**Virtual functions**

* Virtual Function is a function in base class, which is overrided in the derived class, and which tells the compiler to perform **Late Binding** on this function.
* There is a necessity to use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects. But, when base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function.
* A 'virtual' is a keyword preceding the normal declaration of a function.
* When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

Program

#include <iostream>

{

 public:

 virtual void display()

 {

  cout << "Base class is invoked"<<endl;

 }

};

class B:public A

{

 public:

 void display()

 {

  cout << "Derived Class is invoked"<<endl;

 }

};

int main()

{

 A\* a;    //pointer of base class

 B b;     //object of derived class

 a = &b;

 a->display();   //Late Binding occurs

}

Output

Derived Class is invoked

**Early binding and late binding**

The binding means the process of converting identifiers into addresses. For each variables and functions this binding is done. For functions it is matching the call with the right function definition by the compiler. The binding is done either at compile time or at runtime.

**Early Binding**

#include<iostream>

using namespace std;

class Base {

public:

void display() {

cout<<"In Base class" << endl;

}

};

class Derived: public Base {

public:

void display() {

cout<<"In Derived class" <<endl;

}

};

int main() {

Base \*base\_pointer;

Derived d;

base\_pointer=&d;

base\_pointer->display();

return 0;

}

Output

In Base class

## Late Binding

This is run time polymorphism. In this type of binding the compiler adds code that identifies the object type at runtime then matches the call with the right function definition. This is achieved by using virtual function.

#include<iostream>

using namespace std;

class Base {

public:

virtual void display() {

cout<<"In Base class" << endl;

}

};

class Derived: public Base {

public:

void display() {

cout<<"In Derived class" <<endl;

}

};

int main() {

Base \*base\_pointer;

Derived d;

base\_pointer=&d;

base\_pointer->display();

return 0;

}

Output

In Derived class