CS 112 Fall 2020 – Data Structures
Lab for Week #10

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

A solution to last week's lab is posted for you to use, as you'll need this code for this week's lab. Download the provided **main.cpp**, **PlayerChar.cpp**, and **PlayerChar.h** files.

In online role-playing games, a particular type of player-character is called the Tank, whose job it is to draw the attention of "raid bosses" that a team of player-characters are trying to defeat. The tank can withstand large amounts of damage inflicted by the raid boss while the other team members battle the boss.

In this lab, you will create files named **TankPlayerChar.cpp** and **TankPlayerChar.h** in which you will define a class **TankPlayerChar** that will be a derived class from the base class **PlayerChar**. You will define methods in the derived class as specified below.

You will NOT modify or submit the **PlayerChar** files for this lab! You'll also use the supplied **main.cpp** file and add lines of code to test the methods you'll write for the **TankPlayerChar** subclass.

Perform the following tasks:

- At the top of the **TankPlayerChar.cpp** and **TankPlayerChar.h** files, document your module as follows based on your 2-person team:

  ```
  // CS 112 Fall 2020 – Week 10 Lab
  // <Student_Name1> and <Student_Name2>
  ```

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

- The **TankPlayerChar** will inherit all the characteristics of a generic **PlayerChar** object, and will add the following data members (placed in the **private** area of the **TankPlayerChar** class definition):
  aggro (short for "aggravation") which is an amount of aggression the Tank can "hold" in enticing a raid boss to attack only the Tank and not the other members of the team.
  stamina is the amount of time the Tank can withstand an attack by a raid boss before having to withdraw from the fight to be healed.

aggro will be stored as an **int** value, and **stamina** will be stored as a **double** value.
aggro will be initialized as a calculated value, the initial **strength** of the Tank divided by 2.
stamina will be initialized as a calculated value, the initial **strength** times **agility** divided by 2.

- The **TankPlayerChar** class will have three constructor methods that you will write. Each will take the same arguments as the three **PlayerChar** constructors, and will fully initialize the data members of both **PlayerChar** and **TankPlayerChar** classes. These constructors will call the corresponding **PlayerChar** constructor to set the initial values of the **PlayerChar** data members, and will also set the initial values of the **aggro** and **stamina** data members.

- The **TankPlayerChar** class will have two accessor methods **getAggro** and **getStamina** and two mutator methods **setAggro** and **setStamina** that you will will write for the new data members **aggro** and **stamina**.

- You will write a method named **printPlayer**, that writes a **TankPlayerChar**'s info to the screen, including all the data members in both **PlayerChar** and **TankPlayerChar** classes.

  Note that this will have the same name and arguments (that is, no arguments passed) as the existing **printPlayer** method in the **PlayerChar** class. This is called redefining a base method's method in a subclass, and allows a derived class to have methods with identical names and arguments as the base class, but to perform an enhanced version of the base class's method.
  Note that if you want to call the base class's method that has the same name as the derived class's method, you can use the scope operator to specify which method to call. For example,
  `PlayerChar::printPlayer()` calls the method in the **PlayerChar** class
  `TankPlayerChar::printPlayer()` calls the method in the **TankPlayerChar** class

- An operator method for the **==** comparison operator has been supplied for the **PlayerChar** class. You will write an operator method for the comparison operator **==** for the **TankPlayerChar** derived class that makes use of the **PlayerChar** class method. It should return **true** only if all the data member values are equal – **name**, **strength**, **agility**, **role**, **aggro**, and **stamina**.

When finished, submit the Week 10 Lab files to the Week 10 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).