CSCI203/CSCI803 ASSIGNMENT 2

(10 marks + 2 demo marks)

Step-1 demo due during your Week-7 Lab class (2 marks)
Final submit due Week-9, Fri 11:59pm 27 Sept. (10 marks)

You have been hired by a major supermarket chain to models the operation of a proposed supermarket by using a Discrete Event Simulation. The shop has several servers (or checkouts), each of which has a different level of efficiency due to the experience of the server operator. It also takes longer to serve credit card customers. The service time of each customer is calculated as follows:

service_time = (tally_time x efficiency) + payment_time

where:

tally_time:	time it takes to tally up the customer's goods
efficiency:	the efficiency of the server
payment_time:	0.3 for cash or 0.7 for credit card

The input file "ass2.txt" contains the following information.

1. The number of servers. (You may assume a maximum of 20 servers.)
2. The efficiency of each server.
3. Records of each customer consisting of the *arrival time*, *tally time* and *payment method* (cash or card).

Your program should:

1. Open the text file "ass2.txt" (Note: "ass2.txt" should be a hardcoded as a const.)
2. Read the efficiencies of each server.
3. Read and process the customer arrival and service time data.
4. Print service statistics on the screen.

Note:

1. This shop has a single queue of customers.
2. The servers are initially all idle.
3. If more than one idle server is available, the next customer is served by the server with the best efficiency (i.e. the smallest efficiency value).
4. Customers must be served or queued in the order in which they arrive.
5. You should not attempt to read in all the arrival data at the start of the simulation.

At the end of the simulation, when the last customer in the file has been served, your program should print out the following information:

1. The number of customers served.
2. The time that it took to serve all the customers.
3. The greatest length reached by the customer queue.
4. The average length of the customer queue.
5. The average time spent by a customer in the customer queue. (If a customer is served immediately, their queue time is 0.0).
6. The percentage of customers who's waiting time in the customer queue was 0 (zero).
7. For each server:
    a. The number of customers they served
    b. The time they spent idle.

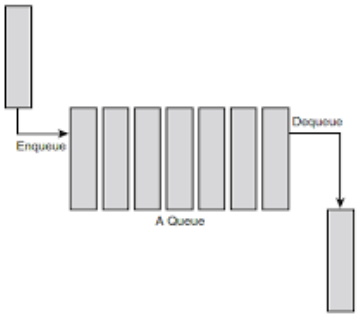
You must choose appropriate data structures and algorithms to accomplish this task quickly. You will lose marks if you use STL, or equivalent libraries, to implement data structures or algorithms. Note: A larger input data file may be used for the final assessment of your program.

Step-1 *(Week-7 demo, 2 marks)*

For step 1 you are to implement the simulator's *customer queue* and *event queue*.

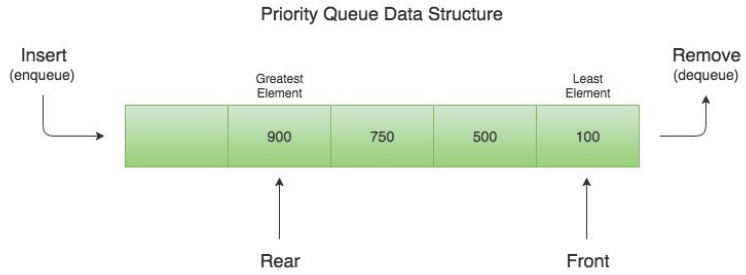
Implement the *customer queue* (FIFO queue). It should have a maximum size of 500 records. Each record in the customer queue should contain two doubles and a boolean (i.e. *arrival time, tally time* and *payment method*). Your customer queue should have functions for adding an item (enqueue), removing an item (dequeue), and for testing if the queue is empty. (Note: C++ and Java coders should also add a constructor for initialsing the queue.)

FIFO Queue



Test your customer queue by declaring an instance of it in the main(). Use a for loop to add 10 records to the queue. Set the *arrival time* for each record between 1 and 100 using rand() or random(). The other fields can be set to 0 or false. Also, print the *arrival times* on the screen as you add them. Now use a while loop to remove all the records from the customer queue and print the *arrival times* on the screen. Is the output correct?

Now implement the simulator's event queue (i.e. a priority queue). The event queue's records should contain an *event type* (int or enum), *event time* & *tally time* (doubles), and *payment method* (boolean). You can assume the maximum number of events in the event queue is 100. The record with the minimum *event time* has the highest priority.



Test the event queue by declaring an instance of it in the main(). Use the while loop (implemented previously) to remove records from the customer queue and add them to the event queue. Set the *event time* with the customer's *arrival time*. Set the other fields to zero. Then implement a while loop in the main() to remove (dequeue) each record from the event queue and print the *event time* on the screen. Is the output correct?

Note: For step-1 (to get the 2 demo marks) you can implement the customer and event queues using any method you like. However, for the final submission, to get full marks, you should ensure all data structures and algorithms are optimised for speed.

Step-2 *(Server array implementation)*

The server array should have a maximum of 20 servers. Each server should have a busy flag (boolean), an efficiency factor (double) and other data members for calculating the stats. C++ and Java coders should implement the server array with a class, preferably, and provide public functions for adding, and removing customers to/from the servers, finding the fasted idle server, etc. according to the specs on page 1.

Step-3 *(Processing in the data)*

When you have the *customer queue*, *event queue* and *server array* correctly completed, delete the main() test code from step 1 and replace it with code for reading the input data file "ass2.txt" and processing the data, as explained on page 1. The following algorithm shows how a typical discrete time simulator can be implemented:

```
main()
    Declare variables and instances and do initialisations
    Open the input data file;  if not found print error and exit
    Read first CustomerArrival event from file and add it to the event queue
    While the event queue is not empty . . .
        Get the next event from the event queue and set CrntTime to the event time
        If the event type = CustomerArrival event . . .
            if an idle server is available . . .
                Find fastest idle serve
                set the server's idle flag to busy
                calculate the server's finish time from event's customer data
                add ServerFinish event to the event queue
            Else
                Add event's customer to the customer queue
            End if
            If not EOF . . .
                Read next customer arrival from file
                add CustomerArrival event to the event queue
            End if
        Else // event type must be a ServerFinish event . . .
            Get server no. from event, set server[no] to idle and do server's stats
            If customer queue is not empty . . .
                Get next customer from the customer queue
                Find fastest idle serve
                set the server's idle flag to busy.
                calculate the server's finish time
                add ServerFinish event to the event queue
            End if
        End if
    End while
    Print stats
End main()
```

Step-4 *(Optimisation and stats)*

When you have the discrete time simulation working correctly, add the necessary data members and variables needed for calculating all the required stats, as explained on page 1, and optimise your simulator for speed.

Step-5 *(Specifications)*

In a comment block at the bottom of your program (no more than 20 lines of text) list all the data structures and algorithms used by your program to process the input data. Include any enhancements you did to speed up your program (if any). For this step, marks will be awarded based on how accurately and clearly you describe your program.

Compilation

All programs submitted must compile and run on banshee:

```
C:        gcc ass2.c
C++:      g++ ass2.cpp
Java:     javac ass2.java
Python:   python ass2.py
```

- Programs which do not compile on banshee with the above commands will receive zero marks. It is your responsibility to ensure that your program compiles and runs correctly.

Marking Guide

- Marks will be awarded for the appropriate use of data structures and the efficiency of the program at processing the input and producing the output.
- Marks may be deducted for untidy or poorly designed code.
- Appropriate comments should be provided where needed. All coding and comments must be your own work.
- There will be a deduction of up to 4 marks for using STL, or equivalent libraries, rather than coding the data structures and algorithms yourself. You may use *string* or *String* type for storing words or text, if you wish.

Submission:

Assignments should be typed into a single text file named "ass2.*ext*" where *"ext"* is the appropriate file extension for the chosen language. You should run your program and copy and paste the output into a text file named: "output.txt"

Submit your files via the *submit* program on banshee:

submit -u *user* -c csci203 -a 2 ass1.*ext output.txt*

- where *user* is your unix userid and *ext* is the extn of your code file.

Late assignment submissions without granted extension will be marked but the points awarded will be reduced by 1 mark for each day late. Assignments will not be accepted if more than five days late. An extension of time for the assignment submission may be granted in certain circumstances. Any request for an extension of the submission deadline must be made via SOLS before the submission deadline. Supporting documentation should accompany the request for any extension.