```
-- file Misc.Mesa
-- last modified by Satterthwaite, April 27, 1978  11:39 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  LitDefs: FROM "litdefs",
  StringDefs: FROM "stringdefs",
  SymDefs: FROM "symdefs",
  SymSegDefs: FROM "symsegdefs",
  TableDefs: FROM "tabledefs",
  TreeDefs: FROM "treedefs";

Misc: PROGRAM
    IMPORTS StringDefs, TableDefs
    EXPORTS LitDefs, SymSegDefs SHARES LitDefs =
 PUBLIC
  BEGIN
  OPEN TableDefs, SymSegDefs, LitDefs;

  ltb: PRIVATE TableBase;       -- literal table base
  stb: PRIVATE TableBase;       -- string table base
  seb: PRIVATE TableBase;       -- se table base
  extb: PRIVATE TableBase;      -- extension table base

  UpdateBases: PRIVATE TableNotifier =
    BEGIN  -- called whenever the main symbol table is repacked
    ltb ← base[lttype];  stb ← base[sttype];
    seb ← base[SymDefs.setype];  extb ← base[SymSegDefs.exttype];  RETURN
    END;


  tableOpen: PRIVATE BOOLEAN ← FALSE;

  LitTabInit: PROCEDURE =
    BEGIN  -- called to set up the compiler's literal table
    shvi: SLitHVIndex;
    IF tableOpen THEN LitTabErase[];
    [] ← ForgetLiterals[];
    FOR shvi IN SLitHVIndex DO sHashVec[shvi] ← MSTNull ENDLOOP;
    stLimit ← localStart ← FIRST[STIndex];  locals ← markBit ← FALSE;
    AddNotify[UpdateBases];
    tableOpen ← TRUE;  RETURN
    END;

  LitTabErase: PROCEDURE =
    BEGIN  --  closes the symbol table blocks
    tableOpen ← FALSE;  DropNotify[UpdateBases];
    RETURN
    END;


 -- literal table management

  LitHVLength: PRIVATE INTEGER = 53;
  LitHVIndex: PRIVATE TYPE = [0..LitHVLength);

  hashVec: PRIVATE ARRAY LitHVIndex OF LTIndex;


  FindLiteral: PROCEDURE [v: WORD] RETURNS [lti: LTIndex] =
    BEGIN
    hvi: LitHVIndex = v MOD LitHVLength;
    FOR lti ← hashVec[hvi], (ltb+lti).link UNTIL lti = LTNull
      DO
      WITH entry: (ltb+lti) SELECT FROM
        short => IF entry.value = v THEN EXIT;
        ENDCASE;
      REPEAT
        FINISHED =>
          BEGIN
          lti ← Allocate[lttype, SIZE[short LTRecord]];
          (ltb+lti)↑ ← LTRecord[datum: short[value: v], link: hashVec[hvi]];
          hashVec[hvi] ← lti;
          END;
      ENDLOOP;
    RETURN
```

```
    END;

  FindMultiWord: PRIVATE PROCEDURE [baseP: TableFinger, desc: LitDescriptor]
      RETURNS [lti: LTIndex] =
    BEGIN
    i: CARDINAL;
    v: WORD;
    hvi: LitHVIndex;
    lLti: POINTER [0..TableLimit/2) TO long LTRecord;
    v ← 0;
    FOR i IN [0 .. desc.length) DO v ← v + (baseP↑ + desc.offset)[i] ENDLOOP;
    hvi ← v MOD LitHVLength;
    FOR lti ← hashVec[hvi], (ltb+lti).link UNTIL lti = LTNull
      DO
      WITH entry: (ltb+lti) SELECT FROM
        long =>
          IF desc.length = entry.length THEN
            FOR i IN [0 .. desc.length)
              DO
              IF entry.value[i] # (baseP↑ + desc.offset)[i] THEN EXIT;
              REPEAT
                FINISHED => GO TO found;
              ENDLOOP;
        ENDCASE;
      REPEAT
        found =>  NULL;
        FINISHED =>
          BEGIN
          lLti ← Allocate[lttype, SIZE[long LTRecord] + desc.length];
          (ltb+lLti)↑ ← LTRecord[
                link: hashVec[hvi],
                datum: long[codeIndex: 0, length: desc.length, value: ]];
          FOR i IN [0 .. desc.length)
            DO  (ltb+lLti).value[i] ← (baseP↑ + desc.offset)[i]  ENDLOOP;
          hashVec[hvi] ← lti ← lLti;
          END;
      ENDLOOP;
    RETURN
    END;

  LiteralValue: PROCEDURE [lti: LTIndex] RETURNS [WORD] =
    BEGIN
    WITH entry: (ltb+lti) SELECT FROM
      short =>  RETURN [entry.value];
      long =>  IF entry.length = 1 THEN RETURN [entry.value[0]];
      ENDCASE;
    ERROR
    END;


  FindLitDescriptor: PROCEDURE [desc: DESCRIPTOR FOR ARRAY OF WORD] RETURNS [LTIndex] =
    BEGIN
    base: TableBase ← 0;
    RETURN [IF LENGTH[desc] = 1
      THEN FindLiteral[desc[0]]
      ELSE FindMultiWord[@base, [offset:LOOPHOLE[BASE[desc]], length:LENGTH[desc]]]]
    END;

  LitDescriptorValue: PROCEDURE [lti: LTIndex] RETURNS [desc: LitDescriptor] =
    BEGIN
    WITH entry: (ltb+lti) SELECT FROM
      short => desc ← [offset: LOOPHOLE[@entry.value-ltb], length: 1];
      long => desc ← [offset: @entry.value-ltb, length: entry.length];
      ENDCASE => ERROR;
    RETURN
    END;


  CopyLiteral: PROCEDURE [literal: LiteralId] RETURNS [lti: LTIndex] =
    BEGIN
    desc: LitDescriptor;
    WITH entry: (literal.baseP↑ + literal.index) SELECT FROM
      short =>  lti ← FindLiteral[entry.value];
      long =>
        BEGIN
        desc ← [offset: @entry.value - literal.baseP↑, length: entry.length];
```

```
      lti ← FindMultiWord[literal.baseP, desc];
      END;
    ENDCASE => ERROR
  END;

ForgetLiterals: PROCEDURE RETURNS [currentSize: CARDINAL] =
  BEGIN
  hvi: LitHVIndex;
  FOR hvi IN LitHVIndex DO hashVec[hvi] ← LTNull ENDLOOP;
  RETURN [TableDefs.TableBounds[lttype].size]
  END;


-- string literal table management

MSTNull: PRIVATE MSTIndex = LOOPHOLE[STNull];
SLitHVLength: PRIVATE INTEGER = 23;
SLitHVIndex: PRIVATE TYPE = [0..SLitHVLength);

sHashVec: PRIVATE ARRAY SLitHVIndex OF MSTIndex;

stLimit, localStart: STIndex;
locals: BOOLEAN;
markBit: BOOLEAN;


FindStringLiteral: PROCEDURE [s: StringDefs.SubString] RETURNS [STIndex] =
  BEGIN  OPEN StringDefs;
  CpW: CARDINAL = AltoDefs.CharsPerWord;
  hash: WORD;
  hvi: SLitHVIndex;
  i, nw: CARDINAL;
  sti: MSTIndex;
  v: STRING;
  desc: StringDefs.SubStringDescriptor;
  hash ← 0;
  FOR i IN [s.offset .. s.offset+s.length)
    DO  hash ← hash + LOOPHOLE[s.base[i], CARDINAL]  ENDLOOP;
  hvi ← hash MOD SLitHVLength;
  FOR sti ← sHashVec[hvi], (stb+sti).link UNTIL sti = MSTNull
    DO
    v ← StringLiteralValue[sti];
    desc ← SubStringDescriptor[base:v, offset:0, length:v.length];
    IF EqualSubStrings[s, @desc] THEN EXIT;
    REPEAT
      FINISHED =>
        BEGIN
        nw ← WordsForString[s.length];
        sti ← Allocate[sttype, SizeSTPrefix + nw];
        (stb+sti)↑ ← STRecord[master[
            info: 0,
            codeIndex: 0,
            local: FALSE,
            link: sHashVec[hvi],
            string: [
              length: 0,
              maxlength: ((s.length + (CpW-1))/CpW) * CpW,
              text: ]]];
        AppendSubString[@(stb+sti).string, s];
        FOR i IN [s.length .. (stb+sti).string.maxlength)
          DO  AppendChar[@(stb+sti).string, 0C]  ENDLOOP;
        (stb+sti).string.length ← s.length;
        stLimit ← stLimit + (SizeSTPrefix + nw);
        sHashVec[hvi] ← sti;
        END;
    ENDLOOP;
  RETURN [sti]
  END;


MasterString: PROCEDURE [sti: STIndex] RETURNS [MSTIndex] =
  BEGIN
  RETURN [WITH s: (stb+sti) SELECT FROM
    master => LOOPHOLE[sti],
    copy => s.link,
    ENDCASE => MSTNull]
```

```
    END;

  StringLiteralReference: PROCEDURE [sti: STIndex] =
    BEGIN
    WITH s: (stb+sti) SELECT FROM
      master => s.info ← s.info + 1;
      ENDCASE => NULL;
    RETURN
    END;

  StringLiteralValue: PROCEDURE [sti: STIndex] RETURNS [STRING] =
    BEGIN
    RETURN[@(stb+MasterString[sti]).string]
    END;

  ResetLocalStrings: PROCEDURE RETURNS [key: STIndex] =
    BEGIN
    IF ~locals
      THEN key ← STNull
      ELSE BEGIN key ← localStart; markBit ← ~markBit END;
    locals ← FALSE; localStart ← LOOPHOLE[TableBounds[sttype].size];
    RETURN
    END;

  FindLocalStringLiteral: PROCEDURE [key: STIndex] RETURNS [sti: STIndex] =
    BEGIN
    next: STIndex;
    master: MSTIndex = MasterString[key];
    FOR sti ← localStart, next UNTIL sti = stLimit
      DO
      WITH s: (stb+sti) SELECT FROM
        master =>
          next ← sti + SizeSTPrefix + StringDefs.WordsForString[s.string.maxlength];
        copy =>
          BEGIN
          IF s.link = master THEN EXIT;
          next ← sti + SIZE[copy STRecord];
          END;
        ENDCASE;
      REPEAT
        FINISHED =>
          BEGIN
          sti ← Allocate[sttype, SIZE[copy STRecord]];
          (stb+sti)↑ ← STRecord[copy[mark: markBit, link: master]];
          stLimit ← stLimit + SIZE[copy STRecord];
          locals ← TRUE;
          END;
      ENDLOOP;
    RETURN
    END;


  EnumerateLocalStrings: PROCEDURE [key: STIndex, proc: PROCEDURE [MSTIndex]] =
    BEGIN
    sti, next: STIndex;
    started, mark: BOOLEAN;
    IF key = STNull THEN RETURN;
    started ← FALSE;
    FOR sti ← key, next UNTIL sti = stLimit
      DO
      WITH s: (stb+sti) SELECT FROM
        master =>
          next ← sti + SizeSTPrefix + StringDefs.WordsForString[s.string.maxlength];
        copy =>
          BEGIN
          IF ~started THEN BEGIN mark ← s.mark; started ← TRUE END;
          IF s.mark # mark THEN EXIT;
          proc[s.link];
          next ← sti + SIZE[copy STRecord];
          END;
        ENDCASE => ERROR;
      ENDLOOP;
    END;

  EnumerateMasterStrings: PROCEDURE [proc: PROCEDURE [MSTIndex]] =
    BEGIN
```

```
    sti, next: STIndex;
    FOR sti ← FIRST[STIndex], next UNTIL sti = stLimit
      DO
      WITH s: (stb+sti) SELECT FROM
        master =>
          BEGIN
          proc[LOOPHOLE[sti]];
          next ← sti + SizeSTPrefix + StringDefs.WordsForString[s.string.maxlength];
          END;
        copy =>  next ← sti + SIZE[copy STRecord];
        ENDCASE =>  ERROR;
      ENDLOOP;
    RETURN
    END;


  -- extension table management

  ISEIndex: TYPE = SymDefs.ISEIndex;
  TreeLink: TYPE = TreeDefs.TreeLink;

  EnterExtension: PROCEDURE [sei: ISEIndex, t: TreeLink] =
    BEGIN
    exti: ExtIndex;
    exti ← Allocate[exttype, SIZE[ExtRecord]];
    (extb+exti)↑ ← ExtRecord[sei: sei, tree: t];
    (seb+sei).extended ← TRUE;
    RETURN
    END;

  FindExtension: PROCEDURE [sei: ISEIndex] RETURNS [TreeLink] =
    BEGIN
    exti: ExtIndex;
    extLimit: ExtIndex = LOOPHOLE[TableBounds[exttype].size];
    FOR exti ← FIRST[ExtIndex], exti + SIZE[ExtRecord] UNTIL exti = extLimit
      DO
      IF (extb+exti).sei = sei THEN RETURN [(extb+exti).tree];
      ENDLOOP;
    ERROR
    END;

  END.
```