Sohaib Ali Khan (vi4362xq)

CSCI 301 – Project 8

Instructor - Dr. Jie Meichsner

Due Date - 8/4/2020

1. **Design Document**

*Introduction*

Project 8 is about solving programming problem by using a sorted queues and priority queues.

*Data Structures –*

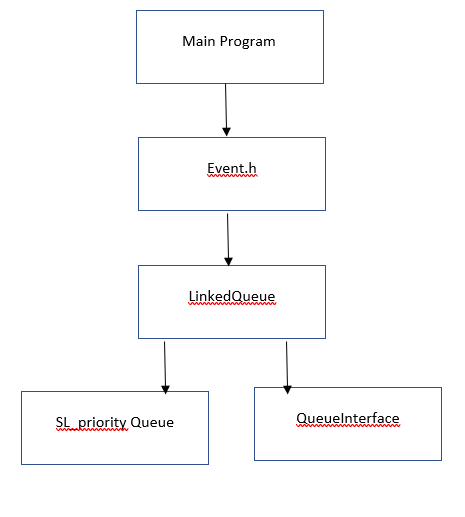
The program uses two main data structure. One ADT queue that is a regular queue ADT based on linked list and another priority queue that is also link based.

*Functions -*

The program has 3 functions in the main program.

* void simulate - Gets the initial list of event times from the input file and stores in the priority queue eventListPQueue.
* void processArrival(Event& newEvent, SL\_PriorityQueue<Event>& eventListPQueue, LinkedQueue<Event>& bankQueuePtr) - Calculates the arrival events according to the algorithm.
* void processDeparture(Event& departureEvent, SL\_PriorityQueue<Event>& eventListPQueue, LinkedQueue<Event>& bankQueuePtr) - Calculates the departure events according to the algorithm.

**Structure Chart of the Main Program**



1. **Code List**

LinkedQueue, SL\_PriorityiQueue, Node, PrecondViolatedExcep & LinkedList codes are not given below.

//Project 8

//Name - Sohaib Ali Khan, StarID - vi4362xq

//Instructor - Dr. Jie Meichsner

//Due Date - 8/4/2020 @ 11:50pm

/\*\* File name - Event.h

Purpose - Header and Implementation file for Event class\*/

#ifndef \_EVENT

#define \_EVENT

#include <string>

#include <iostream>

using namespace std;

class Event

{

private:

string type;

int arrivalTime;

int transactionTime;

public:

Event()

{

} //default constructor

Event(int a, int t)

{

arrivalTime = a;

transactionTime = t;

} //Consturctor with

void setType(string ch)

{

type = ch;

}

void setArrivalTime(int a)

{

arrivalTime = a;

}

void setTransactionTime(int t)

{

transactionTime = t;

}

string getType()

{

return type;

}

int getArrivalTime()

{

return arrivalTime;

}

int getTransactionTime()

{

return transactionTime;

}

friend istream& operator >>(istream& ins, Event& target)

{

ins >> target.arrivalTime >> target.transactionTime;

return ins;

} // end Node

friend ostream& operator <<(ostream& outs, Event target)

{

outs << target.arrivalTime << "\t"<<target.transactionTime << "\t" << target.type;

return outs;

} // end Node

}; //end Event

//nonmemeber overloading operators

bool operator >(Event a1, Event a2)

{

if (a1.getArrivalTime() < a2.getArrivalTime())

return true;

else

return false;

}

bool operator >=(Event a1, Event a2)

{

if (a1.getArrivalTime() <= a2.getArrivalTime())

return true;

else

return false;

}

bool operator <(Event a1, Event a2)

{

if (a1.getArrivalTime() > a2.getArrivalTime())

return true;

else

return false;

}

bool operator <=(Event a1, Event a2)

{

if (a1.getArrivalTime() >= a2.getArrivalTime())

return true;

else

return false;

}

bool operator == (Event a1, Event a2)

{

if (a1.getArrivalTime() == a2.getArrivalTime())

return true;

else

return false;

}

bool operator!= (Event a1, Event a2)

{

if (a1.getArrivalTime() != a2.getArrivalTime())

return true;

else

return false;

}

#endif

//Project 8

//Name - Sohaib Ali Khan, StarID - vi4362xq

//Instructor - Dr. Jie Meichsner

//Due Date - 8/4/2020 @ 11:50pm

/\*\* File name - Project8.cpp

Purpose - Main program file for Project8.cpp\*/

#include "LinkedQueue.h" // ADT Queue operations

#include "SL\_PriorityQueue.h"

#include "Event.h"

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

int customerCount = 0;

int currentTime = 0;

int waitTime = 0;

bool tellerAvailable = true;

//Function prototypes

void processArrival(Event& newEvent, SL\_PriorityQueue<Event>& eventListPQueue,

LinkedQueue<Event>& bankQueuePtr);

void processDeparture(Event& departureEvent, SL\_PriorityQueue<Event>& eventListPQueue,

LinkedQueue<Event>& bankQueuePtr);

/\*\* Gets the initial list of event times from the input file and stores in the priority queue eventListPQueue.

@pre There has to be an external file with the name input.txt containing the original list of event times.

\*/

void simulate()

{

LinkedQueue <Event> bankQueuePtr;

SL\_PriorityQueue <Event> eventListPQueue;

// bool tellerAvailable = true;

Event newArrivalEvent;

ifstream inputFile;

inputFile.open("input.txt");

while (!inputFile.eof())

{

inputFile >> newArrivalEvent;

newArrivalEvent.setType("Arrival");

// cout << newArrivalEvent.getArrivalTime() << "\t" << newArrivalEvent.getTransactionTime() << "\t" << newArrivalEvent.getType() << endl;

eventListPQueue.add(newArrivalEvent);

customerCount++;

} //end while

inputFile.close(); //close the file

Event newEvent;

//int currentTime;

// // Event loop

while (!(eventListPQueue.isEmpty()))

{

newEvent = eventListPQueue.peek();

// cout << "========" << endl;

// cout << newEvent.getArrivalTime() << "\t" << newEvent.getTransactionTime() << "\t" << newEvent.getType() << endl;

currentTime = newEvent.getArrivalTime();

if (newEvent.getType() == "Arrival")

{

processArrival(newEvent, eventListPQueue, bankQueuePtr);

}

else if (newEvent.getType() == "Departure")

{

processDeparture(newEvent, eventListPQueue, bankQueuePtr);

}

} //end while

} //end simulate

/\*\* Calculates the arrival events according to the algorithm.

@param an Event object newEvent, Priority queue - eventListPQueue and LinkedQueue - bankQueuePtr

\*/

// Processes an arrival event.

void processArrival(Event& newEvent, SL\_PriorityQueue<Event>& eventListPQueue,

LinkedQueue<Event>& bankQueuePtr)

{

Event newDepartureEvent;

newDepartureEvent = newEvent;

// Remove this event from the event list

eventListPQueue.remove();

if (bankQueuePtr.isEmpty() && tellerAvailable == true)

{

int departureTime = currentTime + newDepartureEvent.getTransactionTime();

// cout << departureTime << endl;

// Event newDepartureEvent;

newDepartureEvent.setArrivalTime(departureTime);

newDepartureEvent.setTransactionTime(newDepartureEvent.getTransactionTime());

newDepartureEvent.setType("Departure");

eventListPQueue.add(newDepartureEvent);

tellerAvailable = false;

cout << "Processing an arrival event at time: \t" << currentTime << endl;

// SL\_PriorityQueue <Event> testQ(eventListPQueue);

// while (!testQ.isEmpty())

// {

// cout << testQ.peek();

// cout << endl;

// testQ.remove();

// }

}

else

{

bankQueuePtr.enqueue(newEvent);

newDepartureEvent.setType("Departure");

cout << "Processing an arrival event at time: \t" << currentTime << endl;

}

} //end processArrival

/\*\* Calculates the departure events according to the algorithm.

@param an Event object newEvent, Priority queue - eventListPQueue and LinkedQueue - bankQueuePtr

\*/

// Processes a departure event .

void processDeparture(Event& departureEvent, SL\_PriorityQueue<Event>& eventListPQueue,

LinkedQueue<Event>& bankQueuePtr)

{

// Remove this event from the event list

eventListPQueue.remove();

if (!bankQueuePtr.isEmpty())

{

// Customer at front of line begins transaction

Event newDepartureEvent;

newDepartureEvent = bankQueuePtr.peekFront();

bankQueuePtr.dequeue();

if (newDepartureEvent.getArrivalTime() <= currentTime)

{

waitTime = waitTime + (currentTime - newDepartureEvent.getArrivalTime());

}

int departureTime = currentTime + newDepartureEvent.getTransactionTime();

newDepartureEvent.setArrivalTime(departureTime);

newDepartureEvent.setType("Departure");

eventListPQueue.add(newDepartureEvent);

cout << "Processing a departure event at time: \t" << currentTime << endl;

}

else

{

tellerAvailable = true;

cout << "Processing a departure event at time: \t" << currentTime << endl;

}

} //end processDeparture

int main()

{

simulate();

double avgWaitTime = (double)waitTime / (double)customerCount;

cout << "Simulation ends." << endl;

cout << endl;

cout << "Final Statistics: " << endl;

cout << "\tTotal number of customers processed: " << customerCount << endl;

cout << "\tAverage amount of time spent waiting: " << avgWaitTime << endl;

return 0;

} //end main

1. **User Document**

Problem

Implement the event-driven simulation of a bank that Chapter 13 described. A queue of arrival events will represent the line of customers in the bank. Maintain the arrival events and departure events in a priority queue, sorted by the time of event. Use a link-based implementation for the event list.

The input is a text file of arrival and transaction times. Each line of the file contains the arrival time and required transaction time for a customer. The arrival times are ordered by increasing time.

Your program must count customers and keep track of their cumulative waiting time. These statistics are sufficient to compute the average waiting time after the last event has been processed.

Other requirement:

In the class Event, comparison operators (<, <=, >, >= , == and !=)must be defined to use priority queue.

[Hint: You should test the Event class before you use it in this project.]

Output

Display a trace of the events executed and a summary of the computed statistic (total number of the arrivals and average time spent waiting in line). For example, if an input file contains the following customer samples:

Arrival time Transaction time

1 5

2 5

4 5

20 5

22 5

24 5

26 5

28 5

30 5

88 3

The program’s name is Project8.cpp. It is located at the following directory on CentOS.

/home/STCLOUDSTATE/vi4362xq/Project8

To compile the three programs simply enter:

g++ Project8.cpp

To run the programs enter:

./a.out and follow the prompts on the screen.

1. **Test Data**

./a.out

Processing an arrival event at time: 1

Processing an arrival event at time: 2

Processing an arrival event at time: 4

Processing a departure event at time: 6

Processing a departure event at time: 11

Processing a departure event at time: 16

Processing an arrival event at time: 20

Processing an arrival event at time: 22

Processing an arrival event at time: 24

Processing a departure event at time: 25

Processing an arrival event at time: 26

Processing an arrival event at time: 28

Processing an arrival event at time: 30

Processing a departure event at time: 30

Processing a departure event at time: 35

Processing a departure event at time: 40

Processing a departure event at time: 45

Processing a departure event at time: 50

Processing an arrival event at time: 88

Processing a departure event at time: 91

Simulation ends.

Final Statistics:

Total number of customers processed: 10

Average amount of time spent waiting: 5.6

1. **Summary**

After doing Project8 I have gathered greater understanding of Queues and Priority Sorted Queues and how to implement link-based queues in applications. I have also gained greater knowledge of simulation and how to implement algorithms into computer programs using queues and priority queues.