1)Ambiguity in Inheritance

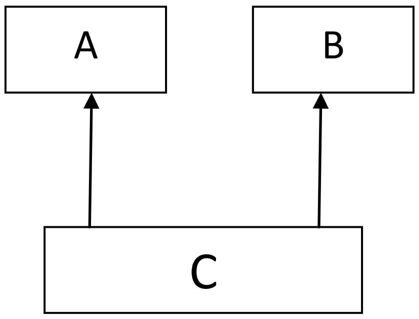
Ambiguity mainly comes in multiple inheritance. Two base classes have functions with the same name, while a class derived from both base classes has no function with this name. When we call the function with derived class's object then compiler can't figure out which of the two functions is meant. This is called ambiguity in inheritance.
Ambiguity can be solved with the help of scope resolution operator(::).

Syntax: Objectname . BaseClassName :: FunctionNAme(..);

Object of base class dot operator Function to be invoked in base class

Here's an example

```
#include <iostream>
#include<conio.h>
class A
{
public:
void show()
{ cout << "Class A"; }
};
class B
{
public:
void show()
 { cout << "Class B; }
};
class C : public A, public B
{
};
 void  main()
{
C objC;        //object of class C
objC.show();   //ambiguous--will not compile
objC.A::show();     //OK
objC.B::show();     //OK
getch();
}
```

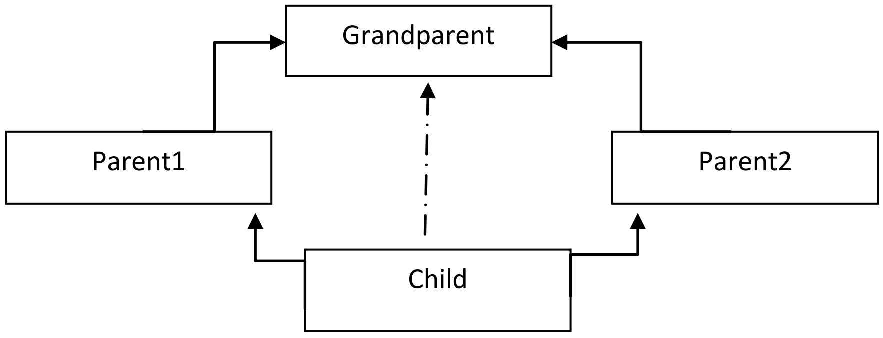


Multiple Inheritance

The problem is resolved using the **scope-resolution operator** to specify the class in which the function lies.
Thus **objC.A::show();** refers to the function show() from A class,
while **objC.B::show();** refers to the function in the B class. This is *disambiguation*.

<u>2)Multipath Inheritance and Virtual Base Class:</u>

The form of inheritance which derives a new class by multiple inheritance of base classes,which are derived earlier from the same base class, is known as **multipath inheritance**.
So, multipath inheritance involves more than one form of inheritance namely multilevel,multiple and hierarchical as shown in fig (1).



Multipath Inheritance(with the concept of virtual base class)

Multipath Inheritance can poses some problems in compilation. The public and protected members of grandparent are inherited into child class **twice,** i) via parent1 class and
ii) via parent2 class. Therefore, the child class would have duplicate sets of members of the grandparent which leads to **<u>ambiguity</u>** during compilation and it should be avoided.

C++ supports another important concept called virtual base classes to handle AMBIGUITY caused due to multipath inheritance. This problem is solved by making the common base class as a virtual base class while declaring the classes during inheritance.

```
#include <iostream>
using namespace std;
class A
{
public:
void func();
};
class B : public A
{ };
class C : public A
{ };
class D : public B, public C
{ };
int main()
{
D objD;
objD.func(); //ambiguous: won't compile
return 0;
}
```

Classes B and C are both derived from class A, and class D is derived by multiple inheritance from both B and C. Trouble starts if you try to access a member function in class A from an object of class D. In this example objD tries to access func(). However, both B and C contain a copy of
func(), inherited from A. The compiler can't decide which copy to use, and signals an error.

This ambiguity can be resolved if the class derived contains only one copy of the class base. This can be done by making the base class a virtual class. This keyword makes the two classes share a single copy of their base class . It can be done as follows :

```
class base
{
:
:
};

class A : virtual public base
{
:
:
};
class B : virtual public base
{
:
:
};
class derived : public A, public
{
:
:
};
```

This will resolve the ambiguity involved.

3)Function Overriding

Function overriding is defined as a member function of a derived class may have the same name as a member function of a base class.

Ex. Program to show the concept of function overriding and function hiding.

```
#include<iostream.h>
#include<conio.h>
class a
{ public: void f1(){}
          void f2{}
};
```