```
-- file ObjectOut.Mesa
-- last modified by Satterthwaite, May 17, 1978  9:49 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [CharsPerWord, PageSize],
  BcdDefs: FROM "bcddefs" USING [VersionStamp, SGRecord, FTNull],
  ComData: FROM "comdata"
    USING [
      compilerVersion, codeSeg, defBodyLimit, definitionsOnly, fgTable,
      fixupLoc, importCtx, mainCtx, moduleCtx, mtRoot, MTRootSize,
      netNumber, objectVersion, rootFile, sourceFile, sourceVersion, symSeg],
  CompilerDefs: FROM "compilerdefs",
  LitDefs: FROM "litdefs" USING [CopyLiteral, ForgetLiterals],
  OsStaticDefs: FROM "osstaticdefs" USING [OsStatics],
  SegmentDefs: FROM "segmentdefs"
    USING [FileHandle, Append, DefaultVersion, Write, NewFile, SetFileAccess],
  StreamDefs: FROM "streamdefs"
    USING [
      StreamHandle, StreamIndex,
      CreateWordStream, GetIndex, NormalizeIndex, SetIndex, WriteBlock],
  StringDefs: FROM "stringdefs" USING [AppendString, WordsForString],
  SymDefs: FROM "symdefs"
    USING [fgHeader, FGTEntry, STHeader, WordOffset, VersionID, 1L],
  SymSegDefs: FROM "symsegdefs"
    USING [
      lttype, httype, sstype, setype, ctxtype, mdtype, bodytype, exttype,
      ExtRecord, ExtIndex],
  SymTabDefs: FROM "symtabdefs" USING [hashblock],
  SystemDefs: FROM "systemdefs" USING [FreeSegment],
  TableDefs: FROM "tabledefs"
    USING [
      TableBase, TableNotifier, TableSelector,
      AddNotify, DropNotify, TableBounds],
  TimeDefs: FROM "timedefs" USING [CurrentDayTime],
  TreeDefs: FROM "treedefs"
    USING [treetype,
      TreeIndex, TreeLink, TreeMap, empty, nullTreeIndex, TreeNodeSize,
      freetree, NodeSize, UpdateTree],
  XrefJournalDefs: FROM "xrefjournaldefs" USING [VersionID, XRJHeader];

ObjectOut: PROGRAM
    IMPORTS
      LitDefs, SegmentDefs, StreamDefs, StringDefs, SymTabDefs, SystemDefs,
      TableDefs, TimeDefs, TreeDefs,
      dataPtr: ComData
    EXPORTS CompilerDefs SHARES StringDefs =
  BEGIN

  stream: StreamDefs.StreamHandle;

  PageSize: CARDINAL = AltoDefs.PageSize;
  BytesPerWord: CARDINAL = AltoDefs.CharsPerWord;
  BytesPerPage: CARDINAL = PageSize*BytesPerWord;

  nextFilePage: PUBLIC PROCEDURE RETURNS [CARDINAL] =
    BEGIN  OPEN StreamDefs;
    fill: ARRAY [0..8) OF WORD ← [0, 0, 0, 0, 0, 0, 0, 0];
    m, n, r: INTEGER;
    r ← GetIndex[stream].byte/BytesPerWord;
    IF r # 0 THEN
      FOR n ← PageSize-r, n-m WHILE n > 0
        DO
        m ← MIN[n, LENGTH[fill]];
        [] ← WriteBlock[stream, BASE[fill], m];
        ENDLOOP;
    RETURN [GetIndex[stream].page + 1]
    END;

  WriteObjectWords: PROCEDURE [addr: POINTER, n: CARDINAL] =
    BEGIN
    [] ← StreamDefs.WriteBlock[stream, addr, n];
    RETURN
    END;

  RewriteObjectWords: PROCEDURE [index: StreamDefs.StreamIndex, addr: POINTER, n: CARDINAL] =
    BEGIN  OPEN StreamDefs;
```

```
    saveIndex: StreamIndex = GetIndex[stream];
    SetIndex[stream, index];
    [] ← WriteBlock[stream, addr, n];
    SetIndex[stream, saveIndex];
    RETURN
    END;


  -- bcd i/o

  bcdOffset: CARDINAL;
  bcdIndex: StreamDefs.StreamIndex;

  BCDIndex: PROCEDURE [offset: CARDINAL] RETURNS [StreamDefs.StreamIndex] =
    BEGIN  OPEN StreamDefs;
    byteOffset: CARDINAL = offset*BytesPerWord;
    RETURN [NormalizeIndex[StreamIndex[
        page: bcdIndex.page + byteOffset/BytesPerPage,
        byte: bcdIndex.byte + byteOffset MOD BytesPerPage]]]
    END;

  StartBCD: PUBLIC PROCEDURE =
    BEGIN
    [] ← nextFilePage[];
    bcdIndex ← StreamDefs.GetIndex[stream];
    bcdOffset ← 0;
    RETURN
    END;

  ReadBCDOffset: PUBLIC PROCEDURE RETURNS [CARDINAL] =
    BEGIN
    RETURN [bcdOffset];
    END;

  ReadBCDIndex: PUBLIC PROCEDURE RETURNS [StreamDefs.StreamIndex] =
    BEGIN
    RETURN [BCDIndex[bcdOffset]];
    END;

  AppendBCDWord: PUBLIC PROCEDURE [word: UNSPECIFIED] =
    BEGIN
    stream.put[stream, word];
    bcdOffset ← bcdOffset + 1;  RETURN
    END;

  AppendBCDWords: PUBLIC PROCEDURE [addr: POINTER, n: CARDINAL] =
    BEGIN
    WriteObjectWords[addr, n];
    bcdOffset ← bcdOffset + n;
    RETURN
    END;

  AppendBCDString: PUBLIC PROCEDURE [s: STRING] =
    BEGIN
    header: StringBody ← [length:s.length, maxlength:s.length, text:];
    AppendBCDWords[@header, SIZE[StringBody]];
    AppendBCDWords[@s.text, StringDefs.WordsForString[s.length] - SIZE[StringBody]];
    RETURN
    END;

  UpdateBCDWords: PUBLIC PROCEDURE [offset: CARDINAL, addr: POINTER, n: CARDINAL] =
    BEGIN
    RewriteObjectWords[BCDIndex[offset], addr, n];
    RETURN
    END;

  EndBCD: PUBLIC PROCEDURE =
    BEGIN
    [] ← nextFilePage[];
    RETURN
    END;

  -- symbol table i/o

  SetObjectStamp: PUBLIC PROCEDURE =
    BEGIN
```

```
  dataPtr.sourceVersion ← [FALSE, 0, 0, [0, 0]];
  dataPtr.objectVersion ← BcdDefs.VersionStamp[
      zapped: FALSE,
      net: dataPtr.netNumber,
      host: OsStaticDefs.OsStatics.SerialNumber,
      time: TimeDefs.CurrentDayTime[]];
  RETURN
  END;


StartObjectFile: PUBLIC PROCEDURE [file: SegmentDefs.FileHandle] RETURNS [StreamDefs.StreamHandle] =
  BEGIN
  objectFile: STRING ← [40];
    BEGIN  OPEN SegmentDefs;
    IF file # NIL
      THEN  SetFileAccess[file, Write+Append]
      ELSE
        BEGIN
        StringDefs.AppendString[objectFile, dataPtr.rootFile];
        StringDefs.AppendString[objectFile, ".bcd"];
        file ← NewFile[objectFile, Write+Append, DefaultVersion];
        END;
    stream ← StreamDefs.CreateWordStream[file, Write+Append];
    END;
  RETURN [stream]
  END;


PageCount: PROCEDURE [words: CARDINAL] RETURNS [CARDINAL] =
  BEGIN
  RETURN [(words+(PageSize-1))/PageSize]
  END;

SetFgt: PROCEDURE [d: SymDefs.WordOffset, sourceFile: STRING]
    RETURNS [fgBase, fgPages: CARDINAL] =
  BEGIN
  np: CARDINAL = PageCount[d];
  dataPtr.symSeg.pages ← np;
  IF dataPtr.definitionsOnly
    THEN
      BEGIN  fgBase ← 0;
      dataPtr.symSeg.extraPages ← fgPages ← 0;
      dataPtr.codeSeg.file ← BcdDefs.FTNull;
      dataPtr.codeSeg.base ← dataPtr.codeSeg.pages ← 0;
      dataPtr.mtRoot.framesize ← 0;
      END
    ELSE
      BEGIN
      fgBase ← np;
      dataPtr.symSeg.extraPages ← fgPages ← PageCount[
          (StringDefs.WordsForString[sourceFile.length]-SIZE[StringBody]) +
          LENGTH[dataPtr.fgTable]*SIZE[SymDefs.FGTEntry] +
          SIZE[SymDefs.fgHeader]];
      END;
  dataPtr.codeSeg.class ← code;  dataPtr.codeSeg.extraPages ← 0;
  RETURN
  END;


WriteSubTable: PROCEDURE [table: TableDefs.TableSelector] =
  BEGIN  OPEN TableDefs;
  base: TableBase;
  size: CARDINAL;
  [base, size] ← TableBounds[table];
  WriteObjectWords[LOOPHOLE[base], size];
  RETURN
  END;


litBias: CARDINAL;

WriteExtension: PROCEDURE RETURNS [size: CARDINAL] =
  BEGIN
  OPEN SymSegDefs, TreeDefs;
  tb, ltb: TableDefs.TableBase;
  treeLoc: TreeIndex;
```

```
  OutputNotify: TableDefs.TableNotifier =
    BEGIN
    tb ← base[treetype];  ltb ← base[lttype];
    seb ← base[setype];   ctxb ← base[ctxtype];
    extb ← base[exttype];  RETURN
    END;

  OutputLiteral: PROCEDURE [t: literal TreeLink] RETURNS [TreeLink] =
    BEGIN  OPEN LitDefs;
    WITH t.info SELECT FROM
      word => index ← CopyLiteral[[baseP:@ltb, index:index]]-litBias;
      ENDCASE => ERROR;
    RETURN [t]
    END;

  SetEmpty: TreeMap =
    BEGIN
    RETURN [empty]
    END;

  OutputTree: TreeMap =
    BEGIN
    s: TreeLink;
    node: TreeIndex;
    nw: CARDINAL;
    WITH link: t SELECT FROM
      literal =>  v ← OutputLiteral[link];
      subtree =>
        IF (s ← UpdateTree[link, OutputTree]) = empty
          THEN  v ← empty
          ELSE
            WITH s SELECT FROM
              subtree =>
                BEGIN  node ← index;
                nw ← NodeSize[@tb, node];
                WriteObjectWords[@(tb+node)↑, nw];
                [] ← freetree[UpdateTree[s, SetEmpty]];
                v ← [subtree[index: treeLoc]];  treeLoc ← treeLoc + nw;
                END;
              ENDCASE =>  v ← s;
      ENDCASE => ERROR;       -- for now
    RETURN
    END;

  extb: TableDefs.TableBase;
  exti, extLimit: ExtIndex;
  seb, ctxb: TableDefs.TableBase;
  TableDefs.AddNotify[OutputNotify];
  WriteObjectWords[@(tb+nullTreeIndex)↑, TreeNodeSize];
  treeLoc ← FIRST[TreeIndex] + TreeNodeSize;
  [extb, LOOPHOLE[extLimit, CARDINAL]] ← TableDefs.TableBounds[exttype];
  FOR exti ← FIRST[ExtIndex], exti + SIZE[ExtRecord] UNTIL exti = extLimit
    DO
    (extb+exti).tree ←
      IF (ctxb+(seb+(extb+exti).sei).ctxnum).ctxlevel < SymDefs.1L
              THEN OutputTree[(extb+exti).tree]
              ELSE empty;
    ENDLOOP;
  TableDefs.DropNotify[OutputNotify];
  RETURN [LOOPHOLE[treeLoc]]
  END;


TableOut: PUBLIC PROCEDURE [sourceFile: STRING] =
  BEGIN
  OPEN TableDefs, SymSegDefs;
  header: SymDefs.STHeader;
  fixupLoc: StreamDefs.StreamIndex;
  d: SymDefs.WordOffset;
  nw: CARDINAL;
  fgheader: SymDefs.fgHeader;
  dataPtr.symSeg.class ← symbols;
  dataPtr.symSeg.base ← nextFilePage[];
    BEGIN
    OPEN header;
```

```
    versionIdent ← SymDefs.VersionID;
    version ← dataPtr.objectVersion;
    sourceVersion ← dataPtr.sourceVersion;
    creator ← dataPtr.compilerVersion;
    definitionsFile ← dataPtr.definitionsOnly;
    directoryCtx ← dataPtr.moduleCtx;
    importCtx ← dataPtr.importCtx;
    outerCtx ← dataPtr.mainCtx;
    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 + TableBounds[bodytype].size;
    bodyBlock.size ← dataPtr.defBodyLimit;
    END;
  IF TableBounds[exttype].size # 0
    THEN  fixupLoc ← StreamDefs.GetIndex[stream]
    ELSE
      BEGIN
      header.treeBlock ← header.litBlock ← header.extBlock ← [d, 0];
      [header.fgRelPgBase, header.fgPgCount] ← SetFgt[d, sourceFile];
      END;
  WriteObjectWords[@header, SIZE[SymDefs.STHeader]];
  WriteObjectWords[SymTabDefs.hashblock[].base, header.hvBlock.size];
  WriteSubTable[httype];
  WriteSubTable[sstype];
  WriteSubTable[setype];
  WriteSubTable[ctxtype];
  WriteSubTable[mdtype];
  WriteSubTable[bodytype];
  IF TableBounds[exttype].size # 0
    THEN
      BEGIN
      litBias ← LitDefs.ForgetLiterals[];
      header.treeBlock.offset ← d;
      header.treeBlock.size ← WriteExtension[];
      d ← d + header.treeBlock.size;
      header.litBlock.offset ← d;
      nw ← TableBounds[lttype].size - litBias;
      WriteObjectWords[LOOPHOLE[TableBounds[lttype].base+litBias], nw];
      d ← d + (header.litBlock.size ← nw);
      header.extBlock.offset ← d;
      d ← d + (header.extBlock.size ← TableBounds[exttype].size);
      WriteSubTable[exttype];
      [header.fgRelPgBase, header.fgPgCount] ← SetFgt[d, sourceFile];
      RewriteObjectWords[fixupLoc, @header, SIZE[SymDefs.STHeader]];
      END;
  IF ~dataPtr.definitionsOnly
    THEN
      BEGIN
      OPEN fgheader;
      [] ← nextFilePage[];
      nw ← StringDefs.WordsForString[sourceFile.length]-SIZE[StringBody];
      fgoffset ← SIZE[SymDefs.fgHeader] + nw;
      fglength ← LENGTH[dataPtr.fgTable];
      sourcefile ← StringBody[
                      length: sourceFile.length,
                      maxlength: sourceFile.length,
                      text: -- written separately -- ];
      WriteObjectWords[@fgheader, SIZE[SymDefs.fgHeader]];
      WriteObjectWords[@sourceFile.text, nw];
      WriteObjectWords[BASE[dataPtr.fgTable], LENGTH[dataPtr.fgTable]*SIZE[SymDefs.FGTEntry]];
      SystemDefs.FreeSegment[BASE[dataPtr.fgTable]];
      END;
  RETURN
  END;

EndObjectFile: PUBLIC PROCEDURE [success: BOOLEAN] =
  BEGIN  OPEN StreamDefs;
  saveIndex: StreamIndex = GetIndex[stream];
  zero: CARDINAL ← 0;
```

```
    IF ~success
      THEN
        BEGIN  -- invalidate bcd
        SetIndex[stream, [0, 0]];  [] ← WriteBlock[stream, @zero, 1];
        END;
    SetIndex[stream, dataPtr.fixupLoc];
    [] ← WriteBlock[stream, @dataPtr.codeSeg, SIZE[BcdDefs.SGRecord]];
    [] ← WriteBlock[stream, @dataPtr.symSeg, SIZE[BcdDefs.SGRecord]];
    [] ← WriteBlock[stream, @dataPtr.mtRoot, dataPtr.MTRootSize];
    SetIndex[stream, saveIndex];
    stream.destroy[stream];  RETURN
    END;

-- xref i/o

  xrefStream: StreamDefs.StreamHandle;

  OpenXrefJournal: PUBLIC PROCEDURE =
    BEGIN  OPEN StringDefs;
    fileName: STRING ← [40];
    header: XrefJournalDefs.XRJHeader;
    nwSource, nwObject: CARDINAL;
    fileName.length ← 0;
    AppendString[fileName, dataPtr.rootFile];  AppendString[fileName, ".XRJ"];
      BEGIN  OPEN StreamDefs, SegmentDefs;
      xrefStream ← CreateWordStream[
        NewFile[fileName, Write+Append, DefaultVersion],
        Write+Append];
      END;
    fileName.length ← 0;
    AppendString[fileName, dataPtr.rootFile];  AppendString[fileName, ".bcd"];
    nwSource ← WordsForString[dataPtr.sourceFile.length];
    nwObject ← WordsForString[fileName.length];
    header ← [
        xrefVersion: XrefJournalDefs.VersionID,
        sourceFile: LOOPHOLE[SIZE[XrefJournalDefs.XRJHeader]],
        objectFile: LOOPHOLE[SIZE[XrefJournalDefs.XRJHeader] + nwSource],
        dataOffset: SIZE[XrefJournalDefs.XRJHeader] + nwSource + nwObject,
        objectVersion: dataPtr.objectVersion];
    [] ← StreamDefs.WriteBlock[xrefStream, @header, SIZE[XrefJournalDefs.XRJHeader]];
    [] ← StreamDefs.WriteBlock[xrefStream, dataPtr.sourceFile, nwSource];
    [] ← StreamDefs.WriteBlock[xrefStream, fileName, nwObject];
    RETURN
    END;

  AppendXrefWords: PUBLIC PROCEDURE [addr: POINTER, n: CARDINAL] =
    BEGIN
    [] ← StreamDefs.WriteBlock[xrefStream, addr, n];
    RETURN
    END;

  CloseXrefJournal: PUBLIC PROCEDURE =
    BEGIN
    xrefStream.destroy[xrefStream];
    RETURN
    END;

  END.
```