**CS-200 Fall 2014**

**LAB 11**

**Time Allowed: 2 Hours and 30 Minutes**

**Instructions:** You are allowed to discuss with your neighbors. But keep your voices down. Don’t standup and move around to discuss.

After lab you will have 72 hours to submit lab tasks on LMS with 40% deductions.

**Abstract Classes and Pure Virtual functions:**

This lab will extend on previous week’s lab on Inheritance. It will introduce to you the concept of abstract classes and how they can be used to write much more generalized code which and prevents repetition of code. We will see a new type of function called *pure virtual functions* which make a class an abstract class. Abstract classes represent abstract concepts in reality such as the concept of shape or tool which cannot be defined by their own self. Hence, it is vital to create derived classes which ‘complete’ the definition of abstract classes.

To demonstrate this concept, we will create an abstract class called Tool and use it to implement the popular game ‘Rock, Paper, Scissors’. For those who don’t know, Rock beats Scissors, Scissors beat Paper and Paper beats Rocks. That will suffice for us.

**Note:** It is not possible to create an instantiation of an abstract class

Write all your code in the files provided. You can make all your members and methods public.

**Task1:**

To begin with create a class called Tool with a single integer variable called ‘strength’ and a string variable called ‘type’ which will store the name of the tool. In the Tool class, create a function setStrength which takes an integer and sets the strength to that integer. Lastly, to make this an abstract class, define a pure virtual function fight like this:

***virtual bool fight(Tool \*t) = 0;***

It is important to pass the Tool object using a pointer which invoked a process known as ‘dynamic binding’ which is used when the exact type of an object is unknown.

Note that this function has no body and it has been a set a value zero. This tells the compiler that the function is a virtual function and that its definition has not been provided on purpose (because we will provide it in the derived classes).

**Task2:**

Create three more classes called Rock, Paper and Scissors which inherit from Tool. Create constructors which take a single integer argument for the classes in which you define the strength of each tool (remember that all objects of derived classes are also objects of base classes) using the ***setStrength*** function. Also, define the type of the tool, ‘rock’ for Rock, ‘paper’ for Paper and ‘scissor’ for Scissor.

**Task3:**

Now that we have our base code up, we can finally start to implement the ***fight*** function in our derived classes to complete our functionality and allow us to play the game. In our declarations of the ***fight*** function we do not need to type virtual again although typing it will not cause an error. Just the normal declarations will work. This function will take in a Tool object as argument. So the prototype will be:

***bool fight(Tool \*t);***

The actual definition for the function will be different for all three classes allowing us to call just one function on different objects and it will behave according to the object it was called on.

To implement this, when rock fights scissors, its strength is temporarily doubled and thus it is able to beat scissors. However, when it fights paper, its strength is reduced to half allowing paper to beat it. Extend this for all three classes. The function should return true if the tool beats its competitor and false otherwise.

To test your code, we have provided a main file. Make sure that you get the correct output. Good luck! ☺

Courtesy of OCW.