LITTLE Newsletter 36A

M. Shimasaki

November 10, 1974

Illustrative Examples of MIDL

In this newsletter supplement several MIDL example programs are given. There are two main motivations behind publication of example programs in advance of implementation of its compiler.

1). To supply test examples for a compiler and aid in debugging the compiler itself, especially to test the facilities not existing in LITTLE but newly introduced into MIDL.

2). To explore the use of the language, and show the ease of writing and/or reading algorithms in the newly designed language for problems of an appropriate level of complication; i.e., to test whether the language's design objectives are attained.

For purpose 1) rather simple, test programs testing separate language facilities are preferable at the initial stage of implementation of the compiler. Test programs for which compiler actions are easily traced will be helpful for debugging the compiler itself. However, for purpose 2), more substantial programs are required.

At this stage we select some simple, well-known - therefore presumably bug-free - programs which serve both for objectives 1) and 2). Special attention is paid to MIDL's recursive call, structure, pointer, and maptable features. As an example illustrating recursive program, Ackermann's (recursive but not primitive recursive) function is given. Except for very small values of m and n, computation of this function invokes many nested recursive calls. As another simple recursive example of recursive subroutines, the "Tower of Hanoi" puzzle is given. A comparatively simple example showing pointer treatment and the use of structures is also given.

We conclude with several more substantial examples: maximum flow in a graph, nodal span parsing, and a macroprocessor. These examples are obtained by transcribing SETL algorithms into MIDL. Explanations and/or references are included in each program.

```
$ EXAMPLE OF RECURSIVE FUNCTION        (2)
$ ACKERMANN FUNCTION
 $ RECURSIVE BUT NOT PRIMITIBE RECURSIVE FUNCTION

        FNCT ACKERMANN( M, N) RECURSIVE;

 $ VARIABLE DECLARATION
        DCL M    BITS(PS),
            N    BITS(PS);
        DCL ACKERMANN BITS(PS);

        IF M = 0  THEN  ACKERMANN = N+1;
                  ELSE  IF  N = 0  THEN ACKERMANN = ACKERMANN(M-1, 1);
                                ELSE ACKERMANN =
                                    ACKRMANN(M-1, ACKERMANN(M, N-1));
                               END IF;
            END IF;
        RETURN
        END FNCT;




$ THE  TOWER OF HANOI
 $ THIS IS A POPULAR ANCIENT PUZZLE, IT CONSISTS OF A HORIZONTAL BOARD
 $ WITH THREE VERTICAL PEGS AND N DISCS OF DIFERENT DIAMETERS. FIRST
 $ DISCS ARE ARRANGED ON ONE OF THE PEGS IN DIAMETER INCREASING ORDER,
 $ NAMELY THE LARGEST DISC IS AT THE BOTTOM AND THE SMALLEST IS AT THE
$ TOP  OF THE PEG. THE OBJECT OF THE PUZZLE IS TO TRANSFER ALL OF THE
$ DISCS FORM THE FIRST PEG TO ONE OF THE OTHERS AND THE FINAL ARRANGEMENT
$ SHOULD BE IDENTICAL TO THE STARTING ONE. ONLY ONE DISC CAN BE TRANSFERED
$ EACH TIME,  FURTHERMORE A DISC CAN BE PUT ONLY ON LARGER ONE THAN ITSELF.
$ WE TENTATIVELY USE INPUT SUBROUTINE INPUTI(N) FOR READING INTEGER
$ AND OUTPUT SUBROUTINE OUTPUTI(I, N, S, Z) IN A SUITABLE FORMAT.
$ PEGS ARE IDENTIFIED AS 1, 2, 3. N DISCS ARE IDENTFIED FROM 1 TO N
$ WITH INCREASING ORDER IN DIAMETERS.
 $ TRANSFER OF  N DISCS FROM THE PEG S TO THE PEG Z IS CARRIED OUT IN
 $ THE FOLLOWING STEPS.
 $      1) MOVE (N-1) DISCS FROM THE PEG S TO THE AUXILIARY PEG 6-S-Z.
 $      2) MOVE THE  N-TH  DISC FROM THE PEG S TO THE PEG Z.
 $      3) MOVE (N-1) DISCS FROM THE AUXILIARY PEG 6-S-Z TO THE PEG Z.
 $ IF N = 0 THEN THERE IS NOTHING TO DO,
 $ NOTE THAT  1 + 2 + 3 = 6  THEREFORE THE LAST PEG OTHER THAN S,Z
 $ CAN BE DENOTED BY  6 - S- Z,
 $ MORE DETAILED ACCOUNT IS GIVEN IN T. KIYONO: FUNDAMENTALS IN PROGRAMMING
 $ ( IN JAPANESE) PP. 173-179,


        SUBR START;
        SIZE NN(PS),              $ NUMBER OF DISCS
             I(PS),               $ I-TH STEP  GLOBAL VARIABLE
              P(PS),
              Q(PS);
        P=1;
        Q=2;
        CALL INPUTI(NN);          $ INPUT NUMBER OF DISCS
        CALL HANOI(NN, P, Q);
        RETURN;
```

```
      END SUBR;


      SUBR HANOI(N, S, Z) RECURSIVE;
 $ THIS SUBROUTINE SPECIFIES TRANSFER OF N DISCS FROM THE PEG S TO THE
 $ PEG Z.

      SIZE S(PS),               $ STARTING PEG
           Z(PS);               $ DESTINATION PEG
           N(PS);               $ NUMBER OF DISCS
      WHILE (N ¬=0) ;
      CALL HANOI(N-1, S, 6-S-Z);
           I = I + 1;
           CALL OUTPUTI( I, N, S, Z);
 $ AT THE I-TH STEP,  N-TH DISC IS TRANSFERRED FROM THE PEG S TO THE PEG Z.
           CALL HANOI(N-1, 6-S-Z, Z);
           END WHILE;
      RETURN;
      END SUBR HANOI;




$ EXAMPLE OF USE OF POINTER
$ THIS EXAMPLE HAS BEEN ADAPTED FROM AN EXAMPLE IN ≠ RECORD HANDLING ≠
$ BY C.A.R. HOARE ( PROGRAMMING LANGUAGES, PP.291-347, A.P.) ON P, 300.

$ TYPE DECLARATION
      TYPE PERSON: DATA-OF-BIRTH       BITS(PS),
                   MALE                BITS(1),
                   FATHER              PTR(PERSON),
                   ELDER-SIBLING       PTR(PERSON),
                   YOUNGEST-OFFSPRING  PTR(PERSON);
      EXPECT YOUNGEST-PATERNAL-UNCLE PTR(PERSON);

$ FUNCTION DEFINITION

      FNCT YOUNGEST-PATERNAL-UNCLE(R);

$ THIS FUNCTION YIELDS AS ITS RESULT A POINTER TO THE YOUNGEST PATERNAL
$ UNCLE OF THE PERSON REFERREDTO AS R, IF HE HAS ONE: OTHERWISE IT
$ YIELDS .OM..  THE FUNCTION MAY BE USED ONLY IF THE GRANDFATHER OF R IS
$ KNOWN TO EXIST;
      DCL YOUNGEST-PATERNAL-UNCLE PTR(PERSON);
      DCL R PTR(PERSON);
      DCL S              PTR(PERSON),
          F              PTR(PERSON),
          GRAND-FATHER   PTR(PERSON);
      F = FATHER R;
      GRAND-FATHER = FATHER F;
      S = YOUNGEST-OFFSPRING GRAND-FATHER;
 /REPEAT/
       YOUNGEST-PATERNAL-UNCLE = .OM.;
       WHILE (S ¬= .OM. );
           IF S = F  ∨ ( MALE S = 0)  THEN
              S = ELDER-SIBLING S;
              END IF;
           YOUNGEST-PATERNAL-UNCLE = S;
```

```
        END WHILE;
       RETURN;
      END FNCT;



$ THIS PROGRAM SOLVES THE ≠≠MAXIMUM FLOW PROBLEM≠≠.  IT HAS BEEN ADAPTED
$ FROM A SETL  PROGRAM EXAMPLE IN ≠ON PROGRAMMING II ≠ P.P.123-124.
$ THIS EXAMPLE SHOWS HOW MAPTABLE IS USED IN MIDL. DETAILED INFORMATION
$ OF THIS PROBLEM IS GIVEN IN THE ABOVE REFERENCE.

$ MACRO DEFINITION
      ** E1 = .F. 1, PS**      $ SUBFIELD DEFINITION THE FIRST NODE
      ** E2 = .F. PS+1, PS**   $ SUBFIELD DEFINITION THE SECOND NODE

$ TYPE DEFINITION
      TYPE SHORTI: BITS(PS);
      TYPE EDGE : BITS(2*PS); $ CONCATENATION OF E1 AND E2
      TYPE G : PTR(*EDGE);    $ TYPE OF GRAPH
      TYPE LISTNODE: VALUE    SHORTI,
                     NEXTNODE PTR(LISTNODE);
      TYPE EDGEFN MAP( 2*PS, SHORTI);       $ TYPE OF MAP TABLE WHOSE
                                            $ ARGUMENT IS EDGE.

 $ WE NOW GIVE GLOBAL VARIABLE DECLARATION.
      EXPECT CAP BITS(PS),
             R   EDGE;

      NAMESET MAXFLOWPR
      DCL MAXNODES SHORTI,                $ NUMBER OF NODES AT THE MOST
          PLENGTH  SHORTI,                $ LENGTH OF PATH P
          GRV      MAP(PS, LISTNODE);     $ GR≤V≥ REPRESENTED AS A MAP TABLE
      DCL P EDGE;   DIMS P(MAXNODES);     $ PATH IN THE FORM OF SEQUENCE OF
                                          $ EDGES
      END NAMESET;



      FNCT R(E);                $ REVERSED EDGE

      DCL R EDGE;
      DCL E EDGE;

      R = E2 E .C. E1 E;
      RETURN;
      END FNCT R;



      SUBR MAXFLOW(X, Y, GRAPH, C);       $ MAIN ROUTINE

      ACCESS MAXFLOWPR;

      DCL X       SHORTI,                 $ STARTING NODE
          Y       SHORTI,                 $ TERMINAL NODE
          GRAPH G,                        $ INPUT GRAPH
          C       EDGEFN;                 $ CAPACITY FUNCTION
```

```
$ END OF PARAMETERS

     DCL GR    G,                              $ GRAPH TYPE  TEMPORARY
         F        EDGEFN;                      $ FLOW VALUE
     DCL  NOEDGE     SHORTI,                   $ NUMBER OF EDGES IN INPUT GRAPH
          NOEDGEGR   SHORTI,                   $ NUMBER OF EDGES IN GR
          I          SHORTI,                   $ DO LOOP COUNTER
          J          SHORTI,                   $ DO LOOP COUNTER
          E          EDGE,
          FLAG       BITS(1);
     DCL LIST        LISTNODE;                 $ TEMPORATY LIST NODE
     DCL U           SHORTI;                   $ NODE
     DCL V           SHORTI;                   $ NODE
     DCL AUXFLOWV SHORTI,                      $ AUXILIARY FLOW VALUE
         TEMP      SHORTI,                     $ TEMPORARY TO COMPUTE MINIMUM
         REDUND    SHORTI,                     $ REDUNDANT FLOW VALUE
         REVES     EDGE;                       $ REVERSED EDGE OF E


$ COPY OF INPUT GRAPH
     NOEDGE = .NELT. GRAPH;
     GR = NEW( G, 2*NOEDGE);
     NOEDGEGR = NOEDGE;
     DO I = 1 TO NOEDGE ;
         GR(I) = GRAPH(I);
         END DO;                               $ END OF COPY OF INPUT GRAPH
$ ADDITION OF REVERSED EDGES TO GR
     DO I = 1 TO NOEDGE;
         FLAG = 0;
         E = R( GRAPH(I) );
         DO J = 1 TO NOEDGE;
             IF ( E = GRAPH(J) ) FLAG=1; $ THE REVERSED EDGE ALREADY
                                          $ EXISTS IN GR.
             END DO J;
         IF ( FLAG = 1 )  CONT DO;
         NOEDGEGR = NOEDGEGR + 1;
         GR(NOEDGEGR) = E;
         END DO I;

$ FORMATION OF  GR≤V≥ AS A MAP TABLE AND ONE WAY LIST
     DO I = 1 TO NOEDGEGR;
         V = E1 GR(I);
         U = E2 GR(I);
         IF(.DEF.GR(V) = 0)
             THEN VALUE GR(V) = U;    $ THE FIRST ELEMENT
             ELSE
                 LIST = GRV(V);
                 WHILE (NEXTNODE LIST ¬= .OM.);
                     LIST = NEXTNODE LIST;
                     END WHILE;
$ END OF LIST HAS BEEN FOUND, CREATE ONE NEW NODE.
                 NEXTNODE LIST = NEW(LISTNODE);
                 LIST = NEXTNODE LIST;
                 VALUE LIST = U;
                 END IF;
             END DO;

$ INITIALIZATION OF F(E)     F(E) IS REPRESENTED AS A MAPTABLE.
     DO I = 1 TO NOEDGEGR;
         F(GR(I) ) = 0;
```

```
        END DO;
    CALL PATH(X, Y, F, C);                    (6)
    WHILE (PLENGTH ¬= 0 );
        AUXFLOWV = CAP(P(1), F, C);
        DO I = 2 TO PLENGTH;
            E = P(I);
            TEMP = CAP(E, F, C);
            IF(TEMP < AUXFLOWV ) AUXFLOWV = TEMP;
            END DO;
        DO I = 1 TO PLENGTH;
            E = P(I);
            F(E) = F(E) + AUXFLOWV;
            REDUND = F(R(E));
            TEMP = F(E);
            IF( REDUND > TEMP ) REDUND = TEMP;
            F(E) = F(E) - REDUND;
            REVES = R(E);
            F(REVES) = F(REVES) - REDUND;
            END DO;
        END WHILE;
    RETURN;
    END SUBR MAXFLOW;



    FNCT CAP(E, F, C);

    DCL CAP SHORTI,
        E   EDGE,
        F   EDGEFN,
        C   EDGEFN;
    DCL TEMP1 SHORTI,                     $ C(E) OR 0
        TEMP2 SHORTI;                     $ F( R(E) )
    TEMP1 = 0;
    IF( .DEF.C(E) = 1 ) TEMP1 = C(E);
    TEMP1 = TEMP1 - F(E);
    TEMP2 = F( R(E) );
    IF( TEMP2 > TEMP1 ) TEMP1 = TEMP2;
    CAP = TEMP1;                          $ MAX   (TEMP1, TEMP2)
    RETURN;
    END FNCT CAP;



    SUBR PATH(X, Y, F, C);

    DCL X         SHORTI,            $ A GRAPH NODE
        Y         SHORTI,            $ A GRAPH NODE
        F         EDGEFN,            $ FLOW VALUE
        C         EDGEFN;            $ CAPACITY FUNCTION
$ END OF PARAMETER

    ACCESS MAXFLOWPR;

$ DECLARATION OF LOCAL VARIABLES
    DCL NONEW     SHORTI,            $ COUNTER FOR ELEMENTS OF NEWS
        NOPRIOR   SHORTI,            $ COUNTER FOR ELEMENTS OF PRIOR
        NOSET     SHORTI,            $ COUNTER FOR ELEMENTS OF SET
        NONEWER   SHORTI,            $ COUNTER FOR ELEMENTS OF NEWER
        I         SHORTI,            $ DO LOOP COUTER
```

```
          J          SHORTI,             $ DO LOCP COUTER
          U          SHORTI,             $ A GRAFH NODE
          V          SHORTI,             $ A GRAFH NODE
          PT         SHORTI,             $ A GRAFH NODE
          TEMP       SHORTI,             $ TEMPORARY FOR A GRAPH NODE
          LIST       LISTNODE,           $ TEMPORARY FOR LISTNODE
          FLAG       BITS(1);            $ A FLAC TO SHOW IF U IS IN U.
     DCL NEWS SHORTI; DIMS NEWS (MAXNODES);    $ NEW IN SETL PROGRAM
$ TO AVOID NAME CONFLICT WITH RESERVED WORD NEW FUNCTION, S IS ADDED.
     DCL SET   SHORTI; DIMS SET  (MAXNODES);
     DCL NEWER SHORTI; DIMS NEWER(MAXNODES);
     DCL PRIOR SHORTI; DIMS PRIOR(MAXNODES);
     DCL NEXT MAP(PS, SHCRTI);    $ NEXT NODE ON THE PATH
     NONEW = 1;
     NEWS(NONEW) = Y;
     NOSET = NONEW;
     SET( NOSET) = Y;
     .DROP, NEXT;          $ NEXT WILL POINT ALONG THE NODES OF A PATH
     WHILE ( NONEW ¬= 0 );
          NONEWER = 0;
          DO J = 1 TO NONEW;
               V = NEWS(J);
               NOPRIOR = 0;
               IF( .DEF.GRV(V) = 0 ) CONT DO;
               LIST = GRV(V);
               WHILE 1;
                    U = VALUE LIST;
                    FLAG = 0;
$ TO CHECK WHETHER U IS CONTAINED IN SET.
$ FLAG = 1 MEANS THAT U IS CONTAINED IN SET.
                    DO I = 1 TO NOSET;
                         IF( U = SET(I) ) THEN FLAG = 1;
                                             QUIT DO;
                                        END IF;
                         END DO;

                    IF FLAG = 0 THEN E = U .C. V ;
                                   IF CAP(E, F, C) > 0 THEN
                                             NOPRIOR = NOPRIOR + 1;
                                             PRIOR(NOPRIOR) = U;
                                             END IF;
                                   END IF;
                    IF(NEXTNODE LIST = .OM.) QUIT WHILE;
                    LIST = NEXTNODE LIST;
                    END WHILE;
               DO I = 1 TO NOPRIOR;
                    U = PRIOR(I);
                    NEXT(U) = V;
                    IF(U = X ) GO TO DONE;
                    NOSET = NOSET + 1;
                    SET(NOSET) = U;
                    NONEWER = NONEWER +1;
                    NEWER(NONEWER) = U;
                    END DO I;
               END DO J;
          NONEW = NONEWER;
          DO I = 1 TO NONEWER;
               NEWS(I) = NEWER(I);
               END DO;
          END WHILE;
```

```
/DONE/
     PLENGTH = 0;
     PT = X;                 $ NOW LOOP TO BUILD UP PATH
     WHILE (.DEF.NEXT(PT) = 1 );
         PLENGTH = PLENGTH + 1;
         TEMP = NEXT(PT);
         P(PLENGTH) = PT,C,TEMP;
         PT = TEMP;
         END WHILE;
     RETURN;
     END SUBR PATH;
```

```
$     THE OUTPUT OF THE ROUTINE -NODPARS- IS A VECTOR OF LISTS OF SPANS
$     -SPANS- AND A FLAG *AMB*, WHICH INDICATES WHETHER THE GRAMMAR IS
$     AMBIGUOUS,
$     SPANS(Q) IS A POINTER TO A LIST OF ALL SPANS OF THE FORM (PAQ).
$     EACH SPAN ELEMENT IN THE LIST CONSISTS OF A SPAN ITEM -PA-
$     A POINTER TO ITS DIVLIS, AND A ONE BIT FLAG -AMBIT-, INDICATING
$     WHETHER THERE ARE MORE THAN ONE DIVISION. A -DIVLIS- ELEMENT
$     CONTAINS 2 POINTERS TO THE TWO PREVIOUS SPANS WHICH FORM THE
$     CURRENT SPAN.

$     WHILE CONSTRUCTING  THE LIST OF SPANS FOR EACH -Q-, EACH NEW SPAN IS
$     ADDED TO THE LIST *CURSPANS*,AND ALSO TO THE HASHTABLE
$     -CURSPANSET-. THE VALUE OF CURSPANSET(PA) IS  A POINTER TO THE SPAN
$     IN THE LIST, SO THAT THE DIVLIS FOR THE SPAN MAY BE LOCATED IF THE
$     SPAN HAS MORE THAN ONE ORIGIN,

$     TO CLEAN UP, THE TOPSPAN IS OBTAINED, AND THE -SPANS- VECTOR
$     CLEARED. BY TRACING THE *DIVLIS* POINTERS, -SPANS- IS REBUILT TO
$     CONTAIN LISTS OF ONLY THOSE SPANS WHICH ARE RELEVANT. TO FIND
$     THE DESCENDENTS OF A SPAN <PAQ>, WE KNOW THAT THE
$     DESCENDENT <RCQ> SHOULD BE ADDED TO THE LIST OF SPANS(Q) AND FROM
$     THIS SPAN, CAN DETERMINE THAT <PBR> SHOULD BE ADDED TO THE LIST
$     SPANS(R).

$     INPUT IS  AN ARRAY OF TOKENS. SYNTYPS IS A VECTOR OF LISTS OF
$     METAVARIABLES FOR EACH TOKEN.

$     GRAM IS  A HASHTABLE SUCH THAT GRAM(BC) IS A LIST OF METAVARIABLES
$     A FOR WHICH THERE ARE PRODUCTIONS A → BC IN THE GRAMMAR.

$ MACROS
      ** RULSZ = 6 **     $ MAX SIZE OF THE INTERNAL REPRESENTATION OF A
                          $ METAVARIABLE.
      ** TOKSZ = 6 **   $ MAX SIZE OF THE POINTER TO INPUT *N*.
      ** INPUTLEN = 63 **       $ DIMENSION OF INPUT

      ** TYP  = .F. 1, RULSZ, **    $ METAVARIABLE FIELD OF A SPAN
      ** MID = .F. RULSZ+1, INPUTSZ, **    $ *P- FIELD OF A SPAN


      ** INITSPLIST(SPANHD, SPANTYPE, SPANITEM) =
          DCL ZZZA PTR(SPANTYPE);
          ZZZA = NEW(SPANTYPE);
          FIRST SPANHD = ZZZA;
          LAST SPANHD = ZZZA;
          SPAN ZZZA = SPANITEM **

      ** ADDSPLIST(SPANHD, SPANTYPE, SPANITEM) =
          DCL ZZZB PTR(SPANTYPE);
          ZZZB = NEW(SPANTYPE);
          NEXT ZZZB = FIRST SPANHD;
          SPAN ZZZB = SPANITEM;
          FIRST SPANHD = ZZZB **

$     TYPE DEFINITIONS

      TYPE RULEP: PTR(RULES);    $ ENTRY IN GRAMMAR HASHTABLE
      TYPE RULES: META PTR(RULSZ), NEXT PTR(RULES); $ ELEMENT OF LIST OF
```

```
                                      $ METAVARIABLES
    TYPE SYNTYP: PTR(RULES);  $ VECTOR CF POINTERS TO LISTS OF RULES
    TYPE SPANLIST: PTR(SPANEL);   $ VECTOR OF LIST OF SPANS
    TYPE SSPANEL: SPAN BITS(RULSZ+INPUTSZ), NEXT PTR(SSPANEL); $ SHORT
                                                        $ SPAN ELENENT
    TYPE SPANEL: SPAN BITS(RULSZ+INPUTSZ), DIVLIS PTR(DIVEL),
                 AMBIT BITS(1),
                 NEXT PTR(SPANEL);
    TYPE SPANHD: FIRST PTR(SPANEL), LAST PTR(SPANEL);
    TYPE SSPANHD: FIRST PTR(SSPANEL), LAST PTR(SSPANEL); $ SHORT SPAN
                                                    $ HD
    TYPE DIVEL: LODIV PTR(SPANEL), HIDIV PTR(SPANEL), NEXT PTR(DIVEL);
                            $ DIVISION LIST ELEMENT

$   GLOBAL VARIABLES
    NAMESET GRAMMAR;
    DCL GRAM MAP(2*RULSZ,RULEP);  $ MAPTABLE STORING RULES OF FORM
                                  $ A → BC
    DCL SYNTYPS PTR(SYNTYP);  $ VECTOR CF LISTS OF RULES
    END NAMESET GRAMMAR;

    NAMESET SOURCE;    $ INPUT STRING TO BE PARSED
    SIZE INPUT(TOKSZ);
    DIMS INPUT(INPUTLEN);
    END NAMESET SOURCE;

    NAMESET PARSEOUT;
    SIZE AMB(1)  ;          $ FLAG FOR AMBIGUOUS GRAMMAR
    DCL SPANS PTR(SPANLIST);   $ VECTOR OF LISTS OF SPANS
    END NAMESET PARSOUT;


    SUBR NODPARSE;
    ACCESS GRAMMAR, SOURCE, PARSEOUT;
    SIZE N(INPUTSZ);    $ PTR TO CURRENT INPUT TOKEN
    DCL TWIGRULE PTR(RULES);    $ PTR TO LIST OF SYNTYPES
    DCL CURSPANS PTR(SPANHD);    $ LIST HEAD OF CURRENT SLAMS
    DCL TODO PTR(SSPANHD);    $ LIST HEAD OF SPANS TO PROCESS
    DCL NEXTODO PTR(SSPANEL);    $ ELEMENT OF TODO
    DCL PREVSPANPT PTR(SPANEL);  $ PTR TO PREVIOUS SPAN GENERATED
    DCL RULEPT PTR(RULES);  $ PTR TO LIST OF METAVARIABLES
    SIZE NEWSPAN(INPUTSZ+RULSZ);    $ NEW SPAN FORMED
    DCL NEWDIVEL PTR(DIVEL);    $ NEW DIVISION LIST ELEMENT
    DCL CURSPANSET MAP(RULSZ+INPUTSZ, SPANLIST);  $ MAPTABLE OF
                                                    $ CURRENT SPANS
    DCL CURSEL PTR(SPANEL);  $ ELEMENT IN RANGE OF CURSPANSET
    DCL SPANTHERE PTR(SPANEL);    $ PTR RETRIENED FROM CURSPANSET
    DCL TOPSPAN PTR(SPANEL);  $ PTR TO SPAN FROM ROOT ELEMENT
    SIZE I(PS);     $ DO LOOP VARIABLE

$ INITIALIZE SPANS(1)
    N = 1;
    TWIGRULE = SYNTYPS(INPUT(1));  $ PTR TO LIST OF METAVARS
    INITSPLIST(CURSPANS, SPANEL, N ,C, META TWIGRULE);
    TWIGRULE = NEXT TWIGRULE;
    WHILE TWIGRULE ¬= .OM.;
        ADDSPLIST(CURSPANS, SPANEL, N .C. META TWIGRULE);
        TWIGRULE = NEXT TWIGRULE;
        END WHILE TWIGRULE;
    SPANS(2) = FIRST CURSPANS;  N = 2;
```

```
    WHILE INPUT(N) ¬= 0;    $ BUILD UP REST OF SPANS
        TWIGRULE = SYNTYPS(INPUT(N));
$   INITIALIZE CURSPANS AND TODO
        INITSPLIST(CURSPANS, SPANEL, N .C. META TWIGRULE);
        INITSPLIST(TODO, SSPANEL, N .C. META TWIGRULE);
        TWIGRULE = NEXT TWIGRULE;
        WHILE TWIGRULE ¬= .OM.;
             ADDSPLIST(CURSPANS, SPANEL, N .C. META TWIGRULE);
             ADDSPLIST(TODO, SSPANEL, N .C. META TWIGRULE);
             END WHILE TWIGRULE;

$ START PROCESSING SPANS IN TODO

        .D. CURSPANSET = 0;  $ INITIAL CURSPANSET
        NEXTODO = FIRST TODO;
        WHILE NEXTODO ¬= .OM.;
             PREVSPANPT = SPANS(MID SPAN NEXTODO);
             WHILE PREVSPANPT ¬= .OM.;
              $ LOOK UP GRAMMAR RULES
                  RULEPT = GRAM(TYP  SPAN PREVSPAN .C.
                         TYP  SPAN NEXTODO);
                  WHILE RULEPT ¬= .OM.;
                     NEWSPAN = MID SPAN PREVSPANPT .C. META RULEPT;
                     NEWDIVEL = NEW(DIVEL);   $ CREATE NEW DIVISION
                               $ LIST ELEMENT
                     HIDIV NEWDIVEL = NEXTODO;
                     LODIV NEWDIVEL = PREVSPANPT;
                     SPANTHERE = CURSPANSET(NEWSPAN); $ SEE IF SPAN
                                          $ ALREADY GENERATED
                     IF SPANTHERE = .OM.  THEN
           $ SPAN NOT YET GENERATED, ADD TO CURSPANS LIST
                       ADDSPLIST(CURSPANS, SPANEL, NEWSPAN);
                       CURSEL = NEW(SPANEL);
$   CREATE ELEMENT OF CURSPANSET;
                       ↑ CURSEL = FIRST CURSPANS;
                       CURSPANSET(NEWSPAN) = CURSEL;
                       $ ADD TO TODO LIST
                       ADDSPLIST(TODO, SSPANEL, NEWSPAN);
                       DIVLIS FIRST CURSPANS= NEWDIVEL;
$   SINCE FIRST OCCURANCE OF SPAN, FIRST DIVELEMENT FOR SPAN
                     ELSE $ SPAN ALREADY GENERATED, ADD DIVLIS EL
                       NEXT NEWDIVEL = DIVLIS SPANTHERE;
                       DIVLIS SPANTHERE = NEWDIVEL;
                       AMBIT SPANTHERE = 1;
                     END IF;
                     RULEPT = NEXT RULEPT;
                     END WHILE RULEPT;
                  PREVSPANPT = NEXT PREVSPANPT;
                  END WHILE PREVSPANPT;
             NEXTODO = NEXT NEXTODO;
             END WHILE NEXTODO;
$   CURSPANS FINISHED, ADD LIST TO VECTOR SPANS
        N = N + 1;  SPANS(N) = FIRST CURSPANS;
        END WHILE INPUT(N);
    $ NOW SEE IF THERE IS A PARSE
    AMB = 0;   $ ASSUME NOT AMBIGUOUS
    TOPSPAN = CURSPANSET(1 .C. ROOT);
$ CLEAR SPANS VECTOR
    DO I = 1 TO N;
```

```
         SPANS(I) = .OM.;
         END DO;
     IF TOPSPAN = .OM.  RETURN;  $ NO PARSES FOUND
     CALL GETDESCS(TOPSPAN, N);
     RETURN;
     END;



     SUBR GETDESCS(TOP, N) RECURSIVE;

     DCL SPANPT PTR(SPANEL);    $ PTR TO LIST OF VALID SPANS
     DCL TOP PTR(SPANEL);    $ PTR TO ROOT SPAN
     DCL DIVPT PTR(DIVEL);    $ PTR TO DIVLIS ELEMENT
     SIZE N(INPUTSZ);     $ END POINT OF SPAN WHICH ISTHE ROOT
     ACCESS PARSEOUT;

     IF SPANS(N) = .OM. THEN
         SPANS(N) = TOP;  NEXT TOP = .OM.;
     ELSE
         SPANPT = SPANS(N);    $ SEARCH FOR SPAN
         WHILE SPANPT ¬= .OM.;
              IF SPAN SPANPT = SPAN TOP RETURN;  $ SPAN ALREADY THERE
              SPANPT = NEXT SPANPT;
              END WHILE SPANPT;
 $   ADD SPAN TO LIST
              NEXT TOP = SPANS(N);   SPANS(N) = TOP;
              END IF;
     IF AMBIT TOP THEN AMB = 1;   $ MORE THAN ONE PARSE

 $   GET DESCENDENTS
     DIVPT = DIVLIS TOP;
     WHILE DIVPT ¬= .OM.;
         CALL GETDESCS(HIDIV DIVPT, N);
         CALL GETDESCS(LODIV DIVPT, MID SPAN HIDIV DIVPT);
         END WHILE DIVPT;
     RETURN;
     END GETDESCS;




$ THIS EXAMPLE ILLUSTRATES HOW SETL OBJECTS AND PRIMITIVES MAY BE
$   USED IN MIDL.   THE CODE CORRESPONDS VERY CLOSELY TO THE ALGORITHM
$ IN ON PROGRAMMING, VOL 2, PP 189-195. AS IN THE ORIGINAL, WE ASSUME
$   THERE IS A FUNCTION -GETOKEN-, WHICH RETURNS A TOKEN AND ITS TYPE,
$   STORED ACCORDING TO THE FOLLOWING TYPE DECLARATION;
$
     TYPE TOKENTYP: TOKT BITS(TYPSZ), TOK SETLOBJ;
$
$   THIS MEANS THAT -GETOKEN- RETURNS A POINTER TO A STRUCTURE IN THE
$   HEAP WHICH CONSISTS OF A TOKEN TYPE FIELD -TOKT- (TYPSZ IS ASSUMED
$   TO BE A MACRO WHICH EXPANDS TO A CONSTANT) AND A SETL OBJECT -TOK-
$   (SRTL ROOT WORD).  THE SETL OBJECT HERE WILL BE AN ARBITRARY
$   LENGTH CHARACTER STRING.
$
$   THE MACRO DICTIONARY -MACDICT-, ALSO  A SETL OBJECT, IS A SET USED
$   AS A FUNCTION.  AS WE ALLOW MIDL    POINTERS TO BE STORED  IN SETS,
```

```
$   MACDICT(TOKEN) STORES A  MIDL   POINTER TO A STRUCTURE DECLARED AS:
$
      TYPE MACDICTENT: GENARGS BITS(ARGSZ),
                       FORMARGS SETLOBJ,
                       MBCD PTR(MBODS);
$
$   GENARGS IS A BITSTRING OF SIZE ARGSZ   (A MACRO WHICH EXPANDS TO
$   A CONSTANT), AND CONTAINS THE NUMBER OF GENERATED MACRO ARGUMENTS.
$   FORMARGS IS A SET WHICH CONTAINS THE MAPPING OF MACRO ARGUMENTS ONTO
$   ARGUMENT NUMBERS.  MBOD IS A POINTER TO THE MACRO BODY, WHICH IS
$   STORED AS AN ARRAY OF POINTERS TO TOKENS, AS SPECIFIED BY THE
$   DEFINITION:
$
      TYPE MBODS: PTR(TOKENTYP);
$
$   RESERVE IS A LINKED LIST OF POINTERS TO TOKENS:
$
      TYPE TOKLIST: ITEM PTR(TOKENTYP), NEXT PTR(TOKLIST);
      TYPE LISTHD: FIRST PTR(TOKLIST), LAST PTR(TOKLIST);
$
$   THE EXPSTACK IS A PUSH DOWN STACK, EACH ENTRY IS A POINTER TO AN
$   EXPSTACK STRUCTURE:
$
      TYPE EXPSTACKP: PTR(EXPSTACKENT);
      TYPE EXPSTACKENT: MBP BITS(PS),
                        MBOD PTR(*MBODS),
                        FORMARGS SETLOBJ,
                        ACTARGS PTR(*ARGTUP);
      TYPE ARGTUP: PTR( MBODS);
$
$   MBOD IS A POINTER TO THE MACRO BODY BEING EXPANDED. MBP IS AN INDEX
$   TO MBOD, INDICATING WHICH TOKEN EXPANSION IS UP TO. FORMARGS IS
$   AS IN MACDICTENT, AND ACTARGS IS AN ARRAY OF MBODS CORRESPONDING
$   TO THE ACTUAL PARAMETERS OF THE MACRO INVOCATION.
$
$   WE NOW GIVE THE GLOBAL VARIABLE DECLARATIONS.
$
      EXPECT NEXTWORD PTR(TOKENTYP);
      EXPECT DEFABSORB PTR(TOKENTYP);
      EXPECT MACEXPAND PTR(TOKENTYP);
      NAMESET LEXMACEXP;
      SIZE EXPSTACKPTR(PS); DATA EXPSTACKPTR=0;
      DCL RESERVE LISTHD,
          MACDICT SETLOBJ,                $ MACRO DICTIONARY
          MACEXPGIVEBACK PTR(TOKENTYP); $ TOKEN RETURNED TO MAXEXPAND
      DCL EXPSTACK EXPSTACKP;
      DIMS EXPSTACK(EXPSTACKDIM);      $ DIMENSION OF EXPSTACK
      END NAMESET;

 $    THE FOLLOWING MACROS ARE USED TO COMPARE A SETL CHARACTER STRING
 $    -W- TO A  MIDL   SELF DEFINED STRING, WHICH IS CONVERTED
 $    TO A SETL STRING BY THE .CN. OPERATOR.

      ** EQSTR(W,S) = (W = .CN. SETLSTR, S) **
      ** NEQSTR(W,S) = (W -= .CN. SETLSTR, S) **


      FNCT NEXTWORD(DUM);
      DCL NEXTWORD  PTR(TOKENTYP);
      DCL WORD      PTR(TOKENTYP),           $ TOKEN
```

```
        MACINF     PTR(MACDICTENT),          $ MACDICT ENTRY OF TOKEN
        ARGFN      SETLOBJ,                  $ ARG - ARGNO MAP
        MBODY      PTR(*MBODS);              $ ARRAY OF PTRS TO STRINGS,
        MCALLINF   PTR(EXPSTACKENT),
        ARGTUPL    PTR(*ARGTUP),
        NEWTOK     PTR(TOKENTYP),            $ NEW TOKEN GENERATED ARG
        CURARGTUPL PTR(*MBODS),              $ CURRENT ACTUAL PARAMETER
        NNEWTOK    PTR(MBODS);               $ PTR TO NEWTOK
    SIZE DUM(PS);
    SIZE FIRSTCALL(1);  DATA FIRSTCALL = 1;
    SIZE NXARGS(ARGSZ),      $ NUMBER OF GENERATED ARGUMENTS
         NARGS (ARGSZ),      $ NUMBER OF NONGENERATED ARGS
         ARGTUPCT(PS),       $ PTR TO ARGTUP
         J(PS),              $ DO LOOP COUNTER
        PARENCOUNT(PS);      $ PARENTHESES COUNT
    ACCESS LEXMACEXP;

    IF FIRSTCALL THEN
        FIRST RESERVE=.OM.;
        LAST RESERVE=.OM.;
        XARGGENCTR = 0;  MACDICT = .NL.;
        FIRSTCALL = 0;
        END IF;
    IF FIRST RESERVE ¬= .OM.  THEN
        WORD = ITEM FIRST RESERVE;
        FIRST RESERVE = NEXT FIRST RESERVE;
        NEXTWORD = WORD;
        RETURN;
        END IF;
$   OTHERWISE, GET ADDITIONAL TOKEN FROM DEFABSORB
/GETWORD/
    WORD = DEFABSORB(0);
    MACINF = MACDICT(WORD);     $ SINCE MACDICT IS A SETL OBJECT,
$   COMPILES INTO CALL TO SRTL
    IF(MACINF = .OM.) THEN
        NEXTWORD = WORD;
        RETURN;
        END IF;
$ WORD IS A MACRO NAME
    NXARGS = GENARGS MACINF;      $ NO GENERATED ARGS
    ARGFN = FORMARGS MACINF;
    MBODY = MBOD     MACINF;
    NARGS = .NELT. ARGFN - NXARGS;
$   .NELT. IS A SYSTEM FUNCTION WHICH , IF ITS OPERAND IS A SETL
$   OBJECT, CALLS SRTL ROUTINE NELT,
$   IF  MIDL   MAPTABLE, COMPUTES NUMBER OF ENTRIES,
$   IF  MIDL   HEAP OBJECT, COMPUTES DIMENSION OF HEAP BLOCK,
$   OTHERWISE, CAN BE COMPUTED AT COMPILE TIME.
    MCALLINF = NEW(EXPSTACKENT); $ BUILD EXPSTACK ENTRY,
    MBP MCALLINF = 1;  MBOD MCALLINF = MBODY;
    FORMARGS MCALLINF = ARGFN;   ARGTUPL = NEWN(ARGTUP, MAXARGS);
    ARGTUPCT = 0;            $ PTR TO ARGTUPL
    IF NARGS > 0 GO TO GETARGS;
$   GENERATE ARGS
/GENARGS/
    DO J = 1 TO NXARGS;
    NEWTOK = NEW(TOKENTYP);
        TOKT NEWTOK = NAMETYPE;  $ SET TYPE OF NEW TOKEN
$        NAMETYPE IS A GLOBAL MACRO TO BE EXPANDED INTO INTEGER
        XARGGENCTR = XARGGENCTR + 1;
```

```
        TOK NEWTOK = .CN. SETLSTR, ≠ZZZZ≠ + .DEC. .CN. SETLINT,
                                 XARGGENCTR;
        ARGTUPCT = ARGTUPCT + 1;
         IF(ARGTUPCT > MAXARGS) THEN
$    OVERFLOW CHECKS COULD BE AVOIDED IF TUPLES WERE SETL TUPLES
$    RATHER THAN  MIDL   OBJECTS
        PRINTERROR(≠MAXIMUM NUMBER OF ARGS EXCEEDED.≠);
             END IF;
        NNEWTOK = NEW(MBODS);
        ↑NNEWTOK  = NEWTOK;          $ THE VALUE OF NNEWTOK IS SET TO
                                      $ POINT TO NEWTOK
        ARGTUPL(ARGTUPCT) = NNEWTOK;
$    TRIM IS A FUNCTION WHICH RETURNS A POINTER TO A HEAP OBJECT
$    WHICH IS REDUCED TO *ARGTUPCT* ENTRIES
        END DO;
    ACTARGS MCALLINF = TRIM(ARGTUPL, ARGTUPCT);
    EXPSTACKPTR = EXPSTACKPTR + 1;
    IF(EXPSTACKPTR > EXPSTACKDIM) THEN
        PRINTERROR(≠TOO MANY EMBEDDED MACROS.≠);
        END IF;
    EXPSTACK(EXPSTACKPTR) = MCALLINF;  $ STORE POINTER TO EXPSTACK
                                 $ ENTRY
    GO TO GETWORD;
/GETARGS/  $ MACRO HAS ARGUMENTS, COLLECT ARGUMENTS OUT OF TOKEN
$    STREAM
    IF NEQSTR(TOK DEFABSORB(0) , ≠(≠) THEN
        PRINTERROR(≠MISSING MACRO ARGUMENTS.≠);
        NEXTWORD = WORD;
        RETURN;
    END IF;
    DO J = 1 TO NARGS;
        PARENCOUNT = 0;  $ UNMATCHED PARENTHESIES COUNT
        CURARGTUPCT = 0;   $ PTR TO CURARGTUPL
        CURARGTUPL = NEWN(MBODS, MAXARGSZ);  $ ALLOCATE STORAGE
        WORD = DEFABSORB(0) ;    $ GET TOKEN
        WHILE(NEQSTR(TOK WORD, ≠,≠) ∨ (PARENCOUNT > 0));
             IF TOKT WORD = EOR THEN
                  PRINTERROR(≠IMPROPER END OF RECORD.≠);
                  NEXTWORD = WORD;
                  RETURN;
                  END IF;
             TOKEN = TOK WORD;
             IF EQSTR(TOKEN, ≠)≠) THEN
                  PARENCOUNT = PARENCOUNT - 1;
                  IF PARENCOUNT = - 1 GO TO ENDARGS;
             ELSE IF EQSTR(TOKEN, ≠(≠) PARENCOUNT = PARENCOUNT + 1;
             END IF;
             CURARGTUPCT = CURARGTUPCT + 1;  $ PTR TO CURARGTUPL
             IF(CURARGTUPCT > MAXARGSZ) THEN
                  PRINTERROR(≠MAX ARG SIZE EXCEEDED.≠);
             ELSE
                  CURARGTUPL(CURARGTUPCT) = WORD;
             END IF;
             END WHILE;
        ARGTUPCT = ARGTUPCT + 1;
        IF(ARGTUPCT > NXARGS) THEN
             PRINTERROR(≠MAXIMUM NUMBER OF ARGS EXCEEDED≠);
             NEXTWORD = WORD;
             RETURN;
             END IF;
```

```
        ARGTUP(ARGTUPCT) = TRIM(CURARGTLPL, CURARGTUPCT);
        END DO;
/ENDARGS/
     IF ARGTUPCT < NARGS THEN
         PRINTERROR(≠MISSING PARAMETERS IN MACRO CALL.≠);
         NXARGS = NXARGS + NARGS - ARGTUPCT;
     ELSE IF PARENCOUNT = -1 THEN
         PRINTERROR(≠SURPLUS PARAMETERS IGNORED IN MACRO CALL.≠);
         END IF;
     END IF;
     GO TO GENARGS;
     END NEXTWORD;

     FNCT DEFABSORB(DUM);
$    ABSORBS MACRO DEFINITIONS
     DCL DEFABSORB PTR(TOKENTYP);
     DCL WORD      PTR(TOKENTYP),
         XWORD     PTR(TOKENTYP),
         TOKEN     SETLOBJ,
         XTOKEN    SETLOBJ,
         MNAME     PTR(TOKENTYP),    $ MACRO NAME
         ARGFN     SETLOBJ,          $ ARGUMENT/ARGNO MAP
         MBODY     PTR(*MBODS),      $ MACRO BODY
     SIZE MBODYCT(PS);
     SIZE DUM(PS);          $ DUMMY ARGUMENT
     SIZE TYP (TYPSZ),
          ARGFNCT(PS),          $ ARGUMENT COUNT
          NARGS(PS);
     DCL NEWMENT PTR(MACDICTENT);
     ACCESS LEXMACEXP;

/SCAN/
     WORD = MACEXPAND(0);$ GET NEXT TOKEN
     TOKEN = TOK WORD;  $ EXTRACT STRING FROM WORD
     XWORD = MACEXPAND(0);
     XTOKEN = TOK XWORD;
     IF EQSTR(TOKEN, ≠ENDM≠) .AND. EQSTR(XTOKEN, ≠;≠) THEN
         PRINTERROR(≠IMPROPER MACRO CLOSE BEFORE OPENING.≠);
     ELSE IF EQSTR(TOKEN, ≠DEFINE≠) .AND. EQSTR(XTOKEN, ≠MACRO≠) THEN
$ GIVE BACK ONE WORD AND RETURN THE OTHER
         MACEXPGIVEBACK = XWORD;
         DEFABSORB = WORD;
         RETURN;
         END IF;
     END IF;
$ READ MACRO DEFINITION
     MNAME = MACEXPAND(0);    $ READ NAME
     TYP   = TOKT MNAME;
     IF TYP  -= NAMETYPE THEN
         PRINTERROR(≠NAME MISSING IN MACRO DEFINITION. DEFINITION ≠
                    .CC.≠IGNORED.≠);
         DEFABSORB = WORD;
         RETURN;
         END IF;
     ARGFN = .NL. ;      ARGFNCT = 0;
     WORD = MACEXPAND(0);
     TOKEN = TOK WORD;
     IF EQSTR(TOKEN, ≠(≠) GO TO GETARGUMS;
     IF EQSTR(TOKEN, ≠;≠) THEN
         IF MACDICT(TOKEN) -= .OM. THEN
```

```
                PRINTWARN(≠PRIOR MACRO DEFINITION IS BEING CHANGED,≠);
                END IF MACDICT;
        GO TO GETBODY;
        END IF;
        IF NEQSTR(TOKEN, ≠ENDM≠) THEN
                PRINTERROR(≠IMPROPER CONTINUATION OF MACRO DEFINITION)≠
                    .CC. ≠DEFINITION IGNORED.≠);
                DEFABSORB = WORD;
                RETURN;
            END IF;
$ HAVE SEEN DEFINE MACRO MACRONAME ENDM.
        WORD = MACEXPAND(0);
        IF NEQSTR(TOK WORD, ≠;≠) THEN
            PRINTERROR(≠IMPROPER TERMINATION OF MACRO DROP≠);
            MACEXPGIVEBACK = WORD;
            END IF;
        IF MACDICT(TOK MNAME) = ,OM, THEN
            PRINTWARN(≠DROP APPLIED TO NON-MACRO NAME≠);
        ELSE
            MACDICT(TOK MNAME) = ,OM,;
        END IF;
        GO TO SCAN;
/GETARGUMS/    $ SCAN FOR ARGUMENTS OF A MACRO
        WORD = MACEXPAND(0);
        IF EQSTR(TOK WORD, ≠/≠) GO TO GETXARGS;
        MACEXPGIVEBACK = WORD;
        GETARGS(ARGFN, ARGFNCT);      $ ARGFNCT IS NUMBER OF NON-GENERATED
        WORD = MACEXPAND(0);
        TOKEN = TOK WORD;
        IF EQSTR(TOKEN, ≠/≠) GO TO GETXARGS;
        NARGS = ARGFNCT;    $ NUMBER OF TRUE ARGS
/TESTALISTEND/
        IF NEQSTR(TOKEN, ≠)≠) THEN
            PRINTERROR(≠ILLEGAL TERMINATION OF MACRO ARGUMENT LIST.≠ ,CC.
                ≠DEFINITION IGNORED.≠);
            DEFABSORB = WORD;
            RETURN;
            END IF;
$ NOW CHECK FOR SEMI-COLON FOLLOWING ARGUMENT LIST
        WORD = MACEXPAND(0);
        IF NEQSTR(TOK WORD, ≠;≠) THEN
            PRINTERROR(≠ILLEGAL TERMINATION OF MACRO ARGUMENT LIST.≠);
            MACEXPGIVEBACK = WORD;
            END IF;
$ BEGIN TO COLLECT BODY OF MACRO
/GETBODY/
        MBODY = NEWN(MBODS,    MAXMACROSIZE);  $ ALLOCATES TUPLE FOR BODY
        MBODYCT = 0;  $ NUMBER OF WORDS IN MAC BODY
/LOOP/
        WORD = MACEXPAND(0);
        TOKEN = TOK WORD;
        XWORD = MACEXPAND(0);
        XTOKEN = TOK XWORD;
        IF EQSTR(TOKEN,≠ENDM≠) ,A, EQSTR(XTOKEN, ≠;≠) THEN     $ END OF DEF

            NEWMENT = NEW(MACDICTENT);     $ BUILD MACDICT ENTRY
            GETARGS NEWMENT = ARGFNCT - NARGS; $ NUMBER OF GENERATED ARGS
            FORMARGS NEWMENT = ARGFN;      $ MAPPING OF ARGS - ARGNO
            MBOD NEWMENT = TRIM(MBODY, MBODYCT);
            MACDICT(TOK MNAME) = ,ON, SETLPTR, NEWMENT; $ ADD DEF TO DIC-
```

```
                                                           $ TIONARY
          GO TO SCAN;
      ELSE
          IF EQSTR(TOKEN, ≠DEFINE≠) .A. EQSTR(XTOKEN, ≠MACRO≠) THEN
               PRINTERROR(≠IMPROPER MACRO DEFINITION WITHIN MACRO BODY≠)
          ELSE IF TOKT WORD = EOR THEN
               PRINTERROR(≠END OF FILE ENCOUNTERED IN MACRO DEFINITION≠)
               DEFABSORB = WORD;
               RETURN;
               END IF;
          END IF;
      END IF;
      MBODYCT = MBODYCT +1;
      $ CHECK FOR OVERFLOW
      IF(MBODYCT .GT. MAXMACROSIZE) THEN
          PRINTERROR(≠MACRO BODY TOO LONG.≠)
          DEFABSORB = WORD;
          RETURN;
          END IF;
      MBODY(MBODYCT) = WORD;
      MACEXPGIVEBACK = XWORD;
      GO TO LOOP;
/GETXARGS/
      NARGS = ARGFNCT;    $ NUMBER OF TRUE ARGS
      CALL GETARGS(ARGFN, ARGFNCT);
      WORD = MACEXPAND(0);
      TOKEN = TOK WORD;
      GO TO TESTALISTEND;
      END DEFABSORB;


      SUBR GETARGS(ARGFN, ARGFNCT);
      DCL WORD      PTR(TOKENTYP);         $ TOKEN
      SIZE ARGFNCT(PS);      $ ARGUMENT COUNT

/GETALOOP/
      WORD = MACEXPAND(0);
      IF TOKT WORD  ¬= NAMETYPE THEN
          PRINTERROR(≠MISSING ARGUMENT NAME IN MACRO ARGUMENT LIST≠);
          RETURN;
      ELSE IF ARGFN(TOK WORD) ¬= ,OM. THEN    $ .OM. IS SETL OMEGA
                                              $ EVALUATION OF TOK WORD
                                              $ GIVES ROOT WORD FOR SETL
                                              $ CHARACTER STRING,
               PRINTERROR(≠DUPLICATE ARGUMENT NAME IN ARGUMENT LIST, ≠
                              ,CC, ≠DUPLICATE IGNORED.≠);
          ELSE ARGFNCT = ARGFNCT + 1;    $ INCREMENT ARG COUNT
               ARGFN(TOK WORD) = ,CN. SETLINT ARGFNCT; $ CONVERT MIDL
                                                       $ INTEGER TO
                                                       $ SETL INTEGER,
               END IF;
          END IF;
      WORD = MACEXPAND(0);
      IF EQSTR(TOK WORD, ≠,≠) GO TO GETALOOP;
      MACEXPGIVEBACK = WORD;
      RETURN;
      END SUBR GETARGS;


      FNCT MACEXPAND(DUM);
```

```
    DCL MACEXPAND PTR(TCKENTYP);
    DCL GETTOKEN  PTR(TCKENTYP),      $ FUNCTION WHICH RETURNS TOKEN
        KEEP      PTR(TCKENTYP),      $ TEMPORARY
        EXPTOP    PTR(EXPSTACKENT),   $ CURRENT MACRO BEING EXPANDED
        BODY      PTR(*MBODS),        $ MACRO BODY BEING EXPANDED
        SYMBOL    PTR(TOKENTYP),      $ NEXT ITEM IN MACRO BODY
        NEWEXPTOP PTR(EXPSTACKENT),   $ ARGUMENT SET UP TP LOOK LIKE MACRO
        ARGFN     SETLOBJ;            $ MAPPING OF ARGS ONTO ARG NO,S
    DCL ACTA PTR(*ARGTUP);
    SIZE DUM(PS);          $ DUMMY ARGUMENT
    SIZE  SYMBNO(PS),               $ POINTER TO MACRO BODY
       ARGNO (PS);                  $ ARGUMENT NUMBER
    ACCESS LEXMACEXP;

    IF MACEXPGIVEBACK -= .OM.  THEN
        KEEP = MACEXPGIVEBACK;
        MACEXPGIVEBACK =.OM. ;
        MACEXPAND = KEEP;
        RETURN;
        END IF;
/START/
    IF(EXPSTACKPTR = 0) THEN
        MACEXPAND = GETTOKEN(0);
        RETURN;
        END IF;
$    OTHERWISE WE ARE IN PROCESS OF EXPANDING A MACRO
/EXPAND/
    EXPTOP = EXPSTACK(EXPSTACKPTR);
    SYMBNO = MBP EXPTOP;
    BODY  = MBOD EXPTOP;
    IF SYMBNO > .NELT. MBOD THEN   $ .NELT. COMPUTES DIMENSION
        EXPSTACKPTR = EXPSTACKPTR - 1;
        GO TO START;
        END IF;
    SYMBOL = BODY(SYMBNO);  ARGFN = FORMARGS EXPTOP;
    IF ARGFN(TOK SYMBOL )  = .OM. THEN
        MBP EXPTOP = SYMBNO + 1;
        MACEXPAND = SYMBOL;
        RETURN;
        END IF;
$ SYMBOL IS AN ARGUMENT
    NEWEXPTOP = NEW(EXPSTACKENT);
    ARGNO = .CN. BITS(PS), ARGFN(TOK SYMBOL); $ CONBERT TO MIDL
                                              $ INTEGER
    MBP NEWEXPTOP = 1;
    ACTA = ACTARG EXPTOP;
    MBOD NEWEXPTOP = ACTA(ARGNO);
    EXPSTACKPTR = EXPSTACKPTR + 1;
    IF(EXPSTACKPTR > EXPSTACKDIM) THEN
        PRINTERROR(≠TOO MANY EMBEDDED MACROS.≠);
        END IF;
    EXPSTACK(EXPSTACKPTR) = NEWEXPTOP;
    GO TO EXPAND;
    END MACEXPAND;
```