**Objectives:**

* To gain an understanding of Virtual Functions in C++.

**Tools and Libraries Used:**

* Programming Language: C++
* IDE: G++
* Libraries: include <iostream>, include <string>

**Theory:**

Virtual Functions:

Virtual functions are a core concept of Object-Oriented Programming (OOP) in C++ that enable runtime polymorphism. They allow a function defined in a base class to be overridden in derived classes. When a base class pointer or reference points to a derived class object, the correct overridden function (from the derived class) is executed at runtime.

**Types of Virtual Functions:**

A. Pure Virtual Functions:

These functions have no implementation in the base class and must be overridden in derived classes. A class containing at least one pure virtual function becomes an abstract class.

Syntax:

class Base {

public:

virtual void functionName() = 0;

};

B. Virtual Functions:

These are regular virtual functions that may have an implementation in the base class but can be overridden by derived classes.

Syntax:

class Base {

public:

virtual void functionName() {

}

};

**Rules for Using Virtual Functions:**

* Cannot be static: Since static functions belong to the class itself (not an instance), they do not support runtime polymorphism and cannot be virtual.
* Matching signatures: The function signature in the derived class must match the base class to ensure proper overriding and runtime dispatch.
* Access via pointers/references: Virtual functions are generally called through base class pointers or references to achieve runtime polymorphism.
* Friend functions: Virtual functions can be declared as friends of other classes. However, friend functions themselves cannot be virtual as they are not class members.
* Optional overriding: Derived classes are not required to override a virtual function. If they don’t, the base class version will be executed.
* Constructors and destructors: Constructors cannot be virtual because objects are not fully established during construction. However, destructors should be virtual to ensure proper cleanup when deleting derived class objects via base class pointers.