$\boxed{\text{Burt Rosenberg}}$

Problem Set 7

OUT: 21 OCTOBER, 1993
DUE: 29 OCTOBER, 1993

Goals

Practice with trees and heaps. Exercise in algorithm implementation.

Reading Assignment

Read Chapters 6–9, 11 and 12 from *Algorithms.*

Programming Assignment

You are to implement a *heap* using a pointer-based implementation of ordered binary trees. Implement heap operations *insert* and *delete-min.* Two versions of your program are required: a *test* and a *run* version. Switching between versions should be accomplished by changing the value of a `CONST` variable and recompiling the program.

Use the following `TYPE` definitions as a model for your heap:

```
type
  TreeNodeP = ^ TreeNodeR ;
  TreeNodeR = record
      data : DataType ;
      { perhaps some more fields }
      l, r : TreeNodeP ;
      end ;
  HeapR = record
      h : TreeNodeP ;
      { perhaps some more fields }
      end ;
  Heap = ^ HeapR ;
```

The heap operations you are to implement are:

```
function HeapCreate : Heap ;
{ Returns a pointer to a newly created, empty heap.}
```

```
function HeapEmpty( h : Heap ) : Boolean ;
{ Returns True if h is empty. }

procedure HeapInsert( h : Heap ; d : DataType ) ;
{ Given a heap h, and a datum d, mutates h into a new heap
  which is the elements of h with d inserted. }

function HeapDeleteMin( h : Heap ) : DataType ;
{ Given a heap h, returns the minimum datum of the heap,
  and mutates h to be h with the minimum deleted. }
```

Here is a description of the two versions of the program:

1. The *test version* begins with an empty heap and accepts text from the keyboard. Any word not beginning with a "-" is inserted into the heap. Words beginning with a "-" signal that a delete-min operation is to be performed. After each insert or delete-min the tree is printed pre-order in the fashion of the previous homework.

2. The *run version* begins with an empty heap and accepts text from a file. Words in the file are inserted into the heap or a delete-min is performed, according to whether or not the word does not begin with a "-", as in the test version. However, no pre-order output is given between words. Instead, at end-of-file, the words are printed in ascending dictionary order using repeated applications of delete-min. (Essentially accomplishing heap-sort!)

Algorithmic Notes

You will have to find the rightmost leaf of the bottom level of the tree. How can this be done? Here is a hint for one possible approach. Suppose the heap knowns how big it is, perhaps a field is included in HeapR which records this. Mentally number the tree nodes according to the following pattern:

```
          1                           1
         / \                         / \
        2   3                      10    11
       /|   |\       <==>         / |    | \
      4 5   6 7                100 101 110 111
     /| |\ /| |\              /   |   |\ /|   |   \
    8 9   ...             1000 1001     ...
```

On the left the nodes are numbered in decimal, on the right, they are numbered in binary. Note that the number of a node, when written in binary, describes the path from the root to the node. Reading the binary representation from left to right, discarding the initial 1, read a 0 as "descend left" and a 1 as "descend right."

Let n be the number of the leaf node you are trying to find. Find the integer k such that,

$$2^k \leq n < 2^{k+1}.$$

Then node n is in level k, where level 0 is the level of the root. Repeat the following,

1. If the leftmost bit of n is 1, strip it off,

   $$\text{if } n \geq 2^k \text{ then } n \leftarrow n - 2^k$$

2. Shift all the bits in n to the left one place,

   $$n \leftarrow 2n$$

3. Test the leftmost bit for one or zero, and descend in the tree accordingly,

   $$\text{if } n \geq 2^k \text{ then go right, else go left}$$

4. Termination conditions: Since you are searching for a leaf, you do not have to count bits. When further progress would mean following a nil pointer, you have arrived. Else repeat from Step 1.

Extra Credit

Implement a linear-time build-heap. A possible approach is,

- Build a complete tree on nodes carrying all the elements, not worrying about heap order.
- Traverse the tree, bottommost rightmost node first, moving leftwards and upwards, to slowly establish heap order everywhere.

You might make use of the Breadth-First tree search algorithm described by Sedgewick, page 48, to organize both the build and "heapify" traversals.