```
-- file SymbolTable.Mesa
-- last modified by Sandman, March 23, 1978  10:25 AM

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

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

  SymbolTableBase: TYPE = POINTER TO FRAME[SymbolTable];

  link: SymbolTableBase;
  cacheInfo: POINTER;

  -- 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
    tb: TableDefs.TableBase;            -- tree area
    ltb: TableDefs.TableBase;           -- literal area
    extb: TableDefs.TableBase;          -- extension map

    stHandle: POINTER TO STHeader;

  -- info defining the source file links
    sourceFile: STRING;
    fgTable: DESCRIPTOR FOR ARRAY OF FGTEntry;

 -- the following procedure is called if the base values change
  notifier: PROCEDURE [SymbolTableBase];

  NullNotifier: PROCEDURE [SymbolTableBase] =
    BEGIN
    RETURN
    END;


 -- hash manipulation

  SubString: TYPE = StringDefs.SubString;

  FindString: PROCEDURE [s: SubString] RETURNS [hti: HTIndex] =
    BEGIN
    desc: StringDefs.SubStringDescriptor;
    ss: SubString = @desc;
    hti ← hashVec[HashValue[s]];
    WHILE hti # HTNull
      DO
      SubStringForHash[ss, hti];
      IF StringDefs.EqualSubStrings[s,ss] THEN EXIT;
      hti ← ht[hti].link;
      ENDLOOP;
    RETURN
    END;

  HashValue: PROCEDURE [s: SubString] RETURNS [HVIndex] =
    BEGIN  -- computes the hash index for string s
    CharBits: PROCEDURE [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;

  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 s.length ← ht[hti].ssIndex - (s.offset ← ht[hti-1].ssIndex);
    RETURN
    END;


-- context management

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

  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 id: (seb+sei) SELECT FROM
            terminal => ISENull,
            sequential => sei + SIZE[sequential id SERecord],
            linked => id.link,
            ENDCASE => ISENull]
    END;

  SearchContext: PROCEDURE [hti: HTIndex, ctx: CTXIndex] RETURNS [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 [sei];
          WITH id: (seb+sei) SELECT FROM
            sequential => sei ← sei + SIZE[sequential id SERecord];
            linked => IF (sei ← id.link) = root THEN EXIT;
            ENDCASE => EXIT;
          ENDLOOP;
        END;
    RETURN [ISENull]
    END;


-- type manipulation

  NormalType: PROCEDURE [type: CSEIndex] RETURNS [nType: CSEIndex] =
    BEGIN
    nType ← type;
      DO
      WITH (seb+nType) SELECT FROM
        subrange => nType ← UnderType[rangetype];
        long, real => nType ← UnderType[rangetype];
        ENDCASE => EXIT;
```

```
    ENDLOOP;
  RETURN [nType]
  END;

RecordLink: PROCEDURE [type: recordCSEIndex] RETURNS [recordCSEIndex] =
  BEGIN
  RETURN [WITH t: (seb+type) SELECT FROM
    linked => LOOPHOLE[UnderType[t.linktype], recordCSEIndex],
    ENDCASE => recordCSENull]
  END;

RecordRoot: PROCEDURE [type: recordCSEIndex] RETURNS [root: recordCSEIndex] =
  BEGIN
    DO
    root ← type;
    IF (type ← RecordLink[root]) = SENull THEN EXIT;
    ENDLOOP;
  RETURN
  END;

TransferTypes: PROCEDURE [type: SEIndex]
    RETURNS [typeIn, typeOut: recordCSEIndex] =
  BEGIN
  sei: CSEIndex = UnderType[type];
  WITH t: (seb+sei) SELECT FROM
    transfer => BEGIN  typeIn ← t.inrecord;  typeOut ← t.outrecord  END;
    ENDCASE => typeIn ← typeOut ← recordCSENull;
  RETURN
  END;

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

TypeLink: PROCEDURE [type: SEIndex] RETURNS [SEIndex] =
  BEGIN
  sei: CSEIndex = UnderType[type];
  RETURN [WITH se: (seb+sei) SELECT FROM
    record =>
      WITH se SELECT FROM
        linked => linktype,
        ENDCASE => SENull,
    ENDCASE => SENull]
  END;

TypeRoot: PROCEDURE [type: SEIndex] RETURNS [root: CSEIndex] =
  BEGIN
  link: SEIndex;
  link ← type;
    DO
    root ← UnderType[link];
    IF (link ← TypeLink[root]) = SENull THEN EXIT;
    ENDLOOP;
  RETURN
  END;

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

XferMode: PROCEDURE [type: SEIndex] RETURNS [TransferMode] =
  BEGIN
  sei: CSEIndex = UnderType[type];
```

```
    RETURN[WITH t: (seb+sei) SELECT FROM
      transfer => t.mode,
      ENDCASE => none]
    END;


  -- information returning procedures

  WordLength: CARDINAL = AltoDefs.wordlength;
  WordFill: CARDINAL = WordLength-1;
  ByteLength: CARDINAL = AltoDefs.charlength;
  BytesPerWord: CARDINAL = AltoDefs.BytesPerWord;

  BitsForType: PROCEDURE [type: SEIndex] RETURNS [CARDINAL] =
    BEGIN
    sei: CSEIndex = UnderType[type];
    RETURN [IF sei = SENull
      THEN 0
      ELSE
        WITH t: (seb+sei) SELECT FROM
          basic => t.length,
          enumerated => BitsForRange[Cardinality[sei]-1],
          record => t.length,
          subrange => IF t.empty THEN 0 ELSE BitsForRange[Cardinality[sei]-1],
          ENDCASE => WordsForType[sei]*WordLength]
    END;

  BitsForRange: PROCEDURE [maxValue: CARDINAL] RETURNS [nBits: CARDINAL] =
    BEGIN
    fieldMax: CARDINAL;
    nBits ← 1;  fieldMax ← 1;
    WHILE nBits < WordLength AND fieldMax < maxValue
      DO  nBits ← nBits + 1;  fieldMax ← 2*fieldMax + 1  ENDLOOP;
    RETURN
    END;

  Cardinality: PROCEDURE [type: SEIndex] RETURNS [CARDINAL] =
    BEGIN
    sei: CSEIndex = UnderType[type];
    RETURN [WITH t: (seb+sei) SELECT FROM
      enumerated => t.nvalues,
      subrange => IF t.empty OR t.flexible THEN 0 ELSE t.range+1,
      basic => IF t.code = codeCHARACTER THEN AltoDefs.maxcharcode+1 ELSE 0,
      ENDCASE => 0]
    END;

  FnField: PROCEDURE [field: ISEIndex] RETURNS [offset: BitAddress, size: CARDINAL] =
    BEGIN
    sei: ISEIndex;
    word: CARDINAL;
    sei ← (ctxb+(seb+field).ctxnum).selist;  word ← 0;
      DO
      size ← IF (seb+sei).constant THEN 0 ELSE WordsForType[(seb+sei).idtype];
      IF sei = field THEN EXIT;
      word ← word + size;  sei ← NextSe[sei];
      ENDLOOP;
    offset ← BitAddress[wd:word, bd:0];
    size ← size * WordLength;
    RETURN
    END;

  WordsForType: PROCEDURE [type: SEIndex] RETURNS [CARDINAL] =
    BEGIN
    sei: CSEIndex = UnderType[type];
    b: CARDINAL;
    RETURN [IF sei = SENull
      THEN 0
      ELSE
        WITH t: (seb+sei) SELECT FROM
          mode => 1,    -- fudge for compiler (Pass4:Binding)
          basic => (t.length + (WordLength-1))/WordLength,
          enumerated => 1,
          record => (t.length + (WordLength-1))/WordLength,
          pointer => 1,
          array =>
            IF (b←BitsForType[t.componenttype]) <= ByteLength AND t.packed
```

```
            THEN (Cardinality[t.indextype] + (BytesPerWord-1))/BytesPerWord
            ELSE Cardinality[t.indextype] * ((b+WordFill)/WordLength),
        arraydesc => 2,
        transfer => IF t.mode = port THEN 2 ELSE 1,
        relative => WordsForType[t.offsetType],
        subrange => IF t.empty OR t.flexible THEN 0 ELSE 1,
        long => WordsForType[t.rangetype] + 1,
        real => 2,
        ENDCASE => 0]
  END;

END.
```