```
-- file OutCode.mesa
-- last modified by Sweet, July 14, 1978  3:25 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [Address, BYTE, PageSize],
  Code: FROM "code" USING [CodePassInconsistancy, codeptr],
  CodeDefs: FROM "codedefs" USING [CCIndex, CCItem, CCNull, ChunkBase, FreeChunk, NULLfileindex],
  ComData: FROM "comdata" USING [codeSeg, dStar, fgTable, linkCount, mainBody, mtRoot, nBodies, nSigCod
**es, objectFrameSize, objectStream, stopping],
  CompilerDefs: FROM "compilerdefs" USING [nextFilePage],
  ControlDefs: FROM "controldefs" USING [codebaseOffset, CSegPrefix, EntryVectorItem, EPRange, InstWord
**, MaxAllocSlot],
  FOpCodes: FROM "fopcodes" USING [qBLTC, qGADRB, qLADRB],
  InlineDefs: FROM "inlinedefs" USING [BITOR, BITSHIFT],
  LitDefs: FROM "litdefs" USING [EnumerateLocalStrings, EnumerateMasterStrings, MSTIndex, STIndex, stty
**pe],
  Mopcodes: FROM "mopcodes" USING [zJIB, zJIW, zNOOP],
  P5ADefs: FROM "p5adefs" USING [Ciout0, Ciout1, computeframesize, P5Error, RequireStack, wordsforsei],
**
  P5BDefs: FROM "p5bdefs" USING [C1W, pushlitval],
  StreamDefs: FROM "streamdefs" USING [GetIndex, SetIndex, StreamIndex, WriteBlock],
  StringDefs: FROM "stringdefs" USING [WordsForString],
  SymDefs: FROM "symdefs" USING [BodyInfo, bodytype, BTIndex, ByteIndex, CBTIndex, CSEIndex, CTXIndex,
**FGTEntry, HTIndex, ISEIndex, recordCSEIndex, SEIndex, SERecord, setype],
  SymTabDefs: FROM "symtabdefs" USING [UnderType],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, AllocatePages, AllocateSegment, FreeHeapNode,
**FreePages, FreeSegment],
  TableDefs: FROM "tabledefs" USING [TableBase, TableLimit, TableNotifier],
  TreeDefs: FROM "treedefs" USING [treetype];

DEFINITIONS FROM CodeDefs;

OutCode: PROGRAM
    IMPORTS MPtr: ComData, CPtr: Code, CodeDefs, CompilerDefs, LitDefs, P5ADefs, P5BDefs, StreamDefs, S
**tringDefs, SymTabDefs, SystemDefs
    EXPORTS CodeDefs, P5BDefs
    SHARES LitDefs =
  BEGIN
  OPEN P5ADefs, P5BDefs;


  -- imported definitions

  Address: TYPE = AltoDefs.Address;
  BYTE: TYPE = AltoDefs.BYTE;
  PageSize: INTEGER = AltoDefs.PageSize;
  InstWord: TYPE = ControlDefs.InstWord;
  codebaseOffset: CARDINAL = ControlDefs.codebaseOffset;

  BodyInfo: TYPE = SymDefs.BodyInfo;
  BTIndex: TYPE = SymDefs.BTIndex;
  CBTIndex: TYPE = SymDefs.CBTIndex;
  ByteIndex: TYPE = SymDefs.ByteIndex;
  CSEIndex: TYPE = SymDefs.CSEIndex;
  CTXIndex: TYPE = SymDefs.CTXIndex;
  FGTEntry: TYPE = SymDefs.FGTEntry;
  HTIndex: TYPE = SymDefs.HTIndex;
  ISEIndex: TYPE = SymDefs.ISEIndex;
  recordCSEIndex: TYPE = SymDefs.recordCSEIndex;
  SEIndex: TYPE = SymDefs.SEIndex;
  SERecord: TYPE = SymDefs.SERecord;

  STIndex: TYPE = LitDefs.STIndex;
  MSTIndex: TYPE = LitDefs.MSTIndex;


  cb: ChunkBase;                -- code base (local copy)
  seb: TableDefs.TableBase;
  bb: TableDefs.TableBase;
  stb: TableDefs.TableBase;

  OutCodeNotify: PUBLIC TableDefs.TableNotifier =
    BEGIN  -- called by allocator whenever table area is repacked
    cb ← LOOPHOLE[base[TreeDefs.treetype]];
    seb ← base[SymDefs.setype];  bb ← base[SymDefs.bodytype];
    stb ← base[LitDefs.sttype];
```

```
  RETURN
  END;

FileSequenceError: SIGNAL = CODE;

fgt: DESCRIPTOR FOR ARRAY OF FGTEntry;
fgti, fgtpages: CARDINAL;
codebase, entrybase: StreamDefs.StreamIndex;
entryvector: DESCRIPTOR FOR ARRAY OF ControlDefs.EntryVectorItem;

parity: {even, odd};
codeindex: CARDINAL;
buffer: InstWord;


startcodefile: PUBLIC PROCEDURE =
  BEGIN -- called to set up bodytable and init binary file header
  OPEN MPtr, ControlDefs, SystemDefs, StreamDefs;
  prefix: CSegPrefix;
  ngfi: CARDINAL = (MAX[nBodies, nSigCodes] + (EPRange-1))/EPRange;
  IF ngfi ~IN [1..4] THEN P5ADefs.P5Error[833];
  -- should be 256 (fix ControlDefs)
  IF linkCount > 377B THEN P5ADefs.P5Error[834];
  prefix ← [
      swapinfo: 0,
      stops: MPtr.stopping,
      fill: IF MPtr.dStar THEN 1 ELSE 0,
      ngfi: ngfi,
      nlinks: linkCount,
      entry: ];
  codeSeg.base ← CompilerDefs.nextFilePage[];
  fgti ← -1; fgtpages ← 1;
  codebase ← GetIndex[objectStream];
  [] ← WriteBlock[objectStream, @prefix, SIZE[CSegPrefix]];
  entrybase ← GetIndex[objectStream];
  codeindex ← SIZE[CSegPrefix]+nBodies*SIZE[EntryVectorItem];
  parity ← even;
  SetIndex[objectStream, StreamIndex[page: codebase.page, byte: 2*codeindex]];
  fgt ← DESCRIPTOR[AllocatePages[fgtpages], (fgtpages*PageSize)/SIZE[FGTEntry]];
  entryvector ← DESCRIPTOR[AllocateSegment[nBodies*SIZE[EntryVectorItem]], nBodies];
  RETURN
  END;

movetocodeword: PUBLIC PROCEDURE RETURNS [CARDINAL] =
  BEGIN
  IF parity = odd THEN
    BEGIN
    buffer.oddbyte ← 377B; MPtr.objectStream.put[MPtr.objectStream, buffer];
    parity ← even; codeindex ← codeindex+1;
    END;
  RETURN [codeindex]
  END;


writecodeword: PUBLIC PROCEDURE [w: WORD] =
  BEGIN
  IF parity # even THEN P5ADefs.P5Error[835];
  MPtr.objectStream.put[MPtr.objectStream, w];
  codeindex ← codeindex+1;
  RETURN
  END;


writecodebyte: PROCEDURE [b: BYTE] =
  BEGIN
  IF parity = odd THEN
    BEGIN
    buffer.oddbyte ← b; MPtr.objectStream.put[MPtr.objectStream, buffer];
    parity ← even; codeindex ← codeindex+1;
    END
  ELSE BEGIN buffer.evenbyte ← b; parity ← odd; END;
  RETURN
  END;


newfgtentry: PROCEDURE [fi, ci: ByteIndex] =
```

```
    BEGIN -- enters new value into fgt
    i: INTEGER;
    oldfgt: DESCRIPTOR FOR ARRAY OF FGTEntry;

    IF (fgti ← fgti+1) >= LENGTH[fgt] THEN
      BEGIN
      OPEN SystemDefs;
      oldfgt ← fgt;  fgtpages ← fgtpages+1;
      fgt ← DESCRIPTOR[
            AllocatePages[fgtpages],
            (fgtpages*PageSize)/SIZE[FGTEntry]];
      FOR i IN [0..LENGTH[oldfgt]) DO fgt[i] ← oldfgt[i] ENDLOOP;
      FreePages[BASE[oldfgt]];
      END;
    fgt[fgti] ← FGTEntry[findex: fi, cindex: ci];
    RETURN
    END;


  outbinary: PUBLIC PROCEDURE [bti: CBTIndex, start: CCIndex] =
    BEGIN -- outputs binary bytes for body bti starting at start
    cfi: ByteIndex;
    c, cj, nextc: CCIndex;
    bodystart: Address;
    offset, e, fs, nw: CARDINAL;
    bytetable, even: BOOLEAN;
    leftbyte: WORD;
    bodysei: POINTER [0..TableDefs.TableLimit) TO transfer constructor SERecord;
    sei: recordCSEIndex;

    bodystart ← movetocodeword[];
    offset ← bodystart * 2;
    FOR c ← start, cb[c].flink UNTIL c = CCNull DO
      WITH  cc:cb[c] SELECT FROM
        code => offset ← offset + cc.isize + cc.pad;
        other => WITH cc SELECT FROM
          table =>
            BEGIN
            OPEN InlineDefs;
            offset ← offset + tablecodebytes + pad;
            taboffset ← bodystart;
            bytetable ← btab ← byteablejumps[flink];
            even ← TRUE;
            FOR cj ← flink, cb[cj].flink DO
              WITH cb[cj] SELECT FROM
                jump =>
                  IF jtype = JumpC THEN
                    BEGIN
                    IF bytetable THEN
                      BEGIN
                      IF even THEN
                        leftbyte ← BITSHIFT[jbytes, 8]
                      ELSE
                        writecodeword[BITOR[leftbyte, jbytes]];
                      even ← ~even;
                      END
                    ELSE writecodeword[jbytes];
                    END
                  ELSE EXIT;
                ENDCASE => EXIT;
              ENDLOOP;
              IF bytetable AND ~even THEN
                writecodeword[BITOR[leftbyte,377B]];
            bodystart ← codeindex;
            END;
          ENDCASE;
        ENDCASE;
      ENDLOOP;
    e ← (bb+bti).entryIndex;
    WITH (bb+bti).info SELECT FROM
      Internal =>
        BEGIN
        IF bti = MPtr.mainBody THEN
          BEGIN
          writecodeword[MPtr.objectFrameSize];
          bodystart ← bodystart+1;
```

```
        END;
      fs ← computeframesize[frameSize];
      IF fs >= ControlDefs.MaxAllocSlot THEN
        BEGIN
        writecodeword[fs];
        bodystart ← bodystart+1;
        fs ← ControlDefs.MaxAllocSlot;
        END;
      offset ← bodystart*2;
      entryvector[e].framesize ← fs;
      newfgtentry[cfi ← sourceIndex, offset];
      END;
    ENDCASE => P5ADefs.P5Error[836];
  bodysei ← LOOPHOLE[SymTabDefs.UnderType[(bb+bti).ioType]];
  sei ← (seb+bodysei).inrecord;
  entryvector[e].nparams ← wordsforsei[sei];
  entryvector[e].defaults ← FALSE;
  entryvector[e].initialpc ← [bodystart];
  (bb+bti).info ←
      BodyInfo[External[origin: offset, bytes: , startIndex: fgti, indexLength: ]];
  FOR c ← start, nextc UNTIL c = CCNull DO
    WITH cc:cb[c] SELECT FROM
      code =>
        BEGIN
        IF cc.sourcefileindex # NULLfileindex THEN
          BEGIN
          IF cfi < cc.sourcefileindex THEN
            newfgtentry[cfi ← cc.sourcefileindex, offset];
          IF cfi > cc.sourcefileindex THEN
            BEGIN SIGNAL FileSequenceError; cfi ← cc.sourcefileindex; END;
          END;
        SELECT cc.isize FROM
          1 =>
            BEGIN
            writecodebyte[cc.inst];
            IF cc.pad # 0 THEN [] ← movetocodeword[];
            END;
          2 =>
            BEGIN
            IF cc.pad # 0 THEN
              BEGIN
              IF parity = even THEN SIGNAL CPtr.CodePassInconsistancy;
              writecodebyte[Mopcodes.zNOOP];
              END;
            writecodebyte[cc.inst];
            writecodebyte[cc.parameters[1]];
            END;
          3 =>
            BEGIN
            writecodebyte[cc.inst];
            IF cc.pad # 0 THEN
              BEGIN
              IF parity = even THEN SIGNAL CPtr.CodePassInconsistancy;
              [] ← movetocodeword[];
              END;
            writecodebyte[cc.parameters[2]];
            writecodebyte[cc.parameters[1]];
            END;
          ENDCASE => P5ADefs.P5Error[837];
        offset ← offset+cc.isize+cc.pad;
        END;
      other => WITH cc SELECT FROM
        table =>
         BEGIN
         CPtr.codeptr ← c;
         C1W[IF btab THEN Mopcodes.zJIB ELSE Mopcodes.zJIW, taboffset];
         cb[CPtr.codeptr].pad ← pad;
         END;
        startbody =>
          BEGIN
          WITH (bb+index).info SELECT FROM
            Internal =>
              newfgtentry[cfi ← sourceIndex, offset];
            ENDCASE => P5ADefs.P5Error[838];
          (bb+index).info ← BodyInfo[External[origin: offset, bytes: ,
              startIndex: fgti, indexLength: ]];
```

```
          END;
        endbody =>
          BEGIN
          WITH (bb+index).info SELECT FROM
            External =>
              BEGIN
              indexLength ← fgti-startIndex+1;
              bytes ← offset - origin;
              END;
            ENDCASE;
          END;
        ENDCASE;
      ENDCASE;
    nextc ← cb[c].flink;
    WITH cb[c] SELECT FROM
      code => nw ← isize-1+SIZE[code CCItem];
      label => nw ← SIZE[label CCItem];
      jump => nw ← SIZE[jump CCItem];
      other => nw ← SIZE[other CCItem];
      ENDCASE;
    CodeDefs.FreeChunk[c, nw];
    WITH (bb+bti).info SELECT FROM
      External =>
        BEGIN
        indexLength ← fgti-startIndex+1;
        bytes ← offset - (bodystart*2);
        END;
      ENDCASE;
    ENDLOOP;
  RETURN
  END;


byteablejumps: PROCEDURE [j: CCIndex] RETURNS [BOOLEAN] =
  BEGIN
  DO
   WITH cb[j] SELECT FROM
    jump =>
      IF jtype = JumpC THEN
        BEGIN
        IF jbytes > LAST[BYTE] THEN RETURN[FALSE];
        j ← cb[j].flink;
        END
      ELSE RETURN[TRUE];
    ENDCASE => RETURN[TRUE]
   ENDLOOP
  END;


ProcessGlobalStrings: PUBLIC PROCEDURE [framestart: CARDINAL] RETURNS [nextnewframe: CARDINAL] =
  BEGIN
  firstnewcode, nextnewcode: CARDINAL ← movetocodeword[];
  stsize: CARDINAL;

  dostring: PROCEDURE [msti: MSTIndex] =
    BEGIN
    nw: CARDINAL;
    IF (stb+msti).info = 0 THEN
      BEGIN (stb+msti).local ← TRUE; RETURN END;
    nw ← StringDefs.WordsForString[(stb+msti).string.length];
    (stb+msti).info ← nextnewframe;
    nextnewframe ← nextnewframe+nw;
    (stb+msti).codeIndex ← nextnewcode;
    nextnewcode ← nextnewcode + nw;
    [] ← StreamDefs.WriteBlock[MPtr.objectStream, @(stb+msti).string, nw];
    codeindex ← codeindex+nw;
    END; -- of dostring

  nextnewframe ← framestart;
  LitDefs.EnumerateMasterStrings[dostring];
  stsize ← nextnewframe - framestart;
  IF stsize > 0 THEN BLTStrings[firstnewcode, stsize, framestart, FALSE];
  END;
```

```
  ProcessLocalStrings: PUBLIC PROCEDURE [framestart: CARDINAL, first: STIndex] RETURNS [nextnewframe: C
**ARDINAL] =
    BEGIN
    nstrings: CARDINAL ← 0;
    countstrings: PROCEDURE [msti: MSTIndex] =
      BEGIN
      IF (stb+msti).local AND (stb+msti).codeIndex # 0 THEN
        nstrings ← nstrings+1;
      END;
    firstnewcode, nextnewcode: CARDINAL ← movetocodeword[];
    stsize, i, nw: CARDINAL;

    cursize: CARDINAL ← 0;
    StringInfo: TYPE = RECORD [offset: CARDINAL, sti: MSTIndex];
    star: DESCRIPTOR FOR ARRAY OF StringInfo;
    insertstrings: PROCEDURE [msti: MSTIndex] =
      BEGIN
      i, co, nw: CARDINAL;
      IF (stb+msti).local THEN
        BEGIN
        co ← (stb+msti).codeIndex;
        IF co # 0 THEN
          BEGIN
          FOR i ← cursize, i-1 WHILE i>0 AND co < star[i-1].offset DO
            star[i] ← star[i-1];
            ENDLOOP;
          star[i] ← [co, msti];
          cursize ← cursize+1;
          END
        ELSE
          BEGIN
          nw ← StringDefs.WordsForString[(stb+msti).string.length];
          (stb+msti).info ← nextnewframe;
          nextnewframe ← nextnewframe+nw;
          (stb+msti).codeIndex ← nextnewcode;
          nextnewcode ← nextnewcode + nw;
          [] ← StreamDefs.WriteBlock[MPtr.objectStream, @(stb+msti).string, nw];
          codeindex ← codeindex+nw;
          END;
        END;
      END; -- of insertstrings

    nextnewframe ← framestart;
    LitDefs.EnumerateLocalStrings[first, countstrings];
    IF nstrings # 0 THEN
      star ← DESCRIPTOR[
        SystemDefs.AllocateHeapNode[nstrings*SIZE[StringInfo]],
        nstrings];
    LitDefs.EnumerateLocalStrings[first, insertstrings];
    stsize ← nextnewframe - framestart;
    IF stsize > 0 THEN BLTStrings[firstnewcode, stsize, framestart, TRUE];
    i ← 0;
    WHILE i < nstrings DO
      framestart ← nextnewframe;
      nextnewcode ← firstnewcode ← star[i].offset;
      WHILE i < nstrings AND star[i].offset = nextnewcode DO
        nw ← StringDefs.WordsForString[(stb+star[i].sti).string.length];
        nextnewcode ← nextnewcode + nw;
        (stb+star[i].sti).info ← nextnewframe;
        nextnewframe ← nextnewframe+nw;
        i ← i+1;
        ENDLOOP;
      stsize ← nextnewframe - framestart;
      BLTStrings[firstnewcode, stsize, framestart, TRUE];
      ENDLOOP;
    IF nstrings # 0 THEN SystemDefs.FreeHeapNode[BASE[star]];
    END;

  BLTStrings: PROCEDURE [coffset, length, foffset: CARDINAL, local: BOOLEAN] =
    BEGIN OPEN FOpCodes;
    RequireStack[0];
    pushlitval[coffset];
    pushlitval[length];
    Ciout1[IF local THEN qLADRB ELSE qGADRB, foffset];
    Ciout0[qBLTC];
    END;
```

```
endcodefile: PUBLIC PROCEDURE RETURNS [nbytes: CARDINAL] =
  BEGIN
  OPEN SystemDefs, StreamDefs;
  saveindex: StreamIndex;
  [] ← movetocodeword[];
  MPtr.fgTable ← DESCRIPTOR[BASE[fgt], fgti+1];
  MPtr.codeSeg.pages ← (codeindex+(PageSize-1))/PageSize;
  saveindex ← GetIndex[MPtr.objectStream];
  SetIndex[MPtr.objectStream, entrybase];
  [] ← WriteBlock[MPtr.objectStream,
      BASE[entryvector],
      LENGTH[entryvector]*SIZE[ControlDefs.EntryVectorItem]];
  FreeSegment[BASE[entryvector]];
  MPtr.mtRoot.framesize ← MPtr.objectFrameSize;
  MPtr.mtRoot.fsi ← computeframesize[MPtr.objectFrameSize];
  MPtr.mtRoot.code.length ← codeindex*2;
  SetIndex[MPtr.objectStream, saveindex];
  RETURN [codeindex*2]
  END;

END...
```