COMP710: Studio Session 03 – Exercise:

EXERCISE: C++ – *Using OOP*

Create a Visual Studio Solution named "SS03" and C++ Project named "Using OOP" for this exercise.

Compile and test your program for this exercise as you progress through each of the steps. Fix any issues that arise before moving onto the next step!

1) Create a class named **`Player`**. Create a **`.h`** file for the class declaration, and a **`.cpp`** file for the class implementation. Add **`private`** member data (properties/fields) to the **`Player`** classes that store the player's *health*, *experience level*, and *power-up level*. Pick appropriate types for these data members. Add **`public`** accessor methods for each of these fields. Also, ensure the **`Get`** methods maintain **`const`** correctness. Remember to **`#include "player.h"`** at the top of the **`player.cpp`** file.

2) Ensure the **`player.h`** class declaration has preprocessor header guards.

3) Add a default *constructor* to the **`Player`** class that prints out "A Player object was created!" to the console. Use the constructor's member initializer list to set each of the class's data members to good default values. Also add a *destructor* to the **`Player`** class that prints out "A Player object was destroyed!".

4) Add a method to the **`Player`** class called **`PrintStats()`** which outputs the current player's statistics (health, experience level and power-up level) to the console. Remember to **`#include <iostream>`** at the top of the **`player.cpp`** file.

5) Add another **`.cpp`** file, named **`main.cpp`**. Inside this file, declare the **`main`** function. In the **`main`** function, declare a local stack instance of the **`Player`** class named **`testPlayer`**. Remember to **`#include "player.h"`** at the top of the **`main.cpp`** file.

6) Call the **`PrintStats`** method on the newly created **`testPlayer`** object instance.

7) In **`main`**, call the **`Set`** methods to mutate the **`testPlayer`** object by setting values for health, experience level and power-up level.

8) In **`main`**, call the **`Get`** methods individually, and print the results returned from each method call.

9) Next, in **`main`**, call the **`PrintStats`** method on the **`testPlayer`** object again.

10) In the **`main`** function, add a declaration of a pointer to a **`Player`** variable named **`pPlayer`**. Assign the pointer the null pointer value.

11) Next, allocate a **`Player`** object on the Free Store with **`new`**. Store the resulting allocation's address in the **`pPlayer`** pointer variable.

12) Call the **`Set`** accessor methods of **`pPlayer`** using the **`->`** operator, and then make a call to the **`PrintStats`** method.

13) At the end of the **`main`** function, deallocate the **`Player`** object (whose pointer is stored in the **`pPlayer`** pointer variable) from the Free Store using the **`delete`** command.

Once all steps are complete, reflect upon and contrast the different between a locally stored stack-based **`Player`** object (**`testPlayer`**), and one stored on the Free Store (**`*pPlayer`**). Using the Visual Studio 2017 debugger, breakpoints, stepping and watch window may help your investigation of the objects and reflection.

Finally, commit your program's source code to your individual SVN folder – include the **`.sln`**, **`.vcxproj`**, **`.cpp`** and **`.h`** files, and ensure you do not commit any build output files.