```
-- file Debug.Mesa
-- last modified by Satterthwaite, July 16, 1978  11:15 AM

DIRECTORY
  ComData: FROM "comdata" USING [bodyRoot, definitionsOnly],
  CompilerDefs: FROM "compilerdefs" USING [CloseStringTable, OpenStringTable],
  ControlDefs: FROM "controldefs" USING [ControlLink],
  ErrorTabDefs: FROM "errortabdefs" USING [CSRptr],
  IODefs: FROM "iodefs"
    USING [CR, TAB, WriteChar, WriteDecimal, WriteOctal, WriteString],
  LitDefs: FROM "litdefs"
    USING [
      lttype, LitDescriptor, LitDescriptorValue,
      MasterString, StringLiteralValue],
  StringDefs: FROM "stringdefs" USING [SubString, SubStringDescriptor],
  SymDefs: FROM "symdefs"
    USING [
      setype, ctxtype, mdtype, bodytype,
      BitAddress, CTXRecord, TransferMode, TypeClass,
      HTIndex, SEIndex, ISEIndex, recordCSEIndex, CTXIndex, BTIndex,
      HTNull, SENull, BTNull,
      lG, lZ, typeTYPE],
  SymSegDefs: FROM "symsegdefs" USING [FindExtension],
  SymTabDefs: FROM "symtabdefs"
    USING [NextSe, SubStringForHash, TypeLink, XferMode],
  TableDefs: FROM "tabledefs"
    USING [TableBase, TableNotifier, AddNotify, DropNotify, TableBounds],
  TreeDefs: FROM "treedefs"
    USING [
      treetype, NodeName, TreeLink, TreeIndex, TreeMap, nullTreeIndex,
      UpdateTree];

Debug: PROGRAM
    IMPORTS
        CompilerDefs, IODefs, LitDefs, SymSegDefs, SymTabDefs, TableDefs,
        TreeDefs,
        dataPtr: ComData
    EXPORTS CompilerDefs =
  BEGIN
  OPEN IODefs, SymDefs;

  tb: TableDefs.TableBase;
  seb: TableDefs.TableBase;
  ctxb: TableDefs.TableBase;
  mdb: TableDefs.TableBase;
  bb: TableDefs.TableBase;
  ltb: TableDefs.TableBase;

  DebugNotify: TableDefs.TableNotifier =
    BEGIN
    tb ← base[TreeDefs.treetype];
    seb ← base[setype];  ctxb ← base[ctxtype];  mdb ← base[mdtype];
    bb ← base[bodytype];
    ltb ← base[LitDefs.lttype];
    RETURN
    END;

  SubString: TYPE = StringDefs.SubString;

  -- csrP and desc.base are set by SwapInStringTab

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

  SwapInStringTab: 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  WriteChar[ss.base[i]]  ENDLOOP;
  RETURN
  END;


-- tree printing

PrintLiteral: PROCEDURE[t: literal TreeDefs.TreeLink] =
  BEGIN  OPEN LitDefs;
  desc: LitDescriptor;
  i: CARDINAL;
  v: WORD;
  WITH t.info SELECT FROM
    string =>
      BEGIN  WriteChar['"];
      WriteString[StringLiteralValue[index]];
      WriteChar['"];
      IF index # MasterString[index] THEN WriteChar['L];
      END;
    word =>
      BEGIN
      desc ← LitDescriptorValue[index];
      IF desc.length # 1 THEN WriteChar['[];
      FOR i IN [0 .. desc.length)
        DO
        IF (v ← (ltb+desc.offset)[i]) < 1000
          THEN WriteDecimal[v]
          ELSE WriteOctal[v];
        IF i+1 # desc.length THEN WriteChar[',];
        ENDLOOP;
      IF desc.length # 1 THEN WriteChar[']];
      END;
    ENDCASE;
  END;

WriteNodeName: PROCEDURE[n: TreeDefs.NodeName] =
  BEGIN
  ss.offset ← csrP.NodePrintName[n].offset;
  ss.length ← csrP.NodePrintName[n].length;
  WriteSubString[ss];  RETURN
  END;

PrintSubTree: PROCEDURE [t: TreeDefs.TreeLink, nBlanks: CARDINAL] =
  BEGIN  OPEN TreeDefs;

  Printer: TreeMap =
    BEGIN
    node: TreeIndex;
    Indent[nBlanks];
    WITH s: t SELECT FROM
      hash =>  PrintHti[s.index];
      symbol =>  PrintSei[s.index];
      literal =>  PrintLiteral[s];
      subtree =>
        BEGIN  node ← s.index;
        SELECT node FROM
          nullTreeIndex =>  WriteString["<empty>"L];
          ENDCASE =>
            BEGIN  OPEN (tb+node);
            WriteNodeName[name];
            WriteChar['[];  PrintIndex[node];  WriteString["] "L];
            IF info # 0
              THEN  BEGIN  WriteString["  info="L];  PrintIndex[info]  END;
            IF attr1 OR attr2
              THEN
                BEGIN
                IF info = 0 THEN WriteChar[' ];
                WriteChar['(];
                IF attr1 THEN WriteChar['1];
                IF attr2 THEN WriteChar['2];
                WriteChar[')];
                END;
            nBlanks ← nBlanks + 2;
            [] ← TreeDefs.UpdateTree[s, Printer];
            nBlanks ← nBlanks - 2;
            END;
```

```
        END;
      ENDCASE;
    RETURN [t]
    END;

  [] ← Printer[t];  RETURN
  END;


PrintTree: PUBLIC PROCEDURE [t: TreeDefs.TreeLink] =
  BEGIN
  TableDefs.AddNotify[DebugNotify];  SwapInStringTab[];
  PrintSubTree[t, 0];  WriteChar[CR];
  CompilerDefs.CloseStringTable[];  TableDefs.DropNotify[DebugNotify];  RETURN
  END;

PrintBodies: PUBLIC PROCEDURE =
  BEGIN
  bti, prev: BTIndex;
  TableDefs.AddNotify[DebugNotify];  SwapInStringTab[];
  bti ← dataPtr.bodyRoot;
    DO
    PrintBody[bti];
    IF (bb+bti).firstSon # BTNull
      THEN bti ← (bb+bti).firstSon
      ELSE
        DO
        prev ← bti;  bti ← (bb+bti).link.index;
        IF bti = BTNull THEN GO TO Done;
        IF (bb+prev).link.which # parent THEN EXIT;
        ENDLOOP;
    REPEAT
      Done => NULL;
    ENDLOOP;
  WriteChar[CR];
  CompilerDefs.CloseStringTable[];  TableDefs.DropNotify[DebugNotify];  RETURN
  END;

PrintBody: PROCEDURE [bti: BTIndex] =
  BEGIN
  OPEN body: (bb+bti);
  WriteChar[CR];  WriteChar[CR];  WriteString["Body: "L];
  WITH b: body SELECT FROM
    Callable =>
      BEGIN
      PrintSei[b.id];
      WriteString[", ep: "L];  WriteDecimal[b.entryIndex];
      WITH b SELECT FROM
        Inner =>
          BEGIN
          WriteString[", frame address: "L];  WriteDecimal[frameOffset];
          END;
        ENDCASE;
      END;
    ENDCASE =>  WriteString["(anon)"L];
  WriteChar[CR];
  WriteString["  context: "L];  PrintIndex[body.localCtx];
  WriteString[", static level: "L];  WriteDecimal[body.level];
  WITH body.info SELECT FROM
    Internal =>
      BEGIN
      WriteString[", frame size: "L];  WriteDecimal[frameSize];
      IF body.kind = Callable
        THEN
          BEGIN  WriteChar[CR];
          PrintSubTree[[subtree[index: bodyTree]], 0];
          END
        ELSE
          BEGIN
          WriteString[", tree root: "L];  PrintIndex[bodyTree];
          END;
      END;
    ENDCASE;
  WriteChar[CR];  RETURN
  END;
```

```
PrintSymbols: PUBLIC PROCEDURE =
  BEGIN
  ctx: CTXIndex;
  limit: CTXIndex = LOOPHOLE[TableDefs.TableBounds[SymDefs.ctxtype].size];
  ctx ← FIRST[CTXIndex] + SIZE [nil CTXRecord];
  UNTIL ctx = limit
    DO
    WriteChar[CR];  PrintContext[ctx];
    ctx ← ctx + (WITH (ctxb+ctx) SELECT FROM
      included => SIZE [included CTXRecord],
      imported => SIZE [imported CTXRecord],
      ENDCASE => SIZE [simple CTXRecord]);
    ENDLOOP;
  WriteChar[CR];  RETURN
  END;

PrintContext: PROCEDURE [ctx: CTXIndex] =
  BEGIN
  sei, root: ISEIndex;
  TableDefs.AddNotify[DebugNotify];  SwapInStringTab[];
  WriteChar[CR];
  WriteString["Context: "L];  PrintIndex[ctx];
  IF (ctxb+ctx).ctxlevel # 1Z THEN
    BEGIN  WriteString[", static level: "L];
    WriteDecimal[(ctxb+ctx).ctxlevel];
    END;
  WITH (ctxb+ctx) SELECT FROM
    included =>
      BEGIN  WriteString[", copied from [file: "L];
      PrintHti[(mdb+ctxmodule).mdhti];
      WriteString[", context: "L];  PrintIndex[ctxmap];
      WriteChar[']];
      END;
    imported =>
      BEGIN  WriteString[", imported from file: "L];
      PrintHti[(mdb+(ctxb+includeLink).ctxmodule).mdhti];
      END;
    ENDCASE;
  root ← sei ← (ctxb+ctx).selist;
    DO
    IF sei = SENull THEN EXIT;
    PrintSE[sei, 2];
    IF (sei ← SymTabDefs.NextSe[sei]) = root THEN EXIT;
    ENDLOOP;
  CompilerDefs.CloseStringTable[];  TableDefs.DropNotify[DebugNotify];
  RETURN
  END;


PrintSE: PROCEDURE [sei: ISEIndex, nBlanks: CARDINAL] =
  BEGIN  OPEN (seb+sei);
  typeSei: SEIndex;
  addr: BitAddress;
  link: ControlDefs.ControlLink;
  Indent[nBlanks];
  PrintSei[sei];
  WriteString[" ["L];  PrintIndex[sei];  WriteChar[']];
  IF public THEN WriteString[" [public]"L];
  IF mark3
    THEN
      BEGIN
      WriteString[", type = "L];
      IF idtype = typeTYPE
        THEN
          BEGIN  typeSei ← idinfo;
          WriteString["TYPE, equated to: "L];
          PrintType[typeSei];
          IF SymTabDefs.TypeLink[sei] # SENull
            THEN
              BEGIN WriteString[", tag code: "L]; WriteDecimal[idvalue] END;
          END
        ELSE
          BEGIN  typeSei ← idtype;  PrintType[typeSei];
          SELECT TRUE FROM
            constant => WriteString[" [const]"L];
```

```
            writeonce => WriteString[" [init only]"L];
            ENDCASE;
          IF ~mark4
            THEN
              BEGIN WriteString[", # refs: "L]; WriteDecimal[idinfo] END
            ELSE
              SELECT TRUE FROM
                constant =>
                  IF ~ extended THEN
                    BEGIN  WriteString[", value: "L];
                    SELECT SymTabDefs.XferMode[typeSei] FROM
                      procedure, program, signal, error =>
                        BEGIN  link ← idvalue;
                        WriteChar['[];
                        WriteDecimal[link.gfi];  WriteChar[',];
                        WriteDecimal[link.ep];  WriteChar[',];
                        WriteDecimal[LOOPHOLE[link.tag]];  WriteChar[']];
                        END;
                      ENDCASE =>
                        IF LOOPHOLE[idvalue, CARDINAL] < 1000
                          THEN  WriteDecimal[idvalue]
                          ELSE  WriteOctal[idvalue];
                    END;
                (dataPtr.definitionsOnly AND (ctxb+ctxnum).ctxlevel = 1G) =>
                  BEGIN WriteString[", index: "L]; WriteDecimal[idvalue] END;
                ENDCASE =>
                  BEGIN  addr ← idvalue;
                  WriteString[", address: "L];
                  WriteDecimal[addr.wd];  WriteChar[' ];
                  WriteChar['[];  WriteDecimal[addr.bd];
                  WriteChar[':];  WriteDecimal[idinfo];
                  WriteChar[']];
                  IF linkSpace THEN WriteChar['*];
                  END;
          END;
        PrintTypeInfo[typeSei, nBlanks+2];
        IF extended
          THEN PrintSubTree[SymSegDefs.FindExtension[sei], nBlanks+4];
        END;
    RETURN
    END;


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

  PrintSei: PROCEDURE [sei: ISEIndex] =
    BEGIN
    PrintHti[IF sei=SENull THEN HTNull ELSE (seb+sei).htptr];
    RETURN
    END;


  WriteTypeName: PROCEDURE [n: TypeClass] =
    BEGIN
    ss.offset ← csrP.TypePrintName[n].offset;
    ss.length ← csrP.TypePrintName[n].length;
    WriteSubString[ss];  RETURN
    END;

  WriteModeName: PROCEDURE [n: TransferMode] =
    BEGIN
    ss.offset ← csrP.ModePrintName[n].offset;
    ss.length ← csrP.ModePrintName[n].length;
    WriteSubString[ss];  RETURN
    END;

  PrintType: PROCEDURE [sei: SEIndex] =
```

```
    BEGIN
    tSei: SEIndex;
    IF sei = SENull
      THEN WriteChar['?]
      ELSE
        WITH t: (seb+sei) SELECT FROM
          constructor =>
            WITH t SELECT FROM
              transfer => WriteModeName[mode];
              ENDCASE =>  WriteTypeName[t.typetag];
          id =>
            BEGIN
            PrintSei[LOOPHOLE[sei, ISEIndex]];  tSei ← sei;
            UNTIL (tSei ← SymTabDefs.TypeLink[tSei]) = SENull
              DO
              WITH (seb+tSei) SELECT FROM
                id =>
                  BEGIN  WriteChar[' ];  PrintSei[LOOPHOLE[tSei, ISEIndex]];
                  END;
                ENDCASE;
              ENDLOOP;
            END;
          ENDCASE;
    WriteString[" ["L];  PrintIndex[sei];  WriteChar[']];
    RETURN
    END;

  PrintTypeInfo: PROCEDURE [sei: SEIndex, nBlanks: CARDINAL] =
    BEGIN
    IF sei # SENull
      THEN
        WITH s: (seb+sei) SELECT FROM
          constructor =>
            BEGIN  Indent[nBlanks];
            WriteChar['[];  PrintIndex[sei];  WriteString["] "L];
            WITH s SELECT FROM
              transfer => WriteModeName[mode];
              ENDCASE => WriteTypeName[s.typetag];
            WITH t: s SELECT FROM
              basic =>
                BEGIN  WriteString[", length: "L]; WriteDecimal[t.length]  END;
              enumerated =>
                BEGIN
                IF t.ordered THEN WriteString[" (ordered)"L];
                WriteString[", value ctx: "L];
                PrintIndex[t.valuectx];
                END;
              record =>
                BEGIN
                IF ~t.lengthUsed THEN WriteChar['*];
                IF t.machineDep THEN WriteString[" (md)"L];
                IF t.monitored THEN WriteString[" (monitored)"L];
                IF t.variant THEN WriteString[" (variant)"L];
                OutRecordCtx[", field ctx: "L, LOOPHOLE[sei, recordCSEIndex]];
                WITH (ctxb+t.fieldctx) SELECT FROM
                  included =>
                    IF ~ctxcomplete THEN WriteString[" [partial]"L];
                  imported => WriteString[" [partial]"L];
                  ENDCASE;
                WITH t SELECT FROM
                  linked =>
                    BEGIN  WriteString[", link: "L];
                    PrintType[linktype];
                    END;
                  ENDCASE;
                END;
              pointer =>
                BEGIN
                IF ~t.dereferenced THEN WriteChar['*];
                IF t.ordered THEN WriteString[" (ordered)"L];
                IF t.basing THEN WriteString[" (base)"L];
                WriteString[", pointing to: "L];
                PrintType[t.pointedtotype];
                PrintTypeInfo[t.pointedtotype, nBlanks+2];
                END;
              array =>
```

```
            BEGIN
            IF ~t.lengthUsed THEN WriteChar['*];
            IF t.packed THEN WriteString[" (packed)"L];
            WriteString[", index type: "L]; PrintType[t.indextype];
            WriteString[", component type: "L]; PrintType[t.componenttype];
            PrintTypeInfo[t.indextype, nBlanks+2];
            PrintTypeInfo[t.componenttype, nBlanks+2];
            END;
          arraydesc =>
            BEGIN
            WriteString[", described type: "L]; PrintType[t.describedType];
            PrintTypeInfo[t.describedType, nBlanks+2];
            END;
          transfer =>
            BEGIN
            OutRecordCtx[", input ctx: "L, t.inrecord];
            OutRecordCtx[", output ctx: "L, t.outrecord];
            END;
          definition =>
            BEGIN
            WriteString[", ctx: "L];  PrintIndex[t.defCtx];
            WriteString[", ngfi: "L];  WriteDecimal[t.nGfi];
            END;
          union =>
            BEGIN
            IF t.overlayed THEN WriteString[" (overlayed)"L];
            IF t.controlled
              THEN  BEGIN WriteString[", tag: "L]; PrintSei[t.tagsei]  END;
            WriteString[", tag type: "L];
            PrintType[(seb+t.tagsei).idtype];
            WriteString[", case ctx: "L];  PrintIndex[t.casectx];
            IF t.controlled
              THEN PrintSE[t.tagsei, nBlanks+2];
            END;
          relative =>
            BEGIN
            WriteString[", base type: "L]; PrintType[t.baseType];
            WriteString[", offset type: "L]; PrintType[t.offsetType];
            WriteString[", result type: "L]; PrintType[t.resultType];
            PrintTypeInfo[t.baseType, nBlanks+2];
            PrintTypeInfo[t.offsetType, nBlanks+2];
            PrintTypeInfo[t.resultType, nBlanks+2];
            END;
          subrange =>
            BEGIN
            WriteString[" of: "L];  PrintType[t.rangetype];
            IF t.filled
              THEN
                BEGIN
                WriteString[" origin: "L];  WriteDecimal[t.origin];
                WriteString[", range: "L];
                IF t.flexible
                  THEN WriteChar['*]
                  ELSE WriteDecimal[t.range];
                END;
            PrintTypeInfo[t.rangetype, nBlanks+2];
            END;
          long, real =>
            BEGIN
            WriteString[" of: "L];  PrintType[t.rangetype];
            PrintTypeInfo[t.rangetype, nBlanks+2];
            END;
          ENDCASE;
        END;
      ENDCASE;
  RETURN
  END;

OutRecordCtx: PROCEDURE [message: STRING, sei: recordCSEIndex] =
  BEGIN
  WriteString[message];
  IF sei = SENull
    THEN WriteString["NIL"L]
    ELSE PrintIndex[(seb+sei).fieldctx];
  RETURN
  END;
```

```
  PrintIndex: PROCEDURE [v: UNSPECIFIED] = LOOPHOLE[WriteDecimal];

  Indent: PROCEDURE [n: CARDINAL] =
    BEGIN
    WriteChar[CR];
    THROUGH [1..n/8] DO WriteChar[TAB] ENDLOOP;
    THROUGH [1..n MOD 8] DO WriteChar[' ] ENDLOOP;
    RETURN
    END;

  END.
```