Inheritance & Its Different Types with Examples in C++

Inheritance in C++ an Overview

- Reusability is a very important feature of OOPs
- In C++ we can reuse a class and add additional features to it
- Reusing classes saves time and money
- Reusing already tested and debugged classes will save a lot of effort of developing and debugging the same thing again

What is Inheritance in C++?

- The concept of reusability in C++ is supported using inheritance
- We can reuse the properties of an existing class by inheriting it
- The existing class is called a base class
- The new class which is inherited from the base class is called a derived class
- Reusing classes saves time and money
- There are different types of inheritance in C++

Forms of Inheritance in C++

- Single Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Multilevel Inheritance
- Hybrid Inheritance

Single Inheritance in C++

Single inheritance is a type of inheritance in which a derived class is inherited with only one base class. For example, we have two classes “employee” and “programmer”. If the “programmer” class is inherited from the “employee” class which means that the “programmer” class can now implement the functionalities of the “employee” class.

Multiple Inheritances in C++

Multiple inheritances are a type of inheritance in which one derived class is inherited with more than one base class. For example, we have three classes “employee”, “assistant” and “programmer”. If the “programmer” class is inherited from the “employee” and “assistant” class which means that the “programmer” class can now implement the functionalities of the “employee” and “assistant” class.

Hierarchical Inheritance

A hierarchical inheritance is a type of inheritance in which several derived classes are inherited from a single base class. For example, we have three classes “employee”, “manager” and “programmer”. If the “programmer” and “manager” classes are inherited from the “employee” class which means that the “programmer” and “manager” class can now implement the functionalities of the “employee” class.

Multilevel Inheritance in C++

Multilevel inheritance is a type of inheritance in which one derived class is inherited from another derived class. For example, we have three classes “animal”, “mammal” and “cow”. If the “mammal” class is inherited from the “animal” class and “cow” class is inherited from “mammal” which means that the “mammal” class can now implement the functionalities of “animal” and “cow” class can now implement the functionalities of “mammal” class.

Hybrid Inheritance in C++

Hybrid inheritance is a combination of multiple inheritance and multilevel inheritance. In hybrid inheritance, a class is derived from two classes as in multiple inheritances. However, one of the parent classes is not a base class. For example, we have four classes “animal”, “mammal”, “bird”, and “bat”. If “mammal” and “bird” classes are inherited from the “animal” class and “bat” class is inherited from “mammal” and “bird” classes which means that “mammal” and “bird” classes can now implement the functionalities of “animal” class and “bat” class can now implement the functionalities of “mammal” and “bird” classes.

Inheritance Syntax & Visibility Mode in C++

Inheritance is a process of inheriting attributes of the base class by a derived class.

```
// Derived Class syntax

class {{derived-class-name}} : {{visibility-mode}} {{base-class-name}}

{

   class members/methods/etc...

}
```

Note:

Default visibility mode is private

Public Visibility Mode: Public members of the base class becomes Public members of the derived class

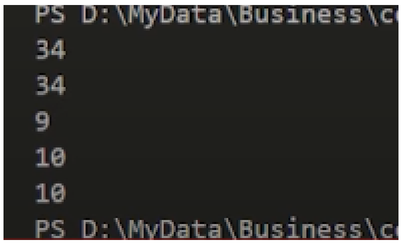
Private Visibility Mode: Public members of the base class become private members of the derived class

Private members are never inherited

```
#include <iostream>

using namespace std;


// Base Class

class Employee
```

```
{
public:
   int id;
   float salary;
   Employee(int inpId)
   {
      id = inpId;
      salary = 34.0;
   }
   Employee() {}
};

// Creating a Programmer class derived from Employee Base class
class Programmer : public Employee
{
public:
   int languageCode;
   Programmer(int inpId)
   {
      id = inpId;
      languageCode = 9;
   }
   void getData(){
      cout<<id<<endl;
   }
};
int main()
{
   Employee harry(1), rohan(2);
   cout << harry.salary << endl;
   cout << rohan.salary << endl;
```

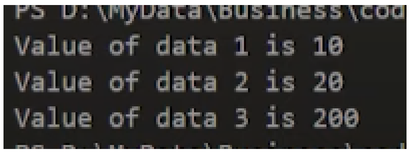
```
    Programmer skillF(10);

    cout << skillF.languageCode<<endl;

    cout << skillF.id<<endl;

    skillF.getData();

    return 0;

}
```



```
PS D:\MyData\Business\c
34
34
9
10
10
PS D:\MyData\Business\c
```

Single Inheritance

```
class Base

{

    int data1; // private by default and is not inheritable

public:

    int data2;

    void setData();

    int getData1();

    int getData2();

};


void Base ::setData(void)

{

    data1 = 10;

    data2 = 20;

}


int Base::getData1()

{
    return data1;
}

int Base::getData2()
{
    return data2;
}
class Derived : public Base
{ // Class is being derived publically
    int data3;

public:
    void process();
    void display();
};

void Derived ::process()
{
    data3 = data2 * getData1();
}

void Derived ::display()
{
    cout << "Value of data 1 is " << getData1() << endl;
    cout << "Value of data 2 is " << data2 << endl;
    cout << "Value of data 3 is " << data3 << endl;
}
int main()
{
    Derived der;
    der.setData();
    der.process();

    der.display();


    return 0;

}
```



```
PS D:\MyData\Business\cod
Value of data 1 is 10
Value of data 2 is 20
Value of data 3 is 200
```

Protected Access Modifier in C++

Protected Access Modifiers in C++

Protected Access Modifiers in C++

Protected access modifiers are similar to the private access modifiers but protected access modifiers can be accessed in the derived class whereas private access modifiers cannot be accessed in the derived class.

- If the class is inherited in public mode then its private members cannot be inherited in child class.
- If the class is inherited in public mode then its protected members are protected and can be accessed in child class.
- If the class is inherited in public mode then its public members are public and can be accessed inside child class and outside the class.
- If the class is inherited in private mode then its private members cannot be inherited in child class.
- If the class is inherited in private mode then its protected members are private and cannot be accessed in child class.
- If the class is inherited in private mode then its public members are private and cannot be accessed in child class.
- If the class is inherited in protected mode then its private members cannot be inherited in child class.
- If the class is inherited in protected mode then its protected members are protected and can be accessed in child class.
- If the class is inherited in protected mode then its public members are protected and can be accessed in child class.

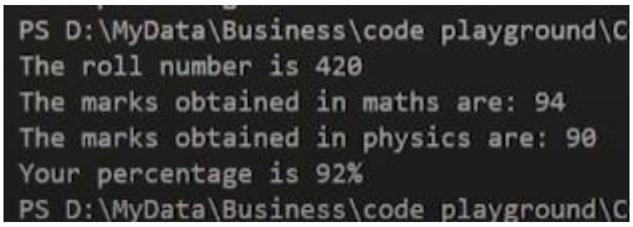
```
#include<iostream>
using namespace std;
class Base{
   protected:
      int a;
   private:
      int b;
};
class Derived: protected Base{
 };
int main(){
   Base b;
   Derived d;
   // cout<<d.a; // Will not work since a is protected in both base as well as derived class
   return 0;
}
```

Multilevel Inheritance in C++

```
#include <iostream>
using namespace std;
class Student
{
protected:
   int roll_number;
public:
   void set_roll_number(int);
   void get_roll_number(void);
};
void Student ::set_roll_number(int r)
{
   roll_number = r;
}
```

```
void Student ::get_roll_number()
{
   cout << "The roll number is " << roll_number << endl;
}
class Exam : public Student
{
protected:
   float maths;
   float physics;
public:
   void set_marks(float, float);
   void get_marks(void);
};
void Exam ::set_marks(float m1, float m2)
{
   maths = m1;
   physics = m2;
}
void Exam ::get_marks()
{
   cout << "The marks obtained in maths are: " << maths << endl;
   cout << "The marks obtained in physics are: " << physics << endl;
}
class Result : public Exam
{
   float percentage;
public:
   void display_results()
   {
      get_roll_number();
```

```
        get_marks();

        cout << "Your result is " << (maths + physics) / 2 << "%" << endl;

    }

};

int main()

{

    Result harry;

    harry.set_roll_number(420);

    harry.set_marks(94.0, 90.0);

    harry.display_results();

    return 0;

}
```



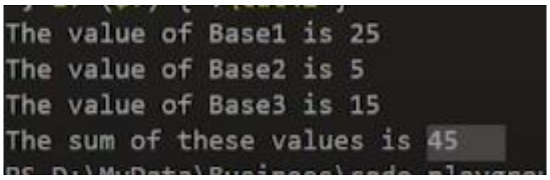
```
PS D:\MyData\Business\code playground\C
The roll number is 420
The marks obtained in maths are: 94
The marks obtained in physics are: 90
Your percentage is 92%
PS D:\MyData\Business\code playground\C
```

Multiple Inheritance

```
class Base1{

protected:

    int base1int;

public:

    void set_base1int(int a)

    {

        base1int = a;

    }

};

class Base2{

protected:

    int base2int;
```

```
public:
   void set_base2int(int a)
   {
       base2int = a;
   }
};
class Base3{
protected:
   int base3int;
public:
   void set_base3int(int a)
   {
       base3int = a;
   }
};
class Derived : public Base1, public Base2, public Base3
{
   public:
       void show(){
           cout << "The value of Base1 is " << base1int<<endl;
           cout << "The value of Base2 is " << base2int<<endl;
           cout << "The value of Base3 is " << base3int<<endl;
           cout << "The sum of these values is " << base1int + base2int + base3int << endl;
       }
};
int main()
{
   Derived harry;
   harry.set_base1int(25);
   harry.set_base2int(5);
```

```
    harry.set_base3int(15);

    harry.show();

   return 0;

}
```



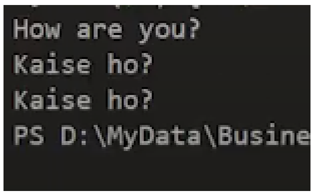
```
The value of Base1 is 25
The value of Base2 is 5
The value of Base3 is 15
The sum of these values is 45
```

Ambiguity Resolution in Inheritance

Ambiguity Resolution in Inheritance

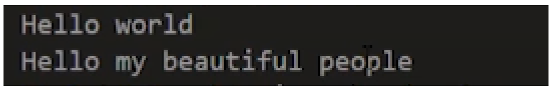
Ambiguity in inheritance can be defined as when one class is derived for two or more base classes then there are chances that the base classes have functions with the same name. So it will confuse derived class to choose from similar name functions. To solve this ambiguity scope resolution operator is used "::". An example program is shown below to demonstrate the concept of ambiguity resolution in inheritance.

```
class Base1{

    public:

        void greet(){

            cout<<"How are you?"<<endl;

        }

};

class Base2{

    public:

        void greet()

        {

            cout << "Kaise ho?" << endl;

        }

};

class Derived : public Base1, public Base2{

   int a;

   public:

    void greet(){
        Base2 :: greet();
    }
};
int main(){
  // Ambibuity 1
     Base1 base1obj;
     Base2 base2obj;
     base1obj.greet();
     base2obj.greet();
     Derived d;
     d.greet();

    return 0;
}
```



```
How are you?
Kaise ho?
Kaise ho?
PS D:\MyData\Busine
```

```
class B{
    public:
        void say(){
            cout<<"Hello world"<<endl;
        }
};
class D: public B{
    int a;
    // D's new say() method will override base class's say() method
    public:
        void say()
        {
            cout << "Hello my beautiful people" << endl;
        }
};
int main(){
    // Ambibuity 2
    B b;
    b.say();
   D d;
    d.say();
return 0;
}
```



```
Hello world
Hello my beautiful people
```