CptS **122 – Data Structures**

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Programming Assignment 5: Grocery Store Simulation

Assigned: Monday, October 8, 2018
Due: Friday, October 19, 2018 by midnight

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:
- Design and implement a dynamic queue
- Allocate and de-allocate memory at runtime
- Manipulate links in a dynamic queue
- Insert items into a dynamic queue
- Delete items from a dynamic queue
- Traverse a dynamic queue
- Design, implement, and apply test cases in C++
- Design test classes in C++

II. Prerequisites:

Before starting this programming assignment, participants should be able to:
- Analyze a basic set of requirements for a problem
- Compose C++ language programs
- Create basic test cases for a program
- Apply arrays, strings, and pointers
- Summarize differences between array notation and pointer notation
- Apply pointer arithmetic
- Apply basic string handling library functions
- Define and implement classes in C++
- Summarize the operations of a linked list
- Describe the operations of a queue including: enqueue ( ), dequeue ( ), printQueue ( )

III. Overview & Requirements:

Note: parts of this assignment were inspired by Deitel and Deitel's Supermarket Simulation problem. We've all had the pleasant experience of standing in line at the grocery store. As I'm standing in line I'm always trying to figure out if I chose the line with the fastest service. In most cases, I fail miserably. Let's write a program to simulate two lines in a grocery store, which will allow us to better understand how to select the fastest line. Note: you'll be required to write some test cases for this program. I **highly** recommend that you start with your tests before you implement any other aspect of the program. Starting with your tests will also allow for you to better design and implement your main application code.

For this assignment you will need to simulate two lines using queues. This will require that you develop enqueue ( ) (insert), dequeue ( ) (delete), and printQueue ( ) operations for a queue. Although you will instantiate two queues, each one of these will consist of the same kind of queue nodes. Define a queue node in the following manner:

```
class Data
{
public: // Member functions

private:
     int customerNumber;  // Unique identifier; starts at 1; after 24 hours should be reset to 1
     int serviceTime;     // Random time; varies between express and normal lanes; units in minutes
     int totalTime;       // totalTime = serviceTime + sum of serviceTimes of customers in line before this
customer; units in minutes
};  // This memory needs to be allocated on the heap!
```

```
class QueueNode
{
public: // Member functions

private:
     Data *pData;     // The memory for Data will need to be allocated on the heap as well!
     QueueNode *pNext;
};
```

You must also define a queue in the following manner:

```
class Queue
{
public: // Member functions

private:
     QueueNode *pHead;
     QueueNode *pTail;
};
```

One of your queues will represent the express lane and the other a normal lane. You will randomly generate arrival times and service times of customers into each lane. The express lane has customers arrive every one to five minutes, and customers arrive every three to eight minutes in the normal lane. Service times vary from one to five minutes, and three to eight minutes, for express and normal lane customers, respectively. As customers arrive into each line print out a message indicating in which line each customer arrives, along with the overall corresponding arrival time and customer number. When customers have finished checking out, print out a message indicating which line the customer was in, along the corresponding customer number and `totalTime` in the line. Allow for the simulation to run for `n` number of minutes, where `n` is inputted by the user.

The general program flow is as follows:

1. Generate a random number between 1 – 5 and 3 – 8 for express and normal lanes, respectively. This represents the arrival time of the first customer into each lane. Set the variable for total time elapsed to 0.
2. As customers arrive into each line, randomly generate a service time for each. Start processing the customers in the lanes based on service time. Randomly generate the arrival time of the next customer into each line. Elapsed time should be updated by one unit.
3. As each minute elapses, a new customer may arrive and/or another customer may be done checking out. Display the appropriate messages as described above.
4. For every 10 minutes, print out the entire queue for each line
5. Repeat steps 2 through 4 for `n` minutes of simulation.

Hints: since this is a simulation one minute is really one unit of time. Thus, the incrementing of an integer variable could represent one minute of time elapsing.

Required test cases:

Declare and define a test *class* for your application. You must declare your test class in a separate .h file from your other classes. You must also place your test case implementations in a separate .cpp file from your other classes. You must implement a total of 5 test cases. You must write the following test cases:
- One test case that executes your `enqueue()` operation on an empty queue
- One test case that executes your `enqueue()` operation with one node in the queue
- One test case that executes your `dequeue()` operation on a queue with one node
- One test case that executes your `dequeue()` operation on a queue with two nodes
- One test case that runs your simulation for 24 hours

BONUS:

Modify `QueueNode` such that it contains a pointer to the start of a dynamic singly linked list. The linked list will consist of grocery items corresponding to one person. These items should be strings like “cereal”, “milk”, “steak”, etc. Adjust the `serviceTime` of the `QueueNode` so that it is no longer random, but proportional to the number of items for the person served.

IV. Submitting Assignments:

1. Using the OSBLE+ MS VS plugin, please submit your solution. Please visit https://github.com/WSU-HELPLAB/OSBLE/wiki/Submitting-an-Assignment for more information about submitting using OSBLE+.
2. Your project must contain at least two header files (a .h file) and three C++ source files (which must be .cpp files).
3. Your project must build properly. The most points an assignment can receive if it does not build properly is 65 out of 100.

V. Grading Guidelines:

This assignment is worth 100 points. Your assignment will be evaluated based on a successful compilation and adherence to the program requirements. We will grade according to the following criteria:

- 5 pts – Appropriate top-down design, style, and commenting according to class standards
- 15 pts – Correct usages of two queues in simulation
- 5 pts – Correct `Data` class
- 5 pts – Correct `QueueNode` class
- 7 pts – Correct `Queue` class
- 10 pts – Correct constructors/destructors for the classes
- 5 pts – Correct getters/setters for the classes
- 5 pts – Correct `printQueue()`
- 3 pts – Correct `isEmpty()`
- 15 pts – Correct `enqueue()`
- 15 pts – Correct `dequeue()`
- 10 pts (2 pts/test case) – Correct test cases for the application

- BONUS: Up to 20 pts for implementation of grocery item list/customer