CSCI 1120 (Low-Level Computing), Fall 2008

Homework 5

Assigned: November 26, 2008.

Due: December 8, 2008, at 5pm.

Credit: 20 points.

1 Reading

Be sure you have read, or at least skimmed, the readings for 11/10 and 11/17, linked from the "Lecture topics and assignments" page[1].

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 5"). 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. (10 points) Write a C program that performs some of the same functions as the Unix command `grep` — searches in a file for all lines that match a given search string, and prints the ones that do. For example, if you have a file containing the following lines:

   ```
   Hello world!
   There is a German word for worldview that I have forgotten.
   World without end.
   Goodbye!
   ```

   and you search for `world`, the first and second lines should print. (It's easiest to do the match in a case-sensitive way, so the third line does not match.)

   The program should accept two command-line arguments, the first containing the text to search for and the second containing the name of the file to search. (For extra credit, make the program work more like `grep`, which allows searching any number of files, or none, in which case it searches for the specified text in standard input. If you do this, and more than one file is specified, print matching lines in a way that makes it clear which file they came from. `grep` does this by preceding each matching line with the name of the file it came from, e.g.,

   ```
   input.txt: There is a German word for worldview that I have forgotten.
   ```

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2008fall/HTML/schedule.html`

but you may think of some format you like better.)

Be as careful as you can about reading text — it is very easy to read more characters than will fit into the character array you're putting them in. (I usually recommend using the library function `fgets` for reading text line by line.) If the input file contains lines that are too long, you can just print warning messages about them.

You may find the library function `strstr` useful.

2. (10 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] — it's the program for 11/17.) 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")
     - 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 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)

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS1120_2008fall/SamplePrograms/`

  - 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.