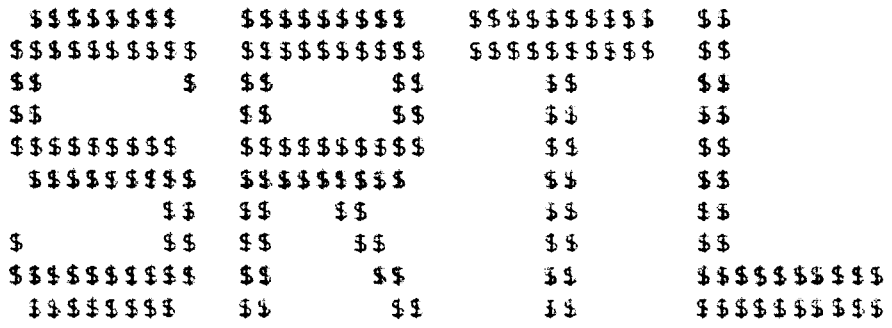
SETL NEWSLETTER NUMBER 73

REVISION 3, MAY 16, 1973

MAY 25, 1972
KURT MALY
HANK WARREN



USER≠S GUIDE TO THE SETL RUN TIME LIBRARY

TABLE OF CONTENTS

1. INTRODUCTION

THE SETL RUN TIME LIBRARY IS A COLLECTION OF SUBROUTINES AND MACROS, WRITTEN IN THE LITTLE LANGUAGE, THAT ARE CAPABLE OF EVALUATING SETL EXPRESSIONS. SUBROUTINES ARE PROVIDED TO BUILD SETS, TEST SETL OBJECTS FOR EQUALITY, CONCATENATE STRINGS, ETC. THE LIBRARY INCLUDES A STORAGE ALLOCATION MECHANISM THAT IS BASED ON A COMPACTING GARBAGE COLLECTOR.

ALTHOUGH THE SETL RUN TIME LIBRARY IS DESIGNED PURELY FOR THE SETL IMPLEMENTATION, IT MAY ALSO BE OF VALUE IN WRITING LITTLE PROGRAMS THAT HAVE NOTHING TO DO WITH SETL. IT IS TOWARD THIS END THAT THIS MANUAL IS DIRECTED. HOWEVER, THE MATERIAL IS ALSO A USEFUL INTRODUCTION FOR THOSE INVOLVED IN MODIFYING OR ADDING TO THE RUN TIME LIBRARY.

THERE ARE THREE LEVELS AT WHICH THE SETL RUN TIME LIBRARY MIGHT BE USED:

1. THE MACRO LEVEL.
2. THE STORAGE MANAGEMENT LEVEL.
3. THE SUBROUTINE LEVEL.

A HIGHER LEVEL INTRODUCES MORE RESTRICTIONS ON THE DATA STRUCTURES PERMITTED, BUT PROVIDES A MORE SIGNIFICANT PROGRAMMING AID.

2. USE OF SRTL GENERAL PURPOSE MACROS

THERE ARE A FEW MACROS IN THE SETL RUN TIME LIBRARY THAT ARE OF GENERAL INTEREST. THESE MAY BE USED WITHOUT IMPOSING ANY RESTRICTIONS ON DATA STRUCTURES.

THERE IS A GROUP OF MACROS RELATED TO THE COMPUTER≠S WORD SIZE THAT ARE FREQUENTLY NEEDED WHEN CODING IN LITTLE. THESE ARE LISTED BELOW.

MISCELLANEOUS MACROS RELATING TO WORD SIZE

```
+*WS = 60**     /* WORD SIZE IN BITS. */
+*WSP1 = 61**   /* WS + 1. */
+*WSM1 = 59**   /* WS - 1. */
+*WSD2 = 30**   /* WS/2. */
+*WSD2P1 = 31**  /* WS/2 + 1. */
+*PTS = 17**    /* POINTER SIZE IN BITS. */
+*PTSM1 = 16** /* PTS - 1. */
+*PTSP1 = 18** /* PTS + 1. */
+*CS = 6**      /* CHARACTER SIZE IN BITS. */
+*CSP1 = 7**    /* CS + 1. */
+*CPW = 10**    /* CHARACTERS PER WORD. */
+*WSMCS = 54** /* WORD SIZE - CHARACTER SIZE.  */
+*WSMCSP1 = 55**     /* WSMCS + 1. */
+*CPWM1 = 9**  /* CPW - 1. */
+*REMWS(X) = (X) - ((X)/WS)*WS**  /* REMAINDER. */
+*QREMWS(X,Q,R) = Q = (X)/WS;  R = (X) - Q*WS**
          /* COMPUTES QUOTIENT AND REMAINDER MOD WS. */
+*REMCPW(X) = (X) - ((X)/CPW)*CPW**  /* REMAINDER. */
+*QREMCPW(X,Q,R) = Q = (X)/CPW;  R = (X) - Q*CPW**
          /* COMPUTES QUOTIENT AND REMAINDER MOD CPW.*/
```

SOME MACROS THAT ARE NEEDED WHEN CODING NESTED MACRO DEFINITIONS ARE:

```
+*Q3(A,B,C) = A B C**
+*MACDEF(TEXT) = Q3(+,*TEXT*,*)**
+*MACDROP(MNAME) = MACDEF(MNAME=)**
```

FOR AN EXAMPLE OF THE USE OF THESE MACROS, SEE THE THEN AND ELSE MACROS BELOW.

LITTLE DOES NOT PROVIDE A FACILITY FOR PRESETTING VARIOUS FIELDS OF A WORD WITH A DATA STATEMENT. TO MORE-OR-LESS SIMULATE THIS CAPABILITY, THE ≠SETWORD≠ FAMILY OF MACROS IS PROVIDED. THE FIRST TWO ARE:

MACROS FOR PRESETTING WORDS WITH FIELDS

```
+*SETWORD1(WORD,FIELD,VALUE) = SIZE WORD(WS);  DATA WORD=0;
     FIELD WORD = VALUE**
+*SETWORD2(WORD,F1,V1,F2,V2) = SIZE WORD(WS);  DATA WORD=0;
     F1 WORD = V1;  F2 WORD = V2**
```

THERE ARE PRESENTLY THREE SUCH MACROS. NOTE THAT THEY EXPAND INTO EXECUTABLE CODE, AND HENCE THEIR CALLS MUST BE PLACED IN THE INSTRUCTION STREAM. THE CALL MUST BE FOLLOWED BY THE STATEMENT TERMINATOR, E.G.

```
SETWORD2(MASK, FIELD1, 15, FIELD2, 15);
```

HERE IT IS ASSUMED THAT FIELD1 AND FIELD2 ARE MACROS, E.G. +*FIELD1 = .F.57,4,**. THE SETWORD MACROS ARE AWKWARD TO USE WITH ≠ABSOLUTE≠ FIELD DESCRIPTORS:

```
SETWORD1(MASK, .F.57 COMMA 4 COMMA, 15);
```

WHERE +*COMMA = ,**. HOWEVER, ABSOLUTE FIELD DESCRIPTORS ARE SELDOM USED.

THE MOST SIGNIFICANT MACROS OF GENERAL INTEREST IN SRTL ARE THE FLOW CONTROL MACROS. THESE IN EFFECT EXPAND LITTLE SO THAT IT INCLUDES THREE TYPES OF FLOW OF CONTROL IN ADDITION TO THE GO TO, GOBY, AND IF(C) GO TO THAT ARE BUILT INTO THE LANGUAGE. THESE ARE THE

DO ITERATION,
WHILE ITERATION, AND
IF-THEN-ELSE.

SOME EXAMPLES OF THE USE OF THESE:

```
DOM(I, 1, MAX);      /* CLEAR ARRAY. */
  ARRAY(I) = 0;
  EDOM;

P = LISTHEAD;
WHILE (P .NE. 0);    /* SEARCH LIST. */
  IF (C(P)) THEN QUITWHILE(P);  ENDIF;
  P = NEXT(P);
  ENDWHILE(P .NE. 0);
```

MORE GENERALLY:

```
DOM(I, LOW, HIGH);   I IS THE ITERATION INDEX,
    .                WHICH RANGES FROM LOW
    .                TO HIGH, INCLUSIVELY. THE
    .                INCREMENT IS FIXED AT 1.

  CONTDO;            THIS CAUSES INCREMENTATION OF
    .                I AND REPETITION OF THE
    .                ITERATION IF THE NEW VALUE
    .                OF I ≤ HIGH.
    .

  QUITDO;            THIS CAUSES A BRANCH TO
    .                THE STATEMENT FOLLOWING THE EDOM
    .                BELOW.  I IS NOT ALTERED.
    .
  EDOM;              THIS SIGNIFIES THE END OF THE
                     SCOPE OF THE DO ITERATION.
```

THE ITERATION VARIABLE I MAY BE ANY ≠VALUE RECEIVING≠ EXPRESSION, E.G. .F.1,8,A(K). LOW AND HIGH MAY BE EXPRESSIONS. HIGH IS EVALUATED EACH TIME THROUGH THE LOOP, AND HENCE FOR EFFICIENCY REASONS IT WOULD USUALLY BE SIMPLY A VARIABLE NAME.

IF HIGH < LOW, THE LOOP IS NOT EXECUTED, BUT I IS SET TO LOW. ON NORMAL FALL-THROUGH, I IS SET TO HIGH + 1. ON EXIT VIA QUITDO, I IS SET TO THE LAST USED VALUE. FOR OTHER DETAILS SUCH AS THE EFFECT OF CHANGING I, LOW, AND HIGH IN THE LOOP, REFER TO THE MACRO DEFINITIONS GIVEN BELOW.

```
+*DOM(J,LOW,HI)=J=LOW;MACDEF(DOITERVAR=J)
     MACDEF(DLOOPLABEL = ZZZA)  MACDEF(OLOOPLABEL = ZZZB)
     MACDEF(CLOOPLABEL = ZZZC)
     /DLOOPLABEL/ IF(J .GT. HI) GO TO OLOOPLABEL**
+*CONTDO = GO TO CLOOPLABEL**
+*QUITDO = GO TO OLOOPLABEL**
+*EDOM = ; /CLOOPLABEL/ DOITERVAR = DOITERVAR + 1;
     GO TO DLOOPLABEL;  /OLOOPLABEL/
     MACDROP(DOITERVAR)  MACDROP(DLOOPLABEL)
     MACDROP(OLOOPLABEL)  MACDROP(CLOOPLABEL)**
```

THE WHILE ITERATION IS SIMILAR TO THE DO:

```
WHILE(C);              C IS ANY CONDITIONAL EXPRESSION
    .                  (ANYTHING VALID IN AN IF).
    .
    .
  CONTWHILE(TEXT);     CAUSES A BRANCH BACK TO THE
    .                  TEST OF CONDITION C.  TEXT IS
    .                  ANYTHING NOT CONTAINING AN
    .                  EXPOSED COMMA.  IT IS NOT
    .                  MATCHED WITH C OR USED IN ANY
    .                  WAY, BUT SOMETHING MUST BE
    .                  PRESENT.
    .
  QUITWHILE(TEXT);     CAUSES A BRANCH TO THE
    .                  STATEMENT FOLLOWING ENDWHILE
    .                  BELOW.
    .
  ENDWHILE(TEXT);      SIGNIFIES THE END OF THE SCOPE
                       OF THE WHILE ITERATION (THIS IS
                       SIMPLY A BRANCH BACK TO THE
                       WHILE, AND SOME MACRO DROPS).
```

DO ITERATIONS MAY BE NESTED TO A DEPTH OF TWO, AND WHILE ITERATIONS MAY BE NESTED TO A DEPTH OF FOUR. THE INNER WHILE ITERATIONS MUST EMPLOY THE WORDS WHILE1, CONTWHILE1, QUITWHILE1, ENDWHILE1, WHILE2, ETC. FOR EXAMPLE:

```
WHILE (X .GT. 0);
  WHILE1 (Y .LT. X);
    ...
    CONTWHILE1(Y);   /* CONTINUE INNER ITERATION. */
    ...
    QUITWHILE(X);    /* EXIT FROM OUTER ITERATION. */
    ...
    ENDWHILE1(Y);
  ENDWHILE(X .GT. C);
```

THERE IS UNFORTUNATELY NO DOING CLAUSE FOR A WHILE ITERATION. THIS LIMITS THE UTILITY OF THE CONTWHILE, AS THE ITERATION STEPS MUST BE WRITTEN REPEATEDLY.

THE IF-THEN-ELSE IS WRITTEN AS FOLLOWS.

```
IF (C) THEN          C IS ANY CONDITIONAL
    .                EXPRESSION.
    .
    .
ELSE                 THE ELSE CLAUSE IS
    .                OPTIONAL.
    .
    .
  ENDIF;             SIGNIFIES END OF IF GROUP.
```

THE THEN AND ELSE CLAUSES CONTAIN A SEQUENCE OF EXECUTABLE STATEMENTS SEPARATED BY THE NORMAL STATEMENT TERMINATOR. THE THEN CLAUSE MUST BE PRESENT, BUT IT MAY BE A NULL STATEMENT. THE ELSE CLAUSE, INCLUDING THE WORD ELSE, MAY BE OMITTED. THE ENDIF IS REQUIRED.

THE THEN CLAUSE IS EXECUTED UNDER THE SAME CONDITIONS AS THE GO TO IN THE NORMAL LITTLE IF (C) GO TO STATEMENT, I.E., IF C EVALUATES TO A NONZERO BIT STRING. THE ELSE CLAUSE IS EXECUTED IF C EVALUATES TO ALL ZEROS.

UP TO FIVE LEVELS OF NESTING OF THE IF MAY BE ACHIEVED BY USING THE WORDS THEN1, ELSE1, ENDIF1, ..., ENDIF5. THE WORD IF NEVER HAS A DIGIT APPENDED. FOR EXAMPLE:

```
IF (C1) THEN
  IF (C2) THEN1 S1;  ELSE1 S2;  ENDIF1;
ELSE
  IF (C3) THEN1 S3;  ELSE1 S4;  ENDIF1;
  ENDIF;
```

THE THEN, ELSE, AND ENDIF MACRO DEFINITIONS FOLLOW.

```
++THEN = MACDEF(THENLABEL=ZZZA)  MACDEF(ELSELABEL=ZZZB)
     MACDEF(ENDIFLABEL=ZZZC)  GO TO THENLABEL;
     GO TO ELSELABEL;  /THENLABEL/;**
++ELSE = GO TO ENDIFLABEL;  /ELSELABEL/;
     MACDROP(ELSELABEL)  MACDEF(ELSELABEL = ZZZD)**
++ENDIF = /ELSELABEL/;  /ENDIFLABEL/  MACDROP(THENLABEL)
     MACDROP(ELSELABEL)  MACDROP(ENDIFLABEL)**
```

THE DO, WHILE, AND IF-THEN-ELSE CONTRUCTIONS MAY BE NESTED IN ONE ANOTHER, GIVING A MAXIMUM DEPTH OF THIRTEEN (3 + 5 + 6 - 1).

3. USE OF SRTL STORAGE MANAGEMENT

THE NEXT LEVEL OF USAGE OF THE SETL RUN TIME LIBRARY IS THE USE OF ITS STORAGE ALLOCATION ROUTINES AND ITS GARBAGE COLLECTOR. THE USE OF THESE FACILITIES IMPOSES A NUMBER OF RESTRICTIONS ON THE PERMISSIBLE DATA STRUCTURES.

3.1 THE STORAGE ARRAY

TO USE THE DYNAMIC STORAGE FACILITIES, A LARGE LINEAR ARRAY CALLED ≠STORAGE≠ MUST BE PROVIDED. ONE END OF THE ARRAY IS USED FOR THE RUN TIME STACK (WHICH MUST BE USED TO USE THE GARBAGE COLLECTOR), AND THE OTHER END IS USED FOR THE GENERAL STORAGE POOL OR ≠HEAP≠. WHEN ONE OF THESE PARTITIONS OVERLAPS THE OTHER, THE GARBAGE COLLECTOR IS INVOKED TO COMPACT THE HEAP.

THE TWO PARTITIONS OF THE STORAGE ARRAY ARE REFERRED TO BY THE VARIABLES STACK AND HEAP. THIS IS PARTLY FOR MNEMONIC VALUE AND PARTLY TO FACILITATE A CHANGE IN DESIGN IN WHICH THE STORAGE AREA IS PARTITIONED INTO TWO DISTINCT AREAS EACH OF FIXED SIZE (THIS CHANGE IS NOT CONTEMPLATED AT THIS TIME, HOWEVER).

THE SETL RUN TIME LIBRARY INCLUDES THE FOLLOWING DEFINITIONS PERTINENT TO THE STORAGE ARRAY:

```
+*HEAP = STORAGE**    /* LOCATED IN LOW INDEX END (FOR EASE
                           OF COMPACTION).                  */
+*STACK = STORAGE**   /* LOCATED IN HIGH INDEX END.       */
+*STORAGESZ = 1000** /* FAIRLY SMALL FOR DEBUGGING.  */
                      /* NOTE: WHEN CHANGING STORAGESZ, ALSO
                      CHANGE MACRO USEDWORDSZ BELOW (IN THE
                      ≠MACROS RELATED TO THE GARBAGE COLLECTOR≠
                      SECTION).  WHEN LITTLE ALLOWS EXPRESSIONS
                      INVOLVING ONLY CONSTANTS IN A DIMS STATE-
                      MENT, USEDWORDSZ SHOULD BE CHANGED TO
                      MAKE ITS SETTING BECOME AUTOMATIC. */
SIZE STORAGE(WS);  DIMS STORAGE(STORAGESZ);
```

HERE WS IS THE WORD SIZE MACRO MENTIONED IN SECTION 2.

3.2 DATA STRUCTURES FOR THE GARBAGE COLLECTOR

THE GARBAGE COLLECTOR IS NOT DEPENDENT UPON THE DATA TYPES USED IN THE SETL IMPLEMENTATION. IT DOES, HOWEVER, RELY ON CERTAIN WORD FORMATS IN THE STACK AND HEAP, AS FOLLOWS.

A WORD IN THE STACK MUST BE IN THE FOLLOWING FORMAT.

LEFTMOST TWO BITS = ANYTHING.
NEXT TWO BITS = NUMBER OF POINTERS IN THE WORD, FROM 0 TO 3.
BITS 1 TO PTS (RIGHTMOST, PTS = POINTER SIZE FOR THE MACHINE) = FIRST POINTER (IF ANY).
BITS 1+PTS TO 2*PTS = SECOND POINTER (IF IT EXISTS).
BITS 2*PTS+1 TO 3*PTS = THIRD POINTER (IF IT EXISTS).

BITS BETWEEN THE FIRST FOUR POSITIONS AND THE POINTER FIELDS MAY BE ANYTHING. ANY POINTER FIELD MAY BE ZERO, SIGNIFYING THAT IT DOESN≠T PRESENTLY POINT TO ANYTHING (THE NULL POINTER).

A WORD WITH THE RIGHTMOST POINTER FIELD ALL ONES HAS SPECIAL SIGNIFICANCE TO THE GARBAGE COLLECTOR. IT IS THE ≠GARBAGE COLLECTOR SKIP WORD≠, AND IT CAUSES THE GARBAGE COLLECTOR TO SKIP OVER, WITHOUT SCANNING, THE CURRENT WORD AND THE NEXT N WORDS, WHERE N IS THE VALUE IN THE SECOND POINTER FIELD OF THE WORD (IT IS USED IN THE SETL IMPLEMENTATION TO SKIP OVER THE LOCAL VARIABLES OF AN INACTIVE PROCEDURE).

THERE ARE THREE TYPES OF BLOCKS IN THE HEAP, AS IDENTIFIED BY THE FIRST TWO BITS OF THE BLOCK (FIRST WORD, LEFTMOST TWO BITS).

0 = STANDARD BLOCK,
1 = ONE-WORD BLOCK,
2 = TWO-WORD BLOCK, AND
3 IS UNUSED.

THE ONE- AND TWO-WORD BLOCKS MAY CONTAIN ONLY WORDS THAT FOLLOW THE FORMAT OF A STACK WORD.

THE STANDARD BLOCK CONTAINS FIVE FIELDS IN ITS FIRST WORD, AS FOLLOWS (THE NAMES OF THE FIELD EXTRACTOR MACROS ARE USED).

EBLOCKTYPE = 0.
ENREF = REFERENCE COUNT.
EBLKSIZE = BLOCK SIZE, INCLUDING FIRST WORD.
EHDRSIZE = SIZE OF ≠HEADER≠ PORTION OF THE BLOCK, EXCLUSIVE OF THE FIRST WORD.
EPTRSIZE = SIZE OF ≠POINTER≠ PORTION OF THE BLOCK.

ALL POINTERS THAT MAY BE CONTAINED IN THE BLOCK MUST BE IN THE POINTER AREA, AND THESE WORDS MUST FOLLOW THE FORMAT OF A STACK WORD. THE HEADER AREA AND THE REMAINDER OF THE BLOCK ARE NOT EXAMINED BY THE GARBAGE COLLECTOR. THESE AREAS MAY CONTAIN FLOATING POINT NUMBERS, CHARACTER STRINGS, ETC.

THE GARBAGE COLLECTOR MUST BE ABLE TO LOCATE ALL POINTERS TO THE HEAP, IN ORDER TO ADJUST THEM. POINTERS MAY ONLY EXIST IN THESE PLACES:

THE RUN TIME STACK,
THE HEAP, AND
ARRAY ≠HEAPPTRS≠.

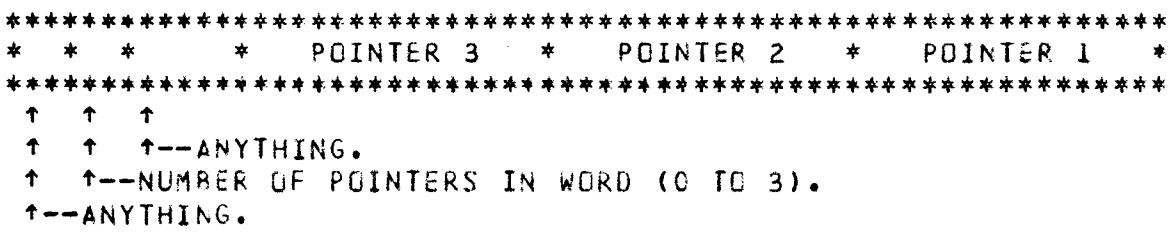
VARIABLES MAY BE ASSIGNED TO THE BASE OF THE RUN TIME STACK BY USE OF THE ≠TACK≠ MACRO. WRITING THE DECLARATORY MACRO CALL ≠TACK(NAME)≠ GENERATES THE MACRO ≠+*NAME = STACK (STORAGESZ-N+1)**≠, WHERE N IS A CONSTANT GENERATED BY COUNTER ZZYZ (THIS COUNTER IS RESERVED FOR THIS PURPOSE). HENCE THE EFFECT IS TO EQUIVALENCE THE VARIABLE ≠NAME≠ TO A PARTICULAR STACK LOCATION. SIMILARLY, THE MACRO CALL ≠PTR(NAME)≠ EQUIVALENCES ≠NAME≠ TO A PARTICULAR LOCATION IN ≠HEAPPTRS≠. COUNTER ZZYY IS RESERVED FOR THIS PURPOSE. THE GARBAGE COLLECTOR USES THE STACK AS A STARTING POINT TO TRACE THROUGH STRUCTURES IN THE HEAP, AND IT ADJUSTS ALL POINTERS IN THE STACK AND HEAP. IT ALSO ADJUSTS POINTERS IN ARRAY ≠HEAPPTRS≠, BUT IT DOES NOT USE THEM AS A STARTING POINT TO LOCATE THE ACTIVE BLOCKS IN THE HEAP.

ARRAY HEAPPTRS IS USED TO HOLD POINTERS THAT ARE USED TO SCAN A BLOCK IN THE HEAP, WHEN THERE IS A POSSIBILITY THAT THE GARBAGE COLLECTOR MIGHT BE INVOKED DURING THE SCAN, AND WHEN IT IS DESIRED FOR EFFICIENCY REASONS TO SCAN USING A ≠POINTER VARIABLE≠ (INDEX INTO THE STORAGE ARRAY) RATHER THAN AN OFFSET. IF P IS SUCH A POINTER VARIABLE, IT SHOULD BE DECLARED ≠PTR(P)≠. P MAY THEN POINT TO FLOATING POINT NUMBERS, CHARACTER STRINGS, ETC.

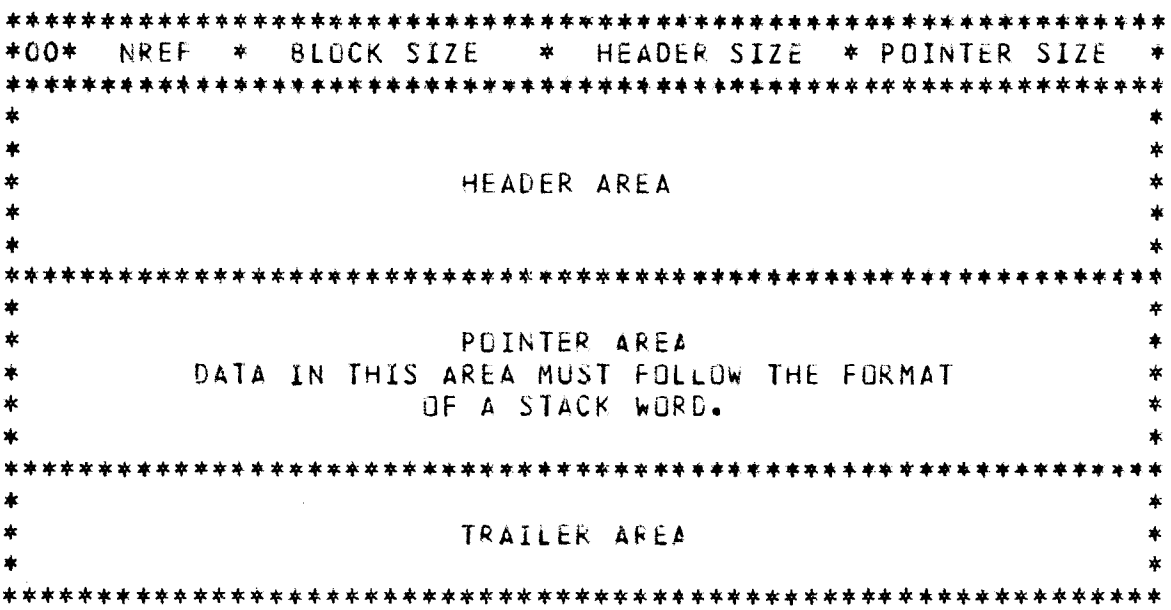
THE SIZE OF ARRAY HEAPPTRS IS SET BY THE MACRO ≠HEAPPTRSSZ≠.

THE FORMATS OF THE STACK WORD AND THE STANDARD BLOCK ARE SHOWN BELOW.

STACK WORD:



STANDARD BLOCK:



3.3 REQUESTING STACK AND HEAP SPACE

STORAGE MANAGEMENT IS QUITE DIFFERENT FOR THE STACK AND THE HEAP. SPACE THAT HAS BEEN ALLOCATED ON THE STACK MUST BE EXPLICITLY FREED, AND THE REQUESTING AND FREEING MUST BE DONE IN A STACK-LIKE ORDER. A SINGLE FREEING OPERATION MAY RELEASE A PORTION OF WHAT WAS LAST REQUESTED, OR THE TOTAL SPACE FOR THE MOST RECENT REQUESTS. TO MINIMIZE THE OVERHEAD OF CHECKING TO SEE IF A GARBAGE COLLECTION IS NECESSARY, A RESERVE OPERATION IS PROVIDED FOR THE STACK. THIS MAKES ALLOCATION AND FREEING SO SIMPLE THAT THE STEPS ARE PLACED IN LINE BY THE MACROS (SEE BELOW).

SPACE THAT HAS BEEN ALLOCATED FROM THE HEAP IS NEVER EXPLICITLY FREED; FREEING IS DONE BY THE GARBAGE COLLECTOR. BLOCKS MAY BE REQUESTED AND FREED IN ANY ORDER. A PORTION OF A BLOCK MAY IN EFFECT BE FREED BY REDUCING THE SIZE FIELDS IN THE BLOCK HEADER. IF THIS IS DONE, THE PORTION FREED MUST BE GIVEN A LEGITIMATE BLOCK HEADER WORD; NORMALLY ONE WITH THE CORRECT BLOCK SIZE AND ZERO HEADER AND POINTER SIZES (SO THAT IT LOOKS TO THE GARBAGE COLLECTOR LIKE ALL TRAILER AREA).

THERE IS NO WAY TO EXPAND THE SIZE OF A BLOCK; IT CAN ONLY BE REALLOCATED (AND MOVED). FOR THE HEAP, THERE IS NOTHING ANALOGOUS TO THE RESERVE OPERATION. THE STEPS REQUIRED FOR ALLOCATING FROM THE HEAP ARE OUT OF LINE (SUBROUTINE CALL), ALTHOUGH THEY ARE SIMPLE ENOUGH SO THAT THEY COULD CONCEIVABLY BE PLACED IN LINE.

THE STACK IS USED BY MEANS OF THE RESERVE MACRO, AND THE PUSH AND POP FAMILY OF MACROS. FOR EXAMPLE, TO STACK THREE ITEMS AND THEN UNSTACK TWO OF THEM, ONE WOULD CODE:

```
RESERVE(3);
PUSH3(A, B, C);
POP2(X, Y);
```

THE RESERVE STATEMENT RESERVES THREE WORDS ON THE STACK. THE PUSH3(A,B,C) IS EQUIVALENT TO PUSH1(A); PUSH1(B); PUSH1(C). POP2(X,Y) IS EQUIVALENT TO POP1(Y); POP1(X) (NOTE THE REVERSAL). HENCE AFTER THE ABOVE SEQUENCE Y = C AND X = B. AFTER PUSH3(A,B,C), ... , POP3(A,B,C), A, B, AND C ARE RESTORED. MACROS ARE PROVIDED FOR STACKING AND UNSTACKING UP TO EIGHT ITEMS AT A TIME. THERE IS NO SIGNIFICANT LIMIT TO THE VALUE OF THE RESERVE PARAMETER.

IN MANY CASES THE RESERVE STATEMENT CAN BE FACTORED OUT OF A PROGRAM LOOP, WITH A RESULTING SAVING OF EXECUTION TIME. PUSH AND POP ARE FAST OPERATIONS.

ITEMS IN THE STACK MAY BE DIRECTLY REFERENCED BY USING THE GLOBAL VARIABLE T, WHICH ALWAYS POINTS TO THE TOP OF THE STACK - 1. HENCE THE TOP OF THE STACK IS STACK(T+1), THE NEXT IS STACK(T+2), ETC. (THE STACK IS IN WHAT WOULD PROBABLY BE REGARDED AS REVERSE ORDER).

BELOW IS SOME SRTL CODE RELATING TO THE STACK.

```
+*RESERVE(NWDS) = CALL RSVSTK(NWDS)**
+*PUSH1(A) = STACK(T) = A;  T = T - 1**
+*PUSH2(A,B) = STACK(T) = A; STACK(T-1) = B; T = T - 2**
+*POP1(A) = T = T + 1;  A = STACK(T)**
+*POP2(A,B) = T = T + 2;  B = STACK(T-1);  A = STACK(T)**
SIZE T(PTS);        /*  TOP OF STACK - 1.                */
SIZE TRES(PTS);     /*  TOP OF RESERVED PORTION - 1.     */
```

TO REQUEST A BLOCK FROM THE HEAP, ONE CODES:

```
GET(N, P);
```

HERE N IS AN EXPRESSION GIVING THE TOTAL NUMBER OF WORDS REQUESTED, INCLUDING THE BLOCK HEADER WORD (IF THERE IS TO BE ONE). P IS A VALUE-RECEIVING EXPRESSION INTO WHICH A POINTER, OR INDEX, WILL BE STORED. THE POINTER INDEXES THE FIRST OF THE N WORDS. N AND P SHOULD BE SIZED WS (WORD SIZE FOR THE MACHINE). THE FIRST WORD OF THE BLOCK RETURNED WILL BE SET WITH THE APPROPRIATE BLOCK SIZE FIELD AND OTHER FIELDS CLEARED. THE CALLER IS RESPONSIBLE FOR COMPLETING THE FORMATTING OF THE BLOCK BEFORE THE GARBAGE COLLECTOR GETS CALLED.

TO REQUEST A SINGLE WORD FROM THE HEAP, ONE MAY CODE

```
GET1(P);
```

FOR A SLIGHT INCREASE IN EFFICIENCY OVER GET(1, P).

CAUTION: WITH THE PRESENT LITTLE COMPILER, PARAMETER P MAY NOT BE DECLARED PTR(P). THIS IS BECAUSE THIS CAUSES P TO BE REPLACED BY AN EXPRESSION OF THE FORM A(I), AND FOR SUCH AN EXPRESSION THE PARAMETER IS MOVED TO A COMPILER TEMPORARY, AND THE VALUE ASSIGNED TO THE TEMPORARY BY THE GET ROUTINE IS NOT PASSED BACK TO LOCATION A(I). NORMALLY ONE WOULD CODE SOMETHING LIKE:

```
SIZE PTEMP(WS);
PTR(P)
...
GET(N, PTEMP);
P = PTEMP;
```

3.4 RECURSIVE CALLS

LITTLE DOES NOT SUPPORT RECURSIVE CALLS, BUT THE USER OF THE SRTL STORAGE MANAGEMENT FACILITY MAY USE THE SRTL RECURSIVE LINKAGE MACROS WITH NO ADDITIONAL RESTRICTIONS ON HIS DATA STRUCTURES. CODING RECURSIVE CALLS WITH THESE MACROS IS VERY AWKWARD; THEIR MAIN VALUE IS THAT THEY PROVIDE A STANDARD WAY TO IMPLEMENT RECURSIVE CALLS, THUS MINIMIZING THE CHANCE FOR ERROR.

IN MANY CASES RECURSIVE LINKAGE CAN BE AVOIDED BY REDESIGNING AN ALGORITHM SO THAT IT IS ITERATIVE, AND USES THE PUSH AND POP MACROS TO SAVE AND RESTORE DATA. WHEN AN ALGORITHM YIELDS TO THIS TECHNIQUE, IT IS PROBABLY THE PREFERABLE WAY TO CODE IT.

RECURSIVE ROUTINES IN SRTL HAVE THE FOLLOWING UNUSUAL FEATURES:

1. THEY ARE NOT DELIMITED BY SUBR OR FNCT AND END. THEY ARE MERELY BLOCKS OF CODE CONTAINED IN A SUBROUTINE OF FUNCTION, PROBABLY DELIMITED BY A FEW BLANK CARDS AND SOME COMMENTARY.
2. THEY THEREFORE HAVE NO FORMAL PARAMETERS.
3. THEY CONTAIN CODE THAT IS DEPENDENT UPON ALL THE PLACES THEY ARE CALLED FROM.

(3) ABOVE STEMS FROM THE FACT THAT LITTLE HAS NO LABEL VARIABLES, AND THE RECURSIVE RETURN MACRO EXPANDS INTO A GOBY.

TO INVOKE A RECURSIVE ROUTINE, ONE CODES

```
RCALL (ROUTINE, RETINDEX, RETLABEL);
```

HERE ROUTINE IS A LABEL AT THE BEGINNING OF THE ROUTINE BEING CALLED (THIS WILL BE REFERRED TO AS THE NAME OF THE ROUTINE). RETINDEX IS A POSITIVE INTEGER IDENTIFYING THE PLACE OF THE CALL, AND RETLABEL IS A NAME THAT SERVES SIMILARLY AS A LABEL. FOR EXAMPLE, IF ROUTINE ≠SEARCH≠ IS CALLED FROM THREE PLACES (SOME OF WHICH MAY BE WITHIN SEARCH ITSELF), THEN THE CALLS MIGHT BE CODED AS:

```
RCALL (SEARCH, 1, L1);
RCALL (SEARCH, 2, L2);
RCALL (SEARCH, 3, BACKHERE);
```

THE RETURN FROM A RECURSIVE ROUTINE IS CODED WITH THE

RRETURN MACRO. FOR EXAMPLE, THE ROUTINE SEARCH WOULD RETURN WITH A STATEMENT SUCH AS:

```
RRETURN (L1, L2, BACKHERE);
```

A RECURSIVE CALL INVOLVES STACKING THE RETURN POINT. THEREFORE, BEFORE THE RCALL MAY BE USED, IT IS NECESSARY TO EXECUTE:

```
RESERVE(1); PUSH1(RETPT);
```

AND UPON RETURN:

```
POP1(RETPT);
```

THE DEFINITIONS OF RCALL AND RRETURN SHOULD CLARIFY HOW THEY WORK.

```
+*RCALL(ROUTINE,RETINDEX,RETLABEL) = RETPT = RETINDEX;
     GO TO ROUTINE; /RETLABEL/ **
+*RRETURN = GOBY RETPT**
```

RETPT IS A GLOBAL VARIABLE DEFINED AS:

```
SIZE RETPT(WS);
```

WHEN MAKING A RECURSIVE CALL, IT IS USUALLY NECESSARY TO STACK THE LOCAL VARIABLES (AT LEAST THOSE THAT ARE NOT DEAD), AND THEN UNSTACK THEM ON RETURN. THIS MAY BE ACCOMPLISHED BY USING THE RCALLSAVE FAMILY OF MACROS. THESE MACROS ALSO SAVE THE RETURN POINT. FOR EXAMPLE:

```
RCALLSAVE2(ROUT, X, Y, 3, L3);
```

STACKS X,Y, AND THE RETURN POINT, EXECUTES A RECURSIVE CALL, AND THEN POPS X,Y, AND THE RETURN POINT. THE DEFINITION OF THIS MACRO IS:

```
+*RCALLSAVE2(R,A,B,I,L) = PUSH3(A,B,RETPT);
                          RCALL(R,I,L);
                          POP3(A,B,RETPT)**
```

THIS MACRO SHOULD BE PRECEDED BY A RESERVE(3) MACRO CALL, WHICH MAY BE FACTORED OUT OF LOOPS.

TO ACCOMPLISH A RECURSIVE CALL THAT SAVES THE RETURN POINT BUT NO VARIABLES, RCALLSAVE0 MAY BE USED. THE MAXIMUM NUMBER OF VARIABLES THAT MAY BE SAVED IS SEVEN.

THE FOLLOWING EXAMPLE SHOULD HELP TO TIE THIS TOGETHER. IT IS TAKEN FROM THE SRTL EQUALITY TEST.

```
          /*                        EQUAL                              */

          SUBR EQUAL;
          /* ARG1, ARG2 = ROOT WORDS OF OBJECTS TO BE
          COMPARED. */

          CALL EQBASIC;
          GOBY (RESULT) (NEQ, EQ, SETS);
/NEQ/     RESULT = FALSE;
          RETURN;
/EQ/      RESULT = TRUE;
          RETURN;
/SETS/    RCALL (SUBSETNT, 1, EQUALR1);
          RETURN;

          /*                      SUBSETNT                             */

/SUBSETNT/;
          ...
          RESERVE(7);
          RCALLSAVE6(SUBSETNT, A1, A2, I1, I1LIM, P1, P2,
                     2, SUBSETNTR1);
          ...
          RRETURN (EQUALR1, SUBSETNTR1);
          /* END OF SUBSETNT. */

          END;  /*  EQUAL  */


          /*                    EQBASIC                                */

          SUBR EQBASIC;
          ...
          END;  /* EQBASIC  */
```

NOTE THAT IF THE FLOW OF CONTROL WITHIN SUBSETNT IS SUCH THAT NO RECURSION OCCURS, THEN NO STACKING AND UNSTACKING IS DONE. THIS APPLIES EVEN TO THE RETURN POINT.

4. USE OF SRTL SUBROUTINES

4.1 DATA STRUCTURES FOR SRTL SUBROUTINES

TO USE THE SRTL SUBROUTINES THAT IMPLEMENT SETL, THE DATA STRUCTURES MUST CLOSELY FOLLOW THOSE USED IN THE SETL IMPLEMENTATION. THERE ARE ELEVEN DATA TYPES THAT MAY BE USED:

0.	INTEGER	6.	SUBROUTINE
1.	REAL	7.	FUNCTION
2.	BOOLEAN STRING	8.	UNDEFINED
3.	CHARACTER STRING	9.	TUPLE
4.	BLANK ATOM	10.	SET
5.	LABEL		

SEVERAL OF THESE HAVE A LONG AND A SHORT FORM. FOR EXAMPLE, CHARACTER STRINGS ARE CONTAINED IN ONE WORD IF THEY WILL FIT, AND OTHERWISE THEY ARE CONTAINED IN THE STORAGE HEAP, WITH A ROOT WORD THAT POINTS TO IT.

EVERY OBJECT IS REPRESENTED BY A ROOT WORD, WHICH FOLLOWS THE FORMAT OF A STACK WORD (AS DESCRIBED IN THE PREVIOUS SECTION). THE FIRST TWO BITS OF A ROOT WORD ARE SET TO 01, WHICH IS THE CODE FOR A ONE-WORD BLOCK. THIS IS SO THE ROOT WORD MAY BE MOVED TO A ONE-WORD BLOCK WITHOUT THE NECESSITY OF SETTING THESE BITS.

THE NEXT TWO BITS OF A ROOT WORD INDICATE THE NUMBER OF POSSIBLE POINTERS IN THE WORD, WHICH IS ALWAYS 1 OR 2 FOR SETL OBJECTS.

THE NEXT FIVE BITS GIVE THE TYPE CODE OF THE ITEM, WITH THE RIGHTMOST BIT INDICATING WHETHER THE ITEM IS SHORT(0) OR LONG(1). THUS 0 = SHORT INTEGER, 1 = LONG INTEGER, 4 = SHORT BOOLEAN STRING, 5 = LONG BOOLEAN STRING, ETC.

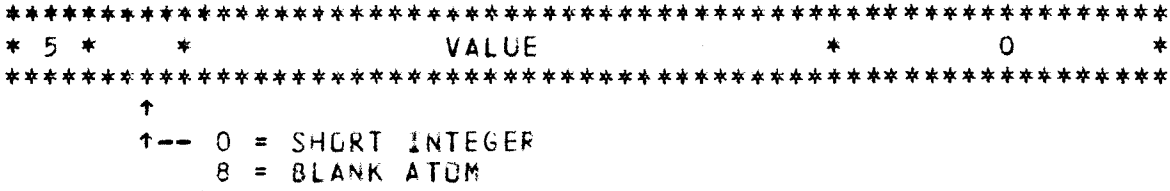
THE NEXT FIELD CONTAINS THE VALUE OF THE ITEM IF IT IS A SHORT ITEM, OR A POINTER TO A BLOCK IN THE HEAP IF IT IS A LONG ITEM.

THE LAST FIELD IS ONLY USED IF THE ITEM IS PUT INTO A SET. ITS PURPOSE IS TO CHAIN TOGETHER ITEMS WHICH GO INTO THE SAME SLOT OF THE HASH TABLE OF THE SET.

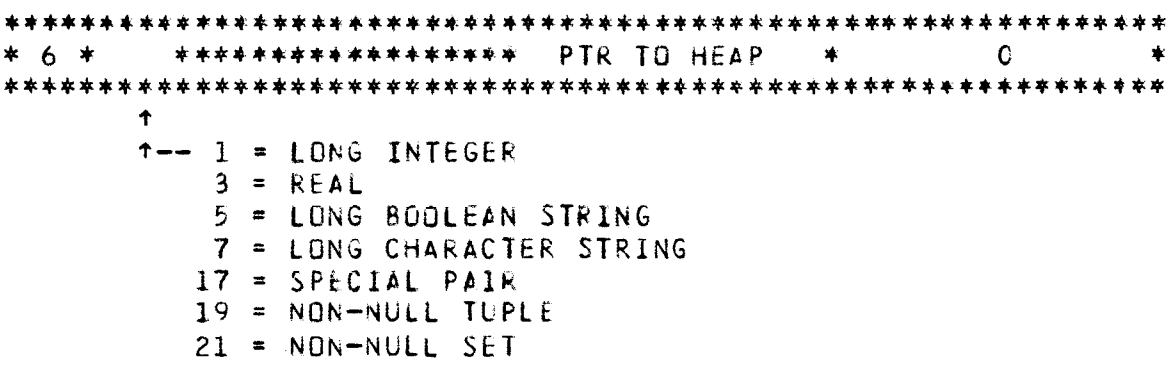
THE ROOT WORD FORMATS ARE DEPICTED BELOW.

ROOT WORDS

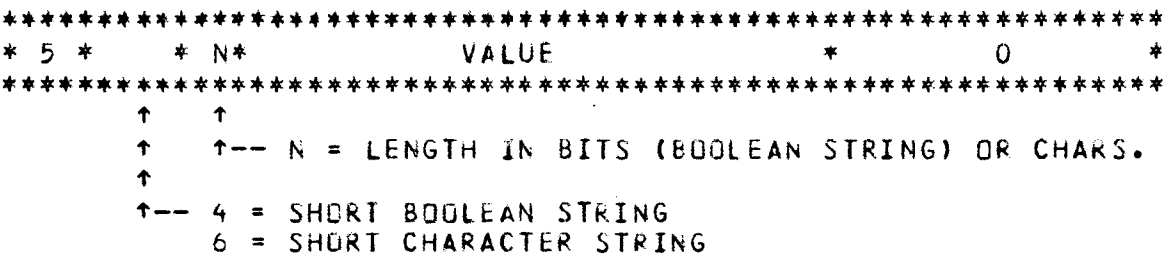
SHORT INTEGER AND BLANK ATOM



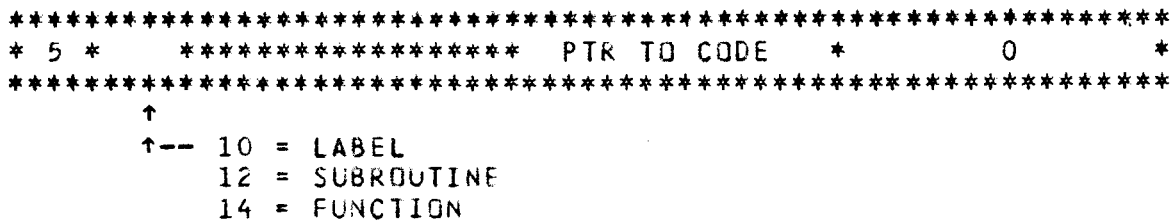
LONG ITEM



SHORT STRING

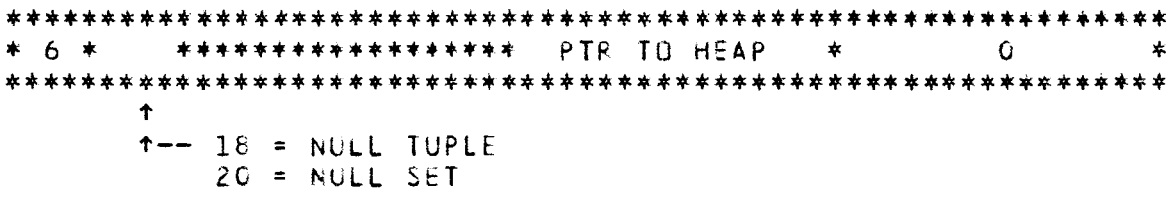


LABEL, SUBROUTINE, AND FUNCTION



UNDEFINED

NULL TUPLE AND NULL SET



THE VALUE FIELD OF THE SHORT INTEGER IS ENCODED IN SIGNED-TRUE FORM, WITH THE LEFTMOST BIT BEING THE SIGN (0=PLUS). THE SRTL ARITHMETIC ROUTINES NEVER RETURN A MINUS ZERO.

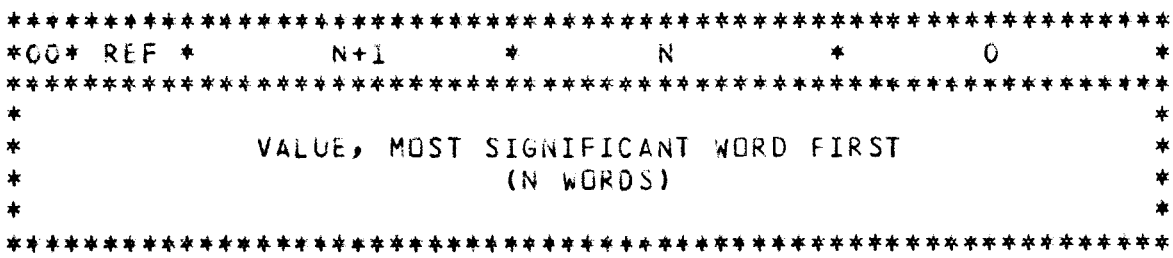
THE VALUE FIELD OF THE BLANK ATOM IS AN UNSIGNED NUMBER, OR IT CAN BE REGARDED AS SIMPLY A FIXED LENGTH BIT STRING, SINCE IT IS NEVER AN OPERAND OF AN ARITHMETIC OPERATION IN A VALID SETL PROGRAM.

BOOLEAN STRINGS ARE RIGHT JUSTIFIED, AND CHARACTER STRINGS ARE LEFT JUSTIFIED.

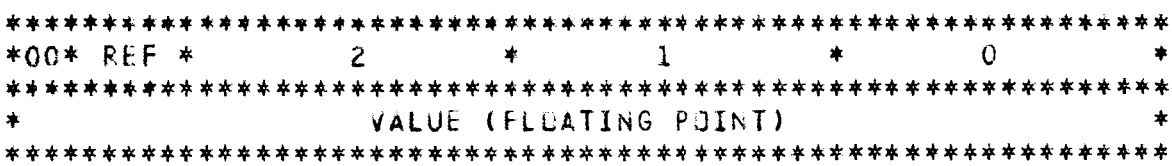
THE HEAP STRUCTURES THAT HOLD THE VALUE OF LONG ITEMS ARE SHOWN BELOW.

HEAP DATA STRUCTURES FOR SETL OBJECTS

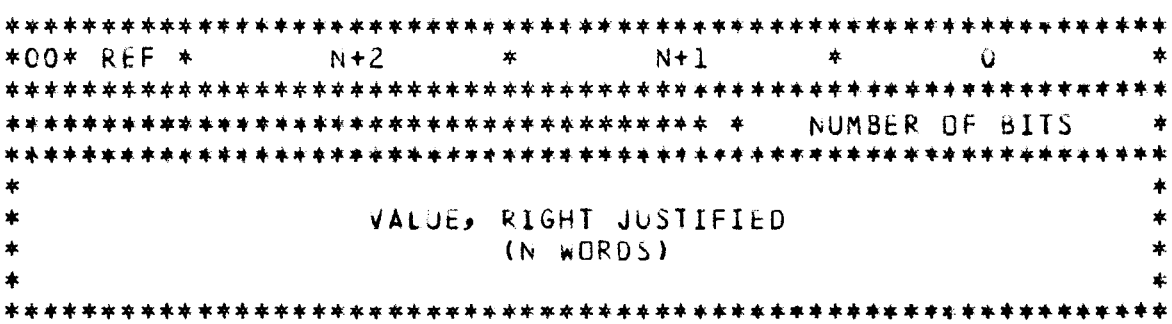
LONG INTEGER



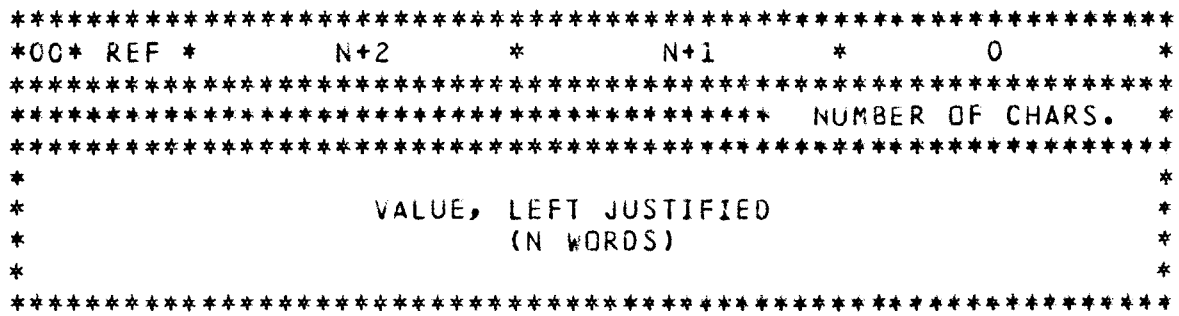
REAL



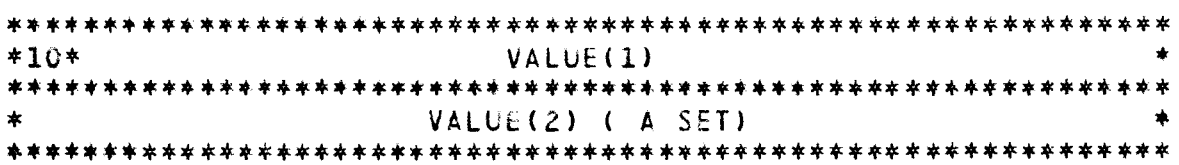
LONG BOOLEAN STRING



LONG CHARACTER STRING

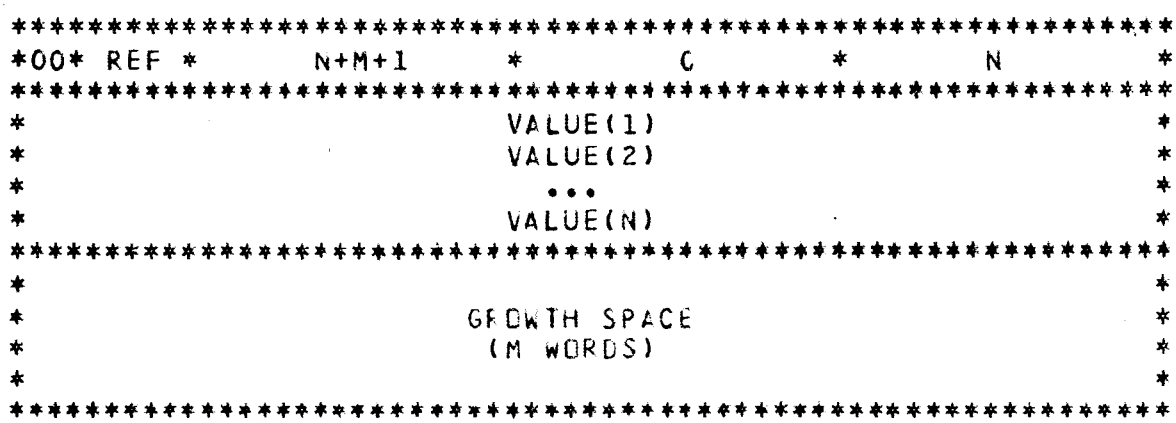


SPECIAL PAIR

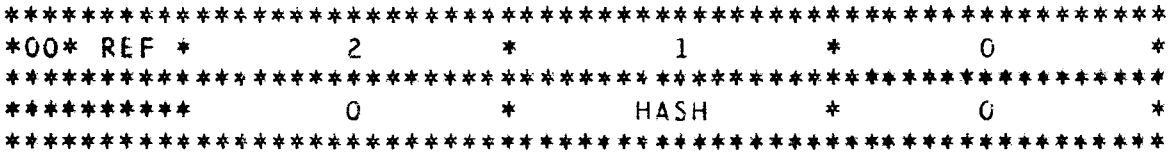


NULL TUPLE

TUPLE

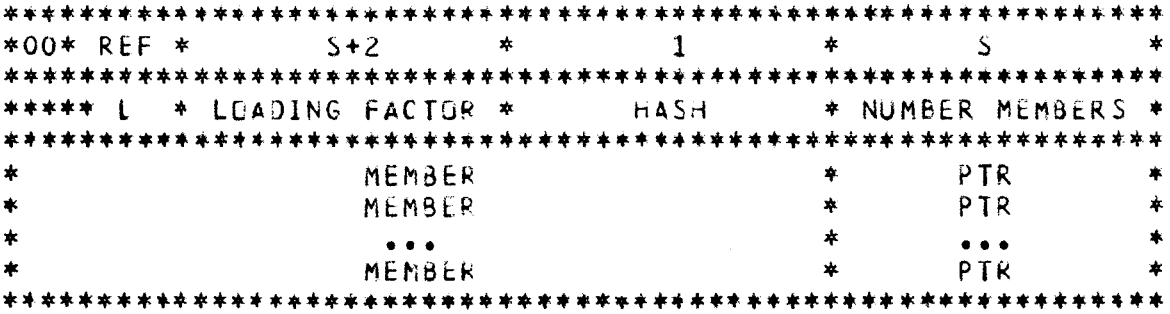


NULL SET



HASH = HASH CODE OF NULL SET (A CONSTANT).

SET



S = HASH TABLE SIZE, L = LOG(BASE 2) S, PTR = POINTER TO NEXT MEMBER IN CHAIN, MEMBER IS UNDEFINED IF NOT PRESENT.

A VARIETY OF MACROS, VARIABLES, AND ROUTINES ARE AVAILABLE TO SIMPLIFY THE CREATION AND MANIPULATION OF SETL OBJECTS. THESE WILL BE BRIEFLY DESCRIBED HERE, BUT THE SRTL SOURCE LISTING SHOULD BE CONSULTED FOR DETAILS.

THERE ARE A NUMBER OF FIELD EXTRACTOR MACROS FOR ACCESSING THE FIELDS OF ROOT WORDS AND HEAP STRUCTURES. FOR EXAMPLE:

```
+*ETYPE = .F.52,5,**      /* OBJECT TYPE.                    */
+*ETYP = .F.53,4,**       /* TYPE WITHOUT LONG/SHORT FLAG.   */
+*EBLKSIZE = .F.35,17,**      +*OFFBLKSIZE = 0**
```

EACH EXTRACTOR FOR A FIELD IN A HEAP STRUCTURE HAS AN OFFSET ASSOCIATED WITH IT. IF P POINTS TO A BLOCK IN THE HEAP, THEN THE BLOCK SIZE FIELD, FOR EXAMPLE, IS REFERENCED BY THE EXPRESSION:

```
EBLKSIZE HEAP(P+OFFBLKSIZE)
```

THERE ARE A NUMBER OF MACROS THAT GENERATE IN LINE CODE FOR REFERENCING A FIELD OF A BLOCK IN THE HEAP FROM THE ROOT WORD (RATHER THAN FROM A POINTER TO THE BLOCK). FOR EXAMPLE, TO OBTAIN THE NUMBER OF MEMBERS OF A SET, ONE CODES

```
NMEMBS(SET)
```

WHERE SET IS A VARIABLE DESIGNATING THE ROOT WORD. THE NMEMBS MACRO IS DEFINED AS FOLLOWS:

```
+*NMEMBS(ROOT) = ESETSIZE HEAP(EPTR ROOT + OFFSETSIZE)**
```

THUS NMEMBS(X) EXPANDS TO:

```
.F.1,17,STORAGE(.F.18,17,X + 1)
```

SOME MACROS IN THIS CATEGORY ARE MORE COMPLICATED. FOR EXAMPLE, TO ACCESS CHARACTER I OF A LONG CHARACTER STRING (USING IN LINE CODE), ONE MAY CODE LCSCHAR(ROOT, I, R)., HERE R GETS THE RESULT. THE MACRO DEFINITIONS FOR THIS ARE:

```
        /* WORD I OF A LONG CHARACTER STR.  I=0,...*/
+*LCSWRD(ROOT,I) = HEAP(EPTR ROOT + OFFCSTR + (I))**

        /* R = CHAR I OF A LONG CHAR STR. I = 0,...*/
+*LCSCHAR(ROOT,I,R) = SIZE ZZZQ(WS), ZZZR(WS);
        QREMCPW(I, ZZZQ, ZZZR);
        R = .F.(WSMCSP1-CS*ZZZR),CS,LCSWRD(ROOT,ZZZQ)**
```

THE TYPE CODES MAY BE REFERENCED SYMBOLICALLY BY MACROS

SUCH AS:

```
++INT = 0**       ++SINT = 0**       ++LINT = 1**
++REAL = 1**                         ++LREAL = 3**
++BOOL = 2**      ++SBOOL = 4**      ++LBOOL = 5**
```

HERE INT, BOOL, ETC. ARE USED IN CONJUNCTION WITH ETYP, AND SINT, LINT, SBOOL, LBOOL, ETC., (SHORT INTEGER, LONG INTEGER, ETC.) ARE USED IN CONJUNCTION WITH ETYPE.

THERE ARE SOME MISCELLANEOUS MACROS SUCH AS:

```
++IFSHORT(X) = IF (ELONG X .EQ. 0)**
++IFATOM(X) = IF (ETYPE X .LE. SUNDEF)**
```

USE OF THESE IS ENCOURAGED AS IT TENDS TO MAKE A PROGRAM LESS SENSITIVE TO CHANGES IN TYPE CODE ASSIGNMENTS, ETC.

SRTL INCLUDES TWO TABLES FOR MISCELLANEOUS USES SUCH AS MASKING. THESE ARE POWTWO = 2, 4, 8, 16, ..., 65536, AND ONEBITS = 1, 3, 7, 15, ..., (2 EXP WS)-1. POWTWO IS SIZED PTS AND ONEBITS IS SIZED WS. ONEBITS(WS) IS THE SUGGESTED WAY TO OBTAIN A WORD CONTAINING ALL ONE BITS.

A NUMBER OF SKELETAL ROOT WORDS ARE DEFINED TO FACILITATE BUILDING SETL OBJECTS. THESE WORDS HAVE A FEW FIELDS, SUCH AS THE TYPE FIELD, ALREADY SET. SOME OF THESE ARE:

```
ROOTSINT        SHORT INTEGER.
ROOTLINT        LONG INTEGER.
ROOTREAL        REAL.
ROOT1BOOL       SHORT BOOLEAN STRING, LENGTH 1.
ROOTLBOOL       LONG BOOLEAN STRING.
ETC.
```

FOR EXAMPLE, TO CREATE THE SETL OBJECT FOR THE INTEGER FIVE, ONE COULD CODE:

```
SIZE FIVE(WS);
FIVE = ROOTSINT;
EVAL FIVE = 5;
```

TO SIMPLIFY THE CREATION OF SETL OBJECTS, THERE IS A SET OF OBJECT ≠GENERATOR≠ ROUTINES. THESE ARE:

```
GENINT(N, V)    GENSUBR(V)
GENSINT(V)      GENFUN(V)
GENREAL(V)      GENPAIR(A, B)
GENBOOL(N, V)   GENTUP(N, V)
GENCHAR(N, V)   GENSET(N, V)
GENLABL(V)
```

GENINT GENERATES AN INTEGER, GENSINT A SHORT INTEGER, ETC. HERE N IS THE NUMBER OF WORDS IN AN INTEGER, BITS IN A BOOLEAN STRING, CHARACTERS IN A CHARACTER STRING, COMPONENTS IN A TUPLE, OR MEMBERS IN A SET. V IS THE ≠VALUE≠ OF THE OBJECT, STORED AS AN ORDINARY ARRAY (LITTLE DIMS STATEMENT). THE PARAMETERS ARE NOT SETL OBJECTS, EXCEPT IN THE CASES OF A AND B IN GENPAIR, AND V IN GENTUP AND GENSET. THE VALUE OF EACH FUNCTION IS A ROOT WORD FOR THE GENERATED SETL OBJECT. FOR SEVERAL OF THE ROUTINES, EITHER A LONG OR A SHORT OBJECT IS CREATED, DEPENDING ON THE VALUE OF N. FOR GENSINT, V IS KNOWN TO FIT IN A SHORT INTEGER. FOR DETAILS SUCH AS WHETHER THE DATA IS LEFT OR RIGHT ADJUSTED IN ARRAY V, CONSULT THE SRTL LISTING.

AS AN EXAMPLE IN THE USE OF THE GENERATOR ROUTINES, THE ABOVE ITEM ≠FIVE≠ COULD BE SET AS FOLLOWS:

```
SIZE FIVE(WS);
FIVE = GENSINT(5);
```

OR FIVE = GENINT(1, 5), AS LITTLE TREATS ONE-WORD ARRAYS AND NON-ARRAYS THE SAME.

THE READER MAY WONDER WHETHER HE SHOULD USE THE GENERATOR FUNCTIONS OR CREATE OBJECTS ≠BY HAND≠. THE ANSWER IS THAT IN ALMOST ALL CASES YOU SHOULD USE THE GENERATOR FUNCTIONS. THIS WILL RESULT IN A PROGRAM THAT IS MORE COMPACT, EASIER TO READ, LESS SENSITIVE TO CHANGES IN SRTL, AND LESS APT TO HAVE BUGS. IT WILL BE SLOWER RUNNING, BUT THIS WILL PROBABLY NOT BE SIGNIFICANT EXCEPT IN A FEW CASES.

4.2 CALLING SEQUENCES

IN MOST CASES THE ARGUMENTS FOR SRTL SUBROUTINES ARE PASSED VIA THE GLOBAL VARIABLES ARG1, ARG2, AND ARG3. IF THE PURPOSE OF THE ROUTINE IS TO ALTER AN ARGUMENT (E.G., THE AUGMENT ROUTINE), THEN THE RESULT IS PASSED BACK VIA THE APPROPRIATE ARGUMENT. ON THE OTHER HAND, IF THE ROUTINE IS GENERALLY THOUGHT OF AS FUNCTION-LIKE (E.G., THE EQUALITY TEST), THEN THE RESULT IS PASSED BACK VIA THE GLOBAL VARIABLE ≠RESULT≠. AS AN EXAMPLE, HERE IS A CODE SEGMENT THAT ADDS INTEGERS TO A SET S UNTIL IT BECOMES EQUAL TO ANOTHER SET P:

```
SIZE K(WS);
ARG2 = S;
K = 0;
RESULT = FALSE;
WHILE (RESULT .EQ. FALSE);
  K = K + 1;
  ARG1 = GENSINT(K);        /* BUILD SETL INTEGER. */
  CALL AUGMENT;             /* ADD ARG1 (K) TO ARG2 (S). */
  ARG1 = P;
  CALL EQUAL;           /* COMPARE ARG1 (P) TO ARG2 (S). */
  ENDWHILE(RESULT);
S = ARG2;                   /* MOVE RESULT TO S. */
```

NOTE THAT IT IS GENERALLY NECESSARY TO MOVE THE RESULT TO THE DESIRED VARIABLE.

THIS TYPE OF CALLING SEQUENCE IS AWKWARD TO USE, BUT IT AVOIDS THE OVERHEAD OF PARAMETER PASSING VIA THE LITTLE MECHANISM, AND IT ALLOWS ONE TO GAIN EFFICIENCY BY SUCH TRICKS AS FACTORING THE ASSIGNMENT TO AN ARGUMENT OUT OF A LOOP (AS IN THE CASE OF ARG2 ABOVE).

THE DISCUSSION ABOVE APPLIES ONLY WHEN THE ARGUMENTS AND RESULT ARE SETL OBJECTS. THIS IS BECAUSE ARG1, ETC., ARE DEFINED AS FOLLOWS:

```
TACK(ARG1)  TACK(ARG2)  TACK(ARG3)  TACK(RESULT)
```

THAT IS, THESE QUANTITIES ARE EQUIVALENCED TO A STACK LOCATION (SO THE GARBAGE COLLECTOR CAN MARK THE STRUCTURES THEY POINT TO).

A FEW SRTL ROUTINES HAVE POINTERS AS PARAMETERS. FOR THESE, ARG1P, ETC., ARE USED, WHICH HAVE THE DEFINITIONS:

```
PTR(ARG1P)  PTR(ARG2P)  PTR(ARG3P)  PTR(RESULTP)
```

TO PASS MISCELLANEOUS DATA TO A ROUTINE, SUCH AS A SMALL INTEGER, IT IS SAFE TO USE ARG1, ETC. THIS IS BECAUSE THE THIRD AND FOURTH BITS (FROM THE LEFT) WILL BE ZERO, INDICATING TO THE GARBAGE COLLECTOR THAT THE WORD CONTAINS NO POINTERS.

4.3 SAMPLE ROUTINES

A FEW ROUTINES WILL BE DISCUSSED HERE TO GIVE THE FLAVOR OF WHAT TO EXPECT IN THE SETL RUN TIME LIBRARY.

ATOM ROUTINE:

```
ARG1 = X;          /* ROOT WORD OF OBJECT TO BE TESTED*/
CALL ATOM;         /* RESULT IS TRUE (THE SETL OBJECT)
                      IF ARG1 IS AN ATOM, AND FALSE IF IT
                      IS A SET OR TUPLE. */
```

NELT ROUTINE:

```
ARG1 = X;
CALL NELT;         /* RESULT IS #X (THE SETL NUMBER SIGN
                      OPERATOR), CODED AS A SETL INTEGER
                      (SHORT). AN ERROR MESSAGE IS WRITTEN
                      OUT AND EXECUTION TERMINATES IF ARG1
                      IS NOT A VALID DATA TYPE FOR THIS
                      OPERATION. */
```

THE NELT ROUTINE ACCEPTS AS INPUT A STRING (BOOLEAN OR CHARACTER, LONG OR SHORT), TUPLE, OR SET. FOR ANYTHING OTHER THAN COMPILER GENERATED CODE, ONE USUALLY KNOWS THE TYPE OF OBJECT INVOLVED. IN THIS CASE, A MACRO CAN BE USED, WHICH WILL GENERATE IN LINE CODE FOR THE NUMBER OF ELEMENTS FUNCTION. THE MACROS TO USE ARE:

```
ELSBS X          FOR A SHORT BOOLEAN STRING.
NBITS(X)         FOR A LONG BOOLEAN STRING.
ELSCS X          FOR A SHORT CHARACTER STRING.
NCHARS(X)        FOR A LONG CHARACTER STRING.
NCOMPS(X)        FOR A TUPLE (NULL OR OTHERWISE).
NMEMBS(X)        FOR A SET (NULL OR OTHERWISE).
```

EQUAL ROUTINE:

```
ARG1 = X;
ARG2 = Y;
CALL EQUAL;     /* RESULT IS TRUE OR FALSE ENCODED AS
                   A SETL OBJECT. */
```

IN SOME CASES ONE CAN USE A LOWER LEVEL, NON-RECURSIVE ROUTINE NAMED EQBASIC. THIS HAS THE SAME INPUT ARGUMENTS, AND IT SETS ≠RESULT≠ TO AN ORDINARY INTEGER ENCODED AS FOLLOWS:

1: OBJECTS ARE NOT EQUAL.
2: OBJECTS ARE EQUAL.
3: OBJECTS ARE NON-NULL TUPLES OF EQUAL LENGTH OR ARE SPECIAL PAIRS (THE COMPONENTS HAVE NOT BEEN EXAMINED).
4: OBJECTS ARE NON-NULL SETS OF EQUAL SIZE AND EQUAL HASH CODES (THE MEMBERS HAVE NOT BEEN EXAMINED).

IN CASES (1) AND (2), THE OBJECTS MAY BE ATOMS, NULL TUPLES, NULL SETS, OR OBVIOUSLY UNEQUAL TUPLES OR SETS.

ELMT ROUTINE

```
ARG1 = X;
ARG2 = S;       /* STRING, TUPLE, OR SET. */
CALL ELMT;      /* RESULT IS TRUE OR FALSE, DEPENDING
                   ON WHETHER OR NOT ARG1 IS AN ELEMENT
                   OF ARG2 */
```

IF THE TYPE OF OBJECT INVOLVED (S) IS KNOWN, THEN A LOWER LEVEL ROUTINE MAY BE USED, WITH THE SAME CALLING SEQUENCE. THESE ARE:

ELMTBST:	FOR BOOLEAN STRINGS (SHORT OR LONG).
ELMTCST:	FOR CHAR. STRINGS (SHORT OR LONG).
ELMTTUP:	FOR TUPLES (NULL OR OTHERWISE).
ELMTSET:	FOR SETS (NULL OR OTHERWISE).
ELSSMP:	FOR S A SET, X KNOWN NOT TO BE A TUPLE OF LENGTH ≥ 3.
ELSTUP:	FOR S A SET, X A TUPLE (ANY LENGTH).

AUGMENT ROUTINE

```
ARG1 = X;
ARG2 = S;        /* A SET. */
CALL AUGMENT;    /* OUTPUT IS SET S AUGMENTED BY X,
                    PASSED BACK IN ARG2.  THE GLOBAL
                    VARIABLE ≠RESULT≠ IS DESTROYED. */
```

HERE AGAIN THERE ARE SEVERAL LOWER LEVEL ROUTINES THAT MAY BE USED FOR INCREASED EFFICIENCY.

AUGAOK: SAME AS AUGMENT, EXCEPT ARGUMENTS ARE NOT VALIDITY CHECKED (ARG2 MUST NOT BE UNDEFINED).

AUGSIMP: SAME AS AUGAOK, EXCEPT THAT ARG1 IS KNOWN NOT TO BE A TUPLE OF LENGTH .GE. 3.

AUGTUP: SAME AS AUGAOK, EXCEPT ARG1 IS KNOWN TO BE A TUPLE (OF ANY LENGTH).

SETWTH1 ROUTINE

```
ARG1 = X;
CALL SETWTH1;    /* RESULT IS THE SET CONTAINING
                    ONLY ARG1 AS A MEMBER, PASSED BACK
                    VIA ARG2 (FOR COMPATIBILITY WITH
                    AUGMENT). */
```

EVEN HERE THERE IS A LOWER LEVEL ROUTINE:

SET1SMP: ARG1 IS KNOWN NOT TO BE A TUPLE OF LENGTH .GE. 3.

OTHER ROUTINES CURRENTLY EXISTING IN SRIL ARE:

EXPAND:	REALLOCATES A SET, GIVING IT A LARGER HASH TABLE.
DIMINIS:	REMOVES A MEMBER FROM A SET (SETL LESS.).
CONTRCT:	REALLOCATES A SET, GIVING IT A SMALLER HASH TABLE (HALF SIZE).
ARB:	OBTAINS AN ARBITRARY ELEMENT FROM A SET, TUPLE, OR STRING (SETL ARB.).
TUPADD1:	APPENDS AN OBJECT TO THE END OF A TUPLE (ARG1(NELT(ARG1)+1) = ARG2).
CONCATT:	CONCATENATES TUPLES.
EXPANDT:	REALLOCATES SPACE FOR A TUPLE. PROVIDING MORE (OR LESS) GROWTH SPACE, SO THAT THE GROWTH SPACE IS ABOUT 25 PERCENT OF THE LENGTH OF THE TUPLE.
OF:	SETL F(X) FOR RETRIEVAL, ONE ARGUMENT. F MAY BE A SET, TUPLE, STRING, OR FUNCTION.

IN ADDITION TO THE ABOVE ROUTINES, WHICH ARE DIRECTLY AIMED AT IMPLEMENTING VARIOUS SETL EXPRESSIONS, THERE ARE A FEW UNDERLYING SUPPORT ROUTINES. SOME OF THESE ARE:

START:	ROUTINE TO BE ENTERED FROM THE OPERATING SYSTEM (OR FROM A LITTLE INTERFACING ROUTINE, IF THERE IS ONE). START CALLS ≠SETLMPG≠, A COMPILER-GENERATED NAME, AFTER PERFORMING A FEW INITIALIZATION STEPS.
RSVSTK:	CALLED BY RESERVE MACRO.
GETSTG:	CALLED BY GET MACRO.
GET1STG:	CALLED BY GET1 MACRO.
GARBCOL:	THE GARBAGE COLLECTOR.

ERRIMP:	WRITES OUT THE ERROR MESSAGE ≠*-*-*IMPLEMENTATION ERROR. UNUSED TYPE CODE DETECTED BY ...≠.
ERRTYPE:	WRITES OUT THE ERROR MESSAGE ≠*-*-* INVALID TYPE CODE FOR ROUTINE ...≠.
ERRVAL:	WRITES OUT THE ERROR MESSAGE ≠*-*-* INVALID DATA VALUE FOR ROUTINE ...≠.
ERRMSG:	WRITES OUT THE ERROR MESSAGE ≠*-*-* ...≠.
ERRIMPL:	WRITES OUT THE ERROR MESSAGE ≠*-*-* IMPLEMENTATION ERROR ...≠.
DISPLAY:	CONVERTS A FULL WORD TO A CHARACTER STRING SUCH AS ≠OCTAL 0123456701 2345670123≠. THIS IS USED IN THE ERROR MESSAGES, AND IT WOULD BE CHANGED TO GENERATE SOMETHING LIKE ≠HEX 0123ABCD≠ ON THE SYSTEM/360.
HASH:	COMPUTES THE HASH CODE FOR ANY SETL OBJECT.