Data Structures and Algorithms

EasyTrip Bus Company

Project Requirements

Objectives

By the end of this project, the student should be able to:

- Understand unstructured, natural language problem description and derive an appropriate design.
- Intuitively modularize a design into independent components and divide these components among team members.
- Build and use data structures to implement the proposed design.
- Write a **complete object-oriented C++ program** that performs a non-trivial task.

Introduction

A bus company needs to handle passenger transportation in the most efficient and profitable way. The company needs to automate the bus assignment process to achieve good and fair use of its buses. Fortunately, the company manager discovered your programming skills and your deep knowledge of different data structures and decided to hire you to develop a program that **simulates** the **operation of the** transportation process and **calculates** some related **statistics** in order to help improve the overall process.

Project Phases

Project Phase	*%*	*Deadline*
Phase 1	**30%**	**Week 9**
Phase 2	**70%**	**Week 14**

Late Submission is not allowed

NOTE: Number of students per team = 4 students.

The project code must be totally yours. The penalty of cheating any part of the project from any other source is not ONLY taking ZERO in the project grade but also taking MINUS FIVE (-5) from other class work grades. So, it is better to deliver an incomplete project rather than cheating it. It is totally your responsibility to keep your code away from others.

Simulation Time

- Any reference to time in this document we mean time in ***hh:mm*** form
- We assume that we are simulating the operation for one full day only. Time starts at 00:00 and ends at 23:59

Passengers, Buses, and Stations

Passengers:

The following pieces of information are available for each passenger:

- **Arrival Time:** the time (hh:mm) when the passenger arrives at the station and is ready to get on a bus.
- **Passenger's start and end stations.**
- **Passenger Get ON/OFF Time:** it is the time (**in seconds**)needed for one passenger to get on/off the bus.
- **Passenger Type:** There are 3 types of passengers:
  - **(SP) Special categories passengers:** Those are the passengers that have the highest priority to get on the bus before others. SP type includes 3 subcategories: aged passengers, POD (people of determination), and pregnant women
  - **(WP) passengers with wheelchairs** have to use special types of buses that are equipped for that purpose.
  - **(NP) Normal passengers:** all other passenger

Stations:

- The company serves **S** stations along the bus road. Stations are numbered from 1 to S. The number of stations and the time between each two stations is loaded from the input file.
- Initially (at the start of every day), all buses are at a special station (**Station#0**). During working hours, buses are moving from station#1 to station #S (***Forward direction***) and back from #S to #1 (***Backward direction***).
- A bus completes one **journey** when it moves from Station #1 to station #S or vice versa.
- After working hours ALL buses should return to station #0.
- Bus maintenance takes place at station #0.

Buses:

At startup, the system loads (from a file) information about the available **buses**. For each bus, the system will load the following information:

- **Bus Type:** There are 2 types of buses:
  - **Wheel-chair bus (WBus):** can be used by WP **ONLY**.
  - **Mixed bus (MBus)** can be used by both SP or NP. But WP are not allowed to use this type of buses.

- **Bus Capacity (BC)**: A bus can carry many passengers in the same trip. BC is the number of passengers a bus can carry.
- **Maintenance (Checkup) time:**
  After finishing **J journeys**, a bus must encounter a checkup and maintenance to make sure it is working properly. The checkup time (**in hours**) is the number of hours a bus needs to finish its checkup. Both J and checkup times are loaded from the input file.

-

Basic Rules

- ❑ The Company working hours are from 4:00 to 22:00. Otherwise, it is **off-hours**.
- ❑ At the same station, there may be passengers waiting to go in the forward direction, and some are waiting to go in the backward direction
- ❑ A passenger can get on the bus during working hours only. If a passenger is waiting for a bus and the time passes 22:00, he will not be allowed to ride any of the buses, and he should leave immediately. However, if a passenger is already on a bus and the time passes 22:00 before he reaches his destination, the bus must drive him to his destination even after working hours.

- ❑ **The allowed activities during off-hours are:**
- ✔ A bus with passengers who haven't reached their destination yet must drive them all to their destinations.
- ✔ A bus that has transported all its passengers and is returning back to station #0.
- ✔ Bus maintenance operations can start/continue during off hours.
- ❑ All buses of the same type have the same capacity.
- ❑ At the beginning of the working hours, the company releases one bus from station #0 every 15 minutes.
- ❑ A bus stops at a station only if it has passengers to get off at this station or there are some passengers waiting to ride ***in the same direction*** (forward/backward) of the bus.
- ❑ When a bus stops at a station, first let its passengers get off, then allow waiting passengers to get on according to the boarding criteria described below. Once boarding is done or the bus is full, the bus should move immediately.
- ❑ If a passenger can't get on a bus due to bus capacity, it should wait for the next bus.
- ❑ If more than one bus is waiting at the same station, they should be boarded in the same order they reach the station.
- ❑ A bus cannot be assigned a passenger during its checkup time

- Checkup duration and bus speed are the same for all buses of the same type.

Definitions

- **AT:** The passenger's arrival time
- **Move Time (MT):** The time at which the bus carrying the passenger starts to move away from the station.
- **Waiting passenger:** The passenger that has arrived at a station but hasn't gotten on a bus yet.
- **Passenger Waiting Time (WT):** ***WT = MT - AT***
- **Moving passenger:** The passenger who is on his way to his station.
- **Finish Time (FT):** The time when a passenger has reached his station **and** left his bus.
  Assume a bus reaches a station at time T.
  Then FT for a certain passenger who is getting off = T + Get OFF time for all passengers getting off before him + Get OFF time for this passenger.
- **Passenger Trip Time (TT):** ***TT = FT - MT***
- **Bus Busy Time:** Time a bus is NOT empty. This doesn't include time for a bus to return to Station #0 or the time after off hours.
- **Bus utilization:** (percentage)
  = **tDC/(BC*N) * (tBT/TSim)** , N ≠ 0 (if N=0, utilization = 0%)
  tDC: total passengers transported by this bus, BC: bus capacity,
  N:total delivery trips of this bus
  tBT: total busy time for this bus
  TSim: total Simulation time

Passenger Boarding Criteria

To determine the next passenger to get on a bus (if a bus is available), the following **criteria** should be applied for all the waiting passengers **at each minute:**

Boarding Order:

1. If a WBus is ready at a station, allow ONLY WP to get on the bus.
2. If a MBus is ready at a station, first allow aged passengers, then POD passengers, then pregnant women. Then allow normal passengers to get on.
3. If a bus gets filled at any time, it should stop boarding and move immediately.

For normal passengers ONLY

- A **waiting** passenger may decide to leave the station at any time. Such info is loaded from the input file.
- if a passenger waits more than a maximum waiting time, **maxW,** minutes he should be **automatically promoted** and the system gives him the same priority as aged passengers (i.e. He joins the end of the current aged passengers list). (**maxW** is read from the input file).

Input/Output File Formats

Your program should receive all information to be simulated from an input file and produce an output file that contains some information and statistics. This section describes the format of both input and output files and gives a sample for each.

The Input File

- Line 1 contains ***S, ST*** where S is the number of stations and ST is the time (in minutes) between each two successive stations
- Line 2 contains two integers representing the total number of buses of each type. (WBus_count, M-Bus_count)
- Line 3 contains two integers for bus capacities for WBus and MBus respectively.
- Line 4 contains three integers:
  - **J:** is the number of trips the bus completes before performing a checkup
  - **C-WBus:** is the checkup duration in hours for WBus
  - **C-MBus:** is the checkup duration in hours for MBus
- Line 5 contains two integers **maxW** (in mins) and get on/off time (in secs)
- Line 6 contains a number ***E*** which represents the number of **events** following this line.
- Then the input file contains ***E*** lines (a line for **each event**). An event can be:
  - **Arrival** event for a new passenger. Denoted by letter **A**, or
  - **Leave** of an existing passenger. Denoted by letter **L**

NOTE: The input lines of all events are sorted by the event time in ascending order

Events

- **Arrival event line** has the following information:
  - **A** denoted an arrival event
  - **TYP** is the passenger type (SP/WP/NP) For SP, an extra word is added at the end of the line to show if the passenger is aged, POD, Pregnant
  - **ET** is the event timestep (in hh:mm)
  - **ID** is a unique sequence number that identifies each passenger
  - **STRT:** start station
  - **END**: start station
- **LEAVE event line** has the following information:
  - **L** means this is a LEAVE event line
  - **ET** is the event timestep.

- **ID** is the ID of the passenger who decided to leave.

Ignore this event if the visitor of the given ID has already been picked.

Sample Input File

```
10   12          ☐ 10 stations with 12 min between each 2 stations
3    11          ☐ 3 WBuses , 11 MBuses
17   45          ☐ Capacity of each bus type (WBus then MBus)
12   3  2        ☐ no. of trips before checkup and the checkup durations

35       10      ☐ maxW (minutes), get on/off time (seconds)
150      ☐ no. of events in this file. This line should be followed by 150 lines
A  NP   4:3   1   2   5       ☐ Arrival event example (NP)
A  NP   4:6   2   10  4
A  WP   4:10  3   7   3       ☐ Arrival event example (WP)
A  SP   4:10  4   5   9  aged  ☐ Arrival event example (SP, aged)
L  4:12  1                    ☐ Leave event example
A  WP   5:7   5   6   2
A  SP   11:6  6   4   8   POD
And so on for the following lines to have 150 event lines
```

The Output File

The output file you are required to produce should contain **M** output lines (a line for each passenger that arrives his destination) of the following format:
FT ID AT WT TT
(Read the "Definitions Section" mentioned above)
The output lines **must be sorted** by **FT** in ascending order.

Then the following statistics should be shown at the end of the file:

1. Total number of passengers and number of passengers of each type
2. Average waiting time for all passengers
3. Average trip time for all passengers
4. Percentage of passengers (relative to the total number of normal passengers) who wait for maxW and got auto-promoted.
5. Total number of buses and number of buses of each type
6. Average busy time of all buses
7. Average utilization of all buses

Sample Output File

The following numbers are just for clarification and are not produced by actual calculations.

FT	**ID**	**AT**	**WT**	**TT**
4:33	**1**	**4:4**	**0:9**	**0:20**
4:52	**10**	**4:1**	**0:15**	**0:36**
5:55	**4**	**4:0**	**0:10**	**1:45**
..... and so on				
.....				

passengers: 124 [NP: 100, SP: 15, WP: 9]
passenger Avg Wait time= 0:10
passenger Avg trip time = 0:35
Auto-promoted passengers: 4%
buses: 20 [WBus: 3, MBus: 17]
Avg Busy time = 91%
Avg utilization = 87%

Program Interface

The program can run in one of two modes: **interactive,** or **silent mode**. When the program runs, it should ask the user to select the program mode.

1. Interactive Mode: Allows the user to monitor the passengers and buses. The program should print an output like that shown below. In this mode, the program prints the current time then pauses for an input from the user ("Enter" key for example) to display the output of the next time.

```
Current Time (Hour:Min)==> 4:9
==============      STATION #1     =================
2 Waiting SP: FWD[1(Ag)] BCK[11(PD)]
2 Waiting WP: FWD[13,15] BCK[]
5 Waiting NP: FWD[5,12] BCK[4,6,9]
2 Buses at this station:
B7[FWD, MBus, 45] {10, 37}
B3[BCK, WBus, 7]  {66, 22}
-----------------------------------------------------
2 In-Checkup buses: 13, 15
-----------------------------------------------------
5 finished passengers: 2, 3, 8, 55, 89
Press any key to display next station
```

Output Screen Explanation

- The screen shows the status at one of the stations. When the user presses any key, the next station is displayed at the same time.
- The numbers shown are the IDs of passengers of different types. The screen shows also the direction of each waiting passenger (FWD/BCK)
- Then the buses at this station (if any) are displayed. For each bus, bus ID, direction, type and capacity are shown. Then IDs of passengers on that bus.
- The remaining is self-explanatory.
- The above screen is just for explanation and is not generated by actual simulation.

2. Silent Mode, the program produces only an output file. It does not print any simulation steps on the console. It just prints the following screen

```
Silent Mode
Simulation Starts...
Simulation ends, Output file created
```

NOTE: No matter what mode of operation your program is running in, **the output file** should be produced.

Project Phases

You are required to write **object-oriented** code and use classes to implement different data structures in your system.

Before explaining the requirement of each phase, all the following are NOT allowed to be used in your project:

- You are not allowed to use **C++ STL** or any external resource that implements the data structures you use. ***This is a data structures course where you should build data structures yourself from scratch.***

- You need to get instructor's approval before making any **custom (new)** data structure.
  NOTE: No approval is needed to use the known data structures.

- ***Do NOT allocate the same passenger more than once***. Allocate it once and make whatever data structures you chose point to it (use lists of pointers). Then, when another list needs access to the same passenger, DON'T create a new copy of the same passenger; just **share** it by making the new list point to it or **move** it from the current list to the new one.

  SHARE, MOVE, DON'T COPY...

- You are not allowed to use **global variables** in your code.

- You need to get instructor approval before using **friendships.**

Phase 1

In this phase you should decide the data structures that you will use in your project.
Selecting the appropriate DS for each list is the core target of phase 1 and the project as a whole.

Phase1 Report You should deliver a report in the following format:

Data Structures and Algorithms	Spring 2023

Data Structures and Algorithms
Project Phase1 Report

Team Name: **Number of members:**

Team Email:

Members' Info:

Member Name	ID	Email

Selected Data Structures

List Name	Chosen DS	Justification
e.g. NEW List	Write the DS you have chosen e.g. Queue/Stack/ Pri-Q/ Tree .. etc	- Why you have chosen that DS - the **operations** you need to perform on this DS and the **complexity** of each operation e.g. Insert a process = O(1) Remove process = O(log n) ….etc

☐ **Repeat the above for each list in the project**

You should cover all lists in your project

See "**Data Structures Selection Guidelines"** before writing this report

Data Structures Selection Guidelines:

1- Do you need a separate list for each passenger type? passenger assignment criteria described above should affect your decision.
2- Do you need a separate list for each bus type? What about the list of stations?
3- Do you need a separate list for each passenger status? i.e. a list for waiting and another one for moving (on the bus) and a third one for finished or just one list for all?
4- Do you need a separate list for each bus status? For example, would you make a list for available buses and another one for those under checkup,…etc. or just one list for all.
5- Do you need to store **finished** passengers? When should you get rid of them to save memory?
6- **Which list type** is most suitable to represent the lists taking into account the **complexity of the operations** needed for each list (e.g. insert, delete, retrieve, shift, sort, etc.). You may need to make a survey about the complexity of the operations. Then, decide what are the most frequent operations needed according to the project description. Then, for this list, choose the DS with the best complexity regarding those frequent operations.
7- Keep in mind that you are **NOT** selecting the DS that would **work in phase1**. **You should choose the DS that would work efficiently for both phases.**
8- **Important**: if you find out that a list can be implemented using two types, you should choose the more restricted type. For example if a list can be implemented using queue and a general list, you should use queue.

Note: You need to read the *"File Format"* section and to see how the input data and output data are sorted in each file because this will affect your DS selection.

Phase1 Code

You have to finalize the following in phase1:

1. *Full data members of classes (See Appendix A below)*
2. *Class Event and all its subclasses.*
3. *Full implementation for all data structures that you will use to represent the lists of passengers and buses.*
4. *File loading function.*
5. *Simple Simulator function for Phase 1. The main purpose of this function is to test different lists and make sure all operations on them are working properly. This function should:*
    a. *Perform any needed initializations*
    b. *Call file loading function*
    c. *At each timestep do the following:*
        i. *Get the events that should be executed at current timestep*
        ii. *For arrival events, generate a passenger of the required type and add it to the appropriate waiting passengers list.*

iii. *For Leave event, delete the corresponding normal passenger (if found in one of the waiting lists. Ignore otherwise)*

iv. For **each station**, Generate a random number from 1 to 100.
   1. If 1<=number<=25, Move one SP passenger to the Finish list.
   2. If 20<=number<=30, Move one WP passenger to the Finish list.
   3. If 50<=number<=60, Move one NP passenger to the Finish list.

d. *Print all applicable info to the interface as described in the "Program Interface" section without bus info.*

Notes:

- *The simulation function stops when there is no more events and all passengers are in the finish list(s).*
- *NO actual passenger-bus assignment is required in Phase 1.*

Phase1 Video

You should record a short video (less than 10 minutes) showing a demo for phase1. In the video, you should do the following:

- Give a **very** brief explanation for the types of data structures lists you have chosen for each list in the project.
- Show an input file with at least 30 passengers of non-trivial scheduling requirements
- Run the code and explain (for a small part of the input) how passengers are moved from one list to another. Map your explanation on the data loaded from input file,

Phase 1 Deliverables:

Each team is required to submit:

1. A text file named ***ID.txt*** containing team members' names, IDs, and emails.
2. **Phase1 Report**
3. **Phase1 code** [Do not include executable files].
4. One sample input file (test case). The sample file must include at least 30 events with 10% cancellation and 10% promotion events**.**
5. **Phase1 Video**

Phase 2

In this phase, you should extend code of phase 1 to build the full application and produce the final output file. Your application should support the different operation modes described in the "Program Interface" section.

Phase 2 Deliverables:

Each team is required to deliver the following:

1. A text file named ***ID.txt*** containing team members' names, IDs, and emails.
2. **Final project code** [Do not include executable files].
3. **Six comprehensive sample input files (test cases) and their output files. Sample input files must cover simple to complex scenarios.**
4. **Workload document**. A document that contains the table shown below:

Phase2 - Workload Table TO BE EDITED	
Phase2 Item	**Done By**
	member name who did it
	Done By

Phase2 Items weights:
TO BE EDITED

Notes:
You must adhere to these items when distributing load among team members
load should be equally distributed based on the following table

Phase2 Item	notes	Relative points

Penalties		
Memory management	**alloc / dealloc appropriately. No copy of passengers/buses between lists**	**minus 0 to 10% of item grade when applicable**
item not running		**minus 30% of item grade**
class responsibility		**minus 20% of item grade**

Project Evaluation

These are the main points for project evaluation:

- **Successful Compilation:** Your program must compile successfully with zero errors. Delivering the project with any compilation errors will make you lose a large percentage of your grade.

- **Object-Oriented Concepts:**
  - **Modularity:** A **modular** code does not mix several program features within the same unit (module). For example, the module that does the core of the simulation process should be separate from the module that reads the input file which, in turn, is separate from the module that implements the data structures. This can be achieved by:
    - Adding classes for each different entity in the system and each DS used.
    - Dividing the code in each class into several functions. Each function should be responsible for a single job. Avoid writing very long functions that do everything.
  - **Maintainability:** A maintainable code is the one whose modules are easily modified or extended without a severe effect on the other modules.
  - **Class responsibilities:** No class is performing the job of another class.

- **Data Structures & Algorithms**: After all, this is what the course is about. You should be able to provide a concise description and a justification for: (1) the data structure(s) and algorithm(s) you used to solve the problem, (2) the **complexity** of the chosen algorithm, and (3) the logic of the program flow.

- **Interface modes**: Your program should support the two modes described in the document.

- **Test Cases**: You should prepare different comprehensive test cases (at least 6). Your program should be able to simulate different scenarios, not just trivial ones.

- **Coding style**: How elegant and **consistent** is your coding style (indentation, naming convention, ...)? How useful and sufficient are your comments? This will be graded.

Individuals Evaluation

Each member must be responsible for writing some project modules (e.g. some classes or some functions) and must answer some questions showing that he/she understands both the program logic and the implementation details. The workload between team members must be almost equal.

✔ The grade of **each student** will be divided as follows:

- **[70%]** for his individual work (the project part he/she is responsible for).
- **[25%]** for integrating his work with the work of ALL students who Finished or nearly finished their project part.
- **[5%]** for cooperation with other team members and helping them if they have any problems with their parts (helping does NOT mean to implement their parts for them).

✔ If one or more members don't do their work, the other members will NOT be affected **as long as** the students who finished their part integrated all their parts together AND their part can be tested to see the output.

✔ If a member couldn't finish his part, other members would integrate together to at least take the integration grade.

You should **inform the TAs** before the deadline **with sufficient time (some weeks before)** if any problems happen between team members to be able to take grading action appropriately.

Penalties:

Cheat Penalty

The project code must be fully yours. The penalty of cheating any part of the project from any other source is not ONLY taking ZERO in the project grade but also taking MINUS FIVE (-5) from other class work grades. So it is better to deliver an incomplete project rather than cheating it. It is totally your responsibility to keep your code away from others.

Late Penalty:

For phase 1: every one day late = -25% of phase1 grade

For phase 2: one day late = -50%. No more latency is allowed.

Appendix A – Guidelines for Project Code

The main classes of the project should be Company, passenger, bus, Event, and UI (User Interface). In addition, you need lists of appropriate types to hold events, passengers, buses,..etc.

Event Classes:

There are two types of events: Arrival and Leave events. You should create a base class called "***Event***" that stores the event time and the related passenger's info. This should be an abstract class with a pure virtual function "**Execute**". The logic of the Execute function should depend on the Event type.

For each type of the two events, there should be a class derived from ***Event*** class. Each derived class should store data related to its operation. Also, each of them should override the function **Execute** as follows:

1. arrivalEvent::Execute should create a new passenger and add it to the appropriate list
2. leaveEvent::Execute should cancel the requested normal passenger (if found and hasn't been on a bus yet)

Passenger and Bus Classes:

Should have data members to store all info of passengers and buses. In addition to appropriate member functions.

Company Class

This class is the maestro class that manages the system.

It should have an appropriate ***list of Event pointers*** to store all events loaded from the file at system startup. Also it should have lists to hold passengers and buses of different types and status. In addition to a list of stations.

It should have member functions to:

1- At program startup, open the input file and load buses data to fill buses list(s) and to rid the events list.

2- At each minute, (**and among other things**)

   a. Execute the events that should be executed at that min
   b. Check waiting passenger to assign them to available buses
   c. Move passengers from waiting to moving to finish lists
   d. Move buses from available to moving to checkup to available again
   e. Collect statistics that are needed to create output file
   f. Calls UI class functions to print details on the output screen

3- Produce the output file at the end of simulation

UI Class

This should be the class responsible for reading any inputs from the user and printing any information on the console. It contains the input and output functions that you should use to show status of the system at each hour.
At the end of each hour, this class should print the output screen (see Program Interface section) showing what has happened during that hour.
The is the only class whose functions can have input and output (cin and cout) lines
Input and output lines must NOT appear anywhere else in the program
This class is NOT responsible for files read/write