Week 10: Lecture 20

Pure Virtual Function and Abstract Class

Function overriding is a feature that allows us to have a same function in child class which is already present in the parent class. A child class inherits the data members and member functions of parent class, but when you want to override a functionality in the child class then you can use function overriding. It is like creating a new version of an old function, in the child class.

Pure Virtual Function and Abstract

The C++ way of declaring a pure virtual function is to put the expression = 0 in the class declaration.

In the previous example base class Shape member function area do not need any implementation because it is overriding in derived class.

If this is the case, the C++ language permits the programmer to declare the function a pure virtual function.

For example, if a member function double area() is being declared pure virtual, then its declaration in its class looks like

```
virtual double area() = 0;
```

A pure virtual function is sometimes called an abstract function, and a class with at least one pure virtual function is called an abstract class. The C++ compiler will not allow you to instantiate an abstract class. Abstract classes can only be subclassed: that is, you can only use them as base classes from which to derive other classes.

A class derived from an abstract class inherits all functions in the base class, and will itself be an abstract class unless it overrides all the abstract functions it inherits. The usefulness of abstract classes lies in the fact that they define an interface that will then have to be supported by objects of all classes derived from it.

```
#include <iostream>
using namespace std;

class Shape
{
protected:
      double width, height;
public:
      void set_data (double a, double b)
      {
      width = a;
      height = b;
      }
      virtual double area() = 0;
};

class Rectangle: public Shape
{
public:
      double area ()
      {
      return (width * height);
      }
};
```

```
class Triangle: public Shape
{
public:
     double area ()
     {
     return (width * height)/2;
     }
};

int main ()
{
Shape *sPtr;
Rectangle Rect;

sPtr = &Rect;

sPtr -> set_data (5,3);
cout << "Area of Rectangle is " << sPtr -> area() << endl;

Triangle Tri;

sPtr = &Tri;

sPtr -> set_data (4,6);
cout << "Area of Triangle is " << sPtr -> area() << endl;

return 0;
}
```

Output : Area of Rectangle is 15 Area of Triangle is 12