C++ Virtual Functions

## **C++ virtual functions : Definition and Description**

A virtual function is a member function of the base class, that is overridden in derived class. The classes that have virtual functions are called polymorphic classes.

The compiler binds virtual function at runtime, hence called runtime polymorphism. Use of virtual function allows the program to decide at runtime which function is to be called based on the type of the object pointed by the pointer.

In C++, the member function of a class is selected at runtime using virtual function. The function in the base class is overridden by the function with the same name of the derived class.

## **C++ virtual function : Syntax**

The keyword virtual is used for defining virtual function.

Class class\_name

{

public:

virtual return func\_name( args.. )

{

//function definition

}

}

## **Why do we need virtual functions?**

Virtual functions are needed for many reasons, among them to eliminate ambiguity is one.

Let’s make it more clear with this example and know when and why we need to use virtual functions.

#include <iostream>

using namespace std;

class Animal

{

public:

void my\_features()

{

cout << "I am an animal.";

}

};

class Mammal : public Animal

{

public:

void my\_features()

{

cout << "\nI am a mammal.";

}

};

class Reptile : public Animal

{

public:

void my\_features()

{

cout << "\nI am a reptile.";

}

};

int main()

{

Animal \*obj1 = new Animal;

Mammal \*obj2 = new Mammal;

Reptile \*obj3 = new Reptile;

obj1->my\_features();

obj2->my\_features();

obj3->my\_features();

return 0;

}

**Output**

I am an animal.

I am a mammal.

I am a reptile.

**Everything going as we expected, right.**

We created a three pointer objects \*obj1, \*obj2 and \*obj3. And when we call function my\_features()using these pointer objects, the corresponding functions of the classes are executed.

Now, what if we created an intermediate function to which we can pass objects of the corresponding class to invoke functions. This should also do the same.

#include <iostream>

using namespace std;

class Animal

{

public:

void my\_features()

{

cout << "I am an animal.";

}

};

class Mammal : public Animal

{

public:

void my\_features()

{

cout << "\nI am a mammal.";

}

};

class Reptile : public Animal

{

public:

void my\_features()

{

cout << "\nI am a reptile.";

}

};

//intermediate function

void intermediate\_func(Animal \*a1)

{

a1->my\_features();

}

int main()

{

Animal \*obj1 = new Animal;

Mammal \*obj2 = new Mammal;

Reptile \*obj3 = new Reptile;

intermediate\_func(obj1);

intermediate\_func(obj2);

intermediate\_func(obj3);

return 0;

}

**Output**

I am an animal.

I am an animal.

I am an animal.

**Whoa, this is not what we expected, right.**

So let’s decode what went wrong and how it went that way. We created an intermediate function intemediate\_func which takes an object as the argument. And depending on the class pointed by that object it invokes the member function of that class or at least we expected that way.

We supplied objects of Mammal and Reptile class to the intermediate function, which in return should have invoked obj2->my\_features() and obj3->my\_features(). But that didn’t happen because of the ambiguous situation aroused where compiler only invoked base class functions in all three situations.

So to overcome such ambiguity, virtual functions are used in base class. Virtual function dynamically binds function call at runtime and allows it to be overridden by the member function of the derrived class.

Here is how a virtual function implemented in c++.

### **Example: Sample program to illustrate use of virtual function in C++**

#include <iostream>

using namespace std;

class Animal

{

public:

virtual void my\_features()

{

cout << "I am an animal.";

}

};

class Mammal : public Animal

{

public:

void my\_features()

{

cout << "\nI am a mammal.";

}

};

class Reptile : public Animal

{

public:

void my\_features()

{

cout << "\nI am a reptile.";

}

};

//intermediate function

void intermediate\_func(Animal \*a1)

{

a1->my\_features();

}

int main()

{

Animal \*obj1 = new Animal;

Mammal \*obj2 = new Mammal;

Reptile \*obj3 = new Reptile;

intermediate\_func(obj1);

intermediate\_func(obj2);

intermediate\_func(obj3);

return 0;

}

**Output**

I am an animal.

I am a mammal.

I am a reptile.

Now we get desired output as we have used a virtual function in the base class, which lets member functions of derived class to override it.