EE 441 HW#2
Due Date: December 14, 2014 (23:59)

Emin Zerman / ARC-202 / zerman@metu.edu.tr
Tayfun Eylen / EA 403 / eylen@metu.edu.tr
Yeti Ziya Gurbuz* / ARC-202 / yeti@eee.metu.edu.tr

***You may ask HW#2 related questions to yeti@eee.metu.edu.tr.**

Easy Everywhere

1. Introduction

EE Tech Company has developed a novel air transportation system, Easy Everywhere. It is like Dolmuş on the air. In Easy Everywhere, each passenger in a single aircraft can travel different locations unlike traditional aircrafts. A special aircraft, which is composed of stacked capsules, has been invented for this purpose. Each capsule carries a passenger and leaves (separates from) the aircraft when the aircraft is at the longitude corresponding to the location to be travelled. The aircraft drops the capsules starting from the rear and the separated capsules travel to the latitude corresponding to the location to be travelled themselves. Therefore, the passengers and the corresponding capsules are placed in sorted order according to the longitudes of the locations. Hence the capsule to be dropped first is placed at the rear. An aircraft heading towards West from Ankara is depicted in Figure 1.

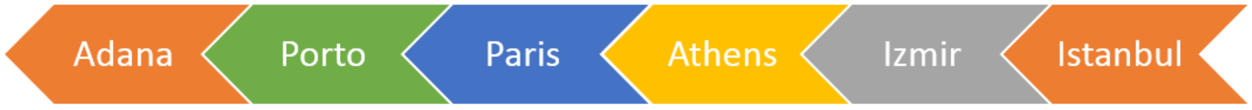

Figure 1: An aircraft composed of 6 capsules and heading towards West from Ankara

An aircraft can be composed of any number of capsules and the number of the capsules depends on the tickets sold for a flight. For every ticket issued, a capsule is constructed and integrated to the aircraft. Integrating a capsule to the aircraft is very costly and a capsule can only be used once. The cost of the capsules can be well-compensated by the price of the flight tickets but EE Tech Company is suffering too much loss due to a lack of its ticket sales handling capability. How many passengers are to be in a flight is not known a priori. Therefore, EE Tech Company constructs a specified number of capsules. Generally, the demand for a flight is less than the number of constructed capsules which causes serious loss. On the other hand, when the number of pre-constructed capsules cannot meet the demand, they lose again from selling no more tickets.

Therefore, EE Tech Company aims to develop a passenger registration system that can overcome the aforementioned situations. They want YOU to design and implement a system that can minimize the cost and maximize the gain. In other words, your system should make sure that the number of capsules is exactly equal to the number of passengers.

2. Requirements

a. Design and implement a class structure to represent the capsules.

b. Design and implement a class structure to represent the aircraft.

c. Implement a simple User Interface (UI) to

- register a passenger to a flight
- simulate a flight

d. Derive the time and space O(.) complexity of the full execution of your homework.

- What is meant by “full execution” will be clear in the next section.
- Space complexity should be no worse than O(N) provided that N is the number of passengers registered for a flight.

3. Homework

In this homework you will implement an Easy Everywhere simulator. A simulation consists of registration of passengers to a flight, construction of the aircraft for the flight and simulating the flight. You will solve this problem by utilizing “classes” and “dynamic memory management” in C++. You will design an “EECapsule” class to represent capsules, an “EEAircraft” class to represent the aircraft and you will write a function to realize the flight. The details and some requirements of the classes and the function will be introduced in the following subsections. Please read every detail and be attentive to encapsulation (whether a member or a method should be public or private), information passing, data types (integer, double, etc.) and some other related issues about class and function implementation.

a. class EECapsule

This is the class to represent a capsule object for a registered passenger. This class should have the following members:

- a variable **Name** to hold the name of the passenger,
- a variable **CID** to hold the citizen ID, which consists of integers, of the passenger,
- a variable **Latitude** to hold the latitude, which is an integer between -90 and 90, of the location to be travelled by the passenger,
- a variable **Longitude** to hold the longitude, which is an integer between -180 and 180, of the location to be travelled by the passenger.

These members above should be set only during construction of the class and should not be reached for editing outside of the class for proper implementation. In other words, the values of these variables cannot be changed once the class is constructed.

This class should also have the following methods (a.k.a. member functions):

- a proper **Constructor**,
- a method which returns **Longitude** value: This method should be able to be called outside of the class.
- a method which displays the information within the capsule according to the following format:

```
========================
Name:           Leeroy Jenkins
CID:            56221111
Destination:    +40, +33
========================
```

Note that the number of ='s is equal to the length of the longest line. It is the name line for this example; however, it can be other lines for different cases as well. Also note that the destination information format is ((sign)Latitude, (sign)Longitude). This method should be able to be called outside of the class.

The implementation detail of the whole class is up to you. Above, the fundamental requirements are given only. The class will be of your design hence you may add members or methods for your own purposes to make things simpler.

b. class EEAircraft

This is the class to represent an aircraft object. This class should be an array structure and have the following members:

- a variable **Latitude** to hold the latitude, which is an integer between -90 and 90, of the departure,

- a variable **Longitude** to hold the longitude, which is an integer between -180 and 180, of the departure,
- a variable **Direction** to hold the flight direction, which is -1 for heading towards West and +1 for heading towards East.

These members above should be set only during construction of the class and should not be reached for editing outside of the class for proper implementation. In other words, the values of these variables cannot be changed once the class is constructed.

This class should also have the following methods (a.k.a. member functions):

- a proper **Constructor**,
- a method, **Integrate**, which inserts an EECapsule object to the class: This method should be able to be called outside of the class,
- a method, **Drop**, which returns the EECapsule object to be dropped and removes the dropped EECapsule object from the class. Note that the dropping order of the capsules is according to the departure location and the flight direction as explained in the introduction section. This method should be able to be called outside of the class,
- a method, **Display**, which displays the current structure of the aircraft according to the following format:

```
========================
EEAircraft
Departure:      +41, +29
Direction:      -1
========================
========================
Name:           Leeroy Jenkins
CID:            56221111
Destination:    +40, +33
========================
========================
Name:           Nat Pagle
CID:            204501
Destination:    +25, -78
========================
                .
                .
                .
========================
Name:           Harrison Jones
CID:            5543154
Destination:    +49, +2
========================
```

Note that the displayed structure of the aircraft is according to the explanation provided in the introduction section. That is, the capsules are placed in the order according to the longitudes of their destinations so that the capsule to be dropped next is placed at the rear. At the top, the information about the flight is displayed. The departure information format is ((sign)Latitude, (sign)Longitude). Just below the departure information, the information about the existing capsules is placed, starting from the front of the aircraft.

The implementation detail of the whole class is up to you. Above, the fundamental requirements are given only. The class will be of your own design. You may and <u>you should</u> add members or methods for your own purposes to make things proper and simpler.

c. function SimulateFlight

This function gets an aircraft object as its input and realizes a flight by displaying the dropped capsules one by one and displaying the structure of the aircraft after each drop. Pseudo code of an example simulator is given in Table 1.

Table 1: Pseudo code for flight simulator

```
SimulateFlight (input EEAircraft myAircraft)
    while myAircraft has capsules
        EECapsule tempCapsule = myAircraft.Drop();
        print: “Dropped Capsule:”
        tempCapsule.DisplayInfo();
        print: “Current Aircraft:”
        myAircraft.Display();
        print: “***”
    endwhile
endSimulateFlight
```

d. User Interface

The purpose of the user interface is getting user inputs for creating a flight, registering passengers to the flight and simulating the flight. The required UI should operate as follows:

- Welcome the user
- Ask for a new flight or termination
  - If termination is selected, exit the program.
  - For the new flight, input the departure location (lat. & long.) and flight direction. Proceed to registration interface.
- Registration Interface: Ask for the registration of a new passenger or closing the registration.
  - For passenger registration, input the name, CID and destination information.
  - If registration is closed, show the resultant aircraft by using Display() method and perform the realization of the flight and return to the very first welcome screen.

An example of a runtime of the UI is provided at the Appendix.

e. **Time and Space O(.) Complexity**

For this part, you will derive the time and space* O(.) complexity of the full execution of your homework. The full execution consists of the following steps:

- Create a flight
- Register passengers
- Simulate the flight

In other words, you may assume that your homework will be used for only one flight and the time and space complexity of that process is required. Note that the space complexity should be no worse than O(N) provided that N is the number of passengers registered for a flight.

You will provide this part of your homework in a separate file, which includes derivations and explanations.

**Space Complexity is the complexity related to required memory space for the execution of a computer program. For example, storing a matrix of R rows and C columns has O(RC) complexity.*

f. **About Implementations**

You should implement **SimulateFlight** as a separate function and call it in the main function. You might simply implement UI within the main function. You should declare the capsule and aircraft classes in a file **ee.h** and implement them in a file **ee.cpp**. Moreover you should implement the simulator in a file **main.cpp** and include necessary headers.

You should implement only what is needed, no more, no less. You should not output anything unnecessary such as warnings, messages, etc. to the screen. You may want to debug your program while doing the homework and you may want to use such messages and warnings for your own purposes. However, you should clean the final implementation to be submitted from these kinds of debug or personal messages.

You might assume that no inappropriate inputs will be given to the system.

4. **Notice**

As a candidate engineer, you are expected to deliver a fully functional homework. In business life, no one would pay for an ill-working program. So, you should check your work and debug your program using different inputs and examine the corresponding outputs carefully in order to expose any unpredicted or hidden errors in your code. In addition, delivering a product/program with an undesired format may harm your reputation in both academic and business life.
As programmers, you should also add comments to your code for your colleagues to understand. You may not see this is an important issue; however, it becomes very important if your program exceed a certain size. You may be working as a programmer next year this time and may be continuing some other programmers work in your new position. So, keep your program understandable.

5. Submission

- Homework creation and sending process are covered in the recitations. You should use Code::Blocks IDE, and you should choose GNU GCC Compiler while you are creating your project. Name your project as "e1XXXXXX_HW2". You should **send the whole project folder and the part-e of the HW compressed** in a rar or zip file. File name of the submissions should be e1XXXXXX_HW2.rar. You **will not get** full score if you do not send your project folder as it is expected.
- Your C++ program must follow object oriented principles, with proper class and method usage and correctly structured private and public components. Your work will be graded on correctness, efficiency and clarity.
- You should comment your source code properly without including any extra unnecessary details.
- Late submissions are welcome, but penalized according to the following policy:
  - 1 day late submission: HW will be evaluated out of 70.
  - 2 days late submission: HW will be evaluated out of 50.
  - 3 days late submission: HW will be evaluated out of 30.
  - 4 or more days late submission: HW will not be evaluated.

We wish you success!

Appendix – Example runtime of a UI

"\>>" : New line in console screen; "_": space keystroke; ↩: enter keystroke

```
>>EE Tech Easy Everywhere Simulator
>>(1) New Flight
>>(2) Exit
>>1↩
>>Input Departure Latitude, Longitude and Flight Direction:
>>41_29_-1↩
>>Input Direction:
>>-1↩
>>EE Tech Easy Everywhere
>>(1) New Passenger
>>(2) Close Registration and Simulate the Flight
>>1↩
>>Name of the passenger:
>>Nat_Pagle↩
>>CID of the passenger:
>>204501↩
>>Destination (Lat. & Long):
>>25_-78↩
>>New passenger has been successfully registered!
>>EE Tech Easy Everywhere
>>(1) New Passenger
>>(2) Close Registration and Simulate the Flight
>>1↩
>>Name of the passenger:
>> Leeroy_Jenkins ↩
>>CID of the passenger:
>>56221111↩
>>Destination (Lat. & Long):
>>40_33↩
>>New passenger has been successfully registered!
>>EE Tech Easy Everywhere
>>(1) New Passenger
>>(2) Close Registration and Simulate the Flight
>>1↩
>>Name of the passenger:
>>Harrison_Jones↩
>>CID of the passenger:
>>5543154↩
>>Destination (Lat. & Long):
>>49_2↩
>>New passenger has been successfully registered!
>>EE Tech Easy Everywhere
>>(1) New Passenger
```

```
>>(2) Close Registration and Simulate the Flight
>>2↩
>>The registration has been closed and the aircraft to take off is:
>>========================
>>EEAircraft
>>Departure:    +41, +29
>>Direction:    -1
>>========================
>>========================
>>Name:         Leeroy Jenkins
>>CID:          56221111
>>Destination:  +40, +33
>>========================
>>========================
>>Name:         Nat Pagle
>>CID:          204501
>>Destination:  +25, -78
>>========================
>>========================
>>Name:         Harrison Jones
>>CID:          5543154
>>Destination:  +49, +2
>>========================
>>The simulation starts…
>>Dropped Capsule:
>>========================
>>Name:         Harrison Jones
>>CID:          5543154
>>Destination:  +49, +2
>>========================
>>Current Aircraft:
>>========================
>>EEAircraft
>>Departure:    +41, +29
>>Direction:    -1
>>========================
>>========================
>>Name:         Leeroy Jenkins
>>CID:          56221111
>>Destination:  +40, +33
>>========================
>>========================
>>Name:         Nat Pagle
>>CID:          204501
>>Destination:  +25, -78
>>========================
>>***
>> Dropped Capsule:
>>=========================
>>Name:          Nat Pagle
>>CID:           204501
>>Destination:   +25, -78
>>=========================
>>Current Aircraft:
>>=========================
>>EEAircraft
>>Departure:     +41, +29
>>Direction:     -1
>>=========================
>>=========================
>>Name:          Leeroy Jenkins
>>CID:           56221111
>>Destination:   +40, +33
>>=========================
>>***
>> Dropped Capsule:
>>=========================
>>Name:          Leeroy Jenkins
>>CID:           56221111
>>Destination:   +40, +33
>>=========================
>>Current Aircraft:
>>=========================
>>EEAircraft
>>Departure:     +41, +29
>>Direction:     -1
>>=========================
>>***
>>Simulation has been ended.
>>EE Tech Easy Everywhere Simulator
>>(1) New Flight
>>(2) Exit
>>2↩
```