# Assignment No. 07

- **TITLE -** Design a base class shape with two double type values and member functions to input the data and compute_area() for calculating area of shape. Derive two classes: triangle and rectangle. Make compute_area() as an abstract function and redefine this function in the derived class to suit their requirements. Write a program that accepts dimensions of triangle/rectangle and displays calculated area. Implement dynamic binding for given case study.

## THEORY –

This C++ program demonstrates the concept of **abstraction**, **inheritance**, and **runtime polymorphism (dynamic binding)** using a base class Shape and two derived classes Triangle and Rectangle. The Shape class contains two double-type data members, length and breadth, along with a member function input() to take user input for these values. It also declares a **pure virtual function** compute_area(), making Shape an **abstract base class** that cannot be instantiated directly. This setup enforces that all derived classes must provide their own implementation of the area calculation.

The derived classes Triangle and Rectangle override the compute_area() function to calculate the area according to their respective formulas:

- For a triangle, the area is 0.5 * length * breadth.
- For a rectangle, the area is length * breadth.

This overriding showcases **function overriding with virtual functions**, allowing each shape type to compute its area in its own way.

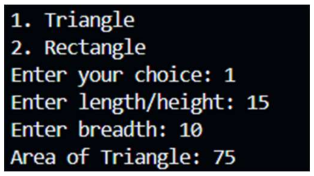
In the main() function, a pointer of type Shape* is used to achieve **dynamic binding**. Based on the user's choice, the pointer is assigned to either a Triangle or a Rectangle object at runtime. The program then invokes input() and compute_area() using the base class pointer, and the correct version of compute_area() executes depending on the actual object type. This is a classic demonstration of **polymorphism**, where the same function call behaves differently depending on the object type it refers to.

## CODE –

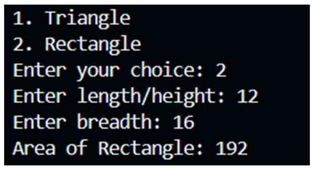
```
#include <iostream>
using namespace std;

class Shape {
    public:
        double length, breadth;
        void input() {
            cout << "Enter length/height: ";
            cin >> length;
            cout << "Enter breadth: ";
```

```
                cin >> breadth;
            }
            virtual double compute_area() = 0;
};

class Triangle : public Shape{
    public:
        double compute_area() override {
            return 0.5 * length * breadth;
        }
};

class Rectangle : public Shape{
    public:
        double compute_area() override {
            return length * breadth;
        }
};

int main(){
    Shape *s;
    int ch;
    cout<<"1. Triangle\n2. Rectangle\nEnter your choice: ";
    cin>>ch;
    if(ch==1){
        s = new Triangle();
        s->input();
        cout<<"Area of Triangle: "<<s->compute_area()<<endl;
        delete s;
    }
    else if(ch==2){
        s = new Rectangle();
        s->input();
        cout<<"Area of Rectangle: "<<s->compute_area()<<endl;
        delete s;
    }
    else{
        cout<<"Invalid choice!"<<endl;
    }
    return 0;
}
```

- **OUTPUT –**



```
1. Triangle
2. Rectangle
Enter your choice: 1
Enter length/height: 15
Enter breadth: 10
Area of Triangle: 75
```



```
1. Triangle
2. Rectangle
Enter your choice: 2
Enter length/height: 12
Enter breadth: 16
Area of Rectangle: 192
```

**Conclusion –**
In this assignment we have studied about implementing abstract class using pure virtual functions in C++. We used arrow pointers for resolving the ambiguity between functions and defined them in different scopes. We've tried multiple methods while implementing this assignment to get core knowledge about abstract classes and pure virtual functions.