```
-- file Error.Mesa
-- last modified by Satterthwaite, May 16, 1978  1:23 PM

DIRECTORY
  ComData: FROM "comdata"
    USING [
      bodyIndex, nErrors, nWarnings, sourceStream, textIndex, warnings],
  CompilerDefs: FROM "compilerdefs" USING [CloseStringTable, OpenStringTable],
  ErrorDefs: FROM "errordefs" USING [ErrorCode],
  ErrorTabDefs: FROM "errortabdefs" USING [CSRptr],
  IODefs: FROM "iodefs"
    USING [ControlZ, CR, WriteChar, WriteDecimal, WriteNumber, WriteString],
  LitDefs: FROM "litdefs" USING [LiteralValue, StringLiteralValue],
  StreamDefs: FROM "streamdefs"
    USING [
      StreamIndex,
      CloseDiskStream, ModifyIndex, NormalizeIndex, OpenDiskStream, SetIndex,
      StreamError],
  StringDefs: FROM "stringdefs" USING [SubString, SubStringDescriptor],
  SymDefs: FROM "symdefs"
    USING [setype, bodytype,
      HTIndex, ISEIndex, HTNull, SENull, BTNull],
  SymTabDefs: FROM "symtabdefs" USING [SubStringForHash],
  TableDefs: FROM "tabledefs" USING [TableBase, TableBounds],
  TreeDefs: FROM "treedefs"
    USING [treetype,
      NodeName, TreeLink, TreeIndex, TreeScan, empty, scanlist];

Error: PROGRAM
    IMPORTS
        CompilerDefs, IODefs, LitDefs, StreamDefs,
        SymTabDefs, TableDefs, TreeDefs,
        dataPtr: ComData
    EXPORTS ErrorDefs =
  BEGIN
  OPEN SymDefs, TreeDefs;

  ErrorCode: TYPE = ErrorDefs.ErrorCode;

  SubString: TYPE = StringDefs.SubString;

 -- source printing

  PrintTextLine: PROCEDURE [i: CARDINAL] =
    BEGIN  OPEN StreamDefs, IODefs;
    start, lineIndex: StreamIndex;
    char: CHARACTER;
    n: [1..100];
    OpenDiskStream[dataPtr.sourceStream];
    start ← lineIndex ← NormalizeIndex[[page:0, byte:i]];
    FOR n IN [1..100] UNTIL lineIndex = [0, 0]
      DO
      lineIndex ← ModifyIndex[lineIndex, -1];
      SetIndex[dataPtr.sourceStream, lineIndex];
      IF dataPtr.sourceStream.get[dataPtr.sourceStream] = CR THEN EXIT;
      start ← lineIndex;
      ENDLOOP;
    SetIndex[dataPtr.sourceStream, start];
    FOR n IN [1..100]
      DO
      char ← dataPtr.sourceStream.get[dataPtr.sourceStream
        !StreamError => EXIT];
      SELECT char FROM
        CR, ControlZ => EXIT;
        ENDCASE => WriteChar[char];
      ENDLOOP;
    WriteChar[CR];
    CloseDiskStream[dataPtr.sourceStream];
    RETURN
    END;


  -- CSRp and desc.base are set by LockStringTable

  CSRp: ErrorTabDefs.CSRptr;
  desc: StringDefs.SubStringDescriptor;
  ss: SubString = @desc;

  LockStringTable: PROCEDURE =
    BEGIN
    CSRp ← CompilerDefs.OpenStringTable[];
    ss.base ← LOOPHOLE[CSRp + CSRp.relativebase, STRING];
    RETURN
    END;


  WriteSubString: PROCEDURE [ss: SubString] =
    BEGIN
    i: CARDINAL;
    FOR i IN [ss.offset..ss.offset + ss.length)
      DO  IODefs.WriteChar[ss.base[i]]  ENDLOOP;
    RETURN
    END;

  WriteErrorString: PROCEDURE [n: ErrorCode] =
    BEGIN
    ss.offset ← CSRp.ErrorMessages[n].offset;
    ss.length ← CSRp.ErrorMessages[n].length;
    WriteSubString[ss];
    RETURN
    END;

  WriteHti: PROCEDURE [hti: HTIndex] =
    BEGIN  OPEN IODefs;
    desc: StringDefs.SubStringDescriptor;
    s: SubString = @desc;
    IF hti = HTNull
      THEN WriteString["(anonymous)"L]
      ELSE  BEGIN SymTabDefs.SubStringForHash[s, hti]; WriteSubString[s] END;
    RETURN
    END;

  WriteSei: PROCEDURE [sei: ISEIndex] =
    BEGIN
    WriteHti[IF sei=SENull
        THEN HTNull
        ELSE (TableDefs.TableBounds[SymDefs.setype].base+sei).htptr];
    RETURN
    END;


  WriteLti: PROCEDURE [t: literal TreeLink] =
    BEGIN  OPEN IODefs;
    WITH t.info SELECT FROM
      word =>  WriteDecimal[LitDefs.LiteralValue[index]];
      string =>
        BEGIN
        WriteChar['"];
        WriteString[LitDefs.StringLiteralValue[index]];
        WriteChar['"];
        END;
      ENDCASE;
    RETURN
    END;


  -- tables used for printing trees


--     pname: ARRAY NodeName[assignx..uparrow] OF STRING ←
--      ["←",
--        " OR ", " AND ", "=", "#", "<", ">=", ">", "<=", " IN ", " ~IN ",
--        "+", "-", "*", "/", " MOD ",
--        ".", ".", ".", ".",
--        "~", "-", "@", "↑"];

  WritePName: PROCEDURE[n: NodeName[assignx..uparrow]] =
    BEGIN
    ss.offset ← CSRp.pname[n].offset;
    ss.length ← CSRp.pname[n].length;
    WriteSubString[ss];  RETURN
    END;
    OpPrec: ARRAY NodeName[assignx..uparrow] OF CARDINAL =
     [1,
      2, 3, 5, 5, 5, 5, 5, 5, 5, 5,
      6, 6, 7, 7, 7,
      10, 10, 10,
      4, 8, 9, 10];


--    fname: ARRAY NodeName[min..memory] OF STRING ←
--     ["MIN", "MAX", "LONG", "ABS",
--      "SIZE", "FIRST", "LAST", "DESCRIPTOR",
--      "LENGTH", "BASE", "LOOPHOLE", "REGISTER", "MEMORY"];

  WriteFName: PROCEDURE[n: NodeName[min..memory]] =
    BEGIN
    ss.offset ← CSRp.fname[n].offset;
    ss.length ← CSRp.fname[n].length;
    WriteSubString[ss];  RETURN
    END;

  cutoff: CARDINAL = 3;

  PrintOperand: PROCEDURE [t: TreeLink, tPrec, depth: INTEGER] =
    BEGIN
    node: TreeIndex;
    prec: INTEGER;
    op: NodeName;
    args: TreeLink;
    tb: TableDefs.TableBase;
    IF t = empty THEN RETURN;
    WITH e: t SELECT FROM
      hash =>  WriteHti[e.index];
      symbol =>  WriteSei[e.index];
      literal =>  WriteLti[e];
      subtree =>
        BEGIN  OPEN TableDefs, IODefs;
        tb ← TableBounds[treetype].base;
        node ← e.index;  op ← (tb+node).name;
        IF depth > cutoff THEN  BEGIN  WriteString["..."L];  RETURN  END;
        SELECT op FROM
          IN [apply .. rowconsx], IN [min .. memory] =>
            BEGIN  OPEN (tb+node);
            SELECT op FROM
              IN [apply .. rowconsx] =>
                BEGIN
                IF son1 # empty THEN PrintOperand[son1, 0, depth];
                args ← son2;
                END;
              IN [min .. memory] =>  BEGIN  WriteFName[op];  args ← son1  END;
              ENDCASE;
            WriteChar['[];
            IF depth = cutoff AND args.tag = subtree
              THEN WriteString["..."L]
              ELSE PrintOperandList[args, depth+1];
            IF op IN [apply .. join] AND nsons > 2
              THEN WriteString[" !..."L];
            WriteChar[']];
            END;
          IN [assignx .. uparrow] =>
            BEGIN  OPEN (tb+node);
            prec ← OpPrec[op];
            IF prec < tPrec THEN WriteChar['(];
            SELECT op FROM
              IN [not .. addr] =>
                BEGIN  WritePName[op];  PrintOperand[son1, prec, depth]  END;
              IN [assignx .. dollar] =>
                BEGIN
                PrintOperand[son1, prec, depth+1];
                WritePName[op];
                PrintOperand[son2, prec+1, depth+1];
                END;
              uparrow =>
                BEGIN  PrintOperand[son1, prec, depth];  WriteChar['↑]  END;
              ENDCASE =>  WriteChar['?];
            IF prec < tPrec THEN WriteChar[')];
          END;
        IN [intOO .. intCC] =>
          BEGIN  OPEN (tb+node);
          WriteChar[IF op = intOO OR op = intOC THEN '( ELSE '[];
          PrintOperand[son1, 0, depth];
          WriteChar['.]; WriteChar['.];
          PrintOperand[son2, 0, depth];
          WriteChar[IF op = intOO OR op = intCO THEN ') ELSE ']];
          END;
        clit =>
          BEGIN  WriteChar[''];
          WITH e1: (tb+node).son1 SELECT FROM
            literal =>
              WITH e1.info SELECT FROM
                word => WriteChar[LOOPHOLE[LitDefs.LiteralValue[index]]];
                ENDCASE;
            ENDCASE;
          END;
        llit, IN [cast .. openexp] =>
          PrintOperand[(tb+node).son1, tPrec, depth];
        ENDCASE =>  WriteString["..."L];
      END;
    ENDCASE;
  RETURN
  END;

PrintOperandList: PROCEDURE [t: TreeLink, depth: INTEGER] =
  BEGIN
  firstSon: BOOLEAN ← TRUE;

  PrintItem: TreeScan =
    BEGIN  OPEN IODefs;
    IF ~firstSon THEN WriteString[", "L] ELSE firstSon ← FALSE;
    IF t # empty THEN PrintOperand[t, 0, depth];
    RETURN
    END;

  scanlist[t, PrintItem];  RETURN
  END;

-- error-handling routines


ErrorLog: PROCEDURE =
  BEGIN  OPEN IODefs;
  bodyId: ISEIndex;
  WriteString[", at "L];
  IF dataPtr.bodyIndex # BTNull
    THEN
      BEGIN  OPEN TableDefs;
      bodyId ← (TableBounds[bodytype].base+dataPtr.bodyIndex).id;
      IF bodyId # SENull THEN WriteSei[bodyId];
      END;
  WriteChar['[];
  WriteNumber[dataPtr.textIndex, [base:8, zerofill:FALSE, unsigned:TRUE, columns:0]];
  WriteChar[']];  WriteChar[':];  WriteChar[CR];
  PrintTextLine[dataPtr.textIndex];
  RETURN
  END;

error: PUBLIC PROCEDURE [code: ErrorCode] =
  BEGIN  OPEN IODefs;
  LockStringTable[];
  WriteChar[CR];  WriteErrorString[code];
  dataPtr.nErrors ← dataPtr.nErrors + 1;
  ErrorLog[];
  CompilerDefs.CloseStringTable[];  RETURN
  END;

errorhti: PUBLIC PROCEDURE [code: ErrorCode, hti: HTIndex] =
  BEGIN
  errortree[code, TreeLink[hash[hti]]];
  RETURN
  END;

errorsei: PUBLIC PROCEDURE [code: ErrorCode, sei: ISEIndex] =
    BEGIN
    errortree[code, TreeLink[symbol[sei]]];
    RETURN
    END;

  errorstring: PUBLIC PROCEDURE [code: ErrorCode, s: STRING] =
    BEGIN  OPEN IODefs;
    LockStringTable[];
    WriteChar[CR];  WriteString[s];  WriteChar[' ];
    WriteErrorString[code];
    dataPtr.nErrors ← dataPtr.nErrors + 1;
    ErrorLog[];
    CompilerDefs.CloseStringTable[];  RETURN
    END;

  errorn: PUBLIC PROCEDURE [code: ErrorCode, n: INTEGER] =
    BEGIN  OPEN IODefs;
    LockStringTable[];
    WriteChar[CR];  WriteDecimal[n];  WriteChar[' ];
    WriteErrorString[code];
    dataPtr.nErrors ← dataPtr.nErrors + 1;
    ErrorLog[];
    CompilerDefs.CloseStringTable[];  RETURN
    END;

  errortree: PUBLIC PROCEDURE [code: ErrorCode, t: TreeLink] =
    BEGIN  OPEN IODefs;
    LockStringTable[];
    WriteChar[CR];  PrintOperand[t, 0, 0];
    WriteChar[' ];  WriteChar[' ];  WriteErrorString[code];
    dataPtr.nErrors ← dataPtr.nErrors + 1;
    ErrorLog[];
    CompilerDefs.CloseStringTable[];  RETURN
    END;

  Warning: PUBLIC PROCEDURE [code: ErrorCode] =
    BEGIN
    IF dataPtr.warnings
      THEN
        BEGIN  OPEN IODefs;
        LockStringTable[];
        WriteChar[CR];  WriteErrorString[code];
        dataPtr.nWarnings ← dataPtr.nWarnings + 1;
        ErrorLog[];
        CompilerDefs.CloseStringTable[];
        END;
    RETURN
    END;

  WarningSei: PUBLIC PROCEDURE [code: ErrorCode, sei: ISEIndex] =
    BEGIN
    IF dataPtr.warnings THEN WarningTree[code, TreeLink[symbol[sei]]];
    RETURN
    END;

  WarningTree: PUBLIC PROCEDURE [code: ErrorCode, t: TreeLink] =
    BEGIN
    IF dataPtr.warnings
      THEN
        BEGIN  OPEN IODefs;
        LockStringTable[];
        WriteChar[CR];  PrintOperand[t, 0, 0];
        WriteChar[' ];  WriteChar[' ];  WriteErrorString[code];
        dataPtr.nWarnings ← dataPtr.nWarnings + 1;
        ErrorLog[];
        CompilerDefs.CloseStringTable[];
        END;
    RETURN
    END;

  END.
```