CSCI 1120 (Low-Level Computing), Spring 2011

Homework 6

Credit: 30 points.

1 Reading

Be sure you have read the assigned readings for classes through 11/28.

2 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course number and the assignment (e.g., "csci 1120 homework 6"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (15 points) Write a C program that sorts the lines in a text file using the library function `qsort`. The program should take the name of the file to sort as a command-line argument (and print appropriate error messages if none is given or the one given cannot be opened) and write the result of the sort to standard output.

   To do this, I think you will need to read the whole file into memory. There are various ways to do this, but the method I have in mind (for learning purposes) involves reading the whole file into memory and then building an array of pointers to individual lines. Here is a function you can use (on Linux systems anyway) to determine how much memory to allocate for the file:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
/* returns size of file *filename in bytes, or -1 on error */
int filesize(char * filename) {
    struct stat status;
    if (stat(filename, &status) == -1) {
        return -1;
    }
    else {
        return (int) status.st_size;
    }
}
```

(The above description is deliberately not very detailed. More detailed hints about how to proceed available on request, but I want you to think about the problem yourself first.)

Hints:

- You can use the library function `strcmp` to compare two strings.
- The "sample programs" page contains an example of using `qsort`. (Notice that while it checks the result of the sort for correctness, your program does not need to do that; instead you are to print the results of the sort.)
  sort-improved-2.c[1]

2. (15 points) Write a C program that implements and tests one of the linked-list-like data structures listed below. For ease of coding, just make the program self-contained (rather than having it get input from a human), similar to the example implementation of unordered singly linked lists we looked at in class. (You can find it linked from the course "sample programs" page here[2]. You can put the whole program in one file, or separate it into multiple files as I did in the example.

   Possible data structures are the following:

   - Sorted singly linked list (of `int`s or another data type). Include functions to
     - create an empty list
     - free all memory associated with a list
     - add an element
     - remove a selected element (everywhere it occurs, or only the first occurrence),
     - search for a selected element (you could have it return something consistent with the C89 idea of a boolean — zero is "false" and anything nonzero is "true" — or have it return a count of how many times the element occurs in the list)
     - print all elements of the list to a specified output stream (you could also include a parameter for the print format, as in the example program, or just hardcode something).

     You can add additional functions for extra credit (amount of credit depending on difficulty).
   - Unsorted or sorted doubly linked list (of `int`s or another data type). A doubly linked list is one in which each element has pointers to both the next element and the previous element. Include functions as for the sorted singly linked list described above, plus a function to print the elements in reverse order. You can add additional functions for extra credit (amount of credit depending on difficulty).
   - Binary search tree (of `int`s or another data type). This is a tree data structure in which every node n has the property that all nodes in its left subtree store values smaller than the value in n and all elements in its right subtree store values larger than the value in

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2011spring/SamplePrograms/Programs/sort-improved-2.c`

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2011fall/SamplePrograms/`

n. (If you haven't encountered this data structure before, I can explain in more detail.) Include functions to

  – create an empty tree
  – free all memory associated with a tree
  – add an element (for simplicity, you can disallow duplicates, and simply do nothing if a request is made to add something that's already present)
  – search for a selected element (you could have it return something consistent with the C89 idea of a boolean — zero is "false" and anything nonzero is "true")
  – print all elements of the tree to a specified output stream (you could also include a parameter for the print format, as in the example program, or just hardcode something).

  (This list doesn't include a function to remove elements because that's trickier.) You can add additional functions for extra credit (amount of credit depending on difficulty).

- Some other data type whose implementation involves dynamically allocated storage and pointers (but ask me before you do this).

You're welcome to use the example from class as a model or starting point, but you will probably learn more if you write most of the implementation of your chosen data structure yourself. Your functions can be recursive, as in the example, or not.