```
-- ListUsing.mesa; modified by Sweet, May 10, 1978  10:27 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  BcdDefs: FROM "bcddefs",
  CommanderDefs: FROM "commanderdefs",
  IODefs: FROM "iodefs",
  ListerDefs: FROM "listerdefs",
  OutputDefs: FROM "outputdefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs",
  StreamDefs: FROM "streamdefs",
  SymbolTableDefs: FROM "symboltabledefs",
  SymDefs: FROM "symdefs",
  SystemDefs: FROM "systemdefs";


ListUsing: PROGRAM IMPORTS CommanderDefs, IODefs, ListerDefs, OutputDefs, SegmentDefs, StreamDefs, Stri
**ngDefs, SymbolTableDefs, SystemDefs =
  BEGIN OPEN ListerDefs, OutputDefs, SymDefs;
  FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
  FileHandle: TYPE = SegmentDefs.FileHandle;

  symbols: SymbolTableDefs.SymbolTableBase;

  PutSubStringLC: PROCEDURE [ss: StringDefs.SubString] =
    BEGIN
    i: CARDINAL;
    c1: CHARACTER;
    FOR i IN [0..ss.length) DO
      c1 ← ss.base[ss.offset+i];
      IF c1 IN ['A..'Z] THEN c1 ← c1 -'A+'a;
      PutChar[c1];
      ENDLOOP;
    END;

  CompareNames: PROCEDURE [n1, n2: StringDefs.SubString] RETURNS [INTEGER] =
    BEGIN
    i: CARDINAL;
    c1, c2: CHARACTER;
    FOR i IN [0..MIN[n1.length, n2.length]) DO
      c1 ← n1.base[n1.offset+i];
      c2 ← n2.base[n2.offset+i];
      IF c1 IN ['A..'Z] THEN c1 ← c1 -'A+'a;
      IF c2 IN ['A..'Z] THEN c2 ← c2 -'A+'a;
      SELECT c1-c2 FROM
        <0 => RETURN [-1];
        >0 => RETURN [1];
        ENDCASE;
      ENDLOOP;
    SELECT INTEGER[n1.length - n2.length] FROM
      <0 => RETURN [-1];
      >0 => RETURN [1];
      ENDCASE => RETURN [0];
    END;

  SortNames: PROCEDURE [na: DESCRIPTOR FOR ARRAY OF StringDefs.SubStringDescriptor] =
    BEGIN
    i: CARDINAL;
    j: INTEGER;
    key: StringDefs.SubStringDescriptor;
    FOR i IN [1..LENGTH[na]) DO
      key ← na[i];
      j ← i-1;
      WHILE j >= 0 AND CompareNames[@na[j],@key]>0 DO
        na[j+1] ← na[j]; j ← j-1; ENDLOOP;
      na[j+1] ← key;
      ENDLOOP;
    END;

  GenCtx: PROCEDURE [ctx: SymDefs.CTXIndex, p: PROCEDURE [SymDefs.ISEIndex]] =
    BEGIN OPEN symbols;
    sei: SymDefs.ISEIndex;
    FOR sei ← FirstCtxSe[ctx], NextSe[sei] UNTIL sei = SENull DO
      p[sei]; ENDLOOP;
    END;
```

```
PrintUsing: PROCEDURE =
  BEGIN OPEN SymDefs, symbols;
  bti: BTIndex;
  ctx: CTXIndex;
  sei: ISEIndex;
  i, n, idir, ndir: CARDINAL;
  first: BOOLEAN ← TRUE;
  desc: StringDefs.SubStringDescriptor;
  modname: StringDefs.SubString = @desc;
  mname: StringDefs.SubString;
  DirRec: TYPE = RECORD [dirname: StringDefs.SubStringDescriptor, dirsei: ISEIndex];
  da: DESCRIPTOR FOR ARRAY OF DirRec;
  na: DESCRIPTOR FOR ARRAY OF StringDefs.SubStringDescriptor;
  countids: PROCEDURE [ISEIndex] = BEGIN n ← n+1; END;
  insertid: PROCEDURE [sei: ISEIndex] =
    BEGIN OPEN symbols;
    SubStringForHash[@na[i], (seb+sei).htptr];
    i ← i+1;
    END;

  PutCR[];

  ndir ← 0;
  FOR sei ← FirstCtxSe[stHandle.directoryCtx], NextSe[sei]
   UNTIL sei = ISENull DO
    ndir ← ndir+1;
    ENDLOOP;
  IF ndir = 0 THEN RETURN;
  da ← DESCRIPTOR[
          SystemDefs.AllocateHeapNode[
            SIZE[DirRec]*ndir],
          ndir];
  ndir ← 0;
  FOR sei ← FirstCtxSe[stHandle.directoryCtx], NextSe[sei]
   UNTIL sei = ISENull DO
    BEGIN i: INTEGER ← ndir-1;
    SubStringForHash[modname, (seb+sei).htptr];
    WHILE i >= 0 AND CompareNames[@da[i].dirname, modname]>0 DO
      da[i+1] ← da[i]; i ← i-1; ENDLOOP;
    da[i+1] ← [modname↑, sei];
    ndir ← ndir+1;
    END;
    ENDLOOP;
  FOR idir IN [0..ndir) DO
    mname ← @da[idir].dirname; sei ← da[idir].dirsei;
    WITH (seb+UnderType[(seb+sei).idtype]) SELECT FROM
      definition =>
        BEGIN
        isei: ISEIndex;
        ctx ← defCtx;
        FOR isei ← FirstCtxSe[stHandle.importCtx], NextSe[isei]
         UNTIL isei = ISENull DO
          WITH (seb+UnderType[(seb+isei).idtype]) SELECT FROM
            definition =>
              WITH (ctxb+defCtx) SELECT FROM
                imported => IF includeLink = ctx THEN
                  BEGIN ctx ← defCtx; EXIT END;
                ENDCASE;
            ENDCASE;
          ENDLOOP;
        END;
      transfer =>
        BEGIN
        bti ← (seb+sei).idinfo;
        ctx ← (bb+bti).localCtx;
        END;
      ENDCASE => ERROR;
    n ← 0;
    GenCtx[ctx, countids];
    WITH (ctxb+ctx) SELECT FROM
      included => NULL;
      imported => GenCtx[includeLink, countids];
      ENDCASE => LOOP; -- main body
    IF n > 0 THEN na ← DESCRIPTOR[
          SystemDefs.AllocateHeapNode[
```

```
            SIZE[StringDefs.SubStringDescriptor]*n],
          n];
      i ← 0;
      GenCtx[ctx, insertid];
      WITH (ctxb+ctx) SELECT FROM
        imported => GenCtx[includeLink, insertid];
        ENDCASE;
      IF first THEN PutString["DIRECTORY"]
      ELSE PutChar[',];
      PutCR[];
      first ← FALSE; PutString["  "];
      PutSubString[mname]; PutString[": FROM """];
      PutSubStringLC[mname]; PutChar['"];
      IF n > 0 THEN
        BEGIN
        SortNames[na];
        PutString[" USING ["];
        PutSubString[@na[0]];
        FOR i IN (0..LENGTH[na]) DO
          PutString[", "];
          PutSubString[@na[i]];
          ENDLOOP;
        PutChar[']];
        SystemDefs.FreeHeapNode[BASE[na]];
        END;
      ENDLOOP;
    SystemDefs.FreeHeapNode[BASE[da]];
    PutChar[';]; PutCR[]; PutCR[]; PutCR[];
    RETURN
    END;

  UsingList: PROCEDURE [cmd: STRING] =
    BEGIN OPEN StringDefs, StreamDefs;
    s: STRING ← [50];
    ch: CHARACTER;
    cs: StreamHandle ← NewByteStream[cmd, Read];
    UNTIL cs.endof[cs] DO
      s.length ← 0;
      WHILE ~cs.endof[cs] AND (ch ← cs.get[cs]) # '  DO AppendChar[s, ch]; ENDLOOP;
      IF s.length > 0 THEN BEGIN IODefs.WriteLine[s]; Using[s]; END;
      ENDLOOP;
    cs.destroy[cs];
    END;

Using: PROCEDURE[root: STRING] =
  BEGIN OPEN StringDefs, SegmentDefs;
  i: CARDINAL;
  defs: BOOLEAN ← FALSE;
  bcdFile: STRING ← [40];
  sseg, cseg: FileSegmentHandle;
  AppendString[bcdFile,root];
  FOR i IN [0..bcdFile.length) DO
    IF bcdFile[i] = '. THEN EXIT;
    REPEAT FINISHED => AppendString[bcdFile,".bcd"];
    ENDLOOP;
  BEGIN
  [code: cseg, symbols: sseg] ← Load[bcdFile !
    NoFGT => RESUME;
    NoCode => BEGIN defs ← TRUE; RESUME END;
    NoSymbols, IncorrectVersion, MultipleModules => GOTO badformat;
    SegmentDefs.FileNameError => GOTO badname];
  symbols←SymbolTableDefs.AcquireSymbolTable[
    SymbolTableDefs.TableForSegment[sseg]];
  IF ~defs THEN SegmentDefs.DeleteFileSegment[cseg];
  ListerDefs.SetRoutineSymbols[symbols];
  OpenOutput[root,".ul"];
  WriteFileID[];
  IF symbols.sourceFile # NIL THEN
    BEGIN
    PutString["  Source: "];
    PutString[symbols.sourceFile];
    PutCR[];
    END;
  PrintUsing[];
  SymbolTableDefs.ReleaseSymbolTable[symbols];
  SegmentDefs.DeleteFileSegment[sseg];
```

```
  CloseOutput[];
  EXITS
    badformat => IODefs.WriteString["Bad Format!"];
    badname => IODefs.WriteString["File Not Found!"];
  END;
  END;


command: CommanderDefs.CommandBlockHandle;

command ← CommanderDefs.AddCommand["Using",LOOPHOLE[Using],1];
command.params[0] ← [type: string, prompt: "Filename"];


command ← CommanderDefs.AddCommand["UsingList",LOOPHOLE[UsingList],1];
command.params[0] ← [type: string, prompt: "Filename"];

END...
```