```
-- file SymbolControl.mesa; edited by Johnsson on July 25, 1978  5:01 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  CompilerDefs: FROM "compilerdefs",
  CopierDefs: FROM "copierdefs",
  CPass1: FROM "cpass1",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  SymbolCompressorDefs: FROM "symbolcompressordefs",
  SymbolTable: FROM "symboltable",
  SymbolTableDefs: FROM "symboltabledefs",
  SymDefs: FROM "symdefs",
  SymSegDefs: FROM "symsegdefs",
  SymTabDefs: FROM "symtabdefs",
  SystemDefs: FROM "systemdefs",
  TableDefs: FROM "tabledefs",
  TreeDefs: FROM "treedefs";

SymbolControl: PROGRAM
  IMPORTS CPass1, CopierDefs, SegmentDefs, StreamDefs, SymbolTable, SymbolTableDefs, SymTabDefs, System
**Defs, TableDefs
  EXPORTS SymbolCompressorDefs, SymSegDefs, TreeDefs =
  BEGIN

  StreamHandle: TYPE = StreamDefs.StreamHandle;
  FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
  FileHandle: TYPE = SegmentDefs.FileHandle;
  TableBase: TYPE = TableDefs.TableBase;
  PageSize: CARDINAL = AltoDefs.PageSize;


  EnterExtension: PUBLIC PROCEDURE [sei: SymDefs.ISEIndex, t: TreeDefs.TreeLink] =
    BEGIN
    IF t # TreeDefs.empty THEN ERROR;
    END;

  FindExtension: PUBLIC PROCEDURE [sei: SymDefs.ISEIndex] RETURNS [TreeDefs.TreeLink] =
    BEGIN
    RETURN[TreeDefs.empty]
    END;

  CopyTree: PUBLIC PROCEDURE [root: TreeDefs.TreeId, map: TreeDefs.TreeMap] RETURNS [TreeDefs.TreeLink]
** =
    BEGIN
    --IF root.link # TreeDefs.empty THEN ERROR;
    RETURN[TreeDefs.empty]
    END;

  CompressSymbols: PUBLIC PROCEDURE [modulename: STRING, sseg: FileSegmentHandle, stream: StreamHandle]
**
    RETURNS [pages: CARDINAL] =
    BEGIN OPEN SegmentDefs;
    tablepages: CARDINAL;
    table: POINTER;
    done: BOOLEAN ← FALSE;

    tablepages←sseg.pages;

    UNTIL done DO
      done ← TRUE;
      table ← SystemDefs.AllocateResidentPages[tablepages];
      TableDefs.InitializeTable[[LOOPHOLE[table], tablepages*PageSize], CompilerDefs.NTableDivisions];
      SymTabDefs.symtabinit[];
      CopierDefs.CopierInit[];
      CopierDefs.CreateFileTable[1];
      CondenseModuleInfo[modulename, sseg
        ! CopierDefs.TableRelocated => RESUME;
          InsufficientVM =>
            BEGIN
            done ← FALSE;
            tablepages ← tablepages - 1;
            CONTINUE
            END];
      CopierDefs.CopierReset[];
      SymTabDefs.symtaberase[];
```

```
    IF done THEN pages ← tableOut[stream];
    TableDefs.EraseTable[];
    SystemDefs.FreePages[table];
    ENDLOOP;
  END;

 -- tables defining the current symbol table

  seb: TableBase;  -- se table
  ctxb: TableBase;  -- context table
  mdb: TableBase;  -- module directory base
  bb: TableBase;  -- body table

condenseNotify: TableDefs.TableNotifier =
  BEGIN OPEN SymDefs;  -- called whenever the main symbol table is repacked
    seb ← base[setype];
    ctxb ← base[ctxtype];
    mdb ← base[mdtype];
    bb ← base[bodytype];
  RETURN
  END;

CondenseModuleInfo: PROCEDURE [modulename: STRING, seg: FileSegmentHandle] =
  BEGIN OPEN SymDefs, SymbolTableDefs, CopierDefs;
  mdi: MDIndex = FIRST[MDIndex];
  table: SymbolTableHandle = TableForSegment[seg];
  ibase: SymbolTableBase ← AcquireSymbolTable[table];
  ibb, iseb, ictxb: TableBase;
  sei: SEIndex;
  isei: ISEIndex;
  ibti, vbti: BTIndex;
  rootbti, prevbti: BTIndex;

  CPass1.P1Unit[];  -- prefills symbol table
  TableDefs.AddNotify[condenseNotify];
  FilePackInit[modulename,ibase.stHandle.version];
  (mdb+mdi).mdFile ← MakeFileTableEntry[seg.file, table];
  compressedHeader ← ibase.stHandle↑;
  ibb ← ibase.bb;
  iseb ← ibase.seb;  ictxb ← ibase.ctxb;
  IF ~OpenIncludedTable[mdi] THEN ERROR;
  compressedHeader.directoryCtx ← FindIncludedCtx[mdi, ibase.stHandle.directoryCtx];
  compressedHeader.outerCtx ← FindIncludedCtx[mdi, ibase.stHandle.outerCtx];

  -- copy all the bodies first
  rootbti ← prevbti ← BTNull;
  ibti ← FIRST[BTIndex];
  UNTIL ibti = LOOPHOLE[ibase.stHandle.bodyBlock.size, BTIndex] DO
    WITH cbti: ibb+ibti SELECT FROM
      Callable =>
        BEGIN
        isei ← cbti.id;
        IF isei # SENull THEN
          BEGIN
          sei ← CopyIncludedSymbol[isei, mdi];
          WITH (seb+sei) SELECT FROM
            id => vbti ← idinfo;
            ENDCASE => ERROR;
          (bb+vbti).localCtx ←
            IF (ibb+ibti).localCtx = ibase.stHandle.outerCtx THEN
              compressedHeader.outerCtx
            ELSE CTXNull;
          IF rootbti = BTNull THEN
            BEGIN
            rootbti ← vbti;
            (bb+rootbti).link ← [parent,BTNull];
            (bb+rootbti).firstSon ← BTNull;
            END
          ELSE IF prevbti = BTNull THEN
            BEGIN
            prevbti ← vbti;
            (bb+prevbti).link ← [parent,rootbti];
            (bb+prevbti).firstSon ← BTNull;
            (bb+rootbti).firstSon ← vbti;
            END
          ELSE
```

```
            BEGIN
            (bb+vbti).link ← (bb+prevbti).link;
            (bb+prevbti).link ← [sibling,vbti];
            (bb+vbti).firstSon ← BTNull;
            prevbti ← vbti;
            END
          END;
        ibti ← ibti +
          (WITH cbti SELECT FROM
            Outer => SIZE[Outer Callable BodyRecord],
            ENDCASE => SIZE[Inner Callable BodyRecord]);
        END;
      ENDCASE => ibti ← ibti + SIZE[Other BodyRecord];
    ENDLOOP;

  -- now copy other PUBLICs and transfer types
  FOR isei ← (ictxb+ibase.stHandle.outerCtx).selist, ibase.NextSe[isei] UNTIL isei = SENull DO
    IF (iseb+isei).public OR
      (SELECT ibase.XferMode[(iseb+isei).idtype] FROM
        procedure, signal, error, program => TRUE,
        ENDCASE => FALSE) THEN
      sei ← CopyIncludedSymbol[isei, mdi];
    ENDLOOP;

  SymbolTableDefs.ReleaseSymbolTable[ibase];
  CloseIncludedTable[];
  FilePackReset[];
  RETURN
  END;

WriteObjectWords: PROCEDURE [s: StreamHandle, addr: POINTER, n: CARDINAL] =
  BEGIN
  IF StreamDefs.WriteBlock[s, addr, n] # n THEN ERROR;
  RETURN
  END;


compressedHeader: SymDefs.STHeader;

tableOut: PROCEDURE [s: StreamHandle] RETURNS [pages: CARDINAL]=
  BEGIN
  OPEN SymSegDefs;
  d: CARDINAL;
    BEGIN
    OPEN TableDefs, compressedHeader;
    importCtx ← SymDefs.CTXNull;
    d ← SIZE[SymDefs.STHeader];
    hvBlock.offset ← d;
      d ← d + (hvBlock.size ← SymTabDefs.hashblock[].length);
    htBlock.offset ← d;  d ← d + (htBlock.size ← TableBounds[httype].size);
    ssBlock.offset ← d;  d ← d + (ssBlock.size ← TableBounds[sstype].size);
    seBlock.offset ← d;  d ← d + (seBlock.size ← TableBounds[setype].size);
    ctxBlock.offset ← d;
      d ← d + (ctxBlock.size ← TableBounds[ctxtype].size);
    mdBlock.offset ← d;  d ← d + (mdBlock.size ← TableBounds[mdtype].size);
    bodyBlock.offset ← d;
      d ← d + (bodyBlock.size ← TableBounds[bodytype].size);
    treeBlock ← litBlock ← extBlock ← [d, 0];
    fgRelPgBase ← fgPgCount ← 0;
    END;
  WriteObjectWords[s, @compressedHeader, SIZE[SymDefs.STHeader]];
  WriteObjectWords[
    s, SymTabDefs.hashblock[].base, compressedHeader.hvBlock.size];
  WriteSubTable[s, httype];
  WriteSubTable[s, sstype];
  WriteSubTable[s, setype];
  WriteSubTable[s, ctxtype];
  WriteSubTable[s, mdtype];
  WriteSubTable[s, bodytype];
  pages ← (d+(PageSize-1))/PageSize;
  RETURN
  END;

WriteSubTable: PROCEDURE [s: StreamHandle, table: TableDefs.TableSelector] =
  BEGIN
  base: TableBase;
```

```
    size: CARDINAL;
    [base, size] ← TableDefs.TableBounds[table];
    WriteObjectWords[s, LOOPHOLE[base], size];
    RETURN
    END;

  START SymbolTable;

  END...
```