```
-- file SymTab.Mesa
-- last modified by Johnsson, April 7, 1978  8:06 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  ControlDefs: FROM "controldefs",
  StringDefs: FROM "stringdefs",
  SymbolTable: FROM "symboltable",
  SymbolTableDefs: FROM "symboltabledefs",
  SymDefs: FROM "symdefs",
  SymSegDefs: FROM "symsegdefs",
  SymTabDefs: FROM "symtabdefs",
  TableDefs: FROM "tabledefs";

SymTab: PROGRAM
    IMPORTS
        StringDefs, TableDefs, SymTabDefs,
        own: SymbolTable
    EXPORTS SymTabDefs SHARES SymDefs =
 PUBLIC
  BEGIN
  OPEN TableDefs, SymTabDefs, SymDefs;

  SubString: TYPE = StringDefs.SubString;

  StaticNestError: SIGNAL = CODE;

-- tables defining the current symbol table

  hashVector: PRIVATE ARRAY HVIndex OF HTIndex;
  ht: PRIVATE DESCRIPTOR FOR ARRAY --HTIndex-- OF HTRecord;

  hashVec: PRIVATE DESCRIPTOR FOR ARRAY OF HTIndex = -- DESCRIPTOR[hashVector]
    DESCRIPTOR[BASE[hashVector], HVLength];     -- ??
  htb: PRIVATE TableBase;                 -- hash table
  ssb: PRIVATE STRING;                    -- id string
  seb: PRIVATE TableBase;                 -- se table
  ctxb: PRIVATE TableBase;                -- context table
  mdb: PRIVATE TableBase;                 -- module directory base
  bb: PRIVATE TableBase;                  -- body table

  UpdateBases: PRIVATE TableNotifier =
    BEGIN  -- called whenever the main symbol table is repacked
    own.hashVec ← hashVec;
    htb ← base[httype];
    own.ssb ← ssb ← LOOPHOLE[base[sstype], STRING];
    own.ht ← ht ← DESCRIPTOR[htb, LENGTH[ht]];
    own.seb ← seb ← base[setype];
    own.ctxb ← ctxb ← base[ctxtype];  own.mdb ← mdb ← base[mdtype];
    own.bb ← bb ← base[bodytype];
    own.tb ← base[SymSegDefs.treetype];
    own.ltb ← base[SymSegDefs.lttype];
    own.extb ← base[SymSegDefs.exttype];
    own.notifier[own];
    RETURN
    END;

  AllocateHash: PRIVATE PROCEDURE RETURNS [HTIndex] =
    BEGIN
    hti: HTIndex = LENGTH[ht];
    [] ← Allocate[httype, SIZE[HTRecord]];
    own.ht ← ht ← DESCRIPTOR[htb, LENGTH[ht]+1];
    ht[hti] ← HTRecord[
        anyInternal: FALSE,
        anyPublic: FALSE,
        link: HTNull,
        ssIndex: ssb.length];
    RETURN [hti]
    END;

  hashblock: PROCEDURE RETURNS [base: POINTER, length: CARDINAL] =
    BEGIN
    base ← BASE[hashVector];  length ← LENGTH[hashVector];  RETURN
    END;

 -- variables for building the symbol string
```

```
  ssw: PRIVATE TableIndex;


  tableOpen: PRIVATE BOOLEAN ← FALSE;

  symtabinit: PROCEDURE =
    BEGIN  -- called to set up the compiler's symbol table
    i: HVIndex;
    IF tableOpen THEN symtaberase[];
    own.notifier ← own.NullNotifier;
    own.stHandle ← NIL;  own.sourceFile ← NIL;
    FOR i IN HVIndex DO hashVector[i] ← HTNull ENDLOOP;
    ht ← DESCRIPTOR[NIL, 0];
    AddNotify[UpdateBases];
    ssw ← Allocate[sstype, SIZE[StringBody]] + SIZE[StringBody];
    ssb↑ ← StringBody[length:0, maxlength:0, text:];
    IF AllocateHash[] # HTNull THEN ERROR;
    IF makenonctxse[SIZE[nil constructor SERecord]] # SENull THEN ERROR;
    (seb+CSENull)↑ ← SERecord[mark3: FALSE, mark4: FALSE,
        sebody: constructor[nil[]]];
    IF makenonctxse[SIZE[mode constructor SERecord]] # typeTYPE THEN ERROR;
    (seb+typeTYPE)↑ ← SERecord[mark3: TRUE, mark4: TRUE,
        sebody: constructor[mode[]]];
    IF Allocate[ctxtype, SIZE [nil CTXRecord]] # CTXNull THEN ERROR;
    (ctxb+CTXNull)↑ ← CTXRecord[snNil, ISENull, 1Z, nil[]];
    tableOpen ← TRUE;  RETURN
    END;

  symtaberase: PROCEDURE =
    BEGIN  --  releases storage allocated for the symbol table blocks
    tableOpen ← FALSE;
    DropNotify[UpdateBases];
    RETURN
    END;


-- hash entry creation

  EnterString: PROCEDURE [s: SubString] RETURNS [hti: HTIndex] =
    BEGIN
    OPEN StringDefs;
    hvi: HVIndex;
    desc: SubStringDescriptor ← [base:ssb, offset:, length:];
    CharsPerWord: CARDINAL = AltoDefs.CharsPerWord;
    offset, length, nw: CARDINAL;
    ssi: TableIndex;
    hvi ← HashValue[s];
    FOR hti ← hashVec[hvi], ht[hti].link UNTIL hti = HTNull
      DO
      desc.offset ← ht[hti-1].ssIndex;
      desc.length ← ht[hti].ssIndex - desc.offset;
      IF EqualSubStrings[s, @desc] THEN RETURN [hti];
      ENDLOOP;
    offset ← ssb.length;  length ← s.length;
    nw ← LOOPHOLE[offset+length+(CharsPerWord-1) - ssb.maxlength, CARDINAL]/CharsPerWord;
    IF nw # 0
      THEN
        BEGIN  ssi ← Allocate[sstype, nw];
        IF ssi # ssw THEN ERROR;
        ssw ← ssw + nw;
        ssb↑ ← StringBody[
                length: ssb.length,
                maxlength: ssb.maxlength + nw*CharsPerWord,
                text:];
        END;
    AppendSubString[ssb, s];
    hti ← AllocateHash[];
    ht[hti].link ← hashVec[hvi];  hashVec[hvi] ← hti;
    RETURN
    END;


-- lexical level accounting

  nextlevel: PROCEDURE [cl: ContextLevel] RETURNS [nl: ContextLevel] =
```

```
    BEGIN  -- increments static height, checking for overflow
    IF cl+1 < MaxContextLevel
      THEN  nl ← cl+1
      ELSE  BEGIN  SIGNAL StaticNestError;  nl ← cl  END;
    RETURN
    END;


  -- context table manipulation

  makenewctx: PROCEDURE [level: ContextLevel] RETURNS [ctx: CTXIndex] =
    BEGIN  -- makes a non-include context entry
    ctx ← Allocate[ctxtype, SIZE[simple CTXRecord]];
    (ctxb+ctx)↑ ← [
        sn: snNil,
        selist: ISENull,
        ctxlevel: level,
        extension: simple[ctxNew: CTXNull]];
    RETURN
    END;

  resetctxlist: PROCEDURE [ctx: CTXIndex] =
    BEGIN  -- change the list for ctx to a proper chain
    OPEN (ctxb+ctx);
    sei: ISEIndex = selist;
    IF sei # SENull
      THEN BEGIN  selist ← NextSe[selist];  setselink[sei, ISENull]  END;
    RETURN
    END;


  firstvisiblese: PROCEDURE [ctx: CTXIndex] RETURNS [sei: ISEIndex] =
    BEGIN
    sei ← (ctxb+ctx).selist;
    WHILE sei # SENull AND (seb+sei).ctxnum # ctx
      DO sei ← NextSe[sei] ENDLOOP;
    RETURN
    END;

  visiblectxentries: PROCEDURE [ctx: CTXIndex] RETURNS [n: CARDINAL] =
    BEGIN
    sei: ISEIndex;
    IF ctx = CTXNull THEN RETURN [0];
    WITH (ctxb+ctx) SELECT FROM
      included =>  IF ~ctxreset THEN RETURN [0];
      ENDCASE;
    n ← 0;
    FOR sei ← (ctxb+ctx).selist, NextSe[sei] UNTIL sei = SENull
      DO
      IF (seb+sei).ctxnum = ctx THEN n ← n+1;
      ENDLOOP;
    RETURN
    END;


  ContextVariant: PROCEDURE [ctx: CTXIndex] RETURNS [ISEIndex] =
    BEGIN
    sei: ISEIndex;
    IF ctx = CTXNull THEN RETURN [ISENull];
    FOR sei ← (ctxb+ctx).selist, NextSe[sei] UNTIL sei = SENull
      DO
      IF TypeForm[(seb+sei).idtype] = union THEN RETURN [sei];
      ENDLOOP;
    RETURN [ISENull]
    END;



  -- semantic entry creation

  makeSEChain: PROCEDURE [ctx: CTXIndex, n: CARDINAL, linked: BOOLEAN] RETURNS [sechain: ISEIndex] =
    BEGIN
    sei: ISEIndex;
    IF n = 0 THEN RETURN [ISENull];
    sechain ← Allocate[setype,
            (n-1)*SIZE[sequential id SERecord] +
```

```
            (IF linked
              THEN SIZE[linked id SERecord]
              ELSE SIZE[terminal id SERecord])];
  sei ← sechain;
  THROUGH [1..n)
    DO
    (seb+sei)↑ ← [mark3: FALSE, mark4: FALSE,
            sebody: id[,,ctx,,,,,,HTNull,,sequential[]]];
    sei ← sei + SIZE[sequential id SERecord];
    ENDLOOP;
  IF linked
    THEN
      (seb+sei)↑ ← SERecord[mark3: FALSE, mark4: FALSE,
            sebody: id[,,ctx,,,,,,HTNull,,linked[ISENull]]]
    ELSE
      (seb+sei)↑ ← SERecord[mark3: FALSE, mark4: FALSE,
            sebody: id[,,ctx,,,,,,HTNull,,terminal[]]];
  RETURN
  END;


makectxse: PROCEDURE [hti: HTIndex, ctx: CTXIndex] RETURNS [sei: ISEIndex] =
  BEGIN  -- makes an id-tagged entry for a declared item
  next, psei: ISEIndex;
  sei ← Allocate[setype, SIZE[linked id SERecord]];
  IF ctx = CTXNull
    THEN  next ← ISENull
    ELSE
      BEGIN  psei ← (ctxb+ctx).selist;
      IF psei = SENull
          THEN next ← sei
          ELSE  BEGIN  next ← NextSe[psei];  setselink[psei, sei]  END;
      (ctxb+ctx).selist ← sei;
      END;
  (seb+sei)↑ ← SERecord[
      mark3: FALSE,
      mark4: FALSE,
      sebody: id[,,ctx,,,,,,hti,,linked[link: next]]];
  RETURN
  END;

NameClash: SIGNAL [hti: HTIndex] = CODE;

fillctxse: PROCEDURE [sei: ISEIndex, hti: HTIndex, public: BOOLEAN] =
  BEGIN
  psei: ISEIndex;
  ctx: CTXIndex = (seb+sei).ctxnum;
  (seb+sei).htptr ← hti;
  IF hti # HTNull
    THEN
      BEGIN
      IF ht[hti].anyInternal AND ctx # CTXNull
        THEN
          FOR psei ← (ctxb+ctx).selist, NextSe[psei] UNTIL psei = sei
            DO
            IF (seb+psei).htptr = hti THEN GO TO duplicate;
            REPEAT
              duplicate => SIGNAL NameClash[hti];
            ENDLOOP;
      ht[hti].anyInternal ← TRUE;
      IF public THEN ht[hti].anyPublic ← TRUE;
      END;
  RETURN
  END;

setselink: PROCEDURE [sei, next: ISEIndex] =
  BEGIN
  WITH (seb+sei) SELECT FROM
    linked => link ← next;
    ENDCASE => ERROR;
  RETURN
  END;


makenonctxse: PROCEDURE [size: CARDINAL] RETURNS [sei: CSEIndex] =
  BEGIN  -- makes a non-ctx se entry for a constructed type
```

```
    sei ← Allocate[setype, size];
    (seb+sei)↑ ← [mark3:FALSE, mark4:FALSE, sebody:constructor[typeinfo: ]];
    RETURN
    END;


  -- attribute testing

  ConstantId: PROCEDURE [sei: ISEIndex] RETURNS [BOOLEAN] =
    BEGIN
    RETURN [IF ~(seb+sei).constant
      THEN FALSE
      ELSE
        SELECT XferMode[(seb+sei).idtype] FROM
          procedure => (seb+sei).mark4 AND (seb+sei).idinfo = BTNull,
          signal, error =>
           (seb+sei).mark4 AND ConstantLink[(seb+sei).idvalue],
          ENDCASE => TRUE]
    END;

  ConstantLink: PROCEDURE [link: ControlDefs.ControlLink] RETURNS [BOOLEAN] =
    BEGIN
    RETURN [link.gfi = ControlDefs.GFTNull]
    END;

  END.
```