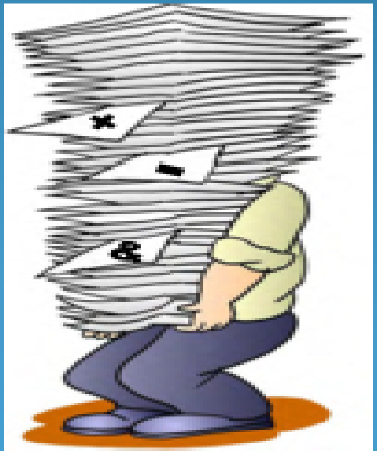
Operator overloading

```
class class_name
{
   -----------
   public: returntype operator sign(args)
            {
               ------------


            }
      -----------------
};
```



Operator overloading

```
int c1,c2,c3;
```

Can you write some this like this?

```
C3=c1+c2;  YES
```

```
class complex
{
            private : float  real;
                      float  iamg;
};
Complex c1,c2,c3;
C3=c1+c2;   X
```

```
Class Time
{
     private: int hr;
                 int min;
    public: void read();
                void print();

};
Time t1,t2,t3;
```

Can we write some thing like this ?

t3=t1+t2; (NO, normally not possible)

With operator overloading u can.

advantages

Using operator overloading we can perform different operations on the same operands.

The advantage to operator overloading is that it makes code much more readable.

operator overloading is used by programmer to make a program more understandable and clear.

For example: you can replace the code like this:

calculation = add(mult(a,b),div(a,b));

with operator overloading

calculation = a*b+a/b;

which is more readable and easy to understand.

Assume you have a class person

```
Class person
{
    private:  int id, age;
                    string name;
    ---------------
};
Person p1(1,20,”rahul”);
Person p2(2,25,”ram”);
```

Say you want to find elder person between two person objects

```
If(p1<p2)
Cout<<“person p1 is elder”
Else
Cout<<“Person p2 is elder”
```

With operator overloading u can.

Operator overloading.

You cannot do any arithmetic or relational operations on objects

```
int a,b,c;
c=a+b;       //for basic Data types.
```

Something like

```
class sample
{
       private : int x,y,z;
}
sample s1,s2,s3;    //objects.
s3=s1+s2;    is not allowed normally.
```

we make this possible by means of operator overloading.

What is operator overloading ?

The mechanism of c++ that permits to add two variables of user defined types with the same syntax that we use for basic data types is called operator overloading.

Operator overloading function

```
returntype classname ::  operator opr(arg_list)
{
   //operations to be performed
}
```

Syntax of calling overloaded operator functions

for unary operators(++,--)

opr object (or) object opr

Ex: ++N

for binary operators(+,*,<,> ....)

object1 opr object2

Ex : c1+c2;

Overloading unary operators :- (++,--)

Write a program to increment a number using operator overloading

----

```
class Number
{
   private: int x;
   public: void read();
          void operator++();
          void print();
};
void Number::read() {
     cout<<"enter a no";
     cin>>x;
 }
void Number::operator++()
{
     ++x;
}
void Number::print() {
   cout<<x;
}
```

```
int main()
{
   Number N;
   N.read();
   ++N;
   N.print();
   return 0;
}
```

N is responsible for invoking member function operator++()

Let us compare

Type 1: in C++

```
void main()
{
    int x;
    cout<<“enter  a No”;
    cin>>x;
    z=increment(x);
     cout<<z;
}
int increment(int x)
{
  ++x;
return x;
}
```

Type 2: in C++

```
Class Number
{
    private: int x;
     public: void read();
                 void increment();
                 void print();
}’;
void Number::read() {
     cout<<“enter a no”;
    cin>>x;
 }
void number :: increment()
{
       ++x;
}
void Number::print() {
   cout<<x;
}
void main()
{
Number N;
N.read();
N.increment();
N.print();
}
```

Type3: with Operator Overloading

```
class Number
{
    private: int x;
    public: void read();
           void operator++();
           void print();
};
void Number::read() {
     cout<<"enter a no";
     cin>>x;
 }
void Number::operator++()
{
     ++x;
}
void Number::print() {
   cout<<x;
}


int main()
{
    Number N;
    N.read();
    ++N;
    N.print();
    return 0;
}
```

Advantages

It makes code more readable.

example: you can replace the code like:

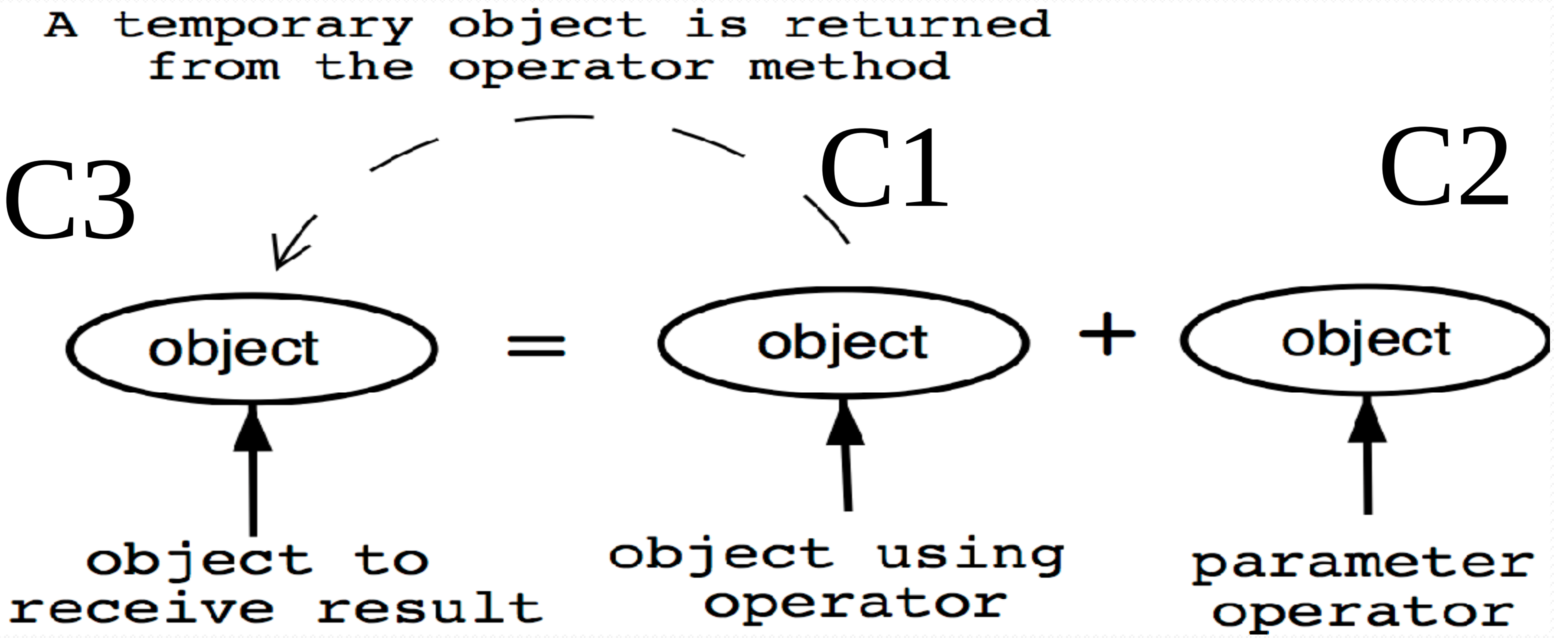
```
calculation =   add(mult(a,b),div(a,b));
```

with operator overloading

```
calculation = a*b+a/b;
```

which is more readable and easy to understand.

Overloading binary operators



C3=C1+c2

Overloading BINARY operators (+,-,*,/)

Program to add 2 complex numbers using operator overloading.

```
class complex
{
                private : float real,imag;
                public :
                     complex (float r, float i)
          {
               real=r;
               imag=i;
          }
                                void print()
          {
                cout<<real<<"+i"<<imag;
          }
          complex operator+(complex c2);
};
```

```
complex complex :: operator+(complex c2)
{
     complex temp(0,0);
                temp.real=real+c2.real;
                temp.imag=imag+c2.imag;
                return(temp);
}

int main()
{
                complex c1(2,3);
                complex c2(4,5);
                complex c3(0,0);
                c3=c1+c2;
                c3.print();
                return 0;
}
```

Overloading binary operators

C3=c1+c2

C3=c1.operator+(c2);

C1 is responsible for invoking the member function

& c2 is passed as parameter

c3=c1.add(c2)

WAP to find elder person among two person objects using operator overloading

Assume you have a class person

```
Class person
{
    private:  int id, age;
                    string name;
     ---------------
};
Person p1(1,20,”rahul”);
Person p2(2,25,”ram”);
```

Say you want to find elder person between two person objects

```
If(p1>p2)
Cout<<“person p1 is elder”
Else
Cout<<“Person p2 is elder”
```

Overloading BINARY operators (+,-,*,/)

Write a Program to add 2 time objects using operator overloading.

```
Class Time
{
      private: int hr;
                   int min;
     public: void read();
                 void print();
                Time operator+(Time t2);
};
Time t1,t2,t3;
T3=t1+t2;
```

```
Class Time
{
      private: int hr;
                    int min;
      public: void read();
                   void print();
              Time operator+(Time t2);

};
Time Time operator+(Time t2)
{
------
----
}
```

```
Int main()
{
          Time t1,t2,t3;
          t1.read();
          t2.read();
          t3=t1+t2;
           t3.print();
 }
```

Lets take an expression.

$$((\underset{c1}{\underline{2+i3}})+(\underset{c2}{\underline{3+i2}}))/(\underset{c3}{\underline{4+i5}})-(\underset{c4}{\underline{3+i2}}))$$

complex c1,c2,c3,c4

complex result;

Using Functions

result=div(add(c1,c2),sub(c3,c4));

With operator overloading

result=(c1+c2)/(c3-c4);

Which is more readable ?

```
int main()
{
    complex c1(2,3);
    complex c2(4,5);
    complex c3(0,0);
    c3=c1+c2;        C3=c1.operator+(c2)
    c3.print();
    c3=c1+2;         C3=c1.operator+(2)
    c3.print();
}
```

```
complex complex ::operator +(complex c2)
{
            complex temp(0,0);
            temp.x=x+c2.x;
            temp.y=y+c2.y;
            return temp;
}
```

```
complex complex:: operator +(int real)
{
            complex temp(0,0);
            temp.x=x+real;
            temp.y=y;
            return temp;
}
```

Rules for overloading operators:

1. Only existing operators can be Overloaded, New operators cannot be created.

2. The overloaded operator must have at least one operand that is of user defined type.

   Example : C=A+B or C=A+2 or C=2+A or ++N

   ++7; wrong, can not be overloaded in this way;

3. We cannot change the Basic meaning of an operator ie., we cannot redefine + operator to subtract one value from another.

Rules for overloading operators

4. There are some operators, that cannot be overloaded sizeof(),

   . , :: , ?:.

   String s1="PES" ;String s2="IT"

   String s3=S1.S2 //Wrong , because dot(.) operator can not be overloaded.

   String S3=S1+S2 // correct

5. When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.

   c3=c1+2; // correct

   c3=2+c1 //wrong

Why is it called operator overloading ?

```
int main()
{
   Complex A,B,C;  int x,y,z;
    C=A+B;  //adding two user defined objects
   C=2+B; //adding constant and user defined object
   C=A+4; //adding user defined and constant
    z=x+y;   //adding built in types
}
```

So + operator is having many forms; it is behaving in many ways; so it is called operator overloading

Overloading :Functions having many forms

Add()

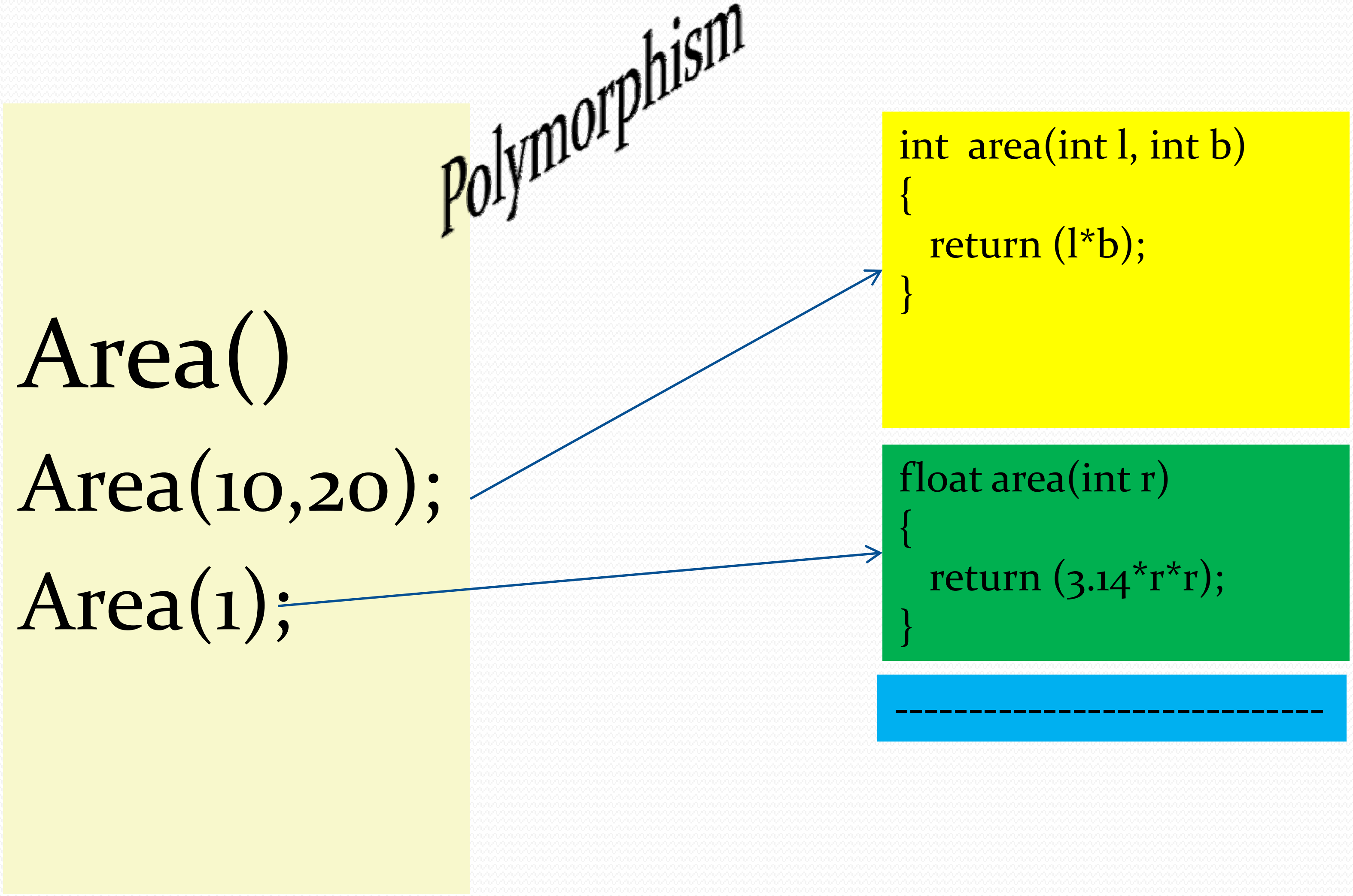
Polymorphism

```
int add(int x, int y)
{
   return (x +y);
}
```

```
int add(int x, int y,int z)
{
   return (x +y+z);
}
```

```
float add(float x, float y)
{
   return (x +y);
}
```

Functions having many forms



Polymorphism

```
Area()
Area(10,20);
Area(1);
```

```
int  area(int l, int b)
{
   return (l*b);
}
```

```
float area(int r)
{
   return (3.14*r*r);
}
```

Functions having many forms

Polymorphism

abs()

```
int abs(int i)
 {
    cout << "Using integer abs()\n";
    return i<0 ? -i : i;
}
```

```
double abs(double d)
{
   cout << "Using double abs()\n";
   return d<0.0 ? -d : d;
}
```

```
long abs(long l)
{
   cout << "Using long abs()\n";
   return l<0 ? -l : l;
}
```

Operator overloading example for Polymorphism

```
int main()
{
   Complex A,B,C;  int x,y,z;
    C=A+B;  //adding two user defined objects
   C=2+B; //adding constant and user defined object
   C=A+4; //adding user defined and constant
    z=x+y;   //adding built in types
}
```

So + operator is having many forms;

Polymorphism

Write a program to create a class called Location with Latitude and Longitude as variables. Implement the following program

```
int main()
{
    Location loc;
    ++loc;
    Print(loc)
    }
```

```
Location operator++()
   {
      latitude++;
      longitude++;

      return(*this);

   }
```

Inheritance

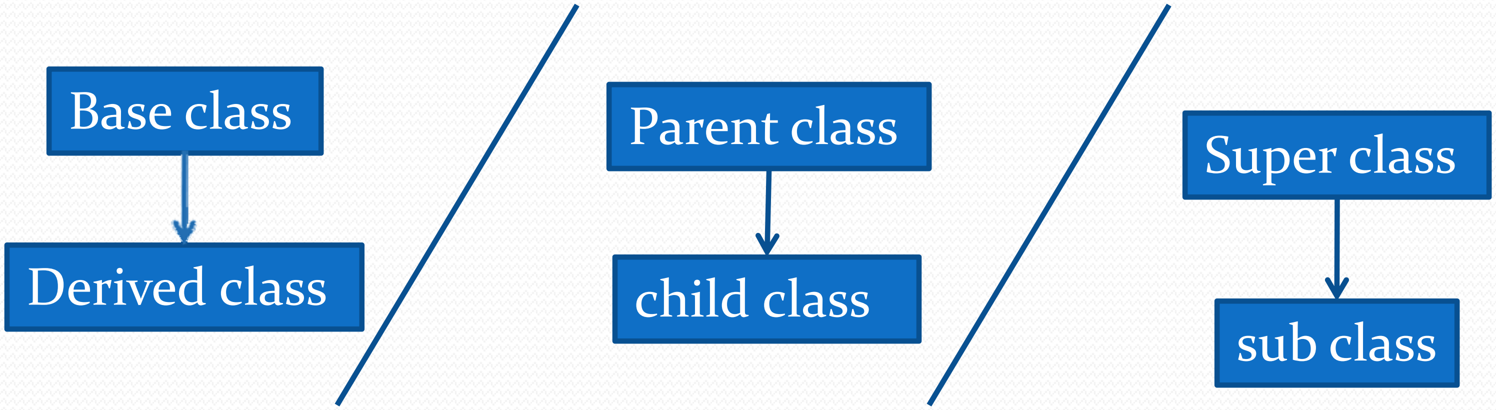
Child inherit properties from father
Derived class inherits properties from its base class

Inheritance

- Inheritance is the process by which one object can acquire the properties of another object.
- Mechanism of deriving a new class from an existing one is called inheritance or derivation.
- It supports the concept of classification

Inheritance:

The old class is referred to as the base class and the new one is called the derived class.



Advantages of inheritance:

1. Code reusability.

2. Saves your time, money and effort.

3. It reduces the burden.

4. Reduces the code size.

5. Debugging is easier.

Types of Inheritance:

1) Single Inheritance.

2) Multiple Inheritance.

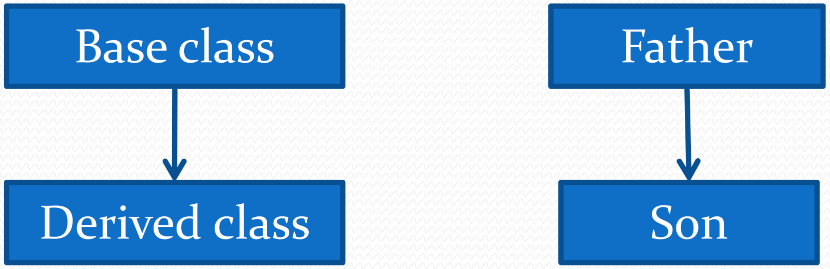
3) Multilevel Inheritance.

4) Hierarchical Inheritance.

5) Hybrid Inheritance.

1) Single inheritance

One base class and one derived class



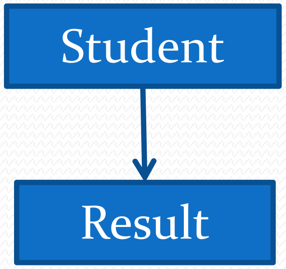
Base class

Derived class

Father

Son

Example:

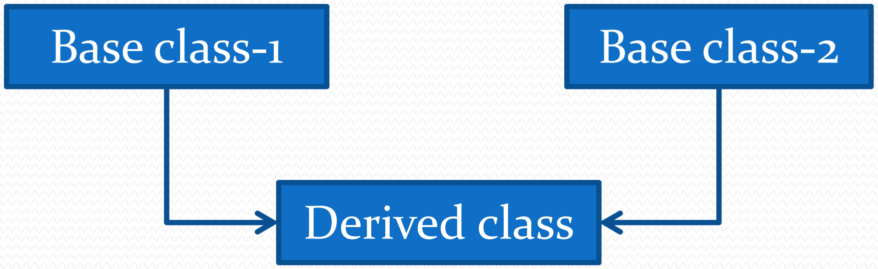


Student

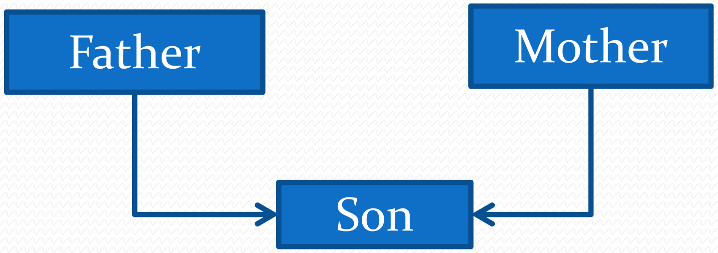
Result

2) Multiple inheritance

a class can inherit the properties of two or more classes, so a derived class is created from multiple classes.



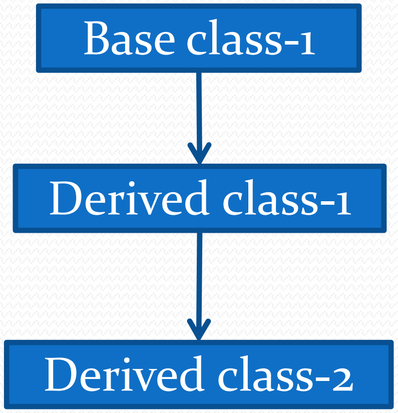
Example:

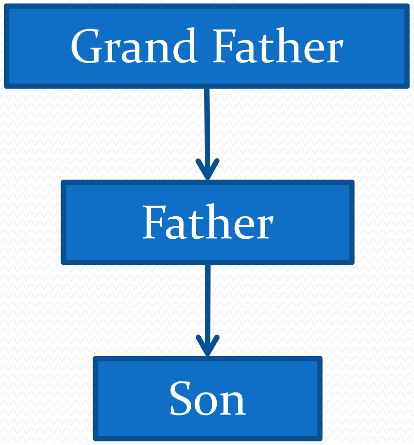


3) Multi level Inheritance.

Derived class is created from a derived class

Example:





Defining derived class/ Base Class Access Control

```
class   derived_class_name  :  access mode    base class_name
{
      //members of the Derived class.
};
```

Ex:

```
class  son: private father
{
   members of the class Result
}
```

Visibility mode: It is optional, either private or public.

(Visibility mode specifies whether the features of the base class are privately derived or publicly.)

Example: Base Class Access Control

```
class ABC : private XYZ  //private derivation
{
   members of ABC;
};

class ABC: public XYZ  //public derivation
{
   members of ABC;
};
class ABC: protected XYZ //protected derivation
{
   members of ABC;
};

class ABC : XYZ                    //private by default.
{
---------
};
```

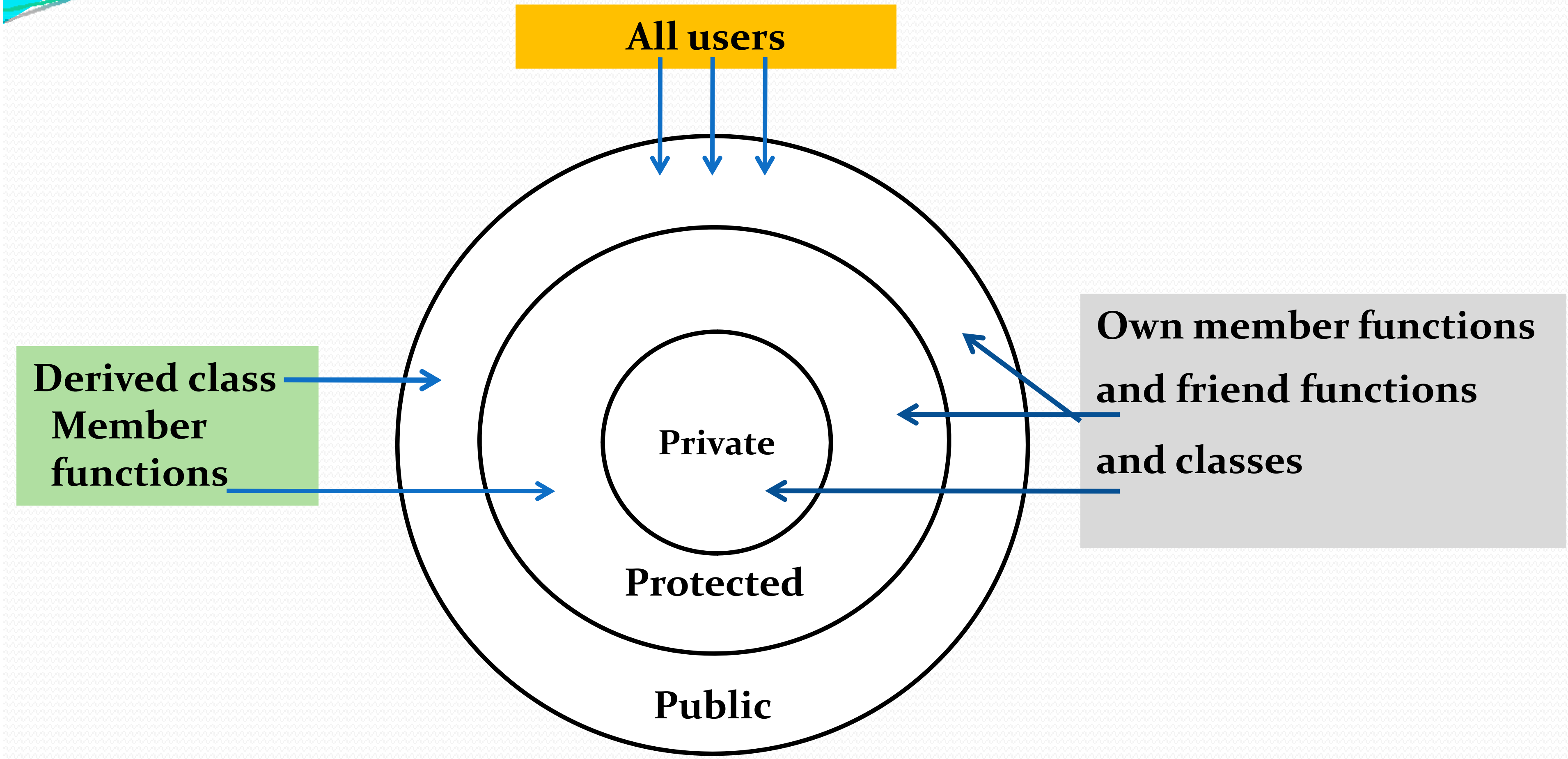
Visibility of inherited members:

OR

Access Control and Inheritance:

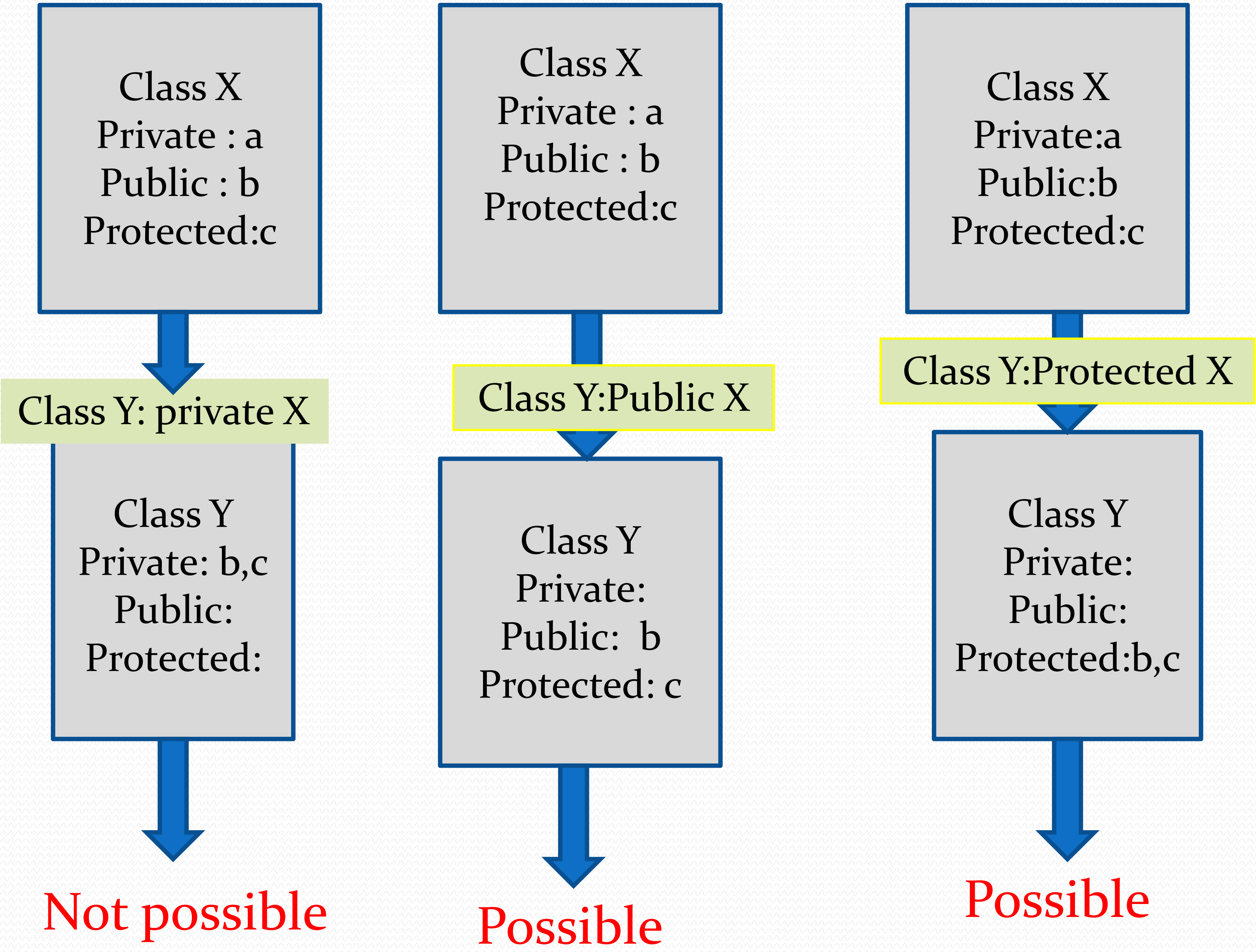
Base class visibility	**Derived class visibility**		
	Public Derivation	**Private Derivation**	**Protected Derivation**
Private →	Not inherited	Not inherited	Not inherited
Protected →	Protected	Private	Protected
Public →	Public	Private	Protected

A simple view of access control to the members of a class:



Access Specifiers with inheritance

Private members will not be Inherited.



Class X
Private : a
Public : b
Protected:c

Class Y: private X

Class Y
Private: b,c
Public:
Protected:

Not possible

Class X
Private : a
Public : b
Protected:c

Class Y:Public X

Class Y
Private:
Public: b
Protected: c

Possible

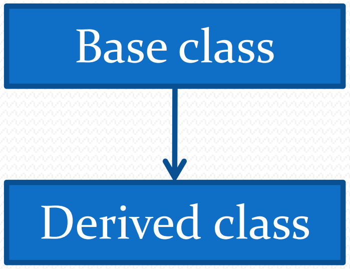
Class X
Private:a
Public:b
Protected:c

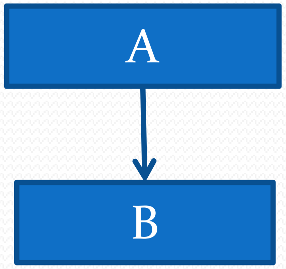
Class Y:Protected X

Class Y
Private:
Public:
Protected:b,c

Possible

WAP to show single inheritance





Create a class student to read and print student details

```
Student
  regno,name,
  Read();
  Print();
```

After 3 months; after all the IA tests, director says, can you announce test marks ?

```
Test
  int T1marks,T2marks;
   ReadMarks();
   FindTotal();
   PrintMarks();
```

Implement single inheritance

Existing class
Created By Programmer 1

```
using namespace std;
class student{
   protected:       int regno;
                    char name[10];

  public:          void readData();
                  void printData();

};   // end of the class definition
void student::readData()
{
    cout<<"enter student details like regno,name";
    cin>>regno>>name;
}
void student::printData()
{
   cout<<"student details are";
  cout<<regno<<name;
}
```

Existing class
Created By Programmer 1

Derived class Programmer 2

```
using namespace std;
class test:public student
{
    private: int t1,t2,esa,total;
    public:void readMarks();
           void printMarks();
};
void test::readMarks(){
cout<<"enter cbt1,cbt2                    ";
    cin>>t1>>t2>>esa;
}
void test ::printMarks(){

 cout<<(t1+t2+esa);
}
int main(){
    test t;
    t.readMarks();
    t.printMarks();
    return 0;
}
```

Reusing

Reusing

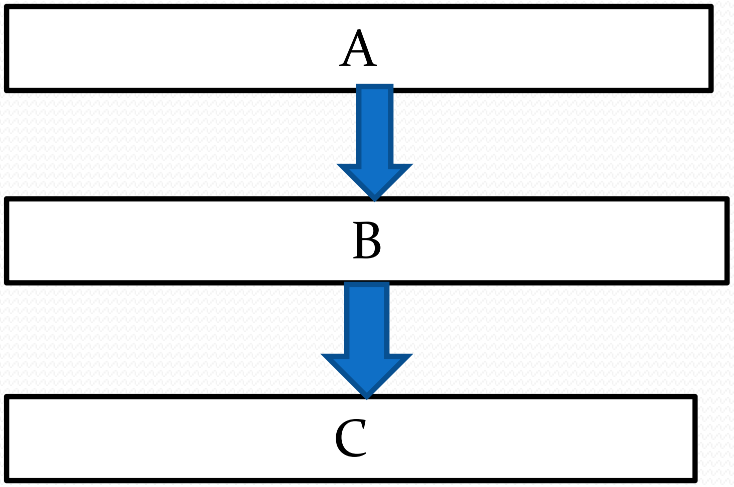
Existing class
Created By Programmer 1

```
using namespace std;
class student{
   protected:       int regno;
                      char name[10];

  public:          void readData();
                     void printData();

};   // end of the class definition
void student::readData()
{
    cout<<"enter student details like
     regno,name";
    cin>>regno>>name;
}
void student::printData()
{
   cout<<"student details are";
  cout<<regno<<name;
}
```

Derived class
Programmer 2

```
using namespace std;
class test:public student
{
    private: int t1,t2,esa,total;
    public:void readMarks();
           void printMarks();
};
void test::readMarks(){
    readData();              Reusing
    cout<<"enter cbt1,cbt2 and ESA marks";
    cin>>t1>>t2>>esa;
}
void test ::printMarks(){
 printData();                 Reusing
 cout<<(t1+t2+esa);
}
int main(){
    test t;
    t.readMarks();
    t.printMarks();
    return 0;
}
```

Multi level Inheritance: Derived class is created from a derived class



Existing class
Created By Programmer 1

```
class student
{
   protected:       int regno;
                        char name[10];
                        float percentage;

  public: void readStudentDetails();
       void printStudentDetails();

};   // end of the class definition
void student::readStudentDetails()
{
    cout<<"enter student details";
    cin>>regno>>name>>percentage;
}
void student::printStudentDetails()
{
   cout<<"student details are";
  cout<<regno<<name<<percentage;
}
```

Existing class
Created By Programmer 2

```
class test:public student
{
    private: int testmarks;
    public:void readTotalIAMarks();
           void printIAMarks();
};
void test::readTotalIAMarks()
{
    cout<<"enter test marks";
    cin>>testmarks;
}
```

Derived class
By   Programmer 3

Input

Regno,name,total IA marks,total final marks

Ex:

```
123 Ram      190/200      700/800
```

Output

```
123 Ram 80
```

Existing class
Created By Programmer 1

```
class student
{
  protected:      int regno;
                     char name[10];
                     float percentage;

 public: void readStudentDetails();
      void printStudentDetails();

};   // end of the class definition
void student::readStudentDetails()
{
    cout<<"enter student details";
    cin>>regno>>name>>percentage;
}
void student::printStudentDetails()
{
   cout<<"student details are";
  cout<<regno<<name<<percentage;
}
```

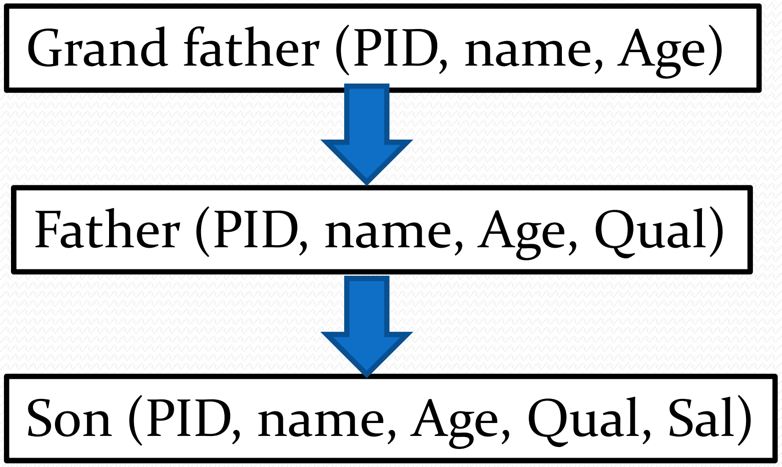
Existing class
Created By Programmer 2

```
class test:public student
{
   private: int testmarks;
   public:void readTotalIAMarks();
         void printIAMarks();
};
void test::readTotalIAMarks()
{
   cout<<"enter test marks";
   cin>>testmarks;
}
```

Derived class
By   Programmer 3

```
#include "st.cpp"
#include "test.cpp"

class Result:public test
{
   int marks;int perg;
   public: void readTotalExternalMarks() {
         cout<<"enter final marks";
         cin>>marks;
         }
         void findPerg()          {
            marks=marks+testmarks;
            perg=(marks*100)/800;
         }
         void printPerg()          {
            cout<<"   Per=    "<<perg<<endl;
         }
};

int main()
{
    Result r;
    r. readStudentDetails();
    r.readTotalIAMarks();
    r. readTotalExternalMarks();
    r.findPerg();
    r.printStudentDetails();
    r.printPerg();
   return 0;
}
```

Write a pgm to illustrate multilevel inheritance for



Grand father (PID, name, Age)

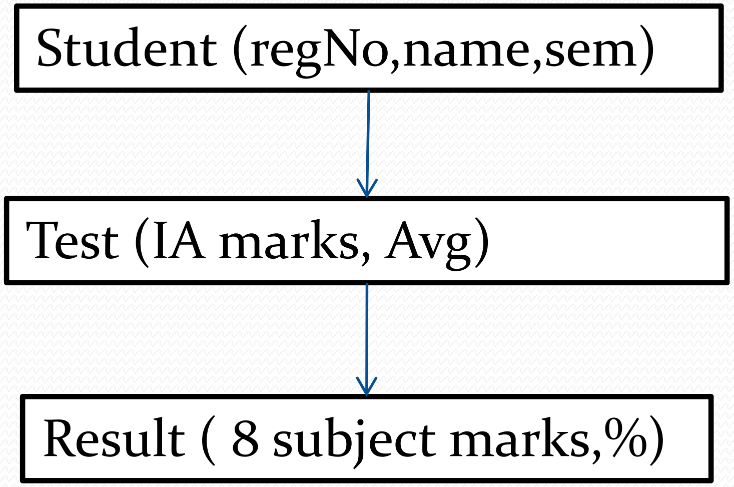
Father (PID, name, Age, Qual)

Son (PID, name, Age, Qual, Sal)

Inheritance

why to create so many classes ?

Why can’t we write all variables and functions in one single class ?

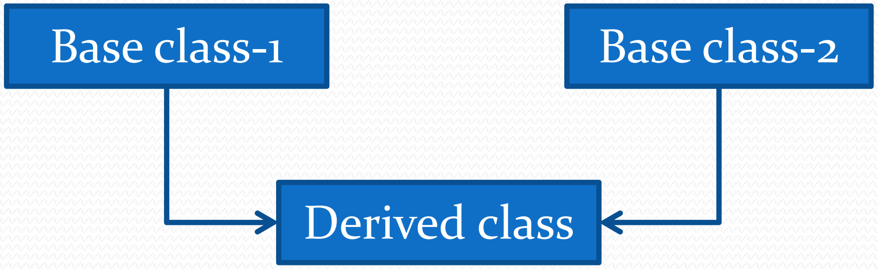


```
class student
{
  protected:      int regno;
                  char name[10];
                  float percentage;
                  int t1,t2,t3; float avg;
                  int totmarks;
                  float per;


  public:       void readData();
                void printData();
                void readTestMarks();
                void readFinalMArks();
                void findPeg();
                void printResult();

};

Student s;
```

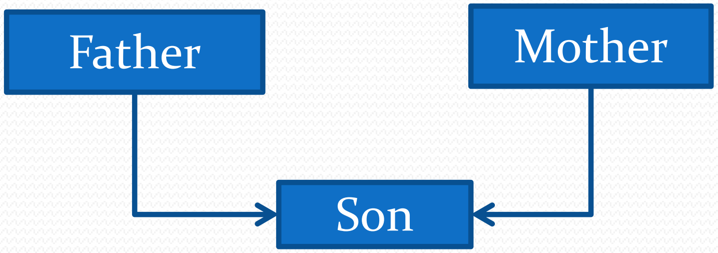
1)At the beginning , we don't need test details
2) Whenever you add a function,
   don't touch the class that is already working
      create new one..

2) Multiple inheritance

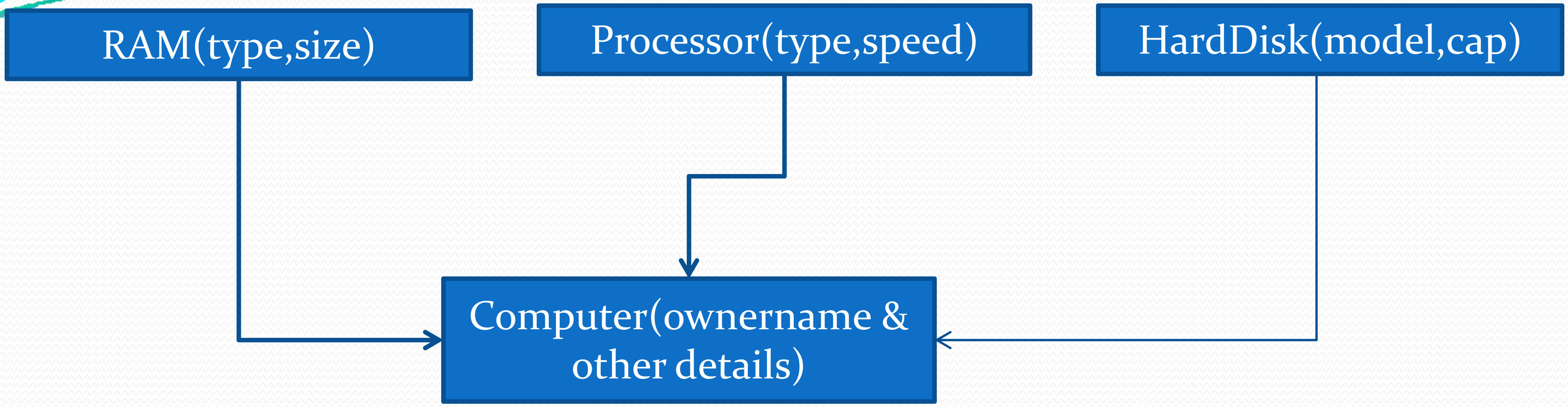
a class can inherit the properties of two or more classes, so a derived class is created from multiple classes.



Example:



Implement multiple inheritance

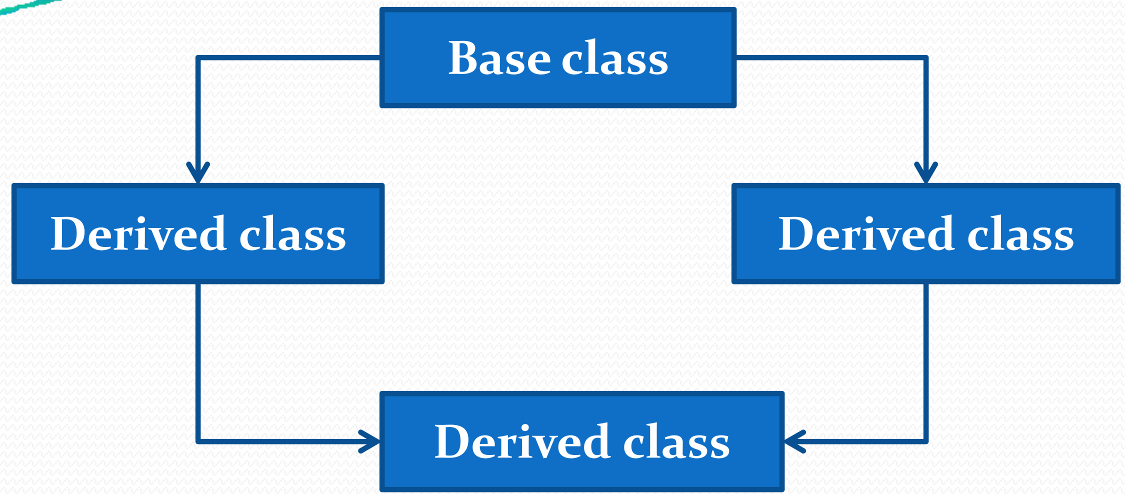


```
Class Computer:public RAM,public Processor,Public HardDisk
{
----
Public: printConfiguration();
};
Main(){
Computer c;
c.readRamDetails();
c.readProcDetails();
c.readHDDetails();
c.printConfiguration();
}
```

```
class RAM           class Processor     class HardDisk
{                   {                   {
-----               ------              ------------
}                   }                   }
```

```
Class Computer:public RAM,public Processor,Public HardDisk
{
----
Public: printConfiguration();
};
Main(){
Computer c;
c.readRamDetails();
c.readProcDetails();
c.readHDDetails();
c.printConfiguration();
}
```

5) Hybrid inheritance:

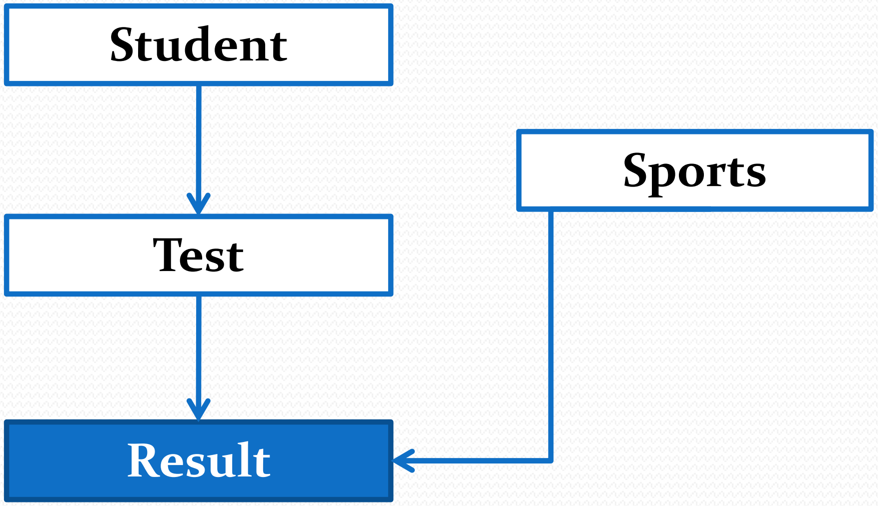


Base class

Derived class

Derived class

Derived class

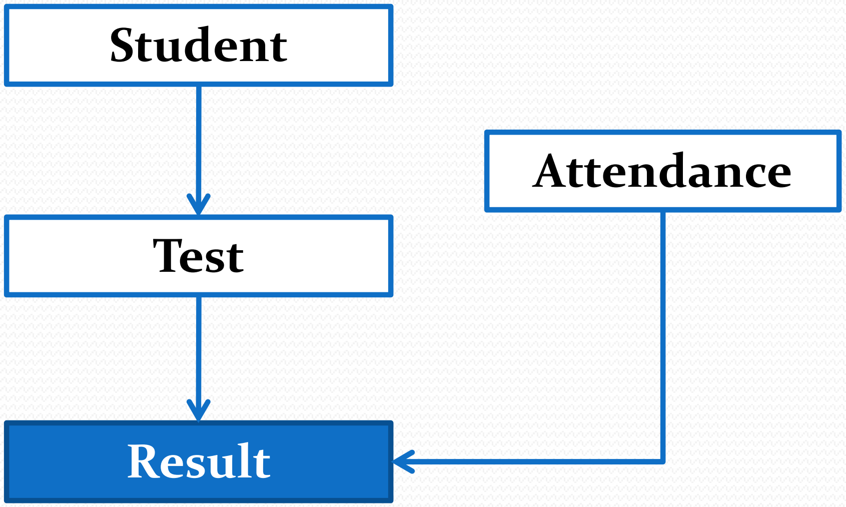


Student

Test

Sports

Result



Student

Test

Attendance

Result

What happens if there are identical functions in Base class and derived class

```
class A
{
  public: void print()
              {
                  cout<<"Hello from A";
              }
};

class B:public A
{
  public: void print()
              {
                  cout<<"Hello from B";
              }
};
```

```
int main()
{
    B objb;
    objb.print();


    return 0;
}
```

Function overriding

Ambiguities in multiple inheritance:

1. Identical members in more than one Base class.
2. Diamond shaped inheritance.

1) Identical members in more than one Base class.

```
class A
{
        public: void print()
                        {
                                cout<<“class A”;
                        }
};
class B
{
        public: void print()
                        {
                                cout<<“class B”;
                        }
};
class C: public A, public B
{
};
```

```
void main()
{
c c1;
c1.print();             //error; Ambiguous call to print.
}
```

To Avoid

use scope resolution operator OR override function print

```
void main()
{
c1.A :: print();
c1.B :: print();
}
```

override function print

ie.,

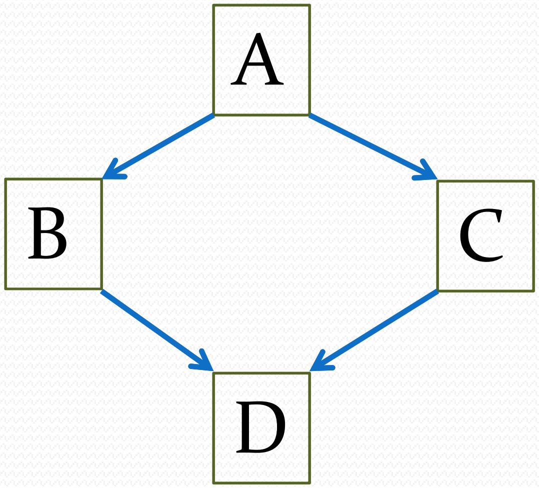
```
class C: public A, public B
{
public: void print()
{
cout<<"class C";
}
}
```

```
void main()
{
c c1;
c1.print();             //no error; after overriding.
}
```

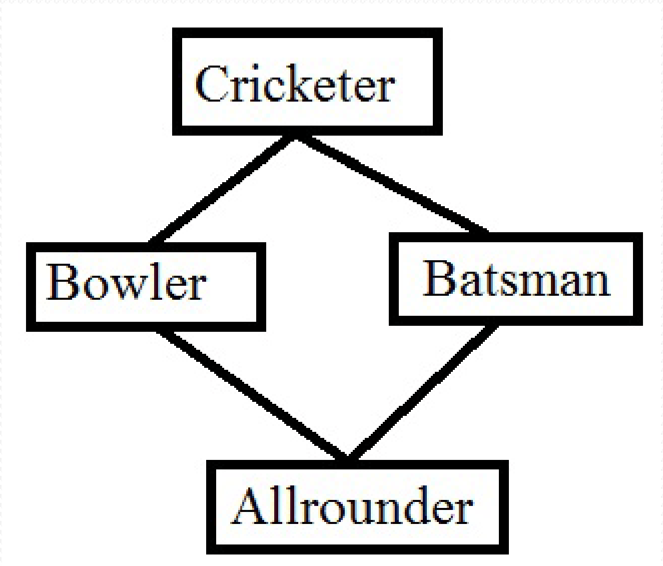
Ambiguities in multiple inheritance:

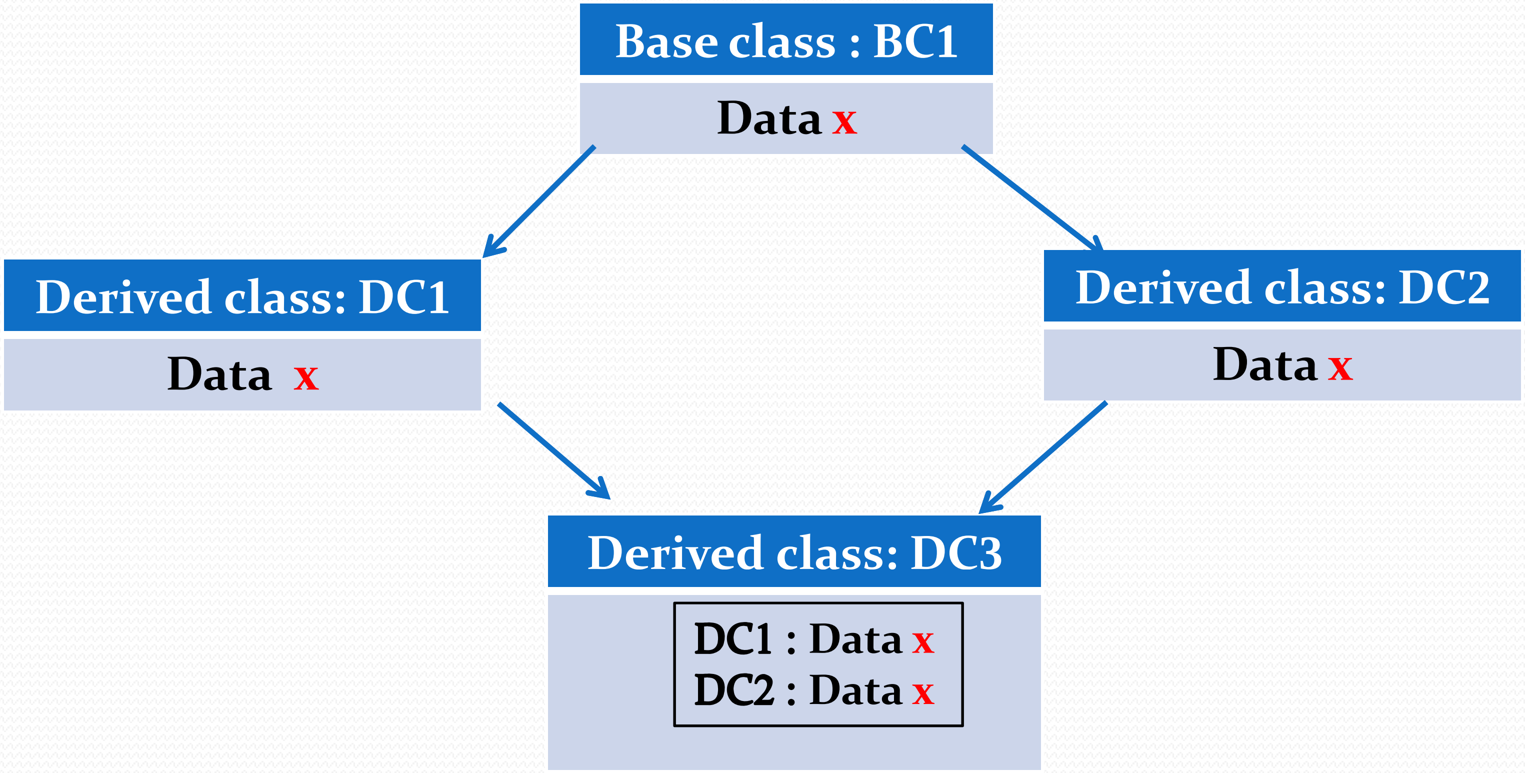
1. Identical members in more than one Base class.
2. Diamond shaped inheritance.

2) Diamond shaped inheritance:



Example





Base class : BC1

Data x

Derived class: DC1

Data x

Derived class: DC2

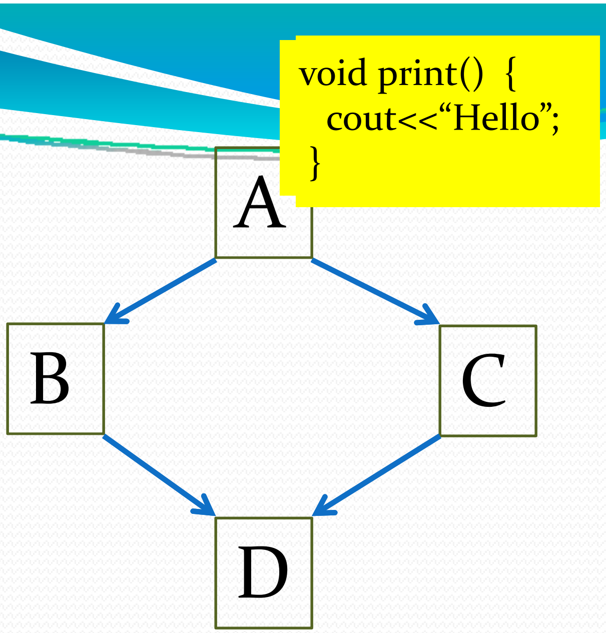
Data x

Derived class: DC3

DC1 : Data x

DC2 : Data x

```
class A
{
     public: void print()
                 {
                    cout<<“Hello”;
                 }
};
class B: pubic A
{
};
class C: public A
{
};
class D: public B, public C
{
};
void main()
{
   D d1;
   d1.print(); //Error
}
```



```
void print()  {
   cout<<“Hello”;
 }
```

To Avoid
use scope resolution operator

OR

Override function print()

OR

Make Virtual Base classes

Virtual base classes:

```
class A
{
   public: void print()
          {
             cout<<"Hello from A";
          }
};
```

```
class B:public virtual A
{

};
```

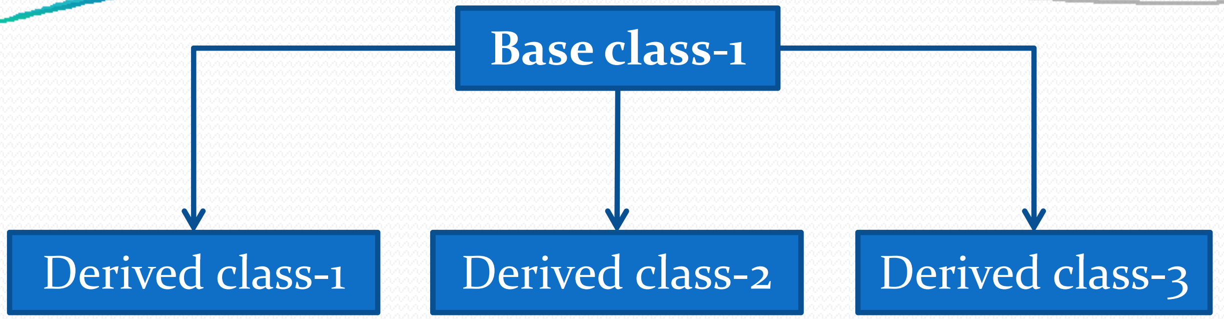
```
class C:public virtual A
{

};
```

```
class D:public B,public C
{

};
```

Duplication of inherited members due to these multiple paths can be avoided by making the common base class as *virtual base class* while declaring the direct or intermediate base classes as shown:

```
int main() {
   D d1;


    d1.print();    // no ERROR


   return 0;
}
```

4) Hierarchical inheritance.

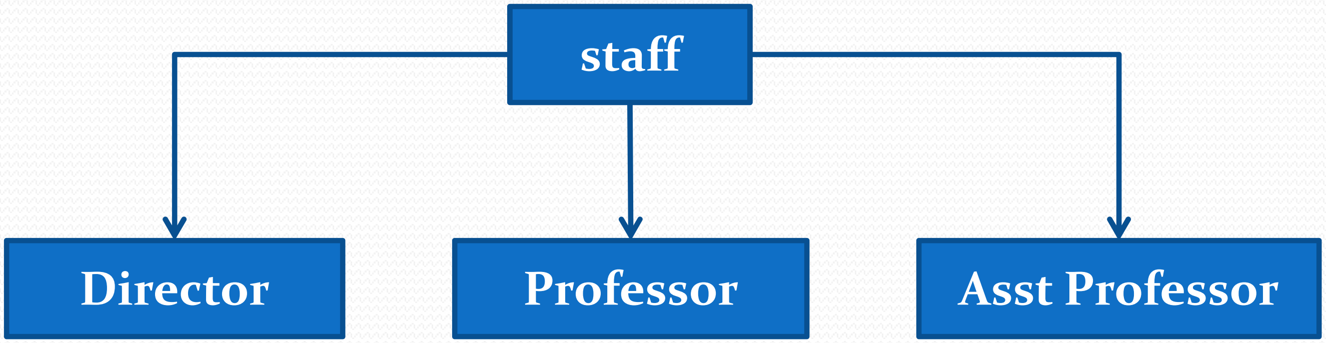


Base class-1

Derived class-1

Derived class-2

Derived class-3

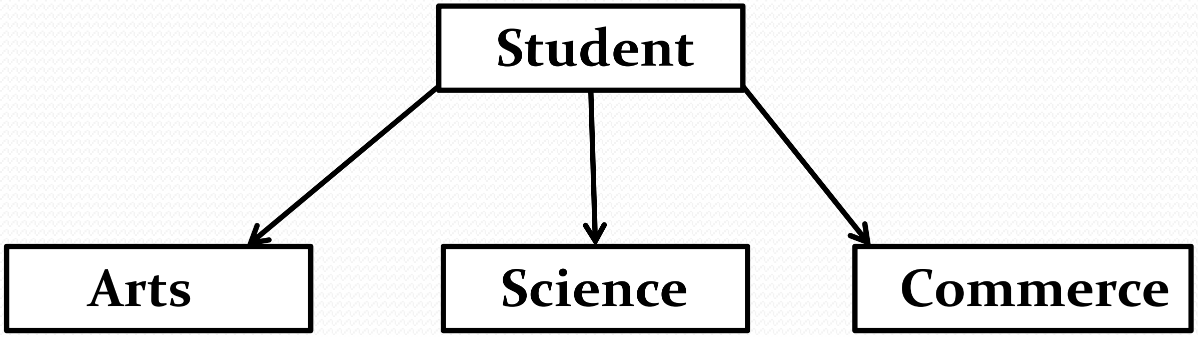


staff

Director

Professor

Asst Professor

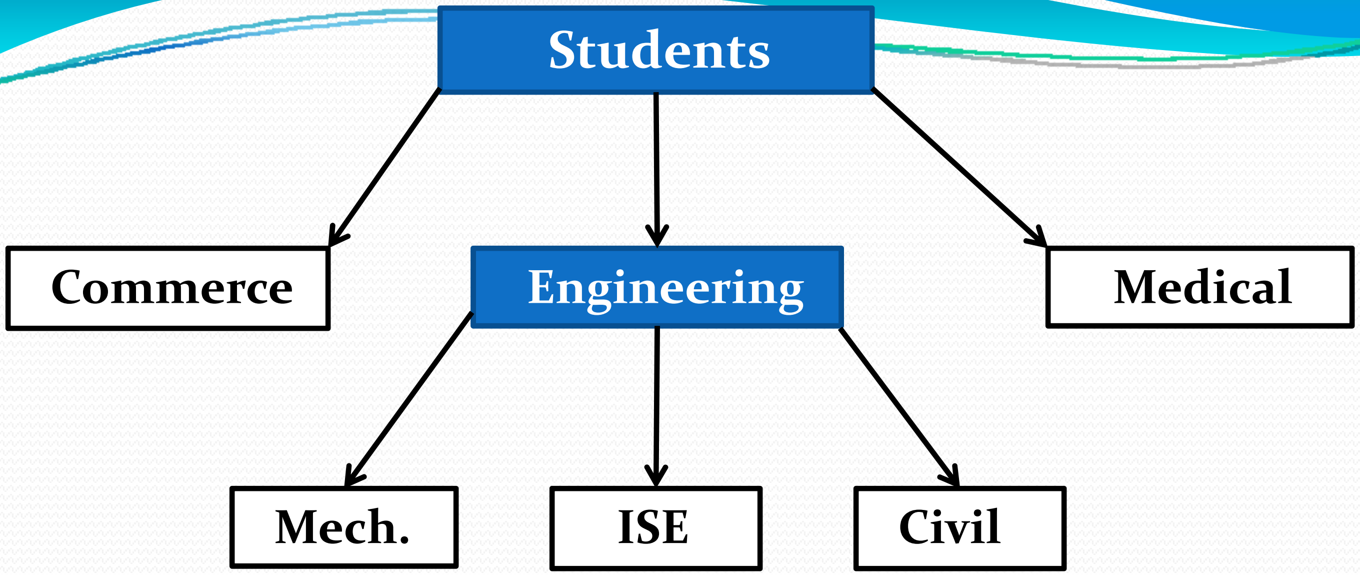


Student

Arts

Science

Commerce



Students

Commerce

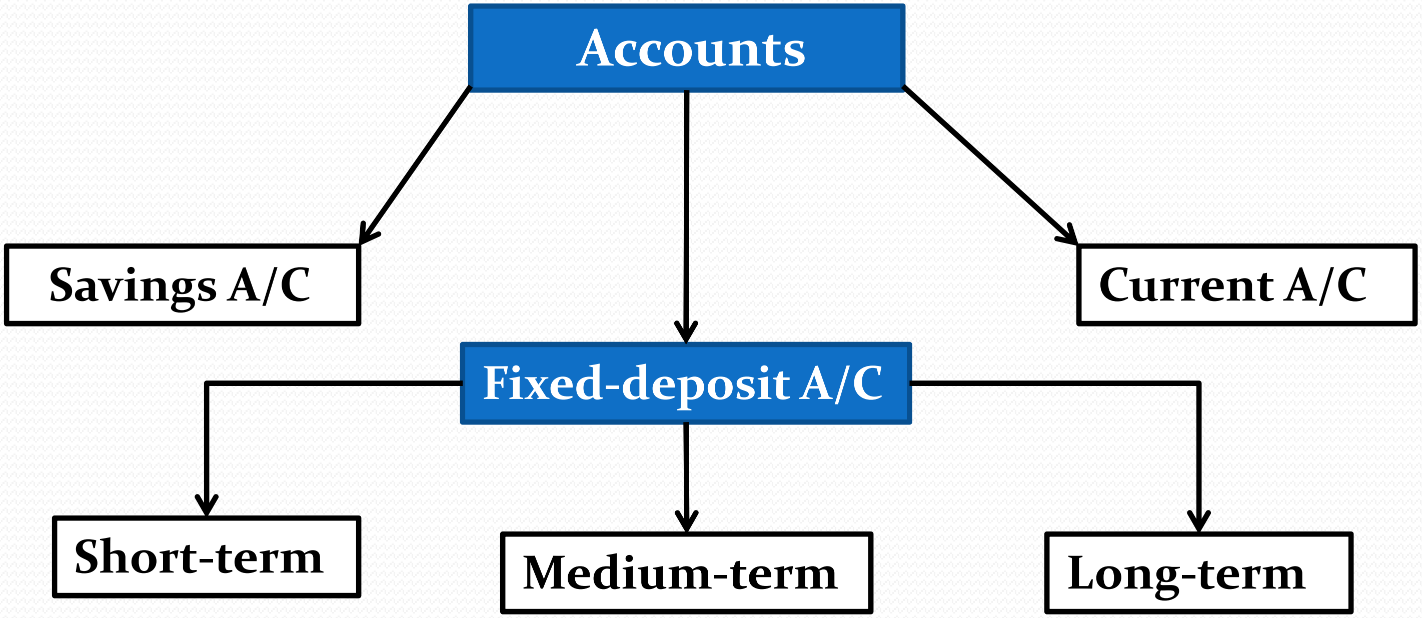
Engineering

Medical

Mech.

ISE

Civil



Accounts

Savings A/C

Current A/C

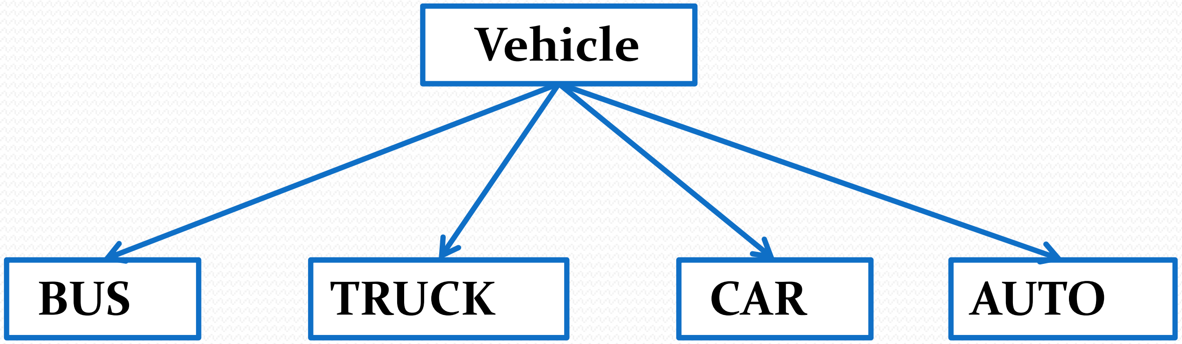
Fixed-deposit A/C

Short-term

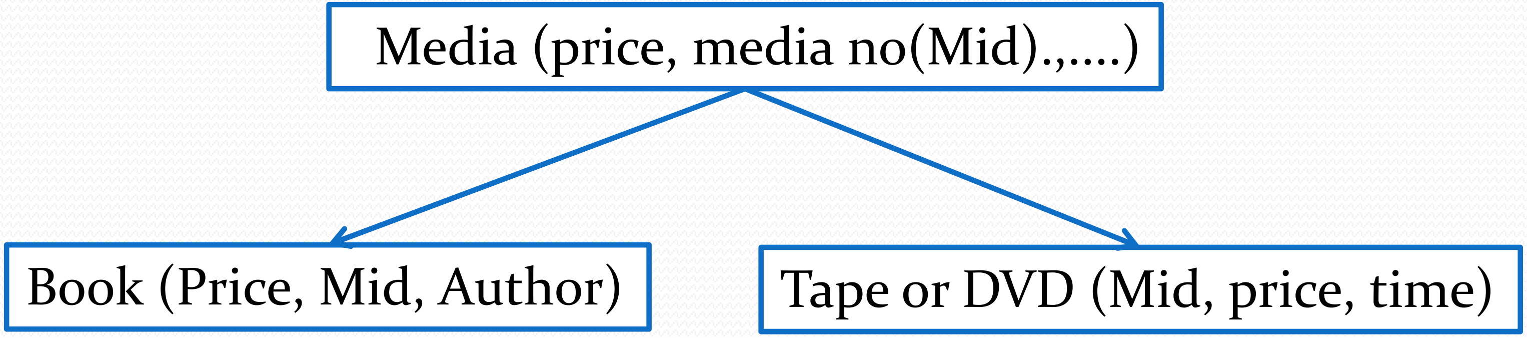
Medium-term

Long-term

Hierarchical inheritance.



WAP to demonstrate hierarchical inheritance for



Media (price, media no(Mid).,....)

Book (Price, Mid, Author)

Tape or DVD (Mid, price, time)

```
class media
          {
                  public: int Mid;
                       float price;
          };
```

```
class book
{
  -----------
};
```

```
class Tape
{
  -----------
};
```

```
class media
   {
     public: int Mid;
             float price;
   };

   class book :public media
   {
   private : char Author[10];
                int nop;
   public :  void read()
               {
                  cin>>Mid>>Author>>nop;
               }
          void print()
          {
                cout<<Mid<<Author<<nop;
          }
   };
```

```
class tape : :public media
{
        private : int time;
        public : void read(){

        cin>>Mid>>price>>time;
        }
        void print(){
            cout<<Mid<<price<<time;
        }
};
```

```
void main()
{
        Book b;  Tape t;
        case 1:
                b.read();  b.print();
                break;
        case 2:  t.read();  t.print();
                break;
}
```

There are 5 Types of Inheritances, They are

1) Single Inheritance(one BC-→ one DC)

2) Multiple Inheritance(more than one BC-> one DC)

3) Multilevel Inheritance(BC->DC1->DC2)

4) Hierarchical Inheritance(1BC-> more than one DC)

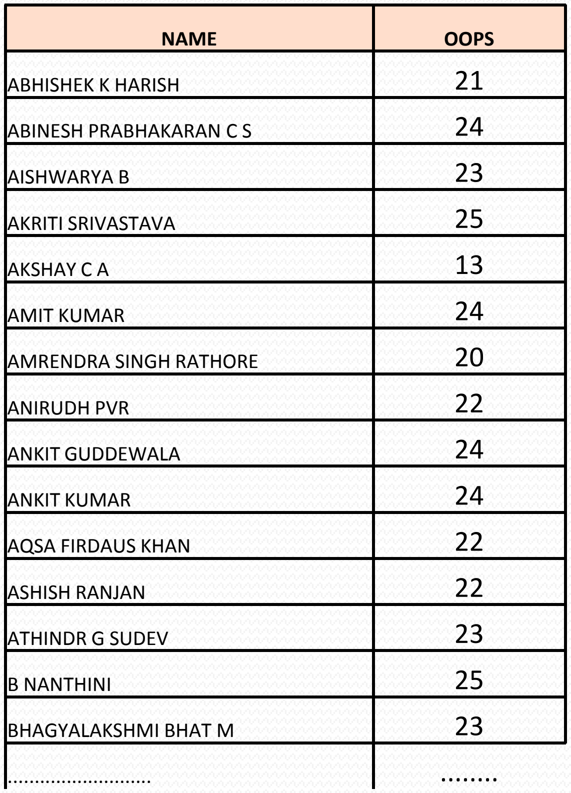
5) Hybrid Inheritance(mixture)

What is the output ?

```
#include <iostream>
using namespace std;
class data{
    private:  int count;
    public:data(){
                     count=0;
               };
       void print(){
          count++;
          cout<<count;
        }
 };
int main(){
      data d1,d2,d3;
      d1.print();  d2.print(); d3.print();
      return 0;
                  cout<<"Total size"<<sizeof(d1)+sizeof(d2)+sizeof(d3);
}
```

Static members in C++

Static data members and functions



NAME	OOPS
ABHISHEK K HARISH	21
ABINESH PRABHAKARAN C S	24
AISHWARYA B	23
AKRITI SRIVASTAVA	25
AKSHAY C A	13
AMIT KUMAR	24
AMRENDRA SINGH RATHORE	20
ANIRUDH PVR	22
ANKIT GUDDEWALA	24
ANKIT KUMAR	24
AQSA FIRDAUS KHAN	22
ASHISH RANJAN	22
ATHINDR G SUDEV	23
B NANTHINI	25
BHAGYALAKSHMI BHAT M	23
....	....

This is only once ,no matter how many objects you create

Avg=total/nos

Static data members

```
class data{
    private: static int count;
    public:data(){
                     count=0;
               };
       void print(){
          count++;
          cout<<count;
        }
 };
int data::count=0;
int main(){
      data d1,d2,d3;
      d1.print();  d2.print(); d3.print();
      return 0;   cout<<"Total size"<<sizeof(d1)+sizeof(d2)+sizeof(d3);
}
```

Static Functions in classes.

A static member function can be called without the class object

The **static** functions are accessed using only the class name

and the scope resolution operator ::

A static member function can only access static data member.

Pgm to show static functions

```
class sample
{
  public:
      static void print()
       {
           cout<<"Hello static";
       }
 };

int main()
{

    sample::print();

    return 0;
}
```

Director is asking to give the class average.

NAME	OOPS
ABHISHEK K HARISH	21
ABINESH PRABHAKARAN C S	24
AISHWARYA B	23
AKRITI SRIVASTAVA	25
AKSHAY C A	13
AMIT KUMAR	24
AMRENDRA SINGH RATHORE	20
ANIRUDH PVR	22
ANKIT GUDDEWALA	24
ANKIT KUMAR	24
AQSA FIRDAUS KHAN	22
ASHISH RANJAN	22
ATHINDR G SUDEV	23
B NANTHINI	25
BHAGYALAKSHMI BHAT M	23
………………………	……..

Avg=total/nos

This is only once ,no matter how many objects you create

```
1: Display Student Info
2: Display Average of the Class
1
Name=Abhishek marks =21
Name=Abinesh   marks=24
Name=Aishwarya marks=23
-----------
1: Display Student Info
2: Display Average of the Class
2
Average of the class is 22
```

……………………………………

Director is asking to give the class average.

NAME	OOPS
ABHISHEK K HARISH	21
ABINESH PRABHAKARAN C S	24
AISHWARYA B	23
AKRITI SRIVASTAVA	25
AKSHAY C A	13
AMIT KUMAR	24
AMRENDRA SINGH RATHORE	20
ANIRUDH PVR	22
ANKIT GUDDEWALA	24
ANKIT KUMAR	24
AQSA FIRDAUS KHAN	22
ASHISH RANJAN	22
ATHINDR G SUDEV	23
B NANTHINI	25
BHAGYALAKSHMI BHAT M	23
....	....

Avg=total/nos

```
class Test
{
    private:
            int marks;
            string name;
    public:
             static int avg;
             static int nos;
            Test(int marks,string name);
             void displayMarks();
             static void calculateAvg();
};
```

This is only once ,no matter how many objects you create

```
class Test
{
    private:
            int marks;
            string name;
    public: void readMarks();
            void displayMarks();
            static float avg;
            static int numOfStudent;
            static void calculateAvg();
};

float  Test::avg;
int  Test::numOfStudent;

void Test::readMarks()
{
     cout<<"Enter name and marrks";
     cin>>name>>marks;
     avg=avg+marks;
     numOfStudent++;
}

void Test:: displayMarks()
{
     cout<<"name= "<<name<<" "<<"marks= "<<marks<<endl;
}

void Test::calculateAvg()
{
     avg=avg/numOfStudent;
}
```

```
int main()
{
    Test t[5];
    int i;
    while(1)
    {
             cout<<"Enter the choice\n 1-> Read Stud info\n"
              "2->display student info\n 3->display average of the class\n";
             cin>>i;
             switch(i)
             {
                          case 1:
                             for(i=0;i<5;i++)
                                        {
           t[i].readMarks();
                                        }
                                        break;
       case 2:

                                        for(i=0;i<5;i++)
                                        {
           t[i].displayMarks();
                                        }
                                        break;
                          case 3:
                                        Test::calculateAvg();
                                        cout<<"Average of the class is : "<<Test::avg<<endl;
                                        break;
                          default:
                                        exit(0);
             }
    }

    return 0;
}
```

Design, develop, and execute a program in C++ based on the following requirements:

An EMPLOYEE class is to contain the following data members and member functions:

Data members:

Employee_Number (an integer),

Employee_Name (a string of characters), Basic_Salary (an integer) , All_Allowances (an integer), IT (an integer), Net_Salary (an integer).

Member functions:

to read the data of an employee,

to calculate Net_Salary and to print the values of all the data members.

(All_Allowances = 123% of Basic; Income Tax (IT) = 30% of the gross salary (= basic_Salary _ All_Allowance); Net_Salary = Basic_Salary + All_Allowances – IT)

Total and avg salary

How would you calculate total_salary paid to all emp
or avg salary of the employee

Static functions

There are some functions in real world entities, they dont belong to any specific object.

Like classavg()

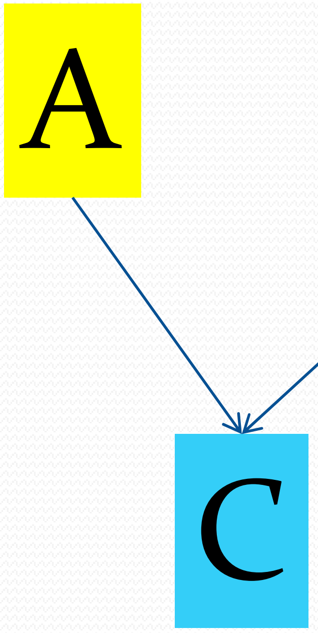
total_salary() or avg_salary() paid to employee;

find_topper();

Such function we make as static ….

Inheritance and Constructors

What is a constructor ?



Calling base class cons through derived class

Inheritance and Constructors

Single Inheritance :Base Constructor then derived Constructor

```
class Alpha
{
   public: Alpha()
        {
          cout<<"hello from alpha"<<endl;
        }
     };
class Beta:public Alpha
{
  public:Beta()
     {
        cout<<"hello from Beta"<<endl;
     }
     };
int main()
{
      Beta b;
}
```

Multilevel inheritance

```
class A
{
  public: A()
        {
          cout<<"hello from a"<<endl;
        }

};
class B:public A
{
 public:B()
      {
         cout<<"hello from B"<<endl;
      }

};


class C:public B
{
  public: C()
        {
          cout<<"hello from c";
         }
};

int main()
{
   C c1;
   return 0;
}
```

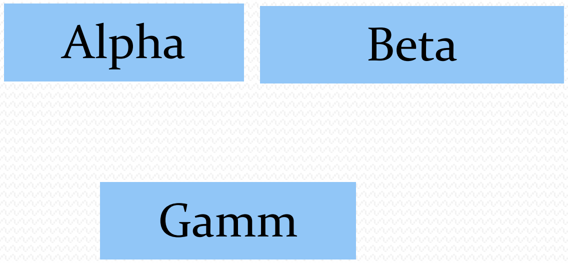
```
Hello from a
Hello from b
Hello from c
```

Multiple inheritance

```
class Alpha
{
  public: Alpha()
        {
          cout<<"hello from alpha"<<endl;
        }
};
```

```
class Beta
{
  public:Beta()
           {
             cout<<"hello from Beta"<<endl;
           }
};
```

```
class Gamma:public Alpha, public Beta
{
  public: Gamma(){
                cout<<"hello from Gamma";
             }
};

int main(){
    Gamma g;
    return 0;
}
```



Constructors in derived classes

```
Class Z:public C,public B,public A
{
}
```

What is the order ?

C

B

A

Z

Passing parameters to base class constructors

```
Derived-constructor(Arglist1):base1( Arglist),base2(arglist) ......base(Arglist)
{

                    Body of derived constructor


}
```

Initializing base class members

```
class Alpha
{
                private: int a;
                public: Alpha(int i)
                  {

                  }
};
```

```
class Beta:public Alpha
{
    private: int b;
    public:Beta() {
                }

};
```

```
class Gamaa:public Beta
{
    private: int g;
    public: Gamaa()
               {
                }
};

Int main()
{
   Gamaa g;
}
```

???

How to Initialize base class members through derived class object?

Single inheritance and init

```
class Alpha
{
   private: int a;
   public: Alpha(int i)   { a=i; }
};
class Beta:public Alpha
{
  private : int b;
  public:Beta(int i,int j):Alpha(i) { b=j;}
};

int main() {
    Beta b(10,20);
    return 0;
}
```

Multilevel inheritance

Init base class members using DC object in Multi inheritance

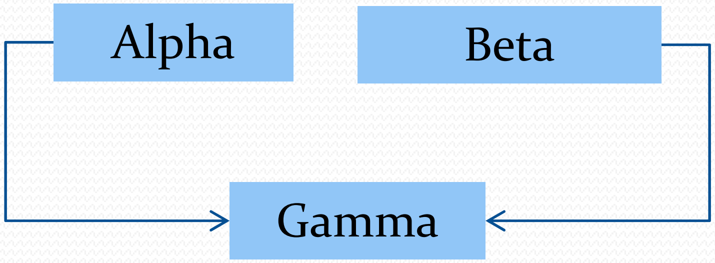
```
class Alpha{
    private: int a;
    public: Alpha(int i)   {
       a=i;
       cout<<"hello from alpha "<<a<<endl;
      }
};
class Beta:public Alpha{
     private : int b;
     public:Beta(int i,int j):Alpha(i)   {
     b=j;
      cout<<" hello from Beta "<<b<<endl;
    }
};
class Gamma:public Beta  {
      private: int c;
      public :Gamma(int i,int j,int k):Beta(i,j)  {
        c=k;
        cout<<"  Hello from Gamma "<<c<<endl;
      }
};

int main()  {
Gamma g(10,20,30);
}
```

Initializing base class members through derived class object

Multiple inheritance

Init base class members using DC object in Multiple inheritance

```
class Alpha  {
   private: int a;
   public: Alpha(int i)   {
           a=i;
           cout<<"hello from alpha "<<a<<endl;
         }
};
class Beta  {
  private : int b;
  public:Beta(int j)  {
        b=j;
         cout<<" hello from Beta "<<b<<endl;
      }
};
class Gamma:public Alpha,public Beta
{
   private: int c;
   public :Gamma(int i,int j,int k):Alpha(i),Beta(j)   {
           c=k;
           cout<<"  Hello from Gamma "<<c<<endl;
         }
};

int main()  {   Gamma g(10,20,30);      return 0; }
```



Alpha

Beta

Gamma