LITTLE NEWSLETTER 42A — JULY 20, 1976
A PARSER-CODE GENERATOR INTERFACE — T. STUART
APPENDIX

MACROS USED OUTSIDE THE INTERFACE

```
*/

++ WS = .WS. **  ++ PS = .PS. **  ++ CS = .CS. **
++ SORG = .F. 1+.SL. , .SO. , **  ++ SLEN = .F. 1 , .SL. , **
++ TEXTL( MESSAGEOUT ) = CALL TEXTLR(MESSAGEOUT); **
++ ENDL = CALL ENDLR ; **



/* COMMAND MACROS - GENERATORS USE THESE TO ACCESS STACKS AND
                    WINDOWS */

++ INIT↗CODE =
        $ INITIALIZE OP STACK AND QUEUE STACK
    CALL RESET↗READ ;   **

++ POP  =
        $ POPS NEXT OPERATOR INTO -OP- AND -COUNT- AND
        $ INITIALIZES STACKS FOR MACROS POP2, PUSH2, AND GET↗ARG.
    CALL POP↗OP ;   **

++ POP2  =
        $ POPS OPERATORS IN THE STACK FOLLOWING -OP- AND
        $ INITIALIZES STACK FOR GET↗ARG2.
    CALL NEXT↗OP2 ;   **

++ PUSH2  =
        $ RETIEVES OPERATORS IN THE STACK PRECEDING -OP- AND
        $ INTIALIZES STACK FOR MACRO GET↗ARG2.
    CALL LAST↗OP2 ;   **

++ POP↗QUEUE  =
        $WINDOWS NEXT OPERAND IN -QUE-.
    CALL QUEUE↗UP ;   **

++ GET↗ARG( ARGCLASS ) =
        $WINDOWS AN ARGUMENT OF THE CURRENT -OP- IN -ARG-.
    CALL OPEN↗ARG( ARGCLASS ) ;   **

++ GET↗ALL  =
        $WINDOWS ALL ARGUMENTS OF THE CURRENT -OP- EXCEPT
        $MULTIPLE CLASS ARGUMENTS AFTER THE FIRST.
    CALL ALL↗ARGS ;   **

++ GET↗ARG2( ARGCLASS ) =
        $WINDOWS AN ARGUMENT OF THE CURRENT -OP2- IN -ARG2-.
    CALL SET↗ARG2( ARGCLASS ) ;   **

++ GET↗ANY( IDENT ) =
        $WINDOWS OPERAND IDENT IN -ANY-.
    CALL ANY↗ARG( IDENT ) ;   **
/*
```

```
*/
    /* OPERATOR TYPES  */

     **   OP.BEGIN       = -6 **   $ FIRST OP IN A ROUTINE
     **   OP.COMMENT     = -5 **   $ COMMENTS FROM SOURCE
     **   OP.BLOCK       = -4 **   $ BLOCK INFO
     **   OP.END         = -3 **   $ END OF ROUTINE
     **   OP.FNCT        = -2 **   $ FNCT ENTRY
     **   OP.SUBR        = -1 **   $ SUBROUTINE ENTRY
                                   $ 0 NOT USED
     **   OP.ADD         =  1 **
     **   OP.SUB         =  2 **
     **   OP.GT          =  3 **
     **   OP.LT          =  4 **
     **   OP.GE          =  5 **
     **   OP.LE          =  6 **
     **   OP.EQ          =  7 **
     **   OP.NE          =  8 **
     **   OP.MUL         =  9 **
     **   OP.DIV         = 10 **
     **   OP.OR          = 11 **
                                   $ 12 NOT USED
     **   OP.AND         = 13 **
     **   OP.EXOR        = 14 **
     **   OP.LIST        = 15 **   $ TURN ON/OFF ASSEMBLY LIST
     **   OP.NB          = 16 **   $ NUMBER OF BITS OPERATION
     **   OP.FB          = 17 **   $ FIRST BIT OPERATION
     **   OP.NOT         = 18 **   $ NOT OPERATION
     **   OP.FCALL       = 19 **
     **   OP.SCALL       = 20 **
     **   OP.ASIN        = 21 **   $ SIMPLE ASSIGNMENT OPERATION
     **   OP.DATA        = 22 **   $ DATA OPERATIO
     **   OP.FASIN       = 23 **   $ FIELD ASSIGNMENT .F.
     **   OP.IO          = 24 **   $ BINARY TRANSPUT
     **   OP.RETURN      = 25 **   $  RETURN
     **   OP.FEXT        = 26 **   $  EXTRACTION OPERATION
     **   OP.IF          = 27 **   $ IF (...) GO TO
     **   OP.LAB         = 28 **   $ LABEL DEFINITION
     **   OP.GOTO        = 29 **   $ GO TO
     **   OP.GOBY        = 30 **
     **   OP.XLOAD       = 31 **   $  INDEXED (ARRAY) LOAD
     **   OP.XASIN       = 32 **   $ INDEXED STORE
     **   OP.XFASIN      = 33 **   $ INDEXED FIELD STORE
     **   OP.IFNOT       = 34 **   $ IF NOT
     **   OP.CCAT        = 35 **   $ .CC. OPERATION
     **   OP.IN          = 36 **   $ .IN. OPERATION
     **   OP.EEXT        = 37 **   $ .E. EXTRACT OP
     **   OP.SEXT        = 38 **   $ .S. EXTRACT OPERATION
     **   OP.EASIN       = 39 **   $ .E. FIELD ASSIGNMENT
     **   OP.SASIN       = 40 **   $ .S. FIELD ASSIGNMENT
     **   OP.XEASIN      = 41 **   $ .E. INDEXED FIELD STORE
     **   OP.XSASIN      = 42 **   $ .S. INDEXED FIELD STORE
     **   ROP.ADD        = 43 **   $ REAL ADD
     **   ROP.SUB        = 44 **   $ REAL SUBTRACT
     **   ROP.GT         = 45 **   $ REAL GREATER THAN
     **   ROP.LT         = 46 **   $ REAL LESS THAN
     **   ROP.GE         = 47 **   $ REAL GREATER THAN OR EQUAL TO
     **   ROP.LE         = 48 **   $ REAL LESS THAN OR EQUAL TO
     **   ROP.EQ         = 49 **   $ REAL EQUAL TO
/*
```

```
*/
     ++  ROP#NE          = 50 **   $ REAL NOT EQUAL TO
     ++  ROP#MUL         = 51 **   $ REAL MULTIPLICATION
     ++  ROP#DIV         = 52 **   $ REAL DIVISION
     ++  ROP#USUB        = 53 **   $ REAL UNARY MINUS
     ++  OP#FLOAT        = 54 **
     ++  OP#IFIX         = 55 **
     ++  OP#ABS          = 56 **
     ++  OP#IABS         = 57 **
     ++  OP#AINT         = 58 **
     ++  OP#INT          = 59 **
     ++  OP#AMOD         = 60 **
     ++  OP#MOD          = 61 **
     ++  OP#SIGN         = 62 **
     ++  OP#ISIGN        = 63 **
     ++  OP#DIM          = 64 **
     ++  OP#IDIM         = 65 **
     ++  OP#EXP          = 66 **   $ EXPONENTIAL
     ++  OP#ALOG         = 67 **   $ NATUAL LOG
     ++  OP#ALOG10       = 68 **   $ COMMON LOG
     ++  OP#SIN          = 69 **   $ SINE
     ++  OP#COS          = 70 **   $ COSINE
     ++  OP#TANH         = 71 **   $ HYPERBOLIC TANGENT
     ++  OP#SQRT         = 72 **   $ SQUARE ROOT
     ++  OP#ATAN         = 73 **   $ ARC TANGENT
     ++  OP#ATAN2        = 74 **   $ ATAN(A1/A2)

     ++  OP#FIRST        = OP#BEGIN    ** $ FIRST OP IN LIST
     ++  OP#LAST         = OP#ATAN2     ** $ LAST ...
     ++  NMBR#OPS = 81 **   $ NO. OF OP SPACES




/*   INDICES TO WINDOW ELEMENTS FOR ATTRIBUTES OF ARGUMENTS

     ARGUMENT        ATTRIBUTE     ATTRIBUTES DEFINED FOR-
     ATTRIBUTES      BINDING
     ----------      ---------     -----------------------
*/
     ++ VALUE=1 **   $ STATIC      ALL EXCEPT TEMPORARIES
     ++ QT=2 **      $ STATIC      (QUANTITY TYPE) ALL
     ++ ID=3 **      $ STATIC      ALL
     ++ SIZ=4 **     $ STATIC      ALL
     ++ DIM=5 **     $ STATIC      VARIABLES
     ++ NSID=6**     $ STATIC      (NAMESET ID) GLOBALS, EXTERNAL FILES
     ++ USE=7 **     $ OP-BOUND    TEMPORARIES
     ++ BYTES=8 **   $ STATIC      (BYTE OR CHAR. COUNT) SEE SUBR STATIC

     ++ NATS =  8 **   $ NUMBER OF ATTRIBUTES (DIMS OF WINDOWS)
     ++ WINDOWSZ = WS **   $ SIZE OF WINDOWS IS USUALLY WS


/*
```

```
*/
/*   QUANTITY TYPES (QT) FOR ARGUMENTS  */

     ** Q#BITS =    1 **  $ BIT STRING
     ** Q#NEGATE =  2 **  $ NEGATIVE NUMBER
     ** Q#REAL =    3 **  $ REAL NUMBER
     ** Q#CHARS=    4 **  $ STRING OF CHARACTERS
     ** Q#CODES=    5 **  $ STRING OF CHARACTER CODES
     ** Q#TEMP =    6 **  $ TEMPORARY
     ** Q#VAR  =    7 **  $ VARIABLE
     ** Q#PARAM=    8 **  $ FORMAL PARAMETER
     ** Q#LABEL=    9 **  $ LABEL
     ** Q#EXTERNAL=10**  $ EXTERNAL SYMBOL


/*   CLASSES OF ARGUMENTS  */

     ** C#SOURCE   =  1  **  $ SOURCE ARGUMENT
     ** C#RESULT   =  2  **  $ RESULT TEMPORARY
     ** C#TARGET   =  3  **  $ TARGET VARIABLE FOR ASSIGNMENT
     ** C#INDEX    =  4  **  $ INDEX
     ** C#LEFT     =  5  **  $ LEFT ARGUMENT IN BINARY OPERATION
     ** C#RIGHT    =  6  **  $ RIGHT...
     ** C#LOCATION=  7  **  $ LABEL OR ROUTINE NAME
     ** C#WIDTH    =  8  **  $ FIELD WIDTH OR CHARACTER COUNT
     ** C#ORIGIN   =  9  **  $ POSITION OF START OF FIELD
     ** C#MULTIPLE= 10  **  $ ONE OF MULTIPLE ARGUMENTS

     ** NMBR#CLASSES = 10  **  $ 10 CLASSES OF OPERANDS


     ** NMSET#LEN = 20 **  $ NUMBER OF CHARACTERS IN NAMESET NAME


/*
```

```
     MACROS USED ONLY BY THE INTERFACE
     ------ ---- ---- -- --- ---------
*/

/*    LEGAL STATUS OF CLASSES (USED TO DEFINE -LEGAL- ARRAY)*/

     ** L#SOURCE   = 1B#1#    **
     ** L#RESULT   = 1B#10#   **
     ** L#TARGET   = 1B#100#  **
     ** L#INDEX    = 1B#1000#**
     ** L#LEFT     = 1B#10000#    **
     ** L#RIGHT    = 1B#100000#   **
     ** L#LOCATION= 1B#1000000#  **
     ** L#WIDTH    = 1B#10000000#**
     ** L#ORIGIN   = 1B#100000000#    **
     ** L#MULTIPLE= 1B#1000000000#   **


     ** ZEROW( WINDOW ) =       $ MACRO TO ZERO OUT A WINDOW
         SIZE ZZZI(PS);
         DO ZZZI = 1 TO NATS ;  $ ZERO OUT EACH ATTRIBUTE ELEMENT
             WINDOW( ZZZI ) = 0 ;
             END DO ZZZI ;
     **



     ** LEGAL#CLASS( OPERATOR , LCLASS )  =  $ TEST OP FOR LEGAL CLASS
         .F. LCLASS , 1 , LEGAL( OPERATOR - OP#FIRST + 1 )    **



/*
     CONDITIONAL COMPILATION FLAG--
         HT       WHEN SET, THE HOST AND TARGET MACHINES ARE IDENTICAL
```

```
*/
     /* VARIABLE DEFINITION SECTION
        -------- ---------- -------   */

     SUBR READ#MODULE ;
     /* THIS ROUTINE MUST BE INCLUDED IN COMPILATIONS OF (1) CODE
        GENERATORS, AND (2) UPPER LEVEL INTERFACE ROUTINES. IT IS
        THE ONLY ROUTINE THAT CODE GENERATORS MUST INCLUDE.  */


     NAMESET CODE;
     /* THIS NAMESET IS ACCESSED BY CODE GENERATORS AND UPPER
        LEVEL INTERFACE ROUTINES.   */

     /* VARIABLES USED TO DEFINE OPERATORS */

     SIZE
      OP(WS),      $ OPERATOR TYPE AS DEFINED BY OP. TYPE MACROS
      COUNT(PS),   $ COUNT OF OPERANDS FOR THIS OPERATOR
      OP2(WS),     $ TYPE OF OPERATOR 2
      COUNT2(PS),  $ CORRESPONDING COUNT FOR OPERATOR 2


     /* WINDOWS THROUGH WHICH OPERANDS MAY BE SEEN*/

     /* OPERATOR BOUND WINDOWS-*/

     SOURCE(WINDOWSZ),    RESULT(WINDOWSZ),    TARGET(WINDOWSZ),
     INDEX(WINDOWSZ),     LEFT(WINDOWSZ),      RIGHT(WINDOWSZ),
     LOCATION(WINDOWSZ),  WIDTH(WINDOWSZ),
     ORIGIN(WINDOWSZ),    MULTIPLE(WINDOWSZ),

     ARG(WINDOWSZ),       ARG2(WINDOWSZ),

     /* INDEPENDENT WINDOWS */

     QUE(WINDOWSZ),       ANY(WINDOWSZ);


     DIMS

     /* OPERATOR BOUND WINDOWS-*/

     SOURCE(NATS),    RESULT(NATS),    TARGET(NATS),
     INDEX(NATS),     LEFT(NATS),      RIGHT(NATS),
     LOCATION(NATS),  WIDTH(NATS),
     ORIGIN(NATS),    MULTIPLE(NATS),

     ARG(NATS),       ARG2(NATS),

     /* INDEPENDENT WINDOWS */

     QUE(NATS),       ANY(NATS);


     END NAMESET CODE ;

     END SUBR READ#MODULE ;

/*
```

```
*/
     SUBR UPPER↗READ ;
     /* THIS ROUTINE IS COMPILED ONLY WITH UPPER LEVEL INTERFACE*/
     NAMESET UPPER ;

     SIZE LEGAL(NMBR↗CLASSES) ;     DIMS LEGAL(NMBR↗OPS) ;
     DATA LEGAL =

     0,                                                  $ BEGIN
     0,                                                  $ COMMENT
     0,                                                  $ BLOCK
     0,                                                  $ END/EXIT
     L↗LOCATION ∨ L↗MULTIPLE ,                           $ FUNCTION ENTRY
     L↗LOCATION ∨ L↗MULTIPLE ,                           $ SUBROUTINE ENTRY
     0 ,                                                 $ NOT USED
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗ADD
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗SUB
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗GT
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗LT
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗GE
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗LE
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗EQ
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗NE
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗MUL
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗DIV
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗OR
     0 ,                                                 $ NOT USED
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗AND
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ OP↗EXOR
     0 ,                                                 $ NOT USED
     L↗SOURCE ∨ L↗RESULT ,                               $ OP↗NB
     L↗SOURCE ∨ L↗RESULT ,                               $ OP↗FB
     L↗SOURCE ∨ L↗RESULT ,                               $ OP↗NOT
     L↗LOCATION ∨ L↗MULTIPLE ∨ L↗RESULT ,                $ FUNCTION CALL
     L↗LOCATION ∨ L↗MULTIPLE ,                           $ SUBROUTINE CALL
     L↗SOURCE ∨ L↗TARGET ,                               $ SIMPLE ASSIGNMENT
     L↗TARGET ∨ L↗INDEX ∨ L↗MULTIPLE ,                   $ DATA ASSIGNMENT
     L↗SOURCE ∨ L↗TARGET ∨ L↗WIDTH ∨ L↗ORIGIN ,          $ FIELD ASSIGNMENT
     L↗SOURCE ∨ L↗TARGET ∨ L↗LEFT ∨ L↗RIGHT ,            $ BINARY I/O
     0 ,                                                 $ RETURN
     L↗SOURCE ∨ L↗WIDTH ∨ L↗ORIGIN ∨ L↗RESULT ,          $ FIELD EXTRACTION
     L↗SOURCE ∨ L↗LOCATION ,                             $ IF ... GO TO
     L↗LOCATION ,                                        $ GO TO
     L↗INDEX ∨ L↗MULTIPLE ,                              $ GOBY
     L↗SOURCE ∨ L↗INDEX ∨ L↗RESULT ,                     $ INDEXED LOAD
     L↗SOURCE ∨ L↗TARGET ∨ L↗INDEX ,                     $ INDEXED STORE
     L↗SOURCE ∨ L↗TARGET ∨ L↗INDEX ∨ L↗WIDTH ∨ L↗ORIGIN ,
               $ INDEXED FIELD ASSIGNMENT
     L↗SOURCE ∨ L↗LOCATION ,                             $ IFNOT ... GO TO
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ .CC. STRING CONCAT
     L↗LEFT ∨ L↗RIGHT ∨ L↗RESULT ,                       $ .IN. STRING SEARCH
     L↗SOURCE ∨ L↗WIDTH ∨ L↗ORIGIN ∨ L↗RESULT ,          $ .E. FIELD EXTRACTION
     L↗SOURCE ∨ L↗WIDTH ∨ L↗ORIGIN ∨ L↗RESULT ,          $ .S. FIELD EXTRACTION
     L↗SOURCE ∨ L↗TARGET ∨ L↗WIDTH ∨ L↗ORIGIN ,          $ .E. FIELD ASSIGNMENT
     L↗SOURCE ∨ L↗TARGET ∨ L↗WIDTH ∨ L↗ORIGIN ,          $ .S. FIELD ASSIGNMENT
     L↗SOURCE ∨ L↗TARGET ∨ L↗INDEX ∨ L↗WIDTH ∨ L↗ORIGIN ,
               $ .E. INDEXED FIELD ASSIGNMENT
/*
```

```
*/
    L↗SOURCE ↘ L↗TARGET ↘ L↗INDEX ↘ L↗WIDTH ↘ L↗ORIGIN ,
                $ .S. INDEXED FIELD ASSIGNMENT
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ROP↗ADD
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ROP↗SUB
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ROP↗GT
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ROP↗LT
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ROP↗GE
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ROP↗LE
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ROP↗EQ
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ROP↗NE
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ROP↗MUL
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ROP↗DIV
    L↗SOURCE ↘ L↗RESULT ,                            $ UNARY NEGATE REAL
    L↗SOURCE ↘ L↗RESULT ,                            $ FLOAT FUNCTION
    L↗SOURCE ↘ L↗RESULT ,                            $ IFIX FUNCTION
    L↗SOURCE ↘ L↗RESULT ,                            $ ABS FUNCTION
    L↗SOURCE ↘ L↗RESULT ,                            $ IABS FUNCTION
    L↗SOURCE ↘ L↗RESULT ,                            $ AINT FUNCTION
    L↗SOURCE ↘ L↗RESULT ,                            $ INT FUNCTION
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ AMOD FUNCTION
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ MOD FUNCTION
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ SIGN FUNCTION
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ ISIGN FUNCTION
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ DIM FUNCTION
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ,                    $ IDIM FUNCTION
    L↗SOURCE ↘ L↗RESULT ,                            $ EXPONENTIAL
    L↗SOURCE ↘ L↗RESULT ,                            $ NATURAL LOG
    L↗SOURCE ↘ L↗RESULT ,                            $ COMMON LOG
    L↗SOURCE ↘ L↗RESULT ,                            $ SINE
    L↗SOURCE ↘ L↗RESULT ,                            $ COSINE
    L↗SOURCE ↘ L↗RESULT ,                            $ HYPERBOLIC TANGENT
    L↗SOURCE ↘ L↗RESULT ,                            $ SQUARE ROOT
    L↗SOURCE ↘ L↗RESULT ,                            $ ARC TANGENT
    L↗LEFT ↘ L↗RIGHT ↘ L↗RESULT ;                    $ ATAN(A1/A2)


    SIZE
        OP↗DUP(WS),         $ DUPLICATE OF -OP-
        COUNT↗DUP(PS),      $ DUPLICATE OF -COUNT-
        OP2↗DUP(WS),        $ DUPLICATE OF -OP2-
        COUNT2↗DUP(PS),     $ DUPLICATE OF -COUNT2-

        OP↗PTR(PS),         $ POINTER TO CURRENT OP; SET BY LOWER LEVEL
        OP2↗PTR(PS),        $ POINTER TO CURRENT OP2

        CLASSPTR(PS),       $ USED DURING SEQUENTIAL ACCESS OF CLASSES
        CLASS2PTR(PS),      $ SEQUENTIAL ACCESS THROUGH ARG2 WINDOW

        MULTPTR(PS),        $ ACCESS TO MULTIPLES WITH GET↗ARG(C↗MULTIPLE
        MULT2PTR(PS),       $ SEQUENTIAL ACCESS WITH GET↗ARG2(C↗MULTIPLE)

        MULT↗COUNT(PS),     $ COUNTER AT A LOWER LEVEL (SUBR SELECT)

        ID↗PTR(PS),         $ POINTER TO QUE ARGUMENT; SET BY LOWER LEVEL

/*
```

```
*/
     MINSETS(PS),       $ MINUMUM ID FOR A NAMESET
     MAXSETS(PS),       $ MAXIMUM
     SELECT#CLASS(PS); $ POINTER TO CURRENT CLASS FOR SUBR SELECT


END NAMESET UPPER ;

END SUBR UPPER#READ ;



SUBR MIDDLE#READ ;
/* THIS ROUTINE IS INCLUDED IN COMPILATIONS OF UPPER AND
   LOWER LEVEL INTERFACE ROUTINES.   */

NAMESET MIDDLE ;
/* THIS NAMESET PROVIDES A COMMUNICATION PATHWAY BETWEEN
   THE UPPER LEVEL ROUTINES IN THE READ MODULE AND THE
   LOWER LEVEL, DATA-STRUCTURE-DEPENDENT ROUTINES.  */

SIZE
     HIDDEN#OP(WS),  $ CURRENT OPERATOR
     HIDDEN#COUNT(PS),  $ CURRENT ARGUMENT COUNT
     HIDDEN#PTR(PS),    $ POINTER TO CURRENT OP
     HIDDEN#ID(PS),    $ ID OF CURRENT ARGUMENT
     HIDDEN#USE(PS),   $ USE OF CURRENT ARGUMENT
     HIDDEN#SIZ(PS),   $ SIZE OF CURRENT ARGUMENT
     HIDDEN#DIM(PS),   $ DIMS OF CURRENT ARGUMENT
     HIDDEN#SET(PS),   $ NAMESET POINTER FOR CURRENT ARGUMENT
     HIDDEN#QT(PS),    $ QUANTITY TYPE OF CURRENT ARGUMENT
     HIDDEN#BYTES(PS);$ CHARACTER COUNT OF CURRENT ARGUMENT

END NAMESET MIDDLE ;


END SUBR MIDDLE#READ ;



SUBR LOWER#READ ;
/* THIS ROUTINE IS USED ONLY IN COMPILATIONS OF LOWER LEVEL
   INTERFACE ROUTINES.   */

NAMESET LOWER ;
/* THE LOWER LEVEL DATA STRUCTURE SHOULD BE DEFINED HERE */


END NAMESET LOWER ;

END SUBR LOWER#READ ;
/*
```

```
UPPER LEVEL INTERFACE ROUTINES
----- - ----- --------- --------
*/

FNCT ANOTHER(DUMMYARG);
/* THIS ROUTINE CALLS A LOWER LEVEL ONE TO PULL THE NEXT
   SUBROUTINE OR FUNCTION OFF THE INPUT FILE.  THE VALUE
   OF THE FUNCTION IS 0 IF NO ROUTINE EXISTS, ELSE 1. */

SIZE ANOTHER(1),DUMMYARG(1),YES#OR#NO(1),FIRSTENTRY(1);
DATA FIRSTENTRY = 1 ;
IF FIRSTENTRY THEN
    FIRSTENTRY = 0 ;  $ IT IS NO LONGER
    CALL OPEN#READ ;  $ OPEN THE READ FILE
    END IF FIRSTENTRY ;

CALL RD#ROUTINE( YES#OR#NO ) ; $ READ IN NEW ONE
ANOTHER = YES#OR#NO ;  IF YES#OR#NO    CALL RESET#READ ;
RETURN ;

END FNCT ANOTHER ;



SUBR RESET#READ ;
/* THIS ROUTINE RESETS THE OPERATOR AND ARGUMENT STACKS. */

ACCESS CODE, UPPER  ;

OP = OP#BEGIN ;       COUNT = 0;  $ SET PRIMARY OPERATOR STACK
CALL INIT#STACK( OP#PTR ) ;  OP2#PTR = OP#PTR ;
OP#DUP = OP#BEGIN ;  OP2#DUP = OP#BEGIN ; $ INTERNAL IDENTICAL
ZEROW( QUE ) ;  $ INITIALIZE THE ARGUMENT STACK
CALL FIRST#ID( ID#PTR ) ;
RETURN ;

END SUBR RESET#READ ;



SUBR POP#OP ;
/* THIS ROUTINE IMPLEMENTS THE POP COMMAND; IT GETS THE NEXT
   OPERATOR ON THE STACK, INITIALIZES COUNTERS FOR THE
   GET#ARG COMMAND AND VALUES FOR THE POP2 AND PUSH2 COMMANDS */

ACCESS CODE , UPPER , MIDDLE ;
IF OP#DUP = OP#END    RETURN;  $ NO-OP IF DONE
CALL UP#STACK( OP#PTR ) ; $ GET NEXT OP; SET HIDDEN VALUES
IF OP#PTR = 0    HIDDEN#OP = OP#END ;  $ SIGNAL IF NO MORE
OP = HIDDEN#OP ;  COUNT=HIDDEN#COUNT ; $ COPY TO USER VALUES
$ AND SAVE ANOTHER COPY FOR THE INTERFACE TO USE
OP#DUP = HIDDEN#OP ;  COUNT#DUP = HIDDEN#COUNT ;
CLASSPTR = 0 ;  MULTPTR = 0 ;  $ COUNTERS FOR GET#ARG
OP2 = 0 ;  COUNT2 = 0 ;  OP2#PTR = OP#PTR ; $ SET FOR POP2
OP2#DUP = HIDDEN#OP ;   $ DUPLICATE FOR PO2 AND PUSH2
RETURN ;

END SUBR POP#OP ;


/*
```

```
*/
    SUBR NEXT↑OP2 ;
    /* THIS ROUTINE IMPLEMENTS THE POP2 COMMAND; IT GETS THE NEXT
       OPERATOR IN THE SEQUENCE FOLLOWING -OP-, AND INITIALIZES
       A COUNTER FOR THE GET↑ARG2 COMMAND.  */

    ACCESS CODE , UPPER , MIDDLE ;
    IF OP2↑DUP = OP↑END    RETURN ;  $ NO-OP IF DONE
    CALL UP↑STACK( OP2↑PTR ) ; $ GET NEXT OP2; SET HIDDEN VALUES
    IF OP↑PTR = 0   HIDDEN↑OP = OP↑END ; $ STACK EXHAUSTED
    OP2 = HIDDEN↑OP ;   COUNT2 = HIDDEN↑COUNT ;  $ COPY
    $ AND SAVE ANOTHER COPY FOR THE INTERFACE TO USE
    OP2↑DUP = HIDDEN↑OP ;  COUNT2↑DUP = HIDDEN↑COUNT ;
    CLASS2PTR = 0 ;  MULT2PTR = 0 ;  $ SET FOR GET↑ARG2

    RETURN ;

    END SUBR NEXT↑OP2 ;


    SUBR LAST↑OP2 ;
    /* THIS ROUTINE IMPLEMENTS THE PUSH2 COMMAND; IT GETS THE NEXT
       OPERATOR IN THE SEQUENCE FOLLOWING -OP-, AND INITIALIZES
       A COUNTER FOR THE GET↑ARG2 COMMAND.  */

    ACCESS CODE , UPPER , MIDDLE ;
    IF OP2↑DUP = OP↑BEGIN    RETURN ;  $ STACK EXHAUSTED
    CALL DOWN↑STACK( OP2↑PTR ) ; $ GET NEXT OP2; SET HIDDEN VALUES
    IF OP2↑PTR = 0    HIDDEN↑OP = OP↑BEGIN ; $ STACK EXHAUSTED
    OP2 = HIDDEN↑OP ;   COUNT2 = HIDDEN↑COUNT ;  $ COPY
    $ AND SAVE ANOTHER COPY FOR THE INTERFACE TO USE
    OP2↑DUP = HIDDEN↑OP ;  COUNT2↑DUP = HIDDEN↑COUNT ;
    CLASS2PTR = 0 ;  MULT2PTR = 0 ;  $ SET FOR GET↑ARG2

    RETURN ;

    END SUBR LAST↑OP2 ;


    SUBR ALL↑ARGS ;
    /* THIS ROUTINE OPENS WINDOWS FOR EACH OPERAND DEFINED FOR
       THE CURRENTLY DISPLAYED OP.  */

    ACCESS UPPER ;

    SIZE CLASS(PS);

    DO CLASS = 1 TO NMBR↑CLASSES ;
        IF LEGAL↑CLASS( OP↑DUP , CLASS )  THEN
            $ IF THE CLASS IS DEFINED FOR THIS OP, OPEN THE WINDOW
            CALL OPEN↑ARG( CLASS ) ;
            END IF ;
        END DO CLASS ;

    RETURN ;

    END SUBR ALL↑ARGS ;


/*
```

```
*/
    SUBR OPEN↗ARG( ARGCLASS ) ;
    /* THIS ROUTINE IMPLEMENTS THE GET↗ARG COMMAND.  THUS, IT
       MAY BE CALLED WITH A ZERO ARGCLASS ( FOR SEQUENTIAL
       PRODUCTION OF ARGUMENTS IN THE WINDOW -ARG-) OR WITH
       THE ARGCLASS SET TO A CLASS (IN WHICH CASE THE CLASS
       WINDOW IS OPENED.)   */

    ACCESS CODE , UPPER , MIDDLE ;

    SIZE ARGCLASS(PS),  $ ZERO OR AN ARGUMENT CLASS
        M↗COUNT(PS);   $ MULTIPLE COUNTER WHEN ARGCLASS = 0

    /* CHECK FOR LEGAL CALLING PARAMETER.  */
    IF ARGCLASS < 0  ∨  ARGCLASS > NMBR↗CLASSES THEN
        TEXTL( ≠IMPROPER GET↗ARG COMMAND ≠ ) ENDL  RETURN ;
        END IF ARGCLASS ;

    $ SET UP OPERATOR AND COUNT FOR LOWER LEVEL ROUTINES.
    HIDDEN↗OP = OP↗DUP ;  HIDDEN↗COUNT = COUNT↗DUP ;
    HIDDEN↗PTR = OP↗PTR ; $ POINT TO OP↗DUP

    IF ARGCLASS = 0 THEN

    /READIN( 0 ) /      $ SEQUENTIAL ACCESS
            /* FIRST SEE IF CURRENT CLASS IS THE MULTIPLE CASE */
            IF CLASSPTR = C↗MULTIPLE THEN
                $ MOVE ON TO THE NEXT MULTIPLE ARGUMENT
                M↗COUNT = M↗COUNT + 1 ;  MULT↗COUNT = M↗COUNT ;
                SELECT↗CLASS = C↗MULTIPLE ; $ SIGNAL SELECT
                CALL SELECT( ARG ) ;  $ OPEN WINDOW
                $ IF FOUND, RETURN; ELSE GO ON TO NEXT ARG
                IF ARG( ID ) ¬= 0   RETURN ;
                END IF CLASSPTR ;

        DO CLASSPTR = CLASSPTR+1  TO NMBR↗CLASSES ;
            $ FIND OUT IF THIS ONE IS DEFINED
            IF LEGAL↗CLASS( OP↗DUP , CLASSPTR ) THEN
                IF CLASSPTR = C↗MULTIPLE THEN
                    $ IF ITS MULTIPLE CLASS, START COUNT AND BRANCH
                    M↗COUNT=0;  GO TO READIN( 0 ) ;
                    END IF CLASSPTR ;
                $ FOR ALL OTHER OPERANDS, GO OPEN THE WINDOW
                SELECT↗CLASS = CLASSPTR ; $ SIGNAL SUBR SELECT
                CALL SELECT( ARG ) ;
                IF ARG( ID ) = 0 CONT DO ; $ IN CASE OF OPTIONAL ARGS
                RETURN;
                END IF ;
            END DO CLASSPTR ;
        ZEROW( ARG ) ;  $ IF NO MORE OPERANDS, ZERO WINDOW

        RETURN;

/*
```

```
*/
     ELSE

     SELECT.CLASS = ARGCLASS ;  $ SET SWITCH FOR SELECT
     GO TO READIN( ARGCLASS ) IN  1  TO  NMBR.CLASSES  ;


     / READIN( C.SOURCE ) /
         CALL SELECT( SOURCE ) ;
         RETURN ;
     / READIN( C.RESULT ) /
         CALL SELECT( RESULT ) ;
         RETURN ;
     / READIN( C.TARGET ) /
         CALL SELECT( TARGET ) ;
         RETURN ;
     / READIN( C.INDEX ) /
         CALL SELECT( INDEX ) ;
         RETURN ;
     / READIN( C.LEFT ) /
         CALL SELECT( LEFT ) ;
         RETURN ;
     / READIN( C.RIGHT ) /
         CALL SELECT( RIGHT ) ;
         RETURN ;
     / READIN( C.LOCATION ) /
         CALL SELECT( LOCATION ) ;
         RETURN ;
     / READIN( C.WIDTH ) /
         CALL SELECT( WIDTH ) ;
         RETURN ;
     / READIN( C.ORIGIN ) /
         CALL SELECT( ORIGIN ) ;
         RETURN ;
     / READIN( C.MULTIPLE ) /
         $ MOVE TO NEXT  MULTIPLE ARGUMENT
         MULTPTR = MULTPTR + 1 ;  MULT.COUNT = MULTPTR ;
         CALL SELECT( MULTIPLE ) ;
         RETURN ;

         END IF ARGCLASS ;

     END SUBR OPEN.ARG ;



/*
```

```
*/
    SUBR SET↗ARG2( ARG2CLASS ) ;
    /* THIS ROUTINE IMPLEMENTS THE GET↗ARG2 COMMAND.  THUS, IT
       MAY BE CALLED WITH A ZERO ARGCLASS ( FOR SEQUENTIAL
       PRODUCTION OF ARGUMENTS IN THE WINDOW -ARG2-) OR WITH
       THE ARG2CLASS SET TO A CLASS .     */

    ACCESS CODE , UPPER , MIDDLE ;

    SIZE ARG2CLASS(PS),  $ ZERO OR AN ARGUMENT CLASS
        M↗COUNT(PS);    $ COUNTER FOR MULTIPLES WHEN ARG2CLASS=0

    /* CHECK FOR LEGAL CALLING PARAMETER.  */
    IF ARG2CLASS < 0  ∨  ARG2CLASS > NMBR↗CLASSES THEN
        TEXTL( ≠IMPROPER GET↗ARG2 COMMAND ≠ ) ENDL  RETURN ;
        END IF ARG2CLASS ;
    IF OP2↗PTR = OP↗PTR   RETURN;  $ HAS THERE BEEN POP2 OR PUSH2

    $ SET UP OPERATOR AND COUNT FOR LOWER LEVEL ROUTINES.
    HIDDEN↗OP = OP2↗DUP ;  HIDDEN↗COUNT = COUNT2↗DUP ;
    HIDDEN↗PTR = OP2↗PTR ; $ POINT TO THIS OP2

    IF ARG2CLASS = 0 THEN

    /READIN( 0 ) /       $ SEQUENTIAL ACCESS
            /* FIRST SEE IF CURRENT CLASS IS THE MULTIPLE CASE */
            IF CLASS2PTR = C↗MULTIPLE THEN
                $ MOVE ON TO THE NEXT MULTIPLE ARGUMENT
                M↗COUNT = M↗COUNT + 1 ;  MULT↗COUNT = M↗COUNT ;
                SELECT↗CLASS = C↗MULTIPLE ; $ SIGNAL SELECT
                CALL SELECT( ARG2 ) ;  $ OPEN WINDOW
                $ IF FOUND, RETURN; ELSE GO ON TO NEXT ARG2
                IF ARG2( ID ) ¬= 0   RETURN ;
                END IF CLASS2PTR ;

        DO CLASS2PTR = CLASS2PTR+1  TO NMBR↗CLASSES ;
            $ FIND OUT IF THIS ONE IS DEFINED
            IF LEGAL↗CLASS( OP↗DUP , CLASS2PTR ) THEN
                IF CLASS2PTR = C↗MULTIPLE THEN
                    $ IF ITS MULTIPLE CLASS, START COUNT AND BRANCH
                    M↗COUNT=0;  GO TO READIN( 0 ) ;
                    END IF CLASS2PTR ;
                $ FOR ALL OTHER OPERANDS, GO OPEN THE WINDOW
                SELECT↗CLASS = CLASS2PTR ; $ SIGNAL SUBR SELECT
                CALL SELECT( ARG2 ) ;
                IF ARG2( ID ) = 0 CONT DO ; $ IN CASE OF OPTIONAL ARGS
                RETURN;
                END IF ;
            END DO CLASS2PTR ;
        ZEROW( ARG2 ) ;  $ IF NO MORE OPERANDS, ZERO WINDOW

        RETURN;

/*
```

```
*/
    ELSEIF ARG2CLASS = C#MULTIPLE THEN

        MULT2PTR = MULT2PTR + 1 ;  MULT#COUNT = MULT2PTR ;
        SELECT#CLASS = ARG2CLASS ; $ SIGNAL SELECT
        CALL SELECT( ARG2 ) ;  $ GET NEXT MULTIPLE ARGUMENT
        RETURN ;

    ELSE        $ ALL OTHER ARGUMENTS

        SELECT#CLASS = ARG2CLASS ; $ SIGNAL SELECT
        CALL SELECT( ARG2 ) ;
        RETURN ;

        END IF ARG2CLASS ;

    END SUBR SET#ARG2 ;



    SUBR ANY#ARG( IDENT ) ;
    /* THIS ROUTINE IMPLEMENTS THE GET#ANY COMMAND.
       IT ZEROES THE WINDOW AND CALLS A LOWER LEVEL
       ROUTINE TO GET THE STATIC ATTRIBUTES.    */

    SIZE IDENT(PS);  $ UNIQUE ID OF OPERAND DESIRED

    ACCESS CODE ;

    ZEROW( ANY ) ;
    ANY( ID ) = IDENT ;
    CALL STATIC( ANY ) ;

    RETURN ;

    END SUBR ANY#ARG ;



    SUBR QUEUE#UP ;
    /* THIS ROUTINE IMPLEMENTS THE POP#QUEUE COMMAND.
       IT ZEROES THE WINDOW AND CALLS A LOWER LEVEL
       ROUTINE TO GET THE STATIC ATTRIBUTES.    */

    ACCESS CODE , UPPER ;

    IF ID#PTR  CALL NEXT#ID( ID#PTR ) ;  $ POP THE ARGUMENT STACK

    ZEROW( QUE ) ;  QUE( ID ) = ID#PTR ;
    IF ID#PTR    CALL STATIC( QUE ) ;

    RETURN ;

    END SUBR QUEUE#UP ;


/*
```

```
*/
     SUBR SELECT( WINDOW ) ;
     /* GIVEN A WINDOW AND A CLASS, THIS ROUTINE WILL
        SELECT A LOWER LEVEL CLASS ROUTINE TO PROVIDE
        THE ID AND USE ATTRIBUTES OF THE ARGUMENT, AND
        THEN CALL THE STATIC ROUTINE FOR THE OTHER
        ATTRIBUTES.   */

     SIZE WINDOW(WINDOWSZ), NON#MULT(PS), ICLASS(PS);
     DIMS WINDOW(NATS);

     ACCESS UPPER , MIDDLE ;

     ZEROW( WINDOW ) ;
     IF LEGAL#CLASS( HIDDEN#OP , SELECT#CLASS ) = 0    RETURN ;

     GO TO READIN( SELECT#CLASS )  IN 1 TO NMBR#CLASSES ;

     / READIN( C#SOURCE ) /
         CALL RD#SOURCE ;
         GO TO STATICS ;
     / READIN( C#RESULT ) /
         CALL RD#RESULT ;
         GO TO STATICS ;
     / READIN( C#TARGET ) /
         CALL RD#TARGET ;
         GO TO STATICS ;
     / READIN( C#INDEX ) /
         CALL RD#INDEX ;
         GO TO STATICS ;
     / READIN( C#LEFT ) /
         CALL RD#LEFT ;
         GO TO STATICS ;
     / READIN( C#RIGHT ) /
         CALL RD#RIGHT ;
         GO TO STATICS ;
     / READIN( C#LOCATION ) /
         CALL RD#LOCATION ;
         GO TO STATICS ;
     / READIN( C#WIDTH ) /
         CALL RD#WIDTH ;
         GO TO STATICS ;
     / READIN( C#ORIGIN ) /
         CALL RD#ORIGIN ;
         GO TO STATICS ;
     / READIN( C#MULTIPLE ) /
     NON#MULT = 0 ;  $ COUNT NUMBER OF NON-MULTIPLE CLASSES
     DO ICLASS = 1 TO NMBR#CLASSES ;
         IF ICLASS = C#MULTIPLE   CONT DO ICLASS ;
         IF LEGAL#CLASS( HIDDEN#OP , ICLASS )  NON#MULT = NON#MULT+1;
         END DO ICLASS ;
     $IF THE ARGUMENT COUNT HAS BEEN EXCEEDED, JUST EXIT
     IF NON#MULT + MULT#COUNT  >  HIDDEN#COUNT    RETURN ;
         CALL RD#MULTIPLE( MULT#COUNT ) ;
         GO TO STATICS ;
/*
```

```
*/
     /STATICS/
          $ DEFINE THE WINDOW ATTRIBUTES FOUND BY LOWER LEVEL
          WINDOW( ID ) = HIDDEN→ID ;
          WINDOW( USE ) = HIDDEN→USE ;
          $ NOW PICK UP THE STATIC ATTRIBUTES
          CALL STATIC( WINDOW ) ;
          RETURN ;

     END SUBR SELECT ;


     SUBR VALU( IDENT , V ) ;
     /* THIS ROUTINE RETURNS THE VALUE OF AN ARGUMENT IN -V- WHEN
        SUPPLIED WITH ITS ID IN -IDENT-.  THE CALLING ROUTINE IS
        RESPONSIBLE FOR SEEING THAT -V- IS SIZED LARGE ENOUGH TO
        ACCOMMODATE THE VALUE.    */

     SIZE IDENT(PS), $ ID OF ARGUMENT DESIRED
          V(WS), $ VALUE RETURNED (SIZE IS INCORRECT AND IMMATERIAL)
          WINDOW(WINDOWSZ); $ TEMPORARY WINDOW
     DIMS WINDOW(NATS);
     ACCESS MIDDLE ;

     /* NOW GET THE STATIC PROPERTIES OF THIS ARGUMENT */
     WINDOW( ID ) = IDENT ;  CALL STATIC( WINDOW ) ;
     $ SET QT AND ID FOR ROUTINES RD→BITS AND RD→CHARS.
     HIDDEN→ID = IDENT ;    HIDDEN→QT = WINDOW( QT ) ;

     IF           $ SEE WHETHER VALUE IS TO BE GIVEN AS BIT OR CHAR. STR
          HIDDEN→QT = Q→BITS
          ∨ HIDDEN→QT = Q→NEGATE
.+HT      ∨ HIDDEN→QT = Q→REAL
.+HT      ∨ HIDDEN→QT = Q→CODES
                THEN
                    CALL RD→BITS( V ) ; $ LOWER LEVEL GETS IT.
     ELSEIF
          HIDDEN→QT = Q→CHARS
          ∨ HIDDEN→QT = Q→VAR
          ∨ HIDDEN→QT = Q→PARAM
          ∨ HIDDEN→QT = Q→LABEL
          ∨ HIDDEN→QT = Q→EXTERNAL
.-HT      ∨ HIDDEN→QT = Q→REAL
.-HT      ∨ HIDDEN→QT = Q→CODES
                THEN
                    IF SORG V < WINDOW(BYTES)*.CS.+.SO.+.SL. THEN
                        TEXTL(≠INSUFFICIENT SPACE TO TRANSFER VALUE ≠)
                        TEXTL(≠IN A CALL TO SUBR VALU≠) ENDL
                    ELSE
                        CALL RD→CHARS( V ) ;  $ GET IT
                        END IF SORG V ;
     ELSE
          $ MUST BE TEMPORARY
          V=0 ;  $  NO VALUE, AND SHOULD NOT HAVE CALLED US
          END IF ;
     RETURN ;

     END SUBR VALU ;

/*
```

```
*/
    SUBR NM( SETS , NAME#SET ) ;
    /* THIS ROUTINE TAKES THE FIRST CALLING PARAMETER, SETS,
       AND USES IT TO FIND THE CHARACTER STRING OF A CORRE-
       SPONDING NAMESET, WHICH IT RETURNS IN THE SECOND
       PARAMETER, NAME#SET   */

    ACCESS UPPER , MIDDLE ;

    SIZE SETS(PS),  $ A UNIQUE ID FOR A NAMESET
        NAME#SET( .SDS. NMSET#LEN ) ; $ NAME OF NAMESET

    /* WE ASSUME THAT THE NAMESET MUST HAVE BEEN PREVIOUSLY
       REFERENCED (THROUGH ACCESS TO ONE OF ITS MEMBERS) AND
       THAT THE UNIQUE IDS ARE CONTIGUOUS WHEN WE TEST FOR
       THEIR LEGALITY HERE.  */
    IF SETS > MAXSETS  v  SETS < MINSETS  THEN
        TEXTL(≠ILLEGAL REQUEST FOR A NAMESET NAME≠) ENDL
        RETURN ;
        END IF SETS ;
    IF SORG NAME#SET < NMSET#LEN*.CS.+.SO.+.SL. THEN
        TEXTL(≠INSUFFICIENT SPACE IN ARGUMENT IN A CALL ≠)
        TEXTL(≠TO SUBR NM≠)  ENDL
        RETURN ;
        END IF SORG NAME#SET ;

    HIDDEN#SET = SETS ;  $ SUBR RD#NM WILL PICK THIS UP

    CALL RD#NM( NAME#SET ) ;  $ RETRIEVE NAMESET NAME

    RETURN ;

    END SUBR NM ;



    SUBR STATIC( WINDOW ) ;
    /* THIS ROUTINE CHOOSES THE APPROPRIATE ROUTINES TO
       CALL IN ORDER TO DEFINE THE STATIC ATTRIBUTES OF
       THE QUANTITY REQUESTED.   A QUANTITY IS REQUESTED
       BY HAVING ITS ID SET PROPERLY IN THE CALLING
       PARAMETER, THE WINDOW.   IF ANY NEW STATIC ATTRI-
       BUTES ARE DEFINED, THEN THIS IS THE PLACE TO ADD
       A CALL TO A ROUTINE THAT SUPPLIES THE ATTRIBUTE.*/

    ACCESS UPPER , MIDDLE ;

    SIZE WINDOW(WINDOWSZ), $ CALLING QUANTITY DESIRED
        TEMP#VALUE(WINDOWSZ); $ TEMPORARY FOR VALUE ATTRIBUTE
    DIMS WINDOW( NATS ) ;

    HIDDEN#ID = WINDOW( ID ) ; $ PICK UP AND SAVE
    CALL CONFIRM#ID ;  $ $ LOWER LEVEL CONFIRMS EXISTENCE
    IF HIDDEN#ID = 0 THEN
        TEXTL(≠UNKNOWN ID REQUESTED OR BAD INPUT DATA≠) ENDL
        RETURN ;
        END IF HIDDEN#ID ;
/*
```

```
*/

     CALL RD#QT ;  WINDOW( QT ) = HIDDEN#QT ;

     CALL RD#SIZ ;  WINDOW( SIZ ) = HIDDEN#SIZ ;

     IF HIDDEN#QT = Q#VAR THEN
          $IF ITS A VARIABLE, IT MAY BE DIMENSIONED
          CALL RD#DIM ;  WINDOW( DIM ) = HIDDEN#DIM ;
          END IF HIDDEN#QT = Q#VAR ;

     IF HIDDEN#QT = Q#VAR  | HIDDEN#QT = Q#EXTERNAL  THEN
          $ IF ITS A VARIABLE OR EXTERNAL, THEN IT MAY BE IN A NAMESET
          CALL RD#SET ;  WINDOW( NSID ) = HIDDEN#SET ;
          /* NOW SET THE LIMITS ON ALLOWED NAMESETS FOR LATER USE
             BY SUBR NM.   */
          IF HIDDEN#SET ¬= 0 THEN  $ IF A NAMESET WAS FOUND
               IF HIDDEN#SET > MAXSETS  MAXSETS = HIDDEN#SET ;
               IF HIDDEN#SET < MINSETS  MINSETS = HIDDEN#SET ;
               END IF ;
          END IF ;

          IF
          HIDDEN#QT = Q#BITS
        | HIDDEN#QT = Q#NEGATE
.+HT    | HIDDEN#QT = Q#REAL
.+HT    | HIDDEN#QT = Q#CODES
          & WINDOW( SIZ ) <= WINDOWSZ  THEN
               $ IF VALUE WILL FIT AND ITS A BIT STRING, GET IT
               CALL RD#BITS( TEMP#VALUE ) ;  WINDOW( VALUE )=TEMP#VALUE;
               END IF ;

     IF
          HIDDEN#QT = Q#CHARS
        | HIDDEN#QT = Q#VAR
        | HIDDEN#QT = Q#PARAM
        | HIDDEN#QT = Q#LABEL
        | HIDDEN#QT = Q#EXTERNAL
.-HT    | HIDDEN#QT = Q#REAL
.-HT    | HIDDEN#QT = Q#CODES
               THEN
               $ IF ITS A CHARACTER STRING, GET THE CHARACTER COUNT
               CALL RD#BYTES ;  WINDOW( BYTES ) = HIDDEN#BYTES ;
               END IF ;

     RETURN ;

     END SUBR STATIC ;



/*
```

SPECIFICATIONS FOR THE LOWER LEVEL ROUTINES

ROUTINE	PURPOSE	INPUT	OUTPUT
OPEN.READ	OPEN THE FILE FROM WHICH INPUT WILL COME; ABORT ON ERRORS	NONE	NONE
RD.ROUTINE	READ A NEW ROUTINE INTO LOWER LEVEL DATA STRUCTURE; IF THERE IS NO ROUTINE, RETURN PRESENCE=0, ELSE =1	NONE	PRESENCE*
INIT.STACK	INITIALIZE A POINTER TO POSITION BEFORE FIRST OPERATOR (ANY VALUE IS OK)	NONE	POINTER*
UP.STACK	MOVE POINTER TO NEXT OPERATOR ON STACK AND SET NEW OP AND COUNT; POINTER=0 AT END OF STACK	POINTER*	POINTER* HIDDEN.OP HIDDEN.COUNT
DOWN.STACK	MOVE POINTER BACK ONE OPERATOR; SET OP,COUNT; POINTER=0 AT TOP OF STACK	POINTER*	POINTER* HIDDEN.OP HIDDEN.COUNT
FIRST.ID	INITIALIZE POINTER TO POSITION BEFORE FIRST OPERAND ON ARGUMENT STACK (BUT NOT A 0)	NONE	POINTER*
NEXT.ID	MOVE TO NEXT OPERAND ON ARGUMENT STACK; RETURN THE NEW POINTER (A UNIQUE ID); POINTER=0 AT END OF STACK	POINTER*	POINTER*
CONFIRM.ID	CONFIRM THAT AN OPERAND ID IS LEGAL; ZERO IT ON RETURN IF IT IS NOT	HIDDEN.ID	HIDDEN.ID
RD.SOURCE	SUPPLY THE ID AND USE ATTRIBUTES OF THE SOURCE ARGUMENT OF THE CURRENT OPERATOR	HIDDEN.OP HIDDEN.PTR	HIDDEN.ID HIDDEN.USE
RD.RESULT	SUPPLY THE ID AND USE ATTRIBUTES OF THE RESULT ARGUMENT OF THE CURRENT OPERATOR	HIDDEN.OP HIDDEN.PTR	HIDDEN.ID HIDDEN.USE

RD↑TARGET	SUPPLY THE ID AND USE ATTRIBUTES OF THE TARGET ARGUMENT OF THE CURRENT OPERATOR	HIDDEN↑OP HIDDEN↑PTR	HIDDEN↑ID HIDDEN↑USE
RD↑INDEX	SUPPLY THE ID AND USE ATTRIBUTES OF THE INDEX ARGUMENT OF THE CURRENT OPERATOR	HIDDEN↑OP HIDDEN↑PTR	HIDDEN↑ID HIDDEN↑USE
RD↑LEFT	SUPPLY THE ID AND USE ATTRIBUTES OF THE LEFT ARGUMENT OF THE CURRENT OPERATOR	HIDDEN↑OP HIDDEN↑PTR	HIDDEN↑ID HIDDEN↑USE
RD↑RIGHT	SUPPLY THE ID AND USE ATTRIBUTES OF THE RIGHT ARGUMENT OF THE CURRENT OPERATOR	HIDDEN↑OP HIDDEN↑PTR	HIDDEN↑ID HIDDEN↑USE
RD↑LOCATION	SUPPLY THE ID AND USE ATTRIBUTES OF THE LOCATION ARGUMENT OF THE CURRENT OPERATOR	HIDDEN↑OP HIDDEN↑PTR	HIDDEN↑ID HIDDEN↑USE
RD↑WIDTH	SUPPLY THE ID AND USE ATTRIBUTES OF THE WIDTH ARGUMENT OF THE CURRENT OPERATOR	HIDDEN↑OP HIDDEN↑PTR	HIDDEN↑ID HIDDEN↑USE
RD↑ORIGIN	SUPPLY THE ID AND USE ATTRIBUTES OF THE ORIGIN ARGUMENT OF THE CURRENT OPERATOR	HIDDEN↑OP HIDDEN↑PTR	HIDDEN↑ID HIDDEN↑USE
RD↑MULTIPLE	SUPPLY THE ID AND USE ATTRIBUTES OF THE DESIRED MULTIPLE ARGUMENT OF THE CURRENT OPERATOR	HIDDEN↑OP HIDDEN↑PTR WHICH↑MULT*	HIDDEN↑ID HIDDEN↑USE
RD↑QT	PROVIDE THE QUANTITY TYPE OF THE OPERAND WITH THE SPECIFIED ID.	HIDDEN↑ID	HIDDEN↑QT
RD↑SIZ	PROVIDE THE SIZE OF THE OPERAND WITH THE SPECIFIED ID.	HIDDEN↑ID	HIDDEN↑SIZ
RD↑DIM	PROVIDE THE DIMENSION OF THE ARGUMENT WITH THE SPECIFIED ID .	HIDDEN↑ID	HIDDEN↑DIM

RD↑SET	PROVIDE THE NAMESET IDENTIFIER OF THE ARGUMENT WITH THE SPECIFIED ID.	HIDDEN↑ID	HIDDEN↑SET
RD↑BYTES	PROVIDE THE CHARACTER COUNT OF THE ARGUMENT WITH THE SPECIFIED ID.	HIDDEN↑ID	HIDDEN↑BYTES
RD↑BITS	PROVIDE THE VALUE OF THE ARGUMENT WITH THE SPECIFIED ID.	HIDDEN↑ID	VALUE*
RD↑CHARS	PROVIDE THE CHARACTER STRING VALUE OF THE ARGUMENT WITH THE SPECIFIED ID (THE ORIGIN FIELD OF VALUE* HAS BEEN SET)	HIDDEN↑ID	VALUE*
RD↑NM	PROVIDE THE CHARACTER STRING FOR THE NAMESET HAVING THE SPECIFIED IDENTIFIER (HIDDEN↑SET).	HIDDEN↑SET	NAMESET*

* INPUT AND OUTPUT QUANTITIES FOR THESE ROUTINES THAT ARE MARKED WITH AN ASTERISK ARE FORMAL PARAMETERS. ALL OTHERS ARE GLOBAL.

*/
/*

EXAMPLES OF CODE THAT USE THE INTERFACE

EXAMPLE 1- SEARCH OF ALL OPERANDS TO FIND THOSE SIZED LARGER THAN THE MACHINE WORD SIZE

```
INIT↛CODE ; $ RESET QUEUE
 -
 -
 -
POP↛QUEUE ; $ POP FIRST ARGUMENT ON THE QUEUE
WHILE QUE(ID) ;  $ SO LONG AS ARGUMENTS KEEP COMING
    IF QUE(SIZ) > WS THEN    $ IF OPERAND IS LARGER
        $ ... DO THE WORK
        END IF QUE ;
    POP↛QUEUE ; $ CONTINUE LOOKING AT OPERANDS
    END WHILE ;
```

EXAMPLE 2A- CHECK WHETHER A RESULT TEMPORARY IS USED ONLY IN THE NEXT OPERATION

```
POP ;   $ POP NEXT OPERATOR OFF THE STACK
 -
 -
 -
GET↛ARG( RESULT ) ;  $ WINDOW THE RESULT TEMPORARY
IF RESULT( USE ) = 1 THEN  $ IF LAST OP USING TEMP. IS ONE AWAY
    $ ACT ON THIS KNOWLEDGE
    END IF RESULT ;
```

EXAMPLE 2B- TEST ALL OPERANDS OF AN OPERATOR FOR THE PRESENCE OF FORMAL PARAMETERS

```
POP ;   $ POP NEXT OPERATOR OFF THE STACK
 -
 -
 -
WHILE COUNT ;  $ WHILE COUNT IS GREATER THAN ZERO
    GET↛ARG( 0 ) ;  $ PICK UP THE NEXT OPERAND IN -ARG- WINDOW
    IF ARG( QT ) = Q↛PARAM THEN  $ IF THIS IS FORMAL PARAMETER
        $ ACT ON THIS KNOWLEDGE
        END IF ARG ;
    COUNT = COUNT - 1 ;  $ DECREMENT NUMBER OF OPERANDS
    END WHILE COUNT ;
```

EXAMPLE 3- THE PRINCIPAL CODE GENERATION LOOP

```
    WHILE 1  ;              $ CONTINUOUS, SEQUENTIAL ACCESS

        POP ;               $ GET A NEW OPERATOR OFF THE STACK

        GET→ALL ;           $ AND RETRIEVE ALL ITS OPERANDS

        GO TO EMIT( OP ) IN OP→FIRST TO OP→LAST ;
         -
         -
         -
         -
         -
        THE FOLLOWING CODE EMISSION FRAGMENT (ALONG WITH
        ITS CALLED SUBROUTINE) IS INCLUDED TO ILLUSTRATE THE
        EASE OF ACCESSING OPERAND ATTRIBUTES.  FOR EXAMPLE,
        THE EXPRESSION - FROM( SIZ ) - IN THE SUBROUTINE IS
        ACCOMPLISHED WITH THE MORE CUMBERSOME EXPRESSION -
                SYZE VOA(INP2 VOA(VOAEP) )
        IN THE PRESENT CODE GENERATORS.
/ EMIT( OP→ASIN ) /
            CALL EMIT→MOVE( SOURCE , TARGET ) ;
            CONT WHILE ;
         -
         -
         -
         -
         -
/ EMIT( OP→END ) /      $ NO MORE OPS ON STACK, SO EXIT
            QUIT WHILE ;
         -
         -
         -
         -
         -
        END WHILE 1 ;


    SUBR EMIT→MOVE( FROM , TO )  ;              $ EMIT CODE FOR MOVE

    SIZE FROM(WS),TO(WS);  DIMS FROM(NATS),TO(NATS);

    IF FROM( SIZ ) <= WS  ^  TO( SIZ ) <= WS THEN
        $ EMIT IN LINE CODE
    ELSE
        $EMIT OFF LINE CALL TO LIBRARY ROUTINE
        END IF FROM  ;

    RETURN ;
    END SUBR EMIT→MOVE ;
```

EXAMPLE 4A- LOOKING AHEAD TO SEE IF THE OPERATOR FOLLOWING
A RELATIONAL OP IS AN IF↛OP, AND EMITTING SPECIAL CODE

```
POP;    GET↛ALL;  $ OPEN WINDOWS OF ALL OPERANDS OF -OP-.
    -
    -
    -
    -
/ EMIT( OP↛GT ) /   $    >   OPERATOR

    POP2 ;   $ GET THE NEXT OPERATOR
    GET↛ARG2(SOURCE);  $ GET THE INPUT ARGUMENT TO THE NEXT OP

    $ THE FOLLOWING QUESTION ASKS WHETHER THE NEXT OPERATOR IS
    $ AN OP↛IF AND WHETHER THE OUTPUT FROM THE OP↛GT IS THE
    $ SAME AS THE INPUT TO THE OP↛IF.
    IF OP2 = OP↛IF  ^  RESULT( ID ) = ARG2( ID ) THEN
            $WE CAN EMIT SPECIAL CODE, SO GET JUMP LOCATION
            GETARG2( LOCATION );
            CALL EMIT↛JUMPGT( LEFT , RIGHT , LOCATION ) ; $ EMIT
            END IF OP2 ;
```

EXAMPLE 4B- SEARCHING AHEAD TO FIND THE NUMBER OF USES OF
A TEMPORARY PRODUCED BY THE CURRENT OP.

```
POP;  GET↛ALL;  $ GET ALL OPERANDS OF CURRENT OP
 -
 -
 -
USE↛COUNT = 0 ; $ NUMBER OF USES OF RESULT TEMPORARY
WHILE RESULT( USE ) ; $ SO LONG AS THERE ARE OPS TO LOOK AT

    POP2 ;  $ GET THE NEXT SUCCEEDING OPERATOR
    WHILE COUNT2 ; $ AND LOOK AT ALL ITS OPERANDS

        IF RESULT( ID ) = ARG2( ID ) THEN $ IF IDS ARE SAME
            USE↛COUNT = USE↛COUNT + 1 ;
            END IF RESULT( ID ) ;
        COUNT2 = COUNT2 - 1 ; $ MOVE ON TO NEXT OPERAND OF OP2
        END WHILE COUNT2 ;

    RESULT( USE ) = RESULT( USE ) - 1 ; $ COUNT OFF SUCCEEDING OP2
    END WHILE RESULT ;
```

EXAMPLE 5- THE HIGHEST CONTROL LEVEL OF A CODE GENERATOR

```
SUBR TOPMOST ;

SIZE ANOTHER(1),    $ FUNCTION TO READ IN ANOTHER ROUTINE
     ROUTINE(1);    $ DUMMY VARIABLE

WHILE  ANOTHER( ROUTINE ) ;

        CALL  PASS1 ;

        INIT#CODE ;      $ PREPARE FOR NEXT PASS

        CALL  PASS2 ;

        INIT#CODE ;      $ PREPARE AGAIN

        CALL  PASS3 ;

    END WHILE ANOTHER ;

CALL EXIT#ROUTINE ;

END SUBR TOPMOST ;
```