**Project 7:   
Queueing Simulation**

**Design Document:**

Introduction:

A Queuing simulation is a theoretical study of waiting lines, expressed in mathematical terms--including components such as number of waiting lines, number of servers, average wait time, probabilities of teller’s transaction times, the probability a customer will arrive or enter a line, the maximum duration of a transaction, and duration of a specific simulation based on the previous inputs described. So, in this program, we will simulate this queueing system which is a representation of a waiting line at a store or in a place with a similar concept of waiting lines. Such that each queue (line) has one server. Therefore, the program will input from the user some numbers for the simulation which are: number of queue/server pairs, the probability that a customer will arrive in a single tick, the longest time a transaction may take, the duration of the simulation in ticks, and a integer seed value for the random number generator. Then, a snapshot of the queues content and tellers’ transaction times are printed to the terminal. Finally, some statics are reported back to the terminal.

Data Structures:

Only one data structure is used for this program which is a Queue. The Queue we are using is a linked Queue (pointer-based). This queue holds the customers' arrival times to the queue or line. In other words, it holds the customers. An array of queue objects will be used in this program. Such that multiple lines or queues can be simulated. In this data structure customers are dequeued or enqueued under specific conditions described later.

Functions:

This program uses 7 functions that belong to the queue class and 3 functions that belong to the main program. We will begin by describing the queue class functions. The constructor “Queue()” just initializes the front node and the rear node of the queue to point to NULL and count (length of the queue) to be zero. The destructor “~Queue();” loops until the end of the linked list attached to the front node of the queue and deletes/deallocates nodes. The enqueue function of the queue “void enqueue(item entry);” adds a new node with an integer entry to the rear of the queue. This is done by allocating a temporary node with the entry integer and a NULL pointer. If the queue is empty, front will point to the new node. Otherwise, the last node in the queue will point to the new node. Also, the rear node will point to the new node in either cases. The dequeue function of the queue “item dequeue();” removes an integer entry from the front of the queue. This is done by getting the data in the first node in the queue, making a temp node point to where the front is pointing, advancing front, and then deleting the temp node. If the queue is empty after the previous process, then the rear should point to NULL. After that, the length of the queue is decreased and the dequeued integer is returned. The size of the queue function “int size()” just returns the length of the queue which is count. The “bool empty()” just checks if the Queue is empty or not. If count is zero, then the function returns true otherwise the function returns false. The “friend ostream& operator << (ostream& out\_s,Queue& q);” declares a node to point to the first element of the list. Then, the function loops till the end of the queue, prints the data of the nodes, advances the declared node to get the data of the next nodes. Also, the function returns the stream. Now let us discuss the none queue functions. The “int shortest(Queue q1[],int length, int teller1[])” returns the index that has the shortest queue or line. If the queues have the same length the function returns the index where a free teller is found. So, first, this function loops until the length of queues and finds which queue has the shortest size by using the queue size function. Then, a second loop in the function checks if there is a line that has a free teller and a queue size of zero and returns that index. Otherwise, the second loop checks if there is a line that has a free teller and a queue size that is equal to the shortest computed queue size but not equal to zero. Then, the function returns the index of such condition. Finally, if the function did not return anything in the second loop, then the index of the shortest computed queue in the first loop is returned. The “int tellerfree( int teller2[],Queue q2[],int length)” loops along the number of tellers and checks which teller is free and has a corresponding not empty queue. If this condition is true, the function returns the index where this condition is found. Otherwise, the function returns -10 which means the condition is not true. The “int sum\_length(int length, Queue q4[])” loops along the number of queues and accumulates the sum of sizes of each available queue, and returns the sum.

The Main Program:

The main program declares a couple of integer variables that will be used to save user input in or will be used in the simulation. The program declares two arrays after the input of the number of queues. One array is an array of objects of the queue class and the other is an array of tellers. So, each queue has only one teller. The program initializes the tellers to be free or contain 0 transaction times. Then, the program loops until the length of the duration inputted from the user. In each loop based on probability, a customer arrival time is enqueued in the shortest queue by using the shortest function. Then, a loop that goes until the number of queues checks if there is a free teller that corresponds to a non-empty queue. If the condition is true which means that the index is not (-10), then the customer is dequeued. After that, the customer wait sum is computed and accumulated to an integer variable and an integer array. Then, the count is incremented which means that a customer is served. Then, a random transaction time is assigned to the teller the customer was served at. After those processes are described, a loop goes over the number of tellers and reduces the transaction times of each teller transaction times except a time of zero. Finally, a snapshot of the queues, the tellers, and the time interval are printed to the terminal. After the simulation is ended which means that the loop exhausted the duration inputted, some statics are printed to the terminal. Those static represents the number of customers served, the average wait sum, the longest time a customer waited before being served (which is found by finding the maximum wait sum in the array of wait sums), and the number of customers who remain in lines (which is computed by the sum\_length() function).

**Code:**

***The main program:***

/\*

Purpose and design: The purpose of the program is to simulate a queueing system. So, this simulation will be used to gather some statics. The program reads the number of queue/server pairs, the probability that a customer will arrive in a single tick, the longest time a transaction may take, the duration of the simulation in ticks, and a integer seed value for the random number generator. Then, the program will simulate the queueing system for the specified time and with the specified inputs. Also, during each tick some events occur. Like a customer is dequeued when a teller is free, and a customer is enqueued to the shortest line. Moreover, transaction times are decreased after each simulation tick. Finally, the program reports some statics. Those statics include: the number of customers served, the probability that a customer will arrive in a single tick, the average wait times of customers served, the longest time a customer waited, and the number of customers left in the queues when the simulation stops.

\*/

// File: project7.cpp

#include<iostream>

#include "Queue.h" // include queue class

#include<cstdlib>

#include<iomanip>

#include<bits/stdc++.h>

using namespace std;

// A function that examines the lengths of the queues and the remaining transactions.

// Precondition: A queue object, the number of the queues or tellers, and the tellers’ array

// Postcondition: correct index is returned to where a customer should enter

int shortest(Queue q1[],int length, int teller1[])

{

int min=0;

int size=0;

for(int x=1;x<length;x++) // loop until end of queues

{

if(q1[min].size()>q1[x].size())

{

min=x; // compute the minimum queue size

size=q1[x].size(); // save minimum size of the queue

}

}

for(int x=0;x<length;x++) // loop again until end of queues

{

if(teller1[x]==0 && q1[x].size()==0) // if teller is free and queue empty then return index

return x;

else if(teller1[x]==0 && q1[x].size() == size && size!=0) // if teller is free, size of a queues is equal to the shortest size, and the size is not zero then return index

return x;

}

return min; // if second loop did not return anything then return the shortest queue length

}

// A function that determines which queue is not empty and has a free server

// Precondition: A queue object, an integer number of the queues or tellers inputted, and an integer tellers’ array

// Postcondition: A correct index is returned to where a customer should be served

int tellerfree( int teller2[],Queue q2[],int length)

{

for(int x=0;x<length;x++) // loop until end of queues

{

if((teller2[x]==0) && (!q2[x].empty())) // if teller free and queue is not empty

return x; // return index

}

return -10; // if nothing is returned previously then return -10

}

// A function that determines the total length of queues

// Precondition: A queue object and an integer number of the queues or tellers inputted

// Postcondition: A sum of queues total lengths is returned

int sum\_length(int length, Queue q4[])

{

int sum=0;

for(int x=0;x<length;x++) // loop until end of queues

{

sum+=q4[x].size(); // Accumulate sum

}

return sum;

}

int main()

{

int QueNum=0;

int Prob=0;

int Trans\_Duration=0;

int Simulation\_Duration=0;

int seed=0;

int trans\_time=0;

int count=0;

int entry\_time=0;

float wait\_sum=0.0;

// get user input for the simulation

cout<<"Enter these parameters of the simulation:"<<endl;

cout<<" The number of queue/server pairs: ";

cin>>QueNum;

cout<<" The probability that a customer arrives in one tick (%): ";

cin>>Prob;

cout<<" The maximum duration of a transaction in ticks: ";

cin>>Trans\_Duration;

cout<<" The duration of the simulation in ticks: ";

cin>>Simulation\_Duration;

cout<<"Enter a random number seed:";

cin>>seed;

srand(seed);

count=0;

Queue q[QueNum]; // declare array of queue objects

int teller[QueNum]; // declare array of tellers

for(int x=0;x<QueNum;x++) // initialize tellers to be free

{

teller[x]=0;

}

int y; // hold shortest queue index

int point=-10;

int time12[700]; // holds all wait sums

for(int time=1;time<Simulation\_Duration+1;++time) // simulate until the length of duration

{

if(rand()%100 < Prob) // if customer arrived

{

y=shortest(q,QueNum,teller); // find shortest index

q[y].enqueue(time); // add the customer to shortest queue

}

for(int x=0;x<QueNum;x++) // loop over the number of queues

{

point=tellerfree(teller,q,QueNum); // if teller free and queue not empty point will hold index

if(point!=-10) // if index returned not -10 then dequeue customer

{

entry\_time=q[point].dequeue(); // dequeue

wait\_sum+=(time-entry\_time); // accumulate wait sums

time12[count]=(time-entry\_time); // accumulate wait sums in an array

++count; // increment number of customers served

trans\_time=(rand()%Trans\_Duration)+1; // random transaction time for new customer

trans\_time+=1; // beacuse the new added teller will be decremented

teller[point]=trans\_time;

}

}

for(int u=0;u<QueNum;u++) // loop over the number of tellers

{

if(teller[u]!=0) // if teller is not free

--teller[u]; // decrement transaction time

}

// formatting snapshot of simulation

cout<<time;

cout<<endl;

for(int x=0;x<QueNum;x++)

{

cout<<setw(9)<<teller[x]<<" "<<q[x];

cout<<endl;

}

cout<<endl;

}

//printing out statics

// number of customers served and wait time average

float avg=0.0;

avg=wait\_sum/count;

cout<<count<<" customers waited an average of "<<avg<<" ticks."<<endl;

// get the longest time a customer waited by getting max number of array of wait sums

cout<<"The longest time a customer waited was "<<\*max\_element (time12, time12+count)<<" ticks."<<endl;

// get the number of customers that are not served yet

cout<<sum\_length(QueNum,q)<<" customers remain in the lines."<<endl;

return EXIT\_SUCCESS;

}

***The Queue class:***

#ifndef QUEUE\_H

#define QUEUE\_H

#include<iostream>

using namespace std;

class Queue

{

public:

typedef int item; // item of type integer

Queue(){front=NULL;rear=NULL;count=0;} // Constructor with front and rear point to NULL and count is zero

~Queue(); // Destructor that removes all nodes used

void enqueue(item entry); // Adds an integer to the last of the queue

item dequeue(); // removes an integer from the first of the queue

int size(){return count;} // return count which is the size of the queue

bool empty(){return count==0;} // checks if the queue is empty by checking count

friend ostream& operator << (ostream& out\_s,Queue& q); // Overloads the inserter operator

private:

struct Node // Node declaration

{

item data; // Data field

Node \*next; // next pointer field

};

Node \*front; // Front node pointer deceleration

Node \*rear; // rear node pointer deceleration

int count; // count declaration

};

// A destructor that deallocates the memory used by the Queue or delete nodes

// Postcondition: Nodes of the Queue will be deleted

Queue::~Queue()

{

Node\* temp;

while(front!=NULL) // While the invoking list is not at the last element or empty

{

temp=front; // temp points to first

front=front->next; // Advance first

delete temp; // delete the node

}

}

// A function that adds an entry integer to the rear of the Queue

// Precondition: An integer number

// Postcondition: number is added to the end of the Queue

void Queue::enqueue(item entry)

{

Node \*temp; // Temporary node

temp=new Node; // Allocate memory

temp->data=entry; // Place entry in node data field

temp->next=NULL; // Place NULL in node pointer field

if(front==NULL) // if queue empty

front=temp; // front points to temporary node

else

rear->next=temp; // the last node in queue point to the temporary node

rear=temp; // rear node points to temporary node

++count; // increment length

}

// A function that removes an entry integer from the front of the Queue

// Precondition: A queue object that is not empty

// Postcondition: A number is removed from the top of the queue

Queue::item Queue::dequeue()

{

Node \*temp; // Temporary node

item dequeued=front->data; // get first node data

temp=front; // temp points to top

front=front->next; // advance front

delete temp; // delete temp

if(front==NULL) // if queue empty

rear=NULL; // rear is empty as well

--count; // decrement length

return dequeued; // give back first removed element

}

// A friend function that prints the Queue content

// Precondition: A ostream operator and a Queue object

// Postcondition: Returns the stream and allows the printing of the Queue with inserter operator "<<"

ostream& operator << (ostream& out\_s,Queue& q)

{

Queue::Node \*p;

p=q.front; // p points to first

while(p!=NULL) // while the invoking list is not at the last element or empty

{

out\_s<<p->data<<' '; //print data

p=p->next; //advance p

}

return out\_s; //return stream

}

#endif

**User Document:**

The program name is (project7.cpp).

An Example of running the program is represented below:

The program reads an inputted requirements for the simulation from the user to start the simulation. The number of queues must an integer at least 1 or greater. The probability is a number from 1 to 100. The duration of the transition can be any integer number greater than or equal to zero. The duration of the simulation must be at least 1. The seed can be any integer number.

**Exchange of the following inputs triggers the program to start the simulation:**

Enter these parameters of the simulation:

The number of queue/server pairs: 2

The probability that a customer arrives in one tick (%): 90

The maximum duration of a transaction in ticks: 10

The duration of the simulation in ticks: 10

Enter a random number seed:2

**This what is outputted when the program terminates:**

**The (1) in this example represent the time interval of the simulation like from (1-10)**

1

7 **Those two numbers represent the tellers**

0

2

6

0

3

5 **This box represents a snapshot of the program at the tick (3)**

0

. . . . . .

8

0 7

4 8

9

3 9 **Those two numbers represents the customers arrival time in the queue**

3 8

10

2 9 10

2 8

**Those represents the statics of simulation:**

4 customers waited an average of 0.5 ticks.

The longest time a customer waited was 2 ticks.

3 customers remain in the lines.

**Tests:**

**The following is a Script of the tests copied from my terminal of (Visual Code Studio) and pasted in this Word document.**

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

**This tests if the program works correctly if 1 queue is used with a low probability of customers entering and with only 2 ticks of maximum duration. The program provides the duration from 1 to 100 but based on the requirements we should not provide the whole simulation.**

PS C:\Users\Khaled\Desktop\c+++> cd "c:\Users\Khaled\Desktop\csci 301\project 7\" ; if ($?) { g++ project7.cpp -o project7 } ; if ($?) { .\project7 }

Enter these parameters of the simulation:

The number of queue/server pairs: 1

The probability that a customer arrives in one tick (%): 20

The maximum duration of a transaction in ticks: 2

The duration of the simulation in ticks: 100

Enter a random number seed:3

1

0

2

0

3

0

..........

94

1 94

95

0 94

96

1

97

0

98

0

99

0

100

0

17 customers waited an average of 0.411765 ticks.

The longest time a customer waited was 3 ticks.

0 customers remain in the lines.

**This tests if the program works correctly if 4 queues are used with the same input of the example of the assignment.**

PS C:\Users\Khaled\Desktop\c+++> cd "c:\Users\Khaled\Desktop\csci 301\project 7\" ; if ($?) { g++ project7.cpp -o project7 } ; if ($?) { .\project7 }

Enter these parameters of the simulation:

The number of queue/server pairs: 4

The probability that a customer arrives in one tick (%): 80

The maximum duration of a transaction in ticks: 12

The duration of the simulation in ticks: 120

Enter a random number seed:3

1

9

0

0

0

2

8

0

0

0

3

7

0

0

0

4

6

10

0

0

5

5

9

8

0

..........

118

8 84 87 89 94 98 104 108 111

0 81 85 91 93 96 100 105 109 117

3 83 90 92 95 101 106 110 112 114

2 80 86 88 97 99 102 107 116 118

119

7 84 87 89 94 98 104 108 111

10 85 91 93 96 100 105 109 117

2 83 90 92 95 101 106 110 112 114

1 80 86 88 97 99 102 107 116 118

120

6 84 87 89 94 98 104 108 111

9 85 91 93 96 100 105 109 117

1 83 90 92 95 101 106 110 112 114

0 80 86 88 97 99 102 107 116 118

64 customers waited an average of 18.2812 ticks.

The longest time a customer waited was 42 ticks.

34 customers remain in the lines.

**This tests if the program works correctly if lots of queues are used**

PS C:\Users\Khaled\Desktop\c+++> cd "c:\Users\Khaled\Desktop\csci 301\project 7\" ; if ($?) { g++ project7.cpp -o project7 } ; if ($?) { .\project7 }

Enter these parameters of the simulation:

The number of queue/server pairs: 8

The probability that a customer arrives in one tick (%): 99

The maximum duration of a transaction in ticks: 13

The duration of the simulation in ticks: 50

Enter a random number seed:8

1

13

0

0

0

0

0

0

0

2

12

8

0

0

0

0

0

0

3

11

7

5

0

0

0

0

0

..........

48

5 47

1 48

0 38

7

5

4

3

7

49

4 47

0 48

10

6 49

4

3

2

6

50

3 47

2

9 50

5 49

3

2

1

5

47 customers waited an average of 1.93617 ticks.

The longest time a customer waited was 12 ticks.

3 customers remain in the lines.

**This tests if the program works and calculates statics correctly if the probability that a customer will arrive is 100%**

PS C:\Users\Khaled\Desktop\c+++> cd "c:\Users\Khaled\Desktop\csci 301\project 7\" ; if ($?) { g++ project7.cpp -o project7 } ; if ($?) { .\project7 }

Enter these parameters of the simulation:

The number of queue/server pairs: 4

The probability that a customer arrives in one tick (%): 100

The maximum duration of a transaction in ticks: 10

The duration of the simulation in ticks: 130

Enter a random number seed:4

1

6

0

0

0

2

5

7

0

0

3

4

6

5

0

..........

128

2 78 82 83 93 95 96 100 105 110 115 121 122

5 88 94 99 101 103 108 112 116 118 123 126 128

3 80 86 90 91 97 102 107 109 114 119 124

9 89 92 98 104 106 111 113 117 120 125 127

129

1 78 82 83 93 95 96 100 105 110 115 121 122

4 88 94 99 101 103 108 112 116 118 123 126 128

2 80 86 90 91 97 102 107 109 114 119 124 129

8 89 92 98 104 106 111 113 117 120 125 127

130

0 78 82 83 93 95 96 100 105 110 115 121 122

3 88 94 99 101 103 108 112 116 118 123 126 128

1 80 86 90 91 97 102 107 109 114 119 124 129

7 89 92 98 104 106 111 113 117 120 125 127 130

82 customers waited an average of 19.9024 ticks.

The longest time a customer waited was 46 ticks.

48 customers remain in the lines.

**This tests if the program works and calculates statics correctly if the maximum duration is so big.**

PS C:\Users\Khaled\Desktop\c+++> cd "c:\Users\Khaled\Desktop\csci 301\project 7\" ; if ($?) { g++ project7.cpp -o project7 } ; if ($?) { .\project7 }

Enter these parameters of the simulation:

The number of queue/server pairs: 6

The probability that a customer arrives in one tick (%): 95

The maximum duration of a transaction in ticks: 90

The duration of the simulation in ticks: 120

Enter a random number seed:4

1

36

0

0

0

0

0

2

35

27

0

0

0

0

3

34

26

55

0

0

0

..........

118

1 13 20 28 35 39 43 49 56 65 72 78 84 91 97 105 111 117

19 29 31 36 40 44 50 57 66 73 79 85 92 100 106 112 118

17 15 22 30 37 45 51 58 60 67 74 80 86 93 101 107 113

7 23 32 38 46 53 61 63 68 75 81 88 94 98 102 108 114

14 17 24 33 41 47 54 62 70 76 82 87 89 95 103 109 115

36 25 27 34 42 48 55 64 71 77 83 90 96 99 104 110 116

119

0 13 20 28 35 39 43 49 56 65 72 78 84 91 97 105 111 117

18 29 31 36 40 44 50 57 66 73 79 85 92 100 106 112 118 119

16 15 22 30 37 45 51 58 60 67 74 80 86 93 101 107 113

6 23 32 38 46 53 61 63 68 75 81 88 94 98 102 108 114

13 17 24 33 41 47 54 62 70 76 82 87 89 95 103 109 115

35 25 27 34 42 48 55 64 71 77 83 90 96 99 104 110 116

120

74 20 28 35 39 43 49 56 65 72 78 84 91 97 105 111 117

17 29 31 36 40 44 50 57 66 73 79 85 92 100 106 112 118 119

15 15 22 30 37 45 51 58 60 67 74 80 86 93 101 107 113 120

5 23 32 38 46 53 61 63 68 75 81 88 94 98 102 108 114

12 17 24 33 41 47 54 62 70 76 82 87 89 95 103 109 115

34 25 27 34 42 48 55 64 71 77 83 90 96 99 104 110 116

17 customers waited an average of 37.1765 ticks.

The longest time a customer waited was 107 ticks.

98 customers remain in the lines.

**Summary:**

In this program, I learned how to implement a pointer-based Queue class which was useful for simulating a queuing system. Moreover, I learned and understood how a queueing simulation works and how dequeuing and enqueueing customers depend on certain conditions. Moreover, those conditions somewhat represent what happens for example in boarding lines in an airport. The interesting thing is that I learned how to determine the shortest queue or the one with a free teller, so the customer joins the best queue. To extend the solution, I might graph the statics rather than providing just numbers. I would like to improve the formatting of the snapshot also. I wanted to make the ticks of the simulation on the same line with the queues and the tellers, however, I found it not working.

**Question**: “If the probability that a customer arrives during one tick is 0.50, and the maximum transaction time is four ticks, what is the smallest number of lines and tellers the system can have and usually keep the average time customers wait below four ticks?”

**Answer**: The smallest number of lines and tellers the system can have and usually keep the average time customers wait below four ticks **is 2 lines and 2 tellers**. I arrived at that conclusion by entering a probability of 50% for customers entering and 4 ticks for maximum transaction times. Then, I entered (1) for the number of queues/tellers and found that the average waiting time will be more than 4 ticks. Then, I entered (2) for the number of queues/tellers and found that the average waiting time is less than 4 ticks. Therefore, that means the smallest number of lines and tellers is (2). Because the least number of tellers and lines the simulation should have is (1) so (2) would be the answer.