**CSCI 241 Assignment 6  
100 points**

**Purpose**

This assignment is an exercise in implementing the queue ADT using an "circular" array-based data structure as discussed in lecture. This assignment also introduces the concepts of templates and exception handling.

**Assignment**

This program creates and implements a class to represent the Queue ADT using an array.

A *driver program* is provided for this assignment to test your implementation. You don't have to write the tests.

**Program**

You will need to write a single template class for this assignment, the Queue class. You will need to implement several methods and functions associated with this class.

**Since this is a C++ template, all of your code should be placed in a single header (.h) file. This includes the implementations of all methods and any other associated functions.**

**class Queue**

*Data members*

This class contains a pointer to an item of the template parameter type that will point to the first element of a dynamically allocated array (the *queue array*). Because the array is allocated dynamically, a data member is also maintained inside the class to determine the maximum number of elements that may be stored in the array (the *queue capacity*). Another data member is used to keep track of the number of data items currently stored in the vector (the *queue size*). Both of these data members should be declared as data type size\_t (which corresponds to an unsigned integer).

The class also requires two integer data members: the subscript of the front item in the queue (the *queue front subscript*) and the subscript of the back item in the queue (the *queue back subscript*).

*Methods and associated functions*

As usual, methods that do not alter the data members of the object that called the method should be declared to be const.

* Constructor

The class should have a default constructor that takes no arguments. The constructor should set the queue size and queue capacity to 0 and the queue array pointer to nullptr. The queue front subscript should be initialized to 0. The queue back subscript should be initialized to the queue capacity - 1. Alternately, the data members can be initialized in the class declaration, in which case this method's body can be empty.

* Destructor

The class should have a destructor that deletes the dynamic memory for the queue array. The destructor should **not** call the clear() method.

* Copy Constructor

The class should also have a proper copy constructor. This will be similar to the copy constructor you wrote for the Stack class for Assignment 5. The main difference is that you will need to copy the contents of the entire queue array (elements 0 to queue capacity - 1), not just elements 0 to queue size - 1.

* Copy Assignment Operator

The copy assignment operator should be properly overloaded to allow one Queue to be assigned to another. Once again, the code here will be similar to what you wrote for Assignment 5. As with the copy constructor, you will need to copy the contents of the entire queue array.

* operator<<

The output operator should be overloaded so that a Queue can be printed on the standard output. As in Assignments 4 and 5, this will need to be a standalone friend function rather than a method.

Declaring a template function to be a friend of a template class requires some special syntax - see the **Implementation Hints** below.

Looping through the queue array to print the elements is complicated by the fact that the queue front subscript will not necessarily be less than the queue back subscript. One way to make this work is to have a counter that controls how many times the loop is done, and a subscript that starts at the front of the queue and is incremented until it reaches the back of the queue:

size\_t current, i;

for (current = rhs.qFront, i = 0; i < rhs.qSize; ++i)

{

// Print queue element at subscript i

lhs << rhs.qArray[current] << ' ';

// Increment i, wrapping around to front of queue array if necessary

current = (current + 1) % rhs.qCapacity;

}

* clear()

This method takes no arguments and returns nothing. It should properly set the queue back to the empty state (set the queue size to 0, the queue front subscript to 0, and the queue back subscript to the queue capacity - 1).

* size()

This method takes no arguments and returns a size\_t. It should return the queue size.

* capacity()

This method takes no arguments and returns a size\_t. It should return the queue capacity.

* empty()

This method takes no arguments and returns a boolean value. It should return true if the queue size is 0; otherwise it should return false.

* front()

This method takes no arguments and returns a reference to a constant item of the template parameter type. If the queue is empty, this method should throw an underflow\_error exception. Otherwise, it should return the front element of the queue array (the one at the queue front subscript).

* back()

This method takes no arguments and returns a reference to a constant item of the template parameter type. If the queue is empty, this method should throw an underflow\_error exception. Otherwise, it should return the back element of the queue array (the one at the queue back subscript).

* push()

This method takes a reference to a constant item of the template parameter type as its argument, the value to insert into the queue. If the queue is full (the queue size is equal to the queue capacity), this method will need to call the reserve() method to increase the capacity of the queue array and make room for the item to insert. If the queue capacity is currently 0, pass a new capacity of 1 to the reserve() method. Otherwise, pass a new capacity of twice the current queue capacity to the reserve() method.

To insert an item, the method should increment the queue back subscript (wrapping around to the front of the queue array if needed) and then copy the method argument into the queue array as the new back item in the queue. The queue size should be incremented by 1.

* pop()

This method takes no arguments and returns nothing. If the queue is empty, this method should throw an underflow\_error exception. Otherwise, it should increment the queue front subscript (wrapping around to the front of the queue array if needed) to effectively remove the front item in the queue array. The queue size should be decremented by 1.

* reserve()

This method takes a size\_t, the new capacity to allocate for the queue array. It returns nothing.

If the new capacity is less than the queue size or is equal to the current queue capacity, simply return. Otherwise, the method should reserve additional capacity for the queue array equal to the new capacity passed in. The logic to do this is similar to that used in Assignment 5.

Note that the "circular" nature of the queue array does complicate copying the items from the queue array to the new temporary array. You will need to modify your code from Assignment 5 accordingly.

**Output**

A driver program, assign6.cpp is provided for this assignment. The purpose of a driver program is to test other pieces that you code. You do not need to write the driver program yourself. A copy of the driver program can also be found on turing at /home/turing/t90kjm1/CS241/Code/Fall2017/Assign6/assign6.cpp.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROGRAM: CSCI 241 Assignment 6

PROGRAMMER: your name

LOGON ID: your z-ID

DUE DATE: due date of assignment

FUNCTION: This program tests the functionality of the Queue

template class.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stdexcept>

#include "Queue.h"

using std::cout;

using std::endl;

using std::underflow\_error;

int main()

{

cout << "Testing default constructor\n\n";

Queue<int> q1;

cout << "q1: " << q1 << endl;

cout << "q1 size: " << q1.size() << endl;

cout << "q1 capacity: " << q1.capacity() << endl;

cout << "q1 is " << ((q1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing push()\n\n";

for (int i = 5; i < 40; i += 5)

q1.push(i);

cout << "q1: " << q1 << endl;

cout << "q1 size: " << q1.size() << endl;

cout << "q1 capacity: " << q1.capacity() << endl;

cout << "q1 is " << ((q1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing pop()\n\n";

for (int i = 0; i < 3; ++i)

q1.pop();

cout << "q1: " << q1 << endl;

cout << "q1 size: " << q1.size() << endl;

cout << "q1 capacity: " << q1.capacity() << endl;

cout << "q1 is " << ((q1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing wrap-around on push()\n\n";

for (int i = 3; i <= 12; i += 3)

q1.push(i);

cout << "q1: " << q1 << endl;

cout << "q1 size: " << q1.size() << endl;

cout << "q1 capacity: " << q1.capacity() << endl;

cout << "q1 is " << ((q1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing wrap-around on pop()\n\n";

for (int i = 0; i < 6; ++i)

q1.pop();

cout << "q1: " << q1 << endl;

cout << "q1 size: " << q1.size() << endl;

cout << "q1 capacity: " << q1.capacity() << endl;

cout << "q1 is " << ((q1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing queue resize on push()\n\n";

for (int i = 10; i < 140; i += 10)

q1.push(i);

cout << "q1: " << q1 << endl;

cout << "q1 size: " << q1.size() << endl;

cout << "q1 capacity: " << q1.capacity() << endl;

cout << "q1 is " << ((q1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing copy constructor()\n\n";

Queue<int> q2 = q1;

cout << "q1: " << q1 << endl;

cout << "q1 size: " << q1.size() << endl;

cout << "q1 capacity: " << q1.capacity() << endl;

cout << "q1 is " << ((q1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "q2: " << q2 << endl;

cout << "q2 size: " << q2.size() << endl;

cout << "q2 capacity: " << q2.capacity() << endl;

cout << "q2 is " << ((q2.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing front() and back()\n\n";

cout << "Front item of q1: " << q1.front() << endl;

cout << "Front item of q2: " << q2.front() << endl << endl;

cout << "Back item of q1: " << q1.back() << endl;

cout << "Back item of q2: " << q2.back() << endl << endl;

cout << "q1: " << q1 << endl;

cout << "q1 size: " << q1.size() << endl;

cout << "q1 capacity: " << q1.capacity() << endl;

cout << "q1 is " << ((q1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "q2: " << q2 << endl;

cout << "q2 size: " << q2.size() << endl;

cout << "q2 capacity: " << q2.capacity() << endl;

cout << "q2 is " << ((q2.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing pop() to empty\n\n";

while (!q1.empty())

{

cout << q1.front() << ' ';

q1.pop();

}

cout << endl;

cout << "q1 size: " << q1.size() << endl;

cout << "q1 capacity: " << q1.capacity() << endl;

cout << "q1 is " << ((q1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing assignment operator\n\n";

Queue<int> q3;

q3 = q2;

cout << "q2 (size " << q2.size() << "): " << q2 << endl;

cout << "q3 (size " << q3.size() << "): " << q3 << endl << endl;

cout << "Testing clear()\n\n";

q2.clear();

cout << "q2 (size " << q2.size() << "): " << q2 << endl;

cout << "q3 (size " << q3.size() << "): " << q3 << endl << endl;

cout << "Testing assignment to self and swap\n\n";

q3 = q3;

q2 = q3;

q3.clear();

cout << "q2 (size " << q2.size() << "): " << q2 << endl;

cout << "q3 (size " << q3.size() << "): " << q3 << endl << endl;

cout << "Testing chained assignment\n\n";

Queue<int> q4;

q4 = q3 = q2;

cout << "q2 (size " << q2.size() << "): " << q2 << endl;

cout << "q3 (size " << q3.size() << "): " << q3 << endl;

cout << "q4 (size " << q4.size() << "): " << q4 << endl << endl;

cout << "Testing other data type\n\n";

Queue<char> q5;

for (char c = 'a'; c < 'k'; c++)

q5.push(c);

cout << "q5 (size " << q5.size() << "): " << q5 << endl << endl;

cout << "Testing const correctness\n\n";

const Queue<char>& r5 = q5;

cout << "q5 (size " << r5.size() << "): " << r5 << endl;

cout << "q5 is " << ((r5.empty()) ? "empty\n" : "not empty\n");

cout << "Front item of q5: " << r5.front() << endl;

cout << "Back item of q5: " << r5.back() << endl << endl;

cout << "Testing exception handling\n\n";

cout << "Testing front() with empty queue\n\n";

try

{

cout << q1.front() << endl;

}

catch (underflow\_error e)

{

cout << "Caught "<< e.what() << endl << endl;

}

cout << "Testing back() with empty queue\n\n";

try

{

cout << q1.back() << endl;

}

catch (underflow\_error e)

{

cout << "Caught "<< e.what() << endl << endl;

}

cout << "Testing pop() with empty queue\n\n";

try

{

q1.pop();

}

catch (underflow\_error e)

{

cout << "Caught "<< e.what() << endl;

}

return 0;

}

**Implementation Hints**

* Implement this similarly to the last assignment. Start off at the beginning of the driver program and try to get it working piece by piece. Maintain a working program as you go.
* Declaring a template function to be a friend of a template class is one of the classic "gotcha's" in C++. We are trying to declare a function to be a friend of **all** classes that might be instantiated from the Queue template class, and most C++ compilers will quite properly refuse to do that without some special syntax.

The friend declaration must contain an extra set of <> to indicate that it is a template function (however, do not code this in the actual function definition - only the friend declaration). You'll also usually need to *forward declare* both the template class and the template function, as shown below.

How much of this you actually need to do can vary from compiler to compiler, but the code shown below works in Dev-C++ and with g++ on turing/hopper.

#ifndef QUEUE\_H

#define QUEUE\_H

#include <iostream>

#include <stdexcept>

// Forward declaration of the Queue template class

template <class T>

class Queue;

// Forward declaration of the operator<< template function

template <class T>

std::ostream& operator<<(std::ostream&, const Queue<T>&);

template <class T>

class Queue

{

// friend declaration for the template function - note the

// special syntax

friend std::ostream& operator<< <>(std::ostream&, const Queue<T>&);

...

};

// Method definitions for the Queue class

#endif

**Output**

Output from the correctly functioning driver program should look like the following:

Testing default constructor

q1:

q1 size: 0

q1 capacity: 0

q1 is empty

Testing push()

q1: 5 10 15 20 25 30 35

q1 size: 7

q1 capacity: 8

q1 is not empty

Testing pop()

q1: 20 25 30 35

q1 size: 4

q1 capacity: 8

q1 is not empty

Testing wrap-around on push()

q1: 20 25 30 35 3 6 9 12

q1 size: 8

q1 capacity: 8

q1 is not empty

Testing wrap-around on pop()

q1: 9 12

q1 size: 2

q1 capacity: 8

q1 is not empty

Testing queue resize on push()

q1: 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

q1 size: 15

q1 capacity: 16

q1 is not empty

Testing copy constructor()

q1: 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

q1 size: 15

q1 capacity: 16

q1 is not empty

q2: 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

q2 size: 15

q2 capacity: 16

q2 is not empty

Testing front() and back()

Front item of q1: 9

Front item of q2: 9

Back item of q1: 130

Back item of q2: 130

q1: 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

q1 size: 15

q1 capacity: 16

q1 is not empty

q2: 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

q2 size: 15

q2 capacity: 16

q2 is not empty

Testing pop() to empty

9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

q1 size: 0

q1 capacity: 16

q1 is empty

Testing assignment operator

q2 (size 15): 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

q3 (size 15): 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

Testing clear()

q2 (size 0):

q3 (size 15): 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

Testing assignment to self and swap

q2 (size 15): 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

q3 (size 0):

Testing chained assignment

q2 (size 15): 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

q3 (size 15): 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

q4 (size 15): 9 12 10 20 30 40 50 60 70 80 90 100 110 120 130

Testing other data type

q5 (size 10): a b c d e f g h i j

Testing const correctness

q5 (size 10): a b c d e f g h i j

q5 is not empty

Front item of q5: a

Back item of q5: j

Testing exception handling

Testing front() with empty queue

Caught queue underflow on front()

Testing back() with empty queue

Caught queue underflow on back()

Testing pop() with empty queue

Caught queue underflow on pop()

**Other Points**

* A makefile is required. Same as always. Make sure it has appropriate rules for all the pieces involved. Obviously, with only one .cpp file (assign6.cpp), there won't be very many rules to write!
* Note that the driver program assumes that your header file is called Queue.h. If you name your header file differently, you'll need to modify the driver program accordingly.
* **Do not compile the Queue.h header file directly in a g++ command.** The header file is copied into your source file by the #include directive, and then the source file is compiled. Compiling the header file directly will create a giant "pre-compiled header file" with the extension .gch which will use up your entire disk quota on Unix. If you do that accidentally, you need to remove the .gch file and then fix your makefile.
* Programs that do not compile on turing/hopper automatically receive 0 points.
* Submit your program using the electronic submission guidelines posted on the course web site.