BALM

THE BALM PROGRAMMING LANGUAGE

MALCOLM C. HARRISON

STEPHANIE BROWN

September 1974

COURANT INSTITUTE OF MATHEMATICAL SCIENCES

New York University

/ BALM

THE BALM PROGRAMMING LANGUAGE

MALCOLM C. HARRISON

"

STEPHANIE BROWN

September 1974

COURANT INSTITUTE OF MATHEMATICAL SCIENCES

New York University

The purpose of this manual is to give an informal introduction to the programming language BALM implemented on the CDC 6600 at the Courant Institute at New York University. This is a powerful language, possessing a number of highly sophisticated features, including vector, string, and list processing, programs as data, and extendibility.

Table of Contents

Chapter

Appendix

CHAPTER 1. BASIC PROGRAM COMPONENTS

1.1 Variables, Values, Names, and Constants

A BALM program may refer to several kinds of data-objects. Some of these are constants which do not vary during the execution of the program. In this chapter we will deal mainly with integers. Constant integers can be written in the usual way. Examples of integer constants are

10, 246, -5.

Description of other types of data-object are postponed until Chapter 2.

Variables in a BALM program are used to refer to data-objects whose values can vary during the execution of the program. Each variable has a name, which is used to identify it, and a value, which may be changed by the execution of theprogram. The value of any variable can be any type of data-object which can be created by a BALM program. That is, in BALM a variable does not have a type associated with it, as in many other programming languages. In BALM the type of a data-object is determined by the data-object itself.

A variable name is written as a sequence of letters or digits starting with a letter. There are some exceptions which permit other characters to appear in names. But we shall postpone discussion of these for later chapters. Names with more than 8 characters are truncated by BALM with a warning message. Thus X, ABC, P1234 are all names of variables. ALONGNAME is also a variable but it would be seen by BALM as ALONGNAM. A variable can be given a new value in a number of ways, but the simplest is by use of an assignment expression. For example, the expression

ABC=123

would assign to the variable whose name was ABC the number 123 as its value. Subsequent references to ABC would then be taken to refer to this number, until its value was changed by a further assignment. The expression

XYZ19 = ABC

would assign to the variable whose name was XYZ19 the current value of the variable whose name is ABC.

1.2 Expressions and Values and Operators

Expressions are the most impprtant components of a BALM program. A constant or a variable is an expression, and expressions can be combined with operators to make other expressions. Integer operators include multiply, divide, add, and subtract. The value of an expression is the value obtained by evaluating the expression using the current values of the variables. For example, if the value of the variable named ABC is 123, the value of the expression

ABC + 5

is the number 128, while the value of the expression

5 + ABC/3

is 46. The operators * and / are used to indicate multiplication and division respectively. The evaluation of an expression involving multiplication and addition is done so that the multiplication is done before the addition. In accordance with the ordinary rules of arithmetic, parentheses can be used to modify this order, so that the value of

(5 + ABC)/4

is 128/4, i.e. 32. That is, the expression inside parentheses is always evaluated first.

To give more flexibility in writing expression which are functions of more than two other expressions, the standard functional notation is used. For example

MAX(X+1,Y,2*Z)

is an expression whose value is given by applying the function named MAX to the values of the expressions X+1, Y, and 2*Z, which are refe-red to as the arguments of MAX. The same notation can be used for applying functions to one or two arguments, so that the expression:

SQRT(2*Z)

will have the value which will result from applying the function named SQRT to the value of the expression 2*Z.

In BALM there are functions which may have other effects than simply returning a result. The more general term procedure is usually used for such functions. In fact, some procedures may not even return any value, or may always return the same value. Such procedures are usually evaluated for their effect rather than their value. One such procedure in BALM is the PRINT procedure which will print out the values of its arguments and whose value is the value of its last argument. For example, execution of

PRINT(A+1)

will print out 21 if the value of A is 20.

In addition to facilities for manipulating integers, BALM can handle strings of characters, true and false, and various ways of specifying collections of these objects. For simplicity, however, we will restrict consideration for the time being to integers, which are very simple and familiar to most people.

1.3 Assignment

In BALM every expression has a value, though sometimes this value is inaccessible to the programmer. The assignment expression used above to show how a variable may receive a value is an expression whose value is the value of the expression used on the right-hand side of the = , and whose evaluation has the side-effect of assigning this value to the left-hand side. Evaluation of the right-hand side takes place before the assignment, so

COUNT= COUNT + 1

will take the current value of COUNT, add 1 to this value, and assign the result to the variable COUNT, thus increasing the value assigned to COUNT by 1. Note that

K = COUNT + 1

will use the value of COUNT to calculate the value of K, but will not change the value of COUNT. Since the assignment expression has a value, evaluation of the expression

A = (B = 1)

will have the value 1, and the side effect of assigning 1 to A and B.

1.4 Program Setup

BALM programs may either be punched on cards and submitted for batch processing or, if available at your installation, typed in at an interactive terminal. Suppose for the moment such a terminal is available and you have logged in and are ready to enter a BALM program. A semicolon after an expression is a signal to the BALM system to evaluate the expression it has just read, and then to read the next expression. Any line with an * in column 1 is treated as a comment.

Examples

Suppose the problem that we wish to solve is that of calculating the cost of painting a room. We can do this by calculating the area of the walls, and then multiplying this by the cost of the paint. For simplicity, suppose the room has one wall 8 by 16 and one wall 9 by 17, so we can find the area by typing in:

```
AREA = 8*16*9*17;
           +
```

This will be executed, so we can then calculate the cost by multiplying the area by the cost per unit area, say 2 cents per square foot:

```
COST = 2*AREA;
```

and then printing out the result

```
PRINT(COST);
```

will print out 562 on the teletype. If we have $7.00 to spend, we might then want to consider if we can also paint the closet, an additional area of 6 by 8. We can then type in:

```
PRINT(COST+2*6*8);
```

which will print out:

658

so we can make it, with 42 cents left for a beer.

If we don't have access to a teletype, we could have punched the same program on cards as follows:

```
*  PROGRAM TO DETERMINE COST OF PAINTING A ROOM
AREA = 8*16*9*17;
COST = 2*AREA;
PRINT(COST);
PRINT(COST+2*6*8);
```

The output from the program would be printed on the line printer as the original program with a blank line separating the expressions, and with any printed output interspersed in the appropriate place. In the present example this would be as follows:

```
AREA = 8*16*9*17;

COST = 2*AREA;

PRINT(COST);
562

PRINT(COST+2*6*8);
658
```

1.5 Conditional Expressions

In very simple problems it is possible to write programs in which the expressions evaluated are independent of the data. In most interesting problems, however, this is not possible, and the programmer must be given some mechanism to permit the sequence of commands to be dependent on the values of expressions.

The usual way in which the course of the calculation can be made to depend on the objects being processed is by use of conditional expressions. These are usually of the form:

IF X THEN Y ELSE Z

where X is an expression whose value is to be tested, and Y and Z are arbitrary expressions. Expressions such as X are called logical expressions, and are considered to be false if their value is NIL, and true otherwise. If X is true, Y is evaluated, but if X is false, Z is evaluated.

There are a number of operators which can be used to construct logical expressions. In general, they return the value TRUE or NIL, and we will refer to them as predicates. The following predicates can be used to compare the values of two integers.

GT LT GE LE EQ NE

These are used to test if an integer is greater than, less than, greater than or equal to, less than or equal to, equal to, or not equal to another integer. For example

```
IF X EQ Y THEN Z=1 ELSE Z=2;
```

will set Z to 1 if X and Y have the same value, and to 2 otherwise. The value of a conditional is the value of the expression which is selected. For example,

```
Z = IF X EQ Y THEN 1 ELSE 2;
```

could have been written instead of the example above. Other predicate operators will be introduced later.

The ELSE clause can be left out, in which case the conditional behaves as if it had an

```
ELSE NIL
```

appended to it. For example

```
IF X EQ Y THEN Z=1;
```

will set Z to 1 if X and Y have the same value, and leave the value of Z unchanged otherwise.

An example of the use of simple conditionals is given in the program below, which does the calculations for the first nine lines of the New York State income tax form for 1969:

```
INCOME = 8000;
ADDITIONS = 200;
SUBTRACTIONS = 95;
TOTNYINC = INCOME + ADDITIONS - SUBTRACTIONS;
STANDED = TOTNYINC / 10;
IF STANDED GT 1000 THEN STANDED = 1000;
FITMDEDS = 950;
LIPREMS = 106;
ITAXDEDS = 500;
TOTITMDEDS = FITMDEDS + LIPREMS - ITAXDEDS;
IF STANDED GT TOTITMDEDS THEN DEDUCTIONS = STANDED
                         ELSE DEDUCTIONS = TOTITMDEDS;
NUMEXEMPTS = 3;
EXEMPTIONS = NUMEXEMPTS * 600;
NYTAXABLEINC = TOTNYINC - DEDUCTIONS - EXEMPTIONS;
PRINT(NYTAXABLEINC);
```

The random figures inserted are for a hypothetical resident who is married with one child (3 exemptions), earned $8000 salary, with $200 additional income and $95 expenses, whose itemized deductions came to $950 on his Federal return including $500 in income tax deductions, and who paid $106 in life insurance premiums. Of course the same program could be used for someone else simply by changing the appropriate figures. The two conditionals are inserted in accordance with the requirement that the standard deduction cannot exceed $1000, and to insure that the resident gets the larger of two possible deductions. The expression:

STANDED GT TOTITMDEDS

has the value true only if the value of STANDED is greater than the value of TOTITMDEDS.

If there are several alternatives to be selected, the following form of conditional can be used.

IF P1 THEN X1 ELSEIF P2 THEN X2 ELSEIF ... ELSE X

with the obvious interpretation.

Several conditions may be tested in one conditional by using the connectives AND and OR. For example

```
IF INCOME LT 3000 THEN TAX=0
ELSEIF INCOME LT 5000 AND NUMEXEMPT GT 4 THEN TAX=10
ELSEIF INCOME LT 10000 AND NUMEXEMPT GT 6 THEN TAX=20
ELSE TAX=30;
```

1.6 Loops

Most programming languages and computers recognize that the most useful kinds of algorithms contain sequences of instructions which are repeated many times. Special features are normally provided for permitting the construction and rapid execution of such repetitive sequences, which are usually referred to as loops.

In BALM there are two forms of loops provided. The first is called a while loop, and is of the form:

WHILE P REPEAT C

This is evaluated as follows.

1. The expression P is evaluated.
2. If the value is NIL, the while loop is finished.
3. The expression C is evaluated.
4. Go to step 1.

Note that the condition represented by P is evaluated at each step, so it is necessary that the evaluation of C be able to affect the value of P. The usual way of doing this is for C to contain an assignment which modifies a variable which is referenced in P.

For example, the following program will print out the smallest number whose square is greater than S:

```
I=1;
WHILE I*I LE S REPEAT I=I+1;
PRINT(I);
```

Suppose we only want to check the numbers up to some limit

```
I=1;
WHILE I LT LIMIT AND I*I LE S REPEAT I=I+1;
PRINT(I);
```

will terminate if I exceeds LIMIT or when the number whose square is greater than S is found.

The second kind of loop is the for-loop, which takes the following form:

FOR V=(I1,I2,I3) REPEAT C

where V stands for a variable, I1,I2,I3 stand for expressions whose values are integers, and C stands for an arbitrary expression. The action of the for loop is as follows:

1. The value of V is set to the value of I1.
2. The values of I2 and I3 are calculated.
3. If the value of V is greater than the value of I2 calculated in step 2, the for loop is finished.
4. The expression C is evaluated.
5. The value of V is increased by the value of I3 calculated in step 2.
6. Go to step 3.

As in the example, if I3 is left out, a value of 1 is assumed. If I3 is negative, step 3 above tests for less than, and in step 5 V is decreased.

An example of a for loop is the following program which will calculate the value of 1*2*3...*N and put the result in NN:

```
NN = 1;
FOR I=(1,N) REPEAT NN = NN*I;
```

Note that, like all expressions, while loops and for loops have values. In each case the value of the loop is the value of the last C. Thus

```
NN = 1;
PRINT(FOR I=(1,N) REPEAT NN = NN*I);
```

will print out the final value assigned to NN.

Nesting of loops and conditional expressions give us the tools to evaluate some interesting problems.

```
FOR I=(1,1) REPEAT
   FOR J=(1,9) REPEAT
      FOR K=(1,9) REPEAT
      IF (I+J+K)*(I*J*K) EQ 100*I+10*J+K
      THEN PRINT(100*I+10*J+K);
```

will find all numbers between 1 and 199 whose value is equal to the sum of digits times the product of the digits. Of course, this is a rather inefficient method. We shall see how the use of compound expressions increases our capabilities.

1.7 Compound expressions

A useful form of expression which can be used with loops and conditionals permits several expressions which are to be evaluated in sequence to be written as a single expression. The form of this is

DO C1,C2, ... CN END;

where C1,C2...,CN are expressions. Expressions within a compound expression are separated by commas. An example of this is

```
NN = 1; I = 1;
WHILE I LE N REPEAT DO NN = NN*I, I = I+1 END;
```

I LE N is an expression which is true as long as I is less than or equal to N.

The value of a compound is the value of the last expression before the end. A compound expression may occur anywhere a single expression could appear.

Example 1.7.1

Although we still have not introduced the more powerful properties of BALM, it is possible at this point to solve some fairly interesting problems. The following program, for instance, will print out all the triples X,Y,Z of numbers below 100, for which X*X + Y*Y EQ Z*Z .

```
FOR Z=(1,100) REPEAT DO ZZ=Z*Z,
  FOR Y=(1,Z-1) REPEAT DO YY=Y*Y,
    FOR X=(1,Y) REPEAT
      IF X*X+YY EQ ZZ THEN
        PRINT(X,Y,Z)
  END
END;
```

The reader should note that the values taken on by X, Y and Z for successive executions of the conditional start out with

1,1,2 1,1,3 1,2,3 2,2,3 1,1,4 and so on.

If this seems confusing the reader should consider that a loop is not executed if the initial value is greater than the limit. Consider the case when Z=1. Then the loop beginning for Y=(1,Z -1) will be skipped, and Z=2 will be the next value. This is by no means the perfect program for doing this -- for example, we know that if X*X+Y*Y EQ Z*Z then X+Y is greater than Z so there is no point in testing values of X which are less than Z-Y. The adjustments of the above program to accomplish this are left to the reader. There are, of course, many other ways of solving the above problem which the reader could implement at this point.

1.8 Blocks

A block in its simplest form has the following structure

BEGIN(V1,...,VN),X1,...,XM END

where V1,...,VN are variables which are to be considered local to the block, and X1,...,XM are expressions. In this form a block is similar to a compound, in that the expressions X1,...,XM are executed in order, with the value of the block being the value of XM. Variables used in the block which are not declared to be local are called global. Usually blocks can be included in other blocks, so variables which are local to an outer block may be global to an inner block. For example, the following extract from a program would print out the value 6:

```
X = 1; Y = 10;
BEGIN(X), X = 5, Y = 5 END;
PRINT(X*Y);
```

The variable X inside the block is declared local, so the assignment to X will not affect the X which has been assigned the value 1. The variable Y, however, is global to the block, and so its value is changed to 5.

The value of being able to declare local variables will be more clear when we discuss user defined procedures. In many languages, FORTRAN for example, variables are always local unless they are specifically declared global by appearing in a COMMON statement. In BALM we have just the opposite case. We will however postpone further discussion until the section on name scoping.

1.8.1 Labels and Gotos

In the examples we have given so far, our program has simply consisted of a number of expressions which were to be evaluated in the order in which they were written. We will now consider how this order can be controlled in a more flexible way.

Suppose we have a number of expressions whose evaluation order we wish to specify. In BALM we can do this by grouping them into a block, and inserting extra expressions to specify the execution order. We will call these jumps. If no jumps are specified, the expressions will be evaluated in the order they are written. For example

```
AREA = 8*16*9*17;
COST = 2*AREA;
PRINT(COST);
```

could be written

```
BEGIN (),
AREA = 8*16*9*17,
COST = 2*AREA,
PRINT(COST)
END;
```

This would have the same effect, but if it was being typed on a teletype nothing would be executed until the user typed in the semicolon.

When expressions are grouped in a block we may transform from one expression to another.

The simplest kind of jump in BALM is of the form

GOTO L

where L is a name. When evaluated, this will cause the next expression to be evaluated to be the one immediately following the name L in the list of expressions. Such a use of a name is called a label. For example

```
BEGIN(),
MOR, I = 10*1,
     J = I+2,
     GOTO MOR
     END;
```

would not terminate, or not at least until stopped by the computer operator. Each time the GOTO was evaluated, evaluation would continue with the expression following MOR.

Jumps are usually used in close conjunction with conditionals. This permits the execution of the jump to be dependent on the value of some expressions, and so permits the sequence of evaluation to be dependent on the data supplied to the program. In particular, the use of conditional jumps permits the programmer to write more complex loops than are provided by the for-loop and while-loop expressions.

As an example of a program which uses loops, let us consider the program which will print out the perfect numbers below 100. A perfect number is defined as a number which is the sum of the numbers which divide it exactly. The ancient Greeks knew of the existence of perfect numbers, one of which is the number 6, which is exactly divisible by 1, 2, and 3. The program can be written as follows:

```
BEGIN(),
     I=1,
 NXT,J=2, S=1,
 NXJ,IF J GE I THEN GOTO FIN,
     Q=I/J,
     IF I EQ J*Q THEN S=S+J,
     J=J+1,
     GOTO NXJ,
 FIN,IF I EQ S THEN PRINT(I),
     I=I+1,
     IF I LT 100 THEN GOTO NXT
END;
```

Jumps may only take place within a block. BALM does not permit us to transfer from one block to another. Thus

```
BEGIN(),
PRINT(10+20),
GOTO DONE
END;
```

is invalid since it implies transfer to a label not defined in the block.

Labels are only valid within blocks and may not occur within compound expressions or conditional expressions. Thus

```
     BEGIN(I),
     I=1,
NXT, I=I+1,
     IF I/2*2 EQ I THEN PRINT(I),
     IF I LT 100 THEN GO NXT
     END;
```

is valid but

```
     BEGIN(I),
     FOR I=(1,100) REPEAT DO
NXT, IF I/2*2 EQ I THEN PRINT(I) ELSE GOTO NXT
     END END;
```

is an invalid use of a label and will result in a run time diagnostic because labels may not appear within a compound.

Indirect Jumps

In an expression of the form GOTO X, X can be an arbitrary expression whose value is a label. X could have been set, for example, to a label prior to execution of the GOTO, as in the following

```
        BEGIN(L),
        IF SWTCH EQ 4 THEN L=L1
        ELSEIF SWTCH EQ 3 THEN L=L2
        ELSEIF SWTCH EQ 2 THEN L=L3
        ELSE L=L4,
        GOTO L,
    L1, ....
    L2, ...
    L3, ....
    L4, ...
        END;
```

More uses of this facility are possible using the more powerful data-objects which are described in chapter 2.

1.8.2 Use of RETURN

In BALM a block is an expression and has as its value the value of the last expression evaluated. The operator return can be used to terminate the evaluation of the expressions in the block, and to give it a value. For example, code to assign to R the value of factorial N can be written

```
    R = BEGIN(I,NN),
         NN=1, I=1,
         NXT,IF I GT N THEN RETURN NN;
         NN=NN*I,
         I=I+1,GOTO NXT
         END;
```

RETURN causes a skip to the end of the block and can be thought of as a block exit. The value returned becomes the value of the entire block. In a later section we will show how the user can write code which will permit him to write:

R=FACT(N);

which is more convenient.

EXERCISES

1.1 Write a program to convert numbers less than 100 from decimal to octal.

1.2 Rewrite example 1.7.1 so that values of X which are less than Z-Y are not tested. Compare the running times.

CHAPTER 2. MANIPULATION OF DATA OBJECTS AND STRUCTURES

2.1 General Properties of Data Objects

The objects which can be manipulated by a BALM program are called items. Each item has a type, which is one of the following.

logical
integer
label
pointer-to-identifier
pointer-to-code
pointer-to-string
pointer-to-pair
pointer-to-vector

For convenience, we will often delete the pointer-to prefix and refer to a pointer-to-vector, for example, simply as a vector. A pointer-to-code is usually called a procedure. The last three types of item, strings, pairs and vectors, are themselves ordered sets of items, while the other types can be considered to be more primitive.

All items in BALM are treated uniformly, in the sense that any item can be assigned as the value of any variable, given as an argument of a procedure, or returned as the value of a procedure call. This is in contrast to many other programming languages, in which a variable is restricted to have a particular type of item as its value. In BALM the type of an item is determined by the item itself rather than the variable whose value it is.

The following expressions can be used to test the type of an item.

Expression	Value
PAIRQ(X)	TRUE if X is a pair, NIL otherwise
VECTQ(X)	TRUE if X is a vector, NIL otherwise
IDQ(X)	TRUE if X is an identifier, NIL otherwise
INTQ(X)	TRUE if X is an integer, NIL otherwise
LOGQ(X)	TRUE if X is true or NIL, NIL otherwise
LBLQ(X)	TRUE if X is a label, NIL otherwise
CODEQ(X)	TRUE if X is code, NIL otherwise
STRQ(X)	TRUE if X is a string, NIL otherwise
X SIM Y	TRUE if X and Y are of the same type, NIL otherwise.

These take an arbitrary item as argument, and have the value true if the item is the specified type, and nil otherwise. For example, to take an action which is dependent on the type of X, we can write code of the form

```
IF INTQ(X) THEN ..., ELSEIF PAIRQ(X) THEN ...
```

Note that all types are mutually exclusive, so that if a particular item satisfies one type test, it cannot also satisfy a different one. The operator

SIM

can be used to test if two items have the same type, so that

P(X,Y) SIM 123

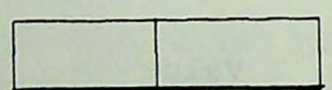
has the same value as

INTQ(P(X,Y))

The three forms of item which can be used to represent ordered sets have different characteristics and different uses, which we will be outlining in the rest of this chapter. Of these, the string is the simplest, consisting of an ordered set of charactres (letters and digits etc.). The vector and the list are more powerful, being ordered sets of arbitrary items (including other vectors and lists).

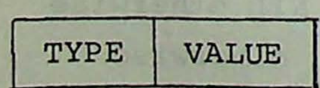
2.2 Internal Representation

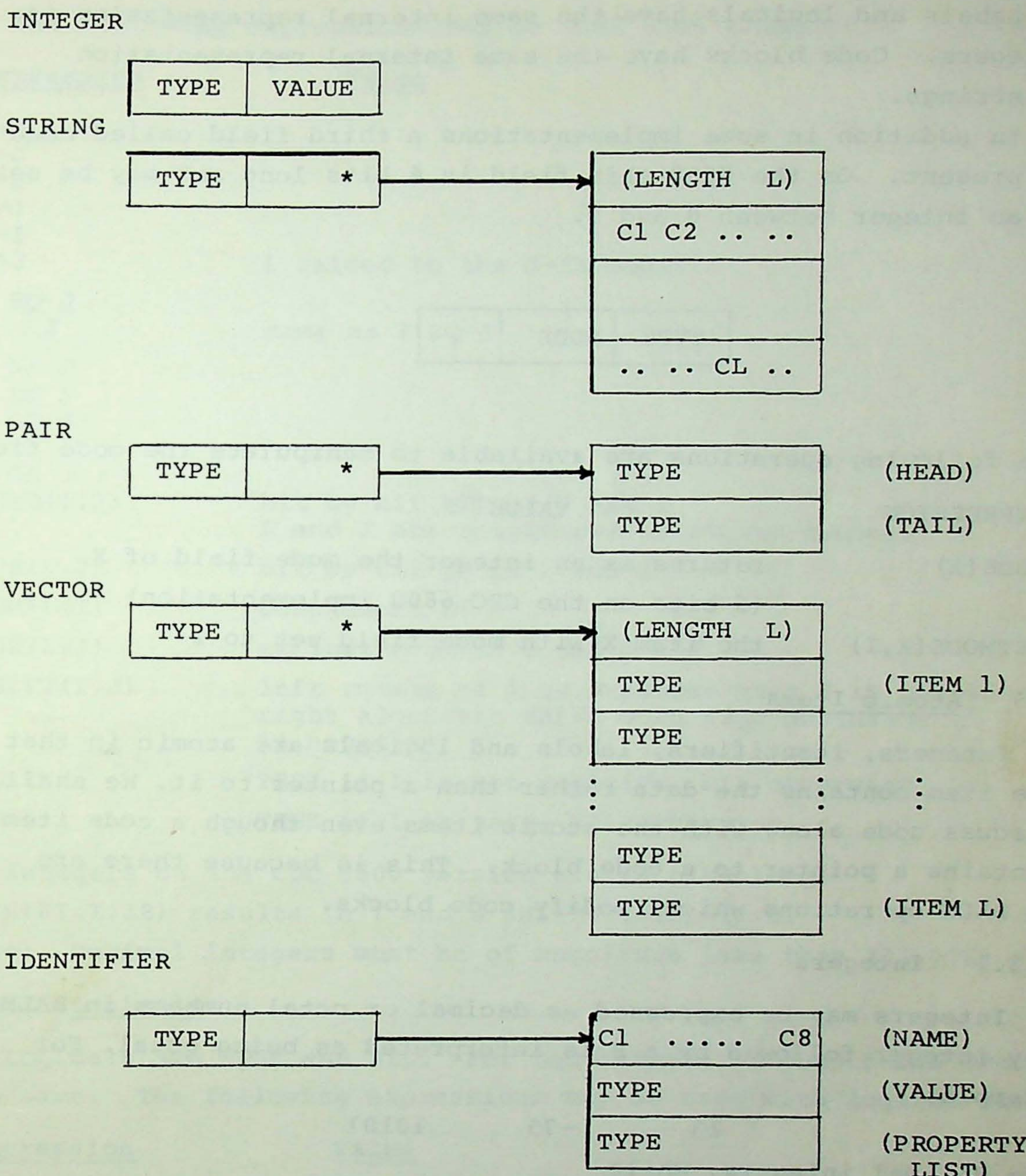
In the diagrams below we show informally how items could be represented in memory in a typical implementation. Items are as boxes of the form



Each such box represents an item, and contains a type field and a value field, which may contain a pointer. Boxes shown on top of each other will usually be adjacent in memory.

INTEGER




INTEGER
TYPE
VALUE
STRING
TYPE
*
(LENGTH L)
C1 C2 .. ..
.. .. CL ..
PAIR
TYPE
*
TYPE
(HEAD)
TYPE
(TAIL)
VECTOR
TYPE
*
(LENGTH L)
TYPE
(ITEM 1)
TYPE
TYPE
TYPE
(ITEM L)
IDENTIFIER
TYPE
C1 ..... C8
(NAME)
TYPE
(VALUE)
TYPE
(PROPERTY LIST)

Labels and logicals have the same internal representation as integers. Code blocks have the same internal representation as strings.

In addition in some implementations a third field called mode is present. On the 6600 this field is 6 bits long and may be set to an integer between 0 and 7.

TYPE	MODE	

The following operations are available to manipulate the mode field.

EXPRESSION	VALUE
MODE(X)	returns as an integer the mode field of X (6 bits on the CDC 6600 implementation)
SETMODE(X,I)	the item X with mode field set to I

2.3 Atomic Items

Integers, identifiers, labels and logicals are atomic in that the item contains the data rather than a pointer to it. We shall discuss code along with the atomic items even though a code item contains a pointer to a code block. This is because there are no BALM operations which modify code blocks.

2.3.1 Integers

Integers may be expressed as decimal or octal numbers in BALM. Any integer followed by a B is interpreted as being octal. For example

13 -75 10101

are decimal integers, while

77B 101010B 10B

are all octal numbers.

The following expressions may be used with integers.

Expression	Value
I+J	
I-J	
I*J	
I/J	
-I	
I↑J	I raised to the J-th power
I EQ J	
I ≡ J	same as I EQ J
I NE J	
I LT J	
I GT J	
I LE J	
I GE J	
LAND(I,J)	bit by bit AND of I and J I and J are considered as binary numbers
LOR(I,J)	bit by bit OR of I and J
COMPL(I)	complement of I
XOR(I,J)	exclusive OR of I and J
SHIFT(I,J)	left rotate of I by J places when J is positive right algebraic shift with sign extension if J is negative
PL I	TRUE if I is not negative, NIL otherwise
ZR I	TRUE if I is zero, NIL otherwise

Integers on the CDC 6600 version are 18 bits long. A SHIFT(I,18) results in I and a SHIFT(I,-18) in 0.

Note: Decimal integers must be of magnitude less than 131,072.

2.3.2 Logicals

Logicals are TRUE and NIL. For convenience NIL and FALSE are the same. The following expressions may be used with logical items.

Expression	Value
X AND Y	if X then Y else NIL
X OR Y	if ¬X then Y else TRUE
NOT X	if X then NIL else TRUE
NULL X	if X then NIL else TRUE
¬X	if X then NIL else TRUE

The reader should note that the above expressions test for a value of NIL. Any value other than NIL will be interpreted in the same way as TRUE.

2.3.3 Identifiers

An identifier may be thought of as an ordered triple, whose components are called name, value and property-list. The name component is always a string, while the value and property-list components may be of arbitrary type. Names of identifiers may be only eight characters long and will be truncated with a warning message if they exceed eight characters.

Constant identifiers used as data-objects may be specified by a name preceded by a quote to distinguish them from their values. Either $\geq$ or = may be used as a quote. Thus

$\geq$ABC   =XYZ $\geq$ALONGNAME

are valid identifiers. Note that ABC without the quote would normally refer to the value of the variable ABC.

The following expressions may be used with identifiers.

Expression	Value
SFROMID(ID)	the string which is the name comppnent of ID thus SFROMID($\geq$ABC) results in ≠ABC≠
VALUE ID	the value component of ID For example: X=VALUE(=ABC) is equivalent to X=ABC
PROPL(ID)	the property-list component of ID We shall discuss property lists in more detail when we discuss language extensions. An identifier may have a property or properties as well as a value.
VALUE ID = X	has the value X and the side-effect of changing the value component of ID to X
PROPL(ID) = X	has the value X and the side-effect of changing the property-list component of ID to X
ID1 EQ ID2 ID1 ≡ ID2	TRUE if ID1 and ID2 are the same identifier, NIL otherwise.

2.3.4 Code Blocks

Code blocks in BALM are the items used to represent procedures Both the BALM system procedures and user-defined procedures are represented as code blocks. Each time the BALM compiler encounters a semicolon it generates a block of code and then executes it.

The following expressions can be used with code.

Expression	Value
CFROMV(V)	The block of code corresponding to the instructions specified as integers in the vector V.
IDFROMC(C)	The identifier which appeared on the left-hand side when the block of code was defined as a procedure (see Chapter 3).
APPLY(C,X)	returns the value of procedure C when applied to argument X, if X is a list (see Section 2.6) it is considered to be a list of arguments and C will be applied to the members of the list. If X is not a list it is considered to be the sole argument of C.
C1 EQ C2 C1 ≡ C2	TRUE if C1 and cC2 refer to the same code block; NIL otherwise.

2.3.5 Labels

Labels are used with GOTOs to determine the flow of control of the program. Constant labels are defined by using an identifier as described in Section 1.8.1. This identifier can be thought of as a local variable of the block whose value is the label item.

The following expressions can be used with labels.

Expression	Value
GOTO L GO L	Value is not accessible to the programmer, control is transferred to the instruction at L, L may be an arbitrary expression, but its value must be a label defined in the current block.

2.4 Strings

A string in BALM is an ordered set of characters, where we use the term character to refer to those symbols which can be punched on a card or typed on a teletype. Strings may be of arbitrary length, and may contain arbitrary characters. A constant string is specified in a program by preceding the first character by the symbol ≠ and following the last character by the symbol ≠.

In order to include the character ≠ in a string it must appear twice. For example

```
S1=≠≠≠≠;
```

is the string consisting of one character which is ≠.

```
S2=≠A QUOTE ≠≠ MAY APPEAR IN A STRING≠;
```

will print as

```
≠A QUOTE ≠ MAY APPEAR IN A STRING≠;
```

The primitive operations on strings provided in BALM include facilities for extracting part of a string (a substring), and for changing a substring. The operator sub used for this purpose takes three arguments, a string, the index of the first character of the substring, and the length of the substring. Thus Thus the program

```
S = ≠ABCDEFGHI12345≠; SS = SUB(S,3,4);
SUB(S,5,7) = ≠+++≠;
```

will assign the string ≠CDEF≠ to the variable SS, and leave the string ≠ABCD+++345≠ as the value of the variable S.

Strings may be concatenated using the operator concat. For example

```
PRINT(CONCAT(≠ABC≠,≠1234567890123≠));
```

will print ≠ABC1234567890123≠.

The following expression may be used to create a string.

Expression	Value
STRING(I1,I2,...,IN)	the string whose characters are represented by I1,I2,...,IN .

This is only useful when the user knows the integer representation of the character. For example:

```
STRING(1,2,3);
```

results in ≠ABC≠.

The reader should note that the variable whose name is ABC is not the same thing as the string ABC. However, the BALM system does permit one to be converted to the other.

The following expressions may be used with strings.

Expression	Value
SIZE S	the number of characters in S
IDFROMS(S)	the unique identifier in the system whose name is S. For example: IDFROMS(≠ALONGNAME≠) has ALONGNAM as its value. Note that the string is truncated to 8 characters.

[continued]

Expression	Value
VFROMS(S)	creates a new vector whose elements are the integer representations of the characters of S (see Section 2.5).
CONCAT(S1,S2)	creates a new string whose characters are the same as S1 followed by S2
SUB(S,I,J)	creates a new string whose characters are the I-th through the (I+J-1)-st character of S
SUB(S1,I,J)=S2	has the value of S2 and the side effect of changing the I-th through the (I+J-1)-st characters of S1 to the first through the J-th characters of S2
S1 EQ S2 S1 ≡ S2	TRUE if S1 and S2 refer to the same string, NIL otherwise
EQSTR(S1,S2)	returns TRUE if S2 is the same as or a copy of S1; otherwise NIL.

If a string exceeds 1 line, trailing blanks are deleted. For example

```
≠A STRING
ON TWO LINES≠
```

is equivalent to

```
≠A STRINGON TWO LINES≠
```

2.4.1 Copying and Assignment

In addition to these operations a series of useful procedures and functions are part of the BALM system. They are listed in entirety in Appendix E. PRINT is one such procedure which we have discussed previously. Below we will describe COPY.

In BALM the assignment operation is done by assigning a copy of the item which is the value of the right-hand side to the left-hand side. However, in the case of a pointer-to type only the pointer is copied, not the object pointed to. In a similar way, arguments to procedures are copies of the actual arguments, but in the case of pointer-to items only the pointer is copied. For example:

```
A=10;
SAVE=A;
A=A+1;
PRINT(SAVE);
10
```

Changing the value of A does not affect SAVE. Consider, however, what happens when the item is not atomic and contains instead a pointer to a data structure. For example:

```
ALPHA=≠ABCDEFGHIJKLMNOPQRSTUVWXYZ≠;
MOD=ALPHA;
SUB(MOD,2,3)=≠234≠;
PRINT(MOD);
≠A234EFGHIJKLMNOPQRSTUVWXYZ≠
PRINT(ALPHA);
```

The value of ALPHA is also ≠A234EFGHIJKLMNOPQRSTUVWXYZ≠. Two things should be noted about the above example: (1) the expression MOD=ALPHA sets MOD to an item whose type is string and whose value is a pointer to the same data structure referred to by ALPHA. In other words the structure representing ≠ABCDE.....≠ occurs only once in memory but the assignment created a second reference to it. (2) The expression SUB(MOD,2,3)=≠234≠ has the side effect of changing the representation in memory. Since the structure only exists once any item referencing it reflects the change.

COPY is a function which returns a copy of its argument. If the assignment expression in the above example were changed to: MOD=COPY(ALPHA); then MOD would contain a reference to a string containing the same characters as ALPHA but not represented in memory by the same structure as ALPHA.

2.5 Vectors

A vector in BALM is an ordered set of items in which an element of the set is referred to by specifying its position within the vector. For example, if the value of the variable V is a vector the first element of the vector is referred to by the expression V(1), the second by V(2), etc. The integer used to specify the element is usually called an index, and in BALM it can be a constant, a variable, or an expression of arbitrary complexity.

A piece of program which will assign to the variable M the largest number in a vector V whose elements are all numbers can be written

```
L=LENGTH(V);
M=V[1];
FOR I=(2,L) REPEAT
   IF V[I] GT M THEN M=V[I];
```

Here we have used the procedure length which will return as its value the number of items in the vector (or list or string) given as its argument.

An element of a vector can have its value changed by an assignment in which the left-hand side specifies the element. For example, the following piece of program will reverse the order of the elements of the vector V.

```
L = LENGTH(V);
FOR I=(1,(L+1)/2) REPEAT
    DO S=V[I], V[I]=V[L+1-I], V[L+1-I]=S END;
```

As a more complicated example, we give below a piece of program which will sort a vector V of numbers into increasing order.

```
L = LENGTH(V);
FOR I=(1,L-1) REPEAT
  IF V[I] GT V[I+1] THEN DO
    X = V[I+1], V[I+1] = V[I], J = I,
    WHILE J GE 2 AND V[J-1] GT X REPEAT DO
     V[J] = V[J-1], J = J-1 END,
    V[J] = X
    END;
```

This way of implementing the sorting operation is called a bubble sort. Essentially the algorithm looks at successive pairs of elements, checking to see if they are in the correct order. When an incorrectly ordered pair is found, the second element is moved backwards towards the beginning of the vector until it is in the correct position, all the intervening elements being moved forward one place. The search for incorrectly ordered pairs is then continued. When the last pair has been examined, the vector elements are in the correct order. The reader should trace through through this program with a sample vector to check that he follows the logic.

A vector can be created in BALM in several ways. The simplest uses the operator VECTOR, which can have an arbitrary number of arguments, and which will return as its value the vector which has as its elements the values of those arguments. Thus

ABC = VECTOR(A+B,12,MAX(V))

will assign to the variable ABC the 3-element vector whose elements are the value of A+B, the number 12, and the maximum element in the vector V, in that order. Alternatively we can use the procedure MAKVECTOR to create a vector of specified size, but without assigning particular values to the elements (they will all be set to NIL actually). Thus

```
DO ABC = MAKVECTOR(3), ABC[1] = A+B,
   ABC[2] = 12, ABC[3] = MAX(V) END;
```

would have precisely the same effect as the single expression above.

The following expressions can be used to create vectors.

Expression	Vector
VECTOR(X1,X2,...,XN)	a vector whose elements are X1,X2,...,XN
MAKVECTOR(I)	a vector with I undefined elements

In the above examples, the only expressions we have used with indices have been simple variables. In fact any expression whose value is a vector can be indexed. One very useful data-structure is a vector whose elements are themselves vectors -- sometimes called a two-dimensional array in other programming languages or a matrix by mathematicians. If M is such a data-structure then the 3rd element is M[3], and the 2nd element of its 3rd element is M[3][2]. This element can be changed by putting this expression on the left-hand side of an assignment in the usual way.

In BALM a simple notation is available for printing a vector This is illustrated by the program

```
V = VECTOR(12, 34, 56); PRINT(V);
```

which would print

```
[12 34 56]
```

In more elaborate cases, such as when the elements of the vector are themselves vectors or other complex data-structures, an extension of the notation is used. In general, a vector is printed as a left square bracket followed by the appropriate notation for the vector elements, separated by spaces, and terminated by a right square bracket.

This same notation can also be used to specify constant vectors in the program. When used in this way, the vector should be preceded by a quote operator which distinguishes this use of the square brackets from others. We use ≥ or = as a quote.

```
V ==[12 34 56]
```

will actually have the same effect as the assignment to V given above. A vector constant may not contain variables. Identifiers appearing in a vector constant are treated as identifier constants even though they arenot preceded by a quote. Thus

```
V=≥[A B C];
```

is equivalent to

```
V=VECTOR(≥A,≥B,≥C);
```

and both will print as

```
[A B C]
```

Similarly

```
V1=≥[X Y Z];
X=10; Y=20; Z=30;
V2=VECTOR(X,Y,Z);
X=≠A NEW VALUE≠;
PRINT(V1); PRINT(V2);
```

will print

```
[X Y Z]
[10 2U 30]
```

The following expressions may be used with vectors.

Expression	Value
V[I]	the I-th element of V
V[I] = X	has the value X and the side-effect of changing the I-th element of V to X
SIZE V	the number of elements in V
CONCAT(U,V)	the vector whose elements are U(1),...,U[SIZE U],V[1],...,V[SIZE V]
SUB(U,I,J)	the vector whose elements are U[I],...,U[I+J-1]
SUB(U,I,J)=V	has the value V and the side-effect of changing the I-th through (I+J-1)-st elements of U to V[1],...,V[J]
SFROMV(V)	the string whose characters are represented by the elements of V, assumed to be integers less than 128
CFROMV(V)	the code corresponding to the elements of V, assumed to be integers less than 128
V EQ U V ≡ U	TRUE if V and U point to the same vector, NIL otherwise
COPY(V)	described in the previous section. COPY can also be used with vectors and makes a copy of all elements of V, and all their elements, etc. The reader should note that two vector operations V[I]=X and SUB(U,I,J)=V have side effects which result in the data structure being changed.

[continued]

	Any items which reference a vector are affected by V[I]=X and SUB(U,I,J)=V.
EQUAL(X1,X2)	a utility procedure which returns TRUE if X2 is the same as or a copy of X1, NIL otherwise.

With the availability of vectors, the range of problems that we can conveniently solve is considerably increased. Many objects which require a number of properties for their description can be conveniently represented by vectors whose elements are the values of those properties. For instance, a triangle can conveniently be represented by a vector of length three, whose elements are the lengths of its sides, and the position of a town can be represented by its latitude and longitude. Since in BALM a vector can itself be an element of a vector, we can represent a trip across the country by a vector whose elements are the 2-element vectors representing the positions of the towns on the route.

In some algorithms, it is found to be convenient to be able to have other types of operations as the basic primitives for manipulating ordered sets. For example, an algorithm which frequently requires that an element be added to or deleted from an ordered set will be rather inefficient using vectors. For this reason we provide an alternative data-structure capable of representing general ordered sets, but in which the primitive operations are different. We refer to thse data-structures as lists, which are described in more detail in the next section.

2.6 Pairs

A pair is an ordered pair of items, the first component being called head and the second component tail. Any two items may be combined to form a pair. Pairs are created by the following expression.

Expression	Value
X : Y	a pair whose head component is X and whose tail component is Y.

The reader should note that the expression X:Y:Z is equivalent to X:(Y:Z) and not to (X:Y):Z .

The BALM notation for printing a pair is illustrated below.

```
P=10:20;  PRINT(P);
(10,20)
```

A pair may also be specified as a constant. We again use a quote character, either $\geq$ or =.

```
P= ≡(10,20);  PRINT(P);
(10,20)
```

The following expressions may be used with pairs.

Expression	Value
HD P	the head component of P
TL P	the tail component of P
HD P = X	has the value X and the side-effect of changing the head component of P to X
TL P = X	has the value X and the side-effect of changing the tail component of P to X
P1 EQ P2	TRUE if P1 and P2 refer to the same pair,
P1 ≡ P2	NIL otherwise
COPY(P)	described in the section of strings is also very useful with pairs
EQUAL(X1,X2)	described in the previous section is useful in comparing copies of pairs

Since the components of a pair may themselves be pairs we can construct very complex groups of items. An especially useful form of a collection of pairs is called a list.

2.6.1 Pairs and Lists

A list in BALM is an ordered set of items in which the two primitives provided for referring to the components give respectively the first element of the list and the remainder of the list.

Lists can be created by using the operator LIST, which can have an arbitrary number of arguments, and which will give as its value a list with the values of its arguments as elements. A list is constructed out of pairs according to the following definition.

Expression	Value
LIST(X1,X2,...,XN)	X1:(X2: ... (XN:NIL) ...)

For example:

```
ABC = LIST(A+B,12,MAX(V));
```

will create a list whose elements are the value of the expression A+B, the number 12, and the value of MAX(V), and assign this list to ABC. The procedure list is analogous to the operator vector described previously. Of course lists may also be constructed from a given head and a given tail using the colon operator. For example:

```
ABC = A+B:12:MAX(V):NIL;
```

would have the same effect as the expressions above.

We can think of a list as having two components, a head and a tail which can be referred to by the two primitives, HD and TL. For example, if the value of the variable L is a list of three numbers 1, 2 and 3, then the value of the expression HD L is the number 1, and the value of the expression TL L is the list whose elements are the numbers 2 and 3.

The second element of a list can be referred to as the head of the tail of the list, so HD TL L would be the number 2. Similarly the third element would be referred to as HD TL TL L, and so on. The tail of a one-element list is the empty list which is written as NIL.

A significant property of lists is that in general the tail of a list is also a list. Thus those algorithms which require the elements of a list to be processed in order can be conveniently implemented using an auxiliary variable to keep the unprocessed portion of the list. For example, a piece of program to sum the elements of a list L can be

```
S = 0; R = L;
WHILE R REPEAT DO S=S+HD R, R=TL R END;
```

Here the variable S is used to accumulate the sum, and the variable R is used to keep the unprocessed part of the list. We are using the fact that the WHILE loop will continue until R is NIL, which denotes the end of the list.

The predicate PAIRQ can be used to test if an arbitrary expression is a non-empty list. For example, we could have written

```
WHILE PAIRQ(R) REPEAT DO S=S+HD R, R=TL R END;
```

in the example given above.

A part of a list can be changed by putting the appropriate reference on the left-hand side of an assignment, as might be expected. However, there is a certain danger in using this facility without care, for the following reason. Lists are represented in memory in such a way that two lists which end in the same elements may use the same portion of memory to store those common elements. This means that any operation which changes an element in one of the lists also changes in an underhand way the element of the other list. This situation can arise in code such as:

```
L = LIST(1,2,3);
X = 4!L; Y = 5!L;
```

At this point the value of X is the list containing the numbers 4, 1, 2, 3 while Y contains 5, 1, 2, 3. If now we executed:

```
HD TL TL X = 9;
```

we would change X to contain 4, 1, 9, 3, but also Y to contain 5,1,9,3. In spite of these warnings, the user can use this facility without problem if he keeps track of which elements are common to which lists.

Lists can be printed by the print routine in a convenient and readable format. This is similar to the format used for vectors, but using parentheses instead of square brackets. Thus the three element list containing the numbers 12, 13, and 14 would be printed:

```
(12 13 14)
```

instead of:

```
(12 13 14,NIL)
```

Similarly this notation can also be used to specify constant lists, if preceded by the quote mark = . Vectors can be elements of lists and vice versa, so we can write things such as:

```
VL = =[(12 13) [14 15 16] ([17 18] 19)]
```

and so on. Recall that an identifier in such a constant will be treated as a constant, not a variable, and expressions will not be evaluated.

Several useful procedures are available which act upon or result in lists:

LENGTH(L) returns the number of elements in a list

VFROML(L) returns a vector whose elements are the same as those of the list given as its argument.

LFRMOV(V) returns a list whose elements are the same as those of the vector given as its argument.

MEMBER(X,L) called with two arguments, the second of which must be a list, returns true if the first argument is EQ to one of the elements of the second, For example: if MEMBER(=RED,L) THEN...

SUBST(X,Y,L) called with 3 arguments, the third of which must be a list. It returns a new list in which argument 1 has replaced argument 2. For example:

```
L=LIST(=RED,=BLUE,=YELLOW,=PURPLE);
NEWL=SUBST(=GREEN,=RED,L);
PRINT(L);
(GREEN BLUE YELLOW PURPLE)
```

2.6.2 Examples

Remember that the tail of a one element list is always NIL. In other words NIL marks the end of a list. Thus we can print successive elements of a list in the following way.

```
WHILE L REPEAT DO
  PRINT(HD L),
  L=TL L
END;
```

This will repeat the loop until L is NIL. Suppose L is a list of strings

```
L=LIST(≠ONE≠,≠TWO≠,≠THREE≠,≠FOUR≠);
```

The first time through the loop

```
≠ONE≠
```

is printed and the value of L is LIST(≠TWO≠,≠THREE≠,≠FOUR≠). Next

```
≠TWO≠
```

is printed and the value of L is LIST(≠THREE,≠FOUR≠). Then

```
≠THREE≠
```

is printed and the value of L is LIST(≠FOUR≠). Finally

```
≠FOUR≠
```

is printed, the value of L is NIL and the loop is terminated.

Suppose we have a list of pairs consisting of an identifier and an integer. A program which will ascertain whether an identifier is on the list and if so print the integer or if not add the identifier and the next highest integer to the list is as follows:

```
          LST= =((RED,1) (BLUE,2) (YELLOW,3));
          ID= =GREEN;
          BEGIN(PREV,N,L),
          L=LST,
          WHILE L REPEAT DO
            IF HD HD L EQ ID THEN GO FOUND,
            PREV=L,  N=TL HD L,  L=TL L
            END,
          TL PREV=(ID;N+1):NIL,
          RETURN N+1,
     FOUND, PRINT(ID,TL HD L)
          END;
          PRINT(L);
```

The first time through the loop HD L will be =(RED.1) thus HD HD L will be =RED.

Note that HD HD L is equivalent to HD(HD L).

TL HD L is 1 the first time through the loop, and the last time HD HD L is =YELLOW, N is set to 3, PREV is set to LIST(=YELLOW:3) and L becomes NIL. To add the new identifier to the list we need to make the tail of PREV point to a new element. This is done by replacing TL PREV which is NIL with a pointer to a new pair whose head is =(GREEN,4) and whose tail is NIL. The final print of L will result in

```
((RED,1) (BLUE,2) (YELLOW,3) (GREEN,4))
```

EXERCISES

2.1 Write a program to test a string and print it if it is palindromic, i.e. ≠MADAM≠.

3.1 A Little About Utility Procedures.

We have already mentioned a number of built-in procedures: PRINT, COPY, EQUAL, etc. A complete list of these predefined or built-in functions exists in Appendix E. All of these are available to the user as part of the BALM system.

All of these procedures are written in BALM and can be found in the BALM system listing in Appendix F either near the front in the section labeled utility procedures or at the end in the section marked supplement. They provide further examples of BALM programs.

In this chapter we will be concerned with user-written procedures. In most cases a BALM program will consist mainly of a number of definitions of user-written procedures, interspersed with one or more commands to invoke these procedures.

3.2 User Defined Procedures and Functions

In general a procedure consists of an expression written in terms of a number of variables, some of which may be designated as standing for arguments. For example, we might want to define a procedure which will take two arguments and return as its value the sum of their squares. The mathematical way of defining this function is something like:

```
F(X,Y) = X*X+Y*Y
```

In BALM we could write this as:

```
SUMSQ = PROC(X,Y),X*X+Y*Y END,
```

Here the expression on the right-hand side of the assignment is a procedure, delimited by the operators PROC and END. Subsequently the procedure can be used in a form such as:

```
A = B + SUMSQ(C,D+3) - 5,
```

Here the arguments of the procedure are the value of C and the value of the expression D+3. The procedure invocation is evaluated by effectively carrying out the following operations:

1. Save the current values of X and Y
2. Assign the value of C to X and the value of D+3 to Y.
3. Evaluate the expression X*X+Y*Y.
4. Restore the original values of X and Y.
5. The value of SUMSQ(C,D+3) is given by step 3.

The reason for steps 1 and 4 is that it is desirable to be able to define a procedure so that any changes made to the values of the variables used to indicate its arguments will not affect the values which they may have elsewhere.

The expression used to define a procedure can be one of considerable complexity, including a block. The procedure mentioned in a previous section which calculates 1*2*3...*N for an argument N, could be defined:

```
FACT = PROC(N),
      BEGIN(I,NN),
        NN=1, I=1,
        NXT,IF I GT N THEN RETURN NN,
        NN=NN*I,
        I=I+1,GOTO NXT
        END END;
```

and could then subsequently be applied like any other procedure.
Thus `PRINT (FACT(5))`
would print out the value 120.

Below are some expressions which are useful when writing procedures.

Expression	Value
NUMARGS()	returns the number of arguments of the current procedure
ARGUMENT(I)	returns the Ith argument of the current procedure

These make it possible to call a procedure with a variable number of arguments. For example

```
PRTARGS=PROC(),
  BEGIN(N,I),
  N=NUMARGS(),
  FOR I=(1,N) REPEAT
  PRINT(ARGUMENT(I))
  END END;
PRTARGS(10,20,30);
```

would print

```
10
20
30
```

and `PRTARGS(≠STRING≠);`
would print

```
≠STRING≠
```

Note: Procedure PRTARGS must be defined before it is executed.

In BALM all procedures are functions. That is they all have values. The value of a procedure is the value of its expression. Normally a procedure consists of a block or a compound expression but it may as in the case of SUMSQ in the example above be a simple expression.

Procedures may also have side effects such as modifying a variable or modifying a data structure pointed to by an argument. Procedures may not, however, return data to the calling program by modifying values of variables given as arguments since only the values of the arguments are transmitted to the procedure. We shall discuss this in more detail in Section 3.3.

We need not call a procedure directly in BALM but may use APPLY.

Expression	Value
APPLY(C,X)	returns the value of procedure C where applied to argument X. If X is a list it is considered to be a list of arguments and C will be applied to the members of the list. If X is not a list it is considered to be the sole argument of C.

Suppose we want to write a function of two arguments defined as follows. If argument 1 is an integer then the function should return that element of the vector specified by argument 2. If the 1st argument is code then it should return the result of executing the procedure with the arguments which are members of the list specified by the 2nd argument.

```
OF=PROC(F,Y),
   IF INTQ(F) THEN Y[F]
   ELSEIF CODEQ(F) THEN APPLY(F,Y)
   ELSE NIL
   END;
V==[10 20 30];
PRINT(OF(3,V));
```

would print

```
30
```

and

```
PRINT(OF(MAX,LIST(10,20)) );
```

would print

```
20
```

3.3 Name Scoping

So far we have not really dealt with the problem of name scoping or what it means to make a variable local to a block. It is important to make variables local to blocks within procedures to prevent unwanted side effects. Perhaps an example will serve to illustrate the seriousness of the problem.

Suppose we wish to determine how many elements of a list and a vector are the same. Suppose we know that those elements are themselves vectors and we write a procedure named SAMEV to return TRUE when its arguments are the same, NIL otherwise.

```
SAMEV=PROC(X,Y),
  BEGIN(),
  IF SIZE X NE SIZE Y THEN RETURN NIL,
  IF NOT VECTQ(X) OR NOT VECTQ(Y) THEN RETURN NIL,
  FOR I=(1, SIZE X) REPEAT
  IF X[I] NE Y[I] THEN RETURN NIL,
  TRUE
  END END;

V= =[[1 2] [2 3] [1 3]];
L= =([1 3] [2 3] [1 2]);
N=0;
I=1;
WHILE L REPEAT DO
  IF SAMEV(HD L, V[I]) THEN N=N+1,
  I=I+1,  L=TL L
  END;

PRINT(N,=MEMBERS,=ARE,=SAME);
```

Note that the procedure must be defined before it is executed.

We can see that element 2 is the same and is in the same position in both the list and the vector. Therefore, we would expect the print to have the following effect.

```
1 MEMBERS ARE SAME
```

However, SAMEV uses I and I is not local to its block. When SAMEV is called I has the value 1 but upon exit from SAMEV I has a value of 2 so the statement I=I+1 will set I to 3. The second time through the loop, element 2 of the list will be compared to element 3 of the vector. The third time through the loop element 3 of the list will be compared to element 2 of the vector. The answer is

```
0 MEMBERS ARE SAME
```

This example illustrates the need for local variables. Procedure SAMEV should not be modifying I in the calling program. I can be made local to SAMEV by inserting it in the list following BEGIN.

```
BEGIN(I),
```

While the rules for name scoping in BALM are simple and straightforward they may seem confusing to users with experience in some other programming languages. In most programming languages storage allocation and, therefore, name scoping is determined at compile time and remains static during execution. In BALM name scoping is dynamic, that is the meaning of variables is determined during exectuion rather than procedure definition.

A name used in a BALM expression normally refers to a variable. Corresponding to each name is a unique identifier. Any occurrence of the name refers to the current value of that identifier.

A variable is made local to a block by appearing in the declaration following a begin. For example:

```
BEGIN (A,B,C)
```

makes A, B, and C local to the block. On entry to the block the current value of each local identifier is saved. On exit from the block that value is restored. Therefore any use of an identifier between entering and leaving a block to which it is declared local will not change the value it had prior to entering the block. As a general rule it is good practice to declare all variables to be local whose values are not needed outside a block. The user should be particularly careful to declare as local all index variables used in FOR loops. They are frequently forgotten and are rarely needed outside the block.

A variable used in a block in which it is not declared local is called global (to that block). Such variables can be used to transmit parameters to the block or procedure, and any changes made to their values within the block will have effect outside the block. Note that a variable can be local to block or procedure A, but be global to a block or procedure invoked between entering and leaving A. This is often convenient when writing a package of procedures whose subprocedures have many arguments or always the same arguments. For example

```
OUTLINE= PROC(),
BEGIN(ITM),
    ITM=READ(INPUT),
    IF EOF(ITM) THEN RETURN NIL,
    IF PAIRQ(ITM) THEN MLIST()
    ELSEIF VECTQ(ITM) THEN  MVECT()
    .
    .
    ELSE PRINT(≠ERROR ON INPUT FILE≠)
END END;
MLIST= PROC(),
DO PRINT(=LIST,ITM),

    .
    .
END END;
```

The variable ITM is declared local to procedure OUTLINE and thus has no effect on any procedure entered before OUTLINE is invoked. The variable ITM is however, global to procedure MLIST, MVEC, etc. and to any procedure called by OUTLINE where it is not specifically declared to be local.

Similarly on entry to a procedure the current values of the identifiers named as arguments are saved. The actual value of the argument then replaces the value of the identifier. The original value is restored upon exit from the procedure.

3.4 Recursive Procedures

In an earlier section we gave the code to sum the elements of a list. This can be written as a procedure in the following way.

```
LSUM = PROC(L), BEGIN(S,R),
       S = 0, R = L,
       WHILE R REPEAT DO S=S+HD R, R=TL R END,
       RETURN S    END END;
```

An alternative way of writing this procedure makes use of the fact that we can say that the sum of a non-empty list is the sum of the head and the sum of the tail of the list. That is, when the list is not empty, we can say

```
LSUM(L) = HD L + LSUM(TL L)
```

Putting in the test for an empty list, whose sum is zero, we get the following concise definition:

```
LSUM = PROC(L),
       IF L EQ NIL THEN 0 ELSE HD L + LSUM(TL L)
       END;
```

This type of definition, called a recursive definition, is quite often convenient for operations on lists. In the simple case given here, a non-recursive definition is also simple, but in more complex cases recursion permits the complexity of an algorithm to be significantly reduced, so the reader should make himself familiar with the technique.

As a simple example, suppose we write a procedure to append two lists, that is, to construct a list whose elements include those of both arguments. At first sight this is a little tricky, since we have to add the last element of the first list to the beginning of the second list, then the second to last, and so on. However, the following recursive implementation accomplishes this:

```
APPEND = PROC(X,Y),
         IF X=NIL THEN Y ELSE HD X:APPEND(TL X,Y)
         END;
```

This is another example of the usefulness of recursion. A non-recursive implementation is more complicated, but can be written as follows:

```
APPEND = PROC(X,Y), REVAPP(REVAPP(X,NIL),Y) END,
REVAPP = PROC(L,R),
         BEGIN(),
    MOR, IF L EQ NIL THEN RETURN R,
         R = HD L:R, L = TL L, GO MOR
         END END;
```

Here we have used an auxiliary routine REVAPP which is similar to APPEND, but reverses its first argument. Thus the value of REVAPP(X,NIL) is simply a list whose elements are the same as those of X, but in the reverse order. REVAPPing this onto the list Y will then give the required result.

Example: TOWER OF HANOI

Tower of Hanoi is a game which consists of three spindles and some rings of different sizes. The rings are to be moved from one spindle to another without violating the following rules.

(1) only one ring may be moved at a time.

(2) a ring may be moved from any spindle to any other.

(3) at no time may a larger ring rest upon a smaller ring.

The following recursive function prints the steps necessary to move N rings from one spindle to another.

```
HANOI=PROC(N,ST,DEST,INTERM),
IF ZR N THEN NIL ELSE DO
   HANOI(N-1,ST,INTERM,DEST),
   PRINT(=MOVE,=RING,N,=FROM,ST,=TO,DEST),
   HANOI(N-1,INTERM,DEST,ST)
   END
END;
```

```
HANOI(5,=A,=C,=B);
MOVE RING 1 FROM A TO C
MOVE RING 2 FROM A TO B
MOVE RING 1 FROM C TO B
MOVE RING 3 FROM A TO C
MOVE RING 1 FROM B TO A
MOVE RING 2 FROM B TO C
MOVE RING 1 FROM A TO C
MOVE RING 4 FROM A TO B
MOVE RING 1 FROM C TO B
MOVE RING 2 FROM C TO A
MOVE RING 1 FROM B TO A
MOVE RING 3 FROM C TO B
MOVE RING 1 FROM A TO C
MOVE RING 2 FROM A TO B
MOVE RING 1 FROM C TO B
MOVE RING 5 FROM A TO C
MOVE RING 1 FROM B TO A
MOVE RING 2 FROM B TO C
MOVE RING 1 FROM A TO C
MOVE RING 3 FROM B TO A
MOVE RING 1 FROM C TO B
MOVE RING 2 FROM C TO A
MOVE RING 1 FROM B TO A
MOVE RING 4 FROM B TO C
MOVE RING 1 FROM A TO C
MOVE RING 2 FROM A TO B
MOVE RING 1 FROM C TO B
MOVE RING 3 FROM A TO C
MOVE RING 1 FROM B TO A
MOVE RING 2 FROM B TO C
MOVE RING 1 FROM A TO C
```

The program logic can be seen by induction. Clearly, moving zero rings requires no steps. Moving one ring requires one step.

```
MOVE RING 1 FROM A TO C
```

Moving two rings requires three steps.

```
MOVE RING 1 FROM A TO B
MOVE RING 2 FROM A TO C
MOVE RING 1 FROM B TO C
```

The general solution is:

(1) move N-1 rings from A to B

(2) move ring N from A to C

(3) move N-1 rings from B to C

In procedure HANOI the arguments are as follows:

(1) N - the number of rings to move

(2) ST - the starting spindle

(3) DEST- the destination spindle

(4) INTERM-the intermediate spindle

The logic follows the general solution. If N is zero then the function is done and returns NIL. Otherwise HANOI is called recursively to move N-1 rings from the starting spindle to the intermediate spindle. When that is done, the print statement directing the user to move disc N from start to destination is executed. Finally, HANOI is called recursively to move N-1 rings from intermediate to destination.

EXERCISES

3.1 A tree is constructed out of pairs and integers. Write a BALM procedure COUNT which will return as its value the number of integers in the tree given as its argument.

3.2 Write a procedure EQLVQ(L,V) which will have the value TRUE if the list L and the vector V contain the same elements in the same order. Assume the elements are integers.

3.3 One representation of a set in BALM is as a list whose elements are the elements of the set. Assuming that the elements are either sets or integers, write a procedure EQSETQ(X,Y) which will return TRUE if sets X and Y contain the same elements.

CHAPTER 4. INPUT/OUTPUT.

4.1 Printing and Reading

BALM has two predefined files

INPUT and OUTPUT

and several predefined utility procedures for doing I/O. Other files and other I/O procedures may be specified by the user. We will discuss how to do this in the next section.

The simplest output operation uses the standard procedure PRINT:

PRINT(X1,X2,X3, ... ,XN)

X1 through XN are written on the file which is the value of OUTPUT. The value of PRINT is XN. PRINT may output more than one line. An output line is defined as having 73 characters. The first character is used for carriage control

Printing conventions for BALM items are as follows:

INTEGERS

10 08 778

Note, if a user wishes numbers printed as octal he must set a switch

OCTMODE=TRUE;

The default is for integers to be decimal

OCTMODE=NIL;

LABELS

/3/ /10/ /9/

Labels are assigned a unique number within their block by the BALM compiler. They are printed with slashes to avoid confusion with integers.

LOGICALS

TRUE NIL

IDENTIFIERS

ABC P1234 ALONGNAM

CODE

/PROC TEST/ /PROC SUMSQ/

Code is printed with the name which appeared on the left-hand side of the equal sign when the procedure was defined.

STRINGS

≠A LONG STRING≠ ≠NUMBERS 1-2-3≠

Strings are enclosed in quotes for printing.

VECTORS

[A B CD]   [[1 2] [1 3] [1 4]]

Vectors are enclosed in square brackets. The print program keeps track of how many levels of nesting have occurred. It puts as much as possible on a single line but each time it starts a new line it indents one space for each level of nesting.

```
[ONLY ONE LEVEL OF NESTING OCCURS
 HERE]
[[TWO LEVELS OF NESTING OCCUR
  HERE]]
[ONLY ONE [LEVEL OF NESTING]
 OCCURS HERE]
```

If there are more than 20 levels the print program starts over at the left margin.

PAIRS

(A . B)   (X Y Z)   (W X Y , Z)

Pairs are printed as

(1 . 2)

except when the tail is

(1) another pair

1:2:3:4 prints as (1 2 3 . 4)

(2) NIL

1:NIL prints as (1)

1:2:3:4:NIL prints as (1 2 3 4)

The same indenting rules described with vectors are used to indicate levels of nesting of pairs.

```
(THERE IS ONLY ONE LEVEL OF
 NESTING HERE)
(THIS IS AN EXAMPLE OF (TWO
  LEVELS OF NESTING))
```

A correct program should never produce anything but BALM data-objects. However, a program with a bug in it could conceivably produce a value which is not a valid BALM item. In this PRINT prints ***

This is an indication of a program bug; a correct program should never produce anything but BALM data-objects.

Frequently a user wishes to write a program which reads its data from an input file. This allows the user to vary the data without modifying the program.

The BALM system has several input functions for this purpose. The simplest is READ. READ has one argument which is a file name. We shall see in the next section how to create files, but if the user wishes to read from the input he uses the predefined variable INPUT.

```
X=READ(INPUT);
```

The value of READ is the next BALM data-object on the file. The same conventions used in printing BALM items apply to reading. Thus if

```
(1 2 3)
```

is on the input file and READ(INPUT) is executed then its value is the list whose members are 1, 2 and 3.

Since execution in BALM takes place as soon as the system encounters a semicolon, data must be interspersed with BALM programs. In the following examples, we shall follow the convention of prefacing lines input by the user with => . For example

```
=> DO A=READ(INPUT), B=READ(INPUT), C=READ(INPUT) END;
=> (A B C)  [X Y Z]  10B
=> PRINT(A);
   (A B C)
=> PRINT(B);
   [X Y Z]
=> PRINT(C);
   8
=> OCTMODE=TRUE;
=> PRINT(C);
   10B
```

READ recognizes strings, lists, identifiers, vectors, and integers, but not labels or code. Thus, it is not possible to read labels or code.

Non-alphanumeric characters such as , * = + - ; which do not have any special meaning in representing BALM items are treated as one character identifiers by READ. For example:

```
=> DO X=READ(INPUT), Y=READ(INPUT), Z=READ(INPUT) END;
=> A , B
=> PRINT(X);
   A
=> PRINT(Y);
   ,
=> PRINT(Z);
   B
```

Spaces are ignored by READ except when they serve as separators between names or numbers.

4.2 Defining Other Files

Since I/O is the most implementation dependent part of the BALM system we shall assume the user has a CDC 6600 but we shall note those functions which are general and are applicable to other implementations.

Before a file can be read it must be defined using the BALM function MAKFILE.

```
MAKFILE(FN,I)
```

returns a vector which contains information about the file necessary for reading and writing. The first argument is the file name and the second argument is the line length. The file name is an integer N meaning TAPEN in the FORTRAN sense. For example:

```
MYFILE=MAKFILE(4,72);
X=READ(MYFILE);
```

defines MYFILE as being TAPE4 and causes X to be set to the first item on that file. Alternatively on the 6600, the file name may be specified as a string.

```
NEWF=MAKFILE(≠FILZZ≠,80);
Z=READ(NEWF);
```

NEWF is defined as being the local file named FILZZ with a line length of 80. The value of Z is the first item on that file.

On the 6600 buffer space for files can be reused when the file is no longer needed. The BALM system is informed that a program no longer needs a file by use of the CLOSE operation.

```
CLOSE(FN)
```

FN may be either a string containing the file name or an integer. CLOSE puts an end of file on the file if the last operation was a WRITE. Buffers are released and may be used for another file.

Example: `CLOSE(4);`

closes the file known as TAPE4.

```
CLOSE(≠FILZZ≠);
```

closes the file known as FILZZ.

4.3 Other I/O Functions

WRITE(X,FIL);

X is written on file FIL. The value of WRITE is X. Write uses the same conventions as PRINT but the item is written on the file which is its second argument. Another difference is that WRITE is always called with 2 arguments while PRINT accepts a variable number. Example:

```
OUTF=MAKFILE(≠PUNCH≠,72);
DATA=≠ THIS IS AN OUTPUT MESSAGE≠;
WRITE(DATA,OUTF);
```

These expressions define OUTF as being the local file PUNCH, and write the string ≠THIS IS AN OUTPUT MESSAGE≠ on that file. On the 6600 this would result in one card being punched.

RDTOKEN(FIL)

RDTOKEN is quite similar to READ but slightly more primitive. RDTOKEN behaves in the same way as READ for integers, identifiers and strings, but doesn't construct lists or vectors. If it encounters a parenthesis, bracket, or period it returns an identifier whose name is a parenthesis, bracket, or period respectively. In the following example lines input by the user are prefaced by =>. For example:

```
=> DO A=RDTOKEN(INPUT), B=RDTOKEN(INPUT), C=RDTOKEN(INPUT) END+
=> ( 1 . 2 )
=> PRINT(A);
   (
=> PRINT(B);
   1
=> PRINT(C);
   .
```

The parentheses and brackets are returned as one character identifiers. For example:

```
BEGIN(),
A=RDTOKEN(INPUT),
IF A EQ IDFROMS(≠(≠) THEN MAKLIST()
ELSEIF A EQ IDFROMS(≠[≠) THEN MAKVECT()
ELSEIF A EQ *. THEN MAKD()
.
.
.
END;
```

Note that the expression IDFROMS must be used to create an identifier from parentheses or brackets. This is to avoid confusion with the notation for constant lists and vectors.

EOF(X)

EOF is a function which returns TRUE if its argument is an end of file, NIL otherwise. It is used in conjunction with READ or RDTOKEN. For example:

```
        BEGIN(ITM),
        NEWF=MAKFILE(3,89),
     RD,ITM=READ(NEWF),
        IF EOF(ITM) THEN GO FIN
        ELSE
        .
        .
        .
        GO RD,
    FIN,PRINT(≠PROCESSING COMPLETE≠)
        END;
```

SAVEBALM(FN)

SAVEBALM is a procedure which provides a facility quite different from the other read and write routines. It allows the user to write a copy of the current machine state to a file This file can subsequently be used to resume execution at a different time.

Suppose that you have written a very useful BALM procedure which several people would like to use. You might copy your BALM source program and give them each a deck of cards to place in front of their BALM programs. But, if your program is very long then may be spending most of their computer time recompiling. It would be more efficient for you to compile it and then create a saved file which they can use when they wish to call your procedure.

```
MYPROC=PROC(A,B,C),
BEGIN(),
.
.
.
END;
```

SAVEBALM(≠OWNSFIL≠); A local file known as ≠OWNSFIL≠ is written and contains the state of the machine, that is all variables, registers, procedures, etc. This file can only be read by the RESUMEAL operation.

RESUMEAL(FN)

reads the machine state from file FN and resumes execution.

```
RESUMEALL(≠OWNSFIL≠);
MYPROC(10,20,30);
```

Suppose that ≠OWNSFIL≠ is the one created by SAVEBALM above. Once the RESUMEALL has been executed MYPROC is defined and can be used. In Chapter 6 we shall discuss how to effect automatic resumption from a saved file.

There are also several more primitive I/O operations which can be used to construct read and write procedures in case the predefined procedures do not suit a user's needs.

Expression	Value
RDLINE(FN)	a string which represents the next line of the file with trailing blanks deleted. All blanks are represented by a string of length 1 consisting of ≠ ≠. When an end of file is read a string of length 0 is returned.

If the user wishes to use RDLINE with the input file he may use the integer 1 as it is synonymous with the input file.

Example:

```
S=RDLINE(1);
ABCDEFGHIJKLMNOPQRSTUVWXYZ
PRINT(S);
≠ABCDEFGHIJKLMNOPQRSTUVWXYZ≠
```

OPEN(FN,I)	value is the first argument. The first argument gives the file name and may be either an integer or string. Argument 2 gives the line length. If a file is read without first executing OPEN then the line length is assumed to be 72.

Example:

```
OPEN(≠MYFILE≠,90);
Z=RDLINE(≠MYFILE≠);
```

WRLINE(S,FN)	the string S is written on file FN and S is returned.

The integer 2 is synonymous with the output file and should be used with WRLINE on output. Example:

```
WRLINE(≠ THIS IS AN OUTPUT MESSAGE≠,2);
THIS IS AN OUTPUT MESSAGE
```

Expression (Cont'd).	Value
ENDFILE(FN)	FN is returned and an end of file is written on the designated file.
REWIND(FN)	FN is returned and the designated file is rewound.
BACKSPACE(FN)	FN is returned and the designated file is backspaced.
SAVEALL(FN)	FN is returned and the machine state is written on the designated file.

Note: the procedure SAVEBALM uses the operation SAVEALL.

CHAPTER 5. SUMMARY

We have covered now most of the features of the BALM language. Chapter 1 detailed how to combine operators to construct

expressions:

```
        A = 10+20/5
```

programs:

```
        A=10;
        PRINT(A);
```

conditional expressions:

```
        IF X THEN Y ELSE Z
        IF X THEN Y
        IF X THEN Y ELSEIF A THEN B ELSE Z
```

loops:

```
        WHILE P REPEAT C
        FOR V=(I1,I2,I3) REPEAT C
```

compound expressions:

```
        DO C1, C2, C3, ... CN END
```

and blocks:

```
        BEGIN(V1,V2, ... VN),
           X1, X2, ,.. XM
        END
```

We also learned that labels and gotos and returns are associated with blocks.

Chapter 2 describes BALM items

logical
integer
label
pointer-to-identifier
pointer-to-code
pointer-to-string
pointer-to-pair
pointer-to-vector

All the primitive operations available to construct expressions involving BALM items are listed in Chapter 2.

Of particular note are the problems inherent in assignment and the necessity for copying when items are pointers. The built-in procedures COPY(X) and EQUAL(X1,X2) provide copying capability and the ability to determine if items point to equivalent objects.

How to define a procedure is described in Chapter 3.

```
        PROCNAME= PROC(ARG1,ARG2, ... ARGN), X END
```

A procedure can contain only one expression although that expression may be a block or compound. Procedures can be written to accept a variable number of arguments by using

```
        NUMARGS()
```

which returns the number of arguments and

ARGUMENT(I)

which returns the I-th argument.

Name scoping is dynamic in BALM, that is, the meaning of a variable is determined at execution time, not compile time. All variables are global unless specifically declared local by appearing in a BEGIN list. Local means that a variable is known in the block where it is defined and in all blocks entered from that block. Likewise procedure arguments are known in the procedure where they are defined and in all procedures called from that procedure. The calling program does not receive a new value assigned to an argument by a procedure. Procedures can pass information to a calling program as a function or by modifying a string, list, or vector pointed to by an argument.

Print conventios discussed in Chapter 4 are as follows

integers	10 101B 89
logicals	TRUE NIL
labels	/10/ /3/ /2/
identifiers	ABC HELLO P12345
code	/PROC TEST/ PROC FACT/
string	≠A STRING≠ ≠1234 /// 10≠
vector	[1 2 3] [X YY Z]
pair	(10 . 20) (10) (10 20 30)

Input data should be in the same format as printed data. Procedure READ constructs the appropriate BALM item.

5.1 COMMENT

We have already noted that any line with a star in column 1 is treated as a comment. There exists a further mechanism for placing comments on a line or imbedding them in code.

COMMENT x — no value. comments are ignored. x may be any BALM constant: list, string, vector, identifier or integer, but a string is the usual choice.

COMMENT may follow any infix operator, but usually follows a comma. The program should be syntactically correct when COMMENT x is removed, so a comma should not be found both before and after the comment. Example.

```
* A PROGRAM TO DEMONSTRATE BALM
* INPUT AND OUTPUT
DEMO = PROC(),
   BEGIN(ITM),
ST, ITM = READ(INPUT),
   IF EOF(ITM) THEN GO DONE,
   IF INTQ(ITM) THEN PRINT(=TYPE,=INTEGER,ITM)
   ELSEIF LOGQ(ITM) THEN PRINT(=TYPE,=LOGICAL,ITM)
      .
      .
   ELSE PRINT(=UNKNOWN,=TYPE), COMMENT ≠ALL TYPES TESTED≠
   GOTO ST,
DONE, PRINT(≠END OF FILE ON INPUT≠)
   END
END;
```

CHAPTER 6. RUNNING A BALM PROGRAM

6.1 Source Input

The control card to direct the system to execute BALM4 on the CDC 6600 is

```
BALM4.
```

or

```
BALM4(INPUT,OUTPUT)
```

These two cards have exactly the same effect. Input and output are the default files. BALM4 expects to find the BALM source program on file INPUT. It places its output on file OUTPUT.

If the source program is on local file BINP then

```
BALM4(BINP)
```

or

```
BALM4(BINP,OUTPUT)
```

tells BALM4 that BINP is the file containing the source program. If the user desires the output from BALM programs to go to the local file BOUT then

```
BALM4(,BOUT)
```

or

```
BALM4(INPUT,BOUT)
```

are appropriate cards. Finally

```
BALM4(BINP,BOUT)
```

directs BALM4 to read from BINP and write on BOUT.

6.2 Deck Setup

Two files are necessary to run a BALM program. One is a saved file which can be created by procedure SAVEBALM described in Chapter 4. The other is a program which simulates an MBALM machine on the target machine, in our case a CDC 6600. See Appendix B for details about other target machines. Chapter 10 describes the MBALM machine and its part in the BALM system implementation.

The program which simulates the MBALM machine on the CDC 6600 expects to find the saved file on local file BLM4SVD. There are two versions available on the 6600. One version requires a field length of about 60000 octal and is suitable for short programs. The control cards to run this version are

```
X123456,CM60000.    USER NAME
ATTACH(BALM4,BLM4SVD)
BALM4,
GREEN END OF RECORD CARD
.
   BALM SOURCE PROGRAM
.
PINK END OF FILE CARD
```

The other version translates MBALM code directly into machine language. It runs between 6 and 10 times faster than the previously described version. It requires a field length of about 120000 octal and is more suitable for long programs. It can be run with the following control cards.

```
X123456,CM120000.    USER NAME
ATTACH(TBALM4,BLM4SVD,TBLM4SVD)
TBALM4.
GREEN END OF RECORD CARD
.
   BALM SOURCE PROGRAM
.
PINK END OF FILE CARD
```

The following deck setup gives an example of a program which reads cards from a file other than input.

```
X123456,CM60000.    USER NAME
ATTACH(BALM4,BLM4SVD)
COPYBR(INPUT,INFIL)
REWIND(INFIL)
BALM4,
GREEN END OF RECORD CARD
.
   DATA CARDS
.
GREEN END OF RECORD CARD
.
INP=MAKFILE(≠INFIL≠,80);
.
.
ITM=READ(INP);
.
PINK END OF FILE CARD
```

If a user wishes to automatically resume execution from a saved file which he created in an earlier run that file must be local file BLM4SVD.

```
X12330456,CM60000.   * USER NAME *
ATTACH(BALM4,BLM4SVD=OWNSAVEDFILE)
BALM4,
```

or

```
X123456,CM120000,   USER NAME
ATTACH(BALM4=TBALM4,BLM4SVD,OWNTSAVEDFILE)
BALM4,
```

Note: A saved file must be used with the same version of BALM4 with which it was created.

INTERCOM OPERATION

BALM may be run interactively using the CDC INTERCOM time-sharing system. The following commands are necessary to execute BALM programs. The system messages are in lower case, the user responses in upper case.

```
LOGIN,
.
.
command- ATTACH(BALM4,BLM4SVD)
command- EFL(60000)
command- CONNECT(INPUT)
command- BALM4.
```

In batch operation the BALM compiler echos each line of input on the output file. This can be inhibited on the teletype by setting

```
TTYFLAG=TRUE;
```

The system subsequently prints a > each time it is ready to read a line of input.

```
TTYFLAG=TRUE;
TTYFLAG=TRUE;
>PRINT(≠HI≠);
≠HI≠


>
```

The user may issue commands to the SCOPE operating system by prefacing the line with an exclamation point.

```
>PRINT(≠TESTING≠);
≠TESTING≠


>↓ETL(100)
>PRINT(≠HELLO≠);
≠HELLO≠


>
```

The INTERCOM user is normally advised to enter a semicolon as soon as an error message is received since complete compilation will not be attempted after an error. He may then start typing his program again from the previous semicolon.
For example,

```
>BEGIN(I),
>WHILE I LT 10 I=I+1
*** (I IS NOT AN OPERATOR) ***
>;
(1 ERRORS)


>
```

There is an editor available for the intercom user which permits the user to make corrections without retyping everything since the last semicolon. It is available on request and is not part of the normal BALM system. It is described in BALM Bulletin No. 12.

6.3 Compilation Errors and Messages

The name of the BALM procedure which produced the error message is listed following the message. If the explanation is unclear the user may consult the BALM system listing in Appendix F.

(ASSIGN ERROR)
ASSIGN,
caused by placing a constant on the left-hand side of an equal sign.
10=A+B

(BRACKETS DONT MATCH)
GETV,
A constant vector has a missing right bracket,
ABC= =[10 20 [1 2];

*** (N COMPILE ERRORS IN ---) ***
CODEGEN,
The number of errors produced by the code generator for the procedure whose name appears in the message.
No code is generated

(N ERRORS)
EXECUTE
The number of syntax errors in the current program.
No code is generated.

*** (ERROR IN OCTAL NUMBER) ***
LXSCAN,
An octal number with an 8 or 9 appears in the text,
1019B

*** (EXPECT ELSE HAVE ---) ***
GELSE
A conditional statement is illformed.

*** (FOR ERROR) ***
GFOR
A FOR loop is ill formed.
FOR I=1,10 REPEAT ...

*** (IMPROPER USE OF --) ***
ANALYSE,
A syntax error has occurred, caused by using a predefined infix operator incorrectly.
A=B+*X;

***(--- IS NOT AN OPERATOR) ***
OPERROR, (called from ANALYZE)
A syntax error has occurred, caused when the parser expects to find an infix operator and does not.
WHILE A EQ B PRINT(≠A≠);

```
*** (INPUT INVALID NO CODE COMPILED) ***
    SUBLBLS,
```

A syntax tree of incorrect format has been passed to the code generator or one of the generator routines created incorrect results, probably the result of user extensions.

```
*** (MISSING END AFTER BEGIN) ***
    GPROG

*** (MISSING END AFTER DO) ***
    GPROGN

*** (MISSING END AFTER PROC) ***
    GLAMBDA

*** (MISSING OPERAND BEFORE -) ***
    ANALYSE.
```

The - in the message is a) or a] or a ; . This is a syntax error caused by improper use of a built-in operator.

```
*** (MISSING --) ***
    ANALYSE.
```

The -- in the message is either a) or a]. Syntax error caused by omitting a closing) or].

```
A[I = 10;

*** (--- NOT VALID LHS) ***
    GSETG
```

An expression appeared on the left-hand side of an equal sign.

```
A(I)=10;

*** (----- NOT VALID THEN CLAUSE) ***
    GELSE
```

Probably caused by an unparenthesized nested IF clause:
IF cond THEN IF cond1 THEN expr1 ELSE expr2 ELSE expr;
Add parentheses around the inner if; correct form is
IF cond THEN(IF cond1 THEN expr1 ELSE expr2) ELSE expr;

```
(PARENS DONT MATCH)
    GETLIST,
```

A constant list is defined with missing parentheses,

```
L= =(10 A C;

*** (WARNING -------- GREATER 8 CHARACTERS) ***
    LXSCAN,
```

This is only a warning and can be ignored. Since BALM only accepts 8 character names anything else is truncated. This message serves to remind the user that BALM is seeing only 8 characters.

```
*** (WHILE ERROR) ***
    GWHILE
```

A while loop is ill formed.

```
WHILE C LT 10 THEN PRINT(C);
```

6.4 Run Time Errors and Messages

++++ ATTEMPT TO APPLY A NON PROC IN ------

The procedure named in the message contains an expression of the form

APPLY(F,L)

and F is not a BALM item of type code; possibly F had syntax errors and no code was generated. Also recall that a procedure must be defined before it can be executed.

+++ ATTEMPT TO GOTO A NON LABEL IN PROCEDURE ------

The procedure named in the message contains an expression of the form

GOTO L

where L is not a BALM item of type LABEL, possibly the definition of L occurs within a compound, loop or conditional. For example

BEGIN(), ... DO L,... END END;

BLOCK BOUNDS ERROR

Might be caused if there is no way to exit from the block. Another possibility is a reference to a label outside the block.

+++ ------ CALLS A NON PROCEDURE

The procedure named in the message contains an expression of the form

F(X)

where F is not a BALM item of type code. Possibly caused by syntax errors in F so that no code was generated. F must be defined before it can be executed.

DATA STRUCTURE EXCEEDS GARBAGE COLLECTOR STACK

The garbage collector is unable to trace a data structure. This is most likely the result of trying to use a BALM primitive with an item of incorrect type. For example

HD NIL = Y

ERROR NOT A PROCEDURE - PROGRAM ABORTED

The error recovery procedure (described in 7.1) has been redefined as an item whose type is not code.

GARBAGE COLLECTION CAN RECOVER LESS 1/20 HEAP SPACE EXECUTION TERMINATED

The garbage collector cannot recover enough space to continue executing. The user should verify that his program is not in a loop and then increase the field length. The statistics at the end indicate the maximum height of the stack. A height greater than 4000 may be an indication of infinite recursion. Infinite recursion can be caused by attempting to copy or print a structure which points to itself.

GARBAGE COLLECTOR FOUND AN INVALID TYPE
SYMBOL TABLE ENTRY ---
NAME -----

This is probably the result of attempting to use a BALM primitive with an item of incorrect type. For example

HD VECT = LIST1[I]

GARBAGE COLLECTOR FOUND AN INVALID TYPE
SYMBOL TYPE STRING ENTRY ---
NAME -----

Probably the result of attempting to use a BALM primitive with an item of incorrect type.

GARBAGE COLLECTOR FOUND AN INVALID TYPE
STACK ENTRY -----

Probably the result of attempting to use a BALM primitive with an item of incorrect type.

GARBAGE COLLECTOR FOUND AN INVALID POINTER
SYMBOL TABLE ENTRY ---
NAME -----

Probably the result of attempting to use a BALM primitive with an item of incorrect type.

GARBAGE COLLECTOR FOUND AN INVALID POINTER
SYMBOL TYPE STRING ENTRY
NAME ---

Probably the result of attempting to use a BALM primitive with an item of incorrect type. For example attempting to take the HD of a vector.

GARBAGE COLLECTOR FOUND AN INVALID POINTER
STACK ENTRY ----

Probably the result of attempting to use a BALM primitive with an item of incorrect type. For example attempting to index a list.

INCOMPLETE TRANSLATION

INSUFFICIENT SPACE FOR BALM EXECUTION

Increase the field length on the job card.

INVALID ARGUMENT TO I/O FUNCTION

Primitives RDLINE, WRLINE, REWIND, ENDFILE, BACKSPACE, SAVEALL. RESUMEAL must be called with a string or integer indicating the local file name.

INVALID DATA ON RESTORE FILE

Either the local file known as BLM4SVD at start of execution is not a BALM saved file or is incompatible with the current system, or the user's program contains an expression of the form

RESUMEAL(FN)

where FN does not contain a saved file. In the second case the user should be sure his file has been rewound.

*** ----- IS NOT A PROCEDURE

One of the user's programs contains an expression of the form

--------(X)

The value of the variable named in the message is not of type code. It may have had syntax errors when defined so that code was not generated or it has not yet been defined. In the case where the user receives the message

*** , IS NOT A PROCEDURE

he has probably used an illformed expression not detected by the parser. For example

F=PROC(X,Y), X1, X2 END;

MORE THAN -- CALLS TO ERROR - PROGRAM ABORTED

The current default allows for 10 errors. After that execution is terminated.

NOT ENOUGH SPACE FOR NEW FILE -----

Either there is not enough space to allocate buffers or the user has too many files open at the same time. Closing unused files should solve the problem.

NOT ENOUGH SPACE NEED -- MORE WORDS

The garbage collector was able to recover at least 1/20 of the total space available for data. However, a request was made for more than the space now available. If this is not the result of a program bug, the user can increase the field length.

STACK HEIGHT MISMATCH

SUBSTRING ERROR -- RESULT TRUNCATED IN PROC ----

UNDEFINED MBALM --

An attempt was made to use an MBALM opcode which does not have an instruction associated with it. Probably a system error, although could be caused by using a saved file recently created with an old simulator.

The user's job may be aborted with one of the following scope error messages.

ADDRESS OUT OF RANGE OR ARITHMETIC ERROR

This is probably the result of using a primitive operation or procedure with a BALM item of inappropriate type. For example

CONCAT(PAIR1,PAIR2); OR HD VECTOR1 OR TL NIL=1

6.5 Storage Allocationand Management

In Section 6.2 a field length of 60000 octal or 120000 octal was suggested for the two versions of BALM, respectively. Actually BALM uses as much core as it can and is limited only by the amount specified on the job card. If you have written a fairly long BALM program you may need to increase the field length.

BALM has a garbage collector which is invoked each time the system runs out of space. Execution continues as long as the garbage collector is able to recover 1/20 of the total amount of space allotted for data. This limit was put in to prevent the program from calling the garbage again and again without receiving enough space to continue execution. At the end of BALM execution some statistics are printed including the number of garbage collections. The user may assume that each garbage collection takes approximately 1 second on the CDC 6600. If the amount of time spent garbage collecting is more than half of the total execution time, the field length should be increased.

The garbage collector recovers space which was used by BALM items. As soon as a program is finished with a string, list or vector the variable could be set to NIL so that space can be recovered.

Garbage collection takes place automatically but the user may force a garbage collection if he wishes.

GARBCOLL() returns NIL, forces a garbage collection.

CHAPTER 7. PROGRAM CONTROL, TRACING AND DEBUGGING

7.1 Controls

BALM execution stops when the compiler encounters an end of file on its input. The user may also cause execution to terminate by using

```
STOP()
```

For example

```
          .
          .
IF IND GT LIMIT THEN DO
    PRINT(≠NO SOLUTION FOUND≠),
    STOP()
    END,
          .
          .
```

There is a facility for making timing studies.

TIME() returns the time since start of execution in 10ths of seconds

For example

```
BEGIN(CHK1,CHK2),
CHK1=TIME(),
F(X),
CHK2=TIME(),
PRINT(CHK2-CHK1,=SPENT,=IN,=F)
END;
```

will print the time in 10ths of seconds spent in procedure F.

ERROR RECOVERY

There is a procedure named ERROR in the BALM system. It has one argument indicating type of ERROR. Whenever one of the run time errors (listed in Section 6.4) occurs, ERROR is called. The argument is an integer with the following values.

(1) no more space

(2) invalid pointer or type found by garbage collector or structure exceeds garbage collector stack

(3) attempt to call or apply a non procedure

(4) attempt to execute an undefined op code

(5) invalid argument to an I-O instruction, i.e. not a string or integer.

(6) attempt to GOTO a non label.

The user may redefine error to suit his own needs. The current procedure is

```
ERROR=PROC(TYPE),
   IF TYPE EQ 3 THEN DO EXECUTE(INPUT,OUTPUT), STOP() END
   ELSE STOP()
END;
```

Execution terminates except when the program attempts to call an undefined procedure. In that case the compiler is called recursively. This has the effect of skipping ahead to read the next card on the input file. Compilation continues from there.

7.2 Debugging Aids

There is a debugging facility in BALM which compiles all procedures with an entry message, a print of arguments, and a message upon return giving the value of the procedure. It also compiles all GOTO's with a print of the label. The user signals that he wishes debugging aids by executing

```
DEBUG();
```

This will affect all procedures compiled from then until an end of file or

```
NODEBUG();
```

is encountered. For example

```
FACT1=PROC(X), IF ZR X THEN 1 ELSE X*FACT1(X-1) END;


DEBUG();
 FACT2=PROC(X), IF ZR X THEN 1 ELSE X*FACT2(X-1) END;


PRINT(FACT1(5));
120


FACT2(4);
FACT2 ENTERED
ARG 1 IS 4
FACT2 ENTERED
ARG 1 IS 3
FACT2 ENTERED
ARG 1 IS 2
FACT2 ENTERED
ARG 1 IS 1
FACT2 ENTERED
ARG 1 IS 0
FACT2 RETURNED 1
FACT2 RETURNED 1
FACT2 RETURNED 2
FACT2 RETURNED 6
FACT2 RETURNED 24
```

An example of goto tracing is as follows.

```
DEBUG();


BEGIN(FIRST),
 FIRST=1,
L, IF FIRST EQ 1 THEN FIRST=FIRST+1 ELSE RETURN TRUE,
 GOTO L
 END;
 GOTO L
```

There are two switches to allow the user to control printouts after the procedures have been compiled with debugging prints:

```
PRCTRACE
```

and

```
GOTRACE
```

For example

```
DEBUG();

FACT=PROC(X), IF ZR X THEN 1 ELSE X*FACT(X-1) END;

PRCTRACE=NIL;

PRINT(FACT(5));
120


PRCTRACE=TRUE;

FACT(2);
FACT ENTERED
ARG 1 IS 2
.
.
.
```

To assist in debugging on the 6600 a user may use the BALM identifier TRACE. The user is cautioned to use this facility only when for some reason he does not wish to recompile. He must be careful to turn TRACE on in the same block as the programs he wishes to trace are executed.

```
DO TRACE=2, ..... TRACE=0 END;
```

This avoids tracing the compiler procedures

TRACE - Default value is 0

1 - messages areprinted from the garbage collector indicating how many words of heap are used for lists,

vectors, etc. Also a message is printed at each request for more stack or symbol table space.

If a SAVEALL is executed the actual core used and an estimated minimal field length for RESUMEALL are printed.

2 - The names of the calling and called procedures are printed each time a call is executed. The names of the returning procedure and the procedure it is returning to and the value returned are printed when a return is executed.

3 - Both 1 and 2

4 - At end of run a core dump of the MBALM machine is printed.

5 - Both 1 and 4.

6 - Both 2 and 4.

7 - Combination of 1,2, and 4.

There exists a further tracing capability.

Expression	Value
STKTRACE(I)	returns a list of procedures and their arguments as they occur on the stack, i.e. in reverse calling sequence. I defines the depth of the trace. Each member of the list is itself a pair whose head is code and whose tail is a vector containing the arguments.

This is particularly useful when a procedure wishes to know who called it. Suppose you have a generalized message printer

```
MESS=PROC(ARG), BEGIN(N),
   N=STKTRACE(2),
   N=TL N,
   PRINT(HD N, =CALLS,=MESS),
.
.
END END;
```

The arguments in the tree will not generally be meaningful to the user. They come from the stack and areonly useful when the system capability of stack variables is used. See Chapter 11 for an explanation.

7.3 Talkative

It is possible to run the BALM compiler in talkative mode. This is useful when studying the compiler. The variable TALKATIV is the switch with which to trigger the printouts.

TALKATIVE	-	default value is 0
1	-	the compiler prints a list of global variables used following each compilation
2	-	the same as the above plus a print of the syntax tree
3	-	the same as both 1 and 2 plus a print of the syntax tree following macro expansion
4	-	the same as 1, 2 and 3 plus a print of the MBALM code generated for each statement.

CHAPTER 8. SYNTAX

8.1 Parsing

The compiler consists of three phases, which are called the parser, the macro-expander, and the code generator. The parser transforms an expression into a form called the syntax tree which shows the syntactic structure of the expression explicitly.

To illustrate theform of the syntax tree, consider the following expression

```
X * X
```

The syntax tree of this expression can be represented as

```
(* X X)
```

Here we have used the symbols (and) as a form of brackets to group together the components of the expression, with the operator being given first. Any part of a syntax tree between corresponding (and) is called a subtree, and that part of the expression which corresponds to a subtree is called a subexpression. The syntax tree of the expression

```
X*X+Y*Y
```

can be represented as

```
(+ (* X X) (* Y Y))
```

This shows that the expression is the sum of two subexpressions whose syntax trees are (* X X) and (* Y Y). The expression

```
PRINT(≠RESULT≠,A*(B+C))
```

has the syntax tree

```
(PRINT (, ≠RESULT≠ (* A (+ B C))))
```

Syntax trees may be printed by setting the switch

```
TALKATIV=2;
```

In this example each line input by the user is prefaced by =>. For example

```
=> TALKATIV=2;
=> A = B + C;
   SYNTAX TREE
   (= A (+ B C))
```

8.2 Precedence and Associativity

The BALM language is what is sometimes called an operator precedence language. That is, the syntax of the language is determined primarily by specifying a set of operators, and the precedence relations between them. Thus in the expression

X*X+Y*Y

the * and + are operators, with the * having higher precedence than +. This means that in an expression of the form

(A+B)*C

the parentheses are necessary to associate C with the expression A+B. In BALM every operator has two numbers associated with it, called its left and right precedence. If the right precedence is higher than or equal to the left, the operator is right associative.

A=B=C=D is equivalent to A=(B=(C=D))

However + and - which have the same precedence are left associative.

A-B+C-D is equivalent to ((A-B)+C)-D

Operators in BALM are classified into three types, referred to as infix, unary, and bracket. Infix operators take two arguments, with the operator being placed between the arguments. Examples of infix operators are +, *, =, THEN, etc. Unary and bracket operators are both forms of prefix operators, that is, they take one argument, and are placed before that argument. The difference between a unary operator and a bracket operator is that after the argument of a bracket operator an infix operator is expected. The function of this infix operator is to terminate the scope of the bracket operator. Examples of unary operators are HD, RETURN, and IF. Examples of bracket operators are PROC, BEGIN, and DO, all of which can be terminated by the infix operator END. The whole of the BALM language is defined in terms of these three types of operators, with the exception of parentheses, square brackets, the quote operator, the NOOP operator, and the COMMENT operator. A list of the predefined operators in BALM, together with their precedences, is given in Section 8.3.

The rules for determining whether a symbol is an operator, and whether it is infix or prefix, are as follows.

(1) An expression must start with an operand or a prefix operator.

(2) A prefix operator must be followed by another prefix operator or operand.

(3) An operand must be followed by an infix operator.

(4) An infix operator must be followed by an operand or a prefix operator (unless it is used to terminate the scope of a bracket operator, in which case it must be followed by an infix operator.

Note that these rules permit the same symbol to be a prefix and an infix operator simultaneously. The determination of the type of an operator is made in a left-to-right scan, so omitting an operator can propagate errors.

8.3 Built-In Operators

Unary and Bracket Operators

Operator	Definition	Precedence
$\geq$	Quotation of constants	9999
=	Quotation of constants	9999
COMMENT		9999
NOOP	Following operator is treated as a variable	9999
HD	Head of a list	2000
TL	Tail of a list	2000
SIZE		1900
VALUE		1900
$	Local variables	1900
-	Arithmetic negation	1700
ZR	Zero test	1200
PL	Positive test	1200
NOT	Logical negation	1200
NULL	Logical negation	1200
¬	Logical negation	1200

[continued]

Operator	Definition	Precedence
GO		500
GOTO		500
FOR		500
WHILE		500
RETURN	Block exit	500
IF		200
PROC	Procedure definition	100
BEGIN	Block definition	100
DO	Compound statement	100

Infix Operators

Operator	Definition	Associativity	Precedence
↑	Exponentiation	left	1800
/	Division	left	1600
*	Multiplication	left	1600
+	Addition	left	1500
-	Subtraction	left	1500
EQ	Equality test	right	1400
≡	Equality test	right	1400
NE	Not equal	right	1200
GE	Greater or equal	right	1200
GT	Greater than	right	1200
LT	Less than	right	1200
LE	Less or equal	right	1200
SIM	Same type	right	1200
AND	Logical and	right	1100
OR	Logical or	right	1100
:	Pair formation	right	800
=	Assignment	right	700
REPEAT		right	600
THEN		right	400
ELSE		right	300

Operator	Definition	Associativity	Precedence
ELSEIF		right	300
,	Separator	right	100
MEANS	Macro definition	right	0
END	Bracket terminator	right	0
;	Terminator	right	-1

8.4 Examples

The following is a syntax tree from a complete BALM procedure. Lines input by the user are prefaced by =>.

```
=> SUBST = PROC(A,X,L),
=>    IF A EQ L THEN A
=>    ELSEIF NOT PAIRQ(L) THEN L
=>    ELSE SUBST(A,X,HD L):SUBST(A,X,TL L)
=>    END;
SYNTAX TREE (= SUBST (PROC (, (, A (, X L))
  (IF (ELSEIF (THEN (EQ A L) A)
   (ELSE (THEN (NILQ (PAIRQ L)) L)
    (: (SUBST (, A (, X (HD L))))
     (SUBST (, A (, X (TL L))))))))))
  END))
```

This syntax is considerably more complex than any seen so far. However, if we consider that its form is the same as

```
A = B
```

which parses as

```
(= A B)
```

we can select and examine subtrees. The first subtree begins with

```
(PROC ...
```

PROC is a bracket operator and it forms a tree like the following.

```
( PROC (, ARG EXPR) END)
```

The second subtree following (PROC .. is thus its list of arguments. The third subtree is its expression. If we examine the expression we can see how a conditonal parses.

```
IF COND THEN X ELSE Y
```

parses as

```
(IF (ELSE (THEN COND X) Y))
```

CHAPTER 9. LANGUAGE EXTENSIONS

The BALM system consists of a number of routines, written in BALM, which read, compile, and execute the user's program. There is essentially no distinction between these routines and those which constitute the user's program. In a similar way, there is no distinction between the data-objects manipulated by the user's program, and the data-objects manipulated by the system. This lack of distinction permits the user to modify parts of the system to meet his own needs if necessary. In particular, the user can design his own language by modifying or extending the syntax of BALM. There are certain safeguards built into the system to prevent this being done unintentionally.

The BALM compiler has been designed so that it can be extended in a number of ways. These range from rewriting the compiler entirely, which requires considerable work and knowledge, to rather simple extensions which can be done without extensive knowledge of the workings of the system. We will concentrate on two simple but powerful extendability mechanisms. First, the user can specify new operators, their precedences and the procedures which they should correspond to. Second, the operator MEANS can be used to define new forms of expression in terms of old forms.

9.1 Defining New Operators

The following expressions, when evaluated, will define new operators.

```
UNARY(=OP,LPR,=PR)
BRACKET(=OP,LPR,=PR)
INFIX(=OP,LPR,RPR,=PR)
```

In each case OP is the operator to be defined, and PR is the identifier to be used to represent it in the syntax tree. LPR and RPR are the left and right precedences of the operator.

For example, suppose the user was writing a program in which pairs of items were associated, and the program used lists to specify these associations. The user would probably write procedures to manipulate these lists, so that, for example,

```
SEARCH(X,L)
```

would find the item associated with X on the list L. Rather than write this, he could define an operator OF so that he could write

```
X OF L
```

The following expression will define the operator

```
INFIX(=OF,1451,1450,=SEARCH);
```

The expression

```
X OF L
```

will now parse as

```
(SEARCH X L)
```

The precedence selected is between the arithmetic and the logical operators. Thus the user may write

```
IF X+N OF L1  EQ  Y*NN OF L2  THEN ......
```

which is equivalent to

```
IF ((X+N) OF L1) EQ ((Y*NN) OF L2)  THEN ...
```

The operator is left associative because its left precedence is greater than its right precedence. Thus

```
X OF L1 OF L2
```

is equivalent to

```
(X OF L1) OF L2
```

A user may also redefine existing operators. Thus if the user has written a matrix multiply procedure,

```
MMPY= PROC(X,Y),
BEGIN(),
    IF INTQ(X) AND INTQ(Y) THEN RETURN X*Y
    ELSEIF INTQ(X) AND VECTQ(Y) THEN ...
    ELSEIF INTQ(Y) AND VECTQ(X) THEN ....
    ELSEIF VECTQ(X) AND VECTQ(Y) THEN .....
    ELSE DO PRINT(≠MULTIPLY ERROR≠), RETURN NIL END
END END;
```

he may now change the meaning of * so that

```
X*Y
```

parses as

```
(MMPY X Y)
```

by writing

```
INFIX(=*,1601,1600,=MMPY);
```

Note that as long as MMPY is compiled before the operator is redefined then * will have its former meaning in MMPY. In fact any procedures compiled before * is redefined are not affected. However, suppose the user subsequently discovered an error in MMPY and wished to recompile. He might make the necessary correction and resubmit his deck. However, if he has created a saved file with a number of procedures or if he is working under time sharing, he may not wish to start fresh. There are two utility procedures which permit a user to remove the most recent definition of an operator.

```
REMINFIX(=OP);
REMUNARY(=OP);
```

He may therefore execute

```
REMINFIX(=*);
```

and

```
X * Y
```

will again parse as

```
(* X Y)
```

Now he is ready to redefine MMPY.

USE OF NOOP

Suppose a user defines a new operator

```
(=IN,1451,1450,=IN);
```

and then wishes to associate a procedure with it.

```
IN=PROC(A,B), ..... END;
```

He will receive the following diagnostic from the parser.

```
*** (IMPROPER USE OF IN) ***
```

He could reverse the order and define the procedure before declaring it to be an operator or he could use NOOP.

```
NOOP IN = PROC(A,B), ... END;
```

NOOP tells the parser to treat the identifier following it as a variable not an operator.

9.2 Macros and Use of MEANS

In an earlier example we defined an operator of so that

```
X OF Y
```

would parse as

```
(SEARCH X Y)
```

Suppose the user writes a procedure which can be used to construct these lists, so

```
L=DEF(X,Y,L)
```

will associate X with Y on list L. However, the user would like to to be able to write

```
X OF L = Y
```

to accomplish this. He may make use of macro expansion. The MEANS operator permits the user to specify how an expression in his extended language is to translate into BALM. It takes the form exprl MEANS expr2

Macro expansion in BALM takes place after parsing. Thus all operators used in exprl and expr2 must be defined and both expressions must be acceptable to the parser.

The expression

```
X1 OF X2 = X3 MEANS X2 = DEF(X1,X3,X2);
```

defines the transformation and the user may now write

```
A+10 OF LIST1 =≠ALPHA≠;
```

and it will be transformed to

```
(= LIST1 (DEF(,(+A 10) (,LIST1 ≠ALPHA≠)))
```

The variables X1,X2, ...X10 play a special part in MEANS. Occurrences of these variables in exprl match any subexpression in a parsed tree. These subexpressions are then substituted in expr2 of the MEANS. All other variables must appear in a subsequent expression or no transformation takes place. For example if the user writes

```
X OF Y = X1 MEANS Y = DEF(X,Y,X1);
```

only expressions containing those variables, such as

```
X OF Y = 10;
X OF Y = A+B+C;
```

will be transformed. Expression with other variables

```
A OF L = 10;
C + D OF LST = X+Y;
```

will not be transformed.

Note that if an XI occurs twice or more in exprl, the corresponding subexpressions must be the same. Occurrences of an XI in expr2 which did not occur in exprl will not be substituted.

As another example of the use of MEANS, suppose the programmer was dealing with two-dimensional arrays implemented as vectors of vectors, and wanted to refer to the (I,J)th element of an array M as M(I,J). Instead of M[I][J], he would do this by first specifying

```
X1[X2,X3]   MEANS   X1[X2][X3];
```

In subsequent code expressions matching X1[X2,X3] would be changed to the form X1[X2][X3], so that, for example, ABC[2*K,L+1] would be changed to ABC[2*K][L+1].

The transformations specified by MEANS definitions can be considered to take place during a left-to-right scan over the syntax tree. The actual algorithm used is the following.

(1) Search the syntax tree from the left until a subtree is found which is matched by the left-hand side of a MEANS definition. If there are two such definitions, choose the most recent.

(2) Transform the subtree according to the MEANS definition.

(3) Continue scanning from the start of the subtree just transformed. However, if the first operator in the subtree was not changed by the transformation, only consider MEANS definitions defined before the one just applied for application to this subtree. Subsequent subtrees will be scanned in the regular way.

To illustrate the algorithm let us examine the earlier example

```
SEARCH= PROC(X,Y), ,.... END;
INFIX(=OF,1451,1450,=SEARCH);
DEF=PROC(A,B,C), ..... END;
X1 OF X2 = X3 MEANS X2 = DEF(X1,X2,X3);
```

If the following statement is encountered

```
A OF LIST3 = A+10;
```

it will parse as

```
(= (SEARCH A LIST3) (+ A 10))
```

The macro expander will check to see if it has patterns starting with =, and will find

```
(= (SEARCH X1 X2) X3)
```

which does match and the syntax tree will be transformed into

```
(=LIST2 (DEF( , A (, (+A 10) LIST3))) )
```

If another MEANS expression is added

```
X1(X2) = X3 MEANS SETLIST(X1,X2,X3);
```

Then when the expression

```
A OF B = C;
```

is encountered, it parses as

```
(= (SEARCH A B ) C)
```

The macro expander first tries to match it against

```
(= (X1 (X2)) X3)
```

and fails. Then it tries to match it against

```
(= (SEARCH X1 X2) X3)
```

finds a match and makes the transformation. Thus the MEANS definitions are applied in the reverse order of definition;

that is, most recent first. The result of this process is a new syntax tree, hopefully in standard form, which is passed to the code generator for compilation.

Note that a MEANS definition in which the left-hand side subtree matches the right-hand side subtree, but not at the top level, will be regarded as inherently recursive and will not be permitted. A diagnostic will be issued. For example

```
X1 = X2 MEANS PRINT(=X1,==,X1=X2);
```

is not allowed for if expanded it would result in

```
(PRINT(, A (, (QUOTE =) (, PRINT(, A (, (QUOTE =) (,PRINT ...
```

Both the syntax tree and the expanded tree can be printed by setting

```
TALKATIV=3;
```

There is a predefined procedure which enables a user to remove the most recent macro associated with an operator.

```
REMMACRO(=OP);
```

In order to remove

```
X1(X2) = X3 MEANS SETLIST(X1,X2,X3);
```

the following expression must be executed.

```
REMMACRO(==);
```

At times it is necessary to use a variable in expr2 which does not appear in expr1; for example:

```
INFIX(=TO,1550,1550,=TO);
FOR X1 = X2 TO X3 REPEAT X4 MEANS DO G1=IF X2 LE X3 THEN 1 ELSE -1,
                                  FOR X1=(X2,X3,G1) REPEAT X4
                                  END;
```

For this purpose generated names are provided. Occurrences of G1,...,G9 in expr2 will be replaced by identifiers which are unique for the program in which the expansion occurs.

```
FOR I= 10 TO 1 REPEAT J=J+I;
```

translates into

```
DO **001= IF 10 LE 1 THEN 1 ELSE -1,
   FOR I=(10,1,**001) REPEAT J=J+I
END
```

where **001 is the generated name which replaces G1 in the expansion.

Examples

ALTERNATIVE FORMS

The following extensions permit alternative forms for procedure definition and conditionals.

```
X1=PROC(X2),X3,X4 END    MEANS    X1=PROC(X2),DO X3,X4 END END;
UNARY(=DEFINE,3000,=DEFINE);
DEFINE X1(X2) = X3    MEANS    X1=PROC(X2),X3 END;
INFIX(=↓,200,200,=↓);
X1↓X2↓X3    MEANS    IF X1 THEN X2 ELSE X3;
```

The following extension will permit the use of an arbitrary identifier as an operator of high precedence.

```
INFIX(=.,3000,3000,=.);
X1.X2    MEANS    X1(X2);
UNARY(=.,3000,=.);
.X1    MEANS    X1;
```

For example, this would permit the programmer to write PRINT.A instead of PRINT(A).

DEBUGGING

The following extensions will print out a trace of assignments and jumps

```
X1=X2    MEANS    X1=PRINT(=X1,==,X2);
GO X1    MEANS    GO DO PRINT(=GO,=X1), X1 END;
```

If required, these could be made more specific, so that only selected variables or labels were traced. The following extension will permit a selected procedure to be traced by preceding its definition with the operator trace.

```
UNARY(=TRACE,3000,=TRACE);
TRACE X1=PROC(X2),X3 END    MEANS
        X1=PROC(X2), DO PRINT(=X1,=ARGS,X2),
                        PRINT(=X1,=VALUE,X3) END END;
```

The following extension will generate traceable versions of all subsequently defined procedures.

```
TRACE(TRUE)    MEANS
   X1=PROC(X2),X3 END    MEANS
      X1=PROC(X2), IF MEMBER(=X1,TRACELIST) THEN
                   DO PRINT(=X1,=ARGS,X2),PRINT(=X1,=VALUE,X3) END
                   ELSE X3  END;
```

Those procedures will be traced whose names are on the list TRACELIST.

MORE POWERFUL LOOPS

The following expressions permit more powerful forms of loops, with the loop variable modification not restricted to integer steps.

```
INFIX(=WHILE,500,500,=WHILE);
FOR X1=X2 THEN X3 WHILE X4 REPEAT X5        MEANS
   DO X1=X2, WHILE X4 REPEAT DO X5, X1=X3 END END;
```

This permits expressions such as

```
FOR L=LL THEN TL L WHILE L NE NULL REPEAT PRINT(HD L);
```

which will print the members of list LL.

OPERATORS AS PROCEDURES

The following extensions permit unary and prefix operators to be manipulated as procedures.

```
UNARY(=PRF,3000,=PRF);
PRF=X1    MEANS    PROC(X),X1(X)END;
UNARY(=INF,3000,=INF);
INF=X1    MEANS    PROC(X,Y),X1(X,Y)END;
```

STRUCTURES

The following extensions permit the definition of structures, implemented as vectors with named components.

```
UNARY(=STRUCTURE,3000,=STRUCTURE);
STRUCTURE X1(X2)    MEANS
  DO (X1(X4) MEANS VECTOR(X4)),  COMPONENTS((1,2,3,4,5),X2) END;
COMPONENTS(X1,X2)    MEANS
  DO (X2(X4) MEANS X4[X1]), (X2(X4)=X3 MEANS X4[X1]=X3) END;
COMPONENTS((X1,X5),X2)    MEANS
  DO (X2(X4) MEANS X4[X1]), (X2(X4)=X3 MEANS X4[X1]=X3) END;
COMPONENTS((X1,X2),(X3,X4))    MEANS
  DO COMPONENTS(X1,X3), COMPONENTS(X2,X4) END;
```

As defined here this is limited to structures with up to five components. An alternative definition which permits an arbitrary number of components is defined

```
STRUCTURE X1(X2)    MEANS
  DO (X1(X4) MEANS VECTOR(X4)),  COMPS(X2) END;
COMPS(X1)    MEANS    COMPS2(=X1);
COMPS1(X1,X2)    MEANS    COMPS2(=X1,X2);
RECURSIVE(X1)    MEANS    X1;
COMPS1((X1,X2),X3)    MEANS    RECURSIVE(COMPS1(X2,(=X1,X3)));
COMPS(X1,X2)    MEANS    COMPS1(X2,=X1);
COMPS2 = PROC(), BEGIN(I), FOR I=(1,NUMARGS()) REPEAT DO
         TRANSLATE(LIST(ARGUMENT(I),=(X1),=MEANS,=X1,VECTOR(I))),
         TRANSLATE(LIST(ARGUMENT(I),=(X1),==,=X2,=MEANS,
                                   =X1,VECTOR(I),==,=X2))
         END END END;
```

CASE EXPRESSIONS

The following extensions permit a form of conditional expression called a CASE expression.

```
BRACKET(=CASE,100,=CASE);
INFIX(=ESAC,99,99,=ESAC);
CASE X1 IN X2 ESAC   MEANS   VECTOR(PROCS(X2))[X1]();
PROCS(X1)   MEANS   PROC(),X1 END;
PROCS(X1,X2)   MEANS   (PROC(),X1 END, PROCS(X2));
```

This will permit expressions such as

```
CASE MONTH IN (31,28,31,30,31,30,31,31,30,31,30,31) ESAC
```

Note that the implementation of the CASE expression as a vector of procedures will not permit the use of jumps in the list of expressions. This requires a more elaborate implementation.

PATTERN MATCHING

The BREAKUP procedure can be made available in a convenient format by the following definition.

```
=X1=X2   MEANS   BREAKUP(=X1,X2);
```

Thispermits expressions such as

```
WHILE =(H,L)=L REPEAT PRINT(H)
```

which will print the elements of the list L.

OPERATOR DEFINITION

A convenient form for defining operators of high precedence can be defined

```
INFIX(=,,3000,3000,=.);
UNARY(=OPERATOR,=100,=OPERATOR);
OPERATOR X1,X2 = X3   MEANS
    DO X1=PROC(X2),X3 END, UNARY(=X1,3000,=X1) END;
OPERATOR X1,X2,X3 = X4   MEANS
    DO X2=PROC(X1,X3),X4 END, INFIX(=X2,3000,3000,=X2) END;
```

This will permit operators to be defined in the following ways.

```
OPERATOR X.SUMSQ,Y = X*X+Y*Y;
OPERATOR SIGN.X = IF X GE 0 THEN 1 ELSE -1;
```

ALGEBRAIC MANIPULATION

The following definitions will permit the user to do algebraic manipulation.

```
UNARY(=LET,3000,=LET);
LET X1=X2 MEANS X1=SYMB(X2);
SYMB(X1) MEANS X1;
UNARY(=$,3000,=$);
SYMB($X1) MEANS CONSTSYMB(=X1);
SYMB(X1+X2) MEANS ADDSYMB(SYMB(X1),SYMB(X2));
SYMB(X1*X2) MEANS MPYSYMB(SYMB(X1),SYMB(X2));
```

This will translate, for example, the expression

```
LET X=2*Y+4*$A;
```

into the same code as

```
X=ADDSYMB(MPYSYMB(2,Y),MPYSYMB(4,CONSTSYMB(=A)));
```

9.3 Lexical Changes

The lexical scanner is that part of the BALM system which determines that

```
123ABC
```

is two items, the integer 123 and the name ABC, whereas

```
ABC123
```

is one item, a name. The rules for lexical scanning can be varied to a degree by the user if his extended language does not conform to BALM scanning rules.

The BALM system divides the characters into the following classifications.

Letters	A - Z
Numbers	0 - 9
Special	All others

The user may change a character from one class to another. Each class is associated with a number; letters are 10, specials are 11 and numbers have their integer value, 8 is 8, 4 is 4, etc. A procedure has been provided which allows the user to change a character from one class to another.

```
CHGCHAR(=CHAR,CLASSNO);
```

Thus

```
CHGCHAR(=-,10);
```

adds - to the letters and removes it from the special characters. Names like

```
A-NAME
ITEM-A
```

are now acceptable.

BALM scanning rules are as follows.

(1) An integer is begun by a digit and terminated by a letter or special (the letter B is considered part of the number and denotes octal)

(2) A name begins with a letter and is terminated by a special character.

(3) Special characters combine with nothing and are considered to be one character names, except for space which is ignored.

In addition two other classifications of special characters exist but are not used by the BALM compiler. They areincluded to give BALM users additional flexibility in extending the language. Characters with a value of 12 act as terminators. That is they can appear as the last character of an identifier. For example, if HD. and TL. are unary operators and . has a value of 12 then a user can write

```
HD.TL.Y
```

without spaces and the lexical scanner will return the 3 appropriate items. A value of 13 denotes a combining special character. It will concatenate only with other characters in its class. For example if * is given a value of 13 then the operator ** can be defined

```
CHGCHAR(=*,13);
INFIX(**,1601,1600,=↑);
```

permits a user to write

```
A=A**2;
```

as an alternative way of denoting exponentiation.

9.4 Adding or Modifying Code Generators

Adding a new code generator is more difficult and requires a more thorough knowledge of the BALM system than the extensions described in the previous sections. In the next section we present an outline of the code produced by the generators present in the current system.

In Chapter 10 a description of the MBALM machine is presented. The code produced by the generators is either calls on BALM system procedures or MBALM machine operations. The main code generator procedure is COMP which is called recursively and compiles code for each BALM expression and subexpression in the expanded tree.

The code generator is driven by two lists CODEGENLIST and OPLIST. These lists are created by procedures INITCODG and INITOPL.

Procedure COMP examines each node of the tree to see if the operator appears on the CODEGENLIST. If it does, the procedure associated with that operator is called with the tree as an argument. If the operator on a node is not on the CODEGENLIST, it is assumed to be either an expression whose value is a code block or an MBALM operation. In this case COMP calls procedure CALLS.

Procedure CALLS checks to see if the operator on the code is a member of the OPLIST. If it is, the MBALM operation code associated with it is generated. Otherwise code to evaluate it is generated. The result is assumed to be a code block, and a call to that code block is generated.

Setting switch TALKATIVE=4 produces a listing of the generated MBALM code as well as the parsed and expanded trees.

9.5 Code generated for BALM Expressions

An informal outline of the way the compiler works is given below. This shows the correspondence between a BALM expression and the code compiled for it. In each case the code compiled will stack the value of the expression. The procedure that does this in the compiler is called COMP. In some cases COMP invokes other procedures to process particular expressions. The names of these procedures are given below also.

EXPR Code Generated Compiler Procedures

```
PROC(ID1,,...,IDN),X END                                 (GLAMBDA,CODEGEN)
        (START NEW BLOCK OF CODE)
        ARG 1
        GLOB ID1
        ASTORE 1
        POP 1
        GSTORE ID1
        POP 1
        ...
        ARG N
        GLOB IDN
        ASTORE N
        POP 1
        GSTORE IDN
        POP 1
        (COMPILE CODE TO STACK VALUE OF X)
        ARG 1
        GLOB ID1
        ASTORE 1
        POP 1
        GSTORE ID1
        POP 1
        ...
        ARG N
        GLOB IDN
        ASTORE N
        POP 1
        GSTORE IDN
        POP 1
        (TERMINATE THIS BLOCK OF CODE, ASSEMBLE IT, INVENT AN
         IDENTIFIER NAM TO REFER TO IT BY, ASSIGN THE BLOCK OF
         CODE AS THE VALUE OF NAM, AND RETURN TO COMPILING CODE INTO
         THE PREVIOUS BLOCK)
        GLOB NAM
BEGIN(ID1,...,IDN),...,X,...,L,...END                  (GPROG)
        BLOCKST N
        GLOB ID1
        VSTORE 1
        ...
        GLOB IDN
        VSTORE N
        POP N
        ...
        (COMPILE CODE TO STACK VALUE OF X)
        ...
  L     SETSTK
        ...
  RET   VAR 1
        GSTORE ID1
        ...
        VAR N
        GSTORE IDN
        POP N
        BLOCKEND
```

EXPR	Code Generated	Compiler Procedures

```
FOR I=(J,K,L), REPEAT X                                         (GFOR)
        (COMPILE CODE TO STACK VALUE OF K)
        (COMPILE CODE TO STACK VALUE OF L)
        (COMPILE CODE TO STACK VALUE OF J)
        STACK NIL
        GSTORE I
        JMP LAST
   LOOP POP 1
        GSTORE I
        (COMPILE CODE TO STACK VALUE OF X)
        STEPLOOP
   LAST TLOOP LOOP
        POP 3
WHILE X1 REPEAT X2                                              (GWHILE)
        STACK NIL
   MORE (COMPILE CODE TO STACK VALUE OF X1)
        JMPF NTRUE
        POP 1
        (COMPILE CODE TO STACK VALUE OF X2)
        JMP MORE
  NTRUE ...
RETURN X                                                        (GRETURN)
        (COMPILE CODE TO STACK VALUE OF X)
        JMP RET
GOTO X                                                          (GGO)
                        ,IF X IS AN IDENTIFIER USED AS A LABEL
        JMP X
GOTO X
                        ,OTHERWISE
        (COMPILE CODE TO STACK VALUE OF X)
        JMPI
IF X1 THEN X2 ELSE X3                                           (GCOND,ETC.)
        (COMPILE CODE TO STACK VALUE OF X1)
        JMPF NO
        (COMPILE CODE TO STACK VALUE OF X2)
        JMP YES
     NO (COMPILE CODE TO STACK VALUE OF X3)
    YES
DO X1,...,XN END                                                (GPROGN)
        (COMPILE CODE TO STACK VALUE OF X1)
        POP 1
        ...
        (COMPILE CODE TO STACK VALUE OF XN)
```

EXPR	Code Generated	Compiler Procedures

```
X1 = X2                                                    (GSETQ)
            (COMPILE CODE TO STACK VALUE OF X2)
            GSTORE X1
HD X1 = X2
            (COMPILE CODE TO STACK VALUE OF X1)
            (COMPILE CODE TO STACK VALUE OF X2)
            RPLACA
X1[X2] = X3
            (COMPILE CODE TO STACK VALUE OF X1)
            (COMPILE CODE TO STACK VALUE OF X2)
            (COMPILE CODE TO STACK VALUE OF X3)
            SETINDEX
= X                                                        (GQUOTE)
            (CREATE A NEW IDENTIFIER NAM, ASSIGN X AS THE VALUE OF NAM)
            GLOB NAM
X1[X2]                                                     (CALLS)
            (COMPILE CODE TO STACK VALUE OF X1)
            (COMPILE CODE TO STACK VALUE OF X2)
            INDEX
X1 + X2
            (COMPILE CODE TO STACK VALUE OF X1)
            (COMPILE CODE TO STACK VALUE OF X2)
            +
X1(X2,...,XN)
                                 .IF X1 IS A BUILT-IN OPERATOR
            (COMPILE CODE TO STACK VALUE OF X2)
            (COMPILE CODE TO STACK VALUE OF XN)
            X1
X1(X2,...,XN)
                                 .OTHERWISE
            (COMPILE CODE TO STACK VALUE OF X2)
            (COMPILE CODE TO STACK VALUE OF XN)
            (COMPILE CODE TO STACK VALUE OF X1)
            CALL N-1
ID                                                         (GVAR)
                                 .IF ID IS AN IDENTIFIER USED AS A LABEL
            LBL ID
ID
                                 .OTHERWISE
            GLOB ID
I                                                          (GCON)
                                 .IF I NOT NEGATIVE
            NUM3 I
I
                                 .OTHERWISE
            NUM3 -I
            NEG
```

CHAPTER 10. THE MBALM MACHINE

The BALM system is designed around a virtual machine called the MBALM. Both the system routines and the user's routines are translated into machine code for this machine prior to execution. The code for the MBALM machine is executed by simulation on the target machine, or by translation into the machine code of the target machine. The BALM system is designed so that the expert user can make use of a knowledge of the MBALM machine code, but need not know how the MBALM machine is implemented.

The memory of the MBALM contains three components, called the stack, the heap and the symbol table. The basic data-object of the MBALM is called an item, and is an ordered pair whose components are called TYPE and INF. The HEAP and the STACK are ordered sets whose members are integers or items. The type of an item is either INT, LOG, ID, STR, PAIR, VECT, CODE, or LBL. The INF component of an item of TYPE INT, LOG, or LBL is an integer. The INF component of an item of TYPE STR, PAIR, VECT, or CODE is an index into the HEAP. The INF component of an item of TYPE ID is an index into the symbol-table. The symbol-table is a set of ordered triples, the components of the triples being called the name, the value, and the property-list. The NAME component is an item of TYPE STR, while the VALUE and property-list components are items.

For convenient we will use the following notation to refer to the components of items and symbol-table elements. If ITM is an ITEM, its components are referred to as ITM.TYPE and ITM.INF. If SYMB is an element of SYMTAB, its components are referred to as SYMB.NAME, SYMB.VALUE, and SYMB.PROPL. Assignment to these components is assumed to change no other elements of the stack, symbol-table or heap. An item whose TYPE and INF components are T and I is written ITEM(T,I).

10.1 Definition of Operations

In the definition of MBALM instructions given below, the following representations are used.

S	a vector represeting the stack
HEAP	a vector representing the heap
SYMTAB	a vector representing the symbol-table

The MBALM is conveniently defined using a number of variables whose values are integers, and which might be considered to be registers.

PO	the index in the heap of the current procedure
PA	the relative address of the current instruction
ST	the index of the top element of the stack
AB	the index in the stack of the zeroth argument
NA	the number of arguments
VB	the index in the stack of the zeroth variable
NV	the number of variables
NS	the number of symbol-table entries.

The following expressions are used in defining the instructions.

PUSH(X)	add X to the top of the stack
POP(X)	remove the top of the stack and assign it to X
POP()	removes the top of the stack and returns its value
CHECK(T1,,..,TN)	check that the items at the top of the stack are of types T1,...,TN with TN being the top
CHECKEITHER(T1,...,TN)	check that the item on the top of the stack has one of types T1,...,TN
INT, LOG, ID, STR, PAIR, VECT, CODE, LBL	mutually distinct integers used to distinguish types
ANY	value assumed to specify arbitrary type
LSBYTE(N)	returns I modulo 128
PARAM(I)	the integer value of the I bytes following the current instruction
GETHEAP(N)	returns the index in HEAP of a block of N consecutive unused locations
GETID(STRING)	returns the index in SYMTAB of the entry whose name component contains the same characters as STRING. If there is no such entry, creates one and returns its index.

The format used for defining the various operations of the MBALM is the following

OP(I), ..., ... ;

This represents the sequence of operations which must be executed for the instruction whose opcode is I.

The BALM code below represents an actual model of the MBALM machine. If this code is compiled the user can execute a file of MBALM instructions. Assume FORTRAN file 9 contains MBALM code. Then the following BALM instructions will load and execute it.

```
BINF=MAKFILE(9,72);
SETUP(10000,1000,500);
LOADER();
MBALMSIM();
```

```
 COMMENT
≠*****************************   CONSTANTS   ****************************≠
INFIX(=,,5001B,5000B,=,);
REMUNARY(=VALUE);
BITAND=LAND; BITOR=LOR; BITXOR=XOR;
LPREN=IDFROMS(≠(≠); RPREN=IDFROMS(≠)≠);
ASTERISK==*;
  COMMENT
≠*****************************************************************************
       AUXILIARY DEFINITIONS
*****************************************************************************≠
LLENGTH = PROC(L),
    BEGIN(I),
        I=0,
        WHILE L REPEAT DO I=I+1, L=TL L END,
        RETURN I
    END END;
PARAM = PROC(I),
    BEGIN(J,K),
        J=0,
        FOR K=(1,I) REPEAT J=J*128+NXTCODBYT(),
        RETURN J
    END END;
LSBYTE = PROC(X), X-(X/128)*128 END;
NXTCODBYT = PROC(),HEAP[PO-1+(PA=PA+1)] END;
REPLACE = PROC(LST,X), HD LST=X END;
GETHEAP = PROC(N),
    BEGIN(I),
        IF HT GT MAXH-N THEN
            DO PRINT(≠ NO MORE HEAP ≠), STOP() END
        ELSE DO I=HT+1, HT=HT+N, RETURN I END
    END END;
EXIT = PROC(),
    DO PRINT(≠ NO. OF INSTRUCTIONS EXECUTED ≠,INSTCT),
        PRINT(), PRINT(≠ NO. OF SYMBOL TABLE ENTRIES ≠,NS), PRINT(),
        PRINT(≠ SIZE OF HEAP ≠,HT,≠ SIZE OF STACK ≠,ST),
        PRINT(≠ END MBALM MACHINE SIMULATION ≠),PRINT(),STOP()
    END END;
SETUP = PROC(A,B,C),
    DO MAXH=A, MAXST=B, MAXSYM=C,
        HEAP=MAKVECTO(MAXH), S=MAKVECTO(MAXST),
        SYMTAB=MAKVECTO(MAXSYM)
    END END;
INOUT = PROC(),
```

```
  BEGIN(),
      PRINT(), PRINT(≠ PO = ≠,PO,≠ PA = ≠,PA),
      PRINT(≠ HEAP ≠), PRINT(SUBV(HEAP,1,HT)),
      PRINT(), PRINT(≠ STACK ≠), PRINT(SUBV(S,1,ST)), PRINT();
      PRINT(≠ SYMBOL TABLE ≠), PRINT(SUBV(SYMTAB,1,NS))
  END END;
 COMMENT
≠*************************************************************************
      THE FOLLOWING CODE MAKES UP A VECTOR XEQ WHOSE
      ELEMENTS ARE PROCEDURES WHICH EXECUTE THE VARIOUS
      INSTRUCTIONS
*************************************************************************≠
XEQ = MAKVECTOR(177B);
FOR I=(1,177B) REPEAT XEQ[I]=PROC(),PRINT(OP,≠IS NOT AN OP≠) END;
NOTIMPL()  MEANS  DO PRINT(OP,≠NOT IMPLEMENTED≠), STOPSIM=TRUE END;
OP(X1),X2  MEANS  XEQ[X1]=PROC().DO X2 END END;
POP(X1)  MEANS  X1=S[1+(ST=ST-1)];
POP()  MEANS  S[1+(ST=ST-1)];
PUSH(X1) MEANS DO G1=X1,S[ST=ST+1]=G1 END;
MREAD(X1,X2)  MEANS  X2=READ(X1);
MREAD(X1,X2,X3)  MEANS  DO X2=READ(X1), MREAD(X1,X3) END;
MWRITE(X1,X2)  MEANS  WRITE(X2,X1);
MWRITE(X1,X2,X3)  MEANS  DO WRITE(X2,X1), MWRITE(X1,X3) END;
X1.TYPE  MEANS  X1[1];
X1.TYPE=X2  MEANS  X1=VECTOR(X2,X1[2]);
X1.INF  MEANS  X1[2];
X1.INF=X2  MEANS  X1=VECTOR(X1[1],X2);
X1.NAME  MEANS  X1[1];
X1.NAME=X2  MEANS  X1=VECTOR(X2,X1[2],X1[3]);
X1.VALUE MEANS X1[2];
X1.VALUE=X2  MEANS  X1=VECTOR(X1[1],X2,X1[3]);
X1.PROPL  MEANS  X1[3];
X1.PROPL=X2  MEANS  X1=VECTOR(X1[1],X1[2],X2);
ITEM(X1,X2)  MEANS  VECTOR(X1,X2);
MBALMSIM = PROC(),
    BEGIN(),
        INT=1, LOG=2, ID=3, STR=4, PAIR=5, VECT=6, CODE=7, LBL=8,
        ITMNIL=ITEM(LOG,0), ITMTRUE=ITEM(LOG,1), ANY=0,
        PO=SYMTAB[1].VALUE.INF, PA=3, STOPSIM=NIL,
        INSTCT=0,
        WHILE ¬STOPSIM REPEAT DO OP=NXTCODBYT(),
              INSTCT=INSTCT+1, XEQ[OP]() END,
        RETURN NIL
    END END;
ST=0;
IDENT = PROC(S1,S2),
    BEGIN(I,IP1,IP2),
        IF HEAP[(IP1=S1.INF)] NE HEAP[(IP2=S2.INF)] THEN RETURN NIL
        ELSE FOR I=(1,HEAP[IP1]) REPEAT
            IF HEAP[IP1+I] NE HEAP[IP2+I] THEN RETURN NIL,
         RETURN TRUE
    END END;
GETID = PROC(W),
```

```
    BEGIN(I),
       I=1,
       WHILE I LE NS AND NOT IDENT(SYMTAB[I].NAME,W) REPEAT I=I+1,
       IF I LE NS THEN RETURN I,
       NS=NS+1,
       IF NS GT MAXSYM THEN DO PRINT(≠ NO MORE SYMBOL TABLE≠),
          STOP() END,
       SYMTAB[NS]=VECTOR(W,NIL,ITMNIL), RETURN NS
    END END;
CHECK = PROC(),
    BEGIN(N,I),
       N=NUMARGS(),
       FOR I=(1,N) REPEAT
          IF ARGUMENT(I) NE S[ST-N+I].TYPE AND
          ARGUMENT(I) NE ANY THEN DO PRINT(OP,≠ ARGS IN ERROR ≠),
          PRINT(SUBV(HEAP,PO,PA+10)),PRINT(SUBV(S,1,ST)),EXIT() END
    END END;
CHECKEITHER = PROC(),
    BEGIN(N,I),
       N=NUMARGS(),
       FOR I=(1,N) REPEAT IF ARGUMENT(I) EQ S[ST].TYPE THEN RETURN I,
       PRINT(OP,≠ OP INVALID ARGUMENT ≠), PRINT(SUBV(HEAP,PO,PA+10)),
       PRINT(SUBV(S,1,ST)), EXIT()
    END END;
  COMMENT
≠*******************************************************************
       DEFINITIONS OF INSTRUCTIONS
*******************************************************************≠

*********          THREE BYTE OPERATIONS
*********       OPCODE PARAMBYTE1 PARAMEBYTE2
**JMPT L2(X)
   OP(1B),   IF EQUAL(POP(),ITMNIL) THEN PA=PA+2 ELSE PA=PARAM(2);
*JMPF L2(X)
   OP(2B), IF EQUAL(POP(),ITMNIL) THEN PA=PARAM(2) ELSE PA=PA+2;
*JMP L2
   OP(3B),   PA=PARAM(2);
*NUM2 I2
   OP(4B),   PUSH(ITEM(INT,PARAM(2)));
*GLOB ID2
   OP(5B),   I=PARAM(2), PUSH(SYMTAB[I].VALUE);
*GSTORE ID2(X)
   OP(6B),   I=PARAM(2), SYMTAB[I].VALUE=S[ST];
*LIST(X1,X2,...,XI,I)
   OP(10B), PUSH(NIL), FOR I=(1,PARAM(2)) REPEAT DO
            NEW=GETHEAP(2), POP(HEAP[1+NEW]), POP(HEAP[NEW]),
            PUSH(ITEM(PAIR,NEW)) END;
*LBL L2
   OP(11B), PUSH(ITEM(LBL,PARAM(2)));
*VECTOR(X1,X2,...,XI,I)
   OP(12B), K=PARAM(2), NEW=GETHEAP(K+1), HEAP[NEW]=K,
            FOR I=(K,1,-1) REPEAT HEAP[NEW+I]=POP(X),
            PUSH(ITEM(VECT,NEW));
```

```
*STRING(I1,I2,...,IJ,J)
   OP(13B), N=PARAM(2), NEW=GETHEAP(N+1), HEAP[NEW]=N,
            FOR I=(N,1,-1) REPEAT DO CHECK(INT), HEAP[NEW+I]=POP().INF
            END, PUSH(ITEM(STR,NEW));
*TLOOP L2(I,I,I)
   OP(14B), POP(X),CHECK(INT,INT,INT),PUSH(X),I=S[ST-1].INF,
            J=S[ST-2].INF,   K=S[ST-3].INF,
            IF J GT 0 AND I LE K OR J LT 0 AND I GE K THEN PA=PARAM(2)
            ELSE DO PA=PA+2,S[ST-3]=S[ST] END;
*
**********          TWO BYTE OPERATIONS
**********       OPCODE PARAMETER
*NUM1 I1
   OP(26B), PUSH(ITEM(INT,PARAM(1)));
*CALL I1(X1,X2,...,XI1,C)
   OP(27B), CHECK(CODE),POP(C),PUSH(ITEM(CODE,PC)),PUSH(PA+1),PUSH(AB),
            PUSH(NA),
            NA=PARAM(1),PO=C.INF,PA=3,AB=ST-4-NA;
*VAR I1
   OP(31B), PUSH(S[VB+PARAM(1)]);
*VSTORE I1(X)
   OP(32B), S[VB+PARAM(1)]=S[ST];
*ARG I1
   OP(33B), PUSH(S[AB+PARAM(1)]);
*ASTORE I1(X)
   OP(34B), S[AB+PARAM(1)]=S[ST];
*POP I1
   OP(35B), ST=ST-PARAM(1);
*BLOCKST I1
   OP(36B), PUSH(VB), PUSH(NV), VB=ST, NV=PARAM(1), ST=VB+NV,
            FOR I=(1,NV) REPEAT S[VB+I]=ITMNIL;
*
**********          FOUR BYTE OPERATIONS
**********       OPCODE PARAMBYTE1 PARAMBYTE2 PARAMBYTE3
*NUM3 I3
   OP(37B), PUSH(ITEM(INT,PARAM(3)));
*
**********          ONE BYTE OPERATIONS
**********       OPCODE
*ID(I)
   OP(41B), CHECK(INT), PUSH(ITEM(ID,POP().INF));
*ARG(I)
   OP(42B), CHECK(INT), PUSH(S[AB+POP().INF]);
*IDFROMC(C)
   OP(43B), CHECK(CODE), C=POP().INF,
            PUSH(ITEM(ID,HEAP[C+1]*128+HEAP[C+2]));
*SFROMV(V)
   OP(45B), CHECK(VECT), V=POP().INF, L=HEAP[V], NEW=GETHEAP(L+1),
            FOR I=(1,L) REPEAT HEAP[NEW+I]=LSBYTE(HEAP[V+I].INF),
            HEAP[NEW]=L, PUSH(ITEM(STR,NEW));
*VFROMS(S)
   OP(46B), CHECK(STR), SU=POP().INF, L=HEAP[SU], NEW=GETHEAP(L+1),
            FOR I=(1,L) REPEAT HEAP[NEW+I]=ITEM(INT,HEAP[SU+I]),
            HEAP[NEW]=L, PUSH(ITEM(VECT,NEW));
```

```
*NARGS
   OP(47B), PUSH(ITEM(INT,NA));
*VALUE(ID)
   OP(50B), CHECK(ID), I=POP().INF, PUSH(SYMTAB[I].VALUE);
*VALUE(ID) = X
   OP(51B), POP(X), CHECK(ID), SYMTAB[POP().INF].VALUE=X; PUSH(X);
*JMPI(L)
   OP(52B), CHECK(LBL), PA=S[ST].INF;
*STEPLOOP (I,I,I)
   OP(53B), CHECK(INT,INT,ANY),S[ST-1].INF=S[ST-1].INF+S[ST-2].INF;
*IDFROMS(S)
   OP(60B), CHECK(STR), POP(NAM), PUSH(ITEM(ID,GETID(NAM)));
*PAIR(X,X)
   OP(61B), NEW=GETHEAP(2), POP(HEAP[NEW+1]), POP(HEAP[NEW]),
            PUSH(ITEM(PAIR,NEW));
*XOR(I,J)
   OP(67B), CHECK(INT,INT), J=POP().INF, I=POP().INF,
            PUSH(ITEM(INT,BITXOR(I,J)));
*SHIFT(I,J)
   OP(70B), CHECK(INT,INT), J=POP().INF, I=POP().INF,
            PUSH(ITEM(INT,SHIFT(I,J)));
*I+J
   OP(71B), CHECK(INT,INT), POP(J), POP(I),
            PUSH(ITM(INT,I.INF+J.INF));
*I-J
   OP(72B), CHECK(INT,INT), POP(J), POP(I),
            PUSH(ITEM(INT,I.INF-J.INF));
*I*J
   OP(73B), CHECK(INT,INT), POP(J), POP(I),
            PUSH(ITEM(INT,I.INF*J.INF));
*I/J
   OP(74B), CHECK(INT,INT), POP(J), POP(I),
            PUSH(ITEM(INT,I.INF/J.INF));
*I↑J
   OP(75B), CHECK(INT,INT), POP(J), POP(I),
            PUSH(ITEM(INT,I.INFJ.INF));
*-I
   OP(76B), CHECK(INT), PUSH(ITEM(INT,-POP().INF));
*INTQ(X)
   OP(77B), IF POP().TYPE EQ INT THEN PUSH(ITMTRUE) ELSE PUSH(ITMNIL);
*STRQ(X)
   OP(101B),IF POP().TYPE EQ STR THEN PUSH(ITMTRUE) ELSE PUSH(ITMNIL);
*VECTQ(X)
   OP(102B),IF POP().TYPE EQ VECT THEN PUSH(ITMTRUE) ELSE PUSH(ITMNIL);
*PAIRQ(X)
   OP(103B),IF POP().TYPE EQ PAIR THEN PUSH(ITMTRUE) ELSE PUSH(ITMNIL);
*CODEQ(X)
   OP(104B),IF POP().TYPE EQ CODE THEN PUSH(ITMTRUE) ELSE PUSH(ITMNIL);
*IDQ(X)
   OP(105B),IF POP().TYPE EQ ID  THEN PUSH(ITMTRUE) ELSE PUSH(ITMNIL);
*LBLQ(X)
   OP(107B),IF POP().TYPE EQ LBL THEN PUSH(ITMTRUE) ELSE PUSH(ITMNIL);
*PROP(ID) = X
```

```
   OP(110B),POP(X),CHECK(ID), I=POP().INF, SYMTAB[I].PROPL=X,PUSH(X);
*PL(I)
   OP(111B),CHECK(INT), PUSH(IF POP().INF GE 0 THEN ITMTRUE
              ELSE ITMNIL);
*ZR(I)
   OP(112B),CHECK(INT), PUSH(IF POP().INF EQ 0 THEN ITMTRUE
              ELSE ITMNIL);
*IDENTQ(X,X)
   OP(113B),POP(X), POP(Y), IF X.TYPE EQ Y.TYPE AND X.INF EQ Y.INF
              THEN PUSH(ITMTRUE) ELSE PUSH(ITMNIL);
*SIZE(X)
   OP(114B),CHECKEITHER(VECT,STR,CODE), POP(X),
              PUSH(ITEM(INT,HEAP[X.INF]));
*RETURN(X)
   OP(115B),POP(R), ST=AB+NA+4, NEWST=AB,
              POP(NA), POP(AB), POP(PA),PO=POP().INF,  ST=NEWST, PUSH(R);
*STOP
   OP(116B),STOPSIM=TRUE;
*V[I]
   OP(117B),CHECK(VECT,INT), POP(I), POP(V), PUSH(HEAP[V.INF+I.INF]);
*V[I] = X
   OP(120B),POP(X), CHECK(VECT,INT), POP(I), POP(V),
              HEAP[V.INF+I.INF]=X, PUSH(X);
*HD P = X
   OP(121B),POP(X), CHECK(PAIR), POP(P), HEAP[P.INF]=X, PUSH(X);
*TL P = X
   OP(122B),POP(X), CHECK(PAIR), POP(P), HEAP[1+P.INF]=X, PUSH(X);
*HD P
   OP(123B),CHECK(PAIR), POP(P), PUSH(HEAP[P.INF]);
*TL P
   OP(124B),CHECK(PAIR), POP(P), PUSH(HEAP[1+P.INF]);
*LAND(I,J)
   OP(125B),CHECK(INT,INT), J=POP().INF, I=POP().INF,
              PUSH(ITEM(INT,BITAND(I,J)));
*LOR(I,J)
   OP(126B),CHECK(INT,INT), J=POP().INF, I=POP().INF,
              PUSH(ITEM(INT,BITOR(I,J)));
*COMPL(I)
   OP(127B),CHECK(INT), PUSH(ITEM(INT,COMPL(POP().INF)));
*SFROMID(ID)
   OP(130B),CHECK(ID), N=SYMTAB[POP().INF].NAME.INF, L=HEAP[N],
              NEW=GETHEAP(L+1), FOR I=(1,L) REPEAT HEAP[NEW+I]=HEAP[N+I],
              HEAP[NEW]=L,PUSH(ITEM(STR,NEW));
*BLOCKEND(X)
   OP(131B),POP(R), ST=VB, POP(NV), POP(VB), PUSH(R);
*NOT(Q)
   OP(132B),IF EQUAL(POP(), ITMNIL) THEN PUSH(ITMTRUE)
              ELSE PUSH(ITMNIL);
*EQSTR(S1,S2)
   OP(133B), BEGIN(), POP(X), POP(Y), IF X.TYPE NE Y.TYPE OR
             X.TYPE NE STR THEN RETURN PUSH(ITMNIL),
             X=X.INF, LX=HEAP[X], Y=Y.INF, LY=HEAP[Y],
             IF LX NE LY THEN RETURN PUSH(ITMNIL)
```

```
          ELSE FOR I=(1,LX) REPEAT
          IF HEAP[X+I] NE HEAP[Y+I] THEN RETURN PUSH(ITMNIL);
          PUSH(ITMTRUE)
          END;
*SETSX(X)
   OP(134B), S[VB+NV+1]=S[ST], ST=VB+NV+1;
*TRUE
   OP(135B),PUSH(ITMTRUE);
*NIL
   OP(136B),PUSH(ITMNIL);
*SETSTK
   OP(137B),ST=VB+NV;
*MAKVECTOR(I)
   OP(140B),CHECK(INT), L=POP(X).INF, NEW=GETHEAP(L+1), HEAP[NEW]=L,
          PUSH(ITEM(VECT,NEW)), FOR I=(1,L) REPEAT HEAP[NEW+I]=ITMNIL;
*RDLINE(I)
   OP(141B),CHECK(INT), LIN=VFROMS(RDLINE(POP().INF)), L=SIZE(LIN),
          NEW=GETHEAP(L+1), HEAP[NEW]=L, SUB(HEAP,NEW+1,L)=LIN,
          PUSH(ITEM(STR,NEW));
*WRLINE(S,I)
   OP(142B),CHECK(INT), N=POP().INF, CHECK(STR), LIN=S[ST].INF;
          WRLINE(SFROMV(SUB(HEAP,LIN+1,HEAP[LIN])),N);
*REWIND(I)
   OP(143B),CHECK(INT), I=S[ST].INF, REWIND(I);
*BACKSPACE(I)
   OP(144B),CHECK(INT), I=S[ST].INF,BACKSPACE(I);
*SAVEALL(I)
   OP(145B),CHECK(INT), N=S[ST].INF,
          MWRITE(N,PO,PA,ST,AB,NA,VB,NV,HEAP,SYMTAB,S);
*RESUMEALL(I)
   OP(146B),CHECK(INT), N=POP().INF,
          MREAD (N,PO,PA,ST,AB,NA,VB,NV,HEAP,SYMTAB,S);
          PUSH(ITMTRUE);
*ENDFILE(I)
   OP(147B),CHECK(INT), N=S[ST].INF, ENDFILE(N);
*CFROMV(V)
   OP(150B),CHECK(VECT), V=POP().INF, L=HEAP[V], NEW=GETHEAP(L+1),
          FOR I=(1,L) REPEAT HEAP[NEW+I]=LSBYTE(HEAP[V+I].INF),
          HEAP[NEW]=L, PUSH(ITEM(CODE,NEW));
*MODE(X)
   OP(151B),NOTIMPL();
*SETMODE(X,I)
   OP(152B),NOTIMPL();
*GARBAGE COLLECT
   OP(153B),PUSH(ITMNIL);
*TIME
   OP(154B),PUSH(ITEM(INT,TIME()));
*PROTECT(ID)
   OP(155B),CHECK(ID), J=S[ST].INF,
          NAM=SYMTAB[J].NAME.INF, L=HEAP[NAM], NEW=GETHEAP(L+2),
          HEAP[NEW]=L+1, HEAP[NEW+1]=ASTERISK,
          FOR I=(1,L) REPEAT HEAP[NEW+1+I]=HEAP[NAM+I];
          SYMTAB[J].NAME=ITEM(STR,NEW);
```

```
*PROP(ID)
   OP(160B),CHECK(ID), PUSH(SYMTAB[POP().INF].PROPL);
*LOGQ(X)
   OP(161B),IF POP().TYPE EQ LOG THEN PUSH(ITMTRUE) ELSE PUSH(ITMNIL);
*SIMTYPEQ(X,X)
   OP(162B),PUSH(IF POP().TYPE EQ POP().TYPE THEN ITMTRUE ELSE ITMNIL);
*SUB(V,I,J) OR SUB(S,I,J)
   OP(163B),CHECK(INT,INT), L=POP().INF, I=POP().INF,
            CHECKEITHER(STR,VECT), POP(X), I=X.INF+I-1,
            NEW=GETHEAP(L+1), HEAP[NEW]=L,
            FOR  J=(1,L) REPEAT HEAP[NEW+J]=HEAP[I+J],
            PUSH(ITEM(X.TYPE,NEW));
*SUB(V,I,J) = X OR SUB(S,I,J) = X
   OP(164B),CHECKEITHER(STR,VECT),POP(X),CHECK(INT,INT),L=POP().INF,
            I=POP().INF,CHECK(X.TYPE),POP(Y),I=Y.INF+I-1,XX=X.INF,
            FOR J=(1,L) REPEAT HEAP[I+J]=HEAP[XX+J],PUSH(X);
**CONCAT(V,V) OR CONCAT(S,S)
   OP(165B),CHECKEITHER(STR,VECT), POP(X), CHECK(T=X.TYPE), POP(Y),
            X=X.INF, LX=HEAP[X], Y=Y.INF, LY=HEAP[Y], L=LX+LY,
            NEW=GETHEAP(L+1), HEAP[NEW]=L,
            FOR I=(1,LY) REPEAT HEAP[NEW+I]= HEAP[Y+I],
            FOR I=(1,LX) REPEAT HEAP[NEW+LY+I]=HEAP[X+I],
            PUSH(ITEM(T,NEW));
*STKTRACE(I)
   OP(170B),CHECK(INT),IST=ST,POP(J1),N=0,IPO=ITEM(CODE,PO),IARG=NA,
            IAB=AB, WHILE NOT ZR IAB AND N LT J1 REPEAT DO
               K=GETHEAP(IARG+5), J=K+4, L=S[ST].INF,
               PUSH(ITEM(PAIR,K)), IF ST GT IST THEN HEAP[L+1]=S[ST],
               HEAP[J]=IARG ,
               FOR I=(1,IARG) REPEAT HEAP[J+I]=S[IAB+I],
               HEAP[K]=ITEM(PAIR,K+2), HEAP[K+2]=IPO,
               HEAP[K+3]=ITEM(VECT,J), IPO=S[IAB+IARG+1],
               T=IARG,
               IARG=S[IAB+IARG+4], IAB=S[IAB+T+3], N=N+1
               END,
            HEAP[K+1]=ITMNIL, ST=IST;
APPLY(C,X)
   OP(171B),CHECK(CODE,ANY), L=S[ST], J=POP().INF,
            C=S[ST].INF, INA=0,
            BEGIN(),
            LOOP, INA=INA+1, IF L.TYPE EQ PAIR THEN DO
               PUSH(HEAP[J]), L=HEAP[J+1], J=L.INF,
               IF L.TYPE NE NIL THEN GOTO LOOP END
               ELSE PUSH(L) END,
            PUSH(ITEM(CODE,PO)), PUSH(PA+1), PUSH(AB),
            PUSH(NA), PO=C, PA=3, NA=INA, AB=ST-4-NA;
```

To illustrate the sort of code used in the MBALM, consider the following BALM procedure.

```
SUMV=PROC(V), BEGIN(S,I,N),
     N=SIZE(V), S=0, I=1,
NXT, IF I GT N THEN RETURN S,
     S=S+V[I], I=I+1, GOTO NXT,
     END END;
```

This procedure computes the sum of the elements of a vector V, assumed to be integers. The MBALM code produced for this procedure is as follows.

```
≠  BYTE      OP,            INST,                   OPERAND ≠
≠ ---------------------------------------------------------- ≠
     3B      33B    ≠ ARG       I1          ≠       1B
     5B       5B    ≠ GLOB    ID2           ≠     601B      V
    10B      34B    ≠ ASTORE  I1(X)         ≠       1B
    12B      35B    ≠ POP       I1          ≠       1B
    14B       6B    ≠ GSTORE  ID2(X)        ≠     601B      V
    17B      35B    ≠ POP       I1          ≠       1B
    21B      36B    ≠ BLOCKST I1            ≠       3B
    23B       5B    ≠ GLOB    ID2           ≠     662B      S
    26B      32B    ≠ VSTORE   I1(X)        ≠       1B
    30B       5B    ≠ GLOB    ID2           ≠     530B      I
    33B      32B    ≠ VSTORE   I1(X)        ≠       2B
    35B       5B    ≠ GLOB    ID2           ≠     612B      N
    40B      32B    ≠ VSTORE   I1(X)        ≠       3B
    42B     137B    ≠ SETSTK ≠
    43B       5B    ≠ GLOB    ID2           ≠     601B      V
    46B     114B    ≠ LENGTH  (X) ≠
    47B       6B    ≠ GSTORE  ID2(X)        ≠     612B      N
    52B     137B    ≠ SETSTK ≠
    53B      26B    ≠ NUM1      I1          ≠       0B
    55B       6B    ≠ GSTORE  ID2(X)        ≠     662B      S
    60B     137B    ≠ SETSTK ≠
    61B      26B    ≠ NUM1      I1          ≠       1B
    63B       6B    ≠ GSTORE  ID2(X)        ≠     530B      I
    66B     137B    ≠ SETSTK ≠
    67B       5B    ≠ GLOB    ID2           ≠     530B      I
    72B       5B    ≠ GLOB    ID2           ≠     612B      N
    75B      72B    ≠ SUBTRACT I-J ≠
    76B      76B    ≠ NEGATE   -I ≠
    77B     111B    ≠ IPOSQ(I) ≠
   100B     132B    ≠ NOT(Q) ≠
   101B       2B    ≠ JMPF     L2(X)        ≠     112B
   104B       5B    ≠ GLOB    ID2           ≠     662B      S
   107B       3B    ≠ JMP       L2          ≠     150B
   112B     136B    ≠ NIL ≠
   113B     137B    ≠ SETSTK ≠
   114B       5B    ≠ GLOB    ID2           ≠     662B      S
   117B       5B    ≠ GLOB    ID2           ≠     601B      V
   122B       5B    ≠ GLOB    ID2           ≠     530B      I
   125B     117B    ≠ V[I] ≠
```

```
1268    71B    ≠ ADD I+J ≠
1278     6B    ≠ GSTORE ID2(X)        ≠    662B     S
1328   137B    ≠ SETSTK ≠
1338     5B    ≠ GLOB   ID2           ≠    530B     I
1368    26B    ≠ NUM1    I1           ≠      1B
1408    71B    ≠ ADD I+J ≠
1418     6B    ≠ GSTORE ID2(X)        ≠    530B     I
1448   137B    ≠ SETSTK ≠
1458     3B    ≠ JMP     L2           ≠     66B
1508    31B    ≠ VAR     I1           ≠      1B
1528     6B    ≠ GSTORE ID2(X)        ≠    662B     S
1558    35B    ≠ POP     I1           ≠      1B
1578    31B    ≠ VAR     I1           ≠      2B
1618     6B    ≠ GSTORE ID2(X)        ≠    530B     I
1648    35B    ≠ POP     I1           ≠      1B
1668    31B    ≠ VAR     I1           ≠      3B
1708     6B    ≠ GSTORE ID2(X)        ≠    612B     N
1738    35B    ≠ POP     I1           ≠      1B
1758   131B    ≠ BLOCKEND(X) ≠
1768    33B    ≠ ARG     I1           ≠      1B
2008     5B    ≠ GLOB   ID2           ≠    601B     V
2038    34B    ≠ ASTORE I1(X)         ≠      1B
2058    35B    ≠ POP     I1           ≠      1B
2078     6B    ≠ GSTORE ID2(X)        ≠    601B     V
2128    35B    ≠ POP     I1           ≠      1B
2148   115B    ≠ RETURN(X) ≠
```

10.2 MBALM Software -- Loader, Garbage Collector

```
LOADER = PROC(),
    BEGIN(PROK,LLIST,LST,I,LEN,RS,ISAV,IP,N1,P),
       NS=HT=0, PROK=NIL,
       CODE=7, STR=4, LOG=2,
       ITMNIL=ITEM(LOG,0),
 STRT, LLIST=LST=NIL:NIL,
 NXT,  RS=RDTOKEN(BINF), IF EOF(RS) THEN RETURN NIL,
       IF RS EQ LPREN THEN DO PROK=TRUE, GOTO NXT END,
       IF RS EQ RPREN THEN GOTO ENTER,
       IF IDQ(RS) THEN DO L=VFROMS(SFROMID(RS)),
          P=GETHEAP(1+(J=SIZE L)), HEAP[P]=J,
          FOR I=(1,J) REPEAT HEAP[P+I]=L[I],
          RS=GETID(ITEM(STR,P)),
          IF PROK THEN DO PROK=NIL, ISAV=RS END END,
       IF RS GT 128 THEN DO N1=RS/128, RS=RS-N1*128,
          LST=REPLACE(LST,N1) END,
       LST=ADDON(LST,RS), GOTO NXT,
ENTER, LEN=LLENGTH(LLIST=TL LLIST),IP=GETHEAP(LEN+1),
       FOR I=(1,LEN) REPEAT DO HEAP[IP+I]=HD LLIST,LLIST=TL LLIST END,
       HEAP[IP]=LEN, SYMTAB[ISAV].VALUE=ITEM(CODE,IP),
       GOTO STRT
    END END;
```

Input to the loader is a series of lists of MBALM instructions representing procedures. The loader locates each name in the symbol table and if not there makes an entry. It generates code blocks with each occurrence of a name replaced by its symbol table entry number. MBALM instructions are assumed to be in seven bit bytes. Thus the number 256 is represented by two bytes

```
0000010 0000000
```

or in octal

```
2B 0B
```

For example, loading a constant 256 requires the following 3 byte MBALM instruction

```
4B 2B 0B
```

The input to the loader for the example from the end of the previous section is as follows.

```
( 0B SUMV 33B 1B 5B 0B V 34B 1B 6B 0B V 35B 1B 36B 3B
  5B 0B S 32B 1B 5B 0B I 32B 2B 5B 0B N 32B 33B 137B
  5B 0B V 114B 6B 0B N 137B 26B 0B 6B 0B S 137B 26B 1B
  6B 0B I 137B 5B 0B I 5B 0B N 72B 76B 111B 132B 2B 0B
  112B 5B 0B S 3B 0B 150B 136B 137B 5B 0B S 5B 0B V 5B
  0B I 117B 71B 6B 0B S 137B 5B 0B I 26B 1B 71B 6B 0B I
  137B 3B 0B 66B 31B 1B 6B 0B S 35B 1B 31B 2B 6B 0B I
  35B 1B 31B 3B 6B 0B N 35B 1B 131B 33B 1B 5B 0B V
  34B 1B 35B 1B 6B 0B V 35B 1B 115B )
```

GARBAGE COLLECTOR

The garbage collector is not illustrated here as there are sufficient discussions of garbage collectors elsewhere. Whenever GETHEAP is unable to fulfill a request the garbage collector is invoked. The garbage collector scans the value and type entries of the symbol table and the stack. It flags each heap entry pointed to from the stack and symbol table. Unused heap entries are recovered and reused.

10.3 Bootstrapping

The BALM system consists of two separate modules. The first module is the system itself written in BALM, and containing I/O routines, lexical scanner, a parser, a code generator and various utility routines. This module is machine independent in that the code generator produces code for a virtual machine called the MBALM. The second module is a mechanism for executing the MBALM machine code. This module is different for each target machine. There are MBALM simulators available for the CDC 6600, the IBM 360, the UNIVAC 1108, XDS SIGMA 5, and the DEC PDP10. An execution module which translates the MBALM code into machine code is also available for the CDC 6600.

Bootstrapping is the process by which a compiler moves itself from one machine or system to another. Since the BALM compiler is written in BALM, the compiler is capable of compiling itself. To implement BALM on a new computer requires

(1) writing a program to execute MBALM instructions, i.e. an MBALM simulator

(2) writing a loader program to read MBALM instructions and create code blocks and symbol table entries.

(3) writing a garbage collector

(4) a file of MBALM code produced by compiling the BALM compiler. The listing of the BALM system in Appendix F is preceded by a procedure called bootstrap which generates such a file.

Once the loader has read the MBALM instructions, execution begins with the first procedure loaded. This is procedure BALM. It calls INITIATE which in turn calls various procedures necessary to initialize the system: see the flow diagram in Chapter 11. One of these, INITIO, defines the BALM character set for the new machine by reading 3 cards. These cards are the first cards in the BALM supplement. The characters start in column 2 and the first character on the first card is a space.

```
 .*-≠()[]¬$,=:+≡/≥;↑
0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

The special characters appear on the first card. They are in a predefined order so that if one wishes to use a character other than ; for a terminator it should appear in column 21. The second card contains the numbers 0-9 in ascending order. The third card contains the alphabet in alphabetical order but may include special characters at the end. These characters will then be considered as part of the alphabet in forming names.

CHAPTER 11. SYSTEM PROGRAMMER'S GUIDE

This section is intended to be a guide to the source listing which is included in Appendix F. It is directed to the user who wishes a more detailed knowledge of the system in order to extend the compiler. Several features about the compiler require some explanation. Originally all compiler procedures and variables were accessible to the user. This turned out to be quite unsatisfactory because it led to inadvertent modification of the compiler. If a user selected the name BLANK and used it as a variable he would begin getting strange diagnostics because the lexical scanner uses the predefined variable BLANK. To protect a BALM user from these types of problems all critical variables and procedures have a name which includes a special character. In order to compile the BALM compiler . must be treated as a letter, as described in Section 9.3. The way to do this is

```
CHGCHAR(=,.10);
```

Thus a user is protected from inadvertent modification of the compiler since before he can form names with . he must change . to a letter.

11.1 Very Local Variables

Name scoping is described in Chapter 3. Let us review for a moment what happens when a procedure

```
SUMSQ=PROC(X,Y), X*X+Y*Y END;
```

is called, as follows:

```
A=SUMSQ(3,4);
```

The following steps are involved:

(1) save the current values of X and Y

(2) assign 3 to X and 4 to Y

(3) evaluate the expression

(4) restore the original values of X and Y

(5) return the value of the procedure -- given in step 3.

Entering a block of code with local variables is quite similar. Execution of

```
BEGIN(A,B),
   .
   .
END;
```

involves the following steps.

(1) save the current values of A and B

(2) evaluate the expressions in the block

(3) restore the original values of A and B

(4) value of the block is either the value of the last expression or the argument of a return.

Note that in the case of a procedure steps 1, 2 and 4 involve saving and restoring of values; in the case of a block steps 1 and 3. The values are saved in the stack. Each time a procedure or block is entered the stack is used to save these values. Upon exit the stack is popped after values are restored. The code generated to perform the saves and restores takes up both extra time and space. It can be eliminated when blocks and procedures access the stack directly for their local variables and arguments respectively. The compiler procedures and blocks do, in fact, access the stack directly. We shall refer to arguments and variables which exist in the stack as very local. The user may request very local as an option by executing

```
MAKALOCAL(TRUE);
```

for very local arguments and

```
MAKVLOCAL(TRUE);
```

for very local variables.

`MAKALOCAL(NIL);` and `MAKVLOCAL(NIL);`

return the system to its normal state.

A very local variable or argument is known only within its own block or procedure. This means that blocks defined within other blocks do not have access to each other's very local variables, after executing MAKVLOCAL(TRUE); and MAKALOCAL(TRUE); arguments and variables prefaced by $ when defined are treated in the normal way. If they appear without a $ they are considered to be very local. For example:

```
BEGIN ($A,B,$C,D)
```

A and C will be treated as local and B and D are very local.

```
TEST=PROC($A,B,C)
```

A will be treated normally, B and C are very local -- i.e. known only within procedure TEST.

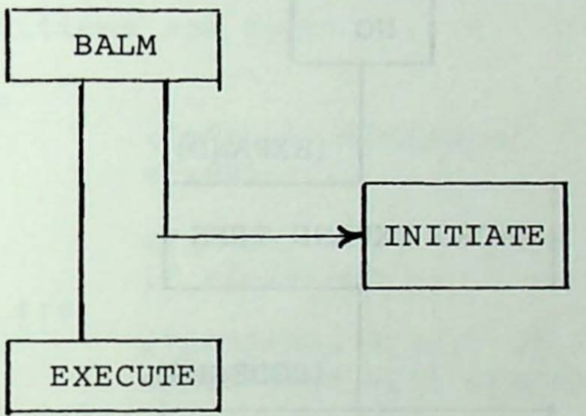
```
MAKVLOCAL(TRUE);
A=10;
BEGIN(A), A=1,
BEGIN(), PRINT(A) END,
PRINT(A)
END;
```

will result in the following prints

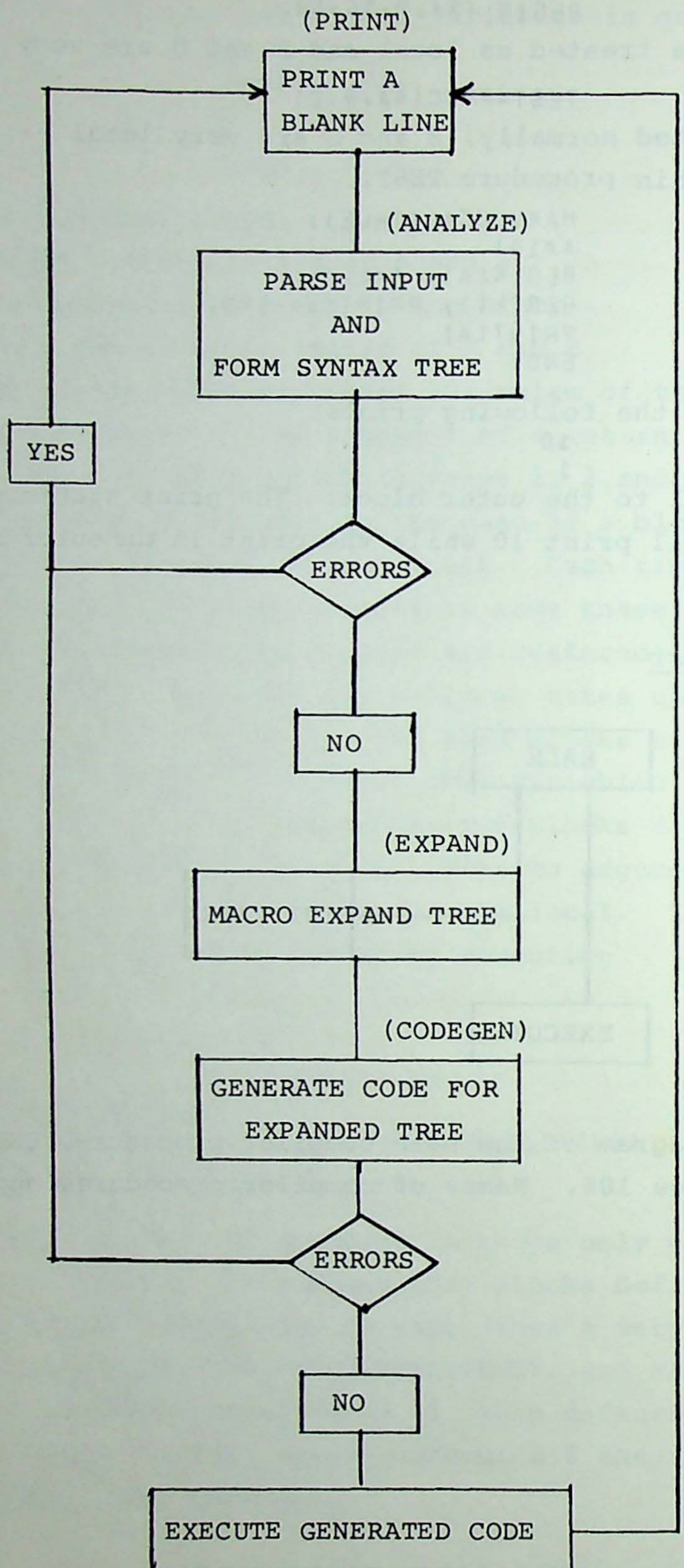
```
10
1
```

A is very local to the outer block. The print statement in the inner block will print 10 while the print in the outer block will result in 1.

11.2 Flow Chart



The flow diagram of the main compiler procedure, EXECUTE, is given on page 106. Names of compiler procedures appear in parentheses.


(PRINT)
PRINT A BLANK LINE
(ANALYZE)
PARSE INPUT AND FORM SYNTAX TREE
YES
ERRORS
NO
(EXPAND)
MACRO EXPAND TREE
(CODEGEN)
GENERATE CODE FOR EXPANDED TREE
ERRORS
NO
EXECUTE GENERATED CODE

APPENDIX A. PARTIAL SYNTAX

We give below a partial definition of the syntax of BALM programs. This is incomplete in the following senses. First, this definition includes programs which are syntactically incorrect. For example, the definition does not prevent the use of expressions which have values of the wrong type, such as as HD 123. This cannot be done in general by syntactic definition, so we have not attempted it at all. Second, there are valid programs which do not conform to the definition. This does not reduce the power of the language very much, and those who wisha more exact definition can consult the more precise definition given elsewhere. Third, the use of blanks in the language is given informally below.

The alternative definitions are given on separate lines, rather than being separated by an alternation mark as is usual. Main definitions are given starting in column 1, while subsidiary definitions are inset.

```
<PROGRAM> ::=
                        <EXPR> ; <PROGRAM>
                        <EXPR> ;
<EXPR> ::=
                        <EXPR2>
                        IF <ELSEXPR>
   <ELSEXPR> ::=
                        <THENEXPR> ELSEIF <ELSEXPR>
                        <THENEXPR> ELSE <EXPR>
   <THENEXPR> ::=
                        <EXPR2> THEN <EXPR2>
<EXPR2> ::=
                        <EXPR3>
                        WHILE <WHILEXPR>
                        FOR <FOREXPR>
                        GO <EXPR3>
                        GOTO <EXPR3>
                        RETURN <EXPR3>
   <WHILEXPR> ::=
                        <EXPR3> REPEAT <EXPR2>
   <FOREXPR> ::=
                        <FORASS> REPEAT <EXPR2>
   <FORASS> ::=
                        <NAME> = <FORLIST>
   <FORLIST> ::=
                        ( <FORPARAMS> )
   <FORPARAMS> ::=
                        <EXPR> , <EXPR>
```

```
                    <EXPR> , <EXPRS>
<EXPRS> ::=
                    <EXPR> , <EXPR>
<EXPR3> ::=
                    <EXPR4>
                    <LHS> = <EXPR4>
<LHS> ::=
                    <NAME>
                    HD <EXPR13>
                    TL <EXPR13>
                    VALUE <EXPR13>
                    <EXPR14> ( <EXPR> )
<EXPR4> ::=
                    <EXPR5>
                    <EXPR5> : <EXPR4>
<EXPR5> ::=
                    <EXPR6>
                    <EXPR6> OR <EXPR5>
<EXPR6> ::=
                    <EXPR7>
                    <EXPR7> AND <EXPR6>
<EXPR7> ::=
                    <EXPR8>
                    <EXPR8> ≡ <EXPR8>
                    <EXPR8> EQ <EXPR8>
                    <EXPR8> NE <EXPR8>
                    <EXPR8> GT <EXPR8>
                    <EXPR8> GE <EXPR8>
                    <EXPR8> LT <EXPR8>
                    <EXPR8> LE <EXPR8>
                    <EXPR8> SIM <EXPR8>
<EXPR8> ::=
                    <EXPR9>
                    <EXPR8> + <EXPR9>
                    <EXPR8> - <EXPR9>
<EXPR9> ::=
                    <EXPR10>
                    <EXPR9> * <EXPR10>
                    <EXPR9> / <EXPR10>
<EXPR10> ::=
                    <EXPR11>
                    - <EXPR11>
<EXPR11> ::=
                    <EXPR12>
                    <EXPR12> ↑ <EXPR11>
<EXPR12> ::=
                    <EXPR13>
                    SIZE <EXPR13>
<EXPR13> ::=
                    <EXPR14>
                    HD <EXPR13>
```

```
                        TL <EXPR13>
                        VALUE <EXPR13>

<EXPR14> ::=
                        <CONST>
                        <NAME>
                        <NOOP> <OP>
                        ( <EXPR> )
                        <EXPR14> ( <EXPRLIST> )
                        <EXPR14> [ <EXPR> ]
                        DO <EXPRLIST> END
                        BEGIN <BLOCKBODY> END
                        PROC <PROCBODY> END
   <EXPRLIST> ::=
                        <EXPR>
                        <EXPR> , <EXPRLIST>
   <BLOCKBODY> ::=
                        <LOCALVARS> , <EXPRLIST>
   <LOCALVARS> ::=
                        ( )
                        ( <VARLIST> )
   <VARLIST> ::=
                        <NAME>
                        <NAME> , <VARLIST>
   <PROCBODY> ::=
                        <ARGLIST> , <EXPR>
   <ARGLIST> ::=
                        ( )
                        ( <VARLIST> )

<NAME> ::=
                        <LETTER> <LETTERORDIGIT>
   <LETTERORDIGIT> ::=
                        <EMPTY>
                        <LETTER> <LETTERORDIGIT>
                        <DIGIT> <LETTERORDIGIT>
<CONST> ::=
                        <INTEGER>
                        <LOGICAL>
                        <STRING>
                        = <ITEM>
<INTEGER> ::=
                        <DIGIT> <DIGITS>
   <DIGITS> ::=
                        <EMPTY>
                        <DIGIT> <DIGITS>
<LOGICAL> ::=
                        TRUE
                        NIL
<STRING> ::=
                        ≠ <CHARACTERS> ≠
```

```
<CHARACTERS> ::=
                    <EMPTY>
                    <CHARACTER> <CHARACTERS>

<ITEM> ::=
                    <INTEGER>
                    <LOGICAL>
                    <STRING>
                    <NAME>
                    <SPECIAL CHARACTER>
                    <VECTOR>
                    <LIST>
<VECTOR> ::=
                    [ <ITEMS> ]
<LIST> ::=
                    ( <ITEMS> )
  <ITEMS> ::=
                    <EMPTY>
                    <ITEM> <ITEMS>
```

The above definition makes no reference to blanks. A blank terminates a <NAME>, an <INTEGER>, a <LOGICAL>, or a <NAME> used as an operator (such as GT). Otherwise, blanks are ignored except in <STRING>s. Note that <OP> is not defined above, but is intended to refer to any operator. Thus the NOOP operator permits operators to be used also as <NAME>s if necessary. <NAME>s and <SPECIAL CHARACTER>s occurring in <CONST>s are used to represent identifiers of the same name.

APPENDIX B. OTHER VERSIONS

360 Operation

The following control cards are necessary to run BALM programs on the 360 at the School of Commerce (NYU).

```
//NAME          JOB
//      EXEC    BALM4
//BALM4.SYSIN   DD  *
    BALM PROGRAM
/*
```

If a user wishes to create a saved file he needs to include a DD card for that file. If his BALM program has a SAVEALL(N) then

```
//BALM4.FTONF001  DD   ETC.
```

must be included. A SAVEALL file requires about 20 tracks and RECFM=VS.

APPENDIX C. EXAMPLES

Example 1

When a function is used frequently with a small number of arguments, it can be more efficient to save previous results in a list and look in this list to see if the result has been calculated for this argument already. If the function is FN, instead of FN(X) we can write MEMO(FN,FNL,X), where FNL is the list used to keep the values calculated for FN.

MEMO can be written:

```
MEMO = PROC(FN,FNL,X),
    BEGIN(FNLPR,SAVE,FNLB,FNLPR2),
        FNLB=FNL, FNL = TL FNL, FNLPR=FNL,
NEXT,   IF HD HD FNL EQ X THEN GOTO FOUND,
        FNLPR2=FNLPR,
        FNLPR = FNL, FNL= TL FNL,
        IF NULL(FNL) THEN GOTO NOTF ELSE GOTO NEXT,
FOUND,  IF FNLPR EQ FNL THEN RETURN TL HD FNL,
        SAVE = HD FNL, HD FNL = HD FNLPR,
        HD FNLPR = SAVE, RETURN TL SAVE,
NOTF,   TL FNLPR2 = NIL, SAVE = FN(X),
        TL FNLB = (X:SAVE):TL FNLB, RETURN SAVE
END END;
```

It is assumed that FNL has been initialized to a list of the form:

```
(Z (X1,Y1)(X2,Y2) .... (XN,YN))
```

where Y1=FN(X1) , and N is the number of arguments that it has been decided to save. The HD of the list, Z, is not used by this routine, but may be useful for other purposes. Each argument, written as X above, is first looked up in this list. If it is found, the corresponding pair is moved one place nearer the beginning of the list and the stored value is returned. If it is not found the last pair in thelist is deleted and a new pair corresponding to the new argument is added to the beginning.

An alternative way of writing this function uses a vector to store the argument-value pairs. We will assume that this is of the form:

```
[(X1,Y1) (X2,Y2) ... (XN,YN)]
```

The function can then be written as follows.

```
MEMO = PROC(FN,FNL,X),
   BEGIN(L,I),
      L=LENGTH(FNL), I=1,
NEXT,  IF I GT L THEN GOTO NOTF
         ELSEIF X EQ HD FNL[I] THEN GOTO FOUND,
      I=I+1, GOTO NEXT,
FOUND, IF I EQ 1 THEN RETURN TL FNL[I],
      L=FNL[I], FNL[I] = FNL[I-1],
      FNL[I-1] = L, RETURN TL L,
NOTF,  FOR I=(L-1,1,-1) REPEAT FNL[I+1] = FNL[I],
      L = FN(X), FNL[1] = X:L, RETURN L
   END END;
```

Note that this probably executes a little more slowly than the other version particularly when the argument is usually not found, because the operation of moving down all the elements of FNL is much slower. However, by a slight modification to this version, an extra entry in the vector, say the first, could be used to keep the index of the top element, and the vector could be treated as circular, with the Lth element being followed logically by the second. This would eliminate the time-consuming move operation.

There are of course several alternative algorithms which may be equally effective. For example, it may be more appropriate to store any new value at the end of the list rather than the beginning, or possibly in the middle. This will be preferable when the arguments fall into two types, a few of which occur frequently, and many of which are rarely repeated.

Example 2

We give below a generalized game-playing routine BEST, using an alpha-beta minimax search, with the appropriate routines POSSMOVES, NEWPOSN, and SCORE added. It will play tic-tac-toe, checkers, chess, or Go with varying degrees of success.

Arguments to BEST are: POSN, the position from which the name must be made; DEPTH, the number of moves the routine should look ahead; WISB and HYB, the range between which scores are acceptable, with HISB being the minimum and HYB the maximum, large scores being good. For tic-tac-toe, for instance, POSN is of the form:

```
[[X--][-0-][---]X]
```

the extra X indicating it is X to move.

The full deck to play a complete game of tic-tac-toe against itself is constructed as follows:

```
A123456,CM60000,      HARRISON
CIMGET(BALM4,BLM4SVD)
BALM4.
   --  GREEN END OF RECORD CARD  --
COMMENT (TEST PROGRAM - TIC-TAC-TOE PLAYER)

 BEST=PROC(POSN,DEPTH,HISB,MYB),
       BEGIN(ML,DM1,BESTSC,BESTM,TRY),
       IF DEPTH≡0 THEN RETURN(-SCORE(POSN):NIL:NIL),
       ML=POSSMOVES(POSN),
       DM1=DEPTH-1,BESTSC=MYB,BESTM=NIL,
 NXT,  IF ¬ML THEN RETURN(-BESTSC:BESTM:NIL),
       TRY=BEST(NEWPOSN(POSN,HD ML),DM1,-BESTSC,-HISB),
       IF HD TRY GT BESTSC THEN
          DO BESTSC=HD TRY,BESTM=HD ML END,
       IF ¬ BESTSC LT HISB THEN RETURN(-BESTSC:BESTM:NIL),
       ML=TL ML, GO NXT
       END END;

SCORE=PROC(P),
        BEGIN(ROWSC,M,H,I,J),
   ROWSC=PROC(INIT,STEP),
        BEGIN(NM,NH,IJ,S),
        INIT(), NM=0, NH=0,
        FOR IJ=(1,3) REPEAT
           DO IF (S=P[I][J]) EQ M THEN NM=NM+1
                 ELSEIF S EQ H THEN NH=NH+1,
           STEP() END,
        RETURN(IF(NM*NH) EQ 0 THEN NM*NM-NH*NH ELSE 0)
        END END,
      M=P[4],H=IF M≡≥X THEN ≥0 ELSE ≥X,
      RETURN
      (ROWSC(PROC(), I=J=1 END, PROC(),J=J+1 END) +
       ROWSC(PROC(),I=1+(J=1) END, PROC(), J=J+1 END) +
```

```
     ROWSC(PROC(),I=2+(J=1) END, PROC(),J=J+1 END) +
     ROWSC(PROC(),I=J=1 END, PROC(), I=I+1 END) +
     ROWSC(PROC(),J=1+(I=1) END, PROC(),I=I+1 END) +
     ROWSC(PROC(),J=2+(I=1) END, PROC(),I=I+1 END) +
     ROWSC(PROC(),I=J=1 END, PROC(),I=J=J+1 END) +
     ROWSC(PROC(),I=2+(J=1) END, PROC(),DO I=I-1,J=J+1 END END) )
    END END;

POSSMOVES=PROC(P),
     BEGIN(I,J,M),
     M=NIL,
     FOR I=(1,3) REPEAT
        FOR J=(1,3) REPEAT
            IF P[I][J]≡≥- THEN M=(I:J:NIL):M,
     RETURN(M)
   END END;

NEWPOSN=PROC(P,M),
     BEGIN(N),
     N=COPY(P),
     N[(HD  M)][(HD  TL  M)]=P[4],
     N[4]=IF P[4]≡≥X THEN ≥O ELSE ≥X,
     RETURN(N)
   END END;

PLAY=PROC(),
     BEGIN(POSN,BM),
     POSN=≥[[- - -][- - -][- - -] X],
  NEXT,PRINT(POSN),
     BM=HD TL BEST(POSN,1,9999,-9999),
     IF BM≡NIL THEN RETURN(POSN),
     POSN=NEWPOSN(POSN,BM),
     GO NEXT
   END END;

PLAY();
STOP;
  --  PINK END OF FILE CARD  --
```

This uses a look-ahead of only one level, prints out each position, and terminates when all squares are occupied.

```
1.1 FOR I=(1,100) REPEAT DO
        R=I-(N=I/8)*8,
        PRINT(10*N+R)
    END;

1.2 FOR Z=(1,100) REPEAT DO ZZ=Z*Z,
        FOR Y=(1,Z-1) REPEAT DO YY=Y*Y,
          FOR X=(Z-Y,Y) REPEAT
             IF X*X+YY EQ ZZ THEN PRINT(X,Y,Z)
        END
    END;

2.1 BEGIN(V,L,I,N,BLANK),
    BLANK=VFROMS(≠ ≠)[1],
    V=VFROMS(S), L=SIZE V,
    I=1,  N=L/2,
    WHILE I LE N REPEAT
       IF V[I] EQ V[L-I+1] THEN I=I+1
       ELSEIF V[I] EQ BLANK THEN DO I=I+1, L=L+1 END
       ELSEIF V[L-I+1] EQ BLANK THEN L=L-1
       ELSE RETURN PRINT(S,≠IS NOT A PALINDROME≠),
    PRINT(S,≠IS A PALINDROME≠)
    END;


3.1 COUNT=PROC(L),
    IF PAIRQ(L) THEN COUNT(HD L)+COUNT(TL L)
    ELSEIF INTQ(L) THEN 1
    ELSE 0
    END;

3.2 EQLVQ=PROC(L,V),
    BEGIN(I),
    FOR I=(1,SIZE V) REPEAT
       IF NOT PAIRQ(L) THEN RETURN NIL
       ELSEIF HD L NE V[I] THEN RETURN NIL
       ELSE L=TL L,
    RETURN TRUE
    END END;

3.3 EQSETQ=PROC(X,Y),
    BEGIN(I),
    IF NOT PAIRQ(X) AND NOT PAIRQ(Y) THEN RETURN X EQ Y
    ELSEIF NOT PAIRQ(X) OR NOT PAIRQ(Y) THEN RETURN NIL
    ELSEIF LENGTH(X) NE LENGTH(Y) THEN RETURN NIL,
    FOR I=(1,LENGTH(X)) REPEAT
       IF NOT ELEM(HD X,Y) THEN RETURN NIL
       ELSE X=TL X,
    RETURN TRUE
    END END;

    ELEM=PROC(X,L),
    BEGIN(),
    WHILE L REPEAT
       IF EQSETQ(X,HD L) THEN RETURN TRUE
       ELSE L=TL L,
    RETURN NIL
    END END;
```

APPENDIX E. UTILITY PROCEDURES

The predefined procedures in the current system, in alphabetical order, are as follows.

ADDON(LST,X)

S is an arbitrary item and LST points to the last element in a list. The last element is replaced by X:NIL, which is returned as the value of ADDON.

Example: L=LAST= =A:NIL, LAST=ADDON(LAST,=B)

L contains (A B) and LAST is =B:NIL

BRACKET(OP,PREC,PROCNAM)

defines OP as a bracket operator with precedence PRED and associated procedure name PROCNAM.

BREAKUP(PATRN,ARG)

BREAKUP performs a multiple assignment of the names in PATRN to structures in ARG. For example

BREAKUP(≥(A [B C (D)]),≥(<STR> [57 (X) (Y)])

will cause the same assignments as A=<STR>;B=57;C=≥(X);D=≥Y;

In general, the names in PATRN are assigned values which are the elements appearing in corresponding positions in ARG. If such an assignment cannot be performed due to a difference in structure between PATRN and ARG then NIL is returned, otherwise TRUE. In the case of an assignment failure, only those names encountered before the structural discrepancy will be assigned values. For example,

BREAKUP(≥(A (B C)),≥(X (3 4)))

will return TRUE and result in the following assignments.

A=≥X;B=3;C=4;

BREAKUP(≥(A(B C) D),≥(X (3 4 5))) will return NIL and cause the same assignments as the previous example. D will remain unchanged. Constants appearing in PATRN must exactly match their corresponding elements in ARG for a successful match. BREAKUP(≥(A B C <STR> 7),≥(1 2 3 <STR> 7)) will return TRUE while BREAKUP(≥(A B C <STR> 7),≥(4 5 6 <STR> 8)) will return NIL. A constant name or structure can be represented by PATRN by preceding it with a ≥ .

Example:

```
BREAKUP(≥(A ≥B B ≥[C D]),≥(1 B 5 [C D]))
```

will return TRUE and will perform the assignments A=1;B=5;

```
BREAKUP(≥(A ≥B C),≥(1 (B) <XYZ>))
```

will return NIL and will perform the assignment A=1;

CHGCHAR(CHAR,N)

CHAR is a vector each item of which represents a character. It is used by procedure LXSCAN to determine BALM items. If the value of an item of CHAR is 0 to 9 the character is than number; 10 the character is a letter; 11 a special character. CHGCHAR changes the value of the CHAR item representing CHR to N. For example,

```
CHGCHAR(=.,10);
```

will make . scan as one of the letters and thus make accessible all the compiler procedures.

CONSTRUCT(ARG)

returns the structure ARG with all names replaced by their values. Example:

```
BREAKUP(≥([A B] C 53),≥([(X Y) Z] R 53));
PRINT(CONSTRUCT(≥[(A B) C 65]));
```

will print [((X Y) Z) R 65]

COMPILE(LST)

LST is a list containing a BALM procedure without the terminating semicolon. COMPILE returns compiled code for that statement.

COPY(ARG)

returns a copy of its argument.

DUMMY(ARG)

returns ARG.

ERROR(TYPE)

called automatically when errors occur. TYPE is set as follows.

1 - no more space
2 - invalid type or pointer found by garbage collector
3 - attempt to execute something which is not a procedure
4 - attempt to execute an undefined op code
5 - invalid argument to I/O instruction, i.e., not a string or integer.
6 - attempt to goto a nonlabel.

EQUAL(ARG1,ARG2)

returns TRUE if ARG1 is the same as (or a copy of) ARG2.

EXECUTE(INFILE,OUTFILE)

EXECUTE translates and executes BALM statements from the file INFILE until a STOP statement is reached. EXECUTE will then return NIL. Translator error messages are written on file OUTFILE.

EXPAND(LST)

returns the result of a macro expansion on list LST. LST is assumed to be in the same form as the output from the pre-edence analyzer.

GENSYM()

returns a unique identifier each time called.

GETPROP(ID,P)

refer to listing of the compiler supplement Appendix F.

IFROMID(ID)

returns an integer which is the symbol table entry number of ID.

INFIX(OP,LPREC,RPREC,PROCNAM)

defines infix operator OP with left precedence LPREC and right precedence RPREC. PROCNAM is the name of the corresponding procedure.

LENGTH(ARG)

returns the length of a string in characters or the number of top level elements in a list or vector.

LFROMV(VECT)

returns a list of the elements of VECT.

LOOKUP(ARG,LST)

refer to listing of the BALM4 compiler, Appendix F. Assumes that LST is a list whose members are lists of two elements. LOOKUP searches LST to see if ARG is the first top level element of a member. If it is not then NIL is returned. If it is then the second top level element of the member is returned.

LOOKUP(ID1,ID2)

If the property field of ID1 is a list it is assumed to be a list whose members are themselves lists with two top level members. LOOKUP examines each member of the list to see if ID2 is its first top level element. If so then the second top level element of the member is returned, otherwise NIL.

MACRO(NAM,PROCED)

defines NAM as a macro with associated procedure PROCED.

MAKPROPS(ID1,ID2)

The value of ID2 is assumed to be a list whose members are lists of two elements. The first element of each member must be an identifier. MAKPROPS examines each member of the list and sets the property field of the first element to a list of ID1 and the second element of the member. When the entire list has been searched the value of ID2 is set to ID1. (Refer to procedure LOOKUP and also to the supplementary listing).

MAKALOCAL(COND)

If COND is TRUE the compiler will make all procedure arguments not preceded by a $ into verylocal variables i.e. known only within that procedure If COND is NIL all arguments will be local and will be known to all procedures called within the scope of the procedures. COND=NIL is the current default mode.

MAKVLOCAL(COND)

If COND is TRUE the compiler will make the variables listed in begin blocks and not preceded by $ into very local variables, i.e. known only within that block. If COND is NIL all variables listed within begin end blocks will be compiled as local regardless of whether they are preceded by $ or not. They will be accessible to all blocks executed within the scope of the defining block. COND = NIL is the current default mode.

MAPX(ARG,PROCED)

MAPX applies the procedure PROCED to each of the top level elements of ARG, returning a list or vector of the results. If ARG is a list, then a list is returned; if ARG is a vector, then a vector is returned. ARG must not be a string.

MEMBER(ARG1,ARG2)

if ARG1 is a top level element of ARG2 (a list or vector) then TRUE is returned, otherwise NIL.

ORDINAL(ARG,LST)

returns I when ARG is the Ith top level member of LST. NIL is returned otherwise. Note ARG must be on the list; if LST contains a copy of ARG, NIL will be returned.

PROCTRACE(LST)

LST is assumed to be the result of a STKTRACE(I) operation. PROCTRACE produces a formatted procedure trace on the output file.

PTRACE(ID)

prints the value of the arguments of procedure ID each time it is called and prints the value of each RETURN.

REMINFIX(OP)

removes the most recently added occurrence of OP from the infix list.

REMMACRO(NAM)

removes the most recently added occurrence of NAM from the MACROLIST. Note that when the user creates a macro using MEANS, NAM must be PROCNAME and not OP as described in procedures INFIX and UNARY.

REMUNARY(OP)

removes the most recently added occurrence of OP from the unary list.

RESTAT(V)

V is the output of procedure SAVSTAT. The states of OPLIST MACROLIST, INFIXLIST, UNARYLIST and CODGENLIST are restored to the same state as when SAVSTAT was executed.

SAVEBALM(S)

opens a file named S, rewinds the file, performs a garbage collection, executes a SAVEALL on the file and closes the file.

SAVSTAT()

output is a vector which can be used as input to RESTAT to restore the state of the compiler.

SETPROPY(NAME,PROP,VAL)

gives the property PROP with value VAL to the name NAME. The previous value of this property (if present) is pushed down, and not lost. The name is returned.

SUBST(NEW,OLD,LST)

returns the list LST with all occurrences of OLD replaced by NEW. For example

```
X=SUBST(≥[X Y],≥(A(B C)),≥(X Y ((A (B C))) Z));
PRINT(X);
```

will print

```
(X Y ([X Y]) Z)
```

TRANSLATE(LST)

LST is a list containing a BALM statement without the terminating semicolon. TRANSLATE will return the statement translated into BALM internal form, suitable for compilation.

UNARY(OP,PREC,PROCNAM)

defines OP as a unary operator with precedence PREC and associated procedure name PROCNAM.

VFROML(LST)

returns a vector containing elements of list LST.

APPENDIX F

The following is in two parts. The first is a listing of the BALM4 compiler preceded by a bootstrap program which enables the compiler to produce a punched object deck. The second is additional BALM4 programs which are used to create a saved file of the BALM4 system.

```
CHGCHAR(=.,10);
MAKALOCA(TRUE); MAKVLOCA(TRUE);
BOOTSTRAP= PROC(), BEGIN(LS,NAM,PR).
MOR,    PRINT.(),
        ERCOUNT.=0, TERMFLA.=NIL.
        LS=ANALYSE.(IEOS.),
        IF ¬ZR ERCOUNT. THEN GOTO ERR,
        IF LS EQ =STOP THEN RETURN NIL.
        LS=EXPAND.(LS),
        IF ¬ZR ERCOUNT. THEN GOTO ERR,
        NAM=HD TL LS,  PR=HD TL TL LS,
        BSCODEGEN(NAM,PR),
        GOTO MOR,
ERR,    ERMSG.(LIST(ERCOUNT.,=SYNTAX,=ERRORS)),
        GOTO MOR
        END END;
BSCODEGEN=PROC(NAM,X),
        BEGIN($ERRCNT,$LIST2,$END2,LIST3,
              $GLOBL,$LBLVALS,$NBYTE,$LBLNO,
              $ARGS,$VARS,$EXPX,SARGS,$LBLIST,
              I,CODEV),
        ERRCNT=0, GLOBL=NIL, NBYTE=3, LBLNO=1,
        LBLVALS=NIL,
        VARS=NIL, LBLIST=NIL,
        IF CGCHECK.(=PROC:(COMMA.:=ARGS:=EXPX:NIL):END2:NIL,X)
           AND END2 EQ =END THEN NIL
        ELSE DO ERMSG.(=PROC:=ERROR:NIL), RETURN NIL END,
        LIST2=END2=(0:NIL), END2=ADDON.(END2,GREFS.(NAM)),
        ARGS=REMCOM.(ARGS),
        IF HD ARGS EQ DOLLAR. THEN ARGS=ARGS:NIL,
        SARGS=ARGS, ARGS=EXCHANG.(SARGS,1),
        COMP.(EXPX),
        EXCHANG.(SARGS,1),
        ASS.(KRTPROC.),
        LIST3=LIST2,
        WHILE LIST2 REPEAT DO
           HD LIST2=SUBLBLS.(HD LIST2), LIST2=TL LIST2 END,
        PRINT.(=GLOBAL,=VARS,GLOBL),
        IF ZR ERRCNT THEN WRITE(LIST3,BINFILE)
           ELSE ERMSG.(LIST(ERRCNT,=COMP,=ERRORS)),
        RETURN NIL
        END END;
        BINFILE=MAKFILE(≠PUNCH≠,72); REWIND(BINFILE[1]);
BOOTSTRAPCOMPILE();
```

```
COMMENT
≠$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
                PROGRAM TO BE BOOTSTRAPPED FOLLOWS
                           UP TO STOP;
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$≠

COMMENT
(********************************************************************
                           MAIN PROGRAM
********************************************************************)
BALM =    PROC(),
     BEGIN(),
        INITIAT.(),
        EXECUTE(INPUT,OUTPUT),
        STOP()
     END
  END;
EXECUTE=  PROC($INPUT,$OUTPUT),
     BEGIN(ST,LS,$CURCOM.,$PREVCOM.,$ERCOUNT.,$TERMFLA.),
  MOR,  PRINT.(),
        ERCOUNT.=0, TERMFLA.=NIL,
        ST=ANALYSE.(IEOS.),
        IF TALKATIV GE 2 THEN DO
           PRINT.(=SYNTAX,=TREE), PRINT.(ST)
        END,
        IF ¬ZR ERCOUNT. THEN GOTO ERR,
        IF ST EQ =RESUME THEN RETURN NIL,
        IF ST EQ =STOP THEN RETURN NIL,
        MACSYMB.=VECTOR(STAR.,STAR.,DNUM.[1],DNUM.[1],DNUM.[1]),
        LS=EXPAND.(ST),
        IF TALKATIV GE 3 THEN DO
           PRINT. (=EXPANDED,=TREE), PRINT.(LS)
        END,
        IF ¬ZR ERCOUNT. THEN GOTO ERR,
        IF PAIRQ(LS) AND HD LS EQ EQSIGN. AND PAIRQ(HD TL TL LS) AND
        HD HD TL TL LS EQ =PROC AND
           IDQ (HD TL LS) THEN DO
              CODEGEN.(HD TL LS,HD TL TL LS),
              GOTO MOR
           END,
           LS = LIST(=PROC,LIST(COMMA.,MTY.,LS),=END),

           CODEGEN.(=CURCOM.,LS),
           IF ¬CODEQ(CURCOM.) THEN GOTO MOR,
           CURCOM.(),
           GOTO MOR,
  ERR,     PRINT.(LIST(ERCOUNT.,=ERRORS)),
           PREVCOM.=ST, GO MOR
        END
     END;
COMMENT
≠********************************************************************
              UTILITY ROUTINES
```

```
*******************************************************************************
VFROML.=  PROC(L),
    BEGIN(N,V,I),
       N=0, V=L,
       WHILE V REPEAT DO
          N=N+1, V=TL V
       END, V=MAKVECTO(N),
       FOR I=(1,N) REPEAT DO
          V[I]=HD L, L=TL L
       END,
       RETURN V
    END
  END;
ADDON.=   PROC(X,Y),
    TL X=Y:NIL
  END;
LOOKUP.=  PROC(X,L),
    BEGIN(P),

       MOR, IF IDQ(L) AND IDQ(X) THEN DO
          P=L, L=PROPL(X), X=P
       END,
       IF ¬PAIRQ(L) THEN RETURN NIL,
       IF X EQ HD HD L THEN RETURN HD TL HD L,
       L=TL L, GOTO MOR
    END
  END;
ORDINAL.= PROC(A,L),
    ORD1.(A,L,1)
  END;
ORD1.=    PROC(A,L,I),

    IF ¬PAIRQ(L) THEN NIL ELSEIF A EQ HD L THEN I
    ELSE ORD1.(A,TL L,I+1)
  END;
LENGTH.=  PROC(X),
    IF PAIRQ(X) THEN 1+LENGTH.(TL X)
    ELSEIF VECTQ(X) OR STRQ(X) OR CODEQ(X) THEN SIZE X
    ELSE 0
  END;
IFROMID.= PROC(X),
    X+0
  END;
GENSYM.=  PROC(),
    BEGIN(I,J),
       FOR I=(5,2,-1) REPEAT
       IF (J=CHAR.[GENSYMB.[I]]) LT 9
       THEN DO
          GENSYMB.[I]=DNUM.[J+2],
          RETURN IDFROMS(SFROMV(GENSYMB.))
       END
       ELSE GENSYMB.[I]=DNUM.[1]
    END
```

```
  END;
MAPX.=     PROC(X,P),
      IF PAIRQ(X) THEN MAPL.(X,P)
      ELSEIF VECTQ(X) THEN MAPV.(X,P)
      ELSE X
   END;
MAPL.=     PROC(L,P),
      BEGIN(B,E),
         B=E=NIL:NIL,
  MOR,   IF ¬PAIRQ(L) THEN RETURN TL B,
         E=ADDON.(E,P(HD L)), L=TL L, GOTO MOR
      END
   END;
MAPV.=     PROC(V,P),
      BEGIN(N,VV,I),
         N=SIZE V, VV=MAKVECTO(N),
         FOR I=(1,N) REPEAT VV[I]=P(V[I]), RETURN VV
      END
   END;
MEMBER.=   PROC(X,L),
      BEGIN(),
  MOR,   IF PAIRQ(L) THEN
         (IF X EQ HD L THEN RETURN TRUE
         ELSE DO
            L=TL L, GOTO MOR
         END)
         ELSE RETURN NIL
      END
   END;
LFROMV.=   PROC(V),
      BEGIN(L,N,I),
         N=SIZE V, L=NIL,
         FOR I=(N,1,-1) REPEAT L=V[I]:L,
         RETURN L
      END
   END;
COMMENT
%**************************************************************************
               INITIATION ROUTINES
**************************************************************************%
INITIAT.= PROC(),
      BEGIN(),
         INITIO.(),
         INITUNA.(),
         INITINF.(),
         INITEXP.(),
         INITCOD.(),
         INITOPL.(),
         INITMIS.(),
         RETURN NIL
      END
   END;
INITIO.=  PROC(),
```

```
*****     FOR MACHINE INDEPENDENCE THE CHARACTER SET IS READ
*****     FROM CARDS AND CONSTANTS ARE SET TO THE CHARACTERS USED BY
*****     COMPILER PROCEDURES
*****     CHARACTERS START IN COLUMN 2 OF THE CARDS
*****     THE FOLLOWING 3 CARDS ARE READ BY INITIO
* ,*-≠()[]¬$,≡:*Ξ/+≥;^>
*0123456789
*ABCDEFGHIJKLMNOPQRSTUVWXYZ.
*****
      BEGIN(LIN,L,I),
        LIN=SUBV(VFROMS(RDLINE(1)),2,22),
        BLANK,=LIN[1], PERIOD,=LIN[2], STAR,=LIN[3],
        MSIGN,=LIN[4], STRQU,=LIN[5], LPAR,=LIN[6],
        RPAR,=LIN[7], LBR,=LIN[8], RBR,=LIN[9],
        SLASHX,=LIN[17],
        SEMIC,=LIN[20],
        PROMPT,=LIN[22],
        LPVAR,=IDFROMS(SFROMV(VECTOR(LPAR,))),
        RPVAR,=IDFROMS(SFROMV(VECTOR(RPAR,))),
        LBVAR,=IDFROMS(SFROMV(VECTOR(LBR,))),
        RBVAR,=IDFROMS(SFROMV(VECTOR(RBR,))),
        PERVAR,=IDFROMS(SFROMV(VECTOR(PERIOD,))),
        STARX, =IDFROMS(SFROMV(VECTOR(STAR,))),
        NOTSIGN,=IDFROMS(SFROMV(VECTOR(LIN[10]))),
        MINUS, =IDFROMS(SFROMV(VECTOR(MSIGN,))),
        DOLLAR, =IDFROMS(SFROMV(VECTOR(LIN[11]))),
        COMMA, =IDFROMS(SFROMV(VECTOR(LIN[12]))),
        EQSIGN, =IDFROMS(SFROMV(VECTOR(LIN[13]))),
        COLON, =IDFROMS(SFROMV(VECTOR(LIN[14]))),
        PLUS, =IDFROMS(SFROMV(VECTOR(LIN[15]))),
        EQUIV, =IDFROMS(SFROMV(VECTOR(LIN[16]))),
        SLASH, =IDFROMS(SFROMV(VECTOR(LIN[17]))),
        EXPON, =IDFROMS(SFROMV(VECTOR(LIN[18]))),
        QSIGN, =IDFROMS(SFROMV(VECTOR(LIN[19]))),
        IEOS, =IDFROMS(SFROMV(VECTOR(LIN[20]))),
        COMCHAR,=IDFROMS(SFROMV(VECTOR(LIN[21]))),
        CHAR,=MAKVECTO(128),
        FOR I=(1,128) REPEAT CHAR,[I]=11,
        LIN=SUBV(VFROMS(RDLINE(1)),2,10),
        DNUM,=MAKVECTO(10),
        FOR I=(1,10) REPEAT DO
           CHAR,[LIN[I]]=I-1, DNUM,[I]=LIN[I]
        END,
        SEVEN,=DNUM,[8],
        LIN=VFROMS(RDLINE(1)), LIN=SUBV(LIN,2,SIZE(LIN)-1),
        LETTB,=LIN[2],
        IDTRUE,=IDFROMS(SFROMV(VECTOR(LIN[20],LIN[18],LIN[21],LIN[5]
        ))),
        IDNIL,=IDFROMS(SFROMV(VECTOR(LIN[14], LIN[9], LIN[12] ))),
        L=SIZE LIN,
        FOR I=(1,L) REPEAT IF LIN[I] NE BLANK, THEN CHAR,[LIN[I]]=10,
        INPUT=MAKFILE(1,72),
        OUTPUT=MAKFILE(2,72),
```

```
            BLANKLI,=MAKVECTO(72),
            FOR I=(1,72) REPEAT BLANKLI,[I]=BLANK,,
            RETURN NIL
        END
      END;
  INITUNA,= PROC(),
        BEGIN(),
            UNARYLI,=LIST(
                LIST(=DO,LIST(=BRCKT,=DO,100,100)),
                LIST(=BEGIN,LIST(=BRCKT,=BEGIN,100,100)),
                LIST(=PROC,LIST(=BRCKT,=PROC,100,100)),
                LIST(≥IF,LIST(≥UNARY,≥IF,200,200)),
                LIST(≥RETURN,LIST(≥UNARY,≥RETURN,500,500)),
                LIST(≥WHILE,LIST(≥UNARY,≥WHILE,500,500)),
                LIST(=FOR, LIST(=UNARY,=FOR,500,500)),
                LIST(≥GOTO,LIST(≥UNARY,≥GO,500,500)),
                LIST(≥GO,LIST(≥UNARY,≥GO,500,500)),
                LIST(NOTSIGN,,LIST(=UNARY,=NILQ,1200,1200)),
                LIST(=NOT,LIST(=UNARY,=NILQ,1200,1200)),
                LIST(≥NULL,LIST(≥UNARY,≥NILQ,1200,1200)),
                LIST(=PL,LIST(=UNARY,=PL,1200,1200)),
                LIST(=ZR,LIST(=UNARY,=ZR,1200,1200)),
                LIST(MINUS,,LIST(=UNARY,INEG,1700,1700)),
                LIST(=SIZE,LIST(=UNARY,=SIZE,1900,1900)),
                LIST(=VALUE,LIST(=UNARY,=VALUE,1900,1900)),
                LIST(DOLLAR,,LIST(=UNARY,DOLLAR,,1900,1900)),
                LIST(≥TL,LIST(≥UNARY,≥TL ,2000,2000)),
                LIST(≥HD,LIST(≥UNARY,≥HD ,2000,2000))
                ),
                RETURN NIL
            END
        END;
  INITINF,= PROC(),
        BEGIN(),
            INFIXLI,=LIST(
            LIST(IEOS,,LIST(=INFIX,IEOS,,0,-1)),
            LIST(≥END,LIST(≥INFIX,≥END,0,0)),
            LIST(COMMA,,LIST(=INFIX,COMMA,,100,100)),
            LIST(≥ELSEIF,LIST(≥INFIX,≥ELSEIF,300,300)),
            LIST(≥ELSE,LIST(≥INFIX,≥ELSE,300,300)),
            LIST(≥THEN,LIST(≥INFIX,≥THEN,400,400)),
            LIST(≥REPEAT,LIST(≥INFIX,≥REPEAT,600,600)),
            LIST(EQSIGN,,LIST(=INFIX,EQSIGN,,700,701)),
            LIST(COLON,,LIST(=INFIX,COLON,,800,800)),
            LIST(≥OR,LIST(≥INFIX,≥OR,1000,1001)),
            LIST(≥AND,LIST(≥INFIX,≥AND,1100,1101)),
            LIST(=NE,LIST(=INFIX,=NE,1200,1200)),
            LIST(=LT,LIST(=INFIX,=LT,1200,1200)),
            LIST(=GE,LIST(=INFIX,=GE,1200,1200)),
            LIST(=GT,LIST(=INFIX,=GT,1200,1200)),
            LIST(=LE,LIST(=INFIX,=LE,1200,1200)),
            LIST(=SIM,LIST(=INFIX,=SIM,1200,1200)),
            LIST(MINUS,,LIST(=INFIX,MINUS,,1501,1500)),
```

```
        LIST(PLUS,,LIST(=INFIX,PLUS,,1501,1500)),
        LIST(STARX,,LIST(=INFIX,STARX,,1601,1600)),
        LIST(EQUIV,,LIST(=INFIX,=EQ,1400,1400)),
        LIST(SLASH,,LIST(=INFIX,SLASH,,1601,1600)),
        LIST(EXPON,,LIST(=INFIX,EXPON,,1800,1800)),
        LIST(=EQ,LIST(=INFIX,=EQ,1400,1400)) ),
    RETURN NIL
    END
  END;
INITEXP,= PROC(),
    BEGIN(),
        MACROLI,=LIST(LIST(=QUOTE,DUMMY,)),
        RETURN NIL
    END
  END;
INITCOD,= PROC(),
    BEGIN(),
        CODGENL, = LIST(
        LIST(=PROC,GLAMBDA),LIST(=BEGIN,GPROG),
        LIST(=RETURN,GRETURN),LIST(=DO,GPROGN),
        LIST(=GO,GGO), LIST(=IF,GCOND),
        LIST(=AND,GAND), LIST(=OR,GOR),
        LIST(=STRING,GSTRING),
        LIST(=QUOTE,GQUOTE), LIST(EQSIGN,,GSETQ),
        LIST(=WHILE,GWHILE), LIST(=FOR,GFOR),
        LIST(=LIST,GLIST), LIST(=VECTOR,GVECTOR)
        ),
        KCALL,=27B, KRTPROC,=115B,
        KNVARS,=36B, KRETPRO,=131B, KSETSTK,=137B, KPOP,=35B,
        KARG,=33B, KVAR,=31B, KGLOB,=5B,
        KASTORE,=34B, KVSTORE,=32B, KGSTORE,=6B,
        KNUM1,=26B, KNUM2,=4B, KNUM3,=37B,
        KNIL,=136B, KTRUE,=135B,
        KJMP,=3B, KJMPT,=1B, KJMPF,=2B,
        KLBL,=11B, KJMPI,=52B,
        KLIST,=10B, KVECTOR,=12B,
        KTLOOP,=14B, KSTEPLO,=53B,
        KINEG,=76B, KMAKVAR,=41B,
        KSTRING,=13B,
        KSETSX,=134B,
        RETURN NIL
    END
  END;
INITOPL,= PROC(),
    BEGIN(),
        OPLIST, = LIST(
        LIST(COLON,,61B), LIST(=HD,123B), LIST(=TL,124B),
        LIST(=RPLACA,121B), LIST(=RPLACD,122B), LIST(=LIST,10B),
        LIST(=MAKVECTO,140B), LIST(=VECTOR,12B),
        LIST(=INDEX,117B), LIST(=SETINDEX,120B),
        LIST(=SUBV,163B), LIST(=SETSUBV,164B),
        LIST(=CONCATV,165B),
        LIST(=SUB,163B), LIST(=SETSUB,164B),LIST(=CONCAT,165B),
```

```
        LIST(=SHIFT,70B), LIST(=LAND,125B),
        LIST(=LOR,126B),LIST(=COMPL,127B),
        LIST(=SIZE,114B), LIST(=STRING,13B), LIST(=XOR,67B),
        LIST(=APPLY,171B), LIST(=IDFROMC,43B),
        LIST(=INTQ,77B), LIST(=STRQ,101B), LIST(=CODEQ,104B),
        LIST(=IDQ,105B), LIST(=LBLQ,107B), LIST(=VECTQ,102B),
        LIST(=PAIRQ,103B), LIST(=LOGQ,161B),
        LIST(=EQ,113B),LIST(=SIM,162B),
        LIST(=PL,111B), LIST(=ZR,112B),
        LIST(=NILQ,132B),
        LIST(=VFROMS,46B), LIST(=SFROMV,45B),
        LIST(=IDFROMS,60B), LIST(=SFROMID,130B),
        LIST(=CFROMV,150B),
        LIST(PLUS.,71B), LIST(MINUS.,72B), LIST(=INEG,76B),
        LIST(STARX.,73B), LIST(SLASH.,74B), LIST(EXPON.,75B),
        LIST(=PROPL,160B), LIST(=SETPROPL,110B),
        LIST(=VALUE,50B), LIST(=SETVALUE,51B),
        LIST(=MODE,152B), LIST(=SETMODE,151B),
        LIST(=RDLINE,141B), LIST(=WRLINE,142B),
        LIST(=REWIND,143B), LIST(=BACKSPAC,144B),
        LIST(=GARBCOLL,153B),
        LIST(=SAVEALL,145B), LIST(=RESUMEAL,146B),
        LIST(=ENDFILE,147B),LIST(=PROTECT,155B),
        LIST(=TIME,154B),
        LIST(=NUMARGS,47B), LIST(=ARGUMENT,42B),
        LIST(=NE,LIST(113B,132B)),
        LIST(=LT,LIST(72B,111B,132B)),
        LIST(=GE,LIST(72B,111B)),
        LIST(=GT,LIST(72B,76B,111B,132B)),
        LIST(=LE,LIST(72B,76B,111B)),
        LIST(=STOP,116B) ),
        RETURN NIL
    END
  END;
INITMIS,= PROC(),
    BEGIN(),
        FALSE=NIL,
        SYSLIST=NIL,
        TTYFLAG=NIL,
        TALKATIV=0,
        OCTMODE=NIL,
        GENSYMB,=VECTOR(STAR,,DNUM,[1],DNUM,[1],DNUM,[1],DNUM,[1]),
        MTY,=SFROMID(=EMPTY),
        NDF,=IDFROMS(SFROMV(VECTOR(SLASHX,,SLASHX,,SLASHX,,SLASHX,))),
        MAKVLOCA(TRUE),
        MAKALOCA(TRUE),
        RETURN NIL
    END
  END;
COMMENT
/*************************************************************************
              I/O ROUTINES
*************************************************************************/
```

```
MAKFILE=  PROC(FN,LLEN),
     BEGIN(LIN,I),

        LIN=MAKVECTO(LLEN), FOR I=(1,LLEN) REPEAT LIN[I]=BLANK.,
        RETURN VECTOR(FN,LIN,LLEN,2,LIN,LLEN,2)
     END
  END;
READ.=     PROC(FIL),
     BEGIN(ITM,$FN,$LIN,$LLEN,$NEXT,$TERMLINE),
        FN=FIL[1], LIN=FIL[2], LLEN=FIL[3], NEXT=FIL[4], TERMLINE=
        READIN.,
        ITM=RDITEM.(),
        FIL[2]=LIN, FIL[4]=NEXT, FIL[3]=LLEN, RETURN ITM
     END
  END;
RDTOKEN.= PROC(FIL),
     BEGIN(ITM,$FN,$LIN,$LLEN,$NEXT,$TERMLINE),
        FN=FIL[1], LIN=FIL[2], LLEN=FIL[3], NEXT=FIL[4], TERMLINE=
        READIN.,
        ITM=LXSCAN.(), FIL[2]=LIN, FIL[4]=NEXT, FIL[3]=LLEN,
        IF ITM EQ IDTRUE. THEN RETURN TRUE,
        IF ITM EQ IDNIL, THEN RETURN NIL,
        RETURN ITM
     END
  END;
RDITEM.=  PROC(),
     BEGIN(ITM),
        ITM=LXSCAN.(),
        IF ITM EQ LPVAR, THEN ITM=GETLIST.()
        ELSEIF ITM EQ LBVAR, THEN ITM=GETVECT.(),
        IF ITM EQ IDTRUE. THEN RETURN TRUE,
        IF ITM EQ IDNIL, THEN RETURN NIL,
        RETURN ITM
     END
  END;
GETLIST.= PROC(),
     BEGIN(ITM,X,Y),
        X=Y=NIL:NIL,
  MOR,  ITM=RDITEM.(),
        IF ITM EQ RPVAR, THEN RETURN TL Y
        ELSEIF ITM EQ PERVAR, THEN TL X=(ITM=RDITEM.())
        ELSEIF ITM EQ IEOS. THEN DO
           NEXT=NEXT-1,
           PRINT.(LIST(=PARENS,=DONT,=MATCH,)). RETURN TL Y
        END
        ELSE X=ADDON.(X,ITM),
        GOTO MOR
     END
  END;
GETVECT.= PROC(),
     VFROML.(GETV.())
  END;
GETV.=    PROC(),
```

```
      BEGIN(ITM,X,Y),
         X=Y=NIL:NIL,
  MOR,   ITM=RDITEM,(),
         IF ITM EQ RBVAR, THEN RETURN TL Y
         ELSEIF ITM EQ IEOS, THEN DO
            NEXT=NEXT-1,
            PRINT,(LIST(=BRACKETS,=DONT,=MATCH,)), RETURN TL Y
         END
         ELSE X=ADDON,(X,ITM),
         GOTO MOR
      END
   END;
 LXSCAN,=  PROC(),
      BEGIN(C,J,E),
*  LXSCAN IS TABLE DRIVEN BY VALUES OF CHAR,
*  0-9  NUMBER
*  10  LETTER
*  11  SPECIAL
*  12 TERMINATING SPECIAL - CAN END A NAME
*  13  COMBINING SPECIAL - COMBINES WITH OTHER 13≠S
         NXT, IF NEXT GT LLEN THEN DO
            TERMLINE(), IF LLEN EQ 0 THEN
            RETURN NDF,
         END,
         C=LIN[NEXT], NEXT=NEXT+1,
         IF C EQ BLANK, THEN GOTO NXT,
         J=NEXT- 1,
         IF (E=CHAR,[C]) EQ 10 THEN GO SYMB
         ELSEIF E LE 9 THEN GO NUMB
         ELSEIF C EQ STRQU, THEN GO STR
         ELSEIF E EQ 13 THEN DO
            WHILE NEXT LE LLEN AND CHAR,[LIN[NEXT]] EQ 13
            REPEAT NEXT=NEXT+1,
            RETURN(IDFROMS(SFROMV(SUBV(LIN,J,NEXT-J))))
         END,
         RETURN IDFROMS(SFROMV(VECTOR(C))),
  SYMB,  WHILE NEXT LE LLEN AND CHAR,[LIN[NEXT]] LE 10 REPEAT
         NEXT=NEXT+1,
         IF NEXT LE LLEN AND CHAR,[LIN[NEXT]] EQ 12 THEN NEXT=NEXT+1,
         E=IDFROMS(SFROMV(SUBV(LIN,J,NEXT-J))),
         IF (NEXT-J) GT 8 THEN
         ERMSG,(=WARNING:E:=GREATER:=8:=CHARS:NIL),
         RETURN E,
  NUMB,  WHILE NEXT LE LLEN AND(C=CHAR,[LIN[NEXT]]) LE 9 REPEAT
         DO
            E=E*10+C, NEXT=NEXT+1
         END,
         IF NEXT GT LLEN OR LIN[NEXT] NE LETTB, THEN RETURN E,
         C=NEXT+1, NEXT=J, J=C, GOTO MAKO,
  STR,   E=MAKVECTO(0),
  MSTR,  IF NEXT GT LLEN THEN DO

            E=CONCATV(E,SUBV(LIN,J+1,LLEN-J)),J=0, TERMLINE(),
```

```
            IF LLEN EQ 0 THEN RETURN NDF,
         END,
         IF LIN[NEXT] NE STRQU. THEN DO
            NEXT=NEXT+1, GOTO MSTR
         END,
         E=CONCATV(E,SUBV(LIN,J+1,NEXT-J-1)),
         NEXT=NEXT+1,
         IF NEXT LE LLEN AND LIN[NEXT] EQ STRQU. THEN
            DO J=NEXT-1, NEXT=NEXT+1, GOTO MSTR END,
         RETURN SFROMV(E),
  MAKO,  E=0, WHILE NEXT LE LLEN AND (C=CHAR.[LIN[NEXT]]) LE 7
         REPEAT DO
            E=LOR(SHIFT(E,3),C), NEXT=NEXT+1
         END,
         IF LIN[NEXT] EQ LETTB. THEN DO
            NEXT=NEXT+1, RETURN E
         END,
         ERMSG.(=ERROR!=IN!=OCTAL!=NUMBER!NIL), NEXT=J, RETURN NIL
      END
   END;
READIN.=   PROC(),
      BEGIN(),
  START, IF FN NE 1 THEN LIN=VFROMS(RDLINE(FN))
         ELSEIF TTYFLAG THEN DO
            WRLINE(STRING(BLANK.,PROMPT.),OUTPUT[1]),
            LIN=VFROMS(RDLINE(FN))
         END
         ELSE DO
            WRLINE(CONCAT(STRING(BLANK.),(LIN=RDLINE(FN))),OUTPUT[1]),
            LIN=VFROMS(LIN)
         END,
         IF (LLEN=SIZE LIN) GT 0 AND LIN[1] EQ STAR. THEN GOTO START,
         NEXT=1
      END
   END;
WRITE=     PROC(L,FIL),
      BEGIN($FN,$LIN,$LLEN,$NEXT,$BPCNT,$TERMLINE),
         FN=FIL[1], LIN=FIL[5], LLEN=FIL[6], NEXT=FIL[7],
         TERMLINE=WRITOUT.,
         BPCNT=0, PUTITEM.(L), TERMLINE(),
         FIL[5]=LIN, FIL[7]=NEXT, RETURN L
      END
   END;
PRINT.=    PROC(),
      BEGIN($FN,$LIN,$LLEN,$NEXT,$BPCNT,$TERMLINE,I,N,FIL,TR),
         TR=TRACE, TRACE=0,
         N=NUMARGS(), FIL=OUTPUT, TERMLINE=WRITOUT.,
         FN=FIL[1], LIN=FIL[5], LLEN=FIL[6], NEXT=FIL[7],
         BPCNT=0, FOR I=(1,N) REPEAT PUTITEM.(ARGUMENT(I)), TERMLINE(),
         FIL[5]=LIN, FIL[7]=NEXT,
         TRACE=TR,
         RETURN ARGUMENT(N)
      END
```

```
   END;
WRITOUT.= PROC(),
      BEGIN(I,J),
         WRLINE(SFROMV(LIN),FN), NEXT=BPCNT-(BPCNT/20)*20+2,
         J=0,
         WHILE (I=LLEN-J) GT 72
         REPEAT DO
            SETSUBV(LIN,J+1,72,BLANKLI.), J=J+72
         END,
         SETSUBV(LIN,J+1,I,BLANKLI.)
      END
   END;
PUTBLAN.= PROC(),
      IF NEXT GT LLEN THEN TERMLINE() ELSE PUTCH.(BLANK.)
   END;
PUTITEM.= PROC(L),

      IF VECTQ(L) THEN PUTVECT.(L)
      ELSEIF PAIRQ(L) THEN DO
         IF NEXT GT LLEN-10 THEN TERMLINE() ELSE NIL, BPCNT=BPCNT+1,
         PUTCH.(LPAR.), PUTLIST.(L)
      END
      ELSEIF STRQ(L) THEN PUTSTR.(L)
      ELSEIF IDQ(L) THEN PUTCHV.(VFROMS(SFROMID(L)))
      ELSEIF INTQ(L) THEN PUTINT.(L)
      ELSEIF L EQ TRUE THEN PUTCHV.(VFROMS(SFROMID(IDTRUE.)))
      ELSEIF L EQ NIL THEN PUTCHV.(VFROMS(SFROMID(IDNIL.)))
      ELSEIF LBLQ(L) THEN PUTLBL.(L)
      ELSEIF CODEQ(L) THEN PUTCODE(L)
      ELSE PUTCHV.(VECTOR(STAR.,STAR.,STAR.))
   END;
PUTVECT.= PROC(L),
      BEGIN(N,I),
         IF NEXT GT LLEN-10 THEN TERMLINE(), PUTCH.(LBR.),
         IF (N=SIZE L) EQ 0 THEN NEXT=NEXT+1, BPCNT=BPCNT+1,
         FOR I=(1,N) REPEAT PUTITEM.(L[I]),
         NEXT=NEXT-1, PUTCH.(RBR.), BPCNT=BPCNT-1,
         PUTBLAN.()
      END
   END;
PUTLIST.= PROC(L),

      IF NULL L THEN DO
         NEXT=NEXT-1, PUTCH.(RPAR.), BPCNT=BPCNT-1,
         PUTBLAN.()
      END
      ELSEIF PAIRQ(L) THEN DO
         PUTITEM.(HD L), PUTLIST.(TL L)
      END
      ELSE DO
         PUTCHK.(PERIOD.), PUTCHK.(BLANK.), PUTITEM.(L),
         NEXT=NEXT-1, PUTCH.(RPAR.), BPCNT=BPCNT-1, PUTCHK.(BLANK.)
      END
```

```
  END;
PUTSTR,=  PROC(S),
    BEGIN(N,I),
        S=VFROMS(S), N=SIZE S,
        PUTCHK.(STRQU,), FOR I=(1,N) REPEAT PUTCHK.(S[I]),
        PUTCHK.(STRQU,), PUTCHK.(BLANK,)
    END
  END;
PUTLBL,=  PROC(L),
    BEGIN(),
        IF NEXT GT LLEN-10 THEN TERMLINE(), PUTCH.(SLASHX,),
        PUTINT.(L), NEXT=NEXT-1, PUTCH.(SLASHX.), PUTCH.(BLANK.)
    END
  END;
PUTCODE=  PROC(L),
    BEGIN(),
        IF NEXT GT LLEN-15 THEN TERMLINE(),
        PUTCH.(SLASHX,), PUTCHV.(VFROMS(SFROMID(=PROC))),
           PUTCHV.(VFROMS(SFROMID(IDFROMC(L)))), NEXT=NEXT-1,
           PUTCH.(SLASHX.), PUTCH.(BLANK.)
        END
    END;
PUTCHV,=  PROC(S),
    BEGIN(N,I),
        N=SIZE S, IF N GE LLEN-NEXT THEN TERMLINE(),
        SETSUBV(LIN,NEXT,N,S), NEXT=NEXT+N, PUTBLAN.()
    END
  END;
PUTCH,=   PROC(C),
    DO
        LIN[NEXT]=C, NEXT=NEXT+1
    END
  END;
PUTCHK,=  PROC(C),
    DO
        IF NEXT GT LLEN THEN TERMLINE() ELSE NIL,
        LIN[NEXT]=C, NEXT=NEXT+1
    END
  END;
PUTINT,=  PROC(N),
    BEGIN(S,NN,Q),
        S=NIL,
        IF NEXT GT LLEN-10 THEN TERMLINE(),
        IF OCTMODE THEN GOTO MAKO,
        IF PL N THEN NN=N ELSE DO
           PUTCH.(MSIGN,), NN=-N
        END,
 MOR,   Q=NN/10, S=DNUM.[NN-Q*10+1]:S, NN=Q,
        IF ¬ZR NN THEN GOTO MOR,
        GOTO PUTN,
 MAKO,  Q=0, S=LETTB.:S, NN=N,
        MORO, S=DNUM.[LAND(N,7)+1]:S, N=SHIFT(N,-3),
        IF PL NN THEN (IF ZR N THEN GOTO PUTN ELSE GOTO MORO)
```

```
        ELSEIF (Q=Q+1) GE 6 THEN GOTO PUTN
        ELSEIF ZR N THEN DO
           WHILE (Q=Q+1) LE 6 REPEAT S=SEVEN.:S,
           GOTO PUTN
        END
        ELSE GOTO MORO,
  PUTN, WHILE S REPEAT DO
           PUTCH.(HD S), S=TL S
        END, PUTCH.(BLANK.)
      END
   END;
ERMSG.=   PROC(L),
      PRINT.(STARX.,STARX.,STARX.,L,STARX.,STARX.,STARX.)
   END;
EOF=       PROC(X),
      IF X EQ NDF, THEN TRUE ELSE NIL
   END;
COMMENT
≠*********************************************************************
                  TRANSLATOR ROUTINES
*********************************************************************≠
TRANSLA.= PROC(B1),
      BEGIN(X),
         ERCOUNT.=0,
         X=FNOTN.(B1),
         MACSYMB.=VECTOR(STAR.,STAR.,DNUM.[1],DNUM.[1],DNUM.[1]),
         IF ZR ERCOUNT, THEN RETURN EXPAND.(X),
         RETURN(X)
      END
   END;
MACDEF.=  PROC(B1,B2),
      MACROLI.=LIST(B1,B2):MACROLI.
   END;
INFIX=     PROC(B1,B2,B3,B4),
      INFIXLI.=LIST(B1,LIST(≥INFIX,B4,B2,B3)):INFIXLI.
   END;
UNARY=     PROC(B1,B2,B3),
      UNARYLI.=LIST(B1,LIST(≥UNARY,B3,B2,B2)):UNARYLI.
   END;
BRACKET=  PROC(B1,B2,B3),
      UNARYLI.=LIST(B1,LIST(≥BRCKT,B3,B2,B2)):UNARYLI.
   END;
COMMENT
≠*********************************************************************
                  PARSER ROUTINES
*********************************************************************≠
FNOTN. =  PROC(L),
      IF ¬PAIRQ(L) THEN L ELSE
      BEGIN($LST,$RDTOKEN.,$READ.,$ENDFLAG,$TERMFLA.),
         LST=L, RDTOKEN.=READ.=GETNEXT.,
         TERMFLA.=NIL, ENDFLAG=SFROMID(=END),
         RETURN ANALYSE.(ENDFLAG)
   END END;
```

```
GETNEXT,= PROC(),
     IF ¬PAIRQ(LST) THEN ENDFLAG ELSE
     BEGIN(NXT),
        NXT=HD LST, LST=TL LST, RETURN NXT
     END
  END;
ANALYSE,= PROC(TERM),
     BEGIN(P,I1,I2,U),
        P=NIL, I1=RDTOKEN,(INPUT),
        IF EOF(I1) THEN GOTO ENDFIL,
        IF I1 EQ TERM THEN RETURN MTY,
        ELSEIF I1 EQ IEOS, THEN DO
           TERMFLA,=TRUE, RETURN MTY,
        END
        ELSE GOTO DOF1,
  DOF,  I1=RDTOKEN,(INPUT),
        IF EOF(I1) THEN GOTO ENDFIL,
  DOF1, (IF PAIRQ(I1) THEN DO
           I1=FNOTN,(I1), GOTO NOTU
        END),
        IF I1 EQ =COMMENT OR I1 EQ COMCHAR,
        THEN DO
           READ,(INPUT), GOTO DOF
        END
        ELSEIF I1 EQ =NOOP THEN DO
           I1=RDTOKEN,(INPUT), GOTO NOTU
        END
        ELSEIF I1 EQ EQSIGN, OR I1 EQ QSIGN, THEN DO
           I1 = =QUOTE:READ,(INPUT):NIL, GOTO NOTU
        END
        ELSEIF I1 EQ LPVAR, THEN DO
           I1=ANALYSE,(RPVAR,), GOTO NOTU
        END
        ELSEIF U=LOOKUP,(I1,UNARYLI,) THEN DO
           P=U:P, GOTO DOF
        END,
        IF LOOKUP,(I1,INFIXLI,) THEN
        DO
           ERMSG,(=IMPROPER:=USE:=OF:I1:NIL),
           ERCOUNT,=ERCOUNT,+1, IF I1 EQ IEOS, THEN TERMFLA,=TRUE
        END
        ELSEIF I1 EQ TERM THEN DO
           ERMSG,(=MISSING:=OPERAND:=BEFORE:TERM:NIL),
           ERCOUNT,=ERCOUNT,+1, RETURN MTY,
        END,
  NOTU, IF TERMFLA, THEN I2=IEOS, ELSE I2=RDTOKEN,(INPUT),
        IF I2 EQ LBVAR, THEN
        DO
           I1=LIST(≥INDEX,I1,ANALYSE,(RBVAR,)), GOTO NOTU
        END
        ELSEIF I2 EQ LPVAR, THEN DO
           I1=LIST(I1,ANALYSE,(RPVAR,)), GOTO NOTU
        END,
```

```
        (IF PAIRQ(I2) THEN DO
            I1=LIST(I1,FNOTN,(I2)), GOTO NOTU
        END
        ELSEIF VECTQ(I2) THEN DO
            I1=LIST(=INDEX,I1,FNOTN,(LFROMV,(I2))), GOTO NOTU
        END
        ELSEIF NULL(I2) THEN DO
            I1=LIST(I1,MTY,), GOTO NOTU
        END ),
  TEST4, IF I2 EQ IEOS, THEN DO
            TERMFLA,=TRUE, U=NIL, GOTO PLL
        END
        ELSEIF EOF(I2) THEN GOTO ENDFIL
        ELSEIF I2 EQ TERM THEN DO
            U=NIL, GOTO PLL
        END,
        IF NULL(U=LOOKUP,(I2,INFIXLI,)) THEN
        DO
        OPERROR,(I2), I2=RDTOKEN,(INPUT), GOTO TEST4
        END,
  TEST3, IF NULL(P) OR HD TL TL HD P LE HD TL TL TL U THEN DO
            P=U:I1:P, GOTO DOF
        END,
  PLL,   IF NULL P THEN DO
            IF TERMFLA, AND TERM NE IEOS,
            THEN DO
                ERMSG,(=MISSING:TERM:NIL), ERCOUNT,=ERCOUNT,+1
            END,
            RETURN I1
        END
        ELSEIF HD HD P EQ =BRCKT THEN DO
            I1=LIST(HD TL HD P,I1,I2), P=TL P, GOTO NOTU
        END
        ELSEIF HD HD P EQ =UNARY THEN DO
            I1=LIST(HD TL HD P,I1), P=TL P
        END
        ELSE DO
            I1=LIST(HD TL HD P,HD TL P,I1), P=TL TL P
        END,
        IF U THEN GOTO TEST3 ELSE GOTO PLL,
  ENDFIL,ERMSG,(=END:=OF:=FILE:=ON:=INPUT:NIL), STOP()
      END
  END;
REMCOM,=  PROC(B1),
     BEGIN(),
        IF NULL(B1) THEN RETURN(NIL),
        IF B1 EQ MTY, THEN RETURN NIL,
        RETURN(REMSEP,(B1,COMMA,))
     END
   END;
REMSEP,=  PROC(B1,B2),
     BEGIN(),
```

```
        IF ¬PAIRQ(B1) THEN RETURN B1:NIL,
        IF HD B1 EQ B2 THEN RETURN
      HD TL B1:REMSEP.(HD TL TL B1,B2),
        RETURN B1:NIL
    END
  END;
OPERROR.=  PROC(B1),
    BEGIN(),
       ERCOUNT.=ERCOUNT. + 1,
       ERMSG.(B1:LIST(=IS,=NOT,=AN,=OPERATOR)),
       RETURN(NIL)
    END
  END;
COMMENT
¢*****************************************************************
              SYNTAX TREE PROCESSORS
*****************************************************************¢
EXPAND.=  PROC(B1),
     BEGIN(OP,M,L,E),
        L=E=NIL:NIL,
        IF ¬PAIRQ(B1) THEN RETURN B1,
        IF PAIRQ(OP=HD B1) THEN GOTO NOTM,
        M=LOOKUP.(OP,MACROLI.),
        IF NULL(M) THEN GOTO NOTM,
        RETURN (M(B1)),
  NOTM,  E=TL E=EXPAND.(HD B1):NIL,
        B1=TL B1,
        IF PAIRQ(B1) THEN GOTO NOTM ELSE RETURN TL L
    END
  END;
EXLIS.=   PROC(B1),
     BEGIN(L,E),

        L=E=NIL:NIL,
  MOR,  IF NULL(B1) THEN RETURN TL L,
        E=TL E=EXPAND.(HD B1):NIL,
        B1=TL B1, GOTO MOR
    END
  END;
DUMMY.=   PROC(X),
     X
  END;
COMMENT
¢*****************************************************************
              MAIN CODE GENERATOR
*****************************************************************¢
CODEGEN.= PROC(NAM,X),
     DO VALUE NAM =
     BEGIN($ERRCNT,$LIST2,$END2,LIST3,
        $GLOBL,$LBLVALS,$NBYTE,$LBLNO,
        I,CODEV,
        $ARGS,$VARS,$EXPX,SARGS,$LBLIST),
        ERRCNT=0, GLOBL=NIL, NBYTE=3, LBLNO=1, LBLVALS=NIL,
```

```
        VARS=NIL, LBLIST=NIL,
        IF CGCHECK.(=PROC:(COMMA.:=ARGS:=EXPX:NIL):=END2:NIL,X)
        AND END2 EQ =END THEN NIL
        ELSE DO
           ERMSG.(=MISSING:=END:=AFTER:=PROC:NIL),
           ERRCNT=ERRCNT+1,
           GOTO ERR
           END,
           LIST2=END2=(0:NIL), END2=ADDON.(END2,(NAM=GREFS.(NAM))),
           ARGS=REMCOM.(ARGS),
           IF HD ARGS EQ DOLLAR, THEN ARGS=ARGS:NIL,
           SARGS=ARGS, ARGS=EXCHANG.(SARGS,1),
           COMP.(EXPX),
           EXCHANG.(SARGS,1),
           ASS.(KRTPROC.),
           IF TALKATIV GE 1 THEN PRINT.(=GLOBAL,=VARS,GLOBL),
           IF TALKATIV EQ 4 THEN DO
              PRINT.(=BINARY,=CODE),
              I=OCTMODE, OCTMODE=TRUE, PRINT.(LIST2), OCTMODE=I
           END,
           IF TALKATIV GE 2 THEN PRINT.(=LABEL,=LIST,LBLVALS),
           LIST3=LIST2, WHILE LIST2 REPEAT DO
              HD LIST2=SUBLBLS.(HD LIST2), LIST2=TL LIST2
           END,
           IF ZR ERRCNT THEN DO
              NBYTE=NBYTE-1, CODEV=MAKVECTO(NBYTE),
              FOR I=(1,NBYTE) REPEAT DO
                 CODEV[I]=IFROMID.(HD LIST3), LIST3=TL LIST3
              END,
              FOR I=(NBYTE,2,-1) REPEAT
              IF CODEV[I] GT 177B THEN DO
                 CODEV[I-1]=CODEV[I]/200B,
                 CODEV[I]=CODEV[I]-200B*CODEV[I-1]
              END
              ELSE NIL,
              RETURN CFROMV(CODEV)
           END,
  ERR,     ERMSG.(ERRCNT:=COMPILE.:=ERRORS:=IN:NAM:NIL),
           RETURN NIL
     END, NAM
        END
     END;
SUBLBLS.= PROC(X),
     IF ¬PAIRQ(X) THEN X ELSE
     BEGIN(Y),
        Y=LOOKUP.(HD X,LBLVALS),
        IF Y THEN RETURN Y,
        ERRCNT=ERRCNT+1,
           ERMSG.(X:=INVALID:=TREE:=NO:=CODE:=COMPILED:NIL),
        RETURN NIL
     END
  END;
COMP.=    PROC(X),
```

```
    BEGIN(FN,ARGL,GENR),
        IF IDQ(X) THEN RETURN GVAR.(X),
        IF ¬PAIRQ(X) THEN RETURN GCON.(X),
        FN=HD X, ARGL=TL X,
        IF IDQ(FN) THEN DO
            GENR=LOOKUP.(FN.CODGENL.),
            IF GENR THEN RETURN GENR(X) ELSE NIL
        END,
        RETURN CALLS.(X)
    END
  END;
GVAR.=     PROC(ATM),

    BEGIN(Y),
        IF Y=ORDINAL.(ATM,ARGS) THEN ASS.(KARG.,Y)
        ELSEIF Y=ORDINAL.(ATM,VARS) THEN ASS.(KVAR.,Y)
        ELSEIF MEMBER.(ATM,LBLIST) THEN ASS.(KLBL.,0,LIST(ATM))
        ELSE ASS.(KGLOB.,0,GREFS.(ATM)),
        RETURN NIL
    END
  END;
GCON.=     PROC(X),
    BEGIN(N),
        IF NULL(X) THEN ASS.(KNIL.)
        ELSEIF X EQ TRUE THEN ASS.(KTRUE.)
        ELSEIF X EQ MTY, THEN NIL
        ELSEIF INTQ(X) THEN
        (IF X LT 0 THEN DO
            GCON.(-X), ASS.(KINEG.)
        END
        ELSEIF X LE 177B THEN ASS.(KNUM1.,X)
        ELSEIF X LE 37777B THEN ASS.(KNUM2.,0,X)
        ELSE ASS.(KNUM3.,0,0,X) )
        ELSEIF IDQ(X) THEN DO

            ASS.(KNUM2.,0,GREFS.(X)), ASS.(KMAKVAR.)
        END
        ELSE DO
            N=GENSYM.(), VALUE N=X, ASS.(KGLOB.,0,N)
        END
    END
  END;
CALLS.=    PROC(X),
    BEGIN(ARG,FN,ARGL,SARGL,OP),
        FN=HD X, ARGL=SARGL=ARGLIST.(TL X),
        WHILE ARGL REPEAT DO
            COMP.(HD ARGL), ARGL=TL ARGL
        END,
        IF OP=LOOKUP.(FN,OPLIST.) THEN
        (IF INTQ(OP) THEN RETURN ASS.(OP)
        ELSE RETURN MAPX.(OP,ASS.) ),
        COMP.(FN),
        ASS.(KCALL.,LENGTH.(SARGL))
```

```
      END
   END;
ARGLIST. = PROC(L),
      IF NULL L THEN NIL
      ELSEIF HD L EQ MTY. THEN NIL
      ELSEIF ¬PAIRQ(HD L) THEN L
      ELSEIF HD HD L EQ COMMA. THEN REMSEP.(HD L,COMMA.)
      ELSE L
   END;
GREFS.=    PROC(A),
      DO
          IF MEMBER.(A,SYSLIST) THEN
          DO
              A=VFROMS(SFROMID(A)),
              A=CONCATV(VECTOR(STAR.).A), A=IDFROMS(SFROMV(A))
          END
          ELSE NIL,
          IF MEMBER.(A,GLOBL) THEN NIL ELSE GLOBL=A:GLOBL,
          A
      END
   END;
ASS.=      PROC(),
      BEGIN(I),
          NBYTE=NBYTE+NUMARGS(),
          FOR I=(1,NUMARGS()) REPEAT END2=TL END2=ARGUMENT(I):NIL
      END
   END;
LBL. =     PROC(X),
      LBLVALS=(X:NBYTE:NIL):LBLVALS
   END;
GENLBL.=   PROC(),
      LBLNO=LBLNO+1
   END;
COMMENT
≠**************************************************************************
                   CODE GENERATORS
**************************************************************************≠
GLAMBDA=   PROC(X),
      BEGIN(N),
          N=GENSYM.(),
          CODEGEN.(N,X),
          IF VALUE N EQ NIL THEN ERRCNT=ERRCNT+1.
          ASS.(KGLOB.,0,GREFS.(N)), RETURN NIL
      END
   END;
EXCH1.=    PROC(L,I),
      IF NULL(L) THEN NIL
      ELSEIF ¬PAIRQ(HD L) THEN HD L : EXCHANG.(TL L,I+1)
      ELSE DO
          ASS.(KARG.,I), ASS.(KGLOB.,0,HD TL HD L),
          ASS.(KASTORE.,I), ASS.(KPOP.,1),
          ASS.(KGSTORE.,0,HD TL HD L), ASS.(KPOP.,1),
          I : EXCHANG.(TL L,I+1)
```

```
        END
    END;
EXCH2,=     PROC(L,I),
        BEGIN(X),
            IF NULL(L) THEN RETURN NIL
            ELSEIF ¬PAIRQ(HD L) THEN X=HD L
            ELSE X=HD TL HD L,
            ASS,(KARG.,I), ASS,(KGLOB.,0,X),
            ASS,(KASTORE.,I), ASS,(KPOP.,1), ASS,(KGSTORE.,0,X),
            ASS,(KPOP.,1), I:EXCHANG,(TL L,I+1)
        END
    END;
MAKALOCA= PROC(COND),
        IF COND THEN EXCHANG,=EXCH1,
        ELSE EXCHANG,=EXCH2,
    END;
GPROG=      PROC(X),
        BEGIN($VARS,$PROGRAM,T1,$LBLIST,$RET,SVARS),
            IF CGCHECK,(=BEGIN:(COMMA,:=VARS:=PROGRAM:NIL):=RET:NIL,X)
            AND RET EQ =END THEN NIL
            ELSE DO
                ERMSG,(=MISSING:=END:=AFTER:=BEGIN:NIL),
                RETURN ERRCNT=ERRCNT+1
            END,
            LBLIST=NIL,
            VARS=REMCOM,(VARS), PROGRAM=REMCOM,(PROGRAM),
            IF HD VARS EQ DOLLAR, THEN VARS=VARS:NIL,
            ASS,(KNVARS.,LENGTH,(VARS)),
            SVARS=VARS, VARS=SAVLOCS,(SVARS,1),
            RET=GENLBL,(),
            X=PROGRAM,
            WHILE PROGRAM REPEAT DO
                T1=HD PROGRAM, PROGRAM=TL PROGRAM,
                IF ¬PAIRQ(T1) THEN LBLIST=T1:LBLIST ELSE NIL
            END,
            PROGRAM=X,
            WHILE PROGRAM REPEAT DO
                T1=HD PROGRAM, PROGRAM=TL PROGRAM,
                IF ¬PAIRQ(T1) THEN LBL,(T1)
                ELSE DO
                    ASS,(KSETSTK.), COMP,(T1)
                END
            END,
            ASS,(KSETSX.),
            LBL,(RET),
            RSTLOCS,(SVARS,1),
            ASS,(KRETPRO.)
        END
    END;
SAVL1,=     PROC(L,I),
        IF NULL(L) THEN NIL
        ELSEIF ¬PAIRQ (HD L) THEN HD L : SAVLOCS,(TL L,I+1)
        ELSE DO
```

```
            ASS,(KGLOB,,0,HD TL HD L), ASS,(KVSTORE,,I),
            I:SAVLOCS.(TL L,I+1)
        END
    END;
SAVL2,=    PROC(L,I),
        BEGIN(),
            IF NULL(L) THEN RETURN (NIL)
            ELSEIF ¬PAIRQ(HD L) THEN ASS,(KGLOB,,0,HD L)
            ELSEIF HD HD L EQ DOLLAR, THEN ASS,(KGLOB,,0,HD TL HD L)
            ELSE RETURN(HD TL HD L : SAVLOCS,(TL L,I+1)),
            ASS,(KVSTORE,,I), I:SAVLOCS,(TL L,I+1)
        END
    END;
RESTL1,=   PROC(L,I),
        IF NULL(L) THEN NIL
        ELSEIF ¬PAIRQ(HD L) THEN RSTLOCS,(TL L,I+1)
        ELSE DO
            ASS,(KVAR,,I), ASS,(KGSTORE,,0,HD TL HD L),
            ASS,(KPOP,,1), RSTLOCS,(TL L,I+1)
        END
    END;
RESTL2,=   PROC(L,I),
        BEGIN(),
            IF NULL(L) THEN RETURN NIL
            ELSEIF ¬PAIRQ(HD L) THEN DO
                ASS,(KVAR,,I), ASS,(KGSTORE,,0,HD L)
            END
            ELSEIF HD HD L EQ DOLLAR, THEN DO
                ASS,(KVAR,,I),
                ASS,(KGSTORE,,0,HD TL HD L)
            END
            ELSE RETURN RSTLOCS,(TL L,I+1),
            ASS,(KPOP,,1), RSTLOCS,(TL L,I+1)
        END
    END;
MAKVLOCA= PROC(COND),
        IF COND THEN DO
            SAVLOCS,=SAVL1,, RSTLOCS,=RESTL1,
        END
        ELSE DO
            SAVLOCS,=SAVL2,, RSTLOCS,=RESTL2,
        END
    END;
GRETURN=  PROC(X),
        DO
            COMP.(HD TL X),ASS,(KSETSX,), ASS,(KJMP,,0,LIST(RET))
        END
    END;
GPROGN=   PROC($L),
        BEGIN($E),
            IF CGCHECK,(=DO:=L:=E:NIL,L) AND E EQ =END
            THEN NIL ELSE DO
                ERMSG,(=MISSING:=END:=AFTER:=DO:NIL),
```

```
            ERRCNT=ERRCNT+1, RETURN NIL
         END,
         L=REMCOM.(L), COMP.(HD L), L=TL L,
         WHILE L REPEAT DO
            IF (E=HD L) NE MTY. THEN DO
               ASS.(KPOP,.1), COMP.(E)
            END,
            L=TL L
         END
      END
   END;
GGO=        PROC(X),
      BEGIN(ARG),
         ARG=HD TL X,
         IF MEMBER.(ARG,LBLIST) THEN ASS.(KJMP,.0,LIST(ARG))
         ELSE DO
            COMP.(ARG), ASS.(KJMPI,)
         END
      END
   END;
GCOND=      PROC(X),
      BEGIN($LAST),
         LAST=GENLBL.(), X=HD TL X,
         IF HD X EQ=THEN THEN DO
            GTHEN.(HD TL X,HD TL TL X),
            ASS.(KNIL,)
         END
         ELSE GELSE.(HD X,HD TL X,HD TL TL X),
         LBL.(LAST)
      END
   END;
GELSE.=     PROC(OP,TH,REM),
      BEGIN(),
         IF OP EQ =THEN THEN DO
            GTHEN.(TH,REM),
            RETURN ASS.(KNIL,)
         END
         ELSEIF ¬PAIRQ(TH) OR ¬(HD TH EQ =THEN) THEN DO
            ERMSG.(TH:=NOT:=VALID:=THEN:=CLAUSE:NIL), ERRCNT=ERRCNT+1,
            RETURN NIL
         END
         ELSE GTHEN.(HD TL TH,HD TL TL TH),
         IF OP EQ =ELSEIF THEN
         GELSE.(HD REM,HD TL REM,HD TL TL REM)
         ELSEIF OP EQ =ELSE THEN COMP.(REM)
         ELSE DO
            ERMSG.(=EXPECT:=ELSE:=HAVE:OP:NIL), ERRCNT=ERRCNT+1
         END
      END
   END;
GTHEN.=     PROC(P,E),
      BEGIN(NTRUE),
         IF HD E EQ =GO AND MEMBER.(HD TL E,LBLIST) THEN
```

```
        DO
           COMP.(P), ASS.(KJMPT.,0,LIST(HD TL E))
        END
        ELSE DO
           COMP.(P), NTRUE=GENLBL.(),
           ASS.(KJMPF.,0,LIST(NTRUE)), COMP.(E),
           IF HD E NE =GO AND HD E NE =RETURN THEN
           ASS.(KJMP.,0,LIST(LAST)) ELSE NIL,
           LBL.(NTRUE)
        END,
        RETURN NIL
     END
   END;
GAND=      PROC(X),
     BEGIN(L),
        L=GENLBL.(), ASS.(KNIL.),
        COMP.(HD TL X) ,ASS.(KJMPF.,0,LIST(L)),
        ASS.(KPOP.,1), COMP.(HD TL TL X),
        LBL.(L)
     END
   END;
GOR=       PROC(X),
     BEGIN(L),
        L=GENLBL.(), ASS.(KTRUE.),
        COMP.(HD TL X) ,ASS.(KJMPT.,0,LIST(L)),
        ASS.(KPOP.,1), COMP.(HD TL TL X),
        LBL.(L)
     END
   END;
GQUOTE=    PROC(X),
     BEGIN(ARG),

        ARG=HD TL X,
        GCON.(ARG)
     END
   END;
GSETQ=     PROC(X),
     BEGIN(LHS,LHSOP,LHSARGS,RHS),
        LHS=HD TL X, RHS=HD TL TL X,
        IF ¬PAIRQ(LHS) THEN DO
           COMP.(RHS), ASSIGN.(LHS), RETURN NIL
        END,
        LHSOP=HD LHS, LHSARGS=ARGLIST.(TL LHS),
        IF LHSOP EQ =HD THEN COMP.(LIST(=RPLACA,HD LHSARGS,RHS))
        ELSEIF LHSOP EQ =TL THEN COMP.(LIST(=RPLACD,HD LHSARGS,RHS))
        ELSEIF LHSOP EQ =INDEX THEN
        COMP.(LIST(=SETINDEX,HD LHSARGS,HD TL LHSARGS,RHS))
        ELSEIF LHSOP EQ =VALUE THEN
        COMP.(LIST(=SETVALUE,HD LHSARGS,RHS))
        ELSEIF LHSOP EQ =SUBV OR LHSOP EQ =SUB THEN
        COMP.(LIST(=SETSUBV,HD LHSARGS,HD TL LHSARGS,
        HD TL TL LHSARGS,RHS))
        ELSE DO
```

```
            ERMSG.(LHS:=NOT:=VALID:=LHS:NIL), ERRCNT=ERRCNT+1
         END
      END
   END;
ASSIGN.=   PROC(ATM),
      BEGIN(X),
         IF ¬IDQ(ATM) THEN DO
            ERRCNT=ERRCNT+1,
            PRINT.(ATM:EQSIGN,:VAL:NIL),
            PRINT.(=ASSIGN,:=ERROR:NIL),
            RETURN NIL
         END,
         IF X=ORDINAL.(ATM,ARGS) THEN ASS.(KASTORE.,X)
         ELSEIF X=ORDINAL.(ATM,VARS) THEN ASS.(KVSTORE.,X)
         ELSE ASS.(KGSTORE.,0,GREFS.(ATM))
      END
   END;
GWHILE=    PROC(X),
      BEGIN(MORE,NTRUE,$P,$E),
         IF ¬CGCHECK.(=WHILE:(=REPEAT:=P:=E:NIL):NIL,X) THEN
         DO
            PRINT.(=WHILE:=ERROR:NIL), RETURN ERRCNT=ERRCNT+1
         END,
         ASS.(KNIL.),
         MORE=GENLBL.(),
         LBL.(MORE),
         COMP.(P),
         NTRUE=GENLBL.(),
         ASS.(KJMPF.,0,LIST(NTRUE)),
         ASS.(KPOP.,1),
         COMP.(E),
         ASS.(KJMP.,0,LIST(MORE)),
         LBL.(NTRUE)
      END
   END;
GFOR =     PROC(X),
      BEGIN($E,$I,$J,$K,L,LAST,FORL),
         L = =FOR:(=REPEAT:(EQSIGN,:=I:(COMMA,:=J:=K:NIL):NIL)
         :=E:NIL):NIL,
         IF ¬CGCHECK.(L,X) THEN
         DO
            PRINT.(=FOR:=ERROR:NIL), RETURN ERRCNT=ERRCNT+1
         END,
         IF ¬PAIRQ(K) OR HD K NE COMMA, THEN L=1
         ELSE DO
            L=HD TL TL K, K=HD TL K
         END,
         LAST=GENLBL.(),
         COMP.(K), COMP.(L), COMP.(J),
         ASS.(KNIL.),
         ASSIGN.(I),
         ASS.(KJMP.,0,LIST(LAST)),
         FORL=GENLBL.(), LBL.(FORL),
```

```
        ASS,(KPOP.,1),
        ASSIGN.(I),
        COMP.(E),
        ASS,(KSTEPLO.),
        LBL,(LAST),
        ASS,(KTLOOP.,0,LIST(FORL)),
        ASS,(KPOP.,3)
    END
  END;
CGCHECK.= PROC(FORM,TREE),
    IF IDQ(FORM) THEN DO
       VALUE FORM=TREE,TRUE
    END
    ELSEIF ¬PAIRQ(FORM) THEN FORM EQ TREE
    ELSEIF ¬PAIRQ(TREE) THEN NIL
    ELSEIF HD FORM NE HD TREE THEN NIL
    ELSE CGCHKL.(TL FORM,TL TREE)
  END;
CGCHKL.=  PROC(FL,TR),
    IF NULL FL THEN NULL TR
    ELSEIF ¬PAIRQ(FL) OR ¬PAIRQ(TR) THEN NIL
    ELSEIF CGCHECK.(HD FL,HD TR) THEN CGCHKL.(TL FL,TL TR)
    ELSE NIL
  END;
GLIST=    PROC(X),
    BEGIN(),
        X=ARGLIST.(TL X), MAPX.(X,COMP.),
        ASS,(KLIST.,0,LENGTH.(X))
    END
  END;
GVECTOR=  PROC(X),
    BEGIN(),
        X=ARGLIST.(TL X), MAPX.(X,COMP.),
        ASS,(KVECTOR.,0,LENGTH.(X))
    END
  END;
GSTRING=  PROC(X),
    BEGIN(),
        X=ARGLIST.(TL X), MAPX.(X,COMP.),
        ASS,(KSTRING.,0,LENGTH.(X))
    END
  END;
STOP;
STOP;
COMMENT ≠
```

```
                           SUPPLEMENT   ≠
BREAKUP,= PROC(P,X),
     BEGIN(I),
         IF IDQ(P) THEN DO
             VALUE P=X, TRUE
         END
         ELSEIF PAIRQ(P) THEN
         (IF ¬PAIRQ(X) THEN NIL
         ELSEIF HD P EQ ≥≥ AND EQUAL.(HD TL P,HD X) THEN
         BREAKUP.(TL TL P,TL X)
         ELSEIF BREAKUP.(HD P,HD X) THEN BREAKUP.(TL P,TL X)
         ELSE NIL)
         ELSEIF VECTQ(P) THEN
         ( IF ¬VECTQ(X) THEN NIL
         ELSE DO
             I=1, WHILE BREAKUP.(P[I],X[I]) REPEAT
             IF I LT SIZE P AND I LT SIZE X THEN I=I+1
             ELSE RETURN SIZE P EQ SIZE X,
             NIL
         END)
         ELSE P EQ X
     END
   END;
!OPLIST,=LIST(=EQSTR,133B)!OPLIST.;
!OPLIST, = LIST(=REMARK,172B)!OPLIST.;
!EQUAL,=     PROC(X,Y),
     BEGIN(I),
   MOR,    IF X EQ Y THEN RETURN TRUE
         ELSEIF X SIM Y THEN
         (IF PAIRQ(X) THEN (IF EQUAL.(HD X,HD Y) THEN
         DO
             X=TL X, Y=TL Y, GOTO MOR
         END
         ELSE RETURN NIL)
         ELSEIF VECTQ(X) THEN(IF SIZE X NE SIZE Y THEN RETURN NIL
         ELSE DO
             FOR I=(1,SIZE X) REPEAT IF ¬EQUAL.(X[I],Y[I])
             THEN RETURN NIL,
             RETURN TRUE
         END)
         ELSE RETURN EQSTR(X,Y) )
         ELSE RETURN NIL
     END
   END;
 MACRO=MACDEF,;
 OPLIST,=LIST(=STKTRACE,170B)!OPLIST.;
 OPLIST,=LIST(=OPEN,166B):LIST(=CLOSE,167B);OPLIST.;
 MAKPROPS= PROC(P,ID),
     BEGIN(L,Z),
         L=VALUE(ID),
         SETPROPL(ID,LIST(=PREVAL,L):PROPL(ID)),
         Z=L, L=NIL,
         WHILE PAIRQ(Z) REPEAT DO
```

```
            L=HD Z:L, Z=TL Z
        END,
        WHILE PAIRQ(L) REPEAT DO
            SETPROPY(P,HD HD L,HD TL HD L),
            L=TL L
        END,
        SETVALUE(ID,P)
    END
  END;
SETPROPY= PROC(P,ID,V),
    IF PAIRQ(PROPL(ID)) THEN SETPROPL(ID,LIST(P,V):PROPL(ID))
    ELSE SETPROPL(ID,LIST(P,V):NIL)
  END;
COPY=      PROC(X),
    IF VECTQ(X) THEN BEGIN(V,L,I),
        L=SIZE X, V=MAKVECTO(L),
        FOR I = (1, L) REPEAT V[I] = COPY(X[I]),
        RETURN(V)
    END
    ELSEIF STRQ(X) THEN SUB(X,1,SIZE X)
    ELSEIF CODEQ(X) THEN DO
        PRINT,(X,=CANNOT,=BE,=COPIED),X
    END
    ELSEIF PAIRQ(X) THEN COPY(HD X):COPY(TL X)
    ELSE X
  END;
PRTR.=TRANSLA. ( =(
    PROC(),BEGIN(N,I,R), N=NUMARGS(),
              PRINT,(=ARGUMENT,=OF,=PNAME,=ARE),
              FOR I=(1,N) REPEAT PRINT.(ARGUMENT(I)), PRINT.(),
              IF N EQ 0 THEN R=OLDPR()
              ELSEIF N EQ 1 THEN R=OLDPR(ARGUMENT(1))
              ELSEIF N EQ 2 THEN R=OLDPR(ARGUMENT(1),ARGUMENT(2))
              ELSEIF N EQ 3 THEN R=OLDPR(ARGUMENT(1),ARGUMENT(2),
              ARGUMENT(3))
              ELSE DO
                  PRINT.(≠TOO MANY ARGS FOR TRACER≠), RETURN NIL
              END,
              PRINT,(=VALUE,=OF,=PNAME,=IS), PRINT.(R), PRINT.(),
              RETURN R
          END
    END ));
PTRACE=PROC(ID),
    BEGIN(G,NEWP),
        G=GENSYM,(), SETVALUE(G,VALUE ID ),
        NEWP=SUBST,(ID,=PNAME,PRTR.),
        NEWP=SUBST,(G,=OLDPR,NEWP),
        CODEGEN,(ID,NEWP)
    END
  END;
PROCTRAC= PROC(L),
    BEGIN(V,N,I),
        PRINT,(≠BALM4 TRACE - PROCS IN REVERSE CALLING SEQUENCE≠),
```

```
      WHILE L REPEAT
      DO
         PRINT.(IDFROMC(HD HD L)), V=TL HD L, N=SIZE V,
         FOR I=(1,N) REPEAT PRINT.(=ARG,I,==,V[I]),
         L=TL L
      END
    END
  END;
CONSTRUC= PROC(P),
    IF IDQ(P) THEN VALUE P
    ELSEIF PAIRQ(P) THEN
    (IF HD P EQ ≥≥ THEN HD P:CONSTRUC(TL P)
    ELSE CONSTRUC(HD P):CONSTRUC(TL P) )
    ELSE P
  END;
GETPROP=  PROC(ID,P),
    LOOKUP.(P,PROPL(ID))
  END;
VBLANK.=VECTOR(BLANK.);
MAKFILE = PROC(NAM,LLEN),
    BEGIN(LIN,I),
      LIN=MAKVECTO(LLEN). FOR I=(1,LLEN) REPEAT LIN[I]=BLANK.,
      RETURN VECTOR(OPEN(NAM,LLEN),LIN,LLEN,2,LIN,LLEN,2)
    END
  END;
SUBST.=   PROC(A,X,L),
    IF X EQ L THEN A
    ELSEIF ¬PAIRQ(L) THEN L
    ELSE SUBST.(A,X,HD L):SUBST.(A,X,TL L)
  END;
MMEANS=   PROC(L),
    BEGIN(LS,RS,M,OP,ROP,PREVM),
      LS=HD TL L, RS=HD TL TL L,
      IF ¬PAIRQ(LS) THEN
         DO ERMSG.(LS:=NOT:=VALID:=LHS:=FOR:=MEANS:NIL),
         RETURN NIL
      END,
      OP=HD LS,
      ROP=NIL,
      IF PAIRQ(RS) THEN DO
         ROP=HD RS,
         IF ROP EQ =X1 OR ROP EQ =X2 OR ROP EQ =X3 OR ROP EQ =X4
         OR ROP EQ =X5 OR ROP EQ =X6 OR ROP EQ =X7 OR ROP EQ =X8
         OR ROP EQ =X9 OR ROP EQ =X10 THEN ROP = NIL
         ELSEIF MMATCH.(LS,TL RS) THEN DO
            ERMSG.(LIST(=USE,=OF,OP,=IN,=MEANS,=WILL,=RECURSE,=
            FOREVER)),
            RETURN NIL
         END
      END,
      M=SUBST.(LS,=L,TMAC.),
      M=SUBST.(RS,=R.,M),
      PREVM = LOOKUP.(OP,MACROLI.),
```

```
        IF PREVM EQ NIL THEN
        M=SUBST,(=EXLIS,,=E,,M)
        ELSE M=SUBST,(PREVM,=E,,M),
        IF NULL(ROP) THEN M=SUBST,(=EXPAND,,=EX,,M)
        ELSEIF(PREVM=LOOKUP,(ROP,MACROLI,))
            THEN M=SUBST,(PREVM,=EX,,M)
        ELSE M=SUBST,(=EXLIS,,=EX,,M),
        MACDEF,(OP,VALUE CODEGEN,(GENSYM,(),M)),
        RETURN NIL
     END
   END;
MMATCH,=  PROC(LS,RS),
     BEGIN(X,$X1,$X2,$X3,$X4,$X5,$X6,$X7,$X8,$X9,$X10),
        X1=X2=X3=X4=X5=X6=X7=X8=X9=X10=GENSYMB.,
        WHILE RS REPEAT IF ¬PAIRQ(RS) THEN RETURN NIL
        ELSEIF PAIRQ(X=HD RS) THEN
        (IF MMATCH,(LS,X) THEN RETURN TRUE ELSE RS=TL RS)
        ELSEIF X EQ HD LS THEN RETURN MATCH,(LS,RS)
        ELSE RS=TL RS
     END
   END;
TMAC.=TRANSLA.(=(PROC(S),
     BEGIN ($X1,$X2,$X3,$X4,$X5,$X6,$X7,$X8,$X9,$X10,
        $G1,$G2,$G3,$G4,$G5,$G6,$G7,$G8,$G9,$G10),
        G1=G2=G3=G4=G5=G6=G7=G8=G9=G10=0,
        X1=X2=X3=X4=X5=X6=X7=X8=X9=X10=GENSYMB.,
        IF MATCH,(=L,S) THEN
        RETURN EX,(BUILD,(=R,))
        ELSE RETURN E,(S)
     END
   END)));
MATCH,=    PROC(L,S),
     IF PAIRQ(L) THEN
     (IF PAIRQ(S) THEN
     (IF MATCH,(HD L, HD S) THEN
     MATCH,(TL L,TL S)
     ELSE FALSE)
     ELSE FALSE)
     ELSEIF L EQ =X1 OR L EQ =X2 OR L EQ =X3
     OR L EQ =X4 OR L EQ =X5 OR L EQ =X6
     OR L EQ =X7 OR L EQ =X8 OR L EQ =X9 OR L EQ =X10
     THEN (IF VALUE L EQ GENSYMB. THEN DO
        VALUE L=S,TRUE
     END
     ELSEIF EQUAL,(VALUE L,S) THEN TRUE ELSE NIL)
     ELSEIF L EQ S THEN TRUE
     ELSE FALSE
   END;
BUILD,=    PROC(R),
     IF PAIRQ(R) THEN
     BUILD,(HD R):BUILD,(TL R)
     ELSEIF R EQ =X1 OR R EQ =X2 OR R EQ =X3
     OR R EQ =X4 OR R EQ =X5 OR R EQ =X6
```

```
      OR R EQ =X7 OR R EQ =X8 OR R EQ =X9 OR R EQ =X10
      THEN (IF VALUE R EQ GENSYMB. THEN R ELSE VALUE R)
      ELSEIF R EQ =G1 OR R EQ =G2 OR R EQ =G3 OR R EQ =G4
      OR R EQ =G5 OR R EQ =G6 OR R EQ =G7 OR R EQ =G8 OR R EQ =G9
      OR R EQ =G10 THEN
      ( IF VALUE R EQ 0 THEN VALUE R=GENNAM.() ELSE VALUE R)
      ELSE R
   END;
GENNAM.=  PROC(),
      BEGIN(I,J),
         FOR I=(5,3,-1) REPEAT IF (J=CHAR.[MACSYMB.[I]]) LT 9
         THEN DO
            MACSYMB.[I]=DNUM.[J+2], RETURN IDFROMS(SFROMV(MACSYMB.))
         END
         ELSE MACSYMB.[I]=DNUM.[1]
      END
   END;
INFIX(=MEANS,0,0,=MEANS);
MACDEF.(=MEANS,MMEANS);
MAKPROPS(=INFIX,=INFIXLI.);
MAKPROPS(=UNARY,=UNARYLI.);
MAKPROPS(=MACRO,=MACROLI.);
MAKPROPS(=CODEG,=CODGENL.);
MAKPROPS(=INSTR,=OPLIST.);
COMPILE.= PROC(X),
      VALUE CODEGEN.(GENSYM.(),TRANSLA.(X))
   END;
SAVSTAT=  PROC(),
      VECTOR(UNARYLI.,INFIXLI.,MACROLI.,CODGENL.,OPLIST.)
   END;
RESTAT=   PROC(X),
      IF VECTQ(X) THEN DO
         UNARYLI.=X[1], INFIXLI.=X[2], MACROLI.=X[3],
         CODGENL.=X[4], OPLIST.=X[5]
      END
      ELSE PRINT.(LIST(=RESTAT,=ARG.,=IS,=INVALID))
   END;
REMOVEX.= PROC(LISTID,OP),
      BEGIN(P,L,ID),
         P=NIL, L=VALUE LISTID,
         ID=OP,
  MOR,   WHILE PAIRQ(L) REPEAT DO
            IF ID EQ HD HD L THEN RETURN
            (IF PAIRQ(P) THEN TL P=TL L
            ELSEIF IDQ(P) THEN SETPROPL(OP,TL L)
            ELSE VALUE LISTID=TL L),
            P=L, L=TL L
         END,
         IF IDQ(L) THEN DO
            P=L, L=PROPL(ID), ID=P, GOTO MOR
         END
         ELSE PRINT.(LIST(OP,=NOT,=ON,LISTID))
      END
```

```
   END;
REMMACRO= PROC(ID),
      REMOVEX.(=MACROLI.,ID)
   END;
REMINFIX= PROC(ID),
      REMOVEX.(=INFIXLI.,ID)
   END;
REMUNARY= PROC(ID),
      REMOVEX.(=UNARYLI.,ID)
   END;
ERROR=     PROC(TYPE),
      IF TYPE EQ 3 THEN DO
         EXECUTE(INPUT,OUTPUT), STOP()
      END
      ELSE STOP()
   END;
SAVEBALM= PROC(S),
      DO
         REWIND(S), GARBCOLL(), SAVEALL(S),
         CLOSE(≠BLM4SVD≠)
      END
   END;
SUB=       PROC(I,J,K),
      SUB(I,J,K)
   END;
SUBV=      PROC(I,J,K),
      SUBV(I,J,K)
   END;
GENP(X1) MEANS X1=PROC(X,Y),
      X1(X,Y)
   END;
GENP(WRLINE);
GENP(SHIFT); GENP(CONCATV); GENP(CONCAT);
GENP(LAND); GENP(LOR); GENP(XOR); GENP(EQSTR);
REMMACRO(=GENP);
GENP(X1) MEANS X1=PROC(X),
      X1(X)
   END;
GENP(RDLINE);
GENP(MAKVECTO); GENP(COMPL); GENP(INTQ); GENP(STRQ); GENP(CODEQ);
GENP(IDQ); GENP(LBLQ); GENP(VECTQ); GENP(PAIRQ); GENP(LOGQ);
GENP(NILQ); GENP(VFROMS); GENP(SFROMV); GENP(IDFROMS); GENP(SFROMID);
GENP(IDFROMC); GENP(REWIND); GENP(SAVEALL); GENP(RESUMEAL);
GENP(ENDFILE); GENP(PROTECT); GENP(STKTRACE);
GENP(REMARK);
REMMACRO(=GENP);
TIME = PROC(), TIME() END;
STOP = PROC(), STOP() END;
CHGCHAR = PROC(CHR,NUM),
      CHAR.[VFROMS(SFROMID(CHR))[1]] = NUM
   END;
ADDON = ADDON.; BREAKUP = BREAKUP.; CODEGEN = CODEGEN.;
COMPILE = COMPILE.; DUMMY = DUMMY.; EQUAL = EQUAL.;
```

```
EXPAND = EXPAND.; GENSYM = GENSYM.; IFROMID = IFROMID.;
LENGTH = LENGTH.; LFROMV = LFROMV.; LOOKUP = LOOKUP.;
MACDEF = MACDEF.; MAPX = MAPX.; MEMBER = MEMBER.;
ORDINAL = ORDINAL.; PRINT = PRINT.; RDTOKEN = RDTOKEN.;
READ = READ.; SUBST = SUBST.; TRANSLAT = TRANSLA.;
VFROML = VFROML.;

******     DEBUGGING PROCEDURES ***
GOFLAG.=NIL;
PRCFLAG.=NIL;
PRCTRACE=NIL;
GOTRACE=NIL;
GOSUB. = TRANSLA.(=(DO IF GOTRACE THEN PRINT.(=GOTO,=X1),X1 END));
GOSUB2.= TRANSLA.(=(DO IF GOTRACE THEN
                  PRINT.(=GOTO,RH) ELSE RH END));
PRSUB. = TRANSLA.(=(DO IF PRCTRACE THEN DO PRINT.(=NAME,=ENTERED),
                  ARGPR.(ARGS) END, VAL=BOD, IF PRCTRACE THEN
                  PRINT.(=NAME,
      =RETURNED,VAL), VAL END));
***  THIS IS A FIX FOR PROBLEM OF $ IN ARGLIST --- ,$A,$B,C)
NOOP $ = DUMMY.;
ARGPR.=PROC(),BEGIN(I),
      FOR I=(1,NUMARGS()) REPEAT PRINT.(=ARG,I,=IS,ARGUMENT(I))
END   END;
DEBUG=PROC(),
     DO GOFLAG.=TRUE,PRCFLAG.=TRUE,
       GOTRACE=TRUE, PRCTRACE=TRUE
END   END;
NODEBUG=PROC(),
     DO GOFLAG.=NIL, PRCFLAG.=NIL, GOTRACE=NIL, PRCTRACE=NIL END END ;
MGO.= PROC(L), BEGIN(RHS,BD),
     IF ¬GOFLAG. THEN RETURN EXLIS.(L),
     RHS = HD TL L,
     IF PAIRQ(RHS) THEN BD = SUBST.(RHS,=RH,GOSUB2.)
       ELSE BD = SUBST.(RHS,=X1,GOSUB.),
     HD TL L = BD,
     RETURN EXLIS.(L),
()   END   END;
MEQUAL.= PROC(L),BEGIN(RHS,LP,AR,BODY,BD,G1),
     RHS = HD TL TL L,
     IF ¬PAIRQ(RHS) THEN RETURN EXLIS.(L),
     IF PAIRQ(RHS) AND HD RHS NE =PROC THEN RETURN EXLIS.(L),
     IF ¬PRCFLAG. THEN RETURN EXLIS.(L),
     G1 = GENSYM(),
     LP = HD TL RHS,
     AR = HD TL LP,
     BODY = HD TL TL LP,
     BD = SUBST.(HD TL L,=NAME,PRSUB.),
     BD = SUBST.(G1,=VAL,BD),
     BD = SUBST.(AR,=ARGS,BD),
     HD TL TL LP = SUBST.(BODY,=BOD,BD),
     RETURN EXLIS.(L),
()   END   END;
```

```
MACDEF,(==,MEQUAL,);
MACDEF,(=GO,MGO,);
MAKPROPS(=MACRO,=MACROLI,);

COMMENT ≠ CHANGE PERIOD SO THAT IT IS NO LONGER ACCEPTABLE IN
             AN IDENTIFIER   ≠
CHGCHAR(=,,11);
MAKVLOCA(NIL);
MAKALOCA(NIL);
DO
     SAVEBALM(≠BLM4SVD≠), PRINT(≠BALM4.2.2≠)
END;
STOP;
```

CROSS REFERENCE

A	125 126 128 130 131 1008 1010 1012 1013 1016 1017 1494 1495 1497
ADDON	1679
ADDON.	28 111 161 461 477 901 1679
AFTER	897 1073 1157
AN	849
ANALYSE.	6 66 732 740 765 782 785
APPLY	366
AR	1719 1726 1730
ARE	1444
ARG	989 1170 1171 1172 1174 1241 1243 1244 1472 1610 1703
ARGL	946 949 989 990 991 992
ARGLIST.	990 1001 1253 1343 1349 1355
ARGPR.	1697 1702
ARGS	20 25 29 30 31 891 894 902 903 904 960 1276 1697 1730
ARGUMENT	387 570 573 1023 1444 1445 1447 1448 1449 1450 1703
ASSIGN.	1251 1268 1273 1314 1318
ASS.	34 907 960 961 962 963 969 970 974 976 977 978 981 984 995 996 998 1020 1041 1048 1049 1050 1059 1060 1061 1079 1092 1095 1098 1105 1112 1113 1115 1122 1123 1130 1133 1134 1137 1150 1163 1172 1174 1183 1193 1212 1216 1218 1226 1227 1228 1234 1235 1236 1276 1277 1278 1287 1292 1293 1295 1313 1315 1317 1320 1322 1323 1344 1350 1356
ATM	957 960 961 962 963 1268 1270 1272 1276 1277 1278
B	158 159 160
BACKSPAC	382
BALM	55
BD	1711 1714 1715 1716 1719 1728 1729 1730 1731
BE	1437
BEFORE	776
BINARY	910
BINFILE	39 43
BLANKLI.	257 258 582 584
BLANK.	215 254 258 419 494 546 550 588 625 626 633 639 647 689 1487 1490
BOD	1697 1731
BODY	1719 1727 1731
BOOTSTRA	3 44
BPCNT	558 561 566 570 578 594 609 611 618 626
BRACKET	720
BRACKETS	475
BRCKT	265 266 267 721 817
BREAKUP	1679
BREAKUP.	1364 1372 1373 1378 1679
BSCODEGE	12 17
BUILD.	1549 1570 1572
B1	702 705 711 712 714 715 717 718 720 721 830 832 833 834 837 840 841 842 843 846 849 857 860 861 864 865 866 867 870 874 875 876
B2	711 712 714 715 717 718 720 721 837 841 842

B3	714 715 717 718 720 721
B4	714 715
C	482 493 494 496 498 504 512 514 517 532 534 656 658 661 664
CALLS.	954 988
CANNOT	1437
CFROMV	375 928
CGCHECK.	25 894 1070 1155 1283 1303 1326 1338
CGCHKL.	1333 1335 1338
CHARS	510
CHAR.	144 240 241 245 254 496 500 505 507 512 532 1585 1677
CHGCHAR	1 1676 1740
CHR	1676 1677
CLAUSE	1196
CLOSE	1409 1647
CODE	910 941
CODEG	1597
CODEGEN	1679
CODEGEN.	82 87 886 1039 1463 1529 1600 1679
CODEQ	88 135 367 603 1436 1667
CODEV	21 890 918 920 923 924 925 928
CODGENL.	327 951 1597 1603 1608
COLON.	232 300 356
COMCHAR.	239 754
COMMA.	25 85 230 294 834 894 1005 1070 1301 1307
COMP	40
COMPILE	1680
COMPILED	941
COMPILE.	930 1599 1680
COMPL	364 1667
COMP.	32 905 945 992 997 1092 1150 1160 1163 1174 1202 1212 1215 1216 1227 1228 1235 1236 1251 1254 1255 1257 1259 1261 1290 1294 1312 1319 1343 1349 1355
CONCAT	362 550 1660
CONCATV	361 521 527 1013 1660
COND	1064 1065 1140 1141
CONSTRUC	1477 1480 1481
COPIED	1437
COPY	1429 1432 1439
CURCOM.	63 87 88 89
DEBUG	1705
DNUM.	73 146 149 243 245 247 404 675 679 706 1587 1589
DOF	749 756 768 808
DOF1	748 751
DOLLAR.	30 229 282 903 1078 1113 1132
DONT	459 475
DUMMY	1680
DUMMY.	321 879 1680 1701
E	158 159 161 482 496 497 499 508 510 511 514 516 518 521 527 531 532 534 537 858 859 865 871 873 875 1154 1155 1162 1163 1208 1210 1212 1216 1217 1282 1283 1294 1300 1302 1319
EMPTY	405

ENDFIL	743 750 799 827
ENDFILE	385 1671
ENDFLAG	729 731 732 735
END2	18 25 26 28 888 894 895 901 1023
ENTERED	1696
EOF	695 743 750 799
EQSIGN.	79 231 299 333 761 1272 1301
EQSTR	1386 1403 1661
EQUAL	1680
EQUAL.	1371 1388 1392 1399 1566 1680
EQUIV.	234 312
ERCOUNT.	5 7 10 14 63 65 70 78 91 704 707 773 777 813 848
ERMSG.	14 27 40 510 539 692 772 776 813 827 849 897 930 941 1073 1157 1196 1204 1264 1503 1514
ERR	7 10 14 70 78 91 899 930
ERRCNT	18 22 39 40 888 892 898 917 930 940 1040 1074 1158 1196 1204 1264 1271 1285 1305
ERROR	27 539 1273 1285 1305 1638
ERRORS	14 40 91 930
EXCHANG.	31 33 904 906 1046 1051 1061 1065 1066
EXCH1.	1044 1065
EXCH2.	1054 1066
EXECUTE	58 62 1640
EXLIS.	870 1523 1528 1712 1717 1721 1722 1723 1732
EXPAND	1681
EXPANDED	76
EXPAND.	9 74 707 857 865 875 1525 1681
EXPECT	1204
EXPON.	236 314 377
EXPX	20 25 32 891 894 905
EX.	1525 1527 1528 1549
E.	1523 1524 1550
FALSE	399 1558 1559 1568
FIL	423 425 428 431 433 435 557 559 562 566 568 569 571
FILE	827
FL	1335 1336 1337 1338
FN	416 420 424 425 432 433 544 547 550 558 559 566 569 578 946 949 950 951 989 990 994 997
FNOTN.	705 727 752 788 791
FOREVER	1515
FORL	1300 1316 1322
FORM	1326 1327 1328 1330 1332 1333
G	1459 1460 1462
GAND	331 1224
GARBCOLL	383 1646
GCOND	330 1178
GCON.	948 967 974 1244
GELSE.	1185 1189 1201
GENLBL.	1029 1081 1180 1215 1226 1234 1288 1291 1311 1316
GENNAM.	1580 1583
GENP	1656 1659 1660 1661 1662 1663 1666 1667 1668 1669 1670 1671 1672 1673
GENR	946 951 952

GENSYM	1681 1724
GENSYMB.	144 146 147 149 404 1535 1547 1563 1576
GENSYM.	141 984 1038 1460 1529 1600 1681
GETLIST.	444 451
GETNEXT.	730 734
GETPROP	1484
GETVECT.	445 465
GETV.	466 468
GFOR	334 1299
GGO	330 1169
GLAMBDA	328 1036
GLIST	335 1341
GLOBAL	38 908
GLOBL	19 22 38 889 892 908 1016
GOFLAG.	1689 1706 1710 1712
GOR	331 1232
GOSUB2.	1694 1714
GOSUB.	1693 1715
GOTRACE	1692 1693 1694 1707 1710
GPROG	328 1068
GPROGN	329 1153
GQUOTE	333 1240
GREATER	510
GREFS.	28 901 963 981 1008 1041 1278
GRETURN	329 1148
GSETQ	333 1247
GSTRING	332 1353
GTHEN.	1182 1192 1199 1208
GVAR.	947 957
GVECTOR	335 1347
GWHILE	334 1281
G1	1545 1546 1577 1719 1724 1729
G10	1545 1546 1579
G2	1545 1546 1577
G3	1545 1546 1577
G4	1545 1546 1577
G5	1545 1546 1578
G6	1545 1546 1578
G7	1545 1546 1578
G8	1545 1546 1578
G9	1545 1546 1578
HAVE	1204
I	21 100 105 106 128 130 131 142 143 144 146 149 165 167 181 183 213 241 244 245 254 258 417 419 566 570 577 580 584 607 610 630 632 651 890 911 919 920 922 923 924 925 1021 1023 1044 1046 1048 1049 1051 1054 1059 1060 1061 1101 1103 1105 1106 1109 1114 1115 1118 1120 1122 1123 1126 1130 1133 1136 1137 1300 1301 1314 1318 1365 1378 1379 1389 1399 1430 1432 1443 1445 1467 1472 1489 1490 1584 1585 1587 1589 1650 1651 1653 1654 1702 1703
ID	1410 1412 1413 1422 1425 1426 1427 1458 1460 1461 1463 1484 1485 1613 1615 1617 1624 1629 1630 1632 1633 1635

```
          1636
IDFROMC   366 646 1471 1670
IDFROMS   147 221 222 223 224 225 226 227 228 229 230 231 232
          233 234 235 236 237 238 239 250 252 374 406 502 504
          508 1013 1587 1669
IDNIL.    252 437 447 601
IDQ       81 117 368 598 947 950 979 1270 1327 1366 1478 1619
          1623 1668
IDTRUE.   250 436 446 600
IEOS.     6 66 238 292 457 473 745 773 779 796 811
IFROMID   1681
IFROMID.  138 920 1681
IMPROPER  772
IN        539 930 1514
INDEX     359 782 791 1256
INEG      279 376
INFIX     292 293 294 295 296 297 298 299 300 301 302 303 304
          305 306 307 308 309 310 311 312 313 314 315 714 715
          1592 1594
INFIXLI.  291 715 770 803 1594 1603 1607 1633
INITCOD.  197 325
INITEXP.  196 319
INITIAT.  57 191
INITINF.  195 289
INITIO.   193 203
INITMIS.  199 397
INITOPL.  198 353
INITUNA.  194 262
INPUT     58 62 255 742 749 756 759 762 779 805 827 1640
INSTR     1598
INTQ      367 599 972 995 1667
INVALID   941 1610
IS        849 1454 1610 1703
ITM       424 427 428 432 435 436 437 438 442 443 444 445 446
          447 448 452 454 455 456 457 461 469 471 472 473 477
I1        741 742 743 744 745 749 750 751 752 754 758 759 761
          762 764 765 767 770 772 773 775 782 785 788 791 794
          808 815 818 821 824
I2        741 779 780 784 787 788 790 791 793 796 799 800 803
          805 818
J         142 144 146 482 495 502 508 509 517 521 527 530 539
          577 579 580 582 584 1300 1301 1312 1584 1585 1587 1650
          1651 1653 1654
K         1300 1301 1307 1309 1312 1650 1651 1653 1654
KARG.     339 960 1048 1059
KASTORE.  340 1049 1060 1276
KCALL.    337 998
KGLOB.    339 963 984 1041 1048 1059 1105 1112 1113
KGSTORE.  340 1050 1060 1122 1130 1134 1278
KINEG.    347 974
KJMPF.    343 1216 1227 1292
KJMPI.    344 1174
KJMPT.    343 1212 1235
```

```
KJMP.      343 1150 1172 1218 1295 1315
KLBL.      344 962
KLIST.     345 1344
KMAKVAR,   347 981
KNIL.      342 969 1183 1193 1226 1287 1313
KNUM1.     341 976
KNUM2.     341 977 981
KNUM3.     341 978
KNVARS,    338 1079
KPOP.      338 1049 1050 1060 1061 1123 1137 1163 1228 1236 1293
           1317 1323
KRETPRO.   338 1098
KRTPROC,   34 337 907
KSETSTK,   338 1092
KSETSX,    349 1095 1150
KSTEPLO,   346 1320
KSTRING,   348 1356
KTLOOP.    346 1322
KTRUE.     342 970 1234
KVAR,      339 961 1122 1130 1133
KVECTOR.   345 1350
KVSTORE,   340 1105 1115 1277
L          99 101 106 114 117 118 120 121 122 125 126 128 130 131
           157 160 161 170 172 173 175 181 182 183 184 213 253
           254 557 561 562 590 592 593 595 597 598 599 600 601
           602 603 606 609 610 615 617 621 622 625 636 639 642
           646 692 693 727 728 730 858 859 867 871 873 874 1001
           1002 1003 1004 1005 1006 1044 1045 1046 1048 1050 1051
           1054 1056 1057 1058 1061 1101 1102 1103 1105 1106 1109
           1111 1112 1113 1114 1115 1118 1119 1120 1122 1123 1126
           1128 1129 1130 1132 1134 1136 1137 1153 1155 1160 1161
           1162 1165 1225 1226 1227 1229 1233 1234 1235 1237 1300
           1301 1303 1307 1309 1312 1411 1412 1413 1414 1416 1418
           1419 1420 1430 1431 1432 1466 1469 1471 1473 1494 1495
           1496 1497 1499 1501 1519 1548 1553 1554 1556 1557 1560
           1561 1562 1563 1564 1566 1567 1613 1614 1616 1617 1618
           1619 1620 1621 1623 1624 1711 1712 1713 1716 1717 1719
           1720 1721 1722 1723 1728 1732
LABEL      913
LAND       363 679 1661
LAST       1179 1180 1186 1218 1300 1311 1315 1321
LBLIST     20 24 891 893 962 1069 1076 1085 1172 1210
LBLNO      19 22 889 892 1030
LBLQ       368 602 1668
LBLVALS    19 23 889 892 913 938 1027
LBL.       1026 1090 1096 1186 1219 1229 1237 1289 1296 1316 1321
LBR,       217 223 608
LBVAR.     223 445 780
LENGTH     1682
LENGTH,    133 134 998 1079 1344 1350 1356 1682
LETTB.     249 516 536 678
LFROMV     1682
LFROMV,    180 791 1682
```

```
LHS        1248 1249 1250 1251 1253 1264 1503
LHSARGS    1248 1253 1254 1255 1257 1259 1261 1262
LHSOP      1248 1253 1254 1255 1256 1258 1260
LIN        213 214 215 216 217 218 219 220 227 229 230 231 232
           233 234 235 236 237 238 239 242 245 248 249 250 252
           253 254 417 419 420 424 425 428 432 433 435 493 500
           502 505 507 508 512 516 521 524 527 529 532 536 544
           547 550 551 553 558 559 562 566 569 571 578 582 584
           653 658 664 1489 1490 1491
LIST       14 40 85 91 264 265 266 267 268 269 270 271 272 273
           274 275 276 277 278 279 280 281 282 283 284 291 292
           293 294 295 296 297 298 299 300 301 302 303 304 305
           306 307 308 309 310 311 312 313 314 315 321 327 328
           329 330 331 332 333 334 335 355 356 357 358 359 360
           361 362 363 364 365 366 367 368 369 370 371 372 373
           374 375 376 377 378 379 380 381 382 383 384 385 386
           387 388 389 390 391 392 393 459 475 712 715 718 721
           782 785 788 791 794 818 821 824 849 913 962 1150 1172
           1212 1216 1218 1227 1235 1254 1255 1257 1259 1261 1292
           1295 1315 1322 1386 1387 1408 1409 1413 1426 1427 1514
           1610 1626
LISTID     1612 1614 1620 1626
LIST2      18 28 35 36 37 888 901 911 914 915
LIST3      18 35 39 888 914 920
LLEN       416 419 420 424 425 428 432 433 435 489 490 500 505
           507 512 516 519 521 522 529 532 553 558 559 566 569
           580 588 594 608 638 644 652 663 670 1488 1490 1491
LOGQ       369 1668
LOOKUP     1682
LOOKUP.    114 767 770 803 862 938 951 994 1485 1521 1526 1682
LOR        364 534 1661
LP         1719 1725 1726 1727 1731
LPAR.      216 221 595
LPVAR.     221 444 764 784
LS         3 6 8 9 11 63 74 76 79 80 81 82 85 87 1500 1501 1502
           1503 1506 1513 1519 1533 1538 1539
LST        729 730 735 737
LXSCAN.    435 443 481
M          858 862 863 864 1500 1519 1520 1523 1524 1525 1527 1528
           1529
MACDEF     1683
MACDEF.    711 1407 1529 1593 1683 1734 1735
MACRO      1407 1596 1736
MACROLI.   321 712 862 1521 1526 1596 1603 1607 1630 1736
MACSYMB.   73 706 1585 1587 1589
MAKALOCA   2 408 1064 1742
MAKFILE    43 255 256 416 1488
MAKO       517 532 671 678
MAKPROPS   1410 1594 1595 1596 1597 1598 1736
MAKVECTO   104 166 240 243 257 358 419 518 918 1431 1490 1667
MAKVLOCA   2 407 1140 1741
MAPL.      153 157
MAPV.      154 164
```

MAPX	1683
MAPX,	152 996 1343 1349 1355 1683
MATCH.	459 475 1539 1548 1553 1556 1557
MEMBER	1683
MEMBER,	170 962 1010 1016 1172 1210 1683
MEQUAL,	1719 1734
MGO,	1711 1735
MINUS.	228 279 309 376
MISSING	776 813 897 1073 1157
MMATCH,	1513 1533 1538
MMEANS	1499 1593
MODE	380
MOR	4 13 15 64 83 88 90 92 117 122 160 161 172 175 454 462 471 478 675 676 874 876 1390 1394 1616 1624
MORE	1282 1288 1289 1295
MORO	679 680 686
MSIGN,	216 228 673
MSTR	519 525 530
MTY,	85 405 744 746 777 794 833 971 1003 1162
N	100 101 103 104 105 165 166 167 181 182 183 566 568 570 573 607 609 610 630 631 632 651 652 653 667 672 673 678 679 680 682 968 984 1037 1038 1039 1040 1041 1443 1445 1446 1447 1448 1449 1467 1471 1472
NAM	3 11 12 17 28 886 887 901 930 932 1488 1491
NAME	1696 1698 1728
NBYTE	19 22 889 892 918 919 922 1022 1027
NDF,	406 491 522 696
NEWP	1459 1461 1462 1463
NEXT	424 425 428 432 433 435 458 474 489 493 495 500 501 502 505 506 507 508 509 512 514 516 517 519 524 525 527 528 529 530 532 534 536 537 539 554 558 559 562 566 569 571 578 588 594 608 609 611 618 626 638 639 644 646 652 653 658 663 664 670
NILQ	274 275 276 372 1669
NN	668 672 673 675 676 678 680
NO	941
NODEBUG	1709
NOOP	758 1701
NOTM	861 863 865 867
NOTSIGN,	227 274
NOTU	752 759 762 765 779 782 785 788 791 794 818
NTRUE	1209 1215 1216 1219 1282 1291 1292 1296
NUM	1676 1677
NUMARGS	387 568 1022 1023 1443 1703
NUMB	497 512
NUMBER	539
NXT	489 494 736 737
OCTAL	539
OCTMODE	403 671 911
OF	772 827 1444 1454 1514
OLDPR	1446 1447 1448 1449 1462
ON	827 1626
OP	858 861 862 989 994 995 996 1189 1191 1200 1202 1204

```
          1500 1506 1514 1521 1529 1612 1615 1619 1626
OPEN      1409 1491
OPERAND   776
OPERATOR  849
OPERROR,  805 846
OPLIST,   355 994 1386 1387 1408 1409 1598 1603 1608
ORDINAL   1684
ORDINAL,  125 960 961 1276 1277 1684
ORD1,     126 128 131
OUTPUT    58 62 256 546 550 568 1640
P         115 118 152 153 154 157 161 164 167 741 742 768 807
          808 810 817 818 820 821 824 1208 1212 1215 1282 1283
          1290 1364 1366 1367 1369 1371 1372 1373 1375 1378 1379
          1380 1383 1410 1419 1422 1425 1426 1427 1477 1478 1479
          1480 1481 1482 1484 1485 1613 1614 1618 1619 1621 1624
PAIRQ     79 120 130 134 153 160 172 369 593 621 728 735 751 787
          840 860 861 867 936 948 1004 1046 1057 1085 1090 1103
          1112 1120 1129 1195 1250 1307 1330 1331 1337 1369 1370
          1392 1415 1418 1426 1439 1479 1496 1502 1508 1536 1537
          1554 1555 1571 1616 1618 1668 1714 1721 1722
PARENS    459
PERIOD,   215 225 625
PERVAR,   225 456
PLL       797 801 810 826
PLUS,     233 310 376
PNAME     1444 1454 1461
PR        3 11 12
PRCFLAG,  1690 1706 1710 1723
PRCTRACE  1691 1696 1697 1707 1710
PREVAL    1413
PREVCOM,  63 92
PREVM     1500 1521 1522 1524 1526 1527
PRINT     1684 1744
PRINT,    4 38 64 68 76 91 459 475 565 693 908 910 911 913 1272
          1273 1285 1305 1437 1444 1445 1452 1454 1468 1471 1472
          1610 1626 1684 1693 1695 1696 1698 1703
PROCTRAC  1466
PROGRAM   1069 1070 1077 1082 1083 1084 1087 1088 1089
PROMPT,   220 546
PROPL     118 378 1413 1426 1485 1624
PROTECT   385 1671
PRSUB,    1696 1728
PRTR,     1442 1461
PTRACE    1458
PUTBLAN,  587 612 619 653
PUTCHK,   625 626 632 633 661
PUTCHV,   598 600 601 604 645 646 650
PUTCH,    588 595 608 611 618 626 638 639 645 647 656 673 688
          689
PUTCODE   603 642
PUTINT,   599 639 667
PUTITEM,  561 570 590 610 622 625
PUTLBL,   602 636
```

SAVEBALM	1644 1744
SAVLOCS,	1080 1103 1106 1114 1115 1142 1145
SAVL1.	1101 1142
SAVL2.	1109 1145
SAVSTAT	1602
SEMIC.	219
SETINDEX	359 1257
SETMODE	380
SETPROPL	378 1413 1426 1427 1619
SETPROPY	1419 1425
SETSUB	362
SETSUBV	360 582 584 653 1261
SETVALUE	379 1259 1422 1460
SEVEN.	247 683
SFROMID	374 405 598 600 601 645 646 731 1012 1669 1677
SFROMV	147 221 222 223 224 225 226 227 228 229 230 231 232
	233 234 235 236 237 238 239 250 252 373 406 502 504
	508 531 578 1013 1587 1669
SHIFT	363 534 679 1660
SLASHX.	218 406 638 639 645 647
SLASH.	235 313 377
ST	63 66 68 71 72 74 92
START	544 553
STARX.	226 311 377 693
STAR.	73 215 226 404 553 604 706 1013
STKTRACE	1408 1671
STOP	8 59 72 393 827 1359 1360 1640 1642 1675 1746
STR	498 518
STRING	332 365 546 550
STRQ	135 367 597 1435 1667
STRQU.	216 498 524 529 632 633
SUB	362 1260 1435 1650 1651
SUBLBLS.	37 915 935
SUBST	1685
SUBST.	1461 1462 1494 1497 1519 1520 1523 1524 1525 1527 1528
	1685 1714 1715 1728 1729 1730 1731
SUBV	214 242 248 360 502 508 521 527 1260 1653 1654
SVARS	1069 1080 1097
SYMB	496 505
SYNTAX	14 68
SYSLIST	400 1010
TALKATIV	67 75 402 908 909 913
TERM	740 744 775 776 800 811 813
TERMFLA.	5 63 65 729 731 746 773 779 797 811
TERMLINE	424 425 432 433 490 521 558 560 561 566 568 570 588
	594 608 638 644 652 663 670
TEST3	807 826
TEST4	796 805
TH	1189 1192 1195 1196 1199
TIME	386 1674
TMAC.	1519 1543
TR	566 567 572 1335 1336 1337 1338
TRACE	567 572

```
TRANSLAT   1685
TRANSLA.   702 1442 1543 1600 1685 1693 1694 1696
TREE       68 76 941 1326 1328 1330 1331 1332 1333
TTYFLAG    401 545
TYPE       1638 1639
T1         1069 1084 1085 1089 1090 1092
U          741 767 768 797 801 803 807 808 826
UNARY      268 269 270 271 272 273 274 275 276 277 278 279 280
           281 282 283 284 717 718 820 1595
UNARYLI.   264 718 721 767 1595 1603 1607 1636
USE        772 1514
V          100 101 102 103 104 106 108 164 166 167 180 182 183
           1425 1426 1427 1430 1431 1432 1433 1467 1471 1472
VAL        1272 1697 1699 1729
VALID      1196 1264 1503
VARS       20 24 38 891 893 908 961 1069 1070 1077 1078 1079 1080
           1277
VBLANK,    1487
VECTOR     73 221 222 223 224 225 226 227 228 229 230 231 232 233
           234 235 236 237 238 239 250 252 335 358 404 406 420
           504 604 706 1013 1487 1491 1603
VECTQ      135 154 368 592 790 1375 1376 1397 1430 1606 1668
VFROML     1686
VFROML,    99 466 1686
VFROMS     214 242 248 373 544 547 551 598 600 601 631 645 646
           1012 1669 1677
VV         165 166 167
WARNING    510
WILL       1514
WRITE      39 557
WRITOUT,   560 568 576
WRLINE     381 546 550 578 1659
X          17 25 111 112 114 117 118 121 133 134 135 138 139 152
           153 154 155 170 173 452 453 456 461 469 470 477 695
           696 703 705 707 708 879 880 886 894 935 936 938 941
           945 947 948 949 952 954 967 969 970 971 972 973 974
           976 977 978 979 981 984 988 990 1026 1027 1036 1039
           1055 1057 1058 1059 1060 1068 1070 1082 1087 1148 1150
           1169 1171 1178 1180 1181 1182 1185 1224 1227 1228 1232
           1235 1236 1240 1243 1247 1249 1269 1276 1277 1281 1283
           1299 1303 1341 1343 1344 1347 1349 1350 1353 1355 1356
           1364 1367 1370 1371 1372 1373 1376 1378 1379 1380 1383
           1388 1390 1391 1392 1394 1397 1399 1403 1429 1430 1431
           1432 1435 1436 1437 1439 1440 1494 1495 1497 1534 1537
           1538 1539 1599 1600 1605 1606 1607 1608 1656 1657 1663
           1664
XOR        365 1661
X1         1510 1534 1535 1544 1547 1560 1573 1656 1657 1663 1664
           1693 1715
X10        1512 1534 1535 1544 1547 1562 1575
X2         1510 1534 1535 1544 1547 1560 1573
X3         1510 1534 1535 1544 1547 1560 1573
X4         1510 1534 1535 1544 1547 1561 1574
```