⇒ Inheritance :-

Inheritance is a property of oops by which we can inherit all features of a class (base class) to other class (derived class).

By using inheritance gives us freedom to reuse our code.

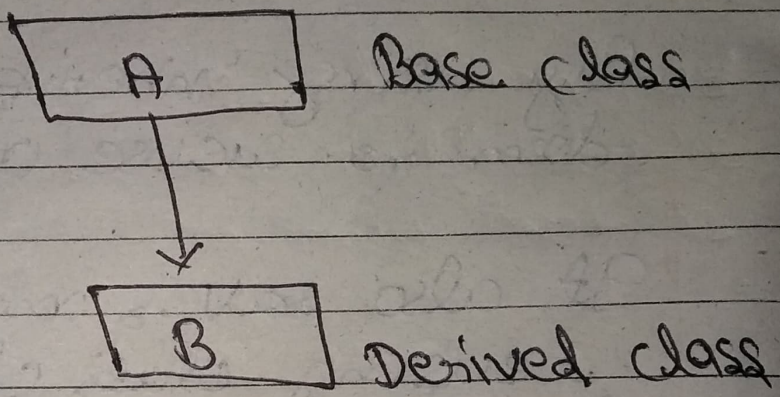
It also saves money & time as reusing older class didn't require testing & debugging again.

If a derived class inherited from base class then we can add some more features in that derived class, or one derived class can inherite from more than one base class, and in this condition ~~base cla~~ derived class will have feature of both base classes.

Types of inheritance :-

i) Single inheritance :-

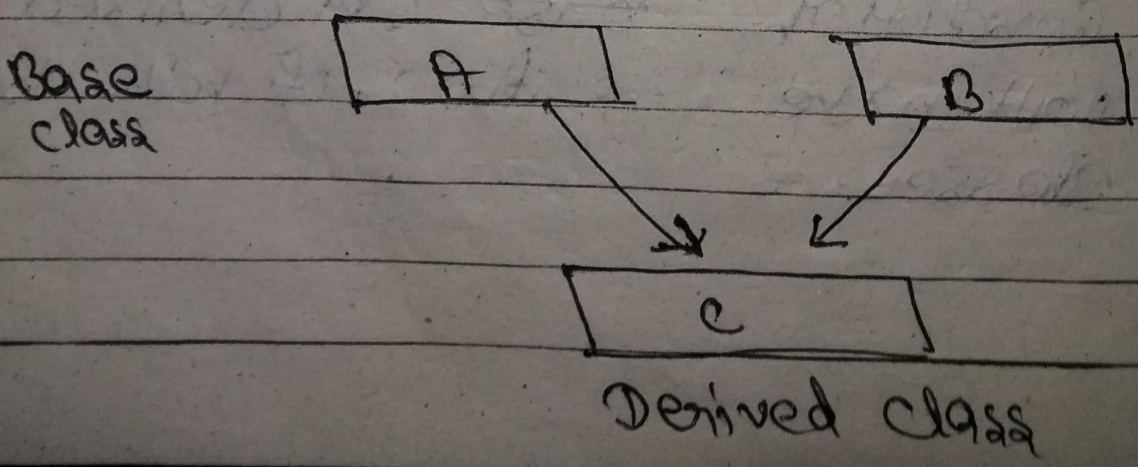
one base class one derived class.



ii) Multiple inheritance :-

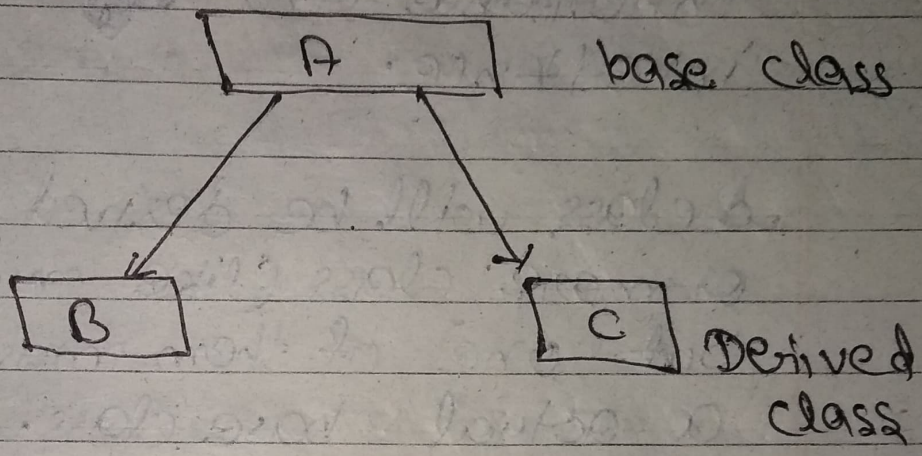
~~Two derived class~~

One derived class from more than one base class.



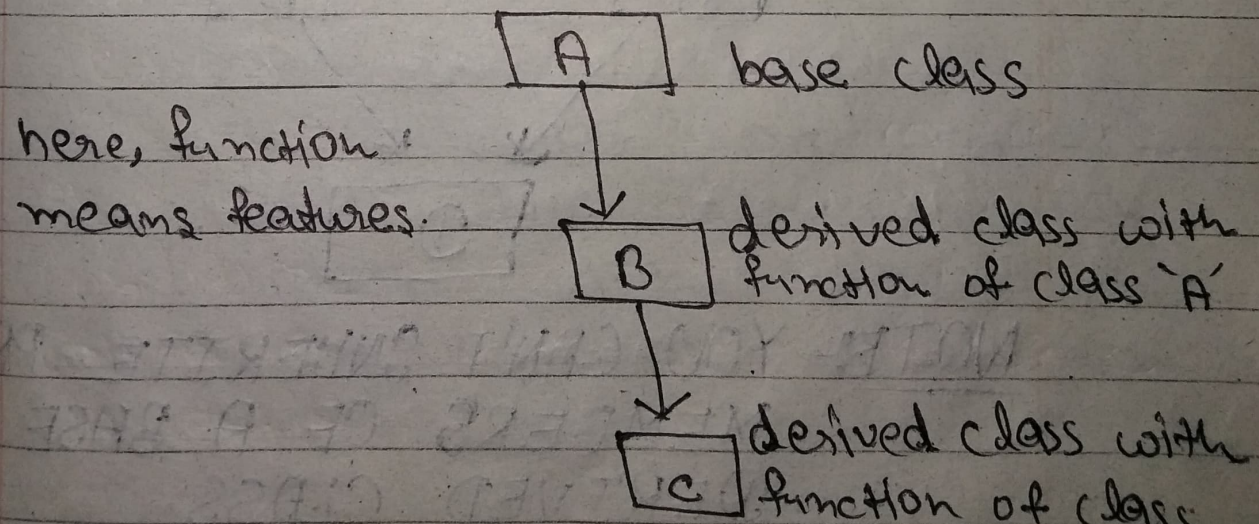
iii) Hierarchical inheritance :-

More than one derived class ~~from~~ from single base class.



iv) Multilevel inheritance :-

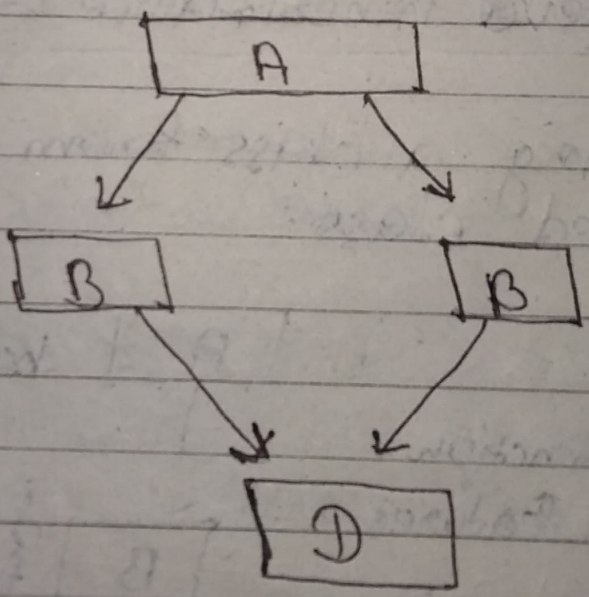
Deriving a class from already derived class.



V) Hybrid inheritance :-

Hybrid inheritance is combination of multiple inheritance, multilevel inheritance and heirarchical inheritance.

A class will be derived from two or more class (like multiple inhe.) but one of them must not be a actual base class.



NOTE:- YOU CAN'T INHERITE PRIVATE MEMBERS OF A BASE TO DERIVED CLASS.

→ Single inheritance demostration & syntax :-

Derived class syntax for single inheritance:

class {{Derived class name}} : {{Visibility mode}} {{base class name}}

Here, there are two types of Visibility mode, public & private.

public visibility mode imphesis that public member of base class will be inherited in derived class as public member.

~~protected~~

private visibility mode imphesis that public member of base class will be inherited in derived class private member.

Also default visibility mode is private.

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

class Employee
{
   public:

      int id;
      float salary;
      Employee(int inpId)
      {
         id = inpId;
         Salary=34.0;
      }
      Employee(){}

};

class programmer : public Employee
{
   public:
```

```
int language_code;

programmer(int inpId)
{
    id = inpId;
    language_code = 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;
}
```

⇒ Protected access modifier and visibility mode :-

While declairing member functions and data members we can use access modifier protected as private or public.

As the name suggest protected, data-members and member functions defined in this can only be accessed by member functions like private. and we can't directly invoke them.

The only difference b/w private access modifiers and protected access modifier is that, that private members can't be inherited but protected members can be inherited.

About protected visibility mode, we can say that if we inherited a class protectedly then private

members will not be inherited, protected members will be inherited as protected and public members will be inherited as protected.

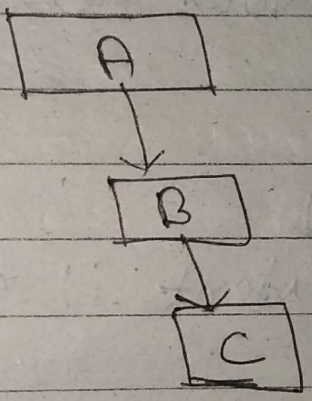
To summerise what happens to all members when they inherited we make a chart.

Members	Public mode	Private mode	Protected mode
Private	NOT INHERITED	NOT INHERITED	NOT INHERITED
protected	protected	Private	protected
Public	Public	Private	protected

→ Multilevel inheritance and its demonstration / Syntax :-

Syntax :-

First, one class 'B' is inherited from 'A' and then class 'C' is inherited from B.




```
Class B : {{visibility mode}} A
{
    // code;
};
Class C : {{visibility mode}} B
{
    // code;
};
```

```
#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<<" Roll number is" << roll_
                    number<<endl;

}
```

```
class exam : public student
{
    Protected :
        float phy;
        float maths;
    public :
        void set_marks(float , float);
        void get_marks(void);
};

void exam :: set marks (float m, float p)
{
    maths = m;
    phy = P;
}

void exam :: get_marks()
{

    cout<<" marks in maths are :" <<
                                maths<<endl;
    cout<<" marks in physics are :"<<phy<<endl;

}
```

```
class result : public exam
{
  public:

    void display_result()
    {
        get_roll_number();
        get_marks();

      cout<<" your result is"<<(maths +
                                phy)/2<<endl;

    }
};


int main()
{
   result harry;

    harry.set_roll_number(420);
    harry.set_marks(96,99);
    harry.display_results();
     return 0;
}
```

→ Multiple inheritance :-

Class {{derived class name}} : {{visibility mode}} base 1 , {{visibility mode}} bass 2
{

    //code;

};

//You can inherite from as many class you want by adding them by comma, as we have done with these two base classes.

Q. WAP to do to these tasks described here.

1. Create two classes

    1) Simplecalculator - takes input of 2 numbers using a utility function and performs +, -, *, / and display result using another function.

ii) ScientificCalculator - Takes input of 2 numbers using a utility functio and performs any four scientific operation of your choice and display the result using another function.

Create another class HybridCalculator and inherit it using those 2 classes. Create object of HybridCalculator and display result of simple and Scientific Calculator.

```
#include <iostream>
#include <math.h>
using namespace std;

class SimpleCalculator
{
  protected:
     int x, y, ch;
     float result;
  public:
     SimpleCalculator
     void getin (void);
```

```
    void calculate(void);
    void show (void);
};

void simpleCalculator :: getin()
{
    cout<<" input two integers on which
            you want to apply simple
            airthmetic operations"<<endl;
    cin>> x >>y;

    cout<<" input 1 to add "<<endl
        <<" input 2 to subtract"<<endl
        <<" input 3 to multiply"<<endl
        <<" input 4 to devide"<<endl;
    cin>> ch;

}
    case 1:

void simpleCalculator :: calculate()
{
```

```
switch (ch)
{
  case 1:
  {
      result = x+y;
      break;
  }
case 2:
{
  result = x-y;
    break;
}
case 3:
  {
      result = x*y;
      break;
  }
case 4:
{
    result = x/y;
    break;
}
default
{
  cout<<"invalid input"<<endl;
  break;
}
}
```

```
void simpleCalculator::show()
{
    cout<<" Result of the operation of "
        <<a<<" and "<<b<<" is "<<result
        <<endl;
}

class ScientificCalculator
{
  protected:
    int a, b, k;
    float result1, result2;
  public:
    void getdata();
    {
      cout<<" Input two integers on which
            you to apply scientific
            arthmetic"<<endl;
      cin>>a>>b;
      cout<<" Input 1 to find log"
          <<endl<<2 to find exponential
          value <<endl<<3 to find
          value of a^b <<endl<<
          4 to find square root <<endl;
```

```
    cin>> k;
}

 void cals(void);
 void shows(void);
};

void ScientificCalculator::cals()
{
    switch(k)
    {
      case 1:
      {
        result1 = log(a);
        result2 = log(b);
        break;
      }
      case 2:
      {
        result1 = exp(a);
        result2 = exp(b);
        break;
      }
```

```
case 3:
{
    result1 = pow(a,b);
    result2 = 0;
}
break;
case 4:
{
    result1 = sqrt(a);
    result2 = sqrt(b);
}
break;
default
{
    cout<<"Invalid input"<<endl;
    break;
}
}
}
void scientificCalculator::shows()
{
    cout<<" Result of operation on "<<a
    <<" and "<<b<<" is "<<result1
    <<" & "<<result2<<endl;
}
```

```
class Hybrid_Calculator : public SimpleCalculator,
public scientificCalculator
{
  Public :
     void display ( )
    {
       calculate ();
       cals ();
    }
};

int main ()
{
    Hybrid_calculator  O;

    O. getin ();
    O. calculate ();
    O. getdata ();
    O. cals ();
    O. display ();

    return 0;
}
```