**Sandesh Ghimire**

**CSCI 301 section 3**

**Computer Science 2**

**Project 6**

**18th October**

**Introduction**

We implemented Queue (ADT) which works on the basis of first in first out. This project contains three files, " *queue.h* “, " *queue.cpp* " and " *main.cpp* “. This program creates probabilistic simulation of a queueing system with several queues and several servers, one server for each queue. There will be several lines so people can join the line that is the shortest. The program starts by asking the number of servers, longest time for transaction, seed value, and probability of arriving.

**Data Structures:**

We used Queue as a data structure which is an Abstract data type. We declared the Queue using an array. The stack has two major function i.e. *enqueue()* and *dequeue().* Enqueue function enters the integer inside the array whereas Dequque function takes the integer out and returns it. Queue uses two pointers(not the data structure) to point the front and end of the queue. According to the enqueue and dequeue the pointer is increased and decreased to keep track of the queue.

Sequential array is also the data structure here. It is used to create the queue and store the data.

**Functions:**

Queue(): It is to initialize the queue.(constructor)

*void enqueue(int entry):* This function takes integer as a parameter. It stores that parameter to the array and then increases the value of *count* by one.

*int dequeue():* This function doesn’t take any parameter. It just decreases the value of *count* by one and returns the current value inside array.

*int Size():* It returns the current used length of array used.

*bool empty():* It checks weather the queue is empty or not.

*friend function:* It is used to overload the << operator.

**The Main Program:**

The main function asks for several inputs by displaying the messages. Firstly, it asks for the number of server, average probability, maximum transaction time, seed, and duration of ticks. The program declares the array of Queue called line[int].

After getting all this inputs a loop is started from 1 to the total duration entered by user. Inside the loop the program checks for the shortest line. After checking the line, the program checks if the probability of arrival is less then the provided percentage of not. If it satisfies the time is enqueued into the queue. Now, the program checks for the transaction time. If it is 0 then the program again goes further checking if the Queue is empty or not. If not, the program puts the value or time to the *entry\_time* variable, increases the value of *counts,* and new value to *trans\_time*. Lastly, the program outputs the lines and their values.

**Code**

**queue.cpp**

#include <iostream>

#include "queue.h"

using namespace std;

Queue::Queue(){ //constructor

front=0;

rear=CAPACITY-1;

count=0;

}

void Queue::enqueue(int entry){//inserts the value into queue

rear=next\_index(rear);

data[rear]=entry;

++count;

}

Queue::Item Queue::dequeue()//removes the item from queue

{

Item temp;

temp=data[front];

front=next\_index(front);

--count;

return temp;

}

ostream& operator <<(ostream& out\_s, Queue& q){//friend function

Queue::Item it;

int n;

n=q.Size();

for(int i=0;i<n;++i){

it=q.dequeue();

out\_s<<it<<" ";

q.enqueue(it);

}

return out\_s;

}

**queue.h**

#include <iostream>

#ifndef QUEUE\_H

#define QUEUE\_H

using namespace std;

class Queue{

public://public functions of the class

typedef int Item; //defining the Item using int

//Item first;

static const int CAPACITY=30;

Queue();

void enqueue(int entry);//to make list empty

int dequeue();//to insert the data in the list

//Member functions

int Size(){return count;};//returns the size of queue

bool empty(){return count==0;}//returns if queue is empty or not

friend std::ostream& operator << (std::ostream& out\_s, Queue& Queue1); //this is a friend function

private://private functions of class

Item data[CAPACITY];

int front;//points front of queue

int rear;//points rear of queue

int count; //counts the number of data in queue

int next\_index(int i){ //Private member function

return (i+1)%CAPACITY;}

};

#endif // QUEUE\_H

**main.cpp**

#include <iostream>

#include<iomanip>

#include<cstdlib>

#include "queue.h"

#include "queue.cpp"

#include <cmath>

using namespace std;

int main ()

{

int ARV\_PROB; //variable for probability

int MAX\_TRANS\_TIME;//maximum transaction time

int DURATION;//duration of the simulation

int server;//number of servers

//asking input to the users

cout<<setw(4)<<"----Enter these parameters of the simulation:----"<<endl;

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

cin>>server;

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

cin>>ARV\_PROB;

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

cin>>MAX\_TRANS\_TIME;

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

cin>>DURATION;

Queue line[server];//declare the array of queue

int trans\_time[server];//array of transaction time

for(int temp=0; temp<server; ++temp)//starting loop for initializing trans\_time

{

trans\_time[temp] = 0;

}

//declaring some variables and initializing them

int count = 0;

int time\_e;

int wait\_sum = 0;

int Count = 0;

int seed;

int Remaining = 0;

int High = 0;

//asking seed to the user

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

cin>>seed;

cout<<endl;

srand(seed);//randomizing the seed

for(int time=1; time<DURATION; ++time)//starting the big loop

{

int temp =0;

if(rand()%100 < ARV\_PROB)//checking if the value is less then the original probability

{

int index = 1;

while (index < server)//looping to select the available queue and storing data

{

if(line[temp].Size() > line[index].Size()) //checking which queue is short

{

temp = index;

}

++index;//increasing the value of index

}

line[temp].enqueue(time);//value goes into queue

}

for (int i = 0;i<server; ++i)//starting loop

{

if(trans\_time[i] == 0)//check is transcation time is 0 or not

{

if(!line[temp].empty())//checking if the queue is empty or not

{

time\_e = line[temp].dequeue();//put the value in time\_e by dequeuing the data

wait\_sum += time - time\_e;//finding out the wait time

Count = time - time\_e;//

if(Count>High)//check if count is greater then high

High = Count;

++count;

trans\_time[i] = rand() % MAX\_TRANS\_TIME + 1;//randomizing the trans time again

}

}

else//if the case is not satisfied

--trans\_time[i];//decrease the value of trans time

cout<<endl;

}

cout<<time<<"::";//output the time

for(int j = 0; j<server; ++j)//loop to display the queue

{

cout<<setw(4)<<trans\_time[j]<<setw(4)<<line[j]<<endl;

}

}

for(int i = 0; i<server; ++i)//loop to find out the remaining line

{

Remaining = Remaining + line[i].Size();

}

cout<<endl;

//printing the outputs

cout<<count<<" customers waited an average of "<<wait\_sum/count<<" ticks."<<endl;

cout<<"The longest time a customer waited was "<<High<<" ticks."<<endl;

cout<<Remaining<<" customers remain in the lines. "<<endl;

return 0;

}

**User Document**

To compile the program simply enter:

*g++ -o main main.cpp*

To run the program, enter. */main,* then you can input an integer. After entering an integer, the program will display the prime factors of the entered number.

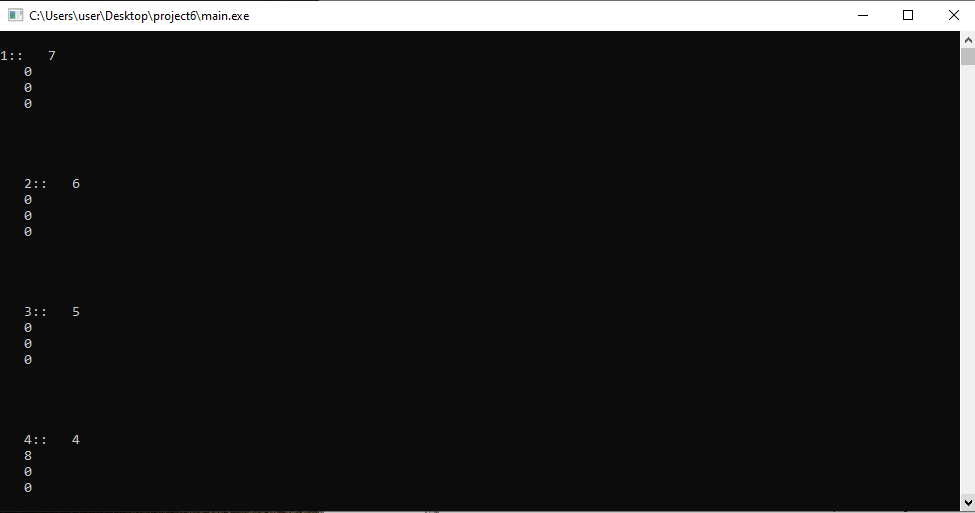
For example, when you run the program it asks input:

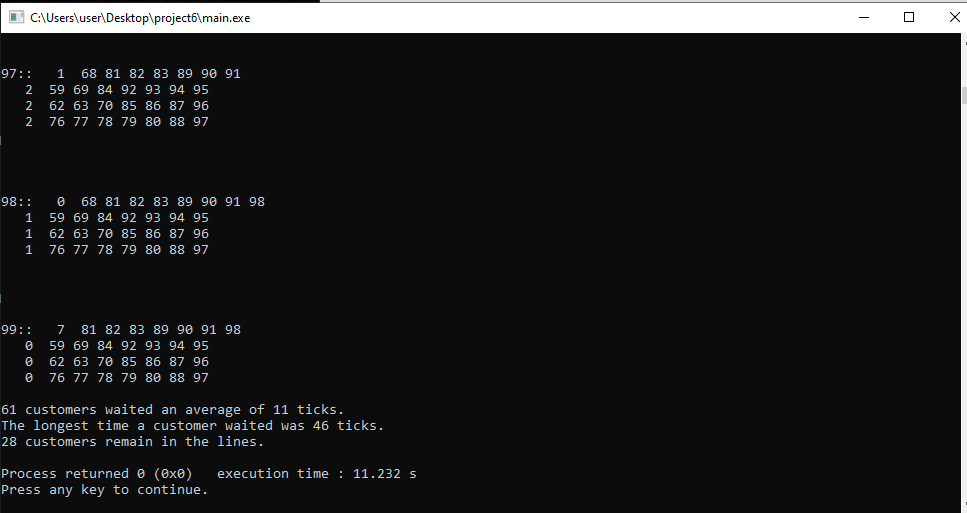
*Prompt>./main*

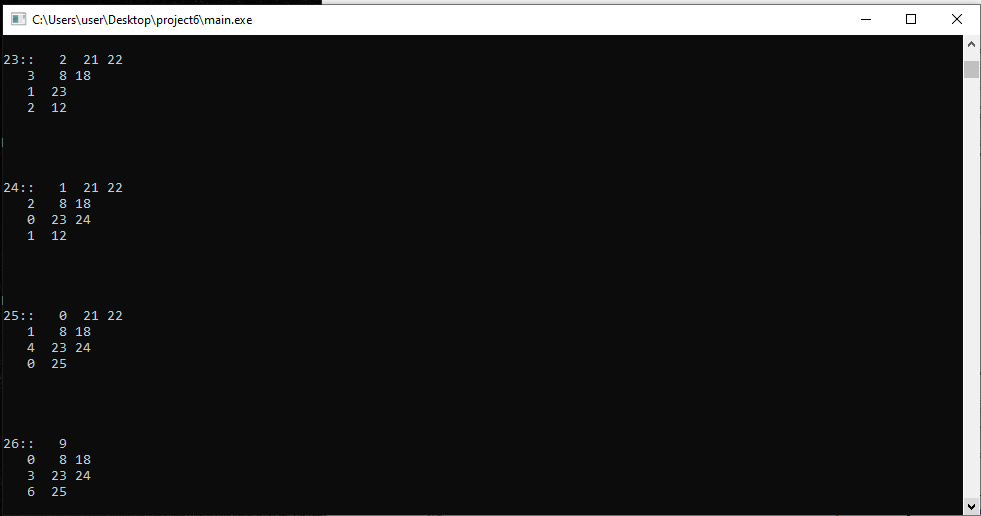
*Enter these parameters of the simulation: 4*

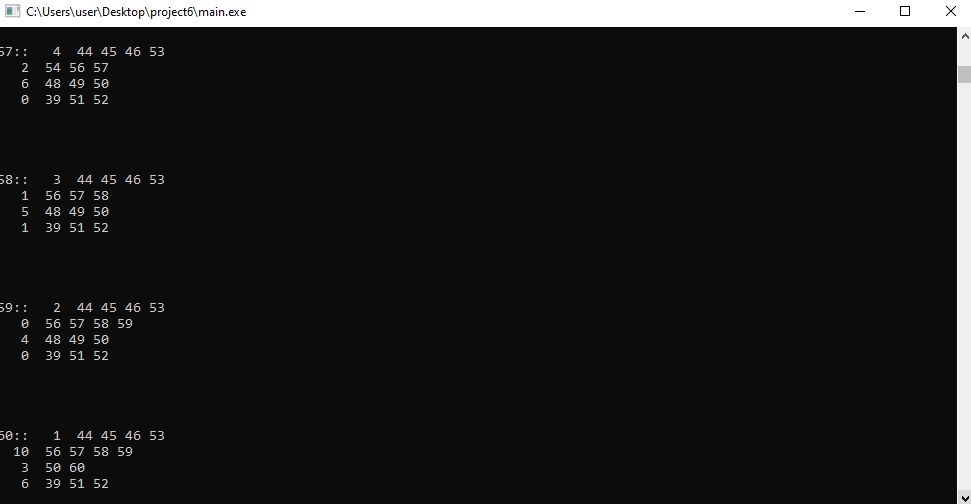
**Testing**

TEST1









**Summary**

In this project we implemented the ADT i.e. queue to do probabilistic simulation of a queueing system with several queues and several servers, one server for each queue, as in a grocery or discount store. This project was perfect implementation of queue in our real world. We constructed a class called queue that can enqueue and dequeue the data. The problem got more challenging when we had to use multiple servers because we had to declare queue more then one time. But the best way to do it was to declare an array of Queue itself. We could use linked list to build queue instead of array. This would be more efficient because we don’t have to care about the size of the list.