EE 441 HOMEWORK #1

Gamers Database

Due: November 17, 2016, 23:59
For questions: ccakmak@metu.edu.tr
eylen@metu.edu.tr

Introduction

Welcome to EE Games Company! You have graduated from METU and it is the first day of your new job as a software developer. Our company provides users a platform to play a good selection of online multiplayer games, pretty much like Steam, Epic Games, Game Ranger and Origin. In EE Games, we come up with network solutions so that gamers from all around the world can get together and have a smooth gaming experience. We deal with issues related to match-making, connection, database control, etc.

Your first job here is to construct data structures for users and games and write a program in C++ that will function as described in the following pages. You will be using this work as a basis for solving further problems such as match-making and database control in the upcoming weeks.

The ***user*** data structure is a list of all the players keeping their personal details and the list of games in their accounts. The ***game*** data structure is a list of all the games of a user and the information related to that.

Assume that there are some users who already purchased and played the games. The information about these users are kept somewhere else but to be inserted to the user dataset. Additionally, in future there will be new users.

Description of the Classes

You need to create two classes; ***User*** class and ***Game*** class.

Game class will have the following attributes and methods:

- *Constructor*: Creates a *Game* object. Doesn't take any inputs.
- *SetID, SetScore, SetExperience*: Sets the values of the respective attributes of *Game* object.
- *GetID, GetScore, GetExperience*: Returns the values of the respective attributes of *Game* object.
- Game ID: The name of the game (A single capital letter). Default value: 'N'.
- Score: The user's score in that game. The default value is 1200.
- Experience: The number of times the user played the game.

REMARK: You may add other methods to this class as you think it is necessary.

User class will hold the following attributes:

- A list of games the user has in their account (*Game_Array*). This will be an array of *Game* objects. Assume that there are 5 different games in EE Games Company.
- User ID: A number between 100 and 999. It is expected that the maximum number of users is 10. Default value is 100, representing an empty/null user.
- The number of games the user has.

User class will have the following methods:

- *Constructor*: Creates a *User* object. Doesn't take any inputs. When a user is created, its default *Game_Array* is also created, which is an array of 5 *Game* objects that represents empty/null objects.

  REMARK: If we had used dynamic memory, we would create a *Game_Array* with NULL objects while constructing *User*. Since you have only studied static memory so far, we'd like you to implement it as follows: First construct an array of 5 *Game* objects, which have the default Game IDs 'N', meaning that the user hasn't bought any games yet. To be able to create an array of 5 Game objects, constructor of *Game* Class should not take any inputs.
- *Add_Game*: The user buys a new game. Find the first empty location in *Game_Array* (i.e. a *Game* object with the Game ID 'N') and change the Game ID to the newly bought game.
- *Add_Existing_Game*: Similar to *Add_Game*. Adds a game which the user has bought before. This time Experience and Score will not get default values.
- *AssignID*: Assign a given user ID to the user.
- *Get_Game_Data*: Explained below.

REMARK: You may add other methods to this class as you think it is necessary. As in the example of *Game* class, you may need to write set and get methods to access private data.

For both *User* and *Game* classes, you have to decide which methods/attributes are private and which are public.

Description of the Program

1) Create an array of 10 *User*s in your *main* function.
2) In main, create the following list of *User* objects, add them to the array of users and assign the given IDs to them.

Existing Players	ID
User_1	101
User_2	102
User_3	103

3) In *main,* add the given existing games for each user. Please note that the experience and score values are assigned randomly, so they may not be consistent.

User_1 - Game_Array	ID	Experience	Score
Game_1	C	34	1412
Game_2	A	14	1630
Game_3	B	2	1080

User_2 - Game_Array	ID	Experience	Rank
Game_1	A	23	1570
Game_3	C	11	1708

User_3 - Game_Array	ID	Experience	Rank
Game_2	B	60	1886

4) Write a function, which lets the programmer to add a newly bought game (i.e. using *Add_Game* method) for a specific user from terminal. The program will ask the programmer for the ID of the user and the name of the game, then output a message confirming that the given game is added to the given user's account.

5) Write a function, *Show_Players,* which takes a Game ID as input and prints out the ID, score and experience of each user who has that game. To be able to implement this, you will need to write an additional method (or multiple methods) *Get_Game_Data* in *User* class, which checks whether that game exists in the user's account or not, and retrieves the related data of the game e.g. *Score* and *Experience*.

6) Write a function *Play,* which takes two user IDs and a Game ID (Assume that all games are 1-to-1 for simplicity). This function simulates a match and **updates** the *Score* and *Experience* of both players upon match result. It prints out the updated score and experience of the winner and the loser, as well as stating who the winner is.

   The decision on the winner (i.e. the simulation of a match result) is given based on the following calculations:

   S_i = Score of user i

   S_j = Score of user j

   P_i : Probability of winning for user i

   P_j : Probability of winning for user j

   Assume that *Si > Sj*

$$\Delta p = \frac{Si - Sj}{1000}$$

$Pi = Pj + (\Delta p \times 0.8)$ and $Pi + Pj = 1$

You will first calculate P_i (or P_j, whichever you prefer). You will also write a function that outputs a random value between 1 and 100 (This can be done by using modular arithmetic on the output of *rand()* function of C++). Then by making use of P_i and the random number obtained, you can decide on whether user i wins the match or not.

Once the winner is decided, the scores of the winner and the loser will be updated as follows:

Swb: Score of winner before match,

Swa: Score of winner after match,

Slb: Score of loser before match,

Sla: Score of loser after match,

c: Constant (=1/5)

$$Swa = Swb + (\frac{2000 - Swb}{1000} \times (Slb - 1000) \times c)$$

$$Sla = Slb - (\frac{2000 - Swb}{1000} \times (Slb - 1000) \times c)$$

Example case: Si = 1600, Sj=1400 → Pi = %58

Assume that j has won the match → Swb = 1400, Slb = 1600

→ Swa = 1472, Sla = 1528

EXAMPLE OUTPUT 1: Function menu and *Show_Players* function

```
Welcome to Gamers Database and Match Simulation

Enter a function number
1- Display users having the same game
2- Simulate a match between two players
3- Add new game
1
Enter a valid game ID (Example games are A, B, C)
(Enter ESC to return function selection)
6
6 is not a valid game ID
Enter a valid game ID (Example games are A, B, C)
(Enter ESC to return function selection)
B
Game B
User ID         Experience      Score
101             2               1080
103             60              1886
Number of users is 2

Enter a function number
1- Display users having the same game
2- Simulate a match between two players
3- Add new game
k
k is not a valid function number
```

EXAMPLE OUTPUT 2: *Play* function

```
Enter a function number
1- Display users having the same game
2- Simulate a match between two players
3- Add new game
2
Enter first user ID
(Enter ESC to return function selection)
asd
asd is not a valid user ID
Enter first user ID
(Enter ESC to return function selection)
101
Enter second user ID
(Enter ESC to return function selection)
103
Enter a valid game ID (Example games are A, B, C)
(Enter ESC to return function selection)
A
User 103 does not have Game A
Enter a valid game ID (Example games are A, B, C)
(Enter ESC to return function selection)
B
Chance of user 101 is 17.76 %
Chance of user 103 is 82.24 %
User 101 wins
New score of user 101 is 1243
New score of user 103 is 1723
```

EXAMPLE OUTPUT 3: *Add_Game* function

```
Enter a function number
1- Display users having the same game
2- Simulate a match between two players
3- Add new game
1
Enter a valid game ID (Example games are A, B, C)
(Enter ESC to return function selection)
B
Game B
User ID         Experience      Score
101             3               1243
103             61              1723
Number of users is 2

Enter a function number
1- Display users having the same game
2- Simulate a match between two users
3- Add new game
3
Enter user ID
(Enter ESC to return function selection)
102
Enter game ID
(Enter ESC to return function selection)
B
Game B is added to user 102

Enter a function number
1- Display users having the same game
2- Simulate a match between two users
3- Add new game
1
Enter a valid game ID (Example games are A, B, C)
(Enter ESC to return function selection)
B
Game B
User ID         Experience      Score
101             3               1243
102             0               1200
103             61              1723
Number of users is 3

Enter a function number
1- Display users having the same game
2- Simulate a match between two users
3- Add new game
```

Regulations:

1. You should insert comments to your source code at appropriate places without including any unnecessary detail. Comments will be graded. You have to write to-the-point comments in your code, otherwise it would be very difficult to understand. If your output is wrong, the only way we can grade your homework is through your comments.

2. Use **Code::Blocks IDE** and choose GNU GCC Compiler while creating your project. Name your project as "e<student_ID>_HW1". Send the whole project folder compressed in a rar or zip file. You will not get full credit if you fail to submit your project folder as required.

3. Your C++ program should follow object oriented principles, including proper class and method usage and should be correctly structured including private and public components. Your work will be graded on its correctness, efficiency and clarity as a whole.

4. 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.

Good Luck!