COMP 11 – Introduction to Computer Science – Fall 2002

Project pp8 – Revised Version

Programming project due the week of 11/4

Due dates:
Section 01 (Block H): Wednesday 11/13 11:00 pm
Section 02 (Block E): Thursday 11/14 11:00 pm

Submissions may begin on Monday 11/4 at 5pm.

Project pp8: 8 points for correctness and 1 point for style.
The car wash simulator with multiple cars

Overview

We take one step closer to the full car wash simulator by simulating multiple cars and multiple washing bays. Our goal in this project is to determine the average and maximum length of the queue during the day, and also the average time it took for each car to go through the car wash (including the time waiting in line).

Here is the basic structure. There is one car wash, which has one queue and one washing bay. We are simplifying the car wash for this project by combining the cashier and washing bay into one entity, which we refer to as the washing bay. So, in this project there are only 3 states the car can be in.

queued: Waiting in the queue to get to the washing bay.

washing: Being washed.

elsewhere: Not at the car wash.

There will be multiple cars in each simulation. Also, many cars can be in the queue at a given time but only one car can be in the washing bay.

You must define a class that represents the cars. Since you will have many cars, you will need an array of car objects (i.e., a car “object” is an instance of the car class). As you remember, in C++ the size of an array is a constant, which will limit the number of cars you can simulate. Make this constant 10,000.

This project is different from the earlier car wash projects in several ways. First, you will simulate several days. Second, during each day, each simulated car will be washed exactly once. At the beginning of each day, you will determine how many cars will be washed that day. The “leftover” cars in the array will be ignored. Also, before simulating the day, you will also determine the minute at which each car will arrive at the car wash and will store that time in each simulated car.

You will also be told how long it takes to go through the washing bay.

Once you have that information, you can perform the simulation. You write 2 nested loops. The outer loop simulates each day and the inner loop simulates each minute of that day.

The simulation of each car for a given day goes roughly as follows. Before simulating the day, you calculate the number of cars for that day and the time each will be washed. See below for how to do that. All times will be in minutes. Record this time in the car object and set the initial state of the car (note that it may not be ‘elsewhere’). For each simulated minute, check the state of every car and update it as needed. If it’s time for the car to be washed, the state changes from 'elsewhere’ to 'queued'.
However, changing from ‘queued’ to ‘washing’ is not easy because only one car can be in the bay at a time. To do this correctly, you must keep track of which car is in the bay and also keep track of when it will finish. Only at that time does a car move from the queue to the bay.

But which car comes out of the queue? You must determine the car that arrived first in the queue. If several arrived at the same time, you can pick any of them that tied.

In addition, you must track enough data to be able to report the average (i.e., mean) time each car spent in the car wash, as well as the longest time any car spent at the car wash. The time at the car wash for a given car is the queue time + washing time. You must also track enough data so that you can calculate the average length of the queue per minute, as well as the longest the queue was in any given minute. All this reporting is over all the simulated days.

**Parameters for each simulation**

Your program must read in these parameters at the beginning.

- # of minutes spent per car in the washing bay: integer, minimum 1.
- Number of hours per day that the car wash is open: integer, minimum 1.
- Number of days to simulate: integer, minimum 1.
- Average number of customers that arrive at the car wash per hour: integer, minimum 1.

For a given day, the number of cars that arrive in a day is the “average # per hour” * “the number of hours the car wash is open”. Your simulation must randomly distribute the arrival of those customers over the course of the day.

You need not check for valid data. Assume it is valid.

**Data to report**

Report the average and maximum of the following.

- The queue length per minute.
- The time each car spent at the car wash.

For the averages, your answers must be accurate to 2 decimal places.

**Overall Design**

1. Read the 4 input parameters.
2. Perform the simulation.
   (a) Set up initial parameters for simulation.
   (b) Set up objects and variables as necessary.
   (c) Loop one iteration per simulated day. For each day:
      i. Calculate the number of cars needed for the day.
      ii. Set up objects and variables as necessary.
      iii. Determine when each car will arrive at the car wash.
iv. Loop one iteration per simulated minute. For each minute:
   A. Update the states of all the objects.
   B. Update your measurements.

3. Calculate and report the required information.

Since you already have the design for the program, please turn in the following for your design homework. Design the class definition you will need for the car class. Be sure to include the data and the member functions needed. For each of these, provide a brief description of their purpose.

**Determining when each car arrives at the car wash**

Let’s assume that the car wash is open 10 hours per day, which is 600 minutes. For each car, pick a random number between 0 and 599 (inclusive) and store that value in the car object. That is the time that car will arrive at the car wash. Remember that the first minute is minute zero!

You may have several cars arriving at the same time. That’s OK.

Also, you may have some cars that arrive so late in the day that the simulation stops while they are still in the queue or the bay. That’s OK also. In your calculations of statistics, just treat those cars like the ones that finished.

To do the above, you need a function that generates a random number. We provide you with:

```c
int randomInt (int n);
```

which returns a randomly chosen integer \( x \) between 0 and \( n-1 \) inclusive, i.e., where \( 0 \leq x < n \). So, for the above example, call randomInt(600) for each car.

As you may discover, programs that rely on randomness are very hard to debug. To help you, the function randomInt is *not* really random. Each time you call it, you will get a different “random” number. But when you run your program again, randomInt will return the exact same “random” numbers in the exact same order.

If you want to test your program with a different sequence of random numbers, edit the file containing randomInt: you will find instructions in the file. Also, if you want real random numbers (i.e., where randomInt returns a different sequence of numbers each time your program is run), you can do that also: again, instructions are in the file.\(^1\)

This function is located on two files named `/comp/11/lab/randomint.h` and `/comp/11/lab/randomint.cpp`. To use these files, do the following.

1. Copy both files into the directory where you have your code for pp8.

2. In your header file for pp8, add the line:

   ```c
   #include "randomint.h"
   ```

3. When you compile your program, be sure to add “randomint.cpp” to the list of files you give to g++.

4. But when you submit your program, do **not** submit either randomint.h nor randomint.cpp.

When you submit your program, the grading program will substitute a different randomint.cpp file. This one won’t really be random either, so we can test your outputs.

\(^1\)We are lying here. You cannot get truly random numbers from a computer, but you can get numbers that are “almost” random. This is usually good enough for applications that require real randomness.
Requirements
You must do the following in your program.

- Use objects to represent the cars. You can use objects for items also but this is not required.
- Use member functions to manipulate your objects. *All* variables within an object must be private. For example, you might have a member function for cars called “update” that updates the state of the car for the next simulated minute.
- You must use a header (.h) file and at least one source code (.cpp) file. You can use more than one source code file but it is not required.
- The grader will completely ignore your output except for the 4 numbers that you report. Precede each of these numbers by a ’#’ and do not print ’#’ elsewhere.

Sample output
Here is a sample run with output for your program. The bold characters are those typed by the user.

```
% a.out
Number of minutes spent in the washing bay: 5
Number of hours car wash is open per day: 1
Number of days to simulate: 1
Average number of customers per hour: 5
Average queue length was # 0.17
Maximum queue length was # 2
Average time spent at car wash was # 7.00
Maximum time spent at car wash was # 11
```

Why do you get the above? The simulation is for 60 minutes with 5 cars. We assume that the random choice of wash times for each car is as follows.

Car 0: 15
Car 1: 30
Car 2: 30
Car 3: 45
Car 4: 52

Here are the states of the cars (the ’elsewhere’ states are not shown) for each of the relevant minutes.

<table>
<thead>
<tr>
<th>Minute</th>
<th>Queued</th>
<th>Washing</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1,2</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
The average queue size is $\frac{10}{60}$ or 0.17. The maximum queue size was 2. The average time spent at the car wash was $\frac{35}{7}$ or 7 minutes. The maximum time spent at the car wash was 11 minutes.

**Style**

To get the style point, you must satisfy **all** of the following requirements.

- Produce output that is easily read.
- Have an introductory comment at the top of your program and preceding each function.
- Have a comment for each variable describing its purpose.
- Separate functions by blank lines.
- Within each function, separate logical sections by blank lines.
- Have a comment at the start of every section and every function.
- Choose meaningful names for variables and functions.

Your *first* submission will be used for style grading, even if your best grading score is on a different submission.

**How to submit your program**

Assume that your program is in two files named pp8.h and pp8.cpp. To submit it, type:

```
provide comp11 pp8 pp8.h pp8.cpp
```

from an Andante computer. You can submit up to 3 times. Your grade will be the highest grade you receive out of all your submissions.