C++ Multilevel Inheritance

Multilevel inheritance is a process of deriving a class from another derived class.

$$\begin{tikzcd} \boxed{A} \arrow[r] & \boxed{B} \arrow[r] & \boxed{C} \end{tikzcd}$$

↳ multilevel inherited.

→ Inheritance is transitive so the last derived class acquires all the members of its base class.

eg.

```
class Animal {
    public :
        void eat () {
            cout << "Eating" << endl;
        }
    };
class Dog : public Animal {
    Public :
        void bark () {
            cout << " Barking" << endl;
        }
    };
class BabyDog : Public Dog {
    Public:
        void weep () {
            cout << "Weeping" << endl
        }
};
```

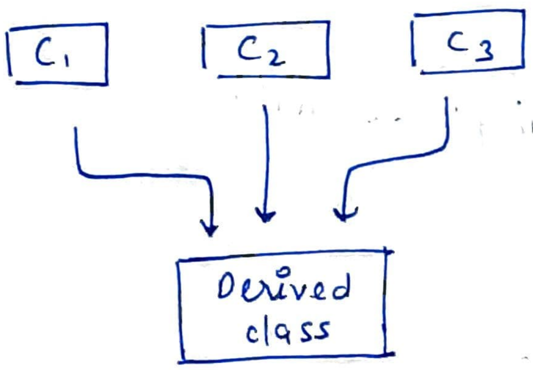
```
int main () {
     Baby Dog d1 ;
     d1 . eat ();
     d1 . bark ();
     d1 . weep ();
  }
```

output

```
Eating
Barking
Weeping
```

C++ Multiple Inheritance

Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.



syntax

```
class D: visibility C1, visibility C2
{
    // Body
}
```

eg.

```
class A {
   protected:
       int a;
   Public :
       void get_a (int n) {
              a = n;
       }
   };
```

```
class B {
   protected:
      int b;
   Public:
      void get-b(int n) {
            b = n;
      }
   };

class C : public A, public B {

   public:
      void display() {
                                          a
            cout << "value of a:" << ~~[illegible]~~ << endl;
            cout << "value of b:" << b << endl;
            cout << " Addition " << a+b << endl;
        }
   };

int main() {

    C c;
    c.get_a(10);
    c.get_b(20);
    c.display();
 }
```

output

```
    value of a: 10
    value of b: 20
     Addition    30
```

Ambiguity Resolution in Inheritance

Ambiguity can occur in using multiple inheritance when a function with same name occurs in more than one $\overset{\text{base}}{\wedge}$ class.

eg.

```
class A {
    public :
        void display {
                cout << " Class A" << endl;
            }
        };

class B {
    public ;
        void display () {
                cout << " Class B" << endl;
            }
        };

class C: public A, public B {
        void view () {
                display () ;
            }
    };

int main () {
        C c;
        c. display ();
    }
```

Output :- error: reference to 'display' is ambiguous.

This issue can be resolved by using class resolution operator.

eg.

```
class C : public A, public B {
        void view () {
                A :: display ();
                B :: display ();
            }
    };
```

output

```
class A
class B.
```

→ Ambiguity can also occur in single inheritance.

eg.

```
class A {
    public:
        void display() {
                cout << "class A" << endl;
        }};
class B : public A {
    public:
        void display() {
                cout << "class B" << endl;
            }
    };
```

```
int main() {
    B b;
    b. display ()
    b. A :: display()
    b. B :: display()
}
```

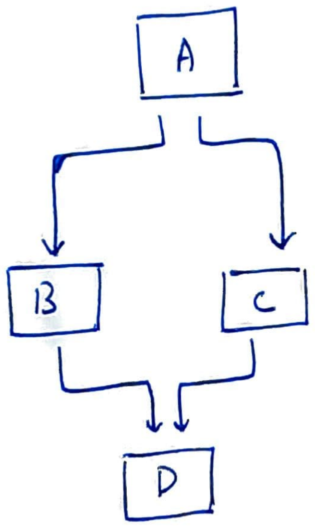
output

```
class B
class A
class B .
```

In above case, the function of derived class overrides the method of base class. Thus, call to display() function will simply call the function defined in derived class. If we want to invoke the base class function, we can use class resolution operator.

C++ Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance.



In this case, methods of all A, B and C would be inherited by D.

eg:

```
class A {
        protected: int a;
public:  void get_a() {
              cout << " Enter a" << endl;
              cin >> a;
         }
        };

class B : public A {
         protected : int b;
         public :
              void get_b() {
                  cout << " Enter b " << endl;
                  cin >> b;
              }
         };
```

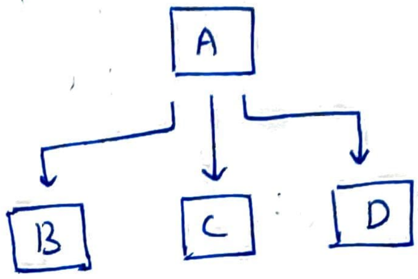
```
class C {
    Protected :  int c;
    Public :
        void get_c() {
            cout << "Enter c" << endl;
            cin >> c;
        }
};

class D : public B, Public C {
    Protected : int d;
    public :
        void mul() {
            get_a();
            get_b();
            get_c();
            cout << "Multiplication:" << axbxc <<endl;
        }
};

int main () {
    D d;
    d.mul();
}
```

output

```
Enter a
 10
Enter b
 20
Enter c
 30
Multiplication   6000
```

C++ Heirarchical Inheritance

→ Process of deriving more than one class from a base class.



Syntax :

```
class A { // body };

class B : public A { }
class C : Poblic A { };
class D: Public A { };
```

} → Nothing new here.

eg.

```
class shape {
    public :
        int a, b;
        void get_data (int n, int m){
            a = n;
            b = m;
        }
};

class Rectangle : Poblic shape {
    Public :
        int rect_area() {
            int result = a x b ;
            retun result;
        }
};
```

```
class Triangle : public shape {
      Public :
          int triangle_area () {
                  float result = 0.5 x a x b ;
                  int result;
          }
      };

int main () {

    Rectangle r;
    Triangle t ;
    int length, breadth, base, height;
    cin>> length >> breadth;
    r. get_data (length, breadth) ;

    int m = r. rect_area ()
    cout << " area of rectangle " << m << endl;

    cin >> base >> height;
    t. get_data (base, height)
    float n = t. triangle_area () ;
    cout << "area of triangle" << n << endl;

}
```

output

```
        23                                   2
        20                                   5
      area of rectangle 460        area of triangle . 5.0
```