

Himalaya College of Engineering

**Advanced C++ Programming Lab Report**

Lab 7: Virtual Functions in C++

**Prepared By :** Monsoon Sapkota (HCE081BEI022)

**Subject :** Object-Oriented Programming (OOP)

**Program :** Bachelor of Electronics, Communication and Information Engineering

**Institution :** Himalaya College of Engineering

**Date :** July 04, 2025

**OBJECTIVE**

➢ To understand the concept of Virtual Functions in C++.

# BACKGROUND THEORY

**Virtual Functions:**

Virtual functions in C++ are a fundamental concept in Object-Oriented Programming (OOP) that enable **runtime polymorphism**. They allow you to define a function in a base class that can be overridden by derived classes and then call the correct overridden version of the function based on the actual type of the object at runtime, even if you're using a pointer or reference to the base class.

**Types of Virtual Functions:**

1. **Pure Virtual Functions:** These are the types of virtual functions that do not include any declaration within it rather it is used to further override by base classes. These functions when in parent class themselves do not have any operation/execution in the code.

**Syntax:** class Base { public:

virtual void functionName() = 0; //Pure Virtual function declaration };

1. **Virtual Functions:** These are the general virtual functions which may or may not have any declaration and are meant to be overridden by child classes where the actual use of virtual function is implemented.

**Syntax:** class Base { public:

virtual void functionName() {

// Virtual function declaration

};

**Rules of Using Virtual Functions**

1. Virtual functions cannot be declared as static: Since static functions belong to the class rather than an object instance, they do not support runtime polymorphism and therefore cannot be virtual.
2. The function signature (prototype) of the virtual function must be identical in both the base and derived classes. This ensures proper overriding and correct function dispatch at runtime.
3. Virtual functions are typically accessed through pointers or references to the base class. This mechanism enables runtime polymorphism, allowing the correct derived class function to be invoked.
4. Virtual functions can be declared as friend functions of other classes. However, friend functions are not members of the class and cannot themselves be virtual.
5. Overriding a virtual function in the derived class is optional. If the derived class does not override the virtual function, the base class version is called at runtime.
6. Constructors cannot be virtual. Object construction happens before the object type is fully established, so virtual dispatch is not possible. However, destructors can and should be declared virtual to ensure proper cleanup of derived class objects when deleted through base class pointers.

**Example:** #include <iostream> using namespace std; class Shape { public:

virtual void draw() = 0; // Pure virtual function

}; class Circle : public Shape { public:

void draw() override { cout << "Drawing a Circle" << endl;

} }; class Rectangle : public Shape { public:

void draw() override { cout << "Drawing a Rectangle" << endl; } }; int main() {

Shape\* shape1 = new Circle(); Shape\* shape2 = new Rectangle(); shape1->draw(); shape2->draw(); delete shape1; delete shape2; return 0; }