Lab No.9 Polymorphism and Virtual Functions

9.1 Objectives of the lab

Introducing the concepts of polymorphism such as

34 Basics of polymorphism
35 Virtual functions
36 Concrete classes
37 Abstract classes

9.2 Pre-Lab

9.2.1 Polymorphism

- A generic term that means 'many shapes'. In C++ the simplest form of Polymorphism is overloading of functions.
- Ability for objects of different classes to respond differently to the same function call
- Base-class pointer (or reference) calls a **virtual** function
  - C++ chooses the correct overridden function in object
- Attained by making a function virtual in base class.
- Keyword **virtual** is used in function declarator.
- Keyword **virtual** is not necessary to be used in derived classes. The overridden functions in derived classes are **virtual** automatically.

9.2.2 Example

```
#include<iostream.h>
class shape
{
public:
        virtual void draw()
        {
                cout<<"Shape class"<<endl;
        }
};
class triangle: public shape
{
public:
        void draw()
        {
```

```
                cout<<"Triangle class"<<endl;
        }
};
class rectangle: public shape
{
public:
        void draw()
        {
                cout<<"Rectangle class"<<endl;
        }
};

class circle: public shape
{
public:
        void draw()
        {
                cout<<"Circle class"<<endl;
        }
};

void main()
{
        shape   *sh;
        triangle        t;
        rectangle       r;
        circle          c;
        sh=&t; sh->draw();
        sh=&r; sh->draw();
        sh=&c;sh->draw();
}
```

Q. What is the effect of the following statement?

shape s;

9.2.3 Abstract and Concrete Classes

- Abstract classes
  - Sole purpose is to provide a base class for other classes
  - No objects of an abstract base class can be instantiated
    - Too generic to define real objects, i.e. **TwoDimensionalShape**
  - Can have pointers and references

- Concrete classes
  - classes that can instantiate objects
- Provide specifics to make real objects , i.e. **Square**, **Circle**
- An instance of abstract class can not be created
- A derived class of an abstract base class remains abstract unless the implementation of all the pure virtual functions is not provided.
  - Derived class inherits the pure virtual function
  - Any class having a pure virtual function is virtual
  - Override the pure virtual function in derived class. (Do not provide a 0 in declarator)

9.2.4 How to make a class abstract?

- Making abstract classes
- Declare one or more virtual functions as "**pure**" by initializing the function to zero

  virtual double earnings() **const** = 0;
- A class with no pure virtual function is a concrete class.

9.3 In-Lab

9.3.1 Activity

Define an abstract base class **shape** that includes protected data members for area and volume of a shape, public methods for computing area and volume of a shape (make these functions **virtual**), and a display function to display the information about an object. Make this class abstract by making display function pure virtual.

Derive a concrete class **point** from the **shape** class. This **point** class contains two protected data members that hold the position of **point**. Provide no-argument and 2-argument constructors. Override the appropriate functions of base class.

Derive a class **Circle** publicly from the **point** class. This class has a protected data member of **radius**. Provide a no-argument constructor to initialize the fields to some fixed values. Provide a 3-argument constructor to initialize the data members of **Circle** class to the values sent from outside. Override the methods of base class as required.

Derive another class **Cylinder** from the **Circle** class. Provide a protected data member for height of cylinder. Provide a no-argument constructor for initializing the data members to default values. Provide a 4-argument constructor to initialize x- and y-coordinates, radius, and height of cylinder. Override the

methods of base class.

Write a driver program to check the polymorphic behavior of this class.

9.4 Home-Lab

9.3.2 Activity

Create a class called **publication** that stores the **title** (**char** array) and **price** (float) of a publication. From this class derive two classes: **book**, which adds a page count (type **int**) and tape, which adds a playing time in minutes (type **float**). Each of the three classes should have a getdata() function to get its data form the user at the keyboard and a putdata() function to display the data.

Write a main program that creates an array of pointers to publication. In a loop, ask the user for data about a particular type of book or tape to hold the data. Put the pointer to the object in the array. When the user has finished entering the data for all books and tapes, display the resulting data for all the books and tapes entered, using a for loop and a single statement such as

```
pubarr[j]->putdata();
```

To display the data form each object in the array.

9.5 References:

24 Class notes

25 **Object-Oriented Programming in C++ by *Robert Lafore***

26 **How to Program C++ by *Deitel & Deitel***