```
-- file Control.Mesa
-- last modified by Sweet, July 20, 1978  2:30 PM

DIRECTORY
  AllocDefs: FROM "allocdefs" USING [MakeDataSegment],
  AltoDefs: FROM "altodefs" USING [CharsPerPage, PageSize],
  CompilerDefs: FROM "compilerdefs"
    USING [
      Pass1, Pass2, Pass3, Pass4, Code, PassIndex,
      P1Unit, P2Unit, P3Unit, P4Unit, P5module,
      CloseXrefJournal, EndObjectFile, NTableDivisions, OpenXrefJournal,
      PrintBodies, PrintSymbols, PrintTree, SetObjectStamp,
      StartObjectFile, TableOut],
  ComData: FROM "comdata"
    USING [
      compilerVersion, definitionsOnly, dStar, errorStream, linkCount,
      netNumber, nErrors, nWarnings, objectBytes, objectFrameSize,
      objectStream, objectVersion, ownSymbols, rootFile, sort,
      sourceFile, sourceStream, warnings, xref],
  ControlDefs: FROM "controldefs" USING [GlobalFrameHandle],
  CopierDefs: FROM "copierdefs" USING [FilePackInit, FilePackReset, OwnFile],
  DisplayDefs: FROM "displaydefs" USING [DisplayOff, DisplayOn, InitDisplay],
  ErrorDefs: FROM "errordefs" USING [error],
  ErrorTabDefs: FROM "errortabdefs" USING [CSRptr],
  FontDefs: FROM "Fontdefs" USING [CreateFont],
  FrameDefs: FROM "framedefs" USING [SwapOutCode],
  ImageDefs: FROM "imagedefs"
    USING [
      CleanupItem, CleanupMask, CleanupProcedure, FileRequest,
      AddCleanupProcedure, AddFileRequest, ImageVersion,
      AbortMesa, RunImage, StopMesa],
  InlineDefs: FROM "inlinedefs" USING [DIVMOD],
  IODefs: FROM "iodefs"
    USING [
      ControlD, CR, SP, NumberFormat, ReadChar, ReadID, Rubout, WriteChar,
      WriteDecimal,  WriteLine, WriteNumber, WriteOctal, WriteString],
  LitDefs: FROM "litdefs" USING [LitTabErase, LitTabInit],
  MiscDefs: FROM "miscdefs" USING [CallDebugger, GetNetworkNumber],
  OsStaticDefs: FROM "osstaticdefs" USING [OsStatics],
  SegmentDefs: FROM "segmentdefs"
    USING [
      FileHandle, DataSegmentHandle, FileSegmentHandle, PageCount,
      Append, Read, Write, DefaultBase, DefaultPages, DefaultVersion,
      CopyDataToFileSegment, DeleteDataSegment, LockFile, MoveFileSegment,
      NewFile, NewFileSegment, SegmentAddress, SegmentFault, SetEndOfFile,
      SwapError, SwapIn, SwapOut, Unlock],
  StreamDefs: FROM "streamdefs"
    USING [
      StreamHandle, StreamIndex, Append, Read, Write,
      CloseDiskStream, CreateByteStream, GetDefaultDisplayStream, GetIndex,
      ModifyIndex, NewByteStream, OpenDiskStream, SetIndex],
  StringDefs: FROM "stringdefs"
    USING [
      AppendChar, AppendString, EquivalentString,
      MesaToBcplString, WordsForBcplString],
  SystemDefs: FROM "systemdefs"
    USING [AllocateHeapNode, AllocateHeapString, FreeHeapString, PruneHeap],
  SymbolTable: FROM "symboltable",
  SymbolTableDefs: FROM "symboltabledefs"
    USING [RestartSymbolCache, SuspendSymbolCache],
  SymTabDefs: FROM "symtabdefs" USING [symtaberase, symtabinit],
  TableDefs: FROM "tabledefs"
    USING [Region, EraseTable, InitializeTable, TableFailure, TableOverflow],
  TimeDefs: FROM "timedefs" USING [DefaultTime, AppendDayTime, UnpackDT],
  TrapDefs: FROM "trapdefs" USING [SendMsgSignal],
  TreeDefs: FROM "treedefs" USING [TreeLink, mlpop, TreeErase, TreeInit];

Control: PROGRAM [
        creationTime: STRING,
        creator: ControlDefs.GlobalFrameHandle,
        parseTableSeg: SegmentDefs.FileSegmentHandle,
        stringTableSeg: SegmentDefs.FileSegmentHandle]
    IMPORTS
        AllocDefs, CompilerDefs, CopierDefs, DisplayDefs, ErrorDefs, FontDefs,
        FrameDefs, ImageDefs, IODefs, LitDefs, MiscDefs, SegmentDefs,
        StreamDefs, StringDefs, SymbolTableDefs, SymTabDefs, SystemDefs,
```

```
      TableDefs, TimeDefs, TrapDefs, TreeDefs,
      ownSymbols: SymbolTable, dataPtr: ComData
  EXPORTS CompilerDefs SHARES ControlDefs, ImageDefs =
BEGIN
OPEN StreamDefs;

-- overlay control

PassIndex: TYPE = CompilerDefs.PassIndex;
GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;

PassLink: TYPE = RECORD [
  frame: GlobalFrameHandle,
  link: POINTER TO PassLink];

passRoot: ARRAY PassIndex OF POINTER TO PassLink;

LoadPass: PROCEDURE [pass: PassIndex] =
  BEGIN
  p: POINTER TO PassLink;
  IF ~initialized THEN InitializeSwapping[];
  FOR p ← passRoot[pass], p.link UNTIL p = NIL
    DO  OPEN SegmentDefs;
    -- Don't use SwapInCode, it will mess up Start Traps
    SwapIn[p.frame.codesegment];
    Unlock[p.frame.codesegment];
    ENDLOOP;
  RETURN
  END;

UnloadPass: PROCEDURE [pass: PassIndex] =
  BEGIN
  p: POINTER TO PassLink;
  IF ~initialized THEN InitializeSwapping[];
  FOR p ← passRoot[pass], p.link UNTIL p = NIL
    DO
    FrameDefs.SwapOutCode[p.frame !SegmentDefs.SwapError => CONTINUE];
    ENDLOOP;
  [] ← SystemDefs.PruneHeap[];
  RETURN
  END;

initialized: BOOLEAN ← FALSE;

InitializeSwapping: PROCEDURE =
  BEGIN
  i: PassIndex;
  FOR i IN PassIndex DO passRoot[i] ← NIL ENDLOOP;
  initialized ← TRUE;  RETURN
  END;

MakeSwappable: PUBLIC PROCEDURE [module: PROGRAM, pass: PassIndex] =
  BEGIN
  frame: GlobalFrameHandle = LOOPHOLE[module];
  p: POINTER TO PassLink = SystemDefs.AllocateHeapNode[SIZE[PassLink]];
  q: POINTER TO PassLink;
  IF ~initialized THEN InitializeSwapping[];
  p↑ ← PassLink[frame:frame, link:NIL];
  IF passRoot[pass] = NIL
    THEN passRoot[pass] ← p
    ELSE
      BEGIN  q ← passRoot[pass];
      UNTIL q.link = NIL DO q ← q.link ENDLOOP;
      q.link ← p;
      END;
  RETURN
  END;


-- command line input control

commandStream: StreamHandle;
comCmRequest: short ImageDefs.FileRequest ← [
  body: short[fill:, name: "Com.Cm."],
  file: NIL,
  access: Read,
```

```
    link: ];

  SetCommandInput: PROCEDURE =
    BEGIN
    c: CHARACTER;
    IF comCmRequest.file = NIL
      THEN commandStream ← NIL
      ELSE
        BEGIN OPEN IODefs;
        SegmentDefs.LockFile[comCmRequest.file];
        commandStream ← CreateByteStream[comCmRequest.file, Read];
        [] ← SkipStreamBlanks[];
        UNTIL commandStream.endof[commandStream] OR
          (c←commandStream.get[commandStream]) = SP OR c = CR DO NULL ENDLOOP;
        [] ← SkipStreamBlanks[];
        IF commandStream.endof[commandStream] THEN
          BEGIN commandStream.destroy[commandStream]; commandStream ← NIL; END
        ELSE SetIndex[commandStream,ModifyIndex[GetIndex[commandStream],-1]];
        END;
    RETURN
    END;

  SkipStreamBlanks: PROCEDURE RETURNS [c: CHARACTER] =
    BEGIN OPEN IODefs;
    UNTIL commandStream.endof[commandStream]
      DO
      c ← commandStream.get[commandStream];
      IF c # SP AND c # CR THEN EXIT;
      ENDLOOP;
    END;

  CommandLineID: PROCEDURE [s: STRING] =
    BEGIN  OPEN IODefs;
    c: CHARACTER;
    s.length ← 0;
    c ← SkipStreamBlanks[];
    IF commandStream.endof[commandStream] THEN RETURN;
    UNTIL c = SP OR c = CR DO
      StringDefs.AppendChar[s, c];  WriteChar[c];
      IF commandStream.endof[commandStream] THEN EXIT;
      c ← commandStream.get[commandStream];
      ENDLOOP;
    RETURN
    END;


-- special output stream control

  ds: StreamHandle;
  displayPut: PROCEDURE [StreamHandle, CHARACTER];

  typescript: StreamHandle;
  mtName: STRING = "Mesa.TypeScript.";
  mesaTSRequest: short ImageDefs.FileRequest ← [
    body: short[fill:, name: mtName],
    file: NIL,
    access: SegmentDefs.Write+SegmentDefs.Append,
    link: ];

  SetTypescript: PROCEDURE =
    BEGIN
    IF commandStream = NIL
      THEN  typescript ← NIL
      ELSE
        BEGIN OPEN SegmentDefs;
        IF mesaTSRequest.file = NIL
          THEN mesaTSRequest.file ← NewFile[mtName, Write+Append, DefaultVersion];
        typescript ← CreateByteStream[mesaTSRequest.file, Write+Append];
        displayPut ← ds.put;  ds.put ← SplitPut;
        END;
    RETURN
    END;


  SplitPut: PROCEDURE [s: StreamHandle, c: CHARACTER] =
    BEGIN
```

```
    typescript.put[typescript, c];  displayPut[s, c];  RETURN
    END;


  savedPut: PROCEDURE [StreamHandle, CHARACTER];

  ErrlogPut: PROCEDURE [s: StreamHandle, c: CHARACTER] =
    BEGIN
    errorFile: STRING;
    IF dataPtr.errorStream = NIL
      THEN
        BEGIN
        errorFile ←
          SystemDefs.AllocateHeapString[dataPtr.rootFile.length + (".errlog"L).length];
        StringDefs.AppendString[errorFile, dataPtr.rootFile];
        StringDefs.AppendString[errorFile, ".errlog"L];
        dataPtr.errorStream ← NewByteStream[errorFile, Write+Append];
        WriteHerald[errorFile];
        SystemDefs.FreeHeapString[errorFile];
        END;
    dataPtr.errorStream.put[dataPtr.errorStream, c];
    RETURN
    END;

  WriteHerald: PROCEDURE [id: STRING] =
    BEGIN OPEN TimeDefs, IODefs;
    now: STRING ← [20];
    WriteString ["Alto/Mesa Compiler 4.1 of "L];
    IF creationTime # NIL THEN WriteString[creationTime];
    WriteChar[CR];
    AppendDayTime[now, UnpackDT[DefaultTime]];
    IF id # NIL THEN BEGIN WriteString[id]; WriteString[" -- "L] END;
    WriteLine[now];
    RETURN
    END;

  WriteTime: PROCEDURE [sec: CARDINAL] =
    BEGIN  OPEN IODefs;
    hr, min: CARDINAL;
    f: NumberFormat ← [base:10, unsigned:TRUE, zerofill:FALSE, columns:1];

    W: PROCEDURE [t: CARDINAL] =
      BEGIN
      IF t # 0 OR f.zerofill THEN
        BEGIN
        WriteNumber[t, f];  WriteChar[':];
        f ← [base:10, unsigned:TRUE, zerofill:TRUE, columns:2];
        END;
      END;

    [min, sec] ← InlineDefs.DIVMOD[sec, 60];
    [hr, min] ← InlineDefs.DIVMOD[min, 60];
    W[hr];  W[min];  WriteNumber[sec, f];
    END;

  WriteClosing: PROCEDURE =
    BEGIN OPEN IODefs;
    WriteChar[CR];  WriteString[sourceFile];  WriteString[" -- "L];
    IF aborted
      THEN
        BEGIN  errors ← TRUE;
        WriteString["aborted, "L];
        WriteDecimal[dataPtr.nErrors];  WriteString[" errors "L];
        IF dataPtr.nWarnings # 0
          THEN
            BEGIN  WriteString["and "L];
            WriteDecimal[dataPtr.nWarnings];  WriteString[" warnings "L];
            END;
        WriteString["on "L];
        WriteString[dataPtr.rootFile];  WriteString[".errlog"L];
        END
      ELSE
        BEGIN
        WriteString["source chars: "L];
          WriteNumber[sourceChars,
            [base:10,zerofill:FALSE,unsigned:TRUE,columns:1]];
```

```
        WriteString[", time: "L];
        WriteTime[secondsClock.low-moduleStartTime];
        IF ~dataPtr.definitionsOnly
          THEN
            BEGIN
            WriteChar[CR];
            WriteString["  code bytes: "L]; WriteDecimal[dataPtr.objectBytes];
            WriteString[", links: "L]; WriteDecimal[dataPtr.linkCount];
            WriteString[", frame size: "L];
            WriteDecimal[dataPtr.objectFrameSize];
            END;
        IF dataPtr.nWarnings # 0 THEN
          BEGIN  warnings ← TRUE;  WriteChar[CR];
          WriteDecimal[dataPtr.nWarnings];  WriteString[" warnings on "L];
          WriteString[dataPtr.rootFile];  WriteString[".errlog"L];
          END;
        END;
    RETURN
    END;

  sourceChars: CARDINAL;

  SaveSourceLength: PROCEDURE =
    BEGIN OPEN StreamDefs;
    i: StreamIndex = GetIndex[dataPtr.sourceStream];
    sourceChars ← i.page*AltoDefs.CharsPerPage+i.byte;
    RETURN
    END;

-- display control

  sysFontRequest: short ImageDefs.FileRequest ← [
    body: short[fill:, name: "SysFont.al."],
    file: NIL,
    access: Read,
    link: ];

-- cursor control

  Cursor: TYPE = MACHINE DEPENDENT RECORD [
    top: PRIVATE CursorRow,
    row1: CursorRow,
    m12: PRIVATE CursorFill,
    row2: CursorRow,
    m23: PRIVATE CursorFill,
    row3: CursorRow,
    bottom: PRIVATE CursorRow];

  TheCursor: POINTER TO Cursor = LOOPHOLE[431B];
  savedCursor: Cursor;

  CursorRow: TYPE = ARRAY [0..2) OF WORD;
  CursorFill: TYPE = ARRAY [0..3) OF WORD;

  Two: CursorRow = [147763B, 147763B];
  L1: CursorRow = [147777B, 147777B];
  R1: CursorRow = [177763B, 177763B];
  M1: CursorRow = [177177B, 177177B];

  ClearCursor: PROCEDURE =
    BEGIN
    i: CARDINAL;
    FOR i IN [431B .. 431B+16) DO MEMORY[i] ← -1 ENDLOOP;
    RETURN
    END;


-- cleanup of files/streams

  compilerCleanupItem: ImageDefs.CleanupItem ← [
      proc: CompilerCleanup,
      mask: ImageDefs.CleanupMask[InLd] + ImageDefs.CleanupMask[OutLd],
      link: ];

  CompilerCleanup: ImageDefs.CleanupProcedure =
    BEGIN
```

```
    SELECT why FROM
      OutLd =>
        IF dataPtr.errorStream # NIL
          THEN CloseDiskStream[dataPtr.errorStream];
      InLd =>
        IF dataPtr.errorStream # NIL
          THEN OpenDiskStream[dataPtr.errorStream];
      ENDCASE;
    RETURN
    END;


  -- table storage management

   tableRequest: short ImageDefs.FileRequest ← [
     body: short[fill:, name: "Swatee."],
     file: NIL,
     access: Read+Write+Append,
     link: ];

   PageCount: TYPE = SegmentDefs.PageCount;

   TablePageStart: PageCount = 64;
   TablePageStep:  PageCount = 8;
   TablePageLimit: PageCount = 96;

   tableDataSegment: SegmentDefs.DataSegmentHandle;
   tableFileSegment: SegmentDefs.FileSegmentHandle;
   tablePages: PageCount;

   tableRegion: TableDefs.Region;

   LoadTable: PROCEDURE [nPages: PageCount] =
     BEGIN
     OPEN SegmentDefs;
     IF nPages # tablePages
       THEN
         BEGIN
         IF tableFileSegment = NIL
           THEN
             BEGIN
             IF tableRequest.file = NIL THEN
               tableRequest.file ←
                 NewFile["Swatee.", Read+Write+Append, DefaultVersion];
             tableFileSegment ← NewFileSegment[
                 file: tableRequest.file,
                 base: DefaultBase,
                 pages: tablePages,
                 access: Read+Write];
             CopyDataToFileSegment[tableDataSegment, tableFileSegment
               ! SegmentFault =>
                 BEGIN
                 SetEndOfFile[tableRequest.file, tableFileSegment.base+nPages, 0];
                 RETRY
                 END];
             DeleteDataSegment[tableDataSegment];  tableDataSegment ← NIL;
             END
           ELSE
             BEGIN  Unlock[tableFileSegment];  SwapOut[tableFileSegment]  END;
         MoveFileSegment[tableFileSegment, DefaultBase, nPages];
         tablePages ← nPages;
         SwapIn[tableFileSegment ! SegmentFault =>
           BEGIN
           SetEndOfFile[tableRequest.file, tableFileSegment.base+nPages, 0];
           RETRY
           END];
         END;
     tableRegion ← [
         origin: LOOPHOLE[SegmentAddress[IF tableDataSegment # NIL
                       THEN tableDataSegment ELSE tableFileSegment]],
         size: tablePages*AltoDefs.PageSize];
     RETURN
     END;


  -- string table management
```

```
  tableLocked: BOOLEAN ← FALSE;

  OpenStringTable: PUBLIC PROCEDURE RETURNS [ErrorTabDefs.CSRptr] =
    BEGIN
    IF tableLocked THEN ERROR;
    SegmentDefs.SwapIn[stringTableSeg];
    tableLocked ← TRUE;
    RETURN [LOOPHOLE[SegmentDefs.SegmentAddress[stringTableSeg]]];
    END;

  CloseStringTable: PUBLIC PROCEDURE =
    BEGIN
    IF tableLocked
      THEN
        BEGIN
        SegmentDefs.Unlock[stringTableSeg];  tableLocked ← FALSE;
        END;
    RETURN
    END;


-- compiler sequencing

  Initialize: PROCEDURE =
    BEGIN
    dataPtr.errorStream ← dataPtr.objectStream ← NIL;
    dataPtr.sourceStream.reset[dataPtr.sourceStream];
    savedPut ← ds.put;  ds.put ← ErrlogPut;
    IF typescript # NIL THEN CloseDiskStream[typescript];
    IF commandStream # NIL THEN CloseDiskStream[commandStream];
    DisplayDefs.DisplayOff[black];
    savedCursor ← TheCursor↑;  ClearCursor[];
    LoadTable[TablePageStart];
    TableDefs.InitializeTable[tableRegion, CompilerDefs.NTableDivisions];
    SymTabDefs.symtabinit[];  LitDefs.LitTabInit[];  TreeDefs.TreeInit[];
    RETURN
    END;

  Finalize: PROCEDURE =
    BEGIN
    IF dataPtr.objectStream # NIL
      THEN CompilerDefs.EndObjectFile[dataPtr.nErrors=0];
    TreeDefs.TreeErase[];  LitDefs.LitTabErase[];  SymTabDefs.symtaberase[];
    TableDefs.EraseTable[];
    TheCursor↑ ← savedCursor;
    DisplayDefs.DisplayOn[];
    IF commandStream # NIL THEN OpenDiskStream[commandStream];
    IF typescript # NIL THEN OpenDiskStream[typescript];
    ds.put ← savedPut;
    IF dataPtr.errorStream # NIL
      THEN dataPtr.errorStream.destroy[dataPtr.errorStream];
    IF pass # '1 THEN OpenDiskStream[dataPtr.sourceStream];
    dataPtr.sourceStream.destroy[dataPtr.sourceStream];
    RETURN
    END;

  StopCompiler: PROCEDURE =
    BEGIN
    IF moduleCount > 1 THEN
      BEGIN OPEN IODefs;
      WriteChar[CR];  WriteString["Total elapsed time: "L];
      WriteTime[secondsClock.low-compilerStartTime];
      WriteChar[CR];
      END;
    IF typescript # NIL
      THEN
        BEGIN  ds.put ← displayPut; typescript.destroy[typescript];
        IF (errors OR warnings) AND pauseForErrors
          THEN
            BEGIN  OPEN IODefs;
            WriteChar[CR];
            IF errors THEN
              BEGIN
              WriteString["Errors"L];
              IF warnings THEN WriteString[" and "L];
```

```
            END;
          IF warnings THEN WriteString["Warnings"L];
          WriteString[" logged; type any character to finish."L];
          [] ← ReadChar[];  WriteChar[CR];
          END;
      END;
  RETURN
  END;

debugPass: PassIndex ← debug;

Debug: PROCEDURE [tree, symbols: PROCEDURE] =
  BEGIN
  LoadPass[debug];
  tree[];  symbols[];
  CloseStringTable[];  UnloadPass[debug];  RETURN
  END;

PrintTreeRoot: PROCEDURE = BEGIN CompilerDefs.PrintTree[root]; RETURN END;


sourceFile: STRING ← [40];
sortDefault, warningsDefault, xrefDefault, dStarDefault: BOOLEAN;
debugDefault: BOOLEAN;
debugFlag, pauseForErrors, localPause: BOOLEAN;
pass: CHARACTER ['1..'5];
nParseErrors: CARDINAL;
parsed, aborted, errors, warnings: BOOLEAN;
moduleCount: CARDINAL ← 0;
moduleStartTime, compilerStartTime: CARDINAL;
secondsClock: POINTER TO MACHINE DEPENDENT RECORD [high, low: CARDINAL] =
  LOOPHOLE[572B];

objectFileHint: SegmentDefs.FileHandle;

msg, signal: UNSPECIFIED;

root: TreeDefs.TreeLink;

-- * * * * * *  M A I N   B O D Y   C O D E  * * * * * *

-- add cleanup procedure
ImageDefs.AddCleanupProcedure[@compilerCleanupItem];
ImageDefs.AddFileRequest[@comCmRequest];
ImageDefs.AddFileRequest[@mesaTSRequest];
ImageDefs.AddFileRequest[@sysFontRequest];
ImageDefs.AddFileRequest[@tableRequest];
STOP;  -- wait for restart

START dataPtr; -- initialize STRING variables, etc.
START ownSymbols; -- initialize STRING variables, etc.

dataPtr.sourceFile ← sourceFile;        -- shared string body
dataPtr.ownSymbols ← ownSymbols;
dataPtr.errorStream ← NIL;

-- set up swapping
BEGIN  OPEN CompilerDefs;
MakeSwappable[Pass1, pass1];  --START Pass1;  UnloadPass[pass1];
MakeSwappable[Pass2, pass2];  --START Pass2;  UnloadPass[pass2];
MakeSwappable[Pass3, pass3];  --START Pass3;  UnloadPass[pass3];
MakeSwappable[Pass4, pass4];  --START Pass4;  UnloadPass[pass4];
MakeSwappable[Code, pass5];   --START Code;   UnloadPass[pass5];
END;

compilerStartTime ← secondsClock.low;

-- start the display
BEGIN OPEN SegmentDefs;
IF sysFontRequest.file = NIL THEN ERROR;
DisplayDefs.InitDisplay[
  dummySize: 72,
  textLines: 6,
  nPages: 6,
  f: FontDefs.CreateFont[
      NewFileSegment[sysFontRequest.file, 1, DefaultPages, Read]]];
```

```
  END;

 -- consider deleting the compiler builder here

  dataPtr.compilerVersion ← ImageDefs.ImageVersion[];
  dataPtr.netNumber ← MiscDefs.GetNetworkNumber[];

 -- obtain the scratch area
  tableDataSegment ← AllocDefs.MakeDataSegment[
        base: SegmentDefs.DefaultBase,
        pages: TablePageStart,
        info: [0, easy, bottomup, initial, other, FALSE, FALSE]];
  tablePages ← TablePageStart;
  tableFileSegment ← NIL;

 -- do the compilation
  sortDefault ← warningsDefault ← TRUE;
  debugDefault ← xrefDefault ← dStarDefault ← FALSE;
  pauseForErrors ← TRUE;
  SetCommandInput[];
  ds ← GetDefaultDisplayStream[];
  SetTypescript[];  WriteHerald[NIL];  errors ← warnings ← FALSE;

    DO
    OPEN IODefs;
      BEGIN
      i, sourceLength: CARDINAL;
      sense, sourceExtension: BOOLEAN;

      CompleteFileName: PROCEDURE RETURNS [fileName: STRING, bcpl: BOOLEAN] =
        BEGIN  OPEN StringDefs;
        j: CARDINAL;
        extension: STRING ← [40];
        fileName ← SystemDefs.AllocateHeapString[40];
        AppendString[fileName, dataPtr.rootFile];
        IF ~sourceExtension
          THEN  AppendString[extension, "image"L]
          ELSE
            FOR j IN [dataPtr.rootFile.length+1 .. sourceLength)
              DO  AppendChar[extension, sourceFile[j]]  ENDLOOP;
        bcpl ← EquivalentString[extension, "run"L];
        AppendChar[fileName, '.];  AppendString[fileName, extension];
        RETURN
        END;

      WriteCommandFile: PROCEDURE [fileName: STRING] =
        BEGIN
        j: CARDINAL;
        copy: StreamHandle;
        copy ← CreateByteStream[comCmRequest.file, Write+Append];
        FOR j IN [0..fileName.length) DO copy.put[copy, fileName[j]] ENDLOOP;
        IF sourceFile.length > i+1
          THEN
            BEGIN
            copy.put[copy, '/];
            FOR j IN (i..sourceFile.length)
              DO copy.put[copy, sourceFile[j]] ENDLOOP;
            END;
        IF commandStream = NIL OR commandStream.endof[commandStream]
          THEN  copy.put[copy, CR]
          ELSE
            BEGIN
            copy.put[copy, ' ];
            UNTIL commandStream.endof[commandStream]
              DO  copy.put[copy, commandStream.get[commandStream]]  ENDLOOP;
            END;
        IF commandStream # NIL THEN commandStream.destroy[commandStream];
        copy.destroy[copy];
        RETURN
        END;

      Run: PROCEDURE [fileName: STRING, bcpl: BOOLEAN] =
        BEGIN
        IF bcpl
          THEN
            BEGIN
```

```
        p: POINTER = OsStaticDefs.OsStatics.EventVector;
        EVItem: TYPE = MACHINE DEPENDENT RECORD [
            type: [0..7777B], length: [0..17B]];
        p↑ ← EVItem[6, StringDefs.WordsForBcplString[fileName.length]+1];
        StringDefs.MesaToBcplString[fileName, p+1];
        ImageDefs.StopMesa[];
        END
      ELSE
        BEGIN
        OPEN SegmentDefs;
        ImageDefs.RunImage[NewFileSegment[
          file: NewFile[fileName, Read, DefaultVersion],
          base: 1,
          pages: 1,
          access: Read]];
        END;
    END;

  dataPtr.xref ← xrefDefault;  dataPtr.warnings ← warningsDefault;
  dataPtr.sort ← sortDefault;  dataPtr.dStar ← dStarDefault;
  debugFlag ← debugDefault; localPause ← FALSE;
  WriteChar[CR];  WriteString["Compile: "];
  IF commandStream # NIL
    THEN CommandLineID[sourceFile]
    ELSE ReadID[sourceFile !Rubout => GO TO abort];
  IF sourceFile.length = 0 THEN EXIT;
  IF sourceFile[0] = ControlD THEN GO TO debugger;        -- ↑D => debug
  moduleStartTime ← secondsClock.low;
  dataPtr.rootFile.length ← 0;  sourceExtension ← FALSE;
  FOR i IN [0..sourceFile.length)
    DO
    SELECT sourceFile[i] FROM
      '. => sourceExtension ← TRUE;
      '/ => GO TO Switches;
      ENDCASE;
    IF ~sourceExtension
      THEN StringDefs.AppendChar[dataPtr.rootFile, sourceFile[i]];
    REPEAT
      Switches =>
        BEGIN
        sourceLength ← i;  i ← i+1;  sense ← TRUE;
        WHILE i < sourceFile.length
          DO
          SELECT sourceFile[i] FROM
            '-, '~ => sense ← ~sense;
            'a, 'A => BEGIN dataPtr.dStar ← ~sense; sense ← TRUE END;
            'd, 'D => BEGIN debugFlag ← sense; sense ← TRUE END;
            'p, 'P => BEGIN localPause ← sense; sense ← TRUE END;
            'r, 'R =>
              BEGIN
              fileName: STRING;
              bcpl: BOOLEAN;
              [fileName, bcpl] ← CompleteFileName[];
              WriteCommandFile[fileName];
              StopCompiler[];
              Run[fileName, bcpl ! ANY => ImageDefs.AbortMesa];
              -- never returns
              END;
            's, 'S => BEGIN dataPtr.sort ← sense; sense ← TRUE END;
            'w, 'W => BEGIN dataPtr.warnings ← sense; sense ← TRUE END;
            'x, 'X => BEGIN dataPtr.xref ← sense; sense ← TRUE END;
            IN ['1..'5] =>
              BEGIN
              debugPass ← LOOPHOLE[sourceFile[i]-'0]; sense ← TRUE;
              END;
            'c, 'C =>
              BEGIN  sense ← TRUE;
              FOR i IN [0..sourceLength)
                DO
                SELECT sourceFile[i] FROM
                  '-, '~ => sense ← ~sense;
                  'a, 'A => BEGIN  dStarDefault ← ~sense;  EXIT  END;
                  'd, 'D => BEGIN  debugDefault ← sense;  EXIT  END;
                  'p, 'P => BEGIN  pauseForErrors ← sense;  EXIT  END;
                  's, 'S => BEGIN  sortDefault ← sense;  EXIT  END;
                  'w, 'W => BEGIN  warningsDefault ← sense;  EXIT  END;
```

```
                'x, 'X => BEGIN  xrefDefault ← sense;  EXIT  END;
                IN ['1..'5] =>
                  BEGIN
                  debugPass ← LOOPHOLE[sourceFile[i]-'0];  EXIT
                  END;
                ENDCASE => EXIT;
              ENDLOOP;
            GO TO skip
            END;
          ENDCASE;
        i ← i+1;
        ENDLOOP;
      END;
    FINISHED => sourceLength ← sourceFile.length;
  ENDLOOP;
sourceFile.length ← sourceLength;
IF ~sourceExtension THEN StringDefs.AppendString[sourceFile, ".mesa"];

moduleCount ← moduleCount + 1;
dataPtr.sourceStream ← NewByteStream[sourceFile, Read
  ! ANY => GO TO noSource];

dataPtr.nErrors ← dataPtr.nWarnings ← 0;  aborted ← FALSE;
Initialize[];

  BEGIN
  ENABLE
    BEGIN
    TableDefs.TableOverflow =>
      BEGIN
      IF tablePages >= TablePageLimit THEN GO TO outOfSpace;
      LoadTable[tablePages+TablePageStep];  RESUME[tableRegion]
      END;
    TableDefs.TableFailure =>  GO TO outOfSpace;
    UNWIND =>  Finalize[];
    ANY => IF ~debugFlag
      THEN
        BEGIN
        [msg, signal] ← SIGNAL TrapDefs.SendMsgSignal;
        GO TO uncaughtSignal
        END
    END;

 -- first pass
  pass ← '1;  LoadPass[pass1];
  TheCursor.row2 ← M1;
  parsed ← CompilerDefs.P1Unit[parseTableSeg];
  nParseErrors ← dataPtr.nErrors;
  ClearCursor[];  UnloadPass[pass1];
  IF ~parsed THEN GO TO failed;
  root ← TreeDefs.mlpop[];
  SaveSourceLength[];
  CloseDiskStream[dataPtr.sourceStream];
  IF debugPass <= pass1
    THEN Debug[PrintTreeRoot, CompilerDefs.PrintSymbols];

 -- second pass
  pass ← '2;  LoadPass[pass2];
  TheCursor.row1 ← L1; TheCursor.row3 ← R1;
  root ← CompilerDefs.P2Unit[root];
  ClearCursor[];  UnloadPass[pass2];
  IF debugPass <= pass2
    THEN Debug[PrintTreeRoot, CompilerDefs.PrintSymbols];
  IF dataPtr.nErrors # 0 THEN dataPtr.xref ← FALSE;

 -- third and fourth passes
  CompilerDefs.SetObjectStamp[];
  CopierDefs.FilePackInit[dataPtr.rootFile, dataPtr.objectVersion];
  objectFileHint ← NIL;
  BEGIN
    ENABLE
      BEGIN
      TableDefs.TableOverflow =>
        BEGIN
        IF tablePages >= TablePageLimit THEN GO TO noSpace;
        SymbolTableDefs.SuspendSymbolCache[];
```

```
            LoadTable[tablePages+TablePageStep];
            SymbolTableDefs.RestartSymbolCache[];
            RESUME[tableRegion]
            END;
          CopierDefs.OwnFile => BEGIN  objectFileHint ← file;  RESUME  END
          END;

        IF dataPtr.xref THEN CompilerDefs.OpenXrefJournal[];

        pass ← '3;  LoadPass[pass3];
        TheCursor.row1 ← R1; TheCursor.row2 ← M1; TheCursor.row3 ← L1;
        root ← CompilerDefs.P3Unit[root];
        ClearCursor[];  UnloadPass[pass3];

        IF dataPtr.xref THEN CompilerDefs.CloseXrefJournal[];
        IF debugPass <= pass3
          THEN Debug[PrintTreeRoot, CompilerDefs.PrintSymbols];
        IF dataPtr.nErrors > nParseErrors THEN GO TO DeleteFiles;

        dataPtr.objectStream ← CompilerDefs.StartObjectFile[objectFileHint];

        pass ← '4;  LoadPass[pass4];
        TheCursor.row1 ← TheCursor.row3 ← Two;
        CompilerDefs.P4Unit[root];
        ClearCursor[];  UnloadPass[pass4];
        IF debugPass <= pass4
          THEN Debug[CompilerDefs.PrintBodies, CompilerDefs.PrintSymbols];
        GO TO DeleteFiles;

        EXITS
          DeleteFiles => CopierDefs.FilePackReset[];
          noSpace =>
            BEGIN  CopierDefs.FilePackReset[];  GO TO outOfSpace  END;
        END;
        IF dataPtr.nErrors # 0 THEN GO TO failed;

      -- fifth pass
        IF ~dataPtr.definitionsOnly
          THEN
            BEGIN
            ENABLE  UNWIND =>  CompilerDefs.EndObjectFile[FALSE];
            pass ← '5;  LoadPass[pass5];
            TheCursor.row1 ← TheCursor.row3 ← Two; TheCursor.row2 ← M1;
            CompilerDefs.P5module[];
            ClearCursor[];  UnloadPass[pass5];
            END;

        TheCursor.row1 ← TheCursor.row2 ← TheCursor.row3 ← Two;
        CompilerDefs.TableOut[sourceFile];
        IF dataPtr.nErrors # 0 THEN GO TO failed;

        EXITS
          failed =>  BEGIN  aborted ← TRUE;  CloseStringTable[]  END;
          uncaughtSignal =>
            BEGIN  aborted ← TRUE;
            IF pass = '1
              THEN WriteString["Compiler Error "]
              ELSE ErrorDefs.error[compilerError];
            WriteString["Pass = "];  WriteChar[pass];
            WriteString[", signal = "];  WriteOctal[signal];
            WriteString[", message = "];  WriteOctal[msg];
            WriteChar[CR];
            END;
          outOfSpace =>
            BEGIN  aborted ← TRUE;
            WriteChar[CR];  WriteString["Storage Overflow in Pass "];
            WriteChar[pass];  WriteChar[CR];
            END;
        END;

      Finalize[];

      WriteClosing[];

      EXITS
        skip =>  NULL;
```

```
        abort => WriteString[" XXX"];
        noSource =>
          BEGIN errors ← TRUE; WriteString["  File error"] END;
        debugger => MiscDefs.CallDebugger[NIL];
      END;
    WriteChar[CR];
    IF (errors OR warnings) AND localPause THEN
      BEGIN pauseForErrors ← TRUE; GO TO truncateList END;
    REPEAT
      truncateList => NULL;
    ENDLOOP;

  StopCompiler[];

  END.
```