Burt Rosenberg

Answer Set 4

Out: 28 September, 1993

```
program unique( input, output ) ;

{
  A solution to Homework4,
    Counting the Unique words in a text.

  Math 220/317 Fall 1993
  University of Miami
  Prof. Burt Rosenberg
}

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

const
  STRLEN = 25;
  EOS = chr(0) ;
  INFILE = 'test.txt' ;

type

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

  ListData = MyStringType ;
  List = ^ListNode ;
  ListNode = record
      data : ListData ;
      count : integer ;
      next : List ;
      end ;
  Handle = ^ListNode ;

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

function EmptyString( s : MyStringType ) : boolean ;
{
   Test if s is the empty string ;
}
begin
  EmptyString := (s[0]=EOS) ;
end ;
```

```
procedure StringPrint( s : MyStringType ) ;
{
   Print the given string.
}
var i : integer ;
begin
  i := 0 ;
  while ((s[i]<>EOS) AND (i<STRLEN)) do begin
    write(s[i]) ;
    i := i + 1 ;
  end ;
end ;

function StringToLower( s : MyStringType ) : MyStringType ;
{
   Converts all alpha's in s to lower case.
   Assumes that s is properly terminated w/ an EOS.
}
var i, d : integer ;
begin
  i := 0 ;
  d := ord('a') - ord('A') ;
  while ( s[i]<>EOS ) do begin
    if ( (s[i]>='A') AND (s[i]<='Z') )
      then s[i] := chr( ord(s[i]) + d ) ;
    i := i + 1 ;
  end ;
  StringToLower := s ;
end ;

function OneWord(var f : TEXT) : MyStringType ;
{
  OneWord will: scan past all delimiters, collect
     characters until the next delimiter.
  Delimiters include end of line.

  If called on an eof, or sequence of delimiters until
  and eof, returns the empty string.

  We will assume that a delimiter precedes an eof.
}
```

```
  function delimiter(ch : Char) : Boolean;
  {Checks to see if character is a delimiter ie isn't a letter}
  begin
    delimiter := (ch<'A') OR (ch>'z') OR ((ch<'a') AND (ch>'Z')) ;
  end ;

VAR
  ch   : Char;
  i    : Integer;
  wasADel, flag : Boolean ;
  s : MyStringType ;
BEGIN
  i := 0 ;
  flag := true ;
  { invariant: s[i] is the next open slot.}
  while flag do begin
    if (eof(f)) then
      flag := false
    else begin
      if eoln(f) then begin
         wasADel := true ;
         readln(f) ;
      end else begin
         read( f, ch ) ;
         wasADel := delimiter(ch) ;
      end ;
      { delimiter is properly set and the
        text cursor is properly advanced. }
      if (not wasADel) then begin
        { not a delimiter, concat. char onto string.}
        s[i] := ch ;
        if (i<(STRLEN-1)) then i := i + 1 ; {truncate too long words}
      end else
        { was a delimiter (includes nl) }
        if (i>0) then
          { this ends a word }
          flag := false ;
        { else there is nothing to do }
    end ; {not eof }
  end ; {while flag }
  s[i] := EOS ;
  OneWord := s;
end ;
```

```
function StringCompare( s, t : MyStringType ) : integer ;
{
   Compares to strings in lex. order.
   Input: s, t : the two strings to compare
   Returns:  -1 if s<t,
              0 if s=t
              1 if s>t
}
var i, j : integer ;
begin
  i := 0 ;
  while ((s[i]=t[i]) AND (s[i]<>EOS) AND (i<(STRLEN-1))) do
    i := i + 1 ;
  j := 0 ;
  if (s[i]<t[i]) then j := -1 ;
  if (s[i]>t[i]) then j := 1 ;
  StringCompare := j ;
end ;

function StringEqual( s, t : MyStringType ) : boolean ;
{
   Returns True is string s and t are equal.
}
begin
  StringEqual := ( 0 = StringCompare(s,t) ) ;
end ;



{=========== list stuff ============}

function ListCreate : List ;
{
   Returns a pointer to a new empty list
}
var p : List ;
begin
  new(p) ;
  p^.data[0] := EOS ; { this is a bad prograaming practice! }
  p^.count := 0 ;
  p^.next := NIL ;
  ListCreate := p ;
end ;
```

```
procedure ListAddAtHead( l : List ; d : ListData ) ;
{
  Adds data element at head of list l, making count 1
}
var p : List ;
begin
  new(p) ;
  p^.data := d ;
  p^.count := 1 ;
  p^.next := l^.next ;
  l^.next := p ;
end ;

procedure ListPrint( l : List ) ;
{
  Prints list l.
}
begin
  while (l^.next<>NIL) do begin
    write( l^.next^.count:5, ' ' ) ;
    StringPrint( l^.next^.data ) ;
    writeln ;
    l := l^.next ;
   end ;
end ;

procedure ListDestroy( l : List ) ;
{
  Frees all space allocated to l.
}
var p : List ;
begin
  while ( l<> NIL ) do begin
    p := l^.next ;
    Dispose(l) ;
    l := p ;
  end ;
end ;
```

```
function ListSearch( l : List; d : ListData ) : Handle ;
{
  Search for data d in list l. Returns a Handle to
  found element, actually a pointer to the node previous
  to the found node. If pointer is to the last node, then
  the search failed.
}
begin
  while (l^.next<>NIL) do begin
    if (StringEqual( l^.next^.data, d )) then break ;
    l := l^.next ;
  end ;
  ListSearch := l ;
end ;

function NotFound( h : Handle ) : boolean ;
{
  Interprets the result of ListSearch: returns true
  if and only if h is not the pointer to the last
  node of the list.
}
begin
  NotFound := (h^.next=NIL ) ;
end ;

function ListLength( l : List ) : integer ;
{
   Returns the length of list l.
}
var i : integer ;
begin
  i := 0 ;
  l := l^.next ;
  while ( l<>NIL ) do begin
    i := i + 1 ;
    l := l^.next ;
  end ;
  ListLength := i ;
end;
```

```
function ListSumCnt( l : List ) : integer ;
{
   Returns the sum of count fields over all
   elements in list l.
}
var i : integer ;
begin
  i := 0 ;
  l := l^.next ;
  while ( l<>NIL ) do begin
    i := i + l^.count ;
    l := l^.next ;
  end ;
  ListSumCnt := i ;
end ;

procedure ListIncCnt( h : Handle ) ;
{
  Increments element referenced by handle h,
  that is, node following node pointed to by h.
}
begin
  h^.next^.count := h^.next^.count + 1 ;
end ;


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

var
  f : TEXT ;
  s : MyStringType ;
  h : Handle ;
  l : List ;
  numWords, numUWords : integer ;

begin

  { create a list and open the text file.}
  l := ListCreate ;
  reset( f, INFILE ) ;
```

```
  s := OneWord( f ) ;
  s := StringToLower( s ) ;
  while ( NOT EmptyString(s) ) do begin
    { for each word, check the list...}
    h := ListSearch( l, s ) ;
    if ( NotFound(h) ) then
      {if not found, add word to head of the list,}
      ListAddAtHead( l, s )
    else
      {if found, increment the word's count.}
      ListIncCnt( h ) ;
    s := OneWord( f ) ;
    s := StringToLower( s ) ;
  end ;
  close(f) ;

  {report results and clean up.}
  numWords := ListSumCnt( l ) ;
  numUWords := ListLength( l ) ;
  writeln( numWords:5, ' Words in file,', numUWords:5, ' unique. They are:' ) ;

  ListPrint( l ) ;
  ListDestroy( l ) ;
end.

impala> cat > test.txt
It was organized in the best of times.
But it opened in the worst of times.
impala> a.out
   16 Words in file,   11 unique. They are:
    1 worst
    1 opened
    1 but
    2 times
    2 of
    1 best
    2 the
    2 in
    1 organized
    1 was
    2 it
```