ECS 40 **Program #9 (30 pts, 1.25 hour)** **Spring 2017**

Due: Friday, June 9th, 11:59 PM using handin to p9 of cs40.
New concepts: stacks, queues, binary search trees, exceptions, and assert
Filenames: authors.csv main.cpp, stack.cpp, stack.h, queue.cpp, queue.h, BST.cpp, BST.h., and Makefile

You are to create three templates container classes that implement a Stack class implemented using the STL list class, a Queue class implemented using a dynamically allocated array, and a BST implemented using a binary search tree. All three classes will use push() to insert, and pop() to remove. The size of the queue array will be passed to its constructor. Your main() will be quite long because it is the only function in main.cpp. Since it should contain a switch statement based on the container chosen, the style checker will not object to its length.

Specifications

1. Operation Files
   - 1.1. The first line of the file contains an integer that indicates the size with which to construct the Queue array.
   - 1.2. Each succeeding line of the file starts with a two chars.
     - 1.2.1. The first char indicates the container to use: 'B' = BST, 'Q' = queue, and 'S' = stack.
     - 1.2.2. The second char indicates the operation: 'P' = pop(), 'U' = push(). push() lines have the inserted integer appended to the line.
2. main()
   - 2.1. Your program should open and read the file specified by the first command line argument of the program.
   - 2.2. main() will produce exactly one line of output to the screen for each line in the operation file. Except for the first line, each line of output begins with the <line number><space><first char><second char>.
   - 2.3. After the switch statement, main() should print the integer involved. Just in case a pop() fails, the integer should be initialized to -1.
   - 2.4. To allow for flexibility of T, both push() and pop() should take a reference to a T.
3. BST
   - 3.1. The BST class has only one data member, a BSTNode<T> *root.
   - 3.2. The BST will be composed of BSTNodes<T>. Like the ListNode class, the BSTNode class should have the BST class as a friend. The standard constructor for the BSTNode class should be defined in BST.h, and implemented in BST.cpp.
   - 3.3. Since a binary search tree is recursively defined, you should write two private recursive functions that take both a some form of BSTNode <T>* and T object to work through the tree. If the root is not NULL, the recursions are started by passing root to these private functions.
   - 3.4. The pop() of the BST will remove the smallest item in the tree. Note that this guarantees that the node deleted will not have a left child. If the deleted node has no children, then its parent's left pointer should be set to NULL. If the deleted node has a right child, then its parent's left pointer should be set to the address of the deleted node's right child. In either case, you must delete the original node in the function. Surpisingly, this pointer manipulation is simple if you pass a "`BSTNode <T> & *`" (that is a reference to a pointer!) to the private pop().
4. Exceptions
   - 4.1. All exceptions should be caught and reported in main(). The program should continue after printing the error message.
   - 4.2. You must specify the exact exception you are catching, and may not use the elipse (…) in your catch.
   - 4.3. For any of the classes, if a container is empty, then a pop() will cause an under flow exception with a string reference as the exception object type.
   - 4.4. If the Queue constructor size is invalid, then the Queue constructor will throw a bad_alloc as the exception object type. You will need to #include <stdexcept> for this.
   - 4.5. If the program tries to push() when the queue is full, then the queue will throw an exception of a class named Whoops. The Whoops class has a constructor that takes a const string reference as its only parameter, and a what() method that returns the const string reference that is passed the constructor. The Whoops class should be declared in queue.h, and implemented in queue.cpp.
   - 4.6. There should be only one statement in each catch block. It will print either the string passed as the exception, or the string returned from a call to what().
5. assert is found in <cassert>

5.1. You must add assertions that would catch the three explicit exceptions of 4.3, 4.4, and 4.5 before they reach the throw statements. Because these are so simple, I wrote and tested them before writing the exception code.

6. Makefile
   6.1. Your Makefile should have an "all:" rule that is dependent on p9.out, and p9b.out, and has simply a tab as the second line of the rule.
   6.2. p9.out is compiled with the NDEBUG defined using the D option of g++.
      6.2.1. It is dependent on every file, and is created directly from main.cpp
      6.2.2. In Netbeans, you would right click on the project, and then select Properties->C++Compiler->Preprocessor Definitions->…->Add. Then just enter NDEBUG.
   6.3. p9b.out is compiled without NDEBUG defined so that the asserts will work.
      6.3.1. It is dependent on every file, is created directly from main.cpp files.
   6.4. You need not compile a main.o file for this assignment, so only two lines are needed for each executable.
7. Remember to use const wherever appropriate.
8. Suggestion: Since the push() and pop() methods have many things in common in all three classes, write the Stack class completely first. When it works perfectly, copy and paste it into the other classes, and do global search and replace to convert to the other classes.

Running the program. Note that p9.out and p9b.out will have the same output for non-error files.

```
[ssdavis@lect1 p9]$ p9.out stackPush.txt
1 Queue size 5
2 SU  8
3 SU  10
4 SU  3
5 SU  5
 [ssdavis@lect1 p9]$ p9.out stackPop.txt
1 Queue size 9
2 SU  23
3 SU  43
4 SU  19
5 SP  19
6 SP  43
7 SP  23
[ssdavis@lect1 p9]$ p9.out stackUnderflow.txt
1 Queue size 9
2 SU  23
3 SU  43
4 SU  19
5 SP  19
6 SP  43
7 SP  23
8 SP Stack underflow error. -1
9 SU  8
[ssdavis@lect1 p9]$ p9b.out stackUnderflow.txt
1 Queue size 9
2 SU  23
3 SU  43
4 SU  19
5 SP  19
6 SP  43
7 SP  23
p9b.out: stack.cpp:20: void Stack<T>::pop(T&) [with
T = int]: Assertion `! stackList.empty()' failed.
8 SP Abort (core dumped)
[ssdavis@lect1 p9]$ p9.out queuePush.txt
1 Queue size 4
2 QU  17
3 QU  29
4 QU  67
5 QU  3
[ssdavis@lect1 p9]$ p9b.out queuePop.txt
1 Queue size 17
2 QU  23
3 QU  88
4 QU  4
5 QP  23
6 QP  88
7 QP  4
[ssdavis@lect1 p9]$
```

```
[ssdavis@lect1 p9]$ p9.out queueSizeError.txt
1 Queue size std::bad_alloc -5
2 SU  3
[ssdavis@lect1 p9]$ p9b.out queueSizeError.txt
p9b.out: queue.cpp:21: Queue<T>::Queue(int) [with T
= int]: Assertion `size >= 0' failed.
1 Queue size Abort (core dumped)
[ssdavis@lect1 p9]$ p9.out queueOverflow.txt
1 Queue size 3
2 QU  9
3 QU  12
4 QP  9
5 QU  13
6 QU  19
7 QP  12
8 QU  6
9 QU Queue overflow error. 15
10 QP  13
11 QP  19
[ssdavis@lect1 p9]$ p9b.out queueOverflow.txt
1 Queue size 3
2 QU  9
3 QU  12
4 QP  9
5 QU  13
6 QU  19
7 QP  12
8 QU  6
p9b.out: queue.cpp:48: void Queue<T>::push(const T&)
[with T = int]: Assertion `count < size' failed.
9 QU Abort (core dumped)
[ssdavis@lect1 p9]$ p9.out BSTPop.txt
1 Queue size 13
2 BU  12
3 BU  8
4 BP  8
5 BU  23
6 BP  12
7 BU  3
8 BU  19
9 BP  3
[ssdavis@lect1 p9]$ p9.out BSTUnderflow.txt
1 Queue size 29
2 BU  8
3 BU  12
4 BP  8
5 BP  12
6 BU  9
7 BP  9
8 BP BST underflow error. -1
9 BU  7
[ssdavis@lect1 p9]$
```