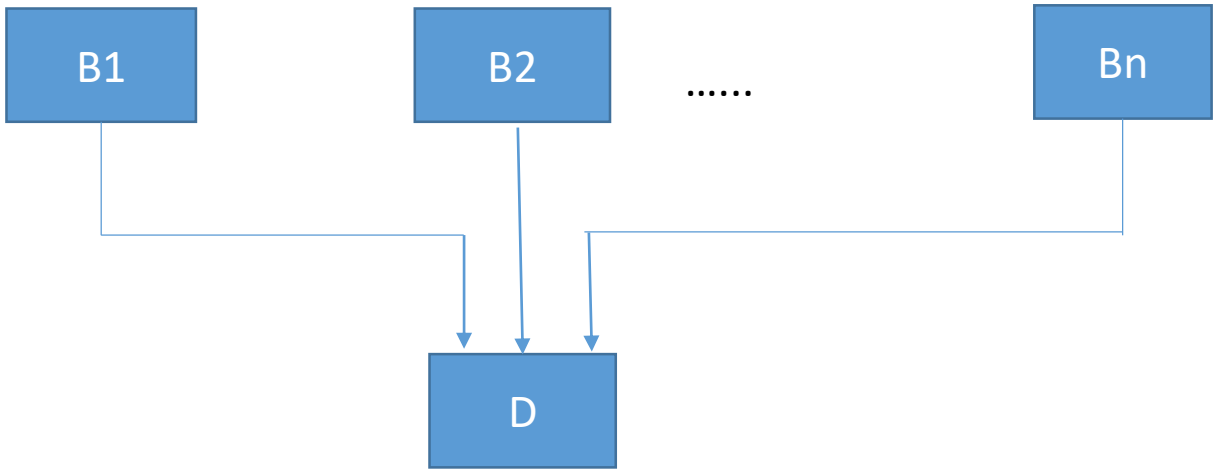
Multiple inheritance

- A class can inherit the attributes of two or more classes. This is known as multiple inheritance.
- Allows us to combine the features of several existing classes as a starting point for defining new classes.
- Let B1, B2,….,Bn be the base classes and D be the derived class.



- Syntax

```
class D: visibility B1, visibility B2 ……
{
        .....
        …….
        ……
};
```

Here visibility specifies whether public or private

Example program

```
#include<iostream>
using namespace std;
class M
{
  protected:
          int m;
 public:
         void get_m(int);
};
class N
{
  protected:
          int n;
 public:
         void get_n(int);
};
```

```
class P: public M, public N
{
  public:
        void display(void);
}
 void M::get_m(int x)
{
   m = x;
}
void N::get_n(int y)
{
   n = y;
}
void P:: display(void)
{
  cout<<“m= “ <<m<<“\n”;
  cout<<“n= “ <<n<<“\n”;
  cout<<“m*n= “ <<m*n<<“\n”;
}
```

```
int main()
{
        P p;
        p.get_m(10);
        p.get_n(20);
        p.display();
        return 0;
}
```

OUTPUT

```
m = 10
n =  20
m*n = 200
```

Ambiguity resolution in Inheritance

- If a function with the same name appears in more than one base class, then that problem can be solved using a scope resolution operator.

- For example, if a function named display() is present in both base class and derived class, then a simple call to display() will result in invoking the display() in the derived class.(because derived class overrides the base class function).

- In this case, invoke the function using the scope resolution operator in the main function.

```
class A
{ public:
  void display()
  {  cout<<“A\n”;  }
};
class B: public A
{  public:
    void display()
  { cout<<“B\n”;  }
};
int main()
{
 B b;
b.display();
b.A::display();
return 0;
}
```

OUTPUT

B
A