```
-- file DIParser.Mesa
-- last modified by
--                  Sandman, April 17, 1978  4:01 PM
--                  Barbara, May 15, 1978  11:13 AM

DIRECTORY
  DIDefs: FROM "didefs" USING [
    Atom, ConstOrQual, ESPointer, ParseHandle, ScanInit, ScanReset],
  DILALRDefs: FROM "dilalrdefs" USING [
    ActionEntry, ActionTag, AsstlEntry, endmarker, LALRTableHandle,
    lastntstate, ProductionInfo, State, Symbol, SymbolRecord],
  StreamDefs: FROM "streamdefs",
  SystemDefs: FROM "systemdefs" USING [AllocateResidentSegment, FreeSegment];

DIParser: PROGRAM
    IMPORTS SystemDefs, DIDefs
    EXPORTS DIDefs SHARES DILALRDefs =
  BEGIN  -- Debugger Interpreter parser with no error recovery
  OPEN DILALRDefs;

  InitialState: State = 1;
  FinalState: State = 0;

  currentState: State;
  inputSymbol, lhs: Symbol;
  DefaultMarker: Symbol = endmarker+1;

  input: PROCEDURE RETURNS [symbol: SymbolRecord];
  inputLoc: CARDINAL;
  inputValue: UNSPECIFIED;
  qI, top: CARDINAL;

  s: DESCRIPTOR FOR ARRAY OF State;
  l: DESCRIPTOR FOR ARRAY OF CARDINAL;
  v: DESCRIPTOR FOR ARRAY OF UNSPECIFIED;
  h: DESCRIPTOR FOR ARRAY OF DIDefs.ConstOrQual;

  q: DESCRIPTOR FOR ARRAY OF ActionEntry;

  stackSize: CARDINAL;
  queueSize: CARDINAL;

  -- transition tables for terminal input symbols

  tState: DESCRIPTOR FOR ARRAY OF State;
  asstl: DESCRIPTOR FOR ARRAY OF AsstlEntry;
  tSymbol: DESCRIPTOR FOR ARRAY OF Symbol;
  tAction: DESCRIPTOR FOR ARRAY OF ActionEntry;

  -- transition tables for nonterminal input symbols

  nState: DESCRIPTOR FOR ARRAY OF State;
  nLength: DESCRIPTOR FOR ARRAY OF CARDINAL;
  nSymbol: DESCRIPTOR FOR ARRAY OF Symbol;
  nAction: DESCRIPTOR FOR ARRAY OF ActionEntry;
  nDefaults: DESCRIPTOR FOR ARRAY OF ActionEntry;

  -- production information

  prodData: DESCRIPTOR FOR ARRAY OF ProductionInfo;


-- initialization/termination

  ParseInit: PROCEDURE
    [string: STRING, tablePtr: LALRTableHandle, ph: DIDefs.ParseHandle] =
    BEGIN
    DIDefs.ScanInit[string, tablePtr];

      BEGIN  OPEN tablePtr;
      tState ← DESCRIPTOR[parsetable.tstate];
      asstl ← DESCRIPTOR[parsetable.asstl];
      tSymbol ← DESCRIPTOR[parsetable.tsym];
      tAction ← DESCRIPTOR[parsetable.tact];
      nState ←DESCRIPTOR[parsetable.nstate];
      nLength ← DESCRIPTOR[parsetable.nlen];
```

```
      nSymbol ← DESCRIPTOR[parsetable.nsym];
      nAction ← DESCRIPTOR[parsetable.nact];
      nDefaults ← DESCRIPTOR[parsetable.ntdefaults];
      prodData ← DESCRIPTOR[parsetable.proddata];
      END;

    stackSize ← queueSize ← 0;  ExpandStack[64, ph];  ExpandQueue[256, ph];
    RETURN
    END;

  InputLoc: PUBLIC PROCEDURE RETURNS [CARDINAL] =
    BEGIN
    RETURN [inputLoc]
    END;

  ParseError: PUBLIC SIGNAL [errorLoc: CARDINAL] = CODE;

  SyntaxError: PUBLIC SIGNAL [errorLoc: CARDINAL] = CODE;

--  the main parsing procedures

  Parse: PUBLIC PROCEDURE
    [string: STRING, table: LALRTableHandle, ph: DIDefs.ParseHandle,
    proc: PROCEDURE [esp: DIDefs.ESPointer], locals: BOOLEAN]
    RETURNS [complete: BOOLEAN] =
    BEGIN
    i, valid, k, m: CARDINAL;           -- stack pointers
    j, j0: CARDINAL;
    tj: ActionEntry;

    input ← DIDefs.Atom;
    ParseInit[string, table, ph];
    i ← top ← valid ← 0;  qI ← 0;
    s[0] ← currentState ← InitialState;
    [inputSymbol, inputValue, inputLoc] ← input[].symbol;

    WHILE currentState # FinalState DO
      BEGIN
      j0 ← tState[currentState];
      FOR j IN [j0 .. j0 + asstl[currentState].tlen)
        DO
        SELECT tSymbol[j] FROM
          inputSymbol, DefaultMarker => EXIT;
          ENDCASE;
        REPEAT
          FINISHED => GO TO SyntaxError;
        ENDLOOP;

      tj ← tAction[j];
      IF ~tj.rtag.reduce        -- scan or scan reduce entry
        THEN
          BEGIN
          IF qI > 0
            THEN
              BEGIN
              FOR k IN (valid..i] DO s[k] ← s[top+(k-valid)] ENDLOOP;
              ph.qp[qI, top, proc, locals !
                DIDefs.ParseError => GO TO SyntaxError;
                UNWIND => BEGIN EraseQueue[]; EraseStack[]; END];
              qI ← 0;
              EXITS
                SyntaxError =>
                  BEGIN
                  EraseQueue[];  EraseStack[];
                  SIGNAL DIDefs.SyntaxError[inputLoc];
                  END;
            END;
          top ← valid ← i ← i+1;
          v[i] ← inputValue;  l[i] ← inputLoc;
          [inputSymbol,inputValue,inputLoc]  ← input[].symbol;
          END;

      WHILE tj.rtag # ActionTag[FALSE, 0]
        DO
        IF qI >= queueSize THEN ExpandQueue[256, ph];
        q[qI] ← tj;  qI ← qI + 1;
```

```
      i ← i-tj.rtag.plength;  -- pop 1 state per rhs symbol
      currentState ← s[IF i > valid THEN top+(i-valid) ELSE (valid ← i)];
      lhs ← prodData[tj.transition].lhs;
        BEGIN
        IF currentState <= lastntstate
          THEN
            BEGIN  j ← nState[currentState];
            FOR j IN [j..j+nLength[currentState])
              DO
              IF lhs = nSymbol[j] THEN
                BEGIN  tj ← nAction[j]; GO TO nfound  END;
              ENDLOOP;
            END;
        tj ← nDefaults[lhs];
        EXITS
          nfound => NULL;
        END;
      i ← i+1;
      ENDLOOP;
    IF (m ← top+(i-valid)) >= stackSize THEN ExpandStack[64, ph];
    s[m] ← currentState ← tj.transition;
    EXITS
      SyntaxError =>
        BEGIN
        EraseQueue[];  EraseStack[];
        SIGNAL DIDefs.SyntaxError[inputLoc];
        END;
    END;
  ENDLOOP;

  BEGIN
  ph.qp[qI, top, proc, locals !
    DIDefs.ParseError => GOTO SyntaxError;
    UNWIND => BEGIN EraseQueue[]; EraseStack[]; END];
  EXITS
    SyntaxError =>
      BEGIN
      EraseQueue[];  EraseStack[];
      SIGNAL DIDefs.SyntaxError[inputLoc];
      END;
  END;

  EraseQueue[];  EraseStack[];
  RETURN [DIDefs.ScanReset[]]
  END;

ExpandStack: PROCEDURE [delta: CARDINAL, ph: DIDefs.ParseHandle] =
  BEGIN  OPEN SystemDefs;
  i, sS, sL, sV, sH: CARDINAL;
  p: POINTER;
  newS: DESCRIPTOR FOR ARRAY OF State;
  newL: DESCRIPTOR FOR ARRAY OF CARDINAL;
  newV: DESCRIPTOR FOR ARRAY OF UNSPECIFIED;
  newH: DESCRIPTOR FOR ARRAY OF UNSPECIFIED;
  newSize: CARDINAL = stackSize + delta;
  sS ← newSize*SIZE[State];
  sL ← newSize*SIZE[CARDINAL];
  sV ← newSize*SIZE[UNSPECIFIED];
  sH ← newSize*SIZE[DIDefs.ConstOrQual];
  p ← AllocateResidentSegment[sS+sL+sV+sH];
  newS ← DESCRIPTOR[p, newSize];
  newL ← DESCRIPTOR[p+sS, newSize];
  newV ← DESCRIPTOR[p+sS+sL, newSize];
  newH ← DESCRIPTOR[p+sS+sL+sV, newSize];
  FOR i IN [0..stackSize)
    DO
    newS[i] ← s[i];
    newL[i] ← l[i];
    newV[i] ← v[i];
    newH[i] ← h[i]
    ENDLOOP;
  EraseStack[];
  s ← newS;  l ← newL;  v ← newV;  h ← newH;  stackSize ← newSize;
  ph.da[qd:q, vd:v, ld:l, hd:h, pd:prodData];
  RETURN
  END;
```

```
  EraseStack: PROCEDURE =
    BEGIN
    IF stackSize # 0 THEN SystemDefs.FreeSegment[BASE[s]];
    RETURN
    END;

  ExpandQueue: PROCEDURE [delta: CARDINAL, ph: DIDefs.ParseHandle] =
    BEGIN  OPEN SystemDefs;
    i: CARDINAL;
    newQ: DESCRIPTOR FOR ARRAY OF ActionEntry;
    newSize: CARDINAL = queueSize + delta;
    newQ ←
      DESCRIPTOR[AllocateResidentSegment[newSize*SIZE[ActionEntry]], newSize];
    FOR i IN [0..queueSize)  DO  newQ[i] ← q[i]  ENDLOOP;
    EraseQueue[];
    q ← newQ;  queueSize ← newSize;
    ph.da[qd:q, vd:v, ld:l, hd:h, pd:prodData];
    RETURN
    END;

  EraseQueue: PROCEDURE =
    BEGIN
    IF queueSize # 0 THEN SystemDefs.FreeSegment[BASE[q]];
    RETURN
    END;

END...
```