C++ virtual function

A C++ virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword.

It is used to tell the compiler to perform dynamic linkage or late binding on the 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.

Late binding or Dynamic linkage

In late binding function call is resolved during runtime. Therefore compiler determines the type of object at runtime, and then binds the function call.

Rules of Virtual Function

Virtual functions must be members of some class.

Virtual functions cannot be static members.

They are accessed through object pointers.

They can be a friend of another class.

A virtual function must be defined in the base class, even though it is not used.

The prototypes of a virtual function of the base class and all the derived classes must be identical. If the two functions with the same name but different prototypes, C++ will consider them as the overloaded functions.

We cannot have a virtual constructor, but we can have a virtual destructor

Consider the situation when we don't use the virtual keyword.

```
#include <iostream>

using namespace std;

class A
```

```
{
  int x=5;
   public:
   void display()
   {
      std::cout << "Value of x is : " << x<<std::endl;
   }
};
class B: public A
{
   int y = 10;
   public:
   void display()
   {
      std::cout << "Value of y is : " <<y<< std::endl;
   }
};
int main()
{
   A *a;
   B b;
   a = &b;
  a->display();
   return 0;
}
```

Output: Value of x is : 5

In the above example, * a is the base class pointer. The pointer can only access the base class members but not the members of the derived class. Although C++ permits the base pointer to point to any object derived from the base class, it cannot directly access the members of the derived class. Therefore, there is a need for virtual function which allows the base pointer to access the members of the derived class.

C++ virtual function Example

Let's see the simple example of C++ virtual function used to invoked the derived class in a 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

Pure Virtual Function

A virtual function is not used for performing any task. It only serves as a placeholder.

When the function has no definition, such function is known as "do-nothing" function.

The "do-nothing" function is known as a pure virtual function. A pure virtual function is a function declared in the base class that has no definition relative to the base class.

A class containing the pure virtual function cannot be used to declare the objects of its own, such classes are known as abstract base classes.

The main objective of the base class is to provide the traits to the derived classes and to create the base pointer used for achieving the runtime polymorphism.

Pure virtual function can be defined as:

```
virtual void display() = 0;
```

Let's see a simple example:

```
#include <iostream>

using namespace std;

class Base

{

   public:

   virtual void show() = 0;

};

class Derived : public Base
```

```
{
   public:
   void show()
   {
      std::cout << "Derived class is derived from the base class." << std::endl;
   }
};
int main()
{
   Base *bptr;
   //Base b;
   Derived d;
   bptr = &d;
   bptr->show();
   return 0;
}
```

Output:

```
Derived class is derived from the base class.
```

In the above example, the base class contains the pure virtual function. Therefore, the base class is an abstract base class. We cannot create the object of the base class.

Pure Virtual Functions and Abstract Classes in C++

Sometimes implementation of all function cannot be provided in a base class because we don't know the implementation. Such a class is called abstract class.

For example, let Shape be a base class.

We cannot provide implementation of function draw() in Shape, but we know every derived class must have implementation of draw(). Similarly an Animal class doesn't have implementation of move() (assuming that all animals move), but all animals must know how to move. We cannot create objects of abstract classes.

A pure virtual function (or abstract function) in C++ is a virtual function for which we don't have implementation, we only declare it. A pure virtual function is declared by assigning 0 in declaration. See the following example.

```
// An abstract class
class Test
{
   // Data members of class
public:
   // Pure Virtual Function
   virtual void show() = 0;
     /* Other members */
};
```

A complete example:

A pure virtual function is implemented by classes which are derived from a Abstract class. Following is a simple example to demonstrate the same.

```
#include<iostream>
using namespace std;
 class Base
{
  int x;
public:
   virtual void fun() = 0;
```

```
    int getX() { return x; }
};
 // This class inherits from Base and implements fun()
class Derived: public Base
{
    int y;
public:
    void fun() { cout << "fun() called"; }
};
 int main(void)
{
    Derived d;
    d.fun();
    return 0;
}
```

Output:

```
fun() called
```

Some Interesting Facts:

1) A class is abstract if it has at least one pure virtual function.

In the following example, Test is an abstract class because it has a pure virtual function show().

```
// pure virtual functions make a class abstract
#include<iostream>
using namespace std;
 class Test
{
   int x;
```

```
public:
   virtual void show() = 0;
   int getX() { return x; }
};
 int main(void)
{
   Test t;
   return 0;
}
```

Output:

```
Compiler Error: cannot declare variable 't' to be of abstract
 type 'Test' because the following virtual functions are pure
within 'Test': note:     virtual void Test::show()
```

2) We can have pointers and references of abstract class type.

For example the following program works fine.

```
#include<iostream>
using namespace std;
 class Base
{
public:
   virtual void show() = 0;
};

class Derived: public Base
{
public:
```

```
    void show() { cout << "In Derived \n"; }
};

int main(void)
{
    Base *bp = new Derived();
    bp->show();
    return 0;
}
```

Output:

```
In Derived
```

3) If we do not override the pure virtual function in derived class, then derived class also becomes abstract class.

The following example demonstrates the same.

```
#include<iostream>
using namespace std;
class Base
{
public:
    virtual void show() = 0;
};

class Derived : public Base { };
  int main(void)
{
  Derived d;
  return 0;}
```

Compiler Error: cannot declare variable 'd' to be of abstract type

'Derived' because the following virtual functions are pure within

'Derived': virtual void Base::show()

4) An abstract class can have constructors.

For example, the following program compiles and runs fine.

```
#include<iostream>
using namespace std;
 // An abstract class with constructor
class Base
{
protected:
   int x;
public:
  virtual void fun() = 0;
  Base(int i) { x = i; }
};
 class Derived: public Base
{
    int y;
public:
    Derived(int i, int j):Base(i) { y = j; }
    void fun() { cout << "x = " << x << ", y = " << y; }
};
  int main(void)
{
    Derived d(4, 5);
```

```
    d.fun();

    return 0;

}
```

Output:

x = 4, y = 5

C++ Abstract Class

Abstract class is used in situation, when we have partial set of implementation of methods in a class. For example, consider a class have four methods. Out of four methods, we have an implementation of two methods and we need derived class to implement other two methods. In these kind of situations, we should use abstract class.

- A virtual function will become pure virtual function when you append "=0" at the end of declaration of virtual function.
- A class with at least one pure virtual function or abstract function is called abstract class.
- Pure virtual function is also known as abstract function.
- We can't create an object of abstract class b'coz it has partial implementation of methods.
- Abstract function doesn't have body
- We must implement all abstract functions in derived class.

Example of C++ Abstract class

```
#include<iostream.h>

#include<conio.h>

class BaseClass       //Abstract class

{

     public:

      virtual void Display1()=0;     //Pure virtual function or abstract function

      virtual void Display2()=0;     //Pure virtual function or abstract function

      void Display3()

      {

           cout<<"\n\tThis is Display3() method of Base Class";

      }

};
```

```
class DerivedClass : public BaseClass
{
    public:
     void Display1()
     {
          cout<<"\n\tThis is Display1() method of Derived Class";
     }
     void Display2()
     {
          cout<<"\n\tThis is Display2() method of Derived Class";
     }
};
void main()
{
     DerivedClass D;
     D.Display1();          // This will invoke Display1() method of Derived Class
     D.Display2();          // This will invoke Display2() method of Derived Class
     D.Display3();          // This will invoke Display3() method of Base Class
}
```

Output :

```
     This is Display1() method of Derived Class
     This is Display2() method of Derived Class
     This is Display3() method of Base Class
```