

## Background:

There are two fundamental data structures that all programmers should know, stacks and queues. For this assignment, you will be programming one of each. You will use a dynamic array as the internal data structure for each. Your array will grow and shrink exactly like the last project. However, since you will now have dynamic data inside of your class, you will need to write a copy constructor, an assignment operator, and a destructor.

## Details:

Your project will accept what are known as command line parameters. We will discuss how this is done in class in the next class or two. The command line parameters will be up to 7 data files and 1 command file. The command file will always be the last file specified. The data files will contain information about products you can buy at Kroger. All you have to do is read them in and store them in your stack. I will write two functions for you, one called `printStack` and one called `printQueue`.

The interface for `printStack` is as follows:

```
Queue printStack( Stack, ostream&, Queue );
```

The return is meant for you to capture the return statement in your queue so that it can subsequently be printed out. This is how the data will be entered into the queue.

The interface for `printQueue` is as follows:

```
void printQueue( Queue, ostream& );
```

The functions will simply take the data structures and print them out. The stack will print in reverse order and the Queue will print in forward order.

The Stack class **must** implement the following methods:

```
bool push( Record );  
bool pop ( Record& );
```

The Queue class **must** implement the following methods:

```
bool enqueue( Record );  
bool deque( Record& );
```

(Where `record` is the name of the class that stores the data record)

In all cases true is returned if the operation was successful and false is returned if the operation fails.

Both the queue class and the stack class **must** implement a copy constructor, an assignment operator and a destructor or this project will not work correctly.

The Record class **must** implement the following methods:

```
void print( ostream& );
```

You **must** use the following class names: Queue, Stack and Record. Any deviation from these names will not work.

I will write two files for you: Function.h and Function.cpp.

Function.h will include the function prototypes. At the top of your main program you simply need to type `#include "Function.h"` and you will be able to use the functions in your program. Additionally, both Function.h and Function.cpp will need to be in the same directory as your project and be included in your project. To do this, create the project as you normally would. Once, you have created your project and all the files you need to create, download and save the two files mentioned above and place them in the same directory will all your other cpp and h files. Now from the file menu choose add existing item and select Function.cpp and Function.h. You should now be able to use without trouble both of those files.

The two functions described above may also do some things to double check that you have indeed successfully created a copy constructor, assignment operator and destructor in addition to simply printing out the contents of the stack and queue.

For the record, below is a sample of the data from one of the files.

UPC	Prd Name	UnitPrc	Units	Krgr	Prc	Krgr	Crđ	Id Num
012-34230-6574	Chocolate Milk	\$1.54	13	\$1.04	Yes			X3-42F9
123-24524-6543	Shampoo	\$5.32	2	\$2.34	No			
532-43576-7862	Dog Bones	\$3.67	4	\$3.01	Yes			52-AG52-RR

The first line is a header line indicating the contents of each column. The subsequent rows are the items. All you need to do is read in the contents of the files, create a record for each one, and store the record in the stack. Each token will be tab delimited. The sample above is not for ease of readability.

The command file will indicate which file to load and which print function to call. For example:

```
print stack
load file1
print stack
load file4
print queue
print stack
load file2
print queue
```

load file3
print stack
print queue

When you load a file you simply push the contents onto the stack each time. You do not clear the stack between loads. This means that by the end of the program's execution there might be a considerable amount of data stored in the stack.

Conversely, each called to printStack with its subsequent assignment into your queue, should be an assignment. That means that the elements in the queue will be lost with each call to printStack and replaced with a new set. At the end of the program run, the stack will contain all the loaded elements in one order and the queue will hold the elements in reverse order.

The records are of no consequence and could be any information; I simply choose some data to represent. The key issue is that the record class implements the print function as specified.

Oh yeah, the output file is to be called "StackQueuePrinting.txt". You should format your output in such a way that it presents a nice neat table. I will take care of any headers for each print.

Finally, there are only three commands:

load file\* - This will cause your program to load the numbered file and push each record onto the stack. Where \* is any integer number. The only valid numbers it can take on is 1 – 7. I may give numbers bigger than 7. See error checking below.  
print stack – This will cause your program to call the given printStack function.  
print queue – This will cause your program to call the given printQueue function.

As for error checking, if a file is given and there is not a valid data file to go along with it, print an error indicating that the file does not exist. For example, the program is invoked with 4 data files and the command is given: load file\*, where \* > 4 then an error is written an execution continues. If the data file has no contents, then no elements get added to the stack.

## Programming Considerations:

Like always you need to follow all the software engineering guidelines covered in this class and others. You should choose valid identifier names and function names. You need to have header comments for each function, method, class, or major block of code.

Specifically you must:

1. Write three classes called: Stack, Queue, and Record
2. Both Stack and Queue must implement a copy constructor, assignment operator, and destructor.
3. The initial size of the both the stack and queue is 10
4. They both double in size when full and half their size when they are more than half empty to a minimum size of 10. For example, the current size is 40, and you have 20 elements, you do not resize. You remove one more element, now you resize.

5. When doubling or resizing, write a message to the file indicating the old size and the new size. This message should have a blank line before it and a blank line after it to make it easily identifiable.
6. Record must implement a print function.
7. You must use the interfaces from above.
8. Print record should produce a nice neat table.
9. Use the output file "StackQueuePrinting.txt"
10. You may not use the STL stack or queue.
11. You must turn in a design for the Stack and Queue class with your project. This does not need to be turned in earlier, just with the final submission.

## Submitting your project:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded by the TAs. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and talk to the TAs. The last file submitted will be considered for grading.

The *Student Guide* and submission link can be found at:

<http://www.cs.vt.edu/curator/>

## Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:  
//  
// - I have not discussed the C++ language code in my program with  
// anyone other than my instructor or the teaching assistants  
// assigned to this course.  
//  
// - I have not used C++ language code obtained from another student,  
// or any other unauthorized source, either modified or unmodified.  
//  
// - If any C++ language code or documentation used in my program  
// was obtained from another source, such as a text book or course  
// notes, that has been clearly noted with a proper citation in  
// the comments of my program.  
//  
// - I have not designed this program in such a way as to defeat or  
// interfere with the normal operation of the Curator System.  
//  
// <Student Name>
```

**Failure to include this pledge in a submission is a violation of the Honor Code.**