```
-- BcdTab.Mesa  Edited by Sandman on August 23, 1977  10:37 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  BcdDefs: FROM "bcddefs",
  BcdTableDefs: FROM "bcdtabledefs",
  BcdTabDefs: FROM "bcdtabdefs",
  InlineDefs: FROM "inlinedefs",
  StringDefs: FROM "stringdefs";

DEFINITIONS FROM BcdTabDefs, BcdDefs;

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

  SubString: TYPE = StringDefs.SubString;

  -- tables defining the current symbol table

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

  hashvec: DESCRIPTOR FOR ARRAY OF HTIndex = DESCRIPTOR[hashvector];
  htb: BcdTableDefs.TableBase;          -- hash table
  ssb: STRING;                          -- id string

  updatebases: BcdTableDefs.TableNotifier =
    BEGIN OPEN BcdTableDefs;
    htb ← base[httype];  ssb ← LOOPHOLE[base[sstype], STRING];
    ht ← DESCRIPTOR[htb, LENGTH[ht]];
    RETURN
    END;

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

  -- variables for building the symbol string

  ssw: BcdTableDefs.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 BcdTableDefs;
    IF tableopen THEN BcdTabErase[];
    AddNotify[updatebases];
    BcdTabReset[];
    tableopen ← TRUE;
    RETURN
    END;

  BcdTabErase: PUBLIC PROCEDURE =
    BEGIN OPEN BcdTableDefs;
    tableopen ← FALSE;
    DropNotify[updatebases];
    RETURN
    END;

  BcdTabReset: PUBLIC PROCEDURE =
    BEGIN OPEN BcdTableDefs;
    i: HVIndex;
    ResetTable[sstype];
    ResetTable[httype];
    FOR i IN HVIndex DO hashvector[i] ← HTNull ENDLOOP;
    ht ← DESCRIPTOR[NIL, 0];
    ssw ← Allocate[sstype, StringHeaderSize] + StringHeaderSize;
    LOOPHOLE[ssb, StringPointer].length ← LOOPHOLE[ssb, StringPointer].maxlength ← 0;
    [] ← allocatehash[];
    RETURN
    END;


  -- hash entry creation

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

END.
```