**COMSC-210 Lab 8  
Stacks and Queues**

In this assignment you will develop your own implementations of stacks and queues. You will have to decide upon design considerations and any additional functionality for yourself.

After you complete this lab assignment, post the required files to the [COMSC server](http://cs.dvc.edu/upload-files.jsp) so that you receive the proper credit for this 50-point lab. Your posted files are due at midnight of the evening of the due date indicated in the course outline. Use the "Submit your SP2011 work" link on the class website to post your file for this lab, using the **lab8** folder provided.

**EXERCISE 1: Write And Apply A Stack Template (10 points)**   
**Purpose**. In this exercise you will write your own stack template, using either array or linked-list implementation, per your choice.

**Requirements.** Rewrite lab 2's Calculator2.cpp as**Calculator8.cpp**, but use your own templated stack class. Name your class Stack, and write its specification in a file named**Stack.h**, and its implementation in a file named**Stack.cpp**. Use either a dynamic array or a linked list to implement the class, and be sure to include dynamic memory management. If you use a linked list, use Node as the node struct name, and include it in Stack.h, making sure to enclose it in its own #ifndef NODE / #endif container -- that will prevent compiler problems when you write a queue template later.

In addition to the calculator application, and prior to developing it, write a driver program,**StackDriver.cpp**, to *fully* test your templated stack class. Refer to useArray.cpp on page 118 and 119 of the Childs textbook to see the things they did to test a templated class with dynamic memory management. Make sure that you test *every* function that you write.

*You* decide on the "design consideration" issues that are listed in the lecture topic 8 notes. Decide on the pop, push, and peek options -- you do not have to choose the same as is indicated in the notes. If you use a linked list, decide if you want a header node or not. Decide any extra functionality that you may wish to include, such as size tracking or status tracking. But do not use the Arraytemplate that you developed, and do not use any STL containers in your Stack, and do *not* include any cout statements in the template.

Submit the *four* files to the class website for credit.

**Program I/O.** Same as for Calculator2.cpp.

**Scoring.** Full points will be awarded upon submission of a fully correct and working version of the required files. Incorrect submissions will receive zero points, and you will be asked to resubmit for full points. No submissions of remaining exercises in this lab, or of future labs, will be considered for grading until full points are awarded for this exercise.

**EXERCISE 2: Use A Stack (20 points)**   
**Purpose**. In this exercise you use a stack to solve the "matching brackets" problem. Your program will read an input text file, and report the first-found container mismatches, such as unbalanced parentheses.

**Requirements.** Write**PreProcessor.cpp**, that prompts the user for an input filename, and checks it for matching container brackets. The input file should be expected to be a text file, of type H, CPP, JAVA, HTML, or JS. Check for these bracket combinations:

* parentheses ( and )
* curly braces { and }
* square brackets [ and ]
* comment offsets /\* and \*/ -- in a CPP, H, JAVA, or JS file

Just report the first found occurrence of a mismatch.

Do NOT check for angle brackets, because there can be less-than and greater-than symbols that make that hard to manage. And don't worry about the appearance of these symbols inside statis strings, like "]". They will appear as mismatches in your program, and that's okay -- we're not writing a compiler... yet.

Use a stack data structure as the basis of your solution, but *you* decide whether to use your own Stack template that you developed for exercise 1, or the STL stack.

Fully test, and then submit the CPP file to the class website for credit.

Note that when using a string for a filename, the "open" statement in ifstream is supposed to use a C-string as a parameter, and not a C++ string. Your compiler may overlook this non-standard behavior, but your instructor will not. [[ Read more... ]](http://www.cplusplus.com/reference/iostream/ifstream/open/)

**Program I/O.** Input: using cin, a filename. Include the expected filename extensions in the prompt (H, CPP, JAVA, HTML, or JS) for reference only -- that is, don't validate. Output: If there are no mismatches, say so in your own words. Otherwise, print the line number of the line containing the opening bracket, say what that bracket is, and say that you cannot find the closing backet and what that bracket should be. Or, print the line number of the line containing the closing bracket for which there is no opening bracket, say what that bracket is, and say that you cannot find the opening backet and what that bracket should be.

**Scoring.** Points will be awarded upon submission of a fully correct and working version of the required file. Incorrect submissions will receive zero points, and you will be asked to resubmit, but a 1 point per day penalty will apply. Submissions that are 20 or more days late will not be graded, and zero points awarded, but grading of future labs will NOT be affected.

**Example (computer prompts in bold).**

*Example: file with no mismatches*

**Enter filename (H, CPP, JAVA, HTML, or JS):** x.cpp

**No mismatches found by my preprocessor in x.cpp**

*Example: file with no closing parenthesis*

**Enter filename (H, CPP, JAVA, HTML, or JS):** x.cpp

**Opening parenthesis found in line 10 of x.cpp**

**But no matching closing parenthesis found**

*Example: file with no opening square bracket*

**Enter filename (H, CPP, JAVA, HTML, or JS):** x.cpp

**Closing square bracket found in line 10 of x.cpp**

**But no matching opening square bracket found**

**EXERCISE 3: Write And Test A Queue Template (20 points)**   
**Purpose**. In this exercise you will write your own queue template, using either array or linked-list implementation, per your choice.

**Requirements.** Rewrite your own templated queue class. Name your class Queue, and write its specification in a file named**Queue.h**, and its implementation in a file named**Queue.cpp**. Use either a dynamic array or a linked list to implement the class, and be sure to include dynamic memory management. If you use a linked list, use Node as the node struct name, and include it inQueue.h, making sure to enclose it in its own #ifndef NODE / #endif container, to prevent compiler issues when using your queue with other linked-list classes.

Write a driver program,**QueueDriver.cpp**, to *fully* test your templated stack class.

*You* decide on the "design consideration" issues that are listed in the lecture topic 8 notes. Decide on the pop, push, and peek options -- you do not have to choose the same as is indicated in the notes. If you use a linked list, decide if you want a header node or not. Decide any extra functionality that you may wish to include, such as size tracking or status tracking. But do not use the Arraytemplate that you developed, and do not use any STL containers in your Queue, and do *not* include any cout statements in the template.

Submit the three files to the class website for credit.

**Program I/O.** All console I/O, in the driver program only.

**Scoring.** Full points will be awarded upon submission of a fully correct and working version of the required files. Incorrect submissions will receive zero points, and you will be asked to resubmit for full points. No submissions of future labs will be considered for grading until full points are awarded for this exercise.