```
-- file Pass1T.Mesa
-- last modified by Satterthwaite, June 22, 1978  11:01 AM

DIRECTORY
  ComData: FROM "comdata"
    USING [
      definitionsOnly,
      idANY, idBOOLEAN, idCARDINAL, idCHARACTER, idINTEGER, idLOCK,
      idREAL, idSTRING],
  LALRDefs: FROM "lalrdefs"
    USING [ActionEntry, ProductionInfo, Symbol, tokenID],
  P1Defs: FROM "p1defs" USING [InputLoc, LockId],
  SymDefs: FROM "symdefs" USING [HTNull],
  TreeDefs: FROM "treedefs" USING [
      NodeName, TreeLink, TreeMap,
      empty, nullid,
      freetree, listlength, maketree, mlextract, mlinsert, mlpop, mlpush,
      pushhashtree, pushlist, pushlittree, pushproperlist, pushsymtree,
      pushtree, pushstringlittree, setattr, setinfo, testtree, updatelist];

Pass1T: PROGRAM
    IMPORTS
        P1Defs, TreeDefs,
        dataPtr: ComData
    EXPORTS P1Defs SHARES LALRDefs =
  BEGIN  -- parse tree building
  OPEN TreeDefs;

 -- local data base (supplied by parser)

  v: DESCRIPTOR FOR ARRAY OF UNSPECIFIED;
  l: DESCRIPTOR FOR ARRAY OF CARDINAL;

  q: DESCRIPTOR FOR ARRAY OF LALRDefs.ActionEntry;

  prodData: DESCRIPTOR FOR ARRAY OF LALRDefs.ProductionInfo;

 -- initialization/termination

  AssignDescriptors: PUBLIC PROCEDURE [
        qd: DESCRIPTOR FOR ARRAY OF LALRDefs.ActionEntry,
        vd: DESCRIPTOR FOR ARRAY OF UNSPECIFIED,
        ld: DESCRIPTOR FOR ARRAY OF CARDINAL,
        pd: DESCRIPTOR FOR ARRAY OF LALRDefs.ProductionInfo] =
    BEGIN
    q ← qd;
    v ← vd;  l ← ld;
    prodData ← pd;
    RETURN
    END;


 -- the interpretation rules

  LinkToSource: PROCEDURE [index: CARDINAL] =
    BEGIN
    setinfo[l[index]];  RETURN
    END;

  access: BOOLEAN;
    private: BOOLEAN = FALSE;
    public: BOOLEAN = TRUE;
  -- initialization
    init: BOOLEAN = FALSE;
    equate: BOOLEAN = TRUE;
  -- machine dependent segments
    machineDep: BOOLEAN;

  ProcessQueue: PUBLIC PROCEDURE [qI, top: CARDINAL] =
    BEGIN
    i: CARDINAL;
    newV: UNSPECIFIED;
    sv1, sv2: TreeLink;
    FOR i IN [0..qI)
      DO
      top ← top-q[i].rtag.plength+1;  newV ← v[top];
```

```
SELECT prodData[q[i].transition].rule FROM

  0  => -- (no action)
     NULL;

 -- basic tree building
  1  => -- lhs             ::= id
        -- typeexp         ::= id
        -- range           ::= id
     pushhashtree[v[top]];
  2  => -- primary         ::= num
     pushlittree[v[top]];
  3  => -- pointerprefix   ::= POINTER
        -- begin           ::= BEGIN
        -- do              ::= DO
        -- statement       ::= NULL
     BEGIN
     mlpush[empty];  newV ← 1;
     END;
  4  => -- directory       ::=
        -- definitions     ::=
        -- imports         ::=
        -- exports         ::=
        -- shares          ::=
        -- indextype       ::=
        -- arglist         ::=
        -- returnlist      ::=
        -- elsepart        ::=
        -- otherpart       ::=
        -- statementlist   ::=
        -- enables         ::=
        -- forclause       ::=
        -- dotest          ::=
        -- optargs         ::=
        -- optexp          ::=
     BEGIN
     mlpush[empty];  newV ← 1;  l[top] ← P1Defs.InputLoc[];
     END;
  5  => -- declist         ::=
        -- catchitem       ::= ANY => statement
     newV ← 0;
  6  => -- includelist     ::= includeitem
        -- modulelist      ::= moduleitem
        -- pairlist        ::= pairitem
        -- variantlist     ::= variantitem ,
        -- bindlist        ::= binditem
        -- statementlist'  ::= statement ;
        -- casestmtlist    ::= casestmtitem ;
        -- caselabel       ::= casetest
        -- exitlist        ::= exititem
        -- catchhead       ::= catchcase ;
        -- lhslist         ::= lhs
        -- orderlist       ::= expitem
        -- keylist         ::= keyitem
        -- caseexplist     ::= caseexpitem ,
     newV ← 1;
  7  => -- includelist     ::= includelist , includeitem
        -- modulelist      ::= modulelist , moduleitem
        -- declist         ::= declist declaration ;
        -- pairlist        ::= pairlist , pairitem
        -- variantlist     ::= variantlist variantitem ,
        -- bindlist        ::= bindlist , binditem
        -- statementlist'  ::= statementlist'  statement ;
        -- casestmtlist    ::= casestmtlist casestmtitem ;
        -- caselabel       ::= caselabel , casetest
        -- exitlist        ::= exitlist ; exititem
        -- catchhead       ::= catchhead  catchcase ;
        -- lhslist         ::= lhslist , lhs
        -- orderlist       ::= orderlist , expitem
        -- keylist         ::= keylist , keyitem
        -- caseexplist     ::= caseexplist caseexpitem ,
     newV ← v[top]+1;
  8  => -- idlist          ::= idlist'
        -- identlist       ::= identlist'
        -- statementlist   ::= statementlist'
        -- explist         ::= orderlist
```

```
        -- explist          ::= keylist
    pushlist[v[top]];
  9 => -- directory         ::= DIRECTORY includelist ;
        -- imports          ::= IMPORTS modulelist
        -- fieldlist        ::= [ pairlist ]
        -- fieldlist        ::= [ typelist ]
        -- exits            ::= EXITS exitlist
        -- exits            ::= EXITS exitlist ;
    pushlist[v[top+1]];
 66 => -- array             ::= ARRAY
        -- initialization   ::= ← initvalue
        -- casehead         ::= SELECT exp FROM
    newV ← FALSE;
 85 => -- monitored         ::=
        -- ordered          ::=
        -- base             ::=
        -- enables          ::=
    BEGIN
    newV ← FALSE;  l[top] ← P1Defs.InputLoc[];
    END;
 67 => -- ordered           ::= ORDERED
        -- base             ::= BASE
        -- array            ::= PACKED ARRAY
        -- initialization   ::= = initvalue
        -- casehead         ::= WITH binditem SELECT optexp FROM
    newV ← TRUE;

  -- declaration processing
 10 => -- unit              ::= directory definitions module
    BEGIN
    pushtree[unit,3];  LinkToSource[top];
    END;
 11 => -- includeitem       ::= id : FROM string
    BEGIN
    mlpush[empty];  pushstringlittree[v[top+3]];  pushhashtree[v[top]];
    pushtree[diritem,-3];  LinkToSource[top];
    END;
222 => -- includeitem       ::= id : FROM string USING [ idlist ]
    BEGIN
    pushstringlittree[v[top+3]];  pushhashtree[v[top]];
    pushtree[diritem,-3];  LinkToSource[top];
    END;
 12 => -- module            ::= id : classhead = attributes block
        -- module           ::= id : defhead = attributes defbody
    BEGIN
    IF ~v[top+5] THEN mlinsert[empty, 2];
    mlpush[empty];
    pushtree[body,4];  setattr[1,FALSE];  setattr[1,FALSE];
    mlpush[mlextract[3]];  pushhashtree[v[top]];
    pushtree[declitem,-3];
    LinkToSource[top];  setattr[1,equate];  setattr[2,public];
    pushtree[module,5];  LinkToSource[top];
    END;
 13 => -- classhead          ::= PROGRAM arguments interface
    BEGIN
    dataPtr.definitionsOnly ← FALSE;  access ← private;
    mlpush[mlextract[5]];  mlpush[mlextract[5]];
    pushtree[programTC,2];
    mlpush[empty];  machineDep ← FALSE;
    END;
201 => -- classhead          ::= MONITOR arguments locks interface
    BEGIN
    dataPtr.definitionsOnly ← FALSE;  access ← private;
    sv1 ← mlextract[4];
    mlpush[mlextract[5]];  mlpush[mlextract[5]];
    pushtree[programTC,2];
    mlpush[sv1];  machineDep ← FALSE;
    END;
 14 => -- defhead           ::= DEFINITIONS shares
    BEGIN
    dataPtr.definitionsOnly ← TRUE;  access ← public;  sv1 ← mlpop[];
    mlpush[empty];  mlpush[empty];  mlpush[sv1];
    pushtree[definitionTC,0];  mlpush[empty];
    machineDep ← FALSE;
    END;
 21 => -- defbody           ::= begin declist END
```

```
  BEGIN
  pushlist[v[top+1]];  mlpush[empty];  newV ← TRUE;
  END;
202 => -- locks           ::=
  BEGIN
  pushhashtree[P1Defs.LockId[]];
  mlpush[empty];  pushtree[lambda,-2];  setattr[1,TRUE];
  END;
203 => -- locks           ::= LOCKS primary
  BEGIN
  mlpush[empty];   pushtree[lambda,-2];  setattr[1,FALSE];
  END;
204 => -- locks           ::= LOCKS primary USING id : typeexp
  BEGIN
  pushhashtree[v[top+3]];  mlinsert[empty,3];
  pushtree[declitem,-3];  LinkToSource[top+3];
  setattr[1,FALSE];  setattr[2,private];
  pushtree[lambda,-2];  setattr[1,FALSE];
  END;
15 => -- moduleitem      ::= id
  BEGIN
  pushhashtree[v[top]];  pushhashtree[v[top]];
  pushtree[item,2];  setattr[1,FALSE];  LinkToSource[top];
  END;
16 => -- moduleitem      ::= id : id
  BEGIN
  pushhashtree[v[top]];  pushhashtree[v[top+2]];
  pushtree[item,2];  setattr[1,TRUE];  LinkToSource[top];
  END;
22 => -- declaration     ::= identlist attributes entry typeexp initialization
  BEGIN
  IF v[top+2] # NodeName[none] THEN pushtree[v[top+2],1];
  pushtree[declitem,3];  LinkToSource[top];
  setattr[1,v[top+4]]; setattr[2,access];
  access ← v[top+1];
  END;
23 => -- declaration     ::= identlist attributes TYPE = attributes typeexp
  BEGIN
  access ← v[top+4];
  sv1 ← mlpop[];  pushtree[modeTC,0];  mlpush[sv1];
  pushtree[declitem,3];  LinkToSource[top];
  setattr[1,equate]; setattr[2,access];
  access ← v[top+1];
  END;
24 => -- attributes      ::=
  newV ← access;
25 => -- attributes      ::= PUBLIC
  BEGIN
  newV ← access;  access ← public;
  END;
26 => -- attributes      ::= PRIVATE
  BEGIN
  newV ← access;  access ← private;
  END;
223 => -- entry           ::=
  BEGIN
  newV ← NodeName[none];  l[top] ← P1Defs.InputLoc[];
  END;
224 => -- entry           ::= ENTRY
  newV ← NodeName[entry];
225 => -- entry           ::= INTERNAL
  newV ← NodeName[internal];
27 => -- idlist'          ::= id
      -- identlist'       ::= id :
  BEGIN
  pushhashtree[v[top]];
  newV ← -1;
  END;
28 => -- idlist'          ::= id , idlist'
      -- identlist'       ::= id , identlist'
  BEGIN
  pushhashtree[v[top]];
  newV ← v[top+2]-1;
  END;
29 => -- typeid           ::= INTEGER
  pushsymtree[dataPtr.idINTEGER];
```

```
30 => -- typeid          ::= CARDINAL
  pushsymtree[dataPtr.idCARDINAL];
31 => -- typeid          ::= CHARACTER
  pushsymtree[dataPtr.idCHARACTER];
32 => -- typeid          ::= BOOLEAN
  pushsymtree[dataPtr.idBOOLEAN];
217 => -- typeid          ::= REAL
  pushsymtree[dataPtr.idREAL];
33 => -- typeid          ::= STRING
  pushsymtree[dataPtr.idSTRING];
34 => -- typeid          ::= id . id
  BEGIN
  pushhashtree[v[top]];  pushhashtree[v[top+2]];
  pushtree[dot,2];
  END;
35 => -- typeid          ::= id id
  BEGIN
  pushhashtree[v[top+1]];  pushhashtree[v[top]];
  pushtree[discrimTC,2];
  END;
36 => -- typeid          ::= id typeid
  BEGIN
  pushhashtree[v[top]];  pushtree[discrimTC,2];
  END;
37 => -- typecons        ::= interval
  BEGIN
  pushsymtree[dataPtr.idINTEGER];  pushtree[subrangeTC,-2];
  END;
38 => -- typecons        ::= id interval
      -- range          ::= id interval
  BEGIN
  pushhashtree[v[top]];  pushtree[subrangeTC,-2];
  END;
39 => -- typecons        ::= typeid interval
      -- range          ::= typeid interval
  pushtree[subrangeTC,2];
40 => -- typecons        ::= { idlist }
  BEGIN
  pushtree[enumeratedTC,1];  setattr[1,access];
  END;
41 => -- typecons        ::= monitored dependent RECORD reclist
  BEGIN
  IF ~v[top]
    THEN pushtree[recordTC,1]
    ELSE
      BEGIN
      sv1 ← mlpop[];  v[top+2] ← listlength[sv1];
      sv1 ← updatelist[sv1, DetachItem];  sv1 ← freetree[sv1];
      pushlist[v[top+2]+1];  pushtree[monitoredTC,1];
      END;
  setattr[1,machineDep]; setattr[2,v[top+3]];
  machineDep ← v[top+1];
  END;
42 => -- typecons        ::= ordered base pointertype
  BEGIN
  sv2 ← maketree[pointerTC,1];
  sv1 ← mlpop[];
  mlpush[sv2];  setattr[1,v[top]];  setattr[2,v[top+1]];
  IF sv1 # empty
    THEN  BEGIN  mlpush[sv1];  pushtree[subrangeTC,2]  END;
  END;
43 => -- typecons        ::= array indextype OF typeexp
  BEGIN
  pushtree[arrayTC,2]; setattr[1,v[top]];
  END;
44 => -- typecons        ::= DESCRIPTOR FOR typeexp
  pushtree[arraydescTC,1];
45 => -- typecons        ::= transfermode arguments
  pushtree[v[top],2];
212 => -- typecons        ::= id RELATIVE typeexp
  BEGIN
  pushhashtree[v[top]];  pushtree[relativeTC,-2];
  END;
213 => -- typecons        ::= typeid RELATIVE typeexp
  pushtree[relativeTC,2];
46 => -- typecons        ::= LONG typeexp
```

```
  pushtree[longTC,1];
47 => -- typecons       ::= FRAME [ id ]
  BEGIN
  pushhashtree[v[top+2]];  pushtree[frameTC,1];
  END;
205 => -- monitored      ::= MONITORED
  BEGIN
  pushhashtree[P1Defs.LockId[]];  pushsymtree[dataPtr.idLOCK];
  mlpush[empty];
  pushtree[declitem,3];  LinkToSource[top];
  setattr[1,FALSE];  setattr[2,access];
  newV ← TRUE;
  END;
48 => -- dependent      ::=
  newV ← machineDep;
49 => -- dependent      ::= MACHINE DEPENDENT
  BEGIN
  newV ← machineDep;  machineDep ← TRUE;
  END;
50 => -- reclist        ::= [ pairlist ]
      -- reclist        ::= [ typelist ]
  BEGIN
  pushlist[v[top+1]];  newV ← FALSE;
  END;
51 => -- reclist        ::= [ pairlist , variantpair ]
  BEGIN
  pushlist[v[top+1]+1];  newV ← TRUE;
  END;
52 => -- reclist        ::= [ variantpair ]
  newV ← TRUE;
53 => -- reclist        ::= [ variantpart ]
  BEGIN
  AnonField[mlpop[],top];  newV ← TRUE;
  END;
54 => -- pairitem       ::= identlist attributes typeexp
      -- variantpair    ::= identlist attributes variantpart
  BEGIN
  mlpush[empty];
  pushtree[declitem,3];  LinkToSource[top];
  setattr[1,FALSE]; setattr[2,access];
  access ← v[top+1];
  END;
55 => -- typelist       ::= typecons
      -- typelist       ::= typeid
  BEGIN
  AnonField[mlpop[],top];
  newV ← -1;
  END;
56 => -- typelist       ::= id
  BEGIN
  pushhashtree[v[top]];  AnonField[mlpop[],top];
  newV ← -1;
  END;
57 => -- typelist       ::= typecons , typelist
      -- typelist       ::= typeid , typelist
  BEGIN
  AnonField[mlextract[-(v[top+2]-1)],top];
  newV ← v[top+2]-1;
  END;
58 => -- typelist       ::= id , typelist
  BEGIN
  pushhashtree[v[top]];  AnonField[mlpop[],top];
  newV ← v[top+2]-1;
  END;
59 => -- variantpart    ::= SELECT vcasehead FROM variantlist ENDCASE
  BEGIN
  pushlist[v[top+3]];  pushtree[unionTC,2];  setattr[1,v[top+1]];
  END;
60 => -- vcasehead      ::= id : attributes tagtype
  BEGIN
  pushhashtree[v[top]];  mlinsert[empty,3];
  pushtree[declitem,-3];  LinkToSource[top];
  setattr[1,FALSE]; setattr[2,access];
  access ← v[top+2];  newV ← FALSE;
  END;
61 => -- vcasehead      ::= COMPUTED tagtype
```

```
  BEGIN
  AnonField[mlpop[],top];  newV ← FALSE;
  END;
62 => -- vcasehead      ::= OVERLAID tagtype
  BEGIN
  AnonField[mlpop[],top];  newV ← TRUE;
  END;
63 => -- tagtype        ::= *
  pushtree[implicitTC,0];
64 => -- variantitem    ::= idlist => subreclist
  BEGIN
  sv1 ← maketree[variantTC,1];
  pushtree[modeTC,0];  mlpush[sv1];
  setattr[1,machineDep];  setattr[2,v[top+2]];
  pushtree[declitem,3];  LinkToSource[top];
  setattr[1,TRUE]; setattr[2,access];
  END;
65 => -- subreclist     ::= NULL
  BEGIN
  mlpush[empty];  newV ← FALSE;
  END;
68 => -- pointertype    ::= pointerprefix
  pushsymtree[dataPtr.idANY];
69 => -- transfertype   ::= PROCEDURE
  newV ← NodeName[procTC];
70 => -- transfertype   ::= PORT
  newV ← NodeName[portTC];
71 => -- transfertype   ::= SIGNAL
  newV ← NodeName[signalTC];
72 => -- transfertype   ::= ERROR
  newV ← NodeName[errorTC];
73 => -- transfertype   ::= PROCESS
  newV ← NodeName[processTC];
74 => -- transfertype   ::= PROGRAM
  newV ← NodeName[programTC];
75 => -- initialization ::=
  BEGIN
  mlpush[empty];  newV ← FALSE;
  END;
76 => -- initvalue      ::= procaccess block
  BEGIN
  IF ~v[top+1] THEN  mlinsert[empty,2];
  mlpush[empty];
  pushtree[body,4];  setattr[1,FALSE];  setattr[2,FALSE];
  access ← v[top];
  END;
77 => -- initvalue      ::= CODE
  pushtree[signalinit,0];
78 => -- initvalue      ::= MACHINE CODE BEGIN codelist END
  BEGIN
  pushproperlist[v[top+3]];  pushtree[inline,1];
  END;
214 => -- codelist       ::= orderlist
  BEGIN
  pushlist[v[top]];  newV ← 1;
  END;
215 => -- codelist       ::= codelist ; orderlist
  BEGIN
  pushlist[v[top+2]];  newV ← v[top]+1;
  END;
79 => -- procaccess     ::=
  BEGIN
  newV ← access;  access ← private;
  END;

80 => -- statement      ::= lhs
  BEGIN
  sv1 ← mlpop[];  mlpush[sv1];
  IF ~testtree[sv1,apply]
    THEN  BEGIN  mlpush[empty];  pushtree[apply,2]  END;
  LinkToSource[top];
  END;
81 => -- statement      ::= lhs ← exp
  BEGIN
  pushtree[assign,2];  LinkToSource[top];
  END;
```

```
82 => -- statement      ::= [ explist ] ← exp
  BEGIN
  pushtree[extract,2];  LinkToSource[top];
  END;
83 => -- statement      ::= block
  BEGIN
  IF v[top] THEN BEGIN  pushtree[block,2];  LinkToSource[top]  END;
  sv1 ← mlextract[2];
  IF sv1 # empty
    THEN
      BEGIN
      mlpush[sv1];  pushtree[openstmt,-2];  LinkToSource[top];
      END;
  END;
84 => -- statement      ::= IF exp THEN statement elsepart
  BEGIN
  pushtree[ifstmt,3];  LinkToSource[top];
  END;
86 => -- statement      ::= casehead casestmtlist ENDCASE otherpart
  BEGIN
  sv1 ← mlpop[];  pushproperlist[v[top+1]];  mlpush[sv1];
  IF v[top]
    THEN pushtree[bindstmt,4]
    ELSE pushtree[casestmt,3];
  LinkToSource[top];
  END;
87 => -- statement      ::= forclause dotest do enables statementlist doexit ENDLOOP
  BEGIN
  IF v[top+3]
    THEN
      BEGIN
      sv1 ← mlpop[];  sv2 ← mlpop[];
      pushtree[enable,2];  LinkToSource[top+3];
      mlpush[sv2];  mlpush[sv1];
      END;
  pushtree[dostmt,6];  LinkToSource[top];
  END;
90 => -- statement      ::= EXIT
  BEGIN
  pushtree[exit,0];  LinkToSource[top];
  END;
216 => -- statement      ::= LOOP
  BEGIN
  pushtree[loop,0];  LinkToSource[top];
  END;
91 => -- statement      ::= GOTO id
  BEGIN
  pushhashtree[v[top+1]];  pushtree[goto,1];
  LinkToSource[top];
  END;
92 => -- statement      ::= GO TO id
  BEGIN
  pushhashtree[v[top+2]];  pushtree[goto,1];
  LinkToSource[top];
  END;
93 => -- statement      ::= RETURN optargs
  BEGIN
  pushtree[return,1];  LinkToSource[top];
  END;
94 => -- statement      ::= transfer lhs
  BEGIN
  pushtree[v[top],1];  LinkToSource[top];
  END;
207 => -- statement      ::= WAIT lhs
  BEGIN
  pushtree[wait,1];  LinkToSource[top];
  END;
95 => -- statement      ::= ERROR
  BEGIN
  pushtree[syserror,0];  LinkToSource[top];
  END;
96 => -- statement      ::= STOP
  BEGIN
  mlpush[empty];  pushtree[stop,1];  LinkToSource[top];
  END;
97 => -- statement      ::= STOP [ ! catchlist ]
```

```
  BEGIN
  sv1 ← mlpop[];
  pushlist[v[top+3]];  mlpush[sv1];  pushtree[catchphrase,2];
  pushtree[stop,1];  LinkToSource[top];
  END;
98 => -- statement      ::= NULL
  BEGIN
  pushtree[nullstmt,0];  LinkToSource[top];
  END;
99 => -- statement      ::= RESUME optargs
  BEGIN
  pushtree[resume,1];  LinkToSource[top];
  END;
100 => -- statement     ::= CONTINUE
  BEGIN
  pushtree[continue,0];  LinkToSource[top];
  END;
101 => -- statement     ::= RETRY
  BEGIN
  pushtree[retry,0];  LinkToSource[top];
  END;
102 => -- statement     ::= lhs ← STATE
  BEGIN
  pushtree[dst,1];  LinkToSource[top];
  END;
89 => -- block          ::= blockhead exits END
  BEGIN
  IF v[top]
    THEN
      BEGIN
      sv1 ← mlpop[];  pushtree[block,2];  LinkToSource[top];
      mlpush[sv1];  newV ← FALSE;
      END;
  pushtree[label,2];  LinkToSource[top];
  END;
17 => -- blockhead      ::= begin enables declist statementlist
  BEGIN
  IF v[top+2] = 0
    THEN  newV ← FALSE
    ELSE
      BEGIN
      sv1 ← mlpop[];  pushlist[v[top+2]];  mlpush[sv1];  newV ← TRUE;
      END;
  IF v[top+1]
    THEN
      BEGIN
      IF newV
        THEN  BEGIN  pushtree[block,2];  LinkToSource[top+2]  END;
      pushtree[enable,2];  LinkToSource[top+1];  newV ← FALSE;
      END;
  END;
18 => -- begin          ::= BEGIN OPEN bindlist ;
      -- do             ::= DO OPEN bindlist ;
  pushlist[v[top+2]];
19 => -- binditem       ::= exp
  BEGIN
  mlpush[nullid];  pushtree[item,-2];  LinkToSource[top];
  END;
20 => -- binditem       ::= id : exp
  BEGIN
  pushhashtree[v[top]];  pushtree[item,-2];
  LinkToSource[top];
  END;
105 => -- casestmtitem   ::= caselabel => statement
       -- caseexpitem    ::= caselabel => exp
       -- catchcase      ::= lhslist => statement
  BEGIN
  sv1 ← mlpop[];
  pushlist[v[top]];  mlpush[sv1];
  pushtree[item,2];  LinkToSource[top];
  END;
106 => -- casetest       ::= optrelation
  BEGIN
  mlpush[empty];  pushtree[v[top],-2];
  END;
107 => -- casetest       ::= exp
```

```
  BEGIN
  mlpush[empty];  pushtree[relE,-2];
  END;
108 => -- forclause      ::= FOR id ← exp , exp
  BEGIN
  sv1 ← mlpop[];  sv2 ← mlpop[];
  pushhashtree[v[top+1]];  mlpush[sv2];  mlpush[sv1];
  pushtree[forseq,3];
  END;
109 => -- forclause      ::= FOR id direction IN range
  BEGIN
  pushhashtree[v[top+1]];  pushtree[v[top+2],-2];
  END;
110 => -- forclause      ::= THROUGH range
  BEGIN
  mlpush[empty];  pushtree[upthru,-2];
  END;
111 => -- direction      ::=
       -- direction      ::= INCREASING
  newV ← NodeName[upthru];
112 => -- direction      ::= DECREASING
  newV ← NodeName[downthru];
113 => -- dotest         ::= UNTIL exp
  pushtree[not,1];
114 => -- doexit         ::=
  BEGIN
  mlpush[empty];  mlpush[empty];
  END;
115 => -- doexit         ::= REPEAT exitlist
       -- doexit         ::= REPEAT exitlist ;
  BEGIN
  pushlist[v[top+1]];  mlpush[empty];
  END;
116 => -- doexit         ::= REPEAT exitlist ; FINISHED => statement
       -- doexit         ::= REPEAT exitlist ; FINISHED => statement ;
  BEGIN
  sv1 ← mlpop[];  pushlist[v[top+1]];  mlpush[sv1];
  END;
117 => -- doexit         ::= REPEAT FINISHED => statement
       -- doexit         ::= REPEAT FINISHED => statement ;
  mlinsert[empty,2];
118 => -- exititem       ::= idlist => statement
  BEGIN
  pushtree[item,2];  LinkToSource[top];
  END;
119 => -- enables        ::= ENABLE catchitem ;
  BEGIN
  sv1 ← mlpop[];
  pushlist[v[top+1]];  mlpush[sv1];  pushtree[catchphrase,2];
  newV ← TRUE;
  END;
120 => -- enables        ::= ENABLE BEGIN catchlist END ;
  BEGIN
  sv1 ← mlpop[];
  pushlist[v[top+2]];  mlpush[sv1];  pushtree[catchphrase,2];
  newV ← TRUE;
  END;
121 => -- enables        ::= ENABLE BEGIN catchhead END ;
  BEGIN
  pushlist[v[top+2]];  mlpush[empty];  pushtree[catchphrase,2];
  newV ← TRUE;
  END;
122 => -- catchlist      ::= catchhead catchitem
  newV ← v[top] + v[top+1];
123 => -- catchitem      ::= catchcase
  BEGIN
  mlpush[empty];  newV ← 1;
  END;
124 => -- statementlist  ::= statementlist' statement
  pushlist[v[top]+1];
125 => -- transfer       ::= SIGNAL
       -- transferop     ::= SIGNAL
  newV ← NodeName[signal];
126 => -- transfer       ::= ERROR
       -- transferop     ::= ERROR
  newV ← NodeName[error];
```

```
218 => -- transfer       ::= RETURN WITH ERROR
  newV ← NodeName[xerror];
127 => -- transfer       ::= START
       -- transferop     ::= START
  newV ← NodeName[start];
128 => -- transfer       ::= RESTART
  newV ← NodeName[restart];
208 => -- transfer       ::= JOIN
       -- transferop     ::= JOIN
  newV ← NodeName[join];
209 => -- transfer       ::= NOTIFY
  newV ← NodeName[notify];
210 => -- transfer       ::= BROADCAST
  newV ← NodeName[broadcast];
129 => -- transfer       ::= TRANSFER WITH
  newV ← NodeName[lst];
130 => -- transfer       ::= RETURN WITH
  newV ← NodeName[lstf];

-- expression processing
140 => -- keyitem        ::= id : optexp
  BEGIN
  pushhashtree[v[top]];  pushtree[item,-2];
  END;
141 => -- exp            ::= transferop lhs
       -- primary        ::= typeop [ typeexp ]
  pushtree[v[top],1];
142 => -- sum            ::= sum addop product
       -- product        ::= product multop factor
  pushtree[v[top+1],2];
143 => -- exp            ::= IF exp THEN exp ELSE exp
  pushtree[ifexp,3];
144 => -- exp            ::= casehead caseexplist ENDCASE => exp
  BEGIN
  sv1 ← mlpop[];
  pushproperlist[v[top+1]];  mlpush[sv1];
  IF v[top]
    THEN pushtree[bindexp,4]
    ELSE pushtree[caseexp,3];
  LinkToSource[top];
  END;
145 => -- exp            ::= lhs ← exp
  pushtree[assignx,2];
146 => -- transferop     ::= NEW
  newV ← NodeName[new];
211 => -- transferop     ::= FORK
  newV ← NodeName[fork];
147 => -- disjunct       ::= disjunct OR conjunct
  pushtree[or,2];
148 => -- conjunct       ::= conjunct AND negation
  pushtree[and,2];
149 => -- negation       ::= not relation
  pushtree[not,1];
150 => -- relation       ::= sum optrelation
  pushtree[v[top+1],2];
151 => -- optrelation    ::= not relationtail
  newV ← NodeName[SELECT NodeName[v[top+1]] FROM
    relE => relN,
    relN => relE,
    relL => relGE,
    relLE => relG,
    relG => relLE,
    relGE => relL,
    in => notin,
    notin => in,
    ENDCASE => v[top+1]];
152 => -- relop          ::= =
  newV ← NodeName[relE];
153 => -- relop          ::= #
  newV ← NodeName[relN];
154 => -- relop          ::= <
  newV ← NodeName[relL];
155 => -- relop          ::= <=
  newV ← NodeName[relLE];
156 => -- relop          ::= >
  newV ← NodeName[relG];
```

```
157 => -- relop          ::= >=
  newV ← NodeName[relGE];
158 => -- relationtail   ::= IN range
  newV ← NodeName[in];
159 => -- interval       ::= [ bounds ]
  pushtree[intCC,2];
160 => -- interval       ::= [ bounds )
  pushtree[intCO,2];
161 => -- interval       ::= ( bounds ]
  pushtree[intOC,2];
162 => -- interval       ::= ( bounds )
  pushtree[intOO,2];
163 => -- addop          ::= +
  newV ← NodeName[plus];
164 => -- addop          ::= -
  newV ← NodeName[minus];
165 => -- multop         ::= *
  newV ← NodeName[times];
166 => -- multop         ::= /
  newV ← NodeName[div];
167 => -- multop         ::= MOD
  newV ← NodeName[mod];
168 => -- factor         ::= - prim
  pushtree[uminus,1];
226  => -- primary        ::= lnum
  BEGIN
  pushlittree[v[top]];  pushtree[mwconst,1];
  END;
169 => -- primary        ::= char
  BEGIN
  pushlittree[v[top]];  pushtree[clit,1];
  END;
170 => -- primary        ::= string
  pushstringlittree[v[top]];
219 => -- primary        ::= lstring
  BEGIN
  pushstringlittree[v[top]];  pushtree[llit,1];
  END;
171 => -- primary       ::= [ explist ]
  BEGIN
  mlpush[empty];  pushtree[apply,-2];
  END;
172 => -- primary        ::= prefixop [ orderlist ]
  BEGIN
  pushlist[v[top+2]];  pushtree[v[top],1];
  END;
220 => -- primary        ::= INTEGER [ explist ]
  BEGIN
  pushsymtree[dataPtr.idINTEGER];  pushtree[apply,-2];
  END;
221 => -- primary        ::= CARDINAL [ explist ]
  BEGIN
  pushsymtree[dataPtr.idCARDINAL];  pushtree[apply,-2];
  END;
173 => -- primary        ::= @ lhs
  pushtree[addr,1];
174 => -- primary        ::= DESCRIPTOR [ desclist ]
  pushtree[arraydesc,1];
175 => -- desclist       ::= exp , exp
  BEGIN
  mlpush[empty];  pushlist[3];
  END;
176 => -- desclist       ::= exp , exp , typeexp
  pushlist[3];
177 => -- prefixop       ::= LONG
  newV ← NodeName[lengthen];
178 => -- prefixop       ::= ABS
  newV ← NodeName[abs];
179 => -- prefixop       ::= MIN
  newV ← NodeName[min];
180 => -- prefixop       ::= MAX
  newV ← NodeName[max];
181 => -- prefixop       ::= BASE
  newV ← NodeName[base];
182 => -- prefixop       ::= LENGTH
  newV ← NodeName[length];
```

```
        182 => -- prefixop       ::= LENGTH
          newV ← NodeName[length];
        183 => -- typeop         ::= SIZE
          newV ← NodeName[size];
        184 => -- typeop         ::= FIRST
          newV ← NodeName[first];
        185 => -- typeop         ::= LAST
          newV ← NodeName[last];
        186 => -- lhs            ::= LOOPHOLE [ exp ]
          BEGIN
          mlpush[empty]; pushtree[loophole,2];
          END;
        187 => -- lhs            ::= LOOPHOLE [ exp , typeexp ]
          pushtree[loophole,2];
        188 => -- lhs            ::= memory [ exp ]
          pushtree[v[top],1];
        189 => -- qualifier      ::= [ explist ]
          pushtree[apply,2];
        190 => -- qualifier      ::= [ explist ! catchlist ]
          BEGIN
          sv1 ← mlpop[];
          pushlist[v[top+3]]; mlpush[sv1]; pushtree[catchphrase,2];
          pushtree[apply,3];
          END;
        191 => -- qualifier      ::= . id
          BEGIN
          pushhashtree[v[top+1]]; pushtree[dot,2];
          END;
        192 => -- qualifier      ::= ↑
          pushtree[uparrow,1];
        193 => -- memory         ::= MEMORY
          newV ← NodeName[memory];
        194 => -- memory         ::= REGISTER
          newV ← NodeName[register];

       -- error or unimplemented
        ENDCASE => ERROR;

      v[top] ← newV;
      ENDLOOP;
    RETURN
    END;

    DetachItem: TreeMap =
      BEGIN
      mlpush[t]; RETURN [empty]
      END;


-- shared processing routines

  AnonField: PROCEDURE [type: TreeLink, top: CARDINAL] =
    BEGIN
    mlpush[nullid]; mlpush[type]; mlpush[empty];
    pushtree[declitem,3]; LinkToSource[top];
    setattr[1,FALSE]; setattr[2,access];
    RETURN
    END;


-- error recovery

  TokenValue: PUBLIC PROCEDURE [s: LALRDefs.Symbol] RETURNS [UNSPECIFIED] =
    BEGIN
    OPEN LALRDefs;
    RETURN [SELECT s FROM
      tokenID => SymDefs.HTNull,
      ENDCASE => 0]
    END;

  END.
```