Object Oriented Programming (CT-260)
Lab 06

Introduction to polymorphism

Objectives

The objective of this lab is to familiarize students with the concept of polymorphism in object oriented programming. By the end of this lab, students will be able to understand and implement function and operator overloading in C++ language.

Tools Required

DevC++ IDE / Visual Studio / Visual Code

Course Coordinator –
Course Instructor –
Lab Instructor –
Prepared By Department of Computer Science and Information Technology
NED University of Engineering and Technology

INTRODUCTION TO POLYMORPHISM

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

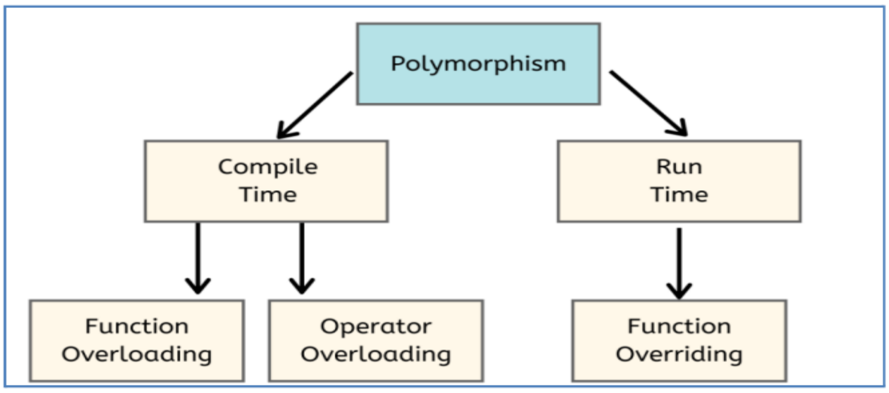
Real World Example:

A real-life example of polymorphism is that a person at the same time can have different characteristics. A man at the same time is a father, a husband, an employee, so the same person possesses different behavior in different situations. This is called as polymorphism. Polymorphism is considered as one of the important features of Object-Oriented Programming.

TYPES OF POLYMORPHISM:

In C++ polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism



Compile time Polymorphism

This type of polymorphism is achieved by function overloading or operator overloading.

Function Overloading:

- When there are multiple functions with same name but different parameters then these functions are said to be overloaded.
- Functions can be overloaded by a change in the number of arguments or/and change in the type of arguments.

Example Code for Function Overloading:

```
// C++ program for function overloading
#include<iostream>
 using namespace std;
class overload {
public:
// function with 1 int parameter
        void func(int x) {
            cout << "value of x is " << x << endl;
        }
// function with same name but 1 double parameter
        void func(double x) {
                cout << "value of x is " << x << endl;
        }
// function with same name and 2 int parameters
        void func(int x, int y) {
                cout << "value of x and y is " << x << ", " << y << endl;
        }
};
int main( ) {
        overload obj1;
// Which function is called will depend on the parameters passed. The first 'func' is called
        obj1.func(7);
// The second 'func' is called
        obj1.func(9.132);
// The third 'func' is called
        obj1.func(85,64);
        return 0;
}
```

Sample Run:

```
value of x is 7
value of x is 9.132
value of x and y is 85, 64
```

Operator Overloading

In C++, we can change the way operators work for user-defined types like objects and structures. This is known as **operator overloading**.

Syntax for C++ Operator Overloading

To overload an operator, we use a special operator function. We define the function inside the class or structure whose objects/variables we want the overloaded operator to work with.

```
class className {
  ... .. ...
  public
    returnType operator symbol (arguments) {
      ... .. ...
    }
  ... .. ...
};
```

Here,

- returnType is the return type of the function.
- operator is a keyword.
- symbol is the operator we want to overload. Like: +, <, -, ++, etc.
- arguments is the arguments passed to the function.

Example Code for Function Overloading:

Suppose we have created three objects c1, c2 and result from a class named Complex that represents complex numbers.
Since operator overloading allows us to change how operators work, we can redefine how the + operator works and use it to add the complex numbers of c1 and c2 by writing the following code:

```
// C++ program to overload the binary operator +
// This program adds two complex numbers
#include <iostream>
using namespace std;
class Complex {
  private:
   float real;
   float imag;

  public:
   // Constructor to initialize real and imag to 0
   Complex( ) : real(0), imag(0) { }

   void input( ) {
      cout << "Enter real and imaginary parts respectively: ";
      cin >> real;
      cin >> imag;
   }

   // Overload the + operator
   Complex operator + (const Complex& obj) {
      Complex temp;
      temp.real = real + obj.real;
      temp.imag = imag + obj.imag;
      return temp;
   }
```

Run time Polymorphism

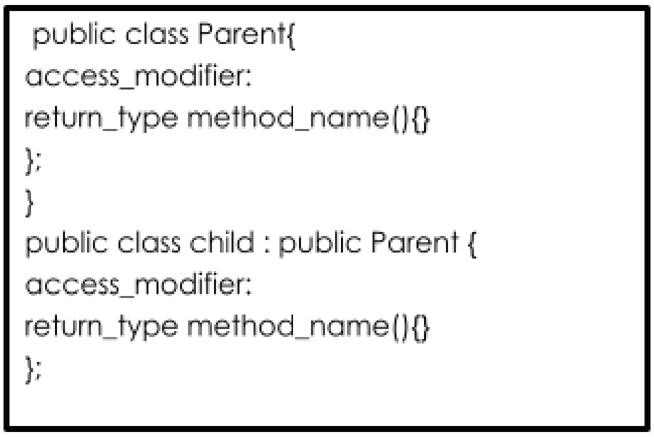
This type of polymorphism is achieved by Function Overriding.

Function Overriding

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.
- To override a function you must have the same signature in the child class.

Syntax for Function Overriding:



```
 public class Parent{
access_modifier:
return_type method_name(){}
};
}
public class child : public Parent {
access_modifier:
return_type method_name(){}
};
```

Example Code for Function Overriding:

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

```
class BaseClass {
public:
        void disp( ){
               cout<<"Function of Parent Class";
        }
};
class DerivedClass: public BaseClass{
public:
        void disp( ) {
                cout<<"Function of Child Class";
        }
};
int main( ) {
        DerivedClass obj = DerivedClass( );
        obj.disp( );
        return 0;
}
```

```
Sample Run:
Function of Child Class
```

Note: In function overriding, the function in parent class is called the overridden function and function in child class is called overriding function.

Function Overriding - during runtime

```
Class a{
public:
    virtual void display( ){ cout << "hello"; }
};

Class b:public a{
public:
     void display( ){ cout << "bye";}
};
```

Example code:

```
// Function Overriding
#include<iostream>
using namespace std;
class BaseClass{
public:
        virtual void Display( ){
                cout << "\nThis is Display( ) method of BaseClass";
        }
        void Show( ){
                cout << "\nThis is Show() method of BaseClass";
        }
};

class DerivedClass : public BaseClass {
public:
        // Overriding method - new working of base class display method
        void Display( ){
```

```
                    cout << "\nThis is Display( ) method of DerivedClass";
          }
};

// Driver code
int main( ){
          DerivedClass dr;
          BaseClass &bs = dr;
          bs.Display( );
          dr.Show( );
}
```

Sample run

```
This is Display() method of DerivedClass
This is Show() method of BaseClass
```

Exercise

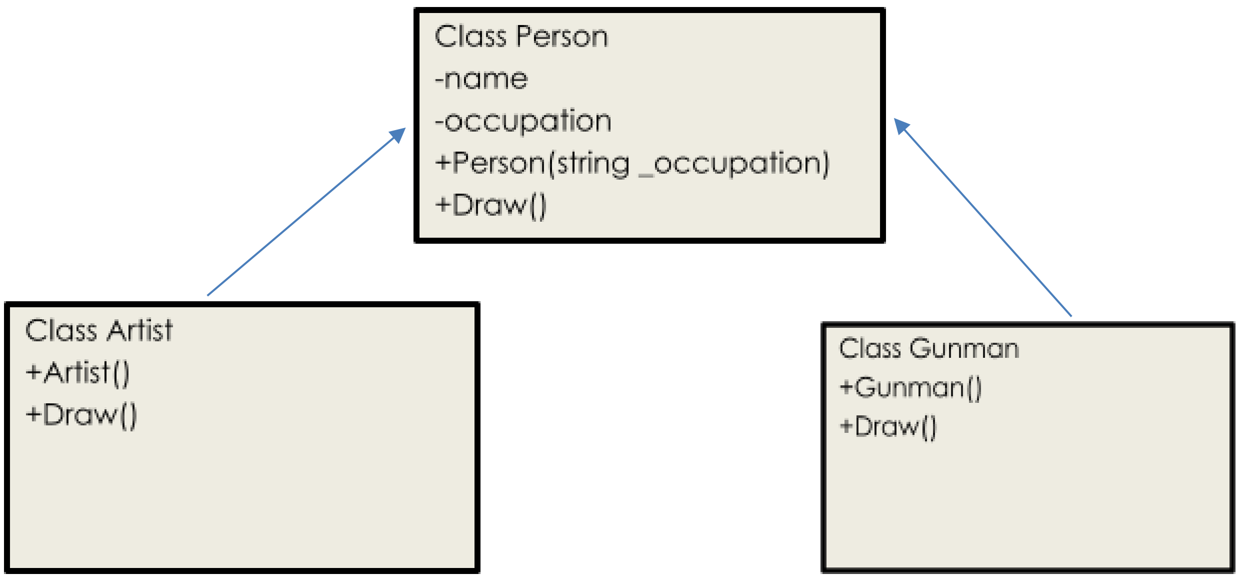
1. Create a class named Shape. All our shapes will be inherited from this class. It will contain the following data members and functions: numberOfSides, area, parametrized constructor, accessors and mutators for the data members.Create classes called Rectangle, Circle and Triangle, which are all inherited from the class Shape. Create a class called Square which is inherited from Rectangle. The derived classes will have the following members:
   Rectangle: length, width, parameterized constructor, generateArea( ) – should place the result in area.
   Circle: radius, parameterized constructor, generateArea( ) – should place the result in area
   Triangle: height, base, parameterized constructor, generateArea( ) – should place the result in area (Area = height*base/2)
   Square:
   - It should have a parameterized constructor that takes one side as input. The constructor should call the constructor for the Rectangle class with that value as parameters.
   - checkSides( ); - checks if both sides are equal. Sides are inherited from Rectangle.
   - generateArea( ) – should place the result in area

   You must make use of parameterized constructors to initialize the values.
1. Create a class called **Calculator** that has three private member variables Num1, Num2, Num3. In this class, you have to overload the functions for addition and multiplication such that they take two and three inputs respectively. You also have to make methods for subtraction and division. **For example**: add(1,2) and add(1,2,3) similarly for multiply, it would be multiply(1,2) and multiply(1,2,3). You may ask the user for input at the time of object creation. Afterwards just demonstrate how the functions are being called.
2. Create a class called **Vector** which represents a two-dimensional vector with **x** and **y** components. The class should have the following member functions:
   - A constructor that initializes the x and y components of the vector.
   - An overloaded operator + that adds two Vector objects and returns a new Vector object.
   - An overloaded operator - that subtracts two Vector objects and returns a new Vector object.

- An overloaded operator * that multiplies a Vector object by a scalar value and returns a new Vector object.
- An overloaded operator / that divides a Vector object by a scalar value and returns a new Vector object.
- A member function magnitude that returns the magnitude of the vector.

In the main function, create two Vector objects and demonstrate the use of all the overloaded operators.

4. Create the classes following the diagram given below. Keep the following things in mind:
   - When an object of Artist is created, the value “artist” will be set to occupation.
   - When an object of Gunman is created, the value “gunman” will be set to occupation.
   - Person::Draw() will print out “A person can draw in many ways”
   - Artist::Draw() will print out “An artist can draw with a paint brush”
   - Gunman::Draw() will print out “A gunman draws a gun to shoot”
   - Write a test code by creating an array of pointers of type Person. Dynamically create objects of each class and store address in the array. After that call Draw function for each object.


Class Person
-name
-occupation
+Person(string _occupation)
+Draw()
Class Artist
+Artist()
+Draw()
Class Gunman
+Gunman()
+Draw()