```
-- file Pass1.Mesa
-- last modified by Satterthwaite, July 16, 1978  9:47 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [charlength, maxword, wordlength],
  ComData: FROM "comdata"
    USING [
      idANY, idBOOLEAN, idCARDINAL, idCHARACTER, idFALSE,
      idINTEGER, idLOCK, idREAL, idSTRING, idTRUE, idUNWIND,
      nErrors, outerCtx, seAnon, sourceStream, tC0, tC1,
      typeBOOLEAN, typeCHARACTER, typeCONDITION, typeINTEGER, typeLOCK,
      typeREAL, typeSTRING],
  CompilerDefs: FROM "compilerdefs" USING [MakeSwappable],
  ControlDefs: FROM "controldefs" USING [ControlLink, EPRange, GFTNull],
  LitDefs: FROM "litdefs" USING [FindLiteral],
  P1Defs: FROM "p1defs" USING [Parse, Scanner, Parser, Pass1T],
  SegmentDefs: FROM "segmentdefs"
    USING [FileSegmentHandle, FileSegmentAddress, SwapIn, SwapOut, Unlock],
  StringDefs: FROM "stringdefs" USING [SubStringDescriptor],
  SymDefs: FROM "symdefs"
    USING [
      ctxtype, setype,
      BitAddress, SERecord,
      HTIndex, SEIndex, ISEIndex, CSEIndex, recordCSEIndex, CTXIndex,
      codeANY, codeINTEGER, codeCHARACTER, typeANY, typeTYPE,
      HTNull, recordCSENull, lZ],
  SymTabDefs: FROM "symtabdefs"
    USING [
      EnterString, fillctxse, makectxse, makenewctx, makenonctxse,
      makeSEChain, NextSe, resetctxlist, UnderType],
  TableDefs: FROM "tabledefs"
    USING [TableBase, TableNotifier, AddNotify, DropNotify],
  TreeDefs: FROM "treedefs" USING [empty];

Pass1: PROGRAM
    IMPORTS
        CompilerDefs, LitDefs, P1Defs, SegmentDefs, SymTabDefs, TableDefs,
        dataPtr: ComData
    EXPORTS CompilerDefs, P1Defs =
  BEGIN
  OPEN SymTabDefs, SymDefs;

  -- symbol table bases
    seb: TableDefs.TableBase;   -- semantic entry base
    ctxb: TableDefs.TableBase;  -- context table base

  P1Notify: TableDefs.TableNotifier =
    BEGIN
    seb ← base[setype];  ctxb ← base[ctxtype];  RETURN
    END;

 -- definition of standard symbols

  WordLength: CARDINAL = AltoDefs.wordlength;

  PrefillSymbols: PROCEDURE =
    BEGIN  -- called to prefill the compiler's symbol table
    OPEN dataPtr;
    tSei, ptrSei: CSEIndex;
    rSei: recordCSEIndex;
    tCtx: CTXIndex;
    sei: ISEIndex;
    outerCtx ← makenewctx[lZ];
    idANY ← MakeBasicType["UNSPECIFIED"L, codeANY, TRUE, WordLength];
      IF UnderType[idANY] # typeANY THEN ERROR;
    idINTEGER ← MakeBasicType["INTEGER"L, codeINTEGER, TRUE, WordLength];
      typeINTEGER ← UnderType[idINTEGER];
    idCHARACTER ← MakeBasicType["CHARACTER"L, codeCHARACTER, TRUE, AltoDefs.charlength];
      typeCHARACTER ← UnderType[idCHARACTER];
    -- make BOOLEAN type
      typeBOOLEAN ← makenonctxse[SIZE[enumerated constructor SERecord]];
      idBOOLEAN ← MakeNamedType["BOOLEAN"L, typeBOOLEAN];
      tCtx ← makenewctx[lZ];
      (seb+typeBOOLEAN)↑ ← SERecord[mark3: TRUE, mark4: TRUE,
        sebody: constructor[
            enumerated[
            ordered: TRUE,
            valuectx: tCtx,
            nvalues: 2]]];
    [] ← MakeConstant["FALSE"L, tCtx, idBOOLEAN, 0];
    [] ← MakeConstant["TRUE"L, tCtx, idBOOLEAN, 1];
    resetctxlist[tCtx];
  idCARDINAL ← MakeSubrangeType["CARDINAL"L, 0, AltoDefs.maxword];
  [] ← MakeNamedType["WORD"L, UnderType[idCARDINAL]];
  -- make REAL type
    typeREAL ← makenonctxse[SIZE[real constructor SERecord]];
    (seb+typeREAL)↑ ← SERecord[mark3: TRUE, mark4: TRUE,
      sebody: constructor[real[rangetype: idINTEGER]]];
    idREAL ← MakeNamedType["REAL"L, typeREAL];
  -- make STRING type
    rSei ← MakeRecord[nFields:3, nBits:2*WordLength];
    [] ← MakeField["length"L, idCARDINAL, [wd:0, bd:0], WordLength];
    sei ← MakeField["maxlength"L, idCARDINAL, [wd:1, bd:0], WordLength];
    (seb+sei).writeonce ← TRUE;
    tSei ← makenonctxse[SIZE[array constructor SERecord]];
    (seb+tSei)↑ ← SERecord[mark3: TRUE, mark4: TRUE,
      sebody: constructor[array[
            packed: TRUE,
            indextype: idCARDINAL,  -- a fudge
            componenttype: idCHARACTER,
            comparable: FALSE,
            lengthUsed: FALSE]]];
    sei ← MakeField["text"L, tSei, [wd:2, bd:0], 0];
    tSei ← MakePointerType[MakeNamedType["StringBody"L, rSei]];
    idSTRING ← MakeNamedType["STRING"L, tSei];
    typeSTRING ← UnderType[idSTRING];
  -- make LOCK type
    rSei ← MakeRecord[nFields:1, nBits:WordLength];
    (seb+rSei).unifield ← FALSE;
    [] ← MakeField[NIL, idANY, [wd:0, bd:0], WordLength];
    idLOCK ← MakeNamedType["MONITORLOCK"L, rSei];
    typeLOCK ← UnderType[idLOCK];
  -- make CONDITION type
    rSei ← rSei ← MakeRecord[nFields:2, nBits:2*WordLength];
    [] ← MakeField[NIL, idANY, [wd:0, bd:0], WordLength];
    [] ← MakeField["timeout"L, idCARDINAL, [wd:1, bd:0], WordLength];
    typeCONDITION ← UnderType[MakeNamedType["CONDITION"L, rSei]];
  -- make a universal pointer type
    ptrSei ← MakePointerType[typeANY];
  -- enter the Boolean constants
    idTRUE ← MakeConstant["TRUE"L, outerCtx, idBOOLEAN, 1];
    idFALSE ← MakeConstant["FALSE"L, outerCtx, idBOOLEAN, 0];
  -- make a universal NIL
    [] ← MakeConstant["NIL"L, outerCtx, ptrSei, 0];
  -- make a neutral entry for error recovery
    seAnon ← MakeVariable[
      name: "?"L,
      ctx: outerCtx,
      type: typeANY,
      offset: [wd:0, bd:0],
      nBits: WordLength];
  -- predeclare UNWIND
    tSei ← makenonctxse[SIZE[transfer constructor SERecord]];
    (seb+tSei)↑ ← SERecord[mark3: TRUE, mark4: TRUE,
      sebody: constructor[
          transfer[
            mode: error,
            inrecord: recordCSENull,
            outrecord: recordCSENull]]]];
    idUNWIND ← MakeConstant["UNWIND"L, outerCtx, tSei,
          ControlDefs.ControlLink[procedure[
            gfi: ControlDefs.GFTNull,
            ep: ControlDefs.EPRange-1,
            tag: procedure]]];
  -- make some constants
    BEGIN
    tC0 ← [literal[info: [word[index: LitDefs.FindLiteral[0]]]]];
    tC1 ← [literal[info: [word[index: LitDefs.FindLiteral[1]]]]];
    END;
  resetctxlist[outerCtx];
  RETURN
  END;
```

```
SubStringDescriptor: TYPE = StringDefs.SubStringDescriptor;

MakeNamedType: PROCEDURE [s: STRING, type: SEIndex] RETURNS [sei: ISEIndex] =
  BEGIN
  desc: SubStringDescriptor ← [base:s, offset:0, length:s.length];
  sei ← makectxse[EnterString[@desc], dataPtr.outerCtx];
    BEGIN  OPEN (seb+sei);
    idtype ← typeTYPE;  idinfo ← type;  idvalue ← TreeDefs.empty;
    writeonce ← constant ← TRUE;
    extended ← public ← linkSpace ← FALSE;
    mark3 ← mark4 ← TRUE;
    END;
  RETURN
  END;

MakeBasicType: PROCEDURE
      [s: STRING, code: [0..16), ordered: BOOLEAN, nBits: CARDINAL]
      RETURNS [ISEIndex] =
  BEGIN  -- makes an se entry for a built-in type --
  sei: CSEIndex = makenonctxse[SIZE[basic constructor SERecord]];
  (seb+sei)↑ ← SERecord[mark3: TRUE, mark4: TRUE,
      sebody: constructor[
          basic[ordered:ordered, code:code, length:nBits]]];
  RETURN [MakeNamedType [s, sei]]
  END;

MakeConstant: PROCEDURE
      [name: STRING, ctx: CTXIndex, type: SEIndex, value: UNSPECIFIED]
      RETURNS [sei: ISEIndex] =
  BEGIN  -- makes an se entry for a built-in constant --
  desc: SubStringDescriptor ← [base:name, offset:0, length:name.length];
  sei ← makectxse[EnterString[@desc], ctx];
    BEGIN  OPEN (seb+sei);
    idtype ← type;  idinfo ← 0;  idvalue ← value;
    writeonce ← constant ← TRUE;
    extended ← public ← linkSpace ← FALSE;
    mark3 ← mark4 ← TRUE;
    END;
  RETURN
  END;

MakeVariable: PROCEDURE
      [name: STRING, ctx: CTXIndex, type: SEIndex, offset: BitAddress, nBits: CARDINAL]
      RETURNS [sei: ISEIndex] =
  BEGIN
  desc: SubStringDescriptor ← [base:name, offset:0, length:name.length];
  sei ← makectxse[EnterString[@desc], ctx];
    BEGIN  OPEN (seb+sei);
    idtype ← type;  idvalue ← offset;  idinfo ← nBits;
    writeonce ← constant ← public ← extended ← linkSpace ← FALSE;
    mark3 ← mark4 ← TRUE;
    END;
  RETURN
  END;


rCtx: CTXIndex;
seChain: ISEIndex;

MakeRecord: PROCEDURE [nFields, nBits: CARDINAL] RETURNS [rSei: recordCSEIndex] =
  BEGIN
  rSei ← LOOPHOLE[makenonctxse[SIZE[notlinked record constructor SERecord]]];
  rCtx ← makenewctx[lZ];
  (ctxb+rCtx).selist ← seChain ← makeSEChain[rCtx, nFields, FALSE];
  (seb+rSei)↑ ← SERecord[mark3: TRUE, mark4: TRUE,
      sebody: constructor[
          record[
              machineDep: TRUE,
              unifield: nFields = 1,
              argument: FALSE,
              defaultFields: FALSE,
              fieldctx: rCtx,
              length: nBits,
              comparable: FALSE,
              privateFields: FALSE,
              lengthUsed: FALSE,
              monitored: FALSE,
              variant: FALSE,
              linkpart: notlinked[]]]];
    RETURN
    END;

  MakeField: PROCEDURE
        [name: STRING, type: SEIndex, offset: BitAddress, nBits: CARDINAL]
        RETURNS [sei: ISEIndex] =
    BEGIN
    desc: SubStringDescriptor;
    hti: HTIndex;
    IF name # NIL
      THEN
        BEGIN
        desc ← [base:name, offset:0, length:name.length];
        hti ← EnterString[@desc];
        END
      ELSE hti ← HTNull;
    sei ← seChain;  seChain ← NextSe[seChain];
    fillctxse[sei, hti, FALSE];
      BEGIN  OPEN (seb+sei);
      idtype ← type;  idvalue ← offset;  idinfo ← nBits;
      writeonce ← constant ← public ← extended ← linkSpace ← FALSE;
      mark3 ← mark4 ← TRUE;
      END;
    RETURN
    END;

  MakePointerType: PROCEDURE [refType: SEIndex] RETURNS [sei: CSEIndex] =
    BEGIN
    sei ← makenonctxse[SIZE[pointer constructor SERecord]];
    (seb+sei)↑ ← SERecord[mark3: TRUE, mark4: TRUE,
        sebody: constructor[
            pointer[
              ordered: FALSE,
              readonly: FALSE,
              basing: FALSE,
              pointedtotype: refType,
              dereferenced: FALSE]]];
    RETURN
    END;

  MakeSubrangeType: PROCEDURE
        [s: STRING, origin: INTEGER, range: CARDINAL]
        RETURNS [ISEIndex] =
    BEGIN
    sei: CSEIndex;
    sei ← makenonctxse[SIZE[subrange constructor SERecord]];
    (seb+sei)↑ ← SERecord[mark3: TRUE, mark4: TRUE,
        sebody: constructor[
            subrange[
              filled: TRUE,
              empty: FALSE,
              flexible: FALSE,
              rangetype: dataPtr.idINTEGER,
              origin: origin,
              range: range]]];
    RETURN [MakeNamedType[s, sei]]
    END;


  LockId: PUBLIC PROCEDURE RETURNS [HTIndex] =
    BEGIN
    desc: SubStringDescriptor ← [base:"LOCK"L, offset:0, length:("LOCK"L).length];
    RETURN [EnterString[@desc]]
    END;


  P1Unit: PUBLIC PROCEDURE [tableSeg: SegmentDefs.FileSegmentHandle]
      RETURNS [success: BOOLEAN] =
    BEGIN  OPEN SegmentDefs;
    TableDefs.AddNotify[P1Notify];
    PrefillSymbols[];
    SwapIn[tableSeg];
    [complete:success, nErrors:dataPtr.nErrors] ←
      P1Defs.Parse[dataPtr.sourceStream, LOOPHOLE[FileSegmentAddress[tableSeg]]];
    Unlock[tableSeg];  SwapOut[tableSeg];
    TableDefs.DropNotify[P1Notify];
    RETURN
    END;


 -- initialization code
  CompilerDefs.MakeSwappable[P1Defs.Scanner, pass1];
  CompilerDefs.MakeSwappable[P1Defs.Parser, pass1];
  CompilerDefs.MakeSwappable[P1Defs.Pass1T, pass1];

  END.
```