**Intro to C# - Assessment Requirements**

This subject will involve completing 4 exercises. We will start them in class time, but you may need to do additional work on them at home if you don’t get them completed in the allotted time. With the exception of Exercise 2 which is written, for each exercise:

* Create a separate Visual Studio solution containing a C# console project.
* Make sure your code is well commented before you submit.
* Submit the entire solution, complete with the project, and all source files. I recommend you just zip up the whole folder.
* Submit a build for each project, so after compiling make sure you copy the executable into its own folder and submit that in addition to the solution.
* Conduct testing and fill out a testing report containing at least 2 entries. A template for this report will be provided.

**Exercise 1**

Write a Sort() function and call it from your Main():

* The Sort() function should take in an array of ints as a parameter.
* It will not need to return anything.
* It should contain a bubble sort, implemented so that the outer loop is a while loop and the inner one is a for loop.

In the Main() after calling the Sort() function, loop through the array and print out the values to show that they were sorted.

Comment your code.

Complete testing and write a report about errors encountered while creating this project.

**Exercise 2**

Complete the following exercise on debugging. This task can be completed during the completion of any of the other exercises or programs created for this subject.

1. Run the Debugger  
   Upload a screenshot of Visual Studio running in debug mode.
2. Compiler Errors  
   Upload a screenshot of 3 different compiler errors.   
   Write a brief description of what each error means.
3. Reference Material  
   List at least one website that provides material to aid you in understanding the C# language.
4. Using the Debugger  
   Upload screenshots showing:  
    A breakpoint  
    A variable watch, and  
    The callstack

**Exercise 3**

Write an Actor class:

* Give it two constructors: one a default constructor with no parameters, the other should have a parameter for a Weapon.
* Give the Actor a Weapon member variable.
* Give it a name variable of type string.
* Give it a strength variable of type int.
* Give it a health variable of type int.
* Set all variables to default values.
* Give it a virtual Print() function that prints out a message about the creature, lists its name, its strength, health, the weapon it’s using, and the weapon’s stats.

Write a Monster class that inherits from Actor:

* Set the Actor’s variables to different values.
* Override the Print() function to print out a different message, but one that still displays all the variables.

Write a Goblin class that inherits from Monster:

* Set the Actor’s variables to different values for the Goblin.
* Override the Print() function to print out a message specific to the Goblin, but one that still displays all the variables.

Write a Skeleton and an Orc class that both inherit from Monster and set different values to the variables and override the Print() function with their own message.

Write a Weapon class:

* Give it a name variable of type string
* Give it a damage variable of type int
* Give it a range variable of type float
* Give it a category variable as type string (e.g. axe, sword, bow, etc)
* Set all variables to default values

Create an Arena class:

* Give it a constructor that will initialize the below arrays.
* Give the class an array of Weapons and initialize them with different data in the constructor.
* Give the class an array of Actors, making each of them a different type (Goblin, Skeleton, or Orc) in the constructor.
* As you create each Actor, give them one of the above Weapons through its constructor.
* Give it a PrintAll() function that will loop through all Actors and call their Print() function.

In your Main() create an instance of the Arena class and call it’s PrintAll() function to print everything out and verify that it all worked.

Comment your code.

Complete testing and write a report about errors encountered while creating this project.

**Exercise 4**

Create a program that writes the Unix time (seconds since epoch) to a text file along with the number of seconds since the program was last run (this will be 0 the first time the program runs).

Every time you execute the program, you will read the last time logged, calculate the time that has passed in seconds by comparing that time to the current time, and update the file with the new Unix time and the time since the program was run.

To get the Unit time you can use:

using System;

long unixTime = DateTimeOffset.UtcNow.ToUnixTimeSeconds();

For example: The first time you run the program, it might write out:

1603789295

0

This indicates that 1603789295 seconds have passed since January 1st, 1970. The zero on the second line indicates that no time has passed since the program was last run.

The next time you run the program it might print out:

1603789395

100

The Unix time has advanced by 100 seconds and the second line confirms that this is the amount of time that has passed since the last time the program was run.

Comment your code.

Complete testing and write a report about errors encountered while creating this project.