CSCI 241 Assignment 6
100 points

Purpose

This assignment is an exercise in implementing the queue ADT using an array-based data structure.

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 class for this assignment, the `Queue` class. You will need to implement several methods and functions associated with this class.

The class should be implemented as two separate files. The class definition should be placed in an appropriately named header (`.h`) file. The implementations of the class methods and any other associated functions should be placed in a separate `.cpp` file for the class.

`class Queue`

Data members

This class contains a pointer to an integer that will point to the first element of a dynamically allocated array of integers (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 `VectorN` 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` rather than a method.

  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 an integer. 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 an integer. 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 an integer argument, the item 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 an unsigned integer argument, the new capacity to allocate for the queue array. It returns nothing. It should reserve additional capacity for the queue array equal to the new capacity passed in.

  Note that the "circular" nature of the queue array complicates copying the items from the queue array to the new temporary array.

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/Fall2016/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
               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 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 = 10; i < 80; 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 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 = 2; i <= 8; i+= 2)
```

```
   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 = 5; i < 70; 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 copy constructor()\n\n";

Queue 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 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 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 const correctness\n\n";

const Queue& r4 = q4;

cout << "q4: " << r4 << endl;
```

```
cout << "q4 size: " << r4.size() << endl;
cout << "q4 capacity: " << r4.capacity() << endl;
cout << "q4 is " << ((r4.empty()) ? "empty\n" : "not empty\n");
cout << "Front item of q4: " << r4.front() << endl;
cout << "Back item of q4: " << r4.back() << endl << endl;

q1 = r4;

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

q1.clear();

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;
}
```

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: 10 20 30 40 50 60 70
q1 size: 7
q1 capacity: 8
q1 is not empty

Testing pop()

q1: 40 50 60 70
q1 size: 4
q1 capacity: 8
q1 is not empty

Testing wrap-around on push()

q1: 40 50 60 70 2 4 6 8
q1 size: 8
q1 capacity: 8
q1 is not empty

Testing wrap-around on pop()

q1: 6 8
q1 size: 2
q1 capacity: 8
q1 is not empty

Testing queue resize on push()

q1: 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q1 size: 15
q1 capacity: 16
q1 is not empty

Testing copy constructor()

q1: 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q1 size: 15
q1 capacity: 16
q1 is not empty

q2: 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q2 size: 15
q2 capacity: 16
q2 is not empty

Testing front() and back()

Front item of q1: 6
Front item of q2: 6

Back item of q1: 65
Back item of q2: 65

q1: 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q1 size: 15
q1 capacity: 16
```

```
q1 is not empty

q2: 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q2 size: 15
q2 capacity: 16
q2 is not empty

Testing pop() to empty

6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q1 size: 0
q1 capacity: 16
q1 is empty

Testing assignment operator

q2 (size 15): 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q3 (size 15): 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65

Testing clear()

q2 (size 0):
q3 (size 15): 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65

Testing assignment to self and swap

q2 (size 15): 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q3 (size 0):

Testing chained assignment

q2 (size 15): 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q3 (size 15): 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q4 (size 15): 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65

Testing const correctness

q4: 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q4 size: 15
q4 capacity: 16
q4 is not empty
Front item of q4: 6
Back item of q4: 65

q1: 6 8 5 10 15 20 25 30 35 40 45 50 55 60 65
q1 size: 15
q1 is not empty

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.
- The class should have a header file for its class definition and a source code file for its implementation. Note that the driver program assumes that your header file is called `Queue.h`.
- Write this program in a fashion similar to the last two assignments. 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.
- 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.