

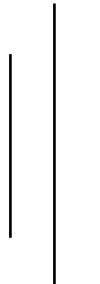


TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING



HIMALAYA COLLEGE OF ENGINEERING CHYASAL, LALITPUR



Lab Report No: - Virtual Function
Title: -06

Submitted by: -
Name: -Aditya Man Shrestha
Roll NO: -HCE081BEI005

Submitted To: -
Department Of
Checked by: -

Date of submission: -

Objectives:

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

Tools and Libraries Used:

- Programming Language: C++
- IDE: Code::Blocks
- Libraries: include <iostream>, include <string>

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:

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  
};
```

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  
    };  
};
```

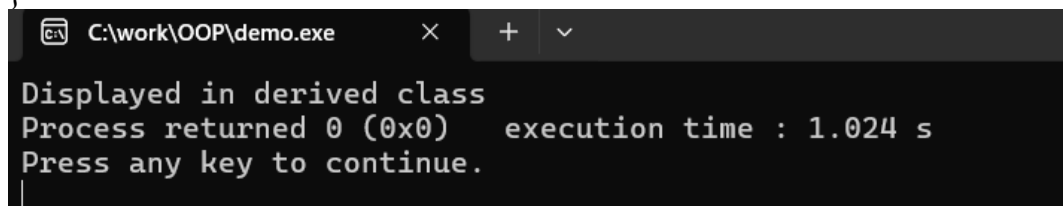
Rules of Using Virtual Functions

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.

Lab Assignment

Qn1.

```
#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;
}
```



```
C:\work\OOP\demo.exe
Displayed in derived class
Process returned 0 (0x0)   execution time : 1.024 s
Press any key to continue.
```

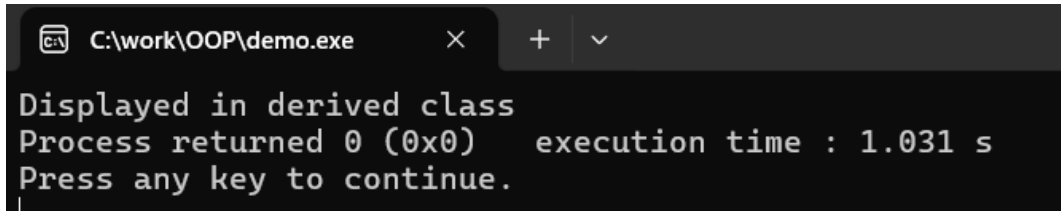
Qn2.

```
#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;  
}
```



```
C:\work\OOP\demo.exe  
Displayed in derived class  
Process returned 0 (0x0) execution time : 1.031 s  
Press any key to continue.
```

DISCUSSION

In this lab we were able to understand the core concept of virtual functions where we learnt about overriding functions, pure and virtual functions. We also learnt about dynamic cast, typeid and typeidinfo got information on interpreting cast and many more. It was very difficult to understand these concepts but with the help of online mediums on the internet I was able to understand these concepts.

CONCLUSION

From this lab, we can conclude that the use of virtual functions enhances code maintainability, reduces the need for rewriting code, and facilitates polymorphism which is a core concept in OOP.