**INHERITENCE**

inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.

 the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

**SINGLE INHERITENCE**

Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.

A->B

Where 'A' is the base class, and 'B' is the derived class.

When one class inherits another class, it is known as single level inheritance.

#include <iostream>

using namespace std;

class Account {

public:

float salary = 60000;

};

class Programmer: public Account {

public:

float bonus = 5000;

};

int main(void) {

Programmer p1;

cout<<"Salary: "<<p1.salary<<endl;

cout<<"Bonus: "<<p1.bonus<<endl;

return 0;

}

Output:

Salary: 60000

Bonus: 5000

**MULTILEVEL INHERITENCE**

When one class inherits another class which is further inherited

by another class, it is known as multi level inheritance

A->B->C

#include <iostream>

using namespace std;

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...";

}

};

int main(void) {

BabyDog d1;

d1.eat();

d1.bark();

d1.weep();

return 0;

}

Output:

Eating...

Barking...

Weeping...

**MULTIPLE INHERITENCE**

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

#include <iostream>

using namespace std;

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()

{

std::cout<<"The value of a is:"<<a<<std::endl;

std::cout<<"The value of b is:"<<b<<std::endl;

cout<<"addition of a and b is:"<<a+b;

}

};

int main()

{

C c;

c.get\_a(10);

c.get\_b(20);

c.display();

return 0;

}

Output:

The value of a is : 10

The value of b is : 20

Addition of a and b is : 30

**HYBRID INHERITENCE**

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

#include <iostream>

using namespace std;

class A

{

protected:

int a;

public:

void get\_a()

{

std::cout << "Enter the value of 'a' : " << std::endl;

cin>>a;

}

};

class B : public A

{

protected:

int b;

public:

void get\_b()

{

std::cout << "Enter the value of 'b' : " << std::endl;

cin>>b;

}

};

class C

{

protected:

int c;

public:

void get\_c()

{

std::cout << "Enter the value of c is : " << std::endl;

cin>>c;

}

};

class D : public B, public C

{

protected:

int d;

public:

void mul()

{

get\_a();

get\_b();

get\_c();

std::cout << "Multiplication of a,b,c is : " <<a\*b\*c<< std::endl;

}

};

int main()

{

D d;

d.mul();

return 0;

}

Output:

Enter the value of 'a' :

10

Enter the value of 'b' :

20

Enter the value of c is :

30

Multiplication of a,b,c is : 6000

**HIERARCHICAL INHERITANCE**

Hierarchical inheritance is defined as the process of deriving more than one class from a base class.

#include <iostream>

using namespace std;

class Shape // Declaration of base class.

{

public:

int a;

int b;

void get\_data(int n,int m)

{

a= n;

b = m;

}

};

class Rectangle : public Shape // inheriting Shape class

{

public:

int rect\_area()

{

int result = a\*b;

return result;

}

};

class Triangle : public Shape // inheriting Shape class

{

public:

int triangle\_area()

{

float result = 0.5\*a\*b;

return result;

}

};

int main()

{

Rectangle r;

Triangle t;

int length,breadth,base,height;

std::cout << "Enter the length and breadth of a rectangle: " << std::endl;

cin>>length>>breadth;

r.get\_data(length,breadth);

int m = r.rect\_area();

std::cout << "Area of the rectangle is : " <<m<< std::endl;

std::cout << "Enter the base and height of the triangle: " << std::endl;

cin>>base>>height;

t.get\_data(base,height);

float n = t.triangle\_area();

std::cout <<"Area of the triangle is : " << n<<std::endl;

return 0;

}

Output:

Enter the length and breadth of a rectangle:

23

20

Area of the rectangle is : 460

Enter the base and height of the triangle:

2

5

Area of the triangle is : 5

**AMBIGUITY IN INHERITENCE**

#include<iostream>

using namespace std;

class A {

public: void func()

{

cout << " I am in class A" << endl;

}

};

class B { public: void func()

{

cout << " I am in class B" << endl;

}

};

class C: public A, public B { };

int main()

{

C obj;

obj.func();

return 0;

}

you need to specify explicitly which base class function you want to call. You can do this by using the scope resolution operator.

#include <iostream>

using namespace std;

class A {

public:

void func() {

cout << "I am in class A" << endl;

}

};

class B {

public:

void func() {

cout << "I am in class B" << endl;

}

};

class C : public A, public B {

};

int main() {

C obj;

obj.A::func(); // Specify that we want to call func() from class A

obj.B::func(); // Specify that we want to call func() from class B

return 0;

}

Output:

I am in class A

I am in class B

**Ambiguity resolution in inheritance**

The above issue can be resolved by using the class resolution operator with the function. In the above example, the derived class code can be rewritten as:

class C public A, public B

{

void view()

{

A :: display();

B:: display();

}

};

INPUT:

#include <iostream>

using namespace std;

class A {

public:

void display() {

cout << "Display from class A" << endl;

}

};

class B {

public:

void display() {

cout << "Display from class B" << endl;

}

};

class C : public A, public B {

public:

void view() {

A::display();

B::display();

}

};

int main() {

C obj;

obj.view();

return 0;

}

OUTPUT:

Display from class A

Display from class B

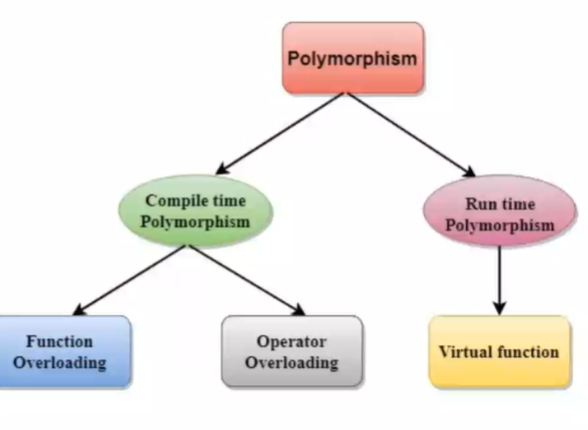
**POLYMORPHISM**

The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms.

Let's consider a real-life example of polymorphism. A lady behaves like a teacher in a classroom, mother or daughter in a home and customer in a market. Here, a single person is behaving differently according to the situations.

There are two types of polymorphism in C++: Compile time polymorphism

Run time polymorphism



**Function overloading**

Function Overloading is defined as the process of having two or more function with the same name, but different in parameters is known as function overloading in C++.

In function overloading, the function is redefined by using either different types of arguments or a different number of arguments.

It is only through these differences compiler can differentiate between the functions.

The advantage of Function overloading is that it increases the readability of the program because you don't need to use different names for the same action

