BLG 252E - Object Oriented Programming Assignment #2

Due: April, 30th 23:55

Introduction

In this homework, you are going to implement a fictional space agency simulation. You will simulate a space agency that operates space missions. Each mission has an operating cost. Agency covers these costs by selling tickets to the passengers. Every mission has a probability to fail at launch, however, these failures are non fatal incidents. Agency only reschedules failed missions to a further date. However, agency has to pay operation cost of the mission.

For any issues regarding the assignment, please contact Alperen Kantarcı (**kantarcia@itu.edu.tr**).

Implementation Notes

Following implementation details are included in grading.

1. Please follow a consistent coding style (indentation, variable names, etc.) with comments.
2. You are **not** allowed to use STL containers.
3. You may (and should) implement any getter and setter methods when needed.
4. Allocate memories dynamically for lists/arrays and implement necessary destructors where the allocated memory parts are freed. Make sure that there is no memory leak in your code.
5. Make use of constants and call by reference appropriately.
6. Use cerr for printing error messages cout.
7. Print your messages to the terminal. Do not create a .txt file to print into.
8. Your codes will be tested with Calico. Therefore your outputs should be **exactly** same with the given example output. To make sure about your output correctness, please checkout the supplied "output.txt" file. All available print messages are available in the "output.txt".

Submission Notes

1. Your program should compile on Linux environment using g++ Agency.cpp Person.cpp Mission.cpp main.cpp. You can test your program on ITU's Linux Server using **SSH** protocol. Include all necessary header files to your code. Do not use precompiled header files and Windows specific header files and functions.
2. You should create "**Person.h**", "**Mission.h**", and "**Agency.h**" header files for the class declarations. You should create "**Person.cpp**", "**Mission.cpp**", and "**Agency.cpp**" for the method implementations.
3. "**Person.h**" and "**Person.cpp**" should contain **Person, Astronaut and Passenger** class definition and implementations together. "**Mission.h**" and "**Mission.cpp**" should contain **Mission** class definition and implementations. "**Agency.h**" and "**Agency.cpp**" should contain **Agency** class definition and implementations.

4. You should compress your files into an archive file named "<your_student_number>.zip". Do not include any executable or project files in the archive file. You should only submit necessary files. Also, write your name and ID on the top of each document that you will upload.

5. An example "main.cpp" file will be supplied for testing your implementations. Please note that your code will be further tested with different main functions as well.

6. Submissions are made through **only** the Ninova system and have a strict deadline. Assignments submitted after the deadline will not be accepted. If you send your homework via e-mail, you will not get any points. Don't wait until the last minute. Upload whatever you have, you can always overwrite it afterwards.

7. This is not a group assignment and getting involved in any kind of cheating is subject to **disciplinary actions**. Your homework should not include any copy-paste material (from the Internet or from someone else's paper/thesis/project).

1 Implementation Details

You are expected to implement five classes; **Person**, **Passenger**, **Astronaut**, **Mission**, and **Agency**. You have to implement the given attributes and methods below. However, to achieve a list/array functionality you can add attributes or methods to the classes.

1.1 Person

Private Attributes:

Person *has* a **name** (eg. Neil) and a **surname** (eg. Armstrong).

Methods:

1. **Constructor(s):** the constructor should *optionally* take the attributes.

1.2 Passenger

Passenger class is a subclass of the **Person** class.

Private Attributes:

Passengers have **cash** (eg. 1250), and **ticket** (a boolean that shows whether a passenger has a ticket).

Methods:

1. **Constructor(s):** the constructor should *optionally* take the attributes. Note that cash can't be negative. If it's given negative then cash should be set as zero and an error message should be shown. Error message will be the following: "Passenger cash can't be negative. It is set to 0."

2. **buyTicket:** when this method is invoked, it buys a ticket if the Passenger has enough cash to pay the ticket price. This method should take a ticket price (int) and return whether the person bought the ticket or not as a boolean.

1.3 Astronaut

Astronaut class is a subclass of the **Person** class.

Private Attributes:

Astronauts have **numMissions** (eg. 4) counter which shows number of successful flights that an astronaut completed.

Methods:

1. **Constructor(s):** the constructor should *optionally* take the attributes.
2. **completeMission:** when this method invoked, it will increment the **numMissions** attribute of the Astronaut by one.

1.4 Mission

Private Attributes:

Missions have **name** (eg. ST-21), **missionNumber** (eg. 2), **cost** (eg. 3500), **faultProbability** (eg. 20), **completed** (eg. true), **passengers** list/array and **crew** list/array. Missions also have a *static* **numMissions** variable which keeps total number of created missions.

- Mission name has a special naming convention. Each mission has an "XX-YY" name where X is an *uppercase* letter and Y is a *digit*. If the mission name does not satisfy this condition, an error message should be printed and the default name ("AA-00") should be set as mission name. You can use **string::substr** and **string::find** functions to parse the string.
- **faultProbability** should be an integer between 0-100 which represents the failure probability of the mission. If 100 is given, then the mission fails at every launch.
- **completed** should be a boolean that indicates the mission success. All successful missions are considered as completed, rescheduled and not executed missions are considered as not completed.
- **passengers** list/array should keep passengers that added to the mission.
- **crew** list/array should keep crew members that assigned to the mission.
- **numMission** is initialized once and it is initialized to 0. You can see the initialization in the given "main.cpp". This static variable should be increased by 1 at each mission object creation. After increasing the variable result should be assigned to **missionNumber** variable. For example first mission should increase the static variable from 0 to 1 and assign the 1 to itself.

Methods:

1. **Constructor(s):** the constructor should *optionally* take the attributes.
2. **operator+=:** overload += operator for adding a **Passenger** or an **Astronaut** to the mission passenger or crew list/array respectively. Passengers can be added only if they have a ticket. If they don't have a ticket, error message should be shown. Astronauts are directly added to the crew list/array. You can see the usage of the overloaded operator in the example main.cpp file.
3. **executeMission:** this method invokes a random number generator (std::rand) and if the number is bigger than fault probability then mission will be successful. It will print whether mission is successful or not. If mission is successful it will invoke **completeMission** method of each astronaut in the crew list/array. Then it will print each astronaut name with number of completed missions. In order to see the behaviour of the method, please check out the provided output. Finally, it will set the **completed** attribute according to the result of the mission and return the result as a boolean.
4. **calculateProfit:** this method gets a ticket price as an argument. It checks whether the mission is completed by checking the class attribute **completed**. If the mission is successful, then the function multiplies the ticket price with the number of passengers on board. Cost of the mission will be subtracted from the profit whether mission is successful or not. After that the method returns net profit which can be a positive or negative integer value.

1.5 Agency

Private Attributes:

- Agency has a **name** (eg. NASA), **cash** (eg. 25000), **ticketPrice** (eg. 1200), **completedMissions** list/array and **nextMissions** list/array.

Methods:

1. **Constructor(s):** the constructor should *optionally* take the attributes.
2. **addMission:** this method takes a mission as an argument and adds the mission to the agency's nextMissions list/array.
3. **executeNextMission:** this method takes the next mission from the nextMissions list/array and executes the mission. If the mission is successful, it should be added to completedMissions list/array and the profit of the mission should be added to the agency cash. Otherwise, the mission should be moved to the end of the nextMissions list/array. Failed mission profit also should be added to the agency cash.
4. **Overload operator≪:** by overloading operator≪ you should put information about the Agency to the out stream. It should contain name, current cash, ticket price, information (name, cost) of the nextMissions and completedMissions. For every mission you should print mission number, mission name and mission cost. For more information you can check out the given output.

Notice: An example main function will be supplied to you. You are expected to write your classes compatible with the given main function. Please also consider that your implementations will be tested with different inputs.

For the given "main.cpp" function, your code should give the following output. Your output should be exactly same with the given "output.txt" file. Be careful about newline and empty spaces in the "output.txt". Please note that because of the mission failure probability if you change the random number generator seed you may get a different output. You can see the expected output for seed 1773 as below.

```
(base) alperen@Titan2x:/storage/alperen/blg_252e_homeworks/homework2$ ./a.out
*** Agency creation ***
Ticket price can't be negative. It is set to 0.
Agency name: NASA, Total cash: 20000, Ticket Price: 0
Next Missions:
No missions available.
Completed Missions:
No missions completed before.
Agency name: NASA, Total cash: 20000, Ticket Price: 10000
Next Missions:
No missions available.
Completed Missions:
No missions completed before.
No available mission to execute!

*** Astronaut creation ***
Number of missions that astronaut completed can't be negative. It is set to 0.

*** Passenger creation ***

*** Moon mission creation ***

*** Venus mission creation ***
Given name does not satisfy the mission naming convention. Please set a new name!
Name of the mission: AA-00
Passenger Alperen Kantarcı does not have a valid ticket!

*** Mission creation finished ***
Agency name: NASA, Total cash: 20000, Ticket Price: 10000
Next Missions:
Mission number: 1 Mission name: MN-01 Cost: 5000
Mission number: 2 Mission name: VS-01 Cost: 35000
Completed Missions:
No missions completed before.

*** Lift off ***
MISSION MN-01 SUCCESSFUL!
Astronaut Neil Armstrong successfully completed 4 missions.
Astronaut Buzz Aldrin successfully completed 1 missions.
Astronaut Sally Ride successfully completed 1 missions.
Agency name: NASA, Total cash: 45000, Ticket Price: 10000
Next Missions:
Mission number: 2 Mission name: VS-01 Cost: 35000
Completed Missions:
Mission number: 1 Mission name: MN-01 Cost: 5000

*** Lift off ***
MISSION VS-01 FAILED!
Agency reschedules the mission.
Agency name: NASA, Total cash: 10000, Ticket Price: 10000
Next Missions:
Mission number: 2 Mission name: VS-01 Cost: 35000
Completed Missions:
Mission number: 1 Mission name: MN-01 Cost: 5000
```