```
-- BcdTreePack.Mesa  Edited by Johnsson on April 12, 1978  5:13 PM

DIRECTORY
  BcdControlDefs: FROM "bcdcontroldefs",
  BcdDefs: FROM "bcddefs",
  BcdTabDefs: FROM "bcdtabdefs",
  BcdTreeDefs:  FROM "bcdtreedefs",
  SystemDefs: FROM "systemdefs",
  TableDefs: FROM "tabledefs";

DEFINITIONS FROM BcdTreeDefs;

BcdTreePack: PROGRAM
  IMPORTS TableDefs, SystemDefs
  EXPORTS BcdControlDefs, BcdTreeDefs = PUBLIC
  BEGIN

  treeopen: PRIVATE BOOLEAN ← FALSE;

  TreeLinkStack: PRIVATE TYPE = DESCRIPTOR FOR ARRAY OF TreeLink;

  Kstack: PRIVATE TreeLinkStack;
  Kindex: PRIVATE CARDINAL;

  tb: PRIVATE TableDefs.TableBase;      -- tree base

  updatebase: PRIVATE TableDefs.TableNotifier =
    BEGIN
    tb ← base[BcdDefs.treetype];  RETURN
    END;


  treeinit: PROCEDURE =
    BEGIN
    IF treeopen THEN treeerase[];
    Kstack ← allocStack[100];  Kindex ← 0;
    TableDefs.AddNotify[updatebase];
    treeopen ← TRUE;  RETURN
    END;

  treeerase: PROCEDURE =
    BEGIN
    treeopen ← FALSE;
    TableDefs.DropNotify[updatebase];
    freeStack[Kstack];  RETURN
    END;


  allocStack: PRIVATE PROCEDURE [size: CARDINAL] RETURNS [s: TreeLinkStack] =
    BEGIN
    OPEN SystemDefs;
    base: POINTER;
    base ← AllocateSegment[size*SIZE[TreeLink]];
    s ← DESCRIPTOR[base, SegmentSize[base]/SIZE[TreeLink]];
    RETURN
    END;

  freeStack: PRIVATE PROCEDURE [s: TreeLinkStack] =
    BEGIN
    OPEN SystemDefs;
    IF LENGTH[s] # 0 THEN FreeSegment[BASE[s]];
    RETURN
    END;

  expandStack: PRIVATE PROCEDURE [s: TreeLinkStack, delta: CARDINAL] RETURNS [t: TreeLinkStack] =
    BEGIN
    i: CARDINAL;
    t ← allocStack[LENGTH[s]+delta];
    FOR i IN [0 .. MIN[LENGTH[s], LENGTH[t]]) DO t[i] ← s[i] ENDLOOP;
    freeStack[s];  RETURN
    END;


  TreeStackError: PRIVATE ERROR [CARDINAL] = CODE;

  mlpush: PROCEDURE [v: TreeLink] =
```

```
    BEGIN
    IF Kindex >= LENGTH[Kstack] THEN  Kstack ← expandStack[Kstack, 25];
    Kstack[Kindex] ← v;  Kindex ← Kindex+1;
    RETURN
    END;

  mlpop: PROCEDURE RETURNS [TreeLink] =
    BEGIN
    IF Kindex = 0 THEN ERROR;
    RETURN [Kstack[Kindex←Kindex-1]]
    END;

  maketree: PROCEDURE [name: NodeName, count: INTEGER] RETURNS [TreeLink] =
    BEGIN
    nsons: CARDINAL = ABS[count];
    node: TreeIndex = TableDefs.GetChunk[TreeNodeSize+nsons];
    p: TreeXIndex;
    d: INTEGER;
    IF nsons > Kindex THEN ERROR TreeStackError[Kindex];
    p ← LOOPHOLE[node + TreeNodeSize + (IF count<0 THEN 0 ELSE nsons-1)];
    d ← IF count<0 THEN 1 ELSE -1;
    THROUGH [1 .. nsons]
      DO
      (tb+p).soni ← Kstack[Kindex←Kindex-1];  p ← p + d;
      ENDLOOP;
    (tb+node).name ← name;  (tb+node).nsons ← nsons;
    RETURN[TreeLink[subtree[index: node]]]
    END;

  makelist: PROCEDURE [size: INTEGER] RETURNS [TreeLink] =
    BEGIN
    pushlist[size];
    RETURN [mlpop[]]
    END;


  pushtree: PROCEDURE [name: NodeName, count: INTEGER] =
    BEGIN
    mlpush[maketree[name, count]];
    RETURN
    END;

  pushlist: PROCEDURE [size: INTEGER] =
    BEGIN
    nsons: CARDINAL = ABS[size];
    node: TreeIndex;
    p: TreeXIndex;
    d: INTEGER;
    SELECT nsons FROM
      1 =>  NULL;
      0 =>  mlpush[empty];
      ENDCASE =>
        BEGIN
        IF nsons > Kindex THEN ERROR TreeStackError[Kindex];
        IF nsons IN (0..MaxNSons]
          THEN
            BEGIN
            node ← TableDefs.GetChunk[TreeNodeSize+nsons];
            p ← LOOPHOLE[node + TreeNodeSize+(nsons-1)];
            END
          ELSE
            BEGIN
            node ← TableDefs.GetChunk[TreeNodeSize+(nsons+1)];
            p ← LOOPHOLE[node + TreeNodeSize+nsons];
            (tb+p).soni ← endmark;  p ← p-1;
            END;
        IF size > 0
          THEN  d ← -1
          ELSE
            BEGIN  d ← 1;  p ← LOOPHOLE[node + TreeNodeSize];
            END;
        THROUGH [1 .. nsons]
          DO
          (tb+p).soni ← Kstack[Kindex ← Kindex-1];  p ← p+d;
          ENDLOOP;
        (tb+node).name ← list;
```

```
      (tb+node).nsons ← IF nsons IN (0..MaxNSons] THEN nsons ELSE 0;
      mlpush[TreeLink[subtree[index: node]]];
      END;
  RETURN
  END;


pushhash: PROCEDURE [hti: BcdTabDefs.HTIndex] =
  BEGIN
  mlpush[TreeLink[hash[index: hti]]];
  RETURN
  END;

pushsym: PRIVATE PROCEDURE [sti: BcdTabDefs.STIndex] =
  BEGIN
  mlpush[TreeLink[symbol[index: sti]]];
  RETURN
  END;


setsourceindex: PROCEDURE [source: CARDINAL] =
  BEGIN
  v: TreeLink = Kstack[Kindex-1];
  WITH v SELECT FROM
    subtree =>
      IF index = nullTreeIndex THEN ERROR
      ELSE (tb+index).sourceindex ← source;
    ENDCASE => ERROR;
  RETURN
  END;

setattribute: PROCEDURE [attr: Attribute, value: BOOLEAN] =
  BEGIN
  v: TreeLink = Kstack[Kindex-1];
  WITH v SELECT FROM
    subtree =>
      IF index = nullTreeIndex THEN ERROR
      ELSE SELECT attr FROM
        links => (tb+index).codelinks ← value;
        ENDCASE => ERROR;
    ENDCASE => ERROR;
  RETURN
  END;


freenode: PROCEDURE [node: TreeIndex] =
  BEGIN
  p: TreeXIndex;
  n: CARDINAL;
  IF node # nullTreeIndex
    THEN
      BEGIN  p ← LOOPHOLE[node + TreeNodeSize];
      IF (tb+node).name # list OR (tb+node).nsons # 0
        THEN
          BEGIN   n ← (tb+node).nsons;
          THROUGH [1 .. n]
            DO
            WITH (tb+p).soni SELECT FROM
              subtree =>  freenode[index];
              ENDCASE;
            p ← p+1;
            ENDLOOP;
          END
        ELSE
          BEGIN  n ← 1;
          UNTIL (tb+p).soni = endmark
            DO
            WITH (tb+p).soni SELECT FROM
              subtree =>  freenode[index];
              ENDCASE;
            n ← n+1;  p ← p+1;
            ENDLOOP;
          END;
      TableDefs.FreeChunk[node, TreeNodeSize+n];
      END;
```

```
  RETURN
  END;

freetree: PROCEDURE [t: TreeLink] RETURNS [TreeLink] =
  BEGIN
  WITH t SELECT FROM
    subtree => freenode[index];
    ENDCASE;
  RETURN [empty]
  END;


Ktop: PRIVATE PROCEDURE RETURNS [TreeLink] =
  BEGIN
  IF Kindex = 0 THEN ERROR TreeStackError[0];
  RETURN [Kstack[Kindex-1]]
  END;

KHeight: PRIVATE PROCEDURE RETURNS [CARDINAL] =
  BEGIN
  RETURN [Kindex]
  END;


-- procedures for tree testing

testtree: PROCEDURE [t: TreeLink, name: NodeName] RETURNS [BOOLEAN] =
  BEGIN
  RETURN [WITH t SELECT FROM
    subtree => index # nullTreeIndex AND (tb+index).name = name,
    ENDCASE => FALSE]
  END;

listlength: PROCEDURE [t: TreeLink] RETURNS [CARDINAL] =
  BEGIN
  node: TreeIndex;
  p: TreeXIndex;
  n: CARDINAL;
  IF t = empty THEN RETURN [0];
  WITH t SELECT FROM
    subtree =>
      BEGIN  node ← index;
      IF (tb+node).name # list THEN RETURN [1];
      n ← (tb+node).nsons;
      IF n # 0 THEN RETURN [n];
      FOR p ← LOOPHOLE[node+TreeNodeSize], p+1 UNTIL (tb+p).soni = endmark
        DO
        n ← n+1;
        ENDLOOP;
      RETURN [n]
      END;
    ENDCASE => RETURN [1]
  END;

listhead: PROCEDURE [t: TreeLink] RETURNS [TreeLink] =
  BEGIN
  node: TreeIndex;
  IF t = empty THEN ERROR;
  WITH t SELECT FROM
    subtree =>
      BEGIN  node ← index;
      IF (tb+node).name # list THEN RETURN [t];
      IF (tb+node).son1 # endmark THEN RETURN [(tb+node).son1];
      ERROR
      END;
    ENDCASE => RETURN [t]
  END;

listtail: PROCEDURE [t: TreeLink] RETURNS [TreeLink] =
  BEGIN
  node: TreeIndex;
  IF t = empty THEN ERROR;
  WITH t SELECT FROM
    subtree =>
      BEGIN  node ← index;
      IF (tb+node).name # list THEN RETURN [t];
```

```
      IF (tb+node).son1 # endmark
        THEN RETURN [(tb + LOOPHOLE[node+TreeNodeSize+(listlength[t]-1), TreeXIndex]).soni];
      ERROR;
      END;
    ENDCASE => RETURN [t]
  END;

-- procedures for tree traversal

scanlist: PROCEDURE [root: TreeLink, action: TreeScan] =
  BEGIN
  node: TreeIndex;
  p: TreeXIndex;
  n: CARDINAL;
  t: TreeLink;
  IF root # empty
    THEN
      WITH root SELECT FROM
        subtree =>
          BEGIN node ← index;
          IF (tb+node).name # list
            THEN action[root]
            ELSE
              BEGIN p ← LOOPHOLE[node + TreeNodeSize];
              IF (n ← (tb+node).nsons) # 0
                THEN
                  THROUGH [1 .. n]
                    DO
                    action[(tb+p).soni]; p ← p+1;
                    ENDLOOP
                ELSE
                  UNTIL (t←(tb+p).soni) = endmark
                    DO
                    action[t]; p ← p+1;
                    ENDLOOP;
              END;
          END;
        ENDCASE => action[root];
  RETURN
  END;

reversescanlist: PROCEDURE [root: TreeLink, action: TreeScan] =
  BEGIN
  node: TreeIndex;
  p: TreeXIndex;
  n: CARDINAL;
  IF root # empty
    THEN
      WITH root SELECT FROM
        subtree =>
          BEGIN node ← index;
          IF (tb+node).name # list
            THEN action[root]
            ELSE
              BEGIN n ← listlength[root];
              p ← LOOPHOLE[node + TreeNodeSize + n];
              THROUGH [1 .. n]
                DO
                p ← p - 1; action[(tb+p).soni];
                ENDLOOP;
              END;
          END;
        ENDCASE => action[root];
  RETURN
  END;

updatelist: PROCEDURE [root: TreeLink, action: TreeMap] RETURNS [TreeLink] =
  BEGIN
  node: TreeIndex;
  p: TreeXIndex;
  n: CARDINAL;
  t: TreeLink;
  IF root = empty THEN RETURN [empty];
  WITH root SELECT FROM
    subtree =>
      BEGIN node ← index;
```

```
      IF (tb+node).name # list THEN RETURN [action[root]];
      p ← LOOPHOLE[node + TreeNodeSize];
      IF (n ← (tb+node).nsons) # 0
        THEN
          THROUGH [1 .. n]
            DO
            (tb+p).soni ← action[(tb+p).soni];  p ← p+1;
            ENDLOOP
        ELSE
          UNTIL (t←(tb+p).soni) = endmark
            DO
            (tb+p).soni ← action[t];  p ← p+1;
            ENDLOOP;
      RETURN [root]
      END;
    ENDCASE => RETURN [action[root]];
  END;

END ...
```