CPSC 2120/2121
Lab 03

Stack implemented with a vector
Queue implemented with a linked list
Due Friday September 14 at 11:59pm

Learning Objectives

- To improve our ability to implement algorithms and data structures in C++
- To implement a data structure using a class template in C++
- To use a simple data structure (a vector) as a building block to construct another data structure (a stack)
- To use a simple data structure (a linked list) as a building block to construct another data structure (a queue)
- To become proficient in fundamental data structures used throughout computer science
- To improve our analytical skills while gaining familiarity with mathematical tools used in the analysis of algorithms
- To implement a data structure to understand its intricacies

Problem / Exercise

For this lab, you will implement a stack data structure by using a vector from the C++ STL. You are free to use the vector class and any of its functions to implement your stack for this assignment. Your stack data structure must be implemented in a file called Stack.h. Remember that when adding to and removing from a stack the rules are always last in first out (LIFO).

You will also implement a queue data structure by using a linked list from the C++ STL. You are free to use the list class and any of its functions to implement your queue for this assignment. Your queue data structure must be implemented in a file called Queue.h. Remember that when adding to and removing from a queue the rules are always first in first out (FIFO).

Before you begin coding for this lab, it is recommended that you read Stack.h, Queue.h, lab03.cpp (a testing program), Makefile, and the examples found in this document to understand how the functions should operate. Once you have an idea of what to do, then implement the functions in Stack.h, and Queue.h, and follow the additional requirements below.

Additional Requirements

- Your Stack and Queue class must compile and run with unmodified versions of our provided lab03.cpp and our Makefile to produce the same output as our examples. If your class does not, then you'll need to fix it. After you get your class working with our test program, then you should create several examples on your own to further test your program to make sure it works with any valid set of test cases.
- You CANNOT modify any function prototypes (function declarations); otherwise, your program may not compile with our tester program when we grade it, which may result in a grade of zero on the assignment.
- Your Stack class must use the vector data structure and your Queue must use the linked list data structure. Besides those two data structures You CANNOT use data structures found in the C++ STL for your Stack or Queue class, which includes, stack, queue, set, array, map, etc. The purpose of this lab is to implement a Stack and Queue class, not to use the already implemented Stack and Queue classes in the STL. The use of any data structures in C++ STL besides the ones given in the instructions will result in a grade of 0 on this lab. However, you may use the string class.

Examples

Your C++ source code should be in a file called Stack.h and Queue.h, and it should be compiled (along with our provided lab03.cpp file) into an executable called lab03.out using our provided Makefile. The output of our testing program in lab03.cpp with your class implementation must look exactly like the examples below when run on the command line on our Unix machines.

```
./lab03.out
s1: size = 11, stack = (top) 29 28 25 22 19 16 13 10 7 4 1 (bottom)
s2: size = 18, stack = (top) z y v u # s # o n m j i # g # c b a (bottom)
s3: size = 6, stack = (top) 3 2.5 2 1.5 1 0.5 (bottom)
q1: size = 11, queue = (front) 19 20 21 22 23 24 25 26 27 28 29 (back)
q2: size = 18, queue = (front) I j k L m n O p q R s t U v w X y z (back)
q3: size = 6, queue = (front) 999995 999996 999997 999998 999999 1e+06 (back)
```

Source Code Requirements

- Put a comment at the top of your source code file(s) with your name (first and last), the date of your submission, your lab section, and the assignment's name.
- All functions should be commented. Use inline documentation, as needed, to explain ambiguous, tricky parts, or important portions of your code.
- All source code must follow good programming style standards such as properly indenting source code and all variables, functions, and classes should be well-named.
- Your class must use dynamic memory allocation and deallocation properly, which means your class cannot contain a memory leak. You may use valgrind to check for memory leaks. Refer to the Unix manual and valgrind's online documentation for more information on valgrind.

Submission

Before the date/time stated on the CPSC 2121 Handin webpage, you need to submit your code to our CPSC 2121 Handin webpage under the correct lab assignment. Make sure to submit all of the following.

1. All source files required for this lab (Stack.h, Queue.h)
2. The tests that you created (lab03.cpp)

After you submit, always double check that the file(s) you submitted were the correct version. To double check, download the submitted file(s), put them on one of our Unix machines, and make sure they compile and run correctly.

Grading: 10 points

If your class does not compile on our Unix machines (or your assignment was not submitted on time), then you'll receive a grade of 0 on this assignment. Otherwise, your class will be graded using the criteria below.

Your class works correctly with a program we will use to test your class	6 points
Proper variable names, documentation, and code organization	2 points
Your class uses dynamic memory allocation/deallocation properly (no memory leaks)	1 point
At least 3 distinct, passing test cases (not included ones provided) are submitted	1 point
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly and efficiently on our Unix machines with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working class. We will only test your program with valid sets of inputs.