CSCI 241 Assignment 7
100 points

Purpose

This assignment is an exercise in implementing the queue ADT using a singly-linked list. This assignment also introduces the concept of templates.

Assignment

This program creates and implements a class to represent the Queue ADT using a singly-linked list.

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 one template structure and one template class for this assignment, called `Node` and `Queue`. You will need to implement several methods and functions associated with these data types.

Since these are both C++ templates and are closely related to each other, 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.

`struct Node`

Data members

This template structure should have two data members: a member of the template parameter type to store an item to be inserted into the queue, and a pointer to a `Node`. The pointer `next` will point to the next node in the linked list (or be `nullptr` if this is the last node in the list).

Since the `Queue` class will need access to these data members, make them `public` (the default for a `struct`).

Methods

- Constructor

  The structure should have one constructor that takes an argument of the template parameter type. Make this argument a reference to `const` data. The constructor should copy the argument into the queue node and set the node's pointer to `nullptr`.

`class Queue`

Data members

This class should have three data members. The first two are pointers to `Node`s. The pointer `qFront` will point to the front node in the queue (or be `nullptr` if the queue is empty); the pointer `qBack` will point to the back or rear node in the queue (or be `nullptr` if the queue is empty). The third data member is used to keep track of the number of data items currently stored in the vector (the *queue size*). This data member should be declared as data type `size_t` (which corresponds to an unsigned integer).

Methods and associated functions

- Constructor

  The class should have a default constructor that takes no arguments. The constructor should set both pointer data members to `nullptr` and the queue size to 0.

- Destructor

  The class should have a destructor. The destructor can simply call the `clear()` method described below.

- Copy constructor

  The class should also have a proper copy constructor. If you choose to make a copy of the linked list by using the `push()` method, make sure you set both the front and back pointers for the new queue to `nullptr` and the queue size to 0 before you attempt to insert any nodes into it.

- `operator=`

  The copy assignment operator should be properly overloaded.

- `operator<<`

  The output operator should be overloaded so that an entire `Queue` can be sent to the standard output. As usual, this function will need to be a `friend` 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.

- `clear()`

  This method takes no arguments and returns nothing. It should properly set the queue back to the empty state. That means deleting all of the nodes in the queue, setting the front and back pointers back to `nullptr`, and setting the queue size back to 0.

- `size()`

  This method takes no arguments and returns a `size_t`. It should return the current size of the queue; i.e., the number of data items currently stored in the queue.

- `empty()`

  Returns `true` if there are no data items currently stored in the queue; otherwise returns `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 data stored in the front node of the queue.

- `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 data stored in the back node of the queue.

- `push()`

  This method takes a reference to a constant item of the template parameter type as its argument (the item to insert into the queue). It returns nothing. The method should insert the item at the back of the queue and increment the queue size.

- `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 remove the item at the front of the queue and decrement the queue size.

If you like, you may write `private` methods for the `Queue` class in addition to the methods described above. For example, you may want to write a `copyList()` method that can be called by both the copy constructor and overloaded assignment operator.

Driver Program

A driver program, `assign7.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/Assign7/assign7.cpp`.

```
/*****************************************************************
   PROGRAM:    CSCI 241 Assignment 7
   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 (size " << q1.size() << "): " << q1 << endl;
cout << "q1 is " << ((q1.empty()) ? "empty\n" : "not empty\n");
cout << endl;

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

q1.push(17);

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

q1.push(2);
q1.push(6);
q1.push(4);

cout << "q1 (size " << q1.size() << "): " << q1 << 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 (size " << q1.size() << "): " << q1 << endl;
cout << "q2 (size " << q2.size() << "): " << q2 << endl << endl;

cout << "Testing clear()\n\n";
q1.clear();

cout << "q1 (size " << q1.size() << "): " << q1 << endl;
cout << "q2 (size " << q2.size() << "): " << q2 << endl << endl;

Queue<int> q3;

q3.push(36);
q3.push(41);
q3.push(75);
q3.push(28);

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

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

Queue<int> q4;

q4 = q3;

cout << "q3 (size " << q3.size() << "): " << q3 << endl;
cout << "q4 (size " << q4.size() << "): " << q4 << endl << endl;


q3.clear();

cout << "q3 (size " << q3.size() << "): " << q3 << endl;
cout << "q4 (size " << q4.size() << "): " << q4 << endl << endl;

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

q4 = q4;
q3 = q4;
q4.clear();
```

```
cout << "q3 (size " << q3.size() << "): " << q3 << endl;
cout << "q4 (size " << q4.size() << "): " << q4 << endl << endl;

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

Queue<int> q5;

q5 = q4 = q3;

cout << "q3 (size " << q3.size() << "): " << q3 << endl;
cout << "q4 (size " << q4.size() << "): " << q4 << endl;
cout << "q5 (size " << q5.size() << "): " << q5 << endl << endl;

cout << "Testing front(), push(), pop()\n\n";

Queue<char> q6, q7;

for(char c = 'a'; c < 'k'; c++)
   q6.push(c);

cout << "q6 (size " << q6.size() << "): " << q6 << endl << endl;

for(int i = 0; i < 10; i++)
   {
   int val;

   val = q6.front();
   q7.push(val);
   q6.pop();
   }

cout << "q6 (size " << q6.size() << "): " << q6 << endl;
cout << "q7 (size " << q7.size() << "): " << q7 << endl << endl;

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

int val1, val2;

q6 = q7;

val1 = q6.back();
val2 = q7.back();
val1 = q6.back();     // Make sure that back() doesn't remove a value.

cout << ((val1 == val2) ? "back() works\n\n" : "back() failure\n\n");

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

val1 = q6.front();
val2 = q7.front();
val1 = q6.front();    // Make sure that front() doesn't remove a value.

cout << ((val1 == val2) ? "front() works\n\n" : "front() failure\n\n");

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

q7.clear();
const Queue<char>& r6 = q6;

cout << "q6 (size " << r6.size() << "): " << r6 << endl;
```

```
cout << "q6 is " << ((r6.empty()) ? "empty\n" : "not empty\n");
cout << "Front item of q6: " << r6.front() << endl;
cout << "Back item of q6: " << r6.back() << endl << endl;

q7 = r6;
Queue<char> q8(r6);

cout << "q7 (size " << q7.size() << "): " << q7 << endl;
cout << "q8 (size " << q8.size() << "): " << q8 << endl << endl;

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

try
   {
   cout << q1.front() << endl;
   }
catch (underflow_error e)
   {
   cout << "Exception: "<< e.what() << endl << endl;
   }

try
   {
   cout << q1.back() << endl;
   }
catch (underflow_error e)
   {
   cout << "Exception: "<< e.what() << endl << endl;
   }

try
   {
   q1.pop();
   }
catch (underflow_error e)
   {
   cout << "Exception: "<< 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>

template <class T>
struct Node
   {
   ...

   };

// Method definitions for the Node class

// 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 /* QUEUE_H */
```

Sample Output

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

```
Testing default constructor

q1 (size 0):
q1 is empty

Testing push()

q1 (size 1): 17
q1 is not empty

q1 (size 4): 17 2 6 4
q1 is not empty

Testing copy constructor
```

```
q1 (size 4): 17 2 6 4
q2 (size 4): 17 2 6 4

Testing clear()

q1 (size 0):
q2 (size 4): 17 2 6 4

q3 (size 4): 36 41 75 28

Testing assignment operator

q3 (size 4): 36 41 75 28
q4 (size 4): 36 41 75 28

q3 (size 0):
q4 (size 4): 36 41 75 28

Testing assignment to self

q3 (size 4): 36 41 75 28
q4 (size 0):

Testing chained assignment

q3 (size 4): 36 41 75 28
q4 (size 4): 36 41 75 28
q5 (size 4): 36 41 75 28

Testing front(), push(), pop()

q6 (size 10): a b c d e f g h i j

q6 (size 0):
q7 (size 10): a b c d e f g h i j

Testing back()

back() works

Testing front()

front() works

Testing const correctness

q6 (size 10): a b c d e f g h i j
q6 is not empty
Front item of q6: a
Back item of q6: j

q7 (size 10): a b c d e f g h i j
q8 (size 10): a b c d e f g h i j

Testing exception handling

Exception: Queue underflow on front()

Exception: Queue underflow on back()

Exception: 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 (`assign7.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.
- 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.