APL
Advanced Techniques and Utilities

by Gary A. Bergquist

Zark Incorporated
53 Shenipsit Street
Vernon, Connecticut  06066
# TABLE OF CONTENTS

**INTRODUCTION** ... 1

1 **LIMBERING UP** ... 4

2 **BRANCING AND LOOPING** ... 8
   - Conditional Branching ... 8
   - Multi-target Branching ... 9
   - Looping ... 11
   - When to Loop in APL ... 15
   - PROBLEMS ... 20

3 **COMPUTER EFFICIENCY CONSIDERATIONS** ... 23
   - Timing Alternative Algorithms ... 23
   - A Utility Function for Timing Algorithms ... 25
   - Fine-tuning Production Applications for Efficiency ... 28
   - APL Efficiency Considerations ... 33
   - PROBLEMS ... 38

4 **POSITIONING CHARACTER DATA** ... 40
   - Removing Extra Blanks from Character Vectors ... 40
   - Justifying Nonblank Segments within Character Arrays ... 42
   - Restructuring Skinny Matrices into Fat Ones ... 43
   - Delimited Character Vector to Justified Matrix ... 46
   - PROBLEMS ... 49

5 **SORTING AND SEARCHING** ... 53
   - Major-to-minor Sorting ... 53
   - Character Matrix Sorting ... 56
   - Uppercase/Lowercase Sorting ... 62
   - Array Searching ... 64
   - Range Searching ... 70
   - Character Substring Searching ... 75
   - Character Substring Replacement ... 76
   - PROBLEMS ... 78

6 **SELECTING** ... 80
   - Selection and Selection Assignment ... 80
   - Scattered Point Indexing ... 82
   - Unique (Distinct) Values ... 84
   - Translating Distinct Values to Distinct Indices ... 88
   - PROBLEMS ... 89

7 **FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABULATIONS** ... 92
   - One-Way Plus Reductions ... 94
   - N-Way Plus Reductions ... 97
   - N-Way Maximum and Minimum Reductions ... 100
   - N-Way Logical Reductions ... 104
   - N-Way Reduction Utility Functions ... 105
   - N-Way Reductions on Files ... 109
   - Milky-Way Reductions ... 111
   - PROBLEMS ... 114
TABLE OF CONTENTS (continued)

8 WRITING USER-FRIENDLY INTERACTIVE FUNCTIONS ... 116
    Primitive Interactive Functions ... 116
    Utility Interactive Functions ... 119
    Utility Validation Functions ... 122
    PROBLEMS ... 127

9 MANIPULATING DATES ... 129
    Representation of Dates in APL ... 129
    Entering and Validating Dates ... 131
    Formatting Dates for Output ... 133
    Manipulating Dates ... 135
    PROBLEMS ... 144

10 WRITING REPORTS ... 145
    Viewing the Report ... 146
    Constructing Titles and Headings ... 147
    Row Oriented Formatting ... 150
    Formatting Multi-Row Records Using Newlines ... 151
    Directing Report Output to Print Files ... 152
    PROBLEMS ... 156

11 SYSTEM DEVELOPMENT PROCEDURE ... 157
    Familiarization ... 158
    Specification ... 159
    File Design ... 160
    Workspace Design ... 160
    User Documentation ... 161
    Flow Charting ... 162
    Coding, Typing, Testing ... 162
    Delivery, Training ... 163

12 PROGRAMMING STANDARDS ... 165
    Familiarization ... 165
    Specification ... 166
    File Design ... 166
    Workspace Design ... 167
    User Documentation ... 168
    Flowcharting ... 169
    Coding, Typing, Testing ... 170
    Delivery, Training ... 174

13 WORKSPACE DESIGN AND DOCUMENTATION ... 175
    Subfunction Design ... 175
    Starting an Application ... 185
    Function Documentation ... 188
    Workspace Documentation ... 191
    Function Identifiers ... 194
    Workspace Identifiers ... 200
    PROBLEMS ... 202
TABLE OF CONTENTS (continued)

14 FILE DESIGN AND UTILITIES ... 203
   APL Database File Organization ... 204
   File Design Considerations ... 208
   Efficient Record Location ... 210
   File Documentation ... 212
   File Utility Functions ... 215
   Multi-Set Transposed File Organization ... 222
   An Illustration of File Utilities ... 225
   PROBLEMS ... 228

15 BOOLEAN TECHNIQUES ... 230
   Logical Scalar Functions ... 231
   Logical Reductions and Scans ... 233
   Logical Shift-and-Compare (Map) Operations ... 236
   Logical Partition Operations ... 237
   An Illustration of Boolean Techniques ... 244
   PROBLEMS ... 252

16 IRREGULAR ARRAYS ... 253
   Constructing Irregular Arrays ... 255
   Emulating Nested Arrays on Non-Nested Systems ... 257
   Catenating to Irregular Arrays ... 262
   Selecting from Irregular Arrays ... 264
   Replacing Items of Irregular Arrays ... 265
   Determining Shapes of Irregular Items ... 266
   Sorting Character Nests ... 267
   Searching Character Nests ... 268
   Reducing Numeric Nests ... 269
   PROBLEMS ... 269

17 CURVE FITTING ... 271
   Using Quad-Divide ... 272
   Forecasting ... 274
   Fitting Data to a Nonlinear Formula ... 277
   Finding the Best Formula ... 278
   PROBLEMS ... 279

18 FINANCIAL UTILITIES ... 282
   Interest and Annuities ... 282
   Loan Amortization Schedules ... 290
   Internal Rate of Return ... 293
   Bond Calculations ... 297
   PROBLEMS ... 301

19 EXCEPTION HANDLING ... 303
   Detecting the Error ... 304
   Signalling the Error ... 307
   Detecting the Attention ... 308
   Suspending the Function ... 309
   Controlling the State Indicator ... 311
   PROBLEMS ... 314

POSTSCRIPT ... 316
| Solutions: | 1 | LIMBERING UP ... 320 |
| Solutions: | 2 | BRANCHING AND LOOPING ... 324 |
| Solutions: | 3 | COMPUTER EFFICIENCY CONSIDERATIONS ... 327 |
| Solutions: | 4 | POSITIONING CHARACTER DATA ... 330 |
| Solutions: | 5 | SORTING AND SEARCHING ... 334 |
| Solutions: | 6 | SELECTING ... 337 |
| Solutions: | 7 | FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABS ... 340 |
| Solutions: | 8 | WRITING USER-FRIENDLY INTERACTIVE FUNCTIONS ... 362 |
| Solutions: | 9 | MANIPULATING DATES ... 364 |
| Solutions: | 10 | WRITING REPORTS ... 366 |
| Solutions: | 13 | WORKSPACE DESIGN AND DOCUMENTATION ... 367 |
| Solutions: | 14 | FILE DESIGN AND UTILITIES ... 383 |
| Solutions: | 15 | BOOLEAN TECHNIQUES ... 449 |
| Solutions: | 16 | IRREGULAR ARRAYS ... 459 |
| Solutions: | 17 | CURVE FITTING ... 474 |
| Solutions: | 18 | FINANCIAL UTILITIES ... 476 |
| Solutions: | 19 | EXCEPTION HANDLING ... 479 |

INDEX ... 483
INTRODUCTION

It is a mystery that APL is more than 20 years old and there are no APL textbooks which treat the reader as if he or she has some understanding of the language. The introductory APL textbooks available are excellent at accomplishing their objectives. However, they leave the novice APL programmer stranded in the real world. The novice APLer has the tools but not the techniques, the knowledge but not the experience.

This book picks up where introductory APL textbooks leave off. Its goal is to build your experience quickly by exposing you to applications of APL. This is accomplished by presenting real world problems and their APL solutions.

Most sections of the book begin with the presentation of a problem. You should read the problem and formulate a solution to it, given your knowledge of APL. Then read on. The problem is followed by a "good" APL solution. Compare it to yours. If different, learn from the differences.

Each chapter is followed by a set of problems. The purpose of the problems is to confirm your understanding of the material presented in the chapter. You will reinforce that understanding by working on the problems. The solutions to the problems are in the back of the book.

Some of the most valuable material in the book is presented as utility function solutions to problems. Therefore, you should at least scan the problems and solutions after reading each chapter, even if you feel you need no reinforcement.

The book assumes you understand the APL primitive functions. If you encounter a primitive function with which you are unfamiliar, look it up in an introductory APL textbook. Though the book does not assume you are using any particular implementation of APL, it does make specific references to three versions: APL2, APL*PLUS, SHARP APL. APL2 is a product of IBM; APL*PLUS is the trademark of a product of STSC, Inc.; SHARP APL is the trademark of a product of I. P. Sharp Associates, Ltd.

If you are using a different implementation of APL, the material in the book will still be pertinent. Only one primitive function is
assumed which you may not have: replicate (/). If your version of APL supports compression but not replicate, i.e. generates a DOMAIN ERROR on 3/4, you will need to substitute your own replicate function, say REPL, whenever replicate is used. The listing of one such REPL function is included at the end of this Introduction.

Most experienced APL programmers collect a set of their favorite utility functions. These utility functions are used to increase programmer productivity by solving the same problem over and over again. This book contains and describes more than 150 commented utility functions. These functions are available on a floppy disk. See the Postscript at the end of the book.

Notice the workspace ID (WSID:) displayed above the header of the REPL function below. The WSID refers to the name of the workspace in which the REPL function may be found. This convention is used throughout the book. Every function for which a WSID is provided is available on floppy disk.

In several sections of the book, you are asked to imagine extensions to the APL language. Each extension is then implemented via a utility function. These imaginary extensions to APL are intended as instructive and mnemonic devices to help you to quickly understand and remember the definition of the utility function. Please do not misinterpret my intent. I do not seek to have these extensions implemented in the current versions of APL. In some cases, the extensions are half-baked or are inconsistent with existing APL conventions. No matter. Use them as they are intended. Allow yourself to imagine the extension and then view the utility function as the implementation of that extension.

There is much emphasis in the book on efficiency considerations. A chapter is devoted to the topic. In addition, relative efficiencies of alternative algorithms are considered throughout the book. Emphasis is placed upon efficiency because of its importance. At an introductory level of APL, you can concentrate on the conciseness of APL and on its elegance. But in the real world, the practical APL programmer must take a blue collar approach.

Sometimes a concise and elegant algorithm requires much more processing time than a somewhat more complicated algorithm. The difference may be significant enough to make an application feasible or infeasible depending upon the algorithm chosen. However, efficiency does not need to come at the cost of clarity. If comments are used generously and subfunctions used judiciously, you can have the best of both worlds: fast, readable functions.

I solicit comments and suggestions about the topics, presentation and utility functions contained herein. In fact, if you make a suggestion that is incorporated into the text or utility functions of the next edition of the book, you will receive a free copy of the current version of the utility functions.
INTRODUCTION

I wish to acknowledge the efforts of everyone who has contributed to the creation of this book, especially: Bob Richmond, Christine Bell, Daryl Burbank-Schmitt, George Dobbs, Joe Hatfield, Bruce Hitchcock, Roger Hui, Don Lagosz-Sinclair, Lori McNichols, Jack Reynolds, David Routhier, Tapan Roy, Jerry Turner and Andi.

Gary A. Bergquist
Zark Incorporated
53 Shenipsit Street
Vernon, CT 06066

March 1987

[WSID: UTILITY]

\[
\nabla \text{ R} \leftarrow \text{B REPL V;I:N:P:T;II} \text{O}
\]

1. \textbf{A} Emulates the replicate function (R\leftarrow B/V), where B
2. \textbf{A} may be non-Boolean. Works for scalar/vector right
3. \textbf{A} argument only.
4. \textbf{A} Branch if right argument not a singleton:
5. \quad \rightarrow (1 \neq N \times \rho V) \rho L1
6. \quad \text{R} \leftarrow (+/B) \rho V
7. \quad \rightarrow 0
8. \textbf{A} Branch if left argument not a singleton:
9. \quad L1: \rightarrow (1 \neq \rho B) \rho L2
10. \quad \text{R} \leftarrow ,\times (B,N) \rho V
11. \quad \rightarrow 0
12. \textbf{A} Origin 0 logic is simpler:
13. \quad L2:II0=0
14. \textbf{A} Flag nonzero replication factors:
15. \quad P\times B
16. \textbf{A} Indices into V of values to replicate:
17. \quad N\times I\times P/1N
18. \textbf{A} Indices into result of starts of runs:
19. \quad T\times \downarrow \backslash P/B
20. \textbf{A} All-zero vector with length of result:
21. \quad R\leftarrow (-1T)\rho 0
22. \textbf{A} Insert 1st differences for subsequent +\backslash:
23. \quad R[N\rho 0,T]\leftarrow I-N\rho 0,I
24. \textbf{A} Replicate selected elements of V:
25. \quad R\leftarrow V[+\backslash R]

\]
The purpose of this chapter is to give you an opportunity to crack your knuckles and stretch you muscles on some APL problems. If you have not used APL for awhile, you will want to spend some time solving these problems. The effort will put your mind in the proper APL orientation to get the most out of the book. If the solutions are different from your own, spend some time studying them. Review any primitive functions with which you are unfamiliar.

If you use APL daily and the problems seem simple to you, skip this chapter altogether. (Solutions on pages 320 to 323).

1. What expression will change the value 645 in the vector AMOUNT to 845?

2. What expression will return the scalar 1 if all elements of the numeric vector PREMS are between 100 and 500, and will return the scalar 0 otherwise?

3. What expression will return the number of elements in the numeric vector WEIGHT which are approximately equal to 24? "Approximately" means the numbers are rounded to the nearest integer before comparing to 24.

4. What expression will return the number of elements in the matrix MAT?
5. Given a variable ANS which represents a numeric scalar (say 56.5), what expression will return the character vector 'ANSWER IS 56.5 YEARS'?

6. What expression will cause the character vector NAME to be catenated as a new row of the character matrix NAMES, assuming the number of elements in NAME is less than or equal to the number of columns in NAMES?

7. What is the effect of the expression, \( \rightarrow p 12 \) ?
   A. Proceed to the next statement
   B. Proceed to line 12
   C. Proceed to line 1
   D. Exit the function
   E. RANK ERROR

8. Given an integer vector V of length 2\( \times \)N, construct an N-element vector R by adding the odd elements of V (1, 3, 5, ...) to the even elements (2, 4, 6, ...) times 256.

9. What is the result of l/\( \backslash 0 \) and why?

10. What is the meaning of -\( \backslash \)VECTOR?

11. How do you resume execution of a function after an error has occurred and been corrected?
12. What happens when closing function definition mode after editing the header of a suspended function?

13. What is the meaning of ••VECTOR?

14. a. What expression describes the shape of the result of A+.×B?
   
   b. If either argument is a scalar?

15. a. What expression describes the shape of the result of A.>B?
   
   b. If either argument is a scalar?

16. What system function can be used to determine the amount of CPU time consumed by an APL expression?

17. What expressions may be used to display the character vector PROMPT and to allow the user to enter a response (R) on the same line as, and following, the display of PROMPT?

18. a. What expression will construct a character matrix which will generate N blank lines when displayed?
   
   b. What expression will construct a character vector which will produce the same effect?

19. What expression will cause all variables in the workspace to be erased?
20. a. What would you type before running the function MODEL to cause the computer to stop before executing each of the lines 12 and 14?

b. The following is a partial display of the function MODEL:

[12] Q=INTERPOLATE T
[13] T=T,Q

By executing MODEL with the stops specified above, the value of T after the first stop is 6 11 13 and after the second stop is 6 11 10. What corrective action would you take?

21. What expression will return the number of lines in the function CALC?

22. Given two vectors, V1 and V2, an INDEX ERROR is signalled by the last of the following expressions:

IND=V1\ll V2
GOOD=IND\leq V1
K=(V1\times V1)[GOOD/IND]

Why?
Branching in APL is a paradox. The definition of the branch function (→) is simple but its application is not. In this chapter we discuss applications of the branching function for: conditional branching, multi-target branching and looping. Finally, the efficiency considerations of looping in APL are discussed.

PROBLEM: Branch to the line labeled CALC if the value of the variable X is greater than 5.

TOPIC: Conditional Branching

This problem requires a conditional branch statement. If the condition (X>5) is true, you want the program's flow of execution to proceed to the line labeled CALC. If untrue, you want to continue at the next statement. The conditional branch statement can be expressed in many ways, the following being some of the more typical:

1. →(X>5)/CALC  4. →CALC×:\X>5  7. →(X≤5)\CALC
2. →(5>5)\CALC  5. →CALC\\X>5  8. →CALC UNLESS X≤5
3. →(X>5)↑CALC  6. →CALC IF X>5

While all of these expressions appear to be adequate, there exist subtle differences between them. Algorithm 1 (/) is the most commonly used conditional branching algorithm. Algorithm 2 (p) is the fastest. Algorithm 3 (↑) is a graphic complement to algorithm 7 (\). Algorithms 4 (×\) and 5 (\↑) are more readable than algorithms 1, 2 and 3 since the word "IF" may be read in place of the ×\ or \↑. However, ×\ does not work in index origin 0 and \↑ does not allow branching to line 0 (i.e. exiting the function) in origin 1.
Algorithm 6 (IF) is the most readable algorithm but requires the existence of the subfunction IF (which is a problem if your function must be self-contained) and is slightly slower than the other algorithms.

Algorithms 7 (feeds) and 8 (UNLESS) require the logical negation of the condition and so may be read as "unless". They may be used when the condition is expressed in such a way that the opposite condition requires the branch (e.g. \( (M \in \text{MVEC}) \) instead of \( (\neg M \in \text{MVEC}) / \text{APPEND} \)). Algorithm 8 (UNLESS) has the same slight disadvantages of algorithm 6 (IF).

Given such a variety of conditional branching algorithms, which should you use? I prefer to use IF and UNLESS when extreme efficiency and self-containment are unnecessary (most of the time). Otherwise, I use \( \rho \) and \( \downarrow \). Whichever algorithm you use, be consistent. APL code is easier to read when conventions are used consistently.

---

PROBLEM: Branch to the line labeled CALC if the value of the variable X is 4, to ENTER if X is 7, to STOP if X is 9 and to LOOP if X is 10.

TOPIC: Multi-target Branching

This problem requires a multi-target branch statement. You want the program's flow of execution to jump to one of four different locations within the program depending upon the value of the variable X. The branch statement can be constructed in many ways, the following being two of the more typical:

1. \( (X = 4 7 9 10) / \text{CALC, ENTER, STOP, LOOP} \)  
2. \( (\text{CALC, ENTER, STOP, LOOP})[4 7 9 10 \ X] \)

In general, the first algorithm (/) is used unless the branch variable (X) is an index value (1, 2, 3,...) which corresponds to the index of the desired label in the list of labels. Then indexing ([[]]) is used. For example, if the problem is restated such that X will have the value 1, 2, 3 or 4, then use the expression:

\( (\text{CALC, ENTER, STOP, LOOP})[X] \)

Notice that the first algorithm (/) actually causes a branch to one of five locations, not four. The fifth location is the next...
statement and is reached when the condition is untrue for all values supplied (e.g. if X is 6). This bonus branch location is not available when using the second algorithm ([[]]) since an invalid branch value (e.g. if X is 6) causes an INDEX ERROR. You may avoid the INDEX ERROR by including an additional label to which the branch should take place if there is no match:

\[ \rightarrow (\text{CALC, ENTER, STOP, LOOP, OTHER})[4, 7, 9, 10, \text{iX}] \]

The additional label not only prevents an INDEX ERROR but has a possible advantage over the first algorithm (/) in that you are not forced to drop through to the following statement when there is no match.

Both algorithms cause a branch to the label corresponding to the first true condition. In the above example, the conditional expression (X= 4 7 9 10) may have at most one true condition. However, if the expression is rewritten (e.g. X=4 7 9 10), there may be more than one true condition (e.g. if X=i8), in which case only the first true condition will be honored.

When using multi-target branching algorithms, it is easy to overlook the fact that the expression

\[ \rightarrow(L1, L2, L3, L4, L5, L6, L7)[\text{TYPE}] \]

requires 8 primitive functions (\[.,.,.,[],[]\]), not 2 (\[[]\]). The catenation commas are readily dismissed as mere aesthetic punctuation. When extreme efficiency is important, the labels should be catenated once, outside of any loops in which the multi-target branching is being employed:

\[
\text{LABS}+L1, L2, L3, L4, L5, L6, L7 \\
\vdots \\
\text{LOOP:} \\
\vdots \\
\rightarrow\text{LABS[TYPE]} \\
\vdots \\
\rightarrow\text{LOOP}
\]

```
PROBLEM: Construct looping logic which will allow the function PROCESS to be executed N times. The right argument of PROCESS is I, where I is the index number of the iteration (1, 2, 3, ..., N).

TOPIC: Looping

The simplest looping logic is:

```
I=1
LOOP: PROCESS I
I=I+1
→LOOP IF I≤N
```

However, this logic breaks down when N=0. The check for completion is not made until after PROCESS has been executed at least once. A safer, but less simple, set of logic is:

```
I=1
LOOP: ENDLOOP IF I>N
PROCESS I
I=I+1
→LOOP
ENDLOOP:
```

Naturally, the conditional branch in both sets of logic above may be replaced by any valid form of conditional branching (discussed above).

Notice the looping overhead which takes place within each iteration. In particular, the counter (I) is incremented, a comparison (>) is made and a conditional branch (→ENDLOOP IF ...) is performed. When extreme efficiency is important, some of this overhead can be removed from the looping logic by precalculating the branch labels:

```
I=1
→LAB←(N→LOOP),ENDLOOP
LOOP: PROCESS I
I=I+1
→LAB[I]
ENDLOOP:
```

Notice that this logic works correctly for the N=0 case. This looping logic is the most efficient possible. However, you should be careful when using it. The shortcoming of this approach is that you must have available workspace for the entire label vector. For example, if you plan to iterate 5000 times, you must have room for a 5001 element integer label vector.
There are two other rather unconventional algorithms for looping which "loop" without branching back. One involves the use of execute ('):.

\[ (N \geq I \rightarrow 1) / \text{LOOP} \gets ' \text{PROCESS} \ I \Diamond \ I \leftarrow I + 1 \Diamond \ (I \leq N) / \text{LOOP} ' \]

(Note: \( \Diamond \) is an APL statement separator and is not available in all APL installations.) The other involves the use of a recursive function:

\[ \text{LOOP} \ N \]

where the function LOOP is defined as:

\[
\begin{align*}
\text{LOOP} \ I \\
[1] & \rightarrow I \leftarrow 0 \\
[2] & \text{PROCESS} \ 1 + N - I \\
[3] & \text{LOOP} \ I - 1 \\
\end{align*}
\]

These two looping algorithms are confusing, inefficient and may cause unexpected complications (e.g. STACK FULL or WS FULL).

Some extended APL systems which support nested arrays have a primitive "iterating" operator named "each" ("·"). The problem stated above can be solved via:

\[ \text{PROCESS} \cdot \n N \]

This expression is significantly simpler and more efficient than the sets of looping logic above. However, it has two drawbacks. The first is the same drawback which the precalculated label vector logic has, namely that you must have available workspace for the entire vector of counter (I) values.

The second drawback is that this expression will not work (as is) if \( N = 0 \). A DOMAIN ERROR or NONCE ERROR will result because the nested array system does not know what fill value (prototype) to associate with the empty result of PROCESS, should it have a result. For example, if the normal result of PROCESS is a character scalar for a numeric scalar argument, you would expect the result of \( \text{PROCESS} \cdot \n 0 \) to be an empty character vector, not an empty numeric vector. The APL system has no way of knowing the nature of the result of PROCESS without executing it at least once.

Viewing the "each" operator as an "iterator" rather than as a parallel processor can quickly lead to expressions which over-kill a problem. For example, consider this file-summarizing logic:
N←1000
SUM←0
I←1
LOOP: SUM←SUM+/READ I
I←I+1
→LOOP IF I≤N

This logic loops through 1000 components of an APL file, reading and summing the 5000-element numeric vectors found in each component. The equivalent nested arrays expression is:

\[ \text{SUM} ← +/\text{READ} \]

The expression is certainly concise. However, at one point during the execution of the expression, the contents of the entire 5,000,000 element file exist in the workspace as a temporary nested variable. This problem can be circumvented by writing a function SUMREAD which returns the sum of the elements of a specified component:

\[ \text{v R} ← \text{SUMREAD I} \]
\[ [1] \text{R} ← +/\text{READ I} \]
\[ \text{v} \]

Then, the expression can be rewritten as:

\[ \text{SUM} ← +/\text{SUMREAD} \]

Now we have only the temporary 1000-element vector result of \text{SUMREAD} as extra baggage from the nested arrays approach.

One of the recurring criticisms of APL is its lack of primitive looping constructs. Because of APL's array-handling capabilities, looping is not required as often as in other programming languages. However, despite nested array extensions to APL, the need to loop still exists.

Imagine a looping primitive (\(\oplus\)) which solves our PROCESS problem as follows:

\[ I\oplus\text{ENDLOOP,N} \]
\[ \text{PROCESS I} \]
\[ \oplus I \]
\[ \text{ENDLOOP:} \]

The left argument of dyadic loop (\(\oplus\)) or the right argument of monadic loop is the counter variable. The right argument of dyadic loop may contain from 1 to 4 elements:
[1] the line number (exit line) to which execution will proceed at the completion of the loop;

[2] the number of iterations (infinite if omitted);

[3] the value of the counter variable during the first iteration (0 if omitted);

[4] the amount by which the counter variable is to be incremented or decremented after each iteration (1 if omitted).

The monadic loop function increments the counter variable and branches back to the line immediately following the line containing the dyadic loop function if there are more iterations to perform. Otherwise, the counter variable is erased and the flow of execution proceeds to the exit line.

APL utility functions can be written to approximate this behavior:

```
~LOOPI ENDLOOP,N
PROCESS I
~NEXTI
ENDLOOP:
```

The definitions of the LOOPI and NEXTI functions follow:

```
[WSID: LOOP]
\[ R\leftarrow LOOPI \text{ LAB} \]
[1] a Initializes globals (I,loopi) for looping.
[2] a LAB: line to branch to when loop complete,
[4] a Used in conjunction with NEXTI as:
[5] a
[6] a \( \rightarrow \text{LOOPI END,100} \)
[7] a \( \rightarrow \text{PROCESS I} \)
[8] a \( \rightarrow \text{NEXTI} \)
[9] a \( \rightarrow \text{END} \):
[10] a
[11] a Default values of l+right arg if omitted:
[12] a +infinity, 0IO, 1
[13] R\leftarrow \text{LAB,I(φ,LAB)\downarrow 0,(I/10),0IO,1}
[14] a Exit if no iterations at all:
[15] \( \rightarrow (R[1+0IO]<1)\phi 0 \)
[16] I\leftarrow R[2+0IO]
[17] a Top line, exit line, number of iterations,
[18] a current I, increment, current counter:
[19] R\leftarrow \text{loopi}+(1+\text{ILC}[1+0IO]),R,1
\]```
PROBLEM: Suppose you want to compute the running balance of your savings account for the last 24 months. You have made a single deposit at the end of each month. DEPOSIT is a 25 element vector of the opening balance and the 24 monthly deposits. RATE is a 24 element vector of the monthly interest rates during this time, expressed as fractions (e.g. .0075 .0081 .0078 ...). You may compute the balances iteratively with the following formula:

\[
\text{BALANCE}[T+1] = \text{DEPOSIT}[T+1] + (\text{BALANCE}[T] \times (\text{RATE}[T]+1))
\]

where T goes from 2 to 25 and where \text{BALANCE}[1]=\text{DEPOSIT}[1]. What APL algorithm may be used to compute this stream of cash balances without looping?

TOPIC: When to Loop in APL

Because APL code is interpreted and not compiled, a looping algorithm is generally less efficient than a non-looping algorithm. For example, the expression SUM+/VECT will be significantly faster than the looping algorithm:
During each iteration of the loop, every symbol of code is reinterpreted. The addition function is actually a small portion of the processing being performed during the loop.

This situation leads to two conclusions:

1. Avoid looping in APL when possible;
2. When looping is necessary, remove as much code as possible from within the loop.

To illustrate, let us solve the above problem in a casual, looping fashion:

```
[WSID: CASHBAL]

v BALANCE←RATE CASH1 DEPOSIT;I;N
[1] v Returns stream of cash balances for deposits
[2] v DEPOSIT and corresponding rates RATE.
[3] N←DEPOSIT
[4] BALANCE←(DEPOSIT)₀
[6] I←1
[7] LOOP:→END IF I≥N
[8] I←I+1
[10] →LOOP
[11] END:
```

-16-
Now let's squeeze everything possible from the loop:

```apl
v BALANCE←RATE CASH2 DEPOSIT;B;I;LAB;N
[1] a Returns stream of cash balances for deposits
[2] a DEPOSIT and corresponding rates RATE.
[3] N←DEPOSIT
[4] BALANCE←0
[6] RATE←RATE+1
[7] LAB←(N←LOOP), END
[8] →LAB[I+2]
[10] →LAB[I+1]
[11] END:
```  

Given these modest modifications, we can expect the function CASH2 to take perhaps 60% to 70% as long to run as CASH1.

Now let's look for a non-looping solution. Let's refer to the elements of 1+RATE as R1, R2, R3,... Let's refer to the elements of DEPOSIT as D1, D2, D3,... Then the elements of BALANCE which we seek may be computed by the following expressions:

\[
\begin{align*}
D1 , & \quad R1\times D1 , \quad R1\times R2\times D1 , \quad R1\times R2\times R3\times D1 , & \ldots \\
+D2 , & \quad +R2\times D2 , \quad +R2\times R3\times D2 , & \quad +R3\times D3 , \quad +D4 \\
\end{align*}
\]

Our objective is to find some APL expression which will generate this vector. We will accomplish this by performing a series of transformations to these elements until the resulting elements can be easily produced with an APL expression. We will then apply APL expressions which will reverse the transformations.

Let's begin by defining the vector RSCAN:

\[
\begin{align*}
1 , & \quad R1 , \quad R1\times R2 , \quad R1\times R2\times R3 , & \ldots \\
\end{align*}
\]

We will divide our desired result by RSCAN, giving:

\[
\begin{align*}
D2 , & \quad D2 , \quad D3 , \quad D4 \\
D1 , & \quad D1\times R1\times R2 \quad \times \quad D1\times R1\times R2\times R3 \\
R1 , & \quad \times \quad R1\times R2 \quad \times \quad R1\times R2\times R3 \\
\end{align*}
\]

Take the first difference (V[I+1]-V[I]) of these elements, giving:

\[
\begin{align*}
D2 , & \quad D3 , \quad D4 \\
D1 , & \quad D3 \quad \times \quad D4 \\
R1 , & \quad R1\times R2 \quad \times \quad R1\times R2\times R3 \\
\end{align*}
\]
Multiply the result by RSCAN, giving DEPOSIT:

\[ D_1, D_2, D_3, D_4, \ldots \]

Now, undo each transformation in reverse order. Undo the multiplication by RSCAN:

\[ \text{DEPOSIT} \div \text{RSCAN} \]

To undo the first difference, you must realize that the cumulative sum (⁺\(\backslash\)) is the inverse of the first difference (⁻\(\backslash\)):

\[ +\backslash \text{DEPOSIT} \div \text{RSCAN} \]

Undo the division by RSCAN:

\[ \text{RSCAN} \times +\backslash \text{DEPOSIT} \div \text{RSCAN} \]

There it is. Expressed as a function:

\[ \text{CASHBAL} \]

\[ \nu \text{ BALANCE} \div \text{RATE} \text{ CASH3} \text{ DEPOSIT; RSCAN} \]

[1] \(\nu\) Returns stream of cash balances for deposits

[2] \(\nu\) DEPOSIT and corresponding rates RATE.

[3] \(\nu\) Performs:

[4] \(\nu\) BALANCE\[I\] ← DEPOSIT\[I\] + BALANCE\[I-1\] × RATE\[I-1\] + 1

[5] RSCAN\(\nu\) DEPOSIT\(\backslash\)RATE\(\backslash\) + 1

[6] BALANCE\(\nu\) RSCAN\(\times\) +\(\backslash\)DEPOSIT\(\div\)RSCAN

We can expect the function CASH3 to take perhaps 2% to 5% as long to run as CASH1! This significant improvement in speed does come at the cost of clarity. The algorithm in CASH3 screams out for comments, the least of which should be:

\(\nu\) Performs: \(\nu\) BALANCE\[I\] ← DEPOSIT\[I\] + BALANCE\[I-1\] × RATE\[I-1\] + 1

After seeing an elegant application of the APL scan functions to perform an inherently iterative function, it is easy to become obsessed with the pursuit of non-looping algorithms. Beware! You may invest a greater value of human time than is saved in machine time. As a further irony, you may find that your elegant and sophisticated non-looping algorithm is slower than a compact looping algorithm.

As a guideline, do not spend your time looking for a non-looping algorithm unless all of the following are true:

1. You suspect one exists;

2. The function is used frequently and the looping algorithm is a "bottleneck" in the function;
Chapter 2  

3. The loop involves at least 20 iterations;

4. Transplanting all possible logic from within the loop to outside the loop does not give you satisfactory performance.

5. You do not have access to an APL "compiler". For example, STSC provides a product used in conjunction with its mainframe APL*PLUS System product which can be used to compile selected APL functions to improve their execution speed. The compiling process requires a good deal of both programmer and computer time but can produce dramatic efficiency improvements, especially on highly iterative functions.

If you choose, or are forced, to employ a looping algorithm, you may still solve the overall problem in an efficient manner by considering the context in which the loop is performed. To illustrate, let us consider the problem of computing the yield-to-maturity rates for 1000 coupon-bearing bonds.

Given the parameters which define the cash flows of a bond, it is necessary to solve for the yield by a method of successive approximations (looping). The APL solution to this problem is described in detail in the Financial Utilities chapter. For now, let us assume that we have an algorithm which can be used to determine the yield rate (to satisfactory precision) in no more than 10 iterations.

To compute the yield rates for the 1000 bonds, are we compelled to perform 10,000 iterations?

No. We are forced to loop by successive approximation (10 iterations) but we are not forced to loop by bond. To efficiently solve the problem, we may perform the 10 iterations on the parameters of all 1000 bonds at once. After 10 iterations, we will have the 1000 desired yield rates. Computing yield rates by such an "iterative" APL approach is quite efficient. The processing speed will rival or surpass that of any compiled language.

~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~
PROBLEMS:  

1. What are the problems with the following conditional branch expression?

\[ \rightarrow \text{CALC} \times (X > 5) ? 1 \]

2. Assuming index origin 1, what expression will cause a branch to the line labeled NEGATIVE if \( N \) is negative, ZERO if \( N \) is zero or POSITIVE if \( N \) is positive?

3. Assuming index origin 1, write the looping logic which will add together the 100 matrices in file components 11, 14, 17, ..., 308. Assume the existence of the monadic function READ whose right argument is the number of the component to be read and whose explicit result is the matrix stored in that component (e.g. \( \text{MAT} \rightarrow \text{READ} 11 \)). Use each of the following techniques:

   a. Normal APL looping logic (increment, compare, branch);
   
   b. Precalculated label vector logic;
   
   c. The hypothetical looping primitive (\( \bullet \));
   
   d. The \text{LOOPI}, \text{NEXTI} utility functions.
   
   e. The each (\( ^{\prime} \)) operator.

4. Write non-looping APL logic which is equivalent to the following formula:

\[ \text{OPRIN}[I] = \text{OPRIN}[I-1] - (\text{PMT} \times \text{RATE} \times \text{OPRIN}[I-1]) \]

for \( I \) from 1 to \( \text{TERM} \), where \( \text{OPRIN}[0] = \text{LOAN} \).
5. Write a function CASH4 which uses another approach to perform the same task as that of functions CASH1, CASH2 and CASH3 listed in this chapter. Begin by defining a vector ACCUM:

\[
R_1 \times R_2 \times R_3 \times \ldots, \quad R_2 \times R_3 \times R_4 \times \ldots, \quad R_3 \times R_4 \times R_5 \times \ldots, \quad \ldots, \quad 1
\]

Perform the following transformations on the elements of BALANCE:

A. Multiply by ACCUM
B. Take the first difference
C. Divide by ACCUM

What is the result? Undo the transformations to construct the new algorithm and use it to write CASH4.

6. The following function WRAPLP modifies its character vector right argument so that it will display in the width (number of characters) specified in the left argument. The modification consists of inserting a newline (carriage return) character in place of the last blank character on each line. In that way, words (groups of contiguous nonblank characters) are not broken from one line to the next. Existing newline characters are left unaltered and are used to separate "sentences" within which the above word-wrap logic takes place. For example:

```
pTEST
70
TEST
THIS EXAMPLE IS NOT VERY BIG.
THE FUNCTION WORKS ON LARGE VECTORS TOO.
15 WRAPLP TEST
THIS EXAMPLE
IS NOT VERY
BIG.
THE FUNCTION
WORKS ON LARGE
VECTORS TOO.
```

Rewrite the WRAPLP function to eliminate looping where possible.
Chapter 2
BRANCHING AND LOOPING

[WSID: WP]

\[ R \leftarrow \text{WID} \text{ WRAPLP } \text{CVEC}; \text{I} \leftarrow \text{IO}; \text{BL} \leftarrow \text{BREAK}; \text{I} \leftarrow \text{L}; \text{LAST} \leftarrow \text{LEN}; \text{LIM} \leftarrow \text{NL}; \text{S} \leftarrow \text{START}; \text{TCNL} \]

[1] \text{a} \text{ Wraps text } \text{CVEC} \text{ into lines of length } \text{WID}
[2] \text{a or less by inserting newline characters.}
[3] \text{a Origin 1:}
[4] \text{a 0} \leftarrow \text{IO} + 1
[5] \text{a Newline character:}
[6] \text{TCNL} \leftarrow \text{DTNL} \quad \text{a APL*PLUS}
[7] \text{a TCNL} \leftarrow \text{TC[2]} \quad \text{a APL2}
[8] \text{a TCNL} \leftarrow \text{DAV[157]} \quad \text{a SHARP APL}
[9] \text{a Flag newline characters:}
[10] \text{NL} \leftarrow \text{CVEC} = \text{TCNL}
[11] \text{a Index before start of each sentence:}
[12] \text{START} \leftarrow 0, \text{NL} \leftarrow \text{I} \leftarrow \text{NL}
[13] \text{a Lengths of sentences (between newlines):}
[14] \text{LEN} \leftarrow 1 + (1 \leftarrow \text{START}, 1 + \text{IO} \leftarrow \text{CVEC}) - \text{START}
[15] \text{a Flag valid break points (blank followed by nonblank):}
[16] \text{BL} \leftarrow \text{CVEC} = '\ ' \leftarrow 0
[17] \text{BREAK} \leftarrow \text{IO} \leftarrow 1 \leftarrow \text{BL}
[18] \text{a Initialize result from argument:}
[19] \text{R} \leftarrow \text{CVEC}
[20] \text{a Loop by sentence:}
[21] \text{I} \leftarrow 0
[22] \text{LIM} \leftarrow \text{I} \leftarrow \text{LEN}
[23] \text{LOOP1} \leftarrow \text{((LIM} \leftarrow 0 + \text{I} \leftarrow 1) / 0
[24] \text{L} \leftarrow \text{LEN} \leftarrow \text{I}
[25] \text{S} \leftarrow \text{START} \leftarrow \text{I}
[26] \text{a Loop by line within sentence:}
[27] \text{LOOP2} \leftarrow \text{((L} \leftarrow \text{WID}) / \text{LOOP1}
[28] \text{a Find last break point within WID chars of line:}
[29] \text{LAST} \leftarrow \text{I} \leftarrow \text{BREAK} \leftarrow \text{S} \leftarrow \text{I} \leftarrow \text{WID}
[30] \text{a Advance start to new break point:}
[31] \text{S} \leftarrow \text{S} \leftarrow \text{LAST}
[32] \text{a Insert newline:}
[33] \text{R} \leftarrow \text{S} \leftarrow \text{TCNL}
[34] \text{a Decrement remaining length:}
[35] \text{L} \leftarrow \text{L} \leftarrow \text{LAST}
[36] \text{a Repeat:}
[37] \rightarrow \text{LOOP2}

\]
Chapter 3

COMPUTER EFFICIENCY CONSIDERATIONS

Time is money. The faster an APL function will run, the less it will cost. This is true whether you are using APL on a commercial remote timesharing service or on your dedicated personal computer. In this chapter, we discuss computer efficiency: measuring time consumption and understanding the factors which affect processing efficiency.

PROBLEM: Which expression will execute quicker on a 4000 element numeric vector V?

1. R←+/V÷ρV
2. R←(+/V)÷ρV

TOPIC: Timing Alternative Algorithms

The niladic APL system function DAI (accounting information) returns a numeric vector of miscellaneous usage statistics. The meanings of the elements of the result vary among the different implementations of APL. One element (usually the second) measures the amount of processing time (CPU time) consumed since the current APL session began. It is usually expressed in milliseconds, or 60ths of a second or seconds. We will assume in our discussion that the index origin is 1 and that DAI[2] is the measure of processing time.

Timing an algorithm is then a simple matter of checking the "stopwatch" before and after executing the algorithm:

TIME1←DAI[2]
R←(+/V)÷ρV
TIME2←DAI[2]
USED←TIME2-TIME1

-23-
These expressions should not be executed in immediate execution mode unless they are executed all at once on a single line:

\[
\text{TIME1} \leftarrow \text{DAI}[2] \quad R \leftarrow (+/V) \div \rho V \quad \text{TIME2} \leftarrow \text{DAI}[2] \quad \text{USED} \leftarrow \text{TIME2} - \text{TIME1}
\]

If your implementation of APL does not have a statement separator (e.g. ◇), the expressions should be specified as lines of a function. The reason for avoiding immediate execution mode is that CPU time is being consumed as you are typing each expression. In fact, on a dedicated (i.e. personal) computer, the measure of CPU time is equivalent to the measure of clock time. That is, the value for \text{DAI}[2] increases by 60 seconds every minute whether or not APL expressions are being executed. Therefore, the measured time will include typing time.

Let us solve the problem above:

\[
V \leftarrow 4000 \div 4000 \\
T \leftarrow \text{DAI}[2] \quad R \leftarrow (+/V) \div \rho V \quad \text{DAI}[2] - T \\
675 \\
T \leftarrow \text{DAI}[2] \quad R \leftarrow (+/V) \div \rho V \quad \text{DAI}[2] - T \\
162
\]

From this example, we can begin to see the importance of well placed parentheses. In the first algorithm, the computer performs 4000 divisions and 4000 additions. In the second, it performs 4000 additions and 1 division.

What happens if we time the algorithms again?

\[
T \leftarrow \text{DAI}[2] \quad R \leftarrow (+/V) \div \rho V \quad \text{DAI}[2] - T \\
692 \\
T \leftarrow \text{DAI}[2] \quad R \leftarrow (+/V) \div \rho V \quad \text{DAI}[2] - T \\
155
\]

We get the same approximate results but they are not exactly the same. Why? On a multi-user computer, the results will vary primarily because of the varying requirements of other users at the moment of execution. Even on a dedicated computer, the results may vary because of "house-cleaning" operations performed automatically and sporadically by the APL system and because of imprecise clock resolution. Therefore, if the accuracy of your timings is important, you should perform several timings and average the results.
Chapter 3

PROBLEM: Write a dyadic function TIMER to time the execution of a specified algorithm. The algorithm is provided as an executable character vector right argument. TIMER runs the algorithm N times, where N is the left argument, and returns the average CPU time consumed. For example:

25 TIMER 'R+/V÷pV'
638.44

TOPIC: A Utility Function for Timing Algorithms

In writing the TIMER function, we will attempt to isolate the time consumed during the execution of the algorithm. Any time consumed during the overhead of the timing process itself will be deducted. In this way, and by averaging many samples, we can get timing results which are as precise as possible.

There are two methods in APL for executing a character vector expression under program control. The first is to use the execute (⍴) primitive and the second is to construct and execute a local function which has the expression as one of its lines.

The first approach (⍴) is simpler but not as accurate. When a character vector expression is executed, the expression must first be "parsed" so that the APL interpreter may correctly identify variables, APL primitive functions, character constants and numeric constants. It is during this parsing phase that variable names and function names are translated into pointers and addresses, the natural vocabulary of the computer. This parsing takes place when you enter an expression in immediate execution mode or in function definition mode or when you define a function under program control. Therefore, when you execute the expression as the line of a function, it has already been parsed and will execute quicker than if the expression is executed as the argument to the execute (⍴) primitive.

We will therefore use the second method, constructing and executing a local function, to time the specified algorithm. Our task is to define a function local to TIMER under program control which looks something like the following (say, to time R+/V÷pV):

```
V ELAPSED←RUN1 N;I
[2]  I←0
[3]  LOOP:←(N<I←I+1)ρEND
[5]  →LOOP
```

-25-
This function will run the specified algorithm (on line [4]) \( N \) times, where \( N \) is the right argument (e.g. RUN1 25). It will return the amount of elapsed processing time consumed during its execution.

We will define a second local function RUN2 which is identical to RUN1 except for line [4], which is defined to do nothing:

```apl
[4] DOIT:
```

We may then execute both RUN1 and RUN2 with the same argument, subtract the results (to eliminate the non-algorithm overhead) and divide by the number of iterations to determine the average processing time for a single execution of the specified algorithm. Therefore, TIMER will look something like:

```apl
V R←N TIMER CVEC;RUN1;RUN2
:
:  define RUN1 and RUN2
:
[7]  R←(RUN1 N)←RUN2 N
[8]  R←R÷N
```

How do we define RUN1 and RUN2? There are two popular methods for defining functions under program control. One is to build a character matrix which "looks" like the function, less the dels (\( V \)) and the bracketed line numbers. Each function line, including the header, occupies exactly one row of the character matrix. Each function line is padded with blanks to have as many characters as the longest function line. Such an array is called the "canonical representation" of the function and may be used as the definition of a new function in the workspace. The function is defined by a system function (DFX in APL2, DDEF in APL*PLUS or DF in SHARP APL).

The second method is to build a character vector which "looks" exactly like the function. The lines of the function are separated by the newline (i.e. carriage return) character. Such an array is called the "visual representation" of the function. The function is defined by a system function (ODEF in APL*PLUS or OFD in SHARP APL).

There is one final comment to make before defining the TIMER function. Since the algorithm being timed may involve variables or functions having any valid names, it is possible that these identifiers may coincidentally be the same as the variables local to TIMER and RUN1. We should take some effort to name the local variables so that the chances of a name conflict are minimized. The RUN1 function we will construct will thus look like:
Let us define the TIMER function using the "visual representation" method. We will leave the "canonical representation" method as an exercise at the end of the chapter. We shall use the monadic DDEF system function to define the function. Its result is the character vector name of the function defined. The niladic system function DTCNL returns a character scalar newline character.

```apl
V ⊢E∆←FA ANA; ∆I∆
[1] ⊢E∆←DAI[1+DIO]
[2] '∆I∆←0
[3] ⊢L∆:→(ANA<∆I∆←∆I∆+1)ρ∆Z∆
[4] ⊢D∆:R←(+/V)÷ρV
[5] →∆L∆
```

Let us define the TIMER function using the "visual representation" method. We will leave the "canonical representation" method as an exercise at the end of the chapter. We shall use the monadic DDEF system function to define the function. Its result is the character vector name of the function defined. The niladic system function DTCNL returns a character scalar newline character.

```apl
[V: TIMING]
V ⊢RA←ANA TIMER ⊢CA; ⊢AA; ⊢BA; ⊢FA; ⊢GA; ⊢NL∆
[1] ⊢Times the execution of the character vector
[2] ⊢CA by running it ANA times. Returns a numeric
[3] ⊢a scalar of the average CPU time consumed per run.
[4] ⊢a
[5] a Prepare to build local functions...
[6] a Newline character:
[7] ⊢NL∆←DTCNL a APL*PLUS
[8] a ⊢NL∆←DAV[156+DIO] a SHARP APL
[9] ⊢AA←'V∆E∆←FA ANA; ∆I∆', ⊢NL∆,'[1]∆E∆←DAI[1+DIO]'
[10] ⊢AA←∆AA, ⊢NL∆, '2∆I∆+0', ⊢NL∆
[12] ⊢AA←∆AA, ⊢NL∆, '4∆D∆:'
[15] a
[16] a
[17] a Define local fn ⊢FA to run ⊢CA:
[18] ⊢RA←DDEF ⊢AA, ⊢CA, ⊢BA a APL*PLUS
[19] a ⊢RA←3 DEF ⊢AA, ⊢CA, ⊢BA a SHARP APL
[20] a
[21] a Define local fn ⊢GA to run nothing:
[22] ⊢AA[∆AA1'F']←'G'
[23] ⊢RA←DDEF ⊢AA, ⊢BA a APL*PLUS
[24] a ⊢RA←3 DEF ⊢AA, ⊢BA a SHARP APL
[25] a
[26] a Run the functions (disallow negative result):
[27] ⊢RA←OF( ⊢FA ANA)−GA ANA
[28] a Return the average:
[29] ⊢RA←∆RA+∆NA
```

-27-
PROBLEM: Define a procedure whereby the individual lines of a specified function may be timed so that possible inefficiencies can be quickly located within the function. The ideal end result of such a procedure will be a display like the following (for a 5 line function):

<table>
<thead>
<tr>
<th>LINE</th>
<th>TIMES</th>
<th>TOTAL</th>
<th>AVG</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RUN</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>450</td>
<td>150</td>
<td>122</td>
<td>171</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>15</td>
<td>5</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>153</td>
<td>918</td>
<td>6</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>150</td>
<td>9,280</td>
<td>62</td>
<td>55</td>
<td>66</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1,065</td>
<td>355</td>
<td>240</td>
<td>380</td>
</tr>
</tbody>
</table>

TOPIC: Fine-tuning Production Applications for Efficiency

After designing and implementing an application system in APL, you may find that it operates slower than you anticipated. In fact, the system may be so slow or so expensive that it is infeasible to operate. What can you do?

A procedure such as the one suggested in the problem above allows you to examine the functions for bottlenecks. You begin with the highest level cover functions and work your way into suspicious subfunctions. Having identified the major inefficiencies, you are in an ideal position to correct them. A discussion of the causes and cures for some of the inefficiencies encountered in APL is contained later in this chapter.

To aid in our discussion, suppose the function we wish to time is the following:

```
V MODEL
[1] SETUP
[2] LIM←50 ◆ I←0
[3] LOOP:→END IF LIM<I+I+1
[4] PROCESS ◆ →LOOP
[5] END:CLOSE
V
```

To time the lines of this function, we need to click our "stopwatch" at the beginning and end of each line. This suggests the placement of timer functions at the start and end of each line. For example:

```
[1] START ◆ SETUP ◆ STOP
```
This idea breaks down for lines which involve branches:

[4] START ◆ PROCESS ◆ LOOP ◆ STOP

In this example, the branch to the line labeled LOOP occurs before the STOP function is executed.

Since functions placed at the end of function lines are not reliably executed (due to branching), we must be satisfied with placing timer functions only at the start of function lines. The timer function must then perform two tasks: to stop the stopwatch for the previous line (which may not be the line directly above the current line) and to start the stopwatch for the current line. If we call our timer function \( \triangle \), we may be able to time the MODEL function above by placing the timer function as follows:

\[
\text{v MODEL}
\begin{align*}
[1] & \triangle ◆ SETUP \\
[2] & \triangle ◆ LIM←50 ◆ I←0 \\
[3] & \text{LOOP: } \triangle ◆ \rightarrow \text{END IF } LIM<I+1 \\
[4] & \triangle ◆ \text{PROCESS ◆ } \rightarrow \text{LOOP} \\
[5] & \text{END: } \triangle ◆ \text{ CLOSE} \\
[6] & \triangle
\end{align*}
\]

Notice that a new line must be added (line 6) to stop the stopwatch for the last line of the function (line 5).

If your implementation of APL does not support statement separators (e.g. ◆), you are compelled to insert the timer function on the line before each function line:

\[
\text{v MODEL}
\begin{align*}
[1] & \triangle \\
[2] & \text{SETUP} \\
[3] & \triangle \\
[4] & \text{LIM←50} \\
[5] & \triangle \\
[6] & I←0 \\
[7] & \text{LOOP: } \triangle \\
[8] & \rightarrow \text{END IF } LIM<I+1 \\
[9] & \triangle \\
[10] & \text{PROCESS} \\
[11] & \triangle \\
[12] & \rightarrow \text{LOOP} \\
[13] & \text{END: } \triangle \\
[14] & \text{CLOSE} \\
[15] & \triangle
\end{align*}
\]
It is the job of the timer function (Δ) to do the following:

1. Record DAI[2].

2. Look at a global variable (say ΔT) which contains the value of DAI[2] when Δ was last executed. Subtract this value from the value recorded in step 1. The result is the time consumed by the previously executed line.

3. Look at a global variable (say ΔL) which contains the number of the line for which Δ was last executed. Using this line number and the consumption value computed in step 2, update the global variable accumulation matrix (say ΔM) which has one row per function line and 4 columns: times run, total CPU, minimum CPU, maximum CPU.

4. Update ΔL to contain the number of the current line.

5. Update ΔT to contain the value of DAI[2] at the start of the current line.

To write the timer function (Δ), we will assume the following initial values for the required global variables:

\[
\Delta L \gets 0 \\
\Delta M \gets (N, 4) \rho 0, (1/10), 0
\]

No initial value is set for ΔT since Δ will not refer to it when it is first executed (on line 1) since ΔL is 0. The N used in the assignment of ΔM is the number of lines in the function being timed. The (1/10) is used to generate the largest possible number (the identity element for minimum) for your APL system. We cannot initialize the minimum value to 0 since the 0 will remain as the minimum value. No timing result could be less.

Let us write the timer function (assuming statement separators):
\[\text{TIME} \dot{\text{IO}}\]

1. \text{TIME} \leftarrow \text{OAI}
2. \text{A Records time since last called and resets stopwatch. Checks the 'stopwatch' before anything else.}
3. \text{DIO} \leftarrow 1
4. \text{A Branch if first time called:}
5. \rightarrow (\times \text{IO} \downarrow \text{L1})
6. \text{A Compute time consumed since last called:}
7. \text{USED} \leftarrow \text{TIME}(2) - \text{AT}
8. \text{A Update accumulation matrix:}
9. \Delta M[\Delta L; 1 2] \leftarrow \Delta M[\Delta L; 1 2] + 1, \text{USED}
10. \Delta M[\Delta L; 3] \leftarrow \Delta M[\Delta L; 3] \text{USED}
11. \Delta M[\Delta L; 4] \leftarrow \Delta M[\Delta L; 4] \text{USED}
12. \text{A Update line number:}
13. \text{L1} \leftarrow \Delta L \leftarrow \text{OCL}[2]
14. \text{A Set } \Delta L \text{ to 0 if bottom of function:}
15. \Delta L \leftarrow \Delta L \times \Delta L \leftarrow (\rho \Delta M)[1]
16. \text{A Update 'stopwatch' as last step:}
17. \Delta T \leftarrow \text{OAI}[2]

Notice the use of \text{OCL} (line counter) to compute the number of the current line on which \text{A} is being called. If your implementation of APL does not support statement separators, and the number of lines in your function has been doubled because of the insertions of \text{A}, the computation of \Delta L would be changed to:

\[\Delta L \leftarrow (1 + \text{OCL}[2]) \div 2\]

Along with the \text{A} timer function, we need 3 other functions:

**TIME\text{DEFINE} 'MODEL'** The TIME\text{DEFINE} function modifies the function named in its character vector right argument by inserting the \text{A} timer function before each function line. It also initializes the global variables \Delta L and \Delta M.

**TIME\text{DISPLAY}** The niladic TIME\text{DISPLAY} function generates and displays a formatted report of the contents of the global accumulation matrix \Delta M.

**TIME\text{RESET}** The niladic TIME\text{RESET} function resets the global variables \Delta L and \Delta M to their initial (zeroed out) settings. Then, the function whose lines are being timed may be rerun and the results redisplayed.
The definition of the TIMEDEFINE function is fairly complex, especially if you attempt to write it without looping. Techniques for writing the function are discussed in a subsequent chapter (Boolean Techniques). The definition of TIMEDEFINE is a problem at the end of that chapter.

The definition of the TIMEDISPLAY function is a straightforward formatting problem.

\[\text{TIMEDISPLAY}:A;B;M;\text{IO}\]

- Displays timing data stored in global arrays.
- TIMES TOTAL AVG MIN MAX
- LINE RUN CPU CPU CPU CPU
- ---- ----- ----- ----- ----- ----- ----- 

- 010+--1
- M+[;1] /11ıp M; 1 1 2 2 3 4

- A[M;2]
- B+M[;3]
- M[;1]+ıp A
- M[;4]+B÷A

Some final notes on this line-timing procedure:

1. Since TIMEDEFINE will permanently modify the function being timed, be sure to save a copy of the function before running TIMEDEFINE on it.
2. Functions timed by this procedure are subject to certain constraints:

a. Avoid sudden exits (e.g. \( \rightarrow 0 \)) and insure that the function always exits through its bottom (last line);

b. Avoid branches to absolute or relative line numbers (e.g. \( \rightarrow 5 \) or \( \rightarrow \text{ULC}-1 \) or \( \rightarrow \text{NEXTL} \)) if your implementation of APL does not support statement separators, since \text{TIME\&DEFINE} will change the line numbers.

The mainframe implementation of APL*PLUS provides a system function DMF (monitor facility) which enables you to time the individual lines of a function in much the same way as the utility functions above. If you use a mainframe APL*PLUS system, you should read the documentation to learn how to use DMF and its companion utility functions.

PROBLEM: What factors influence the efficiency of a user-defined APL function?

TOPIC: APL Efficiency Considerations

So far in this chapter we have discussed techniques for timing segments of APL code. Using these techniques, you can isolate inefficiencies in existing functions and you can choose between alternate algorithms. But how can you learn to write efficient APL functions in the first place?

You can develop a feel for the efficiencies and inefficiencies of APL.

If you were to go on a timing rampage and time every stitch of APL code in sight, some patterns would begin to emerge. You would begin to anticipate the relative speeds of algorithms without timing them. More important, you would find yourself formulating a mental model of the inner workings of the computer. Multiplication is more painful to the computer than addition and exponentiation more painful than multiplication. Out of sympathy for the machine, you will find yourself writing \( N+N \) instead of \( 2\times N \), and \( A\times A \) instead of \( A\times 2 \).

While there is no substitute for such an encounter, below are some of the efficiency considerations which you may want to assimilate.
1. Addition (e.g. A+2) is faster than subtraction (e.g. A-2) which is faster than multiplication which is faster than division which is faster than exponentiation which is faster than doing it by hand. For example, (\times N\times 2) will typically be faster than (2\times 11N).

2. Integers are easier (quicker) to manipulate than fractional (floating point) numbers. An array of integers will generally be stored internally in a more compact form (2 or 4 bytes per element) than an array of floating point numbers (8 bytes per element). Integers can be moved about (e.g. \Phi, \dashv, [], /) quicker than floating point numbers and can be more easily operated on computationally (e.g. +, \times, \tau, \\).

3. Boolean arrays (all 0s and 1s) are stored as bits (one-eighth byte per element) in many implementations of APL. On such implementations, operations involving Boolean arrays are either lightening fast or quicksand slow depending upon the implementation. If they are fast, it is because the bits are being manipulated one byte (8 bits) or so at a time or because the CPU is optimized for Boolean operations. If they are slow, it is because each bit has to be yanked from its byte and processed by a CPU which is better suited to working with 4, 8, 16, 32 or 64 bits at a time. Examples of functions so influenced include: \^/, /, \x2190, \x2193, \\.

4. Elements of arrays are stored internally in raveled order. If you picture in your mind's eye this internal "vector" representation of a matrix M, you should appreciate the reason that +/M is somewhat faster than +/\#M. If M is a Boolean matrix (in which each element occupies one-eighth byte) the difference can be dramatic.

5. Execute (\$) is slower than branching because its argument must be parsed. For example:

\$ (I>99) /'/M[15;] \leftrightarrow A+B'

is slower than:

\$ (I\leq 99) \rho L1
M[15;] \leftrightarrow A+B
L1:

6. The workspace may be viewed as a chain of bytes. In a clear workspace, all of the bytes are "clean" (unused). As you execute expressions (e.g. A\leftarrow 2 or B\leftarrow 150 or C\leftarrow A+B), the bytes become occupied by variables. When variables are reassigned (e.g. A\leftarrow 1+A), the old value of the variable is left as so much garbage. The same fate befalls temporary results which are the products of multiple expressions. For example, the expression A+3+2\times 15 produces the temporary results from 15 and from 2\times 15. As you proceed, the
workspace becomes cluttered with a mixture of active variables and functions and garbage (unused) variables and functions. The symbol table is used to keep track of the locations of the active objects.

Eventually, the computer will be asked to perform a function for which it cannot find sufficient clean space for the result. At that moment, the CPU will take a break from its APL function execution chores and will perform spring cleaning. All of the valid objects are shuffled back to the beginning of the workspace chain, the symbol table is updated and the remaining bytes are swept clean, ready to be reused. The CPU then resumes its APL function execution chores.

It is because of these occasional workspace cleanups that you may notice seemingly random peaks when doing timings. Logic which requires a lot of temporary storage will tend to be less efficient than that which requires less storage. For example, index assignment tends to be more efficient than catenation reassignment:

```
LOOP: VEC[I] ← CRUNCH I
vs.
LOOP: VEC ← VEC, CRUNCH I
```

Consider the storage requirements of the second expression when constructing VEC to be 1000 elements, one element at a time. For example, the catenate (,) function must find space for its 932 element result while the 931 element VEC still exists. Of course, a moment later the 932 element vector is assigned the name VEC and the 931 element vector is left to smolder in the ashes. On the other hand, the index assignment approach creates a 1000 element vector just once and then changes individual elements. Much less data shuffling is involved.

7. Shape (ρ) and reshape (ρ) are the most primitive of primitives. The rank and shape of a variable are included as part of its internal representation. The shape function does not have to count its elements; it simply extracts the shape directly. Shape and reshape are extremely fast. For example, $B$ is faster than $B/L$ or $B↑L$. Also, $1ρMAT$ is faster than $1↑ρMAT$. To construct a 100 by 100 identity matrix (all zeros, but ones along the diagonal), the expression $(1100)^{^\circ} = 1100$ may seem simple enough to you but that's because you do not have to perform the 10,000 mindless comparisons. The less intuitive expression $100 100ρ1, 100ρ0$ is dramatically faster because of its use of reshape.

8. When performing scalar operations, time consumption can be measured with a ruler. Because APL is interpretive, it does not check for syntax errors, value errors or argument conformability until it executes the expression. When working with scalars, the time consumed making these checks tends to dwarf the time consumed performing the desired function. Therefore, when considering the efficiency of APL expressions dealing with scalars, it is more
Chapter 3

COMPUTER EFFICIENCY CONSIDERATIONS

pertinent to count the number of functions being executed than to dwell on the nature of the functions. Consider the expression:

\[ R[I-1] \leftrightarrow R[I-1] + (1 + T[I-1]) \div R[I-1] \times G[I-1] \times 2 \]

If we count the functions being performed (counting index assignment as 2 functions), the result is 16. Let us rewrite the expression (assuming \( T1 \leftrightarrow 1 + T \)):

\[ J \leftrightarrow I-1 \]
\[ R[J] \leftrightarrow R[J] - T1 [J] \div R[J] \times G[J] \times 2 \]

These expressions have the same effect as the original expression but involve 12 functions instead of 16. We can expect these expressions to run approximately 25% faster.

Let us label the approximate time consumed when performing a function on scalars a "tad". Then the original expression used 16 tads and the second used 12 tads.

It is important to develop the proper perspective on tads. APL is an efficient language and performs tads extremely quickly. A tad is a miniscule unit of time. APL can perform a hundred tads in a blink of the eye. Tads do not become important until you write functions which consume thousands or tens of thousands of tads, i.e. when you loop.

The lesson here is to avoid looping when APL's array handling capabilities can be effectively employed. Your avoidance should not develop into a mania, however. Looping in APL is fast if there are not too many iterations (say, two dozen) or if the number of tads within the loop is not too large. Do what you can to keep the tads from getting into the tens or hundreds of thousands.

9. Get to know the peculiarities of your APL implementation. For example, say you have a 100,000 element Boolean vector BV which contains only five 1s. Further, say you have only 1000 bytes of available workspace. Many implementations of APL will allow you to execute the expression \( I \leftrightarrow \text{BV} \div \text{\text{\text{lopBV}} \times 2} \) without producing a WS FULL error message. How can this be when \( \text{\text{\text{lopBV}} \times 2} \) results in an integer vector of 100,000 elements (400,000 bytes or so)?

In one set of implementations, the APL interpreter is clever enough to construe the two symbols \( \div \) as a single function. Therefore, the monadic \( \div \) is never executed. Instead, the \( \div \) "function" scans its Boolean left argument for 1s and returns their indices. In these implementations, it is ironic to find a section of "optimized" code like the following:

\[ I \leftrightarrow \text{lopA} \]
\[ IA \leftrightarrow A / I \]
\[ IB \leftrightarrow B / I \]

-36-
The more conventional expressions will typically be much more efficient since they employ /ı as a single function and do not require as much workspace:

IA←A/ıpA
IB←B/ıpB

In a second set of implementations, the result of monadic ı is an arithmetic progression vector and is stored internally as a "J vector". Specifically the computer stores only the vector's length, its starting value and its value-to-value increment. When executing the expression I+BV/ıpBV, the compression (/) function works with its "J" vector right argument without ever building the entire index vector.

Expressions like 2×10+ı1000 are extremely fast on APL systems which use J vectors. In this expression, a single addition and a single multiplication are performed to generate the 1000 element J vector result.

If you do not know whether your APL system employs J vectors, try V←1E9. If no WS FULL message is generated, you have J vectors.

10. Use "compiled" functions when available. Some vendors of APL provide workspaces of utility functions which are written in machine code rather than in APL. These functions are extremely fast and behave like regular APL utility functions. They may be copied into or erased from your workspace and they can have arguments and results. Take some time to explore available workspaces of utility functions.

In addition, much research has been conducted toward compiling APL code. For example, STSC provides a product used in conjunction with its mainframe APL*PLUS System product which can be used to compile selected APL functions to improve their execution speed. The compiling process requires a good deal of both programmer and computer time but can produce dramatic efficiency improvements when applied to bottleneck functions which consume a large portion of the processing time of an application system.

Some APL systems and some related software products allow you to run non-APL programs from within the APL environment. For example, if you have available a program written in another language (say, C or COBOL) which is very efficient and which performs a desired task, you may be able to invoke the program without ever leaving the APL workspace environment.

A final caveat. This list of computer efficiency considerations can create a distorted perspective. Your primary goal as an APL programmer is not to write APL functions which run fast. Your goal
is to get the job done. If getting the job done means writing faster functions, then keep the above efficiency considerations in mind.

In any case, you should also keep in mind these nonefficiency considerations:

1. Your time is more valuable than the computer's. If you find yourself laboring to find a faster algorithm, ask yourself, "Why?" Will the savings in computer time result in a more responsive system which is less frustrating to use and which saves people time? Will the efficiency improvements result in lower computer allocation charges or lower timesharing bills? If you cannot foresee material benefits from your efforts, you are wasting your time.

In general, when writing a function which will perform a one-time task, forget efficiency. Use the code which flows most rapidly from your mind. Get the job done.

2. A readable function is better than a fast one. It is a crime against nature to insert fast, obscure, uncommented code in a production application. Any algorithm which can be understood can be adequately commented. If you do not have the inclination to insert the comments, then do not use the code. By taking a moment to include comments with your efficient algorithm, you will write code which is both fast and readable. Remember, in six months the person who cannot understand your code may be you.

~~~ ~~~ ~~~ ~~~ ~~~ ~~~ ~~~ ~~~

PROBLEMS: (Solutions on pages 327 to 329)

1. Write a niladic function COST which will produce a display like the following:

   13.15 DOLLARS CONSUMED
   65.12 DOLLARS SINCE SIGNON

The first line of the display will not appear the first time COST is run and will thereafter display the dollars consumed since the prior execution of COST. Assume your CPU charge is 75 cents per unit of DAI[2].

-38-
2. Write the TIMER function described in this chapter to work with the canonical representation method of local function definition.

3. In the Sorting and Searching chapter, a function CMIOTA is presented for searching through the rows of one character matrix for the location of the rows of a second. The function is designed to use one of two different algorithms depending upon the number of rows in its arguments. Time CMIOTA (as defined in that chapter) for character matrix arguments of 10, 50, 100, 500 and 1000 rows (12 columns) in all combinations (e.g. 50 row left argument and 100 row right argument). Then, change the line

\[ L2:\rightarrow(F\#1)\rho L4 \]

to

\[ L2:\rightarrow(F\#1)\rho L5 \]

and do all the timings again. The first set of timings uses a sorting algorithm and the second set uses a looping algorithm. Record these numbers. They are required by a problem in the Curve Fitting chapter which determines the constants to be plugged into CMIOTA for automatically choosing the fastest algorithm.

Construct your character matrices such that the rows of your left argument are distinct (or nearly so) and the rows of your right argument are found throughout the left argument. For example:

\[
\begin{align*}
L\leftarrow & 50 \; 12\rho \text{DAV}[(50\times12)\rho 256] & \text{(50 row left argument)} \\
R\leftarrow & L[?100\rho 1\rho L] & \text{(100 row right argument)}
\end{align*}
\]
Many problems in APL involve the realignment of characters. For example, the title of a report may need to be centered above the body of the report; or a character vector entered by the user may need to have any extraneous blanks deleted from it. In this chapter, we discuss techniques for positioning the nonblank character elements of an array for a variety of different applications.

PROBLEM: Write the monadic functions DLB, DTB and DEB for deleting leading, trailing and extraneous blanks from a specified character vector.

TOPIC: Removing Extra Blanks from Character Vectors

The DLB, DTB and DEB functions are frequently used when accepting character input or when generating report output. For example, say you have a character matrix MONTHS of month names, left justified, an integer scalar MNO of the current month (1 to 12) and an integer scalar YR of the current year (e.g. 1987). You want to construct a character vector of the current month and year (e.g. 'JUNE 1987'). The following expression will perform the task:

\[(DTB \text{MONTHS}[MNO;]),', \, \&YR\]

Say you want to build a 30 column character matrix NAMES of employee names by prompting for one name at a time. You want each name to be left-justified in the matrix and to contain no extra spaces between the segments of the name. Use the following expression:

\[\text{NAMES+\text{NAMES},[1]30\text{DEB},[]}\]
The following functions will perform the desired tasks. Notice that alternative algorithms are included in each function. The relative speed of each of the algorithms depends upon the implementation of APL you use. You may want to time them (as discussed in the Computer Efficiency Considerations chapter) to determine which is fastest for your APL environment.

\[ \text{v } R\leftarrow \text{DLB } C \]

1. A deletes leading blanks from character vector C.
2. \[ R\leftarrow (\forall \backslash C\neq \text{}')/C \]
3. \[ R\leftarrow (+/\backslash C\neq \text{}') \downarrow C \]
4. \[ R\leftarrow (((C\neq \text{}') \downarrow (1)-\Pi 0) \downarrow C \]

\[ \text{v } R\leftarrow \text{DTB } C \]

1. A deletes trailing blanks from character vector C.
2. \[ R\leftarrow (+/\backslash ' \neq \text{C})\rho C \]
3. \[ R\leftarrow (\Phi \backslash ' \neq \text{C})/C \]
4. \[ R\leftarrow (-+/\backslash ' \neq \text{C})\downarrow C \]
5. \[ R\leftarrow (\Pi 0-( ' \neq \text{C})\downarrow C \]
6. \[ R\leftarrow (1-(C=' \text{}')\downarrow C \]

\[ \text{v } R\leftarrow \text{DEB } C;N \]

1. A deletes extraneous (leading, trailing, contiguous) blanks from character vector C.
2. \[ N\leftarrow C\neq \text{} ' \]
3. \[ R\leftarrow (\sim 1\uparrow N)\downarrow (N\uparrow 1\downarrow N, 0)/C \]
4. \[ A \]
5. \[ A \]
6. \[ A \]
7. \[ A \]
8. \[ A \]

\[ \text{v } R\leftarrow 1\downarrow (N\uparrow 1\Phi N)/C \]

\[ \text{v } R\leftarrow \text{DTB } C \]

1. A deletes trailing blanks from character vector C.
2. \[ R\leftarrow (+/\backslash ' \neq \text{C})\rho C \]
3. \[ R\leftarrow (\Phi \backslash ' \neq \text{C})/C \]
4. \[ R\leftarrow (-+/\backslash ' \neq \text{C})\downarrow C \]
5. \[ R\leftarrow (\Pi 0-( ' \neq \text{C})\downarrow C \]
6. \[ R\leftarrow (1-(C=' \text{}')\downarrow C \]

\[ \text{v } R\leftarrow \text{DEB } C;N \]

1. A deletes extraneous (leading, trailing, contiguous) blanks from character vector C.
2. \[ N\leftarrow C\neq \text{} ' \]
3. \[ R\leftarrow (\sim 1\uparrow N)\downarrow (N\uparrow 1\downarrow N, 0)/C \]
4. \[ A \]
5. \[ A \]
6. \[ A \]
7. \[ A \]
8. \[ A \]

\[ \text{v } R\leftarrow 1\downarrow (N\uparrow 1\Phi N)/C \]
PROBLEM: Write the monadic functions LJUST, CJUST and RJUST for left-justifying, centering and right-justifying the nonblank text within a specified character vector or matrix.

TOPIC: Justifying Nonblank Segments within Character Arrays

The LJUST, CJUST and RJUST functions are useful for constructing report titles and for merging character matrices. For example, to display ACME INC. centered within a width of 75 characters, use the following expression:

\[
\text{CJUST } 75\uparrow\text{'ACME INC.'}
\]

As another illustration, say you have two 15 column character matrices of left-justified names, LNAMES and FNAMES. You would like to construct a 32 column character matrix of left-justified names in which the names of LNAMES precede the names of FNAMES and are separated by a comma and a single space (e.g. SMITH, JOHN). Use the following expression:

\[
\text{LJUST(RJUST LNAMES),',',' ',FNAMES}
\]

The following are the definitions of these functions:

\[
\text{V R+LJUST C}
\]

\[
\begin{array}{l}
[1] \text{R+(+/
\text{V C=' ')\Phi C}
\end{array}
\]

\[
\text{V R+RJUST C}
\]

\[
\begin{array}{l}
[1] \text{R+(+/
\text{V ' '})\Phi C}
\end{array}
\]

\[
\text{V R+CJUST C;B}
\]

\[
\begin{array}{l}
[1] \text{B+C=' '}
[2] \text{R+(+/
\text{V B)-+/\Phi B}+2)\Phi C}
\end{array}
\]
Chapter 4

PROBLEM: Given a 20 column character matrix of employee names (in alphabetical order), construct an 80 column matrix of the names such that the names run down the resulting matrix in 4 "columns". The resulting matrix will have one-fourth (or so) as many rows as the original matrix has.

TOPIC: Restructuring Skinny Matrices into Fat Ones

Let's illustrate this problem on a simple character matrix of first names.

ANNE
BILL
CAL
DOT
ED
FRED
GAIL
HAL
IKE
JOAN
KEN
LISA
MIKE
NED
PAT
RICK
VI

In this illustration, the initial matrix has 7 columns instead of the specified 20 and the resulting matrix has 28 columns instead of 80. Still, you can see what we want to do. We will solve the problem for this simple 7 column matrix and then modify the solution to work for the specified 20 column matrix.

The brute-force approach to this problem involves breaking the matrix apart into 4 pieces and then sticking them together side-by-side. Assume the name of the character matrix is CMAT. The number of rows in the desired result is computed as:

$$NR = \left\lceil \frac{1}{4} \cdot \text{height}(\text{CMAT}) \right\rceil$$

NR is 5 in our illustration.

The pieces can be extracted by using the take (↑) and drop (↓) functions:

$$P_1 = \left( \frac{NR}{7}, 7 \right) \uparrow \text{CMAT}$$
$$P_2 = \left( \frac{NR}{7}, 0 \right) \uparrow \text{CMAT}$$
$$P_3 = \left( \frac{2 \times NR}{7}, 0 \right) \downarrow \text{CMAT}$$
$$P_4 = \left( \frac{3 \times NR}{7}, 0 \right) \downarrow \text{CMAT}$$
In our illustration, P2 is:

    FRED
    GAIL
    HAL
    IKE
    JOAN

Notice that the last expression pads P4 at the bottom with blank rows if there are fewer than 4×NR rows in CMAT. The last step concatenates the 4 pieces together:

    R←P1,P2,P3,P4

A more elegant solution to this problem involves the use of dyadic transpose. We begin by padding CMAT so that its number of rows is divisible by 4:

    NR←(1↑⍴CMAT)÷4
    CMAT←((4×NR),7)↑CMAT

Second, reshape the matrix into a 3 dimensional array:

    CMAT←(4,NR,7)⍪CMAT

In our illustration, CMAT is now:

    ANNE
    BILL
    CAL
    DOT
    ED

    FRED
    GAIL
    HAL
    IKE
    JOAN

    KEN
    LISA
    MIKE
    NED
    PAT

    RICK
    VI
Notice that each of the planes in CMAT corresponds to one of the "columns" of names in the desired result.

Third, use dyadic transpose to shuffle the planes and rows so that the shape changes from \((4, NR, 7)\) to \((NR, 4, 7)\). Since the first coordinate \((4)\) becomes the 2nd coordinate, the next \((NR)\) becomes the 1st and the last \((7)\) remains the 3rd, use \(2 1 3\) as the left argument (or \(1 0 2\) in origin 0):

\[
\text{CMAT} \leftrightarrow 2 \ 1 \ 3 \% \text{CMAT}
\]

In our illustration, CMAT is now:

<table>
<thead>
<tr>
<th>ANNE</th>
<th>FRED</th>
<th>KEN</th>
<th>RICK</th>
</tr>
</thead>
<tbody>
<tr>
<td>BILL</td>
<td>GAIL</td>
<td>LISA</td>
<td>VI</td>
</tr>
<tr>
<td>CAL</td>
<td>HAL</td>
<td>MIKE</td>
<td></td>
</tr>
<tr>
<td>DOT</td>
<td>IKE</td>
<td>NED</td>
<td></td>
</tr>
<tr>
<td>ED</td>
<td>JOAN</td>
<td>PAT</td>
<td></td>
</tr>
</tbody>
</table>

By performing this transpose, the characters of the array (if raveled) are in the same order as those in the desired result (if raveled).

Finally, reshape the array into the desired two-dimensional result:

\[
\text{R} \leftarrow \text{NR} \times (28) \rho \text{CMAT}
\]

The final solution for the 20 column problem is:

\[
\text{NR} \leftarrow \lceil (1 \rho \text{CMAT}) / 4 \rceil
\]
\[
\text{R} \leftarrow \text{NR} \times (80) \rho 2 \ 1 \ 3 \% (4, NR, 20) \rho ((4 \times NR), 20) \uparrow \text{CMAT}
\]
The dyadic transpose approach is generally more efficient than the brute-force approach. Its work is performed primarily by the relatively efficient reshape and transpose functions. The brute-force approach makes heavy use of the less efficient take, drop and catenate functions.

PROBLEM: Write a function TITLES which will return a character matrix of titles which will be displayed at the top of a report. The left argument is an integer scalar of the width of the resulting character matrix (i.e. the width of the report). The right argument is a delimited character vector (e.g. 'nOPERATING STATEMENTnDEC. 31, 1987n($000''S)') whose "partitions" each begin with one of the delimiters < (left-justify), n (center) or > (right-justify). The result has one row per partition. Each partition is justified within the row according to the delimiter. For example:

```
HDG<50 TITLES'>PAGE 1nOPERATING STATEMENTn1987<($000s)'
HDG
OPERATING STATEMENT
1987
($000s)
```

TOPIC: Delimited Character Vector to Justified Matrix

Let us define the header of the TITLES function:

```
\[ \text{v R<\text{WID TITLES CS}} \]
```

We will use origin 0 throughout:

```
\[ \text{DIO<0} \]
```

Determine which elements of CS are justification symbols:

```
JUST<\text{'<n>'\text{CS}}
BJUST<JUST<3
```

BJUST is a Boolean vector with 1s corresponding to justification symbols.
JUST←BJUST/JUST
NROWS←#JUST

JUST is an integer vector with one element per justification symbol and whose values indicate which symbol (0: left; 1: center; 2: right). NROWS is the number of titles (delimiters). Determine the length (LEN) of each delimited partition (i.e. each title), excluding the justification symbol:

T←BJUST/1#BJUST
LEN←(1↓T#BJUST)-T+1

Reduce (truncate) those lengths which exceed the specified width WID:

LEN←WID↓LEN

Determine the indices into CS of the characters which start each title (i.e. the character after the justification symbol):

ARGSTART←1+T

Determine the number of leading blanks required per title to justify it (left, center or right) within the matrix result:

LEAD←(JUST≠0)×1(WID-LEN)+1+JUST=1

Determine the indices into the raveled matrix result of the characters which start each title:

RESSTART←LEAD+WID×1#NROWS

Initialize the result to have the correct number of characters but to be all-blank and raveled:

R←(NROWS×WID)pter

All that remains is to extract the titles from CD (we know the starting positions, ARGSTART, and the lengths, LEN, of each title), insert them into R (we know the starting positions, RESSTART, and the lengths, LEN) and then reshape R to the proper shape. If there was but one title, we could do the following:

R[RESSTART+1LEN]←CS[ARGSTART+1LEN]

Unfortunately, monadic # will only work with a one element argument. Imagine an enhanced monadic # function which exhibits the following vector behavior:

<table>
<thead>
<tr>
<th>5 2 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 0 1 0 1 2 3</td>
</tr>
</tbody>
</table>

(remember: #IO=0)
Let us assume a function MONIOTA which will work with vectors as above. Then we can finish the function:

\[ R[(\text{LEN}/\text{RESSTART})+\text{MONIOTA \text{LEN}}]\leftarrow CS[(\text{LEN}/\text{ARGSTART})+\text{MONIOTA \text{LEN}}] \]
\[ R+(\text{NROWS}, WID)\rho R \]

The definition of the MONIOTA function we need follows:

\[ \text{WSID: UTILITY} \]
\[ \text{\V} R+\text{MONIOTA \text{LEN}} \]
\[ \text{[1]} \quad \text{\a} \text{Performs: } (\text{\text{LEN}[1]}), (\text{\text{LEN}[2]}), (\text{\text{LEN}[3]}), \ldots \]
\[ \text{[2]} \quad \text{\a} \text{In APL2: } R+\text{\i''\text{LEN}} \]
\[ \text{[3]} \quad R+\text{LEN}/\text{\i''\text{LEN}} \]
\[ \text{[4]} \quad R+\text{\text{LEN}}/\text{\i''\text{LEN}} \]
\[ \text{\V} \]

As an exercise, you should reread the TITLES logic above to see what happens when some of the partitions are empty (e.g. 80 TITLES 'nBALANCE SHEETnnnDEC. 31n').

Finally, let us redefine the TITLES function slightly to allow it to function as a typical character "vector to matrix" converter. Such a function takes a delimited character vector argument and converts it to a character matrix with one row per partition and with as few columns as possible (equal to the length of the longest partition). The rows of such a matrix result are usually left-justified (padded to the right). We will redefine the left argument of TITLES to be either the width of the resulting matrix or an empty vector if the width is to be automatically determined as the length of the longest partition.

To implement this enhancement, we need only precede the line, LEN+\text{WID} \leftarrow \text{LEN}, by the following:

\[ \text{WID}+\text{1} \uparrow \text{WID} , \text{\i''\text{LEN}} \]

Now the TITLES function may be used in the following way:

\[ ' ' \text{TITLES 'cREDcORANGEcYELLOWcGREENcBLUE'} \]
\[ \text{RED} \]
\[ \text{ORANGE} \]
\[ \text{YELLOW} \]
\[ \text{GREEN} \]
\[ \text{BLUE} \]

(The APL purist may prefer to express the empty vector left argument to TITLES as an empty numeric vector such as 0\rho 0 rather than the empty character vector ''). In that way, the WID, \text{\i''\text{LEN}} operation does not engender a conceptual domain error from the catenation of character and numeric data. However, since most implementations of APL "forgive" the catenation of character and numeric datatypes when one of the arguments is empty, this preference is academic.)
The completed TITLES function is listed below.

```plaintext
[WSID: FORMAT]
\n\v R\leftarrow WID TITLES CS; IO; ARGSTART; BJUST; JUST; LEAD; LEN; NROWS ; RESSTART; T
[1] \a Creates report titles from text CS within page
[2] \a width WID. CS is delimited by 'c\n' indicating
[3] \a left, center, right justification respectively.
[4] \a IO\leftarrow 0
[5] \a 0: left; 1: center; 2: right; 3: not a delimiter:
[6] \a BJUST+3\rightarrow JUST\leftarrow 'c\n'; CS\rightarrow CS
[7] \a Select just delimiters; determine no. titles:
[8] \a NROWS\leftarrow 0; JUST\leftarrow BJUST/JUST
[9] \a Title lengths:
[10] \a T\leftarrow BJUST/\\p BJUST
[11] \a LEN\leftarrow (1\downarrow T, 0BJUST)-T+1
[12] \a Set WID as largest title length if empty WID
[13] \a provided; truncate titles to specified width:
[14] \a LEN\leftarrow LEN\downarrow WID\leftarrow 1\downarrow WID, /LEN
[15] \a Index of char following each delimiter:
[16] \a ARGSTART\leftarrow 1\downarrow T
[17] \a Leading blanks per title, to justify:
[18] \a LEAD\leftarrow (JUST\neq 0)\times (WID-LEN)\downarrow 1+JUST=1
[19] \a Ind in raveled result where each segm. starts:
[20] \a RESSTART+LEAD+WID\times 1NROWS
[21] \a Blank, raveled result:
[22] \a R\leftarrow (NROWS\times WID) 000
[23] \a T\leftarrow MONIOTA LEN:
[24] \a T\leftarrow T+1\times T+LEN/-1\downarrow 0, +\downarrow LEN
[25] \a R[T+LEN/RESSTART]\leftarrow CS[T+LEN/ARGSTART]
[26] \a R\leftarrow (NROWS, WID) 0 R
\v
```

PROBLEMS: (Solutions on pages 330 to 333)

1. Given a character vector TEXT which contains embedded newline characters (carriage returns) and given the scalar NL which is the newline character, what expression will return the first line of text (up to, but not including, the first newline)?
2. Given a character vector CODE, find all occurrences of the string '/i'. Return a bit vector which has the same length as CODE, with a 1 in each element which corresponds to the '/i' in a '/i' pair. All other elements are zero.

3. Write a dyadic function CENTER which returns a character vector whose length is specified by the left argument and in which the character vector right argument is centered. For example:

```r
rhoD<50 CENTER 'ACME'
ACME
50
```

Test your function on each of the following:

- 50 CENTER 'ACME'
- 49 CENTER 'ACME'
- 3 CENTER 'ACME'
- 11 CENTER 'A'

4. Suppose you have a dyadic function COLFMT which formats a numeric matrix into a character matrix. Its right argument NMAT is the numeric matrix to be formatted and its left argument CTL is an integer vector with one element per column of NMAT. The integers indicate the number of decimal places, for each numeric column, to be displayed in the character matrix result CMAT. Each number is formatted in a width of <width> characters (e.g. 10), where <width> is an integer scalar global variable. For example:

```r
rhoD<3 0 1 COLFMT 4 3p12
1.000 2 3.0
4.000 5 6.0
7.000 8 9.0
10.000 11 12.0
4 30
```

Write a function ROWFMT which has the same syntax as COLFMT except the elements of its left argument correspond to the rows of the numeric matrix argument rather than to the columns. For example:
ROWFMT should use COLFMT.

5. Write a dyadic function COLUMNIZE which will restructure a skinny matrix into a fat one as described in this chapter. The right argument of COLUMNIZE, CMAT, is the original skinny character matrix and the left argument is the number of "columns" of CMAT across the width of the fat character matrix result. For example, to solve the problem presented in that section, you would use:

\begin{verbatim}
R←4 COLUMNIZE CMAT
\end{verbatim}

Allow a 1 or 2 element left argument. If 2 elements, the first is the number of rows per "page" and the second is the number of "columns" as discussed above. The result is a 3 dimensional character array with one plane per page. For example, using the 7 column character matrix illustrated in this chapter:

\begin{verbatim}
2 3 COLUMNIZE CMAT
\end{verbatim}

\begin{verbatim}
ANNE CAL ED
BILL DOT FRED
GAIL IKE KEN
HAL JOAN LISA
MIKE PAT VI
NED RICK
\end{verbatim}

6. Write a function HEADINGS which will behave as described below:

SYNTAX:  \texttt{CMAT←WIDS HEADINGS CVEC}

DESCRIPTION:

HEADINGS is used to convert a delimited character vector into a character matrix of column headings whose respective widths are given by the vector WIDS. Each substring of CVEC is preceded by a delimiter (\texttt{)) and may contain any number of newline delimiters (\texttt{~}). The newline delimiters therefore separate sub-substrings. Typically, one width (element of WIDS) is provided for each
heading (substring). However, if fewer widths are provided, they are repeated to match the number of headings in CVEC. The headings are formatted into a character matrix according to the following procedure: the sub-substrings of each heading are truncated if necessary to the corresponding width for that heading; the sub-substrings are padded to the left and right with spaces to bring each sub-substring up to the width for that heading; the sub-substrings are catenated together as rows (centered with respect to one another); a row of underlines (hyphens) is catenated to the bottom of each heading; the headings are padded on the top so that each heading has the same number of rows; the headings are catenated together separating them by 2 columns of blanks (if there are more elements in WID than there are headings defined by the right argument, the remaining elements are used as the numbers of columns of blanks to be inserted between each of the pairs of headings):

\[
\begin{array}{cccc}
10 & 13 & 8 & \text{HEADINGS 'nNAMESnHIRE\textbackslash nDATE\textbackslash nAGE\textbackslash nAT\textbackslash nHIRE'} \\
\text{AGE} & \\
\text{HIRE} & \text{AT} \\
\text{NAMES} & \text{DATE} & \text{HIRE} \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
10 & 13 & 8 & 4 & 1 & \text{HEADINGS 'nNAMESnHIRE\textbackslash nDATE\textbackslash nAGE\textbackslash nAT\textbackslash nHIRE'} \\
\text{AGE} & \\
\text{HIRE} & \text{AT} \\
\text{NAMES} & \text{DATE} & \text{HIRE} \\
\hline
\end{array}
\]

Empty sub-strings in CVEC are displayed without underlines. To include an all-blank heading which is underlined, insert at least one blank character in the corresponding substring.
Many applications in the real world deal with lists of things. In APL those things are typically represented as numbers and the lists as vectors; or the things are represented as character vectors (rows) and the lists as matrices. That is, real world lists are usually represented in APL as numeric vectors or character matrices.

Since the most common operations performed on lists include sorting, searching and selecting, these too are among the most important APL operations on vectors and matrices. In this chapter, we discuss primitive and utility APL functions for performing sorting and searching. In the next chapter, we discuss selecting.

PROBLEM: Suppose you have three 1000 element numeric vectors, ENUM (employee identification number), AGE (employee age) and OFFICE (office identification number), that these vectors are in one-to-one correspondence and that each element corresponds to a single employee. How can you reorder these three vectors such that they remain in one-to-one correspondence (i.e. the same index in each vector still corresponds to a single employee) but are sorted by office, and within office by age, and within age by employee number?

TOPIC: Major-to-minor Sorting

Sorting in APL is a two-step process: determine the "grade vector" of the vector to be sorted; and reorder the original vector by indexing the original vector with the grade vector. Therefore, to sort a vector SALARY in ascending order, you would employ the following expression:

`SORTEDSAL ← SALARY[⍋SALARY]`
The grade-up function (↑) returns the grade vector and the indexing function ([ ]) reorders the elements. Note that while we have "sorted SALARY", the variable SALARY remains unsorted (unless we reassign it: SALARY ← SALARY ↑ &SALARY).

Why does sorting require two steps in APL? Because the grade vector is required if we are dealing with several corresponding vectors whose elements must remain in one-to-one correspondence. For example, if we want to reorder ENUM, AGE and OFFICE such that the values of ENUM are in ascending order but still have corresponding elements in AGE and OFFICE, we must do the following:

```
GRADE ← ↑ ENUM
ENUM ← ENUM [GRADE]
AGE ← AGE [GRADE]
OFFICE ← OFFICE [GRADE]
```

Note that the values of AGE and OFFICE are now not in ascending order. They have simply been reordered to continue to correspond to ENUM which is in ascending order.

The sort required in the problem stated above is called a "major-to-minor" sort. It is not possible (usually) to sort all three variables and to maintain the one-to-one correspondence. Only one variable can be strictly sorted (the "major" sort variable). The other variables can at best be sorted within each of the distinct values of the major variable, since only such reordering will maintain the sorted order of the major variable's values. The second variable sorted is said to be "more minor" than the major variable. The third variable is more minor still and may be sorted only within each combination of the distinct values of the two more major sort variables. And so it goes. The last sort variable is called the "minor" sort variable.

How do you do a major-to-minor sort in APL? Backwards. Reorder all the sort variables (to maintain correspondence) by sorting the minor sort variable. Then reorder them by the next more major variable. And so on. The last variable sorted will be the major sort variable and so it will be in strictly sorted order. Since sorting does not change the relative order of the values which are equal, the effects of the earlier sorts will be preserved within each of the distinct values of the major sort variable.

The solution is therefore:

```
GRADE ← ↑ ENUM
ENUM ← ENUM [GRADE]
AGE ← AGE [GRADE]
OFFICE ← OFFICE [GRADE]

GRADE ← ↑ AGE
ENUM ← ENUM [GRADE]
AGE ← AGE [GRADE]
OFFICE ← OFFICE [GRADE]
```
GRADE←OFFICE
ENUM←ENUM[GRADE]
AGE←AGE[GRADE]
OFFICE←OFFICE[GRADE]

If you study this solution, it may strike you that there is much reordering (indexing) going on needlessly. In particular, rather than reorder every variable after each grade operation, you can just reorder the grade vector. Using this approach, the solution becomes:

GRADE←ENUM
GRADE←GRADE[AGE[GRADE]]
GRADE←GRADE[OFFICE[GRADE]]
ENUM←ENUM[GRADE]
AGE←AGE[GRADE]
OFFICE←OFFICE[GRADE]

The processing cost using this latter solution increases linearly as the number of sort variables increases. Using the former solution, the processing cost increases exponentially. However, the latter solution lacks the clarity of the first solution and requires comments. In fact, the latter solution is sufficiently unclear that many APL programmers feel little remorse at jamming the first three lines together using embedded assignment. That solution is included here so that you will recognize it, not as an endorsement:

GRADE←GRADE[OFFICE[GRADE←GRADE[AGE[GRADE←ENUM]]]]
ENUM←ENUM[GRADE]
AGE←AGE[GRADE]
OFFICE←OFFICE[GRADE]

Some implementations of APL support numeric matrix right arguments to grade-up and grade-down. If so, the resulting grade vector is the result of grading the columns of the matrix (from left to right) as major-to-minor variables. If your APL implementation supports this feature, you may solve the above problem with the following expressions:

GRADE←OFFICE,AGE,[1.5]ENUM           (in origin 1)
ENUM←ENUM[GRADE]
AGE←AGE[GRADE]
OFFICE←OFFICE[GRADE]
PROBLEM: Suppose you have a 1000 row, 8 column character matrix, CMAT, of part numbers, one per row (e.g. 'AKI0632B'). How can you construct a 1000 element grade vector which can be used to reorder the rows of CMAT such that the part numbers are in ascending (alphabetic) order?

TOPIC: Character Matrix Sorting

In the previous problem, all sorting was performed on numbers. When numbers are sorted up, they are in ascending order. That means smaller numbers precede bigger numbers. In this problem, we will sort characters, not numbers. What does that mean? If the characters are letters of the alphabet, it means that the earlier (in the alphabet) letters precede the later letters. For characters not in the alphabet we must decide their relative sorting "magnitude" and extend the alphabet accordingly. Such an extended alphabet is called a "collating sequence" and is used as a reference to determine which characters are bigger or smaller than a given character for sorting purposes. The following character vector represents a typical collating sequence.

\[ \text{CS} = '.,:;/G123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ\Delta\Omega' \]

A different collating sequence may define a different result when sorting a given character array. Therefore the collating sequence is a necessary parameter to the solution of the problem. To solve the problem, we will define a dyadic function CGRADEUP whose left argument is the collating sequence, whose right argument is the character matrix to be sorted and whose result is the desired grade vector:

\[ \n\text{V GRADE} \oplus \text{CS CGRADEUP CMAT} \n\]

If the rows of a character matrix are in sorted order, what characteristics do they have? The characters of the first column are in strictly ascending order (as defined by the collating sequence). The characters of the second column are in strictly ascending order within any distinct character in the first column. The third column is sorted within distinct combinations of values in the first and second columns. And so on. In other words, the rows of the matrix are reordered by using the columns (first to last) as the major-to-minor sort keys.

Some APL systems have defined primitive dyadic grade-up and grade-down to solve this problem directly. The left argument of \( \triangleleft \) or \( \triangledown \) is the collating sequence. There is no need to define a CGRADEUP function. The solution to this problem is:

\[ \text{GRADE} \oplus \text{CS} \triangledown \text{CMAT} \]
On APL systems for which dyadic grade-up and grade-down are not implemented, a different approach is required. The most straightforward converts the character matrix to an integer matrix with the same shape whose values are the indices into the collating sequence of the corresponding characters. Then the integer matrix is sorted in major-to-minor order as done in the previous section. Since dyadic \( \iota \) is used to convert characters to indices, the characters in the character matrix which are not in the collating sequence will translate to 1 greater than the length of the collating sequence (or to the length of the collating sequence if the index origin is 0). Therefore, all characters not included in the collating sequence are treated as if they are at the end of the collating sequence.

\[
\text{v GRADE-CS CGRADEUP1 CMAT;I}
\]

[1] \( \text{a Returns grade vector for sorting rows of} \)

[2] \( \text{CMAT with collating sequence CS.} \)

[3] \( \text{a Convert characters to indices:} \)

[4] \( \text{CMAT-CS} \)\( \text{CMAT} \)

[5] \( \text{a Index of last column as a scalar:} \)

[6] \( \text{I} \in (\rho \text{CMAT})[1+\text{IIO}] - \text{IIO} \)

[7] \( \text{a Return trivial result of no columns:} \)

[8] \( \rightarrow (\text{IIO} \rho \text{Ll}) \)

[9] \( \text{GRADE} \)\( \text{Ll} \rho \text{CMAT} \)

[10] \( \rightarrow 0 \)

[11] \( \text{a Grade rightmost (minor) column:} \)

[12] \( \text{Ll:GRADE} \)\( \text{CMAT}[1] \)

[13] \( \text{a Decrement column index and exit if done:} \)

[14] \( \text{L2:} \rightarrow (\text{IIO} \rho \text{I} \rho \text{I}-1) \rho 0 \)

[15] \( \text{a Grade next more major column:} \)

[16] \( \text{GRADE} \)\( \text{CMAT[GRADE;I]} \)

[17] \( \rightarrow \text{L2} \)

A more sophisticated technique packs several columns together at once so that fewer applications of grade-up (\( \downarrow \)) are required. For example, suppose you have a 9 column character matrix whose indices into the collating sequence are the following:

\[
\begin{array}{cccccccccc}
3 & 27 & 16 & 9 & 8 & 4 & 15 & 8 & 33 \\
31 & 30 & 19 & 9 & 10 & 8 & 24 & 2 & 23 \\
2 & 3 & 19 & 16 & 12 & 4 & 19 & 14 & 15 \\
\end{array}
\]

By grouping the matrix into 3 groups of 3 columns and by packing each group into 1 column by respectively multiplying its columns by 10000, 100 and 1 and adding, the result is:
Because of the nature of major-to-minor sorting and because of the scheme used to pack these numbers, you can then determine the grade vector of this 3 column matrix (third column first) and it will be the same as that of the 9 column matrix. The approach will probably be more efficient than the original approach because only 3 grade-up operations are needed rather than the original 9.

Taking this approach to its logical extreme, you may argue to pack all 9 columns into a single column (vector) of large numbers:

32716090804150833
313019091008240223
20319161204191415

However, the computer internally maintains only 16 or 17 digits of precision on any number. It sees the numbers as:

3271609080415083
3130190910082402
2031916120419141

Therefore, the last digit or two of these large packed numbers are insignificant to the computer when it is grading the vector and may produce incorrect grade indices for rows of the character matrix which are identical except in the last column or two.

So how many characters can be packed together at once? This is a function not only of the internal precision of your APL implementation but also of the length of the collating sequence. In the illustration above, the indices were packed by multiplying by consecutive powers of 100. Smaller powers (say 80) can be used to result in smaller packed numbers and to allow more columns to be packed at once. But if the powers used are too small, the indices will not always pack to distinct numbers.

For example, if the power 10 is used to pack the numbers 3 2 4 and 3 1 4, the results will be the same. This problem arises only if the range of indices is greater than the power used. Since the range of indices is one greater than the length of the collating sequence, that is the power you should use.

The following solution packs as many columns at once and performs as few grade-up operations as possible.
This solution is an improvement over the prior solution only if the packing operation is fast relative to the grade operation. The relative speeds differ among APL implementations and hardware configurations. You should time the two solutions for your implementation. Use the fastest, unless you are paid by the hour.

If you are familiar with the issue of comparison tolerance (system variable OCT), you are aware that APL systems typically do not distinguish between values which differ only beyond the 14th (or so) significant digit. Yet, here we are packing numbers out to 16 significant digits. We can do this because grade-up (↑) and grade-down (↓) are primitive functions which do not consider comparison tolerance (as do =, >, ≤, ∈, ≠, etc.) If your implementation of grade-up and grade-down does consider comparison tolerance, you should modify the above function to localize OCT in the header and to set OCT←0 (full precision) on the first line of the function.

Finally, APL implementations store small integer numbers more compactly than large integer numbers. Because of these differences in internal storage, grade-up is faster on small integers than on large ones. This difference may be so dramatic that you should pack fewer columns and do more grade-up operations on the small integer values. If so, you should change the reference to 1E16 in the above function to 2147483647 or 32767 or whatever your largest integer is (i.e. the largest number not stored as an 8 byte floating point number).
When working with character matrices which are wide and which have rows whose values are all significantly different (e.g. names), another solution to this problem becomes practical. The approach is to work with the columns in major-to-minor order.

Sort the first column. Compare the sorted characters to their neighbor (prior and next row) characters. If both neighbor characters are different, the row is distinct and belongs in its current (sorted) position. If one or both of its neighbors have the same value as it has, we must proceed to the second column. Consider the second column for only the rows whose value is not distinct for the first column. Sort the second column within the values of the first column. Compare the sorted characters to its neighbors and again identify the rows which are still not distinct for the first two columns. And so on.

Consider each successive column until all rows are known to be distinct or until you run out of columns. As fewer and fewer nondistinct rows remain, the grade-up operation will be performed on shorter and shorter vectors. Since the grade-up operation is quicker on short vectors than on long ones, this solution can be quite fast on matrices whose row values are mostly different.

[WSID: SORT]

\[
\begin{align*}
\text{\texttt{GRADE}} & \leftarrow \text{CS} \quad \text{CGRADEUP} \quad \text{CMAT};C;F;G;I;M;N;R;\text{ROWS} \\
[1] & \text{a Returns grade vector for sorting rows of} \\
[2] & \text{a CMAT with collating sequence CS}. \\
[3] & \text{a Index of last column as a scalar:} \\
[4] & N \leftarrow (\rho \text{CMAT})[1+\Pi IO]-\Pi IO \\
[5] & \text{a Return trivial result of no columns:} \\
[6] & \rightarrow (N=\Pi IO) \rho L1 \\
[7] & \text{GRADE} \leftarrow l \rho p \text{CMAT} \\
[8] & \rightarrow 0 \\
[9] & \text{a Select first column:} \\
[10] & L1;C \leftarrow \text{CMAT}[;I \Pi IO] \\
[11] & \text{a Convert characters to indices and grade them:} \\
[12] & \text{GRADE} \leftarrow A CS1C \\
[13] & \text{a Exit if 1 column or 1 or less rows:} \\
[14] & \rightarrow ((I=N) v l \geq l \rho p \text{CMAT}) \rho 0 \\
[15] & \text{a Sort characters:} \\
[16] & C \leftarrow C[\text{GRADE}] \\
[17] & \text{a Flag first of groups of equal values:} \\
[18] & F \leftarrow C \neq l \rho C \\
[19] & \text{a Handle incorrect result if all values equal:} \\
[20] & F[\Pi IO] \leftarrow 1 \\
[21] & \text{a Flag values still unresolved (i.e. more than} \\
[22] & \text{a 1 equal value):} \\
[23] & M \leftarrow F \neq l \rho F \\
[24] & \text{a Squeeze down flag-first vector:} \\
[25] & F \leftarrow M/F \\
[26] & \text{a Exit if none left to resolve:} \\
[27] & \rightarrow (\rho F) \downarrow 0 \\
[28] & \text{a Indices into GRADE of unresolved values:} \\
[29] & \text{ROWS} \leftarrow M/l \rho M
\end{align*}
\]
\noop (continued)
\[30\] a Indices into CMAT of unresolved values:

\[31\] LOOP: R \rightarrow GRADE[ROWS]

\[32\] a Increment column index:

\[33\] I \leftarrow I + 1

\[34\] a Select Ith column for unresolved rows:

\[35\] C \leftarrow CMAT[R;I]

\[36\] a Convert and grade characters:

\[37\] G \leftarrow ACSiC

\[38\] a Reorder grade vec to maintain sorted prior columns:

\[39\] G \leftarrow G[\phi (+\phi)[G]]

\[40\] a Insert reordered grade vec:

\[41\] GRADE[ROWS] \leftarrow R[G]

\[42\] a Exit if no more columns:

\[43\] \rightarrow (I \geq N) \rho 0

\[44\] a Sort characters:

\[45\] C \leftarrow CG]

\[46\] a Flag first, considering prior columns too:

\[47\] F \leftarrow FvC \neq 1\phi C

\[48\] a Flag values still unresolved:

\[49\] M \leftarrow F \# 1\phi F

\[50\] a Squeeze down flag-first vector:

\[51\] F \leftarrow M/F

\[52\] a Exit if none left to resolve:

\[53\] \rightarrow (\rho F) \downarrow 0

\[54\] a Squeeze down unresolved indices into GRADE:

\[55\] ROWS \leftarrow M/ROWS

\[56\] \rightarrow LOOP

\v

~~~~~~~~~~~~~~~

PROBLEM: Sort the following character matrix, SUBJECTS.

Lincoln
troops
liberty
lasting
brothers
Grant
Lee
Chapter 5

TOPIC: Uppercase/Lowercase Sorting

Since this matrix contains both uppercase and lowercase letters, the collating sequence must contain both the uppercase and lowercase alphabets. Let's try catenating them:

```plaintext
CS' ABCDEF...XYZabcdef...xyz'
SUBJECTS[CS\SUBJECTS;]
```

No good. Words beginning with the letter L should be together, whether the L is uppercase or lowercase. Let's try interleaving the uppercase and lowercase alphabets:

```plaintext
CS' AaBbCcDcEeFf...XxYyZz'
SUBJECTS[CS\SUBJECTS;]
```

Not quite. Although words beginning with the letter L are now together, those beginning with an uppercase L precede those beginning with a lowercase l, regardless of the second letter in each word. We want all Ls to be treated equally, regardless of case.

Since equality is our aim, let us promote each lowercase letter to an uppercase letter and try again. Suppose UPPERCASE is a monadic function which converts its character array argument to an array of the same shape and values except each lowercase letter has been replaced by the corresponding uppercase letter. Then, lowercase letters can be omitted from the collating sequence. The following solution does the job.

```plaintext
CS' ABCDEF...XYZ'
SUBJECTS[CS\UPPERCASE SUBJECTS;]
```
Notice that the character matrix is converted to uppercase letters for purposes of grading only. The original mixed case matrix is used for indexing.

The technique of converting an array to uppercase letters is also useful for searching through mixed case arrays whenever the uppercase and lowercase characteristics of letters are to be ignored. For example, to list the words which begin with "LI":

```
(.UPPERCASE SUBJECTS[;1 2])^.=.'LI')#SUBJECTS
Lincoln
liberty
```

The following function will perform the desired translation to uppercase letters.

```
[WSID: SORT]
\[\]
R←UPPERCASE C;FOUND;IND;LOWER;UPPER
[1] R Converts the lowercase letters in the character array C into the corresponding uppercase letters. Useful for sorting or searching character arrays when the case distinction is to be ignored.
[2] LOWER←'abcdefghijklmnopqrstuvwxyz'
[3] UPPER←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
[4] R←,C
[5] IND←LOWER R
[6] FOUND←IND<0IO+ρLOWER
[7] R←(0FOUND/,R)←UPPER[FOUND/IND]
[8] R←(ρC)ρR
[9] Alternate algorithm...
[10] DAV←DAV
[11] DAV←DAV\LOWER←UPPER
[12] (PAV\DAV\LOWER)←UPPER
[13] PAV←DAV\C
[14] PAV←DAV\C
```

Implementations of APL which provide dyadic grade-up typically also provide a facility for handling this uppercase/lowercase problem directly. Specifically, the collating sequence left argument may be
a matrix which contains both alphabets as two corresponding rows. For example:

```r
CS<- 'ABCDEF...,XYZ',[0.5]' abcdef...,xyz'
SUBJECTS[CS|SUBJECTS;]
brothers
Grant
lasting
Lee
liberty
Lincoln
troops
```

You should read your documentation for dyadic grade-up to understand the subtleties of this facility. Given the facility, the UPPERCASE function is not needed for sorting uppercase/lowercase character matrices. However, it is still useful for searching through uppercase/lowercase arrays.

---

**PROBLEM:** Given the policy numbers of 1000 existing policyholders (as a 1000 row, 12 column character matrix since the policy "numbers" may contain letters) and the policy numbers of 600 non-smokers (600 rows, 12 columns), determine the index of the existing policy (1 to 1000) to which each of the 600 non-smokers corresponds. The resulting integer vector will have 600 elements. Return the "index" 1001 if the policy number is not found.

**TOPIC:** Array Searching

If the two lists of policy numbers were numeric vectors rather than character matrices, the solution would be trivial. Suppose BASE is the name of the 1000 element vector of existing policy numbers, VALS is the name of the 600 element vector of non-smoker policy numbers and INDS is the desired result. The following expression will solve the problem:

```r
INDS<-BASE|VALS
```

(Note that the default comparison tolerance, i.e. DCT, will need to be reduced to make accurate comparisons between numbers with more than 14 or so digits.)
In APL implementations which support nested arrays, the dyadic \( \downarrow \) function may be used to solve this problem, even on character matrices. The first step is to convert the matrix arguments to nested vector arguments (e.g. \( \langle 2 \rangle \)BASE in APL2). The solution then follows directly:

\[
\text{INDS}\leftarrow<\langle 2 \rangle \text{BASE}\rangle\downarrow<\langle 2 \rangle \text{VALS}\rangle
\]

(in APL2)

\[
\text{INDS}\leftarrow\downarrow<\langle 2 \rangle \text{BASE}\rangle\downarrow<\langle 2 \rangle \text{VALS}\rangle
\]

(in APL*PLUS)

\[
\text{INDS}\leftarrow<\langle \& 1 \rangle \text{BASE}\rangle\downarrow<\langle \& 1 \rangle \text{VALS}\rangle
\]

(in SHARP APL)

In APL implementations which do not support nested arrays, a more creative approach is required since the dyadic \( \downarrow \) function does not operate as hoped for on character matrix lists.

One effective approach is to convert the character matrix arguments into numeric vector arguments by converting the columns of characters into columns of indices and then packing the numbers together by the techniques of the previous sections. As mentioned there, only a limited number of characters may be packed into a single number without losing precision (say 8 to 12 character columns, depending upon the length of the character vector collating sequence used to convert the characters to indices). Further, since dyadic \( \downarrow \) uses comparison tolerance, the value of \( \text{OCT} \) should be set to zero to make comparisons which are as precise as possible.

The following function uses this technique to emulate dyadic \( \downarrow \) on character matrices:

\[
\text{INDS}\leftarrow\langle 2 \rangle \text{BASE}\rangle\downarrow\langle 2 \rangle \text{VALS}; \text{CS}; P; \text{OCT}
\]

[WSID: SEARCH]

\[
\begin{align*}
1 & \text{ A Returns the row indices of } \text{BASE} \text{ at which the} \\
2 & \text{ A rows of } \text{VALS} \text{ first match.} \\
3 & \text{ A Set comparison tolerance to maximum precision:} \\
4 & \text{ OCT}\leftarrow 0 \\
5 & \text{ A Determine collating sequence:} \\
6 & \text{ CS}\leftarrow((\text{DAV}\in \text{BASE})\lor \text{DAV}\in \text{VALS})/\text{DAV} \\
7 & \text{ A Packing factor:} \\
8 & \text{ P}\leftarrow 1+\rho \text{CS} \\
9 & \text{ A Pack and search:} \\
10 & \text{ IND}\leftarrow\langle 2 \rangle (\text{CS}\& \text{BASE})-\& 0)\downarrow \langle 2 \rangle (\text{CS}\& \text{VALS})-\& 0
\end{align*}
\]

Unfortunately, this technique will not work on wide character matrices. Further, in some APL implementations, the decode \( \downarrow \) function is slow. Under either of these conditions, another approach is desired.

Suppose \( \text{BASE} \) is a numeric vector and \( \text{VAL} \) is a numeric scalar. How can we find the index of the first occurrence of \( \text{VAL} \) in \( \text{BASE} \)?

\[
\text{BASE}\downarrow\text{VAL}
\]
How can we identify (by bits) all of the occurrences of VAL in BASE?

BASE=VAL

If VALS is a vector, how can we identify (by bits) all of the occurrences of VALS in BASE?

BASE⋄.=VALS

If BASE is a character matrix and VAL is a character vector (i.e. one policy number), how can we identify (by bits) all of the occurrences of VAL in BASE?

BASE∧.=VAL

If VALS is also a character matrix, how can we identify (by bits) all of the occurrences of VALS in BASE?

BASE∧.=⋄VALS

The information we seek is contained in the Boolean matrix result of this expression. Specifically, the column index of the first bit in each row is the index we seek. By using a Boolean scan, we can extract the indices:

INDS⋄IO+/>∧\~BASE∧.=⋄VALS

We will modify this algorithm somewhat to replace row-wise (e.g. ∧\) functions by the usually faster column-wise (e.g. ^\ ) functions and to eliminate the not (~) function.

This algorithm is an excellent illustration of the power of APL. Unfortunately, it has some drawbacks. For this example, the result of the v.¬ function is a 600 row, 1000 column Boolean matrix (600,000 elements) which might generate a WS FULL error message. Even if it does work, the function will consume a large amount of CPU time while making the 600,000 comparisons.

A different approach takes advantage of the speed of sorting algorithms. The following discussion assumes IDO=1.

1. Combine the two arguments via catenation into a 1600 row matrix:

A=BASE,[1]VALS
Chapter 5 SORTING AND SEARCHING

2. Sort the combined matrix using the CGRADEUP function developed in a previous section (or using dyadic ▲ if available) and using OAV as the collating sequence:

    GRADE←OAV CGRADEUP A
    A←A[GRADE;]

By sorting the matrix, like rows are now contiguous.

3. Shift the rows of the matrix down one row and compare:

    FLAG←∧/A≠1eA

FLAG is a 1600 element Boolean vector whose 1s flag the first of each set of contiguous like-valued rows.

4. For each of the 1600 rows, determine the index into the original unsorted catenated matrix of the first row of each set of contiguous like-valued rows:

    FIRST←(FLAG/GRADE)[+\FLAG]

The 1600 elements of FIRST correspond to the rows of the sorted catenated matrix.

5. Reorder the elements so they correspond to the rows of the unsorted catenated matrix:

    INDS←(FIRST)p0
    INDS[GRADE]←FIRST

6. Select only those elements of INDS which correspond to the rows of VALS (not the rows of BASE):

    L←1↑pBASE
    INDS←L↓INDS

7. Set the elements of INDS which correspond to rows of VALS for which no matching row was found in BASE to the "not found" index (1 plus the number of rows in BASE):

    INDS←INDS\1+L

This approach is quite efficient for large arguments. However, because it requires so many steps, other algorithms may be more efficient for small arguments. In particular, inner product (▲.=) is typically quite fast when one argument is a matrix and the other is a vector. Therefore for a small (few rows) right argument of CMIOTA, it may actually be faster to loop on the rows of the right argument (using ▲.= to search through the rows of the matrix left argument) than to employ this catenating, sorting, shifting, comparing algorithm.
But how small should the right argument to CMIOTA be before we switch to a looping algorithm? Let us assume the arguments to CMIOTA are L and R:

\[ I+L \text{ CMIOTA } R \]

The CPU time consumed by the looping algorithm increases linearly with the number of rows in R (for a constant L) and linearly with the number of rows in L (for a constant R). Therefore, the CPU time consumed will be a function of the formula:

\[ \text{CPUL} = C_1 + (RR \times (C_2 + (C_3 \times RL))) \]

where CPUL is the amount of CPU time consumed by the looping algorithm, RR and RL are the number of rows in R and L respectively, and C1, C2 and C3 are constants to be determined.

The CPU time consumed by the sorting algorithm increases linearly with the sum of the numbers of rows in R and in L. Therefore, the CPU time consumed will be a function of the formula:

\[ \text{CPUS} = C_4 + (C_5 \times (RR + RL)) \]

where CPUS is the amount of CPU time consumed by the sorting algorithm and C4 and C5 are constants to be determined.

The values of C1, C2, C3, C4 and C5 for the formulas above will depend upon the particular machine and APL implementation. To determine them for your environment, you must time the two algorithms for a variety of arguments and then use the techniques of least squares to find the constants which define the "best" curves to fit the empirical data. There is a problem at the end of the chapter on Computer Efficiency Considerations which performs the first task and a problem at the end of the chapter on Curve Fitting which performs the second task. Work these problems and plug the derived values into the CMIOTA function below (in place of C1, C2, C3, C4, C5).

(The formulas above do not consider the number of columns in the matrix arguments nor the nature of the data, i.e. whether and where the values are found. Therefore they are not precise formulas. However, they will be sufficiently accurate to insure that the best algorithm is used in all but borderline cases.)

The following CMIOTA function uses the approaches discussed above and has been extended to handle origin 0 and to treat the trivial cases (empty or 1-row arguments) separately.
\( \text{INDS} \leftarrow \text{BASE} \times \text{VALS}; \text{A}; \text{F}; \text{G}; \text{I}; \text{L} \)

1. Returns the row indices of BASE at which the rows of VALS first match.
2. Branch if right arg a matrix:
3. \( (2=\rho \text{VALS}) \) \( \rho \text{L1} \)
4. Handle vec or scalar right arg:
5. \( \text{INDS} \leftarrow (\text{BASE} \times = \text{VALS}) \) \( \rho \text{L1} \)
6. \( \to 0 \)
7. \( \text{L1}: \text{INDS} \leftarrow (\rho \text{BASE})[\rho \text{L1}] \)
8. \( \text{A} \leftarrow (\rho \text{VALS})[\rho \text{L1}] \)
9. Branch unless no rows in either arg:
10. \( (\times \text{F} + \text{A} \times \text{L}) \rho \text{L2} \)
11. Handle empty arg:
12. \( \text{INDS} \leftarrow \text{A} \rho \text{LIO} \)
13. \( \to 0 \)
14. Branch if both args have more than 1 row:
15. \( \text{L2}: (\times \text{F} \neq 1) \rho \text{L4} \)
16. Branch unless left arg has 1 row:
17. \( (\times \text{L} \neq 1) \rho \text{L3} \)
18. Handle 1 row left arg:
19. \( \text{INDS} \leftarrow \text{LIO} + \text{VALS} \times \# , \text{BASE} \)
20. \( \to 0 \)
21. Handle 1 row right arg:
22. \( \text{L3}: \text{INDS} \leftarrow (\text{BASE} \times = \text{VALS}) \) \( \rho \text{L1} \)
23. \( \to 0 \)
24. Branch if sort alg. costs more than looping alg.:
25. \( \to (\times \text{C} \times \text{C} \times \text{C} \times \text{C} \times \text{C} \times \text{L}) \rho \text{L5} \)
26. \( \to (\times \text{C} \times \text{C} \times \text{C} \times \text{C} \times \text{C} \times \text{L}) \rho \text{L5} \)
27. Combine args. and sort (like values together)
28. \( \text{INDS} \leftarrow \text{LIO} + \text{VALS} \times \# , \text{BASE} \)
29. \( \to 0 \)
30. (use CGRADEUP if dyadic && unavailable):
31. \( \text{G} \leftarrow \text{A} \times \text{A} \times \text{BASE} ;[\rho \text{L1}] \text{VALS} \)
32. \( \text{A} \leftarrow \text{A} \times \text{L} \times \text{G} \)
33. Flag 1st of distinct rows by shifting and comparing:
34. \( \text{F} \leftarrow \text{A} \times \# = 1 \text{e} \text{A} \)
35. Insure 1st elt is 1 (in case all rows the same):
36. \( \text{F} \leftarrow \text{LIO} \) \( + 1 \)
37. Indices of 1st distinct rows:
38. \( \text{I} \leftarrow \text{F} \times \text{G} \)
39. Replicate for each like row:
40. \( \text{F} \leftarrow \text{LIO} \) \( + \text{LIO} \)
41. \( \text{I} \leftarrow \text{I} \) \( + \text{D} \text{F} \)
42. Unsort indices (to catenated order):
43. \( \text{INDS} \leftarrow \text{I} \)
44. \( \text{INDS} \times \) \( \text{G} \times \) \( \text{L} \leftarrow \text{INDS} \)
45. Keep those corresponding to right arg:
46. \( \text{INDS} \leftarrow \text{L} \) \( \text{L} \) \( \text{INDS} \)
47. Set 'not found' inds to 'one greater':
48. \( \text{INDS} \leftarrow \text{INDS} \times \text{L} \times \text{LIO} \)
49. \( \to 0 \)
50. Use looping algorithm if more efficient:
51. \( \text{L5}: \text{INDS} \leftarrow \text{A} \leftarrow 0 \)
52. \( \text{L} \leftarrow (\text{A} \rho \text{L6}) \) \( , 0 \)
Chapter 5	 SORTING AND SEARCHING

\[\text{\textbackslash cmiota (continued)}\]

\[\text{\textbackslash l6:inds[i]=base^[vals[i]]; i=1}\]

\[\rightarrow \text{L[i+1]}\]

\[\text{\textbackslash v}\]

\[\text{PROBLEM: Suppose you have a 1000 element vector of ages. You wish to group the ages into the 10 ranges: 0 to 9, 10 to 19, 20 to 24, 25 to 29, 30 to 34, 35 to 39, 40 to 49, 50 to 59, 60 to 64, 65 and up. What approach would you take to translate these 1000 ages into 1000 corresponding range indices (i.e. numbers between 1 and 10)?}\]

\[\text{TOPIC: Range Searching}\]

Suppose the 1000 element vector of ages is named AGES. Let us define a 10 element vector LOWER of the lower limits for the specified ranges:

\[\text{LOWER=0 10 20 25 30 35 40 50 60 65}\]

We need to compare each element of AGES to each element of LOWER and to determine the index into LOWER of the last element which is less than or equal to the element of AGES. Outer product may be used to solve this directly:

\[\text{INDS+/AGES,LOWER}\]

This expression is simple and powerful but suffers from the malady of all outer product solutions. Since every element of the left argument is being compared to every element of the right argument, the number of comparisons increases exponentially as the lengths of the two arguments increase linearly. Hence, the solution is slow and expensive when performed on two long vectors.

A more efficient (for long arguments) algorithm can be developed using the same sorting technique employed in the prior section.

1. Combine the two arguments and determine the grade vector:

\[\text{A=LOWER,AGES} \]
\[\text{GRADE=}\text{A}\]
2. Rather than reorder the elements of the catenated array, reorder the elements of an array of 1s and 0s where the 1s mark elements of LOWER and the 0s mark elements of AGES:

\[
\text{FLAG} \leftarrow (\rho A \uparrow (\rho \text{LOWER}) \rho 1)[\text{GRADE}]
\]

3. Determine the index into LOWER of each element of the sorted catenated array:

\[
\text{FIRST} \leftarrow +\backslash \text{FLAG}
\]

4. The elements of FIRST correspond to the elements of the sorted catenated array. Reorder the elements so they correspond to the elements of the unsorted catenated array:

\[
\text{INDS} \leftarrow (\rho \text{FIRST}) \rho 0 \\
\text{INDS}[\text{GRADE}] \leftarrow \text{FIRST}
\]

5. Select only those elements of INDS which correspond to the elements of AGES (not the elements of LOWER):

\[
\text{INDS} \leftarrow (\rho \text{LOWER}) \downarrow \text{INDS}
\]

The following function LIOTA uses this approach but is extended to handle origin 0 and to return 1 greater than the largest index if the corresponding value is less than the smallest lower limit. The left argument is assumed to be in ascending order.

```lisp
[V INDS←LOWER LIOTA VALS;A;F;G;I;L
```

[WSID: SEARCH]
The solutions presented here are oriented around lower limits of ranges (in ascending order). If upper limits are considered (e.g. \( \text{UPPER} = 9 19 24 29 34 39 49 59 64 99 \)), the solutions must be modified accordingly:

Lower limits (ascending): \( +/\text{AGES} \geq \text{LOWER} \)

or: \( \text{LOWER LIOTA AGES} \)

Upper limits (ascending): \( 1++/\text{AGES} > \text{UPPER} \)

or: \( \text{UPPER UIOTA AGES} \)

Lower limits (descending): \( 1++/\text{AGES} < \text{LOWER} \)

Upper limits (descending): \( +/\text{AGES} \leq \text{UPPER} \)

The following function \( \text{UIOTA} \) works like \( \text{LIOTA} \) but requires a vector left argument of range upper limits in ascending order.

\[
\text{v IND} \leftarrow \text{UPPER UIOTA VALS;A;F;G;I;L} \\
\text{1 Returns the indices of UPPER at which the} \\
\text{2 elements of VALS last match or are less than.} \\
\text{3 Branch unless right argument empty:} \\
\text{4} \rightarrow (\times \rho \text{VALS}) \rho \text{L1} \\
\text{5} \text{INDS} \leftarrow \text{0} \\
\text{6} \rightarrow \text{0} \\
\text{7} \text{Combine arguments and sort:} \\
\text{8} \text{L1:G} \leftarrow \text{A} \leftarrow \text{VALS,UPPER} \\
\text{9} \text{Flag elements from UPPER in sorted array:} \\
\text{10} \text{L} \leftarrow \rho \text{UPPER} \\
\text{11} \text{F} \leftarrow (\rho \text{A} \leftarrow \text{Lp1})[\text{G}] \\
\text{12} \text{Determine indices into UPPER (origin dependent):} \\
\text{13} \text{I++'1↓0I0,F} \\
\text{14} \text{Unsort indices (to catenated order):} \\
\text{15} \text{INDS} \leftarrow \text{I} \\
\text{16} \text{INDS[Г]} \leftarrow \text{I} \\
\text{17} \text{Keep those corresponding to right argument:} \\
\text{18} \text{INDS} \leftarrow (\rho \text{VALS}) \rho \text{INDS} \\
\text{v}
\]

You should be aware that the \( \text{LIOTA} \) and \( \text{UIOTA} \) functions may not produce correct results when operating on floating point numeric vectors whose values are approximately equal (within comparison tolerance) to elements in the lower limit or upper limit vector. This aberration occurs because the grade-up (\( \downarrow \)) function used in \( \text{LIOTA} \) and \( \text{UIOTA} \) does not consider comparison tolerance when sorting. Thus, two numbers which would be treated as equal by the relational functions (say \( \geq \) or \( > \)) are treated as distinctly different numbers by grade-up. If this is a likely problem for a particular application, you should use the appropriate outer product solution.

Let's consider an alternate algorithm for solving this range searching problem. The algorithm involves "ranking vectors". The
ranking vector of a vector V is computed via \( \triangleup \) and indicates the relative magnitudes of the values of V. The smallest value in V is assigned the index 1 (in origin 1), the second smallest the index 2, the third 3 and so on. For example,

\[
\begin{array}{cccccc}
\triangleup & 15 & 5 & 10 & 15 & 20 \\
3 & 1 & 2 & 4 & 5
\end{array}
\]

Notice that in the event of ties, the earlier values receive the lower rankings. In this example, the first 15 is ranked 3rd and the next is ranked 4th.

Consider what happens to the rankings of these values when more values are catenated to the vector. For example,

\[
\begin{array}{cccccccc}
\triangleup & 15 & 5 & 10 & 15 & 20 & 13 & 17 \\
4 & 1 & 2 & 5 & 7 & 3 & 6
\end{array}
\]

Notice that the corresponding rankings (4 1 2 5 7) have increased by the number of catenated values which are less than the respective values.

\[
\begin{array}{ccccccc}
4 & 1 & 2 & 5 & 7 & 3 & 6 \\
1 & 0 & 0 & 1 & 2
\end{array}
\]

That is, no catenated values are less than the 5 or 10; one catenated value (13) is less than the two 15s; and two catenated values (13 17) are less than the 20.

Consider what happens when some of the catenated values are equal to values in the original vector. For example, catenating 13 15 instead of 13 17:

\[
\begin{array}{cccccccc}
\triangleup & 15 & 5 & 10 & 15 & 20 & 13 & 15 \\
4 & 1 & 2 & 5 & 7 & 3 & 6
\end{array}
\]

We get the same result. However, notice what happens when the catenated values are placed at the front of the vector:

\[
\begin{array}{cccccccc}
\triangleup & 13 & 15,15 & 5 & 10 & 15 & 20 \\
3 & 4 & 5 & 1 & 2 & 6 & 7 \\
\end{array}
\]

The result now indicates the number of catenated values which are less than or equal to each value, not just less than.

Given this behavior, the LIOTA and UIOTA algorithms follow directly:

LIOTA: \( \text{INDS} \leftarrow (\rho \text{LOWER}) \downarrow \triangleup \text{LOWER,AGES}) - \triangleup \text{AGES} \)

UIOTA: \( \text{INDS} \leftarrow 1 + ((\rho \text{AGES}) \uparrow \triangleup \text{AGES,UPPER}) - \triangleup \text{AGES} \)
These algorithms produce correct results for origin 1. The LIOTA algorithm returns 0 (instead of 1+pLOWER) for values of AGES which are less than the smallest value in LOWER. The two functions listed below, LIOTAI and UIOTAI, work like the LIOTA and UIOTA functions above but use these ranking vector algorithms. The algorithms have been modified to work correctly in either origin and to return the correct "not found" value (UIO+pLOWER).

Further, a more efficient method for computing the ranking vector is employed. When sorting a grade vector, traditional sorting logic is not needed. Index assignment will suffice. The following four sets of expressions generate equivalent results:

\[
\begin{align*}
R & \leftarrow G \\
G & \leftarrow G - R \\
R \leftarrow R + G \\
R[G] & \leftarrow RG
\end{align*}
\]

The last set of expressions is the most efficient. Since it is not as clear as the first expression, it should include a comment:

\[
\begin{align*}
& \text{a } R \leftarrow AV \\
& \text{R} \leftarrow G \leftarrow AV \\
& \text{R[G]} \leftarrow RG
\end{align*}
\]

Using this technique, the LIOTAI and UIOTAI functions each perform only two grade-up operations instead of four. However, the LIOTA and UIOTA functions above each perform only one grade-up operation and so will typically be the faster functions. Time them in your APL implementation.

\[
\begin{align*}
\text{v} & \text{ IND}S \leftarrow \text{LOWER LIOTAI VALS;G;L;R;S} \\
& \text{[1] a Returns the indices of LOWER at which the} \\
& \text{[2] a elements of VALS first match or exceed.} \\
& \text{[3] L} \leftarrow \text{pLOWER} \\
& \text{[4] a } R \leftarrow AVALS : \\
& \text{[5] R} \leftarrow G \leftarrow AVALS \\
& \text{[6] R[G]} \leftarrow RG \\
& \text{[7] a } S \leftarrow LLOWERS,VALS : \\
& \text{[8] S} \leftarrow G \leftarrow LLOWERS,VALS \\
& \text{[9] S[G]} \leftarrow RG \\
& \text{[10] a Origin 1 indices:} \\
& \text{[11] IND}S \leftarrow (\text{L} \leftarrow S) - R \\
& \text{[12] a Set 'not found' indices to 'one greater':} \\
& \text{[13] IND}S((\text{IND}S=0)) \leftarrow \text{L} \leftarrow \text{IND}S \leftarrow L+1 \\
& \text{[14] a Change from origin 1 to origin 0 if needed:} \\
& \text{[15] a } \text{UIO} \leftarrow 0 \\
& \text{[16] IND}S \leftarrow \text{IND}S - 1
\end{align*}
\]
Chapter 5

SORTING AND SEARCHING

[WSID: SEARCH]

\[ \text{INDS} \leftarrow \text{UPPER} \cup \text{U} \text{O} \text{T} \text{A} \text{1} \ \text{VAL} \text{S} ; G ; R ; S \]

1. Returns the indices of UP\text{PER} at which the
2. elements of VAL\text{S} last match or are less than.
3. \text{R} \leftarrow \text{G} \uparrow \text{A} \text{V} \text{A} \text{L} \text{S}
4. \text{R} \leftarrow G \uparrow \text{A} \text{V} \text{A} \text{L} \text{S}
5. \text{R}[G] \leftarrow \text{G} \leftarrow \text{G}
6. \text{S} \leftarrow \text{G} \uparrow \text{A} \text{V} \text{A} \text{L} \text{S}, \text{UP} \text{PER}
7. \text{S} \leftarrow \text{G} \uparrow \text{A} \text{V} \text{A} \text{L} \text{S}, \text{UP} \text{PER}
8. \text{S}[G] \leftarrow \text{G}
9. \text{Origin } 0 \text{ indices:}
10. \text{INDS} \leftarrow ((\rho \text{VALS}) \rho S) - \text{R}
11. \text{Change from origin } 0 \text{ to origin } 1 \text{ if needed:}
12. \rightarrow \text{IOJO}
13. \text{INDS} \leftarrow \text{INDS} + 1

\[ \text{INDS} \leftarrow \text{INDS} + 1 \]

PROBLEM: Write a function named \text{ASS} (string search) which will locate every occurrence of a character vector substring (right argument) in a character vector (left argument). The result is a Boolean vector of the same length as the left argument whose 1s flag the indices at which each match begins. For example:

\[ 'T H I S \ \text{A } T E S T' \ \text{ASS} 'I S' \]

\[ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \]

TOPIC: Character Substring Searching

Some APL implementations have primitive functions which solve this problem directly:

\[ \text{APL*PLUS:} \]

\[ \uparrow \text{BIT} \leftarrow \text{CVEC} \ \text{ASS} \ \text{SUB} \]

\[ [1] \ \text{BIT} \leftarrow \text{CVEC} \ \text{ASS} \ \text{SUB} \]

\[ \uparrow \]
Chapter 5  
SORTING AND SEARCHING

APL2:

\[ V \text{BIT} \rightarrow \text{CVEC} \triangleq \text{SS SUB} \]
[1] \( \text{BIT} \rightarrow \text{SUB} \triangleq \text{CVEC} \)
\[ V \]

If such a primitive function is unavailable to you, you must work a little to get the desired result:

\[ V \text{BIT} \rightarrow \text{CVEC} \triangleq \text{SS SUB} \; ; \; C ; \; S \]

[1] A Returns bit vector of length \( \rho \text{CVEC} \) with 1s
[2] A flagging starts of substrings which match \text{SUB}.
[3] C\( \triangleq \rho \text{CVEC} \)
[4] S\( \triangleq \rho , \text{SUB} \)
[5] BIT\( \triangleq \text{C} \uparrow (-\text{S}) \downarrow \text{SUB} \).\( = \text{(S} , \text{C} \uparrow \times \text{C}) \rho \text{CVEC} \)
\[ V \]

~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~

PROBLEM: Write functions REPLACE and \text{BY} which will replace all occurrences of a character vector substring in a character vector by a second substring. For example:

'THIS IS A TEST' REPLACE 'IS' BY 'ARE'
THARE ARE A TEST

Use the function \( \triangleq \text{SS} \) defined in the prior section.

TOPIC: Character Substring Replacement

The \text{BY} function is used simply as a syntactic convenience to provide three arguments to the REPLACE function. One approach is to assign the right argument to a global variable (say \(<\text{by}>\)) and to return the left argument as the explicit result.

\[ V \text{R} \rightarrow \text{A BY B} \]
[1] A Used in conjunction with REPLACE as:
[2] A
[3] A 'THIS IS A TEST' REPLACE 'IS' BY 'ARE'
[4] A
[5] by\( \rightarrow \text{B} \)
[6] R\( \rightarrow \text{A} \)
\[ V \]

-76-
The REPLACE function will generate a result by analyzing its two arguments and its third global "argument" <by>. When done, REPLACE erases <by> so that it will not be left global.

The approach taken by REPLACE is the following:

1. Use ΔSS to find the occurrences of the old substring in the character vector. Convert the bits to indices.

2. Create a replication vector (i.e. left argument to /) which can be used to both squeeze out the old substring and to allow room for the new substring. Perform the replication on the character vector.

3. Since the length of the character vector has changed (unless the new substring has the same length as the old substring), adjust the indices computed in step 1 to point to where the new substrings must be inserted.

4. Insert the new substrings.

```plaintext
[WSID: SEARCH]

V NVEC+OVEC REPLACE SUB;BIT;IND;NHITS;REP;SIZE;OIO
[1] a Replaces all occurrences of SUB in OVEC by <by> (set
[2] a in BY), erases <by> and returns the modified OVEC.
[3] a Requires subfn ΔSS (or DSS or $).
[4] a The logic is a bit simpler using origin 0:
[5] OIO<0
[6] a Locate the starts of the old substring:
[7] BIT+OVEC ΔSS SUB
[8] a Convert the bits to indices:
[9] NHITS<IND<BIT/"BIT
[10] a Initialize replication vector as 1s:
[11] REP<(BIT)<1
[12] a Insert 0s where old substrings are:
[13] REP[IND]+"BIT,SUB)<0
[14] a Insert new substring length where new substrings
[15] a will begin:
[16] REP[IND]+SIZE<BIT,by
[17] a Squeeze and expand OVEC with replicate:
[18] NVEC+REP/OVEC
[19] a Adjust old indices to get new indices:
[20] IND<IND+(SIZE<BIT,SUB)<1NHITS
[21] a Insert new substrings:
[22] NVEC[IND]+"SIZE,LENGTH<1NHITS,SIZE<by
[23] a Erase <by>:
[24] BIT+DEX 'by'
```

```
1. Given a 3 column integer matrix PNUM of telephone numbers (area code, phone number, extension, e.g. 213 5550123 1234), how can you sort the numbers in ascending order?

2. Modify the CMIOTA function described in this chapter to define a function IOTA which works on numeric vector arguments instead of character matrix arguments. Test it in your APL implementation. Which is faster, IOTA or dyadic \( \_ \) (see Computer Efficiency Considerations chapter)? What is the consequence of the dependence of dyadic \( \_ \) on OCT (comparison tolerance) and the independence of \( \_ \) on OCT?

3. Given a 3 column integer matrix PNUM of telephone numbers (area code, phone number, extension) and a 3 element integer vector P which represents a particular telephone number, determine the index of the first row of PNUM in which P is located.

4. Using the LIOTA function developed in this chapter, determine in which salary grouping each of the elements of the vector SALARY belong. The groupings are: (1) 1000 to 9999; (2) 10,000 to 19,999; (3) 50,000 to 69,999; (4) 100,000 and up. Return the index 5 for elements of SALARY in none of these groupings.

5. Using the ASS function developed in this chapter, write a monadic function DEB which will delete extraneous (leading, trailing or contiguous) blanks from its argument and will return the compressed result. For example:

   DEB ' TOO MANY SPACES. '  
   TOO MANY SPACES.

6. In a numeric vector NVEC, the value \(-1\) represents "unknown". Display NVEC, showing each occurrence of \(-1\) as the characters 'N/A' (not applicable).
7. Suppose you have a 25 column character matrix of employee names, ENAMES. Each row contains one name, left-justified. The names contain both uppercase and lowercase letters. Display the names which contain the string "son" anywhere in the name.
This chapter deals with the task of data selection in APL. Selection is the process of extracting elements from an APL array. The reverse process, replacing the values of elements within an array, or selection assignment, is also considered. Finally, a special selection task is covered: the task of selecting those values in an array which are unique (or distinct).

PROBLEM: Given a three element vector NVEC, what APL expression will return the first two elements? What expression will replace these elements in NVEC by the values 10 and 20?

TOPIC: Selection and Selection Assignment

There are basically 3 selection techniques available in APL.

1. Indexing. Use indexing ([ ]) when you know the positions within the array of the elements to be selected. For example (in origin 1):

   NVEC[1 2]

2. Take/drop. Use take (↑) or drop (↓) or both when the elements to be selected are contiguous, especially at the start or end of the array. For example:

   2↑NVEC
   -1↓NVEC
3. Compression. Use compression (/) when you have a corresponding Boolean compression vector whose ones flag elements to be selected. For example:

```
1 1 0/NVEC
```

Typically, the compression vector is the result of a relational or logical expression which defines some criteria by which elements are to be selected.

Though there are 3 selection techniques, the only selection assignment technique available in APL is index assignment. For example:

```
NVEC[1 2]+10 20
```

If the nature of the selection assignment problem is oriented more toward take/drop or compression logic, you must convert the selection values to indices so that you may use index assignment. For example:

```
NVEC[2↓lρNVEC]←10 20
NVEC[¬1↓lρNVEC]←10 20
NVEC[1 1 0/lρNVEC]←10 20
```

It is because of this need to convert to indices when performing selection assignment that the APL idioms lρ and /lρ are so common.

In APL2, the APL language has been extended to allow direct selection assignment without first converting to indices. For example:

```
NVEC[1 2]+10 20
(2↑NVEC)←10 20
(¬1↓NVEC)+10 20
(1 1 0/NVEC)+10 20
```

In fact, fairly complex selection assignment expressions are permitted. The expression,

```
(3ρ1↓ϕNVEC)←10
```

is equivalent to:

```
NVEC[3ρ1↓ϕlρNVEC]←10
```

The enhancement, when not abused, is a welcome extension to the language.
Chapter 6

SELECTING

PROBLEM: Suppose you have constructed a matrix DEPN of annual depreciation rates to be used for assets which have depreciable lives of 20 years. DEPN has 20 rows and 12 columns. DEPN[Y;M] is the fraction of the asset to be depreciated in the Yth year of its life for an asset purchased in month M (1 for January, 2 for February, and so on). Suppose YEAR is a 1000 element vector of the ages (1 to 20) of 1000 assets and MONTH is a 1000 element vector of the months of purchase (1 to 12) for the corresponding assets. Determine the annual depreciation rates for these assets.

TOPIC: Scattered Point Indexing

A common mistake made when solving this problem is to try the following:

DEPN[YEAR;MONTH]

This expression shows nicely what you want to do but unfortunately does not do it. The shape of the result of matrix indexing is the catenation of the shape of the row indices with the shape of the column indices. Since YEAR has shape 1000 and MONTH has shape 1000, the result has shape 1000 1000. These 1,000,000 elements are the rates for every combination of the elements of YEAR and the elements of MONTH.

If you can picture this 1000 by 1000 element matrix in your mind's eye, you can see that the desired rates are sitting on the diagonal. The other rates are superfluous. If you have experimented much with dyadic transpose (~), you know that it can return the diagonal elements of a matrix argument by providing a left argument of 1 1 (in origin 1). Therefore, a correct expression to solve this problem is:

1 1 ~DEPN[YEAR;MONTH]

Unfortunately, this expression requires room in your workspace for the temporary 1000 by 1000 table. This may cause a WS FULL error. Even if available workspace is not a problem, the extraction of 1,000,000 rates when you need only 1000 is extremely inefficient.

An alternate approach to this problem is to view DEPN as a vector. The vector has 240 elements and is derived by raveling the matrix DEPN. Our job is to pack the vectors YEAR and MONTH into a single vector of indices into the raveled DEPN. The desired indices may be computed by the expression,

MONTH+12×YEAR-1
Chapter 6

SELECTING

Thus, the desired result may be computed from the expression,

\[(\text{DEPN})[\text{MONTH}+12\times\text{YEAR}-1]\]

This type of problem is called a "scattered point indexing" problem because the desired elements to be selected from the matrix are scattered throughout it. Normal matrix indexing (\text{MAT}[\text{ROWS};\text{COLS}]) is useful only when the elements to be selected are in a rectangular pattern.

Let us state the scattered point indexing solution in general terms:

\[(\text{MATRIX})[\text{COLUMNINDEX}+\text{NUMCOLS}\times\text{ROWINDEX}-1]\]

For a 3-dimensional array, the solution is:

\[(\text{ARRAY})[\text{COLUMNINDEX}+(\text{NUMCOLS}\times\text{ROWINDEX}-1)+(\text{NUMCOLS}\times\text{NUMROWS})\times\text{PLANEINDEX}-1]\]

or:

\[(\text{ARRAY})[\text{COLUMNINDEX}+\text{NUMCOLS}\times(\text{ROWINDEX}-1)+\text{NUMROWS}\times\text{PLANEINDEX}-1]\]

When performing scattered point indexing in origin 0 (OIO=0), the "-1" portions of the above expressions disappear:

Matrix (origin 0):

\[(\text{MATRIX})[\text{COLUMNINDEX}+\text{NUMCOLS}\times\text{ROWINDEX}]\]

3-D array (origin 0):

\[(\text{ARRAY})[\text{COLUMNINDEX}+\text{NUMCOLS}\times\text{ROWINDEX}+\text{NUMROWS}\times\text{PLANEINDEX}]\]

For this reason, scattered point indexing is frequently done in origin 0.
Chapter 6

PROBLEM: What algorithm may be used to return the unique values (UN) from a numeric vector (NV)? The unique values (UC) from a character vector (CV)?

TOPIC: Unique (Distinct) Values

Determination of the unique, or distinct, values is a common problem in the world of data processing. For example, given 1000 sales transactions which each include the salesperson number, you may want to compile a list of the numbers of the salespeople who had sales. If only 60 salespeople accounted for all 1000 sales, you would want to determine the numbers of those 60 salespeople. (You might also want to know the number of sales and the total dollar value of the sales attributed to each salesperson. These topics are covered in the next chapter.)

To illustrate the algorithms discussed in this section, we will use the following vectors:

\[
\begin{align*}
NV & = 30 \ 20 \ 20 \ 30 \ 10 \ 50 \ 10 \ 10 \\
CV & = \text{BOOKKEEPER}
\end{align*}
\]

Our task is to return the vectors UN and UC:

\[
\begin{align*}
UN & = 30 \ 20 \ 10 \ 50 \\
UC & = \text{BOOKKEEPER}
\end{align*}
\]

Since the problem of determining distinct values can be viewed as a searching problem, the most obvious algorithm uses the APL searching primitive, dyadic \( \tau \). Consider the result when you search the elements of a distinct vector for its own elements:

\[
\begin{align*}
8 \ 9 \ 7 \ 15 \ \tau & \ 8 \ 9 \ 7 \ 15 \\
1 \ 2 \ 3 \ 4
\end{align*}
\]

However, if the elements are not distinct, the pattern of the result is not so regular:

\[
\begin{align*}
NV \ \tau NV & = 1 \ 2 \ 2 \ 1 \ 5 \ 6 \ 5 \ 5 \\
(NV \ \tau NV) \ \tau & NV
\end{align*}
\]

In fact, wherever the result deviates from the vector of generated indices (\( \tau NV \)), the corresponding element is a repeat value, i.e. has occurred earlier in the vector. To flag the distinct values then:

\[
\begin{align*}
1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0
\end{align*}
\]
And to select the distinct values:

\[ D \left( (\text{NV} \text{\forall NV}) = 1 \right) \text{NV} \big/ \text{NV} \]  
\[ 30 \ 20 \ 10 \ 50 \]  

* Algorithm 1 *

Notice that this algorithm returns the distinct values in the same order as they first appear in the target vector. This algorithm also works on character vectors:

\[ (\text{CV} \text{\forall CV}) = 1 \big/ CV \]  

Since this algorithm depends upon the behavior of dyadic \( \text{\forall} \), it may also be used with functions which emulate dyadic \( \text{\forall} \). For example, if your task is to determine the distinct rows in the character matrix NAMES, you may do so with the following expression (given CMIOTA from the previous chapter):

\[ (\text{NAMES CMIOTA NAMES}) = \text{\forall} \) \text{\forall NAMES} \) \big/ \text{NAMES} \]

When using this algorithm on a large vector (say, 2000 or more elements), the dyadic \( \text{\forall} \) portion of the algorithm may require a significant amount of processing time. A more efficient algorithm may be constructed which uses the (typically very efficient) grade-up (\( \uparrow \)) primitive function.

Sort the vector:

\[ D \left( \text{SORTED} \text{\forall NV} \big/ \text{NV} \right) \]  
\[ 10 \ 10 \ 10 \ 20 \ 20 \ 30 \ 30 \ 50 \]

Shift the elements of the sorted vector to the right and compare to the sorted vector to flag the first distinct value in each run of like values:

\[ \text{\uparrow} \) \text{SORTED} \]  
\[ 50 \ 10 \ 10 \ 20 \ 20 \ 30 \ 30 \]  
\[ D \left( \text{FIRST} \left( \text{\forall} \) \text{SORTED} \right) \big/ \text{\uparrow} \) \text{SORTED} \]  
\[ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \]

Unfortunately, if the values in the vector SORTED are all the same, the vector FIRST will be all zeros. Yet the first value of FIRST should be 1. The following expression will set the first (in either origin) element to 1, and will have no effect if FIRST is an empty vector:

\[ \text{FIRST}[\text{\forall} \) \text{FIRST}] \leftarrow 1 \]

Finally, select the first distinct value in each run:

\[ D \left( \text{\forall} \text{\forall FIRST} \big/ \text{SORTED} \right) \]  
\[ 10 \ 20 \ 30 \ 50 \]
Notice that this algorithm returns the distinct values in ascending order (descending if \texttt{↓} is used). The entire algorithm follows:

\begin{verbatim}
SORTED←NL[↑NL]  \hspace{1cm} * Algorithm 2 *
FIRST←SORTED↑0↑SORTED 
FIRST[↓pFIRST]←1 
UN←FIRST/SORTED
\end{verbatim}

The algorithm works on character vectors once you manage to sort the characters. If your implementation of APL supports dyadic grade-up, you may replace the first statement by:

\begin{verbatim}
SORTED←CV[DADCV] 
\end{verbatim}

If it does not, you may replace the first statement by:

\begin{verbatim}
SORTED←CV[DADlCV] 
\end{verbatim}

As with numeric vectors, the distinct elements in the final result are in ascending order (where DAV, the atomic vector, defines the collating sequence).

You may determine the distinct rows (UNAMES) of a character matrix (NAMES) using this algorithm if you are able to sort the matrix. If your implementation of APL supports dyadic grade-up, do the following:

\begin{verbatim}
SORTED←NAMES[DADNAMES;] 
FIRST←v/SORTED↑0↑SORTED 
FIRST[↓pFIRST]←1 
UNAMES←FIRST/SORTED
\end{verbatim}

If your implementation does not support dyadic grade-up, use the CGRADEUP function developed in the previous chapter.

The expression CV[DADlCV] in the statement above suggests another algorithm for determining the distinct elements of a character vector. Consider the meaning of the expression CV\textbf{↓}DAV, or better still, the expression DAV\textbf{e}CV. The result of the latter expression is a 256 element Boolean vector that flags the elements of DAV which are in CV. Since the elements of DAV are distinct by definition, the remaining step is to use the Boolean vector to select the corresponding elements from DAV. The complete algorithm:

\begin{verbatim}
UC←(DAV\textbf{e}CV)/DAV  \hspace{1cm} * Algorithm 3 *
\end{verbatim}

Like algorithm 2, this algorithm returns the distinct values in ascending order (according to DAV). In some implementations of APL, \texttt{e} is extremely fast on character data. In others, it is not. If not, use algorithm 1 or 2.

Extending this algorithm to numeric (or at least integer) data will produce ridiculous expressions like:

\begin{verbatim}
UN←((↓1E30)\textbf{e}NV)/↓1E30
\end{verbatim}
which will hopefully generate a WS FULL error message rather than run into the next century. When we stop to consider the philosophy of the algorithm rather than its implementation, however, a very clever algorithm emerges.

The philosophy is to consider the nature of the values in NV. The expression above assumes that all elements of NV are integers between 1 and 1E30 (origin 1). A more realistic problem would have the values between, say, 1 and 100. Let us view these values as indices rather than numbers and use them as such. Initialize a Boolean vector to have 100 zeros:

\[ \text{BIT} \leftarrow 100 \rho 0 \]

Use the numeric (index) vector to index assign 1s into the Boolean vector:

\[ \text{BIT}[\text{NV}] \leftarrow 1 \]

Duplicates in NV are essentially discarded. The final step is similar to the final step in algorithm 3:

\[ \text{UN} \leftarrow \text{BIT} / 100 \]

Notice that this algorithm returns the distinct values in ascending order, that it works only on positive integers (110 and up), that the maximum value (MAX) must be known and not too large (else WS FULL on MAX\( \rho 0 \)). This algorithm is extremely fast on vectors of any size and for any APL implementation. The entire algorithm follows:

\[
\begin{align*}
\text{BIT} & \leftarrow (\text{MAX} + \sim 10) \rho 0 \\
\text{BIT}[\text{NV}] & \leftarrow 1 \\
\text{UN} & \leftarrow \text{BIT} / 1 \rho \text{BIT}
\end{align*}
\]

```
```

\[
\sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim \sim
```

-87-
PROBLEM: For each of the unique-value algorithms presented in the last section, what additional logic must be included to also return the indices (INDS) of the numeric vector (NV) or character vector (CV) into the vector of distinct values (UN or UC)? For example, suppose NV and UN have the following values:

NV
30 20 20 30 10 50 10 10
UN
30 20 10 50

The desired value if INDS is:

INDS
1 2 2 1 3 4 3 3

TOPIC: Translating Distinct Values to Distinct Indices

The obvious solution to this problem is:

INDS←UN\NV or INDS←UC\CV

These indices are useful when performing frequency counts or accumulations. These topics are covered in the next chapter.

While obvious, the dyadic \ approach may not be the most efficient technique for computing these indices, depending upon the unique-value algorithm being used.

In the case of Algorithm 1, a dyadic \ is already being performed so there is no need to do it again. Instead:

\( UN \leftarrow (B \leftarrow (I \leftarrow NV \div NV) \div \rho NV) / NV \) \hspace{1cm} \textit{Algorithm 1} \hspace{1cm} \textit{or:} \hspace{1cm} \text{INDS} \leftarrow (\omega B)[I] \sim 0IO

The expansion in the second statement or the cumulative sum in the third are typically more efficient than another dyadic \.

In the case of Algorithm 2, the grade-up operation removes the need to perform dyadic \. Instead:

\( \text{SORTED} \leftarrow NV[G \div NV] \hspace{1cm} \textit{Algorithm 2} \hspace{1cm} \text{FIRST} \leftarrow \text{SORTED} \div 1 \div \text{SORTED} \)
\( \text{FIRST} \div [\omega \rho \text{FIRST}] \leftarrow 1 \)
\( \text{UN} \div \text{FIRST} / \text{SORTED} \)
\( \text{FIRST} \div [\omega \rho \text{FIRST}] \leftarrow 0 IO \)
\( \text{INDS} \leftarrow (\rho \text{FIRST}) \div 0 \)
\( \text{INDS}[G] \leftarrow + \backslash \text{FIRST} \)
The fifth statement sets the first element of FIRST to 1 or 0 depending upon the index origin. The statement is needed only if you may be working in origin 0 since the first element of FIRST will already be 1 (unless FIRST is empty). The plus scan (+\) and indexing operations in the last statement are typically more efficient than another dyadic \. The last two statements are required to "unsort" the indices so that they are in the order of the elements of NV rather than of SORTED. A simpler, though less efficient, statement may be used in place of the last two statements:

\[ \text{INDS} \leftarrow (+\text{\textbackslash FIRST})[\&G] \]

In the case of Algorithm 3, there is no way to avoid the dyadic \. So:

\[ \text{UC} \leftarrow (\text{\textbackslash AV} \in \text{CV}) \div \text{\textbackslash AV} \]
\[ \text{INDS} \leftarrow \text{UC} \setminus \text{CV} \]

* Algorithm 3 *

In the case of Algorithm 4, there is no need to do the dyadic \ for the same reason there is no need to do any searching when determining the distinct values:

\[ \text{BIT} \leftarrow (\text{MAX} \div \& IO) \rho 0 \]
\[ \text{BIT[NV]} \leftarrow 1 \]
\[ \text{UN} \leftarrow \text{BIT} \setminus \& \rho \text{BIT} \]
\[ \text{INDS} \leftarrow (\text{BIT} \setminus \& \rho \text{UN})[\text{NV}] \]

* Algorithm 4 *

PROBLEMS: (Solutions on pages 337 to 339)

1. What APL expression will select every other element (1st, 3rd, 5th, ...) of a vector V which has an even number of elements?

2. What APL expression will return a vector of the elements on the diagonal of the square matrix M?

3. What APL expression will return the Nth column of the matrix M as a vector, where N is the value of the last element of the vector V?
4. In a character matrix NAMES of passenger names, the last names precede the first names and are separated from them by a slash (/). What APL expression(s) will replace the slashes with commas (,)?

5. As an actuary, you need to determine the mortality rates for a set of 500 policyholders. You have a 3 dimensional "select and ultimate" table of mortality rates named RATES. RATES has shape 2 100 16. The first dimension is sex (female, male). The second dimension is issue age (0 to 99). The third dimension is duration from issue (0 to 15 years). You have three 500 element vectors: SEX, IAGE, DUR. The elements of these vectors are in 1-to-1 correspondence with the 500 policyholders. SEX indicates the policyholder's sex (0=female; 1=male). IAGE indicates the issue age (0 to 99). DUR indicates duration from issue (0 to 15 years).

A. Construct the 500 element vector MRATES of the mortality rates for these 500 policyholders.

B. Suppose some of the durations from issue (elements of DUR) exceed 15. For these policies, use duration 15 (the "ultimate" duration) but increase the issue age by the amount that the duration exceeds 15. Construct MRATES.

C. Suppose you receive a memo which informs you that 40 of the elements in RATES are incorrect and must be modified. From the information in the memo, you construct 4 vectors of length 40: NEWRATES, the correct rates; NEWSEX, the sexes for the new rates; NEWIAGE, the issue ages for the new rates; NEWDUR, the durations for the new rates. Insert the new rates into RATES.

6. Write the monadic function UNQNV which returns the distinct elements of its numeric vector argument. Write UNQCV to return the distinct elements of its character vector argument. Write UNQCM to return the distinct rows of its character matrix argument.

Write the dyadic functions UNQII and UNQIO which return the distinct elements of their index vector right arguments (origin 1 or 0 indices respectively). The left argument of UNQII or UNQIO is a scalar of the number of possible indices. For example, a left argument of 5 for UNQIO implies that all indices in the right argument are elements of the set 0 1 2 3 4.

In addition to the distinct values, each function should compute the indices of the right argument values into the resulting...
distinct values. Assign the indices to the global variable \(<\text{ind}()>\). Place lamps \((A)\) in front of the lines which compute \(<\text{ind}>\) so that the indices are not computed unless the lamps are removed.
The often used APL expression +/A adds up the elements of the array A in such a way that one of the dimensions of A is eliminated, or "reduced". If A is a 1000 element vector, the result is a scalar whose one element is the sum of the 1000 elements of A. Typically, the elements of a vector represent measurements or counts of respective real world items. Then, the result of the expression +/A represents the sum of the measurements or counts for all items.

Frequently in the business world, we tend to categorize items rather than lump them together. For example, 1000 sales transactions may be categorized by retail outlet or by salesperson or by product or by day, and so on. In such an environment, we may want to "reduce" the 1000 element vector of invoice amounts into 10 sums (say, by salesperson) rather than into just a single grand total. Such a problem has traditionally been called an "accumulation" problem in APL. Naturally, to solve such a problem, we need a corresponding 1000 element vector whose values indicate the salesperson responsible for each transaction (e.g. salesperson number). Such a vector is called a "classification" vector.

By analyzing the classification vector alone, we can answer questions such as, "For how many transactions was each salesperson responsible?" Such a problem has been called a "frequency count" problem in APL.

If a second classification vector is available which represents, say, day of the week of the sale, we may want to look at sales broken down (i.e. added up) by both salesperson and day of the week. In this case, we want to reduce the 1000 element vector of invoice amounts into a 10 (salesperson) by 7 (days of the week) matrix. Such a problem has been called a "cross-tabulation" problem in APL.

To avoid ambiguity from existing terminology, we present the following terminology and definition:
Chapter 7 FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABULATIONS

"n-way plus reduction on dimension d of array A by classification 1, classification 2, ..., and classification n"

The summarization of array A across dimension d such that dimension d is replaced by n new dimensions whose magnitudes are the number of classes defined for the corresponding classifications 1, 2, ..., n.

Let us try this terminology on an example. Suppose you have information on 1000 life insurance policies. A subset of the information is listed below:

<table>
<thead>
<tr>
<th>Issue Age</th>
<th>Underwriting Class</th>
<th>Death Benefit</th>
<th>Annual Premium</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0 to 99]</td>
<td>[M,F]</td>
<td>[S,A,B,C,D]</td>
<td>(DBEN)</td>
</tr>
<tr>
<td>(IAGE)</td>
<td>(SEX)</td>
<td>(UCLASS)</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>M</td>
<td>C</td>
<td>150</td>
</tr>
<tr>
<td>27</td>
<td>M</td>
<td>S</td>
<td>100</td>
</tr>
<tr>
<td>42</td>
<td>F</td>
<td>S</td>
<td>80</td>
</tr>
<tr>
<td>50</td>
<td>M</td>
<td>B</td>
<td>100</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

The names IAGE, SEX, UCLASS, DBEN and APREM represent the names of the APL variables containing the corresponding data. Each variable is a 1000 element vector. SEX and UCLASS are character vectors and the rest are numeric vectors. Here are a few of the plus reductions that can be performed on these data:

1. The 0-way plus reduction of APREM. This is simply +/APREM.

2. The 1-way plus reduction of APREM by SEX. This is the 2 element vector (+/(SEX='M')/APREM), (+/(SEX='F')/APREM).

3. The 3-way plus reduction of DBEN by AGE, SEX and UCLASS. This is a 100 (ages) by 2 (sexes) by 5 (underwriting classes) array whose elements contain the sums of the elements of DBEN for each combination of AGE, SEX and UCLASS. Notice that the result coincidentally has 1000 elements, the same number of elements as DBEN, the vector being "reduced". The number of elements has not been reduced at all. In fact, if another classification were added, the result would contain more elements than the vector being reduced. Most of the elements of the result will be 0. The data will have become so finely classified that each "cell" (i.e. combination of classes) in the result contains at best a few policies.

4. The 1-way plus reduction of 1 (or (pSEX)p1) by SEX. This is the frequency count by sex: (+/SEX='M'), (+/SEX='F').
5. The 2-way plus reduction on dimension 1 of DBEN,[1.5]APREM by AGE and UCLASS. This is a 100 (ages) by 5 (underwriting classes) by 2 (columns...the dimension not reduced) array whose elements contain the sum of the elements of DBEN (column 1) and APREM (column 2) for each combination of age and underwriting class.

PROBLEM: What is the frequency count (or 1-way plus reduction of 1) by issue age (IAGE) for the above insurance policies? Call the result F. Accumulate (or 1-way plus reduce) APREM by the same ages. Call the result A.

TOPIC: One-Way Plus Reductions

Let us solve this problem first by using the classical APL approach, which is simple but ignores efficiency considerations. The distinct issue ages can be determined by using an algorithm discussed in the previous chapter:

\[ \text{DIA} \left( (\text{IAGE} \setminus \text{IAGE}) = \iota \text{IAGE} \right) / \text{IAGE} \]

Next, use outer product to compare the vector of issue ages to the vector of distinct issue ages:

\[ \text{M} \leftarrow \text{DIA} \circ = \iota \text{IAGE} \]

The result is a Boolean matrix with one row per distinct issue age (say 40) and one column per policy (say 1000). Each column has exactly one 1, marking the distinct age (row) to which that policy (column) corresponds. The frequency count by distinct issue age follows directly:

\[ \text{F} \leftarrow +/\text{M} \]

and the accumulation of APREM by distinct issue age is not far behind:

\[ \text{A} \leftarrow \text{M}+\times\text{APREM} \]

The dimensions of M (say 40 by 1000) and APREM (say 1000) conform along their inner coordinates allowing the matrix multiplication (+,x) which reduces the inner coordinates (1000) and returns a vector with the same number of elements as M has rows (40).

This approach is simple. However, it is inefficient and is prone to WS FULL errors. Its inefficiencies stem from the many needless
comparisons and computations which take place in the \( \circ. = \) and \( +. \times \)
functions. Its WS FULL tendencies are caused by the potentially
gigantic result of the outer product \( (\circ. =) \).

A more efficient approach uses the sort-and-shift techniques employed
in the previous chapter. Begin by sorting the ages in ascending
order (retaining the grade vector):

\[
\begin{align*}
\text{GRADE} & \leftarrow \text{IAGE} \\
\text{SORTED} & \leftarrow \text{IAGE}[\text{GRADE}]
\end{align*}
\]

Shift the elements of the sorted vector to the left and compare to
the sorted vector to flag the last distinct value in each run of like
values:

\[
\text{LAST} \leftarrow \text{SORTED} \# 1 \# \text{SORTED}
\]

Unfortunately, if the values in the vector SORTED are all the same,
the vector LAST will be all zeros. Yet the last value of LAST should
be 1. To avoid this problem, you may instead use an odd expression
like the following (which assumes that no policyholder has issue age
\( \sim 99 \)):

\[
\text{LAST} \leftarrow \text{SORTED} \# 1 \# \text{SORTED}, \sim 99
\]

Sometimes, using an arbitrary number such as \( \sim 99 \) is not feasible.
Perhaps the vector could contain any conceivable value. The
following alternate expressions will set the last (in either origin)
element to 1, and will have no effect if SORTED is an empty vector:

\[
\text{LAST} \leftarrow \text{SORTED} \# 1 \# \text{SORTED}
\]

\[
\text{LAST}[(-1 + \rho \text{LAST}) + 1 \times \rho \text{LAST}] \# 1
\]

Select the last distinct value in each run:

\[
\text{DIA} \leftarrow \text{LAST} / \rho \text{LAST}
\]

You may have noticed that we determined the distinct values
differently than in the previous chapter. There, we constructed a
bit vector FIRST which flagged the first 1 of each run of like
values; here, we constructed a bit vector LAST which flagged the last
1 of each run. The reason for this minor change of algorithm is that
LAST is more useful for determining the 1-way plus reductions.
Consider the meaning of the expression:

\[
\text{CUM} \leftarrow \text{LAST} / \rho \text{LAST}
\]

CUM has one element per distinct issue age. The values are the
cumulative frequency counts (in origin 1). If we can undo the
cumulative effect, we will have the frequency counts. Cumulative
sums are undone by taking the first differences:

\[
F \leftarrow \text{CUM} - (\rho \text{CUM}) \rho 0, \text{CUM}
\]
To accumulate APREM, we use the same approach. Reorder APREM to be in 1-to-1 correspondence with the sorted issue ages (SORTED); determine the cumulative sum; select the last element for each run:

\[
CUM \leftarrow \text{LAST} / \sum \text{APREM[GRADE]}
\]

Compute the first differences to get the desired result:

\[
A \leftarrow CUM - (\circ \ CUM) \circ 0, CUM
\]

This algorithm is extremely efficient, especially on large vectors. As vectors get larger, the required processing time generally increases linearly for this grade-up (\(\circ\)) based algorithm; but it increases exponentially for the outer product (\(\circ.\circ\)) based algorithm.

This algorithm works just as well on character classification vectors (e.g. UCLASS or SEX) as it does on numeric classification vectors (e.g. IAGE). Just begin by converting the characters to indices. For example:

\[
\text{GRADE} \leftarrow \text{S}^\circ \text{A}^\circ \text{B}^\circ \text{C}^\circ \text{D} \cup \text{UCLASS}
\]

or

\[
\text{GRADE} \leftarrow \text{S}^\circ \text{A}^\circ \text{B}^\circ \text{C}^\circ \text{D} \uparrow \text{UCLASS}
\]

Use the latter expression if your implementation of APL supports dyadic grade-up, and the former expression if it does not.

---

**PROBLEM:** Generate a 3 by 10 by 2 by 5 array (SMRY) which summarizes (or 3-way plus reduces) the above insurance policies by age, sex and underwriting class. The definition of SMRY follows:

- \(\text{SMRY[1,:,:,:]}\) Total death benefits
- \(\text{SMRY[2,:,:,:]}\) Total annual premiums
- \(\text{SMRY[3,:,:,:]}\) Frequency count (number of policies)
- \(\text{SMRY[4,:,:,:]}\) Age group I (1 is 0-9; 2 is 10-19; ...; 10 is 90-99)
- \(\text{SMRY[5,:,:,:]}\) Sex group J (1 is male; 2 is female)
- \(\text{SMRY[6,:,:,:]}\) Underwriting group K (1 is Standard; 2 is A; 3 is B; 4 is C; 5 is D)
TOPIC: N-Way Plus Reductions

This problem is awkward to solve using inner product and outer product techniques. The solution is confusing, WS FULL prone and inefficient.

A neater solution arises when you look at the problem backwards. Consider the result SMRY. The three elements defined by SMRY[;I;J;K] are affected by just those policies belonging to age group I, sex group J and underwriting group K. Conversely, each policy affects exactly one set of three elements in the result.

To simplify the discussion, let's consider just SMRY[3;;;], i.e. frequency counts. The first element of this array represents the number of policies in age group 1, sex group 1 and underwriting group 1. The second element of this (raveled) array represents the number of policies in age group 1, sex group 1 and underwriting group 2. And so on. The last (100th) element of this (raveled) array represents the number of policies in age group 10, sex group 2 and underwriting group 5.

By considering the result as a vector, you may treat this problem as a 1-way plus reduction (100 classes) rather than as a 3-way plus reduction (10 by 2 by 5 classes). All that remains is to pack together the values of the three classification vectors (IAGE, SEX, UCLASS) such that the resulting packed classification vector has values which distinctly identify the cell of the result affected by the corresponding policy.

The ideal packing scheme is one which converts the classification values for each policy directly into the index of the affected element in the raveled result. For example:

<table>
<thead>
<tr>
<th>IAGE</th>
<th>SEX</th>
<th>UCLASS</th>
<th>Index (RRI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>36 (4)</td>
<td>M (1)</td>
<td>C (4)</td>
<td>34</td>
</tr>
<tr>
<td>27 (3)</td>
<td>M (1)</td>
<td>S (1)</td>
<td>21</td>
</tr>
<tr>
<td>42 (5)</td>
<td>F (2)</td>
<td>S (1)</td>
<td>46</td>
</tr>
<tr>
<td>50 (6)</td>
<td>M (1)</td>
<td>B (3)</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(origin 1)</td>
</tr>
</tbody>
</table>

The formula being used here is:

\[ RRI = UCLASSINDEX + (5 \times SEXINDEX - 1) + (10 \times IAGEINDEX - 1) \]

When working in origin 0, the formula is a bit simpler:
### Chapter 7 FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABULATIONS

<table>
<thead>
<tr>
<th>IAGE</th>
<th>SEX</th>
<th>UCLASS</th>
<th>Result Index (RRI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>36 (3)</td>
<td>M (0)</td>
<td>C (3)</td>
<td>33 (origin 0)</td>
</tr>
<tr>
<td>27 (2)</td>
<td>M (0)</td>
<td>S (0)</td>
<td>20</td>
</tr>
<tr>
<td>42 (4)</td>
<td>F (1)</td>
<td>S (0)</td>
<td>45</td>
</tr>
<tr>
<td>50 (5)</td>
<td>M (0)</td>
<td>B (2)</td>
<td>52</td>
</tr>
</tbody>
</table>

\[ RRI = \text{UCLASSINDEX} + (5 \times \text{SEXINDEX}) + (10 \times \text{IAGEINDEX}) \]

The 5 in the above formula refers to the number of elements in each row of the result (i.e. the number of UCLASS classes). The 10 refers to the number of elements in each plane (i.e. the number of UCLASS classes times the number of SEX classes).

The computations of UCLASSINDEX, SEXINDEX and IAGEINDEX are straightforward:

1. \( \text{UCLASSINDEX} = 'SABCD' \cdot \text{UCLASS} \)
2. \( \text{SEXINDEX} = 'F' = \text{SEX} \)
3. \( \text{IAGEINDEX} = \text{IAGE} + 10 \)

The elements of the vector RRI are all integers between 0 and 99.

You may then use logic from the prior section to determine the distinct values in RRI and the corresponding frequencies (1-way plus reduction of 1) and 1-way plus reductions of APREM and DBEN. If you initialize the result to be an all-zero vector of the desired length (the length of the raveled result), you may simply index assign the derived frequencies and sums using the corresponding distinct indices from RRI. Finally, reshape the result to the proper shape.

The complete logic follows:

1. Use origin 0 throughout: \( \text{OIO} \leftarrow 0 \)
2. Index in result of column affected by policy:
   - \( \text{UCLASSINDEX} = 'SABCD' \cdot \text{UCLASS} \)
3. Index of row affected:
   - \( \text{SEXINDEX} = 'F' = \text{SEX} \)
4. Index of plane affected:
   - \( \text{IAGEINDEX} = \text{IAGE} + 10 \)
5. Index in raveled SMRY[0;;] affected:
   - \( \text{RRI} = \text{UCLASSINDEX} + 5 \times \text{SEXINDEX} + 2 \times \text{IAGEINDEX} \)
6. Sort result indices in ascending order:
   - \( \text{GRADE} \leftarrow \text{RRI} \)
   - \( \text{SORTED} \leftarrow \text{RRI}[\text{GRADE}] \)
7. Shift to left and compare to flag last distinct values:
   - \( \text{LAST} \leftarrow \text{SORTED} \# \text{SORTED}, -1 \)
8. Select last distinct value in each run:
   - \( \text{URRI} \leftarrow \text{LAST} / \text{SORTED} \)
Chapter 7 FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABULATIONS

\begin{verbatim}
\texttt{a Initialize 3 raveled all-zero arrays for SMRY[0;;;],
SMRY[1;;;] and SMRY[2;;;]:
SMRY0 \leftarrow \text{SMRY1} + \text{SMRY2} \leftarrow (10 \times 2 \times 5)P0
\text{a One-way plus reduce DBEN and insert into SMRY0:}
CUM \leftarrow \text{LAST} + \backslash \text{DBEN[GRADE]}
SMRY0[URRI] \leftarrow \text{CUM} - (\rho\text{CUM})P0,CUM
\text{a Ditto for APREM into SMRY1:}
CUM \leftarrow \text{LAST} + \backslash \text{APREM[GRADE]}
SMRY1[URRI] \leftarrow \text{CUM} - (\rho\text{CUM})P0,CUM
\text{a Ditto for frequency count into SMRY2 (origin 0):}
CUM \leftarrow \text{LAST} / \uparrow \text{LAST}
SMRY2[URRI] \leftarrow \text{CUM} - (\rho\text{CUM})P^{-1},CUM
\text{a Catenate and reshape:}
SMRY \leftarrow 31025P\text{SMRY0,SMRY1,SMRY2}
\text{IO} \leftarrow 1

The logic above can be shortened somewhat if you choose to view the
problem as a single 3-way plus reduction on the last dimension of the
three row matrix whose rows are DBEN, APREM and all 1s. Then, the
logic is:

\text{URRI} \leftarrow \text{LAST/SORTED}
SMRY \leftarrow (3,10 \times 2 \times 5)P0
CUM \leftarrow \text{LAST/\backslash \text{(DBEN,[0]APREM,[\sim 0.5])}[;\text{GRADE]}}
SMRY[;URRI] \leftarrow \text{CUM} - (\rho\text{CUM})P0,CUM
SMRY \leftarrow 31025P\text{SMRY}
\text{IO} \leftarrow 1

Once this 4 dimensional array has been constructed, many questions
can be answered by performing a few simple indexing and plus
reduction operations. For example (origin 1):

1. How many males and females?
   \text{+/[1]++/3SMRY[3;;;]}

2. What is the total death benefit by sex and underwriting class
   (2 row, 5 column matrix)?
   \text{+/[1]SMRY[1;;;]}

3. What is the average annual premium by age group and sex (10
   row, 2 column matrix)?
   \text{+/[1]++/[4SMRY[2 3;;;]}

4. What is the death benefit, annual premium and frequency
   breakdown by age group and underwriting class, where ages are
   broken into 5 groups: 0 to 19; 20 to 39; ...; 80 to 99?
   \text{+/[3] 3 5 2 5 \rho+/[3SMRY}
\end{verbatim}

-99-
PROBLEM: Generate a 2 by 10 by 5 array (MAX) which contains the maximums of (or 3-way maximum reduces) the above insurance policies by age, sex and underwriting class. The definition of MAX follows:

\[
\begin{align*}
\text{MAX}[1,:,:] & \quad \text{Maximum death benefits} \\
\text{MAX}[2,:,:] & \quad \text{Maximum annual premiums} \\
\text{MAX}[;,I;J;K] & \quad \text{Maximum death benefit and annual premium for age group I, sex group J, underwriting group K.}
\end{align*}
\]

TOPIC: N-Way Maximum and Minimum Reductions

This problem is no different from the problem in the prior section except an n-way maximum reduction is being performed instead of an n-way plus reduction. Consequently, the same solution works, up to a point. That point in the above solution is:

\[
\begin{align*}
\text{SMRYO} & \leftrightarrow \text{SMRY1} \leftrightarrow \text{SMRY2} \leftrightarrow (10 \times 2 \times 5) \rho 0 \\
\text{CUM} & \leftrightarrow \text{LAST} / \leftrightarrow \text{DBEN}[\text{GRADE}] \\
\text{SMRYO}[\text{URRI}] & \leftrightarrow \text{CUM} - (\rho \text{CUM}) \rho 0, \text{CUM}
\end{align*}
\]

Unfortunately, you may not simply substitute \( - \) for \( + \). The algorithm happens to work for \( + \) because of the nature of addition. We must find a comparable algorithm which will work for maximum (and hopefully minimum).

One such algorithm involves the grade-up (\( \uparrow \)) function. The idea behind the algorithm is to sort the values within their respective classes (i.e. within like values of SORTED) and then use LAST to select the maximum in each class. Stated differently, you must perform a two-key sort where RRI is the major key and the data (DBEN) is the minor key. Do the minor key first:

\[
\text{G} \leftrightarrow \text{DBEN}
\]

and then the major key:

\[
\text{G} \leftrightarrow \text{G}[\uparrow \text{RRI}[\text{G}]]
\]

Use this grade vector to reorder the values, and use LAST to select the maximum in each class:

\[
\text{MAX} \leftrightarrow \text{LAST} / \text{DBEN}[\text{G}]
\]
or quicker:

\[
\text{MAX} \leftarrow \text{DBEN[LAST/G]}
\]

The final step is to index assign MAX into the result variable (MAX0). When performing the n-way plus reduction, we initialized SMRY0 as all zeros so that those classes which were not represented by any policies (i.e. cells not index assigned) would show a plus reduction result of zero (just as \(+/0\) is 0). In mathematical terminology, the "identity element" of plus is 0. Likewise, you should initialize MAX0 as a vector filled with the identity element for maximum, which is negative infinity and is returned as nearly as possible by the expression \(+/0\). The rest of the solution is:

\[
\begin{align*}
\text{MAX0} & \leftarrow \text{MAX1} \leftarrow (10 \times 2 \times 5) \times l0 \\
\text{MAX0}[\text{URRI}] & \leftarrow \text{MAX} \\
\text{G} & \leftarrow \text{APREM} \\
\text{G}[\text{URRI[G]]} & \leftarrow \text{MAX} \\
\text{MAX} & \leftarrow \text{APREM[LAST/G]} \\
\text{MAX1}[\text{URRI}] & \leftarrow \text{MAX} \\
\text{MAX} & \leftarrow 2 \text{ 10 2 5} \times \text{MAX0,MAX1} \\
\text{DIO} & \leftarrow 1
\end{align*}
\]

A different algorithm solves the problem without the use of grade-up. It uses instead maximum scan (\(\backslash\)). In order for maximum scan to be useful, the values in each subsequent class must first be shifted up the number scale so that the maximum value of an earlier class does not shadow the maximum value of a subsequent class. To illustrate, suppose the values of LAST and DBEN[GRADE] are as follows:

<table>
<thead>
<tr>
<th>LAST</th>
<th>0 0 1 0 0 0 1 0 1 1</th>
</tr>
</thead>
</table>

Suppose we add 100 to each value in the first class, 200 to the second class, 300 to the third class and 400 to the fourth class:

<table>
<thead>
<tr>
<th>LAST</th>
<th>120 110 115 215 235 220 225 320 325 430</th>
</tr>
</thead>
</table>

The maximum scan of this vector is:

<table>
<thead>
<tr>
<th>LAST</th>
<th>120 120 120 215 235 235 235 320 325 430</th>
</tr>
</thead>
</table>

Using LAST to select from this vector produces:

<table>
<thead>
<tr>
<th>LAST</th>
<th>120 235 325 430</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBEN[GRADE]</td>
<td>20 35 25 30</td>
</tr>
</tbody>
</table>

Subtracting 100, 200, 300 and 400 from the respective classes gives the desired maximums by class:

<table>
<thead>
<tr>
<th>LAST</th>
<th>20 35 25 30</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBEN[GRADE]</td>
<td></td>
</tr>
</tbody>
</table>
For this logic to work, you must shift (add to) the values of each class an amount which is at least equal to the difference between the maximum value in the preceding class and the minimum value in this class. Since your task is to determine these very maximums, you cannot use these precise numbers. Rather, you can determine the difference between the maximum and minimum values for the entire data vector and use that amount.

The following are expressions which implement this algorithm as well as numbers which illustrate the procedure:

\[
\begin{align*}
DIF &\leftarrow (f/D)-l/D & 25 \\
FIRST &\leftarrow -1\Phi LAST & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
T &\leftarrow \text{FIRST}\backslash \text{DIF} & 25 & 25 & 25 & 50 & 50 & 50 & 75 & 75 & 100 \\
U &\leftarrow last/T & 25 & 50 & 75 & 100 \\
R &\leftarrow D+T & 45 & 35 & 40 & 65 & 85 & 70 & 75 & 95 & 100 & 130 \\
R &\leftarrow f\backslash R & 45 & 45 & 45 & 65 & 85 & 85 & 85 & 95 & 100 & 130 \\
MAX &\leftarrow (\text{LAST}/R)-U & 20 & 35 & 25 & 30 \\
\end{align*}
\]

The solution to the above problem using this algorithm can be written as follows:

\[
\begin{align*}
\text{MAX} &\leftarrow (10\times 2\times 5)f/10 \\
D &\leftarrow DBEN[GRADE] \\
DIF &\leftarrow (f/D)-l/D \\
T &\leftarrow \text{FIRST}\backslash \text{DIF} \\
\text{MAX} &\leftarrow \text{MIN}[\text{URRI}]\backslash (\text{LAST}/\text{T})-\text{LAST}/T \\
\text{MAX} &\leftarrow (\text{URRI}+20)\text{MAX},\text{MAX}0 \\
\text{MAX} &\leftarrow 1 \\
\text{MAX} &\leftarrow \text{APREM}[GRADE] \\
\text{DIF} &\leftarrow (f/D)-l/D \\
T &\leftarrow \text{FIRST}\backslash \text{DIF} \\
\text{MAX} &\leftarrow \text{MIN}[\text{URRI}]\backslash (\text{LAST}/\text{T})-\text{LAST}/T \\
\text{MAX} &\leftarrow 2 \ 10 \ 2 \ 5 \rho \text{MAX},\text{MAX}0 \\
\text{MAX} &\leftarrow 1 \\
\end{align*}
\]

You can do both DBEN and APREM at once with some minor modifications:

\[
\begin{align*}
\text{MAX} &\leftarrow (2,10\times 2\times 5)f/10 \\
D &\leftarrow (\text{DBEN}[.5]\text{APREM}[;\text{GRADE}] \\
\text{DIF} &\leftarrow (f/D)-l/D \\
T &\leftarrow \text{DIF}\times \text{\text{URRI}}\backslash \text{LAST} \\
\text{MAX} &\leftarrow \text{MIN}[\text{URRI}]\backslash (\text{LAST}/\text{T})-\text{LAST}/T \\
\text{MAX} &\leftarrow 2 \ 10 \ 2 \ 5 \rho \text{MAX} \\
\end{align*}
\]
There are potential problems with the maximum scan algorithm. If there are many classes (i.e. 1s in LAST) and if the difference between the maximum and minimum values of the data vector is large, the values in the vector on which the maximum scan is being performed will become immense. If they become too large (beyond 16 or so significant digits), precision will be lost and the results may be incorrect. Even if precision is not lost, the numbers may get large enough to require internal floating point (usually 8 bytes per element) representation rather than integer (usually 2 or 4 bytes per element) representation. Consequently, the chances of a WS FULL error will increase and processing speed will decline since floating point numbers require more processing time than do integers.

Despite these potential problems, the maximum scan algorithm is a useful and efficient algorithm.

If the problem at the beginning of this section were stated as a minimum reduction problem rather than a maximum reduction problem, the same two approaches could have been taken, with slight modifications. Here are the solutions:

[grade-up algorithm]

:  
URRI+LAST/SORTED
MIN0+MIN1←(10×2×5)ρl/\0

G+\DBEN
G+G[\RRI[G]]
MIN0[URRI]+DBEN[LAST/G]

G+\APREM
G+G[\RRI[G]]
MIN1[URRI]+APREM[LAST/G]

MIN←2 10 2 5 ρMIN0,MIN1
UI0←1

[minimum scan algorithm]

:  
URRI+LAST/SORTED
MIN←(2,10×2×5)ρl/\0

D+(DBEN,[−0.5]APREM)[;GRADE]
DIF←(f/D)-l/D
T=DIF.*+1\LAST
MIN[;URRI]←(LAST/\D-T)+LAST/T

MIN←2 10 2 5 ρMIN

-103-
PROBLEM: Suppose the variable ACTIVE is a Boolean vector with the same length as AGE, SEX, DBEN, ... whose values indicate whether the corresponding policies are active. Generate a 10 by 2 by 5 Boolean array (ALL) whose elements indicate whether (1) or not (0) all of the insurance policies (as defined above) are active within each possible age, sex and underwriting class. That is, perform the 3-way and-reduction (^/) of ACTIVE by age, sex and underwriting class.

ALL[I;J;K] The and-reduction of ACTIVE for age group I, sex group J, underwriting group K.

TOPIC: N-Way Logical Reductions

Once again, the solution to this problem is similar to that of an n-way plus reduction. However, you need to devise an algorithm for ^/ comparable to that for +/:

\[
\begin{align*}
\text{SMRYO} &\leftarrow (10 \times 2 \times 5) \rho 0 \\
\text{CUM} &\leftarrow \text{LAST} / + \text{DBEN[GRADE]} \\
\text{SMRYO[URRI]} &\leftarrow \text{CUM} - (\rho \text{CUM}) \rho 0, \text{CUM}
\end{align*}
\]

Unfortunately, you may not simply substitute ^ for + in the above logic. One simple solution may be derived from the knowledge that the ^/ is true if the +/- equals the frequency count. As with the / and \ problems of the last section, you must begin by initializing the result with the identity element for the reduction function. The identity element for ^ is one, i.e. \(1=^/10\). The explanation for the identity element is: for any Boolean array B, \((1 \land B)\) and \((B \land 1)\) always return exactly B, so 1 is the identity element for ^ . The algorithm is:

\[
\begin{align*}
\text{ALL} &\leftarrow (10 \times 2 \times 5) \rho 1 \\
\text{CUM} &\leftarrow \text{LAST} / \land \text{LAST} - \text{LAST} / + \text{ACTIVE[GRADE]} \\
\text{ALL[URRI]} &\leftarrow \text{CUM} - (\rho \text{CUM}) \rho -1, \text{CUM}
\end{align*}
\]

A second algorithm takes advantage of the fact that \((^/B)\) produces the same result as \((\lnot \lor \lnot B)\). To perform the \lor, we need not compute the frequency count. Instead, we know that the \lor is true if the +/- is not equal to 0. The algorithm is:

\[
\begin{align*}
\text{ALL} &\leftarrow (10 \times 2 \times 5) \rho 1 \\
\text{CUM} &\leftarrow \text{LAST} / + \lnot \text{ACTIVE[GRADE]} \\
\text{ALL[URRI]} &\leftarrow \text{CUM} - (\rho \text{CUM}) \rho 0, \text{CUM}
\end{align*}
\]
A third algorithm may be employed which is based entirely on Boolean techniques. Since these techniques are discussed in the Boolean Techniques chapter, the algorithm is presented here without explanation:

\[
\text{ALL}(10 \times 2 \times 5) \rho 1 \\
\text{B} \leftarrow \text{ACTIVE[GRADE]} \\
\text{CUM} \leftarrow (\text{LAST} \geq \text{B}) / \text{LAST} \\
\text{ALL[URRI]} \leftarrow (\text{LAST}/\text{B}) \wedge \text{CUM}/-1\Phi\text{CUM}
\]

Notice that this algorithm uses no arithmetic function and makes heavy use of Boolean functions ($\geq$, $\wedge$ and $/$). In some implementations of APL, Boolean functions have been optimized to be extremely fast. In such implementations, the third algorithm will dramatically out-perform the first two. You should time the alternative algorithms in your own environment before deciding among them.

If the problem at the beginning of this section were stated instead as an or-reduction (any) problem rather than an and-reduction (all) problem, similar approaches could have been taken, with slight modifications. Here are the solutions:

[sum algorithm]

\[
\text{ANY}(10 \times 2 \times 5) \rho 0 \\
\text{CUM} \leftarrow \text{LAST} + \\text{\textbackslash ACTIVE[GRADE]} \\
\text{ANY[URRI]} \leftarrow \text{CUM} \# (\rho \text{CUM}) \rho 0 , \text{CUM}
\]

[Boolean techniques algorithm]

\[
\text{ANY}(10 \times 2 \times 5) \rho 0 \\
\text{B} \leftarrow \text{ACTIVE[GRADE]} \\
\text{CUM} \leftarrow (\text{LAST} \vee \text{B}) / \text{LAST} \\
\text{ANY[URRI]} \leftarrow (\text{LAST} / \text{B}) \geq \text{CUM} / -1\Phi\text{CUM}
\]

PROBLEM: Define the syntax of utility functions for performing frequency counts, accumulations and cross-tabulations in APL.

TOPIC: N-Way Reduction Utility Functions

We will approach this problem by first imagining an extension to APL which can solve the problems of the previous sections. Imagine an
extension to the definition of the primitive APL reduction operator (@/) so that it will accept the following dyadic syntax:

\[ r \leftarrow (dshape;civec1;civec2;...;civecN)@[d]array \]

where:

- \( array \) = array being reduced;
- \( d \) = dimension of array being reduced;
- \( dshape \) = the shape to which dimension \( d \) is "reduced"; \( dshape \) has \( N \) elements if an \( N \)-way reduction is to be performed;
- \( civeci \) = the class index vector for classification \( i \) (\( 1 \leq i \leq N \) for an \( N \)-way reduction); \( civeci \) has the same length as dimension \( d \) and its values are indices (origin dependent) which identify the class indices into which the corresponding dimension \( d \) arrays are to be grouped; the values of \( civeci \) are all elements of \( \{dshape[i] \} \);
- \( r \) = the \( N \)-way (where \( N = \rho dshape \)) @ reduction (where @ is any scalar dyadic function) on dimension \( d \) of array by classifications \( cvec1, cvec2, ..., cvecN \).

This syntax calls for multiple left arguments (\( N+1 \) for an \( N \)-way reduction) where the arguments are separated by semicolons (;) and are enclosed in parentheses. Since the monadic form of reduction is a 0-way reduction, it is equivalent to the dyadic form in which a single empty vector left argument is provided.

\[ r \leftarrow (\emptyset)@[d]array \quad \leftrightarrow \quad r \leftarrow@[d]array \]

Note that the dyadic reduction functions (except 0-way reduction) are origin sensitive since the left arguments contain indices.

We may illustrate the use of this syntax by using it to solve the problems of the previous sections. We assume origin 1.

One-Way Plus Reductions:

\[
\begin{align*}
DIA \leftarrow ((IAGE \cap IAGE) = \rho IAGE) / IAGE \\
IAGEIND \leftarrow DIA \cap IAGE \\
P \leftarrow (\rho DIA; IAGEIND) + / 1 \\
A \leftarrow (\rho DIA; IAGEIND) + / APREM
\end{align*}
\]
Chapter 7  FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABULATIONS

N-Way Plus Reductions:

\[ \text{UCLASSIND} = '\text{SABCD}'; \text{UCLASS} \]
\[ \text{SEXIND} = '\text{MF}'; \text{SEX} \]
\[ \text{IAGEIND} = 1 + \text{IAGE} / 10 \]
\[ \text{SMRY} = (10 \ 2 \ 5; \text{IAGEIND}; \text{SEXIND}; \text{UCLASSIND}) + \text{DBEN}, [1] \text{APREM}, [.5] \]

N-Way Maximum and Minimum Reductions:

\[ \text{MAX} = (10 \ 2 \ 5; \text{IAGEIND}; \text{SEXIND}; \text{UCLASSIND}) / \text{DBEN}, [.5] \text{APREM} \]
\[ \text{MIN} = (10 \ 2 \ 5; \text{IAGEIND}; \text{SEXIND}; \text{UCLASSIND}) / \text{DBEN}, [1.5] \text{APREM} \]

N-Way Logical Reductions:

\[ \text{ALL} = (10 \ 2 \ 5; \text{IAGEIND}; \text{SEXIND}; \text{UCLASSIND}) \wedge / \text{ACTIVE} \]
\[ \text{ANY} = (10 \ 2 \ 5; \text{IAGEIND}; \text{SEXIND}; \text{UCLASSIND}) \vee / \text{ACTIVE} \]

The syntax of dyadic reduction as described in here is adequate as a model for user-defined utility functions. Unfortunately, current APL systems do not allow multiple left arguments (unless they are packed together into a single "nested" array.) Also, optional arguments (e.g. the \( d \) in \( +/[d] \)) are not allowed. Therefore, we must make compromises.

One possible syntax is the following:

\[ r = (d \text{shape}, c \text{ind}1, c \text{ind}2, ..., c \text{ind}N, (d)) \text{ PLUSRED array} \]
\[ \text{MAXRED} \]
\[ \text{MINRED} \]
\[ \text{ANDRED} \]
\[ \text{ORRED} \]

where:

array = array being reduced;

d = dimension of array being reduced;

d\text{shape} = the shape to which dimension \( d \) is "reduced"; d\text{shape} has \( N \) elements if an \( N \)-way reduction is to be performed; \( d \) is optional and defaults to the last dimension of array if omitted;

c\text{indi} = the numeric suffix of the name of the global class index vector for classification \( i \) (1 \leq i \leq N for an \( N \)-way reduction); the name of the vector is 'I',*c\text{indi} (e.g. I3 or I6); the class index vectors have the same length as dimension \( d \) of array and their values are indices (origin dependent) which identify the class indices into
which the corresponding dimension d arrays are to be grouped; the values of the class index vectors are all elements of \( \text{ldshape}[i] \);

\[ r = \text{the } N\text{-way (where } N = p\text{dshape}) \odot \text{reduction (where } \odot \text{ is + for PLUSRED, } \int \text{ for MAXRED, } \ldots \text{) on dimension } d \text{ of array by the classifications identified by } cind1, cind2, \ldots, cindN. \]

In the case of a scalar right argument (e.g. 1), PLUSRED replicates the scalar to a vector with the same length as the class index vectors. The effect is to return a frequency count by class (times the value of the scalar).

We may illustrate the use of these utility functions by using them to solve the problems of the previous sections. We assume origin 1.

One-Way Plus Reductions:

\[
\begin{align*}
\text{DIA} & \leftarrow ((\text{IAGE} \odot \text{IAGE}) = \text{IAGE}) / \text{IAGE} \\
\text{I1} & \leftarrow \text{DIA} \odot \text{IAGE} \\
\text{F} & \leftarrow (p\text{DIA}), 1 \quad \text{PLUSRED } 1 \\
\text{A} & \leftarrow (p\text{DIA}), 1 \quad \text{PLUSRED APREM}
\end{align*}
\]

N-Way Plus Reductions:

\[
\begin{align*}
\text{I2} & \leftarrow 'SABCD' \cup \text{CLASS} \\
\text{I3} & \leftarrow 'MF' \cup \text{SEX} \\
\text{I4} & \leftarrow 1 + \text{IAGE} + 10 \\
\text{SMRY} & \leftarrow (10 2 5, 4 3 2) \quad \text{PLUSRED } \text{DBEN}, [1] \text{APREM}, [0.5] 1
\end{align*}
\]

N-Way Maximum and Minimum Reductions:

\[
\begin{align*}
\text{MAX} & \leftarrow (10 2 5, 4 3 2) \quad \text{MAXRED } \text{DBEN}, [0.5] \text{APREM} \\
\text{MIN} & \leftarrow (10 2 5, 4 3 2, 1) \quad \text{MINRED } \text{DBEN}, [1.5] \text{APREM}
\end{align*}
\]

N-Way Logical Reductions:

\[
\begin{align*}
\text{ALL} & \leftarrow (10 2 5, 4 3 2) \quad \text{ANDRED } \text{ACTIVE} \\
\text{ANY} & \leftarrow (10 2 5, 4 3 2) \quad \text{ORRED } \text{ACTIVE}
\end{align*}
\]

Notice that the following pairs of expressions produce identical results:
Chapter 7  FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABULATIONS

The writing of these utility functions is left as an exercise at the end of this chapter.

---

PROBLEM: Suppose you wish to compute the 3-way plus reduction by age, sex and underwriting class of death benefits, annual premiums and frequency counts (as presented in the N-Way Plus Reduction section above.) How would you do this on one million policies?

TOPIC: N-Way Reductions on Files

So far, we have considered only arrays which can be easily manipulated within the active workspace. In this problem however, we would need several 1,000,000 element vectors: IAGE, SEX, UCLASS, DBEN, APREM. In many implementations of APL, the active workspace is not large enough to contain these variables. They must be broken into smaller pieces and stored on a file.

The chapter, File Design and Utilities, discusses the application of APL files for storing and manipulating large amounts of information. In that chapter, a number of alternative file organizations are described. One of them, the multi-set transposed file organization, is a likely candidate for storing the one million policies introduced above. Using that organization, the information is broken into smaller pieces and each piece is stored as a single file component. Suppose the pieces are 5000 element vectors. The file will consist of 1000 components (200 sets of 5 variables), each component containing a 5000 element vector.

To compute the required 3-way reduction, you will read in one set of the 5 variables (i.e. 5000 policies) at a time and apply the PLUSRED function on them. Accumulate the results as you go. After 200 iterations (i.e. sets), you will be done.

The file utility function EXECUTE is designed for this type of problem. It reads from file one set of information at a time for specified variables and then performs any specified computations on those variables. The left argument of EXECUTE identifies the file being used (FP) and the variables ("fields") required. The right argument of EXECUTE is a character vector representation of an APL
expression to be executed once for each set. The variables F1, F2,... (where the n in Fn is included in the list of field numbers in the left argument) are assigned the values of the respective fields for the current set.

Suppose the variables required for this problem are located in the following fields of the file:

<table>
<thead>
<tr>
<th>Field Number</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>IAGE</td>
</tr>
<tr>
<td>4</td>
<td>SEX</td>
</tr>
<tr>
<td>9</td>
<td>UCLASS</td>
</tr>
<tr>
<td>12</td>
<td>DBEN</td>
</tr>
<tr>
<td>13</td>
<td>APREM</td>
</tr>
</tbody>
</table>

You can add up the APREM variable (field 13) for all records on file with the following statements:

```
SUM=0
(FP,13) EXECUTE 'SUM=SUM+/F13'
```

The EXECUTE function will execute the expression SUM=SUM+/F13 once for each set on file. Before executing the expression, it will read from file the 5000 element vector of APREM values for the current set and will assign it to the variable name F13 (because the number 13 is in the left argument of EXECUTE).

To perform the desired 3-way reduction, we write the following function:

```
\^ SUM=XTAB;I3;I4;I9
[1] \^ Returns the 3-way plus reduction by age, sex and
[2] \^ underwriting class of death benefits, annual
[3] \^ premiums and frequency counts. Requires globals:
[4] \^ F3 (IAGE), F4 (SEX), F9 (UCLASS), F12 (DBEN),
[6] \^ Which age class?
[7] \^ I3=1+I3+10
[8] \^ Which sex class?
[9] \^ I4='MF'+I4
[10] \^ Which underwriting class?
[11] \^ I9='SABCD'+I9
[12] \^ Perform the 3-way plus reduction:
[13] \^ SUM+(10 2 5, 3 4 9) PLUSRED F12,[1]F13,[0,5]1
\`
```

Then we use EXECUTE to execute this function once for each set of 5000 policies:

```
SUM=0
(FP,3 4 9 12 13) EXECUTE 'SUM=SUM+XTAB'
```
PROBLEM: Suppose you need to perform a 2-way plus reduction by A and B, another by B and C, and another by A and C. You could perform the 3-way plus reduction by A, B and C and then respectively plus reduce dimensions 3, 1 or 2 of the result to generate the desired arrays. Taking this approach to its extreme, you would always perform a single n-way plus reduction by every possible classification and then use monadic plus reduction to eliminate those dimensions not needed for particular reports. What are the problems which arise from taking this extreme approach? How can the problems be overcome?

TOPIC: Milky-Way Reductions

We define a new term:

Milky-Way reduction: an n-way reduction in which you reduce by every classification variable under the sun, generating a result which may have an astronomical number of elements.

Performing Milky-Way reductions can improve your productivity and reduce computer processing time consumed. Compare the following:

17: IAGE=10 (by IAGE)
18: 'MF'=SEX (by SEX)
19: 'SABCD'=UCLASS (by UCLASS)

Milky-Way: SMRY=(10 2 5, 7 8 9) PLUSRED APREM

By SEX and UCLASS:

N-way: (2 5, 8 9) PLUSRED APREM
Milky-Way: +/[1]SMRY

By IAGE and UCLASS:

N-way: (10 5, 7 9) PLUSRED APREM
Milky-Way: +/[2]SMRY

By Sex:

N-way: (2, 8) PLUSRED APREM
Milky-Way: +/[1]+/[3]SMRY

Unfortunately, the number of elements in the result of a Milky-Way reduction may be astronomical. Your active workspace may not be big.
enough to contain them. Even if it can contain them, the process of plus reducing the array to a manageable size will be costly and time consuming because of the tremendous number of values.

Ironically, the majority of the values are zeros. For example, if you perform a 7-way reduction in which the 7 classifications respectively involve 5, 6, 7, 8, 9, 10 and 11 classes, the result (on a vector) will contain 1,663,200 elements \((5\times6\times7\times8\times9\times10\times11)\), regardless of the length of the vector being reduced. If the vector contains 5000 elements, the result will contain at most 5000 nonzero values. If the entities represented by the 5000 elements are somewhat similar to one another, many of the entities will be classified the same. Then there will be fewer than 5000 nonzero values in the result, perhaps much fewer.

If we discard the zeros and retain just the nonzero values, the result of a Milky-Way reduction is more manageable. Of course, we must keep track of where the nonzero values belong in the result.

We propose the following functions:

\[
\Delta r \leftarrow (\text{dshape}, \text{cind}_1, \text{cind}_2, \ldots, \text{cind}_N, (d)) \quad \Delta \text{PLUSRED array}
\]

\[
\Delta \text{MAXRED}
\]

\[
\Delta \text{MINRED}
\]

\[
\Delta \text{ANDRED}
\]

\[
\Delta \text{ORRED}
\]

The meaning and syntax of these functions are identical to those of the PLUSRED, MAXRED, MINRED, ANDRED and ORRED functions introduced earlier in the chapter. The only difference between those functions and these is the result. The result of these functions is a "compressed Milky-Way array" which is a numeric vector whose elements are defined as follows (origin 0):

\[
\Delta r[0] \quad R \quad \text{rank of the array right argument}
\]

\[
\Delta r[1] \quad D \quad \text{dimension being reduced (origin 0)}
\]

\[
\Delta r[2] \quad N \quad \text{shape of dshape, i.e. the N for an N-way reduction}
\]

\[
\Delta r[(3+R)+1] \quad S \quad \text{shape of the reduced, but not expanded (i.e. zero-filled), result}
\]

\[
\Delta r[(3+R)+N] \quad DS \quad \text{resulting shape of dimension reduced, i.e. dshape}
\]

\[
\Delta r[(3+R+N)+1S[D]] \quad \text{RIND -- indices (origin 0) into the raveled dshape dimensions of the existing (i.e. nonzero) values}
\]

\[
\Delta r[(3+R+N+S[D])+1\times/S] \quad \text{DATA -- raveled existing (i.e. nonzero) values}
\]

Since the results of these functions are not in very useful forms, we need another set of utility functions to convert them back to the more familiar multi-dimensional forms (including zeros where appropriate). We propose the following functions:
Chapter 7 FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABULATIONS

r<ways ΔPLUSWAY Δr
ΔMAXWAY
ΔMINWAY
ΔANDWAY
ΔORWAY

The right argument of these functions is a compressed Milky-Way array as returned by the corresponding functions ΔPLUSRED, ΔMAXRED, ΔMINRED, ΔANDRED or ΔORRED. The left argument is a vector of the ways (i.e. indices from \(N\) for an \(N\)-way reduction) to be returned. The result is the normal \(X\)-way reduction (where \(X=p\), ways) as returned by the functions PLUSRED, MAXRED, MINRED, ANDRED or ORRED.

For example:

\[ A \leftarrow (5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11, \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7) \ \DeltaPLUSRED \ NVEC \]
\[ \rho \leftarrow 1 \]
\[ 6 \ 8 \ 10 \]
\[ ρ \leftarrow 1 \]
\[ 5 \ 11 \]
\[ ρ \leftarrow 2 \]
\[ 6 \]
\[ ρ_0(l) \]
\[ 0 \]

Finally, we need to address the problem of performing Milky-Way reductions on files. When using PLUSRED, we execute an expression like the following, once for each set of, say, 5000 records on file:

\[ \text{SUM} \leftarrow \text{SUM} + A \ \text{PLUSRED} \ B \]

This solution will not work with the Milky-Way reduction functions since the values in \(\text{SUM}\) (after the first set) and the values in the result of \(\DeltaPLUSRED\) are not corresponding data values. They are, instead, both compressed Milky-Way arrays. We propose the following functions:

\[ ΔC \leftarrow ΔA \ \DeltaPLUS \ ΔB \]
\[ ΔMAX \]
\[ ΔMIN \]
\[ ΔAND \]
\[ ΔOR \]

These functions perform the \(+, \|, \|, \land\) and \(\lor\) functions, respectively, between two compressed Milky-Way arrays, returning a compressed Milky-Way array. If either argument is an empty array, the other argument is returned.

The solution for working with files, then, needs to be modified only slightly when performing Milky-Way reductions:

\[ Δ\text{SUM} \leftarrow l0 \] (before loop)
\[ Δ\text{SUM} \leftarrow Δ\text{SUM} \ \DeltaPLUS \ A \ \DeltaPLUSRED \ B \] (within loop)

-113-
The writing of these functions is left as an exercise below.

---

PROBLEMS: (Solutions on pages 340 to 361)

1. Given a 500 element character vector TZONE whose values represent the time zones (E, C, M, P, H) in which each of your 500 fast food restaurants are located, how many restaurants are located in each time zone?

2. If SALES is a 500 element numeric vector whose values represent the annual sales, in dollars, of the corresponding 500 restaurants of the prior problem, what are the annual sales by time zone?

3. If TYPE is a 500 element character vector whose values represent the restaurant types (B, C, P, S) of the corresponding 500 restaurants, how many (FRQ) restaurants of each type are located in each time zone? What are the annual sales (AMT) for each of these type/zone breakdowns? How large (MAX) was the largest restaurant in each of these type/zone breakdowns?

4. Write one or more of the utility functions PLUSRED, MAXRED, MINRED, ANDRED, ORRED defined in this chapter. Compare your functions to the listings of those functions included in the solutions at the back of the book.

5. Write one or more of the utility functions PLUSRED, MAXRED, MINRED, ANDRED, ORRED, PLUSWAY, MAXWAY, MINWAY, ANDWAY, ORWAY, PLUS, MAX, MIN, AND, OR defined in this chapter. See the listings at the back of the book.
Given the 500 element vectors TZONE, SALES and TYPE defined in the questions above, suppose you also have the following data:

STATE  500 row, 2 column character matrix of state (postal code) abbreviations indicating the states in which the corresponding 500 restaurants are located.

MGR  500 element integer vector whose values represent the regional managers responsible for the corresponding 500 restaurants; there are 6 managers and their respective numeric codes are 301, 304, 310, 322, 329 and 333.

FIT  500 element numeric vector whose values represent the annual federal income tax, in dollars, of the corresponding 500 restaurants.

Using the utility functions presented in this chapter, generate the following information:

1. Number of restaurants whose annual sales were $0 to $1 million, $1 million to $5 million, $5 million and up.

2. Number of restaurants, total sales and total FIT by state (given a 50 row, 2 column matrix ALLSTATES of the distinct state postal codes).

3. Total sales and total FIT by manager, annual sales volume (0-1, 1-5, 5+ million) and type of restaurant.

4. Number of restaurants by state and type.

5. FIT by type and sales volume.
An interactive function is one which "prompts" you to enter information at the keyboard. In this chapter we discuss the primitive capabilities available in APL for writing interactive functions. We also define utility functions which make the primitive functions easier to apply and friendlier to use.

PROBLEM: Write a function ASKUSER which prompts you to "ENTER EMPLOYEE NAME", assigns the character vector response to the variable NAME, prompts to "ENTER SALARY" and assigns the numeric scalar response to SALARY.

TOPIC: Primitive Interactive Functions

The primitive niladic APL functions 0 (quad) and ~ (quote-quad) allow user interaction. When invoked, both functions causes execution to pause while you type information at the keyboard. When you press the RETURN (or ENTER) key, the typed information is returned explicitly and execution resumes. Graphically, the 0 and ~ represent "windows" through which information passes to the computer from the outside world. In general, 0 is used to enter numeric information and ~ is used to enter character information.

A simple solution to the stated problem follows:

\[
\begin{align*}
\text{ASKUSER} \\
[1] & \ 'ENTER \ EMPLOYEE \ NAME' \\
[2] & \ NAME\+\quad \\
[3] & \ 'ENTER \ SALARY' \\
[4] & \ SALARY\+\quad \\
\end{align*}
\]
On the 1st and 3rd lines, the prompts are expressed as constant character vectors. Since the vectors are not assigned to variable names or otherwise used as arguments to functions, they are displayed. This behavior is one of the great simplifications in APL: to generate output, just construct an array (as a constant or as the result of an expression) and do not assign it to a variable or otherwise use it. This is why the expression 2+3 causes 5 to display but the expressions A+2+3 or 6×2+3 do not cause 5 to display, even though the same 2+3 operation is being performed.

This convention for generating output in APL is a mixed blessing. While it is simple to generate output, it is sometimes unclear from context whether or not output is being generated. For example, does the following function line generate output?

\[15\] CRUNCH I

The answer depends upon whether or not the monadic function CRUNCH returns an explicit result. If so, the result is not being assigned and so will be displayed. If not, nothing will appear (unless output is generated during the execution of CRUNCH). Because of this lack of clarity, some APL programmers choose to show output explicitly by assigning it to \(D\). For example:

\[15\] \(D\)\(\leftarrow\)CRUNCH I

Note that the "window" analogy still holds when using \(D\) in this context. Now information is passing from the computer to the outside world. Not only does the \(D\)\(\leftarrow\) convention add clarity, it also enables you to locate (under program control) occurrences of output in case you wish to direct output elsewhere (say, replace \(D\)\(\leftarrow\) by 'OUTPUT') or turn it off altogether (say, replace \(D\)\(\leftarrow\) by '0 \(0_\rho\)').

Using this convention, let us rewrite our simple solution:

\[\begin{align*}
&v \text{ ASKUSER} \\
&[1] \(D\)\(\leftarrow\)’ENTER EMPLOYEE NAME’ \\
&[2] \text{NAME}\(\leftarrow\)\(D\) \\
&[3] \(D\)\(\leftarrow\)’ENTER SALARY’ \\
&[4] \text{SALARY}\(\leftarrow\)\(D\) \\
&v
\end{align*}\]

When executing this function, you will see the following:

ENTER EMPLOYEE NAME

where the "_" symbol represents the location of the cursor (or print mechanism) while the computer is awaiting your response.

Why is the cursor located at the beginning of the line below the prompt? Output in APL (via \(D\)\(\leftarrow\) or automatic output) is automatically followed by a "carriage return" (i.e. a newline). The only way to suppress the succeeding carriage return is by assigning the output to
Chapter 8  WRITING USER-FRIENDLY INTERACTIVE FUNCTIONS

so, the other "window". In fact, the inclusion or exclusion of the carriage return is the only difference between ⍋ and ⍈. For example, consider the following function:

```
∇ DISPLAY
[1] ⍋'VALUES: '
[2] ⍋2 3 4
[3] ⍋'.'
∇
```

When you execute this function, you will see the following:

VALUES: 2 3 4.

Notice that the last statement uses ⍋, causing the cursor to move to the start of the next line for any subsequent output (including the 6 space indent you get in immediate execution mode).

How can we modify our solution to display the prompt and leave the cursor beyond the prompt on the same line?

ENTER EMPLOYEE NAME: _

Given the behavior of ⍋ described above, we are compelled to try the following:

```
[1] ⍋'ENTER EMPLOYEE NAME: '
[2] NAME←⍋
```

Sure enough, it behaves as we want it to:

ENTER EMPLOYEE NAME: LANDER, KEVIN

Unfortunately, when we check the value of the variable NAME, we find that it contains not only the name but the prompt as well:

```
NAME
34
ENTER EMPLOYEE NAME: LANDER, KEVIN
```

The ⍋ function returns every character on the line, whether put there by ⍋ or by user entry. (Some APL systems return the ⍋ characters as blanks or stars or other designated characters.) Clearly, we need to drop the prompt characters from the result of ⍋:

```
[1] ⍋'ENTER EMPLOYEE NAME: '
[2] NAME←21↓⍋
```

The number 21 is the length of the prompt (including the trailing blank).
Can we apply the same technique to the SALARY prompt? No. Quad (¶) input causes the characters "0:" to display. For example:

```
ENTER SALARY
0:  
```

If we use ¶←'ENTER SALARY: ', all we accomplish is to put the "0:" at the end of the prompt line:

```
ENTER SALARY: 0:  
```

If we want to eliminate the "0:" from the prompt, we must use ¶ input. However, the result will then be character valued. To convert the characters to the numbers they represent, we must use execute (¶). Our final solution is therefore:

```
V ASKUSER
[1] ¶←'ENTER EMPLOYEE NAME: '
[2] NAME←21¶
[3] ¶←'ENTER SALARY: '
[4] SALARY←$14¶
V

```

PROBLEM: Write utility functions which prompt for character vectors or numeric vectors and which handle the problems associated with each.

TOPIC: Utility Interactive Functions

First, we will define a monadic function CPROMPT (character prompt) which will display its character vector right argument as a prompt and will allow you to enter a response at the end of the same line, returning your response as a character vector.

```
V R-CPROMPT PROMPT
[1] A Displays character vector PROMPT, allows
[2] A keyboard input on same line and returns
[4] ¶←PROMPT
[5] R←(ρ,PROMPT)↓¶
V

```
Note the use of ravel (,) on the 2nd line to handle the case in which PROMPT is a scalar (e.g. R←CPROMPT '?').

Given this utility function, the solution to the problem of the last section may be rewritten:

\[
\begin{align*}
\text{v ASKUSER} \\
[1] & \text{NAME←CPROMPT 'ENTER EMPLOYEE NAME:'} \\
[2] & \text{SALARY←CPROMPT 'ENTER SALARY:'} \\
\end{align*}
\]

This function is an improvement over the previous solution. However, problems remain. If you accidentally type a non-numeric character (e.g. 3B5) in response to the 'ENTER SALARY:' prompt, an APL error message will appear (from ⌥) and the function will suspend. This is not user-friendly behavior.

We could use ⌥ input ("evaluated input") to avoid the suspension. When an error occurs in evaluated input mode, the error message is displayed and you are reprompted. For example:

\[
\begin{align*}
\text{ENTER SALARY} \\
\text{(Console prompt)} & \text{:
} \\
\text{3B5} \\
\text{VALUE ERROR} \\
\text{(Console prompt)} & \text{:
} \\
\text{3B5} \\
\text{^} \\
\text{(Console prompt)} & \text{:
}
\end{align*}
\]

However, evaluated input mode has several disadvantages. The first is that its appearance (Console prompt:) is odd to a naive user. The second is that the error messages are technical APL messages (e.g. SYNTAX ERROR), not user-oriented messages (e.g. DEPARTMENT NUMBER MUST BE NUMERIC). The third is that you may inadvertently invoke other functions in the workspace. The fourth is that the response will not be accepted on the same line as the prompt.

Therefore, we will stick with ⌥ input ("character input"). We will define a function NINPUT (numeric input) which will display its character vector prompt right argument and will allow you to enter a response at the end of the same line. The response will be converted to numbers if possible and returned. If not possible, the message '** ENTER NUMBERS ONLY **' will be displayed and you will be reprompted.

How then do we convert a character vector which looks like numbers (e.g. '67 15') into a numeric vector (e.g. 67 15)? A number of APL implementations (e.g. APL*PLUS, SHARP APL) have available the companion monadic functions OFI (fix input or format inverse) and OVI (verify input). The OFI function is just what we are looking for. It converts its character vector or scalar right argument into a numeric vector result. Each group of contiguous nonblank characters is converted into a single numeric element. If the group of
characters does not represent a valid number, it is returned as 0. For example:

\[
\text{OFI '67.5 3B6 5 0 HI'}
\]

67.5 0 5 0 0

Since invalid groups are returned as 0, another function is required to tell the good 0s from the bad 0s. This is what DVI does. The DVI function converts its character vector or scalar right argument into a Boolean vector result. Each group of contiguous nonblank characters is converted into a single bit: 1 if a valid number, 0 if not. For example:

\[
\text{DVI '67.5 3B6 5 0 HI'}
\]

1 0 1 1 0

The OFI and DVI functions always return vectors, never scalars. For example:

\[
\rho\text{OFI '6'}
\]

1

\[
\rho\text{DVI ' '}
\]

0

Using OFI and DVI, the definition of NINPUT is:

\[
\text{v R+NINPUT PROMPT}
\]

[1] aDisplays character vector PROMPT, allows
[2] akeyboard input on same line and returns
[4] L1:R←CPROMPT PROMPT
[5] →(∧/DVI R)ρL2
[6] D+'** ENTER NUMBERS ONLY **'
[7] →L1
[8] L2:R←DVI R

\[
\text{v}
\]

If OFI and DVI are unavailable in your implementation of APL, you must write an APL function which performs a similar function. Such a "parsing" function is quite complicated and is not included here.

In APL2, which does not have OFI and DVI, exception handling may be used to execute the character vector and to display appropriate messages if it cannot be successfully executed. The system function OEA (execute alternate) is used. It executes its right argument and returns the result. If an error occurs while executing its right argument, its left argument is executed.
In APL2, the definition of NINPUT is:

\[
\text{\begin{align*}
\text{V R NINPUT2 PROMPT} \\
[1] \text{a Displays character vector PROMPT, allows} \\
[2] \text{a keyboard input on same line and returns} \\
[3] \text{a numeric vector response. Requires CPROMPT.} \\
[4] \text{L1: R CPROMPT PROMPT} \\
[5] \text{a Return empty numeric vector if all-blank response:} \\
[6] \text{pL2} \\
[7] R \leftarrow 0 \\
[8] \text{R} \rightarrow 0 \\
[9] \text{a Allow only characters which may be parts of numbers} \\
[10] \text{a (so that other functions will not be executed):} \\
[11] \text{L2: (N/R \epsilon '0123456789.E') pL3} \\
[12] \text{a Make sure any E (exponential notation) is not} \\
[13] \text{a preceded by a blank:} \\
[14] \text{R} \rightarrow (V/\text{ E'g', }R) pL3 \\
[15] \text{a Ravel when assigning to insure a vector result:} \\
[16] \text{' R', L3 OEA 'R', R} \\
[17] \text{R} \rightarrow 0 \\
[18] \text{L3: (} \leftrightarrow \text{' ** ENTER NUMBERS ONLY **' } \\
[19] \text{L1} \\
\end{align*}}
\]

PROBLEM: Write a function ASKUSER which prompts for employee name, salary and project codes (numeric) and assigns the respective responses to NAME, SALARY and CODES. Verify that the name is not all-blank, that the salary is a positive integer less than 100,000 and that no more than 5 project codes are entered. Terminate ASKUSER immediately if the user types END in response to any question. The explicit result of ASKUSER is 0 if END is typed and is 1 otherwise.

TOPIC: Utility Validation Functions

Let us take a reverse-engineering approach to this problem. We will write the ASKUSER function, employing imaginary utility functions as needed. Then, we will write the utility functions. Here is the finished solution:
\[
\text{This function was easy to write and is easy to read. Comments are unnecessary. Our task now is to write the utility functions such that the function is "user-friendly" as well.}
\]

The \text{CPROMPTE} (character prompt with escape) function behaves like the \text{CPROMPT} function written earlier with one exception. It checks for the "escape" word END and returns the numeric scalar 1 if entered. Otherwise, it returns the character vector entered via the keyboard. The \text{CPROMPTE} function is listed below.

Since some applications permit the use of more than one escape word (e.g. END, QUIT, BACKUP, ABORT, HELP, PRINT, etc.), we have written \text{CPROMPTE} to illustrate the use of several escape words, specifically END, BACKUP, and ABORT. \text{CPROMPTE} returns the scalar 1, 2 or 3 if the respective escape word is entered.
The **ESCAPE** function is a dyadic function which checks to see if its right argument is a scalar. If so, it returns its label left argument. If not, it returns an empty vector so that no branch will take place. The **ESCAPE** function is listed below.

The **ESCAPE** function has been written to accommodate multiple escape words. For example, the expression,

\( \rightarrow (L99, L1, 0) \) **ESCAPE NAME**<CPROMPTE 'ENTER EMPLOYEE NAME: ' \( \)

will cause a branch to one of the "labels" L99, L1 or 0 if the corresponding escape word END, BACKUP or ABORT is entered. If a single label is provided as **ESCAPE**'s left argument, that label is returned if any of the escape words is entered.
Chapter 8 WRITING USER-FRIENDLY INTERACTIVE FUNCTIONS

[WSID: INPUT]
\[
\begin{align*}
\text{R} & \leftarrow \text{LABELS} \text{ ESCAPE CODE} \\
[1] & \text{Used as:} \\
[2] & \text{R} \\
[3] & \rightarrow (L1, L2, 0) \text{ ESCAPE NAME} \leftarrow \text{CPROMPTE} \ '\text{ENTER NAME:}' \\
[4] & \text{R} \\
[5] & \text{Returns LABELS(\text{CODE}) if code is a scalar.} \\
[6] & \text{Otherwise, returns \text{\{0\} so no branch occurs.} \\
[7] & \text{If LABELS is a singleton, it is returned for} \\
[8] & \text{any scalar CODE.} \\
[9] & \text{Return empty vector for non-scalar CODE:} \\
[10] & \text{R} \leftarrow \{0\} \\
[11] & \rightarrow (x \rho \text{CODE}) \rho 0 \\
[12] & \text{Return LABELS for singleton LABELS:} \\
[13] & \text{R} \leftarrow \text{LABELS} \\
[14] & \rightarrow (1 \wedge \rho \text{LABELS}) \rho 0 \\
[15] & \text{Otherwise, return label for corresp escape code:} \\
[16] & \text{R} \leftarrow \text{LABELS(\text{CODE})} \\
\end{align*}
\]

The IF function is the standard conditional branching function.

[WSID: INPUT]
\[
\begin{align*}
\text{R} & \leftarrow \text{L IF C} \\
[1] & \text{Conditional branch function. Used as:} \\
[2] & \text{R} \\
[3] & \rightarrow \text{LABEL IF } I > 50 \\
\end{align*}
\]

The MESSAGE function is a dyadic function which returns its Boolean left argument and which displays its character vector right argument only if the left argument is 1.

[WSID: INPUT]
\[
\begin{align*}
\text{R} & \leftarrow \text{BIT MESSAGE CVEC} \\
[1] & \text{Displays err msg CVEC if BIT=1. Used as:} \\
[2] & \text{R} \\
[3] & \rightarrow \text{ASK IF } (X < 0) \text{ MESSAGE } '\text{VALUE IS NEGATIVE}' \\
[4] & \text{R} \\
[5] & \text{R} \leftarrow \text{BIT} \\
[6] & \rightarrow \text{BIT} \downarrow 0 \\
[7] & \text{G} \leftarrow \text{CVEC} \\
\end{align*}
\]

The NPROMPTE (numeric prompt with escape) function is dyadic. Its left argument is the number of numbers required. Since a check for the exact number of numbers entered (usually 1) is the most common numeric input check, we will build the check into the function. If the left argument is 0, we will accept any number of numbers. Since the result of NPROMPTE is passed as the right argument of ESCAPE, we will use CPROMPTE within NPROMPTE and will return a numeric scalar.
(escape) result directly instead of the normal numeric vector response.

The NPROMPTE function for APL*PLUS or SHARP APL follows:

```
[WSID: INPUT]
\[ R[NUM NPROMPTE PROMPT \\
 1] \Display{Displays character vector PROMPT, allows} \\
 2] \Display{keyboard input on same line and returns} \\
 3] \Display{numeric vector response of length NUM} \\
 4] \Display{or of any length if NUM=0). Returns} \\
 5] \Display{numeric scalar escape code if escape word} \\
 6] \Display{entered. Requires: CPROMPTE.} \\
 7] L1:R[NUM CPROMPTE PROMPT \\
 8] \Display{Exit if scalar escape code:} \\
 9] ->(RpR)↓0 \\
10] ->(^/I\[V R)/L2 \\
11] \Display{ENTER NUMBERS ONLY **} \\
12] ->L1 \\
13] L2:R[NUM DFI R \\
14] \Display{Exit if NUM is 0 or is length of input:} \\
15] ->NUM↓0 \\
16] ->(NUM=R)/0 \\
17] \Display{ENTER ',(NUM),', NUMBER',(NUM=1)↓'S **} \\
18] \Display{} \\
\]
```
The NPROMPTE function for APL2 follows.

```
[WSID: INPUT]

   \( V \) R←NUM NPROMPTE2 PROMPT
[1] \( \alpha \) Displays character vector PROMPT, allows
[2] \( \alpha \) keyboard input on same line and returns
[3] \( \alpha \) numeric vector response of length NUM
[4] \( \alpha \) (or of any length if NUM=0). Returns
[5] \( \alpha \) numeric scalar escape code if escape word
[6] \( \alpha \) entered. Requires: CPROMPTE.
[7] L1:R←CPROMPTE PROMPT
[8] \( \alpha \) Exit if scalar escape code:
[9] \( \rightarrow (p\rho R) \downarrow 0 \)
[10] \( \alpha \) Return empty numeric vector if all-blank response:
[11] \( \rightarrow (Rv.\#'')\rho L2 \)
[12] R←L0
[13] \( \rightarrow L4 \)
[14] \( \alpha \) Allow only characters which may be parts of numbers
[15] \( \alpha \) (so that other functions will not be executed):
[16] L2:=(^/Re'0123456789.-E')\downarrow L3
[17] \( \alpha \) Make sure any E (exponential notation) is not
[18] \( \alpha \) preceded by a blank:
[19] \( \rightarrow (v'/ E'g', R)\rho L3 \)
[20] \( \alpha \) Ravel when assigning to insure a vector result:
[21] \( \rightarrow L3 \) DEA 'R+',',R
[22] \( \rightarrow L4 \)
[23] L3:D←''** ENTER NUMBERS ONLY **''
[24] \( \rightarrow L1 \)
[25] \( \alpha \) Exit if NUM is 0 or is length of input:
[26] L4:←NUM\downarrow 0
[27] \( \rightarrow (NUM=p\rho R) / 0 \)
[28] D←''** ENTER ',($NUM),' NUMBER',(NUM=1)\downarrow 'S **''
[29] \( \rightarrow L1 \)
```

-127-

PROBLEMS:  
(Solutions on pages 362 to 363)

1. Write a function LPROMPTE (letter prompt with escape) which
prompts for a single letter. The right argument is the character
vector prompt. The left argument is a character vector of
allowable single characters which the user may enter. The result
is a one element vector index into the left argument of the
character entered or is the numeric scalar escape code if an escape word is typed. To illustrate:

\[\begin{align*}
[10] & \rightarrow 0 \text{ ESCAPE } \text{ACTION} \leftarrow 'ACD' \text{ LPROMPT 'ADD, CHANGE, DELETE:'} \\
[11] & \rightarrow (\text{ADD,CHANGE,DELETE})[\text{ACTION}] 
\end{align*}\]

2. Suppose you are writing interactive functions for a user who does not have an APL terminal. Without an APL terminal, the user cannot enter a negative symbol \((-\)\). What modifications would you make to the utility functions described in this chapter to allow the user to enter a minus symbol \((-\)\) for negative numbers (e.g. \(-38\)?)

3. Write a niladic function \texttt{PROPOSAL} which generates a proposal for life insurance as follows:

\texttt{PROPOSAL}

\texttt{NAME: Fred}

\texttt{NUMBER OF KIDS: 3}

\texttt{AGES OF KIDS: 3 4 8}

\texttt{PRESS ENTER WHEN READY...} (press ENTER key)

Dear Fred:

As a proud parent of 3 kids (whose average age is 5), you need insurance.

(press ENTER key)

GENERATE ANOTHER PROPOSAL? N

Use the utility functions developed in this chapter.
Chapter 9

MANIPULATING DATES

In the field of data processing, one of the more commonly processed forms of data is dates. Dates tell us when employees were hired, when bonds mature, when insurance policies take effect, when commissions are due, when materials must be reordered, when expenses are incurred, and so on. Despite the many uses of dates, the number of different tasks performed on dates is small. This chapter discusses those tasks: representing dates in APL, entering dates, displaying dates and manipulating dates.

PROBLEM: Given that APL supports only two datatypes (numbers and characters), how should dates be represented?

TOPIC: Representation of Dates in APL

Suppose you wish to keep track of the date March 22, 1986. What are the different possible conventions you might employ to store this date? Here are several:

1. DATE←'MARCH 22, 1986' ('MONTH DD, YYYY')
2. DATE←3221986 (MMDDYYYY)
3. DATE←19860322 (YYYYMMDD)
4. DATE←860322 (YYMMDD)
5. DATE←1986 3 22 (YYYY MM DD)
6. DATE←1986 81 (YYYY DDD, days from December 31 of the previous year)
7. DATE←31127 (Days from December 31, 1899)

In order to choose the best convention, you must consider the ways in which the date is to be used. Different representations are better for different applications.
For example, the first representation (DATE←'MARCH 22, 1986') is ideal if all you want to do with the date is display it. However, the representation requires 14 bytes (characters) of storage which is more than any of the other representations and it does not lend itself to chronological sorting or to date arithmetic (say, adding 3 months to it).

The second representation (DATE←3221986) requires less storage (4 or 8 bytes depending upon the APL implementation) and is still fairly easy to display in a meaningful form (say 3/22/1986). However, it also requires transformation before it can be sorted or used in date arithmetic (since 3221986 is greater than 1221987 but 3/22/1986 occurs earlier than 1/22/1987).

The third representation (DATE←19860322) requires the same storage as the prior representation and is fairly easy to display in a meaningful form (say 1986/03/22) and can be sorted with or compared to other dates directly, without transformation. For example, since 19870122 is greater than 19860322, it occurs later.

The fourth representation (DATE←860322) is similar to the prior representation. Its advantage is that it displays in two fewer character positions (say 86/03/22 vs. 1986/03/22), though the storage requirements are the same. Its disadvantage is that the year is ambiguous and may not sort properly when comparing to dates in the next century (e.g. 15/03/22 for 2015/03/22).

The fifth representation (DATE←1986322) has different storage requirements than the prior three representations (more or less depending upon the APL implementation). It is easier to work with for some manipulations (say, year arithmetic) but harder for others (say, comparing dates to see which is later).

The sixth representation (DATE←198681) makes day arithmetic easier but meaningful display harder. For example, it is simple to see that the date 50 days beyond 1986 81 is 1986 131. However, it is not as simple to see that 1986 131 is May 11, 1986.

The seventh representation (DATE←31127) makes day arithmetic, date comparisons and chronological sorting trivial operations. However, converting the date to a meaningful form (year, month, day) is a complex task.

Depending upon the specific requirements of your application, you may decide to pick any one of these or another form of date representation. If you are undecided, representations 3 (DATE←19860322) and 5 (DATE←1986322) are good choices. These representations seem to provide a nice balance between the extreme forms 1 and 7.
PROBLEM: Write a monadic function IPDATEMDY (input date in month, day, year order) which may be used to convert a date, as entered, into the internal representation of the date. The right argument of IPDATEMDY is the character vector representation of the date in month, day, year order and the result is the numeric scalar representation (YYYYMMDD) of the date. For example: DATE+IPDATEMDY CPROMPT 'ENTER DATE OF HIRE: ' (where CPROMPT returns the character vector response to the prompt provided as its right argument). The result is 0 if the right argument does not represent a valid date.

TOPIC: Entering and Validating Dates

To be as friendly as possible, the function must allow you to enter the date in any reasonable form. For example, if the date being entered is March 22, 1986, the right argument (i.e. your response) may be in any of the following forms:

3/22/86
3.22.1986
3 22 1986
3 22
3-22
3-22-86
and so on

To be as safe as possible, the function must verify that the date entered is a valid date. For example, some dates which should be rejected are 13/25/86 and 2-29-86.

The function follows:
Chapter 9 MANIPULATING DATES

[WSID: DATES]

\[ YYYYYMMDD\rightarrow IPDATEMDY \text{ CVEC;} DD; MM; NVEC; YY; DIO \]

A Converts the character vector representation of a date (e.g. '6/15' or '3-22-1986' or '3 22' or '3.22.86') to an integer scalar.

A The items in the right argument are in month, day, year order. The result is 0 if the date is invalid.

A Ravel CVEC in case a scalar; replace '/-' by space:

A by space:

\[ CVEC\rightarrow CVEC \]

A Set result to 0 and exit if date not valid:

\[ YYYYMMDD\rightarrow 0 \]

A Date must contain only digits and spaces (once / and - are converted):

\[ \rightarrow (^/\text{CVEC}' 0123456789')\|0 \]

A Convert character vector to numeric vector:

\[ NVEC\rightarrow ,\#CVEC \]

A Date must have 2 or 3 elements (MM DD or MM DD YY):

\[ (\rho NVEC)\in 2 3\|0 \]

A Stick on current year if omitted:

\[ (3=\rho NVEC)\rho L1 \]

A Convert YY to YYYY using current century:

\[ L1: YY\rightarrow NVEC[3] \]

\[ \rightarrow (YY>99)\rho L2 \]

\[ YY+YY+100\times[\text{DTS}[1]\div100 \]

A Validate month:

\[ L2: MM\rightarrow NVEC[1] \]

\[ \rightarrow (MM\notin 12)\|0 \]

A Validate day of month:

\[ DD\rightarrow NVEC[2] \]

\[ \rightarrow ((DD<1)\lor DD>(31 29 31 30 31 30 31 31 30 31 30 31[MM]))\|0 \]

A Check 2/29 if a leap year:

\[ \rightarrow ((MM\notin 2)\lor DD\notin 29)\rho L3 \]

A Leap year every 4 years except at centuries

A (except 4th centuries):

\[ \rightarrow ((0\notin 4|YY)\lor (0=100|YY)\lor #400|YY)\rho 0 \]

A Pack date into YYYYMMDD format:

\[ L3: YYYYMMDD\rightarrow 100|YY, MM, DD \]

-132-
Chapter 9  
MANIPULATING DATES

PROBLEM: Given the three element vector DATE which represents a single date (in YYYY MM DD format), write the APL expressions which will generate each of the following date formats (for DATE~1986 3 22):

a. MAR 22, 1986  
b. March 22, 1986  
c. 3/22/86

TOPIC: Formatting Dates for Output

In the first format (MAR 22, 1986), we must convert the month from an integer (e.g. 3) to a 3 element character vector (e.g. 'MAR'). The most direct way to do this is to first construct a 3 column character matrix which has one row per possible month:

\[
\text{MON+12 3ρ'JANFEBMARAPRILMAYJUNJULAUGSEPOCTNOVDEC'}
\]

Then we may use the month number as the row index into MON. Here is one approach (showing 1986 3 5 as MAR 05, 1986):

\[
\text{MON[DATE[2];1,"'(~2↑'0',%DATE[3]),',',%DATE[1]]}
\]

This approach does not work for a matrix (3 columns) of dates. It must be modified, as in the following:

\[
\begin{align*}
\text{DAY+2 0%DATE[3];} \\
\text{DAY["'=DAY)/1ρDAY]←'0'} \\
\text{DAY+((1ρDATE),2)ρDAY} \\
\text{MON[DATE[;2];1,"',DAY,'',',5 0%DATE[;1]}
\end{align*}
\]

If your implementation of APL supports the system function DFMT, you may construct this date format with the following:

One date:

\[
\text{MON[DATE[2];1,"'X1,ZI2,<,>,I5' DFMT (DATE[3];DATE[1])}
\]

Matrix of dates:

\[
'3A1,X1,ZI2,<,>,I5' DFMT (MON[DATE[;2];1;DATE[;3 1])
\]

In APL2, you may do the following:

One date:

\[
\text{MON[DATE[2];1,'05, 5555'%DATE[3 1]}
\]

Matrix of dates:

\[
\text{MON[DATE[;2];1,'05, 5555'%DATE[;3 1]}
\]
In the second format (MARCH 22, 1986), the entire month name is displayed. This format differs from the first in that the length of the formatted result is not fixed but depends upon the length of the month name and upon the number of digits in the day.

One approach to selecting the month name portion of the date is to build a character matrix of month names (padded to the right with blanks), extract the appropriate row and squeeze out the blanks:

```
MONTH=12 9p'JANUARY FEBRUARY MARCH APRIL MAY...
MON=MONTH[DATE[2];]
MON=(MON"' ')/MON
```

A second approach is to build a character vector of month names and a corresponding vector of the lengths of the names. The vector of lengths may be used to locate the corresponding name:

```
MONTH='JANUARYFEBRUARYMARCHAPRILMAYJUNEJULYAUGUST...'
MLEN=7 8 5 5 3 4 4 6 9 7 8 8
MON=MONTH[(0, + \MLEN)[DATE[2]] + \MLEN[DATE[2]]]
```

Using either approach, once the name is selected, the formatting of the date is simple:

```
MON,' ',$DATE[3],', ',<I>$DATE[1]
```

Generally, the second format (MARCH 22, 1986) is not used when formatting many dates at once since they will have a ragged appearance:

```
MARCH 22, 1986
MAY 3, 1986
SEPTEMBER 17, 1987
JULY 4, 1988
```

On the other hand, you may choose to align the days and years:

```
MARCH 22, 1986
MAY 03, 1986
SEPTEMBER 17, 1987
JULY 04, 1988
```

To construct this result, you can simply use the same approaches discussed for the first format (MAR 22, 1986), but create a 9 column character matrix of right-justified month names instead of a 3 column character matrix of abbreviated month names:

```
MON=12 9p' JANUARY FEBRUARY MARCH APRIL MAY...
```

In the third format (3/22/86), the month number does not need to be translated into a month name. However, the first two digits of the year must be truncated. Here is one approach:

```
($DATE[2]),'/',(\"2"'0',\$DATE[3]),'/',\"2"\$DATE[1]
```

-134-
Chapter 9

MANIPULATING DATES

This approach does not work for a matrix (3 columns) of dates. It must be modified, as in the following:

\[
\text{('}',4 \, 0\text{DATE[;1],DATE[;3]}+100\times\text{DATE[;2]}[;6 \, 7 \, 189 \, 145]
\]

If your implementation of APL supports DFMT, you may do the following:

One date:

','I2,</>,ZI2,' DFMT (DATE[2];DATE[3];100\times\text{DATE[1]})

or

'G<Z9/99/99>' DFMT 100\times\text{DATE[2 3]},100\times\text{DATE[1]}

Matrix of dates:

','I2,</>,ZI2,\text{ZI2}' DFMT (DATE[;2 3];100\times\text{DATE[;1]})

or

'G<Z9/99/99>' DFMT (100\times\text{DATE[;1]})+100\times\text{DATE[;3]}+100\times\text{DATE[;2]}

In APL2, you may do the following:

One date:

'56/06/05'~DATE[2 3 1]

Matrix of dates:

'56/06/05'~DATE[;2 3 1]

PROBLEM: Since some tasks involving dates are more easily solved from certain date representations than from others, design and write a set of date conversion functions which can be used to convert between the various date representations.

TOPIC: Manipulating Dates

Suppose you limit yourself to 4 different date representations. You will need 24 different date conversion functions to handle every possible conversion (4 types times 3 remaining types times 2 directions: to or from). If you handle 5 different date representations, you will need 40 different functions.

You can reduce the number of functions needed by assuming a "base" date representation. If you define just enough functions to be able
to convert any date representation to or from the base representation, you can do any possible conversion, though it may require two steps instead of one. For example, to convert dates from a first representation to a second, neither of which is the base representation, you can convert the dates from the first representation to the base representation and then from the base representation to the second representation.

Let us use YYYYMMDD (e.g. DATE~19860322) as the base representation. We choose this representation because it can serve a number of functions directly, without conversions:

- Such dates can be sorted chronologically using grade-up (↑) or grade-down (↓).

- Such dates can be compared chronologically (before or after) using the relational functions (=, ≠, >, <, ≥, ≤).

- Such dates can be displayed directly (or with minor formatting) and be readily interpreted by the reader.

If you prefer a different base date representation for your applications, you may modify the functions below to suit your needs.

Assuming the YYYYMMDD base date representation, we propose the following date utility functions:

1. **MMDDYYYY→TOMDY YYYYMMDD**
   Converts from YYYYMMDD format to MMDDYYYY format. For example:
   
   TOMDY 19860322 19870209
   3221986 2091987

2. **YYYYMMDD→FROMMDY MMDDYYYY**
   Converts from MMDDYYYY format to YYYYMMDD format. For example:
   
   FROMMDY 3221986 2091987
   19860322 19870209

3. **DAYS→TODAYS YYYYMMDD**
   Converts from YYYYMMDD format to number of days since February 29, 0000. For example:
   
   TODAYS 19860322 19870209
   725393 725717
4. **YYYYMMDD-FROMDAYS DAYS**

   Converts from numbers of days since February 29, 0000 to YYYYMMDD format. For example:

   ```
   FROMDAYS 725393 725717
   19860322 19870209
   ```

5. **QTS-TOQTS YYYYMMDD**

   Converts from YYYYMMDD format to 3iOTS format (i.e. YYYY MM DD). The shape of the result is the catenation of 3 and the shape of the right argument. For example:

   ```
   TOQTS 19860322 19870209
   1986 1987
   3 2
   22 9
   ```

6. **YYYYMMDD-FROMQTS QTS**

   Converts from 3iDTS format (i.e. YYYY MM DD) to YYYYMMDD format. The first element of the shape of the right argument must be 3. The shape of the result is all but the first element of the shape of the right argument. For example:

   ```
   FROMQTS 3 2 1986 1987 3 2 22 9
   19860322 19870209
   ```

7. **DAYS360-TODAYS360 YYYYMMDD**

   Converts from YYYYMMDD format to number of days since January 1, 0000, assuming a 30 days per month, 12 months per year, 360 days per year calendar (the 31st day of the month is treated like the 30th day). Financial institutions frequently assume 360 days per year. For example:

   ```
   TODAYS360 19860322 19870209
   715041 715358
   ```

8. **YYYYMMDD-FROMDAYS360 DAYS360**

   Converts from number of days since January 1, 0000 to YYYYMMDD format, assuming a 30 days per month, 12 months per year, 360 days per year calendar. For example:

   ```
   FROMDAYS360 715041 715358
   19860322 19870209
   ```
February 29, 0000 was chosen as a base date in the TODAYS and FROMDAYS functions for computational reasons. At the end of that leap day, a 400 year cycle of leap years began. The conversion from dates to days or vice versa is easier when March 1 is considered the first day of the year. When a leap year occurs, the leap day is the last day of the year.

Let us illustrate the application of these functions by using them to solve a variety of problems. The following problems assume that the variable DATES is a vector of dates whose values are assigned in the YYYYMMDD format (e.g. DATES+19860322 19870209 19851225 ...).

A. How many dates occur in 1987?

    +/(DATES=19870101)^DATES=19871231     (no conversion needed)

B. Display in MM/DD/YYYY format the dates derived by adding 30 years to each date.

    'G<Z9/99/9999>' DFMT TOMDY 300000+DATES        (assuming DFMT)
    '55/55/5555' TOMDY 30000+DATES              (assuming APL2)

C. What dates result when adding 90 days to each date?

    FROMDAYS 90+TODAYS DATES

D. Which dates occur in any September?

    (9=(TOQTS DATES)[2;])/DATES

E. Assuming a 360 day year (as in bond calculations), how many whole 6 month periods (i.e. semiannual coupons) are there from each date to the date July 4, 1995?

    (((TODAYS360 '19950704')-TODAYS360 DATES)+180

F. Display (in YYYY/MM/DD format) the dates in the past (before today's date), in reverse chronological order (present to past).

    D+(DATES<FROMQTS 3{TS})/DATES
    'G<9999/99/99>' DFMT D[\D]            (assuming DFMT)
    '555/55/55' D[\D]            (assuming APL2)
Chapter 9

G. Compute the ages (age last birthday) today of people born on each of the dates.

TODAY←3↑DTS
YMD+TOQTS DATES
(TODAY[1]-YMD[1;])-(100↑TODAY[2 3])<100↑YMD[2 3;]

The definitions of these date utility functions follow. In those instances for which two substantially different algorithms are available to perform the same task, two functions have been provided, one with the name suggested above and the second with the same name followed by 'Δ' (e.g. FROMDAYS and FROMDAYSΔ). You may want to time the alternate functions for your APL installation to determine which is faster (see the Computer Efficiency Considerations chapter).

[WSID: DATES]

\[ V \text{ MMDDYYYY} \leftarrow \text{TOMDY YYYYMDD} \]
[1] a Converts dates in form YYYYMDD to form
[3] a The steps: 19860322 \rightarrow 322 →
[4] a (32200000000-322) \rightarrow 32219860000 \rightarrow 3221986
[5] a
[6] MMDDYYYY←l(YYYYMDD+99999999×10000\mid YYYYMDD)\div10000

[WSID: DATES]

\[ V \text{ MMDDYYYY} \leftarrow \text{TOMDYA YYYYMDD} \]
[1] a Converts dates in form YYYYMDD to form MMDDYYYY
[2] a by unpacking, rotating and re-packing the digits.
[3] a The steps: 19860322 \rightarrow 1986 322 \rightarrow 322 1986 \rightarrow 3221986
[4] a
[5] MMDDYYYY←10000 \� \times \rightarrow 0 \rightarrow 10000 \ \tau \ \YYYMDD
[6] a Alternative:
[7] a MMDDYYYY←0 \rightarrow 10000 \ \theta \rightarrow 0 \rightarrow 10000 \ \tau \ \YYYMDD

[WSID: DATES]

\[ V \text{ YYYYMDD} \leftarrow \text{FROMMDY MMDDYYYY} \]
[1] a Converts dates in form MMDDYYYY to form
[3] a The steps: 3221986 \rightarrow 1986 →
[4] a (198600000000-1986) \rightarrow 198603220000 \rightarrow 19860322
[5] a
[6] YYYYMDD←l(MMDDYYYY+99999999×10000\mid MMDDYYYY)\div10000

-139-
Chapter 9

MANIPULATING DATES

[WSID: DATES]

v YYYYMMDD\---FROMMDY\-- MMDDYYYY
[1] a Converts dates in form MMDDYYYY to form YYYYMMDD
[2] a by unpacking, rotating and re-packing the digits.
[3] a The steps: 3221986 \rightarrow 322 1986 \rightarrow 1986 322 \rightarrow 19860322
[4] a
[5] YYYYMMDD\rightarrow 10000 1 +.x e 0 10000 \tau MMDDYYYY
[6] a Alternative:
[7] a YYYYMMDD\rightarrow 0 10000 \tau e 0 10000 \tau MMDDYYYY

[WSID: DATES]

v DAYS\---TODAYS YYYYMMDD;DD;YYYYMM;MM;YYYY;DIO
[1] a Converts date (YYYYMMDD) to number of days since
[3] DIO\++1
[4] DD\++100\|YYYYMMDD
[5] YYYYMM\+(YYYYMMDD-DD)\div 100
[6] MM\++100\|YYYYMM
[7] YYYY\+(YYYYMM-MM)\div 100
[8] a Treat Jan and Feb as if in prior year (to have
[9] a leap day at end of yr)
[10] YYYY\++YYYY-MM\le 2
[11] a Days from Feb. 29, 0000 to prior Feb. 28/29 (leap
[12] a year every 4th year, no leap year every 100th year,
[13] a leap year every 400th year):
[14] DAYS+(365\times YYYY)+(YYYY-MM)\div 4 \times 100 400
[15] a Add in DD days and days from prior Feb. 28/29
[16] DAYS\++DAYS\++DD+(306 337 0 31 61 92 122 153 184 214 245 275)[MM]

[WSID: DATES]

v DAYS\---TODAYS YYYYMMDD;DD;YYYYMM;MM;YYYY;DIO
[1] a Converts date (YYYYMMDD) to number of days since
[3] DD\++100\|YYYYMMDD
[4] YYYYMM\+(YYYYMMDD-DD)\div 100
[5] MM\++100\|YYYYMM
[6] YYYY\+(YYYYMM-MM)\div 100
[7] a Treat Jan and Feb as if in prior year (to
[8] a have leap day at end of year):
[9] YYYY\++YYYY-MM\le 2
[10] a Days from Feb. 29, 0000 to prior Feb. 28/29
[11] a (146097, 36524, 1461, 365 days in 400, 100,
[12] a 4, 1 year cycles):
[13] DAYS\+146097 36524 1461 365 \times 0 4 25 4 \tau YYYY
[14] a Add in DD days and days from prior Feb. 28/29:
[15] DAYS\++DAYS\++DD+(306 337 0 31 61 92 122 153 184 214 245 275)[MM]

v
\begin{verbatim}
\[ YYYMMDD\leftarrow FROMDAYS \ D; DD; IND; MM; PDAYS; RDAYS; SHAPE; Y; YYYYY; \ DIO \]
[1] a Converts number of days since Feb. 29, 0000
[2] a to date (YYYYMMDD).
[3] \ DIO\leftarrow1
[4] a Work with array as a vector and reshape when done:
[5] SHAPE\leftarrow PDAYS
[6] DAYS\leftarrow DAYS
[7] a Approximate year (only off for some 2/28,
[8] a 2/29 and 3/1 dates); 365.2425 is used because
[9] a there is a leap year every 4th year (+.25),
[10] a no leap year every 100th year (-.01), leap
[11] a year every 400th year (+.0025):
[12] YYYY\leftarrow \ DAYS/365.2425
[13] a Number of days from Feb. 29, 0000 to Feb. 28/29 of
[14] a prior year:
[15] PDAYS\leftarrow (365\times YYYY)\div4\ 100 400
[16] a Number of days from start of year to specified date:
[17] RDAYS\leftarrow DAYS-PDAYS
[18] a Branch unless year may be too small by 1 (e.g. 3/1):
[19] \rightarrow(xpIND\leftarrow (RDAYS=366)/1\rho RDAYS)\downarrow L1
[20] YYYY[IND]\leftarrow YYYY[IND]+1
[21] RDAYS[IND]\leftarrow DAYS[IND]-(365\times Y)\div4\ 100 400
[22] a Branch unless year too big by 1 (e.g. 2/29 looks
[23] a like 3/0):
[24] L1:\rightarrow(xpIND\leftarrow (RDAYS>0)/1\rho RDAYS)\downarrow L2
[25] YYYY[IND]\leftarrow YYYY[IND]+1
[26] RDAYS[IND]\leftarrow DAYS[IND]-(365\times Y)\div4\ 100 400
[27] a Determine month no. from no. days from start of yr:
[28] L2: MM\leftarrow (31 30 31 30 31 30 31 30 31 31 29 /2\Phi 12)[RDAYS]
[29] a Determine day no. from no. days from start of mon.:
[30] DD\leftarrow RDAYS-(306 337 0 31 61 92 122 153 184 214 245 275)[MM]
[31] a Correct for fact that Jan. and Feb. are treated
[32] a as if in prior yr (to have leap day at end of yr):
[33] YYYY+YYYY+MM\leq2
[34] a Repack and reshape result:
[35] YYYYMMDD\leftarrow SHAPE_\rho DD\leftarrow (100\times MM)+10000\times YYYY
\end{verbatim}
Chapter 9	 MANIPULATING DATES

[WSID: DATES]

\[ YYYYMMDD+FROMDAYS \]

\[ \text{DAYS}; L4; L400; MM; N1; N4; N100; N400; YYYY; D10 \]

[1] \( \text{A} \) Converts number of days since Feb. 29, 000 to date (YYYYMMDD).
[2] \( \text{A} \) Reduces no. days by 1 so day 0 is Mar. 1, 0000:
[3] \( \text{DAYS} \)* \( \text{DAYS}+1 \)
[4] \( \text{A} \) No. of 400 year cycles (146097 days) preceding each date:
[5] \( \text{N400} \)* \( \text{DAYS}+146097 \)
[6] \( \text{A} \) No. days since last 400 year cycle:
[7] \( \text{DAYS} \)* \( \text{DAYS}+146097 \)
[8] \( \text{A} \) Flag 400 year leap dates (e.g. Feb. 29, 1600)
[9] \( \text{DAYS} \)* \( \text{DAYS}+146097 \)
[10] \( \text{A} \) and change to Feb. 28:
[11] \( \text{L400} \)* \( \text{DAYS}+146096 \)
[12] \( \text{A} \) No. of 100 year cycles (36524 days) preceding each date:
[13] \( \text{N100} \)* \( \text{DAYS}+36524 \)
[14] \( \text{A} \) No. days since last 100 year cycle:
[15] \( \text{DAYS} \)* \( \text{DAYS}+36524 \)
[16] \( \text{A} \) Flag 4 year leap dates (e.g. Feb. 29, 1988)
[17] \( \text{DAYS} \)* \( \text{DAYS}+36524 \)
[18] \( \text{A} \) and change to Feb. 28:
[19] \( \text{L4} \)* \( \text{DAYS}+1460 \)
[20] \( \text{A} \) No. of 1 year cycles (365 days) preceding each date:
[21] \( \text{N1} \)* \( \text{DAYS}+365 \)
[22] \( \text{A} \) Increase no. days by 1 so days are 1 to 365:
[23] \( \text{DAYS} \)* \( \text{DAYS}+1 \)
[24] \( \text{A} \) Determine month no. from no. days from start of yr:
[25] \( \text{MM} \)* \((31 30 31 30 31 30 31 31 30 31 31 28) \)* \( \text{DAYS} \)
[26] \( \text{A} \) Determine day no. from no. days from start of mon.:
[27] \( \text{DAYS} \)* \( \text{DAYS}-(306 337 0 31 61 92 122 153 184 214 245 275) \)
[28] \( \text{MM} \)
[29] \( \text{A} \) Add back in leap days:
[30] \( \text{DAYS} \)* \( \text{DAYS}+L4+L400 \)
[31] \( \text{A} \) Determine year from no.s of 400, 100, 4, 1 year cycles:
[32] \( \text{YYYY} \)* \( \text{N1}+(4 \times N4)+(100 \times N100)+400 \times N400 \)
[33] \( \text{A} \) Correct for fact that Jan. and Feb. are treated as if in prior year (to have leap day at end of yr):
[34] \( \text{YYYY} \)* \( \text{YYYY}+\text{MM} \)
[35] \( \text{A} \) Pack year, month, day together:
[36] \( \text{YYYYMMDD} \)* \( \text{DAYS}+100 \times \text{MM}+100 \times \text{YYYY} \)

-142-
Chapter 9

MANIPULATING DATES

[WSID: DATES]

\[ V \] QTS±TOQTS YYYYMDD

1. Converts date (YYYMMD) to DTS format (YYYY MM DD).
2. \( QTS \leftarrow 0 \ 100 \ 100 \ \tau \) YYYYMDD

\[ V \]

[WSID: DATES]

\[ V \] YYYYMMD±FROMQTS QTS

1. Converts date from DTS format (YYYY MM DD) to YYYYMMDD format.
2. \( YYYYMMD \leftarrow 10000 \ 100 \ 1 \ +. \times \) QTS
3. Alternative:
4. \( YYY \leftarrow 0 \ 100 \ 100 \ \downarrow \) QTS

[WSID: DATES]

\[ V \] DAYS±TODAYS360 YYYYMDD

1. Converts dates in form YYYYMMDD to days since
2. January 1, 0000 assuming a 30 days per month,
3. 12 months per year, 360 days per year calendar.
4. The 31st day is treated like the 30th.
5. Alternative:
6. Change DD=31 to DD=30 and subtract 1 from all days
7. and 1 from all months:
8. \( YYYYMDD \leftarrow YYYYMDD - 101 + 31 = 1001 YYYYMDD 
9. DAYS \leftarrow 360 \ 30 \ 1 \ +. \times \) 0 100 100 \( \tau \) YYYYMDD
10. Alternative:
11. DAYS \( \leftarrow 0 \ 12 \ 30 \ \downarrow \) 0 100 100 \( \tau \) YYYYMDD

[WSID: DATES]

\[ V \] YYYYMMD±FROMDAYS360 DAYS

1. Converts days since December 30, -1 to dates in
2. form YYYYMMDD assuming a 30 days per month, 12
3. months per year, 360 days per year calendar.
4. Alternative:
5. Add 1 to all days and 1 to all months (101):
6. \( YYYYMDD \leftarrow 101 + 10000 \ 100 \ 1 \ +. \times \) 0 12 30 \( \tau \) DAYS
7. Alternative:
8. \( YYYYMDD \leftarrow 101 + 0 \ 100 \ 100 \ \downarrow \) 0 12 30 \( \tau \) DAYS

~~~~~ ~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~
Chapter 9  MANIPULATING DATES

PROBLEMS:  (Solutions on pages 364 to 365)

1. Most bonds pay semi-annual coupons (interest payments). That is, every six months the holder of the bond receives one coupon payment to compensate the holder for the use of his or her money. The bond has printed on it a maturity date, i.e. the date when the final semi-annual coupon is to be paid and when the face (par) value is to be repaid. When a bond is sold prior to maturity, the purchase date usually falls somewhere between two coupon dates. Since the seller and the buyer each hold the bond during a portion of the 6 month coupon period, each is entitled to a portion of the next coupon payment. Traditionally, the buyer pays the seller a portion (called the accrued interest) of the next coupon in addition to the agreed-upon purchase price. The number of days the bond was held by the seller is compared to the number of days the bond will be held by the buyer and the coupon is divided proportionately. The number of days is computed using a 360 day year (12 months of 30 days).

Given two vectors PDATES and MDATES which respectively represent the purchase dates and maturity dates (in YYYYMMDD representation) of a set of bonds, determine the fractions of the coupons paid for accrued interest at purchase.

2. Suppose you borrow $1000 and agree to pay .1% (.001) of the outstanding balance per day. If the variables BDATE and RDATE represent the dates (in YYYYMMDD format) on which you respectively borrow and repay the loan, how much interest do you pay?

3. Dates stored in the YYYYDDD representation (e.g. 1986081 for March 22, 1986) are sometimes called "Julian" dates. The last three digits represent the number of days from the previous December 31. Write the utility function TOYD and FROMYD which may be used to convert dates in the YYYYMMDD representation to or from the Julian representation.

4. What expressions will return the day of week today as a character vector (e.g. 'TUESDAY')? (Hint: Feb. 29, 000 was a Tuesday.)
Chapter 10

WRITING REPORTS

Report formatting in APL is an afterthought. It was an afterthought to those who designed and implemented APL. And it is frequently an afterthought to those who use APL. The APL language excels at manipulating large multi-dimensional arrays, not at inserting dollar signs and decimal points. The task of designing and implementing reports is slow and tedious and not relished by many programmers, APL or otherwise.

Excellent report formatting capabilities have evolved in the various implementations of APL over the years. These have greatly improved the productivity of the APL programmer but probably not to the extent that report formatting is fun. The capability available in APL*PLUS and in SHARP APL is DFMT. In APL2 it is format by example (dyadic $ with character vector left argument). In unenhanced versions of APL, it is format (dyadic $ with numeric vector left argument).

In this chapter, we will describe techniques and utilities which can be employed to make report formatting easier and almost enjoyable.

PROBLEM: How would you construct the following report?

CAMPBELL CARPET CLEANING
ANNUAL SUMMARY
12/31/1986

<table>
<thead>
<tr>
<th>ACCOUNT</th>
<th>BUDGET</th>
<th>ACTUAL</th>
<th>PERCENT DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROSS REVENUES</td>
<td>650</td>
<td>625</td>
<td>-3.8</td>
</tr>
<tr>
<td>LESS DISCOUNTS</td>
<td>50</td>
<td>45</td>
<td>10.0</td>
</tr>
<tr>
<td>NET REVENUES</td>
<td>600</td>
<td>580</td>
<td>-3.3</td>
</tr>
<tr>
<td>EXPENSES</td>
<td>500</td>
<td>510</td>
<td>-2.0</td>
</tr>
<tr>
<td>NET INCOME</td>
<td>100</td>
<td>70</td>
<td>-30.0</td>
</tr>
</tbody>
</table>
TOPIC: Viewing the Report

Imagine transcribing this report onto a piece of graph paper (evenly spaced vertical and horizontal lines) by placing each letter, digit or other character into a single square. Viewed in this way, the report appears to be a simple character matrix. Your task is to construct the character matrix.

At the simplest conceptual level, you need only use reshape (ρ):

\[
\text{REPORT} \times 12 \ 45 \rho \ ' \text{CAMPBELL CARPET...} \ -30.0 \ '
\]

This is the most computationally direct and efficient way to construct the report. While the burden is light on the computer, however, it is enormous on you. You must count spaces and type them in precisely and you must type in the numbers whose values are probably already in the computer as vector or matrix variables. This is a tremendous waste of your time.

It is more natural to view the report as a set of submatrices which can be pieced together to form the overall report. Then your task is to construct each piece and to concatenate them together to construct the whole report.

How do you subdivide the report? You should break it into as few pieces as possible where each piece may be constructed by a single straightforward procedure. Here is one possibility:

<table>
<thead>
<tr>
<th>CAMPBELL CARPET CLEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANNUAL SUMMARY</td>
</tr>
<tr>
<td>12/31/1986</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACCOUNT</th>
<th>BUDGET</th>
<th>ACTUAL</th>
<th>PERCENT DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GROSS REVENUES</td>
<td>650</td>
<td>625</td>
<td>-3.8</td>
</tr>
<tr>
<td>LESS DISCOUNTS</td>
<td>50</td>
<td>45</td>
<td>10.0</td>
</tr>
<tr>
<td>NET REVENUES</td>
<td>600</td>
<td>580</td>
<td>-3.3</td>
</tr>
<tr>
<td>EXPENSES</td>
<td>500</td>
<td>510</td>
<td>-2.0</td>
</tr>
<tr>
<td>NET INCOME</td>
<td>100</td>
<td>70</td>
<td>-30.0</td>
</tr>
</tbody>
</table>

By using formatting utility functions or APL primitive functions, you can construct each of these blocks with relative ease. Once constructed, they may be pieced together by a single statement:

\[
\text{REPORT} \times \text{TOP},[1]\text{MIDDLE},[1]\text{LEFT},\text{RIGHT}
\]

If you do not have access to utility functions or are using poorly designed functions, you will be forced to break the report into
smaller pieces and construct it in more steps. More steps means less productivity.

Of the four blocks above, the one which may be constructed directly by using APL primitive functions is the matrix of numbers in the lower right of the report. Assuming a 3 column numeric matrix of values named DATA, you may construct the block by the following:

\[
\text{RIGHT} \gets (7 \ 0 \ 8 \ 0 \ 11 \ 1 \ \$\text{DATA}),((1 \rho \text{DATA}),2)\rho'
\]
or:

\[
\text{RIGHT} \gets 2 \phi 9 \ 0 \ 8 \ 0 \ 11 \ 1 \$\text{DATA}
\]
(Note that the 2 columns of trailing blanks could have been considered a fifth block of the report.)

If your APL implementation has an enhanced formatting capability, you may choose to use it rather than $. For example, with DFMT, you may use:

\[
\text{RIGHT} \gets 'I7,I8,F11.1,X2' \ DFMT \text{ DATA}
\]

In APL2, you may use:

\[
\text{RIGHT} \gets '5555550 \ 5555550 \ 5555550.00 \ '$\text{DATA}
\]

```
~~~~~ ~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~
```

PROBLEM: Design utility functions which will construct the three remaining blocks in the report above.

TOPIC: Constructing Titles and Headings

The top block is a block of titles. When you look at that block, you see the words and date, not the blanks. You see three strings of nonblank characters on three separate lines, centered within the width of the report. The information which completely specifies this block is:

* the width of the report (45 characters)

* the fact that each line is centered within the report width
the nonblank strings to be used, one per line (4 lines):

CAMPBELL CARPET CLEANING
ANNUAL SUMMARY
12/31/1986
(EMPTY)

The left block is a block of row names. It is similar to the block of titles except the nonblank strings are left-justified within the width of the block (except for one indented line). The information which specifies this block is:

* the width of the block (17 characters)
* the fact that each line is left-justified within the block
* the nonblank strings to be used, one per line (5 lines):

GROSS REVENUES
LESS DISCOUNTS
NET REVENUES
EXPENSES
NET INCOME

The specifications for these two blocks suggest the syntax for a formatting utility function which we will call TITLES. Let us illustrate the syntax before we define it:

TOP<-45 TITLES 'nCAMPBELL CARPET CLEANINGnANNUAL SUMMARY
12/31/1986n'

LEFT<-17 TITLES 'cGROSS REVENUESc LESS DISCOUNTS
cNET REVENUEScEXPENSEScNET INCOME'

The left argument is an integer scalar of the width of the resulting character matrix. The right argument is a delimited character vector whose partitions each begin with one of the delimiters c (left-justify), n (center) or > (right-justify). The result has one row per partition. Each partition is justified within the row according to the delimiter.

The TITLES function is developed and explained in the Positioning Character Data chapter.

The remaining (middle) block is a block of column headings. When you look at the block, you see four headings. The rightmost heading has two lines. The lines are centered with respect to each other. Every heading is underlined. Each set of underlines is separated from its neighbors by two spaces. The headings are centered with respect to the underlines.

We will illustrate and then define the syntax for a formatting utility function which we will call HEADINGS.
Chapter 10

WRITING REPORTS

MIDDLE+17 6 6 10 HEADINGS 'nACCOUNTnBUDGETnACTUAL
nPERCENT+DIFFERENCE'

The right argument is a delimited (by leading \ symbol) character vector whose partitions each represent one heading. The partitions themselves may be delimited by newline delimiters (+) which indicate the points at which headings are broken into multiple lines. The left argument is an integer vector of the widths of the fields into which the headings are inserted. Typically, one width is provided for each heading (partition). However, if fewer widths are provided, they are repeated to match the number of partitions.

The subpartitions of each partition are truncated if necessary to the corresponding width for that heading. The subpartitions are centered above one another within the width for that heading. A row of underlines (hyphens) is placed below each heading across the width of the heading. The headings are separated by 2 blank columns. If a separation of more or less than 2 blank columns is desired, you may include a vector of the desired separations after the vector of widths in the left argument. They will be repeated if necessary to match the number of partitions. For example, to have one blank column between each heading in the example above:

MIDDLE+17 7 7 11 1 HEADINGS 'nACCOUNTnBUDGETnACTUAL...'

The HEADINGS function is developed as a problem at the end of the Positioning Character Data chapter.

~~~ ~~~ ~~~ ~~~ ~~~ ~~~ ~~~ ~~~

PROBLEM: Most primitive APL formatting capabilities are column oriented. That is, each column of a numeric matrix to be formatted is treated as a separate entity. Every row of the column is formatted in the same way as every other row. This is not true of every column. For example:

```
6 0 6 1 6 2 ¥ 3 3 p19
1 2.0 3.00
4 5.0 6.00
7 8.0 9.00
```

What approach would you take to do row oriented formatting. For example, how would you generate the following?

```
1 2 3
4.0 5.0 6.0
7.00 8.00 9.00
```
TOPIC: Row Oriented Formatting

A common approach to this problem is to simply format each row separately. For example:

\begin{verbatim}
R+3 18p'
R[1;]e6 0&MAT[1;]
R[2;]e6 1&MAT[2;]
R[3;]e6 2&MAT[3;]
\end{verbatim}

This process may be tedious if the matrix has many rows. The problem may be solved noniteratively by transposing the data, formatting it with a column oriented formatting function and then transposing it back. Both monadic and dyadic applications of transpose (\%) are required. The specific logic required is presented as a problem at the end of the Positioning Character Data chapter. Using the ROWFMT function developed there, the solution is:

\begin{verbatim}
width+6
R+0 1 2 ROWFMT MAT
\end{verbatim}

PROBLEM: You wish to print 50 checks. You are given 5 global variables: CNUM (50 element integer vector of check numbers, e.g. 305); CDATE (50 element integer vector of check dates in MMDDYY format, e.g. 121586); VENDOR (50 row, 25 column character matrix of vendor names, e.g. 25i'ACME SUPPLIES'); CAMT (50 element integer vector of check amounts in cents, e.g. 518250); DESC (50 row, 40 column character matrix of check descriptions, e.g. 40t'LOOSELEAF NOTEBOOKS'). Write a function to print the information on continuous form blank checks loaded in the printer. The following is an illustration of the layout and characters to be printed for a single check:

\begin{verbatim}
NO. 00305 DEC 15, 1986
TO: ACME SUPPLIES $5,182.50
FOR: LOOSELEAF NOTEBOOKS
\end{verbatim}
TOPIC: Formatting Multi-Row Records Using Newlines

When doing simple formatting of a numeric matrix (e.g. 7 \texttt{0\$NMAT}), the character matrix result contains one row per row of the matrix being formatted. In the problem above, the result appears to contain one matrix (a check) per row (or element) of the arrays being formatted. The term "appears" is used because the "matrix" may in fact be a vector with embedded newline (carriage return) characters. Let us look at the above example in a special way:

\begin{verbatim}
' ... NO. 00305b... DEC 15, 1986nn TO: ACME SUPPLIES
... $5,182.50nn FOR: LOOSELEAF NOTEBOOKS ... nn
\end{verbatim}

The \texttt{n} represents a newline character and the \texttt{b} represents a backspace character. Notice how the backspaces are being used to underline the check number. The entire vector, including newlines, backspaces and blanks is 174 characters long. The problem becomes much simpler if you consider that your task is to format and combine the arrays into a 174 column matrix, with one row per element or row of the arrays.

Begin by breaking the \texttt{CDATE} variable into two parts: the month (as a 3 letter abbreviation) and the day/year portion (\texttt{DDYY}):

\begin{verbatim}
MON+12 3p'JANFEBMARAPRMYUNJULAOCTNOVDEC'
MON+MON[CDATE/10000;]
DDYY+10000|CDATE
\end{verbatim}

Let us solve the problem first for those APL implementations which have \texttt{DFMT} (e.g. \texttt{APL*PLUS} and \texttt{SHARP APL}). Given the appropriate format string (\texttt{FS}) left argument to \texttt{DFMT}, the desired matrix result (\texttt{CHKS}) may be constructed as:

\begin{verbatim}
CHKS~FS \texttt{DFMT (CNUM;MON;DDYY;VENDOR;CAMT;DESC)}
\end{verbatim}

The format string left argument of \texttt{DFMT} may contain special characters such as newline and backspace. Construct the variables \texttt{NL} and \texttt{BS} to contain the newline and backspace character scalars respectively:

\begin{verbatim}
APL*PLUS SHARP APL
-------- --------
BS~DT CBS BS~DAV[158+DIO]
NL~DT CNL NL~DAV[156+DIO]
\end{verbatim}

The format string is constructed by carefully piecing together the control characters needed to produce the special 174 column matrix.

\begin{verbatim}
FS~'X36,<NO. >,ZI5,<',(5pBS),(5p'_)','>,X3,3A1,'
FS~FS,'S<97>G< ??, 19??,',',(3pNL),' TO: >,'
FS~FS,'25A1,P<$>CK~2F15.2,',',(3pNL),'FOR: >,'
FS~FS,'40A1,','(6pNL),''
\end{verbatim}

To print the checks, just display the variable \texttt{CHKS} on the printer.
Now let us solve the problem using format by example (dyadic ≜ with character vector left argument, an APL2 enhancement). The approach is basically the same. However, ≜ does not allow more than one array in its right argument. We must therefore build the result in sections and then combine the sections.

```apl
BS←₀TC[₀I0]
NL←₀TC[₁+₀I0]
ROWS←pCNUM

S1←((36p',',NO. 05555',(5pBS),(5p','),3p'))≜(ROWS,1)pCNUM
S2←MON (MON and DDYY from above)
S3←(' 05,_5555',(3pNL),' TO: ',)≜(ROWS,1)p1900+100001
  0 100pDDYY
S4←VENDOR
S5←('$555,555,553.50',(3pNL),'FOR: ',)≜(ROWS,1)pCAMT+100
S6←DESC
S7←(ROWS,6)pNL

CHKS←S1,S2,S3,S4,S5,S6,S7
```

PROBLEM: Frequently, when an APL application produces multi-page reports, the reports are sent to a "print file" rather than printed directly. Why? How should such a print file be organized? How should its contents be printed?

TOPIC: Directing Report Output to Print Files

There are several reasons for directing report output to a print file rather than printing/displaying it immediately:

1. To avoid the interruption caused by printing when many different reports are to be generated. You may generate them all (to a print file) and then let them print unattended.

2. To enable simple and inexpensive restarting. If the paper jams or runs out (or the line to a remote computer drops), you may reprint the report without having to regenerate it.

3. To make multiple copies of a report. The print file may be printed repeatedly without regenerating the report.
4. To print the report on a remote (batch) high-speed line printer. A print file must exist in order for you to submit a batch request for remote printing.

5. To spot-check a lengthy report. If a lengthy report is appended to a print file by an application, selected pages may be printed and reviewed when deciding whether or not to print the entire report.

The following is a reasonable organization for a print file (assuming your APL implementation allows APL files or a reasonable emulation):

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 10</td>
<td>(latent, i.e. empty character matrices: 0 0ρ′′)</td>
</tr>
<tr>
<td>9+2×I</td>
<td>Character matrix (or character vector with embedded newline characters) representation of page I (1,2,3,...) of the report. When displayed, the array will require no more lines than can be accommodated by the paper on which the page is to be printed (typically 66, i.e. 6 lines per inch on paper which is 11 inches high).</td>
</tr>
<tr>
<td>10+2×I</td>
<td>One column all blank character matrix with as many rows as are required at the bottom of page I to reach the bottom of the paper (typically 66 minus the number of rows in the matrix stored in component 9+2×I).</td>
</tr>
</tbody>
</table>

The first 10 (latent) components are included in case your implementation has a remote (batch) high-speed line printer capability. Typically, these batch facilities require several control components at the beginning of your print file. The precise significance of these control components is a function of your APL implementation. Include whatever components are needed in your environment.

After the first 10 components, the print file is organized into pairs of components, one pair per page. Two components are used per page instead of one so that your printfile will not be filled with 80 column (or so) blank rows whose only function is to move the printer to the top of the next page. In fact, if file storage is a major consideration, you should break each page into many pieces (some pieces only one line) so that excess spaces can be omitted. However, the file organization would then not allow direct access to a page in the middle of the file since you could not determine the component in which it begins except by trial and error or by maintaining a directory.

With this file organization, you can immediately tell how many pages are on the print file (.5×10+number of components) and you can determine exactly where any page is stored (component 9+2×I for page I).
To send the contents of a print file to a remote (batch) high-speed line printer, you must follow the directions which apply to that facility in your environment. However, if you want to print pages on your local printer (or hardcopy terminal) or just wish to spot-check pages on your CRT terminal, the following PRINT function may be useful. To use it, you tie (or otherwise activate) the print file and provide the tie number (or other file identification) as the right argument of PRINT. The dialog will then look something like this:

73 PAGES ON THE PRINTFILE.
BEGIN ON WHICH PAGE (OR END): 25
ALIGN PAPER TO PERFORATION AND PRESS RETURN.

(page 25 prints)
(page 26 prints)
...
(page 73 prints)

ERASE PAGES? YES
NO PAGES ON FILE.

If you respond YES to the ERASE PAGES? question, all pages on file will be erased. That is, all but the first 10 components of the print file will be dropped.

In this function, some attention handling code has been included (for APL*PLUS or SHARP APL) in case the BREAK key is pressed while the pages are printing. In that event, the printing immediately stops and you are again asked for the page number on which to begin.
Chapter 10  WRITING REPORTS

[WSID: PRTFILE]

\[\text{PRINT TIE;A;I;N;}\text{P;}\text{DAX;}\text{OPW}\]

1. Prints some or all pages in the printfile tied to TIE. Pages are in components 11, 13, 15,...
2. APL*PLUS attention handling (put DAX in header):
3. \text{DAX\text{\textless}1\text{L}\text{I}}
4. A SHARP APL attention handling (put DTRAP in header):
5. \text{DTRAP\textless1000 E \rightarrow L1}
6. A Set print width to avoid APL wrap on long lines:
7. \text{OPW\rightarrow250}
8. A Number of pages in the file:
9. \text{N\rightarrow(DSIZE TIE)2\text{\textless}1\text{+}2} \text{APL*PLUS}
10. \text{N\rightarrow((DSIZE TIE)2\text{\textless}1\text{+}2} \text{SHARP APL}
11. D\rightarrow(\text{PAGE'},(N=1)\text{\textless}1'S \text{ON THE PRINTFILE}.)
12. A Branch if 0, 1 or many pages:
13. I\rightarrow1
14. \rightarrow(END,L2,L1)[1+\text{N}L2]
15. A Ask for starting page if more than one:
16. L1:O\rightarrow'
17. \rightarrowP\rightarrow'\text{BEGIN ON WHICH PAGE (OR END): '}
18. A\rightarrow(pP)!~
19. A Reprompt on empty response:
20. \rightarrow(pA)!~L1
21. A Exit if END entered (even partially):
22. \rightarrow(AA.=(pA)t'END')pL4
23. A Convert response to numeric and validate:
24. I\rightarrowUI A A Available on APL*PLUS and SHARP APL
25. \rightarrow((1=pI)\text{\textless}I\text{\in}N)pL2
26. D\rightarrow'** INVALID PAGE NUMBER. '
27. D\rightarrow'** VALID PAGE NUMBERS ARE 1 THROUGH ',(\text{\#N}),'. '
28. D\rightarrow'** TYPE 'END' TO STOP PRINTING. '
29. L1\rightarrow'
30. A Align paper:
31. L2:O\rightarrowP\rightarrow'ALIGN TO PERFORATION AND PRESS RETURN. '
32. A\rightarrow(pP)!~
33. A Exit if END entered:
34. \rightarrow((1\text{\leq}N)\text{\&\&}(pA)\text{\uparrow}'END')pL4
35. A Print page and spacing to next page:
36. L3:O\rightarrowFREAD TIE,9+2xI A APL*PLUS
37. A L3:O\rightarrowFREAD TIE,9+2xI A SHARP APL
38. D\rightarrowFREAD TIE,10+2xI A APL*PLUS
39. D\rightarrowFREAD TIE,10+2xI A SHARP APL
40. A Branch if more pages:
41. \rightarrow(N\text{\&\&}1+I\text{\uparrow}1)pL3
42. A Erase pages, if desired:
43. L4:O\rightarrowP\rightarrow'ERASE PAGES? '
44. \rightarrow('Y'\#1\text{\uparrow}(pP)!\text{\downarrow}pEND
45. D\rightarrowFDROP TIE,0\text{\downarrow}1-(DSIZE TIE)2 A APL*PLUS
46. D\rightarrowFDROP TIE,0\text{\downarrow}1-(DSIZE TIE)2 A SHARP APL
47. D\rightarrow'NO PAGES ON FILE. '
48. END:
49. V
PROBLEMS:  
(Solutions on page 366)

1. Using the HEADINGS function introduced in this chapter, construct the following set of column headings.

<table>
<thead>
<tr>
<th>LAST YEAR</th>
<th>THIS YEAR</th>
<th>GROWTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG. SALE</td>
<td>TOTAL SALES</td>
<td>AVG. SALE</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
</tr>
</tbody>
</table>

2. Using the numeric scalar DATE which is today's date in the form MMDDYY and the numeric scalar PNO which is the current page number, construct the following set of titles using the TITLES function introduced in this chapter.

PAGE 17

FINANCIAL SUMMARY
12/15/86
WESTERN REGION

3. Given the following numeric matrix,

NMAT

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

how would you format it to appear as follows?

CMAT

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>6.0</td>
<td>7.0</td>
<td>8.0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13.0</td>
<td>14.0</td>
<td>15.0</td>
<td>16.0</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>21.0</td>
<td>22.0</td>
<td>23.0</td>
<td>24.0</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 11
SYSTEM DEVELOPMENT PROCEDURE

Because APL is so concise, powerful and unrestricted, almost anyone can toss together an application system. While experience and discipline are useful to have, they are not essential. This is one of the reasons why so many APL programmers do not come from traditional data processing backgrounds. (The other reason being that prolonged use of COBOL tends to rot the brain.)

If you can solve a problem in APL in one-fifth the time it takes to solve it using FORTRAN, it follows that you can develop 5 unreadable, unmaintainable APL systems in the time it takes you to develop one unreadable, unmaintainable FORTRAN system. To some computer scientists, this improvement in productivity leads to a new philosophy of system development: throw-away code. The basic idea is to write the system fast to get the job done. Then, when requirements change and the system is no longer adequate, throw it away and build a new one.

Those who foster the view that APL is the ideal language for writing throw-away code are those who would like to see APL thrown away. They also tend to kick their pets.

In many respects, APL is not unlike any other programming language. System development should be planned, documented and implemented meticulously. If done properly, the system will be a pleasure to use and to maintain. If done improperly, the system will be a living hell for all those associated with it. Documentation will be scarce, if existent. Code will be mystifying, if readable. The user will be suicidal, if not homicidal.

A well-developed system, on the other hand, is easy to recognize. It is easy to use so the user rarely needs to refer to the extensive user guide. It is reliable and efficient so the technical support person rarely needs to scan through the readable code or the extensive technical documentation. The word "enhancement" is used more frequently than "maintenance" and neither word causes the system developer to tremble with anxiety.
In this chapter we describe eight steps which should be part of any APL system development procedure:

1. Familiarization
2. Specification
3. File design
4. Workspace design
5. User documentation
6. Flow charting
7. Coding, typing, testing
8. Delivery, training

Yes, in that order. Compare this list to the procedure for the usual throw-away APL system:

1. Coding, typing, testing
2. Delivery, training
3. Familiarization

(Oh! So that's what you wanted!)

I implore you. Please do not dismiss the system development procedure outlined here without trying it once. The procedure will increase your productivity, improve the quality of your system and make the development process more fun. Try it.

TOPIC: Familiarization

During this phase, you become familiar with the problem, not the solution. Emphasis is on the needs of the user, not the tools of the programmer.

If a manual system exists and is to be replaced, now is the time to study the manual system. If there is no manual system, you should talk with the user and "brain-storm" about an ideal system. Sketch sample reports and sample input sheets.

Where will the data come from? Is it readily available? Will the value of the system justify the installation and updating of the data? How much will the data requirements of the system grow? Will the system need to supply data to other computer systems? How frequently will reports be generated? Who will use them and why? How often will the report formats change? What is the expected life of the system?

The unasked questions which should permeate your thinking are: Is this system feasible? Does the user have a clear picture of what such a system will be like and how it will interact with the
organization? Is the value of the finished system going to justify the time and effort required to develop it?

The familiarization phase could also be called the feasibility phase. The phase is complete when both you and the user have a clear qualitative understanding of how the system should operate, and are both convinced that the system is a good idea.

### TOPIC: Specification

At the heart of the word "specification" is the word "specific". That is what the specification phase is all about: getting specific. Put in writing all the details which define the system.

Bear in mind during this phase that any system has three major aspects: input, processing, output.

**Input:** What data items must be supplied to the system? Are there other items which are not needed now but may be needed later? How many records of data items will the system contain? How fast will it grow? From what different sources will the data come? How frequently? What is the exact record layout of any data from external media (e.g. computer tape)? What is the exact layout of the input sheets used to manually enter data? How clean will the data be? What data integrity checks must be performed (e.g. salary must be a positive integer less than 50,000)? What inter-data restrictions must be imposed (e.g. date-of-hire must be earlier than date-of-termination)?

**Processing:** What data items must be computed from input data items? With what formulas? What regular processing operations are to be conducted? How frequently? What steps are involved in these operations? How often will these steps change and how dramatically?

**Output:** What reports will need to be generated by the system? How frequently? What is the exact layout of each report? How is each report item derived from the input and computed data items? How often will the report formats change and how dramatically? Will other areas need access to certain data items? In what format?

The specification phase is complete when you have a written document (the "specification") which so thoroughly and specifically describes the system that, ideally, it could be given to any expert programmer who could then develop the system without conferring with the user.
TOPIC: File Design

During this phase, you draw pictures of alternative file designs. You then consider the pros and cons of each design given the input and output requirements of the system.

How many file accesses are required to add one record? To add 100 records? To change a few items on one record? On 100 records? To delete one record? To delete 100 records? To display the entire contents of one record? Of 100 records? To search the entire file for records which match a set of logical criteria? To generate each of the reports included in the specification? How often will each of these operations be performed? Will the cost and response time resulting from these file accesses be acceptable?

The file design phase is complete when the structure of each file, including its name, is completely documented in written form.

TOPIC: Workspace Design

During this phase, you sketch sample terminal sessions. The terminal sessions illustrate the actual operation of the system including system prompts and typical user responses.

You should work closely with the user during this phase since the user must live with the system interaction being designed now.

Begin with a general flow chart of the operations to be performed by the system. Break the flowchart into functional blocks, each of which will be implemented as a single APL function. Then, for each function, write down the dialog (sample terminal session) produced by the function.

Does the dialog allow for all required input? Does it provide control of every processing step? Can any and all reports be requested easily? Can the user gracefully exit the system from anywhere without losing any input? Can the user quickly navigate among the most commonly used operations of the system? Are the prompts meaningful? Does the prompting structure allow for optimal use of the files, given their design?
The workspace design phase is complete when sample terminal sessions have been sketched for all contingencies and the user accepts the flow of the system without reservation. The major functions of the system are identified, named and documented in general terms. The file design is updated if necessary.

TOPIC: User Documentation

User documentation may be written before or after the system is implemented. It is better to write it before. The only reason for writing the documentation after the system is built is that you may be able to talk the user out of any documentation at all. Not a noble reason.

By writing the user documentation before the system is implemented, you will find that the documentation is much easier to write (fewer constraints) and the system is easier to implement. For example, before implementation you can write, "Type STOP at any time to terminate the system." After implementation you must write, "Type STOP to terminate the system when adding records; type END when generating reports; type HALT when closing the accounting period; and type O-backspace-U-backspace-T if none of these works."

If you have never documented a system before implementing it, you may be reluctant to try it now. Please! Please try it! It will make the overall system development task easier and will result in a better system. It's more fun too. Try it once. What can it hurt?

The user documentation is an instruction manual which explains to an inexperienced user how to use the system. It begins with an explanation of the purpose of the system and any background information required to understand the system. After the introduction, the manual consists mainly of the sample terminal sessions along with comments explaining the various options. After reading a well-written manual, the user should be able to use every facet of the system without help from you.

As you write the manual, you should update the specifications and file design if such change is suggested by the documentation process. The user documentation phase is complete when the manual has been read by the user and accepted without reservation.
TOPIC: Flow Charting

During this phase, you will diagram the program logic which underlies each of the major functions identified during the workspace design phase. You will do this with the documented file structure on one side of your desk and the user documentation on the other side.

The flow charting phase can be called the "divide and conquer" phase. Divide each major function into the general steps which it must perform. Then divide the general steps into more specific steps. Continue in this fashion until the steps can be translated directly into APL code.

During this subdivision process, you will identify common steps which are required in several locations of the system. If such steps are not at the low level in which they may be translated into APL code, you may choose to label these steps as subfunctions and write their comprising steps but once. As you identify each subfunction, you should name it, define its syntax, list the variables and functions which are global to its operation and write a brief description of what it does. This will become a permanent part of the technical documentation.

The flow charting phase is complete when coding is all that remains. You will have written descriptions of all functions, subfunctions and global variables. You will have flow charts which diagram every logical step during the use of the system. The steps will be described at such a precise level of detail that they may be translated directly (without much logical reasoning) into APL code.

TOPIC: Coding, Typing, Testing

Without having written a symbol of APL code or pulled your chair up to your APL terminal, you are more than half done with the system. The user has been getting constant feedback from you, has a user's manual on his or her desk and has complete confidence in your ability to deliver the exact system needed. All this without a symbol of APL.

If you do not know APL, now is the time to learn it. Quickly.

During the coding, typing, testing phase, you do just that. The three tasks are clumped together because they need not each be performed to completion before starting the next task. For example, you may want to code 5 or 10 functions, type them, test them and repeat this process for the next 5 or 10 functions.
Chapter 11  

SYSTEM DEVELOPMENT PROCEDURE

While it is possible to code the entire system before typing a single keystroke, there are disadvantages to such extreme behavior. For one thing, a coding flaw will not be picked up until you begin testing. You may have made the same mistake dozens of times throughout the system. Second, when you are testing the code, you may not remember your intentions in a difficult piece of code. Finally, if doing nothing but writing code for 3 weeks does not drive you crazy, then doing nothing but typing APL code for 3 days will. And if that does not, then 2 weeks of testing will.

At the other extreme, you may choose to code, type and test one function at a time. There are disadvantages to this mode of programming. Some design flaws will not be uncovered until you get further into the system. Such flaws may require you to rewrite or scrap functions written earlier. Any time spent typing or testing now obsolete code will have been wasted.

The coding, typing, testing phase is complete when everything is tested and you are ready to turn the system over to the user.

TOPIC: Delivery, Training

During this phase, you will transfer the system from your control to the user’s control. You will initialize any files which have not yet been initialized and will move the workspace or workspaces to the user’s library if they need to be moved.

When the system is ready to roll, you will meet with the user to take a spin. Having read the user documentation (and helped you design the dialog), the user should require little guidance or training from you. As the system is tested, you will need to make two lists. The first list refers to bugs which are encountered. If your testing process was careful and thorough, this list will be empty.

The second list refers to suggested enhancements to the system. There is nothing like a live system to suggest what is wrong with it. The user will be happy to mention these. Since you worked together closely to design and document the system, you will not be blamed for delivering an imperfect system. Rather, you will be commended for delivering what you agreed to deliver.

The delivery and training phase is complete when the user accepts the system as is. If there are enhancements to be made to the system, you should implement them as you did the system: familiarization,
specification, file design, etc. For simple enhancements, you may get through all the phases in a few minutes. Do not forget to update the technical and user documentation.
The reason for programming standards is to create a conformist world in which every programmer thinks and programs the same way. What these programmer clones lose in creativity they more than make up in productivity. After all, when picking up a program written by Clone A, Clone B has no trouble reading it. Not only is the language familiar; so too is the dialect and the handwriting.

The purpose of this chapter is to present a set of APL programming standards. They are not presented as the perfect set nor even as the author's preferred set. Rather, they are a set. Pick and choose as they suit you. The important thing here is that the set you choose be accepted by all those in your organization who will work on the same systems as you.

The standards are organized by the phases of the system development procedures presented in the prior chapter. Along with the dogma of each standard is a brief justification for it. If the justification is omitted, you may assume it is, "To improve readability by use of consistent conventions."

TOPIC: Familiarization

1-1 Select, or have appointed, another programmer to review your work.

WHY: To spot design and logic flaws and otherwise help you see the forest from the trees. This is also one of the best ways to learn new design and programming techniques.
Chapter 12

1-2 Make certain a single user has been assigned the responsibility of working with you.

WHY: You need a single person to accept your design and to be held responsible for it.

TOPIC: Specification

2-1 Include hand drawn input sheets and full-screen input forms.

WHY: To insure that you and the user see input requirements eye to eye.

2-2 Include hand drawn reports containing exact headings, line names and number formats.

WHY: To insure that you and the user see output requirements eye to eye.

2-3 Have someone else review the specification.

TOPIC: File Design

3-1 All file components are assigned a meaningful variable name (first letter underscored) which is used when the component is read into the workspace.

WHY: To help anyone reading the code to identify objects from file.

3-2 The file directory, if there is one, is stored in component 1 of the file.

3-3 On-line file documentation, if any, is stored in component 2 of the file.
3-4 Leave at least 10 latent (empty vector) components at the start of the file for future design modifications.

3-5 Files are documented on paper (preferably using word processing software) and include the component number, variable name, shape and description of each object in the file.

3-6 Have someone else review the file design.

~~~~~~~~~~~~

TOPIC: Workspace Design

4-1 Along with each user prompt, list all possible error messages in the sample terminal sessions.

WHY: To insure consistent error messages.

4-2 Use the following standard user keywords when needed:

   ADD: Add more data to database.

   CHANGE: Replace an existing value with another.

   DELETE: Remove data from database.

   SHOW: Display data from database.

   INSERT: Add more data among existing data in a database where order of data is important.

   END: Normal termination of the current phase of the program.

   HALT: Terminate program abruptly and completely.

WHY: To be consistent so the effect of various responses to a prompt can be anticipated.
4-3 The system is invoked by loading an autostarted workspace (using DLX). The functions do not return to immediate execution mode until the system is terminated. All input is accepted via character input (D) mode or full-screen input mode rather than evaluated input (E) mode.

WHY: To eliminate the possibility of accidentally invoking non-user functions or of getting APL system error messages (e.g. SYNTAX ERROR).

4-4 Applications are terminal independent unless special features are expressly desired.

WHY: To allow switching from one terminal to another without requiring program modifications.

4-5 All reports are directed to a printfile rather than to the terminal. The contents of the printfile must be displayed in a separate step.

WHY: To allow easy, inexpensive report restarting in the event of line noise, line drop, printer malfunction or complete crash; to allow reference by page number when printing; to allow flexibility in directing reports to terminal, line printer or remote printer.

4-6 Have someone else review the workspace design.

TOPIC: User Documentation

5-1 Include all possible prompts in the documentation, along with descriptive text.

WHY: To ease the transition from text to terminal.

5-2 Write the documentation using whatever word processing software is commonly used in your department or company.

WHY: To insure professional appearance of documentation; to allow quick, easy modifications; to ease the transfer of system support since the same word processing software is used by all.
5-3 If the dialog or options of the system are necessarily complicated, include a brief summary of the workspaces, functions, keywords, choices, and so on for quick reference.

WHY: So the user does not have to thumb through the lengthy documentation.

5-4 Include a complete Table of Contents.

5-5 Have someone else review the documentation.

TOPIC: Flowcharting

6-1 The purpose of each function is documented in a one-sentence description, including the function syntax and its dependence upon global variables and subfunctions. All functions have meaningful names or abbreviations.

6-2 The purpose of each global variable is documented in a one-sentence description.

6-3 Functions with same names in different workspaces are identical.

6-4 Subfunctions are chosen judiciously. They have well defined arguments, produce well defined results or effects and require or create a minimum number of global variables or other subfunctions. The state indicator does not get deeper than 5 levels.

WHY: All user defined functions call other functions (at a minimum, APL primitive functions). The key to readability is that the subfunctions can be understood without reference to further documentation. If there are too many subfunctions and they are not neatly defined, the reader will spend too much time flipping back and forth among function listings instead of reading code. The state indicator in the human brain can generally go no deeper than 5 levels without losing track.
6-5 Flowchart using words and diagrams, not APL code.

WHY: To compel the programmer to organize her thoughts and plans before getting bogged down in coding details. If coding is all that remains, flowcharting is done.

6-6 Include all error checks in flowchart.

WHY: To remember them when coding and to not underestimate the complexity of the function.

6-7 Have someone else review the flowcharts.

TOpIC: Coding, Typing, Testing

7-1 All workspaces contain the global variable <wsid> which contains the workspace identification (WSID) of the saved workspace.

7-2 All workspaces contain the global variables <fnums> and <fnames> which contain the file numbers and file names of the files which are assumed to always be tied.

7-3 File tie numbers are assigned to global variables whose meaningful names are prefixed with 'f' (e.g. fSMRY, fEMPL). These variables reference files which may or may not be tied (see <fnums>). The tie number of the printfile is assigned to fPRINT.

7-4 The name of the printfile is 'PRINTFILE'.

7-5 All workspaces contain the function TIEFILES which ties those files in <fnums> and <fnames>.

7-6 Workspaces which alter their DLX contain the global <lx> which is assigned the original value of DLX.

7-7 Variables global to the workspace have meaningful names and are completely underscored (except for global variables containing file tie numbers).
7-8 Variables from file and localized variables used globally by other functions have meaningful names and have their first character underscored.

7-9 Functions on file, like variables on file, have meaningful names with the first character underscored.

7-10 Strictly local variables (localized and not used within subfunctions) contain no underscores in their names. Meaningful variables have meaningful names or abbreviations. Temporary, intermediate or useless (e.g. from DEX) results are assigned to single letter variable names and are not used further than 5 statements beyond the assignment.

7-11 The meaning of a variable is commented when first assigned unless the comment is the name, or the variable is read from a documented file, or the meaning can be inferred from a prompt.

7-12 The first line or two of every function is a comment which explains the purpose and syntax of the function, and lists the global variables and subfunctions required by the function.

7-13 When calling a subfunction, include a comment which lists the global variables and subfunctions required by the subfunction.

7-14 The intent of every function line is commented.

WHY: To help the program maintainer quickly locate and decipher code which needs fixing or enhancing. Writing code is the process of converting the intent to the code. Reading code is the process of attempting to reconstruct the original intent based on the code. Since the intent is obvious during coding, it can be included in a fraction of the time (and mental effort) it would take to reconstruct it later. Lines that contain prompts or error messages are often self-commenting. Examples of comments which unsuccessfully and successfully comment the intent:

- Squeeze out the flagged rows of the matrix. (no good)
- Ignore inactive profit centers. (good)
- Set DELX to capture error messages. (no good)
- Prepare for file reservation errors. (good)
- Increment and repeat if not done. (no good)
- Loop by region. (good)
7-15 Line labels are L1, L2, L3,... and are kept in ascending order even if not sequential. Significant labels, which segment the function or identify key steps, may have meaningful names. For example:

\[ (\text{'ACDE'}=1\uparrow\text{R})/\text{ADD,CHANGE,DELETE,END} \]

7-16 Branching is always to a line label or empty vector (never \( \rightarrow 0 \) or \( \rightarrow \) ).

WHY: Without \( \rightarrow 0 \), the program must exit through its "bottom" which is better style than having many exit points. For example, you may be certain that a statement added to the end of a function will always be executed. The use of naked branch (\( \rightarrow \)) removes control from cover functions which may call the function.

7-17 Recommended branching techniques are:

\[ \rightarrow \text{LABEL} \]
\[ \rightarrow \text{CONDITION/LABEL} \]
\[ \rightarrow \text{CONDITIONS/LABELS} \]
\[ \rightarrow \text{CONDITION\uparrow LABEL} \]
\[ \rightarrow \text{LABELS[INDEX]} \]
\[ \rightarrow \text{CONDITION\uparrow LFALSE,LTRUE} \]

7-18 Recommended looping technique is:

\[
\begin{align*}
I & \leftarrow 1 \\
\text{LOOP:} & \rightarrow \text{ENDLOOP IF } I \text{LE LIM} \\
& \quad \text{process } I \\
& \quad I \leftarrow I + 1 \\
& \quad \rightarrow \text{LOOP} \\
\text{ENDLOOP:}
\end{align*}
\]

7-19 Use \(-2147483647\) for numeric values which are "not applicable"; assign this constant to the variable \(<\text{huge}>\) if used frequently.

7-20 All output to the terminal is through \( \rightarrow \) or \( \leftarrow \). For example:

\[
\leftarrow \text{'ENTER YOUR CHOICE'}
\]

WHY: To help locate all terminal output if the need arises (e.g. to direct output to a file) and to improve readability (e.g. \( \leftarrow \text{PROCESS MAT vs. PROCESS MAT} \)).
7-21 Evaluated input mode (□) is not used.

WHY: To avoid unintentional escapes via )LOAD or )OFF, to avoid unintentional execution of defined functions, and to avoid technical error messages (e.g. SYNTAX ERROR) to the user.

7-22 Maintain □IO=1 globally. Localize □IO if assigned as 0.

7-23 The local result variable is used only for the result and not for temporary values.

WHY: To avoid unintended results upon premature function termination and to avoid confusion when reading the function.

7-24 The local argument variables are never reassigned except to ravel them.

7-25 Every function line is restartable. That is, no other functions should be performed on the same function line once a function has been executed whose effect should not be repeated. For example, T←2+V←V,R is unrestartable.

WHY: So that any function line may be restarted from its beginning after it has been stopped, say by an error. For example, the following suspension cannot be properly restarted via →3 since V will have been extended twice:

WS FULL
^    

7-26 Error messages or other error handling logic immediately follow detection. For example:

[10] →(X>0)/L2
[12] →L1
[13] L2: etc.

WHY: To avoid having to search through the function to find the code which handles each error condition.

7-27 Lines containing multiple statements perform a single, logical operation.
Chapter 12

PROGRAMMING STANDARDS

7-28 All error messages are passed through an error-displaying function named ERRMSG.

WHY: To allow consistent presentation of error messages (e.g. preceding them by two stars or beeping twice or displaying in a specified position on the screen). To enable you to find all error messages for inclusion as an appendix in the user manual.

7-29 Testing is performed methodically by stopping on every function line, not experimentally (i.e. by jumping from bug to bug).

7-30 Test all edge (e.g. empty vector) conditions.

7-31 Have someone else review the code.

~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~

TOPIC: Delivery, Training

8-1 The contents of all workspaces are documented using whatever workspace documentation software is commonly used in your department or company. This software produces a printed, paged listing of the definitions of all functions and at least the names and shapes of all global variables.

8-2 Functions are not locked unless you have a significant reason for doing so.
For a given computer application, two different programmers will design and implement it differently. In fact, a single programmer will develop the application differently at different times in her own career. Because of the flexibility of APL, a spectrum of approaches are both possible and feasible for any problem. How then is one to choose between plausible approaches when designing an application system? In this chapter, we discuss workspace design and documentation considerations. The aims are to expand your appreciation of the trade-offs involved during the design process, and to help you document an existing application.

PROBLEM: Develop an application which will maintain a list of employees. For each employee, maintain the employee's number (4 digits), name (last name first) and age. Do not use files for this application. Rather, store the information in the global variables ENUM (integer vector), ENAME (25 column character matrix) and EAGE (integer vector). Provide capabilities for adding, deleting and listing employees.

TOPIC: Subfunction Design

As simple as this application is, no two programmers will develop it exactly the same way. The most pronounced difference between solutions is the degree to which subfunctions are employed. At one extreme, a single function is written which calls no subfunctions. At the other extreme, a primary (or main or cover or driver) function is written which calls a variety of subfunctions which in turn call subfunctions and so on as desired.
The following is an illustration of the APL code written to implement the application as a single function.

```
[WSID: FLF]

∇ EMPLOYEES;AGE;G;GOOD;NAME;NUM;P;R
[1]  a Ask for choice on same line:
[2]  CHOOSE←P←'ADD, DELETE, LIST OR END: '
[3]  R←(P)↓[]
[4]  a Branch based on 1st char of response:
[5]  →('ADLE'=1R)/ADD,DELETE,LIST,END
[6]  ↔** INVALID CHOICE. CHOOSE FROM: ADLE'
[7]  →CHOOSE
[8]  a
[9]  a
[10] ADD←D←'EMPLOYEE NUMBER (OR 0 IF DONE)'
[12]  a Continue if exactly 1 number entered:
[13]  →(1=ρNUM)/A1
[14]  ↔** ENTER 1 NUMBER'
[15]  →ADD
[16]  a Branch to choice question if 0 entered:
[17]  A1:→(0=NUM)/CHOOSE
[18]  a Continue unless employee number already exists:
[19]  →(NUM∈ENUM)/A2
[20]  ↔** EMPLOYEE ',(≠NUM),' ALREADY IN LIST'
[21]  →ADD
[22]  A2:↔P←'EMPLOYEE NAME (MAX 25 CHARACTERS): '
[23]  a Ask for name at end of same line:
[24]  NAME←(P)↓[]
[25]  a Continue unless name too long:
[26]  →(25≥ρNAME)/A3
[27]  ↔** NAME TOO LONG'
[28]  →A2
[29]  A3:↔P←'EMPLOYEE AGE'
[30]  AGE←[]
[31]  a Continue if exactly 1 number entered:
[32]  →(1=ρAGE)/A4
[33]  ↔** ENTER 1 NUMBER'
[34]  →A3
[35]  a Continue if a valid age:
[37]  ↔** AGE MUST BE INTEGER FROM 17 TO 99'
[38]  →A3
[39]  a Catenate new values and ask for more:
[40]  A5:ENUM←ENUM,NUM
[41]  a Pad name to length 25:
[42]  ENAME←ENUM,[1]25↑NAME
[43]  EAGE←EAGE,AGE
[44]  →ADD
[45]  a
[46]  a
[47] DELETE←D←'ENTER EMPLOYEE NUMBERS TO DELETE'
[48]  a Ravel to insure a vector, not scalar:
[49]  NUM←[],[]
```
EMPLOYEES (continued)

 Continue if all valid numbers:

 \[ \left( \forall \text{GOOD} \in \text{ENUM} \right) / \text{D1} \]

 ** NOT FOUND: 

 \[ \forall \text{~GOOD}/\text{NUM} \]

 \[ \Rightarrow \text{DELETE} \]

 Flag those employees to keep:

 \[ \text{D1: GOOD} \leftarrow \text{ENUM} \in \text{NUM} \]

 Squeeze out deleted employees:

 \[ \text{ENUM} \leftarrow \text{GOOD} / \text{ENUM} \]

 \[ \text{ENAME} \leftarrow \text{GOOD} / \text{ENAME} \]

 \[ \text{EAGE} \leftarrow \text{GOOD} / \text{EAGE} \]

 \[ \Rightarrow \text{CHOOSE} \]

 Flag those employees to keep:

 \[ \text{D1: GOOD} \leftarrow \text{ENUM} \in \text{NUM} \]

 Squeeze out deleted employees:

 \[ \text{ENUM} \leftarrow \text{GOOD} / \text{ENUM} \]

 \[ \text{ENAME} \leftarrow \text{GOOD} / \text{ENAME} \]

 \[ \text{EAGE} \leftarrow \text{GOOD} / \text{EAGE} \]

 \[ \Rightarrow \text{CHOOSE} \]

 Prepare to sort employees by number:

 \[ \text{G} \leftarrow \text{ENUM} \]

 Sort and display:

 \[ \left( 5 \quad 0 \quad 7 \quad 0 \quad \text{ENUM}[G], [1.5] \text{EAGE}[G] \right), ((\rho \text{ENUM}), 3)p', \text{ENAME}[G]; \]

 \[ \Rightarrow \text{CHOOSE} \]

 End:

 The following is an illustration of the APL code written to implement the application in highly subfunctionized fashion. EMPLOYEES is the driver function.

 ADDEMP;AGE;NAME;NUM

 Exit if 0 entered:

 \[ \Rightarrow 0 \text{ IF } 0 = \text{NUM} \]

 Continue unless employee number already exists:

 \[ \Rightarrow \text{A2 UNLESS } \text{NUM} \in \text{ENUM} \]

 ** EMPLOYEE ',(\$\text{NUM}),' ALREADY IN LIST'

 \[ \Rightarrow \text{A1} \]

 Continue unless name too long:

 \[ \Rightarrow \text{A2 IF}(25 < \rho \text{NAME}) \text{MESSAGE '** NAME TOO LONG'} \]

 \[ \Rightarrow \text{A3 IF}((\rho \text{AGE}) \text{AGE} \vee (\text{AGE} < 17) \text{AGE} > 99) \text{MESSAGE '** AGE MUST BE INTEGER FROM 17 TO 99'} \]

 Catenate new values and ask for more:

 \[ \Rightarrow \text{CATEMP} \]

 End:
\[\text{Chapter 13 WORKSPACE DESIGN AND DOCUMENTATION}\]

\[\text{[WSID: MSF]}\]

\> CATEMP
\>[1] ENUM\ <-> ENUM, NUM
\>[2] \> Pad name to length 25:
\>[3] ENAME\ <-> ENAME RCAT NAME
\>[4] EAGE\ <-> EAGE, AGE
\>

\[\text{[WSID: MSF]}\]

\> R\ <-> CINPUT PROMPT
\>[1] \> Display prompt and ask for response on same line:
\>[2] \> PROMPT
\>[3] \> PROMPT
\> (pPROMPT)
\>

\[\text{[WSID: MSF]}\]

\> DELEMP; GOOD; NUM
\>[1] \> L1:\ <-> 'ENTER EMPLOYEE NUMBERS TO DELETE'
\>[2] \> Ravel to insure a vector, not scalar:
\>[3] \> NUM\ <->, []
\>[4] \> Continue if all valid numbers:
\>[5] \> L2 IF\ ^/ GOOD\ <-> NUM\ <-> ENUM
\>[6] \> NOT FOUND: ', ', (~GOOD)/NUM
\>[7] \> L1
\>[8] \> Flag those employees to keep:
\>[9] \> L2: GOOD\ <-> ENUM\ <-> NUM
\>[10] \> Squeeze out deleted employees:
\>[11] \> SQZEMP GOOD
\>

\[\text{[WSID: MSF]}\]

\> EMPLOYEES; R
\>[1] \> Ask for choice:
\>[2] \> CHOOSE: R\ <-> 'ADLE' SELECT 'ADD, DELETE, LIST OR END: '
\>[3] \> Branch based on response:
\>[4] \> (ADD, DELETE, LIST, END)[R]
\>[5] \> ADD: ADDEMP
\>[6] \> CHOOSE
\>[7] \> DELETE: DELEMP
\>[8] \> CHOOSE
\>[9] \> LIST: LISTEMP
\>[10] \> CHOOSE
\>[11] \> END:
\>
\[ \begin{align*} \text{\textbf{Chapter 13}} & \quad \text{WORKSPACE DESIGN AND DOCUMENTATION} \\
\text{\texttt{\textbackslash v \texttt{R\textasciitilde\texttt{\textbackslash LINE IF CONDITION \[1\] \texttt{R\textasciitilde\texttt{\textbackslash CONDITION/LINE}}} \\
\text{\texttt{\textbackslash v}}} \\
\text{\texttt{\textbackslash v \texttt{LISTEMP;G \[1\] \texttt{\textbackslash o\textasciitilde\texttt{\textbackslash 'NUMBER AGE NAME'}} \[2\] \texttt{\textbackslash o\textasciitilde\textbackslash '}}} \]
\text{\texttt{\textbackslash a Prepare to sort employees by number: \[4\] \texttt{G\textasciitilde\texttt{\textbackslash ENUM}}} \]
\text{\texttt{\textbackslash a Sort and display: \[6\] \texttt{\textbackslash o\textasciitilde\texttt{\textbackslash (5 0 7 0 \ast\texttt{ENUM}[G],[1.5]EAGE[G]),((\rho\texttt{ENUM}),3)\rho\,'\})}, \texttt{ENAME}[G;] \]} \]
\text{\texttt{\textbackslash v}} \\
\text{\texttt{\textbackslash v \texttt{R\textasciitilde\texttt{\textbackslash CONDITION MESSAGE CVEC \[1\] \texttt{R\textasciitilde\texttt{\textbackslash CONDITION}}} \[2\] \texttt{\textbackslash a 0 UNLESS CONDITION \[3\] \texttt{\textbackslash o\textasciitilde\texttt{CVEC}}} \texttt{\textbackslash v}}} \\
\text{\texttt{\textbackslash v \texttt{R\textasciitilde\texttt{\textbackslash NINPUT PROMPT \[1\] \texttt{L1:\texttt{\textbackslash o\textasciitilde\textbackslash PROMPT}}} \[2\] \texttt{\textbackslash a Ravel response to insure a vector, not scalar: \[3\] \texttt{R\textasciitilde\texttt{\textbackslash []}}} \]
\text{\texttt{\textbackslash a Exit if exactly 1 number entered: \[5\] \texttt{\textbackslash a L1 IF(1\neq\rho\texttt{R})MESSAGE '** ENTER 1 NUMBER'}} \texttt{\textbackslash v}}} \\
\text{\texttt{\textbackslash v \texttt{R\textasciitilde\texttt{\textbackslash M RCAT V \[1\] \texttt{R\textasciitilde\texttt{\textbackslash M,[1](1\neq\rho\texttt{M})\uparrow V}}} \texttt{\textbackslash v}}} \\
\text{\texttt{\textbackslash v \texttt{IND\textasciitilde\texttt{\textbackslash CHOICES SELECT PROMPT;}R}}} \]
\text{\texttt{\textbackslash a Ask for choice: \[2\] \texttt{ASK:R\textasciitilde\texttt{\textbackslash CINPUT PROMPT}}} \]
\text{\texttt{\textbackslash a Search vector of choices for 1st char of response: \[4\] \texttt{IND\textasciitilde\texttt{\textbackslash CHOICES\textasciitilde\textbackslash 1\uparrow R}}} \]
\text{\texttt{\textbackslash a Ask again if not a valid choice: \[5\] \texttt{\textbackslash a ASK IF(IND>\rho\texttt{CHOICES})MESSAGE '** INVALID CHOICE. \texttt{\textbackslash CHOOSE FROM: '},\texttt{CHOICES}}} \texttt{\textbackslash v}} \end{align*} \]
By any measure, this sample application is tiny. Yet the advantages and disadvantages of these two extreme approaches emerge even in an application of this size. As the application grows, the differences become more important, even critical. For easy reference, we will use the abbreviations FLF (few large functions) and MSF (many small functions) to refer to the two extreme approaches illustrated above. Let us discuss the pros and cons of each. These considerations should be kept in mind when developing an application system so that the cons are minimized.

1. Utility Functions

A utility function is a usually small (under 20 statements) subfunction which performs a common task and which usually gets its inputs entirely from its arguments (vs. from global variables) and returns its outputs as an explicit result. A well designed utility function will resemble a primitive APL function in its behavior. Some examples of the application of utility functions in the illustrations above include:

\[ \rightarrow A2 \text{ UNLESS } \text{NUM} \leq \text{ENUM} \]
\[ \rightarrow A2 \text{ IF } (25 < \text{Name}) \text{ MESSAGE } '** NAME TOO LONG' \]
\[ \text{Name} \leftarrow \text{CINPUT} 'EMPLOYEE NAME (MAX 25 CHARACTERS):' \]

The FLF approach avoids the use of utility functions while the MSF approach uses them generously. This is a pro for MSF and a con for FLF. Utility functions provide two distinct advantages, both of which improve programmer productivity. The first is that a utility function can replace several lines of common code, allowing you to write and test code faster. For example, compare the following:
AGE=NINPUT 'EMPLOYEE AGE'

vs.

L1:[]='EMPLOYEE AGE'
AGE+,[]
→(l=0AGE)/L2
[]='** ENTER 1 NUMBER'
→L1
L2:

The second advantage is that utility functions can improve code clarity, allowing you to read code faster. For example, compare the following:

→A2 UNLESS NUM∈ENUM

vs.

→(NUM∈ENUM)↓A2
or  →(~NUM∈ENUM)/A2

2. Global Changes

After coding and testing an application, you may be asked by the user to make a change which is pervasive. For example, "Remove the dollar signs from all the numbers being displayed," or "Precede all error messages by a 10 space indent and 3 stars," or "When prompting for numbers, await the response on the same line as the prompt." If you have taken the MSF approach and have used your subfunctions consistently, you may be lucky enough to only have to change a single function. For example, if all error messages are being displayed within a subfunction MESSAGE, you may be able to implement the second request above by changing the line of MESSAGE,

[]+CVEC
to
[]='***',CVEC

This is a pro for MSF and a con for FLF. However, if you have not used your subfunctions consistently, much of the advantage will be lost. For example, if some error messages are being displayed directly and not within the subfunction MESSAGE, you will be forced to conduct an extensive search for those messages. This is exactly what you need to do if you used the FLF approach. Since FLF involves fewer functions than MSG, a slight advantage goes to FLF.
3. Function Size

There exists a school of thought in the community of APL programmers that the ideal size of a function is a page. No line should be wider than a page (i.e. 80 characters or so) and no function should have more lines than will fit on a page (i.e. 50 lines or so). The rationale for this conviction is that the eye and the brain can retain no more than a page or so at a time. Further, by confining each function to a page, the programmer is forced to discern the forest from the trees. The outline of the program logic is placed in the higher level function and the detailed logic is included in subfunctions. The FLF approach violates this standard without remorse. The MSF approach adheres to it rigorously.

Score one for MSF if you want to see the forest and not the trees. Score one for FLF if you want to see the forest and the trees. You can find the trees with the MSF approach but not without leaving the forest (i.e. flipping to another page).

At any rate, smaller function size is a definite pro for MSF when it comes to function editing. If you use a full-screen editor, a small function will fit nicely on a single screen. If you use a line-oriented editor, a large function will suffer more from line renumbering (due to line additions or deletions) than will a small function. For example, if line 15 of a 300 line function is deleted, 285 lines will be renumbered and may need to be reprinted. If line 15 of a 30 line function is deleted, only 15 lines will be renumbered.

4. Self-Containment

The issue of self-containment becomes most obvious when you load the workspace and explore its contents. For FLF:

```apl
)FNS
EMPLOYEES
```

For MSF:

```apl
)FNS
ADDEMP CATEMP CINPUT DELEMP EMPLOYEES IF
LISTEMP MESSAGE NINPUT RCAT SELECT
SQZEMP UNLESS
```

If you decided to merge this application with another, it will be easier to accomplish with the FLF approach. Since the entire application is contained within a single function, the application can be moved about (copying or erasing) as easily as moving the function. With the MSF approach however, you need to be careful that name conflicts (i.e. functions with the same name in two different applications) do not exist.
For example, if you copy the above MSF functions into another application workspace in which a different SELECT function is defined, one of the two SELECT functions will be erased (depending on whether COPY or PCOPY is used). Likewise, if you then erase the above MSF functions from the merged application workspace, the remaining application may no longer work without the IF or CINPUT functions.

Finally, any good written technical documentation will include at least a brief description of each function in an application. The task of writing that documentation is considerably simpler for FLF than for MSF since there are fewer functions to document.

5. Global Passing

Another school of thought in the community of APL programmers states that global variables should be avoided as much as possible. Data should be passed to functions as arguments. There are three reasons for this view:

A. Less confusion. When you are reading a function and you encounter an undocumented global variable, your reading flow is interrupted. What is this variable? Where did it come from? Did I just overlook its assignment? Did the programmer just forget to localize it in the header? Is it a niladic function and not a variable at all? Is it global to this function and assigned outside of it or is it local to this function and assigned within a subfunction to which it is global?

B. Less documentation. To alleviate some of the confusion associated with the use of global variables, you should document a global variable at two points: in the first few lines of any subfunction which requires the global variable; and at the point where any subfunction is called which requires the global variable. For example, the ADDEMP and CATEMP functions defined above should include the following comments:

\[
\begin{align*}
\text{V ADDEMP} \\
\text{[0]} & \text{ A Catenate new values and ask for more:} \\
\text{[0]} & \text{ A (Requires globals: NUM,NAME,AGE)} \\
\text{[0]} & \text{ A (Modifies globals: ENUM,ENAME,EAGE)} \\
\text{[0]} & \text{ CATEMP} \\
\text{V CATEMP} \\
\text{[1]} & \text{ A Called by ADDEMP.} \\
\text{[2]} & \text{ A Requires globals: NUM,NAME,AGE} \\
\text{[3]} & \text{ A Modifies globals: ENUM,ENAME,EAGE}
\end{align*}
\]
C. Fewer localization problems. When a system makes extensive use of global variables, it is easier to forget to localize a variable at the proper level or to localize it at the wrong level. Poorly localized variables can cause some of the most mystifying errors.

Since the FLF approach has fewer functions then the MSF approach, it has fewer subfunction calls and less global variable passing. This is a pro for FLF and a con for MSF.

On the other hand, the MSF approach is easier to employ when you need to use a new name in a function. It is easy to scan a small function to see whether a meaningful name has already been used. With FLF, you may inadvertently re-use the name of a variable still containing valuable information.

6. State Indicator Depth

When you pick up a system written by someone else (or written by you a long time ago), you will most likely start by reading the cover function and then each subfunction as it is called. In order to retain the meaningfulness of what is going on, you must maintain a mental state indicator as you delve into subfunctions. As you finish reading each subfunction, you must remember which function called it and in what context so that you can flip back to that function and continue reading. This process is analogous to the procedure followed by the computer as it is executing the code.

Unfortunately, the human brain cannot maintain its state indicator as flawlessly as the computer. At a depth of 5 or 6, our memories get flaky. If the current state indicator is not kept on paper, you may get lost and have to start again.

In the MSF approach, the state indicator depth grows very rapidly. Even in the tiny application above, it occasionally gets 5 levels deep. For example:

```
UNLESS[1] *
MESSAGE[2]
NINPUT[5]
ADDEMP[1]
EMPLOYEES[6]
```

In an MSF system of any respectable size, the state indicator will occasionally get 10 to 15 levels deep and will average 5 to 9 levels. Such a system is extremely difficult to read until you become familiar enough with the subfunctions that you know what they do without looking into them (like primitive APL functions).

The problem of deep state indicators is especially apparent when you are called upon to handle an error in an unknown MSF system. The first natural step after generating the error is to check the state
indicator. If the state indicator is 9 levels deep, the next natural step is to go to lunch.

Having explored some pros and cons of the FLF and MSF approaches, which should you use? Typically, neither. The most readable and maintainable system is one which employs a moderate number of medium-sized functions, each performing a well-defined task. Utility functions which are self-contained and well-documented should be used generously.

PROBLEM: How is the above application invoked?

TOPIC: Starting an Application

In deciding how to start an application, you must first decide whether or not immediate execution mode will be needed. If so, the user should load the application workspace and then execute (from immediate execution mode) whatever functions are user functions (e.g. EMPLOYEES).

If immediate execution mode is not needed, which is most often the case, you should stay out of that mode until the termination of the application. The main reasons to avoid immediate execution mode are simplicity and security. The application will be simpler to use if the user is prompted for choices rather than having to remember the names of functions. The application will be more secure if non-user functions cannot accidentally or intentionally be invoked.

To "autostart" an application, you should assign the system variable DLX (latent expression) to be the name of the desired cover function before saving the application workspace. For example:

```
DLX←'EMPLOYEES'
)SAVE EMPLOYEES
SAVED.....
```

To initiate the application, the user only needs to load the application workspace. In some installations of APL, even the loading step is unnecessary. A workspace may be specified to be automatically loaded when APL is invoked. In either case, the expression assigned to DLX will be automatically executed. For example:
In this simple application, no special steps need to be taken after the workspace is loaded and before the cover function is executed. In larger applications, this is less likely to be true. Files may need to be tied or shared variables activated and global variables may need to be assigned. In such instances, the value of DLX is more likely to be 'START' or '→RESTART'.

A typical START function will have the following layout:

```
\text{\texttt{\begin{verbatim}
\text{\texttt{V START}}
[1] \text{\texttt{\textbackslash a Workspace driver function. Used as:}}
[2] \text{\texttt{\textbackslash a}}
[3] \text{\texttt{\textbackslash a DLX→'START'}
[4] \text{\texttt{\textbackslash a}}
[5] \text{\texttt{\textbackslash a Display any messages. For example:}}
[6] \text{\texttt{\textbackslash o→'}}
[7] \text{\texttt{\textbackslash o→'WELCOME TO THE EMPLOYEE MAINTENANCE SYSTEM'}
[8] \text{\texttt{\textbackslash o→'}}
[9] \text{\texttt{\textbackslash a Assign any global variables. For example:}}
[10] \text{\texttt{\textbackslash opw←150}}
[11] \text{\texttt{\textbackslash femp←345}}
[12] \text{\texttt{\textbackslash a Tie any files or share any variables. For example:}}
[13] \text{\texttt{\textbackslash empdata\texttt{\textbackslash o\texttt{\textbackslash o\texttt{fortie femp}}}}
[14] \text{\texttt{\textbackslash a Read any global variables from file. For example:}}
[15] \text{\texttt{\textbackslash enum←\texttt{dfread femp,1}}}
[16] \text{\texttt{\textbackslash ename←\texttt{dfread femp,2}}}
[17] \text{\texttt{\textbackslash eage←\texttt{dfread femp,3}}}
[18] \text{\texttt{\textbackslash a Call the cover function. For example:}}
[19] \text{\texttt{\textbackslash employees}}
[20] \text{\texttt{\textbackslash a Do any followup work. For example:}}
[21] \text{\texttt{\textbackslash enum←\texttt{dfreplace femp,1}}}
[22] \text{\texttt{\textbackslash ename←\texttt{dfreplace femp,2}}}
[23] \text{\texttt{\textbackslash eage←\texttt{dfreplace femp,3}}}
[24] \text{\texttt{\textbackslash dfuntie femp}}
[25] \text{\texttt{\textbackslash o→'}}
[26] \text{\texttt{\textbackslash o→'HAVE A NICE DAY'}
[27] \text{\texttt{\textbackslash o→'}}
\text{\texttt{\textbackslash v}}
\end{verbatim}}}
```

A RESTART function (used as DLX→'→RESTART') performs the same tasks as the START function but handles "line drops" in those APL environments which support CONTINUE workspaces. For example, suppose you are connected via terminal, modem and telephone line to a remote APL system. If the telephone line is interrupted (say due to telephone line noise or by accidentally unplugging your modem or terminal), the APL system will detect the drop and will save your active workspace into a stored workspace named CONTINUE. When you next sign on, the CONTINUE workspace will be automatically loaded and
its OLX executed. (In some installations of APL, you may need to load the CONTINUE workspace manually.)

In such an instance, you do not want to run the START function again. Rather, you want to redo any steps undone by the drop (e.g. retie the files or reshare the variables) and then resume execution at the line on which the function at the top of the state indicator is suspended. The RESTART function therefore checks the state indicator (via OLX) and either executes START if there is no suspension or performs restart logic. The final task performed by RESTART is to explicitly return the line number of the suspended function so that execution can resume.

A typical RESTART function will have the following layout:

```apl
[WSID: MSF]

V R←RESTART
[1] a Workspace driver function and line drop handler.
[2] a Used as:
[3] a
[4] a OLX←'→RESTART'
[5] a
[6] a Return the line number of any suspended function
[7] a (beyond RESTART):
[8] R←1↓OLX
[9] a Remove 0's from any $:
[10] R←(R≠0)/R
[11] a Branch if restart logic is necessary:
[12] →(≠0R)/L1
[13] a Otherwise, run START and return R as an empty vector:
[14] START
[15] →0
[16] a Restart logic. Display any messages. For example:
[17] L1:O←'
[18] O←'EMPLOYEE MAINTENANCE SYSTEM BEING RESTARTED'
[19] O←'
[20] a Tie any files or share any variables. For example:
[21] 'EMPDATE' O$TIE fEMP
[22] a Upon exit, line number result will be branched to
[23] a (if OLX←'→RESTART') and execution will resume at
[24] a point of suspension.
```

```
```
PROBLEM: Write a niladic function QDOC (quick documentation) which will display the contents of each function in the workspace, as if you typed \texttt{\textbackslash fnname[]} for each function, in alphabetic order.

TOPIC: Function Documentation

The first step is to determine, under program control, which functions exist in the workspace. The system function \texttt{DNL} (name list) can be used to this end. When used monadically with the right argument 3 (for functions), \texttt{DNL} returns a character matrix of the names of the functions in the active workspace, one row per function. If your implementation of \texttt{DNL} does not return the names in sorted order, sort them (see the Sorting and Searching chapter).

(Some APL implementations have different system functions for returning the names of identifiers. For example, APL*PLUS has \texttt{DIDLIST} and SHARP APL has \texttt{lOWS}. However, these implementations also support \texttt{DNL}.)

The next step is to display the functions, under program control, one at a time. Since APL systems generally do not support an expression like \texttt{\textbackslash fnname[]} , a system function must be used. The APL*PLUS system function \texttt{OVR} (visual representation) and the SHARP APL system function \texttt{OFD} (function definition) both return a character vector "visual representation" of the function whose name is provided as the right argument to the system function. The result, when displayed, looks exactly like the display produced by \texttt{\textbackslash fnname[]}. This is possible because the character vector result contains newline (carriage return) characters at the end of each function line substring. For example:

\begin{verbatim}
\texttt{\textbackslash UNLESS[]} \texttt{\textbackslash R+LINE UNLESS CONDITION}

[1] \texttt{\textbackslash R+CONDITION} \texttt{\line}
\texttt{\textbackslash}

\texttt{\textbackslash CV} \texttt{\textbackslash OVR 'UNLESS'}
\texttt{\rho CV}
\texttt{59}
\texttt{\textbackslash CV}

[1] \texttt{\textbackslash R+LINE UNLESS CONDITION}

[1] \texttt{\textbackslash R+CONDITION} \texttt{\line}
\texttt{\textbackslash}

\texttt{\textbackslash R+LINE UNLESS CONDITION} \texttt{\@} \texttt{\textbackslash R+CONDITION} \texttt{\line}
\texttt{\textbackslash}

\texttt{\textbackslash R+LINE} \texttt{UNLESS CONDITION} \texttt{\@} \texttt{\textbackslash R+CONDITION} \texttt{\line}
\texttt{\textbackslash}
\end{verbatim}
The result of DVR is an empty character vector if the function specified is locked. For implementations which provide such visual representation system functions, the solution to this problem is straightforward:

```apl
v QDOC;FNS;I;N;V;IO
[1] A Displays continuous listing of all functions in ws.
[2] A Origin 1:
[3] IO←1
[4] A Determine functions:
[5] FNS←ONL 3
[6] A Exclude QDOC from list:
[7] FNS←(FNSv.¬(1↑FNS)↑'QDOC')/FNS
[8] A FNS←(^/FNSv.¬(2,1↑FNS)↑2 5p'QDOC CRAVR')/FNS A APL2
[9] A Sort fn names if not already: FNS←FNS[DAV/FNS;
[10] A Loop on rows of FNS:
[11] I←0
[12] N←1↑FNS
[13] LOOP:→(N<1←I+1)/0
[14] V←DVR FNS[I;] A APL*PLUS
[15] A V←1 OFD FNS[I;] A SHARP APL
[16] A V←CRAVR DCR FNS[I;] A APL2
[17] A Ignore if function locked:
[18] →(×V)↓LOOP
[19] 0←V
[20] A Blank line:
[21] 0←'
[22] →LOOP
```

Please note that QDOC will not list functions in the workspace whose names happen to be the same as any of the local identifiers in QDOC (e.g. FNS or LOOP). The system function ONL 3 returns the names of identifiers which are functions at the most local level. Since FNS is a variable and LOOP is a label at the local level, any global function with the same name is "shadowed" and will not be seen. Likewise, DVR (or 1 OFD) returns only the visual representation of identifiers which are interpreted as functions at the local level.

For implementations which do not provide a visual representation system function, you must work with the "canonical" (matrix) representation system function DCR (available in SHARP APL as 2 OFD). The function DCR returns a character matrix representation of the function whose name is provided as the right argument. The result has one row per function line (including the header) and as many columns as the length of the longest function line. The function lines are not numbered, are left justified within their respective rows and are padded to the right with blanks. For example:
The result of OCR is an empty character matrix if the function specified is locked.

To write QDOC, we need a function which will convert this canonical representation result to the more aesthetic visual representation form. The following function will do the trick:

```apl
[WSID: FNREP]
[1] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[2] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[3] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[4] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[5] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[6] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[7] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[8] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[9] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[10] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[11] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[12] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[13] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[14] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[15] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[16] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[17] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[18] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[19] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[20] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[21] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[22] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[23] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[24] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[25] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[26] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[27] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[28] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[29] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[30] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[31] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[32] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[33] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[34] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[35] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[36] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
[37] VR+CR\VR CR;\PIO;C;D;KEEP;L;N;TCNL
```
Chapter 13  WORKSPACE DESIGN AND DOCUMENTATION

\[ CR\Delta VR \text{ (continued)} \]

\[[38]\]  CR\(\text{\{,KEEP\}}/\),CR
\[[39]\]  \& Include header and trailer:
\[[40]\]  VR\&VR,CR,TCNL,' \&',TCNL

Given the CR\&VR function, the QDOC function can be rewritten for OCR implementations by replacing the lines:

\[
\begin{align*}
\text{FNS} & \leftarrow (\text{FNS} \& (1 \downarrow \text{FNS}) \uparrow 'QDOC') \& \text{FNS} \\
\text{V} & \leftarrow \text{VR} \text{ FNS[I;]}
\end{align*}
\]

in the QDOC function above by the corresponding lines:

\[
\begin{align*}
\text{FNS} & \leftarrow (\&/\text{FNS} \& (2,1 \downarrow \text{FNS}) \uparrow 2 5\uparrow 'QDOC \text{ CR}\&VR') \& \text{FNS} \\
\text{V} & \leftarrow \text{CR}\&VR \text{ \& CR} \text{ FNS[I;]}
\end{align*}
\]

PROBLEM: Design a function WSDOC (workspace documentation) which will display the entire contents of the workspace.

TOPIC: Workspace Documentation

One possible solution to this problem is to design a single self-contained function WSDOC which may be copied into the workspace to be documented. Since the function is self-contained, it requires no subfunctions. Therefore, it may be copied into the workspace or erased from the workspace with minimal impact.

Let's establish the differences between the proposed WSDOC function and the QDOC function of the previous section:

1. The output of WSDOC is paged, not continuous.

2. WSDOC is monadic. The elements of its integer vector argument represent: number of rows per page (usually 66), number of columns per page (say 85), lines in top margin (say 3), lines in bottom margin (say 3), columns in left margin (say 5), columns in right margin (say 5).
3. At the top of each page is a title which includes the workspace ID, the current date and time and the page number. For example:

36150 MODEL * 11/15/1986 15:43 PAGE 4

4. The nondefault workspace environment is included at the top of the first page. The workspace environment includes the latent expression, the index origin, the print precision, the random link, the comparison tolerance and any other programmer-controlled workspace settings. Only those settings are displayed whose current values differ from those in a clear workspace. For example:

NONDEFAULT WORKSPACE ENVIRONMENT:

DLX←'START'
DPP+12

5. After the nondefault workspace environment, the global workspace variables are listed in alphabetic order, along with their shapes and up to one line of their raveled values. For example:

GLOBAL WORKSPACE VARIABLES:

<table>
<thead>
<tr>
<th>NAME</th>
<th>SHAPE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>'X'</td>
<td></td>
</tr>
<tr>
<td>MONTHS</td>
<td>12 9</td>
<td>'JANUARY FEBRUARY MARCH...'</td>
</tr>
<tr>
<td>MSG</td>
<td>10</td>
<td>'THAT'S ALL'</td>
</tr>
<tr>
<td>TABLE</td>
<td>2 99 15</td>
<td>1.016283 1.11984 1.61582...</td>
</tr>
<tr>
<td>TIE</td>
<td>368</td>
<td></td>
</tr>
</tbody>
</table>

6. After the global workspace variables, the function names are listed in alphabetic order. For example:

FUNCTIONS:

ADDEMP EMPLOYEES NINPUT UNLESS
CATEMP IF RCAT
CINPUT LISTEMP SELECT
DELEMP MESSAGE SQZEMP

7. Finally, the lines of each function are displayed as in QDOC. However, function lines which are too long (for the page width) are broken into multiple lines with care taken not to break a line in the middle of an identifier or numeric constant. If a function will not fit on the remainder of a page, it is started on the top of the next page. Functions which are longer than one page are broken into multiple pages with care taken not to
display a long line on more than one page. At the bottom right corner of each page is a footnote which displays the first and last functions included on the page. For example:

\[
\text{IF} \rightarrow \text{SELECT}
\]

8. All local variables and labels within WSDOC are prefixed by `\`\`AA\`` to minimize the number of variables and functions which are not recognized because of shadowing.

Let's list the possible problems we may encounter when using such a self-contained WSDOC function:

1. A WS FULL error may occur if there is insufficient available workspace to copy WSDOC. This problem is greater in implementations of APL (such as APL2) in which a function is copied by moving its canonical (matrix) representation. The canonical representation of WSDOC is quite large if it has a lengthy header. To get around this problem, you may remove all local variables from the header and erase all variables beginning with 'AA' on the last line of the function. Alternately, you may load the WSDOC workspace and copy the workspace to be documented. However, bear in mind that the nondefault system variables will not be copied.

2. A SYMBOL TABLE FULL error may occur if there are insufficient available entries in the symbol table for the local variables and labels in WSDOC.

3. Another object which happens to be named WSDOC will be erased and replaced by the WSDOC function when it is copied into the workspace.

If your APL implementation does not support a visual representation system function (e.g. DVR or 1 UFD), you will also need to copy in the function CR\AVA\VR. In this event, WSDOC requires CR\AVA\VR and is not strictly self-contained.

The writing of WSDOC is left as an exercise at the end of the chapter.

The use of this WSDOC function is a simple way to get a neat and thorough listing of the contents of your workspace. If your workspace documentation requirements go beyond the capabilities of this function, you may want to acquire a more comprehensive workspace documentation software package available from your APL vendor. Such
packages typically include capabilities for listing cross-reference information. For example, you can list the functions in which each workspace identifier is used, or list the identifiers used within each function, or display a "tree" diagram which shows which functions are called by other functions, and so on.

~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~

PROBLEM: A function named IDENTIFY analyzes the visual representation of a function to determine which identifiers are used within the function and how. It then displays any known or potential errors or inconsistencies (e.g. assigning a value to a name which is also used as a label). Make a list of all such errors or inconsistencies.

TOPIC: Function Identifiers

The IDENTIFY function as described above is useful for a final validation on any function you have written, especially a large function. After using IDENTIFY, it is a simple matter to edit the function to correct the reported problems.

Here are the problems and illustrations:

1. Redundant label.
   
   :
   [17] L6:Q*B*2

2. Unused identifier localized.

   \ MODEL;A

   (A is not mentioned anywhere in MODEL, though it may be used within a character constant argument to \ or in a subfunction called by MODEL)
3. Identifier localized but not assigned.

\[ v \text{ MODEL};A \\
\mbox{[9]} \; Q\!A\!K \]

(A is not directly assigned, e.g. \( A\!\!B+2 \), anywhere in \text{MODEL}, though it may be assigned within a character constant argument to \( \& \) or in a subfunction called by \text{MODEL})

4. Redundant local variable.

\[ v \text{ MODEL};A;B;R \]
\[ \text{or} \]
\[ v \text{ MODEL};A;B;A;R \]

5. Localized label.

\[ v \text{ MODEL};\text{LOOP};i \]
\[ \mbox{[8]} \; \text{LOOP:}=(\text{LIM}<i)/\text{END} \]

6. Unused label.

\[ v \text{ MODEL} \]
\[ \mbox{[6]} \; L4:K\!\!2+B \]

(No reference is made to \( L4 \), e.g. \( \rightarrow L4 \) or \( \rightarrow (L3,L4,L5)[I] \), anywhere in \text{MODEL}, though it may be used within a character constant (e.g. \( \#(T>0)\)'L4') or may be...gasp...referenced in a subfunction called by \text{MODEL})

7. Assigned label.

\[ v \text{ MODEL} \]
\[ \mbox{[3]} \; \text{END}\!\!99 \]
\[ \mbox{[25]} \; \text{END:}O\!V \]

8. Identifier assigned but not localized.

\[ v \text{ MODEL};A;J \]
\[ \mbox{[7]} \; B\!\!\!t\!pJ \]

(B is not localized in \text{MODEL}, though it may be localized in a function which calls \text{MODEL})
Chapter 13

WORKSPACE DESIGN AND DOCUMENTATION

9. Identifier used and not localized.

\[ \text{\texttt{\textbackslash v MODEL;A}} \]

\[ : \]

\[ \text{\texttt{[6] A\textasciitilde2+B}} \]

(B is not localized in MODEL, though it may be localized in a function which calls MODEL or it may be a subfunction required by MODEL)

10. Result not assigned.

\[ \text{\texttt{\textbackslash v R\textasciitilde\texttt{MODEL \texttt{PARAMS}}}} \]

(R is not assigned in MODEL, though it may be assigned within a character constant to \( \texttt{\~} \) or in a subfunction called by MODEL)

11. Argument not used.

\[ \text{\texttt{\textbackslash v R\textasciitilde\texttt{MODEL \texttt{PARAMS}}}} \]

(PARAMS is not used in MODEL, though it may be used within a character constant argument to \( \texttt{\~} \) or in a subfunction called by MODEL)

The task of writing the IDENTIFY function is beyond the scope of this chapter. It is included as a problem in the Boolean Techniques chapter. IDENTIFY is monadic. Its right argument is the visual representation of the function to be analyzed. For example, to analyze the function MODEL, do the following:

\[ \text{\texttt{IDENTIFY DVR 'MODEL' in APL\textasciitildePLUS}} \]

\[ \text{\texttt{IDENTIFY 1 DFD 'MODEL' in SHARP APL}} \]

\[ \text{\texttt{IDENTIFY \texttt{CR\textasciitildeVR DCR 'MODEL' in another APL system}}} \]

(CR\textasciitildeVR is defined earlier in this chapter.)

There are three related functions also developed in the Boolean Techniques chapter which warrant mention here. They are: RELABEL, LOCALIZE and UNCOMMENT. The right argument of each function, like IDENTIFY, is the visual representation of a function. The result of each function is a modified version of the visual representation, modified to accomplish a particular task. The functions are described below.
Chapter 13 WORKSPACE DESIGN AND DOCUMENTATION

Syntax: NEWVR~LABLIST RELABEL OLDVR

The RELABEL function changes the labels in the visual representation so that they become L1, L2, L3 and so on. In many functions, especially large ones, labels serve simply as branch targets for downward flowing logic. It is difficult and pointless to think up a meaningful name for each label. It is more convenient to the reader to have the labels sequentially numbered so that they can be quickly located. Some labels, however, are best left as meaningful names (e.g. LOOP, END, START, CALC). The left argument LABLIST is a character vector of the names of the labels (separated by spaces) which are not to be renamed. Provide an empty character vector left argument (i.e. '') if all labels are to be renamed.

Do not use RELABEL on any function which contains local variables L1, L2, and so on. Otherwise, these names will refer to both labels and local variables. The resulting function will no longer work correctly.

RELABEL ignores all identifiers within quotes so some labels may not be modified as desired. For example, in the expression,

\[ \text{ELX} \leftarrow ' \rightarrow \text{BELOW}' \]

the reference to the label BELOW will not be detected and modified. To handle this potential problem, you may choose to write such expressions in the following way:

\[ \text{ELX} \leftarrow ' \rightarrow ', \# \text{BELOW} \]

Likewise, the names of labels included in comments are not detected by RELABEL. You should avoid placing labels in comments. For example,

\begin{verbatim}
use: a Branch if quota exceeded
not: a Go to L17 if quota exceeded
\end{verbatim}

RELABEL does not correct any of the problems with labels listed by IDENTIFY.

Syntax: NEWVR~VARLIST LOCALIZE OLDVR

The LOCALIZE function changes the local variables in the header of the visual representation so that the header includes only those variables which are assigned within the visual representation. The LOCALIZE function tends to correct problems 2, 3, 4, 5 and 8 listed by IDENTIFY. Some variables, however, are assigned within a function but should be left global or are not assigned (i.e. are assigned in subfunctions) but should be localized. The left argument VARLIST is a character vector of the names of variables (separated by spaces) which are to be included in the header if not assigned or are to be excluded from the header if assigned. Provide an empty character vector.
vector left argument (i.e. ')') if all and only the assigned variables are to be localized. The localized variables will be included in the header in alphabetic order.

Syntax: NEWVR←UNCOMMENT OLDVR

The UNCOMMENT function removes all comments from the visual representation. End-of-line comments are removed completely, including the comment symbol (a). The comment symbols which precede full-line comments are not deleted so that all function lines remain and do not renumber. Strangely, the UNCOMMENT function allows you to include more comments in functions you write. One argument for omitting or skimping on comments is that comments use up valuable workspace. The UNCOMMENT function allows you to write one set of functions which contain extensive comments (the "maintenance version") and another set which is functionally equivalent but contains no comments (the "production version").

Since the functions RELABEL, LOCALIZE and UNCOMMENT each require a visual representation right argument and each return a visual representation result, they may be "chained" together to perform several functions at once. For example:

NEWVR←'LOOP' RELABEL '' LOCALIZE UNCOMMENT DVR 'MODEL'

However, the visual representation of a function is of little value to you unless you can convert it back into a function. Some APL systems have a system function which will do this directly (DDEF in APL*PLUS and 3 OFD in SHARP APL). The right argument of the system function is the visual representation of a function and the result is the character vector name of the function defined.

Therefore, to relabel a function named MODEL:

N←DDEF '' RELABEL DVR 'MODEL' in APL*PLUS
N←3 OFD '' RELABEL 1 OFD 'MODEL' in SHARP APL

For APL implementations which do not have such a system function, you must use the system function OFX (fix). The right argument to OFX is the canonical (i.e. matrix) representation of a function (as returned by OCR) and the result is the character vector name of the function defined (fixed).

Our task then is to write a function VRACR which will convert the visual representation result of RELABEL, LOCALIZE or UNCOMMENT into a canonical representation so that the function may be defined via OFX. Given the VRACR function, we may relabel a function named MODEL as follows:

N←OFX VRACR '' RELABEL CRVR OCR 'MODEL'
The following VR\rightarrow CR function will perform the necessary conversion.

\[\text{[WSID: FNREP]}\]

\[
\begin{align*}
\text{\texttt{VR}\rightarrow CR} & \rightarrow \text{\texttt{VR;DIO;B;C;D;I;LEN;NL;R;TCNL}} \\
[1] & \rightarrow \text{Converts visual representation of fn to canonical} \\
[2] & \rightarrow \text{representation. Return empty matrix if VR empty} \\
[3] & \rightarrow (\text{locked fn}): \\
[4] & \rightarrow (\rho VR) \rightarrow 0 \\
[5] & \rightarrow \text{Use origin 1:} \\
[6] & \rightarrow \text{Construct newline character:} \\
[7] & \rightarrow \text{TCNL}\rightarrow \text{OTCNL} \rightarrow \text{APL*PLUS} \\
[8] & \rightarrow \text{TCNL}\rightarrow \text{OTCNL} \rightarrow \text{APL2} \\
[9] & \rightarrow \text{TCNL}\rightarrow \text{DAV[157]} \rightarrow \text{SHARP APL} \\
[10] & \rightarrow \text{Select header line (less newline):} \\
[11] & \rightarrow \text{CR}\rightarrow\text{I}\rightarrow\text{VR}\rightarrow\text{TCNL}\rightarrow\rho VR \\
[12] & \rightarrow \text{Drop off header:} \\
[13] & \rightarrow \text{VR}\rightarrow\text{I}\rightarrow\text{VR} \\
[14] & \rightarrow \text{Delete leading \texttt{V} and spaces from header:} \\
[15] & \rightarrow \text{CR}\rightarrow(+/\texttt{VCR}' \texttt{V}') \rightarrow CR \\
[16] & \rightarrow \text{Locate newlines which precede and follow} \\
[17] & \rightarrow \text{each line:} \\
[18] & \rightarrow \text{NL}\rightarrow\text{VR}=\text{TCNL} \\
[19] & \rightarrow \text{Flag starts and ends of contiguous digits} \\
[20] & \rightarrow (\text{e.g. line no.s}): \\
[21] & \rightarrow \text{D} \rightarrow \text{VR} \rightarrow '0123456789' \\
[22] & \rightarrow \text{D} \rightarrow \text{D} \rightarrow (\rho D) \rightarrow 0, D \\
[23] & \rightarrow \text{Flag char following ']' after line no.:} \\
[24] & \rightarrow \text{D} \rightarrow 10\Phi D \rightarrow 10\Phi D \rightarrow 20\Phi NL \\
[25] & \rightarrow \text{Flag starts and ends of contiguous blanks} \\
[26] & \rightarrow (\text{e.g. after line no.s}): \\
[27] & \rightarrow \text{B} \rightarrow \text{VR} \rightarrow ' ' \\
[28] & \rightarrow \text{B} \rightarrow \text{B} \rightarrow (\rho B) \rightarrow 0, B \\
[29] & \rightarrow \text{Flag first nonblank char in each line, as indices:} \\
[30] & \rightarrow \text{D} \rightarrow (D>B) \rightarrow 10\Phi B \rightarrow 10\Phi B / D \\
[31] & \rightarrow \text{D} \rightarrow D / \rho D \\
[32] & \rightarrow \text{Compute lengths of lines:} \\
[33] & \rightarrow \text{LEN} \rightarrow (1\downarrow \text{\texttt{I}} \downarrow \text{\texttt{NL}} / \rho \text{\texttt{NL}}) \rightarrow D \\
[34] & \rightarrow \text{No. of columns in result:} \\
[35] & \rightarrow \text{C} \rightarrow (\rho \text{\texttt{CR}}) \rightarrow \text{\texttt{I}} / \text{LEN} \\
[36] & \rightarrow \text{No. of rows in result:} \\
[37] & \rightarrow \text{R} \rightarrow 1 + \rho \text{LEN} \\
[38] & \rightarrow \text{Initialize result as raveled matrix:} \\
[39] & \rightarrow \text{CR}\rightarrow \text{R} \times \text{\texttt{C}} \rightarrow \text{\texttt{I}} \rightarrow \text{\texttt{CR}} \\
[40] & \rightarrow \text{Construct index vector I} \rightarrow (\downarrow \text{\texttt{LEN}[1]}), (\downarrow \text{\texttt{LEN}[2]}), \ldots \\
[41] & \rightarrow \text{I} \rightarrow \text{LEN} / \rightarrow 1 \downarrow 0, + / \text{\texttt{LEN}} \\
[42] & \rightarrow \text{I} \rightarrow \text{I} / \rho \text{I} \\
[43] & \rightarrow \text{Insert fn lines into raveled result:} \\
[44] & \rightarrow \text{CR}[\downarrow \text{\texttt{I}} + \text{\texttt{LEN}} / \rightarrow \rho \text{\texttt{LEN}}] \rightarrow \text{\texttt{VR}}[\downarrow \text{\texttt{I}} + \text{\texttt{LEN}} / \rightarrow 1 + D] \\
[45] & \rightarrow \text{Reshape result to matrix:} \\
[46] & \rightarrow \text{CR}\rightarrow (R, C) \rightarrow \rho \text{CR} \\
\end{align*}
\]
This function borrows a number of the techniques discussed in the Boolean Techniques chapter. See that chapter for clarification.

PROBLEM: Design a monadic function USEDBY whose argument is a list of functions (character matrix with one name per row or a character vector with names delimited by spaces) and which shows all subfunctions and global variables required by those functions.

TOPIC: Workspace Identifiers

When you inherit the maintenance of an APL application, there are three pieces of documentation which are invaluable to your comprehension of the system. They are:

1. Function listings. If missing, you can reconstruct them by using the WSDOC function defined in this chapter.

2. File structure documentation. If missing, you can hopefully reconstruct it by displaying the data from the files and by inferring meaning from the context in which the files are used (by reading the function listings).

3. System flow charts. If missing, you can hopefully reconstruct them by running the USEDBY function on the user level functions and by reading the function listings.

The following is an illustration of USEDBY on the EMPLOYEES function of the MSF workspace listed earlier in this chapter.
```
USEDBY 'EMPLOYEES'
EMPLOYEES
SELECT
  CINPUT
  IF
  MESSAGE
    UNLESS
ADDEMP
  NINPUT
  IF
    MESSAGE
      UNLESS
      IF
      UNLESS
      ENUM (global)
      CINPUT
      MESSAGE
        UNLESS
CATEMP
  ENUM (global)
  NUM (ADDEMP - local)
  ENAME (global)
  RCAT
  NAME (ADDEMP - local)
  EAGE (global)
  AGE (ADDEMP - local)
DELEMP
  IF
    ENUM (global)
SQUAREMP
    ENUM (global)
    ENAME (global)
    EAGE (global)
LISTEMP
    ENUM (global)
    EAGE (global)
    ENAME (global)
```

The USEDBY function does pretty much what you would do to manually diagram the subfunction and global variable structure of a system. It starts by evaluating the visual representation of the highest level function for global identifiers referenced (i.e. all identifiers but labels, results, arguments or localized variables). For those global identifiers which are themselves functions, it evaluates each one in the same fashion. And so on it recurses deeper or less deep in the fashion of the state indicator during execution of the system.

The writing of USEDBY is left as an exercise at the end of the chapter.

The USEDBY function is extremely powerful and quite complex. It draws heavily on the materials presented in the Boolean Techniques
Chapter 13  WORKSPACE DESIGN AND DOCUMENTATION

It may help you to read that chapter before writing the function or reviewing the solution.

PROBLEMS:  (Solutions on pages 367 to 382)

1. Design other "visual representation manipulation" functions which may be useful. Pattern their syntax and behavior after the IDENTIFY, RELABEL, LOCALIZE and UNCOMMENT functions described in this chapter.

2. Sketch a flowchart of the WSDOC function described in this chapter. Compare it to the flow of the WSDOC function listed in the solutions at the back of the book.

3. Sketch a flowchart of the USEDBY function described in this chapter. Compare it to the USEDBY function listed at the back of the book.
When APL was first implemented, the language included no file capabilities. This shortcoming was quickly recognized as an obstacle to the acceptance of APL as a viable business programming language. Two approaches were taken to overcome the obstacle.

In one approach (shared variables), facilities were developed (OSVO, DSVR, ...) to provide access to existing non-APL file structures. From APL, you can do anything with files that you can do from another programming language. While this approach enables the APL user to communicate with non-APL environments, it leaves the APL purist unsatisfied. It is difficult and disappointing to the APL programmer to work with the concise and consistent APL primitive functions on the one hand and the messy world of records, tracks, blocks, cylinders and disks on the other hand.

In the other approach (shared files), facilities were developed (DFCREATE, DFREAD, ...) to provide access to APL file structures. In the spirit of APL simplicity, an APL file was defined as a list of APL objects residing outside of the active workspace. A file can have any number of objects (called "components") and each object can be of any type, rank or shape. The components are numbered consecutively from 1. A large object can replace a small object without the APL programmer knowing or caring how the storage is being managed on the physical storage device.

If you want to work with APL files but your APL implementation supports only shared variables, look for a public library workspace which uses shared variables to emulate APL files (e.g. the IBM workspace 2 VAPLFILE).

In this chapter, we will discuss some of the more common APL file organizations and the trade-offs between them. We will also discuss the value of quality file documentation and file utility functions.
Chapter 14

FILE DESIGN AND UTILITIES

PROBLEM: Suppose you want to build an APL system for maintaining a database of information about insurance policyholders. For each policyholder, you will keep track of: policy number, issue age, issue date, sex, classification and face amount. Design a file organization for this application.

TOPIC: APL Database File Organization

The ideal file organization for a given application depends upon the size of the database and upon how it is used. Since this information is missing in the description of the problem, we will present several alternative organizations. In the next section, we will discuss the factors to consider when deciding among these organizations.

For simplicity, we will assume that all of the policyholder information may be expressed as numbers (e.g. sex as 0 or 1). We will refer to the items by the abbreviations: POLNO, IAGE, IDATE, SEX, CLASS, AMT.

The descriptions of 8 alternative APL file organizations follow:

1. Record Oriented

<table>
<thead>
<tr>
<th>Comp. No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>POLNO, IAGE, IDATE, SEX, CLASS, AMT for 1st policy</td>
</tr>
<tr>
<td>2</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot; &quot; 2nd &quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot; 3rd &quot;</td>
</tr>
</tbody>
</table>

2. Record Oriented with Deletion Flag

<table>
<thead>
<tr>
<th>Comp. No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STATUS, POLNO, IAGE, IDATE, SEX, CLASS, AMT for 1st policy</td>
</tr>
<tr>
<td></td>
<td>(STATUS=1 if active record, 0 if &quot;deleted&quot;)</td>
</tr>
<tr>
<td>2</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot; 2nd &quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot; &quot; &quot; &quot; &quot; &quot; 3rd &quot;</td>
</tr>
</tbody>
</table>
### 3. Directory

<table>
<thead>
<tr>
<th>Comp. No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Numeric vector (directory) of POLNO for every policy</td>
</tr>
<tr>
<td>2</td>
<td>IAGE, IDATE, SEX, CLASS, AMT for 1st policy (whose POLNO is the 1st element of the directory)</td>
</tr>
<tr>
<td>3</td>
<td>( (I-1) \text{th} ) for 2nd policy</td>
</tr>
</tbody>
</table>

### 4. Directory with Deletion Flag

<table>
<thead>
<tr>
<th>Comp. No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Boolean vector of STATUS for every policy (STATUS=1 if active record, 0 if &quot;deleted&quot;)</td>
</tr>
<tr>
<td>2</td>
<td>Numeric vector (directory) of POLNO for every policy</td>
</tr>
<tr>
<td>3</td>
<td>IAGE, IDATE, SEX, CLASS, AMT for 1st policy (whose POLNO is the 1st element of the directory)</td>
</tr>
<tr>
<td>4</td>
<td>( (I-2) \text{nd} ) for 2nd policy</td>
</tr>
</tbody>
</table>

### 5. Transposed

<table>
<thead>
<tr>
<th>Comp. No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Numeric vector of POLNO for every policy</td>
</tr>
<tr>
<td>2</td>
<td>IAGE</td>
</tr>
<tr>
<td>3</td>
<td>IDATE</td>
</tr>
<tr>
<td>4</td>
<td>SEX</td>
</tr>
<tr>
<td>5</td>
<td>CLASS</td>
</tr>
<tr>
<td>6</td>
<td>AMT</td>
</tr>
</tbody>
</table>
### 6. Transposed with Deletion Flag

<table>
<thead>
<tr>
<th>Comp. No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Boolean vector of STATUS for every policy</td>
</tr>
<tr>
<td>2</td>
<td>Numeric vector of POLNO for every policy</td>
</tr>
</tbody>
</table>
| 3         | " " " IAGE " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " &ndash;
8. Multi-Set Transposed with Deletion Flag

<table>
<thead>
<tr>
<th>Comp. No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Boolean vector of STATUS for 1st 2000 policies (STATUS=1 if active record, 0 if &quot;deleted&quot;)</td>
</tr>
<tr>
<td>2</td>
<td>Numeric vector of POLNO for 1st 2000 policies</td>
</tr>
<tr>
<td>3</td>
<td>&quot; &quot; &quot; IAGE &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot; &quot; &quot; IDATE &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot; &quot; &quot; SEX &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot; &quot; &quot; CLASS &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot; &quot; &quot; AMT &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>8</td>
<td>Boolean vector of STATUS for 2nd 2000 policies</td>
</tr>
<tr>
<td>9</td>
<td>Numeric vector of POLNO for 2nd 2000 policies</td>
</tr>
<tr>
<td>10</td>
<td>&quot; &quot; &quot; IAGE &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>(1+7xi-1)</td>
<td>Boolean vector of STATUS for Ith 2000 policies</td>
</tr>
<tr>
<td>(2+7xi-1)</td>
<td>Numeric vector of POLNO for Ith 2000 policies</td>
</tr>
<tr>
<td>(3+7xi-1)</td>
<td>&quot; &quot; &quot; IAGE &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

This list of file organizations is not exhaustive. It merely illustrates some typical APL file organizations.

At the two extremes are the record oriented and the transposed organizations. The directory organization is a hybrid of the two. The multi-set transposed organization is a modification of the transposed organization designed to avoid WS FULL errors when working with large databases.

The "deletion flag" alternative exists for any file organization. When you need to delete records from a database, you have two alternatives: delete the record now (shifting other records if necessary to fill the void); or flag the record to be deleted but do not delete it until later (by a procedure which restructures the file to remove all flagged records or by the gradual process of replacing flagged records by new records as they are added).
PROBLEM: List the factors to consider when choosing among alternative APL file organizations.

TOPIC: File Design Considerations

To choose among file organizations, you must know how much information is to be stored and how it is to be used. For each type of task, consider how well each file organization will stand up to the demands made upon it. In particular, ask yourself:

1. How may file accesses (i.e. read or write operations) will be required? These take time.

2. How CPU efficient will the task be? Does the organization require significant amounts of processing?

3. How efficient will the task be in terms of workspace storage? Are WS FULL errors likely?

4. How complex is the file structure? Will the programs be difficult to write, to read and to debug?

5. Is redundant file storage required? If so, might the file become excessively large? Could its data get out of synch?

The following is a list of representative tasks which are performed on databases. When choosing a file organization, consider each task. Will this task be performed in this application? How often? How well is it performed in this file organization given the performance measures suggested above?

1. Add 1 record
2. Add 100 records
3. Find/change 1 record (1 item)
4. Find/change 1 record (all items)
5. Find/change 100 records (1 item)
6. Find/change 100 records (all items)
7. Find/delete 1 record
8. Find/delete 100 records
9. Find/list 1 record (5 items)
10. Find/list 1 record (all items)
11. Find/list 100 records (5 items)
12. Find/list 100 records (all items)
13. Summarize all records (1 item)
14. Summarize all records (10 items)

The chart below rates the 8 file organizations presented in the last section for each of these 14 tasks. The letters A (excellent) to F (horrible) are used for rating. These ratings are subjective and
will vary from application to application but this chart is a good guideline.

<table>
<thead>
<tr>
<th>File Organization</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>Task</td>
</tr>
<tr>
<td>A A A B C D C D</td>
<td>Add 1 record</td>
</tr>
<tr>
<td>A A A B A B A B</td>
<td>Add 100 records</td>
</tr>
<tr>
<td>F F A A A B A B</td>
<td>Change 1 record, 1 item</td>
</tr>
<tr>
<td>F F A A B B B B</td>
<td>Change 1 record, all items</td>
</tr>
<tr>
<td>E E C C A B A B</td>
<td>Change 100 records, 1 item</td>
</tr>
<tr>
<td>E E B B B B B B</td>
<td>Change 100 records, all items</td>
</tr>
<tr>
<td>F F B A B A B A B</td>
<td>Delete 1 record</td>
</tr>
<tr>
<td>F E C A B A B A B</td>
<td>Delete 100 records</td>
</tr>
<tr>
<td>F F A A B B B B</td>
<td>List 1 record, 5 items</td>
</tr>
<tr>
<td>F F A A C C C C</td>
<td>List 1 record, all items</td>
</tr>
<tr>
<td>E E C C B B B B</td>
<td>List 100 records, 5 items</td>
</tr>
<tr>
<td>E E B B B B B B</td>
<td>List 100 records, all items</td>
</tr>
<tr>
<td>F F F F A A A A</td>
<td>Summarize all records, 1 item</td>
</tr>
<tr>
<td>E E E E A A A A</td>
<td>Summarize all records, 10 items</td>
</tr>
</tbody>
</table>

Several conclusions may be drawn from this chart:

1. If you intend to do much summarizing or cross-tabulating, you should choose a transposed file organization.

2. Unless you intend to add records and do nothing else, you should avoid a record oriented file organization.

3. No file organization is ideal for all tasks. The best file organization is frequently the one which has the fewest and least severe shortcomings rather than the most strengths. Sometimes a hybrid organization will be the best solution for a given application.
Chapter 14

FILE DESIGN AND UTILITIES

PROBLEM: For a 1,000,000 record APL database, it is critical that a specified record (e.g. policy) be located instantly. How would you organize the file?

TOPIC: Efficient Record Location

The record oriented file organization is out. We do not have the time to do up to 1,000,000 file read operations. Even the multi-set transposed file organization has problems. If blocked at 2000 records per set of components, there will need to be up to 500 file read operations. That is fine for ad hoc file analyses but is unacceptable for instant access.

To solve this problem, we need to utilize the information contained within the key value (record identifier) itself. For example, suppose our records are insurance policies and the record identifier is a policy number. We must use a portion of that number to get us quickly to the vicinity in which the record is located. Consider the following "inverted" directory file:

<table>
<thead>
<tr>
<th>Comp. No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Two-row matrix with one column per policy whose policy number ends with 000:</td>
</tr>
<tr>
<td></td>
<td>[1:] policy number</td>
</tr>
<tr>
<td></td>
<td>[2:] record index where policy data are stored</td>
</tr>
<tr>
<td>2</td>
<td>Ditto for policies ending 001</td>
</tr>
<tr>
<td>3</td>
<td>Ditto for policies ending 002</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>1000</td>
<td>Ditto for policies ending 999</td>
</tr>
</tbody>
</table>

This file could be a companion file for any of the file organizations discussed above. The meaning of "record index" depends upon which file organization is used. For example, if the record oriented file organization is used, the record index can simply be the number of the component in which the record is stored. If a transposed file organization is used, the record index can be a number whose format is SSSSIII where SSSS is the number (index) of the set of components in which the record resides and IIII is the exact index within the components of that set where the record is located.

Given this directory file, any one of the 1,000,000 policies may be located with a single file read operation. For example, to find policy 613821904, read component 905 (i.e. 1+904) of the directory.
file and search its first row for this number. The corresponding element of the second row contains the record index for the policy.

The term "inverted" is used to refer to a file which stores record indices (or pointers) rather than data values. The trade-off for realizing such rapid record location is that the directory must be set up initially and must be updated as records are added or deleted (or their policy numbers changed). This will slow down the record maintenance process somewhat and will make it more complex. Consequently, such a directory should be included only if essential.

An alternative to the inverted file organization is the "layered" file organization. Suppose the file is layered by the last three digits of the policy number. Rather than maintaining a list of the record indices for each possible value (000 to 999), the records are physically segregated by the values. For example, all records whose policy number end with 904 are kept together on file.

This "layering" is fairly easily accomplished with the multi-set transposed file organization. Each set of components contains records for only a single layer value. For example, the first set of components could contain the information for policies whose policy number ends with 625, the second set with 904, the third set with 707, and so on. If there are more policies with numbers ending in 904 than you can place in one set, use more than one set for the records with that layer value.

Suppose you employ a multi-set transposed file organization blocked at 2000 (maximum) records per set of components and layered by the last three digits of the policy number to store the 1,000,000 records discussed above. On average, each set will contain 1000 records. Some more. Some less. If any layer value (e.g. 000) is so popular that it belongs to more than 2000 records, the records of that layer will occupy more than one set. A directory of layer values is maintained as a vector with one element per set and is stored as a single component of the file.

Given this file organization, any one of the 1,000,000 policies may be located with 2 or 3 or so file read operations. For example, to find policy 613821904, read the layer values vector and search it for 904. The matching element(s) identify the set(s) whose records have policy numbers ending with 904. Then, read the policy number component for that set (or sets) and search for the policy number. The result is the index within the set at which that record is located.
PROBLEM: What should be included in written file documentation?

TOPIC: File Documentation

Since a file has no value except as employed in an application, its documentation should be couched in terms relevant to the application. For example, if the file is activated by "tying" it to an arbitrary number, show the tie number which is actually used in the application.

When components are read into the active workspace from the file, they may technically be assigned to any variable name. Show the names which are actually used in the application. When describing a file component, indicate its shape and type and the significance of its value.

The following is an illustration of proper file documentation. Do not waste your time studying its intricacies. Rather, use it to become comfortable with the general structure of good file documentation.

FILE NAME: POLICY

DESCRIPTION: Contains policyholder information

<table>
<thead>
<tr>
<th>Comp. No.</th>
<th>VARIABLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TDATE</td>
<td>Integer scalar of the transaction date (YYYYMMDD) when policyholder information was last added to the file from the administration system.</td>
</tr>
<tr>
<td>2</td>
<td>CTYPES</td>
<td>Integer vector of the available underwriting classification codes.</td>
</tr>
<tr>
<td>3</td>
<td>CNAMEs</td>
<td>10 column character matrix of brief names for the underwriting classes; the rows of CNAMEs are in 1-to-1 correspondence with the elements of CTYPES.</td>
</tr>
<tr>
<td>4</td>
<td>FIV</td>
<td>Field identification vector. Integer vector with one element per field of information on file (e.g. policy number, issue age, sex, ...). The value indicates the type of array</td>
</tr>
</tbody>
</table>
used to store the information:
10: Deletion flag Boolean vector (1: active record)
11: Boolean vector
12: Character vector
13: Integer vector
14: Floating point vector
nn2: Character matrix with nn columns

5 FP

File parameters vector.
FP[1]: Number of active records
FP[2]: Number of records (including deleted)
FP[3]: Number of fields (i.e. pFIV)
FP[4]: Maximum number of records, per set of FP[3] components
FP[5]: Number of sets of FP[3] components

6 ARPS

Active records per set. Integer vector with one element per set (FP[5]) of the number of active (not deleted) records per set. Note: (+/ARPS)=FP[1]

7 RPS

Records per set. Integer vector with one element per set (FP[5]) of the number of records (including deleted) per set. Note: (+/RPS)=FP[2]

8-10 (latent) Empty numeric vector

F+10+FP[3]xS-1 Array of data for field F (1 to FP[3]) in set S (1 to FP[5]). The type and rank of this array is defined by FIV[F]. The length of its first dimension is RPS[S].

The fields of data are:

<table>
<thead>
<tr>
<th>FIELD NO. (F)</th>
<th>VARIABLE NAME</th>
<th>FIV</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STATUS</td>
<td>10</td>
<td>Deletion flag (1=active; 0=deleted)</td>
</tr>
<tr>
<td>2</td>
<td>POLNO</td>
<td>14</td>
<td>Policy number (up to 13 digits)</td>
</tr>
<tr>
<td>3</td>
<td>IAGE</td>
<td>13</td>
<td>Issue age (NN)</td>
</tr>
<tr>
<td>4</td>
<td>IDATE</td>
<td>13</td>
<td>Issue date (YYYYMMDD)</td>
</tr>
<tr>
<td>5</td>
<td>SEX</td>
<td>12</td>
<td>Sex ('M' or 'F' or ' ' if unknown)</td>
</tr>
<tr>
<td>6</td>
<td>CLASS</td>
<td>13</td>
<td>Underwriting classification code (an element of CTYPES)</td>
</tr>
<tr>
<td>7</td>
<td>AMT</td>
<td>13</td>
<td>Face amount (cents)</td>
</tr>
</tbody>
</table>
If you inherit the maintenance responsibility for an application system which has no file documentation, your first task is to reconstruct the file documentation. This is usually possible by carefully reviewing the functions which access the files and by reviewing the file components themselves. To display the functions for your review, use the QDOC or WSDOC functions defined in the Workspace Design and Documentation chapter.

To display the file components, use the FILEDOC function described here. The right argument is the same as that of WSDOC: page height, width, top margin, bottom margin, left margin, right margin (e.g. 66 85 3 3 10 5). The left argument identifies the file to be documented (e.g. file tie number). The output is paged and looks like:

FILE: 21368 POLICY (521 COMPONENTS) * 11/27/86 12:06  PAGE 1

COMPONENT SHAPE ρ VALUE

<table>
<thead>
<tr>
<th>COMPONENT</th>
<th>SHAPES</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19861025</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>11 ρ 31 32 33 34 41 42 42 51 52 53 99</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11 10 ρ 'STANDARD P38K4 P39K4 P3...</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7 ρ 10 14 13 13 12 13 13</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5 ρ 801625 801643 7 2000 401</td>
<td></td>
</tr>
<tr>
<td>8-10</td>
<td>0 ρ 10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2000 ρ 13156281325 21065134890 21065338190...</td>
<td></td>
</tr>
</tbody>
</table>

The writing of FILEDOC is left as an exercise at the end of the chapter.
PROBLEM: Design a set of utility functions for accessing APL files. The functions should be intuitive (i.e. be analogous to APL primitive functions) and should have a syntax which is independent of the chosen file organization and independent of your implementation of APL files.

TOPIC: File Utility Functions

By designing and using such a set of file utility functions, you can become more productive. The procedure for working with files becomes:

1. Design the file organization for a given application;
2. Implement these utility functions for the given file organization;
3. Use the utility functions instead of primitive (e.g. DFREAD or DREAD) file access functions.

By using a consistent set of utility functions, you can work on many application systems without having to continually reorient yourself to the different file organizations. In effect, the utility functions shelter you from the intricacies of each file organization.

Below is a recommended set of file utility functions. In each function, the variable FP (file parameters) represents a numeric scalar or vector which distinctly identifies the file to be used (e.g. file tie number or tie number and blocking factors). If your application deals with just one file and its organization is sufficiently unusual that you are unlikely to need these functions for a similar file, you may omit the FP argument altogether.

The philosophy behind the design of these file utility functions is to treat the file as a matrix in your workspace. The rows of the matrix are called "records". The columns are called "fields". Each utility function emulates some common matrix operation. For example, imagine the data stored in a workspace matrix named FILE. A common matrix operation is adding new records:

\[
\text{FILE} \leftarrow \text{FILE}, [1] \text{NEWDATA}
\]

The corresponding file utility function has the syntax:

FP \leftarrow \text{FP CATREC NEWDATA}

Along with the syntax of each utility function is listed the analogous APL expression for operating on a matrix named FILE.

There are two important distinctions to keep in mind. One is that a field does not need to represent a vector of data. It may be a matrix. For example, a field of employee names may be stored on file as a 20 column character matrix. Here, we treat the names as a
single field. Hence, the analogous APL expression will treat them as a single column.

The second distinction is that "record indices" means values that are understood by the utility functions to identify particular records. They do not necessarily mean \( i \) indices. For example, for a multi-set transposed file organization, a single record index might be a two element vector whose first element is the index of the set and whose second element is the array index within the set.

It is not expected that you will implement all of the following utility functions. Rather, you should select those most useful for the application and implement them.

(STSC's File Manager product -- originally marketed as "EMMA" -- provides a comprehensive set of such file utility functions for an APL*PLUS-based multi-set transposed file organization.)

**SYNTAX:**

```
FP INITFILE FT
FILE←FP;FT
```

INITFILE is used to build an "empty" file whose file parameters are FP and whose field types are defined in FT.

**SYNTAX:**

```
FP→FP CATREC MAT
FILE←FILE,|MAT
```

CATREC is used to add (catenate) records to the end of the file. MAT is a matrix of information to be catenated (or inserted into records flagged for deletion). Each row of MAT represents a single new record. Each column of MAT corresponds to a single field, or column of a matrix field, of data (excluding the deletion flag field, if any). If MAT is a vector, it is treated like a one-row matrix. If a scalar or one-element array, it is catenated to the bottom of each field as a single record. The result is the modified value of FP.

**SYNTAX:**

```
FP→FP CATRECWS NREC
FILE←FILE,|F1,F2,F3,...
```

CATRECWS is used to add (catenate) records to the end of the file. NREC is an integer scalar whose value represents the number of records to be catenated (or inserted into records flagged for deletion). The data for these new records are located in the global field variables \( F_1, F_2, F_3, ... \) where \( F_1 \) is a vector of values for the first field of the record (or a matrix with one row per record), \( F_2 \) is for field 2, \( F_3 \) is for field 3, and so on. One variable is required per field (excluding the deletion flag field, if any). Regardless of the magnitude of NREC, any field variable may be a
one-row matrix, a vector with one element per column of the matrix field, or a scalar or one-element array. The data will be reshaped and catenated to the bottom of the field for NREC records. The result is the modified value of FP. The field variables are erased upon successful completion of the function.

**SYNTAX:**

```
RINDS←(FP,KFLD) IOTA VALUES
RINDS←FILE{;KFLD}VALUES
```

IOTA searches through the file (ignoring deleted records) for the first occurrences of records whose key value (e.g. policy number, employee number, transaction number) is specified in VALUES. IOTA behaves like dyadic \( \iota \). That is, its result contains one index per value in the right argument, in 1-to-1 correspondence. The elements of RINDS are record indices which can be used to directly locate the records. The elements of RINDS are -1 for those elements of VALUES not found. The left argument of IOTA may be just FP if there is only one key (identifying) field. Otherwise, the number of the key field (KFLD) is included in the left argument.

**SYNTAX:**

```
RINDS←IOTARHO FP
RINDS←11pFILE
```

IOTARHO returns the record indices of all active (not deleted) records in the file. IOTARHO behaves like monadic \( 1 \). That is, its result contains all of the indices for the specified array (file). The elements of RINDS are record indices which can be used to directly locate the records.

**SYNTAX:**

```
RINDS←SVEC SLASHIOTARHO FP,SFLDS
RINDS←(~SVEC)/11pFILE
```

SLASHIOTARHO returns the record indices of all active (not deleted) records in the file which satisfy a specified criterion. SVEC is a character vector APL expression (e.g. '\((F3>50)\land F2\neq 0') which defines the desired criterion. The expression is stated in terms of field variables F1, F2, F3, ... which represent the data stored in the 1st, 2nd, 3rd, ... fields of each record. The expression should be constructed such that when executed, its result is a Boolean vector with one element per record (i.e. per element or row of F1, F2, F3, ...) in which ones marks records selected. SLASHIOTARHO behaves like dyadic \( /\iota \). That is, its result contains all of the indices for the specified array (file) which satisfy the specified criterion. The elements of RINDS are record indices which can be used to directly locate the records. The right argument (beyond FP) is an integer vector of the indices of the field variables to be constructed before executing the expression (e.g. 2 3 for '\((F3>50)\land F2\neq 0') or 1 3 7 for '0<PROCESS F3' where F1 and F7 are required by PROCESS as global
variables). If SVEC and SFLDS are empty, SLASHIOTARHO returns the record indices of all active records in the file.

SYNTAX:  \[ \text{FP} + \text{FP} \text{ DELREC RINDS} \]
\[ \text{FILE} \leftarrow \{\{1 \land \text{FILE}\} \in \text{RINDS}\} \land \text{FILE} \]

DELREC deletes specified records from the file. The elements of RINDS are the indices of the records to be deleted (as returned by IOTA, IOTARHO or SLASHIOTARHO). The result is the modified value of FP (if modified).

SYNTAX:  \[ \text{FP} + \text{SVEC} \text{ COMPRESS FP,SFLDS} \]
\[ \text{FILE} \leftarrow \{\text{SVEC}\} \land \text{FILE} \]

COMPRESS deletes all records in the file which do not satisfy a specified set of criteria.

\[ \text{FP} + \text{SVEC} \text{ COMPRESS FP,SFLDS} \]

has the same effect as

\[ \text{FP} + \text{FP} \text{ DELREC ('~',SVEC) SLASHIOTARHO FP,SFLDS} \]

Notice that the former expression does not need to construct intermediate record indices for the selected records and so is more efficient than the latter expression. If SVEC and SFLDS are empty, no records are deleted.

SYNTAX:  \[ \text{MAT} + \text{RINDS INDEX FP,FLDS} \]
\[ \text{MAT} + \text{FILE}[(\text{RINDS};\text{FLDS})] \]

INDEX is used to retrieve from file the data (MAT) from selected fields (FLDS) for specified records (RINDS). The elements of RINDS are the indices of the records to be retrieved (as returned by IOTA, IOTARHO or SLASHIOTARHO). The elements of FLDS are indices of the fields to be retrieved. The result is a matrix of the retrieved data with one row per element of RINDS and one column per element of FLDS.

SYNTAX:  \[ \text{RINDS INDEXWS FP,FLDS} \]
\[ \text{F1} + \text{FILE}[(\text{RINDS};1)] \land \text{F2} + \text{FILE}[(\text{RINDS};2)] \land \ldots \]

INDEXWS is used to retrieve from file the data from selected fields (FLDS) for specified records (RINDS). The elements of RINDS are the indices of the records to be retrieved (as returned by IOTA, IOTARHO or SLASHIOTARHO). The elements of FLDS are indices of the fields to be retrieved. The retrieved data are assigned to global variables.
named Fn where n is the number of the field retrieved (e.g. F3 and F7 for FLDS+3 7). The global variables are vectors with one element (or matrices with one row) per element of RINDS.

SYNTAX: \[ \text{MAT} \leftarrow \text{SVEC SELECT FP,FLDS,0,SFLDS} \]
\[ \text{MAT} \leftarrow (\#\text{SVEC})/\text{FILE}[;FLDS] \]

SELECT is used to retrieve from file the data (MAT) from selected fields (FLDS) for all active (not deleted) records in the file which satisfy a specified set of criteria (SVEC).

\[ \text{MAT} \leftarrow \text{SVEC SELECT FP,FLDS,0,SFLDS} \]

has the same effect as

\[ \text{MAT} \leftarrow (\text{SVEC SLASHIOTARHO FP,SFLDS}) \text{ INDEX FP,FLDS} \]

Notice that the former expression does not need to construct intermediate record indices for the selected records and so is more efficient than the latter expression. If SVEC and SFLDS are empty, SELECT retrieves data for all active records in the file.

SYNTAX: \[ \text{SVEC SELECTWS FP,FLDS,0,SFLDS} \]
\[ \text{F1} \leftarrow (\#\text{SVEC})/\text{FILE}[;1] \odot \text{F2} \leftarrow (\#\text{SVEC})/\text{FILE}[;2] \odot \ldots \]

SELECTWS is used to retrieve from file the data from selected fields (FLDS) for all active (not deleted) records in the file which satisfy a specified set of criteria (SVEC).

\[ \text{SVEC SELECTWS FP,FLDS,0,SFLDS} \]

has the same effect as

\[ (\text{SVEC SLASHIOTARHO FP,SFLDS}) \text{ INDEXWS FP,FLDS} \]

Notice that the former expression does not need to construct intermediate record indices for the selected records and so is more efficient than the latter expression. If SVEC and SFLDS are empty, SELECTWS retrieves data for all active records in the file.

SYNTAX: \[ \text{FP} \leftarrow \text{RINDS INDEXA } (\text{FP,FLDS}) \text{ ASSIGN MAT FILE[ RINDS;FLDS]} \leftarrow \text{MAT} \]

INDEXA is used to replace on file the data in selected fields (FLDS) for specified records (RINDS). The elements of RINDS are the indices of the records to be replaced (as returned by IOTA, IOTARHO or SLASHIOTARHO). The elements of FLDS are indices of the fields to be replaced. MAT is a matrix of the data to be replaced with one row.
per element of RINDS and one column per field (or per column of a matrix field) identified in FLDS. Mat may be a vector if FLDS identifies a single vector field. Regardless of the number of records identified by RINDS, MAT may be a one-row matrix or vector with one element per column of the fields, or a scalar or one-element array. The data will be reshaped and assigned to the specified records. The result is the modified value of FP. The ASSIGN function simply assigns its right argument to <assign> and returns its left argument. INDEXA erases the variable <assign> when done with it.

SYNTAX:  
FP~RINDS INDEXWSA FP,FLDS  
FILE[RINDS;1]~F1 ◦ FILE[RINDS;2]~F2 ◦ ...

INDEXWSA is used to replace on file the data in selected fields (FLDS) for specified records (RINDS). The elements of RINDS are the indices of the records to be replaced (as returned by IOTA, IOTARHO or SLASHIOTARHO). The elements of FLDS are indices of the fields to be replaced. The replaced data is taken from global field variables named Fn where n is the number of the field replaced (e.g. F3 and F7 for FLDS~3 7). The global variables are vectors with one element (or matrices with one row) per element of RINDS. If the field variable is a one-row matrix or a vector with one element per column of the matrix field, it will be applied across all records identified by RINDS. If the field variable is a scalar or one-element array, it will be applied across all records and all columns of the field. The result is the modified value of FP. The field variables are erased upon successful completion of the function.

SYNTAX:  
FP~(FP,XFLDS) EXECUTE XVEC  
*XVEC

EXECUTE is used to execute a specified (XVEC) character vector APL expression (e.g. 'SUM~SUM++/F4'). The expression is stated in terms of field variables F1, F2, F3, ... which represent the data stored in the 1st, 2nd, 3rd, ... fields of each active (not deleted) record. The expression is executed once for each block of records on file (each active record in a record oriented file organization or each set of records in a multi-set transposed file organization). The left argument (beyond FP) is an integer vector of the indices of the field variables involved in the expression. Positive indices indicate fields to be read from file before execution of the expression; negative indices indicate fields to be replaced on the file after execution of the expression. The result is the modified value of FP. For example:
SYNTAX: \[ FP\leftarrow (FP, XFLDS, 0, SFLDS) \text{ EXECUTE } XVEC \text{ FOR SVEC} \]

In this context, EXECUTE is used to execute a specified (XVEC) character vector APL expression for only those active (not deleted) records which satisfy a specified set of criteria (SVEC). The FOR function simply concatenates and returns its two character vector arguments, separating them by a newline character. Both expressions are stated in terms of field variables F1, F2, F3, ... which represent the data stored in the 1st, 2nd, 3rd, ... fields of each record. SVEC should be constructed such that, when executed, its result is a Boolean vector with one element per record (i.e. per element or row of F1, F2, F3, ...) in which ones mark records to be selected for subsequent construction of the field variables to be included in the execution of XVEC. XFLDS and SFLDS are integer vectors of the indices of the field variables involved in the respective expressions XVEC and SVEC. Negative elements of XFLDS indicate fields to be replaced after execution of XVEC. The result is the modified value of FP. For example:

\[ FP\leftarrow (FP, -7 7 2 4 0 3) \text{ EXECUTE } 'F7\leftarrow F7rF2+F4' \text{ FOR } 'F3>1' \]

SYNTAX: \[ RINDS\leftarrow (FP, KFLD) \text{ IOTA VALUES LAYERS Z} \]
\[ RINDS\leftarrow IOTARHO FP \text{ LAYERS Z} \]
\[ RINDS\leftarrow SVEC \text{ SLASHIOTARHO FP, SFLDS LAYERS Z} \]
\[ FP\leftarrow SVEC \text{ COMPRESS FP, SFLDS LAYERS Z} \]
\[ MAT\leftarrow SVEC \text{ SELECT FP, FLDS, 0, SFLDS LAYERS Z} \]
\[ SVEC \text{ SELECTWS FP, FLDS, 0, SFLDS LAYERS Z} \]
\[ FP\leftarrow (FP, XFLDS) \text{ EXECUTE XVEC LAYERS Z} \]
\[ FP\leftarrow (FP, XFLDS, 0, SFLDS) \text{ EXECUTE SVEC FOR SVEC LAYERS Z} \]

LAYERS can be used when the file is layered (see Efficient Record Location section in this chapter). When used, the scope of the file utility function is limited to just those records whose layer value is in the list of layer values Z. All other records are ignored. The LAYERS function simply assigns its right argument to <layers> and returns its left argument. The file utility function erases the variable <layers> when done with it.
**PROBLEM:** Design and document a precise file layout to implement the file utility functions of the previous section for a multi-set transposed file organization. Keep the layout general enough to allow a deletion flag or not, and to allow layers or not.

**TOPIC:** Multi-Set Transposed File Organization

Here is a possible file layout for a multi-set transposed file organization. A discussion follows it.

**FILE NAME:** up to you  
**TIE NUMBER:** up to you  

**DESCRIPTION:** Multi-set transposed file with optional record deletion flags and optional layers.

<table>
<thead>
<tr>
<th>COMP. NO.</th>
<th>VARIABLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOC</td>
<td>Character matrix or vector (with embedded newlines) description of the purpose and contents of this file. [optional]</td>
</tr>
</tbody>
</table>
| 2         | FN       | Character matrix of abbreviated field names (as valid identifier names) with one left justified name per row. [optional]  
\[
1 \leq FN \leq FT
\] |
| 3         | FD       | Character matrix of full field descriptions with one description per row. [optional]  
\[
1 \leq FD \leq FT
\] |
| 4         | FT       | Two row integer matrix of field type information with one column per field (FT[3] columns). The meanings of the values are:  
\[
FT[1:6] = \begin{cases} 
0 & \text{Field width. If 0, the field is inactive (latent). If 1, the field is a vector field. Otherwise, the field is a matrix field with this many columns.} \\
1 & \text{Field datatype. Options: 1 (Boolean), 2 (character), 3 (integer), 4 (floating point).} \\
5-6 & \text{(latent) Available for custom requirements. Contain: 10.}
\end{cases}
\] |
<table>
<thead>
<tr>
<th>COMP. NO.</th>
<th>VARIABLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>FP</td>
<td>Integer vector of file parameters. The meaning of the values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[1]: Field number of layer value field (origin 1) or 0 if file not layered.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[2]: Tie number used when accessing file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[3]: Number of fields, including latent fields. FP[3]=((1+pFT)).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[4]: Number of components preceding the data components. FP[4]≥10.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[5]: Maximum number of records per set of components. (^{\wedge}/FP[5]≥RPS).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[6]: Number of sets, including sets with no active records. FP[6]=((pRPS))=((pARPS)).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[7]: Number of records, including records flagged for deletion. FP[7]=(+/RPS).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[8]: Number of active (not latent) fields, including the deletion flag field if one exists. FP[8]=(+/xFT[l;])</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[9]: Number of active (some active records) sets. FP[9]=(+/0#ARPS).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[10]: Number of active (not flagged for deletion) records. FP[10]=(+/!ARPS).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP[11]: Magnitude is field number of deletion flag Boolean vector field (origin 1) or 0 if no deletion flag. If nonzero: 1(^{\wedge}=FT[l;]FP[11]). If negative: FP[5] inactive records are added for each new set, i.e. FP[5](^{\wedge}=RPS).</td>
</tr>
<tr>
<td>8</td>
<td>RPS</td>
<td>Integer vector with one element per set (FP[6] elements) of the number of records, including those flagged for deletion, in each set.</td>
</tr>
<tr>
<td>9</td>
<td>LV</td>
<td>Layer values. LV is a vector with one element per set (FP[6] elements) if FT[l;FP[1]]=1 or a matrix with one row per set if FT[l;FP[1]]&gt;1. LV is not used (empty numeric vector) if FP[1]=0.</td>
</tr>
<tr>
<td>COMP. NO.</td>
<td>VARIABLE</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>10</td>
<td>ARPS</td>
<td>Integer vector with one element per set (FP[6] elements) whose magnitude is the number of active (not flagged for deletion) records in each set. If positive, the active records in the corresponding set are the leading records (no interspersed deleted records). If negative, the active records are not exclusively the leading records. This component is not used (empty numeric vector) if FP[11]=0.</td>
</tr>
<tr>
<td>(F+FP[4]) DATA +(FP[3])x S-1</td>
<td></td>
<td>Array of data for field F in set S. The array is an empty numeric vector (latent) if FT[1;F]=0. It is a vector if FT[1;F]=1. It is an n-column matrix (n=FT[1;F]) if n≠1. The array (if not latent) has RPS[S] elements or rows, of which</td>
</tr>
</tbody>
</table>

Components 1 (DOC), 2 (FN) and 3 (FD) are not essential pieces of the file. However, since they serve to document the file, they are included and recommended.

Components 4 (FT) and 7 (FP) completely define the structure of the file. They are constructed to satisfy the requirements of the particular application. After creating an empty file (via UFCREATE or DCREATE) and assigning values to FT and FP, you initialize the file by calling INITFILE with FP and FT as its arguments.

Components 5 and 6 contain empty vectors and are not used. It is a good practice to leave a few "latent" components for future unanticipated requirements.

Components 8 (RPS), 9 (LV) and 10 (ARPS) are used and updated as needed by the utility functions.

Components 11 to FP[4] are additional latent components in case more than 2 (components 5 and 6) are needed for the particular application.

All components beyond FP[4] are reserved for the data.
The implementation of the file utility functions for this specific file layout is left as an exercise in the problems at the end of the chapter.

This file layout is illustrated and discussed further in the next section.

PROBLEM: Construct a file for maintaining a database of information about 45,900 insurance policyholders. For each policyholder, you will keep track of: policy number (12 digits/letters), issue age, issue date, sex (M or F), classification (S,A,B,C,D) and face amount. Layer the file by classification.

TOPIC: An Illustration of File Utilities

The first step is to define the fields for this file. They are:

1. Policy number
2. Issue age
3. Issue date
4. Sex
5. Classification
6. Face amount
7. Deletion bit

The last field, deletion bit, is optional. You should consider how the file will be used before you decide whether or not to employ a deletion bit field. If employed, record deletion is quick because deleted records are not physically removed from file. They are simply flagged for deletion (bit=0). However, record searching and retrieval will be slower because "deleted" records must be detected and ignored.

For this illustration, we will employ the deletion bit field.

We have defined 7 fields but let's allow room for another 3 fields for future expansion:

8. (latent)
9. (latent)
10. (latent)
The next step is to specify the nature of each field precisely by constructing FT, which will be component 4 of the file.

```
WIDTH=12 1 1 1 1 1 1 0 0 0
TYPE = 2 3 3 2 2 4 1 1 1 1
FT=WIDTH,[.5]TYPE
```

Since the policy numbers (field 1) may contain letters as well as digits, they will be stored in a 12 column character (type 2) matrix field. Issue age (field 2) and issue date (field 3) will be stored as integer (type 3) vector (width 1) fields. The dates will be stored in YYYYMMDD format. Sex (field 4) and classification (field 5) will be stored as character vector fields. Face amount (field 6) will be stored as a floating point (type 4) vector field since the values may contain cents but will be stored in dollar units. Deletion bit (field 7) must be stored as a Boolean (type 1) vector field. Fields 8 to 10 are stored as latent (width 0) Boolean fields.

Next, define the elements of FP, which will be component 7 of the file. The file will be layered by classification which is field 5:

```
LAYER=5
```

Let's pick a tie number to which the file will be tied when used:

```
TIE=987
```

There are 7 active fields and 10 fields in all:

```
AFLDS=7
INCR=10
```

The file begins with 10 reserved components. Let's include another 40 latent components (11 through 50) in case we later need a place to store related items.

```
DISP=50
```

The file will contain 45,000 or so records. Let's keep our components down to some manageable size so WS FULL errors are kept to a minimum and so new records may be added without having to read giant objects. Let's arbitrarily limit each data object to 2000 records. This means the 45,000 record file will contain at least 23 sets of data components. To read one field for all records will require at least 23 file read operations. By increasing the blocking factor, the number of sets decreases, and vice versa. For some applications, you may want a block size of 30,000. For others, a block size of 10 may be ideal. Here, we will use 2000.

```
BLK=2000
```

Since we are using a deletion bit field, we can choose either of two methods for adding new sets. After the 2000th record is added to the last set of the file and another record is to be added, a new set
must be appended to the file. This set may be appended with just the single new record, or with 2000 records, 1999 of which are flagged deleted. If the latter approach is taken, subsequent new records will simply replace "deleted" records.

From your point of view, as the programmer using the file utility functions, the utility functions behave the same regardless of the approach chosen. Which approach you should use is a function of your APL file system implementation. Some systems behave poorly (i.e. gobble up much disk storage) when asked to replace an object in a component by a larger object. On such systems, you should choose the latter approach so that all the records of the set are added at once. Then, the components in that set will not grow.

Here, we will choose the latter approach by specifying the number of the deletion field as a negative number:

DFLD ← -7

Let's construct FP:

FP ← LAYER, TIE, INCR, DISP, BLK, 0, 0, AFLDS, 0, 0, DFLD

We will name the file 'POLDATA'. Create the file and run INITFILE:

'POLDATA' DFCREATE TIE (or DCREATE on SHARP APL)
FP INITFILE PT

At this point the file has 50 (i.e. DISP) components, no records and no sets. Before we start adding records, let's take a valuable moment to construct and replace the 3 documentation components:

DOC ← 'Insurance policyholder database'
FN ← 7 5 'PNUM IAGE IDATESEX CLASSFACE DBIT'
FD ← (15↑'Policy number'), (1...), (15↑'Deletion bit')
DOC DFREPLACE TIE, 1 (or DFREPLACE on SHARP APL)
FN DFREPLACE TIE, 2
FD DFREPLACE TIE, 3

From this point on, we can simply use the file utility functions. Let's catenate 4 records:

F1 ← (12↑'ABCD'), (11↑'A1'), (12↑'XYZ99'), (5↑12↑'P55' (policy number)
   (issue age)
F2 ← 55 45 35
F3 ← 19820715 19850123 19851230 19860402 (issue date)
F4 ← 'M' (sex, all male)
F5 ← 'SASS' (classification)
F6 ← 50000 30000 40000 25000 (face amount)

FP ← FP CATRECWS 4
Which ones are 45 or older?

'F2≥45' SELECT FP,1 0 2
XYZ99
A1

Make the 45 year old a female:

FP←((FP,2) IOTA 45) INDEXA FP,4 ASSIGN 'F'

Increase each face amount by a factor of 100 for those with standard (S) classification:

FP←(FP,−6 6) EXECUTE 'F6←100×F6' LAYERS 'S'

Return the ages of those with standard (S) classification:

'.' SELECT FP,2 LAYERS 'S'
25 45 35

Delete policyholder records with issue dates in 1985:

FP←'1985∥F3÷10000' COMPRESS FP,3

When done using the file, untie it:

OFUNTIE FP[2] (or UNTIE on SHARP APL)

To use the file again later, retie the file and read the file parameters vector from component 7:

'POLDATA' OFTIE 987 (or TIE on SHARP APL)
FP←OFREAD 987 7 (or DREAD on SHARP APL)

You can then use the file utility functions again.

PROBLEMS: (Solutions on pages 383 to 448)

1. Suppose you have developed an application and are encountering frequent WS FULL error messages. Perhaps there are too many functions in the workspace. Design and implement a set of file utility functions which may be used to store some of these functions on file and to retrieve them when needed.
2. Sketch a flowchart of the FILEDOC function described in this chapter. Compare it to the flow of the FILEDOC function listed in the solutions at the back of the book.

3. In the Workspace Design and Documentation chapter, a simple application system is developed for maintaining a list of employees. Rewrite the EMPLOYEES function (the large one) to assume that employee information will be kept on file rather than in global workspace variables. Do not design a precise file organization. Rather, use the file utility functions introduced in this chapter.

4. Try your hand at writing one or more of the file utility functions for the specific multi-set transposed file layout presented in this chapter. Compare your function to the listing of that function included in the solutions at the back of the book. The functions listed require an APL*PLUS system. If you use SHARP APL or APL2, see the next problem.

5. What general modifications must be made to the APL*PLUS system file utility functions written in the previous problem so that they will work on a SHARP APL system? On an APL2 system?
Chapter 15

BOOLEAN TECHNIQUES

The treatment of logical conditions in APL is simple and powerful. The concept of "true" is represented by the numeric value 1 and "false" by 0. These values may be manipulated with the same ease as those of any other numeric values. An array which contains only 1s and 0s is called a "Boolean" array (after the mathematician George Boole) or a "bit" array (after the unit of computer storage).

The APL language contains 7 primitive functions which return exclusively Boolean results (=,≠,>,≥,<,≤,∈). These are called relational functions. There are 5 primitive functions which not only return Boolean results, but also require exclusively Boolean arguments (~,∧,∨,≠,≡). These are called logical functions.

On the surface, these 12 functions provide an adequate, but not wonderful, set of capabilities for working with logical data. However, by applying the APL operators (such as reduction and scan) to these functions and by utilizing the "shift and compare" techniques available in APL, you can begin to appreciate the rich Boolean functionality of APL. Extremely complex problems can be solved directly using noniterative Boolean techniques that would never even be considered in another programming language.

This chapter presents no new or complex APL primitive functions. Rather it presents deeper interpretations of existing simple functions. The aim of the chapter is to develop your Boolean vocabulary and to broaden your thinking when faced with Boolean problems.
PROBLEM: The dyadic logical functions \( \land, \lor, \land, \lor \) require Boolean arguments and return Boolean results. Since they are scalar functions, the result is a scalar when both arguments are scalars. For the set of four possible pairs of arguments (0 and 0; 0 and 1; 1 and 0; 1 and 1), each function returns its four distinct Boolean results. For example, the results of the \( \land \) (and) function may be expressed in the table:

<table>
<thead>
<tr>
<th>Right Argument</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \land )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Viewing this table of \( \land \) results as a vector \((0 \ 0 \ 0 \ 1)\), you can see that the results for \( \lor, \land, \lor \) are respectively: \(0 \ 1 \ 1 \ 1, 1 \ 1 \ 1 \ 0, 1 \ 0 \ 0 \ 0\). There are 16 possible combinations of 4 bits. These 4 functions produce only 4 of them. What functions may be used to generate the rest? What logical interpretations can be given to each of these functions?

TOPIC: Logical Scalar Functions

For a left argument \( L \equiv 0 \ 0 \ 1 \ 1 \) and a right argument \( R \equiv 0 \ 1 \ 0 \ 1 \), the 16 possible combinations of results may be generated by the following expressions:
### Boolean Techniques

#### Expression

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interpretation</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L \And 0)</td>
<td>Always false</td>
<td>(L \And \neg R)</td>
</tr>
<tr>
<td>(L \Or 0)</td>
<td>And (both; &quot;multiplication&quot;)</td>
<td>(L \Or R)</td>
</tr>
<tr>
<td>(L \Imp 0)</td>
<td>Except (unless; and not; &quot;subtraction&quot;)</td>
<td>(L \Imp R)</td>
</tr>
<tr>
<td>(L)</td>
<td>Left argument</td>
<td>(\neg L)</td>
</tr>
<tr>
<td>(L \Land 0)</td>
<td>Nor not</td>
<td>(\neg L \Or R)</td>
</tr>
<tr>
<td>(L \Land R)</td>
<td>Right argument</td>
<td>(L \Imp R)</td>
</tr>
<tr>
<td>(L \Not R)</td>
<td>Toggle if (exclusive or)</td>
<td>(L \Not \Land \Or R)</td>
</tr>
<tr>
<td>(L \Land \Or R)</td>
<td>Or (&quot;addition&quot;)</td>
<td>(L \Or \neg R)</td>
</tr>
<tr>
<td>(L \Or \neg R)</td>
<td>Nor (neither)</td>
<td>(\neg (L \Or R))</td>
</tr>
<tr>
<td>(L = R)</td>
<td>Not right argument</td>
<td>(L \Imp \neg R)</td>
</tr>
<tr>
<td>(L \Not \Imp R)</td>
<td>Or not</td>
<td>(L \Imp R)</td>
</tr>
<tr>
<td>(L \Imp \neg R)</td>
<td>Toggle if not</td>
<td>(L \land \neg R)</td>
</tr>
<tr>
<td>(L \Not \Land \Or R)</td>
<td>Not left argument</td>
<td>(\neg (L \land R))</td>
</tr>
<tr>
<td>(L \land \neg R)</td>
<td>Nand not</td>
<td>(\neg (L \land R))</td>
</tr>
<tr>
<td>(L \land R)</td>
<td>Nand (not both)</td>
<td>(\neg (L \land R))</td>
</tr>
</tbody>
</table>

To illustrate the use of these logical expressions, let us solve some problems.

**A.** Given a vector DEP of bank deposits, how many deposits are between (inclusive) 100 and 200?

\[
+/((\text{DEP} \geq 100) \land \text{DEP} \leq 200) \quad \text{(and)}
\]
\[
+/((\text{DEP} < 100) \lor \text{DEP} > 200) \quad \text{(or)}
\]

**B.** How many are greater than 250, ignoring those which are exactly 500?

\[
+/((\text{DEP} > 250) \land \neg (\text{DEP} = 500)) \quad \text{(except, and not)}
\]
\[
+/((\text{DEP} = 500) \lor \text{DEP} > 250) \quad \text{(nor not)}
\]

(It is easy to see why the second expression is typically read "how many deposits where (DEP>250) except where (DEP=500)", rather than "how many deposits where (DEP=500) nor not where (DEP>250)".)

**C.** How many are either 100 or greater than 250, ignoring those which are exactly 500?

\[
+/((\text{DEP} > 250) \lor \neg (\text{DEP} \leq 100) \lor 500) \quad \text{(toggle if)}
\]
\[
+/((\text{DEP} \leq 250) \lor \neg (\text{DEP} \leq 100) \lor 500) \quad \text{(toggle if not)}
\]
D. How many are smaller than 10 or larger than 1000?

\[ (+/(\text{DEP}<10) \lor \text{DEP}>1000) \text{ (or)} \]
\[ (+/(\text{DEP} \geq 10) \land \text{DEP} \leq 1000) \text{ (nand)} \]
\[ (+/(\text{DEP} < 10) \lor \text{DEP} \geq 1000) \text{ (or not)} \]
\[ (+/(\text{DEP} \geq 10) \land \text{DEP} > 1000) \text{ (nand not)} \]

---

PROBLEM: There are 10 scalar dyadic relational or logical functions
\( (=, \neq, >, \geq, <, \leq, \land, \lor, \neg, \mathcal{V}) \). All of these may be used with the reduction or scan operators to derive functions which can operate on Boolean arrays. Which of these 20 derived functions have useful interpretations? What are the interpretations?

TOPIC: Logical Reductions and Scans

Obviously, the word "useful" is subjective. So let us be subjective. Only two of the reductions have useful interpretations. However, we will throw in a third reduction (+/) since its interpretation becomes "how many" rather than "add up" when its argument is Boolean.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+/R</td>
<td>How many</td>
</tr>
<tr>
<td>^/R</td>
<td>All</td>
</tr>
<tr>
<td>v/R</td>
<td>Any</td>
</tr>
</tbody>
</table>

To illustrate these functions, let us solve some problems.

A. Given a character vector CVEC, how many nonblank elements does it contain?

\[ +/\text{CVEC}' ' \text{ (how many)} \]

B. Are all of the elements nonblank?

\[ ^/\text{CVEC}' ' \text{ (all)} \]
\[ \sim v/\text{CVEC}=' ' \text{ (not any: none)} \]
Six of the scans have useful interpretations.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>^\R</td>
<td>Leading</td>
</tr>
<tr>
<td>\V\R</td>
<td>Not leading not (leading 0s)</td>
</tr>
<tr>
<td>&lt;\R</td>
<td>First</td>
</tr>
<tr>
<td>&lt;\R</td>
<td>Not first not (first 0)</td>
</tr>
<tr>
<td>#\R</td>
<td>Leading 1-poles to 1-maps</td>
</tr>
<tr>
<td>=\R</td>
<td>Leading 0-poles to 0-maps</td>
</tr>
</tbody>
</table>

Let us solve some problems using these functions.

A. How many leading blanks (blanks before the first nonblank) exist in the character vector CVEC?

\+/^\CVEC=''

(how many leading)

B. Delete the leading blanks from CVEC, returning the elements beyond the leading blanks.

(\V\CVEC\#')/CVEC

(leading 0s)

C. Is the first nonblank character in CVEC a digit?

\V/((<\CVEC\#')/CVEC)\epsilon'0123456789'

(first)

(\V/, i.e. any, is used in case there is no first nonblank)

D. Delete the first ',' from CVEC, returning the rest of CVEC.

(\=\CVEC\#',')/CVEC

(first 0)

The remaining two scans (\#\ and =\) require some explanation before using them to solve problems. A "maps" vector is a Boolean vector which consists of sets of contiguous 1s (1-maps) separated by one or more 0s (0-maps). For example, the following bit vector contains 3 1-maps (each of which is underlined) and 4 0-maps:

0 0 1 1 1 0 0 0 1 1 0 0 1 1 1 1 0 0

A "leading 1-poles" vector is a Boolean vector which consists of pairs of 1s, separated by zero or more 0s. The 1s are called "poles". The left pole in each pair may be viewed as the starting element of a set of contiguous elements. The right pole in each pair may be viewed as the next element beyond the ending element of the
set. For example, the following bit vector contains 3 pairs of leading 1-poles.

```
0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0
```

Notice that 1-maps and leading 1-poles are alternate means of conveying the same information. Specifically, they each identify spans of contiguous elements. 1-maps do so by using 1s to flag the elements within the spans. Leading 1-poles do so by using 1s to flag the starts of spans and the starts of non-spans (hence the word "leading").

The \# function converts bit vectors from the leading 1-pole representation to the 1-maps representation:

```
0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0
0 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 0
0 0 1 1 1 1 0 0 0 1 1 0 0 1 1 1 1 1 0
```

The =\ function converts bit vectors from the leading 0-poles representation (use 0s as poles instead of 1s) to the 0-maps representation (use 0s as maps instead of 1s).

```
0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1
1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1
1 1 0 0 0 0 1 1 1 0 0 1 1 0 0 0 0 1 1
```

Let us solve some problems.

E. Given a character vector CVEC, return all characters within quotes.

```
(#\CVEC=\''\)/CVEC
```

(This expression also returns the leading quote of each quote pair and the second quote from each pair of "doubled" quotes within quote pairs.)

F. Return everything in CVEC except quote characters or characters within quotes.

```
T< \CVEC\''
(T^\#1^T)/CVEC
```

```
PROBLEM: In the last expression of the final illustration in the prior section, a shift-and-compare operation (TA¬1ΦT) is performed to produce the effect of extending each 0-map to the right by one element. What other shift-and-compare operations have useful interpretations when applied on Boolean arrays? What are the interpretations?

TOPIC: Logical Shift-and-Compare (Map) Operations

In the following list of shift-and-compare operations, the catenate and drop technique (e.g. ¬1↓0,B) is used instead of the rotate technique (e.g. ¬1ΦB). The reason for this choice is that the rotate technique has the undesirable effect of "filling" the first element (or last element if 1ΦB) with the arbitrary value of the last (or first) element of the array, rather than with the 1 or 0 needed to make the comparison work in every case.

Also notice that the catenate is done before the drop (e.g. ¬1↓0,B) instead of the other way around (e.g. 0,¬1↓B) so that the expression will behave correctly when the argument is empty.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>R¬1↓0,R</td>
<td>1-maps to leading 1-poles</td>
</tr>
<tr>
<td>R=¬1↓1,R</td>
<td>0-maps to leading 0-poles</td>
</tr>
<tr>
<td>R&gt;¬1↓0,R</td>
<td>1-maps to first 1 bits</td>
</tr>
<tr>
<td>R≥¬1↓1,R</td>
<td>0-maps to first 0 bits</td>
</tr>
<tr>
<td>Rv¬1↓0,R</td>
<td>extend 1-maps to right by 1</td>
</tr>
<tr>
<td></td>
<td>(shorten 0-maps from left by 1)</td>
</tr>
<tr>
<td>RA¬1↓1,R</td>
<td>extend 0-maps to right by 1</td>
</tr>
<tr>
<td></td>
<td>(shorten 1-maps from left by 1)</td>
</tr>
</tbody>
</table>

Since each shift-and-compare operation transforms a map, these operations are sometimes called "map" operations.

Let us solve some problems using these operations.

A. Given a character vector SENTENCE, return the lengths of the words in it. A word is any set of contiguous letters.

LETTERS←'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef...
MAPS←(SENTENCE∈LETTERS),0         (0 to insure last element not a letter)
POLES+MAPS¬1↓0,MAPS                (1-maps to leading 1-poles)
INDS+POLES/\pPOLES                  (indices of poles)
INDS+((\pINDS)÷2),2)\pINDS        (reshape to 2 column matrix)
¬/\pINDS                           (subtract in pairs)
B. Determine the indices of the first letter of each word.

\[
\text{MAPS} \cdot \text{SENTENCE} \cdot \text{LETTERS} \\
\left( \text{MAPS} > \text{~1} \cdot 0, \text{MAPS} \right) / \text{~pMAPS} \\
\text{(1-maps to first 1 bits)}
\]

C. Given a numeric vector BAL, how many times do the values go from positive to negative or vice-versa?

\[
\text{MAPS} \cdot \text{BAL} < 0 \\
+/1\text{MAPS} \cdot \text{~1} \cdot 0, \text{MAPS} \\
\text{(1-maps to leading 1-poles)}
\]

D. Delete the extraneous (leading, trailing or redundant) blanks from a character vector CVEC.

\[
\text{NB} \cdot \text{CVEC} \cdot \text{~'} \\
\text{CVEC} \cdot (\text{NB} \cdot \text{~1} \cdot 0, \text{NB}) / \text{CVEC} \\
\text{CVEC} \cdot (-' \cdot \text{~}^{-} \text{1} \cdot \text{CVEC}) \downarrow \text{CVEC} \\
\text{(extend 1-maps to right by 1)}
\]

\[
\text{CVEC} \sim (-' \cdot \text{~}^{-} \text{1} \cdot \text{CVEC}) \downarrow \text{CVEC} \\
\text{(drop last element if blank)}
\]

PROBLEM: Some applications, such as text processing, are well-suited to the Boolean techniques discussed in the prior sections. These techniques allow you to analyze an array without iterating by character. For example, (+/\text{\R'= '}) tells you how many leading blanks are in the array \text{R}. At times, you will want to work with an array which is the concatenation of several arrays (e.g. the sentences in a paragraph). Then you may want to apply the Boolean operations on each respective "subarray" (e.g. the number of leading blanks in each sentence of a paragraph). Design and implement a set of Boolean utility functions which will perform the operations described in the prior sections, for each of the subarrays in a specified array.

TOPIC: Logical Partition Operations

We will use the term "partition" to refer to each subarray of an array. For example, we may view the following character vector as consisting of 3 partitions (sentences):

'HELLO. THIS IS A SAMPLE. WHAT DO YOU THINK?'

There are a number of different methods which may be used to define where in this character vector each of the partitions begins and
ends. For example, we could specify the indices of the starting elements and of the ending elements for each partition; or we could specify the starting indices and the lengths of each partition.

Since we will be using Boolean techniques discussed above, we will define the locations of the partitions by specifying a Boolean vector which has as many elements as the character vector and which has 1s in the indices which correspond to the first element of each partition. Such a Boolean vector will be called a "partition vector".

A partition vector for the character vector above is defined below (with spaces removed from the display of the partition vector for clarity):

```
CVEC~'HELLO. THIS IS A SAMPLE. WHAT DO YOU THINK?'
PV ← 100000100000000000000000010000000000000000000
```

We desire a set of functions which will perform each of the Boolean operations described in prior sections, but will do so independently on each of the partitions of a specified array. Since the functions require the knowledge of how the array is partitioned, the partition vector must be an argument to each function. For example, if +/ and ^\ were defined to permit a partition vector left argument, we could determine the number of leading blanks in each sentence of CVEC with the following expression:

```
PV+/PV\CVEC=' '
```

The result would contain one element per partition (e.g. 0 2 2).

Since +/ and ^\ do not, in fact, accept partition vector left arguments, we will design our own set of "partition functions".

<table>
<thead>
<tr>
<th>Non-Partition Expression</th>
<th>Partition (PV) Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>+/R</td>
<td>PV pPLUSRED R</td>
</tr>
<tr>
<td>^/R</td>
<td>PV pANDRED R</td>
</tr>
<tr>
<td>∨/R</td>
<td>PV pORRED R</td>
</tr>
<tr>
<td>∨\R</td>
<td>PV pANDSCAN R</td>
</tr>
<tr>
<td>^\R</td>
<td>PV pORSCAN R</td>
</tr>
<tr>
<td>&lt;\R</td>
<td>PV pLTSCAN R</td>
</tr>
<tr>
<td>≤\R</td>
<td>PV pLESCAN R</td>
</tr>
<tr>
<td>≠\R</td>
<td>PV pNESCAN R</td>
</tr>
<tr>
<td>%=R</td>
<td>PV pEQSCAN R</td>
</tr>
<tr>
<td>R≠ᶜ11,0,R</td>
<td>PV pNEMAP R</td>
</tr>
<tr>
<td>R≡ᶜ11,1,R</td>
<td>PV pEQMAP R</td>
</tr>
<tr>
<td>R¬¹1,0,R</td>
<td>PV pGTMAP R</td>
</tr>
<tr>
<td>R≡ᶜ11,1,R</td>
<td>PV pGEMAP R</td>
</tr>
<tr>
<td>Rνᶜ11,0,R</td>
<td>PV pORMAP R</td>
</tr>
<tr>
<td>R∧ᶜ11,1,R</td>
<td>PV pANDMAP R</td>
</tr>
</tbody>
</table>
(Notice that the logical scalar functions (e.g. L>R) are not included here since the scalar functions work correctly whether or not their arguments are partitioned.)

The number of leading blanks in each sentence in CVEC is:

\[ \text{PV pPLUSRED PV pANDSCAN CVEC} = ' ' \]

The use of these functions will be further illustrated in the next section.

The definitions of these functions make heavy use of the Boolean techniques described in prior sections. You may want to study the definitions of some of the functions to become better acquainted with actual applications of Boolean techniques.

(The algorithms underlying many of these functions were conceived by Robert A. Smith of STSC and are introduced in the publication, Boolean Functions and Techniques, 1975, Scientific Time Sharing Corporation.)

\[ \text{[WSID: BOOLEAN]} \]

\[ \nabla R \leftarrow P \text{ pPLUSRED B};T \]

[1] R Returns \(+/\) for each partition \(S\) of \(B\).

[2] R where \(P\) is the corresponding Boolean

[3] R partition vector whose 1s mark the first


[5] R

[6] R Works on Boolean \(B\) only. For numeric \(B\):

[7] R \(R \leftarrow (1 \oplus P) +/\ B\)

[8] R \(R \leftarrow R - 1 \downarrow 0,R\)

[9] R

[10] R Compress partition vec for just 1 bits and

[11] R leading bits:

[12] R \(T \leftarrow R \leftarrow (P \vee B)/P\)

[13] R Convert to indices:

[14] R \(R \leftarrow R/\downarrow T\)

[15] R Lengths of compressed partitions:

[16] R \(R \leftarrow (1 \downarrow R, \downarrow I O + T) - R\)

[17] R Deduct 1 for partitions with leading 0 bit:

[18] R \(R \leftarrow R - \sim P/B\)

\[ \nabla \]
Chapter 15  BOOLEAN TECHNIQUES

\[ \texttt{\WSID{boolean}} \]

\[
\forall R \leftarrow P \text{ pANDRED } B
\]
[1] \( \forall \text{ Returns } ^{\wedge} / S \text{ for each partition } S \text{ of } B, \)
[2] \( \forall \text{ where } P \text{ is the corresponding Boolean } \)
[3] \( \forall \text{ partition vector whose 1s mark the first } \)
[4] \( \forall \text{ element of each partition.} \)
[5] \( \forall \text{ Compress partition vec for just 0 bits and } \)
[6] \( \forall \text{ leading bits:} \)
[7] \( R \leftarrow (P \geq B)/P \)
[8] \( \forall \text{ Which partitions have no 0s beyond leading bit?} \)
[9] \( R \leftarrow R/1^0R \)
[10] \( \forall \ldots \text{and have a leading 1 bit:} \)
[11] \( R \leftarrow R \wedge P/B \)

\[
\forall R \leftarrow P \text{ pORRED } B
\]
[1] \( \forall \text{ Returns } ^{\lor} / S \text{ for each partition } S \text{ of } B, \)
[2] \( \forall \text{ where } P \text{ is the corresponding Boolean } \)
[3] \( \forall \text{ partition vector whose 1s mark the first } \)
[4] \( \forall \text{ element of each partition.} \)
[5] \( \forall \text{ Compress partition vec for just 1 bits and } \)
[6] \( \forall \text{ leading bits:} \)
[7] \( R \leftarrow (P \lor B)/P \)
[8] \( \forall \text{ Which partitions have no 1s beyond leading bit?} \)
[9] \( R \leftarrow R/1^0R \)
[10] \( \forall \text{ Leading 1 bit or any trailing 1s:} \)
[11] \( R \leftarrow R \leq P/B \)

\[
\forall R \leftarrow P \text{ pANDSCAN } B; T
\]
[1] \( \forall \text{ Returns } ^{\wedge}\backslash S \text{ for each partition } S \text{ of } B, \)
[2] \( \forall \text{ where } P \text{ is the corresponding Boolean } \)
[3] \( \forall \text{ partition vector whose 1s mark the first } \)
[4] \( \forall \text{ element of each partition. } \text{ Uses fact } \)
[5] \( \forall \text{ that } ^{\wedge}\backslash A \leftrightarrow ^{\lor}\backslash \sim A. \)
[6] \( \forall \text{ Consider just 0 bits and leading bits:} \)
[7] \( T \leftarrow P \geq B \)
[8] \( \forall \text{ All 1s except leading 1 bits (as 0s):} \)
[9] \( R \leftarrow \sim T/B \)
[10] \( \forall 1\text{-maps to } 1\text{-poles and expand:} \)
[11] \( R \leftarrow T \setminus \sim 1 \downarrow 0, R \)
[12] \( \forall 1\text{-poles to } 1\text{-maps and toggle:} \)
[13] \( R \leftarrow \sim \backslash R \)

\[
\forall
\]
Chapter 15 BOOLEAN TECHNIQUES

[WSID: BOOLEAN]

\( \nabla R \leftarrow P \) pORSCAN \( B;T \)

[1] \( \) Returns \( \forall \S \) for each partition \( S \) of \( B \),
[2] \( \) where \( P \) is the corresponding Boolean
[3] \( \) partition vector whose 1s mark the first
[4] \( \) element of each partition.
[5] \( \) Consider just 1 bits and leading bits:
[6] \( T \leftarrow P \lor V \)
[7] \( \) All 1s except leading 0 bits:
[8] \( R \leftarrow T/B \)
[9] \( \) 1-maps to 1-poles and expand:
[10] \( R \leftarrow T \lor \neg 1 \downarrow 0, R \)
[11] \( \) 1-poles to 1-maps:
[12] \( R \leftarrow \# \backslash R \)
\n
[WSID: BOOLEAN]

\( \nabla R \leftarrow P \) pLTSCAN \( B;T \)

[1] \( \) Returns \( <\S \) for each partition \( S \) of \( B \),
[2] \( \) where \( P \) is the corresponding Boolean
[3] \( \) partition vector whose 1s mark the first
[4] \( \) element of each partition.
[5] \( \) Consider just 1 bits and leading bits:
[6] \( T \leftarrow P \lor V \)
[7] \( \) All 1s except leading 0 bits:
[8] \( R \leftarrow T/B \)
[9] \( \) 1-maps to leading 1 bits and expand:
[10] \( R \leftarrow T \lor \neg 1 \downarrow 0, R \)
[11] \( \) Set leading 1 bits to 1:
[12] \( R \leftarrow R \lor P \backslash B \)
\n
[WSID: BOOLEAN]

\( \nabla R \leftarrow P \) pLESCLAN \( B;T \)

[1] \( \) Returns \( \leq \S \) for each partition \( S \) of \( B \),
[2] \( \) where \( P \) is the corresponding Boolean
[3] \( \) partition vector whose 1s mark the first
[4] \( \) element of each partition.
[5] \( \) Consider just 0 bits and leading bits:
[6] \( T \leftarrow P \geq B \)
[7] \( \) All 1s except leading 1 bits (as 0s):
[8] \( R \leftarrow T/B \)
[9] \( \) 1-maps to leading 1 bits and expand:
[10] \( R \leftarrow T \lor \neg 1 \downarrow 0, R \)
[11] \( \) Set leading 0 bits to 0; subtract other
[12] \( \) leading 1 bits:
[13] \( R \leftarrow R < B \geq P \)
\n
-241-
Chapter 15

BOOLEAN TECHNIQUES

[WSID: BOOLEAN]
\[ \forall R \leftarrow P \text{ pNESCAN } B \]
[1] \( R \) Returns \( \# \setminus S \) for each partition \( S \) of \( B \),
[2] \( R \) where \( P \) is the corresponding Boolean
[3] \( R \) partition vector whose \( 1 \)s mark the first
[4] \( R \) element of each partition.
[5] \( R \) 1-poles to 1-maps, shift right, mark overlap
[6] \( R \) leading bits:
[7] \( R \leftarrow P/\sim 1\setminus 0,\setminus B \)
[8] \( R \) 1-maps to 1-poles, expand, marking leading
[9] \( R \) bits to toggle:
[10] \( R \leftarrow P\backslash R\sim \setminus 1\setminus 0, R \)
[11] \( R \) Toggle selected leading bits, 1-poles to 1-maps:
[12] \( R \leftarrow \#\setminus B\# R \)
\[ \forall \]

[WSID: BOOLEAN]
\[ \forall R \leftarrow P \text{ pEQSCAN } B \]
[1] \( R \) Returns \( \# \setminus S \) for each partition \( S \) of \( B \),
[2] \( R \) where \( P \) is the corresponding Boolean
[3] \( R \) partition vector whose \( 1 \)s mark the first
[4] \( R \) element of each partition.
[5] \( R \) 0-poles to 0-maps, shift right, mark overlap
[6] \( R \) leading bits, toggle:
[7] \( R \leftarrow P/\sim 1\setminus 0,\setminus B \)
[8] \( R \) 1-maps to 1-poles, expand, marking leading
[9] \( R \) bits to toggle:
[10] \( R \leftarrow P\backslash R\sim \setminus 1\setminus 0, R \)
[11] \( R \) Toggle selected leading bits, 0-poles to 0-maps:
[12] \( R \leftarrow \#\setminus B\# R \)
\[ \forall \]

[WSID: BOOLEAN]
\[ \forall R \leftarrow P \text{ pNEMAP } B \]
[1] \( R \) Returns \( S\# \setminus 1\setminus 0, S \) for each partition \( S \) of \( B \),
[2] \( R \) where \( P \) is the corresponding Boolean
[3] \( R \) partition vector whose \( 1 \)s mark the first
[4] \( R \) element of each partition.
[5] \( R \) 1-maps to leading 1-poles:
[6] \( R \leftarrow B\# \setminus 1\setminus 0, B \)
[7] \( R \) Toggle leading bits which are not correct:
[8] \( R \leftarrow R\# P\# B\# R \)
\[ \forall \]
Chapter 15  BOOLEAN TECHNIQUES

\[ \text{\texttt{\textbf{V R\texttt{\texttt{-P}} pEQMAP B}}] \]

\[ \text{\texttt{\textbf{[WSID: BOOLEAN]}}] \]

\[ \text{\texttt{\textbf{1]} \ R \ Returns $S^{-1\downarrow 1, S}$ for each partition $S$ of $B$,} \]
\[ \text{\texttt{\textbf{2]} \ R \ where $P$ is the corresponding Boolean} \]
\[ \text{\texttt{\textbf{3]} \ R \ partition vector whose Is mark the first} \]
\[ \text{\texttt{\textbf{4]} \ R \ element of each partition.} \]
\[ \text{\texttt{\textbf{5]} \ R \ 0-maps to leading 0-poles:} \]
\[ \text{\texttt{\textbf{6]} \ R \orthy B^{-1\downarrow 1, B}} \]
\[ \text{\texttt{\textbf{7]} \ R \ Toggle leading bits which are not correct:} \]
\[ \text{\texttt{\textbf{8]} \ R \orthy R\#P\&B\#R} \]
\[ \text{\texttt{\texttt{-}}] \]

\[ \text{\texttt{\textbf{V R\texttt{\texttt{-P}} pGTMAP B}}] \]

\[ \text{\texttt{\textbf{[WSID: BOOLEAN]}}] \]

\[ \text{\texttt{\textbf{1]} \ R \ Returns $S^\downarrow 1\downarrow 0, S$ for each partition $S$ of $B$,} \]
\[ \text{\texttt{\textbf{2]} \ R \ where $P$ is the corresponding Boolean} \]
\[ \text{\texttt{\textbf{3]} \ R \ partition vector whose 1s mark the first} \]
\[ \text{\texttt{\textbf{4]} \ R \ element of each partition.} \]
\[ \text{\texttt{\textbf{5]} \ R \ 1-maps to first 0 bits:} \]
\[ \text{\texttt{\textbf{6]} \ R \orthy B^\downarrow 1\downarrow 0, B}} \]
\[ \text{\texttt{\textbf{7]} \ R \ Toggle leading bits which are not correct:} \]
\[ \text{\texttt{\textbf{8]} \ R \orthy R\#P\&B\#R} \]
\[ \text{\texttt{\texttt{-}}] \]

\[ \text{\texttt{\textbf{V R\texttt{\texttt{-P}} pGEMAP B}}] \]

\[ \text{\texttt{\textbf{[WSID: BOOLEAN]}}] \]

\[ \text{\texttt{\textbf{1]} \ R \ Returns $S\leq^{-1\downarrow 1, S}$ for each partition $S$ of $B$,} \]
\[ \text{\texttt{\textbf{2]} \ R \ where $P$ is the corresponding Boolean} \]
\[ \text{\texttt{\textbf{3]} \ R \ partition vector whose 1s mark the first} \]
\[ \text{\texttt{\textbf{4]} \ R \ element of each partition.} \]
\[ \text{\texttt{\textbf{5]} \ R \ 0-maps to first 0 bits:} \]
\[ \text{\texttt{\textbf{6]} \ R \orthy B^{-1\downarrow 1, B}} \]
\[ \text{\texttt{\textbf{7]} \ R \ Toggle leading bits which are not correct:} \]
\[ \text{\texttt{\textbf{8]} \ R \orthy R\#P\&B\#R} \]
\[ \text{\texttt{\texttt{-}}] \]

\[ \text{\texttt{\textbf{V R\texttt{\texttt{-P}} pORMAP B}}] \]

\[ \text{\texttt{\textbf{[WSID: BOOLEAN]}}] \]

\[ \text{\texttt{\textbf{1]} \ R \ Returns $S^\downarrow 1\downarrow 0, S$ for each partition $S$ of $B$,} \]
\[ \text{\texttt{\textbf{2]} \ R \ where $P$ is the corresponding Boolean} \]
\[ \text{\texttt{\textbf{3]} \ R \ partition vector whose 1s mark the first} \]
\[ \text{\texttt{\textbf{4]} \ R \ element of each partition.} \]
\[ \text{\texttt{\textbf{5]} \ R \ Extend 1-maps to right by 1:} \]
\[ \text{\texttt{\textbf{6]} \ R \orthy B^\downarrow 1\downarrow 0, B}} \]
\[ \text{\texttt{\textbf{7]} \ R \ Toggle leading bits which are not correct:} \]
\[ \text{\texttt{\textbf{8]} \ R \orthy R\#P\&B\#R} \]
\[ \text{\texttt{\texttt{-}}] \]
PROBLEM: Write a monadic function RELABEL which will locate and modify all of the line labels (and references to those labels) in any specified function (whose character vector visual representation is provided as the argument to RELABEL) such that the labels become L1, L2, L3,... The result of RELABEL is the modified visual representation.

TOPIC: An Illustration of Boolean Techniques

The visual representation of a function is a character vector which, when displayed, looks just like the display produced by the command VFNNAME[@]V. In order to be a character vector, not matrix, which displays on several lines, the visual representation necessarily contains "newline" (carriage return) characters. For example, suppose VR is the visual representation of the simple AVG function:

```
36
VR
V R<AVG V
[1] R<(+/V)+ρV
```

To get a more revealing view of VR and why it has the shape displayed above, we can replace all newline characters by the "@" symbol, and all blank characters by the "_" symbol. Suppose NL is a scalar newline character.
The visual representation of a function may be generated by using the OVR system function in APL*PLUS, or 1 DFD in SHARP APL, or the CRAVR function (and OCR) developed in the Workspace Design and Documentation chapter. The newline character may be generated by using DTCNL in APL*PLUS, DTC[2J (origin 1) in APL2 or by selecting the newline character from DAV (the index of the element depends upon the APL implementation; it is 157 in origin 1 for SHARP APL).

Let us define the header of the desired RELABEL function:

\[ \text{NEWVR+RELABEL VR} \]

We will find it convenient to view the visual representation as a partitioned array, where each line of the function is a partition. We will construct a partition vector whose 1s flag the first character on each line. Since every line is preceded by a newline, except the header line, and since the last character in the visual representation is a newline, we may construct the partition vector as follows:

\[ [1] \text{NL¬⌽1ΟVR=ŋTCNL} \]

(Use ſTC[2J in APL2 or DAV[157] in SHARP APL instead of ſTCNL.)

To solve this task, we must locate all identifiers, including labels, referred to in the visual representation. Identifiers are consecutive strings of alphanumeric characters whose first letter is alphabetic. However, we must be careful to ignore all such strings located within comments or within quotes, since such strings are probably words and not identifiers. Further, we cannot simply ignore everything beyond the first comment symbol (A) on each line. The first comment symbol may be located within quotes and so will not actually represent the beginning of a comment. (Note: many APL implementations now permit end-of-line comments, in addition to full-line comments, and so we cannot assume that the comment symbol will appear in a specific position on the line.)

Our first step is to find and ignore all characters within quotes (even quotes within comments). Then, the comment symbols not within quotes begin genuine comments, so we can use them to find and ignore all characters within comments. Finally, by ignoring all characters either within quotes or within comments, we can proceed with our job of finding identifiers.

As we saw in a prior section, \( =\backslash \) can be used to ignore characters within quotes, using an expression like:

\[ (=\text{CVEC#''''})/\text{CVEC} \]
However, this expression cannot be blindly applied to VR since a comment may contain an odd quote character (e.g. A YOU DON'T SAY!) which will cause mismatching of quote characters. Instead, we must perform the =\ (0-poles to 0-maps) on each partition (function line) of VR individually.

\[2\] \ NQ\leftarrow VR\#'''
\[3\] \ NCCON\leftarrow NL\ pEQSCAN\ NQ

NQ flags the non-quote characters. NCCON is a map vector of the characters which are not within quote pairs (i.e. character constants) within each function line. Note that the closing quotes of each quote pair are included in the map while the opening quote is not.

We can now use NCCON to help us locate all valid comment symbols.

\[4\] \ NC\leftarrow NCCON\leftarrow VR='a'

NC flags the non-comment characters (0s flag the valid comment symbols). For each line of the visual representation, we wish to propagate the 0 which flags the comment symbol in NC so that all following characters are flagged with 0s (to subsequently ignore them). For a single partition, we can use ^\/. For all partitions, we must use pANDSCAN:

\[5\] \ NCMT\leftarrow NL\ pANDSCAN\ NC

NCMT is a map vector of the characters which do not follow a valid comment symbol. Note that the comment symbols themselves are not included in the map.

It is now a simple matter to construct a map vector, PARSE, of the characters which are not within quote pairs and which do not follow a comment symbol within each function line:

\[6\] \ PARSE\leftarrow NCMT\leftarrow NCCON

Our next thrust is to look at the characters flagged by PARSE and to construct a pole vector whose poles (pairs of Is) flag the identifiers. First, flag the digit characters, the letters, the alphanumeric characters and blanks:

\[7\] \ NUM\leftarrow PARSE\leftarrow VR\epsilon'0123456789'
\[8\] \ ALP\leftarrow PARSE\leftarrow VR\epsilon'ABCDEFGHIJKLMNOPQRSTUVWXYZ\ldots xyz\ldots aXYZAabcd\ldots xyz\Delta'
\[9\] \ AN\leftarrow NUM\leftarrow ALP
\[10\] \ BL\leftarrow PARSE\leftarrow VR=''

Then, construct a pole vector of the alphanumeric maps.

\[11\] \ PAN\leftarrow AN\leftarrow 1\downarrow 0,AN

Some of these pole pairs flag identifiers (e.g. COUNT or AMT85 or J) while others do not (e.g. 58 or 1E6). We must "turn off" the pole
pairs whose first pole does not correspond to an alphabetic character. The technique used to do this follows:

\[ T \rightarrow \text{PAN/ALP} \]
\[ \text{PID} \rightarrow \text{PAN} \vee \neg 1 \bigvee T \]

Notice that the compression (\( / \)) places the pole values next to each other, the map operation extends the value of the first pole (1 or 0) into the second pole, and the expansion (\( \\backslash \)) replaces the poles (some 0s now) into their original positions. PID is a pole vector which flags identifiers.

Since some identifiers begin with the '0' symbol (e.g. 010 or 0PP or 0EX), we must adjust the poles in PID to include the '0'.

\[ T \rightarrow 1 \bigvee \text{PID} \]
\[ T \rightarrow T \backslash '0' = T / VR \]
\[ \text{PID} \rightarrow T \vee \text{PID} \bigvee 1 \bigvee T \]

Now that we have located all identifiers within the visual representation, we must locate the labels. Labels are identifiers which immediately follow the bracketed function line number (ignoring spaces) and which immediately precede a colon (\( : \)).

Let us find the line numbers. Construct a pole vector of the numeric digit maps.

\[ \text{PNUM} \rightarrow \text{NUM} \bigvee 1 \bigvee 0 \bigvee \text{NUM} \]

If the first pole in the PNUM pole pairs is exactly 2 characters after a newline character, that pair of poles identifies a line number. Flag the character following the "]" character after the line number at the beginning of each line.

\[ \text{START} \rightarrow 1 \bigvee \text{PNUM} \bigvee 1 \bigvee \text{PNUM} \bigvee 1 \bigvee \text{NL} \]

Notice that / and \( \backslash \) are again used to place the pole values next to each other. In this case, the rotate operation changes 1 0 poles to 0 1 poles so that the last pole of the numeric pole vector (corresponding to the "\]") is flagged.

The next step is to move the bits in START to the right so that they correspond to the first nonblank after the bracketed line number. Construct a pole vector of the blank maps.

\[ \text{PBL} \rightarrow \text{BL} \bigvee 1 \bigvee 0 \bigvee \text{BL} \]

Flag the first nonblank character in each line:

\[ \text{START} \rightarrow (\text{START} \backslash \text{BL}) \bigvee \text{PBL} \bigvee 1 \bigvee \text{PBL} / \text{START} \]

The (\( \text{START} \backslash \text{BL} \)) term is included for lines which do not have any blanks between the bracketed line number and the next nonblank character.
By matching up START and PID, we can construct a pole vector of identifiers which begin with the first nonblank character of a function line:

\[
\begin{align*}
[21] & \quad T \leftarrow \text{PID/START} \\
[22] & \quad \text{PSID} \leftarrow \text{PID} \setminus \text{T} \setminus \text{1} \setminus \text{T}
\end{align*}
\]

Construct a pole vector of labels (i.e. identifiers at the beginnings of function lines which are followed by a colon):

\[
\begin{align*}
[23] & \quad T \leftarrow ':=' \text{=PSID/VR} \\
[24] & \quad \text{PLAB} \leftarrow \text{PSID} \setminus \text{T} \setminus \text{1} \setminus \text{T}
\end{align*}
\]

Using the pole vector, determine the starting and ending (plus 1) indices of the identifiers in the function (as a 2 column matrix, one row per identifier):

\[
\begin{align*}
[25] & \quad \text{IND} \leftarrow \text{PID} \setminus \text{1} \setminus \text{pPID} \\
[26] & \quad \text{NID} \leftarrow (\text{pIND}) \div 2 \\
[27] & \quad \text{IND} \leftarrow (\text{NID}, 2) \setminus \text{pIND}
\end{align*}
\]

Determine the length of each identifier name:

\[
\begin{align*}
[28] & \quad \text{IDSTART} \leftarrow \text{IND} \setminus [;010] \\
[29] & \quad \text{IDLEN} \leftarrow \text{IND} \setminus [;1+010] \setminus \text{IDSTART}
\end{align*}
\]

From here on, Boolean techniques are not required. The techniques used are discussed in other chapters, most notably the Positioning Character Data chapter and the Sorting and Searching chapter.

The RELABEL function is presented below in its entirety. The function uses the Boolean techniques described above. However, it has been modified in a number of ways. The logical partition functions (e.g. pANDSCAN) have been replaced by the equivalent logic so that RELABEL does not require their presence. The variables have been localized. RELABEL also has a left argument: a character matrix or vector (blank-delimited) of the names of the labels which are not to be renamed. Provide an empty character vector if all labels are to be renamed. For example, to relabel the function MODEL:

- **DDEF** '
  `' RELABEL D\text{OVR} 'MODEL'
' (on APL*PLUS)
- **3 FD** `' RELABEL 1 D\text{FD} 'MODEL'
' (on SHARP APL)
- **DFX VR\text{A}CR** `' RELABEL CRA\text{VR} D\text{CR} 'MODEL'
' (otherwise)
Chapter 15

BOOLEAN TECHNIQUES

\[ \text{[WSID: FNIDS]} \]

\[ \mathbf{\text{R} \leftarrow \text{KEEP RELABEL VR; ALP; AN; BL; COLS; FOUND; IDLEN; IDS;}} \]
\[ \text{IDSTART; IND; KLABS; KLEN; KSTART; LABS; NB; NC; NCON; NCMT;}} \]
\[ \text{NEW; NID; NKEEP; NL; NLAB; NLEN; NQ; NSTART; NUM; PAN; PARSE; PBL}} \]
\[ \text{; PID; PLAB; PNUM; PSID; S; START; T; DIO} \]

[1] \( \mathbf{R} \) modifies the vector representation (VR) of a function such that its labels and references to same are changed to L1, L2, L3,... for all labels but those specified in KEEP, a character matrix or vector (blank delimited) of label names.

[2] \( \mathbf{R} \) requires subfunction: CMIOTA.

[3] \( \mathbf{DIO} \leftarrow 0 \)

[4] \( \mathbf{NL} \leftarrow 1 \) or VR = DTCNL # APL*PLUS

[5] \( \mathbf{NL} \leftarrow 0 \) or VR = DTC[1] # APL2

[6] \( \mathbf{NL} \leftarrow 1 \) or VR = DAV[156] # SHARP APL

[7] \( \mathbf{NQ} \leftarrow \mathbf{VR} \leftarrow ' ' ' '

[8] \( \mathbf{NC} \leftarrow \mathbf{NCON} \leftarrow \mathbf{VR} = ' ' '

[9] \( \mathbf{NC} \leftarrow \mathbf{NCMT} \leftarrow \mathbf{NCON} \leftarrow 0 \)

[10] \( \mathbf{S} \leftarrow \mathbf{NL} \leftarrow 0 \)

[11] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[12] \( \mathbf{AN} \leftarrow \mathbf{NUM} \leftarrow \mathbf{ALP} \)

[13] \( \mathbf{AN} \leftarrow \mathbf{PAN} \leftarrow 0 \)

[14] \( \mathbf{AN} \leftarrow \mathbf{PID} \leftarrow 0 \)

[15] \( \mathbf{AN} \leftarrow \mathbf{ALP} \leftarrow 0 \)

[16] \( \mathbf{AN} \leftarrow \mathbf{T} \leftarrow 0 \)

[17] \( \mathbf{T} \leftarrow \mathbf{T} \leftarrow 0 \)

[18] \( \mathbf{T} \leftarrow \mathbf{T} \leftarrow \mathbf{NQ} \leftarrow 0 \)

[19] \( \mathbf{S} \leftarrow \mathbf{NL} \leftarrow 0 \)

[20] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[21] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[22] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[23] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[24] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[25] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[26] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[27] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[28] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[29] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[30] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[31] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[32] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[33] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[34] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[35] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[36] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[37] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[38] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[39] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[40] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[41] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[42] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[43] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[44] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[45] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[46] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)

[47] \( \mathbf{S} \leftarrow \mathbf{T} \leftarrow 0 \)
\[ \text{\textbackslash{}RELABEL (continued)} \]

[48] A Shift leading poles of Dnames to include \( \Box \):
[49] \[ \text{PID} \leftarrow \text{T} \lor \text{PID} > \text{T} \]
[50] \[ \text{T} \leftarrow 0 \]
[51] A Flag char following ] after line no.:
[52] \[ \text{START} \leftarrow \text{~T} \lor \text{PNUM} \lor \text{~T} \lor \text{PNUM} \lor \text{~T} \]
[53] \[ \text{NL} \leftarrow \text{PNUM} \leftarrow 0 \]
[54] A Pole vec of contiguous blanks:
[55] \[ \text{PBL} \leftarrow \text{~T} \lor 0, \text{BL} \]
[56] A Flag 1st nonblank char in each line:
[57] \[ \text{START} \leftarrow (\text{START} > \text{BL}) \lor \text{PBL} \lor \text{~T} \lor \text{PBL} \lor \text{START} \]
[58] \[ \text{BL} \leftarrow \text{PBL} \leftarrow 0 \]
[59] A Pole vec of identifiers at start of line:
[60] \[ \text{PSID} \leftarrow \text{PID} \lor \text{~T} \lor \text{PID} \lor \text{START} \]
[61] \[ \text{START} \leftarrow 0 \]
[62] A Pole vec of labels:
[63] \[ \text{PLAB} \leftarrow \text{PSID} \lor \text{~T} \lor \text{PSID} \lor \text{~T} \lor \text{PLAB} \]
[64] \[ \text{PSID} \leftarrow 0 \]
[65] A Start and end (+1) indices of identifiers:
[66] \[ \text{IND} \leftarrow \text{PID} \lor \text{~T} \lor \text{PID} \lor \text{IND} \]
[67] \[ \text{NID} \leftarrow \text{IND} + 2 \]
[68] \[ \text{IND} \leftarrow \text{IND} \lor 2 \lor \text{IND} \]
[69] A Start indices of identifiers:
[70] \[ \text{INDSTART} \leftarrow \text{IND} \lor 0 \]
[71] \[ \text{IDSTART} \leftarrow \text{IND} \lor 0 \]
[72] \[ \text{IDLEN} \leftarrow \text{IND} \lor 1 \lor \text{IDLEN} \]
[73] \[ \text{IND} \leftarrow 0 \]
[74] A Map vec of nonblanks in labels to keep:
[75] \[ \text{NB} \leftarrow \text{~T} \lor \text{KEEP} \lor \text{~T} \lor \text{KEEP} \]
[76] A Start indices in KEEP of identifiers:
[77] \[ \text{NKEEP} \leftarrow \text{IND} \lor \text{~T} \lor \text{IND} \lor \text{~T} \lor \text{NKEEP} \]
[78] \[ \text{KSTART} \leftarrow \text{IND} \lor 0 \]
[79] \[ \text{KLEN} \leftarrow \text{IND} \lor 1 \lor \text{KLEN} \]
[80] \[ \text{KSTART} \leftarrow \text{IND} \lor 0 \]
[81] \[ \text{KLEN} \leftarrow \text{IND} \lor 1 \lor \text{KLEN} \]
[82] \[ \text{COLS} \leftarrow \text{KLEN} \lor 1 \lor \text{IDLEN} \]
[83] A Raveled blank matrix of identifier names:
[84] \[ \text{IDS} \leftarrow \text{IND} \lor \text{COLS} \lor \text{IND} \lor \text{IDS} \]
[85] A Fill them in (T\text{\textbackslash{}MONIOTA IDLEN}):
[86] \[ \text{T} \leftarrow \text{T} \lor \text{T} \lor \text{IDLEN} \lor 1 \lor 0 \lor \text{IDLEN} \]
[87] \[ \text{IDS} \leftarrow \text{IDS} \lor \text{IDLEN} \lor \text{COLS} \lor \text{IND} \lor \text{IDS} \]
[88] A Reshape to mat of identifiers:
[89] \[ \text{IDS} \leftarrow \text{IND} \lor \text{COLS} \lor \text{IND} \lor \text{IDS} \]
[90] A Mat of label names:
[91] \[ \text{LABS} \leftarrow ((\text{IND} \lor 2) \lor \text{PID} \lor \text{PLAB} \lor 0) \lor \text{LABS} \]
[92] \[ \text{PID} \leftarrow \text{PLAB} \leftarrow 0 \]
[93] A Raveled blank matrix of labels to keep:
[94] \[ \text{KLABS} \leftarrow \text{NKEEP} \lor \text{COLS} \lor \text{KLABS} \]
[95] A Fill in (T\text{\textbackslash{}MONIOTA KLEN}):
[96] \[ \text{T} \leftarrow \text{T} \lor \text{T} \lor \text{KLEN} \lor 1 \lor 0 \lor \text{KLEN} \]
[97] \[ \text{KLABS} \leftarrow \text{KLABS} \lor \text{KLEN} \lor \text{COLS} \lor \text{NKEEP} \lor \text{KLABS} \]
[98] A Reshape to mat of labels to keep:
[99] \[ \text{KLABS} \leftarrow \text{NKEEP} \lor \text{COLS} \lor \text{KLABS} \]
Chapter 15  BOOLEAN TECHNIQUES

\[ \text{RELABEL (continued)} \]

100. A Squeeze out labels in KEEP:

101. LABS*(NKEEP=KLABS CMIOTA LABS)/LABS

102. A Row indices in LABS where rows of IDS are found:

103. IND*LABS CMIOTA IDS

104. A No. of labels:

105. NLAB*1pLABS

106. A Flag identifiers which are labels:

107. FOUND*IND<NLAB

108. A Start indices of label identifiers:

109. IDSTART*FOUND/IDSTART

110. A Lengths of same:

111. IDLEN*FOUND/IDLEN

112. A Indices into LABS of same:

113. IND*FOUND/IND

114. A Building new labels: 'L1L2L3...':

115. NEW*’',#1+1NLAB

116. A Start indices in NEW of new labels:

117. NSTART*(NEW=’ ‘)/1pNEW

118. NEW[NSTART]*’L’

119. A Lengths of new labels:

120. NLEN*(1NSTART,pNEW)-NSTART

121. A Lengths replicated for all label identifiers:

122. NLEN*NLLEN[IND]

123. A Start indices replicated as well:

124. NSTART*NSTART[IND]

125. A Initialize replication vec:

126. R*OVRp1

127. A Use 0s to squeeze out old identifiers

128. A (T*MONIOTA IDLEN):

129. T*T+1pT+IDLEN/-IDLEN

130. R[T+IDLEN/IDSTART]*0

131. A Insert lengths to expand for new identifiers:

132. R[IDSTART]*NLEN

133. A Squeeze/expand vis rep as needed:

134. R*R/VR

135. A Adjust for new lengths:

136. IDSTART+IDSTART+-1\NLEN-IDLEN

137. A Insert new labels (T*MONIOTA NLEN):

138. T*T+1pT+NLEN/-1\NLEN

139. R[T+NLEN/IDSTART]*NEW[T+NLEN/NSTART]
PROBLEMS: (Solutions on pages 449 to 458)

1. What expression will tell you whether all of the elements of the numeric vector NVEC are integers?

2. Delete the trailing blanks from the character vector CVEC.

3. Left justify the rows of the character matrix NAMES (i.e. shift each row left until its first character is a nonblank).

4. How many numbers (set of contiguous digit characters) are there in the character vector INPUT?

5. Write the function TIMEDEFINE as described in the Computer Efficiency Considerations chapter.

6. Write one or more of the functions IDENTIFY, LOCALIZE and UNCOMMENT as described in the Workspace Design and Documentation chapter.
IRREGULAR ARRAYS

APL derives much of its power from its conciseness and consistency. Unfortunately, the real world is not nearly so concise and consistent. While APL sees the world as a set of rectangular arrays of data, the world is nonrectangular by nature.

In this chapter, we deal with irregular (nonrectangular) arrays. We will present a typical problem which involves irregular arrays and will attempt to perform a variety of tasks on these arrays. We will examine alternative methods available in APL for performing these tasks on the irregular arrays.

As an illustration of irregular arrays, suppose you wish to keep track of customer information for a business you operate. The following table shows some of the information.

<table>
<thead>
<tr>
<th>CUSTOMER INFORMATION</th>
<th>UNPAID BILLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>TERR NO.</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>3</td>
</tr>
<tr>
<td>74</td>
<td>5</td>
</tr>
<tr>
<td>89</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How would you store this information in APL variables?

The ID numbers can be assigned as a vector with one element per customer:

```apl
ID←3 65 74 89...
```
Likewise for territory numbers:

\[ \text{TERR} \leftarrow 5 \ 3 \ 5 \ 2 \ldots \]

The customer names may be stored as a character matrix with one row per customer and as many columns as the widest customer name (or an arbitrary maximum number of columns). Assuming 500 customers and a width of 25 character columns,

\[ \text{NAME} \leftarrow 500 \ 25 \left(25 \leftarrow 'ACME CORPORATION'), (25 \leftarrow 'FASTENERS INC.') \ldots \]

While this approach to storing the customer names in a matrix is tolerable, it has some disadvantages. First, if a customer's name is longer than the allowable width (here 25), it must be abbreviated or truncated. Second, short customer names must be padded with blanks to the maximum width, implying a waste of storage. Third, when extracting a customer's name from this character matrix, as for a form letter, the trailing blanks may need to be deleted, requiring more complex programming and more processing time.

If these disadvantages are not major, you will do well to store the names in a character matrix and tolerate the disadvantages. If the disadvantages are great enough to be unworkable, you must use some other method to work with this irregular information.

Conceptually, the customer names define an irregular array, a "vector" of character vectors. Each "element" of the "vector" is itself the character vector name for a single customer. In this chapter, we will refer to this type of array as a "nest of character vectors" or simply a "character nest". We will refer to the character vector "elements" as "items".

Likewise, the unpaid bills information cannot be fit neatly into a conventional APL array. If we assign the values to a 3 column numeric matrix, there will not be one row per customer (as there are in a character matrix of customer names) and so we must maintain additional information which tells us which rows belong to which customer.

Conceptually, the unpaid bills information defines an irregular array, a "vector" of 3 column numeric matrices. Each "element" of the "vector" is itself the 3 column numeric matrix for a single customer. In this chapter we will refer to this type of array as a "nest of numeric matrices" or simply a "matrix nest". We will refer to the numeric matrix "elements" as "items".

In this chapter, we will present 4 different approaches to the problem of working with irregular arrays:

1. APL2 nested arrays
2. APL*PLUS nested arrays
3. SHARP APL nested arrays
4. Conventional APL arrays
You will notice that the APL2 and APL*PLUS implementations of nested arrays are similar. The primary differences are in the function symbols chosen for the particular nested array operations.

**PROBLEM:** Construct a character nest of customer names and a matrix nest of unpaid bills information.

**TOPIC:** Constructing Irregular Arrays

In APL2, APL*PLUS and SHARP APL, character nests and matrix nests may be stored as nested arrays. In particular, they are stored as vectors of items, where the items are character vectors (character nests) or 3 column numeric matrices (matrix nests). The conventions for constructing these arrays are different for each of the different implementations of APL.

In APL2 and APL*PLUS, the arrays may be constructed by "vector notation" or "strand notation":

```
NAME←'ACME CORPORATION' 'FASTENERS INC.' ...
UBILLS←(3 3p31586 372 58625 41086 395 40615 42586 407 10000)
     (1 3p42086 405 80216) (0 3p0) ...
```

Alternately, the arrays may be initialized to have the correct number of items, after which the items are individually index assigned (assume 500 customers):

```
NAME←UBILL←500p0
NAME[1]←'ACME CORPORATION'
NAME[2]←'FASTENERS INC.'
...UBILLS[3]←0 3p0
```

The monadic enclose (⊂) function converts its right argument into a "nested scalar". In APL2, a tidier notation may be employed to perform the index assignment:
In SHARP APL, the arrays may be constructed by repeatedly applying the link (⊃) function:

\[
\text{NAME} \leftarrow \text{ACME CORPORATION} \quad \text{⊃} \quad \text{FASTENERS INC.} \quad \text{UBILLS} \leftarrow (3 \ 3 \ 31586 \ 372 \ 58625 \ 41086 \ 395 \ 40615 \ 42586 \ 407 \\
10000 \\
2 \quad (3 \ 3 \ 42086 \ 405 \ 80216 \\
3 \quad (0 \ 3 \ 0) \quad \text{⊃} \quad \ldots
\]

Alternately, the arrays may be initialized to have the correct number of nested items, after which the items are individually index assigned:

\[
\text{NAME} \leftarrow \text{UBILLS} \leftarrow 500 \ 0 \\
\text{NAME}[1] \leftarrow \text{ACME CORPORATION} \\
\text{NAME}[2] \leftarrow \text{FASTENERS INC.} \\
: \\
\text{UBILLS}[1] \leftarrow 3 \ 3 \ 31586 \ 372 \ 58625 \ 41086 \ 395 \ 40615 \ 42586 \ 407 \\
10000 \\
\text{UBILLS}[2] \leftarrow 1 \ 3 \ 42086 \ 405 \ 80216 \\
\text{UBILLS}[3] \leftarrow 0 \ 3 \ 0 \\
:
\]

Notice that the less than (⊂) symbol is used to perform the enclose function.

If you already have your customer names in the form of a character matrix (CMAT) with one row per name, you may convert the matrix directly into a character nest by one of the following:

\[
\text{NAME} \leftarrow \text{CMAT}[2] \\
\text{NAME} \leftarrow \text{CMAT}[2] \\
\text{NAME} \leftarrow \text{CMAT}[2] \\
\]

Each of the items of the resulting nest will be a character vector of the same length, namely the number of columns in the character matrix (CMAT). To delete the trailing blanks from each character vector item of a character nest, you must do the following:

\[
\text{NAME} \leftarrow \text{DTB} \quad \text{NAME} \\
\text{NAME} \leftarrow (+/\backslash\ 'Φ\text{CMAT})p \quad \text{NAME} \\
\]

where DTB (delete trailing blanks) is a user-defined monadic function which deletes the trailing blanks from its character vector right argument. For example:
When generating reports which include the customer names, you may need to convert the character nest back into a character matrix with a specified number of columns. Each name must be left-justified in a single row of the matrix. Suppose you want a 25 column matrix. You may do this as follows:

```
CMAT←[2]25¨NAME  (APL2)
or: CMAT←25[2]≠[2]NAME  (APL2)
CMAT←[1.5]25¨NAME (APL*PLUS)
CMAT←25◊>NAME     (SHARP APL)
```

The second APL2 algorithm is more efficient than the first but will require more workspace storage than the first if any name is longer than 25 characters. If much longer, a WS FULL may occur.

PROBLEM: How can you work with irregular arrays if your implementation of APL does not support nested arrays?

TOPIC: Emulating Nested Arrays on Non-Nested Systems

Skip this section if your implementation of APL has nested arrays.

To solve the tasks above using conventional APL, we must devise a scheme wherein a character nest or a matrix nest may be stored as one or more conventional APL arrays. We will deal first with the character nest problem.

One approach is to catenate the names together, preceding each name by a delimiter character. For example:

```
NAME←'@ACME CORPORATION@FASTENERS INC.@KLINGLEY...'
```

(In SHARP APL, NAME can be converted into the nested array by the expression: `∼18< NAME`).

Unfortunately, this scheme does not allow an efficient means to directly access the name for a specified customer since the array must first be searched for the occurrences of the delimiter character. For example, the name of the fifth customer may be extracted with the expression:

```
NAME←'@ACME CORPORATION@FASTENERS INC.@KLINGLEY...'
NAME[5] (SHARP APL)
```
which works but is inefficient.

If the delimiter character is not going to be used to locate the substrings of the character nest, you may omit them altogether. Let us concatenate the names together as a single character vector:

\[ \text{NAME} \leftarrow 'ACME CORPORATION FASTENERS INC. KLINGLEY...' \]

We need a second array to tell us where each name starts. One possibility is a Boolean "partition" vector which has one element per element of NAME and whose 1s mark the corresponding first elements of each name:

\[ \text{NPV} \leftarrow 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 \ldots \]

Together, these two arrays contain all the information you need to determine the name for a specified customer. For example, the name of the fifth customer may be extracted with the expression:

\[ (5=\oplus\text{NPV})/\text{NAME} \]

(In SHARP APL, NPV and NAME can be converted into the nested array by the expression: \( \text{NPV} \, \up{10} < \, \text{NAME} \)).

Unfortunately, the partition vector approach can be awkward to work with and has an inherent flaw: it will not handle an empty name. For example, if one of the customer names is unknown, you would naturally store it as an empty character vector. However, the partition vector requires each name to have at least one element (for the 1 in the partition vector).

Instead of constructing a Boolean partition vector to keep track of where the names start and end in the vector NAME, construct a vector of the lengths of each name (including 0s for empty names).

\[ \text{NLEN} \leftarrow 16 \, 14 \, 21 \, 9 \, 15 \, 0 \, 11 \ldots \]

The sum of these elements should be the same as the number of elements in NAME, i.e. \( (\up{+/}\text{NLEN}) = \up{\#}\text{NAME} \). Using NLEN, you may extract the name of the fifth customer with the expression:

\[ \text{NAME}[(-1 \, \up{10}, +\text{\up{\text{NLEN}}})[5] + \text{\up{\text{NLEN}}}[5]] \]

This expression works properly even for empty customer names (i.e. for elements of NLEN which are 0).

Note that the \( (-1 \, \up{10}, +\text{\up{\text{NLEN}}}) \) portion of the expression is merely computing the starting indices (i.e. the index before the first character) of the names. To improve the efficiency of the name extraction process, you may perform this operation once:

\[ \text{NSTART} \leftarrow (-1 \, \up{10}, +\text{\text{NLEN}}) \]
after which the extraction process (for customer 5) becomes:

\[
\text{NAME}[\text{NSTART}[5]+\text{NLEN}[5]]
\]

Unfortunately, the price you pay for the greater efficiency is greater complexity. You must maintain 3 arrays (NAME, NLEN, NSTART) when working with customer names. Any utility functions you write to work with this character nest (e.g. to emulate the capabilities illustrated above on nested array systems) must cope with at least these 3 arguments.

Utility function design will be greatly simplified if we can weld these three arrays into a single array. Unfortunately, numbers (NLEN and NSTART) do not cohabitate well with characters (NAME) in the arrays of conventional APL systems. (Note: the APL2 and APL*PLUS nested array implementations allow "heterogeneous" arrays which contain both character and numeric elements.) You must first convert them to a common datatype. Suppose we convert the character vector NAME into a numeric vector (e.g. DAV\text{\textbackslash NAME}) and then concatenate the three arrays together, along with the number of customers:

\[
\text{CNEST} \leftarrow (\rho \text{NLEN}), \text{NLEN}, \text{START}, \text{DAV}\backslash \text{NAME}
\]

This array is a single object which contains all of the customer name information. Since it is a single array, you may design utility functions which take CNEST as one argument and the other parameters of the problem as the other argument. For example, you can write a function EXTRACT whose left argument is this "character nest" array and whose right argument is the index of the nest for which the character vector name is to be returned:

\[
\text{CNEST EXTRACT 3}
\]

KLINGLEY & SONS, INC.

The definition of the EXTRACT function is straightforward:

\[
\begin{align*}
\triangledown & \text{R} \leftarrow \text{CNEST EXTRACT I;N} \\
[1] & \text{N} \leftarrow 1 \downarrow \text{CNEST} \\
[2] & \text{R} \leftarrow \text{DAV}[\text{CNEST}((\text{CNEST}[1+N+I]+1+N+N)+\downarrow \text{CNEST}[1+I])] \\
\end{align*}
\]

Unfortunately, this definition of a character nest is probably no improvement over padded character matrices. On an APL system in which small integers (i.e. 1 to 256) are stored with 4 bytes per element, each name in this character nest requires \(8+4\times C\) bytes, where \(C\) is the number of characters in the name. Since concern for storage is one of the reasons for exploring character nests, a better solution is needed.

A big improvement can be made if you can manage to squeeze more than one character into an integer. Since integers are stored in 2 bytes or 4 bytes, depending upon your implementation of APL, it is possible to translate 2 or 4 characters into a single integer. For example, in APL implementations which store integers in 4 bytes, the range of
Chapter 16

IRREGULAR ARRAYS

integers is \(-2,147,483,648\) to \(2,147,483,647\). Beyond this range, integers are stored in 8 bytes. The procedure for converting 4 characters into one integer is to convert each character into an integer from 1 to 256 (by finding its index in the atomic vector, DAV) and then pack these 4 numbers into a number between \(-2,147,483,648\) and \(2,147,483,647\). Given a 4 element character vector CVEC4, a single integer may be produced as follows:

\[
\text{OIO} \gets 0 \\
\text{INT} \gets \text{f} -2147483648 + 256 \text{AV} \cdot \text{CVEC4}
\]

To convert the integer back to a 4 element character vector:

\[
\text{OIO} \gets 0 \\
\text{CVEC4} \gets \text{AV}[(4 \cdot 256) \cdot \text{INT} + 2147483648]
\]

Using such a packing and unpacking algorithm, each name in the character nest will require only \(8 + 4 \times \text{C} + 4\) bytes, where \(\text{C}\) is the number of characters in the name. This is a big improvement in storage but comes at the expense of processing efficiency. The packing and unpacking takes time.

Some APL implementations provide primitive functions for converting between datatypes, e.g. for converting one 4-byte integer into 4 1-byte characters or vice versa. Such functions are extremely efficient since no modification is made to the internal representation of the data, only to the "header" of the variable which indicates its shape and datatype. Such implementations are ideal candidates for manipulations of character nests by conventional means (i.e. not nested arrays).

APL*PLUS PC is one such implementation. Integers are stored in 2 bytes (\(-32768\) to \(32767\)) per element. The system function DDR (data representation) converts between datatypes. For example, the expression

\[
\text{NVEC} \cdot 163 \cdot \text{DDR CVEC}
\]

converts an \(N\) element character vector to an \(N\div2\) element integer vector. The expression

\[
\text{CVEC} \cdot 82 \cdot \text{DDR NVEC}
\]

converts an \(M\) element integer vector to an \(M\times2\) element character vector.

Here is one possible design for storing character nests on an APL*PLUS PC system (origin 1):
Chapter 16  IRREGULAR ARRAYS

<table>
<thead>
<tr>
<th>CNEST[1]</th>
<th>number of character vector substrings (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNEST[2 to N+1]</td>
<td>index ([I0=0] into this vector of the starts of the substrings (S)</td>
</tr>
<tr>
<td>CNEST[S]</td>
<td>length of this character substring (L)</td>
</tr>
<tr>
<td>CNEST[S+1 to S+[L÷2]]</td>
<td>integer representation (163 DDR) of this character substring, padding with 1 space if necessary to produce an even number of characters</td>
</tr>
</tbody>
</table>

For example, the character nest which stores the 3 character substrings 'TREE', 'DOG' and 'ELEPHANT' is:

```
CNEST←3 4 7 10 4 21076 17733 20292 8263 8 19525 20549 16712 21582
```

Note that:

```
163 DDR 'TREEDOG ELEPHANT'
21076 17733 20292 8263 19525 20549 16712 21582
```

Given this design, you may write utility functions which will allow you to work with character nests with the same (or greater) ease and efficiency as when working with nested array systems. For example, to construct the character nest from a character matrix of customer names:

```
NAME←CNEST CMAT
```

(The CNEST function and all other character nest utility functions suggested below are written for the APL*PLUS PC implementation of APL as exercises at the end of the chapter.)

To convert a character nest back into a character matrix with a specified number of columns (say 25), use the CNAM function:

```
CMAT←25 CNAM NAME
```

We will deal now with the matrix nest problem. In one regard, the problem is more difficult to work with than the character nest problem and in another regard it is simpler. It is more difficult because the items are matrices instead of vectors and it is simpler because the values are numeric, not character, and so do not need to be converted into numbers.

View the matrices as vectors (i.e. ravel them). The number of elements in each vector is a multiple of 3 since the matrices have 3 columns. Viewed in this way, the nest of numeric matrices becomes a nest of numeric vectors, which we will call a "numeric nest".
One possible design for storing numeric nests is patterned after the design proposed above for character nests (origin 1):

\[
\begin{align*}
\text{NNEST}[1] & \quad \text{number of numeric vector segments (N)} \\
\text{NNEST}[2 \text{ to } N+1] & \quad \text{index (DI0=0) into this vector of the starts of the segments (S)} \\
\text{NNEST}[S] & \quad \text{length of this numeric segment (L)} \\
\text{NNEST}[S+1 \text{ to } S+L] & \quad \text{numeric segment}
\end{align*}
\]

For example, the numeric nest which stores the 3 segments \((30 \ 17 \ 15), (25)\) and \((82 \ 93 \ 95 \ 98)\) is:

\[
\text{NNEST} \leftarrow 3 \ 4 \ 8 \ 10 \ 3 \ 30 \ 17 \ 15 \ 1 \ 25 \ 4 \ 82 \ 93 \ 95 \ 98
\]

Given this design, you may write utility functions for working with numeric nests. For example, to construct the numeric nest of unpaid bills information:

- \(A\) Number of elements per customer/segment:
  \(\text{LEN} \leftarrow 3 \times 3 \ 1 \ 0 \ 2\ldots\)

- \(A\) Values:
  \(\text{VAL} \leftarrow 31586 \ 372 \ 58625 \ 41086 \ 395 \ 40615 \ 42586 \ 407 \ 10000 \ 42086\ldots\)

- \(A\) Construct numeric nest:
  \(\text{UBILLS} \leftarrow \text{LEN} \ \text{NNEST} \ \text{VAL}\)

(The NNEST function and all other numeric nest utility functions suggested below are written as exercises at the end of the chapter.)

PROBLEM: Add to your customer information database a new customer, number 33, TOP DOG LTD., territory 4, which has no unpaid bills.

TOPIC: Catenating to Irregular Arrays

Use catenation to update the two regular arrays:

\[
\begin{align*}
\text{ID} \leftarrow \text{ID}, 33 \\
\text{TERR} \leftarrow \text{TERR}, 4
\end{align*}
\]
Likewise, use catenate to update the two irregular arrays if they are stored as nested arrays:

```
NAME NAME, `<TOP DOG LTD.' (APL2, APL*PLUS)
NAME NAME, `<TOP DOG LTD.' (SHARP APL)
UBILLS UBILLS, `<0 3p0 (APL2, APL*PLUS)
UBILLS UBILLS, `<0 3p0 (SHARP APL)
```

The enclose function (< or <) must be used to construct a nested scalar before it is catenated to the nested array. The enclose function is not needed when catenating more than one customer since vector notation (or the SHARP APL link function) performs the enclosing. For example:

```
NAME NAME, 'TOP DOG LTD.' (APL2, APL*PLUS)
NAME NAME, 'BOTTOM CAT CORP.' (SHARP APL)
```

Using conventional APL, you may write a function CNCAT which will catenate a character nest to a character vector, "enclosing" the character vector and catenating it as a new item:

```
NAME NAME CNCAT 'TOP DOG LTD.'
```

If both arguments to CNCAT are character vectors, the result is a 2 item character nest in which each item comes from a corresponding argument. This allows you to catenate more than one customer name at a time via the notation:

```
NAME NAME CNCAT 'TOP DOG LTD.' CNCAT 'BOTTOM CAT CORP.'
```

Likewise, a function NNCAT may be written which will catenate two numeric nests. Unfortunately, the function cannot readily discern between an argument which is a numeric nest and an argument which is a simple numeric vector. (With CNCAT, a character nest is numeric and a character vector is character.) One way to solve this problem is to write 4 functions (NNCATSS, NNCATVS, NNCATSV, NNCATVV) which will respectively handle the 4 possible combinations of arguments. For example, NNCATVS will catenate a numeric nest ("vector") left argument to a simple numeric vector ("scalar") right argument:

```
UBILLS UBILLS NNCATVS ,0 3p0
```

You may catenate the unpaid bills information for more than one customer at a time via the notation:

```
UBILLS UBILLS NNCATVV (,0 3p0) NNCATSS ,1 3p40986 400 1000
```

Another approach is to provide the "scalar" (i.e. not numeric nest) argument as a matrix. Then, the function can tell its numeric nest (vector) arguments from its non-numeric-nest (matrix) arguments:

```
UBILLS UBILLS NNCAT (0 3p0) NNCAT 1 3p40986 400 1000
```
PROBLEM: Return the name and unpaid bills information for the 5th customer, as a character vector and a 3 column numeric matrix. Return the names and unpaid bills information for the 2nd, 8th and 12th customers, as a 3 item character nest and a 3 item matrix nest, respectively.

TOPIC: Selecting from Irregular Arrays

Using nested arrays, the 5th name and matrix of unpaid bills information may be returned by selecting and disclosing (converting from a nested scalar to a simple array):

\[
\begin{align*}
N & \leftarrow NAME[5] & \text{(APL2, APL*PLUS)} \\
U & \leftarrow UBILLS[5] \\
N & \leftarrow NAME[5] & \text{(SHARP APL)} \\
U & \leftarrow UBILLS[5]
\end{align*}
\]

The select-and-disclose operation is performed so frequently when working with nested arrays that a specific function ("pick") is available in APL2 and APL*PLUS to do just that:

\[
\begin{align*}
N & \leftarrow 5 \Rightarrow NAME & \text{(APL2, APL*PLUS)} \\
U & \leftarrow 5 \Rightarrow UBILLS
\end{align*}
\]

To select several items from a nested vector to return a subset nested vector, you may use indexing directly:

\[
\begin{align*}
N3 & \leftarrow NAME[2 8 12] & \text{(APL2, APL*PLUS, SHARP APL)} \\
U3 & \leftarrow UBILLS[2 8 12]
\end{align*}
\]

Using conventional APL, you may write functions CNIDX and NNIDX which will extract the desired information:

\[
\begin{align*}
N & \leftarrow NAME \text{ CNIDX 5} \\
U & \leftarrow UBILLS \text{ NNIDX 5} \\
U & \leftarrow ((\rho U + 3), 3) \rho U \\
N3 & \leftarrow NAME \text{ CNIDX 2 8 12} \\
U3 & \leftarrow UBILLS \text{ NNIDX 2 8 12}
\end{align*}
\]

These functions return nests (character or numeric) if the right argument is a vector and simple arrays (character or numeric vector) if the right argument is a scalar. Since the unpaid bills information is stored as a vector, the third statement above is necessary to reshape the resulting vector into a 3 column matrix.
Chapter 16

IRREGULAR ARRAYS

PROBLEM: Replace the name and unpaid bills information of the 5th customer by the character vector 'NEW CORP.' and by the 3 column numeric matrix NEWBILLS. Replace the names and unpaid bills information of the 2nd, 8th and 12th customers by those of the 4th, 5th and 6th customers.

TOPIC: Replacing Items of Irregular Arrays

Using nested arrays, the 5th name and matrix of unpaid bills information may be replaced by any of the following:

\[
\begin{align*}
\text{NAME}[5] &\leftarrow 'NEW CORP.' \quad (\text{APL2, APL*PLUS}) \\
\text{UBILLS}[5] &\leftarrow \text{NEWBILLS} \\
(5>\text{NAME}) &\leftarrow 'NEW CORP.' \quad (\text{APL2}) \\
(5>\text{UBILLS}) &\leftarrow \text{NEWBILLS} \\
\text{NAME}[5] &\leftarrow 'NEW CORP.' \quad (\text{SHARP APL}) \\
\text{UBILLS}[5] &\leftarrow \text{NEWBILLS}
\end{align*}
\]

The 2nd, 8th and 12th customers may be modified directly by indexing and index assignment:

\[
\begin{align*}
\text{NAME}[2, 8, 12] &\leftarrow \text{NAME}[4, 5, 6] \quad (\text{APL2, APL*PLUS, SHARP APL}) \\
\text{UBILLS}[2, 8, 12] &\leftarrow \text{UBILLS}[4, 5, 6]
\end{align*}
\]

Using conventional APL, you may write functions CNIDXA, NNIDXA and ASSIGN which will replace the desired information:

\[
\begin{align*}
\text{NAME} &\leftarrow \text{NAME CNIDXA 5 ASSIGN 'NEW CORP.'} \\
\text{UBILLS} &\leftarrow \text{UBILLS NNIDXA 5 ASSIGN ,NEWBILLS} \\
\text{NAME} &\leftarrow \text{NAME CNIDXA 2, 8, 12 ASSIGN NAME CNIDX 4, 5, 6} \\
\text{UBILLS} &\leftarrow \text{UBILLS NNIDX 2, 8, 12 ASSIGN UBILLS NNIDX 4, 5, 6}
\end{align*}
\]

The function ASSIGN is a simple function provided for your convenience. The CNIDXA and NNIDXA functions require 3 arguments (original nested array, indices to be replaced, simple or nested array to be inserted). Since APL syntax allows only 2 arguments, the 3rd argument must be passed as a global variable. The function ASSIGN assigns its right argument to the global variable <assign> and returns its left argument as its explicit result.

The CNIDXA and NNIDXA functions require a nest (in <assign>) if their right argument is a vector; they require a simple array if their right argument is a scalar.

-265-
PROBLEM: What is the average length (ANLEN) of customer names? How many unpaid bills (NUBILLS) exist?

TOPIC: Determining Shapes of Irregular Items

Using nested arrays, you must determine the shape of each item and then manipulate the resulting vector accordingly:

- \( \text{ANLEN} \gets (+/ \rho \cdot \text{NAME}) \div \rho \text{NAME} \) (APL2)
- \( \text{ANLEN} \gets (+/ \rho \cdot \text{NAME}) \div \rho \text{NAME} \) (APL*PLUS)
- \( \text{ANLEN} \gets (+/ \rho \cdot \# \cdot \text{NAME}) \div \rho \text{NAME} \) (SHARP APL)

Each of these expressions requires one more step than intuition suggests (\( \epsilon \), \( \varpi \) and \( \cdot \)). This additional step is needed because the result after performing the \( \rho \cdot \# \) (or \( \rho \cdot \# \cdot \)) function is still a nested vector (or 1 column matrix), whose items (or rows) are each one element vectors, since monadic \( \rho \) always returns a vector. The array of one element vectors must be converted into a simple vector of scalars.

The solutions for the second part of the problem are similar:

- \( \text{NUBILLS} \gets +/(\# \cdot \rho \cdot \text{NAME}) \) (APL2)
- \( \text{NUBILLS} \gets (+/ \rho \cdot \text{NAME}) \) (APL*PLUS)
- \( \text{NUBILLS} \gets +/0 -1 \rho \cdot \# \cdot \text{NAME} \) (SHARP APL)

Since the items of UBILLS are 3 column matrices, logic is included (\( \# \cdot \rho \cdot \# \)) to look at only the number of rows in each matrix (i.e. the first element of the shape of each matrix).

Using conventional APL, you may write functions CNLEN and NNLEN which return a vector of the lengths of the items in the specified character or numeric nest:

- \( \text{ANLEN} \gets (+/ \text{CNLEN NAME}) + 1 \text{NAME} \)
- \( \text{NUBILLS} \cdot (+/ \text{NNLEN UBILLS}) \cdot 3 \)
PROBLEM: Sort the database of customer information by customer name, alphabetically.

TOPIC: Sorting Character Nests

To sort the database of customer information, we need the grade-up vector which would put the names in alphabetically sorted order. Given this vector, say GRADE, the database may be reordered as follows:

\[
\begin{align*}
\text{ID} &\leftarrow \text{ID}[\text{GRADE}] \\
\text{TERR} &\leftarrow \text{TERR}[\text{GRADE}] \\
\text{NAME} &\leftarrow \text{NAME}[\text{GRADE}] \\
\text{UBILLS} &\leftarrow \text{UBILLS}[\text{GRADE}] \\
\text{NAME} &\leftarrow \text{NAME} \text{ CNIDX GRADE} \\
\text{UBILLS} &\leftarrow \text{UBILLS} \text{ NNIDX GRADE}
\end{align*}
\]

How do we determine GRADE? Using nested arrays, you must first convert NAME into a character matrix and then apply \( \& \) or CGRADEUP (see Searching and sorting chapter) on the character matrix.  

\[
\begin{align*}
\text{ALP} &\leftarrow '.,;:-/ABCDEFGHIJKLMNOPQRSTUVWXYZ' \\
\text{GRADE} &\leftarrow \text{ALP} \& \Rightarrow \text{NAME} \quad \text{(APL2)} \\
\text{GRADE} &\leftarrow \text{ALP} \& \uparrow \text{;NAME} \quad \text{(SHARP APL)} \\
\text{GRADE} &\leftarrow \text{ALP} \uparrow (\Rightarrow \rho''\text{NAME}) \uparrow''\text{NAME} \quad \text{(APL*PLUS)}
\end{align*}
\]

Using conventional APL, you may write a function CNGRADEUP which returns the grade vector that may be used to reorder the character vector items of a specified character nest into sorted order.

\[
\begin{align*}
\text{GRADE} &\leftarrow \text{ALP} \text{ CNGRADEUP NAME}
\end{align*}
\]

The CNGRADEUP function may be written employing techniques discussed in the Searching and Sorting chapter. As such, it does not need to convert its character nest right argument into a character matrix with as many columns as the longest item. In this way, the CNGRADEUP function is superior to the algorithms listed above for nested arrays. Each of these algorithms constructs a character matrix from the nest. If one name is very long, a WS FULL error may be generated.
PROBLEM: How many times does the name 'FASTENERS INC.' occur as a customer name? Which customer is 'ABC CORP.'? Which is 'DEF CORP.'? Which is 'GHR CORP.'?

TOPIC: Searching Character Nests

Using nested arrays, the number of times the name 'FASTENERS INC.' occurs may be determined by one of the following:

\[ +/\text{NAME} \equiv '\text{FASTENERS INC.'} \quad \text{(APL2, APL*PLUS)} \]
\[ +/\text{NAME} \equiv '\text{FASTENERS INC.'} \quad \text{(SHARP APL)} \]

Using conventional APL, you may write a function CNEQ which compares each item of a specified character nest to a specified character vector and returns a bit vector with one element per item in the character nest, the 1s corresponding to items which match the character vector.

\[ +/\text{NAME} \text{ CNEQ 'FASTENERS INC.'} \]

The CNEQ function may take two character nest arguments, or one character nest and one character vector argument (in either order) or two character vector arguments (returning a bit scalar). In other words, its behavior is parallel to that of the primitive scalar dyadic function equals (=).

The second part of the problem begs for an index result. Therefore, dyadic iota (\(\iota\)) is the logical choice. Using nested arrays,

\[ \text{NAME} \iota '\text{ABC CORP.' 'DEF CORP.' 'GHR CORP.'} \quad \text{(APL2, APL*PLUS)} \]
\[ \text{NAME} \iota '\text{ABC CORP.' 'DEF CORP.' 'GHR CORP.'} \quad \text{(SHARP APL)} \]

Using conventional APL, you may write a function CNIOTA which searches through its character nest left argument for the first occurrence of each of the items in its character nest right argument. If the right argument is a character vector, the result is a scalar.

\[ \text{NAME} \text{ CNIOTA 'ABC CORP.' CNCAT 'DEF CORP.' CNCAT 'GHR CORP.'} \]
Chapter 16

IRREGULAR ARRAYS

PROBLEM: Determine the total amount of unpaid bills for each customer, returning a numeric vector AMT with one element per customer.

TOPIC: Reducing Numeric Nests

Using nested arrays, the "each", "on" or "with" operators are needed:

\[
\text{AMT} \leftarrow 3 \cdot \sum \cdot UBILLS
\]

(\text{APL2, APL*PLUS})

\[
\text{AMT} \leftarrow +/ \left(1,0 \cdot UBILLS \neg \cdot UBILLS \cdot UBILLS \neg 0,2 \cdot UBILLS \right)
\]

(SHARP APL)

Using conventional APL, you may write a function NNSUMCOL which will sum the Nth column of each M-column matrix item stored as a numeric vector item in the specified numeric nest.

\[
\text{AMT} \leftarrow UBILLS \cdot \text{NNSUMCOL} \cdot 3 \cdot 3
\]

The first 3 in the right argument of NNSUMCOL is the number of columns in the raveled matrix items. The second 3 is the index of the column to be summed.

PROBLEMS:

(Solutions on pages 459 to 473)

1. Suppose you have a numeric nest (vector of numeric vectors) named SALES which contains one item per customer and for which each item is a vector of the customer's monthly sales. The length of each vector is a function of how long the customer has been generating sales. What expression will produce a vector (AVG) with one element per customer, where each element is the average monthly sales for the corresponding customer?

2. Given a character nest NAMES, what expression will return a character nest of its distinct (unique) items (UNAMES)?
3. Write one or more of the character nest functions mentioned in this chapter for your implementation of APL (if it does not have nested arrays). The functions are: CNEST, CNAM, CNCAT, CNIDX, CNIDXA, ASSIGN, CNLEN, CNGRADE, CNEQ, CNIOTA.

4. Write one or more of the numeric nest functions mentioned in this chapter if your implementation of APL does not have nested arrays. The functions are: NNEST, NNCATSS, NNCATVS, NNCATSV, NNCATVV, NNCAT, NNIDX, NNIDXA, ASSIGN, NNLEN, NNSUMCOL.
In both the business and scientific disciplines, the need occasionally arises to fit mathematical curves to empirical data. The process of curve fitting is a wonderful, magical process. Like most magic, curve fitting is illusion based upon reality. The reality is that rigorous mathematical algorithms are applied by the computer to determine the precise parameters of a "best" curve. The illusion is that this "best" curve somehow possesses more knowledge than the data; that it can be used to predict the future. This can be a dangerous and foolhardy view.

This is not to say that curve fitting is useless. Far from it. Curve fitting is not only useful; it is fun. Just be careful that you are not lured into thinking that the computer's intuition is better than your own.

In this chapter, we discuss the most complex of all the APL primitive functions, quad-divide (ffi). The chapter does not presume that you understand the concepts of numerical analysis and linear algebra needed to appreciate the algorithms applied within quad-divide.

PROBLEM: Given the following sample data and the function of a curve, find the coefficients of the curve.

<table>
<thead>
<tr>
<th>T</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Curve: \( T=(A\times X)+(B\times Y)+(C\times Z) \)

Problem: find scalars A, B, C

~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~
TOPIC: Using Quad-Divide

To provide some meaning to this example, suppose the three observations represent three salespeople. The first salesperson received no base salary (X) last year, was paid $3 in stock options (Y) and received no bonuses (Z). He generated 9 new customers (T). The second salesperson received $1 in base salary, $2 in stock options, $3 in bonuses and generated 20 new customers. The third salesperson received $4 in base salary, $1 in stock options, $2 in bonuses and generated 19 new customers.

We have a hypothesis that the number of new customers generated by a salesperson (T) is a direct function of the base salary (X), stock options (Y) and bonuses (Z), and that the function is in the form:

\[ T = (A \times X) + (B \times Y) + (C \times Z) \]

If we can determine the constants (coefficients) A, B and C in this formula, we can predict the number of new customers (T) which will be generated by a salesperson who is paid a specified base salary, stock option amount and bonus amount. Since we have three equations (9 = 3X + B; 20 = A + (2X) + (3Z); 19 = (4X) + B + (2Z)) in three unknowns (A, B, C), the problem can be solved by algebraic manipulations and substitutions. After tedious work, we discover that A = 2, B = 3, C = 4.

Quad-divide (Æ) can be used to solve this problem directly. Provide the dependent values (T) as its vector left argument, and the independent values (X, Y, Z) as its 3 column matrix right argument. The result is a vector of the three desired constants (coefficients).

\[ \begin{align*}
T &\leftarrow 9 \ 20 \ 19 \\
XYZ &\leftarrow 3 \ 3 \ 0 \ 0 \ 3 \ 0 \ 1 \ 2 \ 3 \ 4 \ 1 \ 2 \\
C &\leftarrow T \times XYZ \\
\end{align*} \]

\[ C \leftarrow 2 \ 3 \ 4 \]

Once the coefficients are known, it becomes a simple matter to apply the formula to hypothetical values. For example, how many new customers do we expect to be generated by a salesperson who is paid 0 in base salary, $3 in stock options and $4 in bonuses?

\[ 0 \ 3 \ 4 \ \times C \quad A \ (0 \times A) + (3 \times B) + (4 \times C) \]

\[ 25 \]

In order to solve problems like these (systems of linear equations), the data must satisfy certain requirements. For one thing, we must provide one equation (observation) for each unknown. Since there were three unknown constants (A, B, C) in the above example, we needed three observations. If we do not provide enough observations, there are an infinite number of solutions (values of A, B, C) which will satisfy the specified observations. Quad-divide will generate a DOMAIN ERROR.
Second, even if we provide one observation per unknown, there may be insufficient data to determine a unique solution. For example, in the following set of data, the third observation provides the same information as the second. The values are simply doubled.

<table>
<thead>
<tr>
<th>T</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

No amount of algebraic manipulation will produce a unique solution. To have a unique solution, each observation must be independent. That is, no observation may be a linear combination of the other observations.

If observations are not independent, quad-divide will generate a DOMAIN ERROR. The matrix right argument is said to be "singular" and cannot be "inverted". No unique solution exists.

Third, if we provide more observations than there are unknowns, there will likely be contradictory data so that no single set of coefficients will satisfy all of the observations. In such an instance, the best we can hope for is a set of coefficients which defines a formula which is reasonably accurate for all of the observations, though it may not be precise for any of them.

This is, in fact, what quad-divide returns. It does so by using the method of least squares. Simply stated, it returns those coefficients which when applied against the independent values will return a dependent value (the "expected") which is as close as possible to the specified dependent value (the "actual"). One measure of "noncloseness" (called the "mean squared error") is the mean of the squares of the differences between each of the actuals and their corresponding expecteds. The algorithm within quad-divide determines the set of coefficients which produces the smallest value of this noncloseness measure (hence, "least squares").

To illustrate this "best fit" behavior of quad-divide, let us add one more salesperson to our example and then determine the coefficients:

<table>
<thead>
<tr>
<th>T</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
We may determine the expecteds by applying the computed coefficients against the matrix of independent values:

\[
E = XYZ + .xC
\]

\[
\begin{array}{cccc}
8 & 20 & 18 & 13 \\
\end{array}
\]

While only one of these values exactly matches the actuals (T), the other values are reasonably close. The mean squared error is:

\[
\frac{\sum (E-T)^2}{\sum E}
\]

\[
1.5
\]

PROBLEM: During the last 5 years, the annual sales figures for your firm have been 27, 29, 35, 41, 44. If you assume that sales are growing at a linear rate, what do you project sales to be next year?

TOPIC: Forecasting

Forecasting is just curve fitting in disguise. Instead of expressing a formula in terms of several independent variables (such as X, Y, Z), we express it in terms of time (T). In this problem, we are searching for a formula which looks like:

\[
SALES = A + (B \times T)
\]

where T is some measure of time (such as 1, 2, 3, 4, 5 for the last 5 years) and A and B are the constants to be determined.

If we express our data in the same form as the data in the previous section, the solution becomes obvious.
The column of 1s is used to indicate that the first constant (A) is multiplied by 1 in each observation (i.e. in \( \text{SALES} = A + (B \times T) \)).

\[
\begin{array}{l}
\text{MAT} + 5 & 2 \rho 1 \ 1 \ 1 \ 2 \ 1 \ 3 \ 1 \ 4 \ 1 \ 5 \\
\text{SALES} + 27 & 29 \ 35 \ 41 \ 44 \\
C & \text{SALES} \times \text{MAT} \\
C & 21.4\ 4.6
\end{array}
\]

The two elements in C are our constants A and B. To project the sales for next year, we need only plug the time 6 into our formula:

\[
1 \ 6 + .\times C
\]

49

If we wish to see how closely our formula tracks the past five years, we can apply the constants to the time periods 1 to 5:

\[
\text{MAT} + .\times C
\]

26 30.6 35.2 39.8 44.4

We can compare these values to the actual sales (27, 29, 35, 41, 44).

Suppose we assume that sales are growing at a quadratic rate instead of a linear rate. The formula instead looks like:

\[
\text{SALES} = A + (B \times T) + (C \times T^2)
\]

Our data can be expressed in an expanded table:

<table>
<thead>
<tr>
<th>SALES</th>
<th>1 T T*2</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>1 1</td>
</tr>
<tr>
<td>29</td>
<td>1 2</td>
</tr>
<tr>
<td>35</td>
<td>1 3</td>
</tr>
<tr>
<td>41</td>
<td>1 4</td>
</tr>
<tr>
<td>44</td>
<td>1 5</td>
</tr>
</tbody>
</table>
The solution is no more complex:

\[
\text{MAT} \leftarrow (15) \cdot 0 \cdot 1 \cdot 2 \\
\text{SALES} \leftarrow 27 \ 29 \ 35 \ 41 \ 44 \\
C \leftarrow \text{SALES} \cdot \text{MAT} \\
C \\
22.4 \ 3.742857 \ .142857
\]

Nor is the projection:

\[
1 \ 6 \ 36 \ +. \times C \\
50
\]

Nor is the tracking:

\[
\text{MAT} \ +. \times C \\
26.2857 \ 30.4571 \ 34.9143 \ 39.6571 \ 44.6857
\]

Notice that a quadratic formula tracks better than a linear one. This in no surprise since the linear formula is just a special case of the quadratic formula (the case in which the coefficient of the squared term is 0). If the line fits better than any other quadratic form, the third coefficient will be zero.

After realizing that a quadratic formula fits better than a linear formula, we may jump to the conclusion that a cubic formula (e.g. \( \text{SALES} = A + (B \times T) + (C \times T^2) + (C \times T^3) \)) fits better still. And it does. However, we must bear two thoughts in mind before carrying this logic to its extreme. The first is that we must provide at least one observation per unknown constant. Otherwise, quad-divide will generate a DOMAIN ERROR because there are an infinite number of possible solutions. Hence, for our 5 observations (years), the most terms we can have in our formula is 5.

The second thing to consider is that the reason a higher-order formula fits better to the data is that it exhibits more helter-skelter behavior than a lower-order formula. It has more mood swings. Thus, the formula which is selected by quad-divide to perfectly match your 5 observations may send you into outer space when you project the sixth period.

The bottom line in this discussion is that despite the difficulty of the number-crunching task faced by the computer, your task is more difficult. It is your job, after all, to determine which formula you think your data fits. There are an infinity of possible formulas. The computer merely has to crunch out coefficients.
Chapter 17  CURVE FITTING

PROBLEM: As in the previous section, we wish to project forward one time period the sales figures from the past 5 years (27, 29, 35, 41, 44). We believe the data conforms to the formula, \( \text{SALES} = (A + (B \times T)) \).

TOPIC: Fitting Data to a Nonlinear Formula

Our problem here is that we need to determine two constants (A and B) but the formula is not expressed as an additive combination of two terms, each multiplied by one of the constants. We must transform the formula so that it is in a form suitable to quad-divide.

If we multiply both sides of the formula by \( A + (B \times T) \), we get:

\[ 1 = (\text{SALES} \times A) + (\text{SALES} \times B \times T) \]

From this linear form, we can solve directly for A and B:

\[
\begin{align*}
\text{SALES} &\rightarrow 27 \ 29 \ 35 \ 41 \ 44 \\
\text{TIME} &\rightarrow 5 \\
\text{MAT} &\rightarrow \text{SALES} \cdot [1.5] \text{SALES} \times \text{TIME} \\
C &\rightarrow (\rho \text{SALES} \cdot \rho 1) \cdot \text{MAT} \\
C &\rightarrow \text{.040625 -3.7567} \\
\end{align*}
\]

The projection follows directly:

\[ \frac{1}{C} \cdot x \cdot 6 \]

55.295

While we have accomplished our projection via this transformation, we must concede that quad-divide provides us the best coefficients for the transformed formula, not for the original formula. Hopefully, this distinction is not important. However, we must bear it in mind whenever we transform a formula.

Let us try fitting the data to another formula which is nonlinear and requires a transformation.

Formula: \( \text{SALES} = A \times B \times T \)

Transformed formula: \( (\ast \text{SALES}) = (\ast A) + (B \times T) \)

Solution:

\[
\begin{align*}
\text{MAT} &\rightarrow \text{TIME} \cdot \ast 0 \ 1 \\
C &\rightarrow (\ast \text{SALES}) \cdot \text{MAT} \\
A &\text{ Returns: (\ast A) and B} \\
(\ast C[11]) &\times \ast C[2] \times 6 \\
\text{51.426} \\
\end{align*}
\]
PROBLEM: In the previous sections, we use four different formulas to project 5 years of sales figures one year:

1. \( \text{SALES} = A + (B \times T) \)
2. \( \text{SALES} = A + (B \times T) + (C \times T^2) \)
3. \( \text{SALES} = A + (B \times T) \)
4. \( \text{SALES} = A \times B \times T \)

For each of these formulas, we obtain a different set of coefficients and a different projection (sales figure for the sixth year). Which projection is the best?

TOPIC: Finding the Best Formula

The best projection is that which exactly matches the actual. Unfortunately, the actual sixth year sales figure is not yet known. Therefore, we must determine the best projection in another way. Let's choose the projection that comes from the formula which most closely fits the data. (Bear in mind that the best fitting curve is not necessarily the best projector of the data.)

Which formula fits the best? Just as the mean squared error is used to determine the best coefficients for a given formula, so too can it be used to determine the best formula for a given set of data. The formula which produces the least mean squared error is the best fitting formula. Remember: the mean squared error is the mean of the squares of the differences between each of the actuals and their corresponding expecteds.

Let us compute the mean squared error for each formula:

\[
\begin{align*}
\text{SALES} & = 27, 29, 35, 41, 44 \\
\text{TIME} & = 1, 2, 3, 4, 5 \\
1. \text{SALES} & = A + (B \times T) \\
\text{MAT} & = \text{TIME} \times 0.1 \\
C & = \text{SALES} \times \text{MAT} \\
\text{EXP} & = \text{MAT} \times C \\
((\text{SALES} - \text{EXP}) \times 2) / \text{SALES} & = 1.04
\end{align*}
\]
2. \( \text{SALES} = A + (B \times T) + (C \times T^2) \)

\[
\begin{align*}
\text{MAT} &\leftarrow \text{TIME} \times 0.1 \times 2 \\
C &\leftarrow \text{SALES} \times \text{MAT} \\
\text{EXP} &\leftarrow \text{MAT} \times \text{EXP} \\
\left( (\text{SALES} - \text{EXP})^2 \right) &\div \rho \text{SALES} \\
0.983
\end{align*}
\]

3. \( \text{SALES} = (A + (B \times T)) \)

\[
\begin{align*}
\text{MAT} &\leftarrow \text{SALES} \times [1.5] \times \text{SALES} \times \text{TIME} \\
C &\leftarrow ((\rho \text{SALES})^2) \times \text{MAT} \\
\text{EXP} &\leftarrow (\text{TIME}^0 \times 0.1) \times \times C \\
\left( (\text{SALES} - \text{EXP})^2 \right) &\div \rho \text{SALES} \\
1.85
\end{align*}
\]

4. \( \text{SALES} = A \times B \times T \)

\[
\begin{align*}
\text{MAT} &\leftarrow \text{TIME} \times 0.1 \\
C &\leftarrow (\rho \text{SALES}) \times \text{MAT} \\
\text{EXP} &\leftarrow (\times [1] \times \times [2] \times \text{TIME} \\
\left( (\text{SALES} - \text{EXP})^2 \right) &\div \rho \text{SALES} \\
1.10
\end{align*}
\]

The smallest mean squared error for the above formulas is the one for the second formula. Therefore, the second formula is the "best" formula (of the four selected) and the best projection is the one from that formula (50).

-------

PROBLEMS: (Solutions on pages 474 to 475)

1. Each month, you buy 4 styles of envelopes from your paper supplier (styles A, B, C, D). The supplier has never bothered to itemize the different styles on the invoice. The invoice shows a total amount only. Suppose you bought the following quantities of envelopes during the previous 4 months:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Total Invoice</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>------------</td>
</tr>
<tr>
<td>32</td>
<td>61</td>
<td>15</td>
<td>82</td>
<td>$60.62</td>
</tr>
<tr>
<td>35</td>
<td>104</td>
<td>10</td>
<td>82</td>
<td>$59.57</td>
</tr>
<tr>
<td>37</td>
<td>83</td>
<td>5</td>
<td>85</td>
<td>$56.70</td>
</tr>
<tr>
<td>25</td>
<td>62</td>
<td>14</td>
<td>85</td>
<td>$60.42</td>
</tr>
</tbody>
</table>
Assuming prices have remained constant during this period and that the invoice does not include volume discounts or other credit or debit adjustments, what are the unit prices for the 4 styles of envelopes?

2. During the last 2 months, you have been making weekly checks of your inventory of muffler bearings. The results are given here:

<table>
<thead>
<tr>
<th>Week</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>On hand</td>
<td>1850</td>
<td>1772</td>
<td>1705</td>
<td>----</td>
<td>1508</td>
<td>1490</td>
<td>----</td>
<td>----</td>
<td>1250</td>
</tr>
</tbody>
</table>

(During week 4 the plant was closed and during weeks 7 and 8 you were on vacation.) Assuming the supply of bearings is being depleted at a steady (linear) rate, when will the supply be exhausted (assuming no restocking)?

3. What is the radius and center of the circle which best fits the points (4,1), (3,2), (2,3), (4,5), (7,2), (6,4), i.e. for which \(X=4\ 3\ 2\ 4\ 7\ 6\) and \(Y=1\ 2\ 3\ 5\ 2\ 4\)? The general formula for a circle is:

\[(R^2) = ((X-CX)^2) + ((Y-CY)^2)\]

where \(R\) is the radius, and \((CX,CY)\) is the center.

4. In the Sorting and Searching chapter, a function CMIOTA is presented for searching through the rows of one character matrix for the location of the rows of a second. The function is designed to use one of two different algorithms depending upon the number of rows in its arguments. In the Computer Efficiency Considerations chapter, the two algorithms are timed for a variety of different size arguments (in a problem at the end of that chapter). Suppose you have constructed 4 vectors which contain the results of those timings:

L: The number of rows for left arguments;
R: The number of rows for right arguments;
T1: The average time required to run CMIOTA using the first (sorting) algorithm for a left argument with L rows and a right argument with R rows;
T2: The average time required to run CMIOTA using the second (looping) algorithm for a left argument with L rows and a right argument with R rows.

According to the logic in the Sorting and Searching chapter, the vectors are related according to the approximate formulas:

\[
T_1 = C_4 + (C_5 \times (R + L)) \\
T_2 = C_1 + (R \times (C_2 + (C_3 \times L)))
\]

Determine the values for \(C_1, C_2, C_3, C_4\) and \(C_5\). You may replace the like-named variables in the CMIOTA function so that it will use the fastest algorithm for any combination of arguments in your APL environment.
Chapter 18

FINANCIAL UTILITIES

In a variety of business oriented computer applications, sophisticated financial calculations are required. Many of these calculations deal with the time-value-of-money concept, i.e. the concept of interest. In this chapter, we develop utilities which handle some of the more common financial calculation requirements.

This chapter does not provide a comprehensive treatment of the theory of interest or of financial analysis. It provides only as much material as is needed for you to grasp the important concepts. Nor does it provide a comprehensive library of financial software. It provides some generally useful utility functions.

PROBLEM: You are considering the purchase of a new machine. The amount you must borrow from the bank is $225,000 to be repaid in monthly installments at 13% interest over 10 years. What will be the monthly payment?

TOPIC: Interest and Annuities

We will work toward the solution to this problem gradually.

Suppose you deposit $100 in your bank account at the beginning of the year. If at the end of the year your balance is $112, you have earned $12 on your $100 deposit and so your "effective" (annual or APR) interest rate is .12 (12/100). At the end of two years (if interest rates remain the same), you will have $125.44 ($112 + $112×.12 or $112×1.12). After three years, you will have $140.49 ($125.44×1.12). And so on. In general, after N years, you will have $100×1.12×N. The factor 1.12 (1 plus the interest rate) is called the annual "accumulation factor".
Chapter 18
FINANCIAL UTILITIES

Suppose we look at your banking situation in a reverse way, from the present back into the past. If you have a $100 balance today, how much must you have had in the account one year ago? If we call the desired amount AMT, we know from the logic above that AMT \times 1.12 = $100. Therefore, AMT = $100/1.12 = $89.28. Two years ago, you had $79.72 ($89.28/1.12). And so on. In general, N years ago, you had $100/1.12^N. The factor $1.12 (the reciprocal of the annual accumulation factor) is called the annual "discount factor".

If instead of paying you 12% per year, the bank offers to pay you 1% per month, how much money will you have at the end of the year? Since the monthly accumulation factor is 1.01, you will have $101 ($100 \times 1.01) after one month, $102.01 ($101 \times 1.01) after two months, $103.03 ($102.01 \times 1.01) after three months, ... and $112.68 ($111.57 \times 1.01 or $100 \times 1.01 \times 12) after twelve months. Since you have earned $12.68 on your $100 deposit, your effective interest rate is .1268. This rate may also be called a "nominal" rate of .12, compounded monthly (.12 = .01x12 months). The term "compounded" (or converted) refers to the periodic process of applying the accumulation factor to the balance to derive the new balance.

Since an interest rate may be expressed in "effective" or "nominal" terms, it is important to understand the distinction between the two and to be able to convert between them. The relationship between them follows:

\[(1+E_{INT}) = (1+N_{INT}/CONV) \times CONV\]

where:

\[E_{INT} = \text{effective interest rate (e.g. .1268)}\]
\[N_{INT} = \text{nominal interest rate (e.g. .12)}\]
\[CONV = \text{number of interest conversion (compounding) periods per year}\]

It is instructive to note that for a given nominal interest rate, the equivalent effective interest rate increases as the conversion frequency increases. That is, the more frequently the bank compounds your nominal interest, the more you will have at the end of the year. However, the amount of increase in the effective interest rate drops off as the conversion frequency gets bigger and bigger. In fact, it makes little difference whether you compound weekly or every second. This brings up the concept of "continuous" compounding. That is, by using a bigger and bigger value of CONV in the formula above, the value of (1+E_{INT}) will eventually (when CONV becomes infinitely large) become constant. This value (called the "force of interest") may be computed by using a large value of CONV (say, 10000) or by using the following formula (which may be derived mathematically):

\[(1+E_{INT}) = N_{INT} \text{ (when CONV is infinitely large)}\]

Because of marketing appeal or because of the mathematical simplicity, financial institutions occasionally quote interest rates
"compounded continuously". You may convert them to equivalent effective rates by applying the above formula.

The following functions may be used to convert between nominal and effective interest rates.

\[ V \rightarrow E \rightarrow C \text{ EFFECTIVE } N;R;S;T \]

1. Converts the nominal interest rates in \( N \),
2. compounded \( C \) times per year, into effective
3. (annual) interest rates. A value of \(-1\) for
4. \( C \) implies continuous compounding. \( C \) and \( N \)
5. may have any shapes as long as they are
6. scalar conformable.
7. \( R \)
8. Determine shape of result:
9. \( S \rightarrow \rho C + N \)
10. Construct all zero raveled result:
11. \( R \rightarrow \rho E \left( \frac{x}{S} \right) \rho 0 \)
12. Convert arguments to same-length vectors:
13. \( C \rightarrow \rho C \)
14. \( N \rightarrow \rho N \)
15. Select rates with noncontinuous compounding:
16. \( T \rightarrow \rho^{-1} \)
17. \( E[T/\rho T] \leftarrow 1 + (1 + (T/N) \div T/C) \ast T/C \)
18. Select continuous rates:
19. \( T \leftarrow T \)
20. \( E[T/\rho T] \leftarrow 1 + * T/N \)
21. Reshape result:
22. \( E \rightarrow S \rho E \)

\[ V \]
Let us extend our discussion to include annuities. An "annuity" is a regular series of payments. Suppose, for example, you plan to deposit $100 in your bank account at the beginning of each year for the next 10 years. If the effective interest rate is .12, what will be your balance at the end of 10 years?

The first payment will have compounded 10 times, the second 9 times, and so on. The sum of these amounts is the balance:

\[
\text{BALANCE} = 100 \times 1.12^{10} + 100 \times 1.12^9 + \ldots + 100 \times 1.12^1
\]

By dividing each side of this equation by 1.12 to get a second equation, by subtracting the second equation from the first equation and by algebraic manipulation, we discover that:

\[
\text{BALANCE} = 100 \times 1.12 \times ((1.12^{10}) - 1)/.12
\]

By performing the calculations, we learn that the "future value" of this 10 year annuity is $1,965.46.

Suppose we look at a 10 year annuity in a reverse way, from the beginning of the annuity rather than the end. What is the value today (the "present value") of an annuity in which you deposit $100 at the beginning of each year for the next 10 years? That is, what single amount can you deposit today that will compound to $1,965.46 at the end of 10 years, i.e. that will result in the same future value as will the annuity?
If PV (present value) is the amount we are seeking, the equation we need to solve is:

\[(PV \times 1.12^{10}) = 100 \times 1.12 \times ((1.12^{10}) - 1) / 1.12\]

from which we get:

\[PV = 100 \times 1.12 \times (1 - 1.12^{-10}) / 1.12\]

By performing the calculations, we learn that the present value of this 10 year annuity is $632.83. That is, a single payment today of $632.83 is equivalent to a 10 year annuity of $100 starting today. Each will compound to $1,965.46 at the end of 10 years.

Having presented some examples to expose you to the concepts of compounding, annuities and present value, let us now take a leap forward and work with more general annuities. First, some definitions:

TERM: the number of years during which the annuity is paid;

PAY: the annual payment amount;

PER: the number of payments per year (PAY/PER per payment);

PDEF: the fraction of a payment period preceding (deferring) each payment; 0 if the payment occurs at the beginning of the payment period and 1 if at the end;

DEF: the number of years from the valuation date to the first period; 0 if the present value of the annuity is desired and -TERM if the future value is desired;

CONV: the number of interest conversion (compounding) periods per year;

INT: the nominal annual interest rate.

We will illustrate these terms by depicting a sample annuity on a "time" diagram. Below is the time diagram of an annuity for which the present value is desired. The annuity will be paid for 3 years (TERM=3), will be $200 per year (PAY=200) paid semiannually (PER=2) at the end (PDEF=1) of each half year payment period and will commence after 1 year (DEF=1) without payments. The present value is to be computed at a nominal interest rate of 12% (INT=.12) compounded monthly (CONV=12).
The general formula for the value of an annuity is:

\[ \text{VAL} = \text{PAY} \times (V \times \text{DEF} + \text{PDEF} \div \text{PER}) \times (1 - V \times \text{TERM}) \div \text{PER} \times (1 - V \div \text{PER}) \]

where:

- **VAL**: the value (present, future or otherwise) of an annuity as of DEF years before the first payment period;
- **V**: the annual discount factor, computed from INT and CONV,

\[ V = (1 + \text{INT} \div \text{CONV}) \div \text{CONV} \]

Some modifications to this formula are needed for some possible extreme conditions. First, if TERM is infinite (such an annuity is called a "perpetuity"), the V \times TERM term of the formula becomes 0 and may be omitted. Second, if CONV is infinite (continuous interest compounding), the definition of V becomes:

\[ V = \div \text{INT} \]

Third, if PER is infinite ("continuous payments", a theoretical concept), the PDEF \div PER term of the formula becomes 0 and may be omitted, and the PER \times 1 - V \div PER term becomes \(-\div V\).

Given this formula and these possible modifications, a niladic function VALUE can be written which computes the value of an annuity given all of its parameters as global variables. The global variables have the same names as the parameters defined above except the first letter is from the alternate character set (TERM, pAY, pER, pDEF, dEF, cONV, iNT). Infinite values are represented by the value \(-1\) (for TERM, pER or cONV). The global variables may have any shape as long as they are all scalar conformable with one another, that is as long as the expression,

\[ \text{TERM} + \text{PAY} + \text{PER} + \text{PDEF} + \text{dEF} + \text{cONV} + \text{iNT} \]

does not generate a RANK ERROR or a LENGTH ERROR.

All of the global values (except iNT) have default values which are used if that global variable does not exist.
Chapter 18 \hspace{1cm} \textbf{FINANCIAL UTILITIES}

\begin{verbatim}
[WSID: INTEREST]
\[ VAL\leftarrow VALUE; CONV; DEF; E; INT; N; NDPP; PAY; PD; PDEF; PER; R; SHAPE; TERM; V \]

[1] A Returns the present or future value of a stream
[2] A of uniform cash flows defined by the optional
[3] A global variables:
[4] A
[5] A default name description
[6] A -------- ---- -----------------------------
[7] A 1 TERM no. years of payments (-1=perpetuity)
[8] A 1 PAY annual payment amount
[9] A 1 PER no. payments per year (-1=continuous)
[10] A 1 pDEF is payment at start (0) or end (1) of
[11] A payment period (or fractional)
[12] A 0 DEF no. years from valuation date to
[13] A first payment period
[14] A 1 CONV no. interest conversion (compounding)
[15] A periods per year (-1=continuous)
[16] A - INT nominal annual interest rate (or force
[17] A of interest if CONV=-1)
[18] A
[19] A Determine values from globals or defaults:
[20] E\leftarrow E\div N \leftarrow 'TERM'
[21] TERM\leftarrow E(N)/(E-N)\div '1'
[22] E\leftarrow E\div N \leftarrow 'PAY'
[23] PAY\leftarrow E(N)/(E-N)\div '1'
[24] E\leftarrow E\div N \leftarrow 'PER'
[25] PER\leftarrow E(N)/(E-N)\div '1'
[26] E\leftarrow E\div N \leftarrow 'PDEF'
[27] PDEF\leftarrow E(N)/(E-N)\div '1'
[28] E\leftarrow E\div N \leftarrow 'DEF'
[29] DEF\leftarrow E(N)/(E-N)\div '0'
[30] E\leftarrow E\div N \leftarrow 'CONV'
[31] CONV\leftarrow E(N)/(E-N)\div '1'
[32] INT\leftarrow INT
[33] A
[34] A Construct result as a vector; reshape when done:
[35] SHAPE\leftarrow TERM+PAY+PER+PDEF+DEF+INT+CONV
[36] R\leftarrow SHAPE
[37] TERM\leftarrow RpTERM
[38] PAY\leftarrow RpPAY
[39] PER\leftarrow RpPER
[40] PDEF\leftarrow RpPDEF
[41] DEF\leftarrow RpDEF
[42] INT\leftarrow RpINT
[43] CONV\leftarrow RpCONV
[44] A Convert interest to annual discount factor:
[45] V\leftarrow Rp0
[46] A Noncontinuous compounding:
[47] E\leftarrow CONV\div '1'
[49] A Continuous compounding:
[50] E\leftarrow E
[51] V[E/\div V]**-E/INT
\end{verbatim}
Using the VALUE function, you can now answer the problem which began this section. The repayment schedule of a loan is simply an annuity. The present value of that annuity is exactly equal to the amount borrowed from the bank. We know the following:

\[ tE\text{RM} \leftarrow 10 \]
\[ pE\text{R} \leftarrow 12 \]
\[ pD\text{E}F \leftarrow 1 \]
\[ dE\text{F} \leftarrow 0 \]
\[ cO\text{NV} \leftarrow 12 \]
(probably, but ask the bank)
\[ iNT \leftarrow .13 \]

The only unknown parameter is \( pAY \), whose value we seek. However, we know the present value (225000) which will be the result of VALUE if we provide the correct value of \( pAY \). We may use VALUE iteratively, modifying the value of \( pAY \) in a trial and error fashion until the result of VALUE is 225000. Alternatively, since we know that the result of VALUE is directly proportional to the value of \( pAY \), we may arbitrarily set \( pAY \) to 12 (i.e. 1 per month) and divide the result of VALUE into 225000 to determine the factor by which \( pAY \) must be multiplied to produce the desired present value.

\[ pAY \leftarrow 12 \]
\[ 225000 \div \text{VALUE} \]
\[ 3359.49 \]

This is the monthly payment.

If you wish to experiment with the rate of interest to determine its effect upon the monthly payment, you may do it as follows:

\[ iNT \leftarrow .10 .11 .12 .13 .14 .15 \]
\[ 225000 \div \text{VALUE} \]
\[ 2973.39 3099.38 3228.1 3359.49 3493.49 3630.04 \]
Chapter 18  FINANCIAL UTILITIES

To produce a table of the monthly payments for each of these six interest rates, for 5, 10, 15 and 20 years, you may do it as follows:

```
iNT<.10 .11 .12 .13 .14 .15 0 0 0 0
TERM<0 0 0 0 0 5 10 15 20
2*225000/VALUE
4780.59 2973.39 2417.86 2171.30
4892.05 3099.38 2557.34 2322.42
5005.00 3228.10 2700.38 2477.44
5119.44 3359.49 2846.79 2636.05
5235.36 3493.49 2996.42 2797.92
5352.73 3630.04 3149.07 2962.78
```

### PROBLEM:

Having borrowed $225,000 from the bank to be repaid in monthly installments at 13% interest over 10 years, how much will you have paid after one year? Of that amount, how much will have gone toward payment of interest and how much toward repayment of the original principal of $225,000?

### TOPIC: Loan Amortization Schedules

From the function presented in the prior section, we can answer the first part of the problem.

```
TERM<10
PER<12
pDEF<1
def<0
CONV<12
iNT<.13
PAY<1
2*ANNUAL<225000/VALUE
40313.9
```

This annual loan repayment consists of an interest portion and a principal repayment portion. To determine the two portions, we need only determine the remaining principal at the end of the year. The difference between the original principal ($225,000) and the remaining principal is the amount of principal repaid. The difference between the total annual payment ($40,313.90) and the principal repaid is the interest paid.

So how do we determine the remaining principal at the end of the year? The principal is the present value of the remaining payments.
That is, the amount you owe the bank at any given time (the principal) is equivalent to the value at that time of the remaining payments. For this reason, the bank is indifferent (theoretically) to whether you repay the principal of the loan or continue to make the loan payments.

The principal at the end of the year may be computed by the following:

```
tERM←9
pAY←ANNUAL
□+REMAINING+VALUE
213252.48
```

The principal repaid and interest paid follow directly:

```
□+PRINCIPAL+225000-REMAINING
11747.52
□+INTEREST+ANNUAL-PRINCIPAL
28566.38
```

By applying the logic in this section, we may write an APL function which will generate a "loan amortization table". A loan amortization table shows the breakout of each loan payment into its interest and principal repayment portions. The monadic function SCHEDULE takes a 6 element vector right argument which defines the parameters of the loan:

1. TERM: the number of years of loan repayment;
2. PAY: the annual repayment amount;
3. PER: the number of payments per year;
4. DEF: the number of years from the loan to the first repayment period;
5. CONV: the number of interest conversion (compounding) periods per year;
6. INT: the nominal interest rate.

The result of SCHEDULE is a two column matrix of the loan amortization table. The matrix has one row per loan payment (TERM×PER). The values in the first column are the portions of each payment which represent principal repayment. The values in the second column are the portions which represent interest payments.

We may solve the problem above as follows:

```
MAT←SCHEDULE 10,ANNUAL,12 0 12 .13
```

where ANNUAL was computed above by the VALUE function. MAT has 120 rows (one row per loan payment). The principal repaid and interest paid during the first year are:
There are two identities about the loan amortization table which should be noted. The sum of each row is the periodic payment amount:

\[(+/MAT)^{\cdot}=\text{ANNUAL}/12\]

The total of the principal repayment column is the original loan amount:

\[225000=+/MAT[;1]\]
Chapter 18
FINANCIAL UTILITIES

\[ V \] SCHEDULE (continued)

\[ 36 \] a Noncontinuous compounding:
\[ 37 \] L1:V←(1+INT÷CONV)∗-CONV
\[ 38 \] a Determine present value (principal balance) at
\[ 39 \] a start of each payment period:
\[ 40 \] L2:VAL←(PAY×V÷PER)×(1-V*(TERM×PER)÷PER)÷PER×1-V÷PER
\[ 41 \] a Reset first value to loan amount if repayment
\[ 42 \] a is deferred:
\[ 44 \] a Principal repaid:
\[ 45 \] PRIN←VAL-1÷VAL,0
\[ 46 \] a Interest paid:
\[ 47 \] INTER←(PAY÷PER)-PRIN
\[ 48 \] a Adjust interest to keep it from exceeding
\[ 49 \] a periodic payment:
\[ 50 \] INTER←(INTER)÷(INTER)÷PAY÷PER
\[ 51 \] INTER←INTER-1÷0,INTER
\[ 52 \] PRIN←(PAY÷PER)-INTER
\[ 53 \] MAT←PRIN,[1.5]INTER

\[ V \]

##### PROBLEM: You have been presented an opportunity to invest $10,000
of your money now in exchange for the promise of several
future cash payments. The schedule of payments follows:

- May 12, 1987  You pay $10,000
- Jan. 1, 1989  You receive $2,000
- Jan. 1, 1990  $3,000
- Jan. 1, 1991  $4,000
- Jan. 1, 1992  $5,000
- May 12, 1992  $1,000

Is this a good investment? Are you better off leaving your
money in the bank at 10%?

##### TOPIC: Internal Rate of Return

One approach to this problem is to explore two scenarios. In the
first scenario, you deposit $10,000 in your bank now (May 12, 1987)
and leave it there. At the end of 5 years (May 12, 1992), at 10%
effective interest, your money has compounded to $16,105.10
($10,000×1.10×5). In the second scenario, you deposit nothing now
but instead deposit $2,000 on Jan. 1, 1989, $3,000 on Jan. 1, 1990,
and so on as in the schedule above. Compute the balance as of May 12, 1992.

To compute the balance for the second scenario, you must convert the dates to more manageable measurements of time. By using the TODAYS function presented in the Manipulating Dates chapter, you can translate the dates into days from May 12, 1992:

\[
\text{DAYS=TODAYS 19890101 19900101 19910101 19920101} \\
19920512 \\
D=DAYS[5]-DAYS \\
1227 862 497 132 0
\]

We can translate these days into "years" by dividing by 365. Then, it is a simple matter to accumulate each amount (the annual accumulation factor is 1.10) for the corresponding number of years.

\[
2000 3000 4000 5000 1000+1.10\times\text{DAYS/365}
\]

\[
17242.31
\]

Since $17,242.31 is greater than $16,105.10, you will be better off to accept this investment opportunity than to leave your $10,000 in the bank.

Will the conclusion be the same if the rate of interest offered by the bank is 12%? How about 15%? The higher the rate of interest, the less important are future cash flows and the more important are current flows. At what interest rate will the accumulated value of the investment cash flows be exactly the same as the accumulated value of your money left in the bank? That is, at what interest rate will you be indifferent (ignoring the riskiness of the investment) between making the investment and leaving your money in the bank? This rate of interest is called the "internal rate of return" (IROR) of the investment cash flows. If you can realize a higher interest rate than the IROR by an alternative investment (such as leaving your money in the bank), you should not invest. However, an investment is a good investment if the IROR of its cash flows is higher than any alternative investment.

To determine the IROR of a set of cash flows, you must employ an iterative (trial and error) technique known as "successive approximations". Your task is to determine the interest rate (I) for which the following equation is true:

\[
(10000\times(1+I)^{T0}) = (2000\times(1+I)^{T1}) + (3000\times(1+I)^{T2}) + \ldots
\]

where T0, T1, T2, ... are the numbers of years from the corresponding cash flow to the date of the last cash flow.

You may try different interest rates and see which produces the best results. From these observations you can choose better interest rates and try again. By repeating this procedure, you will gradually narrow in on the correct IROR.
Chapter 18  
FINANCIAL UTILITIES

To speed up this process, you may apply a technique known as the Newton-Raphson Method which uses the results of the previous guess to make an intelligent next guess. The guesses converge to the correct result much faster than standard interpolation procedures because the method considers the derivative (slope) of the formula.

The method is:

\[ \text{NEXTI} = f(\text{LASTI}) + f'(\text{LASTI}) \]

where:

\[ \text{NEXTI} \]: the next interest rate to try;
\[ \text{LASTI} \]: the previous interest rate tried;
\[ f(I) \]: the formula as a function of the interest rate \( I \);
\[ f'(I) \]: the derivative with respect to \( I \) of the formula \( f(I) \).

Given this method, we may write a function IROR which returns the internal rate of return for a specified set of cash flows. The left argument is a vector of the dates (YYYYMMDD) of the cash flows and the right argument is a vector of the amounts of the corresponding flows, where outflows are represented as negative numbers and inflows are represented as positive numbers (or vice versa). The solution to this problem is then:

\[
\begin{align*}
\text{DATES} & \leftarrow 19870512 \ 19890101 \ 19900101 \ 19910101 \ 19920101 \ 19920512 \\
\text{AMTS} & \leftarrow -10000 \ 2000 \ 3000 \ 4000 \ 5000 \ 1000 \\
\text{DATES} & \quad \text{IROR} \quad \text{AMTS} \\
0.12165 & 
\end{align*}
\]

From this, we conclude that the investment should be made if we have available no alternative investments which will generate more than 12.165%.

If an investment opportunity involves several outflows interspersed among the inflows, it is possible that the flows may not define a distinct internal rate of return. That is, several different interest rates may be used to accumulate (or discount) all the outflows to the same value as the inflows. Beware.

The IROR function is listed below. Note that it expresses the formulas as present value formulas in terms of the annual discount factor \( V \) (i.e. the reciprocal of the accumulation factor) rather than in terms of the interest rate. Also note that the iterations stop once the discount factor is determined within \( 0.0000001 \) or when 10 iterations have occurred. If the result is not determined in 10 iterations, the result is set to 0. Modify the function if you desire greater accuracy or more iterations or if you wish to try a starting value other than 10%.
Chapter 18

FINANCIAL UTILITIES

[WSID: INTEREST]

\[ \text{INT} \leftarrow \text{DATES} \quad \text{IROR} \quad \text{AMTS} ; \quad \text{DAYS} ; \quad \text{DIFF} ; \quad I ; \quad \text{TAMTS} ; \quad \text{TYRS} ; \quad V ; \quad \text{YRS} \]

[1] Computes the internal rate of return, as an
[2] effective (annual) interest rate, for the stream
[3] of cash flows defined by the left argument vector
[4] and the
[6] Positive amounts are inflows and negative amounts
[7] are outflows. Requires subfunction: TODAYS.

[8] A

[9] Translate dates into days since Feb. 29, 0000:

DAYS \leftarrow \text{TODAYS} \quad \text{DATES}

[10] Translate days into years (365 days per year)

YRS \leftarrow (\text{DAYS} - 1) / \text{DAYS} \times 365

[11] Precompute factors needed in formula below:

\text{TAMTS} \leftarrow \text{AMTS} \times \text{YRS}

\text{TYRS} \leftarrow \text{YRS} - 1

[12] Start with an effective interest rate of 10 pct.:

\text{V} \leftarrow 1.1

[13] Number of iterations performed so far:

I \leftarrow 0

[14] Apply Newton-Raphson to get new V:

\text{LOOP:} \quad \text{V} \leftarrow \text{V} - \text{DIFF} \leftarrow (\text{AMTS} + \text{V} \times \text{YRS}) \div \text{TAMTS} + \text{V} \times \text{TYRS}

[15] Exit if done (change less than 0.000001):

\rightarrow (1 \times 10^{-7} \leq \text{DIFF}) \rightarrow \text{DONE}

[16] Branch to next iteration unless 10 itns. already:

\rightarrow (10 \times \text{I} + 1) \rightarrow \text{LOOP}

[17] Else set discount factor to 1 if unknown in 10 itns.:

\text{V} \leftarrow 1

DONE: \quad \text{INT} \leftarrow 1 \div \text{V}

\text{V}

~~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~ ~~~~~

PROBLEM: Suppose you purchase a bond on February 17, 1987 for
$4225. On April 1, 2003 (the maturity date), the bond will
mature and you will receive $5000 (the par or redemption
value). From the purchase date to the maturity date, you
will receive semiannual interest payments (coupons) of $300
on April 1 and October 1 of each year. (The annual coupon
rate is 12%, i.e. \(0.12 = (2 \times 300) + 5000\)). In addition to the
$4225 purchase price, you must pay the seller his portion
of the upcoming (April 1) coupon. The computation of this
payment (the accrued interest) is a proration based upon
the number of days of the 6 month coupon period during
which the bond was held by the seller. What is the
internal rate of return of this investment?

-296-
Chapter 18  

TOPIC: Bond Calculations

The internal rate of return of a bond is called its "yield to maturity". The IROR may be determined by the methods described in the previous section. That is, you may construct a vector of the 34 dates on which cash flows occur (19870217 19870401 19871001 19880401 ... 20030401) and a corresponding vector of the amounts payable on those dates. You may then use the IROR function directly.

However, since the cash flows of the bond consist of a simple annuity (the coupon payments) and a single maturity payment, a relatively simple formula exists to describe the present value of the bond's future cash flows. By taking the derivative of this formula (tedious but not difficult) and applying the Newton-Raphson Method, the yield to maturity may be determined in a few iterations.

Suppose you need to determine the yields to maturity for an entire portfolio of bonds (say 500 bonds). If you use the former approach (the IROR function), you will need to create the date and amount vectors individually for each bond (since they may be of different lengths and may involve different dates). If you use the second approach (the formula), you may perform the successive approximation process on all the bonds at once. After a few iterations, you will have the yields for all the bonds.

The function FCYIELD ("fixed coupon yield"), listed below, uses this latter approach to compute the yield to maturity for one or more bonds. Once the yield is determined for a given bond, that bond is excluded from the successive approximation process. This function is a extremely efficient solution to a problem which is iterative by nature and is generally viewed as a "processing hog" when solved using APL.

The result of FCYIELD is a "couponly" rate (e.g. a 6 month rate for semiannual coupons). To convert the result to an effective (annual) rate, use the EFFECTIVE function presented earlier in this chapter.

One final note: this approach uses the formula for a regular annuity. However, because of leap years and months of irregular lengths, the coupon payments are not perfectly regular. The FCYIELD function (and the investment community in general) assumes that the year consists of 12 30-day months. Dates which have a day portion of 31 (e.g. 5/31/87) are treated like the 30th day of the same month (e.g. 5/30/87). Therefore, the results are not as precise as they would be if more care were taken when counting days. However, for most purposes the accuracy of the results is adequate.


Chapter 18 FINANCIAL UTILITIES

[WSID: INTEREST]

\[ \text{YLD} \cdot \text{FCYIELD}\ 
\text{PARAMS;}\ 
\text{COST;}\ 
\text{COUP;}\ 
\text{CRATE;}\ 
\text{CRY;}\ 
\text{DAYS;}\ 
\text{DIFF;}\ 
\text{F;}\ 
\text{F2;}\ 
\text{F3;}\ 
\text{I;}\ 
\text{IND;}\ 
\text{MCOST;}\ 
\text{MDATE;}\ 
\text{MORE;}\ 
\text{MVAL;}\ 
\text{N;}\ 
\text{NCOUPS;}\ 
\text{NEW;}\ 
\text{OK;}\ 
\text{PAR;}\ 
\text{PDATE;}\ 
\text{RY;}\ 
\text{W;}\ 
\text{Y} \]

[1] a Returns couponly yield rates for fixed-coupon securities defined in PARAMS, one row per security.
[2] a
[3] a
[4] a PARAMS[;1] par value
[5] a [;2] purchase cost excluding accrued interest (no coupon rec'd nor interest paid if purchased on a coupon date)
[8] a [;5] annual coupon rate
[9] a [;6] number of coupons per year
[10] a [;7] (optional) maturity value (par value if omitted)
[12] a
[13] a
[14] a
[15] a PAR=MVAL=PARAMS[;1]
[16] a COST=PARAMS[;2]
[17] a PDATE=PARAMS[;3]
[18] a MDATE=PARAMS[;4]
[20] a \rightarrow (6=1)pPARAMS)START
[21] a MVAL=PARAMS[;7]
[22] a
[23] a
[24] a
[25] a COST = ((1+Y)*(F)+(MVAL*(1+Y)*-W)+PAR*CRATE*1+
[26] a (1-(1+Y)*-W)/Y
[27] a
[28] a Where: \ Y = couponly yield rate
[29] a F = the fraction (0<F≤1) of a coupon period from PDATE to the next coupon
[30] a W = the number of whole coupon periods remaining from PDATE to MDATE (less 1 if purchased on a coupon date)
[31] a COST = purchase cost including accrued interest
[32] a
[33] a Convert the formula to a function in Y by moving COST to the right side:
[34] a
[35] a
[36] a f(Y) = (-COST)+((1+Y)*(F)+(MVAL*(1+Y)*-W)+PAR*CRATE*1+(1-(1+Y)*-W)/Y
[37] a
[38] a
[39] a The problem is to determine Y for which f(Y)=0. Solve by the method of successive approximations, using different values of Y. Start by trying Y=CRATE. Then use Newton-Raphson method to determine successive values of Y:
[40] a
[41] a
[42] a
[43] a
[44] a
[45] a
[46] a
[47] a
[48] a \ Y(N+1) = Y(N)-f(Y(N))+f'(Y(N))
[49] a
FCYIELD (continued)

[50] a Determine $f'(Y)$ by taking the derivative of $f(Y)$.
[51] a After tedious computations:
[52] a
[53] a $f'(Y) = ((1+Y)^{-F})^x(((1+Y)^{-L-W})^x(W+F)^x(PAR^x$
[54] a $CRATE+Y-MVAL) + (PAR^xCRATE^x1+Y)\div Y^x2)^-1$
[55] a $PAR^xCRATE^x(F+Y)^x1+Y^x2$
[56] a
[57] a Let us define the following:
[58] a
[59] a $F1=1+Y$ $F2=(1+Y)^{-F}$ $F3=(1+Y)^{-W}$ $COUP=PAR^xCRATE$
[60] a $N=F+W$ $RY=\div Y$ $CRY=COUP^xY$ $MCOST=-COST$
[61] a
[62] a The formulas for $f(Y)$ and $f'(Y)$ become:
[63] a
[64] a $f(Y) = MCOST+F2\times(MVAL\times F3)+COUP+CRI\times 1-F3$
[65] a
[66] a $f'(Y) = F2\times((F3\div F1)\times(N\times CRY-MVAL)+F1\times CRY\times RY)$
[67] a $-CRY\times F+RY$
[68] a
[69] a Compute approx days (360 days/yr) from purchase to maturity (change 31 days to 30):
[70] a
[71] a START:DAYS+ 360 30 1 +.x 0 100 100 100 MDATE-31=100\div MDATE
[72] a $DAYS=360 30 1 +.x 0 100 100 100 PDATE-31=100\div PDATE$
[73] a
[74] a The no. coupon periods from purchase to maturity:
[75] a
[76] a $N=\div(DAYS\times NCOUPS)=\div 360$
[77] a
[78] a Fractional and whole coupons from purchase to maturity:
[79] a
[80] a $F=N-W\div N^x1$
[81] a
[82] a Include accrued interest (prorated) in purchase cost:
[83] a
[84] a $COST=COST+COUP^x1-F$
[85] a
[86] a Start with couponly rates from approximate yield:
[87] a
[88] a
[89] a $YLD=\div(MVAL+MCOST+N\times COUP)+(MVAL\times N)+((N+1)\times COST-MVAL)\div 2$
[90] a
[91] a Indices into YLD of yields not yet known:
[92] a
[93] a $IND=\div YLD$
[94] a
[95] a Number of next iteration:
[96] a
[97] a
[98] a LOOP:F1=1+Y
[99] a
[100] a $F2=F1^x-F$
[101] a
[102] a $F3=F1^x-W$
[103] a
[104] a $CRY=COUP^xRY^x1+Y$
[105] a
[106] a Apply Newton-Raphson to get new $Y$:
[107] a
[108] a $DIFF=(MCOST+F2\times(MVAL\times F3)+COUP+CRI\times 1-F3)+F2\times((F3\div F1)\times(N\times CRY-MVAL)+F1\times CRY\times RY)-CRY\times F+RY$
[109] a
[110] a $NEW=Y-DIFF$
[111] a
[112] a Flag those found (changed less than .0000001):
[113] a
[114] a $OK=1E-7=\div DIFF$
[115] a
[116] a Compute indices of remaining elts:
[117] a
[118] a $MORE=\div(\div OK)/\div OK$
[119] a
[120] a Branch if no elts found:
[121] a
We will use the two approaches to solve the problem stated at the beginning of this section.

Approach 1: Using IROR

(note: purchase price includes $226.67 accrued interest)

\[
\text{DATES} \leftarrow 19870217, (10000 \times 1986 + 116) + 401 \times 1001, 20030401 \\
\text{AMTS} \leftarrow 4451.67, 320300, 5300 \\
\text{DATES IROR AMTS} \\
0.150274
\]

Approach 2: Using FCYIELD

\[
\text{Y} \leftarrow \text{FCYIELD} 1 605000 4225 19870217 20030401 .12 2 \\
2 \text{ EFFECTIVE} 2 \times Y \\
0.150305
\]

The slight difference between these two yield rates is the result of using an exact-days assumption (and 365 days per year) in the first approach, and a 30-days-per-month, regular annuity assumption in the second approach.
Chapter 18

PROBLEMS: (Solutions on pages 476 to 478)

1. You deposit $1000 in an 18 month Certificate of Deposit at 11% compounded daily. What will be the value of your deposit when it matures?

2. You make a deposit of $10 each week for 40 weeks into your bank's Christmas Club plan. Your bank pays 8%, converted monthly. How much will you have at the end of the 40 weeks?

3. You borrow $10,000 from your bank to buy a car. The term of the loan is 4 years and the interest rate is 14%. What will your monthly payment be? If you decide to repay the loan after 3 years, how much must you pay the bank (assuming no prepayment penalty)? How much interest will you have paid during the 3 years?

4. Your brother-in-law is opening a new restaurant and has approached you with the opportunity to invest in his venture. He is asking for an immediate outlay of $10,000 and a second outlay of $5,000 in 6 months. Starting 5 years from now, he will pay you $3,000 a year for 15 years. What is the internal rate of return of this investment (assuming all payments will be made as scheduled)?

5. When the purchase price of a bond is less than its par (face) value, the bond is said to be selling at a "discount". When its price is greater than its par value, it is selling at a "premium". In general, a bond sells at a discount when interest rates are higher than the bond's coupon rate, and sells at a premium when interest rates are lower than the bond's coupon rate. Fluctuating interest rates hence cause inverse fluctuations in the market values (purchase costs) of bonds. When interest rates are up, bond prices are down and vice versa. As the maturity date of a bond gets nearer, the fluctuations of its market value are less pronounced. On the maturity date, the market value of the bond is exactly equal to its par value, regardless of prevailing interest rates.

When you buy a bond, its value on your books (its "book value") is the purchase cost. When the bond matures, its value on your
books is the par value. These two values are not the same when you buy the bond at a discount or a premium. Since the book value of the bond changes from the purchase date to the maturity date, and since you do not want the change to appear as an abrupt change at maturity, you must "amortize" the amount of the discount or premium over time, modifying the book value of the bond accordingly.

In general, the book value of a bond on a given date is computed by using its yield-to-maturity and the present value formula in the FCYIELD function to determine the present value of the bond on the coupon dates before and after the given date and by interpolating between the two dates.

Write a function FCBOOK (fixed coupon book value) which returns the book values of a specified portfolio of bonds as of a specified date. The left argument of FCBOOK is the scalar date (YYYYMMDD) as of when the book values are to be computed. The right argument is a matrix of bond parameters with one row per bond. The 6 columns contain, respectively, par value, maturity date (YYYYMMDD), annual coupon rate, number of coupons per year, coupon only yield rate (as from FCYIELD), maturity value (par value if omitted). The result is a vector of book values with one element per bond (row of the right argument).
In this chapter we discuss the concepts of exception handling in APL, and illustrate exception handling techniques on some of the popular implementations of APL.

An exception is an event which, if not handled, will cause a function to suspend. Specifically, it is an error or an attention (pressing the BREAK key). When an exception occurs and is not handled, diagnostic information displays and the user is left in immediate execution mode. In other words, the function being executed no longer has control of what will happen. For better or worse, the user must decide what action to take next.

The concept of exception handling is that facilities are provided to enable the programmer to insert code into a system which will detect an exception when it occurs and will take some action other than simple suspension.

What kinds of action is the function (i.e. the programmer) likely to take when handling an exception? There are 4 typical choices:

1. Do something and then return to immediate execution mode with the function suspended on the exception line. This is the default (unhandled) behavior where the "something" is to display diagnostic information.

2. Do something and then resume execution at the exception line. Hopefully, the "something" (e.g. allocating more disk storage) removes the cause of the exception so that the line will complete without exception this time.

3. Branch to another line of the exception function where special logic has been included to evaluate the exception and to take appropriate action. After taking such action, the function may choose to branch back to the exception line or to branch elsewhere.

4. Leave the exception function altogether by signalling an error (i.e. exception) to the environment which called the exception function. This is the behavior taken by primitive APL functions. For example, if you attempt to divide by 0, the divide primitive (÷) does not suspend within its assembler code
but rather signals an error (DOMAIN ERROR) to the function line on which divide was called:

```
DOMAIN ERROR
CALC[15]  A\B\C
^
```

Likewise, it may be desirable to have the exception function (e.g. SQRT) signal an error to its calling environment:

```
DOMAIN ERROR
CALC[25]  A\SQRT B
^
```

The calling environment may then handle or not handle the exception as appropriate.

There are a number of different implementations of exception handling in APL. Each of these implementations takes a different approach to allow the 4 choices above. We will illustrate the 3 major exception handling implementations (APL*PLUS, SHARP APL, APL2) in this chapter.

---

**PROBLEM:** Given a character vector named INPUT which contains a user-entered APL expression, execute the expression and return its result. If the execution of the expression (or its assignment to the result variable R) generates an error, report the error and reprompt by branching to the line labeled PROMPT.

**TOPIC:** Detecting the Error

In APL*PLUS, the system variable DELX (error latent expression) may be assigned any executable character vector (as with DLX). The character vector is executed only if an error occurs. The niladic system function ODM (diagnostic message) returns a character vector representation (with embedded newline, i.e. carriage return, characters) of the diagnostic message of the most recent exception. The default setting of DELX in a clear workspace is 'ODM'.
The solution to the problem using APL*PLUS:

```apl
[0] ...;DELX
[1] DELX←'ODM'
    If an error occurs on lines 2 to 14, display the diagnostic message and suspend.
 [15] DELX←'ERR'
 [16] `R⁺',INPUT
 [17] DELX←'ODM'
 [25] ERR:O←ODM
    Display diagnostic message.
 [26] →PROMPT
    Ask again.
```

In SHARP APL, the system variable DTRAP may be assigned a delimited character vector (say, delimited by the 'V' character) or a character matrix where each partition or row specifies the action to be taken for a given class of errors. For example, the expression

```
DTRAP←'V0 E →ERR'
```

says to execute (E) the expression →ERR if any (0) error occurs.

The system variable DER (event report) contains a 3-row character matrix representation of the most recent exception. The first 5 characters of the first row of DER contain the "event number" (e.g. 4 for RANK ERROR). The default setting of DTRAP in a clear workspace is ''.

The solution to the problem using SHARP APL:

```apl
[0] ...;DTRAP
[1] DTRAP←'
    Normal message and suspension if error on lines 2 to 14.
 [15] DTRAP←'V 0 E →ERR'
 [16] `R⁺',INPUT
 [17] DTRAP←'
 [25] ERR:O←5OFDER[]IO;
    Show error message but not event number. Show rest of diagnostic
 [26] O←1 O←DER
    message. Ask again.
 [27] →PROMPT
```

In APL2, the dyadic system function DEA (execute alternate) takes an executable expression as its character vector right argument and executes it ala `*. If the expression can be executed without exception, the left argument of DEA is not considered. However, if the execution of the right argument of DEA generates an exception, the left argument is executed. The system variable OEM (event message) contains a 3-row character matrix representation of the diagnostic message of the most recent exception having occurred at the current level of the state indicator.
The solution to the problem using APL2:

```apl
[16] '→ERR' OEA 'R←',INPUT Branch to ERR if an error during #.
    :
[26] →PROMPT Ask again.
```

A different APL2 solution is possible by using the monadic system function DEC (execute controlled). DEC is like OEA in that its character vector right argument is an executable expression. However, the result of DEC is a 3 item nested array which contains information about the attempt to execute the right argument. The result can be evaluated to determine whether or not an error has occurred. DEC distinguishes among the various types of executable expressions and so allows messages which are more specific than those possible from OEA.

See an APL2 reference manual for more complete documentation on DEC. Here is the solution using DEC in APL2:

```apl
[16] Z←DEC INPUT
[17] →ERR UNLESS Z[1]≡1 2 Branch unless result or
[18] R←3≥Z assignment.
    :
[25] ERR:→(Z[1]=0 3 4 5)/ERR1,ERR2,ERR3,ERR3
[26] ERR1:O←OEM
[27] →PROMPT
[28] ERR2:O←'EXPRESSION HAS NO RESULT'
[29] →PROMPT
[30] ERR3:O←'BRANCHING NOT ALLOWED'
[31] →PROMPT
```

PROBLEM: Suppose you have written a dyadic function STAT whose arguments are same-length numeric vectors and whose result is a numeric vector of various statistics. The function is self-contained, i.e. it requires no subfunctions or global variables and it assigns no global variables. Because of its syntax and self-containment, STAT behaves like a dyadic primitive function (once copied into the workspace) in all regards but one: error handling. When provided with faulty arguments, STAT suspends on one of its lines after displaying one of: RANK ERROR, LENGTH ERROR or DOMAIN ERROR. Rewrite STAT to signal these errors and others (say, WS FULL) to its calling environment rather than suspending.
TOPIC: Signalling the Error

In APL*PLUS, the monadic system function DERROR takes a character vector error message argument and signals that message to the environment from which was called the function in which DERROR is executed. For example:

```
V TEST N
[1] DERROR 'OOPS'
[2] V
TEST 5
OOPS
TEST 5
^`
```

Once the error occurs within STAT, the message may be found as the first line of the result of DDM. The first line is defined as all characters up to the first newline character (represented in APL*PLUS as OTCNL).

The solution to the problem using APL*PLUS:

```
[0] ...;DELX
[1] DELX←'ERROR(^\DDM#OTCNL)/\DDM'
```

In SHARP APL, the dyadic system function DSIGNAL takes an optional character vector error message left argument and a corresponding event number right argument and signals the message (and event number as the first 5 characters of DER) to the environment from which was called the function in which DSIGNAL is executed. For example:

```
V TEST N
[1] 'OOPS' DSIGNAL 599
[2] V
TEST 5
OOPS
TEST 5
^`
```

The function DSIGNAL may also be used monadically. Its argument is an integer event number in the range 1 to 999 (e.g. 2 for the standard APL error message SYNTAX ERROR).

The solution to the problem using SHARP APL:

```
[0] ...;DTRAP
[1] DTRAP←'V0 E DSIGNAL$5pDER'
```

or, using DEC (environment condition):

```
[0] ...;DEC
[1] DEC←1 ⍺ Disallow suspension
```
Chapter 19

EXCEPTION HANDLING

In APL2, the monadic system function OES (event simulation) takes a character vector error message argument or a 2-element integer vector event type code (e.g. 1 3 for WS FULL) and signals the message to the environment from which was called the function in which OES is executed. For example:

\[
\begin{align*}
\text{V} & \text{TEST N} \\
[1] & \text{OES 'OOPS'} \\
[2] & \text{V} \\
\text{TEST 5} \\
\text{OOPS} \\
\text{TEST 5} \\
\wedge
\end{align*}
\]

The system variable DET (event type) contains the 2-element integer vector event type code of the most recent exception having occurred at the current level of the state indicator.

The solution to the problem using APL2:

\[
\begin{align*}
[1] & \text{'OES DET' DEA 'line 1 of STAT'} \\
[2] & \text{'OES DET' DEA 'line 2 of STAT'} \\
[3] & \text{'OES DET' DEA 'line 3 of STAT'} \\
\vdots & \text{\vdots} \\
\vdots & \text{\vdots}
\end{align*}
\]

If self-containment were not an issue, the solution using APL2 would be to rename STAT to STAT1 and to write a new STAT function:

\[
\begin{align*}
\text{V} & \text{R+A STAT B} \\
[1] & \text{'OES DET' DEA 'R+A STAT1 B'} \\
\text{V}
\end{align*}
\]

PROBLEM: Given a 70-line function of critical code named PROCESS, incorporate exception handling such that an interrupt (the BREAK key) will cause the message PLEASE BE PATIENT to display and execution to resume.

TOPIC: Detecting the Attention

In APL*PLUS, the system variable OALX (attention latent expression) may be assigned any executable character vector (as with DLX or DELX). The character vector is executed only if an attention is signalled (via the BREAK key). The default setting of OALX in a
clear workspace is 'ODM'. That is, the default behavior of the system is to display the diagnostic message generated by the attention and to suspend.

The niladic system function OLC (line counter) returns an integer vector of the numbers of the suspended or pendent lines in the state indicator. The first number corresponds to the top (most recent) level in the state indicator and the last number corresponds to the bottom level. Since the branch primitive function (→) only considers the first element of its argument, the expression →OLC will cause the flow of execution to proceed to the line on which the expression was executed.

The solution to the problem using APL*PLUS (⋄ is a statement separator):

```
[0] ...⋄ALX
[1] ⋄ALX←'⋄←''PLEASE BE PATIENT'' ⋄ →OLC'
```

In SHARP APL, the event number 1000 represents any interrupt. The solution using SHARP APL:

```
[0] ...⋄TRAP
[1] ⋄TRAP←'V1000 E ⋄←''PLEASE BE PATIENT'' ⋄ →OLC'
```

In APL2, an attention is not considered an exception which can be handled. Therefore, this problem as stated cannot be solved using APL2. If the PLEASE BE PATIENT message is omitted from the problem, you may solve the problem in APL2 by "conditioning" the PROCESS function to not be interruptable:

```
0 0 1 0 ⋄FX ⋄CR 'PROCESS'
```

---

PROBLEM: You wish to install a fully-tested production system such that if any unexpected (i.e. unhandled) error occurs, the message "AN UNEXPECTED ERROR HAS OCCURRED; CONTACT J. SMITH IMMEDIATELY" will display and the system will suspend on the exact line of the error. How will you modify the system to do so?

TOPIC: Suspending the Function

In APL*PLUS, the default behavior of the system when handling an exception is to suspend on the exception line. In fact, if ⋄ELX is
set to '', that is all that will happen. Since this behavior would be confusing to the user (suddenly in immediate execution mode with no indication of an exception), the default setting of DELX is 'DDM'. With this setting, the diagnostic message is returned and displayed and then the function is suspended. The only way to not suspend the function is to branch (e.g. DELX←'ENTER') so that execution may resume or to signal an error (e.g. DELX←'ERROR 'OOPS'' ) which may be handled by a more global DELX.

The solution to the problem using APL*PLUS is to assign DELX globally:

    DELX←''AN UNEXPECTED...IMMEDIATE''

If an error occurs, DELX will be executed causing the message to display. Then execution will suspend. When Smith arrives, he will type DDM to display the diagnostic message and may resume execution by typing →ULC after correcting the error.

If a function lacks exception handling but is a high security function, you may not want to allow the function to be left suspended after an exception. For example, a function which updates a payroll file may contain sensitive salary information in its local variables during execution. To prevent a function from suspending (i.e. to perform a → when entering immediate execution mode) in APL*PLUS, localize the system variable DSA (stop action) and assign it the value 'EXIT'. The default setting of DSA is ''.

In SHARP APL, if an exception is not handled by the current setting of DTRAP, the diagnostic message will display and the function will suspend on the exception line. As with APL*PLUS, the only way to not suspend the function is to branch (e.g. DTRAP←'VO E →ENTER') so that execution may resume or to signal an error (e.g. DTRAP←'VO E ''OOPS'' SIGNAL 599') which may be handled by a more global DTRAP.

The solution to the problem using SHARP APL is to assign DTRAP globally:

    DTRAP←'VO E D←'AN UNEXPECTED...IMMEDIATE''

If an error occurs, the message will display and execution will suspend. When Smith arrives, he will type DER to display the error report and may resume execution by typing →ULC after correcting the error.

To prevent a high security function from suspending in SHARP APL, include the partition 'V2001 D EXIT' in the current (local) definition of DTRAP. Event number 2001 represents the "immediate execution mode" event. D stands for "do".

In APL2, if an exception occurs outside the execution of the right argument to OEA (or OEC, a similar function), the diagnostic message will display and the function will suspend on the execution line. To keep the diagnostic message from displaying, each statement which may generate an error must be executed within the right argument of OEA.
or must be on a line of a function which is executed within the right argument of `OEA`, or must be on a line of a function called by a function called by `OEA`, and so on.

Therefore, if COVERFN is the name of a function to which all of the other functions in the system are subfunctions, you may control the display of the diagnostic message by invoking the system with an expression such as:

```
'HANDLER' `OEA 'COVERFN'
```

Unfortunately, if an unhandled exception occurs anywhere within COVERFN, a suspension will not occur. Rather, the levels of the state indicator associated with COVERFN will be reset and HANDLER will be executed. Hence, we can choose either to detect the error but lose the suspension or to suspend by not detecting the error. For example, the expression

```
'O⁺' 'AN UNEXPLAINED...IMMEDIATELY' `OEA 'COVERFN'
```

will cause the proper message to display. However, when Smith arrives, he will type OEM to display the event message and will find that it is empty and the state indicator is empty (unless an old suspension is lying around).

To prevent a high security function from suspending in APL2, you may "condition" the function (say, PAYROLL) to not be suspendable:

```
0 1 0 0 `OFX `OCR 'PAYROLL'
```

```
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~
```

**PROBLEM:** Given a 20-line function MAINFN which calls a multitude of subfunctions, you would like to incorporate exception handling in MAINFN such that any error (even if in a subfunction with no exception handling) causes a branch to the line labeled ERRCODE in MAINFN and any attention causes a branch to the line labeled ATTNCODE.

**TOPIC:** Controlling the State Indicator

In APL*PLUS, there is no direct capability for solving this problem. Suppose MAINFN calls SETUP which calls READVARS which encounters an error on its 5th line. If there is no exception handling, the READVARS function will suspend and the state indicator will look something like:
A naive solution to the problem would be to include the following in MAINFN:

```
\[1\] ∀ MAINFN:...,DALX;DELX
\[2\]   DALX←'ATTNCODE'
\[3\]   DELX←'ERRCODE'
   ...
\[41\]  ERRCODE:etc.
   ...
```

However, when the error in READVARS occurs, the value of `DELX ('ERRCODE')` will be executed, causing a branch to line 31 (the value of `ERRCODE` unless shadowed by a different `ERRCODE` local to SETUP or READVARS) of READVARS, not to line 31 of MAINFN. Somehow, we must get out of (i.e. \(→0\)) both READVARS and SETUP before branching to the line labeled `ERRCODE`. The only way to do this without landing in immediate execution mode (which would happen if you localized `ＤSＡ→EXIT` in both READVARS and SETUP) is to use `ＯERROR`.

The approach we will take is this: set `DELX` to check whether `DELX` has been localized at the current top level of the state indicator; if so (i.e. if in MAINFN), branch to `ERRCODE`; if not (i.e. if in SETUP or READVARS), use `ＯERROR` to reset the top level of the state indicator and to signal an error (say, the same error message) to the next level; since `ＯERROR` will trigger `DELX`, the process will be repeated until MAINFN is at the top of the state indicator. This process is called "propagating the error message".

The monadic system function `ＤＩＤＬＯＣ` (identifier localization) returns a 1 row integer matrix of localization codes for the identifier whose name is provided as the character vector right argument. Each column of the result corresponds to one level of the state indicator (local to global) and the code `-1` represents unlocalized. Therefore, if the result of `\(DIM[1]·IDLOC OECD\)` is `-1`, then `DELX` is not localized at the top level of the state indicator.

The initial solution to the problem using APL*PLUS:

```
\[1\] ∀ MAINFN:...,DELX
   `g(\(-1=DIM[1]·IDLOC OECD\))/(\'\'ERROR(^\DIM[1]TCNL)/DIM OECD)^\'→ERRCODE'
```

This handles errors. What about attentions? We will use a similar approach: set `DALX` to check whether `DALX` has been localized at the current top level of the state indicator; if so (i.e. if in MAINFN), branch to `ATTNCODE`; if not (i.e. if in SETUP or READVARS), use `ＯERROR` to reset the top level of the state indicator and to signal an error.
(say 'ATTN') to the next level; OELX will be triggered at the next level and will behave as described above except it will branch to ATTNCODE rather than ERRCODE if the error message is 'ATTN'.

The final solution to the problem using APL*PLUS:

\[ \text{V MAINFN}; \ldots; \text{DALX; OELX} \]
\[ [1] \text{DALX} \leftarrow \text{'( } \neg 1 \leftarrow \text{lpDIDLOC' } \text{DALX'} \text{)'/ERROR'''ATTN'''''} \rightleftharpoons \text{ATTNCODE} \]
\[ [2] \text{OELX} \leftarrow \text{'( } \neg 1 \leftarrow \text{lpDIDLOC' } \text{OELX'} \text{)'/ERROR(\text{\textbackslash}\text{\textbackslash}DM\#\text{\textbackslash}TCNL)/DM'} \rightleftharpoons (''\text{ATTN'} \text{.}=4\uparrow \text{DM}) \lands \text{ERRCODE, ATTNCODE} \]

In SHARP APL, there is a direct capability for solving this problem. The action code C (cut) can be specified within the TRAP definition to specify that the state indicator should be "cut back" (i.e. reset) to the level at which TRAP is local.

The solution to the problem using SHARP APL:

\[ \text{V MAINFN}; \ldots; \text{TRAP} \]
\[ [1] \text{TRAP} \leftarrow \text{'VO C } \land \text{ERRCODE V1000 C } \rightarrow \text{ATTNCODE'} \]

In APL2, the only way to solve the problem is to execute each of the 20 lines of MAINFN as the right argument of DEA. If an exception occurs during the execution of the line, the state indicator is automatically "cut back" so that MAINFN is at the top level; then the left argument of DEA is executed. Since attentions cannot be detected as exceptions in APL2, no branch to ATTNCODE is possible.

The solution to the problem using APL2:

\[ \text{V MAINFN}; \ldots; \text{ELX} \]
\[ [1] \text{ELX} \leftarrow \text{'ERRCODE'} \]
\[ [2] \text{ELX} \lands \text{DEA 'line 1 of MAINFN'} \]
\[ [3] \text{ELX} \lands \text{DEA 'line 2 of MAINFN'} \]
\[ [4] \text{ELX} \lands \text{DEA 'line 3 of MAINFN'} \]
\[ : : : \]
\[ [26] \text{ERRCODE}: \text{etc.} \]

The only problem with this solution is its brute force appearance. A typical APL2 solution to the problem involves restating the problem: rename MAINFN to be MAINFN1 and write a new MAINFN which will handle any exceptions occurring within MAINFN1.

Here is the alternative APL2 solution:

\[ \text{V MAINFN} \]
\[ [1] ' \land \text{ERRCODE'} \lands \text{DEA 'MAINFN1'} \]
\[ [2] 0 \]
\[ [3] \text{ERRCODE}: \text{etc.} \]
Chapter 19

---

PROBLEMS: (Solutions on pages 479 to 482)

1. The following function displays the records of a file. The user is asked for the number of the record at which displaying is to begin. Then the function reads and displays every record from the specified record to the end of the file. Incorporate exception handling so that the user may press the BREAK key to halt the display and to ask again for the number of the record at which displaying is to begin. Assume that the READ function uses exception handling such that an attention within READ causes READ to signal the error message 'ATTN' to its calling environment.

```
V SHOWFILE NAME;LIM;N
  [1] LIM←TIEFILE NAME
  [2] ASK:D←'BEGIN WITH WHICH RECORD?'
  [3] N←0
  [4] (0=N)/0
  [5] LOOP:D←''
  [7] D←'
  [8] (LIM≥N+N+1)/LOOP
```

2. Write a dyadic function NXPROMPTE which prompts for and returns a vector of numbers. Its right argument PROMPT is a character vector which is displayed to prompt the user for input. Input is accepted beyond the prompt on the same line. The left argument is the number of numbers required. If 0, the function will allow any number of numbers. If the word END is entered, the result is the scalar 1. Any primitive APL expression may be entered and will be executed. User defined functions or variables may not be included in the response. Note that this function is identical to the NPROMPTE function developed in the chapter, Writing User-Friendly Interactive Functions, except APL expressions are allowed. For example:

```
Q←0 NXPROMPT 'ENTER PROJECT NO.S: '
ENTER PROJECT NO.S: 10+18
Q
11 12 13 14 15 16 17 18
```
3. Write a dyadic function ERRATTN to initialize the exception handling facilities such that any subsequent error will be handled as directed by its arguments. If the left argument, ELX, is a character vector, it will be executed directly if any error should occur; if the left argument is a numeric line label, the state indicator will be reset (cut back) to the level at which ERRATTN is called and a branch to the scalar will take place. The meaning of the right argument, ALX, is the same as that of the left argument but is for attention handling rather than error handling. If either argument is empty (e.g. ' '), the response to the respective exception is to display the diagnostic message and suspend. For example:

```
[1] ' ERRATTN ' :
[6] 'FIXFILE $\rightarrow$LC' ERRATTN L5
[7] L4:APPEND DATA
[8] $\rightarrow$MORE
[9] L5:$\leftarrow$'FILE INCOMPLETE DUE TO INTERRUPT.' :
```

On line 1, the error and attention handling is set to its default behavior (if an exception, display diagnostic message and suspend). On line 6, a character vector error handler and a numeric line label attention handler are provided. If an error occurs (say, within the APPEND function), the FIXFILE function is executed and execution is resumed. If an attention occurs, execution is resumed on the line labeled L5 (even if the error occurs within the APPEND function).
The functions described and listed in this book are available on a floppy disk for the APL*PLUS PC system. To order one or more of these floppy disks, please send your check or money order to:

ADVANCED APL FUNCTIONS
ZARK INCORPORATED
53 SHENIPSIT STREET
VERNON, CT 06066

Specify the number of disks desired and enclose $15 per disk. Postage and handling charges are included. Connecticut residents, please include sales tax.

The following is a list of the workspaces and functions included on the disk.

)`WSID BOOLEAN
pANDMAP pANDRED pANDSCAN pEQMAP pEQSCAN pGEMAP pGTMAP
pLESCAN pLTSCAN pNEMAP pNESCAN pORMAP pORRED pORSCAN
pPLUSRED

)`WSID CASHBAL
CASH1 CASH2 CASH3 CASH4

)`WSID CNFNS
ASSIGN CNCAT CNEQ CNEST CNGRADEUP CNIDX
CNIDXA CNIOTA CNLEN CNAM CNAV

)`WSID COMMENTS
UNCOMMENT UNLAMP

)`WSID CRTIMING
TIMER TIMEDEFINE
)WSID DATES
FROMDAYS FROMDAYS360 FROMDAYSΔ FROMMDY FROMMDYΔ
FROMQTS FROMMYD IPDatemdy TODAYS TODAYS360
TODAYSΔ TOMDY TOMDYΔ TOQTS TOYD

)WSID ERROR
ERRATTNP ERRATTNS

)WSID FILEDOC
FILEDOC

)WSID FLF
EMPLOYEES

)WSID FNIDS
IDENTIFY LOCALIZE OBFUSCATE RELABEL UNDIAMOND
UNOBFUSCATE

)WSID FNREP
CRAVR VRΔCR

)WSID FNSFILE
DROPFN FNCREATE GETFN PUTFN

)WSID FORMAT
CENTER CJUST COLFMT COLUMNIZE DEB DLB
DTB HEADINGS LJUST RJUST ROWFMT THORN
TITLES

)WSID INPUT
CPROMPT CPROMPT2 ESCAPE IF LPROMPT LPROMPT2
MESSAGE NINPUT NINPUT2 NPROMPT NPROMPT2 NPROMPTE
NPROMPT2E2 NXPROMPTE PROPOSAL UNLESS

)WSID INTEREST
EFFECTIVE FCBOOK FCYIELD IROR NOMINAL SCHEDULE
VALUE

)WSID LOOP
LOOPI NEXTI
)`WSID SORT
CGRADEUP  CGRADEUP1  CGRADEUP2  UPPERCASE

)`WSID TIMING
COST   TIMER   TIMEDEFINE   TIMEDISPLAY   TIMERESET
A

)`WSID USEDBY
USEDBY

)`WSID UTILITY
MONIOTA  REPL

)`WSID WP
WRAP   WRAPLP

)`WSID WSDOC
WSDOC
LIMBERING UP

1. \[\text{AMOUNT[AMOUNT=645]} \leftrightarrow 845\] (if exactly one occurrence of 645)
   or \[\text{AMOUNT[(AMOUNT=645)/AMOUNT]} \leftrightarrow 845\] (if 0 or more occurrences)
   or \[\text{((AMOUNT=645)/AMOUNT)} \leftrightarrow 845\] (APL2)

2. \[\wedge/(\text{PREMS}>100)\wedge\text{PREMS}<500\]

3. \[+/24=\text{0.5+WEIGHT}\]

4. \[\times/\rho\text{MAT}\quad \text{or} \quad \rho,\text{MAT}\]

5. \[\text{''ANSWER IS ',(\$ANS),' YEARS'}\]

6. \[\text{NAMES} \leftrightarrow \text{NAMES,}[1](1\downarrow\rho\text{NAMES})\uparrow\text{NAME}\] (origin 1)

7. A. Since a scalar has no shape, its shape is an empty vector. Therefore, the result of \(\rho12\) is an empty vector. When the right argument of branch (\(\rightarrow\)) is an empty vector, no branch takes place. Control proceeds to the next statement.
8. \[ R^+((\rho V)\rho 0)/V)+256\times((\rho V)\rho 0 1)/V \]
   or \[ R^+256\Phi(N,2)\rho V \]
   or \[ R^+((N,2)\rho V)+\times 1 256 \]
   or \[ R^+/(N,2)\rho V\times(\rho V)\rho 1 256 \]

9. The reduction of an empty vector returns the identity element for the dyadic function involved in the reduction. The identity element is that value which when supplied as one of the arguments of the dyadic function will return the other argument. For example, since \(0+5\) is 5 and \(1\times5\) is 5, the identity element for plus (+) is 0 and for times (\(\times\)) is 1. The only argument to minimum (\(\text{l}\)) which will always return the other argument is positive infinity. Since positive infinity cannot be represented as a number, APL returns the next best thing, the largest possible number which can be represented on the computer. This value varies from implementation to implementation but is generally some value like \(1E77\) or \(2E300\). For non-commutative functions (like \(\div\)), the identity element is that value which when supplied as the right argument of the dyadic function will return the left argument. For example, since \(5\div1\) is 5, the identity element for divide is 1.

10. The running alternating sum.

   \[
   \begin{array}{cccc}
   - \backslash 3 & 8 & 6 & 5 \\
   3 & -5 & 1 & -4
   \end{array}
   \]

   \(3,(3+(-8)),(3+(-8)+6),(3+(-8)+6+(-5))\)

11. Display the state indicator (via "\text{}SI\text{}") to see where the suspension occurred and branch to the line number shown on the top of the state indicator. Alternately:

   \(\rightarrow\text{DLC}\)

12. Most implementations of APL insist that the header of a function not be changed once the function is called. It is too tricky to handle the problems which arise when you make a global variable local, or vice versa, while the function is suspended. Likewise, labels may not be added or deleted (or possibly moved to different lines) in a suspended function, since the values of
labels are assigned at the moment the function is called. Most implementations of APL will allow you to make such editing changes to a suspended function but will display a message like SI DAMAGED or SI ERROR or SI WARNING and will not allow you to resume execution at the point of the error. You will need to reset the state indicator and rerun the function from the beginning.

13. The "rank" or "ranking" of VECTOR. For example, if 6 students have these test scores,

```
SCORES←87 99 83 85 65 100
```

the ranks of the respective students are:

```
▲▲SCORES
4 5 2 3 1 6
```

where the 1 corresponds to the student with the lowest score and the 6 corresponds to the student with the highest score.

14.a. \((-1↓ρA),1↓ρB\)

b. ditto

15.a. \((ρA),ρB\)

b. ditto

16. □AI (accounting information), a niladic system function. Usually the second element of the result of □AI is a measure of CPU time consumed since signon to APL.

17. □PROMPT
R←(ρPROMPT)↓□
(or: □ARBOUR t0 in APL*PLUS)
R+□
18.a. \((N,1)p'\)  
(or: \((N,0)p'\) in APL2)

b. \((N-1)p\)UTCNL \((\text{APL*PLUS})\)  
\((N-1)p\)OAV\{156+\|O\} \((\text{SHARP APL})\)  
\((N-1)p\)UTC\{1+\|O\} \((\text{APL2})\)

Note: None of these three expressions works correctly if \(N=0\). They generate a DOMAIN ERROR.

19. \&EX \&NL 2 \((\text{APL*PLUS, SHARP APL, APL2})\)  
\&ERASE \&DIDLIST 2 \((\text{APL*PLUS})\)  
6 \&FD 1 \&WS 2 \((\text{SHARP APL})\)

20.a. S\&MODEL\&-12 14 \((\text{APL*PLUS, SHARP APL, APL2})\)  
12 14 \&DSTOP 'MODEL' \((\text{APL*PLUS PC, APL*PLUS UNX})\)

b. Localize \(T\) in INTERPOLATE. \(T\) was somehow reassigned after line 11 and before line 13. Since the reassignment does not take place on line 12, it must take place within INTERPOLATE (or a subfunction of INTERPOLATE).

21. Including the header in the count of lines,

\(1\|p\)OCR 'CALC' \((\text{APL*PLUS, APL2})\)  
\(1\|p\)2 \&FD 'CALC' \((\text{SHARP APL})\)  
\(\sim\)1++/\)UTCNL=\)|VR 'CALC' \((\text{APL*PLUS})\)  
\(\sim\)1++/\)OAV\{156+\|O\}=1 \&FD 'CALC' \((\text{SHARP APL})\)

22. Because \(\|O=0\) and some elements of \(V2\) are not found in \(V1\).
1. The expression does not work correctly in index origin 0; it modifies the random link (DRL); and it labels the programmer as having brain damage.

2. \[ (\times N) \neq \text{ZERO, POSITIVE, NEGATIVE} \]
   or \[ (\text{NEGATIVE, ZERO, POSITIVE}) [2+\times N] \]
   or \[ (\neg 1 \ 0 \ 1=\times N)/\text{NEGATIVE, ZERO, POSITIVE} \]

3. a. 
   
   SUM$\leftarrow$0
   I$\leftarrow$11
   LOOP: $\rightarrow$ENDLOOP IF I$>$308
   SUM$\leftarrow$SUM$+$READ I
   I$\leftarrow$I+3
   $\rightarrow$LOOP
   ENDLOOP:

   b. 
   
   SUM$\leftarrow$0
   CMPS$\leftarrow$8+3$\times$I+100
   I$\leftarrow$I+1
   LAB$\leftarrow$(100$\times$LOOP), ENDLOOP
   LOOP: SUM$\leftarrow$SUM+READ CMPS[I]
   I$\leftarrow$I+1
   $\rightarrow$LAB[I]
   ENDLOOP:

   c. 
   
   SUM$\leftarrow$0
   I$\leftarrow$ENDLOOP, 100 11 3
   SUM$\leftarrow$SUM+READ I
   $\rightarrow$I
   ENDLOOP:

   d. 
   
   SUM$\leftarrow$0
   $\rightarrow$LOOPI ENDLOOP, 100 11 3
   SUM$\leftarrow$SUM+READ I
   $\rightarrow$NEXTI
   ENDLOOP:

   e. 
   
   SUM$\leftarrow$0
   where: $\neg$ SUMUP CMP
   SUMUP$''$ 8+3$\times$I+100
   [1] SUM$\leftarrow$SUM+READ CMP
   $\neg$
4. \( R\text{IO} = 1 \\
R\text{SCAN} + (1 + \text{RATE}) \times \text{TERM} \\
O\text{PRIN} + R\text{SCAN} \times \text{LOAN} - + \text{PMT} + R\text{SCAN} \)

5. After performing the transformations, the result is DEPOSIT. Undoing the transformations yields the following function:

\[
\text{BALANCE} \times \text{RATE} \text{CASH4 DEPOSIT} \times \text{ACCUM}
\]

1. Returns stream of cash balances for deposits
2. DEPOSIT and corresponding rates RATE.
3. Performs:
4. \( \text{BALANCE}[I] + \text{DEPOSIT}[I] + \text{BALANCE}[I-1] \times \text{RATE}[I-1] + 1 \)
5. \( \text{ACCUM} + \Phi 	imes 1, 1 + \text{RATE} \)
6. \( \text{BALANCE} + (+ \times \text{DEPOSIT} \times \text{ACCUM}) / \text{ACCUM} \)

Note to actuaries: The approach taken here is to compute the future value of each deposit as of the last period, subtotal the future values, and then discount each subtotal back to the deposit date. Alternately, the approach in CASH3 is to compute the present value of each deposit as of the start of the first period, subtotal the present values, and then accumulate each subtotal back to the deposit date.

6. Notice that the WRAP function below will iterate only as many times as the number of lines generated by the longest sentence.

\[
\text{WRAP} \text{CVEC}; \text{IO} \times \text{BL}; \text{BREAK}; \text{LAST}; \text{LEN}; \text{MORE}; \text{NL}; \text{START}; \text{TCNL}
\]

1. Wraps text CVEC into lines of length WID
2. or less by inserting newline characters.
3. Origin 1:
4. \( \text{IO} \times 1 \)
5. Newline character:
6. \( \text{TCNL} + \text{UTCNL} \times \text{APL} \times \text{PLUS} \)
7. \( \text{TCNL} + \text{UTC}[2] \times \text{APL} \times \text{2} \)
8. \( \text{TCNL} + \text{OAV}[157] \times \text{SHARP} \times \text{APL} \)
9. Flag newline characters:
10. \( \text{NL} + \text{CVEC} = \text{TCNL} \)
11. Index before start of each sentence:
12. \( \text{START} + 0, \text{NL} / \text{\rho} \text{NL} \)
13. Lengths of sentences (between newlines):
14. \( \text{LEN} - 1 + (1 \times \text{START}, 1 + \text{\rho CVEC}) - \text{START} \)
\begin{verbatim}
\textbf{WRAP (continued)}
\[15\text{ A Flag valid break points (blank followed by nonblank):}\]
\[16\text{ BL$+$CVEC='}\]
\[17\text{ BREAK$+$BL}$>$1$\Phi BL\]
\[18\text{ A Initialize result from argument:}\]
\[19\text{ R$+$CVEC}\]
\[20\text{ A Flag sentences still to be broken:}\]
\[21\text{ LOOP$+$MORE$+$LEN}$>$WID\]
\[22\text{ A Select just those remaining:}\]
\[23\text{ LEN$+$MORE/LEN}\]
\[24\text{ A Exit if none left:}\]
\[25\text{ (O$=$LEN)/O}\]
\[26\text{ START$+$MORE/START}\]
\[27\text{ A Find last break point within WID chars of line:}\]
\[28\text{ LAST$+$/\BREAK[START$+$WID]}\]
\[29\text{ A Advance start to new break point:}\]
\[30\text{ START$+$START$+$LAST}\]
\[31\text{ A Insert newlines:}\]
\[32\text{ R[START]$+$TCNL}\]
\[33\text{ A Decrement remaining lengths:}\]
\[34\text{ LEN$+$LEN-LAST}\]
\[35\text{ A Repeat:}\]
\[36\text{ LOOP}\]
\end{verbatim}
Chapter 3 Solutions

COMPUTER EFFICIENCY CONSIDERATIONS

1.  [WSID: TIMING]

\[ \text{\n\[1\] \n\[2\] \n\[3\] \n\[4\] \n\[5\] \n\[6\] \n\[7\] \n\[8\] \n\[9\] \n\[10\] \n\[11\] \n\[12\] \n\[13\] \n\[14\] \n\[15\] \n\[16\] \n}\]

2.  [WSID: CRTIMING]

\[ \text{\n\[1\] \n\[2\] \n\[3\] \n\[4\] \n\[5\] \n\[6\] \n\[7\] \n\[8\] \n\[9\] \n\[10\] \n\[11\] \n\[12\] \n\[13\] \n\[14\] \n\[15\] \n\[16\] \n}\]
Chapter 3 Solutions

\[ \text{COMPUTER EFFICIENCY CONSIDERATIONS} \]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]

\[ \text{\n\n\n\n\n\n\n}\]
Change CMIOTA as instructed (to activate the looping algorithm) and run DOIT again:

```
\v CMIOTA
(edit it)
```

\v

```
T2+L DOIT R
```

The variables L, R, T1 and T2 will be needed in a problem at the end of the Curve Fitting chapter. Record and save them:

```
)SAVE CMTIMES
```
Chapter 4 Solutions

POSITIONING CHARACTER DATA

1. \[ R \leftarrow (\backslash \text{TEXT}\#\backslash \text{NL})/\text{TEXT} \]
   or \[ R \leftarrow ((\text{TEXT}\#\backslash \text{NL})-\text{DIO})\rho \text{TEXT} \]
   or \[ R \leftarrow (+/\backslash \text{TEXT}\#\backslash \text{NL})\rho \text{TEXT} \]

2. \[ R \leftarrow (\text{CODE}=\text{''/'})\backslash \text{''}=1\downarrow \text{CODE},\text{''/''} \]
   (What is wrong with: \[ R \leftarrow (\text{CODE}=\text{''/'})\backslash \text{''}=1\uparrow \text{CODE} \] ?)
   (Try it on: \[ \text{CODE}+\text{''I}B/\text{''}=\text{USES} /\] )
   \[ R \leftarrow \text{CODE} \leftarrow \text{DSS} \text{''/''} \] (APL*PLUS)
   \[ R \leftarrow \text{''/''} \leftarrow \text{CODE} \] (APL2)

3. \[ \forall R \leftarrow W \text{ CENTER C} \]
   [1] \& Pads character vector C to width W, centering
   [2] \& it within that width.
   [3] \[ R \leftarrow W \uparrow (((0\downarrow W-\rho,C)+2)\rho ' ' )\],C
   [4] \& Alternative:
   [5] \[ R \leftarrow W \uparrow (-1\downarrow (W+\rho,C)+2)\uparrow C \]
   \[ \forall \]
4. [WSID: FORMAT]

```
\[ \text{R} \leftarrow \text{D} \ \text{ROWFMT} \ N \]
```

1. Formats a numeric matrix N into a character matrix.
2. A D is an integer vector with one element per row.
3. A of N. The integers indicate the number of decimal places, for each numeric row, to be displayed in the character matrix result. Each number is formatted in a width of <width> characters (e.g. 10), where <width> is an integer scalar global variable.
4. Requires subfunction: COLFMT

5. [WSID: FORMAT]

```
\[ \text{R} \leftarrow \text{RC} \ \text{COLUMNIZE} \ \text{CMAT};\text{C};\text{COLS};\text{PGS};\text{ROWS};\text{DIO} \]
```

1. Restructures skinny character matrix CMAT into a fat one. RC is scalar or 1 or 2 element vector. Last element is no. of "columns" of CMAT running down each page of the result.
2. If RC has 1 element, result is a short matrix. If RC has 2 elements, first element is a number of rows per page. Result is a 3 dimensional character array with (1ρCMAT)××/RC planes, RC[1] rows and (RC×1ρCMAT) columns.
3. DIO=1
4. RC←RC
5. COLS←1↑RC
6. C←1↑ρCMAT
7. Branch if 3 dimensional result:
   1. →(2⋯ρRC)ρL1
8. 2 dimensional result:
   1. ROWS←(1↑ρCMAT)÷COLS
   2. R←(ROWS, COLS×C)ρ 2 1 3 ₋(COLS, ROWS, C)ρ((COLS×ROWS), C)↑CMAT
   3. →0
9. 3 dimensional result:
   1. L1:ROWS+1↑RC
   2. PGS←(1↑ρCMAT)÷ROWS×COLS
   3. R←(PGS, ROWS, COLS×C)ρ 1 3 2 4 ₋(PGS, COLS, ROWS, C)ρ((PGS×COLS×ROWS), C)↑CMAT

\[ \text{\#} \]
Chapter 4 Solutions

POSITIONING CHARACTER DATA

6.

[WSID: FORMAT]

\[ R = WID \ \text{HEADINGS} \ \text{CS} ; \text{DIO} ; \text{A} ; \text{ARGSTART} ; \text{B} ; \text{HBDG} ; \text{BSEG} ; \text{HDGLEAD} ; \text{LEN} ; \text{NCOLS} ; \text{NHDG} ; \text{NROWS} ; \text{NSEG} ; \text{RESTART} ; \text{S} ; \text{SEGLEAD} ; \text{SEGROWS} ; \text{T} \]

[1] a Creates column headings from text CS within
[2] a field widths WID. CS is a character vector with
[3] a text for successive headings separated by 'n'.
[4] a The format of WID is: (widths of headings, not
[5] a including spacing), (spacing between columns).
[6] a If WID has fewer elements than CS has segments,
[7] a its values are repeated; if it does not include
[8] a spacing specification, 2 is used. Empty headings
[9] a are not underlined. Separate lines of multi-line
[10] a heading by 'c'.
[12] DIO = 0
[13] a 1s for hdg starts:
[14] BHDG = CS = 'n'
[15] a 1s for segment starts:
[16] BSEG = CS = 'n'
[17] a
[18] a Flag end of hdgs (1 elt per segment):
[19] BHDG = 1 + BSEG / BHDG
[20] a No. segments per hdg:
[21] T = BHDG / 1 + BHDG
[22] NHDG = \rho NSEG + T - 1 \rightarrow 1, T
[23] a Segment lengths:
[24] T = BSEG / 1 + BSEG
[25] LEN = (1 + T, \rho BSEG) - T + 1
[26] a Spacing between hdgs (2s if omitted):
[27] S = NHDG / WID
[28] S = NHDG / S, (0 = \rho S) / 2
[29] a Reshape widths to conform with headings:
[30] WID = NHDG / WID
[31] a Truncated segment lengths:
[32] LEN = LEN / NSEG / WID
[33] a Leading blanks per segment, to center:
[34] SEGLEAD = f((NSEG / WID) - LEN) / 2
[35] a Col in which each hdg begins:
[36] HDGLEAD = \text{10} + \backslash \text{WID} + S
[37] a Index of char. following delim. of each segment:
[38] ARGSTART = 1 + BSEG / 1 + BSEG
[39] a No. of rows and columns in result (A is no. rows
[40] a without underlines):
[41] NROWS = 1 + A + / / NSEG
[42] NCOLS = (1 + (NHDG - 1) / \rho S) + / / WID
[43] a No. whole rows before each segment:
[44] T = NSEG / A - \rightarrow / NSEG
[45] SEGROWS = T = 1 + T
[46] a Index in raveled result where each segment starts:
[47] RESTART = SEGLEAD + (NSEG / HDGLEAD) + SEGROWS * NCOLS
[48] a Create blank, raveled result:
[49] R = (B + NROWS * NCOLS) \rho' , '
\texttt{\textasciitilde{V} HEADINGS (continued)}

\[50\] \texttt{a Flag nonempty headings:}
\[51\] \texttt{T\textasciitilde{NHDG}\textasciitilde{0}}
\[52\] \texttt{T[(\times\text{LEN})/\text{NSEG}/\text{NHDG}]\textasciitilde{1}}
\[53\] \texttt{a Indices of underlines in raveled result:}
\[54\] \texttt{A\textasciitilde{T/WID}}
\[55\] \texttt{A\textasciitilde{A/((T/\text{WID})/1\text{LEN}_(B-\text{NCOLS}),\text{WID+S})-1\text{LEN},+\text{A}}}
\[56\] \texttt{A\textasciitilde{A+\text{LPA}}}
\[57\] \texttt{a Insert underlines:}
\[58\] \texttt{R[\text{A}]\textasciitilde{\text{L}}}  
\[59\] \texttt{a T\textasciitilde{MONIOTA LEN:}}
\[60\] \texttt{T\textasciitilde{T+\text{LEN}}/\text{LEN}-1\text{LEN},+\text{LEN}}
\[61\] \texttt{R[T+\text{LEN/RESSTART}]\textasciitilde{T+\text{LEN/ARGSTART}}}
\[62\] \texttt{a Reshape result to matrix:}
\[63\] \texttt{R\textasciitilde{R(NROWS,NCOLS)R}}
\texttt{\textasciitilde{V}}
1. If your APL implementation supports matrix right arguments to $\oplus$:

   \[
   SPNUM ← PNUM[\oplus PNUM;]
   \]

   Otherwise (major-to-minor sorting done in minor-to-major order):

   \[
   G ← PNUM[;3]
   G ← G[\oplus PNUM[G;2]]
   G ← G[\oplus PNUM[G;1]]
   SPNUM ← PNUM[G;]
   \]

   Or pack and sort:

   \[
   SPNUM ← PNUM[\oplus 1E3 1E7 1E4 \oplus PNUM;]
   \]

   Note: Numbers are packed to 14 digit floating point numbers (the 1E4 assumes 4 digit extensions). Since the 14 digits do not exceed the 16 or 17 digits of available precision and since $\oplus$ works with the full precision (i.e. does not refer to DCT), the result will be accurate.

2. Since dyadic $\iota$ is dependent upon DCT (for floating point arguments), it will "find" matches which are very close but not exact. The IOTA function will overlook such close values and will "find" only those values which match exactly (to 16 or 17 digits of precision). This will usually not be a problem since IOTA will most often be used on integer arguments or on floating point arguments whose values have not been computed. Only through such computations will "equal" values become slightly different.
\[
\begin{align*}
\text{INDS} & \leftarrow \text{BASE} \text{ IOTA VALS; A; F; G; I; L} \\
& \text{Returns the indices of BASE at which the} \\
& \text{elements of VALS first match, i.e.} \\
& \text{INDS} \leftarrow \text{BASE} \updownarrow \text{VALS} \quad \text{(but maybe faster)} \\
& \text{Branch if right arg a vector:} \\
& \quad \text{(1=} \rho \text{VALS}) \rho \text{L1} \\
& \text{Handle scalar right arg:} \\
& \quad \text{INDS} \leftarrow \text{BASE} \updownarrow \text{VALS} \\
& \text{Branch unless no elts in either arg:} \\
& \quad \text{(x=} \text{A} \text{L}) \rho \text{L2} \\
& \text{Branch unless left arg has 1 elt:} \\
& \quad \text{(L=} \rho \text{L3} \\
& \text{Handle 1 elt left arg:} \\
& \quad \text{INDS} \leftarrow \text{O} \text{O} \text{O} \\
& \text{Branch if both args have more than 1 elt:} \\
& \quad \text{L2} \leftarrow \text{(} \text{F} \text{=} \text{L}) \rho \text{L4} \\
& \text{Handle 1 elt right arg:} \\
& \quad \text{INDS} \leftarrow \text{BASE} \updownarrow \text{VALS} \\
& \text{Branch if sort alg. costs more than looping alg.:} \\
& \quad \text{(remove A after replacing C1,C2,C3,C4 by} \\
& \quad \text{computed constants):} \\
& \quad \text{L4} \leftarrow \text{A} \leftarrow \text{C4} \text{C5} \text{X} \text{A} \text{C3} \text{X} \text{L} \rho \text{L5} \\
& \text{Combine args. and sort (like values together):} \\
& \quad \text{G} \leftarrow \text{A} \updownarrow \text{BASE} \updownarrow \text{VALS} \\
& \text{Flag 1st of distinct elts by shifting and comparing:} \\
& \quad \text{F} \leftarrow \text{A} \uparrow \text{I} \uparrow \text{A} \\
& \text{Insure 1st elt is 1 (in case all rows the same):} \\
& \quad \text{F} \leftarrow \text{O} \text{O} \text{O} \text{O} \text{O} \leftarrow \text{F} \\
& \text{Indices of 1st distinct elts:} \\
& \quad \text{I} \leftarrow \text{F} \text{G} \\
& \text{Replicate for each like elt:} \\
& \quad \text{F} \leftarrow \text{O} \text{O} \text{O} \text{O} \leftarrow \text{F} \\
& \text{Unsort indices (to catenated order):} \\
& \quad \text{INDS} \leftarrow \text{I} \\
& \quad \text{INDS} \leftarrow \text{G} \uparrow \text{I} \\
& \text{Keep those corresponding to right arg:} \\
& \quad \text{INDS} \leftarrow \text{L} \uparrow \text{INDS} \\
& \text{Set 'not found' inds to 'one greater':} \\
& \quad \text{INDS} \leftarrow \text{INDS} \uparrow \text{L} \uparrow \text{O} \text{O} \\
& \text{Use looping algorithm if more efficient:} \\
& \quad \text{L5} \leftarrow \text{INDS} \leftarrow \text{BASE} \updownarrow \text{VALS} \\
\end{align*}
\]
Chapter 5 Solutions

3. \( I \left( PNUM \land \neg P \right) \lor I \left( 1E3 \ 1E7 \ 1E4 \land PNUM \right) \lor 1E3 \ 1E7 \ 1E4 \lor P \)

4. \( \text{LOWER} \leftarrow 1000 \ 10000 \ 20000 \ 50000 \ 70000 \ 100000 \)
\( R \leftarrow \left( 1 \ 2 \ 5 \ 3 \ 5 \ 4 \ 5 \right) \left[ \text{LOWER LIOTA SALARY} \right] \)

5. 

[WSID: SEARCH]

\( V \leftarrow \text{DEB CVEC} \)

- \( \text{A} \) Deletes extraneous (leading, trailing or redundant) blanks from its argument and returns the compressed result.
- \( \text{A} \) Put extra blank on beginning and end:
- \( \text{CVEC} \leftarrow ' ', \text{CVEC} ', ' \)
- \( \text{A} \) Search for 2 contiguous blanks:
- \( \text{R} \leftarrow \text{1} \text{1} \left( \sim \text{CVEC} \Delta \text{SS} ' ' \right) / \text{CVEC} \)

6. \( \text{D} \leftarrow \text{1} \text{1} \left( \text{NVEC} ' ' \right) \) REPLACE ' -1 ' BY ' N/A '

7. \( \text{D} \leftarrow \left( \lor / \ 	ext{O} \ 2 \downarrow ( \rho \text{ENAMES}) \rho (. \text{UPPERCASE ENAMES}) \Delta \text{SS} ' \text{SON}' \right) / \text{ENAMES} \)

(Note: the last 2 Boolean columns are dropped to avoid coincidental wrap-around matches, e.g. if row 5 ends with 'SO' and row 6 starts with 'N')

In APL2:

\( \text{D} \leftarrow \left( \lor / ' \text{SON}' . \text{UPPERCASE ENAMES} \right) / \text{ENAMES} \)
Chapter 6 Solutions

SELECTING

1. \[
\frac{((pV)p10)}{V}
\]
   or
   \[
   V1([-OIO]+2\times l(pV)\div2)
   \]
   or
   \[
   (((pV)\div2),2)pV)\[]\cdot\partial\]

2. \[
(2pOIO)\%M
\]
   or
   \[
   (pM)p(1+1pM)\!l\!/\!M
   \]
   or
   \[
   (((-OIO)+1l0pM)\!F\!/\!M)\cdot\partial\]

3. The expressions \(\!lV\) or \(V[pV]\) return one element vectors, not scalars. Therefore, the result of \(M[;\!lV]\) is a one column matrix, not a vector. The following produce correct results:
   \[
   ,M[;\!lV]
   \]
   or
   \[
   ,M[;V[(pV)\!-\!OIO]]
   \]
   or
   \[
   M[;\!l0)p\!-\!lV]
   \]
   or
   \[
   M[;\!l0)p\!FV]
   \]

4. Approach 1:

   SHAPE\!p\!NAMES
   NAMES\!+,NAMES
   NAMES[(NAMES='//')/\!l0\!P\!NAMES]+','
   NAMES\!+SHAPE\!p\!NAMES
Approach 2:

\[
A1 \leftarrow A2 \leftarrow (DAV \in \text{NAMES}) / \text{DAV} \\
A1[(A1='/' / \text{p} A1) \leftarrow ']' \\
\text{NAMES} \leftarrow A1[A2 \setminus \text{NAMES}]
\]

Approach 3 (APL2):

\[
(('/' =, \text{NAMES}) /, \text{NAMES}) \leftarrow '
\]

5. A. \( \quad \Pi IO \leftarrow 0 \)
\[ \text{MRATES} \leftarrow (, \text{RATES})[\text{DUR} + 16 \times \text{IAGE} + 100 \times \text{SEX}] \]

B. \( \quad \Pi IO \leftarrow 0 \)
\[ \Delta \text{DUR} \leftarrow 15 \times \text{DUR} \]  
\[ \Delta \text{IAGE} \leftarrow \text{IAGE} + \text{DUR} - \Delta \text{DUR} \]  
\[ \text{MRATES} \leftarrow (, \text{RATES})[\Delta \text{DUR} + 16 \times \Delta \text{IAGE} + 100 \times \text{SEX}] \]

C. \( \quad \Pi IO \leftarrow 0 \)
\[ \text{VRATES} \leftarrow , \text{RATES} \]  
\[ \text{VRATES}[\text{NEWDUR} + 16 \times \text{NEWIAGE} + 100 \times \text{NEWSEX}] \leftarrow \text{NEWRATES} \]  
\[ \text{RATES} \leftarrow (\text{p} \text{RATES}) \text{p} \text{VRATES} \]

6.  

[WSID: SEARCH]

\[
\n(\n\text{R} \leftarrow \text{UNQNV} \text{ NV;} \text{FIRST}; \text{G}; \text{SORTED} \\
[1] \text{ a Returns the distinct elements of the} \\
[2] \text{ a numeric vector NV.} \\
[3] \text{ SORTED} \leftarrow \text{NV}[\text{G} \leftarrow \text{NV}] \\
[4] \text{ FIRST} \leftarrow \text{SORTED} \setminus 1 \phi \text{SORTED} \\
[5] \text{ a Set 1st elt to 1 in case FIRST all 0s:} \\
[6] \text{ FIRST} \leftarrow \text{FIRST} \setminus 1 \\
[7] \text{ } R \leftarrow \text{FIRST} / \text{SORTED} \\
[8] \text{ a } \text{FIRST} \leftarrow \text{FIRST} + \Pi IO \\
[9] \text{ a } \text{ind} \leftarrow \text{G} \\
[10] \text{ a } \text{ind} \leftarrow \text{G} + \text{\FIRST} \\
\)
\]

[WSID: SEARCH]

\[
\n(\n\text{R} \leftarrow \text{UNQCV} \text{ CV} \\
[1] \text{ a Returns the distinct elements of the} \\
[2] \text{ a character vector CV.} \\
[3] \text{ R} \leftarrow (\text{DAV} \in \text{CV}) / \text{DAV} \\
[4] \text{ a } \text{ind} \leftarrow \text{R} \setminus \text{CV} \\
\)
\]

-338-
Chapter 6 Solutions

SELECTING

[V: R\+UNQCM CM;FIRST;G;SORTED
[1] \(\alpha\) Returns the distinct rows of the character
[2] matrix CM.
[3] G\+DAV\+CM
[4] \(\alpha\) If dyadic \& unavailable:
[5] G\+DAV CGRADEUP CM
[6] SORTED\+CM[G;]
[7] FIRST\+\+SORTED\+\#leSORTED
[8] \(\alpha\) Set 1st elt to 1 in case FIRST all 0s:
[9] FIRST\+\+FIRST\+1
[10] R\+FIRST\+SORTED
[11] \(\alpha\) FIRST[\+\+FIRST]\+\OIO
[12] \(\alpha\) ind\+G
[13] \(\alpha\) ind[G]\+\FIRST

[V: R\+N UNQI1 IV;BIT;OIO
[1] \(\alpha\) Returns the distinct elements of the
[2] origin 1 index vector IV. All elements
[3] of IV must be elements of \(\in\N\).
[4] \(\OIO\+1
[5] BIT\+\N0
[6] BIT[IV]\+1
[7] R\+BIT\+\N
[8] \(\alpha\) ind\+(BIT\+\O\R)[IV]
[9] \(\alpha\) Alternative:
[10] \(\alpha\) ind\+(+BIT)[IV]

[V: R\+N UNQI0 IV;BIT;OIO
[1] \(\alpha\) Returns the distinct elements of the
[2] origin 0 index vector IV. All elements
[3] of IV must be elements of \(\in\N\).
[4] \(\OIO\+0
[5] BIT\+\N0
[6] BIT[IV]\+1
[7] R\+BIT\+\N
[8] \(\alpha\) ind\+(BIT\+\O\R)[IV]
[9] \(\alpha\) Alternative:
[10] \(\alpha\) ind\+(+BIT)[IV]-1

-339-
Chapter 7 Solutions

FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABULATIONS

1. \[ R \leftarrow \left\{ \begin{array}{l} 'ECMPH' = 'TZONE' \\
                \text{(utility function)} \\
      \end{array} \right. \]
   or
   \[ I11 \leftarrow 'ECMPH' \cup 'TZONE' \\
      R \leftarrow 5 11 \text{ PLUSRED } 1 \\
   \text{or} \\
      R \leftarrow (5;'ECMPH' \cup 'TZONE')+/1 \quad \text{(hypothetical)} \]

2. \[ R \leftarrow ( 'ECMPH' \cup 'TZONE') \times \text{SALES} \]
   or
   \[ I11 \leftarrow 'ECMPH' \cup 'TZONE' \quad \text{(utility function)} \\
      R \leftarrow 5 11 \text{ PLUSRED SALES} \\
   \text{or} \\
      R \leftarrow (5;'ECMPH' \cup 'TZONE')+/\text{SALES} \quad \text{(hypothetical)} \]

3. \[ \begin{array}{l}
      A \leftarrow 'ECMPH' = 'TZONE' \\
      B \leftarrow 'TYPE' = 'BCPS' \\
      FRQ \leftarrow A + \cdot B \\
      AMT \leftarrow (A \times (pA) \cdot p\text{SALES}) + \cdot B \\
      MAX \leftarrow (A \times (pA) \cdot p\text{SALES}) \div \cdot B \\
   \end{array} \]
   or
   \[ I11 \leftarrow 'ECMPH' \cup 'TZONE' \quad \text{(utility functions)} \\
      I12 \leftarrow 'BCPS' \cup 'TYPE' \\
      FRQ \leftarrow 5 4 11 12 \text{ PLUSRED } 1 \\
      AMT \leftarrow 5 4 11 12 \text{ PLUSRED SALES} \\
      MAX \leftarrow 5 4 11 12 \text{ MAXRED SALES} \\
   \text{or} \\
      I11 \leftarrow 'ECMPH' \cup 'TZONE' \quad \text{(hypothetical)} \\
      I12 \leftarrow 'BCPS' \cup 'TYPE' \\
      FRQ \leftarrow (5 4; I11; I12) +/1 \\
      AMT \leftarrow (5 4; I11; I12) +/\text{SALES} \\
      MAX \leftarrow (5 4; I11; I12) \div \text{SALES} \]

-340-
Chapter 7 Solutions  FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABS

4.

        [WSID: REDUCE]

        \[ R \leftarrow L \ PLUSRED \ ARRAY; CIND; CUM; DIM; DSHAPE; GRADE; I; LAST; M; N \]

1. No. of ways for N-way reduction:
2. \[ N \leftarrow l(\rho, L) \div 2 \]
3. Which dimension to be "reduced"?
4. \[ DIM \leftarrow \lceil (N+N) \downarrow L \rceil, \rho \uparrow RANK \leftarrow \rho \rho ARRAY \]
5. Branch unless 0-way reduction:
6. \[ \rightarrow (\times N) \rho L L \]
7. \[ R \leftarrow / [DIM] ARRAY \rightarrow 0 \]
8. Separate left arg into its pieces:
9. \[ L1: DSHAPE \leftarrow N \rho L \]
10. CIND \leftarrow N \rho N \downarrow L \]
11. Begin to compute "raveled result indices":
12. \[ I \leftarrow DIO \]
13. Index from \( t \) DSHAPE([I]) to cause index error if
14. invalid indices:
15. \[ \text{RRI} \leftarrow (t \text{DSHAPE}[I])[\{ \# 'I', \# \text{CIND}[I] \}] \]
16. Branch if origin is 1:
17. \[ \rightarrow \text{IO} \rho \text{LOOPl} \]
18. Continue computing RRI (N iterations for N-way reduction):
19. \[ \text{LOOP0} \leftarrow (N \leq I \leftarrow I + 1) \rho \text{ENDLP} \]
20. \[ \text{RRI} \leftarrow (t \text{DSHAPE}[I])[\{ \# 'I', \# \text{CIND}[I] \}] + \text{DSHAPE}[I] \times \text{RRI} \]
21. \[ \rightarrow \text{LOOP0} \]
22. \[ \text{LOOP1} \leftarrow (N < I \leftarrow I + 1) \rho \text{ENDLP} \]
23. \[ \text{RRI} \leftarrow (t \text{DSHAPE}[I])[\{ \# 'I', \# \text{CIND}[I] \}] + \text{DSHAPE}[I] \times \text{RRI} + \text{1} \]
24. \[ \rightarrow \text{LOOP1} \]
25. Determine unique elements of RRI:
26. \[ \text{ENDLP} \leftarrow \text{GRADE} \leftarrow \text{ARRRI} \]
27. \[ \text{SORTED} \leftarrow \text{RRI[GRADE]} \]
28. \[ \text{LAST} \leftarrow \text{SORTED} \uparrow \# \downarrow \text{SORTED} \]
29. \[ \text{LAST} \leftarrow (\times \rho \text{LAST}) \rho (\rho \text{LAST}) \uparrow \# \text{IO} \downarrow 1 \]
30. \[ \text{URRI} \leftarrow \text{LAST} / \text{SORTED} \]
31. Branch unless ARRAY a scalar (i.e. freq. count):
32. \[ \rightarrow (\times \text{RANK}) \rho L L \]
33. Perform partitioned frequency count:
34. \[ \text{CUM} \leftarrow \text{LAST} / \downarrow \rho \text{LAST} \]
35. \[ \text{CUM} \leftarrow \text{CUM} - (\rho \text{CUM}) \rho (\rho \text{LAST}) \downarrow 1 \uparrow \text{IO} \]
36. Branch if ARRAY is 1 (usually):
37. \[ \rightarrow (1 = \text{ARRAY}) \rho L L \]
38. Multiply freq. counts by scalar:
39. \[ \text{CUM} \leftarrow \text{ARRAY} \times \text{CUM} \]
40. Initialize result with 0's:
41. \[ \text{L2} \leftarrow \times / \text{DSHAPE} \rho 0 \]
42. Insert result of partitioned freq. count:
43. \[ \text{R[URRI]} \leftarrow \text{CUM} \]
44. Reshape to desired shape:
45. \[ \text{R} \leftarrow \text{DSHAPE} \rho R \]
46. \[ \rightarrow 0 \]
PLUSRED (continued)

α Reorder ARRAY to conform with SORTED:
L3:M+DIM-010

α Perform partitioned reduction:
CUM+LAST/[DIM]+\[DIM]ARRAY

α Initialize result. Fill with identity elt.
Ravel the DSHAPE dim.s:
R+((-M)Φ(×/DSHAPE),1↓MΦ=ARRAY)ρ0

α Insert result of partitioned reduction:
R+LMAXRED ARRAY;CIND;DIF;DIM;DSHAPE;GRADE;I;LAST;M;N;
RANK;RRI;SORTED;URRI

α No. of ways for N-way reduction:
N+1(ρ, L)÷2

α Which dimension to be "reduced"?
DIM+tt((N+N)↓L),"↓ttRANK=ρ=ARRAY

α Branch unless 0-way reduction:
→((×N)ρ'L1

α Separate left arg into its pieces:
L1;DSHAPE+ΝρL

α Begin to compute "raveled result indices":
I+ΟI0

α Index from iDSHAPE[I] to cause index error if
α invalid indices:
RRI+((iDSHAPE[I])[{i'I}',%CIND[I]]

α Branch if origin is 1:
→ΟΙ0PLOOP1

α Continue computing RRI (N iterations for
α N-way reduction):
LOOP0:+(N≤I<i+1)ρENDLP
RRI+((iDSHAPE[I]){i'I}',%CIND[I])×+DSHAPE[I]×RRI

α ENDLP:
RRI+((iDSHAPE[I]){i'I}',%CIND[I])×+DSHAPE[I]×RRI+~1

α Determine unique elements of RRI:
ENLP:GRADE=§RRI
α SORTED=ΦRRI[GRADE]
γ LAST=ΦSORTED#ΦSORTED
α LAST[(×ρLAST)ρ(ρLAST)↓ΟI0]↓1
α URRI=LAST/SORTED
\( \text{MAXRED (continued)} \)

\[ \text{M} \triangleleft \text{DIM} - \text{IO} \]

\[ \text{ARRAY} \triangleleft \text{"ARRAY} [', (M_p ; '), ', ' \text{GRADE}', ((\text{RANK} - M + 1)\rho ; ')], ''] \]

\( \text{a Reorder ARRAY to conform with SORTED:} \)

\[ \text{DIF} \triangleleft ([/\text{DIM}]\text{ARRAY}) - 1 / [/\text{DIM}]\text{ARRAY} \]

\[ \text{DIF} \triangleleft (\text{DIM} \times [\text{RANK}] \triangleleft \text{DIF} \times [\text{DIM}] \times \text{RANK} - 1) / \text{LAST} \]

\[ \text{DIF} \triangleleft ([/\text{DIM}] \times [\text{DIM}] \times \text{ARRAY} + \text{DIF}) - \text{LAST} / [/\text{DIM}]\text{DIF} \]

\( \text{Perform partitioned reduction:} \)

\[ \text{R} \triangleleft (\text{MINRED} \times \text{CIND} \times \text{DIF} \times \text{DIM} \times \text{DSHAPE} \times \text{GRADE} \times \text{I} \times \text{LAST} \times M \times N \times \text{RANK} \times \text{RRI} \times \text{SORTED} \times \text{URRI} \]

\( \text{a No. of ways for N-way reduction:} \)

\[ N \triangleleft (\rho, L) / 2 \]

\( \text{Which dimension to be "reduced"?} \)

\[ \text{DIM} \triangleleft (N + N \times L), - 1 \triangleleft \text{RANK} \times \rho \triangleleft \text{ARRAY} \]

\( \text{Branch unless 0-way reduction:} \)

\[ \text{R} \triangleleft 1 / [/\text{DIM}]\text{ARRAY} \]

\[ \text{R} \triangleleft 0 \]

\( \text{Separate left arg into its pieces:} \)

\[ \text{L1} : \text{DSHAPE} \triangleleft N \times L \]

\[ \text{CIND} \triangleleft N \times N \times L \]

\( \text{Begin to compute "raveled result indices":} \)

\[ \text{I} \triangleleft \text{DIM} \]

\( \text{Index from \text{tDSHAPE}[I] to cause index error} \)

\( \text{if invalid indices:} \)

\[ \text{RRI} \triangleleft (\text{tDSHAPE}[I])[\text{\$'I'}, \text{\$CIND}[I]] \]

\( \text{Branch if origin is 1:} \)

\[ \text{\$I\text{O}\text{O}\text{P}1} \]

\( \text{Continue computing RRI (N iterations for} \)

\( \text{N-way reduction):} \)

\[ \text{LOOPOP:} \triangleleft (N < I < I + 1) \times \text{ENDLP} \]

\[ \text{RRI} \triangleleft (\text{tDSHAPE}[I])[\text{\$'I'}, \text{\$CIND}[I]] \times \text{DSHAPE}[I] \times \text{RRI} \]

\[ \text{ENDLP} \]

\( \text{\$I\text{O}\text{O}\text{P}1} \)

\( \text{Determine unique elements of RRI:} \)

\[ \text{ENDLP} \triangleleft \text{GRADE} \triangleleft \text{RRI} \]

\[ \text{SORTED} \triangleleft \text{RRI}[\text{GRADE}] \]

\[ \text{LAST} \triangleleft \text{SORTED} \triangleleft \text{\$I\text{O}\text{O}\text{P}1} \]

\[ \text{URRI} \triangleleft \text{\$I\text{O}\text{O}\text{P}1} \]
\( \text{MINRED (continued)} \)

[33] \( \text{a Reorder ARRAY to conform with SORTED.} \)

[34] \( \text{M} = \text{DIM} - \text{IO} \)

[35] \( \text{ARRAY} \leftarrow \emptyset \text{ARRAY}, (\text{M} \rho' ; ') , \text{GRADE} , (\text{RANK} - \text{M} + 1) \rho' ; ')' \)

[36] \( \text{a Perform partitioned reduction.} \)

[37] \( \text{DIF} \leftarrow ([/\text{DIM}] \text{ARRAY}) - 1 / [\text{DIM}] \text{ARRAY} \)

[38] \( \text{DIF} \leftarrow ([\text{DIM}] = \text{RANK} \& \text{DIM} \leftarrow \text{DIM} - 1 \& \text{LAST} \)

[39] \( \text{DIF} \leftarrow (\text{LAST} / [\text{DIM}] \text{ARRAY} - \text{DIF}) + \text{LAST} / [\text{DIM}] \text{ARRAY} \)

[40] \( \text{a Initialize result. Fill with identity elt.} \)

[41] \( \text{Ravel the DSHAPE dim.s:} \)

[42] \( \text{R} \leftarrow (-\text{M}) \& (\times / \text{DSHAPE}) , 1 \& \text{M} \& \text{ARRAY} \rho / 1 \text{R} \)

[43] \( \text{a Insert result of partitioned reduction:} \)

[44] \( \text{R} \leftarrow \text{R}[', (\text{M} \rho' ; ') , ' \text{URRI}' , (\text{RANK} - \text{M} + 1) \rho' ; ') , ')' \& \text{DIF} \)

[45] \( \text{a Reshape to desired shape (unravel DSHAPE dim.s):} \)

[46] \( \text{R} \leftarrow (-\text{M}) \& \text{DSHAPE} , 1 \& \text{M} \& \text{ARRAY} \rho \text{R} \)

\( \text{v} \)

\( \text{[WSID: REDUCE]} \)

\( \text{v} \text{ R} \leftarrow \text{L ANDRED ARRAY; CIND; CUM; DIM; DSHAPE; GRADE; I; LAST; M; N; RANK; RRI; SORTED; URRI} \)

[1] \( \text{a No. of ways for N-way reduction:} \)

[2] \( \text{N} \leftarrow (\rho, \text{L}) \div 2 \)

[3] \( \text{a Which dimension to be "reduced"?} \)

[4] \( \text{DIM} = 11((\text{N} + \text{N}) \downarrow \text{L}) , -1 \uparrow \text{RANK} \& \text{DIM} \leftarrow \text{DIM} \rho \text{L} \)

[5] \( \text{a Branch unless 0-way reduction:} \)

[6] \( \& ((\times \text{N}) \& \text{DIM} \leftarrow \text{DIM} \rho \text{L} \)

[7] \( \text{R} \leftarrow / [\text{DIM}] \text{ARRAY} \)

[8] \( \& 0 \)

[9] \( \text{a Separate left arg into its pieces:} \)

[10] \( \text{L1} = \text{DSHAPE} \& \text{N} \rho \text{L} \)

[11] \( \text{CIND} = \& \text{N} \rho \text{N} \downarrow \text{L} \)

[12] \( \text{a Begin to compute "raveled result indices":} \)

[13] \( \text{I} \leftarrow \text{IO} \)

[14] \( \text{a Index from \& DSHAPE[I] to cause index error} \)

[15] \( \text{a if invalid indices:} \)

[16] \( \text{RRI} \leftarrow (\& \text{DSHAPE[I]} \& [\# 'I', \& \text{CIND[I]}] \)

[17] \( \text{a Branch if origin is 1:} \)

[18] \( \& \text{IO} \& \text{LOOP1} \)

[19] \( \text{a Continue computing RRI (N iterations for} \)

[20] \( \text{a N-way reduction:} \)

[21] \( \text{LOOP0} \& (\text{N} \& \text{I} \& \text{I} + 1) \rho \text{ENDLP} \)

[22] \( \text{RRI} \leftarrow (\& \text{DSHAPE[I]} \& [\# 'I', \& \text{CIND[I]}] + \text{DSHAPE[I]} \& \text{RRI} \times \text{RRI} \)

[23] \( \& \text{LOOP0} \)

[24] \( \text{LOOP1} \& (\text{N} \& \text{I} \& \text{I} + 1) \rho \text{ENDLP} \)

[25] \( \text{RRI} \leftarrow (\& \text{DSHAPE[I]} \& [\# 'I', \& \text{CIND[I]}] + \text{DSHAPE[I]} \& \text{RRI} + \& 1 \times \text{RRI} \)

[26] \( \& \text{LOOP1} \)

[27] \( \text{a Determine unique elements of RRI:} \)

[28] \( \text{ENDLP} \& \text{GRADE} \& \& \text{RRI} \)

[29] \( \text{SORTED} \& \text{RRI[GRADE]} \)

[30] \( \text{LAST} \& \text{SORTED} \& \& \text{SORTED} \)

[31] \( \text{LAST} \& (\& \text{LAST} \rho (\& \text{LAST}) \& \& \text{IO}) \& 1 \)

[32] \( \text{URRI} \& \text{LAST} \& \text{SORTED} \)

\(-344-\)
\[ \text{ANDRED (continued)} \]

\[ \text{\[33\]} \text{ a Reorder ARRAY to conform with SORTED:} \]
\[ \text{\[34\]} \text{ M\langle DIM\rangle\langle IO} \]
\[ \text{\[35\]} \text{ ARRAY\rightarrow\langle ARRAY\rangle, (M\rho';'), 'GRADE', ((RANK-M+1)\rho';'), ']} \]
\[ \text{\[36\]} \text{ a Perform partitioned reduction (note: \(\vee\) \(\leftrightarrow\) \(\sim\vee\sim\))} \]
\[ \text{\[37\]} \text{ CUM+LAST\langle DIM\rangle\langle \sim DIM\rangle\sim ARRAY} \]
\[ \text{\[38\]} \text{ CUM+\langle D\rangle\sim (\rho\text{CUM})\sim 0, \langle DIM\rangle\text{CUM}} \]
\[ \text{\[39\]} \text{ a Initialize result. Fill with identity elt.} \]
\[ \text{\[40\]} \text{ Ravel the DSHAPE dim.s:} \]
\[ \text{\[41\]} \text{ R\left((-M)\phi (x/DSHAPE), 1\downarrow M\phi\text{ARRAY}\phi R} \]
\[ \text{\[42\]} \text{ a Insert result of partitioned reduction:} \]
\[ \text{\[43\]} \text{ *R[',(M\rho';'), 'URRI', ((RANK-M+1)\rho';'), ']} \sim\text{CUM'} \]
\[ \text{\[44\]} \text{ a Reshape to desired shape (unravel DSHAPE dim.s):} \]
\[ \text{\[45\]} \text{ R\left((-M)\phi DSHAPE, 1\downarrow M\phi\text{ARRAY}\phi R} \]

\[ \text{\[\text{WSID: REDUCE}\]} \]
\[ \text{\[\text{\[4\]} \text{ a No. of ways for N-way reduction:} \]
\[ \text{\[\text{\[2\]} \text{ N\left(\phi \langle\rho, L\rangle\rangle/2} \]
\[ \text{\[\text{\[3\]} \text{ a Which dimension to be "reduced"?} \]
\[ \text{\[\text{\[4\]} \text{ DIM\left((N+N)\rangle\rangle/1, } 1\uparrow \text{RANK}\phi\text{ARRAY} \]
\[ \text{\[\text{\[5\]} \text{ a Branch unless 0-way reduction:} \]
\[ \text{\[\text{\[6\]} \text{ } \rightarrow (xN)\rho L \]
\[ \text{\[\text{\[7\]} \text{ R\left(\langle DIM\rangle\sim ARRAY} \]
\[ \text{\[\text{\[8\]} \text{ } \rightarrow 0 \]
\[ \text{\[\text{\[9\]} \text{ a Separate left arg into its pieces:} \]
\[ \text{\[\text{\[10\]} \text{ L\langle DSHAPE\rangle\langle N\rangle\phi L} \]
\[ \text{\[\text{\[11\]} \text{ CIND\langle N\rangle\phi N\phi L} \]
\[ \text{\[\text{\[12\]} \text{ a Begin to compute "raveled result indices":} \]
\[ \text{\[\text{\[13\]} \text{ I\rightarrow IO} \]
\[ \text{\[\text{\[14\]} \text{ a Index from \langle DSHAPE\rangle[I] to cause index error} \]
\[ \text{\[\text{\[15\]} \text{ a if invalid indices:} \]
\[ \text{\[\text{\[16\]} \text{ RRI\left(\langle DSHAPE\rangle[I]\rangle\langle \# I', \# CIND[I]\rangle} \]
\[ \text{\[\text{\[17\]} \text{ a Branch if origin is 1:} \]
\[ \text{\[\text{\[18\]} \text{ } \rightarrow IO\phi LOOP1} \]
\[ \text{\[\text{\[19\]} \text{ a Continue computing RRI (N iterations for} \]
\[ \text{\[\text{\[20\]} \text{ a N-way reduction):} \]
\[ \text{\[\text{\[21\]} \text{ LOOP0\rightarrow (N\leq I\leq I+1)\rho ENDLP} \]
\[ \text{\[\text{\[22\]} \text{ RRI\left(\langle DSHAPE\rangle[1]\rangle\langle \# I', \# CIND[I]\rangle\rangle\sim DSHAPE[1]\times RRI} \]
\[ \text{\[\text{\[23\]} \text{ } \rightarrow LOOP0} \]
\[ \text{\[\text{\[24\]} \text{ LOOP1\rightarrow (N< I< I+1)\rho ENDLP} \]
\[ \text{\[\text{\[25\]} \text{ RRI\left(\langle DSHAPE\rangle[1]\rangle\langle \# I', \# CIND[I]\rangle\rangle\sim DSHAPE[1]\times RRI+1} \]
\[ \text{\[\text{\[26\]} \text{ } \rightarrow LOOP1} \]
\[ \text{\[\text{\[27\]} \text{ a Determine unique elements of RRI:} \]
\[ \text{\[\text{\[28\]} \text{ ENDLP\rangle GRADE\langle\# RRI} \]
\[ \text{\[\text{\[29\]} \text{ SORTED\langle RRI\langle GRADE\rangle} \]
\[ \text{\[\text{\[30\]} \text{ LAST\langle SORTED\rangle\langle\# SORTED} \]
\[ \text{\[\text{\[31\]} \text{ LAST\langle (x\rho LAST)\phi (\rho\text{LAST\rangle\rangle/\langle\# IO\rangle+1} \]
\[ \text{\[\text{\[32\]} \text{ URRI\langle LAST/SORTED} \]}

-345-
ORRED (continued)

[33] A Reorder ARRAY to conform with SORTED:

[34] M←DIM-DIO

[35] ARRAY←'ARRAY[', (Mρ;''), 'GRADE', (((RANK-M+1)ρ;''), ']

[36] A Perform partitioned reduction:

[37] CUM+LAST/[DIM]+\[DIM]ARRAY

[38] CUM←CUM#(ρCUM)↑0,[DIM]CUM


[40] A Ravel the DSHAPE dim.s:

[41] R←((-M)Φ×/DSHAPE),1↓(Mρ)ARRAY)ρ0

[42] A Insert result of partitioned reduction:

[43] *'R[', (Mρ;''), 'URRI', (((RANK-M+1)ρ;''), ')←CUM'

[44] A Reshape to desired shape (unravel DSHAPE dim.s):

[45] R←((-M)ΦDSHAPE,1↓Mρ)ARRAY)ρ R

5.

[WSID: REDUCE]

R←L APLUSRED ARRAY;CIND;CUM;DIM;DSHAPE;GRADE;I;LAST;M;
N;RANK;RRI;SORTED;URRI

[1] A No. of ways for N-way reduction:

[2] N←l(ρ,L)÷2

[3] A Which dimension to be "reduced"?

[4] DIM←l((N+N)↓L),1↓RANK←ρρARRAY

[5] A Branch unless 0-way reduction:

[6] →(×N)ρL1

[7] ARRAY←+/[DIM]ARRAY

[8] A Construct milky-way result:

[9] R←((ρρARRAY),(DIM-DIO),N,ρARRAY),,ARRAY

[10] →0

[11] A Separate left arg into its pieces:

[12] L1:DSHAPE←NρL

[13] CIND←NρN↓L

[14] A Begin to compute "raveled result indices":

[15] I←DIO

[16] A Index from tDSHAPE[I] to cause index error if

[17] A invalid indices:

[18] RRI←(tDSHAPE[I])[*'I', *CIND[I]]

[19] A Branch if origin is 1:

[20] →DIOρLOOP1

[21] A Continue computing RRI (N iterations for N-way

[22] A reduction):

[23] LOOP0:→(N≤I←I+1)ρENDLP

[24] RRI←(tDSHAPE[I])[*'I', *CIND[I]]+DSHAPE[I]×RRI

[25] →LOOP0

[26] LOOP1:→(N<I←I+1)ρENDLP

[27] RRI←(tDSHAPE[I])[*'I', *CIND[I]]+DSHAPE[I]×RRI+~1

[28] →LOOP1

[29] A Determine unique elements of RRI:

[30] ENDLP:GRADE←RRI

[31] SORTED←RRI[GRADE]
\[ \Delta \text{PLUSRED (continued)} \]

[32] \text{LAST} \leftarrow \text{SORTED} \# \text{SORTED}
[33] \text{LAST}(\times \rho \text{LAST}) \rho (\rho \text{LAST}) \rightarrow \text{DIM} \rightarrow 1
[34] \text{URRI} \leftarrow \text{LAST} / \text{SORTED}
[35] \text{Branch unless ARRAY is a scalar (i.e. freq count)}:
[36] \rightarrow (\times \text{RANK}) L3
[37] \text{Perform partitioned frequency count:}
[38] \text{CUM} \leftarrow \text{LAST} / \rho \text{LAST}
[39] \text{CUM} \leftarrow \text{CUM} - (\rho \text{CUM}) \rho (-1 + \text{DIM}), \text{CUM}
[40] \text{Branch if ARRAY is 1 (usually)}:
[41] \rightarrow (1 = \text{ARRAY}) \rho L2
[42] \text{Multiply freq. counts by scalar:}
[43] \text{CUM} \leftarrow \text{ARRAY} \times \text{CUM}
[44] \text{Construct milky-way result:}
[45] L2: R \leftarrow (1, 0, N, (\rho \text{CUM}), \text{DSHAPE}), \text{URRI} - \text{DIM}), \text{CUM}
[46] \rightarrow 0
[47] \text{Reorder ARRAY to conform with SORTED:}
[48] L3: M \leftarrow \text{DIM} - \text{DIM} \rightarrow 0
[49] \text{ARRAY} \leftarrow \# ' \text{ARRAY}'[', (M \rho'; '), ' \text{GRADE'}, ((\text{RANK} - M + 1) \rho'; ')', ']
[50] \text{Perform partitioned reduction:}
[51] \text{CUM} \leftarrow \text{LAST} / \text{DIM} / \text{DIM} \rightarrow \text{ARRAY}
[52] \text{CUM} \leftarrow \text{CUM} - (\rho \text{CUM}) \rightarrow 0, [\text{DIM}], \text{CUM}
[53] \text{Construct milky-way result:}
[54] R \leftarrow ((\text{RANK}, M, N, (\rho \text{CUM}), \text{DSHAPE}), \text{URRI} - \text{DIM}), \text{CUM}

\[ \text{[WSID: REDUCE]} \]

\[ \Delta \text{MAXRED ARRAY; CIND; DIM; DSHAPE; GRADE; I; LAST; M; N; RANK; RRI; SORTED; URRI} \]
[1] \text{No. of ways for N-way reduction:}
[2] N \leftarrow (\rho, L) + 2
[3] \text{Which dimension to be "reduced"?}
[4] \text{DIM} \leftarrow \# ((N + N) \downarrow L), \text{DIM} \leftarrow \# \text{DIM} + \text{DIM} \rightarrow \text{ARRAY}
[5] \text{Branch unless 0-way reduction:}
[6] \rightarrow (\times N) \rho L1
[7] \text{ARRAY} \leftarrow \# [\text{DIM}] \rightarrow \text{ARRAY}
[8] \text{Construct milky-way result:}
[9] R \leftarrow ((\rho \text{ARRAY}), (\text{DIM} - \text{DIM}), N, \rho \text{ARRAY}), \text{ARRAY}
[10] \rightarrow 0
[11] \text{Separate left arg into its pieces:}
[12] L1: \text{DSHAPE} \leftarrow N \rho L
[13] \text{CIND} \leftarrow N \rho N \downarrow L
[14] \text{Begin to compute "raveled result indices":}
[15] I \leftarrow \text{DIM}
[16] \text{Index from \# DSHAPE[I] to cause index error if}
[17] \text{invalid indices:}
[18] \text{RRI} \leftarrow ((\text{DSHAPE[I]})[\# 'I', \# CIND[I]])
[19] \text{Branch if origin is 1:}
[20] \rightarrow \text{DIM} \rho L0 \text{ LOOPI}
[21] \text{Continue computing RRI (N iterations for}
[22] \text{N-way reduction):}
[23] \text{LOOPO} \rightarrow ((N + I + 1) \rho \text{ENDLP}
[24] \text{RRI} \leftarrow ((\text{DSHAPE[I]})[\# 'I', \# CIND[I]]) + \text{DSHAPE[I]} \times \text{RRI}

\[ -347 - \]
\[ \Delta \text{MAXRED} \hspace{1em} (\text{continued}) \]

[25] \( \rightarrow \text{LOOP0} \)
[26] \( \text{LOOP1} \rightarrow (N<I+I+1) \text{p\text{ENDLP}} \)
[27] \( RRI \leftarrow (\text{tSHAPE[I]}[(\#'I', \#\text{CIND[I])}] + \text{DSHAPE[I]} \times RRI + 1 \)
[28] \( \rightarrow \text{LOOP1} \)
[29] \( \ast \) Determine unique elements of \( RRI \):
[30] \( \text{ENDLPI: GRADE} \leftarrow \# RRI \)
[31] \( \text{SORTED} + \# RRI[\text{GRADE}] \)
[32] \( \text{LAST} \leftarrow \# \text{SORTED} \neq 1 \Theta \# \text{SORTED} \)
[33] \( \text{LAST} \leftarrow (\times \text{pLAST}) \rho (\rho \text{LAST} \neq \#0) \leftarrow 1 \)
[34] \( \# \text{URRI} \leftarrow \# \text{LAST} \Theta \# \text{SORTED} \)
[35] \( \ast \) Reorder \( \text{ARRAY} \) to conform with \# \text{SORTED}:
[36] \( \# \text{M} \hspace{1em} \times \text{DIM} \Theta \#0 \)
[37] \( \# \text{ARRAY} \leftarrow \# \text{ARRAY}[', (\text{M} \rho ';'), ' \text{GRADE}', ((\text{RANK} \hspace{1em} \text{M} \times 1 \rho ';'), ')'] \)
[38] \( \ast \) Perform partitioned reduction:
[39] \( \# \text{DIFF} \leftarrow ([/\text{DIM} \# \text{ARRAY}] - \text{l} / [\text{DIM} \# \text{ARRAY}] \)
[40] \( \# \text{DIFF} \leftarrow (\# \text{DIM} = \# \text{RANK}) \Theta \# \text{DIFF} \times (+1 \Theta \# \text{LAST}) \)
[41] \( \# \text{DIFF} \leftarrow ([\text{LAST} / [\text{DIM} \Theta [\text{DIM} \# \text{ARRAY} + \# \text{DIFF}] - \text{LAST} / [\text{DIM} \# \text{DIFF}] \)
[42] \( \ast \) Construct milky-way result:
[43] \( \# R \leftarrow ((\text{RANK}, \text{M}, \text{N}, (\text{pDIFF}), \text{DSHAPE}), \# \text{URRI} \Theta \#0), \#0, \# \text{DIFF} \)

\[ \Delta \text{MINRED} \hspace{1em} (\text{reduce}) \]

[1] \( \ast \) No. of ways for \( \text{N-way reduction} \):
[2] \( \# N \leftarrow (\rho, N) \div 2 \)
[3] \( \ast \) Which dimension to be "reduced"?
[4] \( \# \text{DIM} \leftarrow (\text{N} + N) \Theta \#0 \leftarrow 1 \Theta \# \text{RANK} \times \# \text{DIM} \)
[5] \( \ast \) Branch unless 0-way reduction:
[6] \( \rightarrow (\times \# \text{DIM} \times \# \text{DIM}) \)
[7] \( \# \text{ARRAY} \leftarrow 1 / [\text{DIM} \# \text{ARRAY}] \)
[8] \( \ast \) Construct milky-way result:
[9] \( \# R \leftarrow ((\text{DIM} \Theta \#0), (\text{DIM} \Theta \#0), \# \text{DIM}, \# \text{DIM}, \# \text{DIM}), \# \text{ARRAY} \)
[10] \( \rightarrow 0 \)
[11] \( \ast \) Separate left arg into its pieces:
[12] \( \# \text{L} \times \text{SHAPE} \leftarrow \# \text{N} \times \# \text{L} \)
[13] \( \# \text{CIND} \leftarrow \# \text{N} \Theta \# \text{N} \Theta \# \text{L} \)
[14] \( \ast \) Begin to compute "raveled result indices":
[15] \( \# \text{I} \leftarrow \# \text{DIM} \)
[16] \( \ast \) Index from \# \text{SHAPE[I]} to cause index error if
[17] \( \ast \) invalid indices:
[18] \( \# RRI \leftarrow (\text{tSHAPE[I]}[(\#'I', \# \text{CIND[I]}]) \)
[19] \( \ast \) Branch if origin is 1:
[20] \( \rightarrow \# \text{DIM} \Theta \text{LOOPI} \)
[21] \( \ast \) Continue computing \# \text{RRRI} \hspace{1em} (\text{N iterations for N-way reduction}):
[22] \( \# \text{LOOPO} \rightarrow (N + I + I + 1) \text{p\text{ENDLP}} \)
[23] \( \# \text{RRRI} \leftarrow (\text{tSHAPE[I]}[(\#'I', \# \text{CIND[I]})]) + \text{DSHAPE[I]} \times \text{RRRI} \)
[24] \( \rightarrow \text{LOOPO} \)
[25] \( \# \text{LOOPI} \rightarrow (N + I + I + 1) \text{p\text{ENDLP}} \)
[26] \( \# \text{RRRI} \leftarrow (\text{tSHAPE[I]}[(\#'I', \# \text{CIND[I]})]) + \text{DSHAPE[I]} \times \text{RRRI} \Theta \#0 \)
[28] \( \rightarrow \text{LOOPI} \)
\[ \Delta \text{MINRED (continued)} \]

[29] u Determine unique elements of RRI:
[30] ENDLP: GRADE \leftarrow \Delta RRI
[31] SORTED \leftarrow RRI[GRADE]
[32] LAST \leftarrow SORTED \#1 \Phi \text{SORTED}
[33] LAST[\times \rho \text{LAST}] \rho (\rho \text{LAST} \# \Omega \text{IO}) \rightarrow 1
[34] URRI \leftarrow \text{LAST} / \text{SORTED}

\[ \text{u Reorder ARRAY to conform with SORTED:} \]
[35] M \leftarrow \text{DIM} / \text{IO}
[36] ARRAY \leftarrow \# ' \text{ARRAY}[, (M \rho ';'), ' \text{GRADE}, ((\text{RANK} - M + 1) \rho ';'),']'

[38] u Perform partitioned reduction:
[39] DIF \leftarrow (\text{DIM} / \text{ARRAY}) - 1 / \text{DIM} \text{ARRAY}
[40] DIF \leftarrow (\Delta \text{DIM} = \text{RANK}) \Phi \text{DIM} \Phi . \times + ^{1} \text{LAST}

\[ \text{u Construct milky-way result:} \]
[41] R \leftarrow ((\text{RANK}, M, N, (\rho \text{DIM}), \text{DSHAPE}), \text{URRI} / \text{IO}), \text{DIM}

\[ \text{[WSID: REDUCE]} \]

\[ \nabla R \rightarrow L \Delta \text{ANDRED ARRAY}; \text{CIND}; \text{CUM}; \text{DIM}; \text{DSHAPE}; \text{GRADE}; \text{I}; \text{LAST}; M; N \]
; \text{RANK}; \text{RRI}; \text{SORTED}; \text{URRI}
[1] u No. of ways for N-way reduction:
[2] N \leftarrow (\rho, L) / 2
[3] u Which dimension to be "reduced"?
[4] DIM \leftarrow ((N + N) \# L), \text{DIM} \Phi \text{IO} \Phi 1 \Phi \text{RANK} \Phi \rho \text{ARRAY}
[5] u Branch unless 0-way reduction:
[6] \rightarrow (\times N) \rho L
[7] ARRAY \leftarrow \# / [\text{DIM} \text{ARRAY}]

\[ \text{u Construct milky-way result:} \]
[9] R \leftarrow ((\rho \text{ARRAY}), (\text{DIM} - \text{IO}), N, \rho \text{ARRAY}), \text{ARRAY}
[10] \rightarrow 0

\[ \text{u Separate left arg into its pieces:} \]
[12] L1 \leftarrow \text{DSHAPE} \# N \rho L
[13] CIND \leftarrow N \rho N \Phi L
[14] u Begin to compute "raveled result indices":
[15] I \leftarrow \text{IO}
[16] u Index from \text{DSHAPE}[I] to cause index error if
[17] \text{invalid indices:
[18] RRI \times (\text{DSHAPE}[I]) \Phi ' I', \Phi CIND[I]
[19] u Branch if origin is 1:
[20] \rightarrow \text{IO} \# \text{LOOP1}
[21] u Continue computing RRI (N iterations for
[22] u N-way reduction):
[23] LOOP0: \rightarrow (N \times I \Phi I + 1) \# \text{ENDLP}
[24] RRI \leftarrow (\text{DSHAPE}[I]) \Phi ' I', \Phi CIND[I]) \# \text{DSHAPE}[I] \times \text{RRI}
[25] \rightarrow \text{LOOP0}
[26] LOOP1: \rightarrow (N \times I \Phi I + 1) \# \text{ENDLP}
[27] RRI \leftarrow (\text{DSHAPE}[I]) \Phi ' I', \Phi CIND[I]) \# \text{DSHAPE}[I] \times \text{RRI} \times 1
[28] \rightarrow \text{LOOP1}

\[ \text{u Determine unique elements of RRI:} \]
[29] ENDLP: GRADE \leftarrow \Delta RRI
[30] SORTED \leftarrow RRI[GRADE]
[31] LAST \leftarrow SORTED \# 1 \Phi \text{SORTED}
\[
\text{\(\Delta{\text{ANDRED (continued)}}\)}
\]
\[33\]
\text{LAST}[(\times{\text{LAST}})\,\rho(\text{LAST})-\sim{\text{IO}}]\leftarrow 1
\]
\[34\]
\text{URRI}\leftarrow \text{LAST}/\text{SORTED}
\]
\[35\]
\text{\(\alpha\) Reorder ARRAY to conform with SORTED:}
\]
\[36\]
\text{M}\leftarrow \text{DIM-DIO}
\]
\[37\]
\text{ARRAY}\leftarrow \rho'\text{ARRAY}'\,\rho'\text{'GRADE}',((\text{RANK-M+1})\rho';''),'
\]
\[38\]
\text{\(\alpha\) Perform partitioned reduction (note: \(\wedge/\leftrightarrow \sim\vee/\sim\):)
\]
\[39\]
\text{CUM}\leftarrow \text{LAST}/[\text{DIM}]+\setminus [\text{DIM}]/\text{ARRAY}
\]
\[40\]
\text{CUM}\leftarrow \text{CUM}=(\rho\text{CUM})\uparrow 0,[\text{DIM}]\text{CUM}
\]
\[41\]
\text{\(\alpha\) Construct milky-way result:}
\]
\[42\]
\text{R}\leftarrow ((\text{RANK},\text{M},\text{N},(\rho\text{CUM}),\text{DSHAPE}),\text{URRI-DIO}),,\text{CUM}
\]
\[\text{\(\forall\)}
\]
\[\text{[WSID: REDUCE]}
\]
\[\text{\(\forall\)}
\]
\[\text{R}\leftarrow \text{L} \text{\(\Delta{\text{ORRED ARRAY;CIND;CUM;DIM;DSHAPE;GRADE;I;LAST;M;N;}}\)
\]
\[\text{\(\text{RANK;URRI;SORTED;URRI}
\]
\[1\]
\text{\(\alpha\) No. of ways for N-way reduction:}
\]
\[2\]
\text{N}\leftarrow (\rho,L)+2
\]
\[3\]
\text{\(\alpha\) Which dimension to be "reduced"?
\]
\[4\]
\text{DIM}\leftarrow 1\uparrow ((N+N)\downarrow L),\text{DIM}\downarrow 1\uparrow \text{RANK}+\rho\text{ARRAY
\]
\[5\]
\text{\(\alpha\) Branch unless 0-way reduction:
\]
\[6\]
\text{\(\rightarrow (\times N)\rho L\)
\]
\[7\]
\text{ARRAY}\leftarrow \nu/\,[\text{DIM}]\text{ARRAY
\]
\[8\]
\text{\(\alpha\) Construct milky-way result:}
\]
\[9\]
\text{R}\leftarrow ((\rho\text{ARRAY}),,(\text{DIM-DIO}),\text{N},\rho\text{ARRAY}),,\text{ARRAY}
\]
\[10\]
\text{\(\rightarrow 0
\]
\[11\]
\text{\(\alpha\) Separate left arg into its pieces:
\]
\[12\]
\text{L1}\leftarrow \text{DSHAPE}\leftarrow N\rho L
\]
\[13\]
\text{CIND}\leftarrow N\rho N\downarrow L
\]
\[14\]
\text{\(\alpha\) Begin to compute "raveled result indices":
\]
\[15\]
\text{I}\leftarrow \text{DIM}
\]
\[16\]
\text{\(\alpha\) Index from \text{DSHAPE}[I] to cause index error if
\]
\[17\]
\text{\(\alpha\) invalid indices:
\]
\[18\]
\text{RRI}\leftarrow (\text{DSHAPE}[I])[\#'I',\#\text{CIND}[I]]
\]
\[19\]
\text{\(\alpha\) Branch if origin is 1:
\]
\[20\]
\text{\(\rightarrow \text{DIM}\rho \text{LOOP1}
\]
\[21\]
\text{\(\alpha\) Continue computing RRI (N iterations for
\]
\[22\]
\text{\(\alpha\) N-way reduction:
\]
\[23\]
\text{\(\rightarrow (N<\text{I}\leftarrow \text{I}+1)\rho \text{ENDLP}
\]
\[24\]
\text{\(\alpha\) RRI}\leftarrow (\text{DSHAPE}[I])[([\#'I',\#\text{CIND}[I]])+\text{DSHAPE}[I]\times\text{RRI}
\]
\[25\]
\text{\(\rightarrow \text{LOOP0}
\]
\[26\]
\text{\(\rightarrow (N<\text{I}\leftarrow \text{I}+1)\rho \text{ENDLP}
\]
\[27\]
\text{\(\alpha\) RRI}\leftarrow (\text{DSHAPE}[I])[([\#'I',\#\text{CIND}[I]])+\text{DSHAPE}[I]\times\text{RRI}+^-1
\]
\[28\]
\text{\(\rightarrow \text{LOOP1}
\]
\[29\]
\text{\(\alpha\) Determine unique elements of RRI:
\]
\[30\]
\text{ENDLP:GRADE}\leftarrow \#\text{ARRI}
\]
\[31\]
\text{\(\alpha\) SORTED}\leftarrow \text{RRI}[\text{GRADE}
\]
\[32\]
\text{\(\alpha\) LAST}\leftarrow \text{SORTED}\#1\#\text{SORTED}
\]
\[33\]
\text{\(\alpha\) LAST}[(\times{\text{LAST}})\,\rho(\text{LAST})-\sim{\text{IO}}]\leftarrow 1
\]
\[34\]
\text{URRI}\leftarrow \text{LAST/\text{SORTED}
\]
\[35\]
\text{\(\alpha\) Reorder ARRAY to conform with SORTED:}
\]
\[36\]
\text{M}\leftarrow \text{DIM-DIO}
\]
\[37\]
\text{ARRAY}\leftarrow \rho'\text{ARRAY}'\,\rho'\text{'GRADE}',((\text{RANK-M+1})\rho';''),'
\]
\[A\] Perform partitioned reduction:

\[CUM\leftarrow LAST/DIM+\{DIM\}ARRAY\]
\[CUM\leftarrow CUM\leftarrow (\rho CUM)^+0,\{DIM\}CUM\]

\[R\leftarrow ((RANK,M,N,\langle CUM\rangle,DSHAPE),URRI\leftarrow DIO),,CUM\]

\[V\]

\[\text{WSID: REDUCE}\]

\[R+W \Delta PLUSWAY ARRAY;DIM;DS;GRADE;LAST;M;N;NDS;RANK;RRI;S;SORTED\]

\[\text{\textbf{1} Force } W \text{ to be a vector:}\]
\[W\leftarrow W\]

\[\text{\textbf{2} Rank; dimension reduced (origin 0); no. ways:}\]
\[RANK\leftarrow ARRAY[DIO]\]
\[DIM\leftarrow DIO+M\leftarrow ARRAY[DIO+1]\]
\[N\leftarrow ARRAY[DIO+2]\]

\[\text{\textbf{3} Shape of reduced array; new/old resulting shape}\]
\[S\leftarrow ARRAY[3+1RANK]\]
\[DS\leftarrow ARRAY[(3+RANK)+N]\]

\[\text{\textbf{4} New resulting shape of reduced dimension:}\]
\[NDS\leftarrow DS[W]\]

\[\text{\textbf{5} Branch unless 0-way reduction:}\]
\[\rightarrow (N\rho)\rho L0\]
\[\rightarrow 0\]

\[\text{\textbf{6} Result indices; reduced array:}\]
\[LO:RRI\leftarrow ARRAY[(3+RANK+N)+1S[DIM]]\]
\[ARRAY\leftarrow SP(3+RANK+N+S[DIM])\downarrow ARRAY\]

\[\text{\textbf{7} Treat special if } W \text{ is empty:}\]
\[\rightarrow (0\rho W)\rho L1\]
\[\rightarrow 0\]

\[\text{\textbf{8} Treat special if } W \text{ is } \rho DS \text{ (i.e. all dimensions):}\]
\[L1:\rightarrow ((\rho W)\rho DS)\rho L2\]
\[\rightarrow (W\leftarrow \rho DS)\rho L3\]

\[\text{\textbf{9} New result indices:}\]
\[L2:RRI\leftarrow NDS\downarrow (DS\uparrow RRI)[W:]\]

\[\text{\textbf{10} Determine unique elements of } RRI:\]
\[GRADE\leftarrow ARRI\]

\[\text{\textbf{11} SORTED\leftarrow RRI[GRADE]\}
\[LAST\leftarrow SORTED\downarrow 1\downarrow SORTED\]
\[\rightarrow (X\rho LAST)\rho (\rho LAST)\leftarrow DIO\leftarrow 1\]
\[\rightarrow RRI\leftarrow LAST\downarrow SORTED\]

\[\text{\textbf{12} Reorder ARRAY to conform with SORTED:}\]
\[\rightarrow ARRAY\leftarrow ARRAY[\rho LAST\leftarrow DIM]\]
\[ \text{\textcopyright PLUSWAY (continued)} \]

\[ \text{\textcopyright Insert result of partitioned reduction:} \]
\[ \text{\textcopyright 'R', (M', ';'), 'DIO+RRI', ((RANK-M+1)';', ')\rightarrow ARRAY'} \]

\[ \text{\textcopyright Reshape to desired shape (unravel NDS dim.s):} \]
\[ R\left((-M)\Phi NDS,1\downarrow M\Phi ARRAY\right)\rho R \]

\[ \text{\textcopyright REDUCE} \]

\[ R\leftarrow W \leftarrow W \text{\textcopyright MAXWAY ARRAY; DIF; DIM; DS; GRADE; LAST; M; N; NDS; RANK; RRI; S; SORTED} \]

\[ \text{\textcopyright Force W to be a vector:} \]
\[ W\leftarrow W \]

\[ \text{\textcopyright Rank; dimension reduced (origin 0); no. ways:} \]
\[ \text{\textcopyright RANK\rightarrow ARRAY[0]} \]
\[ \text{\textcopyright DIM\rightarrow DIO+M\rightarrow ARRAY[DIO+1]} \]
\[ \text{\textcopyright N\rightarrow ARRAY[DIO+2]} \]

\[ \text{\textcopyright Shape of reduced array; new/old resulting shape} \]
\[ \text{\textcopyright of reduced dimension:} \]
\[ S\rightarrow ARRAY[3+\text{RANK}] \]
\[ DS\rightarrow ARRAY[(3+\text{RANK})+\text{DIM}] \]

\[ \text{\textcopyright New resulting shape of reduced dimension:} \]
\[ N\rightarrow DS[w] \]

\[ \text{\textcopyright Branch unless 0-way reduction:} \]
\[ \rightarrow (\times N)\rho L0 \]
\[ R\leftarrow S\rho (3+\text{RANK})\uparrow \text{ARRAY} \]
\[ \rightarrow 0 \]

\[ \text{\textcopyright Result indices; reduced array:} \]
\[ L0\leftarrow RRI\rightarrow ARRAY[(3+\text{RANK}+N)+\text{DIM}] \]
\[ \text{\textcopyright ARRAY\rightarrow S\rho (3+\text{RANK}+N+\text{DIM})\uparrow \text{ARRAY}} \]

\[ \text{\textcopyright Treat special if W is empty:} \]
\[ \rightarrow (0 \rho W)\rho L1 \]

\[ \text{\textcopyright Treat special if W is } \iota \rho DS \text{ (i.e. all dimensions):} \]
\[ L1\rightarrow ((\rho W)^{\iota} \rho DS)\rho L2 \]
\[ \rightarrow (W^{\iota} \iota \rho DS)\rho L3 \]

\[ \text{\textcopyright New result indices:} \]
\[ L2\leftarrow RRI\leftarrow DS^{\uparrow} RRI[w;] \]

\[ \text{\textcopyright Determine unique elements of RRI:} \]
\[ \text{\textcopyright GRADE\rightarrow RRI} \]
\[ \text{\textcopyright SORTED\rightarrow RRI[GRADE]} \]

\[ \text{\textcopyright LAST\rightarrow SORTED}\iota \iota \text{SORTED} \]
\[ \text{\textcopyright LAST} \iota \rho \text{LAST} \iota -\iota \text{DIM} \iota 1 \]
\[ \text{\textcopyright RRI\iota LAST/SORTED} \]

\[ \text{\textcopyright Reorder ARRAY to conform with SORTED:} \]
\[ \text{\textcopyright ARRAY\rightarrow 'ARRAY', (M', ';'), 'GRADE', ((RANK-M+1)';', ')'}} \]

\[ \text{\textcopyright Perform partitioned reduction:} \]
\[ \text{\textcopyright DIF\rightarrow (L/\text{DIM} ARRAY)\iota -\text{DIM} ARRAY} \]
\[ \text{\textcopyright DIF\rightarrow (\Delta DIM=\text{DIM})\#DIF, x\downarrow -1\Phi \text{LAST} \]
\[ \text{\textcopyright ARRAY\rightarrow (\text{LAST}/\text{DIM})\\text{DIM ARRAY}DIF -\text{DIM} DIF \]

\[ \text{\textcopyright Initialize result. Fill with identity elt.} \]
\[ \text{\textcopyright Ravel the DSHAPE dim.s:} \]
\[ L3\leftarrow R\left((-M)\Phi (x/NDS),1\downarrow M\Phi ARRAY\right)\rho 1/0 \]
\[ \triangle \text{MAXWAY (continued)} \]

\[ \text{[44]} \] A Insert result of partitioned reduction:
\[ \text{[45]} \] \text{\$'R'[,',(M_p';')',\'_O+RRI',,(RANK-M+1)p';')',']'\rightarrow ARRAY'} \]

\[ \text{[46]} \] A Reshape to desired shape (unravel NDS dim.s):
\[ \text{[47]} \] \text{R\((\mathbf{\rightarrow})(-M)\Phi NDS,1\Phi M\Phi ARRAY)\Phi R} \]

\[ \text{[WSID: REDUCE]} \]

\[ \text{\$ R\=W \triangle \text{MINWAY ARRAY;DIF;DIM;DS;GRADE;LAST;M;N;NDS;RANK; RRI;S;SORTED}} \]

\[ \text{[1]} \] A Force W to be a vector:
\[ \text{[2]} \] W\(\mathbf{\rightarrow}W \)

\[ \text{[3]} \] A Rank; dimension reduced (origin 0); no. ways:
\[ \text{[4]} \] RANK\(\rightarrow\text{ARRAY}[D_{IO}] \)
\[ \text{[5]} \] DIM\(\rightarrow\text{DIM}+M\rightarrow\text{ARRAY}[[\text{DIM}]+1] \)
\[ \text{[6]} \] N\(\rightarrow\text{ARRAY}[[\text{DIM}]+2] \)

\[ \text{[7]} \] A Shape of reduced array; new/old resulting shape
\[ \text{[8]} \] A of reduced dimension:
\[ \text{[9]} \] S\(\rightarrow\text{ARRAY}[3+1\text{RANK}] \)
\[ \text{[10]} \] DS\(\rightarrow\text{ARRAY}[(3+\text{RANK})+\text{N}] \)

\[ \text{[11]} \] A New resulting shape of reduced dimension:
\[ \text{[12]} \] NDS\(\rightarrow\text{DS}[W] \)

\[ \text{[13]} \] A Branch unless 0-way reduction:
\[ \text{[14]} \] \(\rightarrow(\times N)\rho L_0 \)

\[ \text{[15]} \] R\(\Phi (3+\text{RANK})\\downarrow\text{ARRAY} \)

\[ \text{[16]} \] \(\rightarrow0 \)

\[ \text{[17]} \] A Result indices; reduced array:
\[ \text{[18]} \] L_0:RRI\(\rightarrow\text{ARRAY}[(3+\text{RANK}+\text{N})+1\text{DIM}] \)
\[ \text{[19]} \] ARRAY\(\Phi (3+\text{RANK}+\text{N}+\text{DIM})\downarrow\text{ARRAY} \)

\[ \text{[20]} \] A Treat special if W is empty:
\[ \text{[21]} \] \(\rightarrow(0\neq\Phi W)\rho L_1 \)

\[ \text{[22]} \] R\(\Phi l/[\text{DIM}]\text{ARRAY} \)

\[ \text{[23]} \] \(\rightarrow0 \)

\[ \text{[24]} \] A Treat special if W is \(\neq\Phi\text{DS}\) (i.e. all dimensions):
\[ \text{[25]} \] L_1:R\(\rightarrow((\Phi W)\neq\Phi\text{DS})\rho L_2 \)

\[ \text{[26]} \] \(\rightarrow(W=\neq\Phi\text{DS})\rho L_3 \)

\[ \text{[27]} \] A New result indices:
\[ \text{[28]} \] L_2:RRI\(\rightarrow\text{NDS}((\text{DSTRRI})[W];) \)

\[ \text{[29]} \] A Determine unique elements of RRI:
\[ \text{[30]} \] GRADE\(\rightarrow\text{RRI} \)
\[ \text{[31]} \] SORTED\(\rightarrow\text{RRI}[\text{GRADE}] \)
\[ \text{[32]} \] LAST\(\rightarrow\text{SORTED}\Phi 1\Phi \text{SORTED} \)
\[ \text{[33]} \] LAST\(\rightarrow(\times \text{RANK})\rho (\text{RANK}\Phi 1) = \text{DIM} \)
\[ \text{[34]} \] RRI\(\rightarrow\text{LAST}/\text{SORTED} \)

\[ \text{[35]} \] A Reorder ARRAY to conform with SORTED:
\[ \text{[36]} \] ARRAY\(\rightarrow\text{ARRAY}[',(M_p';')',\text{GRADE}',((\text{RANK-M+1})p';')',']' \)

\[ \text{[37]} \] A Perform partitioned reduction:
\[ \text{[38]} \] DIF\(\rightarrow(\mathbf{/[DIM]}\text{ARRAY})-\mathbf{/[DIM]}\text{ARRAY} \)
\[ \text{[39]} \] DIF\(\Phi (\mathbf{4\times DIM}=(\text{RANK})\Phi \text{DIMo.x}+)\}\Phi 1\Phi \text{LAST} \)
\[ \text{[40]} \] ARRAY\(\rightarrow(\text{LAST}/\mathbf{/[DIM]}\mathbf{/[DIM]}\text{ARRAY}-\text{DIF})+\text{LAST}/\mathbf{/[DIM]}\text{DIF} \)

\[ \text{[41]} \] A Initialize result. Fill with identity elt.
\[ \text{[42]} \] A Ravel the DSHAPE dim.s:
\[ \text{[43]} \] L_3:R\(\rightarrow((\mathbf{-M})\Phi (x/NDS),1\Phi M\Phi \text{ARRAY})\Phi l/10 \)
\[ Δ\text{MINWAY} \text{ (continued)} \]

[44] \( \text{Insert result of partitioned reduction:} \)

[45] \( \text{'R'}, \text{(Mp'; ')}, \text{DIM+RRI', ((RANK-M+1)p'; ')}) \leftrightarrow \text{ARRAY'} \)

[46] \( \text{Reshape to desired shape (unravel NDS dim.s):} \)

[47] \( R \leftarrow (-M)Φ NDS, 1↓MΦ \text{ARRAY} \rho R \)

\[ \text{[WSID: REDUCE]} \]

\[ \text{∨} \]

\( R \leftarrow W \Δ\text{ANDWAY ARRAY}; \text{DIM; DS; GRADE; LAST; M; N; NDS; RANK; RRI; S} \)

; \text{SORTED} \]

[1] \( \text{Force W to be a vector:} \)

[2] \( W \leftarrow W \)

[3] \( \text{Rank; dimension reduced (origin 0); no. ways:} \)

[4] \( \text{RANK} \leftarrow \text{ARRAY}[\text{DIM}] \)

[5] \( \text{DIM} \leftarrow \text{DIM}+\text{M} \leftarrow \text{ARRAY}[\text{DIM}+1] \)

[6] \( \text{N} \leftarrow \text{ARRAY}[\text{DIM}+2] \)

[7] \( \text{Shape of reduced array; new/old resulting shape} \)

[8] \( \text{of reduced dimension:} \)

[9] \( \text{S} \leftarrow \text{ARRAY}[1+\text{RANK}] \)

[10] \( \text{DS} \leftarrow \text{ARRAY}[(3+\text{RANK})+\text{N}] \)

[11] \( \text{New resulting shape of reduced dimension:} \)

[12] \( \text{NDS} \leftarrow \text{DS}[\text{W}] \)

[13] \( \text{Branch unless 0-way reduction:} \)

[14] \( \rightarrow (\times N)p L0 \)

[15] \( R \leftarrow S_ρ(3+\text{RANK})↓\text{ARRAY} \)

[16] \( \rightarrow 0 \)

[17] \( \text{Result indices; reduced array:} \)

[18] \( \text{L0} : \text{RRI} \leftarrow \text{ARRAY}[3\text{+RANK}+\text{DIM}] \)

[19] \( \text{ARRAY} \leftarrow S_ρ(3+\text{RANK}+\text{S}[\text{DIM}])↓\text{ARRAY} \)

[20] \( \text{Treat special if W is empty:} \)

[21] \( \rightarrow (0\# W)p L1 \)

[22] \( R \leftarrow /[\text{DIM}]\text{ARRAY} \)

[23] \( \rightarrow 0 \)

[24] \( \text{Treat special if W is Lρ DS (i.e. all dimensions):} \)

[25] \( \text{L1} : \rightarrow ((\text{W})\# ρ DS)p L2 \)

[26] \( \rightarrow (\text{W} .= Lρ DS)p L3 \)

[27] \( \text{New result indices:} \)

[28] \( \text{L2} : \text{RRI} \leftarrow \text{NDS}↓(\text{DS}ρ\text{RRI})[\text{W};] \)

[29] \( \text{Determine unique elements of RRI:} \)

[30] \( \text{GRADE} \leftarrow \text{RRI} \)

[31] \( \text{SORTED} \leftarrow \text{RRI}[\text{GRADE}] \)

[32] \( \text{LAST} \leftarrow \text{SORTED} \# 1Φ \text{SORTED} \)

[33] \( \text{LAST}[(\times ρ \text{LAST})ρ(ρ \text{LAST})\# -\text{DIM}]p L1 \)

[34] \( \text{RRI} \leftarrow \text{LAST} / \text{SORTED} \)

[35] \( \text{Reorder ARRAY to conform with SORTED:} \)

[36] \( \text{ARRAY} \leftarrow 'ARRAY', (\text{M}p'; ')', \text{GRADE}', ((\text{RANK}+\text{DIM})p'; ')', ' \)

[37] \( \text{Perform partitioned reduction:} \)

[38] \( \text{ARRAY} \leftarrow \text{LAST} / [\text{DIM}] + [\text{DIM}] \leftarrow \text{ARRAY} \)

[39] \( \text{ARRAY} \leftarrow \text{ARRAY} \leftarrow (ρ \text{ARRAY})↑0, [\text{DIM}]\text{ARRAY} \)

[40] \( \text{Initialize result. Fill with identity elt.} \)

[41] \( \text{Ravel the DSHAPE dim.s:} \)

[42] \( L3 : R \leftarrow (-M)Φ(\times / \text{NDS}), 1↓Mρ \text{ARRAY} \rho 1 \)
\[ \text{\textcopyright ANDWAY (continued)} \]

[43] \text{a Insert result of partitioned reduction:}
[44] \text{\textquoteleft R', (M', '}'), '\text{\textquoteleft Rlí+RRI', ((RANK-M+1)'},>';'), '{\text{ARRAY'}}
[45] \text{a Reshape to desired shape (unravel NDS dim.s):}
[46] R\rightarrow(-M)\Phi NDS, 1\downarrow M\Phi \text{ARRAY})\rho R

\[ \text{[WSID: REDUCE]} \]

\[ R\leftarrow W \text{\textcopyright ANDWAY ARRAY; DIM; DS; GRADE; LAST; } M; N; NDS; RANK; RRI; S; \text{ SORTED} \]

[1] \text{a Force W to be a vector:}
[2] W\rightarrow W
[3] \text{a Rank; dimension reduced (origin 0); no. ways:}
[4] RANK\rightarrow ARRAY[OIO]
[5] DIM\rightarrow OIO+M\rightarrow ARRAY[OIO+1]
[6] N\rightarrow ARRAY[OIO+2]
[7] \text{a Shape of reduced array; new/old resulting shape}
[8] \text{a of reduced dimension:}
[9] S\rightarrow ARRAY[3+1RANK]
[10] DS\rightarrow ARRAY[(3+RANK)+1N]
[11] \text{a New resulting shape of reduced dimension:}
[12] NDS\rightarrow DS[W]
[13] \text{a Branch unless 0-way reduction:}
[14] \rightarrow(\times N)\rho L0
[15] R\rightarrow S\rho (3+RANK)\downarrow ARRAY
[16] \rightarrow 0
[17] \text{a Result indices; reduced array:}
[18] L0:RRI\rightarrow ARRAY[(3+RANK+N)+1S[DIM]]
[19] ARRAY\leftarrow S\rho (3+RANK+N+S[DIM])\downarrow ARRAY
[20] \text{a Treat special if W is empty:}
[21] \rightarrow(0\neq \rho W)\rho L1
[22] R\leftarrow V/[DIM]ARRAY
[23] \rightarrow 0
[24] \text{a Treat special if W is } \rho DS \text{ (i.e. all dimensions):}
[25] L1: \rightarrow(\rho W)\neq DS)\rho L2
[26] \rightarrow(W. = \rho DS)\rho L3
[27] \text{a New result indices:}
[28] L2:RRI\rightarrow NDS\downarrow (DS\uparrow RRI)[W;]
[29] \text{a Determine unique elements of RRI:}
[30] GRADE\rightarrow \# RRI
[31] SORTED\rightarrow RRI[GRADE]
[32] LAST\rightarrow SORTED\neq 1\Phi SORTED
[33] LAST[\times \rho LAST)\rho (\rho LAST)\sim OIO]\rightarrow 1
[34] RRI\rightarrow LAST/SORTED
[35] \text{a Reorder ARRAY to conform with SORTED:}
[36] ARRAY\leftarrow \# ARRAY', (M', ')
[37] \text{a Perform partitioned reduction:}
[38] ARRAY\leftarrow LAST/[DIM]+\\downarrow [DIM]ARRAY
[39] ARRAY\leftarrow ARRAY\#(\rho ARRAY)\uparrow 0, [DIM]ARRAY
[40] \text{a Initialize result. Fill with identity elt.}
[41] \text{a Ravel the DSHAPE dim.s:}
[42] L3:R\rightarrow(-M)\Phi (\times /NDS), 1\downarrow M\Phi \text{ARRAY})\rho 0
Chapter 7 Solutions FREQUENCY COUNTS, ACCUMULATIONS AND CROSS-TABS

\[ \Delta \text{ORWAY (continued)} \]

[43] \( & \text{Insert result of partitioned reduction:} \\[44] \) \( \text{'R'}, (\rho';')', \text{IIO} + \text{RRI}', ((\text{RANK} - \text{M} + 1)\rho';'), 'I' \rightarrow \text{ARRAY} \)

[45] \( & \text{Reshape to desired shape (unravel NDS dim.s):} \\[46] \) \( R \rightarrow (\text{M} \phi \text{NDS}, \text{I} \downarrow \text{M} \phi \text{ARRAY}) \rho R \)

[WSID: REDUCE]

\[ \Delta \text{PLUS B;ARRAY;CUM;DIM;DS;GRADE;LAST;N;RANK;RRI;S;SORTED;URRI;\text{IIO}} \]

[1] \( & \text{Adds together two compressed Milky-Way arrays,} \\[2] \) \( & \text{returning a third such array.} \\[3] \) \( & \)

[4] \( & \text{Return left argument if right is empty:} \\[5] \) \( \rightarrow (0 \epsilon \rho B) \downarrow L1 \)

[6] \( R \leftarrow A \)

[7] \( \rightarrow 0 \)

[8] \( & \text{Return right if left is empty:} \\[9] \) \( L1 : \rightarrow (0 \epsilon \rho A) \downarrow L2 \)

[10] \( R \leftarrow B \)

[11] \( \rightarrow 0 \)

[12] \( & \text{Extract components from left argument:} \\[13] \) \( L2 : I \leftarrow 0 \)

[14] \( \text{RANK} \leftarrow A[0] \)


[17] \( & \text{Branch unless a 0-way reduction:} \\[18] \) \( \rightarrow (x N) \rho L3 \)

[19] \( R \leftarrow ((3 + \text{RANK}) \rho A), ((3 + \text{RANK}) \downarrow A) + (3 + \text{RANK}) \downarrow B \)

[20] \( \rightarrow 0 \)

[21] \( L3 : S \leftarrow A[3 + l \text{RANK}] \)

[22] \( \text{DS} \leftarrow A[(3 + \text{RANK}) + l N] \)

[23] \( \text{RRI} \leftarrow A[(3 + \text{RANK} + N) + l S[\text{DIM}]] \)

[24] \( \text{ARRAY} \leftarrow S \rho (3 + \text{RANK} + N + S[\text{DIM}]) \downarrow A \)

[25] \( & \text{Include components from right argument:} \\[26] \) \( S \leftarrow B[3 + l \text{RANK}] \)

[27] \( \text{RRI} \leftarrow \text{RRI}, B[(3 + \text{RANK} + N) + l S[\text{DIM}]] \)

[28] \( \text{ARRAY} \leftarrow \text{ARRAY}, [\text{DIM}] S \rho (3 + \text{RANK} + N + S[\text{DIM}]) \downarrow B \)

[29] \( & \text{Use same logic as in } \Delta \text{PLUSRED:} \\[30] \) \( \text{GRADE} \leftarrow \Delta \text{RRI} \)

[31] \( \text{SORTED} \leftarrow \text{RRI}[\text{GRADE}] \)

[32] \( \text{LAST} \leftarrow \text{SORTED} \downarrow \# \text{SORTED} \)

[33] \( \text{LAST} : (x \rho \text{LAST}) \rho^{-1} \rho \text{LAST} \leftarrow 1 \)

[34] \( \text{URRI} \leftarrow \text{LAST} / \text{SORTED} \)

[35] \( \text{ARRAY} \leftarrow a' \text{ARRAY}' [', (\text{DIM} \rho';')', ' \text{GRADE}', ((\text{RANK} - \text{DIM} + 1) \rho';'), ']' \)

[36] \( \text{CUM} \leftarrow \text{LAST} /[\text{DIM}] + /[\text{DIM}] \text{ARRAY} \)

[37] \( \text{CUM} \leftarrow \text{CUM} - (\rho \text{CUM}) \uparrow 0, [\text{DIM}] \text{CUM} \)

[38] \( R \leftarrow ((\text{RANK}, \text{DIM}, N, (\rho \text{CUM}), \text{DS}), \text{URRI}), , \text{CUM} \)

\[ \]
[WSID: REDUCE]

\[ R + A \ \Delta \text{MAX} B; \text{ARRAY}; \text{DIF}; \text{DIM}; \text{DS}; \text{GRADE}; \text{LAST}; \text{N}; \text{RANK}; \text{RRI}; \text{S}; \text{SORTED}; \text{URRI}; \text{OO} \]

[1] \( \text{a} \) Adds together two compressed Milky-Way arrays, returning a third such array.
[2] \( \text{a} \)

[3] \( \text{a} \) Return left argument if right is empty:
[4] \( \rightarrow (0 \in \rho B) \downarrow \text{L1} \)
[5] \( R + A \)
[6] \( \rightarrow 0 \)
[7] \( \text{a} \) Return right if left is empty:
[8] \( \text{L1} \rightarrow (0 \in \rho A) \downarrow \text{L2} \)
[9] \( R + B \)
[10] \( \rightarrow 0 \)

[12] \( \text{a} \) Extract components from left argument:
[13] \( \text{L2} \rightarrow \text{OO} \rightarrow 0 \)
[14] \( \text{RANK} \rightarrow A[0] \)
[17] \( \text{a} \) Branch unless a 0-way reduction:
[18] \( \rightarrow (\times N) \rho \text{L3} \)
[19] \( R + ((3 + \text{RANK}) \rho A),((3 + \text{RANK}) \downarrow A)\downarrow (3 + \text{RANK}) \downarrow B \)
[20] \( \rightarrow 0 \)
[21] \( \text{L3} \rightarrow S \rightarrow A[3 + \text{RANK}] \)
[22] \( \text{DS} \rightarrow A[(3 + \text{RANK}) + 1 \text{N}] \)
[23] \( \text{RRI} \rightarrow A[(3 + \text{RANK} + 1) + 1 \text{S}[\text{DIM}]] \)
[24] \( \text{ARRAY} \rightarrow S \rightarrow (3 + \text{RANK} + 1 + 1 \text{S}[\text{DIM}]) \downarrow A \)
[25] \( \text{a} \) Include components from right argument:
[26] \( \text{S} \rightarrow B[3 + \text{RANK}] \)
[27] \( \text{RRI} \rightarrow RRI, B[(3 + \text{RANK} + 1) + 1 \text{S}[\text{DIM}]] \)
[28] \( \text{ARRAY} \rightarrow \text{ARRAY}, [\text{DIM}] S \rightarrow (3 + \text{RANK} + 1 + 1 \text{S}[\text{DIM}]) \downarrow B \)
[29] \( \text{a} \) Use same logic as in \( \triangle \text{PLUSRED} \):
[30] \( \text{GRADE} \rightarrow \text{ARRI} \)
[31] \( \text{SORTED} \rightarrow \text{RRI}[\text{GRADE}] \)
[32] \( \text{LAST} \rightarrow \text{SORTED} \rightarrow 1 \rightarrow \text{SORTED} \)
[33] \( \text{LAST} \rightarrow (\times N) \rho \text{LAST} \rightarrow 1 + 1 \rho \text{LAST} + 1 \)
[34] \( \text{URRI} \rightarrow \text{LAST} \rightarrow \text{SORTED} \)
[35] \( \text{ARRAY} \rightarrow S \rightarrow ' \text{ARRAY}[',(\text{DIM} \rho ' ; ');', ' \text{GRADE}',((\text{RANK} + 1 \rho ' ; '),',']' \)
[36] \( \text{DIF} \rightarrow ((/ \times [\text{DIM}] \text{ARRAY}) - 1) / ([\text{DIM}] \text{ARRAY}) \)
[37] \( \text{DIF} \rightarrow (\triangle \text{DIM} = \text{RANK}) \triangle \text{DIF}_{x} + \triangle \text{LAST} \)
[38] \( \text{DIF} \rightarrow (\text{LAST} / ([\text{DIM}] \downarrow [\text{DIM}] \text{ARRAY} + \text{DIF}) - \text{LAST} / [\text{DIM}] \text{DIF} \)
[39] \( R + ((\text{RANK}, \text{DIM}, N, (\rho \text{DIF}), \text{DS}), \text{URRI}), \text{DIF} \)

[WSID: REDUCE]

\[ R + A \ \Delta \text{MIN} B; \text{ARRAY}; \text{DIF}; \text{DIM}; \text{DS}; \text{GRADE}; \text{LAST}; \text{N}; \text{RANK}; \text{RRI}; \text{S}; \text{SORTED}; \text{URRI}; \text{OIO} \]

[1] \( \text{a} \) Adds together two compressed Milky-Way arrays, returning a third such array.
[2] \( \text{a} \)

[3] \( \text{a} \) Return left argument if right is empty:
[4] \( \rightarrow (0 \in \rho B) \downarrow \text{L1} \)
\[ \nabla \Delta_{\text{MIN}} \text{(continued)} \]

[6] \( R \leftarrow A \)
[7] \( \rightarrow 0 \)
[8] \( \nabla \text{Return right if left is empty:} \)
[9] \( L_1 \rightarrow (0 \in \rho A) \rightarrow L_2 \)
[10] \( R \leftarrow B \)
[11] \( \rightarrow 0 \)
[12] \( \nabla \text{Extract components from left argument:} \)
[13] \( L_2 : \emptyset \rightarrow 0 \)
[14] \( \nabla \text{RANK} \leftarrow A[0] \)
[15] \( \nabla \text{DIM} \leftarrow A[1] \)
[17] \( \nabla \text{Branch unless a 0-way reduction:} \)
[18] \( \rightarrow (\times N) \rho L_3 \)
[19] \( R \leftarrow ((3 + \text{RANK}) \rho A),((3 + \text{RANK}) \downarrow A)(3 + \text{RANK}) \downarrow B \)
[20] \( \rightarrow 0 \)
[21] \( L_3 : S + A[3 + \text{RANK}] \)
[22] \( \nabla \text{DIM} \leftarrow A[3 + \text{RANK}] + N \)
[23] \( \nabla \text{RANK} + A[3 + \text{RANK} + N] + S[\text{DIM}] \)
[24] \( \nabla \text{ARRAY} \leftarrow S \rho (3 + \text{RANK} + N + S[\text{DIM}]) \downarrow A \)
[25] \( \nabla \text{Include components from right argument:} \)
[26] \( S \leftarrow B[3 + \text{RANK}] \)
[27] \( \nabla \text{RANK} \leftarrow \text{RANK} + B[3 + \text{RANK} + N] + S[\text{DIM}] \)
[28] \( \nabla \text{ARRAY} \leftarrow \text{ARRAY},[\text{DIM}] \rho (3 + \text{RANK} + N + S[\text{DIM}]) \downarrow B \)
[29] \( \nabla \text{Use same logic as in } \Delta_{\text{PLUSRED}}: \)
[30] \( \nabla \text{GRADE} \leftarrow \text{RANK} \)
[31] \( \nabla \text{SORTED} \leftarrow \text{RANK}[\text{GRADE}] \)
[32] \( \nabla \text{LAST} \leftarrow \text{SORTED} \neq 1 \) \( \nabla \text{SORTED} \)
[33] \( \nabla \text{LAST} \leftarrow (\times \text{LAST}) \rho^{-1} \rho \text{LAST} \leftarrow 1 \)
[34] \( \nabla \text{URRI} \leftarrow \text{LAST} / \text{SORTED} \)
[35] \( \nabla \text{ARRAY} \leftarrow \text{ARRAY}[',(\text{DIM}';',')','\text{GRADE}',((\text{RANK} - \text{DIM} + 1) \rho ';''),']' \)
[36] \( \nabla \text{DIF} \leftarrow ([/\text{DIM}] \text{ARRAY}) - [1/\text{DIM}] \text{ARRAY} \)
[37] \( \nabla \text{DIF} \leftarrow (\Delta \text{DIM} = \text{RANK}) \) \( \nabla \text{DIF} \leftarrow . X + 1 \) \( \nabla \text{LAST} \)
[38] \( \nabla \text{DIF} \leftarrow (\text{LAST} / \text{DIM}) \backslash ([\text{DIM}] \text{ARRAY} - \text{DIF}) + \text{LAST} / ([\text{DIM}] \text{DIF} \)
[39] \( R \leftarrow ((\text{RANK}, \text{DIM}, N, (\rho \text{DIF}), \text{DS}), \text{URRI}), \text{DIF} \)

[WSID: REDUCE]

\[ \nabla \text{R} \leftarrow A \) \( \Delta \) \( \text{AND} \) \( B; \text{ARRAY; CUM; DIM; DS; GRADE; LAST; N; RANK; RRI; S; \) \text{SORTED; URRI; } \emptyset \]

[1] \( \nabla \text{Adds together two compressed Milky-Way arrays, } \)
[2] \( \nabla \text{returning a third such array.} \)
[3] \( \nabla \)
[4] \( \nabla \text{Return left argument if right is empty:} \)
[5] \( \rightarrow (0 \in \rho B) \rightarrow L_1 \)
[6] \( R \leftarrow A \)
[7] \( \rightarrow 0 \)
[8] \( \nabla \text{Return right if left is empty:} \)
[9] \( L_1 \rightarrow (0 \in \rho A) \rightarrow L_2 \)
[10] \( R \leftarrow B \)
[11] \( \rightarrow 0 \)

-358-
\( \Delta AND \) (continued)

12. Extract components from left argument:
13. \( L2: DIO + 0 \)
14. \( RANK \leftarrow A[0] \)
15. \( DIM \leftarrow A[1] \)
17. Branch unless a 0-way reduction:
18. \( (\times N)^p L3 \)
19. \( R \leftarrow ((3 + RANK)^p A), ((3 + RANK)^p A) \lor (3 + RANK)^p B \)
20. \( \rightarrow 0 \)
21. \( L3: S \leftarrow A[3 + \lambda RANK] \)
22. \( DS \leftarrow A[(3 + RANK) + \lambda N] \)
23. \( RRI \leftarrow A[(3 + RANK) + N + \lambda S[DIM]] \)
24. \( ARRAY \leftarrow S^\rho (3 + RANK + N + S[DIM])^p A \)
25. Include components from right argument:
26. \( S \leftarrow B[3 + \lambda RANK] \)
27. \( RRI \leftarrow RRI,B[(3 + RANK + N) + \lambda S[DIM]] \)
28. \( ARRAY \leftarrow ARRAY,[DIM]S^\rho (3 + RANK + N + S[DIM])^p B \)
29. Use same logic as in \( \Delta PLUSRED : \)
30. \( GRADE \leftarrow \Delta RRI \)
31. \( SORTED \leftarrow RRI[GRADE] \)
32. \( LAST \leftarrow SORTED \lor 1 \) \( \triangleright \) \( SORTED \)
33. \( LAST[(\times p LAST)\rho^{-1} + p LAST] \leftarrow 1 \)
34. \( URRRI \leftarrow LAST \lor SORTED \)
35. \( ARRAY \leftarrow \#'\text{ARRAY}',(DIM\rho':''),'GRADE',((RANK-DIM+1)^p ';')',' \)
36. \( CUM \leftarrow LAST / [DIM]+[DIM] \leftarrow \text{ARRAY} \)
37. \( CUM \leftarrow CUM \leftarrow (\rho CUM) \lor 0,[DIM] \leftarrow \text{CUM} \)
38. \( R \leftarrow ((RANK, DIM, N, (\rho CUM), DS), URRRI), \text{CUM} \)

\[ \text{[WSID: REDUCE]} \]

\( \Delta AND \) \( \Delta OR \) \( B; ARRAY; \text{CUM}; DIM; DS; GRADE; LAST; N; RANK; RRI; S; \)
\( \text{SORTED; URRRI; DIO} \)

1. Adds together two compressed Milky-Way arrays, returning a third such array.
2. Return left argument if right is empty:
3. \( (0 \in p B) \rightarrow L1 \)
4. \( R \leftarrow A \)
5. \( \rightarrow 0 \)
6. Return right if left is empty:
7. \( L1 \rightarrow (0 \in p A) \rightarrow L2 \)
8. \( R \leftarrow B \)
9. \( \rightarrow 0 \)
10. Extract components from left argument:
11. \( L2: DIO + 0 \)
12. \( RANK \leftarrow A[0] \)
13. \( DIM \leftarrow A[1] \)
15. Branch unless a 0-way reduction:
16. \( (\times N)^p L3 \)
17. \( R \leftarrow ((3 + RANK)^p A), ((3 + RANK)^p A) \lor (3 + RANK)^p B \)
\[ \Delta OR \text{ (continued)} \]

\[ L3: S \rightarrow A[3 + l\text{RANK}] \]
\[ DS \rightarrow A[(3 + \text{RANK}) + lN] \]
\[ RRI \rightarrow A[(3 + \text{RANK}) + lS[DIM]] \]
\[ \text{ARRAY} \rightarrow S_p(3 + \text{RANK} + N + S[DIM]) \downarrow A \]
\[ \text{Include components from right argument:} \]
\[ S \rightarrow B[3 + l\text{RANK}] \]
\[ RRI \rightarrow RRI, B[(3 + \text{RANK} + N) + lS[DIM]] \]
\[ \text{ARRAY} \rightarrow \text{ARRAY}, [\text{DIM}] S_p(3 + \text{RANK} + N + S[DIM]) \downarrow B \]
\[ \text{Use same logic as in } \Delta \text{PLUSRED:} \]
\[ \text{GRADE} \rightarrow \uparrow \text{RRI} \]
\[ \text{SORTED} \rightarrow \text{RRI[GRADE]} \]
\[ \text{LAST} \rightarrow \text{SORTED} \downarrow \text{SORTED} \]
\[ \text{LAST} \rightarrow \text{LAST} \downarrow \text{SORTED} \]
\[ \text{URRI} \rightarrow \text{LAST} \downarrow \text{SORTED} \]
\[ \text{ARRAY} \rightarrow s' \text{ARRAY}', (\text{DIM}', ' \text{GRADE}', ((\text{RANK} - \text{DIM} + 1)'', ')', '')' \]
\[ \text{CUM} \rightarrow \text{LAST} /[\text{DIM}] + /[\text{DIM}] \text{ARRAY} \]
\[ \text{CUM} \rightarrow \text{CUM}', (\text{DIM}', \text{CUM}) \uparrow 0, [\text{DIM}] \text{CUM} \]
\[ R \rightarrow (\text{RANK}, \text{DIM}, N, (\rho \text{CUM}), DS), URRI), \text{CUM} \]

6.

\[ IO \rightarrow 1 \]
\[ I1 \rightarrow 'ECMPH', 'TZONE \quad (5 \text{ classes}) \]
\[ I2 \rightarrow 'BCPS', 'TYPE \quad (4 \text{ classes}) \]
\[ I3 \rightarrow +/ \text{SALES} \geq 0 \quad 1E6 5E6 \quad (3 \text{ classes}) \]
\[ \text{(or: } I3 \rightarrow 0 \quad 1E6 5E6 \text{ LIOTA SALES) } \]
\[ \text{(LIOTA is defined in Sorting and Searching chapter) } \]
\[ I4 \rightarrow \text{ALLSTATES} \quad \text{CMIOTA STATES} \quad (50 \text{ classes}) \]
\[ \text{(CMIOTA is defined in Sorting and Searching chapter) } \]
\[ I5 \rightarrow 301 \quad 304 \quad 310 \quad 322 \quad 329 \quad \text{MGR} \quad (6 \text{ classes}) \]

\[ \text{SUM} \rightarrow (5 \quad 4 \quad 3 \quad 50 \quad 6, 1 \quad 2 \quad 3 \quad 4 \quad 5) \Delta \text{PLUSRED } 1, [1] \text{SALES, [.5]FIT} \]
\[ \alpha \text{ Returns a 3 (frequency, SALES, FIT) by 5 by 4 by 3 by 50 } \]
\[ \alpha \text{ by 6 result as a compressed Milky-Way result.} \]

1. \((3 \Delta \text{PLUSWAY SUM})[1;] \]
   \[ \text{or} \]
   \[ 3 \quad 3 \quad \text{PLUSRED } 1 \]

2. \((4 \Delta \text{PLUSWAY SUM}) \]
   \[ \text{or} \]
   \[ 50 \quad 4 \quad \text{PLUSRED } 1, [1] \text{SALES, [.5]FIT} \]
3. \( (5 \ 3 \ 2 \ \Delta \text{PLUSWAY SUM})[2 \ 3;\ ;;] \)
   or
   \[6 \ 3 \ 4 \ 5 \ 3 \ 2 \ \text{PLUSRED SALES}, [.5] \text{FIT} \]

4. \( (4 \ 2 \ \Delta \text{PLUSWAY SUM})[1;\ ;] \)
   or
   \[50 \ 4 \ 4 \ 2 \ \text{PLUSRED 1} \]

5. \( (2 \ 3 \ \Delta \text{PLUSWAY SUM})[3;\ ;] \)
   or
   \[4 \ 3 \ 2 \ 3 \ \text{PLUSRED FIT} \]
Chapter 8 Solutions

WRITING USER-FRIENDLY INTERACTIVE FUNCTIONS

1. [WSID: INPUT]

\[
\begin{align*}
\triangleright R & \leftarrow \text{CHOICES \ LPROMPTE \ PROMPT} \\
[1] & \text{Prompts for a single letter. PROMPT is the} \\
[2] & \text{character vector prompt. CHOICES is a character} \\
[3] & \text{vector of allowable single characters to be} \\
[4] & \text{entered. } R \text{ is a one element vector index into} \\
[5] & \text{CHOICES of the character entered or is a numeric} \\
[6] & \text{scalar escape code if an escape word is typed.} \\
[7] & \text{Requires: CPROMPTE.} \\
[8] & \triangleright L1 : R \leftarrow \text{CPROMPTE \ PROMPT} \\
[9] & \text{Exit if scalar escape code:} \\
[10] & \rightarrow (\rho \triangleright R) \downarrow 0 \\
[11] & \rightarrow (1=R) \triangleright R \triangleright L2 \\
[12] & \rightarrow '** \text{ ENTER ONE CHARACTER ONLY'} \\
[13] & \rightarrow L1 \\
[14] & \text{Convert } R \text{ to index:} \\
[15] & \rightarrow L2 : R \leftarrow \text{CHOICES} \triangleright R \\
[16] & \rightarrow (R<\Pi 0 + \rho \text{CHOICES}) \triangleright 0 \\
[17] & \rightarrow '** \text{ INVALID CHOICE. ENTER ONE OF: '} \\
[18] & \rightarrow '1 ,',',[\Pi 0 + 0.5] \text{CHOICES} \\
[19] & \rightarrow L1 \\
\end{align*}
\]

2. Insert the expression

\[
R[(R='-')/\triangleright R]\leftarrow ' ' \\
\]

in the functions NINPUT and NPROMPTE just before using \text{\OVI} on \( R \). Insert the APL2 expression

\[
((R='-')/R)\leftarrow ' ' \\
\]

in the functions NINPUT2 and NPROMPTE2 just before checking \( R \) for numeric characters.
3.

\[ \text{[WSID: INPUT]} \]

\[ \triangledown \text{PROPOSAL; AGES; NAME; NKIDS; R} \]

[1] Illustration of input utility functions.

[2] \text{L1:} \rightarrow 0 \text{ ESCAPE NAME} \leftarrow \text{CPROMPTE 'NAME: '}

[3] \text{L2:} \rightarrow 0 \text{ ESCAPE NKIDS} \rightarrow 1 \text{ NPROMPTE 'NUMBER OF KIDS: '}

[4] \rightarrow \text{L2 IF(\neg NKIDS} \geq 0, \land 20) \text{MESSAGE '** 0 TO 20 KIDS ONLY'}

[5] \rightarrow \text{L4 UNLESS} \times \text{NKIDS}

[6] \text{L3:} \rightarrow 0 \text{ ESCAPE AGES} \leftarrow \text{NKIDS NPROMPTE 'AGES OF KIDS: '}

[7] \rightarrow \text{L3 IF(AGES} \neq \text{AGES) \lor (AGES} < 0 \lor \text{AGES} > 99) \text{MESSAGE '** 0 TO 99 AGES ONLY'}}

[8] Allow alignment of paper:

[9] \text{L4:} \rightarrow 0 \text{ ESCAPE CPROMPTE 'PRESS ENTER WHEN READY...'}

[10] \text{3 blank lines:}

[11] \text{\( \rightarrow 3 \) l \( \rho ' ' \)}

[12] \text{\( \rightarrow ' ' \text{Dear ', NAME, '':} \)}

[13] \text{1 blank line:}

[14] \text{\( \rightarrow ' ' \)}

[15] \text{\( \rightarrow ' ' \text{As a proud parent of ', NKIDS, ' kid'} \)}

[16] \text{\( \rightarrow ' ' \text{(1} \neq \text{NKIDS)'} s', ' \text{whose'} \)}

[17] \text{\( \rightarrow ' ' \text{average age is ', AGES, '} \text{NKIDS} \)}

[18] \text{\( \rightarrow ' ' \text{, you need insurance.'} \)}

[19] \text{\( \rightarrow 3 \) l \( \rho ' ' \)}

[20] Allow alignment of paper:

[21] \text{\( \rightarrow 0 \) ESCAPE CPROMPTE ' '}

[22] \text{\( \rightarrow 0 \) ESCAPE R \( \leftarrow ' ' \text{YN} \) LPROMPTE 'GENERATE ANOTHER PROPOSAL? '}

[23] Do another if response is Y:

[24] \text{\( \rightarrow \text{L1 IF R=1} \)}
1. 1-1I((TODAYS360 MDATES)-TODAYS360 PDATES)÷180  
   or  
   1I(TODAYS360 PDATES)-TODAYS360 MDATES)÷180  
   These two expressions return different results if the purchase  
   date occurs on a coupon date (1 and 0 respectively). The first  
   expression is correct if the buyer receives the coupon and the  
   second is correct if the seller receives the coupon.  

2. (1000X1.001I/TODAYS RDATE,BDATE)-1000  

3.  
   [WSID: DATES]  
   V YYYYDDD+TOYD YYYYMMDD;DD;LEAP;MM;MMDD;YYYY;DIO  
   [1] ^ Converts dates (YYYYMMDD) to Julian dates (YYYYDDD  
   [2] ^ where DDD is number of days since prior Dec. 31).  
   [3] DIO+1  
   [4] DD+100IYYYYMMDD  
   [5] MMDD+10000IYYYYMMDD  
   [6] MM+(MMDD-DD)+100  
   [7] ^ Year and year times 1000 (e.g. 1986000):  
   [8] YYYYX(YYYYMMDD-MMDD)+10000  
   [9] YYYYDDD+1000XYYYY  
   [10] ^ Add in days from start of month, and from start  
   [11] ^ of year to start of month:  
   [12] YYYYDDD+YYYYDDD+DD+(0 31 59 90 120 151 181 212 243 273  
        304 334)[MM]  
   [13] ^ Determine whether a leap year:  
   [14] LEAPX(O=4IYYYY)^X(0#100IYYYY)v0=400IYYYY  
   [15] ^ Add in leap day if month is March or later:  
   [16] YYYYDDD+YYYYDDD+LEAP^MM+3  
       V
Chapter 9 Solutions

MANIPULATING DATES

[WSID: DATES]

V YYYYMMDD+FROMYD YYYYDDD;DD;DDD;FEB29;LEAP;MM;YYYY;DIO

[1] a Convert Julian dates (YYYYDDD where DDD is number of
   [2] a days since prior Dec. 31) to YYYYMMDD dates.
   [3] DIO+1
   [4] DDD+10000*YYYYDDD
   [5] a Year and year times 10000 (e.g. 19860000):
   [6] YYYY=(YYYYDDD-DDD)+1000
   [7] YYYYMMDD+10000*YYYY
   [8] a Determine whether a leap year:
   [9] LEAP=(O=4|YYYY)∧(0≠100|YYYY)∨0=400|YYYY
   [10] a Is day a leap day (i.e. Feb. 29)?
   [11] FEB29+LEAP*DDD=60
   [12] a Subtract leap day if Feb. 29 or later to determine
       month:
   [13] DDD=DDD-LEAP*DDD=60
   [15] a Days since start of month:
   [16] DD/DDD-(0 31 59 90 120 151 181 212 243 273 304 334)(MM
   [17] a Add back one day if Feb. 29:
   [18] DD=DD+FEB29
   [19] YYYYMMDD=YYYYMMDD+DD+100×MM

4. WKDAYS←7 9p'MONDAY TUESDAY WEDNESDAYTHURS...SUNDAY '
   WKDAY=WKDAYS[DIO+711+TODAYS FROMQTS 3pQTS;]
   (WKDAY#""')/WKDAY

-365-
Chapter 10 Solutions

WRITING REPORTS

1. H1←15 15 3 HEADINGS 'LAST YEAR\nTHIS YEAR'
   HDG←'AVG.→SALES\nTOTAL→SALES\nAVG.→SALES\nTOTAL→SALES'
   HDG←HDG,'GROWTH→TOTAL→SALES'
   HDG←6 7 6 7 2 3 2 3 HEADINGS HDG
   HDG[1 2↓↓H1]←H1

2. A Format the date:
   FDATE←,'G<Z9/99/99>' ∙FMT DATE ∙ APL*PLUS or SHARP APL
   FDATE←'55/55/50' ∙FDATE ∙ APL2
   FDATE←1 1 0 1 1 0 1 1\6 0 ∙FDATE ∙ A Other APL systems
   FDATE[3 6]←'/'' ∙ A Other APL systems
   TITLE←'PAGE ',(≔PNO),',\nFINANCIAL SUMMARY\n',FDATE
   TITLE←TITLE,','WESTERN REGION'
   TITLE←65 TITLES TITLE

3. Approach 1 (using the newline character):
   TCNL←UTCNL ∙ APL*PLUS
   TCNL←OAV[156+IIIO] ∙ SHARP APL
   TCNL←UTC[1+IIIO] ∙ APL2
   TCNL←OAV[??] ∙ A Other APL systems

   CMAT←('4(I5,X2),<',TCNL,')',4F7.1') ∙FMT 3 8pNMAT ∙ APL*PLUS,
                                                SHARP APL

   CMAT←((28p'55550 '),TCNL,28p' 5550.0') ∙FNMAT ∙ APL2

   CMAT←((8p7 0),8p7 1) ∙3 8pNMAT ∙ A Other APL systems
   CMAT←(0 2↓↓CMAT),TCNL,3 ∙28↑CMAT ∙ A Other APL systems

   Approach 2 (using dyadic transpose):

   CMAT←(6p2 0)↓6 28p2 1 3↓4 6 7p(12p7 0 7 1)↓3NMAT
1. The following are good candidates for "visual representation manipulation" functions. Their listings are not presented here but are available on disk. See the Postscript at the end of the book.

Syntax: NEWVR~UNLAMP OLDVR

The UNLAMP function removes all comments from the visual representation. Both end-of-line and full-line comments are removed completely, including the comment symbol (\%). The function lines are renumbered as needed to allow for deleted full-line comments. UNLAMP is different from UNCOMMENT in that UNCOMMENT does not delete the comment symbol on full-line comments and so has no line-renumbering.

Syntax: NEWVR~OBFUSCATE OLDVR

The OBFUSCATE function modifies all local identifiers (labels, result variable, argument variables, localized variables) by prefixing their names by the characters 'AA'. In this way, the identifiers of the function are obfuscated so that chances of a name conflict are minimized when the function uses execute (\%) to access variables or functions whose definitions are global to the function being obfuscated. WSDOC is an example of an obfuscated function.

Syntax: NEWVR~UNOBFUSCATE OLDVR

The UNOBFUSCATE function deletes all occurrences of the characters 'AA', thereby undoing the effects of OBFUSCATE. Since UNOBFUSCATE does not limit its search to just local identifiers, it will delete even those occurrences of 'AA' which were not inserted by OBFUSCATE (such as in character constants, in comments or in the middle of identifier names).
Syntax: NEWVR~UNDIAMOND OLDVR

The UNDIAMOND function breaks all multi-statement lines (◊ delimited) into single-statement lines, renumbering lines as needed. For example:

```
 v TEST
 v
```

The UNDIAMOND function may be used when moving a function from an APL environment which supports statement separators (◊) to one which does not; or you may find single-statement lines easier to read than multi-statement lines.

2.

```
[WSID: WSDOC]
 v WSDOC △PAGE;△B;△BOTTOM;△C;△D;△DATA;△DONE;△F;
 △FIRST;△FNS;△FOOT;△HEIGHT;△I;△IND;△LAST;△LEFT;
 △LEN;△LIM;△LINES;△MARGIN;△N;△NAME;△NL;
 △NONDISPLAY;△P;△PNO;△QUOTE;△R;△S;△T;△TCNL;
 △TITLE;△TOP;△TXT;△VARS;△VR;△W;△WIDTH
 [1] a Displays paged WS documentation. All output
 [2] a is via ⊕ so replace all ⊕ by custom fn
 [3] a (e.g. PRINT) to redirect output. PAGE: rows,
 [4] a columns, margins (top, bottom, left, right).
 [8] △LEFT←△PAGE[4+OIO]
 [10] a Construct newline character:
 [12] a TCNL←TC[1+OIO] a APL2
 [14] a Format today's date:
 [15] △D←$OTS[1 2 0 +OIO]
 [16] △D(△D=' ')/'p △D]←'/'
 [17] a Format the time:
 [18] △T←($OTS[3+OIO]),':',~2↑'0',♂OTS[4+OIO]
 [19] a Format the WSID:
 [20] △TITLE←$WSID a APL*PLUS
 [21] a TITLE←2 DWS 1 a SHARP APL
 [22] a S+100 0SVO 'C' a APL2 (TSO)
 [23] a S+0 0 1 1 0SVC 'C' a APL2 (TSO)
 [24] a C←')WSID' a APL2 (TSO)
 [25] a TITLE←C[OIO];~' ' a APL2 (TSO)
```
\[ WSDOC \text{(continued)} \]

[26] \( \text{a S} \text{DSVR 'C' a APL2 (TSO)} \)
[27] \( \text{a Provide WSID as left arg otherwise:} \)
[28] \( \text{a TITLE} \text{WSID a APL2 (CMS)} \)
[29] \( \text{a Delete leading/trailing blanks:} \)
[30] \( \Delta \text{TITLE} \langle (+/\backslash \Delta \text{TITLE}^\prime) \rangle \downarrow \text{TITLE} \)
[31] \( \text{a Format page title:} \)
[32] \( \Delta \text{TITLE} \langle \Delta \text{TOP} \Delta \text{TCNL}, \Delta \text{WIDTH} \uparrow (\Delta \text{LEFT}^\prime), \Delta \text{TITLE}^\prime, *', ' \Delta \text{D}, ', \Delta \text{T} \)
[33] \( \text{a Insert page number:} \)
[34] \( \Delta \text{PNO} \langle \)
[35] \( \Delta \text{T}' \text{PAGE } 1' \)
[36] \( \Delta \text{TITLE} \langle (\rho \Delta \text{T}) \uparrow \rho \Delta \text{TITLE} \rangle \uparrow \Delta \text{T} \)
[37] \( \text{a Build first page:} \)
[38] \( \Delta \text{TXT} \langle \Delta \text{TITLE}, \Delta \text{TCNL} \)
[39] \( \text{a Keep track of lines used so far (below} \)
[40] \( \text{a top margin) in TXT:} \)
[41] \( \Delta \text{LINES} \langle 2 \)
[42] \( \text{a Build nondefault environment:} \)
[43] \( \Delta \text{T} \langle \Delta \text{TCNL}, (\Delta \text{LEFT}^\prime), ' \text{'NONDEFAULT WORKSPACE} \
[44] \( \text{ENVIRONMENT':}, \Delta \text{TCNL} \)
[45] \( \Delta \text{MARGIN} \langle (\Delta \text{LEFT}+3)^\prime \rangle ' \)
[46] \( \Delta \text{QUOTE} \langle ' ' ' \rangle ' \)
[47] \( \text{a Define chars which don't display normally:} \)
[48] \( \Delta \text{NONDISPLAY} \langle \text{OTCNL, UTCLF, UTCBS, UTCBEL, UTCDEL, UTCNUL, UTCESC, UTCFF} a \text{ APL*PLUS} \)
[49] \( \text{a AAV[OIO+0 1 156 158 159]} a \text{ SHARP APL} \)
[50] \( \text{a AAVNONDISPLAY} a \text{ APL2} \)
[51] \( \text{a Format nondefaults:} \)
[52] \( \rightarrow (\rho \Delta \text{DATA}^\prime, \Delta \text{DLX}) \uparrow \Delta \text{L2} \)
[53] \( \text{a Replace nondisplayable chars by } \|
[54] \( \Delta \text{DATA} \langle (\Delta \text{DATA} \text{a AAVNONDISPLAY}) \downarrow \rho \Delta \text{DATA} \rangle ' \|
[55] \( \text{a Double up quote chars:} \)
[56] \( \Delta \text{DATA} \langle \Delta \text{QUOTE}^\prime (1+\Delta \text{DATA} = \Delta \text{QUOTE}) / \Delta \text{DATA} \rangle, \Delta \text{QUOTE} \)
[57] \( \text{a Width available (after } ' \text{OIO}' \rangle: \)
[58] \( \Delta \text{W} \langle \Delta \text{WIDTH} - \Delta \text{LEFT}+7 \)
[59] \( \text{a Branch if data will fit on a single line:} \)
[60] \( \rightarrow (\Delta \text{W} \geq \rho \Delta \text{DATA}) / \Delta \text{L1} \)
[61] \( \text{a Else truncate and show '...':} \)
[62] \( \Delta \text{DATA} \langle (\Delta \text{W}+3)^\prime \rangle, '...' \)
[63] \( \Delta \text{L1} \langle \Delta \text{T}+\Delta \text{T}, \Delta \text{TCNL}, \Delta \text{MARGIN}, ' \text{OIO}' \rangle, \Delta \text{DATA} \)
[64] \( \Delta \text{L2} \langle \Delta \text{T}+\Delta \text{T}, (1 \text{ OIO}) / \Delta \text{TCNL}, \Delta \text{MARGIN}, ' \text{OIO}' \rangle, \|
[65] \( \Delta \text{T} \langle \Delta \text{T}, (10 \text{ OPP}) / \Delta \text{TCNL}, \Delta \text{MARGIN}, ' \text{OPP}' \rangle, \|
[66] \( \Delta \text{T} \langle \Delta \text{T}, (16807 \text{ ORL}) / \Delta \text{TCNL}, \Delta \text{MARGIN}, ' \text{ORL}' \rangle, \|
[67] \( \text{a Perform precise comparison for OCT:} \)
[68] \( \Delta \text{C} \langle \text{OCT} \)
[69] \( \text{OCT}^\prime \)
[70] \( \Delta \text{T} \langle (\Delta \text{C} \text{A} 2 \text{~46}) / \Delta \text{TCNL}, \Delta \text{MARGIN}, ' \text{C}' \rangle, \|
[71] \( \text{a Use 1E'13 instead of 2~46 for APL2} \)
[72] \( \text{OCT} \langle \Delta \text{C} \)
[73] \( \text{a Other nondefault workspace environment} \)
[74] \( \text{a parameters which may be included are: state} \)
[75] \( \text{a indicator, workspace size, workspace available,} \)

---

\[ -369 - \]
\[ \text{WSDOC (continued)} \]

[76] A symbols reserved and used, error latent
[77] A expression (trap definition), attention latent
[78] A expression, etc.
[79] A

[80] A Include blank line after any nondefaults:
[81] \( \Delta T \leftarrow \Delta T, \Delta TCNL \)
[82] A Attach nondefaults, if any, to page:
[83] \( \Delta N \leftarrow / \Delta T = \Delta TCNL \)
[84] \( \Delta N \leftarrow \Delta N \times \Delta N > 3 \)
[85] \( \Delta TXT \leftarrow / \Delta TXT, (\times \Delta N) / \Delta T \)
[86] \( \Delta LINES \leftarrow \Delta LINES + \Delta N \)

[87] A

[88] A Variables:
[89] \( \Delta VARS \leftarrow \text{ONL} 2 \)
[90] A Squeeze out local (WSDOC) variables:
[91] \( \Delta VARS \leftarrow (\Delta VARS; 0 1 + \text{III} \text{v}.'A') / \Delta VARS \)
[92] \( \Delta MARGIN \leftarrow \Delta LEFT \rho ' \)
[93] A Branch if no variables:
[94] \( \rightarrow (1 \rho \Delta VARS) \uparrow \Delta L7 \)
[95] \( \Delta TXT \leftarrow / \Delta TXT, \Delta TCNL, \Delta MARGIN, 'GLOBAL WORKSPACE VARIABLES: ', \Delta TCNL \)
[96] \( \Delta LINES \leftarrow \Delta LINES + 2 \)

[97] A Sort variable names if not already:
[98] A VARS \leftarrow VARS[OAV, VARS;]
[99] A Right justify variable names:
[100] \( \Delta T \leftarrow \Delta VARS+.' \)
[101] \( \Delta VARS \leftarrow (-\Delta T) \Phi \Delta VARS \)
[102] A Drop leading all blank columns (possible by
[103] A deleting vars):
[104] \( \Delta VARS \leftarrow (0, l / \Delta T) \downarrow \Delta VARS \)
[105] A Indent variable names for left margin plus 3:
[106] \( \Delta VARS \leftarrow (((1 \rho \Delta VARS), \Delta LEFT + 3) \rho ') / \Delta VARS \)
[107] A Include heading:
[108] \( \Delta T \leftarrow \Delta TCNL, ((- \rho \Delta VARS) \uparrow 'NAME'), ' \leftarrow \text{SHAPE } \rho \text{ VALUE'}, \Delta TCNL \)
[109] \( \Delta TXT \leftarrow / \Delta TXT, ((- \rho \Delta VARS) \uparrow '----'), ' \leftarrow ---- \)

[110] \( \Delta LINES \leftarrow \Delta LINES + 2 \)
[111] A Loop by variable:
[112] \( \Delta I \leftarrow \text{III} + 1 \)
[113] \( \Delta LIM \leftarrow (1 \rho \Delta VARS) \downarrow \text{III} \)
[114] \( \Delta L3 : \leftarrow (\Delta LIM \times \Delta I \leftarrow \Delta I + 1) / \Delta L6 \)
[115] A Format shape:
[116] \( \Delta S \leftarrow (\rho \Delta DATA \times \Delta NAME \times \Delta VARS[\Delta I;]), ' \leftarrow \rho ' \)
[117] A Omit shape and \( \rho \) if a scalar; pad to line up
[118] A with \( \rho \) 's:
[119] \( \Delta S \leftarrow (1 - \rho \Delta S) \uparrow (3 \rho \Delta S) / \Delta S \)
[120] A Combine name and shape; compute remaining width:
[121] \( \Delta S \leftarrow \Delta NAME, ' \leftarrow ', \Delta S \)
[122] \( \Delta W \leftarrow \Delta WIDTH - \rho \Delta S \)
[123] A Branch if a numeric variable:
[124] \( \rightarrow (0 = 1 \rho \Delta DATA) \rho \Delta L4 \)

-370-
WSDOC (continued) [125] Else data is character; consider only up to [126] W chars:
[127] \[
\Delta\Delta\text{DATA}\leftarrow(\Delta\Delta W \times / \rho \Delta\Delta\text{DATA})\rho\Delta\Delta\text{DATA}
\]
[128] Else replace nondisplayable chars by \[\wedge]\[\wedge]
[129] \[
\Delta\Delta\text{DATA}\leftarrow(\Delta\Delta\text{DATA}\downarrow\Delta\Delta\text{NONDISPL})/\rho\downarrow\Delta\Delta\text{DATA}\leftarrow\wedge
\]
[130] Else double up quote chars:
[131] \[
\Delta\Delta\text{DATA}\leftarrow\Delta\Delta\text{QUOTE},((1+\Delta\Delta\text{DATA})/\Delta\Delta\text{DATA})\Delta\Delta\text{QUOTE}
\]
[132] Branch if data will fit on a single line:
[133] \[
\rightarrow(\Delta\Delta W \geq \rho \Delta\Delta\text{DATA})/\Delta\Delta L\]
[134] Else truncate and show '...':
[135] \[
\Delta\Delta\text{DATA}\leftarrow((\Delta\Delta W+3)\rho\Delta\Delta\text{DATA}),'
\]
[136] \[
\rightarrow\Delta\Delta L\]
[137] If data is numeric, consider only up to [138] 1+W+2 elements:
[139] \[
\Delta\Delta L\leftarrow\Delta\Delta\text{DATA}\leftarrow((1+1+\Delta\Delta W+2)\downarrow / \rho \Delta\Delta\text{DATA})\rho\Delta\Delta\text{DATA}
\]
[140] Format numbers to 10 digits (CLEAR WS default):
[141] \[
\Delta\Delta P\leftarrow\Delta\Delta P
\]
[142] \[
\Delta P\leftarrow10
\]
[143] \[
\Delta\Delta\text{DATA}\leftarrow\Delta\Delta\text{DATA}
\]
[144] \[
\Delta P\leftarrow\Delta\Delta P
\]
[145] Format value from empty array as 10:
[146] \[
\Delta\Delta\text{DATA}\leftarrow\Delta\Delta\text{DATA},(0=\rho\Delta\Delta\text{DATA})/'0'
\]
[147] Branch if data will fit on a single line:
[148] \[
\rightarrow(\Delta\Delta W \geq \rho \Delta\Delta\text{DATA})/\Delta\Delta L\]
[149] Else truncate at last space within width and [150] show '...':
[151] \[
\Delta\Delta\text{DATA}\leftarrow((/+\vee '-' =\wedge(\Delta\Delta W+3)\rho\Delta\Delta\text{DATA})\rho\Delta\Delta\text{DATA}),'
\]
[152] \[
\rightarrow\Delta\Delta L\]
[153] Append variable definition to page:
[154] \[
\Delta\Delta L\leftarrow\Delta\Delta L\leftarrow(\Delta\Delta\text{BOTTOM}+\Delta\Delta\text{HEIGHT} - \Delta\Delta\text{LINES})\rho\Delta\Delta\text{TCNL}
\]
[155] \[
\Delta\Delta\text{LINES}\leftarrow\Delta\Delta\text{LINES}+1
\]
[156] Branch for more vars unless bottom of page:
[157] \[
\rightarrow(\Delta\Delta\text{LINES} < \Delta\Delta\text{HEIGHT})/\Delta\Delta L\]
[158] Display page:
[159] \[
\rightarrow\Delta\Delta\text{TXT},(\Delta\Delta\text{BOTTOM}+\Delta\Delta\text{HEIGHT} - \Delta\Delta\text{LINES})\rho\Delta\Delta\text{TCNL}
\]
[160] Format new page:
[161] \[
\Delta\Delta\text{TXT}\leftarrow\Delta\Delta\text{TXT},\Delta\Delta\text{TCNL}
\]
[162] \[
\Delta\Delta\text{PNO}\leftarrow\Delta\Delta\text{PNO}+1
\]
[163] \[
\Delta T\leftarrow'\text{PAGE } ',\wedge\Delta\Delta\text{PNO}
\]
[164] \[
\Delta\Delta\text{TXT}((-\rho\Delta\Delta T)\uparrow \rho\Delta\Delta\text{TITLE})\leftarrow\Delta\Delta T
\]
[165] \[
\Delta\Delta\text{LINES}\leftarrow2
\]
[166] Branch if no more vars:
[167] \[
\rightarrow(\Delta\Delta I=\Delta\Delta LIM)/\Delta\Delta L\]
[168] Else insert new heading:
[169] \[
\Delta\Delta\text{TXT}\leftarrow\Delta\Delta\text{TXT},\Delta\Delta\text{TCNL},\Delta\Delta\text{MARGIN},'\text{GLOBAL WORKSPACE VARIABLES (CONT.):}',\Delta\Delta\text{TCNL}
\]
[170] \[
\Delta\Delta\text{LINES}\leftarrow\Delta\Delta\text{LINES}+2
\]
[171] \[
\Delta\Delta T\leftarrow\Delta\Delta\text{TCNL},((-1+\rho\Delta\Delta\text{VARS})\uparrow '\text{NAME}')', \leftarrow \text{SHAPE } \rho\text{ VALUE'}
\]
[172] \[
\Delta\Delta\text{TXT}\leftarrow\Delta\Delta\text{TXT},\Delta\Delta T,((-1+\rho\Delta\Delta\text{VARS})\uparrow '-----'),'
\]
[173] \[
\Delta\Delta\text{LINES}\leftarrow\Delta\Delta\text{LINES}+2
\]
\[ \text{WSDOC (continued)} \]

[174] \( \triangle \) Branch for more vars:

[175] \( \rightarrow \triangle \) L3

[176] \( \triangle \) No more vars (include blank line):

[177] \( \triangle \triangle L6: \triangle \) TXT \( \rightarrow \triangle \) TXT, \( \triangle \) TCNL

[178] \( \triangle \triangle \) LINES \( \rightarrow \triangle \) LINES + 1

[179] \( \triangle \) No more vars:

[180] \( \triangle \) Functions:

[181] \( \triangle \triangle \) L7: \( \triangle \) FNS \( \rightarrow \) DN L 3

[182] \( \triangle \) Squeeze out this (WSDOC) function:

[183] \( \triangle \triangle \) FNS \( \rightarrow \triangle \) FNS if \( (1 \lor \triangle \) FNS) \( \rightarrow \) 'WSDOC' \( \rightarrow \) FNS

[184] \( \triangle \) Use next, not prior, line if CRAVR is used below:

[185] \( \triangle \) FNS if \( (\triangle / \) FNS) \( \rightarrow \) (2,1 \lor \) FNS) \( \rightarrow \) 5 \( \lor \) 'WSDOCCRAVR' \( \rightarrow \) FNS

[186] \( \triangle \) Branch if some fns:

[187] \( \rightarrow (\times \triangle \) LIM \( \rightarrow \) 1 \lor \) FNS) \( \rightarrow \) L8

[188] \( \triangle \) Exit if nothing on page:

[189] \( \rightarrow (\triangle \) LINES \( \rightarrow \) 2) / 0

[190] \( \triangle \) Else display page and exit:

[191] \( \rightarrow (\triangle \) TXT, (\triangle \) BOTTOM + \( \triangle \) HEIGHT - \( \triangle \) LINES) \( \rightarrow \) \( \triangle \) TCNL

[192] \( \rightarrow 0 \)

[193] \( \triangle \) Branch if at least 4 lines left on page:

[194] \( \rightarrow (\triangle \) LINES \( \rightarrow \) \( \triangle \) HEIGHT + 4) / \( \triangle \) L9

[195] \( \triangle \) Else display page:

[196] \( \rightarrow (\triangle \) TXT, (\triangle \) BOTTOM + \( \triangle \) HEIGHT - \( \triangle \) LINES) \( \rightarrow \) \( \triangle \) TCNL

[197] \( \rightarrow 0 \)

[198] \( \rightarrow (\triangle \) TXT, (\triangle \) BOTTOM + \( \triangle \) HEIGHT - \( \triangle \) LINES) \( \rightarrow \) \( \triangle \) TCNL

[199] \( \rightarrow \) Format new page:

[200] \( \rightarrow \) TXT \( \rightarrow \) \( \triangle \) TITLE, \( \triangle \) TCNL

[201] \( \rightarrow \) FNO \( \rightarrow \) FNO + 1

[202] \( \rightarrow \) T 'PAGE ', \( \nabla \) \( \triangle \) FNO

[203] \( \rightarrow \) TXT \( \rightarrow \) (\( \rightarrow \triangle \) TITLE \( \rightarrow \) \( \triangle \) T

[204] \( \rightarrow \) LINES \( \rightarrow \) 2

[205] \( \rightarrow \)

[206] \( \rightarrow \) L9: \( \rightarrow \) TXT \( \rightarrow \) TXT, \( \rightarrow \) TCNL, \( \rightarrow \) MARGIN, 'FUNCTIONS:', \( \rightarrow \) TCNL

[207] \( \rightarrow \) LINES \( \rightarrow \) LINES + 2

[208] \( \rightarrow \) Sort fn names if not already:

[209] \( \rightarrow \) FNS \( \rightarrow \) FNS if \( \nabla \) (AV \( \rightarrow \) FNS);

[210] \( \rightarrow \)

[211] \( \rightarrow \) Pad fn names with 3 leading blank columns:

[212] \( \rightarrow \) FNS if \( \nabla \) 0 \( \rightarrow \) 3 \( \rightarrow \) FNS

[213] \( \rightarrow \) How many "columns" of fn names will fit across pg?

[214] \( \rightarrow \) W \( \rightarrow \) 1 \lor \) FNS

[215] \( \rightarrow \) N \( \rightarrow \) WIDTH \( \rightarrow \) W

[216] \( \rightarrow \) How many rows will this take?

[217] \( \rightarrow \) R \( \rightarrow \) LIM \( \rightarrow \) N

[218] \( \rightarrow \) Pad bottom of fn list for subsequent reshape:

[219] \( \rightarrow \) FNS if \( \nabla \) (R \( \times \) N), \( \rightarrow \) W \( \rightarrow \) FNS

[220] \( \rightarrow \) Shuffle fn list to get names in desired order:

[221] \( \rightarrow \) F \( \rightarrow \) (R \( \rightarrow \) LEFT \( \rightarrow \) 1) \( \rightarrow \) \( \rightarrow \) W \( \rightarrow \) FNS

[222] \( \rightarrow \) How will this fit on this page?

[223] \( \rightarrow \) L0: \( \rightarrow \) N \( \rightarrow \) (1 \lor \) F \( \rightarrow \) HEIGHT \( \rightarrow \) LINES

[224] \( \rightarrow \) Stick them on:

[225] \( \rightarrow \) TXT \( \rightarrow \) TXT, \( \rightarrow \) TCNL, \( \rightarrow \) LINES, \( \rightarrow \) N \( \rightarrow \) F \( \rightarrow \) F

\( \rightarrow \)
WSDOC (continued)

\[ \Delta \Delta \text{LINES} \leftarrow \Delta \Delta \text{LINES} + \Delta \Delta \text{N} \]

\[ \text{a} \] Drop them off:

\[ \Delta \Delta \text{F} \leftarrow (\Delta \Delta \text{N}, 0) \downarrow \Delta \Delta \text{F} \]

\[ \text{a} \] Branch if none left:

\[ \rightarrow (1 \uparrow \Delta \Delta \text{F}) \downarrow \Delta \Delta \text{L11} \]

\[ \text{a} \] Display page:

\[ \Downarrow \Delta \Delta \text{TXT}, \Delta \Delta \text{BOTTOM}, \Delta \Delta \text{TCNL} \]

\[ \text{a} \] Format new page:

\[ \Delta \Delta \text{TXT} \leftarrow \Delta \Delta \text{TITLE}, \Delta \Delta \text{TCNL} \]

\[ \Delta \Delta \text{PNO} \leftarrow \Delta \Delta \text{PNO} + 1 \]

\[ \Delta \Delta \text{T} \leftarrow \text{PAGE}, \% \Delta \Delta \text{PNO} \]

\[ \Delta \Delta \text{TXT} \leftarrow \left( \neg \Delta \Delta \text{T} \uparrow \neg \Delta \Delta \text{TITLE} \right) \Delta \Delta \text{T} \]

\[ \Delta \Delta \text{LINES} \leftarrow 2 \]

\[ \Delta \Delta \text{TXT} \leftarrow \Delta \Delta \text{TXT}, \Delta \Delta \text{TCNL}, \Delta \Delta \text{MARGIN}, \text{FUNCTIONS (CONT.)} : ' ', \Delta \Delta \text{TCNL} \]

\[ \Delta \Delta \text{LINES} \leftarrow \Delta \Delta \text{LINES} + 2 \]

\[ \text{a} \] Put remaining fns on this page:

\[ \rightarrow \Delta \Delta \text{L10} \]

\[ \text{a} \]

\[ \text{a} \] Include 5 blank lines after fn list:

\[ \Delta \Delta \text{L11} \leftarrow \Delta \Delta \text{N} \leftarrow 5 \uparrow \Delta \Delta \text{HEIGHT} - \Delta \Delta \text{LINES} \]

\[ \Delta \Delta \text{TXT} \leftarrow \Delta \Delta \text{TXT}, \Delta \Delta \text{PNO} \Delta \Delta \text{TCNL} \]

\[ \Delta \Delta \text{LINES} \leftarrow \Delta \Delta \text{LINES} + \Delta \Delta \text{N} \]

\[ \text{a} \]

\[ \text{a} \] Loop by function (keep track of 1st one for footnote):

\[ \Delta \Delta \text{FIRST} \leftarrow \Delta \Delta \text{FNS}[\neg \text{IO};] \]

\[ \Delta \Delta \text{FIRST} \leftarrow (\Delta \Delta \text{FIRST} \neq \neg \Delta \Delta \text{FIRST}) / \Delta \Delta \text{FIRST} \]

\[ \Delta \Delta \text{LAST} \leftarrow \neg \]

\[ \Delta \Delta \text{I} \leftarrow \text{IO} \leftarrow 1 \]

\[ \Delta \Delta \text{LIM} \leftarrow \Delta \Delta \text{LIM} \leftarrow \neg \text{IO} \]

\[ \Delta \Delta \text{L12} \leftarrow \left( \neg \Delta \Delta \text{DONE} \leftarrow \Delta \Delta \text{LIM} \leftarrow \Delta \Delta \text{I} \leftarrow \Delta \Delta \text{I} + 1 \right) / \Delta \Delta \text{L18} \]

\[ \Delta \Delta \text{NAME} \leftarrow \Delta \Delta \text{FNS}[\neg \Delta \Delta \text{I};] \]

\[ \Delta \Delta \text{NAME} \leftarrow (\Delta \Delta \text{NAME} \neq \neg \Delta \Delta \text{NAME}) / \Delta \Delta \text{NAME} \]

\[ \Delta \Delta \text{VR} \leftarrow \text{UVR} \ \Delta \Delta \text{NAME} \ \text{a APL}\star\text{PLUS} \]

\[ \text{a VR} \leftarrow \text{1} \ \text{UDF NAME} \ \text{a SHARP APL} \]

\[ \text{a VR} \leftarrow \text{CRAVR} \ \text{OCR NAME} \ \text{a APL2} \]

\[ \text{a} \] Get next fn if this one locked:

\[ \rightarrow (\neg \Delta \Delta \text{VR}) \downarrow \Delta \Delta \text{L12} \]

\[ \text{a} \] Find newline characters:

\[ \Delta \Delta \text{NL} \leftarrow (\Delta \Delta \text{VR} = \Delta \Delta \text{TCNL}) / \neg \Delta \Delta \text{VR} \]

\[ \text{a} \] Lengths of lines:

\[ \Delta \Delta \text{LEN} \leftarrow \neg 1 \uparrow \Delta \Delta \text{NL} \leftarrow \neg 1 \uparrow (\neg \text{IO} \leftarrow 1), \Delta \Delta \text{NL} \]

\[ \text{a} \] Branch if no lines too long:

\[ \rightarrow (\neg \Delta \Delta \text{T} \leftarrow \Delta \Delta \text{LEN} \leftarrow \neg \Delta \Delta \text{WIDTH} - \Delta \Delta \text{LEFT}) \downarrow \Delta \Delta \text{L15} \]

\[ \text{a} \] Retain starting indices and lengths of too long lines:

\[ \Delta \Delta \text{NL} \leftarrow \neg \Delta \Delta \text{T} / \Delta \Delta \text{NL} \]

\[ \Delta \Delta \text{LEN} \leftarrow \neg \Delta \Delta \text{T} / \Delta \Delta \text{LEN} \]

\[ \text{a} \] Flag chars which shouldn't be broken when contiguous:

\[ \Delta \Delta \text{B} \leftarrow \Delta \Delta \text{VR} \leftarrow \text{ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789.}_0^' \]
WSDOC (continued)

[277] Break points exist everywhere but where these chars occur and are followed by another one of these chars:

[280] \[ \Delta A B \leftarrow \Delta A B \Phi \Delta A B \]

[281] Indices of break chars found so far:

[282] \[ \Delta A I N D \leftarrow 0 \]

[283] Relative indices of first WIDTH chars following known breaks:

[285] \[ \Delta A R \leftarrow (\sim 0 0) + \Phi \sim \Delta A D - \Delta A L E F T \]

[286] Displacement to next break points:

[287] \[ \Delta A L 0 3 : \Delta A D \leftarrow / V \Delta A B [ \Delta A N L \sim 0 \sim + \Delta A R ] \]

[288] Use full length if break point under 1/2 line:

[289] \[ \Delta A D [(\Delta A D < (\rho \Delta A R + 2) / \sim (\rho \Delta A D) \sim + \rho \Delta A R ] \]

[290] Update vars for new break points:

[291] \[ \Delta A N L \leftarrow \Delta A N L + \Delta A D \]

[292] \[ \Delta A I N D \leftarrow \Delta A I N D , \Delta A N L \]

[293] \[ \Delta A L E N \leftarrow \Delta A L E N - \Delta A D \]

[294] Any lines still too long (add 6 space indent)?

[295] Branch if not:

[296] \[ \Delta A R \leftarrow \Delta A W I D T H - \Delta A L E F T + 6 \]

[297] \[ \Delta A T \leftarrow \Delta A L E N \sim + \Delta A R \]

[298] \[ \Delta A L E N \leftarrow \Delta A T / \Delta A L E N \]

[299] \[ (\rho \Delta A L E N) \downarrow \Delta A L 1 4 \]

[300] \[ \Delta A N L \leftarrow \Delta A T / \Delta A N L \]

[301] \[ \Delta A R \leftarrow (\sim 0 0) + \Phi \sim \Delta A R \]

[302] Repeat:

[303] \[ \rightarrow \Delta A L 1 3 \]

[304] Build replication vector to insert line breaks:

[305] \[ \Delta A L 1 4 : \Delta A R \leftarrow (\rho \Delta A V R) \rho 1 \]

[306] Allow 8 positions for char, newline, 6 spaces:

[307] \[ \Delta A R [ \Delta A I N D ] \leftarrow 8 \]

[308] Expand visual representation:

[309] \[ \Delta A V R \leftarrow \Delta A R / \Delta A V R \]

[310] Adjust indices for new VR:

[311] \[ \Delta A I N D \leftarrow \Delta A I N D [ \Delta A I N D ] + 7 \times (\sim 0 0) - 0 0 \]

[312] Insert break characters:

[313] \[ \Delta A V R [ \Delta A I N D \leftarrow (\sim 0 0) + 1 7 ] \leftarrow (\rho \Delta A I N D , 7) \rho \Delta A T C N L , 6 \rho \]

[314] Redefine NL:

[315] \[ \Delta A N L \leftarrow (\Delta A V R = \Delta A T C N L ) / \sim (\rho \Delta A V R \]

[316] Branch if no left margin:

[317] \[ \Delta A L 1 5 : \leftarrow \leftarrow \Delta A L E F T \downarrow \Delta A L 1 6 \]

[318] Build replication vector to insert left margin:

[319] \[ \Delta A R \leftarrow (\rho \Delta A V R) \rho 1 \]

[320] Allow positions for margin:

[321] \[ \Delta A R [ \sim 0 0 \Delta A N L ] \leftarrow \Delta A L E F T + 1 \]

[322] Expand visual representation:

[323] \[ \Delta A V R \leftarrow \Delta A R / \Delta A M A R G I N , \Delta A R / \Delta A V R \]

[324] Adjust newline indices for new VR:

[325] \[ \Delta A N L \leftarrow \Delta A N L + \Delta A L E F T \times (\sim 0 0 \Delta A N L ) \leftarrow \sim 0 0 \]

[326] Insert left margin:

[327] \[ \Delta A V R [ \sim 0 0 \Delta A N L \leftarrow (\sim 0 0) + 1 \sim \Delta A L E F T ] \leftarrow \]'

[328] Branch if fn won't fit on current page:

[329] \[ \Delta A L 1 6 : \leftarrow (\Delta A H E I G H T < 1 + \Delta A L I N E S + \rho \Delta A N L ) / \Delta A L 1 8 \]
3. [WSID: USEDBY]

\[ \text{USEDBY AAFNS;} \text{OIO;} \text{AAPL;} \text{AAPAN;} \text{AAPARGS;} \text{AAPB;} \text{AAPBL;} \text{AAPC;} \text{AAPCOLS;} \text{AAPGDS;} \text{AAPGLOBAL;} \text{AAPNUM;} \text{AAPHDR;} \text{AAPI;} \text{AAPIDLEN;} \text{AAPIDS;} \text{AAPIDSTART;} \text{AAPIDTYPES;} \text{AAPIND;} \text{AAPINDEX;} \text{AAPJ;} \text{AAPLAB;} \text{AAPLCGLOBAL;} \text{AAPLCPARSED;} \text{AAPLDSP;} \text{AAPLEN;} \text{AAPLIM;} \text{AAPNUM;} \text{AAPLOC;} \text{AAPLOCAL;} \text{AAPALTYPE;} \text{AAPNAME;} \text{AAPNAMES;} \text{AAPNC; \text{AAPNCCON;} \text{AAPNCMT; \text{AAPNID;} \text{AAPNL;} \text{AAPNQ;} \text{AAPNUM;} \text{AAPPAN;} \text{AAPPARSE;} \text{AAPPARSED;} \text{AAPPBL;} \text{AAPPID;} \text{AAPPLAB;} \text{AAPNUM;} \text{AAPPSID;} \text{AAPR;} \text{AAPRESULT;} \text{AAPRVAR;} \text{AAPS; \text{AAPSTART;} \text{AAPT;} \text{AAPTCNL;} \text{AAPVR} \]\n
[1] a Displays chart of fns and vars called by the fns
[2] a specified in the character matrix or vector
[3] a (delimited by spaces) argument. Requires subfns:
[4] a CMIOTA; (and CRAVR if not APL*PLUS or SHARP APL).
[5] a Use origin 1:
[6] \( \text{OIO+1} \)
[7] a Indent per level:
[8] \( \text{AAPINDENT+3} \)
[9] a Branch unless fns a matrix:
[10] \( \rightarrow (2 \neq \rho \text{AAPFNS}) \rho \text{AAPL} \)
[11] a Left justify char mat:
[12] \( \text{AAPFNS}^{-=}+/\text{AAPFNS}' \)\( \rho \text{AAPFNS} \)
[13] \( \rightarrow \text{AAPL2} \)
[14] a Flag blank before and last char of each name:
[15] \( \text{AAPL1}; \text{AAPFNS}' \)', \( \text{AAPFNS} \)
[16] \( \text{AAPB}^{-=}\text{AAPFNS}' \', \text{AAPFNS} \)
[17] \( \text{AAPT}^{-=}((\text{AAPB}^{-1}\rho \text{AAPB})/\rho \rho \text{AAPB}) \)
[18] \( \text{AAPT}^{-=}(0((\text{AAPT}^{-1})/2, 2, 2)^{-1}\rho \rho \text{AAPT}) \)
[19] a Lengths and starts of each name:
[20] \( \text{AAPLEN}^{-=}/\text{AAPT} \)
[21] \( \text{AAPSTART}^{-=}\text{AAPT}[;1] \)
[22] a Number of rows and cols in desired matrix:
[23] \( \text{AAPR}^{-=}\rho \text{AAPLEN} \)
[24] \( \text{AAPC}^{-=}\text{AAPLEN}/\text{AAPLEN} \)
[25] a Blank, raveled array:
[26] \( \text{AAPT}^{-=}(\text{AAPR}^{-1}\text{AAPC})^{-1}\rho' \)
[27] a Compute indices: (\( \text{AAPLEN}[1])\), (\( \text{AAPLEN}[2])\), ...
[28] \( \text{AAPI}^{-=}\text{AAPLEN}^{-1}\downarrow 0, +/\text{AAPLEN} \)
[29] \( \text{AAPI}^{-=}\text{AAPI}+1\rho \text{AAPI} \)
[30] a Insert fn names into raveled matrix:
[31] \( \text{AAPT[\text{AAPI}+\text{AAPLEN}/\text{AAPC}^{-1}+1\rho \text{AAPLEN}]^{-=}} \text{AAPFNS[\text{AAPI}+\text{AAPLEN}/\text{AAPSTART}] \)
[32] a Reshape to matrix:
[33] \( \text{AAPFNS}^{-=}(\text{AAPR}, \text{AAPC})^{-1}\rho \text{AAPT} \)
[34] a Exit if no fns specified:
[35] \( \text{AAPL2}^{-=}\text{AAPR}^{-1}\rho \text{AAPFNS} \)
[36] \( \rightarrow \text{AAPR}^{-1} \)
[37] a Construct newline character:
[38] \( \text{AAPTCNL}^{-=}\text{AAPTCNL A APL*PLUS} \)
[39] a \( \text{AAPTCNL}^{-=}\text{AAPTCNL[1+\text{OIO}] A APL2} \)
[40] a \( \text{AAPTCNL}^{-=}\text{AAPTCNL[156+\text{OIO}] A SHARP APL} \)
[41] a Initialize tracking variables...
[42] a Character matrix of all distinct identifiers
[43] a found so far:
[44] \( \text{AAPNAMES}^{-=}(\text{AAPFNS CMIOTA} \text{AAPFNS}=\downarrow \text{AAPR})^{-1}\text{AAPFNS} \)
\`\`\` USEDBY (continued)
[45] \` Index (into NAMES) vector of fns analyzed
[46] \` so far (\`1=initial call):
[47] \`\`\` PARSED< \`1
[48] \` Index (into NAMES) vector of global names for
[49] \` fn in PARSED:
[50] \`\`\` GLOBAL<\`\`\` NAMES CMIOTA \`\`\` FNS
[51] \` No. of globals in GLOBAL for fn in PARSED:
[52] \`\`\` NUM<\`\`\` R
[53] \` No. of globals in GLOBAL before those for fn
[54] \` in PARSED:
[55] \`\`\` DSP+<\`\`\` 0
[56] \` Index (into NAMES) vector of local names for
[57] \` fn in PARSED:
[58] \`\`\` LOCAL<\`\`\` 0
[59] \` No. of locals in LOCAL for fn in PARSED:
[60] \`\`\` LNUM<\`\`\` 0
[61] \` No. of locals in LOCAL before those for fn
[62] \` in PARSED:
[63] \`\`\` DSP+<\`\`\` 0
[64] \` Integer vector of local var types for each elt
[65] \` of LOCAL (1:label; 2:result; 3:argument; 4:local):
[66] \`\`\` TYPE<\`\`\` 0
[67] \` Index into PARSED of object whose globals are
[68] \` currently being evaluated:
[69] \`\`\` LC<\`\`\` 1
[70] \` Index into partition of GLOBAL (for the object
[71] \` whose globals are currently being evaluated) of
[72] \` the object being evaluated:
[73] \`\`\` CGLOBAL<\`\`\` 1
[74] \` \`
[75] \` Determine index into NAMES of object being evaluated:
[76] \`\`\` LOOP:\`\`\` INDEX\`\`\` GLOBAL DGP[LCPARSE[1]]\`\`\` CGLOBAL[1]
[77] \` What's its name?
[78] \`\`\` NAME\`\`\` NAMES[INDEX;]
[79] \`\`\` NAME\`\`\` NAME<\`\`\` '/\`\`\` NAME
[80] \` Loop on elts of LCPARSE from local to global
[81] \` (I=2,3,...) (to check whether object is locally
[82] \` defined at higher level):
[83] \`\`\` I+<\`\`\` 1
[84] \`\`\` LIM<\`\`\` 1+\`\`\` LCPARS
[85] \`\`\` LP=\`\`\` LIM<\`\`\` I+\`\`\` I+<\`\`\` 1\`\`\` L3
[86] \`\`\` IND<\`\`\` LCPARSE[I]
[87] \` List of local objects at this level:
[88] \`\`\` T<\`\`\` LOCAL DLD[LIND]+DNUM[DIND]]
[89] \` Search for this object:
[90] \`\`\` J<\`\`\` T[I\`\`\` INDEX
[91] \` Move up a level if not found:
[92] \`\`\` J=\`\`\` T<\`\`\` LP
[93] \`\`\` T<\`\`\` TYPE DLD[T+DIND]+DJ]
[94] \` Display object name, fn where local, type:
[95] \`\`\` R<\`\`\` NAMES[DPARSE[DIND;]

-377-
v USED BY (continued)
[97] ΔΔR←(ΔΔR≠' ') / ΔΔR
[98] ΔΔT←(4 8 ρ' label result argument local ') [ΔΔT;]
[99] ΔΔT←(ΔΔT≠' ') / ΔΔT
[100] Δ+(ΔΔINDENT×1+ρΔΔLCPARSED)ρ', ΔΔNAME,' (', ΔΔR, ' -
' , ΔΔT, ' )'
[101] →ΔΔL10
[102] a
[103] a Branch if object is a function:
[104] ΔΔL3:→(3= DNC ΔΔNAME) ρ ΔΔL4
[105] a Display object name and global indicator:
[106] Δ+(ΔΔINDENT×1+ρΔΔLCPARSED)ρ', ΔΔNAME,' (global)
[107] →ΔΔL10
[108] a
[109] a Display object name:
[110] ΔΔL4:Δ+(ΔΔINDENT×1+ρΔΔLCPARSED)ρ', ΔΔNAME
[111] a Branch if fn has already been parsed:
[112] →((ρΔΔPARSED)≥ΔΔJ+ΔΔPARSED×ΔINDEX) ρ ΔΔL9
[113] ΔΔPARSED←ΔΔPARSED, ΔΔINDEX
[114] a
[115] a Analyze fn for global, local identifiers...
[116] a Construct visual representation:
[117] ΔΔV R←ΔVR ΔΔNAME A APL*PLUS
[118] a A ΔΔVR←1 ΔFD ΔΔNAME A SHARP APL
[119] a A ΔΔVR←CRΔVR ΔCR ΔΔNAME A Other APL systems
[120] a Update selected vars:
[121] ΔΔGDSP←ΔΔGDSP, ρ ΔΔGLOBAL
[122] ΔΔLDSP←ΔΔLDSP, ρ ΔΔLOCAL
[123] a Branch unless fn locked:
[124] →(ρ ΔΔVR) ρ ΔΔL5
[125] a No identifiers found if locked:
[126] ΔΔGNUM←ΔΔGNUM, 0
[127] ΔΔLNUM←ΔΔLNUM, 0
[128] →ΔΔL9
[129] a Use origin 0 for fn parsing:
[130] ΔΔL5:ΔIO←0
[131] a Where does header end?
[132] ΔΔT←ΔΔVR ΔΔTCNL
[133] a Grab header of fn (less newline):
[134] ΔΔHDR←ΔΔT ρ ΔΔVR
[135] a Drop header from vis rep:
[136] ΔΔV R←ΔΔT ΔΔVR
[137] a Where does fn syntax end?
[138] ΔΔT←ΔΔHDR\';'
[139] a Localized vars:
[140] ΔΔLOC←ΔΔT ΔΔHDR
[141] a Drop local vars:
[142] ΔΔHDR←ΔΔT ρ ΔΔHDR
[143] a Drop leading junk:
[144] ΔΔHDR←(v \~ ΔΔHDR e' v')/ ΔΔHDR
[145] a Is there an explicit result?
[146] ΔΔT←(ρ ΔΔHDR) ΔΔIND ← ΔΔHDR\'→'
[147] a Explicit result (if any):
[148] ΔΔRESULT←(ΔΔT ΔΔIND) ρ ΔΔHDR

-378-
Chapter 13 Solutions

WORKSPACE DESIGN AND DOCUMENTATION

ѵ USEDBY (continued)

[149] a Remove result from header:
[150] ΔΔHDR' ',((×ρΔRESULT)+ρΔΔRESULT)↓ΔΔHDR
[151] a Starting indices of fn name, args:
[152] ΔΔSTART+1+(ΔΔHDR=' ')↓ρΔΔHDR
[153] a Lengths of names:
[154] ΔΔLEN=(1↓ΔΔSTART,1+ρΔΔHDR)-1+ΔΔSTART
[155] a No. of args:
[156] ΔΔT↑1+ρΔΔLEN
[157] a Indices of args (if any):
[158] ΔΔΛN+ρΔΔIND=((ΔΔT=2)ρ0),(×ΔΔT)ρΔΔT
[159] a Don't consider fn name:
[160] ΔΔSTART+ΔΔSTART[ΔΔIND]
[161] ΔΔLEN=ΔΔLEN[ΔΔIND]
[162] a Length of longest arg:
[163] ΔΔCOLS=f/0,ΔΔLEN
[164] a Raveled, blank mat of args:
[165] ΔΔARGS=((ΔΛN×ΔΔCOLS)ρ)''
[166] a Compute indices: (iLEN[1]),(iLEN[2]),...
[167] ΔΔI=ΔΔLEN/-1↓0,+/ΔΔLEN
[168] ΔΔI=ΔΔI+1↓ρΔΔI
[169] a Insert names into raveled matrix:
[170] ΔΔARGS[ΔΔI+ΔΔLEN/ΔΔCOLS×ΔΔN]↓ΔΔHDR[ΔΔI+ΔΔLEN/ΔΔSTART ]
[171] a Reshape to mat of args:
[172] ΔΔARGS=((ΔΛN,ΔΔCOLS)ρ)ΔΔARGS
[173] a Flag newline chars:
[174] ΔΔNL=ΔΔVR=ΔΔTCNL
[175] a Flag nonquotes:
[176] ΔΔNQ=ΔΔVR=ΔΔTCNL
[177] a Map of chars not in quote pairs (i.e. char
[178] a constants) within each fn line (i.e. an open
[179] a quote is closed at the end of the fn line).
[180] a Leading quotes are flagged 0; closing quotes are
[181] a flagged 1 (including 1st of double-quote pairs):
[182] ΔΔNCON=\ΔΔNQ\ΔΔNL\ΔΔT\¬1↓0,ΔΔT¬ΔΔNL/=\ΔΔNQ
[183] ΔΔNQ+0
[184] a Flag non-α chars (includes αs in quotes):
[185] ΔΔNC=ΔΔNCON×ΔΔVR=’α'
[186] a Flag newlines or αs (except as in quotes):
[187] ΔΔS=ΔΔNL=ΔΔNC
[188] a A map of chars which do not follow a α (ignoring αs
[189] a within quotes) within each fn line. αs are
[190] a flagged 0:
[191] ΔΔNCMT¬f\ΔΔS\ΔΔT¬1↓0,ΔΔT¬ΔΔS/ΔΔNC
[192] ΔΔS¬ΔΔNC+0
[193] a A map of chars which are not included within αs or '':
[194] ΔΔPARSE=ΔΔNCMT\ΔΔNCON
[195] ΔΔNCON=ΔΔNCMT+0
[196] a Flag digits and letters:
[197] ΔΔNUM=ΔΔPARSE\ΔΔVR=’0123456789'
[198] ΔΔALP=ΔΔPARSE\ΔΔVR=’ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz’

-379-
USEDBY (continued)

[199] A Flag blanks:
[200] ΔΔBL<ΔΔPARSE∧ΔΔVR=' '
[201] ΔΔPARSE<0
[202] A Flag alphanumeric chars:
[203] ΔΔAN<ΔΔNUM∧ΔΔALP
[204] A Pole vec of contiguous digits:
[205] ΔΔPNUM<ΔΔNUM∧'1'ΔΔT<ΔΔPAN/ΔΔALP
[206] ΔΔNUM<0
[207] A Pole vec of contiguous digits/letters:
[208] ΔΔPAN<ΔΔAN<1<0, ΔΔAN
[209] ΔΔAN<0
[210] A Pole vec of identifiers:
[211] ΔΔPID<ΔΔPAN ∧ΔΔTV<1ΔΔT<ΔΔPAN/ΔΔALP
[212] ΔΔALP<ΔΔPAN<0
[213] A Flag '0' before identifiers (Dnames):
[214] ΔΔT<ΔΔTV<0'= (ΔΔT<1 ΔΔPID)/ΔΔVR
[215] A Shift leading poles of Dnames to include 0:
[216] ΔΔPID<ΔΔTVΔΔPID>1<0ΔΔT
[217] ΔΔT<0
[218] A Flag char following ] after line no.:
[219] ΔΔSTART<1ΔΔPNUM<1ΔΔPNUM/2< ΔΔANL
[220] ΔΔANL<ΔΔPNUM<0
[221] A Pole vec of contiguous blanks:
[222] ΔΔPBL<ΔΔBL<1<0, ΔΔBL
[223] ΔΔBL<0
[224] A Flag 1st nonblank char in each line:
[225] ΔΔSTART<ΔΔSTART>ΔΔPBL<ΔΔPBL<1ΔΔPBL/ΔΔSTART
[226] ΔΔPBL<0
[227] A Pole vec of identifiers at start of line:
[228] ΔΔPSID<ΔΔPIDΔΔTV<1ΔΔT<ΔΔPID/ΔΔSTART
[229] A Pole vec of labels:
[230] ΔΔSTART<0
[231] ΔΔPLAB<ΔΔPSID<ΔΔTV<1 ΔΔT< ': '= ΔΔPSID/ΔΔVR
[232] ΔΔPSID<0
[233] A Start and end (+1) indices of identifiers:
[234] ΔΔIND<ΔΔPID/1ρΔΔPID
[235] A No. of identifiers:
[236] ΔΔIND=(ρΔΔIND)÷2
[237] ΔΔIND=(ΔΔIND, 2)ρΔΔIND
[238] A Start indices of identifiers:
[239] ΔΔIDSTART<ΔΔIND[;0]
[240] A Lengths of identifiers:
[241] ΔΔIDLEN<ΔΔIND[11]-ΔΔIDSTART
[242] A Starting indices of local vars:
[243] ΔΔSTART<1+(ΔΔLOC=' ')/1ρΔΔLOC
[244] A Lengths of local vars:
[245] ΔΔLEN<1ΔΔSTART, 1+ρΔΔLOC-1+ΔΔSTART
[246] A Length of longest ident.:
[247] ΔΔCOLS<((/ΔΔLEN)/(/ΔΔIDLEN))/ρΔΔRESULT/1↓ρΔΔARGS
[248] A Pad arg names to conform:
[249] ΔΔARGS<((1ρΔΔARGS), ΔΔCOLS)↑ΔΔARGS
[250] A 0 row mat if no result:
[251] ΔΔRESULT<((XΔΔRESULT), ΔΔCOLS)ρΔΔCOLS↑ΔΔRESULT
\[
\forall \text{ USED} \text{ BY (continued)}
\]

[252] a Raveled blank mat of local vars:
[253] \(\Delta T \leftarrow (\Delta \text{COLS} \times \Delta \text{N} \leftarrow \rho \Delta \text{START}) \rho' \)
[254] a Compute indices: \((\Delta \text{LEN}[1]),(\Delta \text{LEN}[2]),...\)
[255] \(\Delta I \leftarrow \Delta \text{LEN} / -1 \downarrow 0, +/\Delta \text{LEN}\)
[256] \(\Delta I \leftarrow \Delta I + \rho \Delta I\)
[257] a Insert names into raveled matrix:
[258] \(\Delta T[\Delta I + \Delta \text{LEN} / \Delta \text{COLS} \times \Delta \text{N}] + \Delta \text{LOC}[\Delta I + \Delta \text{LEN} / \Delta \text{START}]\)
[259] a Reshape to mat of local vars:
[260] \(\Delta \text{LOC} \leftarrow (\Delta \text{N}, \Delta \text{COLS}) \rho \Delta T\)
[261] a Raveled blank mat of identifiers:
[262] \(\Delta T \leftarrow (\Delta \text{COLS} \times \Delta \text{N} \leftarrow \rho \Delta \text{IDSTART}) \rho' \)
[263] a Compute indices: \((\Delta \text{IDLEN}[1]),(\Delta \text{IDLEN}[2]),...\)
[264] \(\Delta I \leftarrow \Delta \text{IDLEN} / -1 \downarrow 0, +/\Delta \text{IDLEN}\)
[265] \(\Delta I \leftarrow \Delta I + \rho \Delta I\)
[266] a Insert names into raveled matrix:
[267] \(\Delta T[\Delta I + \Delta \text{IDLEN} / \Delta \text{COLS} \times \Delta \text{N}] + \Delta \text{VR}[\Delta I + \Delta \text{IDLEN} / \Delta \text{IDSTART}]\)
[268] a Reshape to mat of identifiers:
[269] \(\Delta \text{IDS} \leftarrow (\Delta \text{N}, \Delta \text{COLS}) \rho \Delta T\)
[270] a Mat of label names:
[271] \(\Delta \text{LAB} \leftarrow (\Delta S+(\Delta \text{N}, 2) \rho \Delta \text{PID} / \Delta \text{PLAB})[;0]/\Delta \text{IDS}\)
[272] \(\Delta \text{PID} \leftarrow \Delta \text{PLAB} \leftarrow 0\)
[273] a Mat of referenced vars less labels:
[274] \(\Delta \text{RVAR} \leftarrow (\sim \Delta S) / \Delta \text{IDS}\)
[275] a Combine different types of vars and a vector
[276] a of their types:
[277] \(\Delta \text{IDS} \leftarrow (\Delta \text{LAB}, [0] \Delta \text{RESULT}, [0] \Delta \text{ARGS}, [0] \Delta \text{LOC}), [0] \Delta \text{RVAR}\)
[278] \(\Delta \text{IDTYPES} \leftarrow ((1 \rho \Delta \text{LAB}), (1 \rho \Delta \text{RESULT}), (1 \rho \Delta \text{ARGS}), (1 \rho \Delta \text{LOC}), 1 \rho \Delta \text{RVAR}) / 1 2 3 4 5\)
[279] a Select just the first distinct name (and type):
[280] \(\Delta T \leftarrow ((\Delta \text{IDS} \text{ CMIO} \Delta \text{IDS}) \Delta \text{IDS}) / \downarrow \rho \Delta \text{IDTYPES}\)
[281] \(\Delta \text{IDS} \leftarrow \Delta \text{IDS}[\Delta T;]\)
[282] \(\Delta \text{IDTYPES} \leftarrow \Delta \text{IDTYPES}[\Delta T]\)
[283] a Branch if no columns in IDS (i.e. no identifiers):
[284] \(\rightarrow (1 \rho \Delta \text{IDS}) \downarrow \Delta \text{L5A}\)
[285] a Retain \(\Omega\)-names only for system variables (add other
[286] a system vars on your APL system):
[287] \(\Delta S \leftarrow 5 4 \rho ' \Omega \text{IO} \Omega \text{PP} \Omega \text{DRL} \Omega \text{DCT} \Omega \text{DLX} '\)
[288] \(\Delta T ' \leftarrow \Delta \text{IDS}[;0]\)
[289] \(\Delta I \leftarrow \Delta T / \rho \Delta I\)
[290] \(\Delta R \leftarrow (1 \rho \Delta S) \sim \Delta S \text{ CMIO} ((\rho \Delta I), 4) \uparrow \Delta \text{IDS}[\Delta I;]\)
[291] a Branch if all \(\Omega\)-names are in this list:
[292] \(\rightarrow (\sim \Delta R) \downarrow \Delta \text{L5A}\)
[293] \(\Delta T ' \leftarrow \Delta \text{T} \uparrow \Delta \text{RR}\)
[294] a Squeeze out \(\Omega\)-names which are system fns (e.g. \(\Omega \text{EX})\):
[295] \(\Delta \text{IDS} \leftarrow \Delta \text{T} / \Delta \text{IDS}\)
[296] \(\Delta \text{IDTYPES} \leftarrow \Delta \text{T} / \Delta \text{IDTYPES}\)
[297] a Return to origin 1:
[298] \(\Delta \text{L5A} : \Omega \text{IO} \leftarrow 1\)
[299] a Insert new names into NAMES and convert to indices:
[300] \(\rightarrow (x(\Delta S \leftarrow 1 \rho \Delta \text{NAMES}) - \downarrow \Delta T + 1 \rho \Delta \text{IDS}) \uparrow \Delta \text{L8}, \Delta \text{L6}, \Delta \text{L7}\)
Chapter 13 Solutions

WORKSPACE DESIGN AND DOCUMENTATION

\[ \text{USEDBY (continued)} \]

[301] a NAMES has more columns than IDS:

[302] \[ \triangle L6: \triangle IDS \leftarrow (1 \rho \triangle IDS), \triangle S \uparrow \triangle IDS \]

[303] \[ \rightarrow \triangle L8 \]

[304] a IDS has more columns than NAMES:

[305] \[ \triangle L7: \triangle NAMES \leftarrow (1 \rho \triangle NAMES), \triangle T \uparrow \triangle \triangle NAMES \]

[306] a

[307] \[ \triangle L8: \triangle \triangle IND \leftarrow \triangle \triangle NAMES \triangle \triangle IND \]

[308] a Flag those not found:

[309] \[ \triangle T \leftarrow \triangle \triangle IND, \triangle R \leftarrow 1 \rho \triangle \triangle NAMES \]

[310] a Add to NAMES and compute indices:

[311] \[ \triangle \triangle NAMES \leftarrow \triangle \triangle NAMES, [1] \triangle T \uparrow \triangle IDS \]

[312] \[ \triangle S \leftarrow \triangle T, \triangle \rho \triangle T \]

[313] \[ \triangle \triangle IND \leftarrow \triangle \triangle S, \triangle \triangle R \leftarrow 1 \rho \triangle \triangle S \]

[314] \[ \triangle \triangle IDS \leftarrow \triangle \triangle \triangle IND \]

[315] a How many identifiers are locals?

[316] \[ \triangle N \leftarrow -1 + \triangle \triangle IDTYPES \downarrow 5 \]

[317] \[ \triangle \triangle LNUM \leftarrow \triangle \triangle LNUM, \triangle N \]

[318] \[ \triangle \triangle LLOCAL \leftarrow \triangle \triangle LOCAL, \triangle N \rho \triangle \triangle IDS \]

[319] \[ \triangle \triangle LTYPE \leftarrow \triangle \triangle LTYPE, \triangle N \rho \triangle \triangle IDTYPES \]

[320] a How many identifiers are globals:

[321] \[ \triangle \triangle IDS \leftarrow \triangle \triangle N \downarrow \triangle \triangle IDS \]

[322] \[ \triangle \triangle GNUM \leftarrow \triangle \triangle GNUM, \rho \triangle \triangle IDS \]

[323] \[ \triangle \triangle GLOBAL \leftarrow \triangle \triangle GLOBAL, \triangle \triangle IDS \]

[324] a

[325] a Branch unless this object has globals:

[326] \[ \triangle L9: \rightarrow (\times \triangle \triangle GNUM(\triangle \triangle J)) \downarrow \triangle \triangle L10 \]

[327] a Add a level to the "state indicator":

[328] \[ \triangle \triangle LCPARSED \leftarrow \triangle \triangle J, \triangle \triangle LCPARSED \]

[329] \[ \triangle \triangle LCGLOBAL \leftarrow 1, \triangle \triangle LCGLOBAL \]

[330] \[ \rightarrow \triangle \triangle LOOP \]

[331] a

[332] a Increment "state indicator" line:

[333] \[ \triangle \triangle L10: \triangle \triangle T \uparrow \triangle \triangle LCGLOBAL[1] \leftarrow 1 \downarrow \triangle \triangle LCGLOBAL[1] \]

[334] a Resume if more globals at this level:

[335] \[ \rightarrow (\triangle \triangle T \uparrow \triangle \triangle GNUM(\triangle \triangle LCPARSED[1])) \rho \triangle \triangle LOOP \]

[336] a Else drop a level from the "state indicator":

[337] \[ \triangle \triangle LCPARSED \leftarrow 1 \downarrow \triangle \triangle LCPARSED \]

[338] \[ \triangle \triangle LCGLOBAL \leftarrow 1 \downarrow \triangle \triangle LCGLOBAL \]

[339] a Continue if any levels left:

[340] \[ \rightarrow (\times \rho \triangle \triangle LCPARSED) \rho \triangle \triangle \triangle L10 \]
1. Here is a possible file structure for the "functions file":

FILE NAME: FNS  TIE NUMBER: up to you

DESCRIPTION: Contains representations of functions

<table>
<thead>
<tr>
<th>COMP. NO.</th>
<th>VARIABLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
</table>
| 1         | DIR      | Character matrix directory of the names (one per row) of the functions whose representations are stored on file. The matrix has as many columns as the longest function name has elements. The shorter names are left justified (padded to the right with spaces).
| 1+I       | VR       | Character vector visual representation of the function whose name is DIR[I]; (or canonical representation if your APL implementation of APL does not support visual representations).

[WSID: FNSFILE]
[WSID: FNSFILE]

\[ \Delta \Delta R \leftarrow \Delta \Delta T I E \ \text{PUTFN} \ \Delta \Delta \text{NAME}; \Delta \text{IO}; \Delta \Delta \text{DIR}; \Delta \Delta \text{IND}; \Delta \Delta T; \Delta \Delta \text{VR} \]

[1] a Appends or replaces to the functions file tied to $\Delta \Delta T I E$ the visual representation (or canonical representation) of the function named $\Delta \Delta \text{NAME}$.

[4] A Note: will not file functions whose names are local to PUTFN. Returns $\Delta \Delta \text{NAME}$ if successful, else empty vector.

[7] a Origin 1:

[8] $\Delta \text{IO} \leftarrow 1$

[9] a Construct visual representation:

[10] $\Delta \Delta \text{VR} \leftarrow \Delta \Delta \text{VR} \Delta \Delta \text{NAME} \ A \ A P L ^ * \ P L U S$

[11] a $\Delta \Delta \text{VR} \leftarrow 1 \ \text{OFD} \ \Delta \Delta \text{NAME} \ A \ S H A R P \ A P L$

[12] a $\Delta \Delta \text{VR} \leftarrow \Delta \text{OCR} \ \Delta \Delta \text{NAME} \ A \ O n \ o t h e r \ s y s t e m s$

[13] a Return empty vector if function locked:

[14] $\Delta \text{VR} \leftarrow "$

[15] $- \times / \rho \Delta \Delta \text{VR} ) \downarrow 0$

[16] a Read directory of function names:

[17] $\Delta \Delta \text{DIR} \leftarrow \text{OFREAD} \ \Delta \Delta T I E, 1 \ A \ A P L ^ * \ P L U S$

[18] a $\Delta \Delta \text{DIR} \leftarrow \text{OFREAD} \ \Delta \Delta T I E, 1 \ A \ S H A R P \ A P L$

[19] a Delete any blanks in name:

[20] $\Delta \text{R} \leftarrow (\Delta \Delta \text{NAME} \leftarrow (\Delta \Delta \text{NAME}^{'} \\ ' ) ) / \Delta \Delta \text{NAME}$

[21] a Search directory for function name (branch if not found):

[23] $\rightarrow((1 \rho \Delta \Delta \text{DIR}) \leftarrow \rho \Delta \Delta \text{NAME}) \rho \Delta \Delta \text{L} 1$

[24] $\Delta \Delta \text{IND} \leftarrow ((\Delta \Delta \text{DIR}) \leftarrow (1 \rho \Delta \Delta \text{DIR}) \uparrow \Delta \Delta \text{NAME}) \downarrow 1$

[25] $\rightarrow((\Delta \Delta \text{IND} \leftarrow 1 \rho \Delta \Delta \text{DIR}) \rho \Delta \Delta \text{L} 1$

[26] a Replace existing visual representation with a new one:

[28] $\Delta \Delta \text{VR} \leftarrow \text{OFREPLACE} \ \Delta \Delta T I E, 1 + \Delta \Delta \text{IND} \ A \ A P L ^ * \ P L U S$

[29] a $\Delta \Delta \text{VR} \leftarrow \text{OREPLACE} \ \Delta \Delta T I E, 1 + \Delta \Delta \text{IND} \ A \ S H A R P \ A P L$

[30] $\rightarrow 0$

[31] a Add name to directory. Branch unless $\Delta \Delta \text{NAME}$ too long:

[32] a too long:

[33] $\Delta \Delta \text{L} 1 \leftarrow ((\rho \Delta \Delta \text{NAME}) \leftarrow (1 \rho \Delta \Delta \text{DIR}) \rho \Delta \Delta \text{L} 2$

[34] a Pad columns in directory to length of $\Delta \Delta \text{NAME}$:

[35] $\Delta \Delta \text{DIR} \leftarrow ((1 \rho \Delta \Delta \text{DIR}) \rho \Delta \Delta \text{NAME}) \uparrow \Delta \Delta \text{DIR}$

[36] $\Delta \Delta \text{L} 2 \leftarrow \Delta \Delta \text{DIR} \leftarrow \Delta \Delta \text{DIR}, [1] (1 \rho \Delta \Delta \text{DIR}) \uparrow \Delta \Delta \text{NAME}$

[37] a Replace directory:

[38] $\Delta \Delta \text{DIR} \leftarrow \text{OFREPLACE} \ \Delta \Delta T I E, 1 \ A \ A P L ^ * \ P L U S$

[39] a $\Delta \Delta \text{DIR} \leftarrow \text{OREPLACE} \ \Delta \Delta T I E, 1 \ A \ S H A R P \ A P L$

[40] a Append function representation:

[41] $\Delta \Delta T \leftarrow \Delta \Delta \text{VR} \leftarrow \text{OFAPPEND} \ \Delta \Delta T I E \ A \ A P L ^ * \ P L U S \ \text{if } \text{OFAPPEND has result}$

[42] a $\Delta \Delta \text{VR} \leftarrow \text{OFAPPEND} \ \Delta \Delta T I E \ A \ A P L ^ * \ P L U S \ \text{if } \text{OFAPPEND has no result}$

[43] a $\Delta \Delta \text{VR} \leftarrow \text{OAPPEND} \ \Delta \Delta T I E \ A \ S H A R P \ A P L$
Chapter 14 Solutions

FILE DESIGN AND UTILITIES

[WSID: FNSFILE]

\[ \Delta R \leftarrow \Delta TIE \ GETFN \ \Delta \ NAME; DIO; \Delta D IR; \Delta A IND; \Delta A VR \]

1. \( \Delta \) Reads from the functions file tied to \( \Delta TIE \) the
2. \( \Delta \) visual representation (or canonical representation)
3. \( \Delta \) of the function named \( \Delta \ NAME \) and defines that
4. \( \Delta \) function in the active workspace. Note: will not
5. \( \Delta \) define functions whose names are local to \( \GETFN \).
6. \( \Delta \) Returns \( \Delta \ NAME \) if successful, empty vector if
7. \( \Delta \) function not found, numeric code if function not
8. \( \Delta \) definable.
9. \( \Delta \) Origin 1:
10. \( DIO \leftarrow 1 \)
11. \( \Delta \) Read directory of function names:
12. \( \Delta D IR \leftarrow \OFREAD \ \Delta TIE,1 \ \ \ APL^{PLUS} \)
13. \( \Delta D IR \leftarrow \OFREAD \ \Delta TIE,1 \ \ SHARP \ APL \)
14. \( \Delta \) Delete any blanks in name:
15. \( \Delta \ NAME \leftarrow (\Delta \ NAME \#)' / \Delta \ NAME \)
16. \( \Delta \) Search directory for function name (exit if not
17. \( \Delta \) found):
18. \( \Delta R \leftarrow '' \)
19. \( \rightarrow (1 \langle \rho \Delta D IR \rangle \langle \rho \Delta \ NAME \rangle \rho 0 \)
20. \( \Delta A IND \leftarrow (\Delta D IR A.= (1 \langle \rho \Delta D IR \rangle \uparrow \Delta \ NAME) \rho 1 \)
21. \( \rightarrow (\Delta A IND > 1 \langle \rho \Delta D IR \rangle) \rho 0 \)
22. \( \Delta \) Read visual (or canonical) representation from
23. \( \Delta A file: \)
24. \( \Delta A VR \leftarrow \OFREAD \ \Delta TIE,1 + \Delta A IND \ \ APL^{PLUS} \)
25. \( \Delta A VR \leftarrow \OFREAD \ \Delta TIE,1 + \Delta A IND \ \ SHARP \ APL \)
26. \( \Delta \) Define function in workspace (result is function
27. \( \Delta \) name or numeric code indicating WS FULL, SYMBOL
28. \( \Delta \) TABLE FULL,...):
29. \( \Delta R \leftarrow \DEF \ \Delta A VR \ \ APL^{PLUS} \)
30. \( \Delta R \leftarrow 3 \ \ DF \ \Delta A VR \ \ SHARP \ APL \)
31. \( \Delta R \leftarrow \DFX \ \Delta A VR \ \ SHARP \ APL \)

\[ \]

[WSID: FNSFILE]

\[ R \leftarrow TIE \ DROPFN \ NAME; DIO; D IR; A IND; LAST \]

1. \( R \) Removes from the functions file tied to \( TIE \)
2. \( \) the visual representation (or canonical
3. \( \) representation) of the function named \( NAME \).
4. \( R \) Returns \( NAME \) if successful, empty vector if
5. \( R \) function not found.
6. \( R \) Origin 1:
7. \( DIO \leftarrow 1 \)
8. \( \) Read directory of function names:
9. \( D IR \leftarrow \OFREAD \ TIE,1 \ \ APL^{PLUS} \)
10. \( D IR \leftarrow \OFREAD \ TIE,1 \ \ SHARP \ APL \)
11. \( \) Delete any blanks in name:
12. \( NAME \leftarrow (NAME \#)' / NAME \)
13. \( \) Search directory for fn name (exit if not found):
14. \( R \leftarrow '' \)
15. \( \rightarrow (1 \langle \rho D IR \rangle \langle \rho NAME \rangle \rho 0 \)
16. \( IND \leftarrow (DIR A.= (1 \langle \rho D IR \rangle \uparrow NAME) \rho 1 \)

-385-
V DROPFN (continued)

[17] ⍵(IND>1)pDIR)p0
[18] ⍵ Replace this name with last one in directory:
[19] LAST(玎DIR)[1]
[21] DIR←-1 0 4DIR
[22] ⍵ Replace this vis. rep. with last one on file:
[23] (⎨READ TIE,1+LAST) mingle TIE,1+IND ⍵ APL*PLUS
[24] ⍵DROP TIE,←1
[25] ⍵ (⎨READ TIE,1+LAST) mingle TIE,1+IND ⍵ SHARP APL
[26] ⍵ DROP TIE,←1
[27] ⍵ Replace directory:
[28] DIR mingle TIE,1 ⍵ APL*PLUS
[29] ⍵ DIR mingle TIE,1 ⍵ SHARP APL

2.

[WSID: FILEDOC]

V FILE FILEDOC PAGE;IO;PP;BOTTOM;CMPS;DATA;DONE;
FIRST;FOOT;HEIGHT;I;LAST;LEFT;LIM;LINES;MARGIN;N;NEW;
NONDISPLAY;OLDCMP;PNO;QUOTE;S;START;T;TCNL;TITLE;TOP;
TXT;W;WIDTH

[1] ⍵ Displays paged file documentation. All output is
[2] ⍵ via ⍵ so replace all ⍵ by custom fn (e.g. PRINT)
[3] ⍵ to redirect output. PAGE: rows, columns, margins
[4] ⍵ (top, bottom, left, right) FILE: tie no. if
[5] ⍵ APL*PLUS or SHARP APL; otherwise a file name.
[6] ⍵ Use origin 1:
[7] ⍵ IO←1
[8] ⍵ Format no.s to 10 digits (CLEAR WS default):
[9] ⍵ PP+10
[10] ⍵ Activate file if IBM’s workspace 2 VAPLFILE:
[14] ⍵ HEIGHT+PAGE[1]-TOP+BOTTOM
[17] ⍵ Construct newline character:
[18] ⍵ TCNL←QTcnL ⍵ APL*PLUS
[21] ⍵ Format today’s date:
[22] ⍵ D←*DTS[2 3 1]
[23] ⍵ D[(D=' ')\pD←'/'
[24] ⍵ Format the time:
[26] ⍵ Format the file name:
[27] ⍵ TITLE←,DFNAMES[DFNUM[FILE];] APL*PLUS
[28] ⍵ TITLE←,DNAMES[DFNUM[FILE];] SHARP APL
[29] ⍵ TITLE←FILE ⍵ Otherwise
\ FILEDOC (continued)
[30] A Delete leading/trailing blanks:
[31] TITLE*(~+/\TITLE=' ')\(\downarrow(+/\\TITLE=' ')\)\(\downarrow\)TITLE
[32] A Number of file components and first one:
[33] S<DFSIZE FILE A APL*PLUS
[34] A S<FSIZE FILE A SHARP APL
[35] A S<1,1+rHO FILE A 2 VAPLFILE
[36] START=S[1]
[37] LIM=S[2]
[38] N=LIM-START
[39] A If 2 VAPLFILE:
[40] CMPS<EXITttS<RHO FILE
[41] A LIM<N<CMPS<CMPS/\CMPS
[42] A Exit if none:
[43] \(\rightarrow\)N=0
[44] A Format page title:
[45] TITLE*TITLE,'(','(\#N),' COMPONENT',((1\#N)\'S''),') 'D',',',T
[46] A If 2 VAPLFILE:
[47] TITLE*TITLE,'(','(\#N),' COMPONENT',((1\#N)\'S''),') OF 'D',',',T
[48] TITLE*(TOPpTCNL),WIDTH*(LEFTp' '),FILE: ',TITLE
[49] A Insert page number:
[50] PNO=1
[51] T='PAGE 1'
[52] TITLE[(\#T)\#T<TITLE]=T
[53] A Build first page:
[54] TXT<TITLE,TCNL
[55] A Keep track of lines used so far (below top margin) in TXT:
[56] LINES=2
[57] QUOTE='''
[58] A Define characters which don't display normally:
[59] NONDISPLAY<OTCNL,OTCLF,OTCBS,OTCBEL,OTCDEL,OTCNUL,
[60] OTCESC,OTCFF A APL*PLUS
[61] A NONDISPLAY<DAV[1 2 157 159 160] A SHARP APL
[62] A NONDISPLAY<OTC A APL2
[63] MARGIN<LEFTp'
[64] A Include heading:
[65] T=TCNL,MARGIN,'COMPONENT SHAPE \rho VALUE',TCNL
[66] TXT<TXT,T,MARGIN,'-------- ----- -----'
[67] LINES=LINES+2
[68] A Loop by component:
[69] OLDCOMP=0
[70] FIRST=START
[71] I=START+1
[72] A If 2 VAPLFILE:
[73] A FIRST<CMPS[1]
[74] A I=0
[75] LOOP:\(\rightarrow\)(DONE+LIM<\#I+1)/L2
[76] A LOOP:\(\rightarrow\)(DONE+LIM<\#I+1)/L2 A 2 VAPLFILE
[77] A Read data from file:
[78] NEW<DFREAD FILE,I A APL*PLUS
[79] A NEW<DFREAD FILE,I A SHARP APL

-387-
FILEDOC (continued)

[80] A \texttt{NEW} \texttt{\&} \texttt{GET} CMPS[I] A 2 VAPLFILE

[81] A

[82] A \texttt{Branch unless this is the first component:}

[83] A \texttt{\rightarrow(}OLDCMP\texttt{)/L1}

[84] A \texttt{Last distinct object read from file:}

[85] A \texttt{DATA} \texttt{\&} \texttt{NEW}

[86] A \texttt{Earliest such component of that object:}

[87] A \texttt{OLDCMP}+I

[88] A \texttt{OLDCMP} \texttt{\&} \texttt{CMPS[I\text{\|LIM}] A 2 VAPLFILE}

[89] A \texttt{\rightarrow LOOP}

[90] A \texttt{Get next object if this one is identical (to}}

[91] A \texttt{a last one:}

[92] A \texttt{L1:} \texttt{\rightarrow((}p\texttt{DATA})\texttt{\&}p\texttt{NEW)/L2}

[93] A \texttt{If 2 VAPLFILE:}

[94] A \texttt{L1:} \texttt{\rightarrow(CMPS[I]1+CMPS[I+1])/L2}

[95] A \texttt{\rightarrow((}p\texttt{DATA})\texttt{\&}p\texttt{NEW)/L2}

[96] A \texttt{\rightarrow(\texttt{\&},DATA=NEW)/LOOP}

[97] A \texttt{Format DATA (components OLDCMP to I-1)...}

[98] A \texttt{Format shape:}

[99] A \texttt{\rightarrow(3<\texttt{\&}S)/S}

[100] A \texttt{L2:S+(p\texttt{DATA}),' \texttt{\&}'}

[101] A \texttt{Omit shape and } p \texttt{if a scalar; pad to line up}

[102] A \texttt{with } p \texttt{\'s:}

[103] A \texttt{S+(3<\texttt{\&}S)/S}

[104] A \texttt{Combine component no.(s) and shape; compute}

[105] A \texttt{remaining width:}

[106] A \texttt{T+OLDCMP#I+1}

[107] A \texttt{T+OLDCMP\&CMPS[I+1] A 2 VAPLFILE}

[108] A \texttt{T+(p\texttt{OLDCMP}),T/-',p\texttt{I+1}}

[109] A \texttt{T+(p\texttt{OLDCMP}),T/-',p\texttt{CMPS[I+1]} A 2 VAPLFILE}

[110] A \texttt{T+(9f\texttt{pT})\texttt{\&}-5f\texttt{pT})\texttt{\&}T}

[111] A \texttt{S+(MARGIN,T,'-','S}

[112] A \texttt{W+WIDTH-pS}

[113] A \texttt{Branch if a numeric variable:}

[114] A \texttt{\rightarrow(0<1f\texttt{\&}pDATA)=L3}

[115] A \texttt{Else data is char; consider only up to W chars:}

[116] A \texttt{DATA+(W\texttt{\&}\texttt{\&}\texttt{\&}pDATA)}

[117] A \texttt{Replace nondisplayable chars by \texttt{\&}:}

[118] A \texttt{DATA+(DATA\texttt{\&}\texttt{\&}NONDISPLAY)/p\texttt{DATA)}\texttt{\&}\texttt{\&}}

[119] A \texttt{Double up quote chars:}

[120] A \texttt{DATA+QUOTE,((1+DATA=QUOTE)/DATA),QUOTE}

[121] A \texttt{Branch if data will fit on a single line:}

[122] A \texttt{\rightarrow(W\texttt{\&}pDATA)/L4}

[123] A \texttt{Else truncate and show '...' :}

[124] A \texttt{DATA+(W+3\texttt{\&}pDATA),'...'

[125] A \texttt{\rightarrowL4}

[126] A \texttt{If data is numeric, consider only up to}

[127] A \texttt{W+2 elements:}

[128] A \texttt{L3:DATA+(1+\texttt{\&}W+2)/\texttt{\&}pDATA}

[129] A \texttt{DATA+\#DATA}

[130] A \texttt{Format value from empty array as 10:}

[131] A \texttt{DATA+DATA, (0<p\texttt{DATA})/10'}}
FILEDOC (continued)

® Branch if data will fit on a single line:
[132] →((W≥\$DATA))/L4

® Else trunc. at last space within wid and show '...':
[134] DATA←((+/\$' '≡FW+3\$DATA)\$DATA),'

® Append variable definition to page:
[136] L4:TXT←TXT,TCNL,S,DATA

LINES←LINES+1
DATA←NEW
OLDCMP←I

® OLDCMP←CMPS[I] a 2 VAPLFILE

L4:TXT←TXT,TCNL,S,DATA

® Last←CMPS[I−1] a 2 VAPLFILE

® Branch for more unless bottom of page (2 lines
for footnote) or end of file:
[146] →((DONE<LINES<HEIGHT+2)/LOOP

® Construct footnote:

® Construct footnote:

L5:T←FIRST"I-LAST
FOOT←(-WIDTH)↑'COMPONENT',(T'S'),' ',(≠FIRST),T/’ TO
',≠LAST

® Display page:

® Display page:

TXT←((HEIGHT−LINES)\$TCNL),FOOT,BOTTOM\$TCNL
FIRST←LAST+I
FIRST←LAST+CMPS[I−ILIM] a 2 VAPLFILE

® Exit if no more components:
[156] →DONE/0

® Format new page:
[158] TXT←TITLE,TCNL

PNO←PNO+1

TXT←'PAGE ',≠PNO
TXT[(-≠T)↑≠TITLE]←T

LINES←2

® Insert new heading:

® Insert new heading:

TXT←TCNL,MARGIN,'COMPONENT
SHAPE \$ VALUE',TCNL

TXT←TXT,T,MARGIN,'---------- ----- -----'
3. [WSID: MULTIFLO]

   \[ EMPLOYEES;F1;F2;F3;G;GOOD;IND;NUM;P;R \]

   [1] \( v \) ILLUSTRATION OF FILE UTILITY FUNCTIONS.
   [2] \( v \) Assumes employee information is on file. The file
   [3] \( v \) is identified by the global variable FP (file
   [4] \( v \) parameters). Fields of file are:
   [5] \( v \) 1. Employee number
   [6] \( v \) 2. Employee name
   [7] \( v \) 3. Employee age
   [8] \( v \) Uses subfns: IOTA,CATRECWS,DELREC,SELECTWS.
   [9] \( v \) Ask for choice on same line:
   [10] \( v \) CHOOSE: \( \text{ADD, DELETE, LIST OR END} \)'
   [11] \( v \) \( R \leftrightarrow (pP) \leftrightarrow \)'
   [12] \( v \) Branch based on 1st char of response:
   [13] \( v \) \( \leftrightarrow (\text{'ADLE'}=1 \leftrightarrow R) / \text{ADD,DELETE,LIST,END} \)
   [14] \( v \) \( \leftrightarrow '** INVALID CHOICE. CHOOSE FROM: ADLE' \)
   [15] \( v \) \( \rightarrow \text{CHOOSE} \)
   [16] \( v \)
   [17] \( v \)
   [18] \( v \) ADD: \( \leftrightarrow '\text{EMPLOYEE NUMBER (OR 0 IF DONE)}' \)
   [19] \( v \) \( F1 \leftrightarrow ,D \)
   [20] \( v \) Continue if exactly 1 number entered:
   [21] \( v \) \( (1=pF1) / A1 \)
   [22] \( v \) \( \leftrightarrow '** ENTER 1 NUMBER' \)
   [23] \( v \) \( \rightarrow \text{ADD} \)
   [24] \( v \) Branch to choice question if 0 entered:
   [25] \( v \) \( A1: \rightarrow (0=pF1) / \text{CHOOSE} \)
   [26] \( v \) Continue unless employee number already exists:
   [27] \( v \) \( (\text{'-1'=1} p (\text{FP,1}) \leftrightarrow IOTA F1) \leftrightarrow A2 \)
   [28] \( v \) \( \leftrightarrow '** EMPLOYEE ',(\text{'F1}),' ALREADY IN LIST' \)
   [29] \( v \) \( \rightarrow \text{ADD} \)
   [30] \( v \) A2: \( \leftrightarrow '\text{EMPLOYEE NAME (MAX 25 CHARACTERS)}': ' \)
   [31] \( v \) Ask for name at end of same line:
   [32] \( v \) \( F2 \leftrightarrow (pP) \leftrightarrow \)'
   [33] \( v \) Continue unless name too long:
   [34] \( v \) \( (25=pF2) / A3 \)
   [35] \( v \) \( \leftrightarrow '** NAME TOO LONG' \)
   [36] \( v \) \( \rightarrow A2 \)
   [37] \( v \) A3: \( \leftrightarrow '\text{EMPLOYEE AGE} ' \)
   [38] \( v \) \( F3 \leftrightarrow ,D \)
   [39] \( v \) Continue if exactly 1 number entered:
   [40] \( v \) \( (1=pF3) / A4 \)
   [41] \( v \) \( \leftrightarrow '** ENTER 1 NUMBER' \)
   [42] \( v \) \( \rightarrow A3 \)
   [43] \( v \) Continue if a valid age:
   [44] \( v \) \( A4: \rightarrow (F3=1 \leftrightarrow F3) \wedge (F3=17) \wedge (F3=99) / A5 \)
   [45] \( v \) \( \leftrightarrow '** AGE MUST BE INTEGER FROM 17 TO 99' \)
   [46] \( v \) \( \rightarrow A3 \)
   [47] \( v \) Pad name to length 25:
   [48] \( v \) \( A5: F2 \leftrightarrow 25 \leftrightarrow F2 \)
   [49] \( v \) Catenate new values and ask for more:
   [50] \( v \) \( Fp \leftrightarrow Fp \leftrightarrow \text{CATRECWS 1} \)
   [51] \( v \) \( \rightarrow \text{ADD} \)
Chapter 14 Solutions

FILE DESIGN AND UTILITIES

\[ \text{\textcopyright EMPLOYEES (continued)} \]
\[ \text{\[52\]} \text{a} \]
\[ \text{\[53\]} \text{a} \]
\[ \text{\[54\]} \text{DELETE:} \text{\textasciitilde} \text{ENTER EMPLOYEE NUMBERS TO DELETE} \]
\[ \text{\[55\]} \text{a Ravel to insure a vector, not scalar:} \]
\[ \text{\[56\]} \text{NUM\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \]
\[ \text{\[57\]} \text{a Continue if all valid numbers:} \]
\[ \text{\[58\]} \text{IND\textasciitilde} \text{(FP,1)IOTA NUM} \]
\[ \text{\[59\]} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} \text{\textascii
INITFILE (continued)

DEL<FP[11]

→(×DEL)↓L1

ERROR(~(|DEL|≤|INCR|)'/NONEXISTENT DELETION FIELD NUMBER'

ERROR(1v,.#FT[1;|DEL|]')/USE FT=1 1 FOR DELETION FIELD'

L1:L←0

LAY<FP[1]

→(×LAY)↓L2

ERROR(~LAY€|INCR|)'/NONEXISTENT LAYER FIELD NUMBER'

ERROR(0=W<FT[1;LAY|)'/LAYER FIELD INACTIVE'

ERROR(LAY=|DEL|)'/LAYER AND DELETION FIELDS ARE THE SAME'

L←(0,(W#1)ρW)ρ0

→(2#FT[2;LAY])ρL2

L←(0,(W#1)ρW)ρ''

L2:R←' 'OFAPPEND TIE

R←(0 0 ρ''')OFAPPEND TIE

R←(0 0 ρ''')OFAPPEND TIE

R<FT OFAPPEND TIE

R←(10)OFAPPEND TIE

R←(10)OFAPPEND TIE

R<FP OFAPPEND TIE

R←(10)OFAPPEND TIE

R←L OFAPPEND TIE

R←(10)OFAPPEND TIE

→(DISP≤10)ρ0

R←l0

LOOP:→(DISP>R OFAPPEND TIE)ρLOOP

WSID: MULTIFLO

NFP<FP CATREC MAT; ARPS; BIT; BLK; C; CMP; DATA; DCMP; DEL;
DISP; F; FLD; FREC; FT; FILL; GOOD; I; INCR; INDS; LAYER; LC; LEAD;
LV; M; MIN; N; NFLD; NREC; RMAT; RPS; S; SDISP; SETS; T; TIE; VEC;
W; WID; DIO

a Catenates rows of MAT to file.

ERROR((2<ρMAT)v1#ρρFP)'/RANK ERROR'

ERROR(1I#ρFP)'/LENGTH ERROR'

NFP<FP

a Convert scalar or vector to matrix:

→(2=ρρMAT)ρL1

MAT←(2↑1 1 ,ρMAT)ρMAT

a Exit if no records to catenate:

→(×1ρρMAT)↓0

DIO←1

TIE<FP[2]

INCR<FP[3]

DISP<FP[4]

BLK<FP[5]

DEL<FP[11]

FILL<FP[11]<0

FT<OFREAD TIE,4
\(\text{CATREC (continued)}\)

\[\text{[18] Widths (no. cols.) of fields:}\]
\[\text{WID}\leftarrow\text{FT[1;]}\]

\[\text{[19] Indices of active fields:}\]
\[\text{FLD}\leftarrow(\times\text{WID})/\text{INCR}\]

\[\text{[20] Exclude deletion field:}\]
\[\text{NFLD}\leftarrow\text{FLD}-(\text{FLD} \# \text{DEL})/\text{FLD}\]

\[\text{[21] Index of active fields:}\]
\[\text{FLD}(\times\text{WID})/\text{INCR}\]

\[\text{[22] Exclude deletion field:}\]
\[\text{NFLD}\leftarrow\text{FLD}-(\text{FLD} \# \text{DEL})/\text{FLD}\]

\[\text{[23] Error check:}\]
\[\text{ERROR}((1\times/\text{MAT})(1\downarrow\text{MAT})\#+/\text{WID}(\text{FLD})')/\text{LENGTH}\]

\[\text{[24] Domain error:}\]
\[\text{ERROR}((2\times\text{FT}[2;\text{FLD}])v_{0}=1\uparrow0\text{MAT})'/\text{DOMAIN}\]

\[\text{[25] Extend singleton across all columns:}\]
\[\text{MAT}\leftarrow(1,+/\text{WID(FLD)})\text{MAT}\]

\[\text{[26] Branch unless ARPS should be read:}\]
\[\text{ARPS}\leftarrow\text{DFREAD TIE,8}\]

\[\text{[27] Branch unless layered file:}\]
\[\text{L3}\leftarrow(\times\text{FP[1]})\downarrow\text{L7}\]

\[\text{[28] Read layer values:}\]
\[\text{LV}\leftarrow\text{DFREAD TIE,9}\]

\[\text{[29] Compute the col inds of MAT with layer values:}\]
\[\text{LC}\leftarrow0,+/\text{WID(FLD)}[\text{FLD}\times\text{FP[1]}]+\text{WID(FP[1])}\]

\[\text{[30] Convert to scalar if vector fld:}\]
\[\text{VEC}\leftarrow1=\text{LC}\]

\[\text{[31] Use layer value of 1st row of MAT:}\]
\[\text{L4}\leftarrow\text{LAYER}+\text{MAT[1;LC]}\]

\[\text{[32] Branch if a vector layer field:}\]
\[\text{VEC}\leftarrow\text{L5}\]

\[\text{[33] Flag rows of MAT in this layer:}\]
\[\text{GOOD}\leftarrow\text{MAT[;LC]}=\text{LAYER}\]

\[\text{[34] Put remaining rows in RMAT:}\]
\[\text{L6}\leftarrow\text{RMAT}\leftarrow(\sim\text{GOOD})\#\text{MAT}\]

\[\text{[35] Consider only non-full sets in this layer}\]
\[\text{SETS}\leftarrow(\sim\text{SGS})\vee\text{SETS}\text{ARPS}\text{BLK}\]

\[\text{[36] Branch if more than 1 set needed:}\]
\[\text{L8}\leftarrow\text{SETS}\leftarrow\text{SETS}/\text{SETS}\leftarrow\text{SETS}\text{ARPS}\text{BLK}\]

\[\text{[37] No. records filed so far:}\]
\[\text{FREC}\leftarrow0\]

\[\text{[38] Branch if no slots available in existing sets:}\]
\[\text{L9}\leftarrow\sim(\text{BLK}\times\text{MIN})\leftarrow1\downarrow\text{ARPS(SETS)}\text{ARPS}\text{BLK}\]

\[\text{[39] Branch if more than 1 set needed:}\]
\[\text{T}\leftarrow\text{NREC}-\text{FREC}\]

\[\text{[40] \rightarrow(T>\text{BLK-MIN})\text{L10}}\]
\[ \text{\textbf{Chapter 14 Solutions}} \]
\[ \text{\textbf{FILE DESIGN AND UTILITIES}} \]

\[ \text{\textbf{V CATREC (continued)}} \]

\[ \text{(71) \text{A Choose fullest set which will hold all recs:}} \]
\[ S \leftarrow \text{BLK-ARPS[SETS]} \]
\[ S \leftarrow \text{SETS[S\{}(S\geq T)/S\}} \]
\[ \rightarrow \text{L11} \]

\[ \text{(75) \text{A Choose set with most empty slots:}} \]
\[ \text{L10:} S \leftarrow \text{SETS[\{}(ARPS[SETS])\text{MIN}\} \]

\[ \text{(77) \text{A No. records to be inserted within the set:}} \]
\[ \text{L11:} M \leftarrow \text{TRPS[SETS]} \]

\[ \text{(79) \text{A No. records to be catenated within the set:}} \]
\[ N \leftarrow (T-M)\text{BLK-RPS[S]} \]

\[ \text{(81) \text{A Displacement (no. components) before this set:}} \]
\[ \text{SDISP} \leftarrow \text{DISP+INCRxS+1} \]

\[ \text{(83) \text{A Branch if no deletion field:}} \]
\[ \rightarrow ((\text{XDEL})\downarrow \text{L12}} \]

\[ \text{(85) \text{A Read deletion field for set S:}} \]
\[ \text{BIT} \leftarrow \text{DFREAD DCMP+TIE,DEL+SDISP} \]

\[ \text{(87) \text{A Branch if no records to be inserted:}} \]
\[ \rightarrow ((\text{X})\downarrow \text{L12}} \]

\[ \text{(89) \text{A Indices of available insertion slots:}} \]
\[ \text{INDS} \leftarrow \text{M}(\text{BIT})/\text{BIT} \]

\[ \text{(91) \text{A Columns of MAT filed so far:}} \]
\[ \text{L12:} C \leftarrow 0 \]

\[ \text{(93) \text{A Next field index:}} \]
\[ I \leftarrow 1 \]

\[ \text{(95) \text{A Loop by active field:}} \]
\[ \text{L13:} \rightarrow (I>\text{NFLD})\rightarrow \text{L19} \]

\[ \text{(97) \text{A Field number:}} \]
\[ F \leftarrow \text{FLD[I]} \]

\[ \text{(99) \text{A Field width (no. columns):}} \]
\[ W \leftarrow \text{WID[F]} \]

\[ \text{(101) \text{A Read field F for set S:}} \]
\[ \text{DATA} \leftarrow \text{DFREAD CMP+TIE,F+SDISP} \]

\[ \text{(103) \text{A Branch if a matrix field:}} \]
\[ \rightarrow ((W>1)\downarrow \text{L15}} \]

\[ \text{(105) \text{A Column of MAT if a vector field:}} \]
\[ C \leftarrow C+1 \]

\[ \text{(107) \text{A Branch if no records to insert:}} \]
\[ \rightarrow ((\text{X})\downarrow \text{L14}} \]

\[ \text{(109) \text{DATA[INDS]} \leftarrow \text{MAT[FREC+M;C]} \]

\[ \text{(110) \text{A Branch if no records to catenate:}} \]
\[ \rightarrow ((\text{X})\downarrow \text{L18}} \]

\[ \text{(112) \text{L14:} DATA} \leftarrow \text{DATA, MAT[}(\text{FREC+M)+\text{N;C]} \]
\[ \rightarrow \text{L18} \]

\[ \text{(114) \text{A Branch if no records to insert:}} \]
\[ \text{L15:} \rightarrow ((\text{X})\downarrow \text{L16}} \]

\[ \text{(116) \text{DATA[INDS]} \leftarrow \text{MAT[FREC+M;C+W]} \]

\[ \text{(117) \text{A Branch if no records to catenate:}} \]
\[ \rightarrow ((\text{X})\downarrow \text{L17}} \]

\[ \text{(119) \text{L16:} DATA} \leftarrow \text{DATA, MAT[}(\text{FREC+M)+\text{N;C+W]} \]

\[ \text{(120) \text{L17:} C \leftarrow C+W} \]

\[ \text{(121) \text{L18:} DATA} \leftarrow \text{DFREPLACE CMP} \]

\[ \text{(122) I \leftarrow I+1} \]

\[ \text{(123) \rightarrow \text{L13}} \]
Chapter 14 Solutions

FILE DESIGN AND UTILITIES

\[\text{CATREC (continued)}\]

[124] \(\text{a Branch if no deletion field:}\)

[125] L19: LEAD+1

[126] \(\rightarrow (\times \text{DEL}) \downarrow L22\)

[127] \(\text{a Branch if no records to insert:}\)

[128] \(\rightarrow (\times M) \downarrow L20\)

[129] \(\text{a Turn active record bits on:}\)

[130] BIT[INDS] \(\leftarrow 1\)

[131] LEAD \(\^ / \text{BIT} = \backslash \text{BIT}\)

[132] \(\text{a Branch if no records to catenate:}\)

[133] \(\rightarrow (\times N) \downarrow L21\)

[134] L20: BIT \(\leftarrow \text{BIT}, N_0\)

[135] L21: BIT 0FREPLACE DCMP

[136] \(\text{a Increment FREc by no. records added to this set:}\)

[137] L22: FREc+FREc+M+N


[140] \(\text{a Replace layer value if file layered and}\)

[141] \(\text{a set initially empty:}\)

[142] \(\rightarrow (T \times FP[11]) \downarrow L24\)

[143] \(\text{a Branch if a vector layer field:}\)

[144] \(\Rightarrow \text{VECpL23}\)

[145] LV[S;1] \(\leftarrow \text{LAYER}\)

[146] \(\Rightarrow L24\)

[147] L23: LV[S] \(\leftarrow \text{LAYER}\)

[148] L24: ARPS[S] \(\leftarrow (\sim 1) \times L20 \times M+N+1 \times \text{ARPS[S]}\)


[150] \(\text{a Exit if all of MAT filed:}\)

[151] \(\rightarrow (\text{NREC} \times \text{FREC}) \downarrow L34\)

[152] \(\Rightarrow L9\)

[153] \(\text{a Add new set; no. fields to be appended:}\)

[154] L25:NFLD \(\leftarrow \text{FP}[3]\)

[155] \(\text{a No. records to be appended in next set:}\)

[156] N \(\leftarrow \text{BLK} \text{NREC} \times \text{FREC}\)

[157] \(\text{a Columns of MAT filed so far:}\)

[158] C \(\leftarrow 0\)

[159] \(\text{a Next field number:}\)

[160] F \(\leftarrow 1\)

[161] \(\text{a Loop by field:}\)

[162] L26: \(\rightarrow (F \times \text{NFLD}) \downarrow L32\)

[163] W \(\leftarrow \text{WID}[F]\)

[164] \(\text{a Branch unless a latent field:}\)

[165] \(\rightarrow (\times W) \downarrow L27\)

[166] DATA \(\leftarrow 0\)

[167] \(\Rightarrow L31\)

[168] \(\text{a Branch if a matrix field:}\)

[169] L27: \(\rightarrow (W \times 1) \downarrow L29\)

[170] \(\text{a Branch unless it's the deletion field:}\)

[171] \(\rightarrow (\text{DEL} \# F) \downarrow L28\)

[172] DATA \(\leftarrow N_0\)

[173] \(\Rightarrow L30\)

[174] \(\text{a Column of MAT if a vector field:}\)

[175] L28: C \(\leftarrow C+1\)

[176] DATA \(\Rightarrow \text{MAT}[\text{FREC} \times 1; N; C]\)
Chapter 14 Solutions  FILE DESIGN AND UTILITIES

\[ \text{CATREC (continued)} \]

\[ L30 \rightarrow \]

\[ L29: \text{DATA} \leftarrow \text{MAT} \left[ \text{FREC} + N; C + W \right] \]

\[ L30: \rightarrow \text{FILL} \rightarrow L31 \]

\[ \text{DATA} \leftarrow (\text{BLK}, 1 \cdot \text{DATA}) \cdot \text{DATA} \]

\[ L31: T \leftarrow \text{DATA} \cdot \text{FAPPEND TIE} \]

\[ F \leftarrow F + 1 \]

\[ \rightarrow L26 \]

\[ \text{a Branch unless set must be padded to BLK records:} \]

\[ L32: \text{FREC} \leftarrow \text{FREC} + N \]

\[ \text{ARPS} \leftarrow \text{ARPS}, N \]

\[ \text{RPS} \leftarrow \text{RPS}, T \leftarrow N \cdot \text{BLK} \times \text{FILL} \]

\[ \text{FP}[7] \leftarrow \text{FP}[7] + T \]

\[ \text{FP}[6] \leftarrow \text{FP}[6] + 1 \]

\[ \text{FP}[9] \leftarrow \text{FP}[9] + 1 \]

\[ \text{a Catenate layer value if file layered:} \]

\[ (\times \text{FP}[1]) \rightarrow L33 \]

\[ \text{LV} \leftarrow \text{LV}, [1] \text{ LAYER} \]

\[ \text{a Continue unless all of MAT filed:} \]

\[ L33: \rightarrow (\text{NREC} \# \text{FREC}) \cdot L25 \]

\[ L34: \text{FP}[10] \leftarrow \text{FP}[10] + NREC \]

\[ \text{a Branch unless layered:} \]

\[ (\times \text{FP}[1]) \rightarrow L36 \]

\[ \text{a Branch if no data left to file:} \]

\[ (\times \text{RMAT}) \rightarrow L35 \]

\[ \text{a Put remaining rows in MAT, continue:} \]

\[ \text{MAT} \leftarrow \text{RMAT} \]

\[ \rightarrow L4 \]

\[ L35: \text{LV} \cdot \text{FREPLACE TIE}, 9 \]

\[ L36: \text{RPS} \cdot \text{FREPLACE TIE}, 8 \]

\[ (\times \text{DEL}) \rightarrow L37 \]

\[ \text{ARPS} \cdot \text{FREPLACE TIE}, 10 \]

\[ L37: \text{FP} \cdot \text{FREPLACE TIE}, 7 \]

\[ NFP \leftarrow \text{FP} \]

\[ \text{[WSID: MULTIFLO]} \]

\[ \text{NFP} \leftarrow \text{FP} \cdot \text{CATRECWS NREC; ARPS; BIT; BLK; CMP; DATA; DCMP; DEL; DISP; F; FILED; FLD; FREC; FT; FILL; GOOD; I; INCR; INDS; LAYER; LEAD; LV; M; MIN; N; NFLD; NR; RANK; RPS; S; SDISP; SETS; SHAPE; T; TIE; VAR; VEC; W; WID; OIO} \]

\[ \text{a Catenates elements/rows of F1, F2, F3, ... to file.} \]

\[ \text{ERROR}(1 \# \text{RO FP}) \cdot 1 \cdot \text{.#NREC} / \text{'RANK ERROR'} \]

\[ \text{NREC} \leftarrow \text{.O NREC} \]

\[ \text{ERROR}(11 \# \text{RO FP}) / \text{'LENGTH ERROR'} \]

\[ \text{NFP} \leftarrow \text{FP} \]

\[ \text{OIO} \leftarrow 1 \]

\[ \text{TIE} \leftarrow \text{FP}[2] \]

\[ \text{INCR} \leftarrow \text{FP}[3] \]

\[ \text{DISP} \leftarrow \text{FP}[4] \]

\[ \text{BLK} \leftarrow \text{FP}[5] \]
Chapter 14 Solutions

FILE DESIGN AND UTILITIES

\[\text{\textcopyright CATRECWS (continued)}\]

[11] \text{DEL} \leftarrow \text{FP}[11]
[12] \text{FILL} \leftarrow \text{FP}[11] < 0
[13] \text{FT} \leftarrow \text{UFREAD TIE}, 4
[14] \text{Fields (no. cols.) of fields:}
[15] \text{WID} \leftarrow \text{FT}[1];
[16] \text{Indices of active fields:}
[17] \text{FLD} \leftarrow (\times \text{WID}) / \text{INCR}
[18] \text{Exclude deletion field:}
[19] \text{NFLD} \leftarrow \text{FLD} \leftarrow (\text{FLD} \# \text{DEL}) / \text{FLD}
[20] \text{Index of next active field:}
[21] \text{I} \leftarrow 1
[22] \text{Loop by active field to verify field vars}
[23] \text{(bypass this loop to make the fn faster and}
[24] \text{to live dangerously):}
[25] \text{L1:} (I > \text{NFLD}) \rightarrow \text{L2}
[26] \text{Field number:}
[27] \text{F} \leftarrow \text{FLD}[I]
[28] \text{Field width (no. columns):}
[29] \text{W} \leftarrow \text{WID}[F]
[30] \text{Look at field variable:}
[31] \text{RANK} \leftarrow \text{SHAPE} \leftarrow \text{VAR} \leftarrow 'F', # F
[32] \text{ERROR}((2 \times \text{FT}[2; F]) \# 0 = 1 \times \text{VAR}) / 'DOMAIN ERROR'
[33] \text{Continue if singleton data:}
[34] \text{I} \leftarrow I + 1
[35] \rightarrow (I \times . = \text{SHAPE}) \rightarrow \text{L1}
[36] \text{ERROR}((\text{RANK} > 2) / 'RANK ERROR'
[37] \text{ERROR}((\text{RANK} = 1) \times \text{W} = 1 \times \text{SHAPE}) \leftarrow (\text{W} = 1) \times \text{NREC} = 1 \times \text{SHAPE} \leftarrow (\text{RANK} = 2) \times (\text{SHAPE} \times = \text{NREC}, \text{W} \times \text{SHAPE} = 1, \text{W}) / 'LENGTH ERROR'
[38] \rightarrow \text{L1}
[39] \text{Exit if no records to catenate:}
[40] \text{L2:} \rightarrow (\times \text{NREC} \downarrow 0)
[41] \text{ARPS} \leftarrow \text{RPS} \leftarrow \text{UFREAD TIE}, 8
[42] \text{Branch unless ARPS should be read:}
[43] \rightarrow (=/\text{FP}[7, 10]) \rightarrow \text{L3}
[44] \text{ARPS} \leftarrow \text{UFREAD TIE}, 10
[45] \text{Branch unless layered file:}
[46] \text{L3:} \rightarrow (\times \text{FP}[1]) \rightarrow \text{L7}
[47] \text{Read layer values:}
[48] \text{LV} \leftarrow \text{UFREAD TIE}, 9
[49] \text{Flag records filed so far:}
[50] \text{FILED} \leftarrow \text{NREC} \rightarrow 0
[51] \text{Look at layer field:}
[52] \text{L4:} \text{VAR} \leftarrow 'F', # \text{FP}[1]
[53] \text{W} \leftarrow \text{WID}[\text{FP}[1]]
[54] \text{VAR} \leftarrow (\text{NREC}, (\text{W} \times 1) \times \text{VAR}
[55] \text{Branch if vector field:}
[56] \text{VEC} \leftarrow \text{W} = 1
[57] \rightarrow \text{VEC} \rightarrow \text{L5}
[58] \text{Flag records with layer of 1st unfiled rec:}
[59] \text{LAYER} \leftarrow \text{VAR}[\text{FILED} \downarrow 0 ;]
[60] \text{GOOD} \leftarrow \text{VAR} \times = \text{LAYER}
[61] \text{...and sets with this record:}
[62] \text{SETS} \leftarrow \text{LV} \times = \text{LAYER}
\[ \text{CATRECWS (continued)} \]

\[ 63 \] \( \rightarrow L6 \)
\[ 64 \] \( L5:: \text{LAYER} \leftrightarrow \text{VAR} [\text{FILED} \downarrow 0] \)
\[ 65 \] \( \text{GOOD} \leftrightarrow \text{VAR} = \text{LAYER} \)
\[ 66 \] \( \text{SETS} \leftrightarrow \text{LV} = \text{LAYER} \)
\[ 67 \] \( \text{a Convert to indices:} \)
\[ 68 \] \( L6:: \text{GOOD} \leftrightarrow \text{GOOD} \downarrow 1 \text{GOOD} \)
\[ 69 \] \( \text{NR} \leftrightarrow \text{GOOD} \)
\[ 70 \] \( \text{a Consider only non-full sets in this layer or} \)
\[ 71 \] \( \text{a empty sets in any layer:} \)
\[ 72 \] \( \text{SETS} \leftrightarrow (\text{ARPS} = 0) \uparrow \text{SETS} \uparrow \text{ARPS} \uparrow \text{BLK} \)
\[ 73 \] \( \rightarrow L8 \)
\[ 74 \] \( L7:: \text{GOOD} \leftrightarrow \text{NR} \leftrightarrow \text{NREC} \)
\[ 75 \] \( \text{SETS} \leftrightarrow \text{ARPS} \uparrow \text{BLK} \)
\[ 76 \] \( \text{a Convert to indices:} \)
\[ 77 \] \( L8:: \text{SETS} \leftrightarrow \text{SETS} \downarrow 1 \text{SETS} \)
\[ 78 \] \( \text{a No. records filed so far:} \)
\[ 79 \] \( \text{FREC} \downarrow 0 \)
\[ 80 \] \( \text{a Branch if no slots available in existing sets:} \)
\[ 81 \] \( L9:: (\text{BLK} \leq \text{MIN} \downarrow 1) \uparrow 1 \text{ARPS} [\text{SETS}] \uparrow L24 \)
\[ 82 \] \( \text{a Branch if more than 1 set needed:} \)
\[ 83 \] \( T \leftrightarrow \text{NR} \leftrightarrow \text{FREC} \)
\[ 84 \] \( \rightarrow (T > \text{BLK} \downarrow \text{MIN}) \uparrow L10 \)
\[ 85 \] \( \text{a Choose fullest set which will hold all recs:} \)
\[ 86 \] \( S \uparrow \text{BLK} \downarrow 1 \text{ARPS} [\text{SETS}] \)
\[ 87 \] \( S \uparrow \text{SETS} [\text{SETS} / (S \geq T) / S] \)
\[ 88 \] \( \rightarrow L11 \)
\[ 89 \] \( \text{a Choose set with most empty slots:} \)
\[ 90 \] \( L10:: S \leftrightarrow \text{SETS} [1 \uparrow 1 \text{ARPS} [\text{SETS}] \uparrow \text{MIN}] \)
\[ 91 \] \( \text{a No. records to be inserted within the set:} \)
\[ 92 \] \( L11:: M \leftrightarrow T \downarrow \text{RPS} [S] \uparrow \text{ARPS} [S] \)
\[ 93 \] \( \text{a No. records to be catenated within the set:} \)
\[ 94 \] \( N \leftrightarrow (T - M) \downarrow \text{BLK} \downarrow \text{RPS} [S] \)
\[ 95 \] \( \text{a Displacement (no. components) before this set:} \)
\[ 96 \] \( \text{SDISP} \leftrightarrow \text{DISP} \uparrow \text{INCR} \downarrow 1 \)
\[ 97 \] \( \text{a Branch if no deletion field:} \)
\[ 98 \] \( \rightarrow (x \text{DEL}) \downarrow L12 \)
\[ 99 \] \( \text{a Read deletion field for set } S: \)
\[ 100 \] \( \text{BIT} \uparrow \text{UFREAD DCMP} \uparrow \text{TIE, DEL} \uparrow \text{DISP} \)
\[ 101 \] \( \text{a Branch if no records to be inserted:} \)
\[ 102 \] \( \rightarrow (x \text{M}) \downarrow L12 \)
\[ 103 \] \( \text{a Indices of available insertion slots:} \)
\[ 104 \] \( \text{INDS} \leftrightarrow M \downarrow (\text{BIT}) / 1 \uparrow \text{BIT} \)
\[ 105 \] \( \text{a Next field index:} \)
\[ 106 \] \( L12:: I \downarrow 1 \)
\[ 107 \] \( \text{a Loop by active field:} \)
\[ 108 \] \( L13:: (I > \text{NFLD}) \downarrow L18 \)
\[ 109 \] \( \text{a Field number:} \)
\[ 110 \] \( F \leftrightarrow \text{FLD}[I] \)
\[ 111 \] \( \text{a Field width (no. columns):} \)
\[ 112 \] \( \text{W} \leftrightarrow \text{WID}[F] \)
\[ 113 \] \( \text{a Look at field variable:} \)
\[ 114 \] \( \text{VAR} \leftrightarrow \#'F', \#F \)
\[\text{CATRECWS (continued)}\]

\[\text{\# R Reshape singletons, vectors and 1-row matrices}\]
\[\text{VAR\(\lt\)(NREC,\(W\gt1\))pVAR}\]
\[\text{\# to proper rank, shape:}\]
\[\text{VAR\(\lt\)(NREC,\(W\gt1\))pVAR}\]
\[\text{\# Read field F for set S:}\]
\[\text{DATA\(\lt\)OFREAD CMP\(\lt\)TIE,F\(\lt\)SDISP}\]
\[\text{\# Branch if a matrix field:}\]
\[\text{\(\rightarrow\)(\(W\gt1\))pL15}\]
\[\text{\# Branch if no records to insert:}\]
\[\text{\(\rightarrow\)(\(xM\))pL14}\]
\[\text{\# Branch if no records to catenate:}\]
\[\text{\(\rightarrow\)(\(xN\))pL17}\]
\[\text{\text{\(L14:\text{DATA}\rightarrow\text{DATA,VAR[GOOD[(FREC+M)+tN]]}\)\rightarrow L17}}\]
\[\text{\# Branch if no records to insert:}\]
\[\text{\(L15:\text{\(\rightarrow\)(\(xM\))\downarrow L16}\)\text{\(\rightarrow\)(\(xN\))\downarrow L17}\)\}
\[\text{\# Branch if no records to catenate:}\]
\[\text{\(\rightarrow\)(\(xN\))pL17}\]
\[\text{\text{\(L16:\text{DATA}\rightarrow\text{DATA,\{1\}VAR[GOOD[(FREC+M)+tN]]}\}\)}\]
\[\text{\text{\(L17:\text{DATA \(\langle\rangle\)REPLACE CMP}\)\}}\]
\[\text{\text{\(\rightarrow I+I+1\)\}}\]
\[\text{\(\rightarrow L13\)\}}\]
\[\text{\# Branch if no deletion field:}\]
\[\text{\(L18:\text{LEAD\rightarrow 1}\)\}}\]
\[\text{\# Branch if no records to insert:}\]
\[\text{\(L19:\text{\(\rightarrow\)(\(xM\))\downarrow L19}\)\}}\]
\[\text{\# Turn active record bits on:}\]
\[\text{\(\text{BIT[INDS]\rightarrow 1}\)\}}\]
\[\text{\# Branch if no records to catenate:}\]
\[\text{\(\rightarrow\)(\(xN\))pL20}\]
\[\text{\(\text{\(L19:\text{BIT\rightarrow\text{BIT,Np}}\)\downarrow L20}\)\}}\]
\[\text{\(\text{\(L20:\text{BIT \(\langle\rangle\)REPLACE DCMP}\)\}}\]
\[\text{\# Increment FREc by no. records added to this set:}\]
\[\text{\(L21:\text{FREC\rightarrow FREC+M+N}\)\}}\]
\[\text{\(\text{RPS[S]\rightarrow RPS[S]+N}\)\}}\]
\[\# Replace layer value if file layered and\]
\[\# set initially empty:}\]
\[\text{\(\rightarrow\)(\(T\times FP[1]\))\downarrow L23}\]
\[\# Branch if a vector layer field:}\]
\[\text{\(\rightarrow\text{VECpL22}\)\}}\]
\[\text{\(\text{LV[S];\rightarrow LAYER}\)\}}\]
\[\text{\(\rightarrow L23\)\}}\]
\[\text{\(\text{LV[S];\rightarrow LAYER}\)\}}\]
\[\text{\(\text{L22:ARPS[S]\rightarrow \{-1 1\}[(1+\text{LEAD})\times M+N]}\right)\downarrow \text{ARPS[S]}\]
\[\# Exit if all of data in field vars. filed:}\]
\[\text{\(\rightarrow\)(\(NR=FREC)\)\downarrow L33}\]
\[\text{\(\rightarrow L9\)\}}\]
Chapter 14 Solutions  
FILE DESIGN AND UTILITIES

\[ \text{CATRECWS (continued)} \]

[167] \( \text{R No. records to be appended in next set:} \)
[168] \( \text{L24: } N \leftarrow \text{BLK} \backslash N \text{-FREC} \)
[169] \( \text{R Next field number:} \)
[170] \( \text{F} \leftarrow 1 \)
[171] \( \text{R Loop by field:} \)
[172] \( \text{L25: } \rightarrow (F \text{-INC}) \rho L31 \)
[173] \( \text{W} \leftarrow \text{WID}[F] \)
[174] \( \text{R Branch unless a latent field:} \)
[175] \( \rightarrow (XW) \rho L26 \)
[176] \( \text{DATA} \leftarrow \text{L0} \)
[177] \( \rightarrow L30 \)
[178] \( \text{R Branch unless it's the deletion field:} \)
[179] \( \text{L26: } \rightarrow (\text{DEL} \# F) \rho L27 \)
[180] \( \text{DATA} \leftarrow N \rho 1 \)
[181] \( \rightarrow L29 \)
[182] \( \text{R Look at field variable:} \)
[183] \( \text{L27: } \text{VAR} \leftarrow \text{VAR}'F', \# F \)
[184] \( \text{R Reshape singletons, vectors and 1-row matrices} \)
[185] \( \text{to proper rank, shape:} \)
[186] \( \text{VAR} \leftarrow (NREC, (W \geq 1) \rho W) \rho \text{VAR} \)
[187] \( \text{R Branch if a matrix field:} \)
[188] \( \rightarrow (W \geq 1) \rho L28 \)
[189] \( \text{DATA} \leftarrow \text{VAR}[\text{GOOD}[FREC + tN]] \)
[190] \( \rightarrow L29 \)
[191] \( \text{L28: DATA} \leftarrow \text{VAR}[\text{GOOD}[FREC + tN]]; \)
[192] \( \text{R Branch unless set must be padded to BLK records:} \)
[193] \( \text{L29: } \rightarrow \text{FILL} \downarrow L30 \)
[194] \( \text{DATA} \leftarrow (\text{BLK}, 1 \downarrow \rho \text{DATA}) \downarrow \text{DATA} \)
[195] \( \text{L30: } T \downarrow \text{DATA} \leftarrow \text{FAPPEND TIE} \)
[196] \( \text{F} \leftarrow F + 1 \)
[197] \( \rightarrow L25 \)
[198] \( \text{R Increment FREC by no. records added to this set:} \)
[199] \( \text{L31: FREC} \leftarrow \text{FREC} + N \)
[200] \( \text{ARPS} \leftarrow \text{ARPS}, N \)
[201] \( \text{RPS} \leftarrow \text{RPS}, T \leftarrow N \backslash \text{BLK} \times \text{FILL} \)
[202] \( \text{FP}[7] \leftarrow \text{FP}[7] + T \)
[203] \( \text{FP}[6] \leftarrow \text{FP}[6] + 1 \)
[204] \( \text{FP}[9] \leftarrow \text{FP}[9] + 1 \)
[205] \( \text{R Catenate layer value if file layered:} \)
[206] \( \rightarrow (\times \text{FP}[11]) \downarrow L32 \)
[207] \( \text{LV} + \text{LV}, [1] \downarrow \text{LAYERN} \)
[208] \( \text{R Continue unless all of data in field vars. filed:} \)
[209] \( \text{L32: } \rightarrow (\text{NR} \# \text{FREC}) \rho L24 \)
[210] \( \text{R Branch unless layered:} \)
[211] \( \text{L33: } \rightarrow (\times \text{FP}[11]) \downarrow L34 \)
[212] \( \text{R Branch if no data left to file:} \)
[213] \( \text{FILED}[\text{GOOD}] \leftarrow 1 \)
[214] \( \rightarrow (\text{^ /FILED}) \downarrow L4 \)
[215] \( \text{LV} \downarrow \text{DFREPLACE TIE}, 9 \)
[216] \( \text{L34: RPS} \downarrow \text{DFREPLACE TIE}, 8 \)
[217] \( \rightarrow (\times \text{DEL}) \downarrow L35 \)
[218] \( \text{ARPS} \downarrow \text{DFREPLACE TIE}, 10 \)
[219] \( \text{L35: FP}[10] \leftarrow \text{FP}[10] + \text{NREC} \)
\[\text{CATRECWS (continued)}\]
\[\text{FP OREPLACE TIE,7}\]
\[\text{NFP}\uparrow\text{FP}\]
\[\text{\# Erase global field variables:}\]
\[\text{F}\uparrow\text{FLD,1}\uparrow\text{FLD}\]
\[\text{F' \# 'F', (+/' '=' F)\# F}\]
\[\text{\# DEX F}\]

\[\text{INDS}\uparrow\text{FP IOTA VALS;ARPS;BIT;DATA;DEL;DISP;FLD;FOUND;FT;}
\text{INCR;IND;LV;NDEL;NSET;NUM;RPS;SDISP;SET;SETS;SHAPE;T}
\text{TIE;VIND;WID;DIO}\]

\[\text{\# Searches through field FP[12] for VALS and}\]
\[\text{\# returns 2-row matrix of indices. First row}\]
\[\text{\# is set number (origin 1); second row is index}\]
\[\text{\# (origin 1) within set. Result contains -1s}\]
\[\text{\# where corresponding value is not found.}\]
\[\text{\# Shape of result is 2,\rho VALS. Requires CMIOOTA}\]
\[\text{\# function if a character matrix field.}\]
\[\text{\# DERROR(1\#\rho FP)\uparrow 'RANK ERROR'}\]
\[\text{\# DERROR(12\#\rho FP)\uparrow 'LENGTH ERROR'}\]
\[\text{\# DIO\uparrow 1}\]
\[\text{\# INCR\uparrow FP[3]}\]
\[\text{\# DEL\uparrow FP[11]}\]
\[\text{\# FLD\uparrow FP[12]}\]
\[\text{\# DERROR((FLD\#\uparrow INCR)\#FLD\#\uparrow DEL)\uparrow 'INVALID FIELD NUMBER'}\]
\[\text{\# TIE\uparrow FP[2]}\]
\[\text{\# FT\uparrow OFREAD TIE,4}\]
\[\text{\# Width (no. columns) of specified field:}\]
\[\text{\# WID\#FT[1;FLD]}\]
\[\text{\# DERROR(0=WID)\uparrow 'INACTIVE FIELD'}\]
\[\text{\# DERROR((WID\#1)\#(x\rho VALS)\#WID\#1\uparrow \rho VALS)\uparrow 'LENGTH ERROR'}\]
\[\text{\# Numeric field?}\]
\[\text{\# NUM=2\#FT[2;FLD]}\]
\[\text{\# DERROR(NUM\#0=1\#0\rho VALS)\uparrow 'DOMAIN ERROR'}\]
\[\text{\# Branch unless VALS is a scalar for a mat fld:}\]
\[\text{\# (WID=1)\#x\rho VALS)\#L1}\]
\[\text{\# Treat a scalar as a vector if a matrix field:}\]
\[\text{\# VALS\#WID\rho VALS}\]
\[\text{\# Determine shape of result; then 'ravel' VALS:}\]
\[\text{\# L1:SHAPE\#(-WID\#1)\#WID\rho VALS}\]
\[\text{\# VALS\#((x\#SHAPE),(WID\#1)\#WID)\rho VALS}\]
\[\text{\# Initialize 'raveled' result:}\]
\[\text{\# IND\#(2,\#x\#SHAPE)\#0}\]
\[\text{\# Exit if VALS is empty or if no records on file:}\]
\[\text{\# (0\#SHAPEx0=FP[10])\#L14}\]
\[\text{\# DISP\uparrow FP[4]}\]
\[\text{\# ARPS\#RPS\#OFREAD TIE,8}\]
\[\text{\# Branch unless ARPS should be read:}\]
\[\text{\# (/\#FP[7 10])\#L2}\]
\[\text{\# ARPS\#OFREAD TIE,10}\]
\[\n\text{\n\nChapter 14 Solutions\n\nFILE DESIGN AND UTILITIES\n\n\n\n\n\n\nIOTA (continued)\n\n[40] Consider only nonempty sets: \n[41] L2: SETS \( \not\emptyset \) \# ARPS
[42] \rightarrow \ldots \text{and sets specified in} \langle \text{layers} \rangle:\n[43] \rightarrow (x \text{FP}[1]) \rightarrow L6
[44] \rightarrow (x \text{DNC 'layers'}) \rightarrow L6
[45] Read layer values:
[46] LV \rightarrow \text{DFREAD} \ TIE,9
[47] Branch if matrix layers field:
[48] \rightarrow (2=\rho LV) \rightarrow L3
[49] SETS \rightarrow \text{SETSA} LV \text{ layers}
[50] \rightarrow L6
[51] \rightarrow \text{Convert} \langle \text{layers} \rangle \rightarrow \text{matrix if not already:}
[52] L3: \text{DERROR} (FT[1;FP[1]] \not= 1 \rho \text{players})/'LENGTH ERROR'
[53] \rightarrow (2=\rho \text{players}) \rightarrow L4
[54] layers \rightarrow ((x^{-1} \rho \text{players}),-1 \rho \text{players}) \rightarrow \text{players}
[55] Branch if numeric matrix field:
[56] L4: \rightarrow (0=1 \rho LV) \rightarrow L5
[57] SETS \rightarrow \text{SETSA} (\text{layers CMIOTA} LV) \leq 1 \rho \text{players}
[58] \rightarrow L6
[59] L5: SETS \rightarrow \text{V/LVA}.= \rho \text{layers}
[60] Convert to indices; erase \langle \text{layers} \rangle:
[61] L6: SETS \rightarrow \text{SETSA} / \rho \text{SETS}
[62] NSET \rightarrow \rho \text{SETS}
[63] LV \rightarrow \text{DEX 'layers'}
[64] Indices into INDS of values not yet found:
[65] VIND \rightarrow \rho \text{VALS}
[66] Last set index:
[67] S \leftarrow 0
[68] \rightarrow \text{Loop by nonempty set:}
[69] L7: \rightarrow (S \geq NSET) \rightarrow L4
[70] Current set index and number:
[71] S \leftarrow S+1
[72] SET \rightarrow \text{SETSA}[S]
[73] Displacement (no. components) before this set:
[74] SDISP \rightarrow \text{DISP} + \text{INCR} \times SET + -1
[75] Read field FLD for set SET:
[76] DATA \rightarrow \text{DFREAD} \ TIE,FLD+SDISP
[77] Branch if deletion field unneeded:
[78] \rightarrow (NDEL\text{ARPS}[SET]=\text{RPS}[SET]) \rightarrow L9
[79] Branch if deletion field needed:
[80] \rightarrow (NDEL\text{ARPS}[SET]>0) \rightarrow L8
[81] Active records are leading records:
[82] DATA \rightarrow (\text{ARPS}[SET],(\text{WID}>1) \rho \text{WID}) \rho \text{DATA}
[83] \rightarrow L9
[84] Read and apply deletion field for set SET:
[85] L8: \text{BIT} \rightarrow \text{DFREAD} \ TIE,DEL+SDISP
[86] DATA \rightarrow \text{BIT}/DATA
[87] Branch if a matrix field:
[88] L9: \rightarrow (\text{WID}>1) \rightarrow L0
[89] Search algorithm for vector field:
[90] IND \rightarrow \text{DATA} \rightarrow \text{VALS}
[91] \rightarrow L12\n\]
\[ \text{V IOTA (continued)} \]

\[ [92] \text{A Branch if a numeric matrix field:} \]
\[ [93] \text{L10: } \rightarrow \text{NUM} p \text{L11} \]
\[ [94] \text{A Search algorithm for character matrix field:} \]
\[ [95] \text{IND} \rightarrow \text{DATA CM} \text{IOTA VALS} \]
\[ [96] \text{L12} \]
\[ [97] \text{A Search algorithm for numeric matrix field:} \]
\[ [98] \text{L11: IND} \rightarrow 1++/^[^{}\text{\textbackslash VALS} noted\text{\textbackslash DATA} \]
\[ [99] \text{A Determine those values found in this set:} \]
\[ [100] \text{L12: FOUND} \rightarrow \text{IND} \rightarrow 1\text{\textbackslash DATA} \]
\[ [101] \text{A Continue to next set if none found:} \]
\[ [102] \text{IND} \rightarrow \text{FOUND/IND} \]
\[ [103] \rightarrow (\times p \text{IND}) \rightarrow \text{L7} \]
\[ [104] \text{A Consider deletion field if applicable:} \]
\[ [105] \rightarrow \text{NDEL} p \text{L13} \]
\[ [106] \text{IND} \rightarrow (\text{BIT} / 0\text{BIT}) [\text{IND}] \]
\[ [107] \text{A Insert indices in result:} \]
\[ [108] \text{L13: IND} \rightarrow \text{S0}; \text{FOUND/VIND} \rightarrow \text{SET}, [0.5] \text{IND} \]
\[ [109] \text{A Compress indices of remaining values:} \]
\[ [110] \text{BIT} \rightarrow \text{FOUND} \]
\[ [111] \text{VIND} \rightarrow \text{BIT/VIND} \]
\[ [112] \text{A Exit if all found:} \]
\[ [113] \rightarrow (\times p \text{VIND}) \rightarrow \text{L14} \]
\[ [114] \text{A Else, compress remaining values:} \]
\[ [115] \text{VALS} \rightarrow \text{BIT/VALS} \]
\[ [116] \rightarrow \text{L7} \]
\[ [117] \text{L14: IND} \rightarrow (2, \text{SHAPE}) p \text{INDS} \]

\[ \text{\textbackslash WSID: MULTIFLO} \]

\[ \text{\textbackslash IND} \rightarrow \text{IOTA \textbackslash HO FP; \textbackslash ARPS; \textbackslash BIT; \textbackslash CMPS; \textbackslash DEL; \textbackslash FT; I; LV; NSET; RPS; \textbackslash SETS; \textbackslash START; \textbackslash TIE; \textbackslash DIO} \]
\[ [1] \text{A Returns 2 row matrix of indices of all active records} \]
\[ [2] \text{on file. First row is set number (origin 1); second} \]
\[ [3] \text{row is index (origin 1) within set.} \]
\[ [4] \text{\textbackslash DERROR(1\# \textbackslash FP})/^{'}\text{RANK ERROR'} \]
\[ [5] \text{\textbackslash DERROR(11\# \textbackslash FP})/^{'}\text{LENGTH ERROR'} \]
\[ [6] \text{\textbackslash DIO} \rightarrow 1 \]
\[ [7] \text{A Branch if some records of file:} \]
\[ [8] \rightarrow (\times \text{FP[101]}) p \text{L1} \]
\[ [9] \text{INDS} \rightarrow 2 \ 0 \ p0 \]
\[ [10] \rightarrow \text{L8} \]
\[ [11] \text{L8: TIE} \rightarrow \text{FP[2]} \]
\[ [12] \text{ARPS} \rightarrow \text{RPS} \rightarrow \text{\textbackslash DFREAD TIE, 8} \]
\[ [13] \text{A Branch unless ARPS should be read:} \]
\[ [14] \rightarrow (=/ \text{FP[7 101]}) p \text{L2} \]
\[ [15] \text{ARPS} \rightarrow \text{\textbackslash DFREAD TIE, 10} \]
\[ [16] \text{A Consider only nonempty sets:} \]
\[ [17] \text{L2: SETS} \rightarrow 0 \# \text{ARPS} \]
\[ [18] \text{A ... and sets specified in <layers>:} \]
\[ [19] \rightarrow (\times \text{FP[11]} \rightarrow \text{L6} \]
\[ [20] \rightarrow (\times \text{\textbackslash ONC 'layers'}) \rightarrow \text{L6} \]

\[ -403- \]
\[\textbf{IOTARHO (continued)}\]

[21] \textbf{Read layer values and fld types:}

\[\text{L}V \leftarrow \text{DFREAD TIE,9}\]

[22] \textbf{FT \leftarrow \text{DFREAD TIE,4}}

[23] \textbf{Branch if matrix layers field:}

\[\rightarrow (2=\rho LV)pL3\]

[24] \[\text{SETS} \leftarrow \text{SETS} \land LV \in \text{layers}\]

[25] \rightarrow L6

[26] \textbf{Convert <layers> to matrix if not already:}

\[L3:U \text{ERROR}(\text{FT}[1;FP[1]] \neq 1 \rho \text{layers})/\text{'LENGTH ERROR'}\]

[27] \[\rightarrow (2=\rho \text{layers})pL4\]

[28] \[\text{layers} \leftarrow ((x/-1 \rho \text{layers}), -1 \rho \text{layers}) \rho \text{layers}\]

[29] \textbf{Branch if numeric matrix field:}

\[L4: \rightarrow (0=1 \rho LV)pL5\]

[30] \[\text{SETS} \leftarrow \text{SETS} \land (\text{CM} \text{IOTA LV}) \leq 1 \rho \text{layers}\]

[31] \rightarrow L6

[32] \textbf{Convert to indices; erase <layers>:}

\[L5: \text{SETS} \leftarrow \text{SETS} \land \text{LV} / \text{LV} = \rho \text{layers}\]

[33] \[\rightarrow L6\]

[34] \textbf{Branch if numeric matrix field:}

\[L4: \rightarrow (0=1 \rho LV)pL5\]

[35] \[\text{SETS} \leftarrow \text{SETS} \land (\text{CM} \text{IOTA LV}) \leq 1 \rho \text{layers}\]

[36] \rightarrow L6

[37] \textbf{Convert to indices; erase <layers>:}

\[L6: \text{SETS} \leftarrow \text{SETS} / \rho \text{SETS}\]

[38] \[\text{LV} \leftarrow \text{OEX 'layers'}\]

[39] \[\rightarrow (\text{ARPS[SETS]} \land \rho \text{SETS})\]

[40] \textbf{Construct 2 rows: set ind, rec ind:}

\[\text{INDS} \leftarrow !\text{ARPS[SETS]}\]

[41] \[I \leftarrow !\text{MONIOTA IND}\]

[42] \[\rightarrow I + \rho I \leftarrow \text{INDS} / \text{INDS} / -\text{INDS}\]

[43] \[\text{INDS} \leftarrow (\text{INDS} / \text{SETS}), [0.5] I\]

[44] \textbf{Exit if no deleted records:}

\[\rightarrow (\text{ARPS[SETS]} \land = \rho \text{SETS})pL8\]

[45] \textbf{Deletion field number:}

\[\text{DEL} \leftarrow \text{IP[11]}\]

[46] \textbf{Set no.s for which deletion bits are to be read:}

\[\text{SETS} \leftarrow (\text{ARPS[SETS]} < 0) / \text{SETS}\]

[47] \[\text{NSET} \leftarrow \rho \text{SETS}\]

[48] \textbf{Exit if none:}

\[\rightarrow (= \text{NSET})pL8\]

[49] \textbf{Component numbers of deletion bit fields:}

\[\text{CMPS} \leftarrow (\text{DEL} + \text{FP[4]}) + \text{FP[3]xSETS} - 1\]

[50] \textbf{Starts of each selected set's indices in result:}

\[\text{ARPS} \leftarrow !\text{ARPS}\]

[51] \[\text{START} \leftarrow (0, + / \text{ARPS[SETS]}\]

[52] \textbf{Compress ARPS to selected sets:}

\[\text{ARPS} \leftarrow \text{ARPS[SETS]}\]

[53] \[\rightarrow \text{INDEX} '\text{layers'}\]

[54] \[\rightarrow I \leftarrow 1\]

[55] \textbf{Loop by set; read deletion bits:}

\[\text{L7:BIT} \leftarrow \text{OREAD TIE,CMP[1]}\]

[56] \textbf{Insert correct indices in result:}

\[\text{INDS}[2; \text{START}[I] + !\text{ARPS[SETS]}[1]] / \rho \text{BIT}\]

[57] \[\rightarrow \text{INDEX} '\text{layers'}\]

[58] \[\rightarrow (I \leq \text{NSET})pL7\]

[59] \[\text{L8:INDEX} '\text{layers'}\]
Chapter 14 Solutions

FILE DESIGN AND UTILITIES

[WSID: MULTIFLOJ

Diags

\[ \Delta \text{INDS} \leftarrow \Delta \text{EXPR} \ \text{SLASH IOTARHO} \ \Delta \text{FP} ; \ \Delta \text{ACTIVE} ; \ \Delta \text{ALL} ; \ \Delta \text{ARPS} ; \ \\
\] \[ \Delta \text{BIT} ; \ \Delta \text{DATA} ; \ \Delta \text{DEL} ; \ \Delta \text{DISP} ; \ \Delta \text{FLD} ; \ \Delta \text{FLDS} ; \ \Delta \text{FNAM} ; \ \Delta \text{FT} ; \ \\
\] \[ \Delta \text{INCR} ; \ \Delta \text{IND} ; \ \Delta \text{LV} ; \ \Delta \text{NDEL} ; \ \Delta \text{NFLD} ; \ \Delta \text{NSET} ; \ \Delta \text{ARPS} ; \ \Delta \text{AS} ; \ \\
\] \[ \Delta \text{SDISP} ; \ \Delta \text{SEL} ; \ \Delta \text{SET} ; \ \Delta \text{SETS} ; \ \Delta \text{TIE} ; \ \Delta \text{WID} ; \ \Delta \text{UIO} \]

1. Loops through active sets doing the following:
2. Reads the fields specified in \( \text{FP} \) (calling them \( \text{F5, F9, etc.} \), executes the
3. Character vector \( \text{EXPR} \), converts the resulting bit
4. Vector to indices and returns the indices of all
5. Records found for which \( \text{EXPR} \) returns a 1. Result
6. is a 2 row matrix: first row is set number (origin
7. 1); second row is index (origin 1) within set. If
8. \( \text{EXPR} \) and \( \text{FP} \) are empty, all records are selected.
9. Note that \( \text{EXPR} \) is executed in origin 1; e.g.
10. 'F3[;2]' always refers to 2nd column.
11. \( \text{ERROR}( (1 \neq \rho \Delta \text{FP}) \lor (\rho \Delta \text{FP}) \neq 'RANK ERROR' \)
12. \( \text{ERROR}(0 = \text{INCR}' \Delta \text{FP}) \neq 'DOMAIN ERROR' \)
13. \( \text{IO} = 1 \)
14. \( \Delta \text{ALL} = 0 \neq \Delta \text{NFLD} \lor \rho \Delta \text{FLDS} \neq \text{FP} \)
15. \( \text{ERROR}( (\Delta \text{ALL} = \Delta \text{EXPRV} \neq ') \lor (\text{FP} \neq \rho \Delta \text{FP}) \neq 'LENGTH ERROR' \)
16. \( \Delta \text{INCR} \neq \Delta \text{FP}[3] \)
17. \( \Delta \text{DEL} \neq \Delta \text{FP}[11] \)
18. \( \text{ERROR}( (\rho \Delta \text{FLDS} \neq \text{FP}) \neq 'INVALID FIELD NUMBER' \)
19. \( \Delta \text{TIE} \neq \Delta \text{FP}[2] \)
20. \( \Delta \text{FT} \neq \text{FREAD} \Delta \text{TIE}, 4 \)
21. \( \text{Width (no. columns) of specified fields:} \)
22. \( \Delta \text{WID} \neq \Delta \text{FT}[1; \Delta \text{FLDS}] \)
23. \( \text{ERROR}(0 = \Delta \text{WID}) \neq 'INACTIVE FIELD' \)
24. \( \text{Initialize result as empty:} \)
25. \( \Delta \text{INDS} \leftarrow 2 \ 0 \ 0 \)
26. \( \text{Exit if no records on file:} \)
27. \( \rightarrow (\Delta \Delta \text{FP}[10]) \downarrow 0 \)
28. \( \text{Field names (e.g. 'F5 F9') to be erased below:} \)
29. \( \Delta \text{FNAM} \neq (\Delta \text{NFLD}, 1) \rho \Delta \text{FLDS} \)
30. \( \Delta \text{FNAM} \neq 'F', (+/ ' = \Delta \text{FNAM}) \Phi \Delta \text{FNAM} \)
31. \( \Delta \text{DISP} \neq \Delta \text{FP}[4] \)
32. \( \Delta \text{ARPS} \neq \Delta \text{FPS} \neq \text{FREAD} \Delta \text{TIE}, 8 \)
33. \( \text{Branch unless ARPS should be read:} \)
34. \( \rightarrow (\Delta \Delta \text{FP}[7, 10]) \rho \Delta \text{L1} \)
35. \( \Delta \text{ARPS} \neq \text{FREAD} \Delta \text{TIE}, 10 \)
36. \( \text{Consider only nonempty sets:} \)
37. \( \Delta \text{L1} \neq \Delta \text{SETS} \neq 0 \neq \Delta \text{ARPS} \)
38. \( \text{... and sets specified in <layers>:} \)
39. \( \rightarrow (\Delta \Delta \text{FP}[1]) \downarrow \Delta \text{L5} \)
40. \( \rightarrow (\Delta \Delta \text{FLD}) \downarrow \Delta \text{L5} \)
41. \( \rightarrow (\Delta \text{FNC 'layers'}) \downarrow \Delta \text{L5} \)
42. \( \text{Read layer values:} \)
43. \( \Delta \text{LV} \neq \text{FREAD} \Delta \text{TIE}, 9 \)
44. \( \text{Branch if matrix layers field:} \)
45. \( \rightarrow (2 \neq \rho \Delta \text{LV}) \rho \Delta \text{L2} \)
46. \( \Delta \text{SETS} \neq \Delta \text{SETS} \Delta \text{LV} = \text{layers} \)
47. \( \rightarrow \Delta \Delta \text{L5} \)
\[\text{\textbackslash n SLASHIOTARHO (continued)}\]

[48] a Convert \texttt{<layers>} to matrix if not already:

[49] \[\Delta \Delta L_2: \text{\texttt{ERROR(}\Delta \Delta F T[1]; \Delta \Delta P F[1]}\] \[\text{\textbackslash n} \pm 1 \text{\texttt{\rho layers}} / \text{\texttt{LENGTH ERROR}}\]

[50] \[\rightarrow (2=\rho \text{\texttt{layers}}) \rho \Delta \Delta L_3\]

[51] \[\text{\texttt{layers}} \rightarrow (x / 1 \text{\texttt{\rho layers}}, -1 \text{\texttt{\rho layers}}) \rho \text{\texttt{layers}}\]

[52] a Branch if numeric matrix field:

[53] \[\Delta \Delta L_3: \rightarrow (0=1 \text{\texttt{\rho}} \Delta \Delta \text{\texttt{LV}}) \rho \Delta \Delta L_4\]

[54] \[\Delta \Delta \text{\texttt{SETS}} \leftarrow \Delta \Delta \text{\texttt{SETS}} \cap (\text{\texttt{layers}} \cap \text{\texttt{MIOTA}} \cap \text{\texttt{LLEN}}) \leq 1 \text{\texttt{\rho layers}}\]

[55] \[\rightarrow \Delta \Delta L_5\]

[56] \[\Delta \Delta L_4: \Delta \Delta \text{\texttt{SETS}} \leftarrow \Delta \Delta \text{\texttt{SETS}} \cap \sqrt{\Delta \Delta \text{\texttt{LV}}} \cap \Delta \Delta \text{\texttt{SETS}} \cap \Delta \Delta \text{\texttt{SETS}}\]

[57] a Convert to indices; erase \texttt{<layers>}:

[58] \[\Delta \Delta L_5: \Delta \Delta \text{\texttt{SETS}} \leftarrow \Delta \text{\texttt{SETS}} \cap \rho \Delta \Delta \text{\texttt{SETS}}\]

[59] \[\Delta \Delta \text{\texttt{SET}} \leftarrow \rho \Delta \Delta \text{\texttt{SETS}}\]

[60] \[\Delta \Delta \text{\texttt{LV}} \rightarrow \text{\texttt{UEX}} \text{\texttt{\textbackslash n} \texttt{\textbackslash n}} \text{\texttt{layers}}\]

[61] a Last set index:

[62] \[\Delta \Delta \text{\texttt{S}} \leftarrow 0\]

[63] a Loop by nonempty set:

[64] \[\Delta \Delta L_6: \rightarrow (\Delta \Delta \text{\texttt{S}} \leftarrow \Delta \Delta \text{\texttt{SET}}) \rho 0\]

[65] a Current set index and number:

[66] \[\Delta \Delta \text{\texttt{S}} \leftarrow \Delta \Delta \text{\texttt{S}} + 1\]

[67] \[\Delta \Delta \text{\texttt{SETS}} \leftarrow \Delta \Delta \text{\texttt{SETS}}[\Delta \Delta \text{\texttt{S}}]\]

[68] a Displacement (no. components) before this set:

[69] \[\Delta \Delta \text{\texttt{DISP}} \leftarrow \Delta \Delta \text{\texttt{DISP}} + \Delta \Delta \text{\texttt{INCR}} \times \Delta \Delta \text{\texttt{SET}} + -1\]

[70] a Branch if deletion field unneeded:

[71] \[\rightarrow (\Delta \Delta \text{\texttt{NDEL}} \leftarrow \Delta \Delta \text{\texttt{ARPS}}[\Delta \Delta \text{\texttt{SET}}] > 0) \rho \Delta \Delta L_7\]

[72] a Read deletion field for set SET:

[73] \[\Delta \Delta \text{\texttt{BIT}} \leftarrow \text{\texttt{IFREAD}} \Delta \Delta \text{\texttt{TIE}}, \Delta \Delta \text{\texttt{DEL}} + \Delta \Delta \text{\texttt{DISP}}\]

[74] a Branch if no selection expression:

[75] \[\Delta \Delta L_7: \Delta \Delta \text{\texttt{SEL}} \leftarrow 1\]

[76] \[\rightarrow \Delta \Delta \text{\texttt{ALL}} \rho \Delta \Delta L_12\]

[77] a Are all records in this set active?

[78] \[\Delta \Delta \text{\texttt{ACTIVE}} \leftarrow \Delta \Delta \text{\texttt{ARPS}}[\Delta \Delta \text{\texttt{SET}}] = \Delta \Delta \text{\texttt{RPS}}[\Delta \Delta \text{\texttt{SET}}]\]

[79] a Last field index:

[80] \[\Delta \Delta \text{\texttt{F}} \leftarrow 0\]

[81] a Loop by field specified:

[82] \[\Delta \Delta L_8: \rightarrow (\Delta \Delta \text{\texttt{F}} = \Delta \Delta \text{\texttt{NFLD}}) \rho \Delta \Delta L_11\]

[83] a Current field index and number:

[84] \[\Delta \Delta \text{\texttt{F}} \leftarrow \Delta \Delta \text{\texttt{F}} + 1\]

[85] \[\Delta \Delta \text{\texttt{FLD}} \leftarrow \Delta \Delta \text{\texttt{FLDS}}[\Delta \Delta \text{\texttt{F}}]\]

[86] a Read field \texttt{FLD} for set SET:

[87] \[\Delta \Delta \text{\texttt{DATA}} \leftarrow \text{\texttt{IFREAD}} \Delta \Delta \text{\texttt{TIE}}, \Delta \Delta \text{\texttt{FLD}} + \Delta \Delta \text{\texttt{DISP}}\]

[88] a Branch if all records in this set active:

[89] \[\rightarrow \Delta \Delta \text{\texttt{ACTIVE}} \rho \Delta \Delta L_10\]

[90] a Branch if deletion field needed:

[91] \[\rightarrow \Delta \Delta \text{\texttt{NDEL}} \rho \Delta \Delta L_9\]

[92] a Active records are leading records:

[93] \[\Delta \Delta \text{\texttt{DATA}} \leftarrow (\Delta \Delta \text{\texttt{ARPS}}[\Delta \Delta \text{\texttt{SET}}], (\Delta \Delta \text{\texttt{WID}}[\Delta \Delta \text{\texttt{F}}] > 1) \rho \Delta \Delta \text{\texttt{WID}}[\Delta \Delta \text{\texttt{F}}]) \rho \]

[94] \[\Delta \Delta \text{\texttt{DATA}}\]

[95] \[\rightarrow \Delta \Delta L_10\]

[96] a Apply deletion field:

[97] \[\Delta \Delta L_9: \Delta \Delta \text{\texttt{DATA}} \leftarrow \Delta \Delta \text{\texttt{BIT}} \times \Delta \Delta \text{\texttt{DATA}}\]

[98] a Assign data to global variable \texttt{Fn}:

[99] \[\Delta \Delta L_{10}: \text{\texttt{F'}}, (\Delta \Delta \text{\texttt{FLD}}), \leftarrow \Delta \Delta \text{\texttt{DATA}}\]

[100] \[\rightarrow \Delta \Delta L_8\]
Once all fields have been read, execute EXPR:
\[ \Delta L1: \Delta SEL + \# \Delta EXPR \]
Erase field variables (e.g. F5, F9,...):
\[ \Delta IND + \Delta FNAM \]
Continue to next set if none found:
\[ \Delta L2: \Delta IND + \Delta SEL / \Delta ARPS[\Delta SET] \]
\[ \to (\times \rho \Delta IND) \Delta L6 \]
Consider deletion field if applicable:
\[ \Delta DIND = (\Delta BIT / \rho \Delta BIT)[\Delta IND] \]
Catenate indices to result:
\[ \Delta L3: \Delta INDS + \Delta INDS, \Delta SET, [0.5] \Delta IND \]
\[ \to \Delta L6 \]

[WSID: MULTIFLO]
\[ \text{NFP = FP DELREC INDS; BIT; CMP; DEL; DISP; F; FLD; FLDS; INCR; IND; NEW; NFLD; NSET; OLD; RPS; S; SDISP; SET; SETS; TIE; UNQ; \#IO} \]
Deletes records identified by file indices matrix
\[ \text{IND. First row is set number (origin 1); second row is index (origin 1) within set. INDS may be} \]
of any dimension as long as its first coordinate is 2.
\[ \text{DERROR} = (1 \neq \rho \text{FP}) / 'RANK ERROR' \]
\[ \text{DERROR} = (11 \neq \rho \text{FP}) \lor 2 \neq \rho \text{INDS} / 'LENGTH ERROR' \]
Exit if nothing to delete:
\[ \text{NFP = FP} \]
\[ \to (0 \neq \rho \text{INDS}) \rho 0 \]
\[ \text{SETS} = 1 \neq \text{INDS} \]
\[ \text{INDS} = 0 \neq \text{INDS} \]
\[ \#IO = 1 \]
\[ \text{TIE = FP[2]} \]
\[ \text{INCR = FP[3]} \]
\[ \text{DISP = FP[4]} \]
\[ \text{DEL = FP[11]} \]
\[ \text{RPS will be changed if DEL = 0; ARPS will be changed} \]
\[ \text{if DEL > 0. Read and replace only one or the other:} \]
\[ \text{RPS = \#OFREAD TIE, 8 + 2 \times \#DEL} \]
\[ \text{Determine active fields if no deletion field:} \]
\[ \to (\times \text{DEL}) \rho L1 \]
\[ \text{NFLD = \#FLDS} = (\times \#\text{OFREAD TIE, 4})[1;] / \\text{INCR} \]
\[ \text{Determine distinct set numbers (deleting \#1s):} \]
\[ \text{L1: UNQ = SETS[\#SETS]} \]
\[ \text{NSET = \#UNQ = \#UNQ \#1 \#1, UNQ) / UNQ} \]
\[ \text{Last set index:} \]
\[ S = 0 \]
\[ \text{Loop by distinct set:} \]
\[ L2: \to (S = \text{NSET}) \rho L5 \]
\[ \text{Current set index and number:} \]
\[ S = S + 1 \]
\[ \text{SET = UNQ}[S] \]
Chapter 14 Solutions  FILE DESIGN AND UTILITIES

\[\text{\textbackslash v} \text{DELREC (continued)}\]

[34] \text{a Displacement (no. components) before this set:}
[35] \text{SDISP} \leftarrow \text{DISP} + \text{INCR} \times \text{SET} + 1
[36] \text{a Indices to delete in this set:}
[37] \text{IND} \leftarrow \text{(SET} \equiv \text{SETS})/\text{INDS}
[38] \text{a Branch unless deletion field exists:}
[39] \rightarrow (\times \text{DEL}) \downarrow \text{L3}
[40] \text{a Read deletion field for set SET:}
[41] \text{BIT} \leftarrow \text{DFREAD CMP} \equiv \text{TIE}, \text{DEL} + \text{SDISP}
[42] \text{a Turn off specified indices and replace:}
[43] \text{BIT}[\text{IND}] \leftarrow 0
[44] \text{BIT} \leftarrow \text{DFREPLACE CMP}
[45] \text{a Compute new no. records:}
[46] \text{NEW} \leftarrow +/\text{BIT}
[47] \text{OLD} \leftarrow \text{RPS}[\text{SET}]
[48] \text{a Reset parameters:}
[49] \text{NFP}[9] \leftarrow \text{NFP}[9] - \text{NEW} = 0
[50] \text{NFP}[10] \leftarrow \text{NFP}[10] + \text{NEW} - \text{OLD}
[51] \text{RPS}[\text{SET}] \leftarrow \text{NEW} \times (-1) \times (1 + \text{NEW} = +/\wedge \text{BIT})
[52] \rightarrow \text{L2}
[53] \text{a}
[54] \text{a Turn off specified indices:}
[55] \text{L3: OLD} \leftarrow \text{RPS}[\text{SET}]
[56] \text{BIT} \leftarrow \text{OLD} \uparrow 1
[57] \text{BIT}[\text{IND}] \leftarrow 0
[58] \text{a Compute new no. records:}
[59] \text{NEW} \leftarrow +/\text{BIT}
[60] \text{a Reset parameters:}
[61] \text{NFP}[9] \leftarrow \text{NFP}[9] - \text{NEW} = 0
[62] \text{NFP}[7] \leftarrow \text{NFP}[10] + \text{NFP}[7] + \text{NEW} - \text{OLD}
[63] \text{RPS}[\text{SET}] \leftarrow \text{NEW}
[64] \text{a Last field index:}
[65] \text{F} \leftarrow 0
[66] \text{a Loop by active field:}
[67] \text{L4: (F} \equiv \text{NFLD}) \downarrow \text{L2}
[68] \text{a Current field index and number:}
[69] \text{F} \leftarrow \text{F} + 1
[70] \text{FLD} \leftarrow \text{FLDS}[\text{F}]
[71] \text{a Read, compress, replace field FLD for set SET:}
[72] \text{CMP} \equiv \text{TIE}, \text{FLD} + \text{SDISP}
[73] \text{(BIT} \equiv \text{DFREAD CMP}) \equiv \text{DFREPLACE CMP}
[74] \rightarrow \text{L4}
[75] \text{a File either RPS or ARPS:}
[76] \text{L5: RPS} \equiv \text{DFREPLACE TIE}, 8 + 2 \times \text{DEL}
[77] \text{NFP} \equiv \text{DFREPLACE TIE}, 7
\text{\textbackslash v}
Loops through active sets doing the following:

1. Reads the fields specified in the fields (calling them F5, F9, etc. for fields 5, 9, etc.),
2. EXEC executes the character vector EXPR, and deletes the records of that set which correspond to 0s
3. in the resulting bit vector. If EXPR and the fields are empty, all records are selected (none are deleted). Note that EXPR is executed in origin
4. Loops through active sets doing the following:
5. reads the fields specified in the fields (calling them F5, F9, etc. for fields 5, 9, etc.),
6. EXEC executes the character vector EXPR, and deletes the records of that set which correspond to 0s
7. in the resulting bit vector. If EXPR and the fields are empty, all records are selected (none are deleted). Note that EXPR is executed in origin
\[ V \text{ COMPRESS (continued)} \]

\[ \to \Delta \Delta L6 \]

49.7 \(\text{a} \) Convert <layers> to matrix if not already:

\[ \to \Delta \Delta L3 : \text{ERROR}(\Delta \Delta F(1; \Delta \Delta F(\text{FLP}[1]) \neq -1 \to \text{players}) / 'LENGTH ERROR' \]

\[ \to (2 = \text{players}) \to \Delta \Delta L4 \]

50.7 \(\text{a} \) Branch if numeric matrix field:

\[ \to \Delta \Delta L4 : \to (0 = 1 \to \text{players}) \to \Delta \Delta L5 \]

51.7 \(\Delta \Delta \text{SETS} \to \Delta \Delta \text{SETS} \wedge (\text{layers} \wedge \text{CMIOTA} \wedge \Delta \Delta L5 \leq 1 \to \text{players} \to \Delta \Delta L6 \]

52.7 \(\Delta \Delta L5 : \Delta \Delta \text{SETS} \wedge \text{layers} / \Delta \Delta L5 . = \text{layers} \to \Delta \Delta L6 \]

53.7 \(\text{a} \) Convert to indices; erase <layers>:

\[ \to \Delta \Delta L6 : \Delta \Delta \text{SETS} \wedge \Delta \Delta \text{SETS} / \to \text{players} \]

54.7 \(\Delta \Delta \text{NSET} \to \Delta \Delta \text{SETS} \]

55.7 \(\Delta \Delta LV + \text{OEX} '\text{layers}' \]

56.7 \(\text{a} \) Last set index:

\[ \to \Delta \Delta S = 0 \]

57.7 \(\text{a} \) Loop by nonempty set:

\[ \to (\Delta \Delta S \geq \Delta \Delta \text{NSET}) \to \Delta \Delta L7 \]

58.7 \(\Delta \Delta L7 : \to (\Delta \Delta S \geq \Delta \Delta \text{NSET}) \to \Delta \Delta L8 \]

59.7 \(\text{a} \) Current set index and number:

\[ \to \Delta \Delta S \to \Delta \Delta S + 1 \]

60.7 \(\Delta \Delta \text{SET} \to \Delta \Delta \text{SETS}[\Delta \Delta S] \]

61.7 \(\text{a} \) Displacement (no. components) before this set:

\[ \to \Delta \Delta \text{SDISP} \to \Delta \Delta \text{DISP} + \Delta \Delta \text{INCR} \times \Delta \Delta \text{SET} = -1 \]

62.7 \(\text{a} \) Branch if deletion field unneeded:

\[ \to (\Delta \Delta \text{DEL} \leq \Delta \Delta \text{ARPS}[\Delta \Delta \text{SET}] > 0) \to \Delta \Delta L8 \]

63.7 \(\text{a} \) Read deletion field for set SET:

\[ \to \Delta \Delta \text{BIT} \to \text{DFREAD} \Delta \Delta \text{TIE}, \Delta \Delta \text{DEL} \to \Delta \Delta \text{SDISP} \]

64.7 \(\text{a} \) Are all records in this set active:

\[ \to \Delta \Delta L8 : \Delta \Delta \text{ALL} \leq \Delta \Delta \text{ARPS}[\Delta \Delta \text{SET}] = \Delta \Delta \text{RPS}[\Delta \Delta \text{SET}] \]

65.7 \(\text{a} \) Last field index:

\[ \to \Delta \Delta F = 0 \]

66.7 \(\text{a} \) Loop by field specified:

\[ \to (\Delta \Delta F \geq \Delta \Delta \text{NFLD}) \to \Delta \Delta L9 \]

67.7 \(\text{a} \) Current field index and number:

\[ \to \Delta \Delta F \to \Delta \Delta F + 1 \]

68.7 \(\Delta \Delta \text{FLD} \to \Delta \Delta \text{FLDS}[\Delta \Delta F] \]

69.7 \(\text{a} \) Read field FLD for set SET:

\[ \to \Delta \Delta \text{DATA} \to \text{DFREAD} \Delta \Delta \text{TIE}, \Delta \Delta \text{FLD} \to \Delta \Delta \text{SDISP} \]

70.7 \(\text{a} \) Branch if all records in this set active:

\[ \to \Delta \Delta \text{ALL} \to \Delta \Delta \text{L11} \]

71.7 \(\text{a} \) Branch if deletion field needed:

\[ \to (\Delta \Delta \text{DEL} \leq \Delta \Delta L10 \]

72.7 \(\text{a} \) Active records are leading records:

\[ \to \Delta \Delta \text{DATA} \to (\Delta \Delta \text{ARPS}[\Delta \Delta \text{SET}] \wedge (\Delta \Delta \text{WID}[\Delta \Delta \text{FLD}] > 1) \wedge \Delta \Delta \text{WID}[\Delta \Delta \text{FLD}] ) \to \Delta \Delta \text{DATA} \]

73.7 \(\to \Delta \Delta L11 \]

74.7 \(\text{a} \) Apply deletion field:

\[ \to \Delta \Delta L10 : \Delta \Delta \text{DATA} \to \Delta \Delta \text{BIT} \times \Delta \Delta \text{DATA} \]

75.7 \(\text{a} \) Assign data to global variable Fn:

\[ \to \Delta \Delta L11 : 'F', (\Delta \Delta \text{FLD}), ' + \Delta \Delta \text{DATA}' \]

76.7 \(\to \Delta \Delta L9 \]

77.7 \(\text{a} \) Once all fields have been read, execute EXPR:

\[ \to \Delta \Delta L12 : \Delta \Delta \text{SEL} \to \Delta \Delta \text{EXPR} \]

-410-
COMPRESS (continued)

[101] a Erase field variables (e.g. F5, F9, ...):
[102] \( \Delta \Delta \text{OLD}+\Delta \text{EX} \Delta \Delta \text{FNAM} \)
[103] a Continue to next set if all records selected:
[104] \( \rightarrow (\wedge /\Delta \Delta \text{SEL}) \rho \Delta \Delta \text{L7} \)
[105] a Branch unless deletion field exists:
[106] \( \rightarrow (\times \Delta \Delta \text{DEL}) \downarrow \Delta \Delta \text{L15} \)
[107] a Branch if deletion field not read in:
[108] \( \rightarrow \Delta \Delta \text{NDEL} \rho \Delta \Delta \text{L13} \)
[109] a Reset BIT:
[110] \( \Delta \Delta \text{BIT} \leftarrow \Delta \Delta \text{BIT} \backslash \Delta \Delta \text{SEL} \)
[111] \( \rightarrow \Delta \Delta \text{L14} \)
[112] \( \Delta \Delta \text{L13}: \Delta \Delta \text{BIT} \leftarrow \Delta \Delta \text{RPS}[\Delta \Delta \text{SET}] \uparrow \Delta \Delta \text{SEL} \)
[113] a Replace deletion field:
[114] \( \Delta \Delta \text{L14}: \Delta \Delta \text{BIT} \uparrow \Delta \Delta \text{EX} \Delta \Delta \text{TIE}, \Delta \Delta \text{DEL} + \Delta \Delta \text{SDISP} \)
[115] a Compute new no. records:
[116] \( \Delta \Delta \text{NEW} \leftarrow \wedge /\Delta \Delta \text{BIT} \)
[117] \( \Delta \Delta \text{OLD} \leftarrow \wedge \Delta \Delta \text{RPS}[\Delta \Delta \text{SET}] \)
[118] a Reset parameters:
[119] \( \Delta \Delta \text{NFP}[9] + \Delta \Delta \text{NFP}[9] - \Delta \Delta \text{NEW} = 0 \)
[120] \( \Delta \Delta \text{NFP}[10] \leftarrow \Delta \Delta \text{NFP}[10] + \Delta \Delta \text{NEW} - \Delta \Delta \text{OLD} \)
[121] \( \Delta \Delta \text{RPS}[\Delta \Delta \text{SET}] \leftarrow \Delta \Delta \text{NEW} \times (\sim 1) [1+\Delta \Delta \text{NEW} = \wedge /\wedge \Delta \Delta \text{BIT}] \)
[122] \( \rightarrow \Delta \Delta \text{L7} \)
[123] a Compute new no. records:
[124] \( \Delta \Delta \text{L15}: \Delta \Delta \text{NEW} \leftarrow \wedge /\Delta \Delta \text{SEL} \)
[125] \( \Delta \Delta \text{OLD} \leftarrow \wedge \Delta \Delta \text{RPS}[\Delta \Delta \text{SET}] \)
[126] a Reset parameters:
[127] \( \Delta \Delta \text{NFP}[9] + \Delta \Delta \text{NFP}[9] - \Delta \Delta \text{NEW} = 0 \)
[128] \( \Delta \Delta \text{NFP}[10] \leftarrow \Delta \Delta \text{NFP}[10] + \Delta \Delta \text{NEW} - \Delta \Delta \text{OLD} \)
[129] \( \Delta \Delta \text{RPS}[\Delta \Delta \text{SET}] \leftarrow \Delta \Delta \text{NEW} \)
[130] a Last field index:
[131] \( \Delta \Delta \text{F} \leftarrow 0 \)
[132] a Loop by active field:
[133] \( \Delta \Delta \text{L16}: \rightarrow (\Delta \Delta \text{F} \geq \Delta \Delta \text{NAFLD}) \rho \Delta \Delta \text{L7} \)
[134] a Current field index and number:
[135] \( \Delta \Delta \text{F} \leftarrow \Delta \Delta \text{F} + 1 \)
[136] \( \Delta \Delta \text{FLD} \leftarrow \Delta \Delta \text{FLD} \backslash \Delta \Delta \text{FLDS}[\Delta \Delta \text{F}] \)
[137] a Read, compress, replace field FLD for set SET:
[138] \( \Delta \Delta \text{CMP} \leftarrow \wedge \Delta \Delta \text{TIE}, \Delta \Delta \text{FLD} + \Delta \Delta \text{SDISP} \)
[139] \( (\wedge \Delta \Delta \text{SEL} \wedge \text{READ} \wedge \Delta \Delta \text{CMP} \wedge \text{UFREPLACE} \wedge \Delta \Delta \text{CMP} \)
[140] \( \rightarrow \Delta \Delta \text{L16} \)
[141] \( \Delta \Delta \text{L17}: \Delta \Delta \text{NFP} \uparrow \Delta \Delta \text{EX} \Delta \Delta \text{TIE}, 7 \)
[142] a RPS has been changed if DEL=0; ARPS has been changed
[143] a if DEL>0. Replace only one or the other.
[144] \( \rightarrow (\times \Delta \Delta \text{DEL}) \rho \Delta \Delta \text{L18} \)
[145] \( \Delta \Delta \text{RPS} \uparrow \text{UFREPLACE} \wedge \Delta \Delta \text{TIE}, 8 \)
[146] \( \rightarrow 0 \)
[147] \( \Delta \Delta \text{L18}: \Delta \Delta \text{ARPS} \uparrow \text{UFREPLACE} \wedge \Delta \Delta \text{TIE}, 10 \)
\[\]
Chapter 14 Solutions

FILE DESIGN AND UTILITIES

[WSID: MULTIFLO]

V MAT=*INDS INDEX FP;CHR;COL;COLS;DATA;DEL;DISP;F;FLD;
FLDS;FT;INCR;IND;NFLD;NSET;S;SDISP;SET;SETS;SHAPE;
TIE;UNQ;W;WID;DIO

[1] a Returns data from fields 11↑FP for records
[2] a identified by file indices matrix INDS.
[3] a First row of INDS is set number (origin 1);
[4] a second row is index (origin 1) within set.
[5] a INDS may be of any dimension as long as its
[6] a first coordinate is 2. Result has same number
[7] a of columns as fields 11↑FP have columns.
[8] a Leading shape of result is 1↓ρINDS.
[9] \$ERROR(1↑ρFP)'/RANK ERROR'
[10] \$ERROR((1↓ρFP)↑2≠1↓ρINDS)'/LENGTH ERROR'
[11] DIO↑1
[12] NFLD=ρFLDS=1↓FP
[15] \$ERROR((1↓FLDS≤\INCR)<DEL≤FLDS)'/INVALID FIELD NUMBER'
[16] TIE=FP[2]
[17] FT=(DFREAD TIE,4)[;FLDS]
[18] a Width (no. columns) of specified fields:
[19] WID=FT[1;]
[20] \$ERROR(0≤WID)'/INACTIVE FIELD'
[21] a Fields must be all character or all numeric:
[22] CHR=2=1↑FT[2;]
[23] \$ERROR(CHR.≠2=FT[2;])'/DOMAIN ERROR'
[24] a Shape of result (excluding columns):
[25] SHAPE=1↓ρINDS
[26] COLS=+/WID
[27] a Break apart indices:
[28] NREC=ρSETS+, 1 0 #INDS
[29] IND$+, 0 1 #INDS
[30] a Construct all-zero or all-blank result:
[31] →CHRρL1
[32] MAT=(NREC, COLS)ρ0
[33] →L2
[34] L1:MAT=(NREC, COLS)ρ' '
[35] a Exit if no indices or no fields:
[36] L2:→(↑NREC×COLS)↓L6
[37] DISP=FP[4]
[38] a Determine distinct set numbers (deleting -1s):
[39] UNQ=SETS[↓SETS]
[40] NSET=ρUNQ=(UNQ≠1↓1, UNQ)/UNQ
[41] a Last set index:
[42] S=0
[43] a Loop by distinct set:
[44] L3:→(S≥NSET)ρL6
[45] a Current set index and number:
[46] S=S+1
[47] SET=UNQ[S]
[48] a Displacement (no. components) before this set:
[49] SDISP=DISP+INCR×SET+•1
\( V \) INDEX (continued)

[50] \( \text{Indices of INDS to retrieve in this set:} \)
[51] \( \text{IND}(\text{SET}=\text{SETS})/|\text{NREC} \)
[52] \( \text{Last field index, and columns inserted so far:} \)
[53] \( F+\text{COL}=0 \)
[54] \( \text{Loop by specified field:} \)
[55] \( L4:\rightarrow(F\geq\text{NFLD})\rho L3 \)
[56] \( \text{Current field index, number and width:} \)
[57] \( F+\text{F}+1 \)
[58] \( \text{FLD}+\text{FLDS}[F] \)
[59] \( W+\text{WID}[F] \)
[60] \( \text{Read data:} \)
[61] \( \text{DATA} \rightarrow \text{DFREAD TIE,FLD+SDISP} \)
[62] \( \text{Branch if matrix field:} \)
[63] \( \rightarrow(W>1)\rho L5 \)
[64] \( \text{Insert vector of data:} \)
[65] \( \text{COL}+\text{COL}+1 \)
[66] \( \text{MAT}[\text{IND};\text{COL}] \rightarrow \text{DATA}[\text{INDS}[\text{IND}]] \)
[67] \( \rightarrow L4 \)
[68] \( \text{Insert matrix of data:} \)
[69] \( L5:\text{MAT}[\text{IND};\text{COL}+W] \rightarrow \text{DATA}[\text{INDS}[\text{IND}]]; \)
[70] \( \text{COL}+\text{COL}+W \)
[71] \( \rightarrow L4 \)
[72] \( \text{Exit if result has correct shape already:} \)
[73] \( L6:\rightarrow(1=\rho \text{SHAPE})\rho 0 \)
[74] \( \text{MAT}+(\text{SHAPE,COLS})\rho \text{MAT} \)

\[ \text{WSID: MULTIFLO} \]

\( V \) INDS INDEX WS FP;DATA;DEL;DISP;FLD;FLDS;FT;INCR;IND;
LAB;NFLD;NREC;NSET;S;SDISP;SET;SETS;SHAPE;TIE;UNQ;W;
WID;DIO

[1] \( \text{Retrieves data from fields 11\#FP for records} \)
[2] \( \text{identified by file indices matrix INDS.} \)
[3] \( \text{First row of INDS is set number (origin 1);} \)
[4] \( \text{second row is index (origin 1) within set.} \)
[5] \( \text{INDS may be of any dimension as long as its} \)
[6] \( \text{first coordinate is 2. The retrieved data are} \)
[7] \( \text{assigned to global variables named Fn where n} \)
[8] \( \text{is the number of the field retrieved (e.g. F3} \)
[9] \( \text{and F7 for 11\#FP of 3 7). Global variables} \)
[10] \( \text{have same number of columns as corresponding} \)
[11] \( \text{fields. Leading shape is 1\#\rho \text{INDS.}} \)
[12] \( \text{ERROR(1\#\rho \text{FP})/\text{'RANK ERROR'}} \)
[13] \( \text{ERROR((11\#\rho \text{FP})\text{\text{\& 1\#\rho \text{INDS}}}/\text{'LENGTH ERROR'}} \)
[14] \( \text{DIO=1} \)
[15] \( \text{NFLD}+\rho \text{FLDS}+11\#FP \)
[16] \( \text{INCR}+\rho \text{FP}[3] \)
[17] \( \text{DEL}+\rho \text{FP}[11] \)
[18] \( \text{ERROR(__\text{\text{\& FLDS}\text{\& INCR} \text{\& DEL\text{\& FLDS}}}'/\text{'INVALID FIELD NUMBER'}} \)
[19] \( \text{TIE}+\rho \text{FP}[2] \)
[20] \( \text{FT} \rightarrow \text{DFREAD TIE,4}[;\text{FLDS}] \)
INDEXWS (continued)

[21] Width (no. columns) of specified fields:
[22] WID^FT[1,]
[23] ERROR(0<WID)'/INACTIVE FIELD'
[24] Shape of result (excluding columns):
[25] SHAPE^1+pINDS
[26] Break apart indices:
[27] NREC^pSETS^, 1 0 /INDS
[28] IND^0, 1 /INDS
[29] Construct all-zero or all-blank globals.
[30] Label vector needed below based upon field
[31] rank and type (nvec, nmat, cvec, cmat):
[32] LAB^L2,L3,L4,L5[(2!WID)+2x2=FT[2,]]
[33] Last field index:
[34] F^0
[35] Loop by specified field:
[36] L1:(F>=NFLD)^L7
[37] Current field index and number:
[38] F=F+1
[39] FLD^FLDS[F]
[40] Branch based upon type and width:
[41] LAB[F]
[42] L2:DATA^NREC^0
[43] L6
[44] L3:DATA^NREC,WID[F]^0
[45] L6
[46] L4:DATA^NREC^'
[47] L6
[48] L5:DATA^NREC,WID[F]^'
[49] L6:'F',(FLD),''DATA'
[50] L7
[51] Exit if no indices or no fields:
[52] L7:=(xNRECxNFLD)^L11
[53] DISP^FP[4]
[54] Determine distinct set numbers (deleting -1s):
[55] UNQ^SETS[::-SETS]
[56] NSET^pUNQ^UNQ,-1,-1,UNQ)/UNQ
[57] Last set index:
[58] S^0
[59] Loop by distinct set:
[60] L8:(S>=NSET)^L11
[61] Current set index and number:
[62] S=S+1
[63] SET^UNQ[S]
[64] Displacement (no. components) before this set:
[65] SDISP^DISP+INCRxSET+1
[66] Indices of INDs to retrieve in this set:
[67] IND^SET=SETS)/\NREC
[68] Last field index:
[69] F^0
[70] Loop by specified field:
[71] L9:(F>=NFLD)^L8
[72] Current field index, number and width:
[73] F=F+1
INDEXWS (continued)

[74] FLD+FLDS[F]
[75] W+WID[F]
[76] \( \Rightarrow \) Read data:
[77] DATA\(+\)FREAD TIE,FLD+SDISP
[78] \( \Rightarrow \) Branch if matrix field:
[79] \( \Rightarrow (W>1) \Rightarrow L10 \)
[80] \( \Rightarrow \) Insert vector of data:
[81] \( \Rightarrow 'F',(\#FLD),[(IND)\(+\)DATA[INDS[IND]]] \)
[82] \( \Rightarrow L9 \)
[83] \( \Rightarrow \) Insert matrix of data:
[84] \( \Rightarrow L10: 'F',(\#FLD),[(IND;J\(+\)DATA[INDS[IND];J] \)
[85] \( \Rightarrow L9 \)
[86] \( \Rightarrow \) Exit if globals have correct shape already:
[87] \( \Rightarrow L11:\Rightarrow (1=pSHAPE) \Rightarrow 0 \)
[88] \( \Rightarrow \) Last field index:
[89] \( \Rightarrow F+0 \)
[90] \( \Rightarrow \) Loop by specified field:
[91] \( L12:\Rightarrow (F=NFLD) \Rightarrow 0 \)
[92] \( \Rightarrow \) Current field index and number:
[93] \( \Rightarrow F\Rightarrow F+1 \)
[94] \( \Rightarrow \) FLD\(+\)FLDS[F]
[95] \( \Rightarrow \) Reshape globals Fn to conform with shape of indices:
[96] DATA\(+\)T\(+'F','FLD
[97] \( \Rightarrow \) DATA\(+\)(SHAPE,1\(\Rightarrow\)DATA\)\(\Rightarrow\)DATA
[98] \( \Rightarrow 'T,\Rightarrow\)DATA'
[99] \( \Rightarrow L12 \)

\[ WSID: \text{MULTIFLO} \]

\( \Rightarrow \) Loops through active sets doing the following:
[1] \( \Rightarrow \) reads the fields specified in \((11+(11\downarrow F)\downarrow 0)\downarrow F \)
[2] \( \Rightarrow \) (calling them F5, F9, etc. for fields 5, 9, etc.),
[3] \( \Rightarrow \) executes the character vector EXPR, and returns
[4] \( \Rightarrow \) data from fields \( -(1+(11\downarrow F)\downarrow 0)\uparrow 11\downarrow F \) for the
[5] \( \Rightarrow \) records of that set which correspond to 1s in the
[6] \( \Rightarrow \) resulting bit vector. Result has one row per
[7] \( \Rightarrow \) record found and same number of columns as the
[8] \( \Rightarrow \) latter set of fields has columns. If EXPR and
[9] \( \Rightarrow \) selection fields are empty, all records are
[10] \( \Rightarrow \) selected. Note that EXPR is executed in origin 1;
[11] \( \Rightarrow \) e.g. 'F3[;2]' always refers to 2nd column.
[12] \( \Rightarrow \) ERROR\( (1\downarrow p\downarrow F)\downarrow 1<\downarrow p\downarrow EXPR) '/RANK ERROR'
[13] \( \Rightarrow \) ERROR\( (0=1\uparrow 0\downarrow p\downarrow EXPR) '/DOMAIN ERROR'
[14] \( \Rightarrow \) IO\(+\)1
[15] \( \Rightarrow \) Extract 2 sets of fields from FP:
[16] \( \Rightarrow \) T\(+11+(11\downarrow F)\downarrow 0 \)
SELECT (continued)

\[ \Delta \Delta \text{NFLD} - \rho \Delta \Delta \text{FLDS} \leftarrow 1 \downarrow \left( \Delta \Delta T^{-1} \right) \rho \Delta \Delta \text{FP} \]

\[ \Delta \Delta \text{ALL} \leftarrow 0 = \Delta \Delta \text{NFLD} - \rho \Delta \Delta \text{SFLDS} \leftarrow \Delta \Delta T \downarrow \Delta \Delta \text{FP} \]

\[ \text{ERROR}\left( \left( \Delta \Delta \text{ALL} = \Delta \Delta \text{EXPR} \neq \# \right)' \cdot \left( 11 > \rho \Delta \Delta \text{FP} \right) \right) '/' \text{LENGTH ERROR}' \]

\[ \Delta \Delta \text{INCR} - \Delta \Delta \text{FP}[3] \]

\[ \Delta \Delta \text{DEL} \leftarrow \Delta \Delta \text{FP}[11] \]

\[ \Delta \Delta T \Delta \Delta \text{SFLDS}, \Delta \Delta \text{FLDS} \]

\[ \text{ERROR}\left( \left( \wedge \Delta \Delta T \in \Delta \Delta \text{INCR} \right) \leq \Delta \Delta \text{DEL} \in \Delta \Delta T \right) '/' \text{INVALID FIELD NUMBER}' \]

\[ \Delta \Delta \text{TIE} \leftarrow \Delta \Delta \text{FP}[2] \]

\[ \Delta \Delta \text{FT} \leftarrow \text{DFREAD} \Delta \Delta \text{TIE}, 4 \]

\[ \text{Width (no. columns) of fields:} \]

\[ \Delta \Delta \text{WID} \leftarrow \Delta \Delta \text{FT}[1] \]

\[ \text{ERROR}\left( 0 \leq \Delta \Delta \text{WID}[\Delta \Delta T] \right) '/' \text{INACTIVE FIELD}' \]

\[ \text{Fields must be all character or all numeric:} \]

\[ \Delta \Delta T + 2 = \Delta \Delta \text{FT}[2; \Delta \Delta \text{FLDS}] \]

\[ \Delta \Delta \text{CHR} \leftarrow 1 \uparrow \Delta \Delta T \]

\[ \text{ERROR}\left( \Delta \Delta \text{CHR} \neq \left. . \right/ \Delta \Delta T \right) '/' \text{DOMAIN ERROR}' \]

\[ \text{Columns in result:} \]

\[ \Delta \Delta \text{COLS} \leftarrow \Delta \Delta \text{WID}[\Delta \Delta \text{FLDS}] \]

\[ \text{Construct empty result:} \]

\[ \rightarrow \Delta \Delta \text{CHR} \rho \Delta \Delta \text{L1} \]

\[ \Delta \Delta \text{MAT} \leftarrow (0, \Delta \Delta \text{COLS}) \rho \Delta \Delta \text{FILL} \leftarrow 0 \]

\[ \rightarrow \Delta \Delta \text{L2} \]

\[ \Delta \Delta \text{L1} \leftarrow \Delta \Delta \text{MAT} \leftarrow (0, \Delta \Delta \text{COLS}) \rho \Delta \Delta \text{FILL} \leftarrow ' ' \]

\[ \text{Exit if no records:} \]

\[ \Delta \Delta \text{L2} \rightarrow \left( \times \Delta \Delta \text{FP}[10] \right) \leftarrow 0 \]

\[ \text{Field names (e.g. 'F5 F9') to be erased below:} \]

\[ \Delta \Delta \text{FNAM} \leftarrow \left( \Delta \Delta \text{NFLD}, 1 \right) \rho \Delta \Delta \text{SFLDS} \]

\[ \Delta \Delta \text{FNAM} \leftarrow 'F', (+/' = \Delta \Delta \text{FNAM}) \Phi \Delta \Delta \text{FNAM} \]

\[ \Delta \Delta \text{DISP} \leftarrow \Delta \Delta \text{FP}[4] \]

\[ \text{Label vector needed below based upon field rank and whether in WS or onfile when needed} \]

\[ \left( \text{vfile, mfile, vws, mws}: \right) \]

\[ \Delta \Delta \text{LAB} \leftarrow (\Delta \Delta \text{L17}, \Delta \Delta \text{L18}, \Delta \Delta \text{L19}, \Delta \Delta \text{L21}) \left[ (2 \uparrow \Delta \Delta \text{WID}[\Delta \Delta \text{FLDS}]) + 2 \times \Delta \Delta \text{FLDS} \leq \Delta \Delta \text{SFLDS} \right] \]

\[ \Delta \Delta \text{ARPS} \leftarrow \Delta \Delta \text{ARPS} \leftarrow \text{DFREAD} \Delta \Delta \text{TIE}, 8 \]

\[ \text{Branch unless ARPS should be read:} \]

\[ \left( = \times \Delta \Delta \text{FP}[7, 10] \right) \rho \Delta \Delta \text{L3} \]

\[ \Delta \Delta \text{ARPS} \leftarrow \text{DFREAD} \Delta \Delta \text{TIE}, 10 \]

\[ \text{Consider only nonempty sets:} \]

\[ \Delta \Delta \text{L3} : \Delta \Delta \text{SETS} \leftarrow 0 \# \Delta \Delta \text{ARPS} \]

\[ \text{... and sets specified in <layers>:} \]

\[ \left( \times \Delta \Delta \text{FP}[11] \right) \downarrow \Delta \Delta \text{L7} \]

\[ \text{UNCH 'layers')} \downarrow \Delta \Delta \text{L7} \]

\[ \text{Read layer values:} \]

\[ \Delta \Delta \text{LV} \leftarrow \text{DFREAD} \Delta \Delta \text{TIE}, 9 \]

\[ \text{Branch if matrix layers field:} \]

\[ \Delta \Delta \text{L4} = \text{ERROR}(\Delta \Delta \text{FT}[1; \Delta \Delta \text{FP}[11]) \# 1 \uparrow \text{players}) '/' \text{LENGTH ERROR}' \]

\[ \rightarrow (2 = \rho \text{players}) \rho \Delta \Delta \text{L5} \]
V SELECT (continued)

[69] layers+=((x/-ltplayers),-ltplayers)players

[70] a Branch if numeric matrix field:

[71] ΔΔL5:→(0=lt0ρΔΔLV)ρΔΔL6

[72] ΔΔSETS+ΔΔSETS∧(layers CMIOTA ΔΔLV)≤1ρplayers

[73] →ΔΔL7

[74] ΔΔL6:ΔΔSETS+ΔΔSETS∧V/ΔΔLV∧.≥layers

[75] a Convert to indices; erase <layers>:

[76] ΔΔL7:ΔΔSETS+ΔΔSETS/1ρΔΔSETS

[77] ΔΔNSET+ρΔΔSETS

[78] ΔΔLV+ΔΔ 'layers'

[79] a Last set index:

[80] ΔΔS<0

[81] a Loop by nonempty set:

[82] ΔΔL8:→(ΔΔS=ΔΔNSET)ρ0

[83] a Current set index and number:

[84] ΔΔS+ΔΔS+1

[85] ΔΔNSET+ΔΔSETS[ΔΔS]

[86] a Displacement (no. components) before this set:

[87] ΔΔDISP+ΔΔDISP+ΔΔINCR×ΔΔSET+1

[88] a Branch if deletion field unneeded:

[89] →(ΔΔNDEL+ΔΔARPS[ΔΔSET]>0)ρΔΔL9

[90] a Read deletion field for set SET:

[91] ΔΔBIΓT+ΔFREAD ΔΔTIE,ΔΔDEL+ΔΔSDISP

[92] a Branch if no selection expression:

[93] ΔΔL9:ΔΔSEL+1

[94] →ΔΔALLρΔΔL14

[95] a Are all records in this set active:

[96] ΔΔACTIVE+ΔΔARPS[ΔΔSET]=ΔΔRPS[ΔΔSET]

[97] a Last field index:

[98] ΔΔF<0

[99] a Loop by field specified:

[100] ΔΔL10:→(ΔΔF≥ΔΔNSFLD)ρΔΔL13

[101] a Current field index and number:

[102] ΔΔF+ΔΔF+1

[103] ΔΔFLD+ΔΔSFLDS[ΔΔF]

[104] a Read field FLD for set SET:

[105] ΔΔDATA+ΔFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

[106] a Branch if all records in this set active:

[107] →ΔΔACTIVEρΔΔL12

[108] a Branch if deletion field needed:

[109] →ΔΔNDELΔΔL11

[110] a Active records are leading records:

[111] ΔΔDATA←(ΔΔARPS[ΔΔSET],(ΔΔWID[ΔΔFLD])>1)ρΔΔWID[ΔΔFLD])ρ

ΔΔDATA

[112] →ΔΔL12

[113] a Apply deletion field:

[114] ΔΔL11:ΔΔDATA+ΔΔBIT'ΔΔDATA

[115] a Assign data to global variable Fn:

[116] ΔΔL12:→'F',('ΔΔFLD'),'←ΔΔDATA'

[117] →ΔΔL10

[118] a Once all fields have been read, execute EXPR:

[119] ΔΔL13:ΔΔSEL←ΔΔEXPR
\[ V \text{ SELECT (continued)} \]

\[ [120] \text{a Indices to retrieve (after squeezing deletions):} \]
\[ [121] \Delta \Delta L14: \Delta \Delta \text{IND} \leftarrow \Delta \Delta \text{SEL} / r \Delta \Delta \text{ARPS}[\Delta \Delta \text{SET}] \]
\[ [122] \text{a Continue to next set if no records selected:} \]
\[ [123] \rightarrow (\times \Delta \Delta \text{NUM} \cap \Delta \Delta \text{IND}) \downarrow \Delta \Delta L23 \]
\[ [124] \text{a No. records found before this set:} \]
\[ [125] \Delta \Delta \text{ROWS} \leftarrow 1 \cap r \Delta \Delta \text{MAT} \]
\[ [126] \text{a Expand result for new records:} \]
\[ [127] \Delta \Delta \text{MAT} \leftarrow \Delta \Delta \text{MAT}, [1](\Delta \Delta \text{NUM}, \Delta \Delta \text{COLS}) \cap \Delta \Delta \text{FILL} \]
\[ [128] \text{a Indices to retrieve (before squeezing deletions):} \]
\[ [129] \Delta \Delta \text{INDS} \leftarrow \Delta \Delta \text{IND} \]
\[ [130] \text{a Branch if deletion field unneeded:} \]
\[ [131] \rightarrow \Delta \Delta \text{NDEL} \cap \Delta \Delta L15 \]
\[ [132] \text{a Reset INDS, considering deletion field:} \]
\[ [133] \Delta \Delta \text{INDS} \leftarrow (\Delta \Delta \text{BIT}/r \Delta \Delta \text{BIT})[\Delta \Delta \text{IND}] \]
\[ [134] \text{a Last field index and columns inserted so far:} \]
\[ [135] \Delta \Delta L15: \Delta \Delta \text{F} \leftarrow \Delta \Delta \text{COL} + 0 \]
\[ [136] \text{a Loop by field to retrieve:} \]
\[ [137] \Delta \Delta L16: \rightarrow (\Delta \Delta \text{F} \geq \Delta \Delta \text{NFD}) \cap \Delta \Delta L23 \]
\[ [138] \text{a Current field index, number and width:} \]
\[ [139] \Delta \Delta \text{F} \leftarrow \Delta \Delta \text{F} + 1 \]
\[ [140] \Delta \Delta \text{FLD} \leftarrow \Delta \Delta \text{FLDS}[\Delta \Delta \text{F}] \]
\[ [141] \Delta \Delta \text{W} \leftarrow \Delta \Delta \text{WID}[\Delta \Delta \text{FLD}] \]
\[ [142] \text{a Branch depending on field width and whether} \]
\[ [143] \text{a already in workspace:} \]
\[ [144] \rightarrow \Delta \Delta \text{LAB}[\Delta \Delta \text{F}] \]
\[ [145] \Delta \Delta L17: \Delta \Delta \text{DATA} \leftarrow (\Omega \text{READ} \Delta \Delta \text{TIE}, \Delta \Delta \text{FLD} + \Delta \Delta \text{SDISP})[\Delta \Delta \text{INDS}] \]
\[ [146] \rightarrow \Delta \Delta L20 \]
\[ [147] \Delta \Delta L18: \Delta \Delta \text{DATA} \leftarrow (\Omega \text{READ} \Delta \Delta \text{TIE}, \Delta \Delta \text{FLD} + \Delta \Delta \text{SDISP})[\Delta \Delta \text{INDS}] \]
\[ [148] \rightarrow \Delta \Delta L22 \]
\[ [149] \Delta \Delta L19: \Delta \Delta \text{DATA} \leftarrow (\&'F', \&\Delta \Delta \text{FLD})[\Delta \Delta \text{IND}] \]
\[ [150] \Delta \Delta L20: \Delta \Delta \text{COL} \leftarrow \Delta \Delta \text{COL} + 1 \]
\[ [151] \Delta \Delta \text{MAT}[\Delta \Delta \text{ROWS} + l \Delta \Delta \text{NUM}; \Delta \Delta \text{COL}] \leftarrow \Delta \Delta \text{DATA} \]
\[ [152] \rightarrow \Delta \Delta L16 \]
\[ [153] \Delta \Delta L21: \Delta \Delta \text{DATA} \leftarrow (\&'F', \&\Delta \Delta \text{FLD})[\Delta \Delta \text{IND}] \]
\[ [154] \Delta \Delta L22: \Delta \Delta \text{MAT}[\Delta \Delta \text{ROWS} + l \Delta \Delta \text{NUM}; \Delta \Delta \text{COL} + \Delta \Delta \text{W}] \leftarrow \Delta \Delta \text{DATA} \]
\[ [155] \Delta \Delta \text{COL} \leftarrow \Delta \Delta \text{COL} + \Delta \Delta \text{W} \]
\[ [156] \rightarrow \Delta \Delta L16 \]
\[ [157] \text{a Erase field variables (e.g. F5, F9, ...):} \]
\[ [158] \Delta \Delta L23: \Delta \Delta \text{T} \leftarrow \Omega \text{EX} \Delta \Delta \text{FNAM} \]
\[ [159] \rightarrow \Delta \Delta L8 \]
\[ V \]
[WSID: MULTIFLO]

1. A Loops through active sets doing the following:
2. A reads the fields specified in \((11+ (11 \times FP) \times 0) \times FP\)
3. A (calling them F5, F9, etc. for fields 5, 9, etc.),
4. A executes the character vector EXPR, and retrieves
5. A data from fields \((-1 + (11 \times FP) \times 0) \times 11 \times FP\) for the
6. A records of that set which correspond to 1s in the
7. A resulting bit vector. The retrieved data are
8. A assigned to global variables named Fn where n is
9. A the number of the field retrieved (e.g. F3 and F7
10. A if 3 7 are the numbers of the latter set of
11. A fields). Global variables have same number of
12. A columns as corresponding fields. If EXPR and
13. A selection fields are empty, all records are
14. A selected. Note that EXPR is executed in origin 1;
15. A e.g. 'F3[;2]' always refers to 2nd column.
16. A ERROR((1 \times \rho \times \Delta FP) \times 1 < \rho \times \Delta EXPR) / 'RANK ERROR'
17. A ERROR(0 = 1 \times \rho \times \Delta EXPR) / 'DOMAIN ERROR'
18. A I0 \times 1
19. A Extract 2 sets of fields from FP:
20. A \Delta T \times 11 \times (11 \times \Delta FP) \times 0
21. A \Delta NFLD \times 0 \times \Delta FLDS \times 11 \times \Delta T \times -1 \times \Delta FP
22. A \Delta ALL \times 0 \times \Delta NSFLD \times \rho \times \Delta SFLDS \times \Delta T \times \Delta FP
23. A ERROR((\Delta ALL = \Delta EXPR \times .#') \times 11 \times \rho \times \Delta FP) / 'LENGTH ERROR'
24. A \Delta INC \times \Delta FP \times [3]
25. A \Delta DEL \times 1 \times \Delta FP \times [11]
26. A \Delta T \times \Delta SFLDS \times \Delta FLDS
27. A ERROR((\^/ \Delta T \times \Delta INC) \times \Delta DEL \times \Delta T) / 'INVALID FIELD NUMBER'
28. A \Delta TIE \times \Delta FP \times [2]
29. A \Delta FT \times DREAD \times \Delta TIE \times 4
30. A Width (no. columns) of fields:
31. A \Delta WID \times \Delta FT \times [1 ; 1]
32. A ERROR(0 \in \Delta WID [\Delta T]) / 'INACTIVE FIELD'
33. A Exit if no fields:
34. A \times \Delta NFLD \downarrow \Delta L 36
35. A Datatypes \(1, 2, 3, 4\) of fields to be retrieved:
36. A \Delta TYPES \times \Delta FT \times [2 ; \Delta FLDS]
37. A No. preceding cols. for each fld within its datatype:
38. A \Delta T \times (14) \times . \times \Delta TYPES
39. A \Delta START \times \times \Delta T \times (\rho \Delta AT) \times 0, \Delta S \times \times \Delta T \times (\rho \Delta T) \rho \Delta WID[\Delta FLDS]
40. A No. columns of each datatype:
41. A \Delta COLS \times, 4 \times -1 \times \Delta S
42. A Construct empty result matrices (one per datatype):
43. A \Delta M1 \times (0, \Delta COLS[11]) \times 0
44. A \Delta M2 \times (0, \Delta COLS[21]) \times 0
45. A \Delta M3 \times (0, \Delta COLS[31]) \times 0
46. A \Delta M4 \times (0, \Delta COLS[41]) \times 0
\[ SELECTWS \] (continued)

1. Exit if no records:
   \[ (\times \Delta F \text{FP}[-10]19 \]
2. Field names (e.g. 'F5 F9') to be erased below:
   \[ \Delta \text{FNAM}(\times \Delta \text{NSFLD}, 1) \rho \Delta \text{FSLDS} \]
3. Field names (e.g. 'F', (+/-' = \Delta \text{FNAM}) \Phi \Delta \text{FNAM} \]
4. Label vector needed below based upon field rank
   \[ \text{and whether in WS or onfile when needed} \]
5. Field names (e.g. 'F5 F9') to be erased below:
   \[ \text{ANFAM} \left( \text{ANSFLD}, \text{AFLDS} \right) \]
6. Label vector based upon datatype:
   \[ \Delta \text{LAB}((\Delta \text{L15}, \Delta \text{L16}, \Delta \text{L17}, \Delta \text{L19})[(\times \Delta \text{WID}[\Delta \text{FLDS}]) + 2 \times \Delta \text{FLDS}\text{FLDS} \]
7. Branch unless ARPS should be read:
   \[ \text{ARPS[\times \text{FREAD} \Delta \text{TIE}, 8} \]
8. Consider only nonempty sets:
   \[ \Delta \text{L1:}\Delta \text{SETS} \neq 0 \Delta \text{ARPS} \]
9. ...and sets specified in <layers>:
   \[ \times \text{FP}[1] \downarrow \Delta \text{L5} \]
10. Read layer values:
    \[ \Delta \text{LV[\times \text{FREAD} \Delta \text{TIE}, 9} \]
11. Branch if matrix layers field:
    \[ \times \text{FP}[2, 1 \Delta \text{L2} \]
12. Convert <layers> to matrix if not already:
    \[ \text{ERROR(\Delta \text{FT}[1]; \Delta \text{FP}[1])} \times \text{FP}[-1 \Delta \text{players}] / ' \text{LENGTH ERROR}' \]
13. Branch if numeric matrix field:
    \[ \times \text{FP}[2, 1 \rho \Delta \text{L3} \]
14. Convert to indices; erase <layers>:
    \[ \text{CMIOTA} \downarrow \text{LAB} \Delta \text{LV} \leq \text{players} \]
15. Last set index:
    \[ \Delta \text{S} \times 0 \]
16. Loop by nonempty set:
    \[ \Delta \text{L6:} \times \text{ANSET[\times \Delta \text{S} \text{L27} \]
17. Current set index and number:
    \[ \Delta \text{S} \times 0 \Delta \text{S} + 1 \]
18. Displacement (no. components) before this set:
    \[ \Delta \text{DISP}(\times \text{DISP} \times \text{INCR} \times \text{SET} \times 1) \]
19. Branch if deletion field unneeded:
    \[ \times \text{FP}[\Delta \text{NDEL} \times \text{ARPS[\times \text{SET}]} / 0 \times \text{L7} \]
\[ SELECTWS \] (continued)

\[ a \] Read deletion field for set SET:
\[ \Delta A TIE, \Delta D E L + \Delta S D I S P \]
\[ b \] Branch if no selection expression:
\[ \Delta A S E L \rightarrow 1 \]
\[ \rightarrow \Delta A L 1 \]
\[ \rightarrow \Delta A L 2 \]
\[ \rightarrow \Delta A L 3 \]
\[ \rightarrow \Delta A L 4 \]
\[ \rightarrow \Delta A L 5 \]
\[ \rightarrow \Delta A L 6 \]
\[ \rightarrow \Delta A L 7 \]
\[ \rightarrow \Delta A L 8 \]
\[ \rightarrow \Delta A L 9 \]
\[ \rightarrow \Delta A L 10 \]
\[ \rightarrow \Delta A L 11 \]
\[ \rightarrow \Delta A L 12 \]
\[ \rightarrow \Delta A L 13 \]
\[ \rightarrow \Delta A L 14 \]
SELECTWS (continued)

Current field index, number and width:

Branch depending on field width and whether already in workspace:

Branch based upon datatype:

Erase field variables (e.g. F5, F9, ...):

Label vector by datatype needed below:

Last field index:

Loop by field to retrieve:

Current field index, number and width:

Branch if matrix field:

Branch based upon datatype:

Branch based upon datatype:
\( \text{SELECTWS (continued)} \)

[202] a Assign to Fn vars.: 
[203] \( \Delta{\Delta}{L35}: 'F', (\&{\Delta}{FLD}), '\Delta{\Delta}{DATA} \)
[204] \( \rightarrow \Delta{\Delta}{L28} \)
[205] \( \Delta{\Delta}{L36}: \Delta{\Delta}{S<\text{DEX}} '\text{layers}' \)

\[ \text{[WSID: MULTIFLO]} \]

\( \text{v R\text{\textasciitilde}A ASSIGN B} \)

[1] a Used as: rinds INDEXA \((\text{FP,flds})\) ASSIGN mat 
[2] R\text{\textasciitilde}A 
[3] assign\text{\textasciitilde}B 

\[ \text{[WSID: MULTIFLO]} \]

\( \text{v NFP<INDS INDEXA FP;A;AFLD;ARPS;BAD;BIT;BLK;CMP;COL;}
\text{COLS;DATA;DCMP;DEL;DISP;F;FILED;FLD;FLDS;FREC;FT;}
\text{GOOD;I;INDS;INCR;IND;LAY;LAYER;LEAD;LSETS;LV;LVAR;M;}
\text{MIN;N;NFLD;NR;NREC;NSET;RPS;S;SD;SDISP;SET;SETS;}
\text{SHAPE;SIND;SINGLE;T;TIE;UNQ;VAR;VEC;W;WID;IO} \)

[1] a Used as: \( \text{FP+rinds INDEXA (FP,flds)} \) ASSIGN mat 
[2] a Inserts data from global matrix \( \text{<assign>} \) into 
[3] a fields \( \text{11<FP} \) for records identified by 
[4] a indices matrix INDS. First row of INDS is set 
[5] a number (origin 1); second row is index (origin 1) 
[6] a within set. INDS may be of any dimension as long as 
[7] a its first coordinate is 2. \( \text{<assign>} \) has same number 
[8] a of columns as fields \( \text{11<FP} \) have columns. 
[9] a Leading shape of \( \text{<assign>} \) is \( 1\uparrow\rho\text{INDS} \). \( \text{<assign>} \) is 
[10] a erased upon completion. 
[11] \text{ERROR(1\#\rho\text{FP})/'RANK ERROR'} 
[12] \text{ERROR((11>\rho\text{FP})\#21\#1\uparrow\rho\text{INDS})/'LENGTH ERROR'} 
[13] \text{DIO<1} 
[14] \text{NFLD<\rho\text{FLDS}<11<FP} 
[15] \text{NFP<11<\rho\text{FP}} 
[16] \text{INCR<FP[3]} 
[17] \text{DEL<FP[1]} 
[18] \text{FILL<FP[1]} \text{<0} 
[19] \text{ERROR((^\text{/FLDS}\in\text{\&INCR} \leq \text{DEL}\text{\&FLDS})/'INVALID FIELD NUMBER'} 
[20] \text{TIE<FP[2]} 
[21] \text{FT<DREAD TIE,4} 
[22] a Width (no. columns) of specified fields: 
[23] \text{WID<FT[1]} 
[24] \text{COLS<\text{\&W<\text{\&WID[FLDS}]} 
[25] \text{ERROR(0<\text{\&W}')/'INACTIVE FIELD'} 
[26] a Fields must be all character or all numeric: 
[27] \text{S<21\#T<FT[2;FLDS]} 
[28] \text{ERROR(S\#T<2')/'DOMAIN ERROR'} 
[29] a Break apart indices: 
[30] \text{SHAPE<1\#\rho\text{INDS} 
[31] NREC<\rho\text{SETS}, 1 0 \#\text{INDS 
[32] IND$\text{\textasciitilde}, 0 1 \#\text{INDS}}\)
\V INDEXA (continued)

[33] \a Singleton data?
[34] SINGLE\wedge 1\wedge =\rho assign
[35] \rightarrow SINGLE\rho L3
[36] \a Branch unless singleton (vector) record:
[37] \rightarrow ((COLS\neq 1)\wedge 1\wedge =1\wedge \rho assign)\rightarrow L1
[38] \[ERROR(COLS\neq 1\wedge \rho assign)'/LENGTH ERROR'
[39] \rightarrow L2
[40] L1: \[ERROR((\rho SHAPE)\wedge .(\rho assign)+1, (COLS=1)\rho 0)'/RANK ERROR'
[41] S\leftarrow (1+\rho SHAPE)\rho (\rho assign), 1
[42] \[ERROR(SV.\neq SHAPE, COLS)'/LENGTH ERROR'
[43] \rightarrow (2=\rho assign)\rho L3
[44] L2: assign\rightarrow (NREC, COLS)\rho assign
[45] \a Exit if no records or no fields:
[46] L3: \rightarrow \times (NREC\times NFLD)\downarrow L43
[47] LAY\leftarrow FP[1]
[48] DISP\leftarrow FP[4]
[49] BLK\leftarrow FP[5]
[50] \a Read LV, RPS, ARPS if layer fld being assigned:
[51] \rightarrow \times (LAY\in FLDS)\downarrow L4
[52] LV\leftarrow DFREAD TIE, 9
[53] RPS\leftarrow ARPS\leftarrow DFREAD TIE, 8
[54] \rightarrow (=/FP[7 10])\rho L4
[55] ARPS\leftarrow DFREAD TIE, 10
[56] \a Determine distinct set numbers (deleting -1s):
[57] L4: UNQ\leftarrow SETS[\#SETS]
[58] NSET\leftarrow \rho UNQ\leftarrow (UNQ\neq 1\wedge 1, UNQ)/UNQ
[59] \a Last set index:
[60] SIND\leftarrow 0
[61] \a Loop by distinct set:
[62] L5: \rightarrow \times (SIND=NSET)\rho L43
[63] \a Current set index and number:
[64] SIND\leftarrow SIND+1
[65] SET\leftarrow UNQ[SIND]
[66] \a Displacement (no. components) before this set:
[67] SDISP\leftarrow DISP+INCR\times SET\rightarrow 1
[68] \a Indices and elts of INDS to retrieve in this set:
[69] IND\leftarrow (SET=SETS)/\times NREC
[70] I\leftarrow INDS[IND]
[71] \a Elts of INDS of recs whose layer value changes:
[72] BAD\leftarrow 0
[73] \a Last field index, and columns filed so far:
[74] F\leftarrow COL\rightarrow 0
[75] \a Loop by specified field:
[76] L6: \rightarrow \times (F=NFLD)\rho L3
[77] \a Current field index, number and width:
[78] F\leftarrow F+1
[79] FLD\leftarrow FLDS[F]
[80] \a Is this the layer field being changed?
[81] LAYER\leftarrow LAY=FLD
[82] W\leftarrow WID[FLD]
[83] \a Read data:
[84] DATA\leftarrow DFREAD CMP\leftarrow TIE, FLD+SDISP
\( V \) INDEXA (continued)

[85] a Branch if matrix field:
[86] \((W>1)\)pL9
[87] a Insert vector of data.
[88] a Branch if singleton:
[89] \(\rightarrow\)SINGLEpL7
[90] COL+COL+1
[91] DATA[I]+assign[IND;COL]
[92] \(\rightarrow\)L8
[93] L7:DATA[I]+assign
[94] a Branch unless layer field:
[95] L8: \(\rightarrow\)LAYER\(\rightarrow\)L12
[96] a Flag recs whose layer val has changed:
[97] BAD\((LV[SET]\#DATA[I])\)\(\rightarrow\)L12
[98] a Insert matrix of data.
[99] a Branch if singleton:
[100] L9: \(\rightarrow\)SINGLEpL10
[101] DATA[I;]\(\rightarrow\)assign[IND;COL+W]
[102] COL+COL+W
[103] \(\rightarrow\)L11
[104] L10:DATA[I;]\(\rightarrow\)assign
[105] a Branch unless layer field:
[106] L11: \(\rightarrow\)LAYER\(\rightarrow\)L12
[107] a Flag recs whose layer val has changed:
[108] BAD\((DATA[I;]\#LV[SET;])\)\(\rightarrow\)L12
[109] a Replace data on file:
[110] L12: DATA X FREPLACE CMP
[111] a Next set if layer values unchanged:
[112] L13: \(\rightarrow\)(XpBAD)\(\rightarrow\)L5
[113] a Indices of active fields:
[114] AFLD\((xWID)/tINCR
[115] a Exclude deletion field:
[116] NF\(\rightarrow\)AFLD\(\rightarrow\)(AFLD\#DEL)/AFLD
[117] a Flag records filed so far:
[118] FILED\((\rhoBAD)\)\(\rightarrow\)0
[119] a Look at layer field:
[120] LVAR\(\rightarrow\)OFREAD TIE,LAY+SDISP
[121] a Branch if vector field:
[122] VEC+WID[LAY]=1
[123] L14: \(\rightarrow\)VECpL15
[124] a Flag records with layer of 1st unfiled rec:
[125] L15: LAYER\(\rightarrow\)LVAR[BAD[FILED\(\rightarrow\)0];]
[126] a Good recs with layer of 1st unfiled rec:
[127] GOOD\(\rightarrow\)LVAR[BAD;]\(\rightarrow\)LAYER
[128] a ...and sets with this record:
[129] LSETS\(\rightarrow\)LV\(\rightarrow\)=LAYER
[130] a Update FILED; convert to indices:
[131] L16: FILED+FILEDvGOOD
[132] a GOOD=GOOD/BAD

-425-
\textbf{INDEXA (continued)}

[138] \texttt{NR\leftarrow pGOOD}

[139] \texttt{\textbf{a} Consider only non-full sets in this layer or}

[140] \texttt{\textbf{a} empty sets in any layer:}

[141] \texttt{LSETS\leftarrow (ARPS=0) v LSETS\#ARPS\#BLK}

[142] \texttt{\textbf{a} Convert to indices:}

[143] \texttt{LSETS\leftarrow LSETS/\ell pLSETS}

[144] \texttt{\textbf{a} No. records filed so far:}

[145] \texttt{FREC\leftarrow 0}

[146] \texttt{\textbf{a} Branch if no slots available in existing sets:}

[147] \texttt{L17:\rightarrow (BLK\leq MIN\leftarrow !A\leftarrow !ARPS[LSETS])\rho L32}

[148] \texttt{\textbf{a} Branch if more than 1 set needed:}

[149] \texttt{T\leftarrow NR\leftarrow FREC}

[150] \texttt{\rightarrow (T>BLK-MIN)\rho L18}

[151] \texttt{\textbf{a} Choose fullest set which will hold all recs:}

[152] \texttt{S\leftarrow BLK-A}

[153] \texttt{S\leftarrow LSETS[Sl/(S\geq T)/S]}

[154] \texttt{\rightarrow L19}

[155] \texttt{\textbf{a} Choose set with most empty slots:}

[156] \texttt{L18:S\leftarrow LSETS[Al\ell MIN]}

[157] \texttt{\textbf{a} No. records to be inserted within the set:}

[158] \texttt{L19:M\leftarrow T!RPS[S]\leftarrow ARPS[S]}

[159] \texttt{\textbf{a} No. records to be ccatenated within the set:}

[160] \texttt{N\leftarrow (T-M)\ell BLK-RPS[S]}

[161] \texttt{\textbf{a} Displacement (no. components) before this set:}

[162] \texttt{SD\leftarrow DISP+INCR\times S^{-1}}

[163] \texttt{\textbf{a} Branch if no deletion field:}

[164] \texttt{\rightarrow (\times DEL)\downarrow L20}

[165] \texttt{\textbf{a} Read deletion field for set S:}

[166] \texttt{BIT\leftarrow DFREAD DCMP\leftarrow TIE,DEL+SD}

[167] \texttt{\textbf{a} Branch if no records to be inserted:}

[168] \texttt{\rightarrow (\times M)\downarrow L20}

[169] \texttt{\textbf{a} Indices of available insertion slots:}

[170] \texttt{ININDS\leftarrow M_0(\sim BIT)/\ell pBIT}

[171] \texttt{\textbf{a} Next field index:}

[172] \texttt{L20:F\leftarrow 1}

[173] \texttt{\textbf{a} Loop by active field:}

[174] \texttt{L21:\rightarrow (F>NF)\rho L26}

[175] \texttt{\textbf{a} Field number:}

[176] \texttt{FLD\leftarrow AFLD[F]}

[177] \texttt{\textbf{a} Field width (no. columns):}

[178] \texttt{W\leftarrow WID[FLD]}

[179] \texttt{\textbf{a} Look at original set:}

[180] \texttt{VAR\leftarrow DFREAD TIE,FLD+SDISP}

[181] \texttt{\textbf{a} Read field F for set S:}

[182] \texttt{DATA\leftarrow DFREAD CMP\leftarrow TIE,FLD+SD}

[183] \texttt{\textbf{a} Branch if a matrix field:}

[184] \texttt{\rightarrow (W>1)\rho L23}

[185] \texttt{\textbf{a} Branch if no records to insert:}

[186] \texttt{\rightarrow (\times M)\downarrow L22}

[187] \texttt{DATA[ININDS]\leftarrow VAR[GOOD\leftarrow FREC+1M]}

[188] \texttt{\textbf{a} Branch if no records to ccatenate:}

[189] \texttt{\rightarrow (\times N)\downarrow L25}

[190] \texttt{L22:DATA\leftarrow DATA,VAR[GOOD\leftarrow (FREC+M)+1N]}
INDEXA (continued)

[191] → L25
[192] a Branch if no records to insert:
[193] L23:→(X* )L24
[194] DATA[IINDS;]←VAR[GOOD[FREC+1M];]
[195] a Branch if no records to catenate:
[196] → (X* )L25
[197] L24:DATA←DATA,[1]VAR[GOOD[(FREC+M)+1N];]
[198] L25:DATA ←FREPLACE CMP
[199] F←F+1
[200] → L21
[201] a Branch if no deletion field:
[202] L26: LEAD←1
[203] → (X*DEL) L29
[204] a Branch if no records to insert:
[205] → (X* )L27
[206] a Turn active record bits on:
[207] BIT[IINDS]←1
[208] LEAD←∧/BIT=¬\BIT
[209] a Branch if no records to catenate:
[210] → (X* )L28
[211] L27: BIT←BIT,Np1
[212] L28: BIT ←FREPLACE DCMP
[213] a Increment FREe by no. records added to this set:
[214] L29: FREC←FREC+M+N
[217] a Replace layer value if set initially empty:
[218] → T+L31
[219] a Branch if a vector layer field:
[220] → VECpL30
[221] LV[S;]+LAYER
[222] → L31
[223] L30:LV[S]+LAYER
[226] a Exit if all of data in field vars. filed:
[227] → (NR=FREC)pL40
[228] → L17
[229] a No. records to be appended in next set:
[230] L32:N←BLK\NR=FREC
[231] a Next field number:
[232] F←1
[233] a Loop by field:
[234] L33:+(F>INCR)pL39
[235] W←WID[F]
[236] a Branch unless a latent field:
[237] → (XW)pL34
[238] DATA←0
[239] → L38
[240] a Branch unless it's the deletion field:
[241] L34:+(DEL#F)pL35
[242] DATA←Np1
[243] → L37

-427-
\textbf{INDEXA (continued)}

\begin{verbatim}
[244] \textbf{a} Look at original set:
[245] \textbf{L35:} \texttt{VAR+DFREAD TIE,F+SDISP}
[246] \textbf{a} Branch if a matrix field:
[247] \rightarrow(W>1)\rightarrow L36
[248] \texttt{DATA+VAR\{GOOD\{FREC+\downarrow N\}\}}
[249] \rightarrow L37
[250] \texttt{L36:} \texttt{DATA+VAR\{GOOD\{FREC+\downarrow N\};\}}
[251] \textbf{a} Branch unless set must be padded to BLK records:
[252] \texttt{L37:} \rightarrow FILL \rightarrow L38
[253] \texttt{DATA+\{BLK,1\downarrow pDATA\}+DATA}
[254] \texttt{L38:} \texttt{T+DATA DFAPPEND TIE}
[255] \texttt{F+\downarrow F+1}
[256] \rightarrow L33
[257] \textbf{a} Increment FREC by no. records added to this set:
[258] \texttt{L39:} \texttt{FREC+\downarrow FREC+N}
[259] \texttt{ARPS+ARPS,N}
[260] \texttt{RPS+RPS,T+\uparrow N+BLK+\downarrow FILL}
[264] \textbf{a} Catenate layer value:
[265] \texttt{LV+LV,\{1\} LAYER}
[266] \textbf{a} Continue unless all of data in field vars. filed:
[267] \rightarrow (\uparrow (\downarrow FREC)) \rightarrow L32
[268] \textbf{a} Branch if no data left to file:
[269] \texttt{L40:} \rightarrow (\uparrow (\downarrow FILED)) \rightarrow L14
[270] \textbf{a}
[271] \textbf{a} Branch unless deletion field exists:
[272] \rightarrow (\times \downarrow \texttt{DEL}) \rightarrow L41
[273] \textbf{a} Read deletion field for set \texttt{SET}:
[274] \texttt{BIT+OFREAD CMP+TIE,DEL+SDISP}
[275] \textbf{a} Turn off specified indices and replace:
[276] \texttt{BIT[BAD]+0}
[277] \texttt{BIT+DFREPLACE CMP}
[278] \textbf{a} Compute new no. records:
[279] \texttt{N+\downarrow N/BIT}
[280] \texttt{M+\downarrow RPS[SET]}
[281] \textbf{a} Reset parameters:
[283] \texttt{ARPS[SET]+N+(\texttt{-1} 1)[1+N=+/\downarrow N/BIT]}
[284] \rightarrow L5
[285] \textbf{a}
[286] \textbf{a} Turn off specified indices:
[287] \texttt{L41:} \texttt{M+RPS[SET]}
[288] \texttt{BIT+M1}
[289] \texttt{BIT[BAD]+0}
[290] \textbf{a} Compute new no. records:
[291] \texttt{N+\downarrow N/BIT}
[292] \textbf{a} Reset parameters:
[295] \texttt{RPS[SET]+N}
\end{verbatim}

-428-
INDEXA (continued)

v INDEXA (continued)

[296] a Last field index:
[297] F<0
[298] a Loop by active field:
[299] L42:=(F=NF)\p L5
[300] a Current field index and number:
[301] F\p F+1
[302] FLD\p AFLD[F]
[303] a Read, compress, replace field FLD for set SET:
[304] CMP\p TIE,FLD\p SDISP
[305] (BIT\p OFREAD CMP)\p DREPLACE CMP
[306] \p L42
[307] a
[308] a Erase <assign> and exit:
[309] L43:F\p DEX 'assign'
[310] a Replace LV, RPS, ARPS if layer field assigned:
[311] \p (LAY\p FLDS)\p 0
[312] LV \p DREPLACE TIE,9
[313] NFP \p DREPLACE TIE,7
[314] RPS \p DREPLACE TIE,8
[315] \p (xDEL)\p 0
[316] ARPS \p DREPLACE TIE,10

\n
[WSID: MULTIFLO]

\n
v NFP\p INDS INDEXWSA FP:A;AFLD;ARPS;BAD;BIT;BLK;CMP;DATA;
DCMP;DEL;DISP;F;FILED;FLD;FLDS;FREC;FT;GOOD;I;
IINDS;INCR;IND;LAY;LEAD;LSETS;LV;LVAR;M;MIN;N;NF
;NFLD;NR;NREC;NSET;RPS;S;SD;SDISP;SET;SETS;SHAPE;SIND;
T;TIE;UNQ;VAR;VEC;W;WID;\p 10

[1] a Inserts data from global field variables (e.g. F3,
[2] a F5,... for fields 3, 5,...) into fields 11\p FP for
[3] a records identified by file indices matrix INDS.
[4] a First row of INDS is set number (origin 1);
[5] a second row is index (origin 1) within set.
[6] a INDS may be of any dimension as long as its
[7] a first coordinate is 2. Global field variables
[8] a have same number of columns as corresponding
[9] a fields. Leading shape is 1\p INDS. Global field
[10] a variables are erased upon completion.
[11] a ERROR(\p FP)'RANK ERROR'
[12] a ERROR((11>\p FP)\p 2\p \p 1\p INDS)'LENGTH ERROR'
[13] a Exit if no fields:
[14] NFLD\p FLDS+11\p FP
[15] NFP\p 11\p FP
[16] \p (xNFLD)\p 0
[17] \p 1
[18] INCR\p FP[3]
[21] a ERROR((\p FLDS\p INCR)<DEL\p FLDS)'INVALID FIELD NUMBER'
[22] TIE\p FP[2]
[23] FT\p OFREAD TIE,4
INDEXWSA (continued)

24] A Width (no. columns) of specified fields:
25] WID+FT[1;]
26] OERROR(0eWID[FLDS])/'INACTIVE FIELD'
27] A Shape of field variables (excluding columns):
28] SHAPE+10INDS
29] A Last field index:
30] F+0
31] A Loop by specified field to verify field vars
32] A (bypass this loop to make the function faster
33] A and to live dangerously):
34] L1:F>=NFLD)FL3
35] A Current field index, number and width:
36] F=F+1
37] FLD+FLDS[F]
38] W=WID[FLD]
39] A Look at field variable:
40] S=pDATA,="F",pFLD
41] OLERROR((2#FT[2;FLD])#0=10pDATA)/'DOMAIN ERROR'
42] A Done if singleton data:
43] (1\.=S)FL1
44] A Branch unless 'singleton' record:
45] ((W#1)\=\=1\LST)FL2
46] OERROR(W#1\LST)/'LENGTH ERROR'
47] FL1
48] L2:OERROR((\=SHAPE)\.\.(\=S)+\-1,\(W=1)\L)
49] N=(1+\=SHAPE)\=S,1
50] OERROR(N\.\.=SHAPE,W)/'LENGTH ERROR'
51] FL1
52] A Break apart indices:
53] L3:NREC=SETS\-,10 \INDS
54] INDS\-,01 \INDS
55] A Exit if no indices:
56] (\XNREC)\L43
57] LAY=FP[1]
58] DISP=FP[4]
59] BLK=FP[5]
60] A Read LV, RPS, ARPS if layer fld being assigned:
61] LAY=DFREAD TIE,9
62] RPS=DFREAD TIE,8
63] ARPS=DFREAD TIE,10
64] (=/FP[10])FL4
65] L4:UNQ=SETS\[\XSETS]
66] NSET=UNQ=(UNQ\-1\-1,UNQ)/UNQ
67] A Last set index:
68] SIND=0
69] A Loop by distinct set:
70] L5:=(SIND=NSET)FL43
71] A Current set index and number:
72] SIND=SIND+1
73] SET=UNQ[SIND]
INDEXWSA (continued)

[76] A Displacement (no. components) before this set:
[77] S\text{DISP}=\text{DISP}+\text{INCR} \times \text{SET} + 1
[78] A Indices of INDS to retrieve in this set:
[79] IND<\text{(SET}=\text{SETS})/\text{\text{nREC}
[80] I+INDS[IND]
[81] A Elts of INDS of recs whose layer value changes:
[82] BAD<1
[83] A Last field index:
[84] F=0
[85] A Loop by specified field:
[86] L6:(F=NFLD) \rightarrow L13
[87] A Current field index, number and width:
[88] F=F+1
[89] FLD=FLDS[F]
[90] A Is this the layer field being changed?
[91] LAYER=LAY=FLD
[92] W=WID[FLD]
[93] A Read data:
[94] DATA \leftrightarrow \text{DFREAD CMP} \leftrightarrow TIE,FLD+SDISP
[95] A Current field var:
[96] VAR<\#F',\#FLD
[97] A Continue if singleton:
[98] \rightarrow (1<\text{\text{=VAR}) \rightarrow L10
[99] A Branch if 'singleton' record:
[100] \rightarrow ((W\neq 1)^1 \land .{=1<\text{\text{VAR}) \rightarrow L7
[101] A Branch if rank OK already:
[102] \rightarrow ((\text{VAR})=2<\text{\text{W}) \rightarrow L8
[103] L7:VAR<\text{(NREC},(W>1)<\text{\text{W}) \rightarrow VAR
[104] A Branch if matrix:
[105] L8:(W>1) \rightarrow L9
[106] VAR<\text{VAR[IND]}
[107] \rightarrow L10
[108] L9:VAR<\text{VAR[IND;]}
[109] A Branch if matrix:
[110] L10:(W>1) \rightarrow L11
[111] A Insert data:
[112] DATA[I]=VAR
[113] A Branch unless layer field:
[114] \rightarrow \text{LAY} \downarrow L12
[115] A Flag recs whose layer val has changed:
[116] BAD<\text{LV[SET]} \neq \text{VAR})/I
[117] \rightarrow L12
[118] L11:DATA[I;]=VAR
[119] A Branch unless layer field:
[120] \rightarrow \text{LAY} \downarrow L12
[121] A Flag recs whose layer val has changed:
[122] BAD<\text{VAR} \neq \text{LV[SET;]})/I
[123] A Replace data on file:
[124] L12:DATA \leftrightarrow \text{DFREPLACE CMP}
[125] \rightarrow L6
[126] A Next set if layer values unchanged:
[127] L13:(\times BAD) \rightarrow L5
INDEXWSA (continued)

[128] a Indices of active fields:
[129] AFLD+(×WID)/LINCR
[130] a Exclude deletion field:
[131] NF+rAFLD+(AFLD#DEL)/AFLD
[132] a Flag records filed so far:
[133] FILED+(BAD)r0
[134] a Look at layer field:
[135] LVAR+DFREAD TIE, LAY+SDISP
[136] a Branch if vector field:
[137] VEC+LWID(LAY)=1
[138] L14:+VECpL15
[139] a Flag records with layer of 1st unfiled rec:
[140] LAYER+LVAR[BAD[FILED]0];
[141] GOOD+LVAR[BAD];=LAYER
[142] a ... and sets with this record:
[143] LSETS+LVAR=到底
[144] →L16
[145] L15:LAYER=LVAR[BAD[FILED]0]]
[146] GOOD+LVAR[BAD]=LAYER
[147] LSETS+LVAR=LAY
[148] a Update FILED; convert to indices:
[149] L16:FILED+FILED\GOOD
[150] GOOD+GOOD/BAD
[151] NR+rGOOD
[152] a Consider only non-full sets in this layer or
[153] a empty sets in any layer:
[154] LSETS+(ARPS=0)vLSETS\ARPS\BLK
[155] a Convert to indices:
[156] LSETS+LSETS\LSETS
[157] a No. records filed so far:
[158] FREC+0
[159] a Branch if no slots available in existing sets:
[160] L17:+(BLK=MIN\l/A\ARPS[LSETS])rL32
[161] a Branch if more than 1 set needed:
[162] T+NR+FREC
[163] →(T>BLK-MIN)rL18
[164] a Choose fullest set which will hold all recs:
[165] S+BLK-A
[166] S+LSETS[S\l/(S=T)/S]
[167] →L19
[168] a Choose set with most empty slots:
[169] L18:S+LSETS[A\MIN]
[170] a No. records to be inserted within the set:
[171] L19:M+T[ARPS[S]\ARPS[S]
[172] a No. records to be catenated within the set:
[173] N+(T-M)\BLK-RPS[S]
[174] a Displacement (no. components) before this set:
[175] SD+DISP+INCR\S-1
[176] a Branch if no deletion field:
[177] →(×DEL)rL20
[178] a Read deletion field for set S:
[179] BIT+DFREAD DCMP\TIE, DEL+SD
INDEXWSA (continued)

180] a Branch if no records to be inserted:
181] \( \rightarrow (\times M) \) \( \rightarrow L20 \)
182] a Indices of available insertion slots:
183] \( IINDS \leftarrow M_p(\sim BIT)/p \) BIT
184] a Next field index:
185] L20: F = 1
186] a Loop by active field:
187] L21: \( \rightarrow (F>NF)p \) L26
188] a Field number:
189] FLD \( \rightarrow AFLD[F] \)
190] a Field width (no. columns):
191] W \( \leftrightarrow \) WID[FLD]
192] a Look at original set:
193] VAR \( \leftarrow \) DFREAD TIE, FLD + SDISP
194] a Read field F for set S:
195] DATA \( \leftarrow \) DFREAD CMP \( \leftarrow \) TIE, FLD + SD
196] a Branch if a matrix field:
197] \( \rightarrow \) (W > 1)p L23
198] a Branch if no records to insert:
199] \( \rightarrow (\times M) \) \( \rightarrow L22 \)
200] DATA[IINDS] \( \leftarrow \) VAR[GOOD[FREC + M]]
201] a Branch if no records to catenate:
202] \( \rightarrow (\times N) \) \( \rightarrow L25 \)
203] L22: DATA \( \leftarrow \) DATA, VAR[GOOD[(FREC + M) + N]]
204] \( \rightarrow L25 \)
205] a Branch if no records to insert:
206] L23: \( \rightarrow (\times M) \) \( \rightarrow L24 \)
207] DATA[IINDS]; \( \leftarrow \) VAR[GOOD[(FREC + M)];]
208] a Branch if no records to catenate:
209] \( \rightarrow (\times N) \) \( \rightarrow L25 \)
210] L24: DATA \( \leftarrow \) DATA, [1]VAR[GOOD[(FREC + M) + N];]
211] L25: DATA \( \leftarrow \) DFPLACE CMP
212] F \( \leftarrow \) F + 1
213] \( \rightarrow L21 \)
214] a Branch if no deletion field:
215] L26: LEAD \( \leftarrow +1 \)
216] \( \rightarrow (\times D E L) \) \( \rightarrow L29 \)
217] a Branch if no records to insert:
218] \( \rightarrow (\times M) \) \( \rightarrow L27 \)
219] a Turn active record bits on:
220] BIT[IINDS] \( \leftarrow +1 \)
221] LEAD \( \leftarrow \) \( ^{\wedge} \) BIT \( = ^{\wedge} \) BIT
222] a Branch if no records to catenate:
223] \( \rightarrow (\times N) \) \( \rightarrow L28 \)
224] L27: BIT \( \leftarrow BIT, N + 1 \)
225] L28: BIT \( \leftarrow \) DFPLACE DCMP
226] a Increment FREC by no. records added to this set:
227] L29: FREC \( \leftarrow \) FREC + M + N
228] RPS[S] \( \leftarrow \) RPS[S] + N
229] NFP[S] \( \leftarrow \) NF[S] + N \( = 0 \) ARPS[S]
230] a Replace layer value if set initially empty:
231] \( \rightarrow T \) \( \rightarrow L31 \)
INDEXWSA (continued)

1 Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:

Branch if a vector layer field:
\( V \) INDEXWSA (continued)

[284] \( \rightarrow (x\text{DEL}) \downarrow \text{L41} \)

[285] \( \rightarrow (x\text{DEL}) \downarrow \text{L41} \)

[286] \( \rightarrow \)

[287] \( \rightarrow \)

[288] \( \rightarrow \)

[289] \( \rightarrow \)

[290] \( \rightarrow \)

[291] \( \rightarrow \)

[292] \( \rightarrow \)

[293] \( \rightarrow \)

[294] \( \rightarrow \)

[295] \( \rightarrow \)

[296] \( \rightarrow \)

[297] \( \rightarrow \)

[298] \( \rightarrow \)

[299] \( \rightarrow \)

[300] \( \rightarrow \)

[301] \( \rightarrow \)

[302] \( \rightarrow \)

[303] \( \rightarrow \)

[304] \( \rightarrow \)

[305] \( \rightarrow \)

[306] \( \rightarrow \)

[307] \( \rightarrow \)

[308] \( \rightarrow \)

[309] \( \rightarrow \)

[310] \( \rightarrow \)

[311] \( \rightarrow \)

[312] \( \rightarrow \)

[313] \( \rightarrow \)

[314] \( \rightarrow \)

[315] \( \rightarrow \)

[316] \( \rightarrow \)

[317] \( \rightarrow \)

[318] \( \rightarrow \)

[319] \( \rightarrow \)

[320] \( \rightarrow \)

[321] \( \rightarrow \)

[322] \( \rightarrow \)

[323] \( \rightarrow \)

[324] \( \rightarrow \)

[325] \( \rightarrow \)

[326] \( \rightarrow \)

[327] \( \rightarrow \)

[328] \( \rightarrow \)

[329] \( \rightarrow \)

[330] \( \rightarrow \)
Chapter 14 Solutions  FILE DESIGN AND UTILITIES

[WSID: MULTIFLOJ]

\[ R \rightarrow A \text{ FOR } B \]

[1] Catenates arguments together, separating by a newline character. Used as:
[2] (FP, \[7 7 2 4 0 3\]) EXECUTE \( F7 \leftarrow F7 \uplus F2 \div F4 \) FOR \('F3>1'\)

[WSID: MULTIFLOJ]

\[ R \rightarrow A \text{ FOR } B \]

\[ \Delta A N F P \leftarrow \Delta A F P \text{ EXECUTE } \Delta A E X P R ; \Delta A A ; \Delta A A F L D S ; \Delta A A L L ; \Delta A A R P S ; \Delta A A B A D ; \Delta A B I T ; \Delta A B L K ; \Delta A C M P ; \Delta A D A T A ; \Delta A D C M P ; \Delta A D E L ; \Delta A D I S P ; \Delta A F ; \Delta A F I L E D ; \Delta A F I L L ; \Delta A F L D S ; \Delta A F N 1 ; \Delta A F N 2 ; \Delta A F N 3 ; \Delta A F R E C ; \Delta A F T ; \Delta A G O O D ; \Delta A I N D S ; \Delta A I N C R ; \Delta A I N D ; \Delta A I N D S ; \Delta A I N D S ; \Delta A I N D S ; \Delta A I N D S ; \Delta A L A B ; \Delta A L A Y ; \Delta A L A Y E R ; \Delta A L E A D ; \Delta A L S E T S ; \Delta A L V ; \Delta A L V A R ; \Delta A M ; \Delta A M I N ; \Delta A N ; \Delta A N A F L D ; \Delta A N D E L ; \Delta A N F ; \Delta A N F L D ; \Delta A N R ; \Delta A N S E T ; \Delta A N S F L D ; \Delta A R P S ; \Delta A S ; \Delta A S D ; \Delta A S D I S P ; \Delta A S E L ; \Delta A S E T ; \Delta A S E T S ; \Delta A S F L D S ; \Delta A S I N D ; \Delta A T ; \Delta A T I E ; \Delta A V A R ; \Delta A V E C ; \Delta A W ; \Delta A W I D ; \Delta A X E Q ; \Delta I O \]

[1] Used as: (FP, \[7 7 2 4\]) EXECUTE \( F7 \leftarrow F7 \uplus F2 \div F4 \) or:
[2] (FP, \[7 7 2 4 0 3\]) EXECUTE \( F7 \leftarrow F7 \uplus F2 \div F4 \) FOR \('F3>1'\)
[3] Loops through active sets doing the following:
[4] reads the fields specified beyond the 0 in the left argument (calling them F3, F5, etc. for fields 3, 5, etc.); executes the character vector expression beyond the newline character (inserted by FOR) in the right argument; retrieves data from fields specified before the 0 (and positive) in the left argument (calling them F7, F2, etc. for fields 7, 2, etc.) for the records of that set which correspond to 1s in the resulting bit vector (or all active records if no selection expression provided); executes the character vector expression before the newline character in the right argument; replaces on file data for fields specified before the 0 (and negative) in the left argument (assuming their existence in variables F7, F8, etc. for fields \(-7, -8\), etc.) for the records selected. All "Fn" variables are erased upon completion. Note that EXPR is executed in origin 1; e.g. \('F3[;2]' always refers to 2nd column.

[24] \[ \square \text{ERROR}(1 \neq \rho \Delta A F P) \lor 1 < \rho \Delta A E X P R) / 'RANK ERROR' \]
[25] \[ \square \text{ERROR}(0 = 1 \lor \rho \Delta A E X P R) / 'DOMAIN ERROR' \]
[26] \[ \square \text{IO} \leftarrow 1 \]
[27] Extract 3 sets of fields from FP:

\[ \Delta A T \leftarrow 11 + (11 \Delta A F P) \lor 0 \]
\[ \Delta A F L D S \leftarrow 11 \downarrow (\Delta A T - 1) \rho \Delta A F P \]
\[ \Delta A S F L D S \leftarrow \Delta A T \downarrow \Delta A F P \]
\[ \Delta A A F L D S \leftarrow \left( \Delta A T \Delta A F L D S < 0 \right) / \Delta A F L D S \]
\[ \Delta A F L D S \leftarrow (\sim \Delta A T) / \Delta A F L D S \]

[33] Extract 2 expressions from EXPR:

\[ \Delta A E X P R \leftarrow \Delta A E X P R \]
\[ \Delta A T \leftarrow \Delta A E X P R \leftarrow \text{UTCNL} \]

[36] \[ \Delta A X E Q \leftarrow (\sim \Delta A T - 1) \rho \Delta A E X P R \]

-436-
EXECUTE (continued)

[37] \( \Delta \Delta EXPR + \Delta \Delta T \downarrow \Delta \Delta EXPR \)

[38] \( \Omega \text{ERROR}((0=\rho \Delta AFLDS) \vee \langle 11=\rho \Delta \text{FP} \rangle \vee (' ^{\cdot,} = \Delta \Delta EXPR) \neq 0=\rho \Delta \Delta SFLDS) / ' \text{LENGTH ERROR}' \)

[39] \( \Delta \Delta \text{INCR} + \Delta \Delta \text{FP}[3] \)

[40] \( \Delta \Delta \text{DEL} \downarrow / \Delta \Delta \text{FP}[11] \)

[41] \( \Delta \Delta \text{FILL} + \Delta \Delta \text{FP}[11] < 0 \)

[42] \( \Delta \Delta T \downarrow \Delta AFLDS, \Delta AFLDS, \Delta \Delta SFLDS \)

[43] \( \Omega \text{ERROR}((/ \Delta \Delta T \in \Delta \Delta \text{INCR}) \leq \Delta \Delta \text{DEL} \in \Delta \Delta T) / ' \text{INVALID FIELD NUMBER}' \)

[44] \( \Delta \Delta \text{TIE} + \Delta \Delta \text{FP}[2] \)

[45] \( \Delta \Delta \text{FT} \leftarrow \text{OFREAD} \Delta \Delta \text{TIE}, 4 \)

[46] \( \text{A Width (no. columns) of fields:} \)

[47] \( \Delta \Delta \text{WID} \leftarrow \Delta \Delta \text{FT}[1]; \)

[48] \( \Omega \text{ERROR}(0 \in \Delta \Delta \text{WID}[\Delta \Delta T]) / ' \text{INACTIVE FIELD}' \)

[49] \( \text{A Numbers of sets with records in wrong layer:} \)

[50] \( \Delta \Delta \text{BAD} \leftarrow 10 \)

[51] \( \text{A Exit if no records:} \)

[52] \( \Delta \Delta \text{NF} \downarrow 11 \rho \Delta \Delta \text{FP} \)

[53] \( \rightarrow (\times \Delta \Delta \text{FP}[10]) \downarrow \Delta \Delta \text{L} 29 \)

[54] \( \text{A Make field vectors distinct:} \)

[55] \( \Delta \Delta \text{T} \leftarrow \Delta \Delta \text{INCR} \rho 0 \)

[56] \( \Delta \Delta \text{T}[\Delta \Delta \text{SFLDS}] \leftarrow 1 \)

[57] \( \Delta \Delta \text{NSFLD} \leftarrow / \Delta \Delta \text{SFLDS} + \Delta \Delta \text{T} / \rho \Delta \Delta \text{T} \)

[58] \( \Delta \Delta \text{T}[\bot] \leftarrow 0 \)

[59] \( \Delta \Delta \text{T}[\Delta \Delta \text{AFLDS}] \leftarrow 1 \)

[60] \( \Delta \Delta \text{NAFLD} \leftarrow \rho \Delta \Delta \text{AFLDS} + \Delta \Delta \text{T} / \rho \Delta \Delta \text{T} \)

[61] \( \Delta \Delta \text{T}[\bot] \leftarrow 0 \)

[62] \( \Delta \Delta \text{T}[\Delta \Delta \text{FLDS}] \leftarrow 1 \)

[63] \( \Delta \Delta \text{NF} \leftarrow / \Delta \Delta \text{AFLDS} + \Delta \Delta \text{T} / \rho \Delta \Delta \text{T} \)

[64] \( \text{A Names of fields to erase:} \)

[65] \( \Delta \Delta \text{FN} 1 \leftarrow (\sim \Delta \Delta \text{SFLDS} \in \Delta \Delta \text{FLDS}) / \Delta \Delta \text{SFLDS} \)

[66] \( \Delta \Delta \text{FN} 1 \leftarrow (\rho \Delta \Delta \text{FN} 1), 1) \rho \Delta \Delta \text{FN} 1 \)

[67] \( \Delta \Delta \text{FN} 1 \leftarrow ' F', (+/' = \Delta \Delta \text{FN} 1) \Phi \Delta \Delta \text{FN} 1 \)

[68] \( \Delta \Delta \text{FN} 2 \leftarrow (\sim \Delta \Delta \text{SFLDS} \in \Delta \Delta \text{FLDS}) / \Delta \Delta \text{FLDS} \)

[69] \( \Delta \Delta \text{FN} 2 \leftarrow (\rho \Delta \Delta \text{FN} 2), 1) \rho \Delta \Delta \text{FN} 2 \)

[70] \( \Delta \Delta \text{FN} 2 \leftarrow ' F', (+/' = \Delta \Delta \text{FN} 2) \Phi \Delta \Delta \text{FN} 2 \)

[71] \( \Delta \Delta \text{FN} 3 \leftarrow (\Delta \Delta \text{NAFLD}, 1) \rho \Delta \Delta \text{AFLDS} \)

[72] \( \Delta \Delta \text{FN} 3 \leftarrow ' F', (+/' = \Delta \Delta \text{FN} 3) \Phi \Delta \Delta \text{FN} 3 \)

[73] \( \text{A Label vector needed below based upon whether} \)

[74] \( \text{a field in ws or on file when needed (file,ws):} \)

[75] \( \Delta \Delta \text{LAB} \leftarrow (\Delta \Delta \text{L} 17, \Delta \Delta \text{L} 19)[1 + \Delta \Delta \text{AFLDS} \in \Delta \Delta \text{SFLDS}] \)

[76] \( \Delta \Delta \text{LAY} \leftarrow \Delta \Delta \text{FP}[1] \)

[77] \( \Delta \Delta \text{DISP} \leftarrow \Delta \Delta \text{FP}[4] \)

[78] \( \Delta \Delta \text{BLK} \leftarrow \Delta \Delta \text{FP}[5] \)

[79] \( \Delta \Delta \text{ARPS} \leftarrow \Delta \Delta \text{ARPS} \leftarrow \text{OFREAD} \Delta \Delta \text{TIE}, 8 \)

[80] \( \text{A Branch unless ARPS should be read:} \)

[81] \( \rightarrow (\sim / \Delta \Delta \text{FP}[7 \ 10]) \rho \Delta \Delta \text{L} 1 \)

[82] \( \Delta \Delta \text{ARPS} \leftarrow \text{OFREAD} \Delta \Delta \text{TIE}, 10 \)

[83] \( \text{A Consider only nonempty sets:} \)

[84] \( \Delta \Delta \text{L} 1 \leftarrow / \Delta \Delta \text{SETS} \leftarrow 0 \# / \Delta \Delta \text{ARPS} \)

[85] \( \text{A ... and sets specified in <layers>:} \)

[86] \( \rightarrow (\times \Delta \Delta \text{LAY}) \downarrow \Delta \Delta \text{L} 5 \)

[87] \( \Delta \Delta \text{T} \leftarrow \text{INC} ' \text{layers}' \)
EXECUTE (continued)

\[(\Delta T \neq \Delta LAY \in \Delta AFLDS) \lor \Delta L5\]

\[\Delta LV \leftarrow \text{DFREAD} \Delta TIE, 9\]

\[\Delta T \leftarrow \Delta L5\]

\[\text{a Branch if matrix layers field:}\]

\[\Delta WID[\Delta LAY > 1] \lor \Delta L2\]

\[\Delta SETS \leftarrow \Delta SETS \lor \Delta LV \in \text{layers}\]

\[\Delta L5\]

\[\text{a Convert <layers> to matrix if not already:}\]

\[\Delta L2: \text{IFERROR}(\Delta FT[1; \Delta LAY] \neq 1 \lor \text{players}) / '\text{LENGTH ERROR'}\]

\[(2 = \#\text{players}) \lor \Delta L3\]

\[\text{layers} \leftarrow (x / \#\text{players}), \#\text{layers} \times \text{players}\]

\[\text{a Branch if numeric matrix field:}\]

\[\Delta L3: (0 = \#\text{players}) \lor \Delta L4\]

\[\Delta SETS = \Delta SETS \lor (\text{layers} \times \text{CMIOTA} \Delta LV \leq \#\text{players}\]

\[\Delta L5\]

\[\Delta L4: \Delta SETS = \Delta SETS \lor \Delta LV \leq \#\text{layers}\]

\[\text{a Convert to indices; erase <layers>:}\]

\[\Delta L5: \Delta SETS = \Delta SETS / \#\Delta SETS\]

\[\Delta NSET \leftarrow \#\Delta SETS\]

\[\Delta T \leftarrow \text{OEX} '\text{layers}'\]

\[\text{a Last set index:}\]

\[\Delta S + 0\]

\[\text{a Loop by nonempty set:}\]

\[\Delta L6: (\Delta S = \Delta NSET) \lor \Delta L29\]

\[\text{a Current set index and number:}\]

\[\Delta S + \Delta S + 1\]

\[\Delta SETS = \Delta SETS[\Delta S]\]

\[\text{a Displacement (no. components) before this set:}\]

\[\Delta S\text{DISP} = \Delta S\text{DISP} + \Delta INCR \times \Delta SET - 1\]

\[\text{a Branch if deletion field unneeded:}\]

\[\Delta NDEL = \#\Delta ARPS[\Delta SET] \lor \Delta L7\]

\[\text{a Read deletion field for set SET:}\]

\[\Delta BIT \leftarrow \text{DFREAD} \Delta TIE, \Delta DEL + \Delta SDISP\]

\[\text{a Are all records in this set active:}\]

\[\Delta L7: \Delta ALL = \#\Delta ARPS[\Delta SET] \lor \#\Delta ARPS[\Delta SET]\]

\[\text{a Branch if a selection expression:}\]

\[\Delta L8 = \#\Delta NSFLD \lor \Delta L9\]

\[\text{a Branch if all records active:}\]

\[\Delta L8 \lor \Delta L9\]

\[\text{a Branch if deletion field unneeded:}\]

\[\Delta NDEL \lor \Delta L8\]

\[\text{a Indices of active records:}\]

\[\Delta INDS = \#\Delta BIT \lor \Delta BIT\]

\[\Delta L15\]

\[\Delta L8 = \#\Delta INDS \lor \#\Delta ARPS[\Delta SET]\]

\[\Delta L15\]

\[\text{a Last field index:}\]

\[\Delta L9 = \#\Delta F \leftarrow 0\]

\[\text{a Loop by selection field:}\]

\[\Delta L0 = (\Delta F \neq \#\Delta NSFLD) \lor \Delta L13\]

\[\text{a Current field index and number:}\]

\[\Delta F \leftarrow \Delta F + 1\]
EXECUTE (continued)

ΔΔFLD→ΔΔSFLDS[ΔΔF]

A Read field FLD for set SET:

ΔΔDATA←UFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

A Branch if all records in set active:

ΔΔALLpΔΔL12

A Branch if deletion field unneeded:

ΔΔNDELpΔΔL11

A Apply deletion field:

ΔΔDATA←ΔΔBIT/ΔΔDATA

ΔΔL12

A Read field FLD for set SET:

ΔΔDATA←UFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

A Branch if all records in set active:

ΔΔALLpΔΔL12

A Branch if deletion field unneeded:

ΔΔNDELpΔΔL11

A Apply deletion field:

ΔΔDATA←ΔΔBIT/ΔΔDATA

ΔΔL12

A Read field FLD for set SET:

ΔΔDATA←UFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

A Branch if all records in set active:

ΔΔALLpΔΔL12

A Branch if deletion field unneeded:

ΔΔNDELpΔΔL11

A Apply deletion field:

ΔΔDATA←ΔΔBIT/ΔΔDATA

ΔΔL12

A Read field FLD for set SET:

ΔΔDATA←UFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

A Branch if all records in set active:

ΔΔALLpΔΔL12

A Branch if deletion field unneeded:

ΔΔNDELpΔΔL11

A Apply deletion field:

ΔΔDATA←ΔΔBIT/ΔΔDATA

ΔΔL12

A Read field FLD for set SET:

ΔΔDATA←UFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

A Branch if all records in set active:

ΔΔALLpΔΔL12

A Branch if deletion field unneeded:

ΔΔNDELpΔΔL11

A Apply deletion field:

ΔΔDATA←ΔΔBIT/ΔΔDATA

ΔΔL12

A Read field FLD for set SET:

ΔΔDATA←UFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

A Branch if all records in set active:

ΔΔALLpΔΔL12

A Branch if deletion field unneeded:

ΔΔNDELpΔΔL11

A Apply deletion field:

ΔΔDATA←ΔΔBIT/ΔΔDATA

ΔΔL12

A Read field FLD for set SET:

ΔΔDATA←UFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

A Branch if all records in set active:

ΔΔALLpΔΔL12

A Branch if deletion field unneeded:

ΔΔNDELpΔΔL11

A Apply deletion field:

ΔΔDATA←ΔΔBIT/ΔΔDATA

ΔΔL12

A Read field FLD for set SET:

ΔΔDATA←UFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

A Branch if all records in set active:

ΔΔALLpΔΔL12

A Branch if deletion field unneeded:

ΔΔNDELpΔΔL11

A Apply deletion field:

ΔΔDATA←ΔΔBIT/ΔΔDATA

ΔΔL12

A Read field FLD for set SET:

ΔΔDATA←UFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

A Branch if all records in set active:

ΔΔALLpΔΔL12

A Branch if deletion field unneeded:

ΔΔNDELpΔΔL11

A Apply deletion field:

ΔΔDATA←ΔΔBIT/ΔΔDATA

ΔΔL12

A Read field FLD for set SET:

ΔΔDATA←UFREAD ΔΔTIE,ΔΔFLD+ΔΔSDISP

A Branch if all records in set active:

ΔΔALLpΔΔL12

A Branch if deletion field unneeded:
EXECUTE (continued)
[194] \(\Delta\Delta\text{DATA} \Rightarrow 'F', \delta\Delta\FLD\)
[195] \(\text{\textcircled{v}}\) Branch if matrix field:
[196] \(\rightarrow (\Delta\text{WID}[\Delta\FLD] > 1) \rho \Delta\AL\)
[197] \(\Delta\Delta\text{DATA} \Rightarrow \Delta\Delta\text{DATA}[\Delta\IND] \)
[198] \(\rightarrow \Delta\AL\)
[199] \(\Delta\AL\)\(\text{\textcircled{v}}\) \(\Delta\Delta\text{DATA} \Rightarrow \Delta\Delta\text{DATA}[\Delta\IND] \)
[200] \(\text{\textcircled{v}}\) Assign data to global variable \(\text{Fn}:\)
[201] \(\Delta\Delta\text{DATA} \Rightarrow 'F', (\neq \Delta\FLD), '\Delta\text{DATA}' \)
[202] \(\rightarrow \Delta\AL\)
[203] \(\text{\textcircled{v}}\) Once all execution fields read, execute XEQ:
[204] \(\Delta\AL\)\(\text{\textcircled{v}}\) \(\Delta\text{XE}\)
[205] \(\text{\textcircled{v}}\) Erase fields in FLDS and not in AFLDS:
[206] \(\Delta\T \Rightarrow \text{EX} \ \Delta\FN2\)
[207] \(\text{\textcircled{v}}\) Last field index:
[208] \(\Delta\F \Rightarrow 0\)
[209] \(\text{\textcircled{v}}\) Loop by assignment field:
[210] \(\Delta\AL\)\(\text{\textcircled{v}}\) \(\Delta\text{AFLD} \Rightarrow \Delta\text{AFLDS}[\Delta\F] \)
[211] \(\Delta\FLD \Rightarrow \Delta\text{WID}[\Delta\FLD] \)
[212] \(\Delta\COMP \Rightarrow \Delta\text{WIE}, \Delta\FLD + \Delta\DISP \)
[213] \(\Delta\text{DATA} \Rightarrow 'F', \neq \Delta\FLD\)
[214] \(\text{\textcircled{v}}\) Branch if all records selected:
[215] \(\rightarrow \Delta\ALL\rho \Delta\AL\)
[216] \(\Delta\VAR \Rightarrow \Delta\text{DATA}\)
[217] \(\text{\textcircled{v}}\) Read field from file:
[218] \(\Delta\text{DATA} \Rightarrow \text{OFREAD} \ \Delta\COMP\)
[219] \(\text{\textcircled{v}}\) Branch if matrix field:
[220] \(\rightarrow (\Delta\W > 1) \rho \Delta\AL\)
[221] \(\Delta\text{DATA}[\Delta\INDS] \Rightarrow \Delta\VAR\)
[222] \(\rightarrow \Delta\AL\)
[223] \(\Delta\AL\)\(\text{\textcircled{v}}\) \(\Delta\text{DATA} \Rightarrow \Delta\text{DATA}[\Delta\INDS] \Rightarrow \Delta\VAR\)
[224] \(\text{\textcircled{v}}\) Check that data is OK before filing:
[225] \(\Delta\AL\)\(\text{\textcircled{v}}\) \(\Delta\text{ERROR}((2 \Delta\W \neq \rho \Delta\text{DATA})/'\text{RANK ERROR}'\)
[226] \(\text{\textcircled{v}}\) \(\Delta\text{ERROR}((\Delta\W \neq 1 \uparrow 1, \rho \Delta\text{DATA} \vee 1 \rho \rho \Delta\text{DATA})/'\text{LENGTH ERROR}'\)
[227] \(\text{\textcircled{v}}\) \(\Delta\text{ERROR}((2 \neq \rho \Delta\FT[2; \Delta\FLD]) \neq 0 = 1 \rho \Delta\text{DATA})/'\text{DOMAIN ERROR}'\)
[228] \(\text{\textcircled{v}}\) Replace data on file:
[229] \(\Delta\text{DATA} \Rightarrow \text{UFREPLACE} \ \Delta\COMP\)
[230] \(\text{\textcircled{v}}\) Continue loop unless layer field:
[231] \(\rightarrow (\Delta\LAY \neq \Delta\FLD) \rho \Delta\AL\)
[232] \(\rightarrow \Delta\ALL \downarrow \Delta\AL\)
[233] \(\Delta\VAR \Rightarrow \Delta\text{DATA}\)
[234] \(\text{\textcircled{v}}\) Branch if matrix field:
[235] \(\Delta\AL\)\(\text{\textcircled{v}}\) \(\Delta\text{VAR} \Rightarrow \Delta\LV[\Delta\SET] \rho \Delta\AL\)
[236] \(\text{\textcircled{v}}\) Else keep track of bad set:
[237] \(\Delta\BAD \Rightarrow \Delta\BAD, \Delta\SET\)
[238] \(\rightarrow \Delta\AL\)
[239] \(\Delta\AL\)\(\text{\textcircled{v}}\) \(\Delta\text{BAD} \Rightarrow \Delta\BAD, \Delta\SET\)
[240] \(\rightarrow (\Delta\VAR \Rightarrow \Delta\LV[\Delta\SET]) \rho \Delta\AL\)
[241] \(\text{\textcircled{v}}\) End of process.
\V EXECUTE (continued)
[246] \arrow AAL23
[247] \a Erase fields in AFLDS:
[248] AAL28: AAL + QEX AAFN3
[249] \arrow AAL6
[250] AAL29: AAL + QEX 'layers'
[251] \a Exit if no sets with wrong layer recs:
[252] \arrow (\times AANSET + \rho AABAD) \downarrow 0
[253] AASETS + AABAD
[254] \a Indices of active fields:
[255] AAFLDS + (\times AAALWID) / \times AAINCR
[256] \a Exclude deletion field:
[257] AAFLDS + (\times AAFLDS \# ADEL) / AAFLDS
[258] \a Is layer field a vector field?
[259] AAVEC + AAFLDS \# AAALAY = 1
[260] \a Last set index:
[261] AAINS + 0
[262] \a Loop by bad set:
[263] AAL30: \arrow (AAINS \geq AANSET) \rho AAL64
[264] \a Current set index and number:
[265] AAINS + AAINS + 1
[266] \a displacement (no. components) before this set:
[267] AAINS + AAINS + AAINS + 1
[268] \a Branch if deletion field unneeded:
[269] \arrow (\times ANDEL + \rho AARPS[AASET] > 0) \rho AAL31
[270] \a Read deletion field for set SET:
[271] AADVERTISEMENT AATIE, AADEL + AAINS + 1
[272] \a Indices of active records:
[273] AAINS + AAINS / \rho AABIT
[274] \arrow AAINS
[275] A nIndex + AAINS + 1
[276] AAINS + AAINS + AAINS + 1
[277] \a Look at layer field:
[278] AAINS + AAINS + AAINS + 1
[279] \a Branch if vector field:
[280] \arrow AAVEC \rho AAINS
[281] \a Indices of recs with changed layer value:
[282] AAINS + (\times AAINS + AAINS + 1) \neq AAINS + AAINS + 1
[283] \arrow AAINS
[284] AAINS + AAINS + AAINS + 1
[285] \a Flag records filed so far:
[286] AAINS + AAINS + AAINS + 1
[287] \a Branch if vector field:
[288] AAINS + AAINS + AAINS + 1
[289] \a Flag records with layer of 1st unfiled rec:
[290] AANTS + AAINS + AAINS + 1
[291] AANTS + AAINS + AAINS + 1
[292] \a ... and sets with this record:
[293] AANTS + AAINS + AAINS + 1
[294] \arrow AAINS
[295] AANTS + AAINS + AAINS + 1
[296] AANTS + AAINS + AAINS + 1
[297] AANTS + AAINS + AAINS + 1
EXECUTE (continued)

\[298\] a Update FILED; convert to indices:
\[299\] ΔAΔL37: ΔΔFILED ← ΔΔFILED ∨ ΔΔGOOD
\[300\] ΔΔGOOD ← ΔΔGOOD / ΔΔBAD
\[301\] ΔΔNR ← ΔΔGOOD
\[302\] a Consider only non-full sets in this layer or
\[303\] a empty sets in any layer:
\[304\] ΔΔLSETS ← (ΔΔARPS = 0) ∨ ΔΔLSETS ∧ ΔΔARPS ∧ ΔΔBLK
\[305\] a Convert to indices:
\[306\] ΔΔLSETS ← ΔΔLSETS / ΔΔARPS
\[307\] a No. records filed so far:
\[308\] ΔΔFREC ← 0
\[309\] a Branch if no slots available in existing sets:
\[310\] ΔΔL38: (ΔΔBLK ≤ ΔΔMIN / ΔΔARPS[ΔΔLSETS]) ∧ ΔΔL53
\[311\] a Branch if more than 1 set needed:
\[312\] ΔΔT ← ΔΔNR - ΔΔFREC
\[313\] → (ΔΔT > ΔΔBLK - ΔΔMIN) ∧ ΔΔL39
\[314\] a Choose fullest set which will hold all recs:
\[315\] ΔΔS ← ΔΔBLK - ΔΔA
\[316\] ΔΔS ← ΔΔLSETS[ΔΔS \& (ΔΔS ≥ ΔΔT) / ΔΔS]
\[317\] → ΔΔL40
\[318\] a Choose set with most empty slots:
\[319\] ΔΔL39: ΔΔS + ΔΔLSETS[ΔΔA \& ΔΔMIN]
\[320\] a No. records to be inserted within the set:
\[321\] ΔΔL40: ΔΔM + ΔΔTL ∧ ΔΔARPS[ΔΔS] - 1 ∧ ΔΔARPS[ΔΔS]
\[322\] a No. records to be catenated within the set:
\[323\] ΔΔN ← (ΔΔT - ΔΔM) ∧ ΔΔBLK - ΔΔARPS[ΔΔS]
\[324\] a Displacement (no. components) before this set:
\[325\] ΔΔSD ← ΔΔDISP + ΔΔINCR × ΔΔS + 1
\[326\] a Branch if no deletion field:
\[327\] → (×ΔΔDEL) ↓ ΔΔL41
\[328\] a Read deletion field for set S:
\[329\] ΔΔBIT ← OFREAD ΔΔDCMP + ΔΔTIE, ΔΔDEL + ΔΔSD
\[330\] a Branch if no records to be inserted:
\[331\] → (×ΔΔM) ↓ ΔΔL41
\[332\] a Indices of available insertion slots:
\[333\] ΔΔINDS ← ΔΔM (ΔΔBIT) / ΔΔBIT
\[334\] a Next field index:
\[335\] ΔΔL41: ΔΔF ← 1
\[336\] a Loop by active field:
\[337\] ΔΔL42: → (ΔΔF > ΔΔNF) ∧ ΔΔL47
\[338\] a Field number:
\[339\] ΔΔFLD ← ΔΔFLDS[ΔΔF]
\[340\] a Field width (no. columns):
\[341\] ΔΔW ← ΔΔWID[ΔΔFLD]
\[342\] a Look at original set:
\[343\] ΔΔVAR ← OFREAD ΔΔTIE, ΔΔFLD + ΔΔDISP
\[344\] a Read field F for set S:
\[345\] ΔΔDATA ← OFREAD ΔΔCMP + ΔΔTIE, ΔΔFLD + ΔΔSD
\[346\] a Branch if a matrix field:
\[347\] → (ΔΔW > 1) ∧ ΔΔL44
\[348\] a Branch if no records to insert:
\[349\] → (×ΔΔM) ↓ ΔΔL43
\[350\] ΔΔDATA[ΔΔINDS] ← ΔΔVAR[ΔΔGOOD[ΔΔFREC + ΔΔM]]
EXECUTE (continued)

A Branch if no records to catenate:

\[A \times (A \times A) \rightarrow A \times L46\]

A Branch if no records to insert:

\[A \times A \rightarrow A \times L45\]

A Branch if no records to catenate:

\[A \times L46\]

Branch if no records to insert:

\[A \rightarrow A \times L48\]

Branch if no records to catenate:

\[A \rightarrow A \times L49\]

Increment FREc by no. records added to this set:

\[A \times L52\]

A Branch if a vector layer field:

\[A \rightarrow A \times A (A \times A) \rightarrow A \times L51\]

A Branch unless a latent field:

\[A \rightarrow A \times L55\]

No. records to be appended in next set:

\[A \times L53\]

Next field number:

\[A \rightarrow A \times L59\]

Loop by field:

\[A \rightarrow A \times L60\]

A Branch unless a latent field:

\[A \rightarrow A \times L55\]

\[A \rightarrow A \times L55\]
EXECUTE (continued)
\[402\] \(\rightarrow \Delta \Delta L59\)
\[403\] \(\Delta\) Branch unless it's the deletion field:
\[404\] \(\Delta \Delta L55:\rightarrow (\Delta \Delta DEL\#\Delta \Delta F)\rho \Delta \Delta L56\)
\[405\] \(\Delta \Delta DATA\rightarrow \Delta \Delta N\rho 1\)
\[406\] \(\rightarrow \Delta \Delta L58\)
\[407\] \(\Delta\) Look at original set:
\[408\] \(\Delta \Delta L56: \Delta \Delta VAR\rightarrow \Delta \Delta DMA\rightarrow \Delta \Delta TIE, \Delta \Delta F\)+\(\Delta \Delta SDISP\)
\[409\] \(\Delta\) Branch if a matrix field:
\[410\] \((\Delta \Delta W>1)\rho \Delta \Delta L57\)
\[411\] \(\Delta \Delta DATA\rightarrow \Delta \Delta VAR[\Delta \Delta GOOD[\Delta \Delta FREC+1\Delta \Delta N]]\)
\[412\] \(\rightarrow \Delta \Delta L58\)
\[413\] \(\Delta \Delta L57: \Delta \Delta DATA\rightarrow \Delta \Delta VAR[\Delta \Delta GOOD[\Delta \Delta FREC+1\Delta \Delta N]]\)
\[414\] \(\Delta\) Branch unless set must be padded to BLK records:
\[415\] \(\Delta \Delta L58: \rightarrow \Delta \Delta FILL\downarrow \Delta \Delta L59\)
\[416\] \(\Delta \Delta DATA\rightarrow (\Delta \Delta BLK, 1)\rho \Delta \Delta DATA\uparrow \Delta \Delta DATA\)
\[417\] \(\Delta \Delta L59: \Delta \Delta T\rightarrow \Delta \Delta DATA \downarrow \Delta \Delta DMA\rightarrow \Delta \Delta TIE\)
\[418\] \(\Delta \Delta F\rightarrow \Delta \Delta F+1\)
\[419\] \(\rightarrow \Delta \Delta L54\)
\[420\] \(\Delta\) Increment FREC by no. records added to this set:
\[421\] \(\Delta \Delta L60: \Delta \Delta FREC\rightarrow \Delta \Delta FREC+\Delta \Delta N\)
\[422\] \(\Delta \Delta ARPS\rightarrow \Delta \Delta ARPS, \Delta \Delta N\)
\[423\] \(\Delta \Delta ARPS\rightarrow \Delta \Delta ARPS, \Delta \Delta T\rightarrow \Delta \Delta N[\Delta \Delta BLK\times \Delta \Delta FILL]\)
\[424\] \(\Delta \Delta NFP[7]\rightarrow \Delta \Delta NFP[7]+\Delta \Delta T\)
\[425\] \(\Delta \Delta NFP[6]\rightarrow \Delta \Delta NFP[6]+1\)
\[426\] \(\Delta \Delta NFP[9]\rightarrow \Delta \Delta NFP[9]+1\)
\[427\] \(\Delta\) Catenate layer value:
\[428\] \(\Delta \Delta LV\rightarrow \Delta \Delta LV, [1]\Delta \Delta LAYER\)
\[429\] \(\Delta\) Continue unless all of data in field vars. filed:
\[430\] \((\Delta \Delta NR\#\Delta \Delta FREC)\rho \Delta \Delta L53\)
\[431\] \(\Delta\) Branch if no data left to file:
\[432\] \(\Delta \Delta L61: \rightarrow (\wedge \Delta \Delta FILED)\downarrow \Delta \Delta L35\)
\[433\] \(\Delta\)
\[434\] \(\Delta\) Branch unless deletion field exists:
\[435\] \((\Delta \Delta DEL\#\Delta \Delta F)\rho \Delta \Delta L62\)
\[436\] \(\Delta \Delta BIT\rightarrow \Delta \Delta DMA\rightarrow \Delta \Delta TIE, \Delta \Delta DEL+\Delta \Delta SDISP\)
\[437\] \(\Delta\) Turn off specified indices and replace:
\[438\] \(\Delta \Delta BIT[\Delta \Delta BAD]+0\)
\[439\] \(\Delta \Delta BIT \uparrow \Delta \Delta DMA\rightarrow \Delta \Delta CMP\)
\[440\] \(\Delta\) Compute new no. records:
\[441\] \(\Delta \Delta N\rightarrow \Delta \Delta BIT\)
\[442\] \(\Delta \Delta M\rightarrow \Delta \Delta ARPS[\Delta \Delta SET]\)
\[443\] \(\Delta\) Reset parameters:
\[444\] \(\Delta \Delta NFP[9]\rightarrow \Delta \Delta NFP[9]-\Delta \Delta N=0\)
\[445\] \(\Delta \Delta ARPS[\Delta \Delta SET]+\Delta \Delta N\times (\wedge 1)[1+\Delta \Delta N=+/\wedge \Delta \Delta BIT]\)
\[446\] \(\rightarrow \Delta \Delta L30\)
\[447\] \(\Delta\)
\[448\] \(\Delta\) Turn off specified indices:
\[449\] \(\Delta \Delta L62: \Delta \Delta M\rightarrow \Delta \Delta ARPS[\Delta \Delta SET]\)
\[450\] \(\Delta \Delta BIT\rightarrow \Delta \Delta M\rho 1\)
\[451\] \(\Delta \Delta BIT[\Delta \Delta BAD]+0\)
\[452\] \(\Delta\) Compute new no. records:
\[453\] \(\Delta \Delta N\rightarrow \Delta \Delta BIT\)
EXECUTE (continued)

- Reset parameters:
  \[ \Delta \text{NFP}[9] + \Delta \text{NFP}[9] - \Delta \text{N} = 0 \]
  \[ \Delta \text{NFP}[7] + \Delta \text{NFP}[7] + \Delta \text{N} = \Delta \text{M} \]
  \[ \Delta \text{ARPS}[\Delta \text{SET}] + \Delta \text{N} \]

- Last field index:
  \[ \Delta \text{F} = 0 \]

- Loop by active field:
  \[ \Delta \text{L63} \rightarrow (\Delta \text{F} \pm \Delta \text{N}) \rho \Delta \text{L30} \]

- Current field index and number:
  \[ \Delta \text{F} = \Delta \text{F} + 1 \]
  \[ \Delta \text{FLD} + \Delta \text{AFLDS}[\Delta \text{F}] \]

- Read, compress, replace field FLD for set SET:
  \[ \Delta \text{CMP} + \Delta \text{TIE}, \Delta \text{FLD} + \Delta \text{SDISP} \]
  \[ (\Delta \text{BIT} \uparrow \text{FREED} \Delta \text{CMP}) \uparrow \text{FREPLACE} \Delta \text{CMP} \]
  \[ \rightarrow \Delta \text{L63} \]

- Replace LV, FP, RPS, ARPS:
  \[ \Delta \text{L64} + \Delta \text{LV} \uparrow \text{FREPLACE} \Delta \text{TIE}, 9 \]
  \[ \Delta \text{NFP} \uparrow \text{FREPLACE} \Delta \text{TIE}, 7 \]
  \[ \Delta \text{ARPS} \uparrow \text{FREPLACE} \Delta \text{TIE}, 8 \]
  \[ \rightarrow (\times \Delta \text{DEL}) \downarrow 0 \]

- \[ \Delta \text{ARPS} \uparrow \text{FREPLACE} \Delta \text{TIE}, 10 \]

[WSID: MULTIFLO]

- R \[=\] A LAYERS B

  Used as:
  - RINDS \[=\] (FP, KFLD) IOTA VALUES LAYERS Z
  - RINDS \[=\] IOTARHO FP LAYERS Z
  - RINDS \[=\] SVEC SLASHIOTARHO FP, SFLDS LAYERS Z
  - FP \[=\] SVEC COMPRESS FP, SFLDS LAYERS Z
  - MAT \[=\] SVEC SELECT FP, FLDs, 0, SFLDS LAYERS Z
  - SVEC SELECTWS FP, FLDs, 0, SFLDS LAYERS Z
  - (FP, XFLDS) EXECUTE XVEC LAYERS Z
  - (FP, XFLDS, 0, SFLDS) EXECUTE XVEC FOR SVEC LAYERS Z

  where Z is a vector (if FP[1] is a vector field)
  - or a matrix (if FP[1] is a matrix field) of the
  - larger values for those sets of records to be
  - considered in the operation being performed. The
  - global variable <layers> is erased after performing
  - the operation (e.g. by IOTA or EXECUTE).

- R \[=\] A LAYERS Z

- where Z is a vector (if FP[1] is a vector field)
- or a matrix (if FP[1] is a matrix field) of the
- larger values for those sets of records to be
- considered in the operation being performed. The
- global variable <layers> is erased after performing
- the operation (e.g. by IOTA or EXECUTE).
5. The APL*PLUS file utility functions listed above should be changed as follows to work on a SHARP APL system:

A. Make the following direct replacements wherever they occur:

<table>
<thead>
<tr>
<th>APL*PLUS</th>
<th>SHARP APL</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFREAD</td>
<td>DREAD</td>
</tr>
<tr>
<td>OFREPLACE</td>
<td>OREPLACE</td>
</tr>
<tr>
<td>DFAPPEND</td>
<td>OAPPENDR</td>
</tr>
<tr>
<td>OFNUMS</td>
<td>ONUMS</td>
</tr>
<tr>
<td>OTCNL</td>
<td>OAV[156+DIO]</td>
</tr>
</tbody>
</table>

B. Wherever DERROR is used, replace the expression with an equivalent DSIGNAL expression. For example:

```
DERROR (1≠ρFP)/'RANK ERROR'

'RANK ERROR' DSIGNAL (1≠ρFP)/599
```

While not listed in this book, the SHARP APL versions of the file utility functions are available on disk. See the Postscript.

Unfortunately, the modifications required to make the file utility functions operational in APL2 are much more dramatic. Since files are limited to fixed length records rather than variable size components, we must take a different approach.

One approach is to use a set of file access functions which emulate the behavior of the file access functions on APL*PLUS and SHARP APL systems. For example, the functions in the IBM public workspace 2 VAPLFILE (CREATE, USE, SET, GET,...) perform such an emulation. When the file is created, you specify the record length, the number of records and the number of components. The utility functions break apart arrays which are too large for a single record and maintain directories to locate and reassemble the pieces of such arrays. If you take this approach, your most difficult tasks are choosing a record length and estimating the number of records the file will ultimately require.

Another approach is to assume that a record length is chosen which is sufficiently large that any of the objects of the file may fit within it. Since the data components of a multi-set transposed file are limited to a specified block size (FP[5]), the maximum object size can be determined in advance. The maximum size of each data component is a function of its field type, its field width and the block size. Therefore, given the parameters of a file (FT and FP), you can determine a record length which is just long enough to accommodate any single file component. Further, if the maximum number...
of records and sets are specified, you can compute the total number of records required.

Given the record length and number of records, the file may be created with all of its records. These records may then be accessed directly (using BDAM) to read or replace objects. In fact, it is a simple matter to write functions FREAD and FREPLACE which behave like the APL*PLUS system functions DFREAD and DFREPLACE.

Appending new records, however, is not so easy. Therefore, the file should be initialized with its maximum number of records, all of which are flagged deleted. The subsequent "catenation" of new records will actually be performed by replacing these deleted records.

Let's pursue the latter approach. To implement it, you can do the following:

A. Write a dyadic function NREC\_RECL whose left argument is FP, whose right argument is FT and whose result is the 2 element vector indicating the number of records and the record length required for the file. FP[6] must be the maximum number of sets needed, FP[7] the maximum number of records, and FP[11] the number of the deletion bit field as a negative number. For example:

```
FT←2 10 12 1 1 1 1 1 1 0 0 0 2 3 3 2 2 4 1 1 1 1
FP←5 987 10 50 2000 25 50000 7 0 0 -7
NL←FP NREC\_RECL FT
NL
300 24020
```

B. Write a dyadic function FCREATE whose left argument is the name of the file to be created, whose right argument is a 3 element integer vector: "tie number", number of records and record length. The file is created, and variables (e.g. CTL987, REC987) are shared with BDAM using the tie number to distinctly identify this file from other files. The records are appended via QSAM or via the FMT option of BDAM. For example:

```
'POLDATA\_DATA' FCREATE FP[2],NL
```

C. Write functions FREAD and FREPLACE to emulate the APL\_PLUS system functions DFREAD and DFREPLACE. Assume variables shared with BDAM using the tie number to distinctly identify the file from other files.

D. Modify the INITFILE function (listed above for APL\_PLUS systems) to require FP[11] to be negative and to fill the entire file (FP[6] sets) with "deleted" records. Use FREPLACE
in place of DFAPPEND since the file already contains all of its "components". For example:

FP INITFILE FT

E. Modify the APL*PLUS file utility functions (CATREC, CATRECWS, IOTA, IOTARHO,...) with the following direct replacements:

<table>
<thead>
<tr>
<th>APL*PLUS</th>
<th>APL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-------</td>
<td>----</td>
</tr>
<tr>
<td>DFREAD</td>
<td>FREAD</td>
</tr>
<tr>
<td>OFREPLACE</td>
<td>FREPLACE</td>
</tr>
<tr>
<td>DTCNL</td>
<td>DTC[1+OIO]</td>
</tr>
<tr>
<td>DERROR</td>
<td>DES</td>
</tr>
</tbody>
</table>

F. Modify CATREC, CATRECWS, INDEXA, INDEXWSA and EXECUTE to eliminate the "append new sets" logic since the file is initialized with the maximum needed number of sets. If all sets are full and more records are to be catenated, signal a FILE FULL error.

G. Write functions FTIE, FUNTIE and FERASE to emulate the APL*PLUS system functions OPTIE, ODFUNTIE and OFERASE. Share variables with BDAM using the tie number to distinctly identify one file from another. For example:

FUNTIE FP[2]

While not listed in this book, the APL2 versions of the file utility functions (including NRECARECL, FCREATE, FREAD,...) are available on disk. See the Postscript.
Chapter 15 Solutions

BOOLEAN TECHNIQUES

1. \( ^\vee / \text{NVEC} = \{ \text{NVEC} \}
   \text{or}
   \( ^\vee / 0 = 1 \mid \text{NVEC} \)

   (Note: The first expression checks whether the elements are integers within comparison tolerance, i.e. OCT. The second expression checks whether the elements are exactly integers to full internal precision. Comparison tolerance is not meaningful for "0=".)

2. \( ( \Phi \vee \backslash ' \# \Phi \text{CVEC}) / \text{CVEC} \)
   \text{or}
   \( (- + / \vee \backslash ' = \Phi \text{CVEC}) \downarrow \text{CVEC} \)

3. \( (+ / \vee \backslash \text{CMAT} = ' ') \Phi \text{CMAT} \)

4. \( \text{MAP} = \text{INPUT} \in '0123456789' \)
   \(+ / \text{MAP} > -1 \downarrow 0, \text{MAP} \)
5. [WSID: TIMING]

\[ \text{TIMEDEFINE } \triangle\triangle\text{name}::\text{\texttt{DIO}};\triangle\\triangle\text{ALP};\triangle\\triangle\text{AN};\triangle\\triangle\text{BL};\triangle\\triangle\text{I};\triangle\\triangle\text{N};\triangle\\triangle\text{NL};\]
\[ \triangle\\triangle\text{NUM};\triangle\\triangle\text{PAN};\triangle\\triangle\text{PBL};\triangle\\triangle\text{NUM};\triangle\\triangle\text{T};\triangle\\triangle\text{CNL};\triangle\\triangle\text{VR} \]

\[ \text{[1]} \triangle\text{IO} \leftarrow 1 \]
\[ \text{[2]} \triangle\triangle\text{name} \text{ remove blanks from fn name:} \]
\[ \text{[3]} \triangle\triangle\text{name} \leftarrow (\triangle\triangle\text{name} \leftarrow \text{'} ') / \triangle\triangle\text{name} \]
\[ \text{[4]} \triangle\text{PL} \leftarrow \text{PLUS} \]
\[ \text{[5]} \triangle\text{ERROR}(3 \# \text{ONC } \triangle\triangle\text{name}) / \text{NOT A DEFINED FUNCTION } \]
\[ \text{[6]} \triangle\text{ERROR}(v / (', '* , \text{OSI} , ' ' , \triangle\triangle\text{name} , ' [ ' ) / \text{FUNCTION SUSPENDED} \]
\[ \text{[7]} \triangle\text{ERROR}(0 = \rho \triangle\triangle\text{VR} \leftarrow \triangle\text{VR} \triangle\triangle\text{name}) / \text{FUNCTION LOCKED} \]
\[ \text{[8]} \triangle\\triangle\text{CNL} \leftarrow \triangle\text{CNL} \]
\[ \text{[9]} \triangle\text{SHARP} \triangle\text{ALP} \]
\[ \text{[10]} \triangle\text{NOT A DEFINED FUNCTION} \triangle\text{ER} (3 \# \text{ONC NAME}) \rho 599 \]
\[ \text{[11]} \text{T} \leftarrow 2 \text{ OWS 2 } \text{ a STATE INDICATOR} \]
\[ \text{[12]} \triangle\text{FUNCTION SUSPENDED} \triangle\text{ER} (v / ((1 \rho T) , 1 + \rho \text{NAME}) \uparrow T) \leftarrow \text{NAME} , ' [ ' ) / \rho 599 \]
\[ \text{[13]} \triangle\text{FUNCTION LOCKED} \triangle\text{ER} (0 = \rho \triangle\triangle\text{VR} \leftarrow 1 \text{ OFD NAME}) \rho 599 \]
\[ \text{[14]} \triangle\text{CNL} \leftarrow \triangle\text{CNL} \leftarrow \text{AV}[157] \]
\[ \text{[15]} \triangle\text{Flag newline chars (Boolean partition vector):} \]
\[ \text{[16]} \triangle\triangle\text{NL} \leftarrow 1 \triangle\triangle\text{VR} = \triangle\triangle\text{CNL} \]
\[ \text{[17]} \triangle\text{Flag digits:} \]
\[ \text{[18]} \triangle\triangle\text{NUM} \leftarrow \triangle\text{VR} \leftarrow '0123456789' \]
\[ \text{[19]} \triangle\text{Flag vector of contiguous digits:} \]
\[ \text{[20]} \triangle\\triangle\text{PNUM} \leftarrow \triangle\triangle\text{NUM} \leftarrow 1 \triangle\triangle\text{NUM} \]
\[ \text{[21]} \triangle\text{Flag char following ] after line no.:} \]
\[ \text{[22]} \triangle\triangle\text{NL} \leftarrow 1 \triangle\\triangle\text{PNUM} \leftarrow 1 \\triangle\triangle\text{PNUM} \leftarrow 1 \triangle\triangle\text{NL} \]
\[ \text{[23]} \triangle\\triangle\text{PNUM} \leftarrow 0 \]
\[ \text{[24]} \triangle\text{Flag blanks:} \]
\[ \text{[25]} \triangle\\triangle\text{BL} \leftarrow \triangle\text{VR} \leftarrow ' ' \]
\[ \text{[26]} \triangle\text{Flag vector of contiguous blanks:} \]
\[ \text{[27]} \triangle\\triangle\\text{PBL} \leftarrow \triangle\\triangle\text{BL} \leftarrow 1 \triangle\\triangle\text{BL} \]
\[ \text{[28]} \triangle\text{Flag 1st nonblank char in each line:} \]
\[ \text{[29]} \triangle\triangle\text{NL} \leftarrow (\triangle\triangle\text{NL} \leftarrow \triangle\\triangle\text{BL}) \vee \triangle\\triangle\text{PBL} \leftarrow 1 \triangle\\triangle\text{PBL} / \triangle\triangle\text{NL} \]
\[ \text{[30]} \triangle\\triangle\text{BL} \leftarrow 0 \]
\[ \text{[31]} \triangle\text{Flag letters:} \]
\[ \text{[32]} \triangle\\triangle\text{ALP} \leftarrow \triangle\text{VR} \leftarrow ' \text{ABCDEFGHIJKLMNOPQRSTUVWXYZ} ' \]
\[ \text{[33]} \triangle\text{Flag digits or letters:} \]
\[ \text{[34]} \triangle\triangle\text{AN} \leftarrow \triangle\\triangle\text{ALP} \vee \triangle\triangle\text{NUM} \]
\[ \text{[35]} \triangle\triangle\text{NUM} \leftarrow 0 \]
\[ \text{[36]} \triangle\text{Flag vector of contiguous digits/letters:} \]
\[ \text{[37]} \triangle\\triangle\text{PAN} \leftarrow \triangle\triangle\text{AN} \leftarrow 1 \triangle\triangle\text{AN} \]
\[ \text{[38]} \triangle\triangle\text{AN} \leftarrow 0 \]
\[ \text{[39]} \triangle\text{Flag vector of identifiers:} \]
\[ \text{[40]} \triangle\\triangle\text{PAN} \leftarrow \triangle\\triangle\text{PAN} \leftarrow \triangle\\triangle\text{T} \leftarrow 1 \triangle\\triangle\text{T} \leftarrow \triangle\\triangle\text{PAN} / \triangle\\triangle\text{ALP} \]
\[ \text{[41]} \triangle\\triangle\text{ALP} \leftarrow 0 \]
\[ \text{[42]} \triangle\text{Flag vector of identifiers at start of line:} \]
\[ \text{[43]} \triangle\\triangle\text{PAN} \leftarrow \triangle\\triangle\text{PAN} \leftarrow \triangle\\triangle\text{T} \leftarrow 1 \triangle\\triangle\text{T} \leftarrow \triangle\\triangle\text{PAN} / \triangle\\triangle\text{NL} \]
\[ \text{[44]} \triangle\text{Flag vector of labels:} \]
\[ \text{[45]} \triangle\\triangle\text{PAN} \leftarrow \triangle\\triangle\text{PAN} \leftarrow \triangle\\triangle\text{T} \leftarrow 1 \triangle\\triangle\text{T} \leftarrow '\text{'} = \triangle\\triangle\text{PAN} / \triangle\text{VR} \]
\[ \text{[46]} \triangle\text{Flag 1st nonblank after label or at start:} \]
\[ \text{[47]} \triangle\\triangle\text{NL} \leftarrow (\triangle\\triangle\text{NL} \leftarrow \triangle\\triangle\text{PAN}) \leftarrow 1 \triangle\\triangle\text{PAN} / \triangle\\triangle\text{NL} \]
TIMEDEFINE (continued)

\[ \Delta \Delta \text{PAN} + 0 \]

\[ \Delta \Delta \text{VR} \left( 1 + \Delta \Delta \text{NL} + \Delta \Delta \text{NL} \right) / \Delta \Delta \text{VR} \]

\[ \Box \text{IO} + 0 \]

\[ \Delta \Delta \text{I} \left( \Delta \Delta \text{N} + / \Delta \Delta \text{NL} \right) \]

\[ \Delta \text{I} \left( \Box \text{DEF} \right) \Delta \Delta \text{VR} \left( \text{APLPLUS} \right) \]

\[ \Delta \text{I} \left( \Box \text{DEF} \right) \Delta \text{NAME}, \left[ ', (\Delta \Delta \text{N} + 1), ', \right] \Delta \text{V} \left( \text{APLPLUS} \right) \]

\[ \Delta \text{I} \left( \Box \text{DEF} \right) \Delta \text{NAME}, \left[ ', (\Delta \Delta \text{N} + 1), ', \right] \Delta \text{V} \left( \text{SHARP APL} \right) \]

\[ \Delta \text{M} \left( \Delta \Delta \text{N}, 4 \right) \rho 0 0, (1 / 10) \rho 0 \]

\[ \Delta \text{L} \left( 0 \right) \]

\[ \left( \left( \Delta \Delta \text{NL} / \rho \Delta \Delta \text{NL} \right) + \Delta \Delta \text{I} + \Delta \Delta \text{I} \right) + 0 \]

6.

IDENTIFY VR; ALP; AN; ARGs; AVAR; BL; COLS; HDR; IDLEN; IDS;
IDSTART; IND; LAB; LEN; LOC; N; NC; NCMT; NID; NL; NQ; NUM;
PAN; PARSE; PBL; PID; PIDA; PLAB; PNUM; PSID; R; RESULT; RVAR; S;
START; TCNL; TIO

\[ \Delta \text{I} \left( \Box \text{DEF} \right) \Delta \text{NAME}, \left[ ', (\Delta \Delta \text{N} + 1), ', \right] \Delta \text{V} \left( \text{APLPLUS} \right) \]

\[ \Delta \text{I} \left( \Box \text{DEF} \right) \Delta \text{NAME}, \left[ ', (\Delta \Delta \text{N} + 1), ', \right] \Delta \text{V} \left( \text{SHARP APL} \right) \]

\[ \Delta \text{M} \left( \Delta \Delta \text{N}, 4 \right) \rho 0 0, (1 / 10) \rho 0 \]

\[ \Delta \text{L} \left( 0 \right) \]

\[ \left( \left( \Delta \Delta \text{NL} / \rho \Delta \Delta \text{NL} \right) + \Delta \Delta \text{I} + \Delta \Delta \text{I} \right) + 0 \]

\[ \Delta \text{I} \left( \Box \text{DEF} \right) \Delta \text{NAME}, \left[ ', (\Delta \Delta \text{N} + 1), ', \right] \Delta \text{V} \left( \text{APLPLUS} \right) \]

\[ \Delta \text{I} \left( \Box \text{DEF} \right) \Delta \text{NAME}, \left[ ', (\Delta \Delta \text{N} + 1), ', \right] \Delta \text{V} \left( \text{SHARP APL} \right) \]

\[ \Delta \text{M} \left( \Delta \Delta \text{N}, 4 \right) \rho 0 0, (1 / 10) \rho 0 \]

\[ \Delta \text{L} \left( 0 \right) \]

\[ \left( \left( \Delta \Delta \text{NL} / \rho \Delta \Delta \text{NL} \right) + \Delta \Delta \text{I} + \Delta \Delta \text{I} \right) + 0 \]

\[ \Delta \text{I} \left( \Box \text{DEF} \right) \Delta \text{NAME}, \left[ ', (\Delta \Delta \text{N} + 1), ', \right] \Delta \text{V} \left( \text{APLPLUS} \right) \]

\[ \Delta \text{I} \left( \Box \text{DEF} \right) \Delta \text{NAME}, \left[ ', (\Delta \Delta \text{N} + 1), ', \right] \Delta \text{V} \left( \text{SHARP APL} \right) \]

\[ \Delta \text{M} \left( \Delta \Delta \text{N}, 4 \right) \rho 0 0, (1 / 10) \rho 0 \]

\[ \Delta \text{L} \left( 0 \right) \]
Chapter 15 Solutions BOOLEAN TECHNIQUES

\[ \text{IDENTIFY (continued)} \]

[28] a Remove result from header:

\[ \text{HDR} \leftarrow ' ', (\text{T} \times \text{IND} + 1) \downarrow \text{HDR} \]

[29] a Starting indices of fn name, args:

\[ \text{START} \leftarrow 1 + (\text{HDR} = ' ') / \text{pHDR} \]

[30] a Lengths of names:

\[ \text{LEN} \leftarrow (1 \downarrow \text{START}, 1 + \text{pHDR}) - 1 \downarrow \text{START} \]

[31] a No. of args:

\[ \text{T} \leftarrow 1 + \text{LEN} \]

[32] a Indices of args (if any):

\[ \text{N} \leftarrow \text{IND} + (\text{T} = 2) \rho \text{0}, (\times \text{T}) \rho \text{T} \]

[33] a Don't consider fn name:

\[ \text{START} \leftarrow \text{START}[\text{IND}] \]

[34] a Length of longest arg:

\[ \text{COLS} \leftarrow \text{T} \downarrow \text{LEN} / \text{LEN} \]

[35] a Raveled, blank mat of args:

\[ \text{ARGS} \leftarrow (\text{N} \times \text{COLS}) \rho \]

[36] a Fill them in (T+\text{MONIOTA} \text{LEN}):  

\[ \text{T} \leftarrow \text{T} + \text{LEN} / \downarrow \text{LEN} \downarrow \text{LEN} + \text{LEN} \]

[37] a Reshape to mat of args:

\[ \text{ARGS} \leftarrow [\text{T} + \text{LEN} / \text{COLS} \times \text{LEN}] \text{HDR} \leftarrow \text{T} + \text{LEN} / \text{START} \]

[38] a Flag newline chars (Boolean partition vector):  

\[ \text{NL} \leftarrow \text{1} \downarrow \text{VR} = \text{TCNL} \]

[39] a Flag nonquotes:

\[ \text{NQ} \leftarrow \text{VR} \cdot '' \]

[40] a Map of chars not in quote pairs (i.e. char constants)

\[ \text{NC} \leftarrow \text{NCCON} \text{AND} \text{VR} = ' ' \]

[41] a within each fn line (NCCON=NL pEQSCAN NQ):

\[ \text{S} \leftarrow \text{NL} \times \text{NC} = \text{NC} \text{MT} \leftarrow \text{S} \times \text{T} \leftarrow \text{NC} \text{ORDER} \leftarrow \text{NC} \text{MT} \leftarrow \text{NCCON} \]

[42] a (NCMT=NL pANDSCAN NC):

\[ \text{S} \leftarrow \text{NL} \times \text{NC} \]

[43] a Map of chars which do not follow a \text{S} (including as)

\[ \text{AN} \leftarrow \text{NUM} \times \text{ALP} \]

[44] a within quotes. As are flagged 0.

\[ \text{AN} \leftarrow \text{NCMT} = \text{NCCON} \times \text{AN} \]

[45] a Pole vec of contiguous digits:

\[ \text{PNUM} \leftarrow \text{NUM} \leftarrow \text{AN} \times \text{ALP} \]

[46] a Flag digits, letters, blanks:

\[ \text{NUM} \leftarrow \text{NUM} \leftarrow 0 \downarrow \text{NUM} \]

[47] a Pole vec of contiguous digits:

\[ \text{NUM} \leftarrow 0 \]
Chapter 15 Solutions

\[ \] IDENTIFY (continued)

\[ [80] \] a Pole vec of contiguous digits/letters:
\[ [81] \] PAN+AN\#140,AN
\[ [82] \] AN\#0
\[ [83] \] a Pole vec of identifiers:
\[ [84] \] PID\#PAN\Tv\#1T\#PAN/ALP
\[ [85] \] ALP\#PIAN\#0
\[ [86] \] a Flag '0' before identifiers (\#names):
\[ [87] \] T\#1\#PID
\[ [88] \] T\#T'\#T'/T/VR
\[ [89] \] a Shift leading poles of \#names to include 0:
\[ [90] \] PID\TvPID\#1\#T
\[ [91] \] T\#0
\[ [92] \] a Flag char following ] after line no.:
\[ [93] \] START\#1\#PNUM\#1\#PNUM/\#1\#NL
\[ [94] \] NL\#PNUM\#0
\[ [95] \] a Pole vec of contiguous blanks:
\[ [96] \] PBL\#BL\#1\#0,BL
\[ [97] \] a Flag 1st nonblank char in each line:
\[ [98] \] START\#(START\#BL)\#PBL\#1\#PBL/START
\[ [99] \] BL\#PBL\#0
\[ [100] \] a Pole vec of identifiers at start of line:
\[ [101] \] PSID\#PID\Tv\#1\#T\#PID/START
\[ [102] \] START\#0
\[ [103] \] a Pole vec of labels:
\[ [104] \] PLAB\#PSID\Tv1\#T\#':=PSID/VR
\[ [105] \] PSID\#0
\[ [106] \] a Pole vec of direct assignment identifiers:
\[ [107] \] PIDA\#PID\Tv1\#T\#'=PID/VR
\[ [108] \] a Start and end (+1) indices of identifiers:
\[ [109] \] IND\#PID/l\#PID
\[ [110] \] a No. of identifiers:
\[ [111] \] NID\#(IND\#2)/2
\[ [112] \] IND\#(NID,2)\#IND
\[ [113] \] a Start indices of identifiers:
\[ [114] \] IDSTART\#IND[:0]
\[ [115] \] a Lengths of identifiers:
\[ [116] \] IDLEN\#IND[:11-IDSTART
\[ [117] \] a Starting indices of local vars:
\[ [118] \] START\#1+(LOC\#::':')/l\#LOC
\[ [119] \] a Lengths of local vars:
\[ [120] \] LEN\#(1\#START,1\#LOC)-1\#START
\[ [121] \] a Length of longest ident.:
\[ [122] \] COLS\#(\#LEN)\#(\#IDLEN)\#(\#RESULT)\#(1\#ARGS
\[ [123] \] a Pad arg names to conform:
\[ [124] \] ARG\#(1\#ARGS, COLS)\#ARGS
\[ [125] \] a 0 row mat if no result:
\[ [126] \] RESULT\#(\#RESULT), COLS\# COLS\#RESULT
\[ [127] \] a Raveled blank mat of local vars:
\[ [128] \] T\#(COLS\#N\#START)\#'
\[ [129] \] a Fill them in (S\#MONIOTA LEN):
\[ [130] \] S\#S\#S\#LEN/1\#0,+\#LEN
\[ [131] \] T\#S\#LEN/ COLS\#N\#LOC[S\#LEN\#/START]
\[ IDENTIFY (continued)\]

[112] \( \wedge \) Reshape to mat of local vars:
[113] \( \text{LOC} \leftarrow (N, \text{COLS}) T \)
[114] \( \wedge \) Raveled blank mat of identifiers:
[115] \( T \leftarrow (\text{COLS} \times N \leftarrow \text{IDSTART}) \rho' \)
[116] \( \wedge \) Fill them in \((S \leftarrow \text{MONIOTA} \ \text{IDLEN})\):
[117] \( S = S + \rho S + \text{IDLEN} / \sim 1 \downarrow 0, + \text{IDLEN} \)
[118] \( T[S + \text{IDLEN} / \text{COLS} \times \downarrow N] \leftarrow \text{VAR}[S + \text{IDLEN} / \text{IDSTART}] \)
[119] \( \wedge \) Reshape to mat of identifiers:
[120] \( \text{IDS} \leftarrow (N, \text{COLS}) T \)
[121] \( \wedge \) Mat of label names:
[122] \( \text{LAB} \leftarrow (S \leftarrow ((N, 2) \rho \text{PID} / \text{PLAB}) [; 0]) \not/ \text{IDS} \)
[123] \( \text{PLAB} \leftarrow 0 \)
[124] \( \wedge \) Mat of assigned vars:
[125] \( \text{AVAR} \leftarrow (T \leftarrow ((N, 2) \rho \text{PID} / \text{PIDA}) [; 0]) \not/ \text{IDS} \)
[126] \( \text{PID} \leftarrow \text{PIDA} \leftarrow 0 \)
[127] \( \wedge \) Mat of referenced vars:
[128] \( \text{RVAR} \leftarrow (S \leftarrow T) \not/ \text{IDS} \)
[129] \( \wedge \) Make distinct:
[130] \( \text{AVAR} \leftarrow (\text{AVAR} \not/ \text{MONIOTA} \ \text{AVAR}) = \sim 1 \rho \text{AVAR} \not/ \text{AVAR} \)
[131] \( \text{RVAR} \leftarrow (\text{RVAR} \not/ \text{MONIOTA} \ \text{RVAR}) = \sim 1 \rho \text{RVAR} \not/ \text{RVAR} \)
[132] \( \wedge \)
[133] \( \wedge \) Flag distinct labels:
[134] \( \to (\wedge / T \leftarrow (\text{LAB} \not/ \text{MONIOTA} \ \text{LAB}) = \sim 1 \rho \text{LAB}) \rho L_1 \)
[135] \( N \not/ ' \text{#S} ', ' (\sim T) \not/ \text{LAB} \)
[136] \( \text{LAB} \leftarrow T \not/ \text{LAB} \)
[137] \( \ominus ' \text{REduDANt LABEL} : ', 1 \downarrow (N \vee 1 \Phi N) / S \)
[138] \( \wedge \) Flag distinct locals:
[139] \( L_1 \leftarrow R \vee ' ' \)
[140] \( \to (\wedge / T \leftarrow (\text{LOC} \not/ \text{MONIOTA} \ \text{LOC}) = \sim 1 \rho \text{LOC}) \rho L_2 \)
[141] \( R \not/ ' ', (\sim T) \not/ \text{LOC} \)
[142] \( \text{LOC} \leftarrow T \not/ \text{LOC} \)
[143] \( L_2 \leftarrow (\wedge / T \leftarrow (1 \rho \text{RVAR}) \not/ \text{RVAR} \not/ \text{MONIOTA} \ \text{LOC}) \downarrow L_3 \)
[144] \( N \not/ ' \text{#S} ', ' T \not/ \text{LOC} \)
[145] \( \ominus ' \text{IDENTIFIER LOCALIZED BUT NOT USED} : ', 1 \downarrow (N \vee 1 \Phi N) / S \)
[146] \( \wedge \) Any unassigned local vars?
[147] \( L_3 \leftarrow (\wedge / T \leftarrow (1 \rho \text{AVAR}) \not/ \text{AVAR} \not/ \text{MONIOTA} \ \text{LOC}) \downarrow L_4 \)
[148] \( N \not/ ' \text{#S} ', ' T \not/ \text{LOC} \)
[149] \( \ominus ' \text{IDENTIFIER LOCALIZED BUT NOT ASSIGNED} : ', 1 \downarrow (N \vee 1 \Phi N) / S \)
[150] \( \wedge \) Include result and args in local vars:
[151] \( L_4 \leftarrow \text{LOC} \leftarrow (\text{RESULT}, [0]) \not/ \text{ARGS}, [0] \not/ \text{LOC} \)
[152] \( \wedge \) Flag distinct locals:
[153] \( \to (\wedge / T \leftarrow (\text{LOC} \not/ \text{MONIOTA} \ \text{LOC}) = \sim 1 \rho \text{LOC}) \rho L_5 \)
[154] \( R \not/ ' R, ' (\sim T) \not/ \text{LOC} \)
[155] \( \text{LOC} \leftarrow T \not/ \text{LOC} \)
[156] \( L_5 \leftarrow (x \rho R) \downarrow L_6 \)
[157] \( N \not/ ' R \)
[158] \( \ominus ' \text{REduDANt LOCAL VARIABLE} : ', 1 \downarrow (N \vee 1 \Phi N) / R \)
[159] \( \wedge \) Any localized labels?
[160] \( L_6 \leftarrow (\wedge / T \leftarrow (1 \rho \text{LOC}) \not/ \text{LOC} \not/ \text{MONIOTA} \ \text{LAB}) \downarrow L_7 \)
[161] \( N \not/ ' \text{#S} ', ' T \not/ \text{LAB} \)
[162] \( \ominus ' \text{LOCALIZED LABEL} : ', 1 \downarrow (N \vee 1 \Phi N) / S \)
\[ \text{\textcopyright Chapter 15 Solutions  BOOLEAN TECHNIQUES} \]

\[ \text{\textcopyright \hspace{1cm}} \]

\[ \text{\textcopyright IDENTIFY (continued)} \]

[183] \( \text{\textcopyright Any unreferenced labels?} \)

[184] \( \text{\textcopyright L7:}(\vee/T&\langle 1_{\rho_0}RVAR}\leq RVAR\text{ CMIOTA LAB})\downarrow L8 \)

[185] \( \text{\textcopyright N' 'S', 'T/LAB} \)

[186] \( \text{\textcopyright D'UNUSED LABEL: ',1}(N\vee 1\phi N)/S \)

[187] \( \text{\textcopyright A Any assigned labels?} \)

[188] \( \text{\textcopyright L8:}(\vee/T&\langle 1_{\rho_1}AVAR}\geq AVAR\text{ CMIOTA LAB})\downarrow L9 \)

[189] \( \text{\textcopyright N' 'S', 'T/LAB} \)

[190] \( \text{\textcopyright D'ASSIGNED LABEL: ',1}(N\vee 1\phi N)/S \)

[191] \( \text{\textcopyright A Any unlocalized assigned vars?} \)

[192] \( \text{\textcopyright L9:}(\vee/T&\langle 1_{\rho_0}LOC}\leq LOC\text{ CMIOTA AVAR})\downarrow L10 \)

[193] \( \text{\textcopyright N' 'S', 'T/AVAR} \)

[194] \( \text{\textcopyright D'IDENTIFIER ASSIGNED BUT NOT LOCALIZED: ',1}(N\vee 1\phi N)/S \)

[195] \( \text{\textcopyright A Any unlocalized referenced vars?} \)

[196] \( \text{\textcopyright L10:}(\vee/T&\langle 1_{\rho_0}LOC)+1_{\rho_0}LAB\leq(LOC,[0]LAB)\text{ CMIOTA RVAR})\downarrow L11 \)

[197] \( \text{\textcopyright N' 'S', 'T/RVAR} \)

[198] \( \text{\textcopyright D'IDENTIFIER USED AND NOT LOCALIZED: ',1}(N\vee 1\phi N)/S \)

[199] \( \text{\textcopyright L11:}(\vee/T&\langle 1_{\rho_1}AVAR}\leq AVAR\text{ CMIOTA RESULT})\downarrow L12 \)

[200] \( \text{\textcopyright D'RESULT NOT ASSIGNED: ',RESULT} \)

[201] \( \text{\textcopyright L12:}(\vee/T&\langle 1_{\rho_0}RVAR}\leq RVAR\text{ CMIOTA ARGS})\downarrow 0 \)

[202] \( \text{\textcopyright N' 'S', 'T/ARGS} \)

[203] \( \text{\textcopyright D'ARGUMENT NOT USED: ',1}(N\vee 1\phi N)/S} \)

\[ \text{\textcopyright WSID: FNIDS} \]

\[ \text{\textcopyright R\textcopyright KEEP LOCALIZE VR;ALP;AN;ARGS;COLS;HDR;IDLEN;IDS; IDSTART;IND;KIDS;KLEN;KSTART;LOCAL;N;NB;NC;NCCON; NCMT;NID;NKEEP;NL;NQ;NUM;PAN;PARSE;PID;PIDA;RESULT;S;T;TCNL;DIO} \]

[1] \( \text{\textcopyright A Modifies the vector representation (VR) of a} \)

[2] \( \text{\textcopyright function such that its header contains as localized} \)

[3] \( \text{\textcopyright variables all and only those variables which are} \)

[4] \( \text{\textcopyright assigned within the function. KEEP is a character} \)

[5] \( \text{\textcopyright matrix or vector (blank delimited) of names of} \)

[6] \( \text{\textcopyright variables to not localize though assigned, or to} \)

[7] \( \text{\textcopyright localize though not assigned.} \)

[8] \( \text{\textcopyright A Requires subfunction: CMIOTA.} \)

[9] \( \text{\textcopyright DIO} \)

[10] \( \text{\textcopyright A Construct scalar newline character:} \)

[11] \( \text{\textcopyright TCNL\textcopyright DTCNL} \)

[12] \( \text{\textcopyright A TCNL\textcopyright DTC[1]} \textcopyright A APL2} \)

[13] \( \text{\textcopyright A TCNL\textcopyright DAV[156]} \textcopyright A SHARP APL} \)

[14] \( \text{\textcopyright A Flag newline chars (Boolean partition vector):} \)

[15] \( \text{\textcopyright NL\textcopyright 1\phi VR=TCNL} \)

[16] \( \text{\textcopyright A Flag nonquotes:} \)

[17] \( \text{\textcopyright NQ\textcopyright VR#''} \)

[18] \( \text{\textcopyright A Map of chars not in quote pairs (i.e. char constants) \}

[19] \( \text{\textcopyright A within each fn line (NCCON\textcopyright NL pEQSCAN NQ):} \)

[20] \( \text{\textcopyright NCCON\textcopyright \textbackslash NQ\#NL\textbackslash T\textasciitilde1\downarrow 0,T\textasciitilde NL/=\textbackslash ~1\downarrow 1,NQ} \)

[21] \( \text{\textcopyright NQ} \)

\[ -455- \]
Chapter 15 Solutions

LOCALIZE (continued)

[22] A Flag non-A chars (includes AS in quotes):
[23] NC<NCCON\VR='A'
[24] A Map of chars which do not follow a A (ignoring AS
[25] A within quotes) within each fn line. AS are flagged 0.
[26] A (NCMT<NL pANDSCAN NC):
[27] S<NL<NC
[28] NCMT<#\S\T<#1↓0,T<~S/NC
[29] S<T<NC+0
[30] A Map of chars which are not included within AS or ' ':
[31] PARSE+NCMT\NCON
[32] NCON<NCMT<0
[33] A Flag digits, letters:
[34] NUM+PARSE\VR<'0123456789'
[35] ALP+PARSE\VR<"ABCDEFGHIJKLMNOPQRSTUVWXYZ\abedefghijklmnopqrstuvwxyz"'
[36] PARSE<0
[37] A Flag alphanumeric chars:
[38] AN+NUM\ALP
[39] NUM<0
[40] A Pole vec of contiguous digits/letters:
[41] PAN<AN<#1↓0,AN
[42] AN<0
[43] A Pole vec of identifiers:
[44] PID<PAN\Tv~1ΦT<PAN/ALP
[45] ALP<PAN<0
[46] A Flag '0' before identifiers (Onames):
[47] T<1<ΠPID
[48] T<T<"0"=T/VR
[49] A Shift leading poles of Onames to include 0:
[50] PID<TvPID<~1ΦT
[51] T<0
[52] A Pole vec of direct assignment identifiers:
[53] PID<PID\Tv1ΦT<~'='=PID/VR
[54] A Start and end (+1) indices of identifiers:
[55] IND<PID/1<ΠPID
[56] A No. of identifiers:
[57] NID<0<IND<2
[58] IND<0<IND<2<0<IND
[59] A Start indices of identifiers:
[60] IDSTART<IND[;0]
[61] A Lengths of identifiers:
[62] IDLEN<IND[;1]−IDSTART
[63] A Map vec of nonblanks in exception identifiers:
[64] NB<"'##KEEP','",KEEP
[65] A Start indices in KEEP of identifiers:
[66] NKEEP<0KSTART<0<NB>−1<ΦNB)/0<ΠNB
[67] A Lengths of KEEP identifiers:
[68] KLEN<1+(NB>1<ΝΒ)/0<ΝΒ<−KSTART
[69] A Length of longest identifier:
[70] COLS<0/KLEN)/0<IDLEN
[71] A Raveled blank matrix of identifier names:
[72] IDS<0<ΝΙD×COLS<0"
\[ v \text{ LOCALIZE (continued)} \]

\[ a \text{ Fill them in (T+MONIOTA IDLEN):} \]
\[ a \text{ T} <- T + \text{IDLEN} - 1 \downarrow 0, + \text{IDLEN} \]
\[ a \text{ IDS} [\text{T} + \text{IDLEN/COLS} \times \text{NID}] + \text{VR}[\text{T} + \text{IDLEN/IDSTART}] \]
\[ a \text{ Reshape to mat of identifiers:} \]
\[ a \text{ IDS} + (\text{NID}, \text{COLS}) \times \text{IDS} \]
\[ a \text{ Prepare to look at header syntax:} \]
\[ a \text{ T} <- \text{VR} \times \text{TCNL}, ' ; ' \]
\[ a \text{ Take up to local vars:} \]
\[ a \text{ HDR} + (\text{I/T}) \times \text{VR} \]
\[ a \text{ Drop leading junk:} \]
\[ a \text{ HDR} + (v \backslash \text{ HDR} \leftarrow v ) / \text{HDR} \]
\[ a \text{ Explicit result (if any):} \]
\[ a \text{ RESULT} + (\text{T}, \text{COLS}) \times \text{COLS} ^ {\text{TH} \times \text{IDLEN}} \]
\[ a \text{ Remove result from header:} \]
\[ a \text{ HDR} + (\text{T} \times \text{IND} + 1) / \text{HDR} \]
\[ a \text{ No. of arguments:} \]
\[ a \text{ S} + '/ ' = \text{HDR} \]
\[ a \text{ Arguments (if any):} \]
\[ a \text{ ARGS} <- \text{IDS} [\text{T} + ((\text{S} = 2) \times \text{IND}; (\times \text{S}) \times \text{IND}; ] \]
\[ a \text{ Mat of assigned identifiers:} \]
\[ a \text{ LOCAL} + (\text{NID}, 2) \times \text{PID} \times \text{PIDA} ([0]) / \text{IDS} \]
\[ a \text{ PID} + \text{PIDA} = 0 \]
\[ a \text{ Remove redundancies:} \]
\[ a \text{ LOCAL} + (\text{LOCAL} \times \text{CMNIOTA} \times \text{LOCAL}) = \text{IND} \times \text{LOCAL} / \text{LOCAL} \]
\[ a \text{ Raveled blank matrix of exception identifiers:} \]
\[ a \text{ KIDS} + (\text{NKEEP} \times \text{COLS}) \times \text{KIDS} \]
\[ a \text{ Fill in (T+MONIOTA KLEN):} \]
\[ a \text{ T} <- \text{T} + \text{KLEN} - 1 \downarrow 0, + \text{KLEN} \]
\[ a \text{ KIDS} [\text{T} + \text{KLEN/COLS} \times \text{NKEEP}] + \text{KEEP}[\text{T} + \text{KLEN/KSTART}] \]
\[ a \text{ Reshape to mat of exception identifiers:} \]
\[ a \text{ KIDS} + (\text{NKEEP}, \text{COLS}) \times \text{KIDS} \]
\[ a \text{ No. of syntax local vars:} \]
\[ a \text{ N} + \text{IR} \times \text{RESULT}, [0] \times \text{ARGS} \]
\[ a \text{ Squeeze out identifiers in KEEP (globally assigned)} \]
\[ a \text{ or syntax local vars:} \]
\[ a \text{ LOCAL} + (\text{S} + (\text{N} + \text{NKEEP}) \times \text{IND}) = \text{IND} \times (\text{KIDS}, [0] \times \text{CMNIOTA} \times \text{LOCAL}) / \text{LOCAL} \]
\[ a \text{ Include identifiers in KEEP which are not assigned } \]
\[ a \text{ (assigned within subfns):} \]
\[ a \text{ T} <- (1 + \text{N} + \text{NKEEP}) \times 1 \]
\[ a \text{ T} \left[ \text{IND} \right] = 0 \]
\[ a \text{ LOCAL} + \text{LOCAL}, [0] \times (\text{NKEEP} \times \text{T}) / \text{KIDS} \]
\[ a \text{ Place local vars in sorted order:} \]
\[ a \text{ LOCAL} + \text{LOCAL}, [\text{AVA} \times \text{LOCAL}; ] \]
\[ a \text{ Lengths of local vars:} \]
\[ a \text{ LLEN} + / \text{LOCAL} , ' ' \]
\[ a \text{ Indices after each header local semicolon:} \]
\[ a \text{ T} + ' 1 + 0, \text{LLEN} \]
\[ a \text{ Initialize local var segment of header:} \]
\[ a \text{ HDR} + (1 + ' 1 \uparrow \text{T}) \times ' \]
\[ a \text{ Insert (S+MONIOTA LLEN):} \]
\[ a \text{ S} + \text{S} \times \text{S} \times \text{LLEN} - 1 \downarrow 0, + \text{LLEN} \]
\[ a \text{ HDR} + [\text{S} + \text{LLEN} / 1 \uparrow \text{T} \left( \text{S} + \text{LLEN/COLS} \times 1 \text{LLEN} \right) \times \text{LLEN} \times 1 \downarrow] (-, \text{LOCAL}) [\text{S} + \text{LLEN} / \text{COLS} \times 1 \text{LLEN} ] \]

-457-
LOCALIZE (continued)

\[ T \leftarrow VR \setminus TCNL, ' '; \]
\[ R \leftarrow ((1/T) p VR), HDR, T[0] \downarrow VR \]

[WSID: COMMENTS]

\[ R \leftarrow UNCOMMENT VR; BL; NC; NCON; NCMT; NL; NQ; NUM; PARSE; PBL; PNUM; S; START; T \]

1. A Modifies the vector representation (VR) of a
2. A function such that its comments are removed (but
3. A the lamp symbols remain for full-line comments).
4. A Flag newline chars (Boolean partition vector):
5. \[ NL \leftarrow \sim \downarrow VR \times DTCNL \]
6. \[ NL \leftarrow \sim \downarrow VR \div DTC[l+IO] \] A APL2
7. \[ NL \leftarrow \sim \downarrow VR \div DAV[156+IO] \] A SHARP APL
8. A Flag nonquotes:
9. \[ NQ \leftarrow VR \div ' ' \]
10. A Map of chars not in quote pairs (i.e. char constants)
11. A within each fn line (NCON\leftarrow NL \times EQSCAN NQ):
12. \[ NCON\leftarrow=\left(NQ \times NL \div T \div 1 \div 0, T \leftarrow NL /= \sim 1 \div 1, NQ \right) \]
13. \[ NQ \leftarrow 0 \]
14. A Flag non-A chars (includes \( @ \) as in quotes):
15. \[ NC \leftarrow NCON \times VR \div ' A' \]
16. A Map of chars which do not follow a \( @ \) (ignoring \( @ \)
17. A within quotes) within each fn line. \( @ \)s are flagged 0.
18. A (NCMT\leftarrow NL \times PANDSCAN NC):
19. \[ S \leftarrow NL \div NC \]
20. \[ NCMT\leftarrow \# \div S \div T \div 1 \div 0, T \leftarrow S \div NC \]
21. \[ S \leftarrow T \div NC \leftarrow 0 \]
22. A Always include newline chars (extend maps left by 1):
23. \[ NCMT\leftarrow NCMT \times 1 \div FNCMT \]
24. A Map of chars which are not included within \( @ \)s or \( ' ' \):
25. \[ PARSE \div NCMT \times NCON \]
26. \[ NCON \leftarrow 0 \]
27. A Flag digits, blanks:
28. \[ NUM \leftarrow PARSE \times VR \div '0123456789' \]
29. \[ BL \leftarrow PARSE \times VR \div ' ' \]
30. \[ PARSE \leftarrow 0 \]
31. A Pole vec of contiguous digits:
32. \[ PNUM \leftarrow NUM \div \sim 1 \div 0, NUM \]
33. \[ NUM \leftarrow 0 \]
34. A Flag char following \( ] \) after line no.:
35. \[ START \leftarrow \sim 1 \div FNUM \div \sim 1 \div FNUM \div \sim 1 \div NL \]
36. \[ NL \leftarrow PNUM \leftarrow 0 \]
37. A Pole vec of contiguous blanks:
38. \[ PBL \leftarrow BL \div \sim 1 \div 0, BL \]
39. A Flag 1st nonblank char in each line:
40. \[ START \leftarrow (START > BL) \div PBL \div \sim 1 \div PBL \times START \]
41. \[ BL \leftarrow PBL \leftarrow 0 \]
42. A Squeeze out cmnts except cmnt symb for full line:
43. \[ R \leftarrow (NCMT \times START) / VR \]

\[ -458 - \]
Chapter 16 Solutions

IRREGULAR ARRAYS

1. \( \text{AVG} \leftarrow +(+/\text{SALES}) + p \cdot \text{SALES} \)  
   \( \text{AVG} \leftarrow ,\leftarrow (+/\text{SALES}) + p \cdot \text{SALES} \)  
   \( T \leftarrow ,p \cdot \text{SALES} \)  
   \( \text{AVG} \leftarrow (+/T \cdot p \cdot \text{SALES}) + T \)  
   \( \text{AVG} \leftarrow (\text{SALES NNSUMCOLS 1 1}) \cdot \text{NNLEN SALES} \)  
   (Conventional APL)

2. \( \text{UNAMES} \leftarrow (\text{NAMES} \cdot \text{NAMES}) = p \cdot \text{NAMES}) / \text{NAMES} \)  
   (APL2, APL*PLUS, SHARP APL)  
   \( \text{UNAMES} \leftarrow \text{NAMES CNIDX ((NAMES CNIOTA NAMES) =} \) 
   \( \text{t} \cdot \text{t} \cdot \text{NAMES}) / \text{t} \cdot \text{t} \cdot \text{NAMES} \)  
   (Conventional APL)  
   or \( \text{SORTED} \leftarrow \text{NAMES CNIDX DAV CNGRADEUP NAMES} \)  
   (Conventional APL)  
   \( \text{UNAMES} \leftarrow \text{SORTED CNIDX (~t\cdot\text{t}, \text{SORTED CNEQ} \) 
   \( \text{SORTED} \leftarrow \text{CNIDX 1} \cdot \text{t} \cdot \text{t} \cdot \text{SORTED}) / \text{t} \cdot \text{t} \cdot \text{SORTED} \) 

3. \( \text{v R} \leftarrow \text{CNEST M:A;B;L;N;S;E;I;} \) 
   (WSID: CNFNS) 
   [1] A CNEST is used to convert a character vector or 
   [2] A matrix into a character nest. If a character 
   [3] A vector, the first character is taken as the 
   [4] A delimiter; each set of characters between 
   [5] A delimiters defines one segment. If a character 
   [6] A matrix, each row (less the trailing blanks) 
   [7] A defines one segment. The rows of a character 
   [8] A matrix are assumed to be left justified, i.e. 
   [9] A to have no blanks to the left of the leftmost 
   [11] A Branch if scalar or vector: 
   [12] \( \rightarrow (1 \cdot p \cdot M) \cdot p \cdot M \) 
   [13] \( \cdot \text{t} \cdot \text{IO} \leftarrow 0 \)
Chapter 16 Solutions

IRREGULAR ARRAYS

\[ \text{V CNEST (continued)} \]

[14] \text{\texttt{A Insure even no. cols:}}
[15] \text{\texttt{M=M,}}
[16] \text{\texttt{A Lengths:}}
[17] \text{\texttt{L1:L1+/v', \#'FM}}
[18] \text{\texttt{A Cvt to int mat, incl lengths:}}
[19] \text{\texttt{M+L,163 ODR M}}
[20] \text{\texttt{A No. elts to take from each row, incl length:}}
[21] \text{\texttt{N+R\#L+1+L/2}}
[22] \text{\texttt{A Start elts of raveled mat:}}
[23] \text{\texttt{S<(1\#M)\times N}}
[24] \text{\texttt{A R=MONIOTA L:}}
[25] \text{\texttt{R+R\#R+L/-1\#0,+\L}}
[26] \text{\texttt{R+(N,+(+-1\#0,L)+1+N),(,M)[R+L/S]}}
[27] \text{\texttt{\to 0}}
[28] \text{\texttt{A Word starts, lengths:}}
[29] \text{\texttt{L2:OIO<1}}
[30] \text{\texttt{B\#M=1\#M}}
[31] \text{\texttt{N\#S\#B/\#B}}
[32] \text{\texttt{L+R-1+(1\#S,1\#0B)-S}}
[33] \text{\texttt{I++\#-1\#0,L),E+1+B+L+2}}
[34] \text{\texttt{A 8224=163 ODR '':}}
[35] \text{\texttt{\#IO<0}}
[36] \text{\texttt{R+(N,I),E/8224}}
[37] \text{\texttt{R[I]+L}}
[38] \text{\texttt{R\#82 ODR R}}
[39] \text{\texttt{\#IO<1}}
[40] \text{\texttt{A R=MONIOTA L:}}
[41] \text{\texttt{A+A+\#P\#A+/\#-1\#0,+\L}}
[42] \text{\texttt{R[A+L/2+I+I]+M[A+L/S]}}
[43] \text{\texttt{R\#163 ODR R}}
[44] \text{\texttt{\n}}

\[ \text{[WSID: CNFNS]} \]

\[ \text{V R+W CN\#M M;J:L;N;S;X;OIO} \]

[1] \text{\texttt{A CN\#M is used to convert a character nest (M) to a}}
[2] \text{\texttt{character matrix (R) with W columns and one row}}
[3] \text{\texttt{per segment. If W is 0, the matrix will have as}}
[4] \text{\texttt{many columns as the longest segment in M. If W}}
[5] \text{\texttt{has two elements, the first is the number of columns}}
[6] \text{\texttt{of the result and the second is the justification}}
[7] \text{\texttt{indicator: 1 (left justify each segment within its}}
[8] \text{\texttt{row); 2 (right justify each segment within its row);}}
[9] \text{\texttt{3 (center each segment within its row). If the}}
[10] \text{\texttt{second element is omitted, a value of 1 (left}}
[11] \text{\texttt{justify) is assumed. If the first element is}}
[12] \text{\texttt{a negative (e.g. -1), the matrix will have as many}}
[13] \text{\texttt{columns as the longest segment in M.}}
[14] \text{\texttt{A J\in 2 3 (L,R,C):}}
[15] \text{\texttt{J+1\#(1\#W),1}}
[16] \text{\texttt{A Handle char vec or scalar arg:}}
[17] \text{\texttt{\to(82\#ODR M)\#L1}}

-460-
Chapter 16 Solutions

IRREGULAR ARRAYS

\[ CN\Delta M \text{ (continued)} \]
[18] \[ L \leftarrow \rho M \leftarrow M \]
[19] \[ W \leftarrow l \{(0=\rho, W) \lor 0>1 \{W / L \}, W \]
[20] \[ R \leftarrow W + M \]
[21] \[ R \leftarrow (1, W_\rho ((J \neq 1) \times 0 \{ (W-L) \} \lor J-1) \phi R \]
[22] \[ \rightarrow 0 \]
[23] \[ \text{a No., starts, lengths:} \]
[24] \[ L_1: \{0\} \leftarrow l \]
[25] \[ X \leftarrow l N \leftarrow M[\{0\}] \]
[26] \[ \{0\} \leftarrow 0 \]
[27] \[ L \leftarrow M[S \leftarrow M[X]] \]
[28] \[ \rightarrow (0>1 \{W, -1\}) \downarrow L_2 \]
[29] \[ W \leftarrow 0 \{W / L \]
[30] \[ \text{a Truncate too-long segments:} \]
[31] \[ L_2: L \leftarrow L \{W \leftarrow 1 \{W \]
[32] \[ \text{a Blank, raveled result:} \]
[33] \[ R \leftarrow (N \times W) \rho' ' \]
[34] \[ S \leftarrow S + 1 \]
[35] \[ X \leftarrow \text{MONIOTA L:} \]
[36] \[ X \leftarrow X + l \rho X \leftarrow L / \rightarrow 1 \{0, + L \}
[37] \[ M \leftarrow (82 \text{ DDR } M)[X + L / S + S] \]
[38] \[ \rightarrow (L_3, L_4, L_5)[J + 1] \]
[39] \[ \text{a Left:} \]
[40] \[ L_3: R[X + L / W \times \{N\}] \leftarrow M \]
[41] \[ \rightarrow L_6 \]
[42] \[ \text{a Right:} \]
[43] \[ L_4: R[X + L / (W \times \{N\}) + W-L] \leftarrow M \]
[44] \[ \rightarrow L_6 \]
[45] \[ \text{a Center:} \]
[46] \[ L_5: R[X + L / (W \times \{N\}) + 10.5 \times W-L] \leftarrow M \]
[47] \[ L_6: R \leftarrow (N, W) \rho R \]

\[ [\text{WSID: CNFNS}] \]

\[ R \leftarrow C \text{ CNAV CNEST;} N, L, S, T, E, I ; I O \]
[1] \[ \text{a CNAV is used to convert a character nest} \]
[2] \[ \text{a CNEST to a character vector with the} \]
[3] \[ \text{a character string C between each segment.} \]
[4] \[ \rightarrow (82 \# \text{DDR CNEST}) \rho I I \]
[5] \[ R \leftarrow \text{CNEST} \]
[6] \[ \rightarrow 0 \]
[7] \[ \text{a No., starts, lengths:} \]
[8] \[ L_1: \{I O\} \leftarrow 1 \]
[9] \[ T \leftarrow l N \leftarrow \text{CNEST}[\{I O\}] \]
[10] \[ \{I O\} \leftarrow 0 \]
[11] \[ L \leftarrow \text{CNEST}[S + \text{CNEST}[T]] \]
[12] \[ S \leftarrow S + 1 \]
[13] \[ R \leftarrow \rho C \leftarrow C \]
[14] \[ \rightarrow (0 \{1 \} + N + N) \rho 1 \{0 \}
[15] \[ \text{a Lengths of segments (incl separators):} \]
[16] \[ L \leftarrow R + E \{L - R \]
[17] \[ \text{a Starts of segments (in C, CNEST):} \]
[18] \[ S \leftarrow E \{R + S + S \]
\( \text{CN\Delta V (continued)} \)

\[ \text{\[19\] a R\oplus \text{MONIOTA } L:} \]
\[ \text{\[20\] R=R+1_{p} R=L/-110,+L} \]
\[ \text{\[21\] R=(C,82 \text{ DR CNEST})[R+L/S]} \]

\[ \text{\[WSID: \text{CNFNSJ}\text{]}\]}

\( \text{\[22\] R\oplus \text{A CNCAT B;D;E;F;G;L;\oplus IO} } \)
\[ \text{\[1\] a CNCAT is used to catenate two character nests and} \]
\[ \text{\[2\] a return the catenated character nest result. Either} \]
\[ \text{\[3\] a argument may be a character nest or a character} \]
\[ \text{\[4\] a vector (i.e. a character nest "scalar").} \]
\[ \text{\[5\] \text{\&IO}\text{+0} } \]
\[ \text{\[6\] a \text{\&V\&V,\&V\&S,\&S\&V,\&S\&S:}} \]
\[ \text{\[7\] \rightarrow (L, L, L, L, L)[2182=(\text{\&DR A,\&DR B}]} \]
\[ \text{\[8\] L,\text{\&IO}\text{+1} } \]
\[ \text{\[9\] D\text{\&\&A}[\text{\&IO}]} \]
\[ \text{\[10\] F\text{\&\&B}[\text{\&IO}]} \]
\[ \text{\[11\] \text{\&IO}\text{+0} } \]
\[ \text{\[12\] D\text{\&\&A}[D]} \]
\[ \text{\[13\] F\text{\&\&B}[F]} \]
\[ \text{\[14\] R\text{\&}((G+E,D),(F+1+pA),(1+E)\text{\&A}),(1+G)\text{\&B} \]
\[ \text{\[15\] \rightarrow 0} \]
\[ \text{\[16\] L,\text{\&IO}\text{+1} } \]
\[ \text{\[17\] F\text{\&\&L+2} } \]
\[ \text{\[18\] \text{\&IO}\text{+0} } \]
\[ \text{\[19\] D\text{\&\&A}[\text{\&IO}]} \]
\[ \text{\[20\] \text{\&IO}\text{+0} } \]
\[ \text{\[21\] D\text{\&\&A}[D]} \]
\[ \text{\[22\] R\text{\&}(1+E,D,\text{\&A}),(1+E)\text{\&A},L,163 \text{\&DR(F+F)\text{\&B}} \]
\[ \text{\[23\] \rightarrow 0} \]
\[ \text{\[24\] L,\text{\&IO}\text{+1} } \]
\[ \text{\[25\] F\text{\&\&L+2} } \]
\[ \text{\[26\] \text{\&IO}\text{+0} } \]
\[ \text{\[27\] D\text{\&\&A}[\text{\&IO}]} \]
\[ \text{\[28\] \text{\&IO}\text{+0} } \]
\[ \text{\[29\] D\text{\&\&A}[D]} \]
\[ \text{\[30\] R\text{\&}((E+1,2),(D+2+F),L,163 \text{\&DR(F+F)\text{\&A}),(1+E)\text{\&B}} \]
\[ \text{\[31\] \rightarrow 0} \]
\[ \text{\[32\] L,\text{\&IO}\text{+1} } \]
\[ \text{\[33\] E\text{\&\&D+2} } \]
\[ \text{\[34\] F\text{\&\&B}+B} \]
\[ \text{\[35\] G\text{\&\&F+2} } \]
\[ \text{\[36\] R\text{\&}(2+3,E+4),(D,163 \text{\&DR(E+E)\text{\&A}),(1+E)\text{\&B}} \]

\[ \text{\[\text{\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&}\)
Chapter 16 Solutions

IRREGULAR ARRAYS

[WSID: CNFNS]

```plaintext
\textbf{\texttt{\textbackslash{}R+\texttt{CNEST CNIDX INDS;L;S;N}}}
\textbf{\texttt{\textbackslash{}V}}

[1] A CNIDX is used to extract one or more segments
[2] A from a character nest (CNEST). INDS is an
[3] A integer vector or scalar of the indices of the
[4] A segments to be extracted. If a vector, the
[5] A array extracted (R) will be a character nest.
[6] A If a scalar, the array will be a character vector.
[7] A \texttt{\textbackslash{}L+\texttt{CNEST[OIO+S+\texttt{CNEST[1+INDS]}}]
[8] A Branch if nest result:
[9] A \texttt{\textbackslash{}R+\texttt{(O\#N+\texttt{RINDS})*L}}
[10] A \texttt{\textbackslash{}R+L*82 DDR CNEST[(S+1)+L/L/2]}
[11] A \texttt{\textbackslash{}R+0}
[12] A \texttt{\textbackslash{}L+L+1+L/2}
[13] A R+\texttt{MONIOTA L:}
[14] A \texttt{\textbackslash{}R+R+1\texttt{R+L/(-1\downarrow 0,+L}}
[15] A \texttt{\textbackslash{}R+(N,\downarrow\texttt{L}(1+N),L),CNEST[R+L/S]}
```

\[ \textbf{\texttt{\textbackslash{}V}} \]

[WSID: CNFNS]

```plaintext
\textbf{\texttt{\textbackslash{}R+\texttt{A ASSIGN B}}}
```

[WSID: CNFNS]

```plaintext
\textbf{\texttt{\textbackslash{}V}}
```

[WSID: CNFNS]

```plaintext
\textbf{\texttt{\textbackslash{}R+\texttt{CNEST CNIDXA INDS;L;S;NL;N;A;B;C;NS;I;U}}}
\textbf{\texttt{\textbackslash{}V}}

[1] A CNIDXA is used to replace one or more segments
[2] A into a character nest (CNEST). INDS is an
[3] A integer vector or scalar of the indices of the
[4] A segments to be replaced. If a vector, the
[5] A array replaced (assign) will be a character nest.
[6] A If a scalar, the array will be a character
[7] A vector. The function ASSIGN simply assigns its
[8] A right argument to the global variable <assign>
[9] A and returns its left argument. After being
[10] A replaced, <assign> is erased.
[12] A Branch unless a scalar index:
[13] A \texttt{\textbackslash{}R+(N\#\texttt{RINDS})*L2}
[14] A \texttt{\textbackslash{}L+\texttt{CNEST[OIO+S+\texttt{CNEST[1+INDS]}}]
[15] A \texttt{\textbackslash{}NL+\texttt{RASSIGN<assign}}
[16] A Index assign if same length:
[17] A A+L/2
[18] A B+L/2
[19] A \texttt{\textbackslash{}R+(A\#B)*L1}
[20] A R+\texttt{CNEST}
[21] A R[S+L+1+B]+NL,163 DDR(B+B)+assign
[22] A \texttt{\textbackslash{}R+L9}
[23] A L1:R+((S+1)*CNEST
[26] A R[S+OIO]+NL
```

-463-
Chapter 16 Solutions
IRREGULAR ARRAYS

\[ \text{CNIDX}A \ (\text{continued}) \]

[27] \( R+R,(163 \quad \text{DDR}(B+B)\uparrow \text{assign}), (S+A+1)\downarrow \text{CNEST} \)
[28] \( \rightarrow L9 \)
[29] a Branch if nothing to assign:
[30] \( L2: \rightarrow (0=U+\rho \text{INDS})\rho L7 \)
[31] \( I+1+\uparrow N \)
[32] \( L<\text{CNEST}[[\text{IO}+S+\text{CNEST}[I]]] \)
[33] \( A+\rho L+2 \)
[34] a Branch if char vec to assign:
[35] \( \rightarrow (C<82=\text{DDR assign})\rho L3 \)
[36] \( R+\downarrow U \)
[37] \( NL<\text{assign}[[\text{IO}+\text{NS}+\text{assign}[1+R]]]) \)
[38] \( B+\uparrow \downarrow NL+2 \)
[39] \( \rightarrow L4 \)
[40] \( L3: R+\downarrow L \)
[41] \( NL<\rho \text{assign}, \text{assign} \)
[42] \( NS<0 \)
[43] \( B+\uparrow NL+2 \)
[44] \( \text{assign}<\text{NL}, 163 \quad \text{DDR}(B+B)\uparrow \text{assign} \)
[45] a Index assign if same length.
[46] a Branch if any lengths change:
[47] \( L4: \rightarrow (A[I\text{INDS}]=B)\rho L8 \)
[48] a Drop header data off nest if multi-segments:
[49] \( \rightarrow C\rho L5 \)
[50] \( \text{assign}<U+1\downarrow \text{assign} \)
[51] \( \rightarrow L6 \)
[52] \( L5: B+\rho U \)
[53] a Repeat data if scalar assignment:
[54] \( \text{assign}<U\times \rho \text{assign})\rho \text{assign} \)
[55] \( L6: R<\text{CNEST} \)
[56] \( I+S[I\text{INDS}] \)
[57] \( L+1+B \)
[58] a \( S+\text{MONIOTA} L: \)
[59] \( S+S+1\rho S+S/L/\sim L0, +L \)
[60] \( R[S+S/L][I]\downarrow \text{assign} \)
[61] \( \rightarrow L9 \)
[62] \( L7: R<\text{CNEST} \)
[63] \( \rightarrow L9 \)
[64] \( L8: I[I\text{INDS}]<N+R \)
[65] \( L+1+(A, B)[I] \)
[66] \( S+(S, NS+\text{CNEST}[I]) \)
[67] a \( R+\text{MONIOTA} L: \)
[68] \( R+1\rho R+L/\sim L0, +L \)
[69] \( R+(N, +\downarrow L)\downarrow (1+N), L), \text{CNEST}, \text{assign}[R+L/S] \)
[70] \( L9: \text{ERASE } \text{'assign'} \)
Chapter 16 Solutions

IRREGULAR ARRAYS

[WSID: CNFNS]
\[ \text{V L} \leftarrow \text{CNLEN \ CNEST;} \text{IO} \]

[1] A \text{CNLEN} is used to return the lengths of the
[3] \text{IO} \leftarrow 1
[4] L \leftarrow \text{CNEST}[1]
[5] \text{IO} \leftarrow 0
[6] L \leftarrow \text{CNEST}[\text{CNEST}[L]]

\[ \text{V} \]

[WSID: CNFNS]
\[ \text{V R} \leftarrow \text{CSEQ \ CNGRADEUP \ CNEST;} \text{A;} \text{B;} \text{D;} \text{G;} \text{I;} \text{L;} \text{M;} \text{N;} \text{P;} \text{R;} \text{S;} \text{T;} \text{Z} \]

[1] A \text{CNGRADEUP} is used to determine the grade vector
[2] A which can be used (with CNIDX) to sort the
[3] A segments of \text{CNEST} into ascending order. The
[4] A collating sequence used is in \text{CSEQ}.
[5] \text{N} \leftarrow \text{CNEST}[\text{IO}]
[6] A Start, lengths:
[7] S \leftarrow \text{IO} + \text{CNEST}[1 + \text{N}]
[8] L \leftarrow \text{CNEST}[S] / 2
[9] A Empty segments precede others in coll. seq.:
[10] R \leftarrow A \leftarrow L > 0
[11] A Z: indices into \text{CNEST} of remaining elts (i.e. \text{R[A]}):
[12] Z \leftarrow B / \text{N}
[13] A Done if 0 or 1 segments or all empty:
[14] \text{A} \leftarrow \text{(N} > 1 \text{)} \text{AND} \text{RZ} \downarrow 0
[15] A A: indices into \text{R} of remaining elts (always
[16] A ascending), \text{B[R]} / \text{N}:
[17] A \leftarrow \text{(N} - \text{RZ}) + \text{L} \text{RZ}
[18] A 1st col (2 chars as an integer) of data:
[19] D \leftarrow \text{CNEST}[S[Z] + I + 1]
[20] A Reorder grade vec:
[21] \text{R[A]} \leftarrow Z \leftarrow \text{CSEQ}((\text{R} \text{D}), 2) \text{R} 82 \text{ DDR D}
[22] D \leftarrow \text{D[G]}
[23] A Flag 1st elts of grps of like values (partition vec):
[24] P \leftarrow D \leftarrow 1 \text{FD}
[25] A Flag elts of >1 elt grps (map vec):
[26] LP \leftarrow M \leftarrow P \leftarrow 1 \text{FP}
[27] \text{T} \leftarrow \text{(R} \text{A} \leftarrow \text{M/A}) \uparrow 0
[28] P \leftarrow M / P
[29] A Skip following logic if no segments end here:
[30] \text{T} \leftarrow \text{(A} \leftarrow \text{B} + \text{L[Z} \leftarrow \text{R[A]} > 1) \text{R} L 1
[31] A Shift enders to front or end of grps:
[32] G \leftarrow A
[33] G \leftarrow G \leftarrow (T \leftarrow \text{P}) [G]
[34] \text{R[A]} \leftarrow Z \leftarrow G
[35] B \leftarrow B \leftarrow G
[36] D \leftarrow B / T
[37] P \leftarrow D \leftarrow 1 \text{FD}
[38] M \leftarrow P \leftarrow 1 \text{FP}
[39] \text{T} \leftarrow \text{(R} \text{A} \leftarrow \text{M/B/A}) \uparrow 0
[40] Z \leftarrow \text{R[A]}
[41] P \leftarrow M / P

-465-
Chapter 16 Solutions  IRREGULAR ARRAYS

\[\text{CNGRADEUP (continued)}\]

\[L1: I+I+1\]
\[D\leftarrow \text{CNEST}[S[Z]+I]\]
\[G\leftarrow \text{CSEQ}((\rho D), 2)\rho 82 \text{ DDR D}\]
\[G\leftarrow G[+(P)[G]]\]
\[R[A] \leftarrow Z[G]\]
\[D\leftarrow D[G]\]
\[P\leftarrow P\vee D\neq 1\Phi D\]
\[\rightarrow LP\]
\[\downarrow\]

[WSID: CNFNS]

\[\text{R} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.

\[\downarrow V\]

\[\text{V} \leftarrow A \ CNEQ B; C; D; E; S; T; L; K; Z; \Omega O\]

1. A CNEQ is used to compare two character nests of the same length (i.e. number of segments) for matching segments. The result is a Boolean vector with as many elements as segments and with a 1 for each segment which is the same in both arguments. A character vector argument (or arguments) is treated as a character nest "scalar" and is compared against all the segments of the other argument. Thus, if both arguments are character vectors, the result will be a scalar 1 if the vectors are equivalent and will be 0 otherwise.
Chapter 16 Solutions

IRREGULAR ARRAYS

\[ \text{\textcopyright CNEQ (continued)} \]
[41] \( \text{s+1a[1]} \)
[42] \( \text{OIOo0} \)
[43] \( S+\text{A[S]} \)
[44] \( R+\text{A[S]}=\text{L}+\rho B+, B \)
[45] \( \rightarrow(\times L)1O \)
[46] \( OIO+1 \)
[47] \( S+(R/S)\circ. +1L=[L^2 \]
[48] \( OIOo0 \)
[49] \( R+R^{\text{A}[S]} \Rightarrow =163 \Delta DR(L+L)\uparrow B \)
[50] \( \rightarrow 0 \)
[51] \( \text{a Reverse S o V args and use V o S logic:} \)
[52] \( \text{L3:T+} \)
[53] \( \text{A+B} \)
[54] \( \text{B+T} \)
[55] \( \rightarrow L2 \)
[56] \( \text{L4:A,} \)
[57] \( \text{B+}, B \)
[58] \( \rightarrow (R<\rho A)^. =\rho B)\downarrow 0 \)
[59] \( \text{R+A^.. =B} \)
\[ \text{\textcopyright [WSID: CNFNS]} \]

\[ \text{\textcopyright INDS+CNEST CNIOTA VALS; A; B; BASE; C; F; G; I; L; LAB; LL; M; S; SHAPE; SS; cseq} \]
[1] \( \text{a CNIOTA searches through the character nest left} \)
[2] \( \text{a argument CNEST for the character nest, vector} \)
[3] \( \text{a or scalar right argument VALS. The result is} \)
[4] \( \text{a an integer vector (for nest right arg.) or} \)
[5] \( \text{a scalar (for character right arg.) of the} \)
[6] \( \text{a segment index in which the character vector was} \)
[7] \( \text{a first located (as an exact, not partial, match)} \)
[8] \( \text{a or 1 greater than the number of segments if not} \)
[9] \( \text{a found (ala dyadic \( \downarrow \)).} \)
[10] \( \text{a Requires subfns: CNGRADEUP, CNCAT, CNIDX, CNEQ.} \)
[11] \( \text{a Branch if right arg a nest:} \)
[12] \( \text{C+CNEST[OIO] } \)
[13] \( \rightarrow (82\# DDR VALS)\rho L1 \)
[14] \( \text{a Handle char. vec or scalar right arg:} \)
[15] \( \text{L+\rho VALS+, VALS} \)
[16] \( \text{SHAPE+10} \)
[17] \( \text{a Length of arg. in integers:} \)
[18] \( \text{M+1L+2} \)
[19] \( \text{a Convert arg. to integers:} \)
[20] \( \text{VALS+163 DDR(M+M)\uparrow VALS} \)
[21] \( \rightarrow L5 \)
[22] \( \text{L1:A+VALS}[OIO] \)
[23] \( \text{a Branch unless no segments in either arg:} \)
[24] \( \rightarrow(\times F+Al C)\rho L2 \)
[25] \( \text{a Handle empty arg:} \)
[26] \( \text{INDS+A\rho OIO} \)
[27] \( \rightarrow 0 \)
Chapter 16 Solutions IRREGULAR ARRAYS

\text{\textbackslash} CNIOTA (continued)

[28] \text{\textbackslash} A Branch if both args have more than 1 segment:
[29] \text{L2\text{\rightarrow}(F\#1)\text{\textbackslash} L7}
[30] \text{\textbackslash} A Branch unless left arg has 1 segment:
[31] \text{\textrightarrow(C\#1)\text{\textbackslash} L4}
[32] \text{\textbackslash} A Handle 1 segment left arg:
[33] \text{L\textleft\text{\leftarrow CNEST}[2+\text{\textbackslash} 10]}
[34] \text{M\textleft\text{\textleftarrow L\div2}}
[35] \text{\textbackslash} A Extract segment as integers:
[36] \text{CNEST\textleft\text{\leftarrow CNEST}[3+\text{\textbackslash} M]}
[37] \text{S\textleft\text{\leftarrow 10+VALS}[1+\text{\textbackslash} 1A]}
[38] \text{B\textleft\text{\leftarrow VALS}[S]}
[39] \text{\textbackslash} A Branch if segment empty:
[40] \text{\textrightarrow(x\textleft)\textleft L3}
[41] \text{\textbackslash} A Indices of same-length segments:
[42] \text{S\textleft\text{\leftarrow ((\text{\textbackslash} 10)+B/S)+\text{\textbackslash} M}
[43] \text{B\textleft\text{\leftarrow B\textbackslash VALS}[S]\text{\textbackslash CNEST}
[44] \text{L3\textleft\text{\leftarrow IND\textleft 10+\text{\textbackslash} B}
[45] \text{\textrightarrow 0}
[46] \text{\textbackslash} A Handle 1 segment right arg:
[47] \text{L4\textleft\text{\leftarrow SHAPE\textleft 1}
[48] \text{L\textleft\text{\textleftarrow VALS}[2+\text{\textbackslash} 10]}
[49] \text{M\textleft\text{\textleftarrow L\div2}
[50] \text{VALS\textleft\text{\leftarrow VALS}[3+\text{\textbackslash} M]}
[51] \text{L5\textleft\text{\leftarrow 10+CNEST}[1+\text{\textbackslash} C]}
[52] \text{\textbackslash} A Flag same-length segments in left arg:
[53] \text{B\textleft\text{\leftarrow L\textbackslash CNEST[S]}
[54] \text{\textbackslash} A Branch if segment empty:
[55] \text{\textrightarrow(x\textleft)\textleft L6}
[56] \text{\textbackslash} A Indices of same length segments:
[57] \text{S\textleft\text{\leftarrow ((\text{\textbackslash} 10)+B/S)+\text{\textbackslash} M}
[58] \text{B\textleft\text{\leftarrow B\textbackslash CNEST[S]\text{\textbackslash CNEST}
[59] \text{L6\textleft\text{\leftarrow IND\textleft 10+SHAPE\textbackslash B\textbackslash 1}
[60] \text{\textrightarrow 0}
[61] \text{\textbackslash} A Branch if sort alg. costs more than looping alg.:
[62] \text{\textbackslash (remove \textbackslash after replacing C1,C2,C3,C4 by}
[63] \text{\textbackslash computed constants)}:
[64] \text{L7\textleft\text{\leftarrow ((C4+5\times L+A)\times C1+A\times C2+C3\times L)\text{\textbackslash L5}
[65] \text{\textbackslash A Combine args. and sort (like values together)}:
[66] \text{cseq\textleft\text{\textleftarrow OAV}
[67] \text{F\textleft\text{\leftarrow G\textleft\text{\leftarrow OAV CNGRADE\textleft A\textleft CNEST\textleft\text{\textleftarrow CNCAT\textleft VALS}
[68] \text{\textbackslash A \textleft F\textleft AG:
[69] \text{F[G]\textleft\text{\leftarrow \textbackslash G}
[70] \text{\textbackslash A Flag 1st of distinct rows by shifting and comparing:
[71] \text{F\textleft\text{\textleftarrow (~A\textleft\text{\textleftarrow CNEQ\textleft\text{\textleftarrow A\textleft\text{\textleftarrow CNEQ\textleft\text{\textleftarrow ~G}[F])[G]}
[72] \text{\textbackslash A Insure 1st elt is 1 (in case all rows the same)}:
[73] \text{F[10]\textleft\text{\leftarrow 1}
[74] \text{\textbackslash A Indices of 1st distinct rows:
[75] \text{I\textleft\text{\leftarrow F\textleft G}
[76] \text{\textbackslash A Replicate for each like row:
[77] \text{F[10]\textleft\text{\leftarrow 10}
[78] \text{I\textleft\text{\textleftarrow I\textleft I\textleft F]}
[79] \text{\textbackslash A Unsort indices (to catenated order)}:
[80] \text{IND\textleft\text{\leftarrow I}
\[ V \text{ CNIOTA (continued)} \]

[81] \text{INDS}[G] \leftarrow I

[82] \text{a Keep those corresponding to right arg:}

[83] \text{INDS} \leftarrow \text{C} \downarrow \text{INDS}

[84] \text{a Set 'not found' inds to 'one greater':}

[85] \text{INDS} \leftarrow \text{INDS} \downarrow \text{C} \downarrow \text{IO}

[86] \rightarrow 0

[87] \text{a Use looping algorithm if more efficient:}

[88] L8: \text{INDS} \leftarrow \text{A} \rho 0

[89] \text{LAB} \leftarrow (\text{A} \rho \text{L9}) \downarrow 0

[90] \text{LL} \leftarrow \text{CNEST} [[\text{SS} \leftarrow \text{IO} + \text{CNEST} [1 + \text{CNEST} [\text{IO}]])]

[91] \text{a Starting indices from which to add \text{IM}:}

[92] \text{SS} \leftarrow \text{SS} \leftarrow \text{IO}

[93] I \leftarrow \text{IO}

[94] L9: B \leftarrow LL \leftarrow \text{VALS}[S \leftarrow \text{IO} + \text{VALS}[1 + I]]

[95] M \leftarrow [L \leftarrow 2]

[96] B \leftarrow B \leftarrow \text{CNEST} [[(B / \text{SS}) \leftarrow \text{IO}] \leftarrow \text{VALS}[[S \leftarrow \text{IO}] + \text{IM}]

[97] \text{INDS}[I] \leftarrow \text{BI} 1

[98] \rightarrow \text{LAB}[I \leftarrow I + 1]

\[ V \]

4.

\[ V \text{ R} \rho \text{ P NNEST V;} \text{IO;} \text{E;} \text{I;} \text{N} \]

[1] \text{a NNEST is used to convert a numeric vector V into}

[2] \text{a a numeric nest. } \text{ P is the replication vector used}

[3] \text{a to partition the vector. That is,}

[4] \text{a } (\rho \text{P/0})=(\rho \text{V}). \text{ If P is a singleton, it is}

[5] \text{a replicated as much as necessary to encompass V.}

[6] \text{a Replicate left arg.:}

[7] \text{IO} \leftarrow 0

[8] \text{V} \leftarrow \text{V}

[9] \rightarrow (1 \downarrow \text{N} \leftarrow \rho \text{P} \leftarrow \rho \text{L} 1

[10] \text{a Repeat P if singleton:}

[11] \text{N} \leftarrow \rho \text{P} \leftarrow [(\rho \text{V}) \div \rho \text{P}

[12] \text{L} 1: I \leftarrow \leftarrow 1 \downarrow (1 + \text{N}), E \leftarrow \text{L} 1 + \text{P}

[13] \text{a Fill with 2s for now:}

[14] \text{R} \leftarrow (\text{N}, I), \text{E} \leftarrow 2

[15] \text{a Insert lengths:}

[16] \text{R}[I] \leftarrow \text{P}

[17] \text{IO} \leftarrow \text{L}

[18] \text{a E+MONIOTA P:}

[19] \text{E} \leftarrow \text{E} + \downarrow \rho \text{E} \leftarrow \rho \text{P} / -1 \downarrow 0, + \text{P}

[20] \text{IO} \leftarrow 0

[21] \text{a Insert data:}

[22] \text{R}[\text{E} + \rho \text{P}/I] \leftarrow \text{V}

\[ V \]
Chapter 16 Solutions
IRREGULAR ARRAYS

[WSID: NNFNS]

\[ \begin{align*}
\triangleright & \hspace{1em} R \leftarrow A \text{ NNCATSS } B; D; F \\
[1] & \text{ A NNCATSS is used to catenate two numeric vectors (i.e. A numeric nest } "\text{scalars}" \text{) to form a 2-segment numeric } \\
[2] & \text{ A nest.} \\
[3] & \text{ D} \leftarrow \rho A \leftarrow \rho B \leftarrow \rho C \\
[4] & \text{ R} \leftarrow \rho (2, 3), D, A, F, B \\
\end{align*} \]

\[ \begin{align*}
\triangleright & \hspace{1em} R \leftarrow A \text{ NNCATVS } B; D; E; L; \Omega O \\
[1] & \text{ A NNCATVS is used to catenate a numeric nest (A) to a numeric vector (B, i.e. a numeric nest } "\text{scalar}" \text{).} \\
[2] & \text{ L} \leftarrow \rho B \leftarrow \rho C \\
[3] & \text{ D} \leftarrow \rho \Omega O \leftarrow \rho D \\
[4] & \text{ E} \leftarrow \rho L \leftarrow \rho E \\
[5] & \text{ R} \leftarrow \rho (1, 2), (D + 2, L), A, (1 + E) \downarrow B \\
\end{align*} \]

\[ \begin{align*}
\triangleright & \hspace{1em} R \leftarrow A \text{ NNCATSV } B; D; E; F; G; \Omega O \\
[1] & \text{ A NNCATSV is used to catenate a numeric vector } \\
[2] & \text{ A (A, i.e. a numeric nest } "\text{scalar}" \text{) to a numeric nest (B).} \\
[3] & \text{ L} \leftarrow \rho F \leftarrow \rho D \\
[4] & \text{ D} \leftarrow \rho G \leftarrow \rho E \\
[5] & \text{ R} \leftarrow \rho (G + E, D), F + (1 + \rho A), ((1 + E) \downarrow A), (1 + G) \downarrow B \\
\end{align*} \]

\[ \begin{align*}
\triangleright & \hspace{1em} R \leftarrow A \text{ NNCATVV } B; D; E; F; G; \Omega O \\
[1] & \text{ A NNCATVV is used to catenate two numeric nests to form } \\
[2] & \text{ A longer numeric nest.} \\
[3] & \text{ D} \leftarrow \rho L \leftarrow \rho E \\
[4] & \text{ F} \leftarrow \rho G \leftarrow \rho B \\
[5] & \text{ R} \leftarrow \rho (G + E, D), F - 1 + \rho A, ((1 + E) \downarrow A), (1 + G) \downarrow B \\
\end{align*} \]
Chapter 16 Solutions

IRREGULAR ARRAYS

[WSID: NNFNS]

\[ \forall R \rightarrow A \] NNCAT B;D;E;F;G;L;IO

[1] A NNCAT is used to catenate two numeric nests
[2] A and/or vectors. The non-nest arguments are
[4] IO\rightarrow 0
[5] S\rightarrow S;V\rightarrow V;S\rightarrow V;V\rightarrow V;
[6] \rightarrow (L1,L2,L3,L4)(1=\rho A)+2x1=\rho B
[7] L1:D\rightarrow \rho A, A
[8] F\rightarrow \rho B,B
[9] R\rightarrow (2 3,D+4),D,A,F,B
[10] \rightarrow 0
[11] L2:L\rightarrow \rho B,B
[12] IO\rightarrow 1
[13] D\leftarrow E\rightarrow \rho A
[14] IO\rightarrow 0
[16] R\rightarrow (1+E,D,\rho A),(1+E)\downarrow A,L,B
[17] \rightarrow 0
[18] L3:L\rightarrow \rho A,A
[19] IO\rightarrow 1
[20] D\leftarrow E\rightarrow \rho B
[21] IO\rightarrow 0
[22] D\rightarrow B[D]
[23] R\rightarrow ((E+12),(D+2+L),L,A),(1+E)\downarrow B
[24] \rightarrow 0
[25] L4:IO\rightarrow 1
[26] D\leftarrow E\rightarrow \rho A
[27] F\leftarrow G\rightarrow \rho B
[28] IO\rightarrow 0
[29] D\rightarrow A[D]
[31] R\rightarrow ((G+E,D),F\rightarrow 1++A),((1+E)\downarrow A),(1+G)\downarrow B

[WSID: NNFNS]

\[ \forall R \rightarrow NEST NNIDX INDS;I;L;S;N \]

[1] A NNIDX is used to replace one or more segments
[2] A into a numeric nest (NEST). INDS is a
[3] numeric vector or scalar of the indices
[4] A of the segments to be replaced. If a vector,
[5] A the array replaced (assign) will be a
[6] A numeric nest. If a scalar, the array will be
[7] A a numeric vector. The function ASSIGN simply
[8] A assigns its right argument to the global
[10] A After being replaced, <assign> is erased.
[11] L\rightarrow NEST[IO\rightarrow S\rightarrow NEST[1+INDS]]
[12] A Branch if nest result:
[13] \rightarrow (0\not\rho N\rightarrow \rho INDS)\rho L1
[14] R\rightarrow NEST[(S+1)+\downarrow L]
[15] \rightarrow 0
[16] L1:L\rightarrow 1+L
Chapter 16 Solutions
IRREGULAR ARRAYS

\[ V \text{ NNIDX (continued)} \]

[17] A I=\text{MONIOTA L}:
[18] I+I+p\uparrow pL/\sim 1\downarrow 0, +L
[19] R+(N, +1\downarrow (1+N), L), \text{NEST}[I+L/S]

\[ \downarrow \]

[WSID: NNFNS]

\[ V \text{ R=A ASSIGN B} \]

[1] R=A ◦ assign=B

\[ \downarrow \]

[WSID: NNFNS]

\[ V \text{ R=NEST NNIDX A INDS; L; S; NL; N; NS; I; U} \]

[1] A NNIDX A is used to replace one or more segments
[2] A into a numeric nest (NEST). INDS is a
[3] A numeric vector or scalar of the indices of the
[4] A segments to be replaced. If a vector, the
[5] A array replaced (assign) will be a numeric
[6] A nest. If a scalar, the array will be a
[7] A numeric vector. The function ASSIGN simply
[8] A assigns its right argument to the global
[10] A After being replaced, <assign> is erased.
[12] A Branch unless a scalar index:
[13] →(x=pINDS)pL2
[14] L=\text{NEST}[I+O+S=\text{NEST}[1+INDS]]
[15] NL=pD, assign
[16] A Index assign if same length:
[17] →(L\neq NL)pL1
[18] R=\text{NEST}
[19] R[(S+1)+L]+D
[20] →L5
[21] L1: R→(S+1)+NL\text{NEST}
[22] I+1+INDS+\sim O+L\text{N-INDS}+\sim O
[25] R+R, D,(S+L+1)\downarrow \text{NEST}
[26] →L5
[27] A Branch if nothing to assign:
[28] L2:+(0=pINDS)pL3
[29] I+1+pL
[30] R+U+(D=assign)[I+O]
[31] L=\text{NEST}[I+O+S=\text{NEST}[I]]
[32] NL+D[I+O+NS=D[I+R]]:
[33] A Index assign if same length:
[34] →(L[INDS]v., \#NL)pL4
[35] R=\text{NEST}
[36] I+S[INDS]
[37] L+1+NL
[38] A N=\text{MONIOTA L}:
[39] N+N+pN+L/-\sim 1\downarrow 0, +L

-472-
\[ \text{NNIDX} \ (\text{continued}) \]

\begin{align*}
[40] & \text{R}[N+L/I] \leftarrow (U+1)D \\
[41] & \Rightarrow L5 \\
[42] & \text{L3} : \text{R} \leftarrow \text{NEST} \\
[43] & \Rightarrow L5 \\
[44] & \text{L4} : I[\text{INDS}] \leftarrow N + R \\
[45] & L \leftarrow 1 + (L, NL)[I] \\
[46] & S \leftarrow (S, NS + \rho \text{NEST})[I] \\
[47] & a \leftarrow \text{MONIOTA} \ L: \\
[48] & I \leftarrow I + \rho I + L / \sim 1 \downarrow 0, + \backslash L \\
[49] & R \leftarrow (N, + \backslash 1 \leftarrow (1 + N), L), (\text{NEST}, D)[I + L / S] \\
[50] & \text{L5} : \text{DERASE} \ '\text{assign}'
\end{align*}

[WSID: NNFNS]

\[ \text{L} \leftarrow \text{NNLEN} \ \text{NEST} : \Pi \text{IO} \]

1. \text{NNLEN} is used to return the lengths of the
2. \text{segments} of the numeric nest right argument.
3. \Pi \text{IO} \leftarrow 1
4. \text{L} \leftarrow \text{NEST}[1]
5. \Pi \text{IO} \leftarrow 0
6. \text{L} \leftarrow \text{NEST}[\text{NEST}[L]]

[WSID: NNFNS]

\[ \text{R} \leftarrow \text{NEST} \ \text{NNSUMCOL} \ \text{COLS} : I ; L ; N ; S \]

1. \text{NNSUMCOL} is used to sum the \text{N}th column of each
2. \text{M} column matrix item (raveled) in the numeric
3. \text{nest} \text{NEST}. \ \text{COLS} is \text{M}, \text{N}.
4. \text{N} \leftarrow \text{NEST}[\Pi \text{IO}]
5. \text{S} \leftarrow \text{NEST}[1 + \backslash N] + \Pi \text{IO}
6. \text{L} \leftarrow \text{NEST}[S] / \text{COLS}[\Pi \text{IO}]
7. \text{I} \leftarrow I + \rho I + L / \sim 1 \downarrow 0, + \backslash L
8. \text{R} \leftarrow \backslash \text{NEST}[L / S + (\sim \Pi \text{IO}) + \text{COLS}[1 + \Pi \text{IO}]] + \text{COLS}[\Pi \text{IO}] \times I - \Pi \text{IO}]
9. \text{R} \leftarrow \text{R}[(\sim \Pi \text{IO}) + \backslash L]
10. \text{R} \leftarrow \sim 1 \downarrow 0, \text{R}

\[ \text{\textbackslash } \]
Chapter 17 Solutions

CURVE FITTING

1. AMT: 60.62 59.57 56.70 60.42
   MAT: 4 4 32 61 15 82 35 104 10 82 37 83 5 85 25 62 14 85
   AMT\times MAT
   0.15 0.05 0.73 0.51

2. The formula is: \text{SUPPLY} = A + (B \times \text{TIME})

   The supply is exhausted when \text{SUPPLY} = 0, i.e. when:
   \[
   0 = A + (B \times \text{TIME})
   \]
   \[
   -A = B \times \text{TIME}
   \]
   \[
   \text{TIME} = -\frac{A}{B}
   \]

   Use \# to determine A and B and then plug into the last equation:
   \[
   \text{SUPPLY} = 1850 1772 1705 1508 1490 1250
   \]
   \[
   \text{TIME} = 1 2 3 5 6 9
   \]
   \[
   C = \text{SUPPLY} \div [1.5 \times \text{TIME}} \quad (C \text{ is } A, B)
   \]
   \[
   -C
   \]
   \[
   25.6
   \]

   The supply will be exhausted between weeks 25 and 26.

3. The formula is: \(R^2 = ((X-CX)^2) + ((Y-CY)^2)\)
   or: \(R^2 = (X^2) + (Y^2) + (CX^2) + (CY^2) + (-2XXCX) + (-2YCY)\)
   so: \((X^2) + (Y^2) = ((R^2) + (-CX^2) + (-CY^2)) + (2XXCX) + (2YCY)\)
Use 9 to determine the coefficients:

\[
\begin{align*}
X & \leftarrow 4 \ 3 \ 2 \ 4 \ 7 \ 6 \\
Y & \leftarrow 1 \ 2 \ 3 \ 5 \ 2 \ 4 \\
\text{LEFT} & \leftarrow (X*2)+(Y*2) \\
\text{MAT} & \leftarrow 1,2 \times X,[1.5]Y \\
C & \leftarrow \text{LEFT} \cdot \text{MAT} \\
D & \leftarrow CX \cdot C[2] \\
4.6 & \\
D & \leftarrow CY \cdot C[3] \\
2.9 &
\end{align*}
\]

Since: \( C[1] = (R*2)+(-CX*2)+(-CY*2) \)
then: \( (R*2) = C[1]+(CX*2)+(CY*2) \)
so:
\[
D \leftarrow (C[1]+(CX*2)+(CY*2)) \cdot 0.5
\]
\( 2.2 \)

Therefore, the center is (4.6,2.9) and the radius is 2.2.

4. For: \( T1 = C4+(C5 \times (R+L)) \)
\[
\begin{align*}
\text{C} & \leftarrow T1 \cdot 1, [1.5] R+L \\
C4 & \leftarrow C[1] \\
C5 & \leftarrow C[2]
\end{align*}
\]

For: \( T2 = C1+(R \times (C2+(C3 \times L))) \)
\[
\begin{align*}
T2 & \leftarrow C1+(C2 \times R)+(C3 \times R \times L) \\
\text{C} & \leftarrow T2 \cdot 1, R, [1.5] R \times L \\
C1 & \leftarrow C[1] \\
C2 & \leftarrow C[2] \\
C3 & \leftarrow C[3]
\end{align*}
\]
1. \[ 1000 \times (1 + 365 \text{ EFFECTIVE} \cdot .11) \times 1.5 \]
   \[= 1179.36 \]
   
or
   \[ 1000 \times (1 + .11 + 365) \times 365 \times 1.5 \]
   \[= 1179.36 \]

2. 
   \[ \text{TERM} \leftarrow 40 \div 52 \]
   \[ \text{PAY} \leftarrow 10 \times 52 \]
   \[ \text{PER} \leftarrow 52 \]
   \[ \text{PDEF} \leftarrow 0 \]
   \[ \text{DEFF} \leftarrow \text{TERM} \]
   \[ \text{CONV} \leftarrow 12 \]
   \[ \text{INT} \leftarrow .08 \]
   \[ \text{VALUE} \]
   \[= 412.84 \]
   
or
   \[ \text{V} \leftarrow (1 + .08 \div 12) \times 12 \]
   \[10 \times (\text{V} \times 40 \div 52) \div (1 - \text{V} \times 40 \div 52) \div 1 - \text{V} \div 52 \]
   \[= 412.84 \]

3. 
   \[ \text{PI} \leftarrow \text{SCHEDULE 4 12 12 0 12 .14} \]
   \[ (48 \text{ rows}) \]
   \[ \text{TOTPRIN} \leftarrow +/\text{PI}[:1] \]
   \[ \text{PI} \leftarrow \text{PI} \times 10000 + \text{TOTPRIN} \]
   \[ +/\text{PI}[1:] \]
   \[ \text{Loan amt. at } \$1 \text{ per month} \]
   \[ \text{Cvt. loan amt. to } \$10,000 \]
   \[ \text{Monthly pmt.; sum any row} \]
   \[= 273.26 \]
   \[+/\text{PI}[1:] \]
   \[= 3043.47 \]
   \[+/36\text{PI}[2:] \]
   \[= 2881.01 \]
   \[ \text{Repayment amt. is principal outstanding, i.e. remaining principal payments} \]
   \[ \text{Sum 3 years of interest pmts.} \]
Chapter 18 Solutions

4.\

\[
\begin{align*}
\text{DATES} & \leftarrow 19870101, 19870701, 101 + 10000 \times 1991 + 15 \\
\text{AMTS} & \leftarrow 10000 \times 10 - 5, 15, 15, 3 \\
\text{DATES} & \text{ IRR} \text{ AMTS}
\end{align*}
\]

0.1058

5.

[WSID: INTEREST]

\[\text{BV} \leftarrow \text{DATE FCBOOK PARAMS; COUP; CRATE; D; DAYS; F; MDATE; MVAL; N; NCOUPS; P; PAR; S; YLD}\]

[1] Returns book values as of DATE (YYYYMMDD), a scalar,
[2] for fixed-coupon securities defined in the matrix
[3] PARAMS, one row per security. Value is computed
[4] after coupon is received if DATE is a coupon date.
[5] Result has shape: (1\(\times\) PARAMS).
[6] PARAMS;
[7] PARAMS[;1] par value
[8] PARAMS[;2] maturity date (YYYYMMDD)
[9] PARAMS[;3] annual coupon rate
[10] PARAMS[;4] number of coupons per year
[12] PARAMS[;6] (optional) maturity value (par value
[13] if omitted)
[14] PARAMS[;1] MVAL \leftarrow PARAMS[;1]
[15] MDATE \leftarrow PARAMS[;2]
[16] CRATE \leftarrow PARAMS[;3] \times NCOUPS \leftarrow PARAMS[;4]
[17] YLD \leftarrow PARAMS[;5]
[18] \Rightarrow (5=1 \times \text{PARAMS}) \times \text{START}
[19] MVAL \leftarrow PARAMS[;6]
[20] \Rightarrow \text{Formula:}
[21] \Rightarrow \text{BV(t) = (MVAL} \times (1+Y)^{-W(t)} + \text{PAR} \times \text{CRATE} \times (1-(1+Y)^{-W(t)}
[22] \Rightarrow -W(t)) / Y
[23] \Rightarrow \text{where: BV(t) = book value at time t (a coupon date)
[24] Y = couponly yield rate
[25] W(t) = the number of (whole) coupon periods
[26] remaining from time t to MDATE
[27] \Rightarrow \text{Compute approx days (360 days/yr) from specified
[28] DATE to maturity (change 31 days to 30):
[29] \Rightarrow \text{START:DAYS} \leftarrow 360 30 1 + . \times 0 100 100 \times \text{MDATE-31=100!MDATE}
[30] \Rightarrow \text{DAYS} \leftarrow 360 30 1 + . \times 0 100 100 \times \text{DATE-31=100!DATE}
[31] \Rightarrow \text{No. coupon periods from specified date to maturity:
[32] N \leftarrow (\text{DAYS} \times \text{NCOUPS}) / 360
[33] \Rightarrow \text{Coupon payment:
[34] COUP \leftarrow \text{PAR} \times \text{CRATE}
[35] \Rightarrow \text{Book value at prior coupon:
[36] D \leftarrow (1+YLD)^{-N}
[37] \Rightarrow \text{P} \leftarrow (\text{MVAL} \times D) + \text{COUP} \times (1-D) / YLD
[38] \Rightarrow \text{477-}
Chapter 18 Solutions

FINANCIAL UTILITIES

\[ D \times (1 + YLD) \times -1N \]
\[ S + (MVAL \times D) + COUP \times (1 - D) \div YLD \]
\[ F \times N \times -1N \]

Interpolate between prior and subsequent book vals:
\[ BV \times (P \times F) + S \times 1 - F \]
Chapter 19 Solutions

EXCEPTION HANDLING

1. APL*PLUS:

\[
\text{\texttt{v SHOWFILE NAME;LIM;N;\textsc{dalx};\textsc{delx}}}
\]

[1] \text{\texttt{\textsc{dalx}←\textsc{delx}←'DDM'}}
[2] \text{\texttt{LIM←TIEFILE NAME}}
[3] \text{\texttt{\textsc{dalx}←'ASK'}}
[4] \text{\texttt{\textsc{delx}←'( 'ATTN' '∧. = 4↑DDM')/ASK} \textsc{ddm'}}
[5] \text{\texttt{ASK:0←'BEGIN WITH WHICH RECORD?'}}

etc.

SHARP APL:

\[
\text{\texttt{v SHOWFILE NAME;LIM;N;\textsc{dtrap}}}
\]

[1] \text{\texttt{\textsc{dtrap}←''}}
[2] \text{\texttt{LIM←TIEFILE NAME}}
[3] \text{\texttt{\textsc{dtrap}←'V1000 C →ASK ∨0 C →(' 'ATTN' '∧. = 4↑5↓DER[DIO;])/ASK'}}
[4] \text{\texttt{ASK:0←'BEGIN WITH WHICH RECORD?'}}

etc.

APL2: Cannot be solved since attentions are not detected as exceptions

2. [WSID: INPUT]

\[
\text{\texttt{v R←NUM NxPROMPTE PROMPT;\textsc{delx}}}
\]

[1] \text{\texttt{\textsc{a} Displays character vector PROMPT, allows keyboard}}
[2] \text{\texttt{\textsc{a} input on same line and returns numeric vector}}
[3] \text{\texttt{\textsc{a} response of length NUM (or of any length if NUM=0)}}.
[4] \text{\texttt{\textsc{a} Returns numeric scalar escape code if escape word}}
[5] \text{\texttt{\textsc{a} entered. Allows and executes primitive APL}}
[6] \text{\texttt{\textsc{a} expressions. Requires: CPROMPTE.}}
[7] \text{\texttt{\textsc{a} APL*PLUS version.}}
[8] \text{\texttt{\textsc{a} SHARP APL: localize \textsc{dtrap} instead of \textsc{delx}.}}
[9] \text{\texttt{\textsc{a} APL2: localize neither \textsc{delx} or \textsc{dtrap}.}}
[10] \text{\texttt{\textsc{delx}←'DDM' \textsc{a} APL*PLUS}}
[11] \text{\texttt{\textsc{dtrap}←'' \textsc{a} SHARP APL}}
[12] \text{\texttt{L1:R←CPROMPTE PROMPT}}
3. Below are the SHARP APL (ERRATTNS) and APL*PLUS (ERRATTNP) solutions to the problem. The problem cannot be solved with APL2 because attentions are not considered exceptions (to be handled) and because the DEA and DEC system functions are not designed to allow the type of "environment conditioning" required in the problem.

```apl
V ELX ERRATTNS ALX
[2] V Sets UTRAP so that error will cause →ELX at
[3] V the SI level at which UTRAP is local; and
[4] V break will cause →ALX at that level. If ELX
[5] V or ALX is a character vector, it is executed
[6] V at the level at which the error or attention
[7] V occurs. If ELX or ALX is empty, no special
[8] V error or attention handling is included in
[9] V the local UTRAP. Be sure to localize UTRAP.
```
\[V\] ERRATTNS (continued)


[11] \(\text{DTRAP}+\)'

[12] \(\rightarrow (0 \in E L X) \rho L 2\)

[13] A Branch if numeric ELX:

[14] \(\rightarrow (0=1 \in E L X) \rho L 1\)

[15] \(\text{DTRAP}' V 0 E ', E L X\)

[16] \(\rightarrow L 2\)

[17] \(L 1: \text{DTRAP}' V 0 C \rightarrow ', \# E L X\)

[18] \(L 2: \rightarrow (0 \in E L X) \rho 0\)

[19] A Branch if numeric ALX:

[20] \(\rightarrow (0=1 \in E L X) \rho L 3\)

[21] \(\text{DTRAP}+\text{DTRAP}, 'V 1000 E ', A L X\)

[22] \(\rightarrow 0\)

[23] \(L 3: \text{DTRAP}+\text{DTRAP}, 'V 1000 C \rightarrow ', \# A L X\)

\[V\]

\[\text{WSID: ERROR}\]

\[V\] ELX ERRATTNP ALX


[2] A Sets \(\# E L X\) and possibly \(\# A L X\) so that error

[3] A will cause \(\# E L X\) at the SI level at which

[4] A \(\# E L X\) is local; and break will cause \(\# A L X\)

[5] A at the SI level at which \(\# A L X\) is local (uses

[6] A \(\# E R R O R\) to percolate to proper level). If

[7] A \(\# E L X\) or \(\# A L X\) is a character vector, it is

[8] A executed at the level at which the error or

[9] A attention occurs. If \(\# E L X\) is empty, \(\# E L X\) is

[10] A set to 'ODM'. If \(\# A L X\) is empty, \(\# A L X\) is set

[11] A to 'ODM' if \(\# A L X\) is localized in the calling

[12] A function, and otherwise is not modified. Be


[14] A Localize \(\# A L X\) only if \(\# A L X\) is not empty.

[15] \(\rightarrow (0 \in E L X) \rho L 2\)

[16] A Branch if numeric ALX:

[17] \(\rightarrow (0=1 \in E L X) \rho L 1\)

[18] A Character ALX:

[19] \(\# A L X \rightarrow A L X\)

[20] \(\rightarrow L 3\)

[21] A Numeric ALX:

[22] \(L 1: \# A L X \rightarrow ' t (\# L 1=1 \in \# \text{IDLOC}' 'A L X')/ '/' ' \# E R R O R''''A T T N'''''''''''\rightarrow ' ' , \# A L X\)

[23] \(\rightarrow L 3\)

[24] A Empty ALX; branch if \(\# A L X\) not local to calling fn:

[25] \(L 2: \rightarrow (\# L 1=1 \downarrow 1, \# \text{IDLOC} ' ' A L X') \rho L 3\)

[26] \(# A L X \rightarrow 'ODM'\)

[27] \(L 3: \rightarrow (0 \in E L X) \downarrow L 4\)

[28] \(E L X \rightarrow 'ODM'\)

[29] A Branch if numeric ELX:

[30] \(L 4: \rightarrow (0=1 \in E L X) \rho L 6\)

[31] A Branch if non-empty numeric ALX:

[32] \(\rightarrow (0 \neq 1 \in E L X) \lor 0 \in E L X) \downarrow L 5\)
\( \triangledown \text{ERRATTNP (continued)} \)

[33] A Character ELX; character or empty ALX:
\[ \text{DELX} \rightarrow \text{ELX} \]

[36] A Character ELX; numeric ALX:
\[ \text{L5:} \text{DELX} \rightarrow \phi (\sim1\text{IDLOC}'\text{DELX}')^{'} \text{ATTN}'^{.}=4\text{DM})/'' \rightarrow \phi (\text{ALX}),''\phi \text{ERROR}'\text{ATTN}'^{.}=4\text{DM})/4\text{DM}',\text{ELX} \]

[38] \( \rightarrow 0 \)

[40] A Numeric ELX; character or empty ALX:
\[ \text{DELX} \rightarrow \phi (\sim1\text{IDLOC}'\text{DELX}')/''\phi \text{ERROR}(\sim\text{DM} \# \text{TCNL})/\text{DM }'' \rightarrow ' \phi \text{ELX} \]

[42] \( \rightarrow 0 \)

[43] A Numeric ELX; numeric ALX:
\[ \text{L7:} \text{DELX} \rightarrow \phi (\sim1\text{IDLOC}'\text{DELX}')/''\phi \text{ERROR}(\sim\text{DM} \# \text{TCNL})/\text{DM }'' \rightarrow (\text{''ATTN}'^{.}=4\text{DM})\phi ',\phi (1,\text{ELX}),1,\text{ALX} \]

\( \triangledown \)
INDEX

accumulation 92
acknowledgements 3
ADDEMP (wsid: MSF) 177
age computation 139
DAI, accounting information 23
DALX, attention latent expression 308
amortization 302
ANDRED (wsid: REDUCE) 344
annuity 285
APL files 203
APL2 1
APL*PLUS 1
ASSIGN (wsid: CNFNS) 463
ASSIGN (wsid: MULTIFLO) 423
ASSIGN (wsid: NNFNS) 472
attention detection 308
attention handling 154
autostarting 185
DAV, atomic vector 86 151 245 260

BDAM 447
best fit 278
blank manipulation 40
bond accrued interest 144
bond calculations 297
book value 301
Boolean storage 34 105
Boolean techniques 105 230 244
BOOLEAN ws 316
branching 8 172
BY (wsid: SEARCH) 76

C 37
canonical representation 26 189
CASHBAL ws 316
CASH1 (wsid: CASHBAL) 16
CASH2 (wsid: CASHBAL) 17
CASH3 (wsid: CASHBAL) 18
CASH4 (wsid: CASHBAL) 325
CATEMP (wsid: MSF) 178
CATREC 216
CATREC (wsid: MULTIFLO) 392
CATRECWS 216
CATRECWS (wsid: MULTIFLO) 396
CENTER (wsid: FORMAT) 330
centering 50
CGRADEUP 86 267
CGRADEUP (wsid: SORT) 60
CGRADEUP1 (wsid: SORT) 57
CGRADEUP2 (wsid: SORT) 59
character data positioning 40
character matrix searching 64
character matrix sorting 56
character nest 254
character substring replacement 76
character substring searching 75
CINPUT (wsid: MSF) 178
circle 280
CJUST (wsid: FORMAT) 42
classification 92
CMIOTA 39 85 280
CMIOTA (wsid: SEARCH) 69
CMIOTA1 (wsid: SEARCH) 65
CMIOTA2 (wsid: SEARCH) 66
CNCAT (wsid: CNFNS) 462
CNEQ (wsid: CNFNS) 466
CNEST (wsid: CNFNS) 459
CNFNS ws 316
CGRADEUP (wsid: CNFNS) 465
CNIDX (wsid: CNFNS) 463
CNIDXA (wsid: CNFNS) 463
CNIOTA (wsid: CNFNS) 467
CNLEN (wsid: CNFNS) 465
CNA (wsid: CNFNS) 460
CNAV (wsid: CNFNS) 461
COBOL 37 157
coding 162 170
collating sequence 56
COLUMNIZE (wsid: FORMAT) 331
comments 171
COMMENTS ws 316
compiled functions 37
compiler 19 37
compounding 283
COMPRESS 218
COMPRESS (wsid: MULTIFLO) 409
compressed Milky-Way array 112
conditional branching 8
CONTINUE workspace 186
continuous payments 287
COST (wsid: TIMING) 327
CPROMPT (wsid: INPUT) 119
CPROMPTE (wsid: INPUT) 124
CPU time 23
DCA, canonical representation 189 245
cross-tabulation 92
CRTIMING ws 316
CRAVR 193 198 245
CRAVR (wsid: FNREP) 190
DCT, comparison tolerance 59 64 65 72 449
curve fitting 271
data storage 34
date formatting 133
date manipulation 135
date representation 129
date utility functions 136
date validation 131
DATES ws 317
day of week 144
DEB (wsid: FORMAT) 41
DEB (wsid: SEARCH) 336
DDEF, define 26 198
DELEMP (wsid: MSF) 178
deletion flag 207 227
delivery 163 174
DELREC 218
DELREC (wsid: MULTIFLO) 407
diagonal 82 89
diamond 24
diamonds 29
directory files 205
distinct values 84
DLB (wsid: FORMAT) 41
DDM, diagnostic message 304
DDR, data representation 260
DROPFN (wsid: FNSFILE) 385
DTB 256
DTB (wsid: FORMAT) 41
DEA, execute alternate 121 305
each 12
DEC, execute controlled 306
DEC, error conditioning 307
EFFECTIVE (wsid: INTEREST) 284
efficiency 23 33
DELX, error latent expression 304
DEM, event message 305
embedded assignment 55
EMMA 216
EMPLOYEES (wsid: FLF) 176
EMPLOYEES (wsid: MSF) 178
EMPLOYEES (wsid: MULTIFLO) 390
enclose 255
DER, event report 305
ERRATTNP (wsid: ERROR) 481
ERRATTNS (wsid: ERROR) 480
ERROR 307
error detection 304
error messages 173
error signalling 307
ERROR ws 317
escape from input 123
ESCAPE (wsid: INPUT) 125
DES, event simulation 308
DET, event type 308
evaluated input 120
exception handling 303
execute 12 25 34 119
EXECUTE 109 220
EXECUTE (wsid: MULTIFLO) 436
familiarization 158 165
FCBOOK (wsid: INTEREST) 477
FCYIELD (wsid: INTEREST) 298
FDF, function definition 26 188 198 245
DFI, fix input 120
fields 215
file block size 226
file design 160 166 203 208
file documentation 212
file efficiency 210
File Manager 216
file organization 204
file utilities 203 215 225
FILEDOC 214
FILEDOC ws 317
FILEDOC (wsid: FILEDOC) 386
files, inverted 210
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>files, layered</td>
<td>211</td>
</tr>
<tr>
<td>financial utilities</td>
<td>282</td>
</tr>
<tr>
<td>first difference</td>
<td>95</td>
</tr>
<tr>
<td>PLF</td>
<td>180</td>
</tr>
<tr>
<td>PLF ws</td>
<td>317</td>
</tr>
<tr>
<td>floating point storage</td>
<td>34 59 103</td>
</tr>
<tr>
<td>floppy disk</td>
<td>316</td>
</tr>
<tr>
<td>flowcharting</td>
<td>162 169</td>
</tr>
<tr>
<td>OFMT, format</td>
<td>133 135 145 151</td>
</tr>
<tr>
<td>PNCREATE (wsid: PNSFILE)</td>
<td>383</td>
</tr>
<tr>
<td>PNIDS ws</td>
<td>317</td>
</tr>
<tr>
<td>PNREP ws</td>
<td>317</td>
</tr>
<tr>
<td>PNSFILE ws</td>
<td>317</td>
</tr>
<tr>
<td>FOR</td>
<td>221</td>
</tr>
<tr>
<td>FORE (wsid: MULTIFLO)</td>
<td>436</td>
</tr>
<tr>
<td>forecasting</td>
<td>274</td>
</tr>
<tr>
<td>format by example</td>
<td>133 135 145 152</td>
</tr>
<tr>
<td>FORMAT ws</td>
<td>317</td>
</tr>
<tr>
<td>formatting by row</td>
<td>50</td>
</tr>
<tr>
<td>formatting with newlines</td>
<td>151</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>157</td>
</tr>
<tr>
<td>frequency counts</td>
<td>92</td>
</tr>
<tr>
<td>FROMDAYS (wsid: DATES)</td>
<td>141</td>
</tr>
<tr>
<td>FROMDAYSΔ (wsid: DATES)</td>
<td>142</td>
</tr>
<tr>
<td>FROMDAYS360 (wsid: DATES)</td>
<td>143</td>
</tr>
<tr>
<td>FROMMDY (wsid: DATES)</td>
<td>139</td>
</tr>
<tr>
<td>FROMMDYΔ (wsid: DATES)</td>
<td>140</td>
</tr>
<tr>
<td>FROMQTS (wsid: DATES)</td>
<td>143</td>
</tr>
<tr>
<td>FROMYD (wsid: DATES)</td>
<td>365</td>
</tr>
<tr>
<td>function documentation</td>
<td>188</td>
</tr>
<tr>
<td>function identifiers</td>
<td>194</td>
</tr>
<tr>
<td>function size</td>
<td>182</td>
</tr>
<tr>
<td>functions on file</td>
<td>228</td>
</tr>
<tr>
<td>future value</td>
<td>285</td>
</tr>
<tr>
<td>DFX, fix</td>
<td>26 198 309 311</td>
</tr>
<tr>
<td>GETFN (wsid: PNSFILE)</td>
<td>385</td>
</tr>
<tr>
<td>global changes</td>
<td>181</td>
</tr>
<tr>
<td>global passing</td>
<td>183</td>
</tr>
<tr>
<td>global variables</td>
<td>170</td>
</tr>
<tr>
<td>grade vector</td>
<td>53</td>
</tr>
<tr>
<td>grade-up, dyadic</td>
<td>56 63 67 86 96 267</td>
</tr>
<tr>
<td>J vectors</td>
<td>37</td>
</tr>
<tr>
<td>Julian dates</td>
<td>144</td>
</tr>
<tr>
<td>justification</td>
<td>42 47 148</td>
</tr>
<tr>
<td>keywords</td>
<td>167</td>
</tr>
<tr>
<td>label vector</td>
<td>11</td>
</tr>
<tr>
<td>labels</td>
<td>172</td>
</tr>
<tr>
<td>LAYERS</td>
<td>221</td>
</tr>
<tr>
<td>LAYERS (wsid: MULTIFLO)</td>
<td>445</td>
</tr>
<tr>
<td>DLC, line counter</td>
<td>31 187 309</td>
</tr>
<tr>
<td>leap years</td>
<td>138</td>
</tr>
<tr>
<td>least squares</td>
<td>273</td>
</tr>
<tr>
<td>limbering up</td>
<td>4</td>
</tr>
<tr>
<td>link</td>
<td>256</td>
</tr>
</tbody>
</table>
INDEX

LIOTA (wsid: SEARCH) 71
LIOTA1 (wsid: SEARCH) 74
LISTEMP (wsid: MSF) 179
LJUST (wsid: FORMAT) 42
loan amortization schedule 290
LOCALIZE 197
LOCALIZE (wsid: FNIDS) 455
logical partition operations 237
logical reductions 233
logical scalar functions 231
logical scans 234
loop avoidance 15
LOOP ws 317
LOOPI (wsid: LOOP) 14
looping 8 11 36 172
looping primitive 13
lower limits 72
LPROMPT (wsid: INPUT) 362
DLX, latent expression 168 170

major-to-minor sort 54
map operations 236
maps 234
matrix nest 254
matrix sorting 55
maximum scan 101
MAXRED (wsid: REDUCE) 342
mean squared error 273 278
MESSAGE (wsid: INPUT) 125
MESSAGE (wsid: MSF) 179
UMF, monitor facility 33
Milky-Way reduction 111
Milky-Way reduction on files 113
MINRED (wsid: REDUCE) 343
MONIOTA (wsid: UTILITY) 48
mortality tables 90
MSF 180
MSF ws 318
MULTIFLO ws 318
MULTISA ws 318
MULTI2 ws 318
multi-set transposed files 206
222
multi-target branching 9

naming conventions 171
nested array emulation 257
nested arrays 253
Newton-Raphson Method 295
NEXTI (wsid: LOOP) 15
NINPUT (wsid: INPUT) 121
NINPUT (wsid: MSF) 179

NINPUT2 (wsid: INPUT) 122
UNL, name list 188
NNCAT (wsid: NNFNS) 471
NNCATSS (wsid: NNFNS) 470
NNCATSV (wsid: NNFNS) 470
NNCATVS (wsid: NNFNS) 470
NNCATVV (wsid: NNFNS) 470
NNEST (wsid: NNFNS) 469
NNFNS ws 318
NNIDX (wsid: NNFNS) 471
NNIDX (wsid: NNFNS) 472
NNLEN (wsid: NNFNS) 473
NNSUMCOL (wsid: NNFNS) 473
NOMINAL (wsid: INTEREST) 285
nonlinear curve fitting 277
NPROMPT (wsid: INPUT) 126
NPROMPT2 (wsid: INPUT) 127
NRECARecl 447
numeric nest 261
NXPROMPT (wsid: INPUT) 479
n-way logical reduction 104
n-way max/min reduction 100
n-way plus reduction 97
n-way reduction on files 109
n-way reductions 93

OBFUSCATE 367
origin 173
ORRED (wsid: REDUCE) 345
output 172

packing columns 57
pANDMAP (wsid: BOOLEAN) 244
pANDRED (wsid: BOOLEAN) 240
pANDSCAN (wsid: BOOLEAN) 240
partition utilities 238
partition vector 238 258
pEPMAP (wsid: BOOLEAN) 243
pEQSCAN (wsid: BOOLEAN) 242
perpetuity 287
pGEMAP (wsid: BOOLEAN) 243
pGTMAP (wsid: BOOLEAN) 243
pLESCAN (wsid: BOOLEAN) 241
pLTSCAN (wsid: BOOLEAN) 241
PLUSRED (wsid: REDUCE) 341
pNEMAP (wsid: BOOLEAN) 242
pNESCAN (wsid: BOOLEAN) 242
poles 234
pORMAP (wsid: BOOLEAN) 243
pORRED (wsid: BOOLEAN) 240
pORSSCAN (wsid: BOOLEAN) 241

-486-
INDEX

pPLUSRED (wsid: BOOLEAN) 239
precision 58 103
present value 285
print files 152
PRINT (wsid: PRTFILE) 155
programming standards 165
PROPOSAL (wsid: INPUT) 363
prototype 12
PRTFILE ws 318
PUTFN (wsid: FNSFILE) 384

QDOC 214
QDOC ws 318
QDOC (wsid: QDOC) 189
quad input 116 173
quad output 117
quadratic formula 275
quad-divide 271
quote-quad input 116
quote-quad output 118

range searching 70
ranking vector 73 322
rate of return 293
RCAT (wsid: MSF) 179
readability 38
record indices 216
record oriented files 204
records 215
recursion 12
REDUCE ws 318
reduction, dyadic 106
RELABEL 197 244
RELABEL (wsid: FNIDS) 249
REPL (wsid: UTILITY) 3
REPLACE (wsid: SEARCH) 77
report headings 51
report writing 145
RESTART (wsid: MSF) 187
restartability 173
restarting 186
RJUST (wsid: FORMAT) 42
row formatting 149
ROWFMT (wsid: FORMAT) 331

same line input 119
sample terminal sessions 160
DSA, stop action 310
scalar operations 35
scattered point indexing 82 97

SCHEDULE (wsid: INTEREST) 292
SEARCH ws 318
searching 53
SELECT 219
SELECT (wsid: MSF) 179
SELECT (wsid: MULTIFLO) 415
selection 80
selection assignment 81
SELECTWS 219
SELECTWS (wsid: MULTIFLO) 419
self-containment 182
shared variables 203
SHARP APL 1
shift and compare 236
DSIGNAL 307
singular matrix 273
SLASHIOTARHO 217
SLASHIOTARHO (wsid: MULTIFLO) 405
Smith, Robert A. 239
sort and shift 85 95
SORT ws 319
sorting 53
sorting character nests 267
specification 159 166
SQZEMP (wsid: MSF) 180
START (wsid: MSF) 186
state indicator cutting 311
state indicator depth 184
strand notation 255
subfunction design 169 175
successive approximations 294
suspending the function 309
system development procedure 157
system of linear equations 272
tads 36
UTC, terminal control 152 245
UTCBS, backspace 151
UTCNL, newline 151 245
testing 162 170 174
throw-away code 157
time diagram 287
TIMER (wsid: CRTIMING) 327
TIMER (wsid: TIMING) 27
TIMEDEFINE (wsid: TIMING) 450
TIMEDISPLAY (wsid: TIMING) 32
TIMEARESET (wsid: TIMING) 32
timing function lines 28
TIMING ws 319
timings 23
TITLES 148
TITLES (wsid: FORMAT) 49
TODAYS (wsid: DATES) 140
TODAYS\^ (wsid: DATES) 140
TODAYS360 (wsid: DATES) 143
TOMDY (wsid: DATES) 139
TOMDY\^ (wsid: DATES) 139
TOQTS (wsid: DATES) 143
TOYD (wsid: DATES) 364
training 163 174
transposed files 205
transpose, dyadic 44 82
\^TRAP 305
tree diagram 201
UTS, time stamp 137
typing 162 170

UIOTA (wsid: SEARCH) 72
UIOTAT (wsid: SEARCH) 75
UNCOMMENT 198
UNCOMMENT (wsid: COMMENTS) 458
UNDIAMOND 368
unique values 84
UNLAMP 367
UNLESS (wsid: MSF) 180
UNOBFUSCATE 367
UNQCM (wsid: SEARCH) 339
UNQCV (wsid: SEARCH) 338
UNQI1 (wsid: SEARCH) 339
UNQI0 (wsid: SEARCH) 339
UNQNV (wsid: SEARCH) 338
unsorting 89
upper limits 72
UPPERCASE (wsid: SORT) 63
uppercase/lowercase searching 79
uppercase/lowercase sorting 62
USEDBY 201
USEDBY ws 319
USEDBY (wsid: USEDBY) 376
user documentation 161 168
utility functions 180 316
UTILITY ws 319

VALUE (wsid: INTEREST) 288
VAPLFILE 203 446
vector notation 255
\^VI, validate input 120

visual representation 26 188 244
\^VR, visual representation 188
\^VR\^CR 198
\^VR\^CR (wsid: FNREP) 199

word wrap 21
workspace cleanups 35
workspace design 160 167 175
workspace documentation 175 191
workspace environment 192
workspace identifiers 200
WP ws 319
WRAP (wsid: WP) 325
WRAPLP (wsid: WP) 22
WSDOC 214
WSDOC ws 319
WSDOC (wsid: WSDOC) 368

XTAB 110

yield to maturity 297
\^ \ (wsid: TIMING) 31

\^AND (wsid: REDUCE) 358
\^ANDRED (wsid: REDUCE) 349
\^ANDWAY (wsid: REDUCE) 354
\^MAX (wsid: REDUCE) 357
\^MAXRED (wsid: REDUCE) 347
\^MAXWAY (wsid: REDUCE) 352
\^MIN (wsid: REDUCE) 357
\^MINRED (wsid: REDUCE) 348
\^MINWAY (wsid: REDUCE) 353
\^OR (wsid: REDUCE) 359
\^ORRED (wsid: REDUCE) 350
\^ORWAY (wsid: REDUCE) 355
\^PLUS (wsid: REDUCE) 356
\^PLUSRED (wsid: REDUCE) 346
\^PLUSWAY (wsid: REDUCE) 351
\^SS (wsid: SEARCH) 76