CS 270 Data Structures Fall 2020

Program #8 Bank Queue Simulator

Assigned: Monday Nov. 2nd Due: Monday Nov. 9th

Use a linked implementation of the Queue Abstract Data Type to simulate how a bank serves its customers. The bank maintains one Queue to line up customers who arrive at the bank. The bank has four teller service stations. If a teller has no customer, the teller will serve the next customer in the waiting Queue. Each teller will keep their own Queue of customers served. By seeing the Queue of customers served by each teller, we can analyze how efficiently services were given.

The program development will comprise two parts. In the first part, design, code, and individually test each class used by the program. In the second part, bring all the classes together to develop and test the overall bank queue simulator using the algorithm given below.

Part One – Develop and Test Individual Classes

a) Customer Class

Customer HAS-A

name – std::string gives a unique name of this customer

task – std::string gives a description of the task this customer wishes to do at the bank

timeTask – int value gives the estimated amount of time needed for a teller to do this task

timeServed – int value gives the bank clock time when this customer begins service with a teller

public methods – write descriptive pre- and post-condition header comments for each method.

```
Customer – constructor initializes strings to “none” and int values to 0.

Customer – constructor receives arguments that give values for name, task, and
timeTask

setTimeServed – receives an int value to set the value of the timeServed
attribute.  Nothing is returned.

getTimeTask() – returns the timeTask value, const method

getTimeServed() – returns the timeServed value, const method

getTask() – return the task name, const method

getName() – return the customer’s name, const method
```

Unit Test Customer Class

Write a short main test program MainCustomer.cpp that initializes one Customer and exercises each of the public methods.

b) Node Class

Node HAS-A

data – type Customer value

next – pointer to a Node object that follows this Node in a linked structure

public methods – write descriptive pre- and post-condition header comments for each method.

```
Node – constructor initializes next to NULL.

Node – constructor receives a Customer value and assigns Node’s data to be the
given value and sets next to NULL

setNext – receives a pointer to a Node and assigns Node’s next to be the given
pointer value.  No return value.

getData – return Customer data value, const method

getNext – return pointer to next Node in the linked structure, const method
```

Write a short main test program MainNode.cpp that creates a linked chain of three Node objects. Use a while loop to walk along the Node chain printing out each Node’s Customer data. See the sample code in Canvas from our second day of linked list examples.

c) Queue Class

Queue HAS-A

first – pointer to first Node in linked structure, will be NULL when Queue is empty

last – pointer to last Node in linked structure, will be NULL when Queue is empty

public methods – write descriptive pre- and post-condition header comments for each method.

Queue – initializes contents to empty

insert – receives a Customer object, if the Queue is not full, then appends this Customer to the end and returns true; else, return false.

remove – if the Queue is not empty, remove and return the Customer at the front; else, return a Customer object whose attribute values denote an undefined/invalid Customer.

isEmpty – return true if the Queue is empty; else, false. const method.

isFull() – return true if the Queue is fall; else, false. const method.

** Implement your Queue.cpp using the Node object to contain a Customer data.*

Write a short main test program MainQueue.cpp that inserts three Customers into the Queue. Use a while loop continue removing Customers as long as the Queue is not empty. Print each Customer removed to confirm correct behavior.

Part Two – Main Program Bank Simulation

```
Let waitingCustomers be a Queue of Customers, initially empty

Fill waitingCustomers Queue with Customers, each having a unique name, desired banking task, and estimated time for a teller to service that task

Let tellerCustomers be an array of Queue one per teller.  When a teller begins serving a Customer that Customer will be removed from waitingCustomers Queue and inserted into one of these teller Queues.

Let tellerWork be an array of integers one per teller.  Each value represents the remaining time needed for each teller to finish servicing their current Customer’s task

Let clockTime be an integer value that represents the current simulation time, initially 0

While waitingCustomers is not empty and tellerWork is such that at least one teller has work left do:

    Let t be 0

    while t less than number of tellers do:

        if tellerWork for teller #t is equal to 0 and waitingCustomers is not empty then

            Let C be next Customer removed from waitingCustomers Queue

            Set tellerWork for teller #t to be C’s estimated task time

            Set C’s time served to be clockTime

            Insert C into tellerCustomer Queue for teller #t

        Decrement tellerWork by 1 for teller #t

        Increment t by 1 to go to the next teller

    Increment clockTime by 1


    Let t be 0
    while t less than number of tellers do:
        while tellerCustomers Queue for teller #t is not empty
            Let C be Customer removed from tellerCustomers Queue for teller #t
            print information about Customer serviced by teller #t
        Increment teller #t by 1
```

Sample Simulation Run #1

All 20 customers wish to make a deposit that takes 1 unit of time.

Line of Customers

A requests deposit takes 1

B requests deposit takes 1

C requests deposit takes 1

D requests deposit takes 1

E requests deposit takes 1

F requests deposit takes 1

G requests deposit takes 1

H requests deposit takes 1

I requests deposit takes 1

J requests deposit takes 1

K requests deposit takes 1

L requests deposit takes 1

M requests deposit takes 1

N requests deposit takes 1

O requests deposit takes 1

P requests deposit takes 1

Q requests deposit takes 1

R requests deposit takes 1

S requests deposit takes 1

T requests deposit takes 1

Report results by showing Customers served by each teller. The numbers represent the clockTime at which a Customer begins to receive service from that teller.

Teller #0 Served Customers

0: A did deposit

1: E did deposit

2: I did deposit

3: M did deposit

4: Q did deposit

Teller #1 Served Customers

0: B did deposit

1: F did deposit

2: J did deposit

3: N did deposit

4: R did deposit

Teller #2 Served Customers

0: C did deposit

1: G did deposit

2: K did deposit

3: O did deposit

4: S did deposit

Teller #3 Served Customers

0: D did deposit

1: H did deposit

2: L did deposit

3: P did deposit

4: T did deposit

The first simulation run resulted in each teller servicing the same number of customers in lock step timing since all customers needed the same amount of service time.

Sample Simulation Run #2

Customers appear in runs of 4 distinct tasks, each of which takes a longer amount of time.

Line of Customers

A requests deposit takes 1

B requests withdraw takes 2

C requests new account takes 4

D requests loan application takes 8

E requests deposit takes 1

F requests withdraw takes 2

G requests new account takes 4

H requests loan application takes 8

I requests deposit takes 1

J requests withdraw takes 2

K requests new account takes 4

L requests loan application takes 8

M requests deposit takes 1

N requests withdraw takes 2

O requests new account takes 4

P requests loan application takes 8

Q requests deposit takes 1

R requests withdraw takes 2

S requests new account takes 4

T requests loan application takes 8

Teller #0 Served Customers

0: A did deposit

1: E did deposit

2: F did withdraw

4: H did loan application

12: Q did deposit

13: R did withdraw

15: T did loan application

Teller #1 Served Customers

0: B did withdraw

2: G did new account

6: K did new account

10: O did new account

14: S did new account

Teller #2 Served Customers

0: C did new account

4: I did deposit

5: J did withdraw

7: L did loan application

Teller #3 Served Customers

0: D did loan application

8: M did deposit

9: N did withdraw

11: P did loan application

Grading Rubric

Software Development Steps	**Excellent (3)**	**Satisfactory (2)**	**Re-do (1)**
Design and Coding of Customer, Node, and Queue Classes	All class data members and methods are correctly defined with descriptive pre-	At least 70% of the data members and methods are correctly defined with descriptive pre- and post-condition header comments	Does not satisfy excellent nor satisfactory

	and post-condition header comments		
Unit-test main program that demonstrates successful behavior of Customer, Node, and Queue classes	Provides a separate main.cpp file that creates at least one object of its class type and applies all of its methods with descriptive print outs. All main files compile and run.	Provides a separate main.cpp file that creates at least one object of its class type and applies all of its methods with descriptive print outs. All main files can be made to compile and run with a moderate number of edits (approximately no more than 30% of the source code of each)	Does not satisfy excellent nor satisfactory
Main Bank Simulation Program	C++ code correctly imports needed classes, accurately realizes the bank simulation algorithm given above, compiles, and runs being able to produce output that matches that of the two sample test cases	C++ code mostly realizes the bank simulation algorithm given above, but has some typos/syntax errors or logic errors that cause the code to not compile and run or produce inaccurate results. Code in need of improvement comprises no more than approximately 30% of the source code.	Does not satisfy excellent nor satisfactory
Demonstrates output of simulation program	Includes screen shots to document at least two successful simulation runs	At least 70% of the test case outputs matched the expected correct outputs	Does not provide screen shots or documented results

The overall rating is computed by averaging the ratings for each of the four evaluation criteria. Average ratings >= 2.5 are excellent, >= 1.5 satisfactory, and < 1.5 re-do.

Submit your work by uploading the following items:

Customer.h   Customer.cpp   Node.h   Node.cpp   Queue.h   Queue.cpp

Unit-testing demonstration main programs (one per class – Customer, Node, and Queue)

MainCustomer.cpp   MainNode.cpp   MainQueue.cpp

BankSim.cpp (your main simulation program file)

Test Run 1 screen shot image

Test Run 2 screen shot image