CS 112 – Computer Science Fundamentals II
Assignment #3

Deadlines

This assignment is due on **Tuesday, October 27, 2020 @ 11:59 PM**

How to submit your work on the assignment:

Assignments will be accepted via Canvas as CPP and H files. Use the Multiple File submission tool in Canvas to submit all files. Make sure to include the CPP file containing your main() function that contains the tests of your code. Submit all the files in a single submission, and do NOT use ZIP.

It is expected AND REQUIRED that all the code you submit is your work and your work alone. Copying of any code between students is strictly prohibited! Also, all functions you write MUST have the correct name and headers that match the problems specifications. If the instructor's testing statements can't run your functions because of errors in the headers, that will result in major deductions in your assignment grade!

In this assignment, you will work with the **PlayerChar** class you defined in the Week 08 Lab.

See the Week 08 Lab assignment – if you did not complete the lab assignment, then you can finish the lab ON YOUR OWN. If you did NOT participate in the Week 08 Lab assignment on Friday, October 16, then you MUST fully complete that assignment ON YOUR OWN with NO COPYING OF CODE from any fellow student! If you DID participate in the Week 08 Lab assignment, you may use the code you developed with your team, but you MUST complete this assignment as YOUR OWN WORK with NO COPYING OF CODE from any fellow student!

Some additions to the PlayerChar class from the Week 08 Lab code

Part (a)
First, refactor your code to allow the number of "standard" player types to be increased and decreased more easily. if you used the number **5** in the declaration of the stdPlayer array, then change that to use a **const int** value named **NUM_TYPES** instead, and go through your code to make sure that **NUM_TYPES** is used appropriately throughout instead of the literal value of **5**.

Once you've done that, you will then add a sixth type of standard player. You should now be able to do so by just changing **NUM_TYPES** to have the value **6** and then adding to the **stdPlayer** array a sixth standard player with the following characteristics:

Strength = 650, Agility = 2.7, role = Dwarf

Part (b)

Define a mutator method **setName**, with a **string** argument, that changes the **name** of a **PlayerChar**.

Part (c)
Update your **increaseStrength** and **increaseAgility** methods (if necessary) to make sure that they can be passed negative integer values that will result in **strength** or **agility** going down by the percentage indicated. For example, if you have a PlayerChar with the identifier **fred**, then **fred.increaseStrength(– 10)** will result in Fred's strength going down by 10 percent. As before, make sure that the **strength** and **agility** cannot change (either up or down) by more than 10 percent in a single use of the method.

Also, if a value greater than 10 or less than –10 is passed to either method, then the value should change the appropriate value by only 10 percent. Do not set a minimum limit on the character's strength or agility.

Part (d)

Add a new private data member to your **PlayerChar** class, named **hp** (which stands for "hit points"). The value of **hp** will always be calculated, instead of entered by the player. The value of **hp** shall be initialized to the result of multiplying the player's current **strength** by the player's current **agility**. Use a **double** value for **hp**, which is the appropriate data type for **hp** to store the exact value of the product of **strength** and **agility**.

In your constructor methods, always initialize the value of **hp** by multiplying the character's **strength** and **agility**.

Part (e)

Now define a private method (a method whose headers in the **H** file are placed in the **private** area of the class instead of the **public** area, so that the user may never use it but the programmer who defines the class may use it). Name the mutator method **updateHP**. It will be overloaded with two versions of the method:

- If **updateHP** is called with no argument, then this will check to make sure that the current value of **hp** is less than or equal to the current **strength** times the current **agility**. If the value of **hp** is greater than **strength** times **agility**, then **hp** will be adjusted to be equal to **strength** times **agility**. If the **hp** value is already less than or equal to **strength** times **agility**, then the current **hp** value will remain unchanged.
- If **updateHP** is called with one argument of type **int**, then **hp** will be changed so that **int** value will be either added to **hp** (if a positive **int** value) or subtracted from **hp** (if a negative **int** value), with limits. Any attempt to increase **hp** to greater than **strength** times **agility** will instead set **hp** to be equal to **strength** times **agility**. Any attempt to decrease **hp** to less than 0.0 will instead set **hp** to 0.0.

Every time the **strength** or **agility** of a **PlayerChar** changes in the **increaseStrength** or **increaseAgility** methods, your code should be updated to immediately call the **updateHP** method (without an argument) to update the **hp** value as defined in that method. (You can call the **updateHP** method whenever the **strength** or **agility** go up, but in those cases the value of **hp** should be unchanged).

Part (f)

Define a private mutator method **restoreHP**, with no arguments, that resets the **hp** value to be **strength** times **agility**.

Be sure to write sufficient tests in your **main.cpp** file to test all the code! You do NOT need to turn in the **main.cpp** file, as the instructor will test your code using his own **main.cpp** file. This does mean, though, that your code needs to use the method names that exactly match the specifications above!

Use the Multiple File submission tool in Canvas to submit the **PlayerChar.cpp** and **PlayerChar.h** files. Submit all the files in a single submission, and do NOT use ZIP. It is expected AND REQUIRED that any code you create and submit (or modify and submit) is **your work and your work alone**. Copying of any code between students is strictly prohibited!