```
-- file DILiterals.Mesa
-- last modified by
--                  Sandman, April 9, 1978  1:16 AM
--                  Barbara, June 21, 1978  3:03 PM

DIRECTORY
  DILitDefs: FROM "dilitdefs" USING [
    HalfPage, LTIndex, LTNull, LTRecord, STIndex, STNull, STRecord],
  ForgotDefs: FROM "forgotdefs" USING [LowHalf],
  StringDefs: FROM "stringdefs" USING [
    EqualSubStrings, SubString, SubStringDescriptor],
  SystemDefs: FROM "systemdefs" USING [AllocateResidentPages, FreePages];

DILiterals: PROGRAM
  IMPORTS StringDefs, SystemDefs
  EXPORTS DILitDefs
  SHARES DILitDefs = PUBLIC

  BEGIN OPEN DILitDefs;

  tableopen: PRIVATE BOOLEAN ← FALSE;

  LitTabInit: PROCEDURE =
    BEGIN  -- called to set up the compiler's literal table
    lti: LTIndex;
    slti: STIndex;
    p: POINTER;
    IF tableopen THEN LitTabErase[];
    p ← SystemDefs.AllocateResidentPages[1];
    litTable ← LOOPHOLE[DESCRIPTOR[p, LAST[LTIndex], LTRecord]];
    slitTable ← LOOPHOLE[DESCRIPTOR[p+HalfPage, LAST[STIndex], STRecord]];
    FOR lti IN LTIndex DO
      litTable[lti] ← LTRecord[free: TRUE, link: LTNull, datum: ];
      ENDLOOP;
    FOR slti IN STIndex DO
      slitTable[slti] ← STRecord[free: TRUE, link: STNull, string: NullDesc];
      ENDLOOP;
    tableopen ← TRUE;
    free ← LAST[LTIndex];
    freesl ← LAST[STIndex];
    RETURN
    END;

  LitTabErase: PROCEDURE =
    BEGIN  --  closes the symbol table blocks
    tableopen ← FALSE;
    SystemDefs.FreePages[BASE[litTable]];
    RETURN
    END;

 -- literal table management

  litTable: PRIVATE DESCRIPTOR FOR ARRAY LTIndex OF LTRecord;

  free: PRIVATE LTIndex ← LAST[LTIndex]; -- for finding free slot

  TooManyLiterals: SIGNAL = CODE;

  FindLiteral: PROCEDURE [v: WORD] RETURNS [lti: LTIndex] =
    BEGIN
    lti ← hash[v];
    IF litTable[lti].free THEN
      BEGIN
      litTable[lti] ← LTRecord[free: FALSE, link: LTNull, datum: short[value: v, unused:]];
      RETURN
      END;
    FOR lti ← lti, litTable[lti].link DO
      WITH l:litTable[lti] SELECT FROM
        short => IF l.value = v THEN RETURN;
        ENDCASE;
      IF litTable[lti].link = LTNull THEN EXIT;
      ENDLOOP;
    UNTIL free = LTNull DO
      IF litTable[free].free THEN
        BEGIN
        litTable[lti].link ← free;
```

```
        litTable[free] ← [free: FALSE, link: LTNull, datum: short[value: v, unused:]];
        lti ← free;
        free ← free - 1;
        RETURN[lti];
        END
      ELSE free ← free - 1;
      ENDLOOP;
    ERROR TooManyLiterals;
    END;

  FindLongLiteral: PROCEDURE [v: LONG INTEGER] RETURNS [lti: LTIndex] =
    BEGIN
    lti ← hash[ForgotDefs.LowHalf[v]];
    IF litTable[lti].free THEN
      BEGIN
      litTable[lti] ← LTRecord[free: FALSE, link: LTNull, datum: long[value: v]];
      RETURN
      END;
    FOR lti ← lti, litTable[lti].link DO
      WITH l:litTable[lti] SELECT FROM
        long => IF l.value = v THEN RETURN;
        ENDCASE;
      IF litTable[lti].link = LTNull THEN EXIT;
      ENDLOOP;
    UNTIL free = LTNull DO
      IF litTable[free].free THEN
        BEGIN
        litTable[lti].link ← free;
        litTable[free] ← [free: FALSE, link: LTNull, datum: long[value: v]];
        lti ← free;
        free ← free - 1;
        RETURN[lti];
        END
      ELSE free ← free - 1;
      ENDLOOP;
    ERROR TooManyLiterals;
    END;

  LiteralValue: PROCEDURE [lti: LTIndex] RETURNS [WORD] =
    BEGIN
    WITH l:litTable[lti] SELECT FROM
      short => RETURN[l.value];
      ENDCASE => ERROR;
    END;

  LongLiteralValue: PROCEDURE [lti: LTIndex] RETURNS [LONG INTEGER] =
    BEGIN
    WITH l:litTable[lti] SELECT FROM
      long => RETURN[l.value];
      ENDCASE => ERROR;
    END;

  hash: PRIVATE PROCEDURE [value: WORD] RETURNS [LTIndex] =
    BEGIN
    RETURN[(value MOD LAST[LTIndex]) + 1];
    END;

  -- string literal table management

  slitTable: PRIVATE DESCRIPTOR FOR ARRAY STIndex OF STRecord;

  freesl: PRIVATE STIndex ← LAST[STIndex]; -- for finding free slot

  TooManyStringLiterals: SIGNAL = CODE;

  FindStringLiteral: PROCEDURE [s: StringDefs.SubString] RETURNS [slti: STIndex] =
    BEGIN
    slti ← shash[s];
    IF slitTable[slti].free THEN
      BEGIN
      slitTable[slti] ← STRecord[free: FALSE, link: LTNull, string: s↑];
      RETURN
      END;
    FOR slti ← slti, slitTable[slti].link DO
      IF StringDefs.EqualSubStrings[@slitTable[slti].string, s] THEN RETURN;
      IF slitTable[slti].link = STNull THEN EXIT;
```

```
      ENDLOOP;
    UNTIL freesl = STNull DO
      IF slitTable[freesl].free THEN
        BEGIN
        slitTable[slti].link ← freesl;
        slitTable[freesl] ← STRecord[free: FALSE, link: STNull, string: s↑];
        slti ← freesl;
        freesl ← freesl - 1;
        RETURN[slti];
        END
      ELSE freesl ← freesl - 1;
      ENDLOOP;
    ERROR TooManyStringLiterals;
    END;

  StringLiteralValue: PROCEDURE [slti: STIndex] RETURNS [StringDefs.SubString] =
    BEGIN
    RETURN[@slitTable[slti].string]
    END;

  shash: PRIVATE PROCEDURE [s: StringDefs.SubString] RETURNS [STIndex] =
    BEGIN
    hash, i: CARDINAL;
    hash ← 0;
    FOR i IN [s.offset .. s.offset+s.length)
      DO  hash ← hash + LOOPHOLE[s.base[i], CARDINAL]  ENDLOOP;
    RETURN[(hash MOD LAST[STIndex]) + 1]
    END;

  NullDesc: StringDefs.SubStringDescriptor = [base: " ", offset: 0, length: 0];

  END.
```