National University
of computer and emerging sciences

Lab 13

Items	Description
Course Title	Object Oriented Programming
Lab Title	Classes (Polymorphism)
Duration	3 Hours
Tools	Eclipse/ C++
Objective	To get familiar with the use of different concepts in classes in c++

Lab 13

Polymorphism

Definition

Polymorphism is an OOP concept that allows one interface to be used for different types of objects. In C++, polymorphism comes in two forms:

1. **Compile-time Polymorphism** (Static): Achieved through function overloading and operator overloading.
2. **Run-time Polymorphism** (Dynamic): Achieved through inheritance and virtual functions.

Function Overloading

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

```
class Math {
public:
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
};

int main() {
    Math math;
    cout << "Integer Addition: " << math.add(5, 10) << endl; //
Calls int version
    cout << "Double Addition: " << math.add(3.5, 4.5) << endl; //
Calls double version
    return 0;
}
```

National University
of computer and emerging sciences

What is Function Overriding?

Function overriding occurs when a **derived class** provides its own version of a function that is already defined in the **base class**. The function in the derived class must have the **same name, parameters, and return type** as the one in the base class.

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

class Animal {
public:
    void sound() { // Not virtual
        cout << "Animal makes a sound" << endl;
    }
};

class Dog : public Animal {
public:
    void sound() { // Overrides the base class function
        cout << "Dog barks" << endl;
    }
};
```

```
int main() {
    Dog dog;
    dog.sound(); // Calls Dog's version -> Output: Dog barks

    Animal animal = dog; // Object slicing occurs
    animal.sound(); // Calls Animal's version -> Output: Animal
makes a sound
    return 0;
}
```

Binding

Definition

Binding refers to the association of a function call to its corresponding code (implementation). There are two types of binding:

1. **Early Binding (Static Binding)**
    - Occurs at compile-time.
    - Used with normal functions and non-virtual member functions.
2. **Late Binding (Dynamic Binding)**
    - Occurs at runtime.
    - Used with **virtual functions**, enabling polymorphism.

Types of Binding

a. Early Binding

- The compiler determines the function to be called based on the object type at compile-time.
- Used for normal member functions or when a function is not declared as `virtual`.


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

class Animal {
public:
    void sound() { // Non-virtual function
        cout << "Animal makes a sound" << endl;
    }
};

class Dog : public Animal {
public:
    void sound() { // Overrides base class function
        cout << "Dog barks" << endl;
    }
};
int main() {
    Animal animal;
    Dog dog;
    animal.sound(); // Calls Animal's sound
    dog.sound();    // Calls Dog's sound

    return 0;}
```

National University
of computer and emerging sciences

Lab Task

Task 1: Function Overriding

Objective: Create a base class `Shape` with a `draw()` function and a derived class `Circle` that overrides this function.

Instructions:

1. Create a base class `Shape` with a function `draw()` that outputs `"Drawing shape"`.
2. Create a derived class `Circle` that overrides the `draw()` function to output `"Drawing Circle"`.
3. In the `main()` function:
    - Create an object of the `Circle` class and call the `draw()` function.

Task 2: Demonstrate Object Slicing with Function Overriding

Objective: Demonstrate the effect of object slicing when assigning a derived class object to a base class object in the context of function overriding.

Instructions:

1. Create a base class `Vehicle` with a `startEngine()` function that prints `"Vehicle engine started"`.
2. Create a derived class `Car` that overrides the `startEngine()` function to print `"Car engine started"`.
3. In the `main()` function:
    - Create a `Car` object and call `startEngine()`.
    - Assign the `Car` object to a base class `Vehicle` object and call `startEngine()` using the base class object.

National University
of computer and emerging sciences

Task 3: Compile-Time Polymorphism with Function Overloading

Objective: Show how function overloading works in C++ to achieve compile-time polymorphism.

Instructions:

1. Create a class `Printer` with two overloaded functions `print()`:
    - One function that takes an integer parameter and prints `"Integer: <value>"`.
    - Another function that takes a string parameter and prints `"String: <value>"`.
2. In the `main()` function:
    - Create an object of the `Printer` class and call both overloaded `print()` functions (one with an integer and one with a string).