CS 112 Fall 2020 – Data Structures
Lab for Week #08

For this lab exercise, you will work in 2-person teams (a single 3-person team will be allowed is there are an odd total number of students in the lab). The point here is to have teams discuss the lab and offer each other support in making sure the IDE of your choice is working!

For this lab, **you will take turns** in being the "driver" (the team member sharing the screen with the IDE) and the "navigator" (the team member who is viewing the editing being done and collaborating in performing the Lab Exercises).

NOTE: If you are using an IDE other than NetBeans, you should ensure that the code you write and test will run in the NetBeans IDE. This rule will apply more strongly for individual homework assignments than for the weekly lab sessions, but understanding what adaptations you need to make to your code to run in NetBeans is a necessary component of your participation in this course. If incompatible code continues to be a problem, the instructor reserves the right to require the use of NetBeans for all assignments.

Lab Exercise

In this lab, teams will define a C++ class based on the **PlayerChar** class demonstrated in class in Week 5, Lecture 2. In that lecture, we defined the **PlayerChar** class for a role-playing game with the following data members:

```
string name;
int    strength;
double agility;
string role;
```

The lab exercise is to use and expand this as a full-fledged C++ class, which will have more methods associated with it – Constructors, Accessors, Mutators, and Other methods.

We will also introduce "standard" player characteristics, based on an array of type **PlayerType** (which is a **struct**) that defines the allowed **strength**, **agility**, and **role** values of a **PlayerChar**. After all, we don't want our role-playing characters to be too weak or too strong, but instead to have a good balance of **strength** and **agility** appropriate to their character types! The struct for player types shall be defined in **PlayerChar.h** as:

```
struct PlayerType {
    int    strength;
    double agility;
    string role;
}
```

For your allowed **PlayerType** values, create a constant array of size 5 of type **PlayerType**. Use these values:

```
const PlayerType stdPlayer[5] = {{700, 2.5, "Ogre"}, {300, 5.5, "Sprite"},
      {400, 4.5, "Elf"}, {600, 3.0, "Valkyrie"}, {500, 3.5, "Magician"}};
```

The possible roles for a **PlayerChar** will be limited to only the five roles listed above in the **stdPlayer** array.

With that, now populate your **PlayerChar.cpp** file and your **PlayerChar.h** file, containing the appropriate content, in the appropriate places, to meet these requirements.

PlayerChar.cpp will contain code, and **PlayerChar.h** will contain headers, for the following methods:

3 Constructor methods

- One Constructor method will have 2 parameters, both **string** values, for (in order) the **name** and the **role** of the character. The Constructor will set the **strength** and **agility** values for the character based on the **stdPlayer** array. If the **role** string passed to the Constructor does not match any of the standard **PlayerChar** roles, the **PlayerChar** will be created as an **Ogre**.
- One Constructor method will have 1 parameter, a string for the **name** of the PlayerChar. It will be created as an **Ogre** with the corresponding **strength** and **agility** values.
- One Constructor method will have no parameters, that will create an **Ogre** named **Fred**.

4 Accessor methods

- One for each characteristic of the PlayerChar, with these names:
  getName getStrength getAgility getRole

2 Mutator methods

- A mutator named **increaseStrength**, that expects an **int** value representing a percentage increase of the character's current strength (for example, a value of 5 increases the **strength** by 5 percent, while –5 decreases the strength by 5 percent). You method must check to ensure the following:
  * A character's **strength** must not increase (or decrease) more than 10 percent at a time.
  * A character's maximum **strength** will be capped at double the standard strength for a character with that role.
- A mutator named **increaseAgility**, with the same argument and rules as **increaseStrength**.

1 Other method

- A method named **printPlayer**, that writes a **PlayerChar**'s info to the screen. See the **print_player_info** method in the Week 05 Lecture 2 code for an example.

Write code in your **main.cpp** file (or similar file) that calls all these methods to test them.

It is NOT necessary to submit the **main.cpp** file – the instructor has a **main.cpp** with which to test your code. At the top of the **CPP** and **H** files, be sure to place comments with your names, as usual:

```
// CS 112 Fall 2020 – Week 08 Lab
// <Student_Name1> and <Student_Name2>
```

Substitute your own names for the **Student_Name** placeholders above.

When finished, submit the Week 08 Lab files to the Week 08 Lab assignment page. Use the Multiple File Submission tool in Canvas to submit the **CPP** and **H** files (except the main.cpp file, which does not need to be turned in).