ENEE 114: Programming Concepts for Engineers

Fall 2006 **Handout #29**

Project #4: Metro Simulator
Due Date: December 12, 11:59p.m.

In this project, you will build a program that performs a discrete-time simulation of a subway system similar to the Washington D.C. Metro. Your program will simulate passengers entering various metro stations, waiting for trains, boarding and disembarking trains, and then finally leaving the metro system. At the end of the simulation, your program will output the average passenger throughput (passengers processed per unit time), and the average waiting and transit times of passengers. Your program will also output a log file that shows the activity of trains and passengers throughout the system over time.

1 Simulator Structure

To simulate the metro system, your program will create and manipulate a set of dynamic data structures that will represent various components of the metro. Figure 1 illustrates the required dynamic data structures. Your program will maintain 3 types of data objects: stations, trains, and passengers. Each data object will be a different *structure* declared in your program. In this section, we describe the dynamic data structures in detail.

1.1 Lines and Stations

The metro system consists of several train *stations* (*e.g.* "Greenbelt," "College Park-U of MD," etc.) organized by train *lines* (*e.g.*, the Red line, Green line, etc.). Your program should create a pointer array to represent all the stations in the metro system, as illustrated in Figure 1. The pointer array should have 1 pointer for each train line, which points to an array of structures, one structure for each train station in the line. The index of each line in the pointer array is the line number, and the index of each structure in the structure array is the station number. The specification for the train lines and stations you should create in your pointer array is provided in a "stations file," explained in Section 2.1.

Each station structure should contain (at least) the following information. First, the station structure should store the name of the station in a character array. Second, the station structure should store the number of passengers waiting at the station at any given moment in time. Finally, the station structure should keep track of all the passengers that are waiting for trains at the station. You should track waiting passengers for trains traveling up the line (increasing station number) and down the line (decreasing station number) seperately, as indicated by the "up_waiting" and "down_waiting" structure fields in Figure 1. Because the number of waiting passengers is unknown beforehand, you must maintain the waiting passengers as linked lists, with "up_waiting" and "down_waiting" serving as the head pointers to each list.

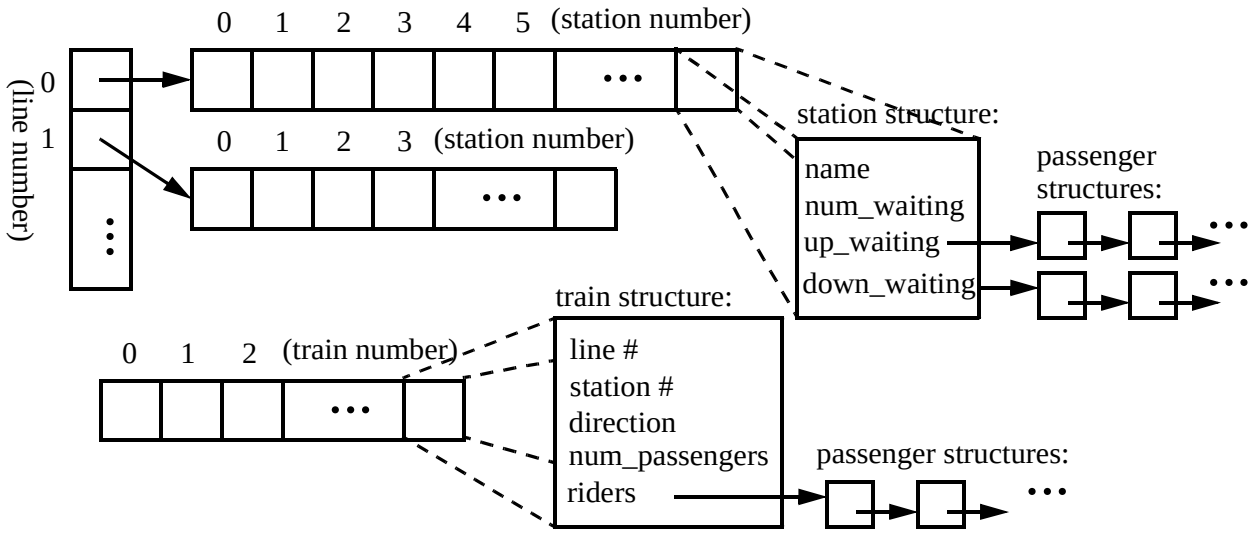

Figure 1: Recommended data structures for implementing the simulated metro system.

1.2 Trains

On each train line, there are trains that travel up and down the line, picking up and dropping off passengers as they pass through each train station. In this project, you will create 2 trains per line. Each train is represented by a structure containing (at least) the following information. First, the train structure should store the train line number it travels on, the train station number where the train is currently stopped, and the direction (either up or down the line) in which the train is currently traveling. Second, the train structure should store the number of passengers currently riding on the train. Finally, the train structure should keep track of all the passengers on-board the train at any given moment in time, as indicated by the "riders" structure field in Figure 1. It's recommended that you maintain the riding passengers as a linked list since the number of passengers will vary dynamically. In that case, "riders" serves as the head pointer to the list. Trains are numbered in the order of the line number on which they travel, with an even-numbered and odd-numbered train on each line.

1.3 Passengers

Waiting in the train stations and riding on the trains are the passengers. Each passenger is represented by a structure containing (at least) the following information. First, the passenger structure should keep track of how long each passenger spends in the metro system, and also how long each passenger spends waiting for trains. In addition, the passenger structure should also store the route information that specifies how to route the passenger to its destination train station. This can be done by storing a sequence of line/station numbers, and the number of hops in the route. (The specification of route information will be discussed in detail in Section 2.2).

2 Input Files

All the input files, described below, are available on the course web page. Just follow the link for the "Project 4 Files."

2.1 Stations File

The "stations file" is a text file that contains all the information about train lines and stations. You are to parse this file to create the pointer array dynamic data structure (described in Section 1.1) that represents all the train stations in your metro system.

The first line in the stations file contains an integer that specifies the number of train lines in the metro system. For each train line, the stations file specifies all the train stations that appear on that train line (in order of the train line number). The specification of each train line's train stations begins with an integer that denotes the number of train stations on the line, followed by a list of train station names (in order of the train station number).

For this project, we have provided two stations files: "stations-red.txt" and "stations-redgreen.txt." The first file specifies a metro system with a single train line (the D.C. Metro's Red line), consisting of 27 train stations. The second file specifies a metro system with two train lines (the D.C. Metro's Red and Green lines); the first train line consists of 27 train stations while the second train line consists of 21 train stations. In general, your program must be able to handle an arbitrary number of train lines, each with an arbitrary number of train stations.

2.2 Passengers File

The "passengers file" is a text file that contains information about passenger arrivals and routing information. Each line in the passengers file represents a single passenger in transit through the metro system, and contains several integer values. The first integer always specifies the passenger's arrival time. The second integer always specifies the number of hops in the passenger's route. The remaining integers specify the passenger's route. In the simplest passengers files, all passenger routes consist of a single hop (so the number of hops is always "1"). This means all passengers arrive at a train station, travel to a destination train station on the same train line, and depart the metro system from that destination train station. Such 1-hop routes are specified using 4 integers: source train line number, source train station number, destination train line number, and destination train station number.

For more complex passengers files, in addition to 1-hop routes, there can also be 2-hop routes (so the number of hops can be "1" or "2"). In a 2-hop route, a passenger first travels to a *transfer* train station, transfers from one train line to another, and then travels to a destination train station. Transfer train stations appear on more than one train line. For example, in the "stations-redgreen.txt" file, Gallery Pl-Chinatown and Fort Totten are both transfer train stations because they appear in both the Red (line #0) and Green (line #1) lines. 2-hop routes are specified using 8 integers rather than just 4. The first 2 integers specify the source train line number and source

train station number. The next 2 integers specify the source train line number and transfer train station number on the source train line. The next 2 integers after that specify the destination train line number to transfer to, and the transfer train station number on the destination train line. Finally, the last 2 integers specify the destination train line number and destination train station number. Notice in 2-hop routes, the middle 4 integers always specify transfer train stations.

We have provided 7 passengers files: passengers.test.txt, passengers.red1.txt, passengers.red2.txt, passengers.redgreen1.txt, passengers.redgreen2.txt, passengers.test-xfer.txt, and passengers.redgreen-xfer.txt. The first 3 passengers files are to be used with the stations-red.txt file (these passengers only travel on a single train line, the Red line), while the last 4 passengers files are to be used with the stations-redgreen.txt file (these passengers travel on two train lines, both the Red and Green lines). Also, the first 5 passengers files contain only 1-hop routes, while the last 2 passengers files contain both 1-hop and 2-hop routes. In all the passengers files, the passengers are listed in order of increasing arrival time.

Notice, passengers.test.txt and passengers.test-xfer.txt are very simple, and are useful for initially debugging your program. The passengers.test.txt file contains only two 1-hop passengers. (The first passenger arrives at station #2 on line #0 at time 1, and travels to station #5 on line #0; the second passenger arrives at station #7 on line #0 at time 2, and travels to station #3 on line #0). The passengers.test-xfer.txt file contains a single 2-hop passenger. (The passenger arrives at station #2 on line #0 at time 1, travels to transfer station #15 on line #0, transfers to station #10 on line #1, and then travels to station #7 on line #1).

3 Simulator Functionality

Your program should take 3 command-line arguments: the stations filename, the passengers filename, and the log filename (the log file will be discussed in Section 4). Your program should open the stations file in read mode, read its contents, and then build the pointer array data structure for the metro stations described in Section 1.1. Then, your program should create the train structures described in Section 1.2, creating 2 trains per train line read in from the stations file. On each train line, the two trains should be initialized such that one train is stopped at train station #0 moving in the upward direction (the even-numbered train), while the other train is stopped at the last train station in the line moving downward (the odd-numbered train). For example, for the Red line, the two trains should be initially stopped at the Shady Grove and Glenmont train stations. Your program should also open the passengers file in read mode, and the log file in write mode.

Next, your program will perform a *discrete-time simulation* of the metro system you just built, using the contents of the passengers file to drive the simulation. The main part of the simulation is a *simulator loop*. Each iteration of the simulator loop represents 1 unit of time, or a *timestep*. During each timestep, you will simulate the activity or actions that take place inside your metro system. Specifically, there are 4 major actions that occur in every timestep:

Passenger Arrival. Your program should read in passengers from the passengers file as the simulation progresses. For each passenger whose arrival time is equal to the current timestep, you should create a passenger structure, initialize it, and insert it into the up_waiting or

down_waiting linked list belonging to the train station where the passenger arrives (*i.e.*, the passenger's source train station), depending on whether the passenger is traveling up or down the train line.

Disembark Trains. For every train in the metro system, you should identify all the passengers riding the train who wish to disembark to the station at which the train is currently stopped. You should remove all such passengers from the riders linked list on the train. For disembarking passengers who have reached their final destination, you should update any relevant statistics (*i.e.*, passenger transit time and passenger waiting time), and free the passenger structure. For disembarking passengers who are transferring to another line, you should insert them into the up_waiting or down_waiting linked list belonging to the train station where they are transferring to.

Board Trains. After disembarking passengers, you should board all passengers waiting at a train station where a train has stopped. Passengers waiting in the up_waiting linked lists should only board trains traveling up the train line; passengers waiting in the down_waiting linked lists should only board trains traveling down the train line. Passengers should board trains in the order they arrive at the train station. If multiple passengers arrive during the same timestep, they should board in the order they appear in the passengers file. Trains should be boarded until they reach their maximum capacity. In your program, you should assume trains can hold up to 100 passengers.

Move Trains. After boarding passengers, you should move every train 1 station up or down their corresponding train lines, depending on the direction they are traveling. In addition, trains that move to a station that is located at an end of the line should change direction (for example, on the Red line, a train moving to the Shady Grove station should change from the down to up direction, while a train moving to the Glenmont station should change from the up to down direction). After all trains have been moved, your program should start the next timestep and repeat the above 4 steps.

Your simulation of the metro system should continue until you have reached the end-of-file in the passengers file (*i.e.*, all passenger arrivals have been processed) *and* all passengers have left the metro system. Before exiting, your program should output the number of timesteps simulated, the number of passengers processed, the passenger throughput (# passengers / # timesteps), the average passenger transit time, and the average passenger wait time.

4 Log File

Your program should create a log file that reports the activity of the simulated metro system on a timestep-by-timestep basis. Every timestep, you should print to the log file the timestep number, followed by a report of the status of every train and train line. For each train, you should print the train number, what station it's currently stopped at, and the number of passengers riding on the train. For each train line, you should print the train line number, and then print the number of passengers waiting at every train station on the train line (for each train station, print the train

station number, name, and the count of waiting passengers). After your simulation terminates, you should print to the log file the statistics described at the end of Section 3.

We have provided 7 output log files, one for each of the passengers files described in Section 2.2. The filename of these log files begin with "log," followed by the same extension used in the corresponding passengers file. (For example, "log.test.txt" is the output log file created by running a simulation with the "passengers.test.txt" file). You can download all the provided log files from the course web page. The output generated by your program should match the log files we provide *exactly*.

5 Grading

As in other projects, we will grade your project #4 by running the input files we provide to you, and matching the output of your program with the log files we provide. We will also run your program on input files that we have not provided to you. 80% of the grade for functionality will be determined by how your program behaves on passengers files with only 1-hop passenger routes, *i.e.*, passengers never transfer train lines. The remaining 20% of the grade for functionality will be determined by passengers files with both 1-hop and 2-hop passenger routes. It's recommended that you first get your program working for 1-hop passenger routes (*i.e.*, using the passengers.test.txt, passengers.red1.txt, passengers.red2.txt, passengers.redgreen1.txt, passengers.redgreen2.txt files) since most of the points for the project can be earned once this functionality exists. After that, you can extend your project to handle 2-hop passenger routes.