Burt Rosenberg

Problem Set 4

OUT: 19 OCTOBER, 1992
DUE: 26 OCTOBER, 1992

Reading Assignment

Read

- Chapters 5 and 7.
- Review your class notes for merge-sort.

Goals

Practice with recursive algorithms, dictionary order on strings and use of subroutine units.

Assignment

Write a merge-sort for list of strings.

The starting point will be two subroutine packages, *string.psu* and *list.psu*, which will soon be placed in your home directory. Each is a Pascal Subroutine Unit. Neither is a complete program, they contain building blocks for programs. The pascal compiler will automatically and temporarily copy the files which contain these units into your programs it directed to do so by the,

```
%INCLUDE 'filename'
```

compiler directive.

These subroutine units do not contain all the routines needed to accomplish merge-sort. The code you must write falls into two groups.

1. Primitive list and string manipulation routines. Logically, these routines can be placed in list.psu and string.psu. However, to make grading your work easier, do not change list.psu and string.psu. Write these new routines in your ps4.pas.

   These routines include:

```
function le_string( s,t:stringType) : boolean ;
{An extension to string.psu which returns }
{true if s <= t, false otherwise.}

function remove_head( l:List) : listElemPntr ;
{An extension to list.psu. Detaches the first }
{element from list l and returns a pointer }
{to it.}

procedure insert_last( l:List; p:listElemPntr ) ;
{An extension to list.psu. Add the element pointed }
{to by p to the end of the list l.}
```

2. Routines dealing directly with the merge-sort algorithm. There include:

```
procedure split( L1,L2,L3: List ) ;
{Given a list L1 and two empty lists L2 and L3,}
{places approximately half the elements of L1 }
{on L2 and the rest on L3. Makes L1 empty.}
{Note: none of these need be passed var!}

procedure merge( L1,L2,L3: List ) ;
{Given two sorted lists L1 and L2 and an empty }
{list L3, merges L1 and L2 onto L3 so that L3 }
{is sorted. L1 and L2 become empty.}
{Note: none of these need be passed var!}
```

Depending on the details of your merge-sort, you may want to define additional operations on lists and strings. For example. it seems likely that a superior implementation of merge-sort would require the list subroutine,

```
procedure append( L1,L2:List ) ;
{Given two lists L1 and L2, appends L2 to the}
{end of L1, making L2 empty.}
```

Recall that merge-sort is a recursive algorithm. In class we developed a pseudo-code implementation of merge-sort similar to the following:

```
merge-sort(l:List) ;

if l is a basis case for the recursion,
  do nothing.
else
  create lists lA and lB
  split(l,lA,lB)
  merge-sort(lA)
  merge-sort(lB)
  merge(lA,lB,l)
  destroy lists lA and lB
end {merge-sort}
```

Your finished program, ps4.pas, will therefore be structure something like,

```
program ps4(input,output,datafile) ;
%INCLUDE 'string.psu'
%INCLUDE 'list.psu'
{--- extensions to string.psu ---}
Code for le_string.
{--- extensions to list.psu ---}
Code for remove-head and insert-last.
{--- merge sort routines ---}
Code for split, merge and merge-sort.
{--- main ---}
var
...
begin
  Open and read datafile into a list.
  List that list (unsorted).
  Sort the list.
  List the list, now it should be sorted.
end.
```

Remember, work in small steps. Compile often. Debug thoroughly. Use a test suite to organize your debugging, starting with simple test cases first.

Getting Started

Begin by copying string.psu, list.psu, ps4test.pas and sampledata.dat into a subdirectory [.ps4] which you create. Compile and run the test to see if all went well. Then read carefully string.psu and list.psu, making sure you understand the list structure and how it can be manipulated. Here is how the copying and compiling goes. I assume you are logged in to your home directory.

```
$ create /directory [.ps4]
$ copy string.psu [.ps4]
$ copy list.psu [.ps4]
$ copy ps4test.pas [.ps4]
$ copy sampledata.dat [.ps4]
$ cd [.ps4]
$ copy sampledata.dat datafile.dat
$ pascal ps4test
$ link ps4test
$ run ps4test
   8 words in text
   5 unique, they are:
   1 blue
   1 red
   1 two
   4 fish
   1 one
$
```