Burt Rosenberg

Answer Set 6

Out: 19 October 1993

```
program treeHomework( input, output ) ;

{
  A solution to homework number 6
  Math 220/317 Fall 1993
  University of Miami
  Prof. Burt Rosenberg

   "Fun with trees", inserting, deleting and
   printing.
}


{========= types and const ==========}

const
   STRLEN = 25;
   EOS = chr(0) ;
   INFILE = 'test.txt' ;
   SPACES = '  ' ;
{   DEBUG = FALSE ; }
   DEBUG = TRUE ;

type

   MyStringType = array [0..STRLEN-1] of char ;
   { A string is an array terminated by an EOS }

   treeNodeP = ^ treeNode ;
   treeNode = record
      l, r : treeNodeP ;
      d : MyStringType ;
      c : integer ;
      end ;
   Tree = treeNodeP ;

{========= string stuff ============}

{see previous Answer sets of definitions of:
```

```
    function EmptyString( var s : MyStringType ) : boolean ;
    procedure StringPrint( var s : MyStringType ) ;
    procedure StringToLower( var s : MyStringType ) ;
    procedure OneWord( var retVal : MyStringType ; var f : TEXT) ;
    function StringCompare( var s, t : MyStringType ) : integer ;
    function StringEqual( var s, t : MyStringType ) : boolean ;

}

procedure StringCopy( VAR s, t : MyStringType ) ;
{
  copies source string t to destination string s
}
var i : integer ;
begin
   i := -1 ;
   repeat
      i := i + 1 ;
      s[i] := t[i] ;
   until s[i] = EOS
end ;


{=========== tree stuff ============}

function TreeCreate : Tree ;
begin
   TreeCreate := nil
end ;

Procedure TreePrint( t : Tree ) ;

   Procedure TPaux( t: Tree; level : integer ) ;
   var i : integer ;
   begin
      write(level:3) ;
      for i := 1 to level do write(SPACES) ;
      if t=nil then
         {terminal condition for recursions }
         writeln('-')
      else begin
         {recurse}
         write(t^.c:0,':') ;
         StringPrint(t^.d) ;
```

```
         writeln ;
         TPaux( t^.l, level+1 ) ;
         TPaux( t^.r, level+1 )
      end
   end ;
begin
   TPaux( t, 1 )
end ;

function TreeUpdate( t : Tree; w : MyStringType ) : Tree ;
{
   Given a tree and a datum, searches for datum in tree.
   Returns a tree with datum inserted (if not found) or
   the count of the datum incremented (if found)
}
var s : TreeNodeP ;
    j : integer ;
begin
   if t=nil then {insert to an empty tree} begin
      new(s) ;
      s^.l := nil ;
      s^.r := nil ;
      s^.c := 1 ;
      StringCopy( s^.d, w ) ;
      TreeUpdate := s
   end else begin
      j := StringCompare( t^.d, w ) ;
      if j<0 then           {recurse on right subtree}
         t^.r := TreeUpdate( t^.r, w )
      else if j>0 then      {recurse on left subtree}
         t^.l := TreeUpdate( t^.l, w )
      else                 {update found w}
         t^.c := t^.c + 1 ;
      TreeUpdate := t
   end
end ;



{deletion}

function detachRightmost( t : Tree; var n : treeNodeP ) : Tree ;
{
   Given a tree, returns the tree with the rightmost node taken
```

```
   off. This node is returned via the VAR variable n.

   Preconditions: t <> nil
}
begin
   if t^.r = nil then begin
      {terminal condition of recursion, rightmost is found}
      n := t ;
      detachRightmost := t^.l
   end else begin
      {ASSERT: t^.r<> nil }
      {recurse}
      t^.r := detachRightmost( t^.r, n ) ;
      detachRightmost := t
   end
end ;

function TreeDelete( t : tree; w : MyStringType ) : tree ;
{
   Given a tree and a data item to delete, returns the tree
   with the item deleted. If w is not in t, or t is nil,
   returns t.
}
var j : integer ;
    n : TreeNodeP ;
begin
   if t=nil then {delete from a nil tree leaves a nil tree.}
      TreeDelete := t
   else begin
      {ASSERT t <> nil }
      j := stringCompare( w, t^.d ) ;
      if j<>0 then begin {t is not the node to delete}
         if j<0 then {recurse left}
            t^.l := TreeDelete( t^.l, w )
         else {recurse right}
            t^.r := TreeDelete( t^.r, w ) ;
         TreeDelete := t
       end else begin
         {w=t^.d, delete this item. If a child is nil
         this is easy ... }
         if t^.r=nil then
            TreeDelete := t^.l
         else if t^.l = nil then
            TreeDelete := t^.r
```

```
         else begin
            {ASSERT neither t^.r or t^.l = nil }
            {Remark: we swap t with the rightmost node
            of the left subtree, because such a node is
            easy to delete.}
            n^.l := detachRightmost( t^.l, n ) ;
            n^.r := t^.r ;
            TreeDelete := n {new root of subtree}
         end ;
         dispose(t)
      end {end case both children non-nil}
   end {end case t not nil}
end { procedure TreeDelete } ;


{======= main line =========}

var
   f : TEXT ;
   s : MyStringType ;
   ta : tree ;
   i : integer ;

begin
   if NOT DEBUG then
      reset(f, INFILE ) ;
   ta := treeCreate ;
   if DEBUG then
      OneWord( s, input )
   else
      OneWord( s, f ) ;
   StringToLower( s ) ;

   while NOT emptystring(s) do begin
      if s[0] = '-' then begin {this is a delete}
         i := 0 ;
         repeat
           s[i] := s[i+1] ;
           i := i + 1 ;
         until s[i]=EOS ;
         ta := TreeDelete( ta, s )
      end else   {this is an insert/increment count }
         ta := TreeUpdate( ta, s ) ;
      if DEBUG then begin
```

```
         TreePrint( ta ) ;
         OneWord( s, input )
      end else
         OneWord( s, f )  ;
      StringToLower( s )
   end ;
   if NOT DEBUG then begin
      close(f) ;
      TreePrint( ta )
   end
end.

impala>a.out
romeo, romeo, wherefore art thou?
     ... {various stages of building the tree}
  1  2:romeo
  2    1:art
  3      -
  3      -
  2    1:wherefore
  3      1:thou
  4        -
  4        -
  3      -
-romeo
  1  1:art
  2    -
  2    1:wherefore
  3      1:thou
  4        -
  4        -
  3      -
```