MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS

CNG213 C Programming – Programming Assignment 3

Date handed out: 11 December 2015, Friday

Date submission due: 24 December 2015, Thursday

"Telephone System Simulator"

In this assignment, you will write a program that simulates the operation of a telephone system that might be found in small business, such as your local restaurant. Only one person can answer the phone (a single-service queue), but there can be unlimited number of calls waiting to be answered.

Queue analysis considers two primary elements, the length of time a requester waits for service (the queue waiting time – in case, the customer is calling for an order) and service time (the time it takes the customer to place an order). Your program will simulate the operation of the telephone and gather statistics during the process.

The program will mainly require two inputs to run the simulation: (1) the length time in hours that the service will be provided and (2) the maximum time it takes for the operator to take an order (the maximum service time).

For elements are required to run the simulation: a timing loop, a call simulator, a call processor and a start call function.

1. **Timing loop**: This is simply the simulation loop. Every iteration of the loop will be considered 1 minute real time. The loop will continue until the service has been in operation the requested amount of time (given input above). When the operating period is complete, however, any waiting calls must be answered before ending the simulation. The timing loop has the following subfunctions:
    a. Determine whether a call was received (call simulator)
    b. Process active call
    c. Start new call
2. **Call simulator**: The call simulator will use a random number generator to determine whether a call has been received. Scale the random number to an appropriate range, such as 1 to 10. The random number should be compared with a defined constant. If the value is less than the constant, a call was received; if it is not, then no call was received. For the simulation, set the call level to %50; that is, on average, a call will be received every 2 minutes. If a call is received, place it in a queue.
3. **Process active call**: If a call is active, test whether it has been completed. In order to check if it is completed, you will again need to randomly decide whether a call is completed or not. In order to do that, you need to check the time of the service and make sure that it is always less than the given maximum service time. If completed, print the statistics for the current call and gather the necessary statistics for the end-of-job report.
4. **Start new call**: If there are no active calls, start a new call if there is one waiting in the queue. Note that starting a call must calculate the time the call has been waiting. This will show the time that the call has been waiting in the queue.

During the processing, print the data shown in the table below after each call is completed. (**Note**: you will not get the same results). It shows the call number, arrival time (the time the call was made), wait time (how long the call waited in the queue to be answered), start time (when the call was answered), service time (how long it took to answer this call) and queue size (shows the total number of calls in the queue).

Clock time	Call number	Arrival time	Wait time	Start time	Service time	Queue Size
4	1	2	0	2	3	2
6	2	3	2	5	2	4

At the end of the simulation, print out the following statistics gathered during the processing: Be sure to use appropriate format for each statistic, such as float for averages:

1. Total calls: calls received during the operating time
2. Total idle time: total time during which no calls were being serviced
3. Total wait time: sum of wait times for all calls
4. Total service time: sum of service time for all calls
5. Maximum queue size: maximum number of calls waiting during the simulation
6. Average wait time: total wait time/number of calls
7. Average service time: total service time/number of calls

Run the simulator twice. Both runs should simulate 2 hours. In the first simulation, use a maximum of service time of 2 minutes. In the second run, use a maximum of service time of 5 minutes.

Programming Requirements:

In order to implement this simulator, you need to enter the two values required to start the simulator from the command line. These values are: (1) the length time in hours that the service will be provided and (2) the maximum time it takes for the operator to take an order (the maximum service time). You need to write the following functions:

run_simulator(): This function will include the main timing loop.

call_simulator(): The call simulator will use a random number generator to determine whether a call has been received.

process_active_call(): If a call is active, test whether it has been completed. If completed, print the statistics for the current call and gather the necessary statistics for the end-of-job report.

start_new_call(): If there are no active calls, start a new call if there is one waiting in the queue.

finalise_report_simulator(): This function will mainly finish the simulator with reporting the required data (see the list above).

Queue Header: You need to create a separate header file that includes all the functionalities of a queue ADT. It is important to separate the functions of your queue ADT from the main application which is the telephone system simulator.

Submission Requirements:

Create a project under the **CNG213_Assignment_3** folder. Separate the major functionality of your Abstract Data Types into header files and put them under the **CNG213_Assignment_3/Project_Lib** folder. Project submission should be compressed version of **CNG213_Assignment_3** folder. If you do not follow this structure, you will loose %10 from the overall grade.

Programming Style Tips!

Please follow the modular programming approach. In C we use functions referred to modules to perform specific tasks that are determined/guided by the solution. Remember the following tips!

- Modules can be written and tested separately!
- Modules can be reused!
- Large projects can be developed in parallel by using modules!
- Modules can reduce the length of the program!
- Modules can also make your code more readable!

Grading:

Your program will be graded as follows:

Grading Point	**Mark (100)**
main()	5 pts
run_simulator()	35 pts
call_simulator()	10 pts
process_active_call()	10 pts
start_new_call()	10 pts
Header files (in particular a queue a header is required!)	15 pts
finalise_report_simulator(): Generating report by calling report function (calculating and displaying statistics properly)	15 pts

NOTE: Remember to have good programming style (Appropriate comments, variable names, formulation of selection statements and loops, reusability, extensibility etc.). Each of the items above will include10% for good programming style.