

OBJECTIVE

- To comprehend the idea of C++ virtual functions.
- To discover how virtual functions are used to provide runtime polymorphism.
- To show how to use virtual functions to invoke derived class functions using base class pointers.

BACKGROUND THEORY

In C++, a virtual function is a special type of member function in a base class that is designed to be overridden in derived classes. It plays a central role in achieving runtime polymorphism, one of the core concepts of Object-Oriented Programming (OOP). By using virtual functions, the program can decide at runtime which version of a function to invoke, depending on the type of object being referenced not the type of pointer or reference used.

When a class has virtual functions, the compiler creates a virtual table (commonly known as a vtable) for that class. The vtable is essentially a lookup table containing pointers to the virtual functions of the class. Each object of that class contains a hidden pointer called vptr, which points to the vtable. When a virtual function is called using a base class pointer or reference, the call is resolved using the vtable, and the function corresponding to the actual type of the object is invoked.

Advantages of Virtual Functions:

- Enables polymorphism and code reusability.
- Makes the system more flexible and maintainable.
- Supports extension of code without modifying existing codebase.

Syntax:

```
class Base {  
public:  
    virtual void functionName(); // Virtual function declaration  
};
```

1. Create a base class Base with a virtual method display(), create a derived class Derived that overrides the display() method and implement a main() function where you create a Base pointer pointing to a Derived object and call the display() method.

```
#include <iostream>

using namespace std;

class Base {
public:
    // Virtual function
    virtual void display() {
        cout << "Display from Base class" << endl;
    }
};

class Derived : public Base {
public:
    // Overriding display function
    void display() override {
        cout << "Display from Derived class" << endl;
    }
};

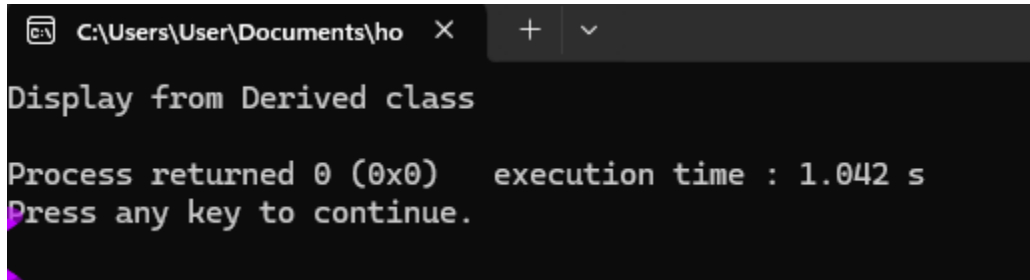
int main() {
    // Base class pointer pointing to Derived class object
    Base* ptr = new Derived();

    ptr->display();

    delete ptr;

    return 0;
}
```

Output:

A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Users\User\Documents\ho' and standard window controls. The command prompt displays the text 'Display from Derived class' on the first line. The second line shows 'Process returned 0 (0x0) execution time : 1.042 s'. The third line shows 'Press any key to continue.' with a cursor at the start of the line.

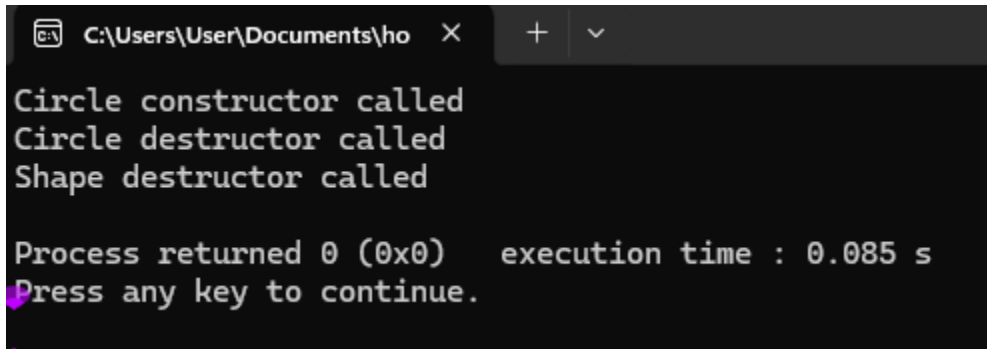
```
C:\Users\User\Documents\ho X + v
Display from Derived class
Process returned 0 (0x0) execution time : 1.042 s
Press any key to continue.
```

2. Create a base class Shape with a virtual destructor, create a derived class Circle that has a constructor and destructor and implement a main() function to demonstrate the use of virtual destructors by creating a Shape pointer pointing to a Circle object.

```
#include <iostream>
using namespace std;
class Shape {
public:
    virtual ~Shape() {
        cout << "Shape destructor called" << endl;
    }
};
class Circle : public Shape {
public:
    // Constructor
    Circle() {
        cout << "Circle constructor called" << endl;
    }
    ~Circle() {
        cout << "Circle destructor called" << endl;
    }
};
```

```
    }  
};  
  
int main() {  
    // Shape pointer pointing to a Circle object  
    Shape* s = new Circle();  
    delete s;  
    return 0;  
}
```

Output:



```
C:\Users\User\Documents\ho X + v  
Circle constructor called  
Circle destructor called  
Shape destructor called  
  
Process returned 0 (0x0) execution time : 0.085 s  
Press any key to continue.
```

Discussion:

In this lab, we explored the concept of virtual functions in object-oriented programming using C++. A virtual function allows derived classes to override a function defined in the base class, enabling dynamic (runtime) polymorphism. By using a base class pointer to point to a derived class object, we demonstrated that the overridden function in the derived class is executed when the function is declared as virtual in the base class.

Conclusion:

The lab successfully demonstrated the importance and functionality of virtual functions in C++. Through practical implementation, it was evident that virtual functions allow for proper function overriding and ensure that the correct function is called for an object, regardless of the type of pointer used.

