EECS 345: Programming Language Concepts

Programming Exercise 3

Due Thursday, March 26

The purpose of this assignment is to get practice at writing C code, and in particular, to practice using structs, pointers, and allocating and freeing memory.

I have written a heap class for you. As you recall from 233, a heap is a binary tree where the minimum element is at the root, and each node has the property the the node's element is not smaller than its parent nor larger than either of its children. The heap is implemented as an array where the root is stored at index 1. For each node, the index of its left child is 2*(index of node) and the index of the right child is 2*(index of node) + 1.

`samplesort.c` is an example that uses the heap to sort the command line arguments.

`huffman.c` is the start of a file that uses the heap to create the Huffman codes for each character in a file. If you remember Huffman coding from 233 or 340, it is a technique to compress a file by using variable length binary encodings for each letter instead of the standard 8 characters per byte. Those character that appear a lot (such as ' ') should be coded with very few bits, but those that rarely appear should use more bits. This homework will not be compressing files (but you are welcome to do that if you wish), just determining the Huffman code for each character.

Huffman coding uses a heap to build a tree. Each leaf node of the tree will store a character. You get the binary encoding for the character by traversing from the root of the tree. If you go left, you use a '0' and if you go write you use a '1'. For example, if you go from the root to the 'e' leaf node by going left, left, right, left, then the encoding of 'e' will be 0010.

You create the Huffman tree by creating a tree node for each character, you make the weight of the node equal to the number of times the character appears in the file, and each node will have no children (because these nodes will be the leaves of the tree). You place all the tree nodes in a heap so that the minimum weight node will be at the root of the tree. Then you repeatedly remove two nodes from the heap, create a new tree node to be the parent of these two nodes, make the weight of the new node equal to the sum of the weights of its two children, and you put this node into the heap. When the heap has only one node left in it, that node is the root of the Huffman tree.

What you are to do

1. Complete the code for `huffman.c`. You will find instructions in the comments inside the file. Be sure to use the special allocation and deallocation functions detailed below.

2. Complete the `heapSort` function in `heap.c`. To do heap sort properly, the heap sort should be "in place", and you should not allocate any memory in this routine.

Important Rules

To help you allocate and free correctly, the file `memory.c` provides a few useful functions:

`eecs345_malloc`, `eecs345_calloc`, and `eecs345_free`. You should call these functions instead of `malloc`, `calloc` and `free`. The functions work exactly the same except that they keep track of the memory allocations. At the end of your program, call `test_for_memoryleaks()`, and the function will print a message if you have memory allocated that you did not free or if you tried to free memory more than once.

How To Use the Heap and Memory Code

You should place `#include "heap.h"` and/or `#include "memory.h"` at the top of any file that will use these routines. When you compile your program, you should compile it with `heap.c` and `memory.c`.

Here is a simple example: `gcc -o samplesort samplesort.c heap.c memory.c`

If you want to use a debugger, you should compile with the debug flag: `gcc -g -o samplesort samplesort.c heap.c memory.c`

(If you do not want to recompile `memory.c` everytime, recall how to use a makefile from EECS 293.)