```
-- BcdTab.Mesa  Edited by Sandman on May 12, 1978  8:48 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [CharsPerWord],
  BcdDefs: FROM "bcddefs" USING [httype, PackedString, sstype],
  BcdTabDefs: FROM "bcdtabdefs" USING [
    HTIndex, HTNull, HTRecord, HVIndex, HVLength],
  InlineDefs: FROM "inlinedefs" USING [BITAND, BITXOR],
  StringDefs: FROM "stringdefs" USING [
    AppendChar, AppendSubString, EqualSubStrings, EquivalentSubStrings,
    SubString, SubStringDescriptor],
  TableDefs: FROM "tabledefs" USING [
    AddNotify, Allocate, DropNotify, TableBase, TableIndex, TableNotifier,
    TrimTable];

DEFINITIONS FROM BcdTabDefs, BcdDefs;

BcdTab: PROGRAM IMPORTS TableDefs, StringDefs EXPORTS BcdTabDefs
  SHARES BcdTabDefs =
  BEGIN

  SubString: TYPE = StringDefs.SubString;

  -- tables defining the current symbol table

  ht: DESCRIPTOR FOR ARRAY --HTIndex-- OF HTRecord;

  hashvec: DESCRIPTOR FOR ARRAY --HVIndex-- OF HTIndex;
  htb: TableDefs.TableBase;        -- hash table
  ssb: POINTER TO BcdDefs.PackedString;   -- id string

  updatebases: TableDefs.TableNotifier =
    BEGIN OPEN BcdDefs;
    htb ← base[httype];
    ssb ← LOOPHOLE[base[sstype], POINTER TO BcdDefs.PackedString];
    hashvec ← DESCRIPTOR[htb, LENGTH[hashvec]];
    ht ← DESCRIPTOR[htb+LENGTH[hashvec]*SIZE[HTIndex], LENGTH[ht]];
    RETURN
    END;

  allocatehash: PROCEDURE RETURNS [hti: HTIndex] =
    BEGIN OPEN TableDefs, BcdDefs;
    next: TableIndex = Allocate[httype, SIZE[HTRecord]];
    hti ← LENGTH[ht];
    IF hti*SIZE[HTRecord]+LENGTH[hashvec] # LOOPHOLE[next, INTEGER] THEN
      ERROR;
    ht ← DESCRIPTOR[BASE[ht], LENGTH[ht]+1];
    ht[hti] ← HTRecord[link: HTNull, offset: ssb.string.length+1];
    RETURN [hti-1]
    END;

  -- variables for building the symbol string

  ssw: TableDefs.TableIndex;
  StringOverlay: TYPE = MACHINE DEPENDENT RECORD [
    length, maxlength: CARDINAL];
  StringPointer: TYPE = POINTER TO StringOverlay;
  StringHeaderSize: CARDINAL = SIZE[StringOverlay];


  tableopen: BOOLEAN ← FALSE;

  BcdTabInit: PUBLIC PROCEDURE =
    BEGIN OPEN TableDefs;
    IF tableopen THEN BcdTabErase[];
    hashvec ← DESCRIPTOR[NIL, HVLength];
    TableDefs.AddNotify[updatebases];
    BcdTabReset[];
    tableopen ← TRUE;
    RETURN
    END;

  BcdTabErase: PUBLIC PROCEDURE =
    BEGIN
    tableopen ← FALSE;
    TableDefs.DropNotify[updatebases];
```

```
    RETURN
    END;

  BcdTabReset: PUBLIC PROCEDURE =
    BEGIN OPEN BcdDefs;
    i: HVIndex;
    nullss: StringDefs.SubStringDescriptor ← [base:, offset:, length:0];
    TableDefs.TrimTable[sstype, 0];
    TableDefs.TrimTable[httype, 0];
    [] ← TableDefs.Allocate[httype, HVLength*SIZE[HTIndex]];
    hashvec ← DESCRIPTOR[htb, HVLength];
    FOR i IN HVIndex DO hashvec[i] ← HTNull ENDLOOP;
    ht ← DESCRIPTOR[htb+LENGTH[hashvec]*SIZE[HTIndex], 0];
    ssw ← TableDefs.Allocate[sstype, StringHeaderSize] + StringHeaderSize;
    ssb.string ← [length: 0, maxlength: 0, text:];
    [] ← allocatehash[];
    IF EnterString[@nullss] # HTNull THEN ERROR;
    RETURN
    END;


  -- hash entry creation

  EnterString: PUBLIC PROCEDURE [s: SubString] RETURNS [hti: HTIndex] =
    BEGIN OPEN StringDefs, BcdDefs;
    hvi: HVIndex;
    desc: SubStringDescriptor ← [base:@ssb.string, offset:, length:];
    CharsPerWord: CARDINAL = AltoDefs.CharsPerWord;
    offset, length, nw: CARDINAL;
    ssi: TableDefs.TableIndex;
    hvi ← hashvalue[s];
    FOR hti ← hashvec[hvi], ht[hti].link UNTIL hti = HTNull
      DO
      desc.offset ← ht[hti].offset;
      desc.length ← ssb.size[ht[hti].offset];
      IF EqualSubStrings[s, @desc] THEN RETURN [hti];
      ENDLOOP;
    offset ← ssb.string.length;  length ← s.length + 1;
    nw ← LOOPHOLE[offset+length+(CharsPerWord-1) - ssb.string.maxlength, CARDINAL]/CharsPerWord;
    IF nw # 0
      THEN
        BEGIN  ssi ← TableDefs.Allocate[sstype, nw];
        IF ssi # ssw THEN ERROR;
        ssw ← ssw + nw;
        LOOPHOLE[ssb, StringPointer].maxlength ← LOOPHOLE[ssb, StringPointer].maxlength + nw*CharsPerWo
**rd;
        END;
    AppendChar[@ssb.string, LOOPHOLE[s.length, CHARACTER]];
    AppendSubString[@ssb.string, s];
    hti ← allocatehash[];
    ht[hti].link ← hashvec[hvi];  hashvec[hvi] ← hti;
    RETURN
    END;


  -- the following copied from symboltable.mesa

  ignorecases: BOOLEAN ← FALSE;

  hashvalue: PROCEDURE [s: SubString] RETURNS [HVIndex] =
    BEGIN  -- computes the hash index for string s
    CharMask: PROCEDURE [CHARACTER, WORD] RETURNS [CARDINAL] =
      LOOPHOLE[InlineDefs.BITAND];
    mask: WORD = 137B;                -- masks out ASCII case shifts
    n: CARDINAL = s.length;
    b: STRING = s.base;
    v: WORD;
    v ← CharMask[b[s.offset], mask]*177B + CharMask[b[s.offset+(n-1)], mask];
    RETURN [InlineDefs.BITXOR[v, n*17B] MOD LENGTH[hashvec]]
    END;

  FindString: PUBLIC PROCEDURE [s: SubString] RETURNS [found: BOOLEAN, hti: HTIndex] =
    BEGIN
    OPEN StringDefs;
    desc: SubStringDescriptor;
    ss: SubString = @desc;
```

```
    hti ← hashvec[hashvalue[s]];
    WHILE hti # HTNull
      DO
      SubStringForHash[ss, hti];
      found ←
        IF ignorecases THEN EquivalentSubStrings[s,ss]
        ELSE EqualSubStrings[s,ss];
      IF found THEN RETURN;
      hti ← ht[hti].link;
      ENDLOOP;
    RETURN [FALSE, HTNull]
    END;

  FindEquivalentString: PUBLIC PROCEDURE [s: SubString] RETURNS [found: BOOLEAN, hti: HTIndex] =
    BEGIN
    oldcase: BOOLEAN = ignorecases;
    ignorecases ← TRUE;
    [found, hti] ← FindString[s];
    ignorecases ← oldcase;
    RETURN
    END;

  SubStringForHash: PUBLIC PROCEDURE [s: SubString, hti: HTIndex] =
    BEGIN -- gets string for hash table entry
    s.base ← @ssb.string;
    IF hti = HTNull
      THEN s.offset ← s.length ← 0
      ELSE
        BEGIN
        s.offset ← ht[hti].offset;
        s.length ← ssb.size[ht[hti].offset];
        END;
    RETURN
    END;

  END.
```