```
-- file Scanner.Mesa
-- last modified by Satterthwaite, June 16, 1978  2:56 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [CharsPerWord, maxword, PageSize],
  IODefs: FROM "iodefs" USING [CR, TAB, WriteChar, WriteNumber, WriteString],
  LALRDefs: FROM "lalrdefs"
    USING [
      LALRTable, Symbol, SymbolRecord, VocabHashEntry,
      endmarker, hashval,
      tokenARROW, tokenCHAR, tokenDOT, tokenDOTS, tokenEQUAL,
      tokenGE, tokenGREATER, tokenID, tokenLE, tokenLESS,
      tokenLNUM, tokenLSTR, tokenMINUS, tokenNUM, tokenSTR],
  LitDefs: FROM "litdefs"
    USING [FindLitDescriptor, FindLiteral, FindStringLiteral],
  P1Defs: FROM "p1defs",
  StreamDefs: FROM "streamdefs"
    USING [
      StreamHandle, StreamIndex,
      GetIndex, ModifyIndex, NormalizeIndex, ReadBlock, SetIndex,
      StreamError],
  StringDefs: FROM "stringdefs" USING [SubStringDescriptor, AppendString],
  SymTabDefs: FROM "symtabdefs" USING [EnterString],
  SystemDefs: FROM "systemdefs"
    USING [
      AllocateHeapString, AllocatePages, FreeHeapString, FreePages,
      PruneHeap];

Scanner: PROGRAM
    IMPORTS IODefs, LitDefs, StreamDefs, StringDefs, SymTabDefs, SystemDefs
    EXPORTS P1Defs SHARES LALRDefs =
  BEGIN
  OPEN LALRDefs;

  dHashTab: DESCRIPTOR FOR ARRAY OF VocabHashEntry;
  dScanTab: DESCRIPTOR FOR ARRAY CHARACTER [40C..177C] OF Symbol;
  vocab: STRING;
  dVocabIndex: DESCRIPTOR FOR ARRAY OF CARDINAL;

  NUL: CHARACTER = 0C;
  CR: CHARACTER = IODefs.CR;
  Control1Z: CHARACTER = 32C;            -- Bravo escape char

  stream: StreamDefs.StreamHandle;       -- the input stream
  streamOrigin: StreamDefs.StreamIndex;

  TextPages: CARDINAL = 6;
  TextWords: CARDINAL = TextPages*AltoDefs.PageSize;
  TextChars: CARDINAL = TextWords*AltoDefs.CharsPerWord;

  tB: POINTER TO PACKED ARRAY [0..TextChars) OF CHARACTER;
  tI, tMax: [0..TextChars];
  tOrigin, tLimit: CARDINAL;
  tEnded: BOOLEAN;

  FillTextBuffer: PROCEDURE =
    BEGIN
    words: [0..TextWords];
    bytes: [0..AltoDefs.CharsPerWord);
    tOrigin ← tLimit;
    IF tEnded
      THEN  tMax ← 0
      ELSE
        BEGIN
        words ← StreamDefs.ReadBlock[stream, tB, TextWords];
        bytes ← StreamDefs.GetIndex[stream].byte MOD AltoDefs.CharsPerWord;
        IF bytes # 0 THEN  words ← words-1;
        tMax ← words*AltoDefs.CharsPerWord + bytes;
        IF tMax < TextChars THEN  tEnded ← TRUE;
        tLimit ← tOrigin + tMax;
        END;
    IF tMax = 0 THEN  BEGIN  tB[0] ← NUL;  tMax ← 1  END;
    tI ← 0;  RETURN
    END;
```

```
  buffer: STRING ← NIL;                    -- token assembly area
  iMax: CARDINAL;                          -- iMax = buffer.maxlength
  desc: StringDefs.SubStringDescriptor; -- initial buffer segment

  nErrors: CARDINAL;                       -- lexical errors

  BufferOverflow: ERROR = CODE;

  ExpandBuffer: PROCEDURE =
    BEGIN
    oldBuffer: STRING ← buffer;
    IF oldBuffer.length > 2000 THEN ERROR BufferOverflow;
    buffer ← SystemDefs.AllocateHeapString[2*oldBuffer.length];
    StringDefs.AppendString[buffer, oldBuffer];
    iMax ← buffer.length ← buffer.maxlength;
    SystemDefs.FreeHeapString[oldBuffer];
    desc.base ← buffer;
    RETURN
    END;


  char: CHARACTER;       -- current (most recently scanned) character

  NextChar: PROCEDURE = -- also expanded inline within Atom
    BEGIN
    IF (tI←tI+1) = tMax THEN FillTextBuffer[];
    char ← tB[tI];
    RETURN
    END;


  Atom: PUBLIC PROCEDURE RETURNS [symbol: SymbolRecord] =
    BEGIN
    OPEN symbol;
      DO
      WHILE char IN [NUL..' ]
        DO
        SELECT char FROM
          ControlZ =>
            UNTIL char = CR
              DO
              IF (tI←tI+1) = tMax THEN
                BEGIN IF tEnded THEN GO TO EndFile; FillTextBuffer[] END;
              char ← tB[tI];
              ENDLOOP;
          ENDCASE;
        IF (tI←tI+1) = tMax THEN
          BEGIN IF tEnded THEN GO TO EndFile; FillTextBuffer[] END;
        char ← tB[tI];
        ENDLOOP;
      index ← tOrigin + tI;  value ← 0;
      SELECT char FROM

        'a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm,
        'n, 'o, 'p, 'q, 'r, 's, 't, 'u, 'v, 'w, 'x, 'y, 'z  =>
          BEGIN
          i: CARDINAL;
          i ← 0;
            DO
            buffer[i] ← char;
            IF (tI←tI+1) = tMax THEN FillTextBuffer[];
            char ← tB[tI];
            SELECT char FROM
              IN ['a..'z], IN ['A..'Z], IN ['0..'9] =>
                IF (i ← i+1) >= iMax THEN ExpandBuffer[];
              ENDCASE =>  EXIT;
            ENDLOOP;
          desc.length ← i+1;
          class ← tokenID;  value ← SymTabDefs.EnterString[@desc];
          GO TO GotNext
          END;

        'A, 'B, 'C, 'D, 'E, 'F, 'G, 'H, 'I, 'J, 'K, 'L, 'M,
        'N, 'O, 'P, 'Q, 'R, 'S, 'T, 'U, 'V, 'W, 'X, 'Y, 'Z  =>
          BEGIN
          first, last: CARDINAL;
```

```
    uId: BOOLEAN;
    i, j, h: CARDINAL;
    s1, s2: CARDINAL;
    i ← 0;  uId ← TRUE;  first ← last ← char-0C;
      DO
      buffer[i] ← char;
      IF (tI←tI+1) = tMax THEN FillTextBuffer[];
      char ← tB[tI];
      SELECT char FROM
        IN ['A..'Z] =>
          BEGIN  last ← char-0C;
          IF (i ← i+1) >= iMax THEN ExpandBuffer[];
          END;
        IN ['a..'z], IN ['0..'9] =>
          BEGIN  uId ← FALSE;
          IF (i ← i+1) >= iMax THEN ExpandBuffer[];
          END;
        ENDCASE =>  EXIT;
      ENDLOOP;
    i ← i+1;
    IF uId
      THEN
        BEGIN
        h ← ((first*128-first) + last) MOD hashval + 1;
        WHILE (j ← dHashTab[h].symptr) # 0
          DO
          IF dVocabIndex[j]-(s2←dVocabIndex[j-1]) = i
            THEN
              FOR s1 IN [0 .. i)
                DO
                IF buffer[s1] # vocab[s2] THEN EXIT;
                s2 ← s2+1;
                REPEAT
                  FINISHED => BEGIN class ← j; GO TO GotNext END;
                ENDLOOP;
          IF (h ← dHashTab[h].link) = 0 THEN EXIT;
          ENDLOOP;
        END;
    desc.length ← i;
    class ← tokenID;  value ← SymTabDefs.EnterString[@desc];
    GO TO GotNext
    END;

  '0, '1, '2, '3, '4, '5, '6, '7, '8, '9  =>
    BEGIN
    v, v10, v8: LONG INTEGER;
    scale: CARDINAL;
    valid, valid10, valid8, octal: BOOLEAN;
    MaxLiteral: CARDINAL = AltoDefs.maxword;
    vRep: ARRAY [0..SIZE[LONG INTEGER]) OF WORD;  -- machine dependent
    v10 ← v8 ← 0;  valid10 ← valid8 ← TRUE;
    WHILE char IN ['0..'9]
      DO
      IF valid10 THEN [v10, valid10] ← AppendDigit10[v10, char];
      IF valid8 THEN [v8, valid8] ← AppendDigit8[v8, char];
      NextChar[];
      ENDLOOP;
    SELECT char FROM
      'B, 'C =>
        BEGIN
        class ← IF char = 'C THEN tokenCHAR ELSE tokenNUM;
        v ← v8;  valid ← valid8;  octal ← TRUE;
        END;
      ENDCASE =>
        BEGIN
        class ← tokenNUM;  v ← v10;  valid ← valid10;  octal ← FALSE;
        END;
    SELECT char FROM
      'B, 'C, 'D =>
        BEGIN
        NextChar[];
        IF class = tokenNUM
          THEN
            BEGIN  scale ← 0;
            WHILE char IN ['0..'9]
              DO
```

```
            scale ← 10*scale + CARDINAL[char-'0];
            NextChar[];
            ENDLOOP;
          THROUGH [1 .. scale]  WHILE valid
            DO
            IF octal
              THEN [v, valid] ← AppendDigit8[v, '0]
              ELSE [v, valid] ← AppendDigit10[v, '0];
            ENDLOOP;
          END;
      END;
    ENDCASE;
  vRep ← LOOPHOLE[v];
  IF v <= MaxLiteral
    THEN value ← LitDefs.FindLiteral[vRep[0]]
    ELSE
      BEGIN
      IF class = tokenCHAR THEN valid ← FALSE;
      class ← tokenLNUM;
      value ← LitDefs.FindLitDescriptor[DESCRIPTOR[vRep]];
      END;
  IF ~valid THEN
    BEGIN
    nErrors ← nErrors + 1;  ScanError[number, index];
    END;
  GO TO GotNext
  END;

',, ';, ':, '←, '#, '~, '+, '*, '/, '↑, '@, '!,
'(, '), '[, '], '{, '}  =>
  BEGIN
  class ← dScanTab[char];  GO TO GetNext
  END;

'' =>
  BEGIN
  NextChar[];
  class ← tokenCHAR;  value ← LitDefs.FindLiteral[char-0C];
  GO TO GetNext
  END;

'" =>
  BEGIN
  i: CARDINAL;
  i ← 0;
    DO
    IF (tI←tI+1) = tMax THEN
      BEGIN  IF tEnded THEN GO TO EOFEnd;  FillTextBuffer[]  END;
    char ← tB[tI];
    SELECT char FROM
      '" =>
        BEGIN
        IF (tI←tI+1) = tMax THEN FillTextBuffer[];
        char ← tB[tI];
        IF char # '" THEN GO TO QuoteEnd;
        END;
      ENDCASE;
    IF i >= iMax
      THEN ExpandBuffer[
        !BufferOverflow =>
          BEGIN
          nErrors ← nErrors + 1;
          ScanError[string, tOrigin+tI];
          i ← 0;
          CONTINUE
          END];
    buffer[i] ← char;  i ← i+1;
    REPEAT
      QuoteEnd => NULL;
      EOFEnd => BEGIN  FillTextBuffer[];  char ← tB[tI]  END;
    ENDLOOP;
  desc.length ← i;
  value ← LitDefs.FindStringLiteral[@desc];
  IF char = 'L
    THEN  BEGIN  class ← tokenLSTR;  GO TO GetNext  END
    ELSE  BEGIN  class ← tokenSTR;   GO TO GotNext  END
```

```
        END;

      '- =>
        BEGIN
        pChar: CHARACTER;
        NextChar[];
        IF char # '-
          THEN  BEGIN  class ← tokenMINUS;  GO TO GotNext  END;
        char ← NUL;
          DO
          pChar ← char;
          IF (tI←tI+1) = tMax THEN
            BEGIN  IF tEnded THEN GO TO EndFile;  FillTextBuffer[]  END;
          char ← tB[tI];
          SELECT char FROM
            '- =>  IF pChar = '- THEN EXIT;
            CR =>  EXIT;
            ENDCASE;
          ENDLOOP;
        NextChar[];
        END;

      '. =>
        BEGIN
        NextChar[];
        IF char = '.
          THEN  BEGIN  class ← tokenDOTS;  GO TO GetNext  END
          ELSE  BEGIN  class ← tokenDOT;   GO TO GotNext  END
        END;

      '= =>
        BEGIN
        NextChar[];
        IF char = '>
          THEN  BEGIN  class ← tokenARROW;  GO TO GetNext  END
          ELSE  BEGIN  class ← tokenEQUAL;  GO TO GotNext  END
        END;

      '< =>
        BEGIN
        NextChar[];
        IF char = '=
          THEN  BEGIN  class ← tokenLE;    GO TO GetNext  END
          ELSE  BEGIN  class ← tokenLESS;  GO TO GotNext  END
        END;

      '> =>
        BEGIN
        NextChar[];
        IF char = '=
          THEN  BEGIN  class ← tokenGE;       GO TO GetNext  END
          ELSE  BEGIN  class ← tokenGREATER;  GO TO GotNext  END
        END;

      ENDCASE =>
        BEGIN
        class ← dScanTab[char];
        IF class # 0 THEN  GO TO GetNext;
        NextChar[];
        nErrors ← nErrors + 1;  ScanError[char, index];
        END;

    REPEAT
      GetNext =>
        BEGIN
        IF (tI←tI+1) = tMax THEN FillTextBuffer[];
        char ← tB[tI];
        END;
      GotNext =>  NULL;
      EndFile =>
        BEGIN
        FillTextBuffer[];  char ← tB[tI];
        class ← endmarker;  index ← tOrigin;  value ← 0;
        END;
    ENDLOOP;
  RETURN
```

```
    END;


  -- numerical conversion

  Digit: ARRAY CHARACTER ['0..'9] OF CARDINAL = [0,1,2,3,4,5,6,7,8,9];

  AppendDigit10: PROCEDURE [v: LONG INTEGER, digit: CHARACTER ['0..'9]]
      RETURNS [newV: LONG INTEGER, valid: BOOLEAN] =
    BEGIN
    MaxV: LONG INTEGER = 214748364;              -- (2**31-1)/10
    MaxD: CARDINAL = 7;                          -- (2**31-1) MOD 10
    d: [0..9] = Digit[digit];
    valid ← v < MaxV OR (v = MaxV AND d <= MaxD);
    newV ← 10*v + d;
    RETURN
    END;

  AppendDigit8: PROCEDURE [v: LONG INTEGER, digit: CHARACTER ['0..'9]]
      RETURNS [newV: LONG INTEGER, valid: BOOLEAN] =
    BEGIN
    MaxV: LONG INTEGER = 1777777777B;            -- (2**31-1)/8
    MaxD: CARDINAL = 7B;                         -- (2**31-1) MOD 8
    d: [0..9] = Digit[digit];
    valid ← (d < 8) AND (v < MaxV OR (v = MaxV AND d <= MaxD));
    newV ← 8*v + d;
    RETURN
    END;


  -- initialization/finalization

  ScanInit: PUBLIC PROCEDURE [
      textStream: StreamDefs.StreamHandle, tablePtr: POINTER TO LALRTable] =
    BEGIN
      BEGIN  OPEN tablePtr.scantable;
      dHashTab ← DESCRIPTOR [hashtab];
      dScanTab ← DESCRIPTOR [scantab];
      vocab ← LOOPHOLE[@vocabbody, STRING];
      dVocabIndex ← DESCRIPTOR [vocabindex];
      END;
    IF buffer = NIL THEN buffer ← SystemDefs.AllocateHeapString[256];
    iMax ← buffer.length ← buffer.maxlength;
    desc.base ← buffer;  desc.offset ← 0;
    stream ← textStream;
    streamOrigin ← StreamDefs.GetIndex[stream];
    tB ← SystemDefs.AllocatePages[TextPages];
    tOrigin ← tLimit ← 0;  tMax ← 0;  tEnded ← FALSE;
    FillTextBuffer[];  char ← tB[tI];
    nErrors ← 0;
    RETURN
    END;

  ScanReset: PUBLIC PROCEDURE RETURNS [CARDINAL] =
    BEGIN
    SystemDefs.FreePages[tB];
    IF buffer # NIL
      THEN BEGIN  SystemDefs.FreeHeapString[buffer];  buffer ← NIL END;
    [] ← SystemDefs.PruneHeap[];
    RETURN [nErrors]
    END;


  -- error handling

  StreamIndex: TYPE = StreamDefs.StreamIndex;

  PrintTextLine: PROCEDURE [origin: StreamIndex] RETURNS [start: StreamIndex] =
    BEGIN  OPEN IODefs;
    lineIndex: StreamIndex;
    char: CHARACTER;
    n: [1..100];
    start ← lineIndex ← origin;
    FOR n IN [1..100] UNTIL lineIndex = [0, 0]
      DO
      lineIndex ← StreamDefs.ModifyIndex[lineIndex, -1];
```

```
    StreamDefs.SetIndex[stream, lineIndex];
    IF stream.get[stream] = CR THEN EXIT;
    start ← lineIndex;
    ENDLOOP;
  StreamDefs.SetIndex[stream, start];
  FOR n IN [1..100]
    DO
    char ← stream.get[stream ! StreamDefs.StreamError => EXIT];
    SELECT char FROM
      CR, Control Z => EXIT;
      ENDCASE => WriteChar[char];
    ENDLOOP;
  WriteChar[CR];  RETURN
  END;


ResetScanIndex: PUBLIC PROCEDURE [index: CARDINAL] =
  BEGIN
  page: CARDINAL;
  IF index ~IN [tOrigin .. tLimit)
    THEN
      BEGIN
      page ← index/(AltoDefs.PageSize*AltoDefs.CharsPerWord);
      tOrigin ← tLimit ← page*(AltoDefs.PageSize*AltoDefs.CharsPerWord);
      tMax ← 0;
      StreamDefs.SetIndex[stream,
         [page: streamOrigin.page+page, byte: streamOrigin.byte]];
      FillTextBuffer[];
      END;
  tI ← index - tOrigin;  char ← tB[tI];  RETURN
  END;


ScanError: PROCEDURE [code: {number, string, char}, tokenIndex: CARDINAL] =
  BEGIN
  ErrorContext[
    SELECT code FROM
      number => "Invalid Number"L,
      string => "String Too Long"L,
      char => "Invalid Character"L,
      ENDCASE => NIL,
    tokenIndex];
  RETURN
  END;


ErrorContext: PUBLIC PROCEDURE [message: STRING, tokenIndex: CARDINAL] =
  BEGIN  OPEN IODefs;
  saveIndex: StreamIndex = StreamDefs.GetIndex[stream];
  origin: StreamIndex = StreamDefs.NormalizeIndex[
      [page: streamOrigin.page, byte: streamOrigin.byte+tokenIndex]];
  char: CHARACTER;
  WriteChar[CR];
  StreamDefs.SetIndex[stream, PrintTextLine[origin]];
  UNTIL StreamDefs.GetIndex[stream] = origin
    DO
    char ← stream.get[stream ! StreamDefs.StreamError => EXIT];
    WriteChar[IF char = TAB THEN TAB ELSE ' ];
    ENDLOOP;
  WriteString["↑ "L];  WriteString[message];  WriteString[" ["L];
  WriteNumber[tokenIndex, [base:8, zerofill:FALSE, unsigned:TRUE, columns:0]];
  WriteChar[']];   WriteChar[CR];
  StreamDefs.SetIndex[stream, saveIndex];  RETURN
  END;

END.
```