Community College of Philadelphia

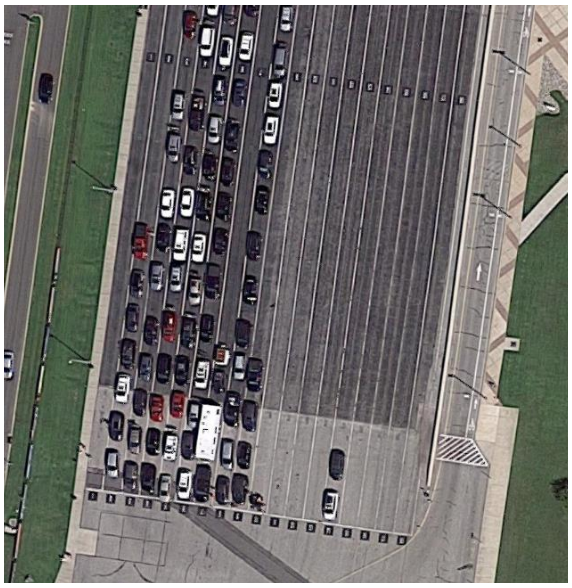
CSCI211 – Data Structures and Algorithms
ASSIGNMENT 5

Instructor: Michael Hackett
Department: Computer Science
Email: mhackett@ccp.edu

Ferry Queues

Using C++, Java, or Python, design a program to simulate the parking queues of the Cape May-Lewes Ferry. The image below shows the layout of the terminal.

The next picture shows the queues of cars waiting to board.



There is a total of 10 queues (Lanes 1 through 10) with each having a capacity of 30 cars, supporting 300 cars in total.

To represent each Car, design a Car object that simply contains an ID number. Every time you create a Car object, make sure its ID is one greater than the previously created Car.

The cars are boarded on the ferry starting with Lane 1. Once Lane 1 is empty, then cars in Lane 2 are boarded and so on. When the boat is full, the remaining cars will need to wait for the next boat. The next boat will start boarding from the lane where the previous boat left off. If, for example, a boat left off on Lane 5, the next boat will start its boarding with Lane 5. Once Lane 10 is emptied, the process starts over with Lane 1.

A ferry has a capacity of 100 cars.

The Driver Class/Main

In a main function, you'll set up your 10 queue structures for storing Car objects. The program will indefinitely ask the user to enter one of the following three commands:

add – Entering this command will cause the program to prompt for the number of cars to add to the lanes. The program will then generate the appropriate number of Car objects, and store them in the next available lane. The program will print how many cars were assigned to each lane.

- If the user tries to add cars for which there is no more room in any lane, add as many cars as possible and print how many cars were turned away.

load – Entering this command will start the boarding process. The program will then remove the appropriate number of cars from the appropriate lanes. The program will print how many cars were removed from each lane and the lane numbers of the affected lanes (the total number of cars should add up to 100).

exit – End/exit the program.

If the user enters and invalid command, print an error message and ask them to try again.

You may **not** use built-in queue/queue-like structures in this program.

Submit any and all related source code files in the Assignment 5 submission link.

Grading

See Assignment Rubric in Canvas.

Sample Input/Output

```
Command: add
Number of cars: 160
Added 30 cars to Lane 1
Added 30 cars to Lane 2
Added 30 cars to Lane 3
Added 30 cars to Lane 4
Added 30 cars to Lane 5
Added 10 cars to Lane 6

Command: load
Loaded 30 cars from Lane 1
Loaded 30 cars from Lane 2
Loaded 30 cars from Lane 3
Loaded 10 cars from Lane 4

Command: add
Number of cars: 100
Added 20 cars to Lane 6
Added 30 cars to Lane 7
Added 30 cars to Lane 8
Added 20 cars to Lane 9

Command: load
Loaded 20 cars from Lane 4
Loaded 30 cars from Lane 5
Loaded 30 cars from Lane 6
Loaded 20 cars from Lane 7

Command: add
Number of cars: 100
Added 10 cars to Lane 9
Added 30 cars to Lane 10
Added 30 cars to Lane 1
Added 30 cars to Lane 2

Command: load
Loaded 10 cars from Lane 7
Loaded 30 cars from Lane 8
Loaded 30 cars from Lane 9
Loaded 30 cars from Lane 10

Command: load
Loaded 30 cars from Lane 1
Loaded 30 cars from Lane 2

Command: load
All queues empty.
```

```
Command: add
Number of cars: 400
Added 30 cars to Lane 3
Added 30 cars to Lane 4
Added 30 cars to Lane 5
Added 30 cars to Lane 6
Added 30 cars to Lane 7
Added 30 cars to Lane 8
Added 30 cars to Lane 9
Added 30 cars to Lane 10
Added 30 cars to Lane 1
Added 30 cars to Lane 2
Could not add 100 cars. All queues full.
```