```
-- file SymbolTable.Mesa
-- last modified by Barbara, October 3, 1977  11:24 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  ControlDefs: FROM "controldefs",
  InlineDefs: FROM "inlinedefs",
  StringDefs: FROM "stringdefs",
  TableDefs: FROM "tabledefs",
  SymTabDefs: FROM "symtabdefs",
  SymDefs: FROM "symdefs";

SymbolTable: PROGRAM
    IMPORTS StringDefs
    EXPORTS SymTabDefs SHARES SymDefs =
 PUBLIC
  BEGIN
  OPEN SymDefs;

   -- tables defining the current symbol table
     hashvec: DESCRIPTOR FOR ARRAY OF HTIndex;    -- hash index
     ht: DESCRIPTOR FOR ARRAY --HTIndex-- OF HTRecord;    -- hash table
     ssb: STRING;                          -- id string
     seb: TableDefs.TableBase;             -- se table
     ctxb: TableDefs.TableBase;            -- context table
     mdb: TableDefs.TableBase;             -- module directory base
     bb: TableDefs.TableBase;              -- body table

     stHandle: POINTER TO STHeader;

   -- info defining the source file links
     sourcefile: STRING;
     fgt: DESCRIPTOR FOR ARRAY OF FGTEntry;

   ignorecases: BOOLEAN;

  -- the following procedure is called if the base values change
   notifier: PROCEDURE [POINTER TO FRAME[SymbolTable]];

   NullNotifier: PROCEDURE [POINTER TO FRAME[SymbolTable]] =
     BEGIN
     RETURN
     END;


   SubString: TYPE = StringDefs.SubString;

   hashvalue: PROCEDURE [s: SubString] RETURNS [HVIndex] =
     BEGIN  -- computes the hash index for string s
     CharBits: MACHINE CODE [CHARACTER, WORD] RETURNS [WORD] =
       LOOPHOLE[InlineDefs.BITAND];
     mask: WORD = 337B;          -- masks out ASCII case shifts
     n: CARDINAL = s.length;
     b: STRING = s.base;
     v: WORD;
     v ← CharBits[b[s.offset], mask]*177B + CharBits[b[s.offset+(n-1)], mask];
     RETURN [InlineDefs.BITXOR[v, n*17B] MOD LENGTH[hashvec]]
     END;

   FindString: PROCEDURE [s: SubString] RETURNS [found: BOOLEAN, hti: HTIndex] =
     BEGIN
     OPEN StringDefs;
     desc: SubStringDescriptor;
     ss: SubString = @desc;
     hti ← hashvec[hashvalue[s]];
     WHILE hti # HTNull
       DO
       SubStringForHash[ss, hti];
       found ←
        (IF ignorecases THEN EquivalentSubStrings ELSE EqualSubStrings)[s,ss];
       IF found THEN RETURN;
       hti ← ht[hti].link;
       ENDLOOP;
     RETURN [FALSE, HTNull]
     END;
```

```
SubStringForHash: PROCEDURE [s: SubString, hti: HTIndex] =
  BEGIN -- gets string for hash table entry
  s.base ← ssb;
  IF hti = HTNull
    THEN s.offset ← s.length ← 0
    ELSE
      BEGIN
      s.offset ← ht[hti].ssIndex;
      s.length ← ht[hti+1].ssIndex - s.offset;
      END;
  RETURN
  END;


hashforse: PROCEDURE [sei: ISEIndex] RETURNS [HTIndex] =
  BEGIN
  RETURN [(seb+sei).htptr]
  END;

searchcontext: PROCEDURE [hti: HTIndex, ctx: CTXIndex] RETURNS [BOOLEAN, ISEIndex] =
  BEGIN
  sei, root: ISEIndex;
  IF ctx # CTXNull AND hti # HTNull
    THEN
      BEGIN  sei ← root ← (ctxb+ctx).selist;
        DO
        IF sei = SENull THEN EXIT;
        IF (seb+sei).htptr = hti THEN  RETURN [TRUE, sei];
        WITH (seb+sei) SELECT FROM
          sequential =>  sei ← sei + SIZE[sequential id SERecord];
          linked =>  IF (sei ← link) = root THEN EXIT;
          ENDCASE => EXIT;
        ENDLOOP;
      END;
  RETURN [FALSE, ISENull]
  END;


-- information returning procedures

BytesPerWord: CARDINAL = AltoDefs.BytesPerWord;

symexternal: PROCEDURE [sei: ISEIndex] RETURNS [BOOLEAN] =
  BEGIN
  RETURN [(seb+sei).external]
  END;

symdefinition: PROCEDURE [sei: ISEIndex, onlypublic: BOOLEAN] RETURNS [BOOLEAN] =
  BEGIN
  RETURN [(seb+sei).ctxnum = stHandle.outerCtx
          AND (~onlypublic OR (seb+sei).public)
          AND (seb+sei).writeonce]
  END;

symtype: PROCEDURE [sei: ISEIndex] RETURNS [SEIndex] =
  BEGIN
  RETURN [(seb+sei).idtype]
  END;

symconst: PROCEDURE [sei: ISEIndex] RETURNS [BOOLEAN] =
  BEGIN
  RETURN [(seb+sei).constant]
  END;


symaddress: PROCEDURE [sei: ISEIndex] RETURNS [bitaddress] =
  BEGIN
  IF (seb+sei).constant THEN ERROR;
  RETURN [(seb+sei).idvalue]
  END;

symvalue: PROCEDURE [sei: ISEIndex] RETURNS [UNSPECIFIED] =
  BEGIN
  IF ~(seb+sei).constant THEN ERROR;
  RETURN [(seb+sei).idvalue]
  END;
```

```
symentry: PROCEDURE [sei: ISEIndex] RETURNS [CARDINAL] =
  BEGIN
  bti: BTIndex;
  IF ~(seb+sei).constant OR xfermode[(seb+sei).idtype] # procedure
    THEN ERROR;
  bti ← LOOPHOLE[(seb+sei).idvalue];
  RETURN [(bb+bti).entryindex]
  END;


transfertypes: PROCEDURE [type: SEIndex] RETURNS [typein, typeout: SEIndex] =
  BEGIN
  sei: CSEIndex ← undertype[type];
  typein ← typeout ← SENull;
  WITH (seb+sei) SELECT FROM
    transfer => RETURN [inrecord, outrecord];
    ENDCASE => NULL;
  RETURN
  END;

fieldcontext: PROCEDURE [type: SEIndex] RETURNS [CTXIndex] =
  BEGIN
  sei: CSEIndex;
  IF type = SENull THEN RETURN [CTXNull];
  sei ← undertype[type];
  RETURN [WITH (seb+sei) SELECT FROM
    record => fieldctx,
    ENDCASE => CTXNull]
  END;


BytePC: TYPE = ControlDefs.BytePC;

localcontext: PROCEDURE [pc: BytePC] RETURNS [CTXIndex] =
  BEGIN
  bti: BTIndex = PcToBti[pc];
  RETURN [IF bti = BTNull THEN CTXNull ELSE (bb+bti).localctx]
  END;

PcToBti: PROCEDURE [pc: BytePC] RETURNS [BTIndex] =
  BEGIN -- maps pc into body table index
  btlimit: BTIndex = LOOPHOLE[stHandle.bodySize];
  bti: BTIndex ← FIRST[BTIndex];
  IF pc = BytePC[0] THEN RETURN[bti];
  UNTIL bti = btlimit
    DO
    WITH body: (bb+bti).info SELECT FROM
      external =>
        IF pc IN [body.origin..body.origin+body.bytes) THEN RETURN [bti];
      ENDCASE => ERROR;
    bti ← bti + (WITH (bb+bti) SELECT FROM
                   inner => SIZE[inner BodyRecord],
                   ENDCASE => SIZE[outer BodyRecord]);
    ENDLOOP;
  RETURN [BTNull]
  END;


-- path following procedures

firstctxse: PROCEDURE [ctx: CTXIndex] RETURNS [ISEIndex] =
  BEGIN
  RETURN [IF ctx = CTXNull
    THEN ISENull
    ELSE (ctxb+ctx).selist]
  END;

nextse: PROCEDURE [sei: ISEIndex] RETURNS [ISEIndex] =
  BEGIN
  RETURN [
    IF sei = SENull
      THEN ISENull
      ELSE
        WITH (seb+sei) SELECT FROM
```

```
            terminal => ISENull,
            sequential => sei + SIZE[sequential id SERecord],
            linked => link,
            ENDCASE => ISENull]
    END;

  ctxentries: 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  n ← n+1  ENDLOOP;
    RETURN
    END;

  -- type manipulation

  undertype: PROCEDURE [type: SEIndex] RETURNS [CSEIndex] =
    BEGIN  -- strips off type identifiers
    sei: SEIndex ← type;
    WHILE sei # SENull
      DO
      WITH (seb+sei) SELECT FROM
        id =>
          BEGIN
          IF idtype # typeTYPE THEN ERROR;
          sei ← idinfo;
          END;
        ENDCASE =>  EXIT;
      ENDLOOP;
    RETURN [LOOPHOLE[sei, CSEIndex]]
    END;

  typeclass: PROCEDURE [type: SEIndex] RETURNS [TypeClass] =
    BEGIN
    RETURN [(seb+undertype[type]).typetag]
    END;

  xfermode: PROCEDURE [type: SEIndex] RETURNS [TransferMode] =
    BEGIN
    sei: CSEIndex = undertype[type];
    RETURN[WITH (seb+sei) SELECT FROM
      transfer => mode,
      ENDCASE => none]
    END;

  wordsfortype: PROCEDURE [type: SEIndex] RETURNS [CARDINAL] =
    BEGIN
    sei: CSEIndex = undertype[type];
    wordlength: CARDINAL = AltoDefs.wordlength;
    RETURN [IF sei = SENull
      THEN 0
      ELSE
        WITH (seb+sei) SELECT FROM
          mode => 1,    -- temporary fudge for P4binding
          basic => (length + (wordlength-1))/wordlength,
          enumerated => 1,
          record => (length + (wordlength-1))/wordlength,
          pointer => 1,
          array => IF packed
                THEN (cardinality[indextype] + (BytesPerWord-1))/BytesPerWord
                ELSE cardinality[indextype]*wordsfortype[componenttype],
          arraydesc => 2,
          transfer => IF mode = port THEN 2 ELSE 1,
          subrange => IF empty OR flexible THEN 0 ELSE 1,
          ENDCASE => 0]
    END;

  cardinality: PROCEDURE [type: SEIndex] RETURNS [CARDINAL] =
    BEGIN
    sei: CSEIndex = undertype[type];
    RETURN [WITH (seb+sei) SELECT FROM
```

```
      enumerated => nvalues,
      subrange => IF empty OR flexible THEN 0 ELSE range+1,
      basic => IF code = codeCHARACTER THEN AltoDefs.maxcharcode+1 ELSE 0,
      ENDCASE => 0]
    END;

  END.
```