CS 311 - HW 2 - 100 points

1. Queue Implementation (100 points)

Write a **queue** application program to generate all possible strings consisting of three characters A, B, and C using a queue.

Steps:

1. Implement the queue: complete the attached queue.h and queue.cpp
2. Test your queue using the attached queuetest.cpp (Test1).
3. Run the attached hw2queuedemo.out to see how client2.cpp should work.
4. Implement the queue application program to generate all possible strings consisting of three characters A, B, and C using a queue: complete the attached client2.cpp
5. Test your application program (Test2). You can double check your results with the attached hw2queuedemo.out. The results must be the same. You may implement your client2.cpp so that your user interface looks nicer, but the results (displayed numbers or errors) must be the same.
6. Copy-paste the tests results to your test1.txt and test2.txt files. These .txt files must also include the commands required to compile your cpp files. We will compile and run your cpp files and also check your .txt files as references.
7. Make a folder containing your files, compress it and submit your zip file to cougar courses. The name of this file must be your first and last name, for example HW2NahidMajd.zip

Test1:

You run this test to ensure your implemented queue works correctly. Once you completed the attached queue.h and queue.cpp, compile them with queuetest.cpp and run it. For Test1, set MAX_SIZE to 10.

Then run the following test cases and enter the results to your test1.txt:

1. display the queue
2. enqueue “a”, “b”, “c”, “d”, “e”, “f”, “g”, “h”, “i”, “j”
3. display the front element
4. dequeue
5. display the front element
6. display the queue
7. enqueue “k”
8. get size

9. display the queue
10. go to back
11. display the queue

Important note: Study queuetest.cpp. For HW1 and HW2, I gave you the test file. For next HWs, you will be asked to implement the test file.

String generator program

Use your implemented queue to generate all possible strings consisting of three characters A, B, and C in the following order:

A

B

C

AA

AB

AC

BA

BB

BC

CA

CB

CC

AAA

AAB

AAC

ABA

ABB

ABC

ACA

ACB

ACC

…

Set MAX_SIZE to 50, so you will get many strings. The program should keep going until your queue overflows. At that point, print out all the remaining elements of the queue using displayAll(). Your program will display all the strings and at the end, an error message describing the error occurred.

Submission

Submit a zip file containing the following files.

1. queue.h (30 points)	-- class header file
2. queue.cpp (30 points)	-- class implementation file
3. client2.cpp (30 points)	-- your implemented application
4. test1.txt (5 points)	-- results of testing the queue class
5. test2.txt (5 points)	-- results of testing the application

Important note1: You will miss up to 10 points if you don't comment your programs.

Important Note2: Always make sure the files you submit can be compiled on **empress.csusm.edu** with no error. We will compile and test your files on empress.

---

2. Vector-based stack Implementation (20 points extra credit)

Re-write the HW1 stack class (copy and then rename the files as vstack.h, vstack.cpp, and vclient1.cpp first)

1. In this new implementation, use a vector instead of an array.
2. There is no maximum size now. The stack starts out having no slots and it will grow as elements are added.
3. No need for top. Take it out. You can call size()-1 to get top.
4. Destructor has to do some work to make sure it leaves no cells behind.
5. Constructor has no work to do.
6. isFull() always returns false.
7. Note that pop_back() removes the top element but does not return any value.
8. Note that back() returns the top element but does not remove it from vector.

Now, use this vstack class with the HW1 vclient1.cpp program to evaluate post-fix expressions. Test with the same cases as before. The file vlient1.cpp does not change except to include the new header file. <u>Do not change the prototypes of member functions. Your new implementation should be invisible to the client.</u>

Submission

Include the following files in your zip file.

1. vstack.h (20 points)	-- class header file
2. vstack.cpp (-5 if not attached)	-- class implementation file
3. vclient1.cpp (-5 if not attached)	-- your implemented application