```
-- ListPub.mesa; modified by Bruce, September 2, 1978  2:03 PM

DIRECTORY
  AltoDefs:  FROM "altodefs" USING [PageNumber, BytesPerPage],
  AltoFileDefs:  FROM "altoFiledefs" USING [FP],
  CommanderDefs:  FROM "commanderdefs" USING [AddCommand, CommandBlockHandle],
  DirectoryDefs:  FROM "directorydefs" USING [DirectoryLookup],
  DisplayDefs:  FROM "displaydefs" USING [DisplayOn, DisplayOff],
  GPsortDefs:  FROM "gpsortdefs" USING [PutProcType,
    GetProcType, LT, EQ, GT, Sort],
  InlineDefs:  FROM "inlinedefs" USING [BITXOR],
  IODefs: FROM "iodefs" USING [CR, WriteString],
  ListerDefs: FROM "listerdefs" USING [IncorrectVersion, Load,
    MultipleModules, NoCode, NoFGT, NoSymbols, PrintSei, SetRoutineSymbols],
  OutputDefs: FROM "outputdefs" USING [GetOutputStream, CloseOutput,
    OpenOutput, PutChar, PutCR, PutDecimal, PutNumber, PutOctal, PutString],
  SegmentDefs: FROM "segmentdefs" USING [DeleteFileSegment, DestroyFile,
    FileNameError, FileSegmentHandle, LockFile, UnlockFile, Read],
  StreamDefs: FROM "streamdefs" USING [CreateByteStream, DiskHandle,
    NormalizeIndex, GetIndex, GrIndex, NewByteStream, StreamIndex],
  StringDefs: FROM "stringdefs" USING [AppendChar, AppendString,
    AppendSubString, SubStringDescriptor, WordsForString],
  SymbolTableDefs: FROM "symboltabledefs" USING [
    AcquireSymbolTable, ReleaseSymbolTable, SymbolTableBase, TableForSegment],
  SymDefs: FROM "symdefs" USING [BodyRecord, BTIndex, codeANY, codeBOOLEAN,
    codeCHARACTER, codeINTEGER, CTXIndex, HTNull, ISEIndex, ISENull, lZ,
    recordCSEIndex, recordCSENull, SEIndex, SENull, TransferMode, typeTYPE];

ListPub: PROGRAM
  IMPORTS CommanderDefs, DirectoryDefs, DisplayDefs, GPsortDefs,
    IODefs, ListerDefs, OutputDefs, SegmentDefs, StreamDefs, StringDefs,
    SymbolTableDefs =
BEGIN OPEN SymDefs;

ProcType: TYPE = PROCEDURE [root: STRING];
cz: CHARACTER = 32C;
FileTooBig: SIGNAL = CODE;
largestItem: CARDINAL;
lastItem: StreamDefs.StreamIndex;
moduleList: STRING ← [40];
inSh, outSh, sortSh: StreamDefs.DiskHandle;
symbols: SymbolTableDefs.SymbolTableBase;

Cap: PROCEDURE [ch: CHARACTER] RETURNS [cap: CHARACTER] =
  BEGIN RETURN[IF ch IN ['a..'z] THEN ch-('a-'A) ELSE ch] END;

CompareStrings: PROCEDURE [p1,p2: POINTER] RETURNS[INTEGER] =
  BEGIN OPEN GPsortDefs;
  s1: STRING ← p1;
  s2: STRING ← p2;
  idx: CARDINAL;
  c1, c2: CHARACTER;
  FOR idx IN [0..MIN[s1.length, s2.length]) DO
    c1 ← Cap[s1[idx]]; c2 ← Cap[s2[idx]];
    SELECT c1 FROM
      < c2 => RETURN[LT];
      > c2 => RETURN[GT];
      ENDCASE;
    ENDLOOP;
  SELECT s1.length FROM
    < s2.length => RETURN[LT];
    = s2.length => RETURN[EQ];
    > s2.length => RETURN[GT];
    ENDCASE;
  END;

GetItem: GPsortDefs.GetProcType =
  BEGIN
  char: CHARACTER ← 0C;
  s: STRING ← p1;
  s↑ ← [length: 0, maxlength: largestItem-2, text:];
  UNTIL sortSh.endof[sortSh] DO
    char ← sortSh.get[sortSh];
    IF char = IODefs.CR THEN EXIT ELSE StringDefs.AppendChar[s,char];
    REPEAT
      FINISHED => RETURN[0];
    ENDLOOP;
  RETURN[StringDefs.WordsForString[s.length]]
  END;

PutItem: GPsortDefs.PutProcType =
  BEGIN OPEN StreamDefs, OutputDefs;
  maxSi: StreamIndex ← NormalizeIndex[[0,50000]];
  trailer: STRING = "13398d2998\b"L;
  namelength: CARDINAL ← 0;
  itemString: STRING ← p;
  PutString[itemString];
  PutChar[cz];
  PutString[trailer];
  UNTIL itemString[namelength] = ': DO
    namelength ← namelength+1;
    IF namelength > itemString.length THEN ERROR;
    ENDLOOP;
  PutDecimal[namelength]; PutChar['B];
  PutCR[];
  IF GrIndex[GetIndex[outSh],maxSi] THEN SIGNAL FileTooBig;
  END;

doPriv, xferOnly: BOOLEAN;

PrintSymbols: PROCEDURE =
  BEGIN OPEN symbols,StringDefs;
  modname: STRING ← [50]; -- :SP[name]SP
  ss: SubStringDescriptor;
  mySei,sei: ISEIndex;
  thisItem: StreamDefs.StreamIndex;
  AppendString[modname,": ["L];  -- set up modname
  FOR sei ← FirstCtxSe[stHandle.directoryCtx], NextSe[sei] UNTIL sei = ISENull DO
    mySei ← sei;
    ENDLOOP;
  SubStringForHash[@ss, (seb+mySei).htptr];
  AppendSubString[modname,@ss];
  AppendString[modname,"] "L];
  AppendSubString[moduleList,@ss];
  BlinkCursor[];
  AppendChar[moduleList,' ];
  FOR sei ← FirstCtxSe[stHandle.outerCtx], NextSe[sei] UNTIL sei = ISENull DO
    IF (doPriv OR (seb+sei).public) AND
        (~xferOnly OR XferMode[(seb+sei).idtype] # none) THEN
      BEGIN
      defaultPublic ← TRUE;
      PrintSym[sei, modname];
      OutputDefs.PutCR[];
      thisItem ← StreamDefs.GetIndex[outSh];
      largestItem ← MAX[largestItem,SiSub[thisItem,lastItem]];
      lastItem ← thisItem;
      END;
    ENDLOOP;
  END;

SiSub: PROCEDURE [si1,si2: StreamDefs.StreamIndex] RETURNS [CARDINAL] =
  BEGIN OPEN AltoDefs;
  pages: PageNumber ← si1.page - si2.page;
  bytes: CARDINAL ← si1.byte - si2.byte;
  RETURN [pages*BytesPerPage+bytes]
  END;

defaultPublic: BOOLEAN;

PrintSym: PROCEDURE [sei: ISEIndex, colonstring: STRING] =
  BEGIN OPEN symbols;
  savePublic: BOOLEAN ← defaultPublic;
  typeSei: SEIndex;
  IF (seb+sei).htptr # HTNull THEN
    BEGIN
    ListerDefs.PrintSei[sei];
    OutputDefs.PutString[colonstring];
    END;
  IF (seb+sei).public # defaultPublic THEN
    BEGIN defaultPublic ← (seb+sei).public;
    OutputDefs.PutString[IF defaultPublic THEN "PUBLIC "L ELSE "PRIVATE "L];
    END;
  IF (seb+sei).idtype = typeTYPE THEN
    BEGIN  typeSei ← (seb+sei).idinfo;
    OutputDefs.PutString["TYPE="L];
    [] ← PrintType[typeSei, NoSub];
    END
  ELSE
    BEGIN vf: ValFormat;
    typeSei ← (seb+sei).idtype;
    vf ← PrintType[typeSei, NoSub];
    IF (seb+sei).constant AND vf # none THEN
      BEGIN OPEN OutputDefs;
      val: UNSPECIFIED = (seb+sei).idvalue;
      PutChar['=];
      SELECT vf FROM
        num => PrintValue[val];
        char => BEGIN PutNumber[val, [8,FALSE,TRUE,0]]; PutChar['C] END;
        bool => PutString[IF FALSE = val THEN "FALSE" ELSE "TRUE"];
        ENDCASE;
      END;
    END;
  defaultPublic ← savePublic;
  END;

PrintFieldCtx: PROCEDURE [ctx: CTXIndex] =
  BEGIN OPEN symbols, OutputDefs;
  isei: ISEIndex ← FirstCtxSe[ctx];
  first: BOOLEAN ← TRUE;
  IF isei # ISENull AND (seb+isei).ctxnum # ctx THEN isei ← NextSe[isei];
  IF isei = ISENull THEN
    BEGIN PutString["NULL"L]; RETURN END;
  PutChar['[];
  FOR isei ← isei, NextSe[isei] UNTIL isei= ISENull DO
    IF first THEN first ← FALSE ELSE PutString[", "L];
    PrintSym[isei, ": "L];
    ENDLOOP;
  PutChar[']];
  END;

PrintValue: PROCEDURE [value: UNSPECIFIED] =
  BEGIN
  IF LOOPHOLE[value, CARDINAL] < 1000
    THEN  OutputDefs.PutDecimal[value]
    ELSE  OutputDefs.PutOctal[value];
  END;

NoSub: PROCEDURE = BEGIN RETURN END;
arraySub: BOOLEAN ← FALSE;

ValFormat: TYPE = {none, num, char, bool, machinecode};

PrintType: PROCEDURE [tsei: SEIndex, dosub: PROCEDURE] RETURNS [vf: ValFormat] =
  BEGIN OPEN SymDefs, OutputDefs, ListerDefs, symbols;
  vf ← none;
  WITH t: (seb+tsei) SELECT FROM
    id =>
      BEGIN OPEN SymDefs;
      printBase: BOOLEAN ← TRUE;
      ifInteger: BOOLEAN ← FALSE;
      bsei: SEIndex ← tsei;
      DO
        WITH (seb+UnderType[bsei]) SELECT FROM
          basic =>
            BEGIN
            SELECT code FROM
              codeINTEGER => BEGIN printBase ← ifInteger; vf ← num END;
              codeBOOLEAN => vf ← bool;
              codeCHARACTER => vf ← char;
              ENDCASE;
            EXIT;
            END;
          subrange => BEGIN bsei ← rangetype; ifInteger ← TRUE END;
          ENDCASE => EXIT;
        ENDLOOP;
      IF printBase OR dosub = NoSub THEN
        BEGIN
        PrintSei[LOOPHOLE[tsei]];
        UNTIL (tsei ← TypeLink[tsei]) = SENull DO
          WITH (seb+tsei) SELECT FROM
            id => BEGIN PutChar[' ]; PrintSei[LOOPHOLE[tsei]] END;
            ENDCASE;
          ENDLOOP;
        END;
      dosub[];
      END;
    constructor =>
      WITH t SELECT FROM
        --basic =>  won't see one, see the id first.
        enumerated =>
          BEGIN isei: ISEIndex; first: BOOLEAN ← TRUE;
          PutChar['{];
          FOR isei ← FirstCtxSe[valuectx], NextSe[isei] UNTIL isei= ISENull DO
            IF first THEN first ← FALSE ELSE PutString[", "L];
            PrintSei[isei];
            ENDLOOP;
          PutChar['}];
          END;
        record =>
          BEGIN
          IF (ctxb+fieldctx).ctxlevel # 1Z THEN
            BEGIN
            fctx: CTXIndex = fieldctx;
            bti: BTIndex ← FIRST[BTIndex];
            btlimit: BTIndex = bti+stHandle.bodyBlock.size;
            PutString["FRAME ["];
            UNTIL bti = btlimit DO
              WITH entry: (bb+bti) SELECT FROM
                Callable =>
                  BEGIN
                  IF entry.localCtx = fctx THEN
                    BEGIN
                    PrintSei[entry.id]; PutChar[']];
                    EXIT
                    END;
                  bti ← bti + (WITH entry SELECT FROM
                    Inner => SIZE[Inner Callable BodyRecord],
                    ENDCASE => SIZE[Outer Callable BodyRecord]);
                  END;
                ENDCASE => bti ← bti + SIZE[Other BodyRecord];
              ENDLOOP;
            END
          ELSE
            BEGIN
            IF monitored THEN PutString["MONITORED "L];
            IF machineDep THEN PutString["MACHINE DEPENDENT "L];
            PutString["RECORD"L];
            PrintFieldCtx[fieldctx];
            END;
          END;
        pointer =>
          BEGIN
          IF ordered THEN PutString["ORDERED "L];
          IF basing THEN PutString["BASE "L];
          PutString["POINTER"L];
          dosub[];
          WITH (seb+UnderType[pointedtotype]) SELECT FROM
            basic => IF code = SymDefs.codeANY THEN GO TO noprint;
            ENDCASE;
          PutString[" TO "L];
          [] ← PrintType[pointedtotype, NoSub];
          EXITS
            noprint => NULL;
          END;
        array =>
          BEGIN
          IF packed THEN PutString["PACKED "L];
          PutString["ARRAY "L];
          arraySub ← TRUE;
          [] ← PrintType[indextype, NoSub];
          arraySub ← FALSE;
          PutString[" OF "L];
          [] ← PrintType[componenttype, NoSub];
          END;
      arraydesc =>
        BEGIN
        PutString["DESCRIPTOR FOR "L];
        [] ← PrintType[describedType, NoSub];
        END;
      transfer =>
        BEGIN
        PutModeName[mode];
        IF inrecord # recordCSENull THEN
          BEGIN PutChar[' ];
          PrintFieldCtx[(seb+inrecord).fieldctx];
          END;
        IF outrecord # recordCSENull THEN
          BEGIN
          PutString[" RETURNS "L];
          PrintFieldCtx[(seb+outrecord).fieldctx];
          END;
        END;
      union =>
        BEGIN
        tagtype: SEIndex;
        PutString["SELECT "L];
        IF ~controlled THEN
          IF overlayed THEN PutString["OVERLAID "L]
          ELSE PutString["COMPUTED "L]
        ELSE
          BEGIN PrintSei[tagsei]; PutString[": "L] END;
        tagtype ← (seb+tagsei).idtype;
        IF (seb+tagsei).public # defaultPublic THEN
          OutputDefs.PutString[IF defaultPublic THEN "PRIVATE "L ELSE "PUBLIC "L];
        WITH (seb+tagtype) SELECT FROM
          id => [] ← PrintType[tagtype, NoSub];
          constructor => PutChar['*];
          ENDCASE;
        PutString[" FROM "L];
        BEGIN isei: ISEIndex; first: BOOLEAN ← TRUE;
        varRec: recordCSEIndex;
        FOR isei ← FirstCtxSe[casectx], NextSe[isei] UNTIL isei= ISENull DO
          IF first THEN first ← FALSE ELSE PutString[", "L];
          PrintSei[isei]; PutString[" => "L];
          varRec ← (seb+isei).idinfo;
          PrintFieldCtx[(seb+varRec).fieldctx];
          ENDLOOP;
        PutString[" ENDCASE"L];
        END;
        END;
      relative =>
        BEGIN
        IF baseType # SENull THEN [] ← PrintType[baseType, NoSub];
        PutString["RELATIVE "L];
        [] ← PrintType[offsetType, dosub];
        END;
      subrange =>
        BEGIN
        org: INTEGER ← origin;
        size: CARDINAL ← range;
        doit: PROCEDURE =
          BEGIN
          PutChar['[];
          PrintValue[org];
          PutString[".."L];
          IF arraySub AND size = 177777B THEN
            BEGIN PrintValue[org]; PutChar[')] END
          ELSE
            BEGIN PrintValue[org+size]; PutChar[']] END;
          END;
        IF ~flexible THEN vf ← PrintType[rangetype, doit];
        END;
      long =>
        BEGIN
        PutString["LONG "L];
        [] ← PrintType[rangetype, NoSub];
        END;
      real => PutString["REAL"L];
      ENDCASE => PutString["Send message to SDSUPPORT"L];
    ENDCASE;
```

```
  END;

PutModeName: PROCEDURE[n: TransferMode] =
  BEGIN
  ModePrintName: ARRAY TransferMode OF STRING = ["PROCEDURE"L, "PORT"L,
    "SIGNAL"L, "ERROR"L, "PROCESS"L, "PROGRAM"L, "NONE"L];
  OutputDefs.PutString[ModePrintName[n]]
  END;

DoSymbols: PROCEDURE [bcdFile: STRING] =
  BEGIN OPEN ListerDefs;
  defs: BOOLEAN ← FALSE;
  sseg: SegmentDefs.FileSegmentHandle;
  BEGIN
  [symbols: sseg] ← Load[bcdFile !
    NoFGT => RESUME;
    NoCode => RESUME;  -- language feature
    NoSymbols, IncorrectVersion, MultipleModules => GOTO badformat;
    SegmentDefs.FileNameError => GOTO badname];
  DisplayDefs.DisplayOff[black];
  symbols ← SymbolTableDefs.AcquireSymbolTable[
    SymbolTableDefs.TableForSegment[sseg]];
  SetRoutineSymbols[symbols];
  PrintSymbols[];
  SymbolTableDefs.ReleaseSymbolTable[symbols];
  SegmentDefs.DeleteFileSegment[sseg];
  EXITS
    badformat =>
      BEGIN OPEN IODefs;
      DisplayDefs.DisplayOn[];
      WriteString[bcdFile];
      WriteString[" Has A Bad Format!"L];
      END;
    badname =>
      BEGIN OPEN IODefs;
      DisplayDefs.DisplayOn[];
      WriteString[bcdFile];
      WriteString[" Not Found!"L];
      END;
  END;
  END;  -- Of DoSymbols

AppendBcd: PROCEDURE [s: STRING] =
  BEGIN
  i: CARDINAL;
  FOR i IN [0..s.length) DO
    IF s[i] = '. THEN BEGIN s.length ← i; EXIT END
    ENDLOOP;
  StringDefs.AppendString[s,".bcd"L];
  END;

globalRoot: STRING;

DoIt: PROCEDURE[root: STRING, myDoPriv, myXferOnly: BOOLEAN] =
  BEGIN OPEN SegmentDefs, OutputDefs;
  list: BOOLEAN;
  bcdFile: STRING ← [40];
  sortFile: STRING ← "2.xref";
  fp: AltoFileDefs.FP;
  globalRoot ← root; doPriv ← myDoPriv;  xferOnly ← myXferOnly;
  StringDefs.AppendString[bcdFile,root];
  AppendBcd[bcdFile];
  list ← NOT DirectoryDefs.DirectoryLookup[@fp,bcdFile,FALSE];
  largestItem ← 0;
  lastItem ← [0,0];
  OutputDefs.OpenOutput[root,".scratch$"L];
  outSh ← LOOPHOLE[GetOutputStream[]];
  IF list THEN
    BEGIN OPEN StreamDefs;
    inSh ← NewByteStream[root,Read !FileNameError => GOTO badname];
    GPsortDefs.Sort[GetName,PutName,CompareStrings,22,22,140];
    PutChar[cz]; PutChar['j]; PutCR[];  -- trailer for module list
    inSh.destroy[inSh];
    EXITS
      badname => BEGIN IODefs.WriteString["File Not Found!"L]; RETURN END;
    END
  ELSE
    BEGIN
    DoSymbols[bcdFile];
    ChangeOutput[];
    PutString[moduleList];
    PutChar[cz]; PutChar['c]; PutCR[];  -- trailer for heading
    END;
  PutChar[cz]; PutCR[];  -- skip a line
  largestItem ← largestItem + 20;  -- a little slop
  BlinkCursor[];
  GPsortDefs.Sort[GetItem,PutItem,CompareStrings,100,largestItem/2,15
    !FileTooBig =>
      BEGIN
      CloseOutput[];
      OpenOutput[root,sortFile];
      outSh ← LOOPHOLE[GetOutputStream[]];
      sortFile[0] ← sortFile[0] + 1;
      RESUME
      END];
  DisplayDefs.DisplayOn[];
  sortSh.destroy[sortSh];
  UnlockFile[sortSh.file];
  DestroyFile[sortSh.file];
  CloseOutput[];
  END;

BlinkCursor: PROCEDURE =
  BEGIN
  map: POINTER TO WORD = LOOPHOLE[431B];
  i: CARDINAL;
  FOR i IN [0..16) DO
    (map+i)↑ ← InlineDefs.BITXOR[(map+i)↑,177777B];
    ENDLOOP;
  FOR i IN [0..1000) DO NULL ENDLOOP;  -- wait a little while
  FOR i IN [0..16) DO
    (map+i)↑ ← InlineDefs.BITXOR[(map+i)↑,177777B];
    ENDLOOP;
  END;

ChangeOutput: PROCEDURE =
  BEGIN OPEN SegmentDefs, OutputDefs;
  LockFile[outSh.file];
  CloseOutput[];
  sortSh ← StreamDefs.CreateByteStream[outSh.file,Read];
  OpenOutput[globalRoot,".xref"L];
  outSh ← LOOPHOLE[GetOutputStream[]];
  PutString["PUBLIC SYMBOLS FOR "L];
  END;

GetName: GPsortDefs.GetProcType =
  BEGIN OPEN StringDefs;
  char: CHARACTER ← 0C;
  file: STRING ← [40];
  s: STRING ← p1;
  s↑ ← [length: 0, maxlength: 40, text:];
  UNTIL inSh.endof[inSh] DO
    char ← inSh.get[inSh];
    SELECT char FROM
      '-, '., '$ => AppendChar[file,char];
      IN ['0..'9] => AppendChar[file,char];
      IN ['A..'Z] => AppendChar[file,char];
      IN ['a..'z] => AppendChar[file,char];
      ENDCASE => IF file.length # 0 THEN EXIT;
    REPEAT
      FINISHED =>
        BEGIN OPEN OutputDefs;
        ChangeOutput[];
        PutChar[cz]; PutChar['c]; PutCR[];  -- trailer for heading
        RETURN[0];
        END;
    ENDLOOP;
  AppendBcd[file];
  DoSymbols[file];
  AppendString[s,moduleList];
  moduleList.length ← 0;
  RETURN[WordsForString[s.length]]
  END;

PutName: GPsortDefs.PutProcType =
  BEGIN
  s: STRING ← LOOPHOLE[p];
  OutputDefs.PutString[s];
  END;

-- mainline
command: CommanderDefs.CommandBlockHandle;

command ← CommanderDefs.AddCommand["Xref", LOOPHOLE[DoIt], 3];
command.params[0] ← [type: string, prompt: "Filename"];
command.params[1] ← [type: boolean, prompt: "Include Private Symbols?"];
command.params[2] ← [type: boolean, prompt: "Procedures Only?"];

END...
```