In multiple inheritances, when one class is derived from two or more base classes then there may be a possibility that the base classes have functions with the same name, and the derived class may not have functions with that name as those of its base classes. If the derived class object needs to access one of the similarly named member functions of the base classes then it results in ambiguity because the compiler gets confused about which base's class member function should be called.

**Example**:

- C++

```
// C++ program to show inheritance ambiguity

#include<iostream>
using namespace std;

// Base class A

class A {
    public:

    void func() {
        cout << " I am in class A" << endl;
    }
};

// Base class B

class B {
    public:

    void func() {
        cout << " I am in class B" << endl;
    }
};

// Derived class C

class C: public A, public B {


};

// Driver Code

int main() {
    // Created an object of class C

    C obj;

    // Calling function func()

    obj.func();

    return 0;
}
```

**Output**:

```
prog.cpp: In function ‘int main()’:
prog.cpp:43:9: error: request for member ‘func’ is ambiguous
     obj.func();
         ^
prog.cpp:21:10: note: candidates are: void B::func()
     void func() {
          ^
prog.cpp:11:10: note:                 void A::func()
     void func() {
          ^
```

In this example, derived class C inherited the two base classes A and B having the same function name func(). When the object of class C is created and called the function func() then the compiler gets confused that which base class member function func() should be called.

**Solution to  Ambiguity:**

To solve this ambiguity ***scope resolution operator*** is used denoted by ‘ **::** ‘

**Syntax:**

`ObjectName.ClassName::FunctionName();`

Below is the program to show the concept of ambiguity resolution in multiple inheritances.

- C++

```
// C++ program to resolve inheritance
// ambiguity
```

```
#include<iostream>
using namespace std;

// Base class A

class A {
    public:

    void func() {
        cout << " I am in class A" << endl;
    }
};

// Base class B

class B {
    public:

    void func() {
        cout << " I am in class B" << endl;
    }
};

// Derived class C
class C: public A, public B {


};

// Driver Code

int main() {

    // Created an object of class C
    C obj;

    // Calling function func() in class A
    obj.A::func();

    // Calling function func() in class B
    obj.B::func();

    return 0;
}
```

**Output**

```
I am in class A
I am in class B
```