CSCI 235 – Software Design & Analysis II
Assignment 3

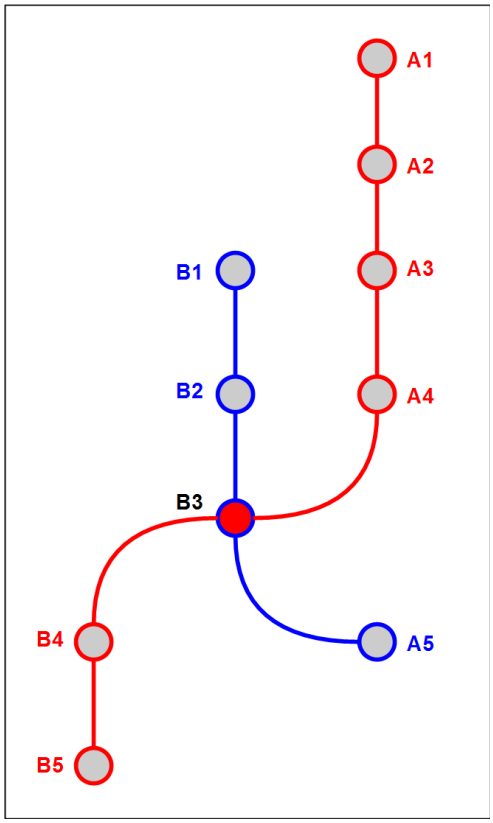
Introduction

This assignment gives you practical programming experience using the ADTs introduced during class. The program specified below must compile and run on the server eniac.geo.hunter.cuny.edu in G-Lab. You may work on this assignment by yourself or in a small team of two or three fellow students. If you choose to work in a group, each of the group members will receive the same grade. Your grade will be based on the following:

60% = Correctness (conformance to the requirements below)
20% = Design
10% = Performance
10% = Documentation and style

Assignment

Your assignment is to write a C++ program to support the simulation of passengers and trains in a system to support answering the question of the number of trains required for the volume of passengers.



The train system has 10 stations, configured as shown in the illustration.

There are two train lines, a red line and a blue line. With one exception, stations are serviced by either a red or a blue train. The one exception, station B3, is serviced by both the red train line and the blue train line.

The blue or red lines between stations indicate the tracks the train(s) run on. For example, the blue train(s) line stops at stations: B1, B2, B3 and A5; the red train(s) stop at stations: A1, A2, A3, A4, B3, B4 and B5. Once a train reaches the end of the track (at stations B1, A5, A1 or B5), the train turns around and heads in the opposite direction.

Station B3 is a potential transfer point between the lines. Passengers on a red line train may change to one on the blue line or vice versa at station B3. For example, a passenger wishing to go from station A2 to station B2 must board a red train at station A2, go through stations A3 and A4, must disembark the red train at station B3, must board a blue train at station B3 and ride it to station B2, disembarking there.

Trains are composed of 3 cars, each with 4 doors. Each car holds a maximum of 64 passengers. As a simplifying assumption, your simulation may assume that each car is identical (i.e. the cars do not

require additional space for the train's operators). After unloading and loading passengers, trains require 8 minutes travel time between “A” and “B” stations (i.e. between “B3” and “A4” … or between “B3” and “A5” in either direction).

The time of day determines the number of passengers, the stations from which they originate and their destination stations. This is determined by the (attached) configuration file. For example, the following line:

```
06.passengers.volume.A = 5
```

… indicates that in hour 6 (between 6:00 AM and 6:59AM), the number of passengers per minute entering the “A” stations (A1 – A5) is an evenly-distributed random number between 0 and 5.

Likewise, passengers enter stations knowing their destinations. For example, the line:

```
06.passengers.destination.B = 5
```

… indicates that there is a 5% chance (evenly distributed) that a passenger's destination is one of the “B” stations, regardless of the station from which the passenger originates. A passenger's destination is always a different station from the one which he or she originally entered.

The location in the station where passengers may enter train doors is predictable and marked in the station. When passengers enter a station, they wait at one of the marked areas. Your implementation may assume that the passengers are polite: that passengers enter trains on a first-come, first-served basis for each of the marked areas.

Your simulation must determine the minimum number of for each line trains required to ensure that a passenger's wait time to board a train (either at the original station or at a transfer point) is no longer than 5 minutes and that a passenger's total wait and transit time is no longer than 35 minutes from entering the train system and disembarking at final destination.

As described above, the program must read a property file which will indicate the various configurations required by the program. The property file will be “#” commented and be in key = value format. In other words, each line will either be blank, will contain a comment (starting with a “#” symbol) or will contain a pair:

```
key = value
```

The name of the property file will be supplied to the program on command line. For example, if your program is named “a3.exe” and the property file “a3.prop”, the following Unix command would start your program and read the property file:

```
./a3.exe a3.prop
```

In addition to the above, your implementation must be properly designed. Among other things, this means that the implementation must use one or more of the ADTs (List, Stack, Queue) discussed in class to date, and these ADTs must be your own implementation, not from the C++ STL.

Your implementation may:

1. Assume that passengers never get lost in the system (i.e. never take the wrong train)
2. Keep track of the (virtual) time each passenger enters and leaves the train system
3. Add any values (eg. the number of trains per line) to the property file

In addition, your implementation may rely on the attached Properties class. If you choose to use it, the associated files must be included and compiled as normal, and the class should be used similar to the

following:

```
// The first two lines should be called once, near the start of the program
Properties * prop = Properties::getInstance();  // Get instance of properties
prop->readPropertyFile("a3.prop");

// This next line may be repeated as many times as needed
string someProperty = prop->getProperty("11.passengers.volume.A");
```

Submission

Submit your source code along with the minimum number of trains per line required to achieve