Computer Science 381
Programming Unix in C
The College of Saint Rose
Fall 2013

Lab 9: Data Structures

Due: 11:59 PM, Thursday, November 7, 2013

This week's lab is a follow on to the previous two. You will write two more C data structures in the object-oriented style.

Unix Utilities

Before we get into this week's C programming tasks, we take a look at some of the most useful Unix utilities. The extensive set of simple but useful utilities, and the ability to string them together with pipes and scripts are the real power of a Unix environment. Experienced Unix users faced with a task often find that they can quickly put together a script, or even a single command that performs the task, avoiding the need to find (or worse yet, write) a single program to perform it. This is especially true when the task involves processing some

Back in the first lab, you looked into the functionality of a list of Unix commands. We will look more closely at some of those, plus a few more, this week and next.

Pipes

Input and output redirection, which you've already been doing this semester, is one of the great sources of the power of the Unix command line. You have seen that you can have a program that is expecting input from the keyboard (standard I/O functions like `getchar` and `scanf` that read from `stdin`) get its input instead from the contents of a file. For example, in the `inputadder` program from earlier this semester, you could have a set of numbers to be added in a file `mynumbers.txt` and without changing your program to know that the input is going to be in a file rather than be typed in at its prompts by issuing the command:

```
./inputadder < mynumbers.txt
```

You have been using output redirection all semester, at least for the "output capture" questions in the labs.

```
ls -laR > ls.out
```

Now if you wanted to take the output of one program and use it as the input to another, you could use a temporary file as a way to store that output from the first program and provide it as input to the second. For example, if you have one program that generates a list of numbers (for whatever purpose), and you want to add those numbers up with your `inputadder` program:

```
./gennumbers > tempfile
./inputadder < tempfile
```

While this would work, it has some problems. We need to pick a name for a file that doesn't already exist. We need space in the filesystem to store the file. We will want to remember to remove the file afterward.

Unix provides the ability to attach the output of one program directly to the input of another using a *pipe*. For the example above, your command line would be:

```
./gennumbers | ./inputadder
```

In addition to avoiding the need for the temporary file, this can be done much more efficiently behind the scenes. The first program can still be running while the second starts its work.

There's no reason to limit this to just two commands in a pipeline. For example, suppose we have an input file `namelist.txt` that contains an unsorted list of names, one per line. We want to consider only those names that contain the word "john" anywhere in the name, and we want to print out the last three alphabetically from that group. This pipelined command line would do it:

```
grep -i john namelist.txt | sort | tail -3
```

Lab Question 1:
Explain what's happening in each component of the above command pipeline and how they combine to work as described. (3 points)

For the following lab questions, describe the effect of the given command pipeline.

Lab Question 2:
`ls -1 | wc -l` (1 point) Note: the parameter to `ls` is the number '1' while the parameter to `wc` is the letter 'l'.

Lab Question 3:
`head -10 myfile | tail -1` (1 point) The parameter to `tail` is the number '1'. We assume that the file `myfile` contains at least 10 lines.

For the following lab questions, give a single Unix command pipeline that would accomplish the task described.

Lab Question 4:
Print the number of files in the current directory. (1 point)

Lab Question 5:
All of the files in a directory modified on Halloween. (2 points) Hint: start with `ls -la`.

? Lab Question 6:

Given a file with a list of several hundred words, one per line, print the word that occurs between lines 100 and 200 of the file which is last alphabetically. (2 points)

Programming Assignment: A Queue of Ratios

Create a queue structure and corresponding functions to operate on queues in C that holds `ratio` values. You may use the `ratio` structure from the `ratios` examples. Again, include an appropriate header file, implementation file and a file containing a `main` function that tests your implementation. Also include a `Makefile` that compiles your queue implementation and your testing code.

Programming Assignment: A Priority Queue of Ratios

Next, create a priority queue structure and corresponding functions to operate on priority queues in C that holds `ratio` values. Your priority queue should remove the smallest ratio when an item is removed. Again, include an appropriate header file, implementation file and a file containing a `main` function that tests your implementation. Also include a `Makefile` that compiles your priority queue implementation and your testing code.

These programs and their `Makefile`s are worth a combined 40 points as broken down below.

Reference solutions to all programs are available on mogul in `/home/cs381/labs/ooc2`.

Submission

Please submit all required files as email attachments to *terescoj@strose.edu* by 11:59 PM, Thursday, November 7, 2013. Be sure to check that you have used the correct file names and that your submission matches all of the submission guidelines listed on the course home page.

Grading

Grading Breakdown	
Lab questions	10 points
`ratio` queue correctness	12 points
`ratio` priortity queue correctness	12 points
Program error checking	3 points
Program memory management	4 points
Program documentation	5 points
Program style	3 points
`Makefile`s	1 point