Grant Bingham

Dean Ouellette

Final Specification Document

This program will simulate the inflow traffic of C273ville, a town with 2000 residents. The simulation will run on a minute-by-minute basis and will simulate a full week of traffic inflow.

This program will have 8 classes:

* class Building
* class Car
* class DestinationStreet
* class ExitStreet
* class Random
* class Resident
* class Simulation
* class Street

And one .cpp file:

* Source.cpp

The main function will call the Simulation class. The Simulation class will read the first and last names of all 2000 residents from two separate files: one file will contain first names and the other will contain last names. The first names will be stored in an array of strings called firstnames[2000], and the last names in string lastnames[2000].

We will use these names to create 2000 Resident objects which take the first and last name strings as arguments.

Car objects will also be created as Residents go into town, each will be composed of a Resident\* driver. In addition to the Resident\* driver, the Car constructor will also take int clock and string destination as arguments, clock being the current time of the Simulation, and destination being where the Car is going, either to the School or the Bank. The Car class will also have the private data fields int currentRouteEntryTime, currentPlaceEntryTime, drivingSpeed and projectedArrivalTime. The currentRouteEntryTime and currentPlaceEntryTime data fields will both be set to “clock” when the Car object is constructed. The this->driver and this->destination will be set equal to the arguments passed into the constructor. Lastly, the drivingSpeed will be set to a random integer between 25 and 35, which will be generated by calling the next\_int function int the Random class. Once the Car object is created, it will begin its trip downtown.

The layout of the map shows that Cars will be travelling (and exiting) to destinations from two directions. Which direction the car comes from will be decided randomly.

All Street objects will have two priority queues of Car objects. One inwardQueue, for Cars travelling to either the bank or the school, and one outwardQueue, for Cars exiting town. Since all the cars are assigned different driving speeds, their order in the queue will shuffle around as they move through the queue. This is why the priority queue data structure is necessary, and this is where the projectedArrivalTime data field comes into play. The Car’s projectedArrivalTime is calculated based on currentPlaceEntryTime (the time at which the car entered the queue it is currently in), and the Car’s drivingSpeed. Each Street object has a data field int length. The Street object’s length / the Car’s speed \* 60 gives the amount of time it will take for the Car object to move through the queue. If this value is added to the currentPlaceEntryTime, the projectedArrivalTime is now calculated. Hence, the Car objects that have the lowest projectedArrivalTime (will move through the queue the quickest), are at the top of the priority queue.

The parent class Street has two child classes, ExitStreet and DestinationStreet. DestinationStreets are streets that are connected to a building. This simulation will have four total: James and Amber which connect to the bank, and Tulip and Birch which connect to the school. ExitStreets are streets that Cars will enter and exit on. They are not connected to any buildings, however, they are each connected to two DestinationStreet objects. There are two ExitStreets in this simulation: Travis, which connects to James (leading to the bank) and Birch (leading to the school), and Jackson, which connects to Amber (leading to the bank) and Tulip (leading to the school).

Also, there are two buildings: the school and the bank. Each building has two adjacent DestinationStreet objects. The building has a priotity\_queue of Cars called visitors, which functions the same as the Street queues, except the projectedArrivalTime data field is based on the random number of minutes between 5 and 10 given to each Car when they enter the queue.

Here is a summary of what a Car’s trip would look like. Let’s say a Car object is added to the inwardQueue of Cars on Jackson St. The Car is now on an ExitStreet moving toward two DestinationStreets. Once the Car is at the top of this ExitStreet inwardQueue, it will be added to one of the DestinationQueues. In this case, the Car will be added to either Amber or Tulip, based on whether it is headed to the bank or the school. This is handled by the void updateInner (int clock) method in DestinationStreet. Basically, this function pops the Car off of the ExitStreet Queue and pushes it into the DestinationStreet queue. Once the Car is in the new queue, it’s data fields are updated and the Car begins moving toward the building. When the Car is at the top of the DestinationStreet queue, the method void update (clock) is called from the Building class. This method acts just like the updateInner method in DestinationStreet and basically moves the car from the DestinationStreet innerQueue to the Building’s (bank or school) visitor queue. Once the Car is at the top of the visitor queue, it moves to the DestinationStreet outward queue, and then to the ExitStreet outward queue.

When the Car leaves town, it is popped from the ExitStreet outwardQueue and the Resident \*driver of that Car will have two of that Resident object’s data fields updated. The Car’s destination will be added to the Resident’s multiset of destinations, and the totaltimeSpentDowntown data field is updated.

This was all what one individual Car’s trip would look like.

Now that relationships between all the classes is more clear, we will go into more detail with the Simulation class, which initiates all of this when its functions are called by the main function. As mentioned before, the simulation class reads in all the residents name from a file and store them in an array and creates resident objects with these names. However, as these residents are created they are simultaneously being into a map that maps the string of the resident’s name to the corresponding resident object. This is all done by the getResidents() function, which is called by the Simulation default constructor when a Simulation object is constructed in main().

Then, the user is prompted to enter values for the visitor rates and the road capacities. Once the user has entered these values, six street objects (four destination, two exit) are created based on the user’s input, as well as two building objects. This all done by the enter\_data() function which is the next thing called in the main function.

The traffic simulation is now ready to begin. The total time is set to 7 \* 24 \* 60, the number of minutes in a week. Then there is a for loop that counts from zero up to this number. In each moment, the inward and outward queues are updated in the roads and buildings, and Car objects are pushed into the Exit streets based on the visitor rate. Once the time is up, the Residents that are currently in town are all collected and stored in a vector toBePutBack. This is all done by the run\_simulation() function which is the next thing called in the main function.

Now, the program will output the stats from the simulation. These stats include the total number of trips made downtown throughout the week, the average time of each trip, as well as the total time spent downtown from everyone. This is done by the show\_stats() function which is called in the main function after run\_simulation().

Lastly, the user is displayed a menu with the options of seeing all the visitors, seeing the record of a specific resident, or exiting the program. This is done by the display\_residents() function which is the last thing called in main.

Use Cases:

There are two main things that the user has control over in this simulation. The user will be prompted to enter both the average hourly visitor rate as well as the road capacity. The range for visitors/ hour is 1 – 60, and the range for the road capacity is 5 - 50.