CSCI 2275 – Programming and Data Structures
Instructor: Hoenigman
Assignment 4
Due: Friday, October 2

Think of a Linked List like a Train

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. In this assignment, we're going to represent a train as a linked list and simulate people buying tickets for a seat on the train. Unlike the real world though, where the train has a fixed number of people that it can hold, we can add new cars to our simulated train to accommodate all of the interested passengers.

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).

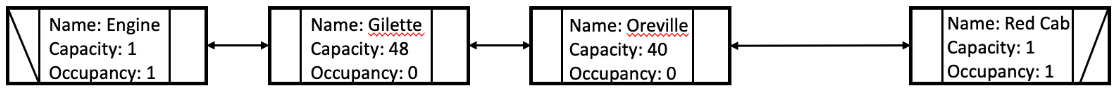
Note: I'll refer to the linked list as "the train" throughout this document.

```
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;
   ```

Structuring your program

The functionality for your network will be implemented in a class called Train. A suggested header file called, Train.h is provided for you on Canvas. You are welcome to structure your program differently than the structure provided in Train.h, or write additional helper functions if needed. Your code needs to be readable, efficient, and accomplish the task provided.

Here are the prototypes for the methods included in the Train.h header file.

*void Train::addCar(string name, int capacity, int passengers, trainCar *previous)*
/*
Insert a new car into the train after the previous. The name of the new car is in the argument name.
*/
void Train::addPassengers(string name, int numPassengers)
/*
Add passengers to the car. The name of the car to add to is the argument *name*. If the car is full, this method should call *addCar.*

*/

void Train::printTrain()
/*
Start at the head of the train and print the name and occupancy of each car in order to the end of the train, and then back again to the head of the train.
*/

void Train::buildTrain()
/*Build the initial train from the cars given in this writeup. The cars can be fixed in the function, you do not need to write the function to work with any list of cars. */

Suggestions for completing this assignment
There are several components to this assignment that can be treated independently. My advice is to tackle these components one by one, starting with building the train, and then print the train to verify that it looks like what you expect. From there, read the file and add passengers and cars 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 in your book, and we will also be covering these concepts in lectures this week.

Your code should be separated into three files - *Train.h, Train.cpp*, and *Assignment4.cpp*. You need to write both the *Train.cpp* and the *Assignment4.cpp* files. You can compile your code on the command-line using g++

```
g++ -std=c++11 Train.cpp Assignment4.cpp -o Assignment4
```

and then run your program on the command-line using

```
./Assignment4 ticketsSold.txt
```

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.