TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: - 06

Title: - Virtual Functions

Submitted by: -

Name: - Gaurab Pandey

Roll NO: - 15

Submitted To: -

Department Of

Checked by: -

Date of submission: -

Objective:

To understand the concept of virtual functions in C++ and implement runtime polymorphism using base and derived classes.

Theory:

In C++, a virtual function is a member function in a base class that can be overridden in a derived class. It is used to support runtime polymorphism, where the function to be invoked is determined at runtime, not compile time. This helps in achieving dynamic dispatch when using base class pointers or references.

A virtual function is declared using the virtual keyword in the base class. When a base class pointer points to a derived class object and a virtual function is called, the derived class version is executed.

Key Concepts:

- Base Class: The class containing the virtual function.
- Derived Class: A class that inherits the base class and overrides the virtual function.
- Pointer to Base Class: Used to access the object of derived class to show runtime polymorphism.
- Virtual Table (V-Table): An internal mechanism that supports dynamic dispatch.

Syntax:

```
class Base {
public:
virtual void display() {
cout << "Base class display" << endl;
}
 };
 class Derived : public Base {
 public:
void display() override {
 cout << "Derived class display" << endl;
}
};
```

1.

Base Class and Virtual Function

The base class is the parent class that defines a function as virtual, allowing it to be overridden by derived classes. Declaring a function virtual tells the compiler to support runtime polymorphism for that function.

Syntax:

```
class Base {
 public:
virtual void show() {
cout << "Base class show function" << endl;
}
};
```

2. Derived Class and Function Overriding

A derived class inherits from the base class and overrides the virtual function using the same name and signature. This allows different behaviors for different derived classes.

Syntax:

```
class Derived : public Base {
 public:
 void show() override {
cout << "Derived class show function" << endl;
 }
 };
```

3. Pointer to Base Class and Runtime Polymorphism

When a base class pointer or reference points to a derived class object, and a virtual function is called, the derived class version of the function is executed. This is known as runtime polymorphism or dynamic dispatch.

Syntax: Base* ptr;

```
 Derived d;

 ptr = &d;

 ptr->show(); // Output: "Derived class show function"
```

Exercise 1: Basic Virtual Function

1. Create a base class Base with a virtual method display().

2. Create a derived class Derived that overrides the display() method.

3. Implement a main() function where you create a Base pointer pointing to a Derived object and call the display() method.

Program:

```
#include<iostream>
using namespace std;
class base{
    public:
    virtual void display () {
        cout<<"Displayed in base class";
    }
};
class derived : public base {
    public:
    void display() override {
        cout<<"Displayed in derived class";
    }
};
int main() {
    base *b;
    derived d;
    b=&d;
    b->display();
    return 0;
}
```

Output:

```
Displayed in derived class
--------------------------------
```

Exercise 2: Virtual Destructor

1. Create a base class Shape with a virtual destructor.

2. Create a derived class Circle that has a constructor and destructor.

3. Implement a main() function to demonstrate the use of virtual destructors by creating a Shape pointer pointing to a Circle object.

Program:

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

class shape {
public:
    virtual ~shape() {
        cout << "Destructor of class shape." << endl;
    }
};

class circle : public shape {
public:
    circle() {
        cout << "Constructor in class circle." << endl;
    }
    ~circle() {
        cout << "Destructor in class circle." << endl;
    }
};

int main() {
    shape* s = new circle();
    delete s;
    return 0;
```

Output

```
Constructor in class circle.
Destructor in class circle.
Destructor of class shape.
```

Discussion:

In this lab, we explored the concept of virtual functions in C++, which are essential for achieving runtime polymorphism. By declaring a function as virtual in the base class, we ensure that the correct derived class function is called based on the object's actual type, even when accessed through a base class pointer or reference.

Conclusion:

Understanding virtual functions is crucial for implementing polymorphism in C++. Through this lab, we learned how to:

- Define virtual functions to enable dynamic method binding.
- Override virtual functions in derived classes.
- Use pure virtual functions to create abstract classes.

These concepts form the foundation for designing flexible and reusable object-oriented systems. Mastery of virtual functions allows developers to write more efficient and adaptable code, making it easier to extend functionality in future developments.