CSCI 2275 – Programming and Data Structures
Instructor: Hoenigman
Assignment 5
Due: Part 1: Wednesday, October 7, 6pm. Part 2: Monday, October 11

**Notice that the first part of this assignment is due this Wednesday by 6pm.**

## Think of a Linked List like a Train and a Queue like the line to board

In a linked list, you have a collection of nodes that contain data and a link to the nodes before and after it. The idea of a linked list is similar to a train, where you have individual cars that contain people and objects and the cars have a physical link to the cars before and after it. Before passengers can board the train, they might have to wait in line, in a queue, where the first person in the line is the first person to board the train.

In this assignment, we're going to build on the previous train assignment by adding a queue to control the order in which passengers board the train, and the ability to delete cars once passengers dis-embark.

## Functionality from previous assignment

**Build your train**
Your first step in this assignment is to build your linked list train. Each node in your linked list represents a train car with a name, capacity (the number of passengers the car can hold), and an occupancy (the number of passengers currently on the car).

Note: I'll refer to the linked list as "the train" throughout this document.

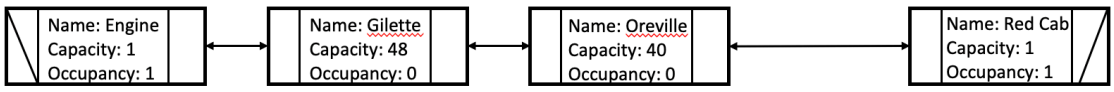
```
Struct trainCar:
      string name;
      int capacity;
      int occupancy;
```

```
trainCar *next;
trainCar *previous;
```

Include the following cars in your train, with these capacities. The initial occupancy should be 0, except for the Engine and the Caboose, which have an occupancy of 1.

**Train Car: Capacity**
The Little Engine That Could: 1
Gilette: 48
Oreville: 40
Keystone: 35
Addie Camp: 25
Redfern: 20
Mystic: 50
Rochford: 52
Harney Canyon: 40
Bluebird: 50
Red Caboose: 1

When you build your train, the order of the cars should be the same as the order listed above. The first three cars are as follows:




The head car – the engine – has a capacity and occupancy of 1, only the conductor rides in the engine. At the tail of the train is the caboose, which also has a capacity and an occupancy of 1.

**Add passengers and cars**
There is a file on Canvas called *ticketsSold.txt*, where each line in the file includes the number of tickets purchased and the name of the car. Your program needs to read the file and add passengers to the train by updating the occupancy of each car to reflect the number of tickets sold for that car. Once a car fills up – when occupancy = capacity, add another car with the same name and capacity to the train right after the existing car. Add the remaining passengers to the new car. For example, if 50 tickets are sold for Keystone, and there are only 20 seats remaining on Keystone, put 20 passengers in Keystone and then add a new Keystone car to the train with the remaining 30 passengers. If there are additional ticket sales for Keystone, those passengers would be added to the second Keystone car, and a new Keystone would be added if necessary. When a new car is added, give the car a number as part of its name, such as Keystone_2 to differentiate it from the other cars with the same name.

**Traverse the train**

After all of the tickets are read, print the name and occupancy of each car on the train, moving from the Engine to the Red Caboose and then back to the Engine. You should only print the Red Caboose once.

**Methods in the Train class**

1. **Build the train:** Build the linked list using the cars listed above in the order they are listed. Each car needs to have a name, a capacity, an occupancy, and a pointer to the next and previous cars. Takes an array of strings that are the names of the cars and an array of integers that are the capacities of each car.

2. **Print Train cars:** This option prints out the train in order from the Engine to the Caboose and then back to the Engine. Printing the path could be very useful to you when debugging your code. Print the name and occupancy of each car.

   ```
   cout<<trainCar->name<<":"<<trainCar->occupancy<<endl;
   ```

3. **Add Passengers**: After reading the line in the file, add passengers to the train. Takes the train car name and the number of passengers to add.

4. **Add Car:** Add a new car to the train when the occupancy of a car reaches the capacity. Takes the name, capacity, new occupancy, and previous car as arguments and adds a new car to the train. When a new car is added to the train, print <Car name>:<Capacity>:<Occupancy>

   ```
   cout<<trainCar->name<<":"<<trainCar->capacity
   <<":"<<trainCar->occupancy<<endl;
   ```

## Assignment 5 functionality

In assignment 5, you will build on the code you completed for assignment 4 to include the following functionality:

**Add a circular queue:**

Instead of reading from the file and directly adding passengers to the train, the car name and number of tickets should be added to a queue. In the queue.h header file, there is a carPassenger object that should be used to store the train car name and number of tickets for that car. Passengers are added to the car with the dequeue operation.

**Allow manual input with a menu**

In addition to reading from the file, the user should be able to enter the name of the train car and the number of passengers and this input will be added to the queue.

**Remove passengers from the train**

Your menu should have options removing passengers from a specified train car and removing all passengers. Reduce the count for the car as passengers are removed, and when the capacity=0, delete the car from the train.

**Delete cars from the train**
When the capacity of a car = 0, delete the car from the train and update the next and previous pointers for the surrounding cars accordingly. You shouldn't delete the engine or the caboose.

**Menu Options**
To incorporate this new functionality, you need to add a menu to your program with the following options.

1. **Enqueue from file**
    a. This option reads the next line in the file and adds an instance of *carPassenger* to the queue. If the queue was full and the object wasn't added successfully, you need to be able to add that same data to the queue once there is room in the queue. For example, if the line read from the file is "Keystone, 12", and the queue is full, the next time the user selects Enqueue from file and there is room in the queue, "Keystone 12" is added to the queue. You should accomplish this by storing the value and using the stored value instead of reading the next line from the file on the next enqueue from file operation. Print a message saying "queue is full", or "added to queue", depending on whether the operation was successful.
2. **Enqueue from user input**
    a. This option asks the user to enter the name of a car and a number of passengers, creates a *carPassenger* object, and adds it to the queue. Print a message saying "queue is full", or "added to queue", depending on whether the operation was successful.
3. **Dequeue**
    a. Dequeue from the queue and add the passengers to the train car. If the car is full, another car should be added using the *addCar* functionality from assignment 4.
4. **Remove passengers**
    a. This option simulates passengers leaving the train. The user enters a number of passengers and the car they are exiting. When a car is empty, delete it from the train. If the number of passengers > capacity, set capacity = 0, and delete the car.
5. **Remove all passengers**
    a. This option simulates all remaining passengers leaving the train. Start from the front of the train and move to the back of the train. Don't delete the engine or the caboose. Before a car is deleted, print the name of the car and the number of passengers.
6. **Print the train**
    a. This option prints the train cars and the number of passengers in each car, staring at the front of the train.

7. **Print the queue**
    a. This option prints the contents of the queue, starting at the head. Printing the queue will be really useful for debugging purposes.

**Assignment 5, Part 1: (10 points)**
**The first part of this assignment is due on Wednesday, Oct 7 by 6pm**, and requires no coding. You need to write a plan for how you're going to test your code as you're developing it. Think through the functionality of the program and write out what you want to tackle first, next, and so on. How will you test that functionality? What input will you create, and what output will you expect. For example, one of the items in your plan should be testing that you are correctly adding to the queue. You will want to test that you can add to the queue, that items appear in the correct order, and that you can't add to the queue when the queue is full. How will you create an input that isolates the queue functionality from other areas of the program?

**What to submit:** Write a one-page summary of the functionality of the program and how you will set up a test case for each unit of functionality.

**Suggestions for completing this assignment**
There are several components to this assignment that can be treated independently. My advice is to tackle the queue and train functionality separately before combining them. Test that you can enqueue and dequeue successfully before adding passengers to the train. Once you get one feature completed, test, test, test, to make sure it works before moving on to the next feature.

There are several examples of how to work with linked lists in Chapter 5, stacks in Chapter 6, and queues in Chapter 7 in your book. The queue code from recitation last week can also be used for this assignment, with some modification.

Your code should be separated into **five files** - *Train.h, Train.cpp*, *Queue.h, Queue.cpp*, and *Assignment5.cpp*. You need to write the *Train.cpp, Queue.cpp* and the *Assignment5.cpp* files. You are provided with a beginning shell of *assignment5.cpp* and the *train.h* and *queue.h* header files.

You can compile your code on the command-line using g++

```
g++ -std=c++11 Train.cpp Queue.cpp Assignment5.cpp -o Assignment5
```

and then run your program on the command-line using

```
./Assignment5 ticketsSold.txt
```

If you want to use gdb or lldb to debug your program, include the -g compiler flag when you build your code.

g++ -std=c++11 -g Train.cpp Queue.cpp Assignment5.cpp -o Assignment5

Also, start early.

**What to do if you have questions**

There are several ways to get help on assignments in 2275, and depending on your question, some sources are better than others. There is a discussion forum on Slack that is a good place to post technical questions, such as how to insert a node in a linked list. When you answer other students' questions on the forum, please do not post entire assignment solutions. The TAs are also a good source of technical information, especially questions about C++. If, after reading the assignment write-up, you need clarification on what you're being asked to do in the assignment, the TAs and the Instructor are the best sources of information.