ECE498SL (409 pending): Engineering Parallel Software **Spring 2012**
Homework #2 **Due: Tuesday 28 February**

Short Code Examples

This assignment is intended to give you an opportunity to observe and correct many of the problems discussed in class. Eight short programs ranging from 70 to about 600 lines serve as the basis for your effort.

The lab associated with this homework must be done individually. Similarly, although this assignment is a homework and therefore allows you to split up the work, I encourage you to solve these problems on your own and then compare your answers with other students.

For each problem, turn in a written explanation of the problem(s) with the program and what you had to do to fix it. In addition to this written explanation, please also turn in your corrected or modified version of the program. Please put all of the programs into a single tarball and mail them to me. If you do end up working together, only one person from your group needs to send the code to me.

Leaky: This program repeatedly creates and destroys instances of a base class, but somehow runs out of memory by doing so. Note that the code takes about 8 seconds to terminate on the `remlnx` machines.

What's going on? Fix one of the class definitions (not `main`!) to remove the leak. Turn in an explanation and a description of the necessary change.

States: This program tries to create an array of "state pointers" and initialize them, then clean up, but something causes a segmentation violation. Once you have fixed that problem, you'll find that the program is leaking memory as well. Turn in an explanation and a description of the necessary changes.

Histogram: This program uses two methods to build a histogram of alphabetic characters from a string and then prints the two. Somehow, the results are different.

Figure out what's going on and explain. Note that one of the two histograms is correct, even though neither one should be. Your answer should include an explanation of why you get the right result in one case.

Palindrome: This program reads words from a file and checks whether or not they are palindromes. Each palindrome found is printed to `cout`.

Use `time(1)` to measure the amount of time required to check the word list file for palindromes, then rewrite the recurse routine to return a bool rather than throwing an exception. You will also have to edit isPalindrome to make use of the return value. Time your new code—it should go roughly 8× faster.

Prime: This program is meant to print all prime numbers less than 1000. The algorithm used is nearly the simplest one possible: check for divisors of up to the square root of the number.

The program has a few bugs, however, which generate exceptions. The exception reporting mechanism is also somehow broken.

Fix these things and get the output to match the desired output file (`prime.cc.output`) exactly.

Ring: This program builds a simple ring graph using a set container, but somehow none of the links show up when they're tested.

Explain why the links are not found and fix the code so that all of the links can be found. There are two (or more) possible fixes, both of which also make the code faster by an order of magnitude or more. Explain this performance gain as well.

Spellck: This program is the by far the largest one in this assignment. The program builds a data structure to hold a set of English words (a dictionary), then reads in a file, breaks it into words, and prints out any words not included in the dictionary. You invoke the program by passing the name of the file to be spell-checked. For example: `spellck spellck.cc.input`

The first task for this code is to fix a dynamic memory error. The problem may be a bit tricky. Try watching what happens to the dictionary (`printWords`) in the first few insertions.

The second task involves extending the array operator overloading. Array notation is used to check for the presence of a word in the dictionary. In this problem, you must extend the design to allow array notation for use in inserting and removing words from the dictionary.

Note that the `insert` and `remove` calls already exist, and that examples with the desired syntax are included in the code (just un-comment them).

Also note that if the temporary object created by your new `operator=` is not marked as constant, you will need to override the default assignment operator for the helper class to get the line

```
dictionary["center"] = dictionary["have"] = dictionary["not"];
```

to work properly. it will **compile** but not work if it copies bits from one helper object to the next without converting to booleans and assigning (inserting/removing) from the trie.

When your code works correctly, the output produced for the sample input should match the sample output file (`spellck.cc.output`) exactly.

World: This program creates a 2-dimensional world of walls and spaces. Two command-line arguments are necessary to specify the dimension of the (square) world and the probability that each location contains a brick wall. After creating a random world based on these parameters, the program checks whether one corner is reachable from the opposite corner. The answer is printed as 1 or 0. The chance that the opposite corner is reachable will pass through a phase transition at some point, an interesting random graph phenomenon that we'll explore further in Lab #2.

Sadly, the search is slightly broken. You may well be able to fix the bug by inspection. Instead, try to debug it using `gdb` by looking at the state of the queue and the set of seen world locations. Then use the functions `printTupleSet` and `printTupleQueue` to watch what's happening. In `gdb`:

```
call printTupleSet (seen)
call printTupleQueue (q)
```

Explain and fix the bug, then comment on your experience trying to debug STL data structures with and without support functions for inspecting them.