**Call.h**

#ifndef call

#define call

#include <iostream>

#include <string>

using namespace std;

class Call{

public:

Call(int time);

//constructor with properly generated data members

Call();

//Constructor with 0 in all data members

Call(const Call & called);

//Copy constructor

int getCallTime();

//returns calltime

int getPriority();

//returns priority

int getTime();

//returns timer

bool DecreaseQuantity();

//decrements calltime

friend bool operator<(Call &call1,Call &call2);

//overrides < operator

friend bool operator<=(Call &call1,Call &call2);

//overrides <= operator

friend bool operator>(Call &call1,Call &call2);

//overrides > operator

friend bool operator>=(Call &call1,Call &call2);

//overrides >= operator

friend bool operator==(Call &call1,Call &call2);

//overrides == operator

friend bool operator!=(Call &call1,Call &call2);

//overrides != operator

// friend ostream& operator<<(ostream & out, const ItemToPurchase & f);

private:

int timer;

int priority;

int calltime;

};

#endif

Call.cpp

#include "Call.h"

#include"time.h"

#include <string>

#include <iostream>

using namespace std;

Call::Call(){

this->timer = 0;

this->priority=0;

this->calltime = 0;

}

Call::Call(int timeTaken){

this->timer = timeTaken;

int temp = (rand()+timer) %10;

if(temp ==0){

this->priority = 1;

}else if(temp <4){

this->priority = 2;

}else{

this->priority = 3;

}

this->calltime = (rand()+timer)%6+3;

}

int Call::getPriority(){

return priority;

}

int Call::getTime(){

return timer;

}

int Call::getCallTime(){

return calltime;

}

bool Call::DecreaseQuantity(){

// cout<<"This was calltime"<<calltime<<endl;

calltime--;

// cout<<"This is the calltime: "<<calltime<<endl;

return (calltime<1);

}

bool operator<(Call &call1,Call &call2){

return(call1.priority <call2.priority);

}

bool operator>(Call &call1,Call &call2){

return(call1.priority > call2.priority);

}

bool operator==(Call &call1,Call &call2){

return(call1.priority == call2.priority);

}

bool operator!=(Call &call1,Call &call2){

return(call1.priority != call2.priority);

}

bool operator<=(Call &call1,Call &call2){

return(call1.priority <=call2.priority);

}

bool operator>=(Call &call1,Call &call2){

return(call1.priority >=call2.priority);

}

Call::Call(const Call & called){

timer = called.timer;

priority = called.priority;

calltime = called.calltime;

//cout<<"Copy called"<<endl;

}

//ostream & operator<<(ostream& out,const ItemToPurchase & f){

// double price = f.price;

// int quantity = f.quantity;

// double total = price\*quantity;

// string info = f.name +" " +to\_string(quantity)+ " $" + to\_string(price) + " = $"+to\_string(total);

// out<<info;

// return out;

//}

CallCenter.h

#ifndef CallCenter\_\_

#define CallCenter\_\_

#include <iostream>

#include <string>

#include "Call.h"

#include "PriorityQueue.h"

using namespace std;

class CallCenter{

public:

CallCenter();

//Constructor

bool updateCenter(Call &call);

//Adds a new Call to PriorityQueue

void display();

//displays the end result of the program

void displayStatus();

//displays the update for the program

bool updateCenter();

//decrements the top of the queue's calltime and if empty, pops the next in

//its place

// friend ostream& operator<<(ostream & out, const ItemToPurchase & f);

private:

PriorityQueue waitlist;

Call top;

double timer;

double callnumber;

double avgPOne;

double avgPtwo;

double avgPthree;

double numberone;

double numbertwo;

double numberthree;

};

#endif

CallCenter.cpp

#include "Call.h"

#include "PriorityQueue.h"

#include "CallCenter.h"

#include <string>

#include <iostream>

using namespace std;

CallCenter::CallCenter(){

timer = 0;

callnumber = 0;

numberone =0;

numbertwo =0;

numberthree = 0;

avgPOne = 0;

avgPtwo = 0;

avgPthree = 0;

}

bool CallCenter::updateCenter(Call &caller){

callnumber++;

int temp = caller.getPriority();

if(temp==1){

numberone++;

}else if(temp==2){

numbertwo++;

}else{

numberthree++;

}

// cout<<"This is point h"<<endl;

waitlist.Enqueue(caller);

// cout<<"This is point i"<<endl;

return true;

}

bool CallCenter::updateCenter(){

timer++;

if (top.getCallTime() <1){

if (waitlist.getLength()==0){

return false;

}else{

top = waitlist.Dequeue();

int temp = top.getPriority();

if(temp==1){

avgPOne = timer - top.getTime();

}else if(temp==2){

avgPtwo = timer - top.getTime();

}else if(temp == 3){

avgPthree = timer - top.getTime();

}

}

}

top.DecreaseQuantity();

return true;

}

void CallCenter::display(){

double tempWait = (avgPOne+ avgPtwo + avgPthree)/callnumber;

cout<<"End of simulation"<<endl;

cout<<"This is the total of calls entered and answered: "<<callnumber<<endl;

cout<<"This is total time taken to answer all calls: "<<timer<<endl;

cout<<"This is the average wait time for a call to be answered: ";

if (callnumber!=0){

cout<<tempWait<<endl;

}else{

cout<<"No calls"<<endl;

}

cout<<"This is the average wait time for a Priority one call to be answered: ";

if (numberone!= 0){

cout<<avgPOne/numberone<<endl;

}else{

cout<<"No Priority one calls"<<endl;

}

cout<<"This is the average wait time for a Priority two call to be answered: ";

if (numbertwo!= 0){

cout<<avgPtwo/numbertwo<<endl;

}else{

cout<<"No priority two calls"<<endl;

}

cout<<"This is the average wait time for a Priority three call to be answered: ";

if (numberthree!= 0){

cout<<avgPthree/numberthree<<endl;

}else{

cout<<"No priority three calls"<<endl;

}

}

void CallCenter::displayStatus(){

cout<<"Simulation Update:"<<endl;

cout<<"Minutes into the simulation: "<<timer<<endl;

cout<<"Calls in the Waiitng List: "<<waitlist.getLength()<<endl;

cout<<"Calls that have been answered: "<<callnumber - waitlist.getLength()-1<<endl;

cout<<endl<<endl;

}

**PriorityQueue.h**

#ifndef PriorityQueue\_

#define PriorityQueue\_

#import <string>

#include "Call.h"

using namespace std;

typedef Call ItemType;

class PriorityQueue {

public:

PriorityQueue();

//Constructor

~PriorityQueue();

//Destructor

bool isEmpty() const;

//returns true if queue is empty, otherwise returns false

int getLength() const;

//returns length of the queue

bool Enqueue(const ItemType & newEntry);

//Adds a node wtih itemtype data type to the queue based on priority

ItemType Dequeue();

//Precondition: Must have atleast one node in the queue

//Returns the top of the queue

ItemType Peek() const;

//Precondition: Must have atleast one node in the queue

private:

class node{

public:

node \*next;

ItemType data;

};

int length;

node \*head;

};

#endif

**PriorityQueue.cpp**

#include "PriorityQueue.h"

#include <string>

#include <iostream>

#include <assert.h>

using namespace std;

PriorityQueue::PriorityQueue(){

head = nullptr;

length = 0;

}

bool PriorityQueue::isEmpty() const{

return (length == 0);

}

int PriorityQueue::getLength() const{

return length;

}

bool PriorityQueue::Enqueue(const ItemType & newEntry){

node\* newNode = new node;

newNode->data = newEntry;

if (length==0){

newNode->next = head;

head = newNode;

}else{

node\* p = head;

// cout<<"This is point j"<<endl;

while((p->next != nullptr)&&(p->next->data<p->data)){

p = p->next;

}

// cout<<"This is point k"<<endl;

newNode->next = p->next;

// cout<<"This is point l"<<endl;

p->next = newNode;

}

// cout<<"This is point m"<<endl;

length++;

return true;

}

ItemType PriorityQueue::Dequeue(){

assert(length!=0);

node\*b = head;

node\*c = head->next;

ItemType str = head->data;

head=c;

delete b;

length--;

// cout<<"Stack popped worked"<<endl;

return str;

}

ItemType PriorityQueue::Peek() const{

assert(length!=0);

return head->data;

}

PriorityQueue::~PriorityQueue(){

node\* a = head;

while(a!=nullptr){

node \*b = a->next;

delete a;

a = b;

}

}

**Main.cpp**

#include "CallCenter.h"

#include "Call.h"

#include "PriorityQueue.h"

#import <string>

#import <vector>

#include <fstream>

#include <sstream>

#include <iostream>

#include <assert.h>

using namespace std;

void getUserInput(int& timeLimit,double& prob);

//input: user input

//output: none

//side effect: validates user input and updates correct variables with it

//Precondition: User input must be correct or code will abort

Call generateCall(const int& timer);

//input: int timer

//output: Call object holding the time

void runCallCenter(int& timeLimit,double& prob,CallCenter& center);

//input:int timeLimit, double prob, Callcenter object

//output: none

//side effects: decrements timeLimit,uses probability to generate Calls,

// updates Callcenter every minute

void display(CallCenter& center);

//input: CallCenter object

//output:none

//side effects: displays the end results of the simulation

int main(){

int timeLimit = -1;

double prob = -1;

srand(time(0));

CallCenter center;

getUserInput(timeLimit,prob);

runCallCenter(timeLimit,prob,center);

return 0;

}

void getUserInput(int& timeLimit,double& prob){

cout<<"What is the number of minutes CallCenter accepts calls?(Mu";

cout<<"st be a positive number): ";

cin>>timeLimit;

assert(timeLimit>0);

cout<<endl<<"What is the probability of a phone call arriving?: ";

cin>>prob;

assert(prob>=0.1&&prob<=1);

}

Call generateCall(const int& timer){

Call newcall(timer);

return newcall;

}

void runCallCenter(int& timeLimit,double& prob,CallCenter& center){

// cout<<"This is point a"<<endl;

int checker = timeLimit%5;

int timer = 0;

while(timeLimit>0){

timer++;

double temp = ((double) rand() / (RAND\_MAX));

if (temp<=prob){

// cout<<"This is point b"<<endl;

Call tempcall = generateCall(timer);

// cout<<"This is point f"<<endl;

center.updateCenter(tempcall);

// cout<<"This is point g"<<endl;

}

// cout<<"This is point c"<<endl;

center.updateCenter();

timeLimit--;

if(timeLimit%5==checker){

// cout<<"This is point d"<<endl;

center.displayStatus();

}

}

cout<<"No longer accepting new calls"<<endl;

int timecheck = 5-checker;

while (center.updateCenter()){

// cout<<"This is point e"<<endl;

timecheck++;

if (timecheck%5==0){

center.displayStatus();

}

}

display(center);

}

void display(CallCenter& center){

center.display();

}

**Ouput1.txt**

What is the number of minutes CallCenter accepts calls?(Must be a number): 25

What is the probability of a phone call arriving?: 0.5

Simulation Update:

Minutes into the simulation: 5

Calls in the Waiitng List: 0

Calls that have been answered: 0

Simulation Update:

Minutes into the simulation: 10

Calls in the Waiitng List: 2

Calls that have been answered: 0

Simulation Update:

Minutes into the simulation: 15

Calls in the Waiitng List: 2

Calls that have been answered: 1

Simulation Update:

Minutes into the simulation: 20

Calls in the Waiitng List: 5

Calls that have been answered: 2

Simulation Update:

Minutes into the simulation: 25

Calls in the Waiitng List: 6

Calls that have been answered: 3

No longer accepting new calls

Simulation Update:

Minutes into the simulation: 30

Calls in the Waiitng List: 6

Calls that have been answered: 3

Simulation Update:

Minutes into the simulation: 35

Calls in the Waiitng List: 5

Calls that have been answered: 4

Simulation Update:

Minutes into the simulation: 40

Calls in the Waiitng List: 5

Calls that have been answered: 4

Simulation Update:

Minutes into the simulation: 45

Calls in the Waiitng List: 3

Calls that have been answered: 6

Simulation Update:

Minutes into the simulation: 50

Calls in the Waiitng List: 2

Calls that have been answered: 7

Simulation Update:

Minutes into the simulation: 55

Calls in the Waiitng List: 2

Calls that have been answered: 7

Simulation Update:

Minutes into the simulation: 60

Calls in the Waiitng List: 0

Calls that have been answered: 9

End of simulation

This is the total of calls entered and answered: 10

This is total time taken to answer all calls: 62

This is the average wait time for a call to be answered: 5.8

This is the average wait time for a Priority one call to be answered: No Priority one calls

This is the average wait time for a Priority two call to be answered: 3.33333

This is the average wait time for a Priority three call to be answered: 6.85714

**Output2.txt**

What is the number of minutes CallCenter accepts calls?(Must be a number): 15

What is the probability of a phone call arriving?: 0.6

Simulation Update:

Minutes into the simulation: 5

Calls in the Waiitng List: 1

Calls that have been answered: 0

Simulation Update:

Minutes into the simulation: 10

Calls in the Waiitng List: 2

Calls that have been answered: 1

Simulation Update:

Minutes into the simulation: 15

Calls in the Waiitng List: 3

Calls that have been answered: 3

No longer accepting new calls

Simulation Update:

Minutes into the simulation: 20

Calls in the Waiitng List: 3

Calls that have been answered: 3

Simulation Update:

Minutes into the simulation: 25

Calls in the Waiitng List: 2

Calls that have been answered: 4

Simulation Update:

Minutes into the simulation: 30

Calls in the Waiitng List: 1

Calls that have been answered: 5

Simulation Update:

Minutes into the simulation: 35

Calls in the Waiitng List: 0

Calls that have been answered: 6

End of simulation

This is the total of calls entered and answered: 7

This is total time taken to answer all calls: 40

This is the average wait time for a call to be answered: 5

This is the average wait time for a Priority one call to be answered: 1.5

This is the average wait time for a Priority two call to be answered: 3.5

This is the average wait time for a Priority three call to be answered: 8.33333

**Output3.txt**

What is the number of minutes CallCenter accepts calls?(Must be a number): 16

What is the probability of a phone call arriving?: 2

Assertion failed: (prob>=0.1&&prob<=1), function getUserInput, file Main.cpp, line 45.

Abort trap: 6

**Output4.txt**

What is the number of minutes CallCenter accepts calls?(Must be a positive number): -3

Assertion failed: (timeLimit>0), function getUserInput, file Main.cpp, line 42.

Abort trap: 6