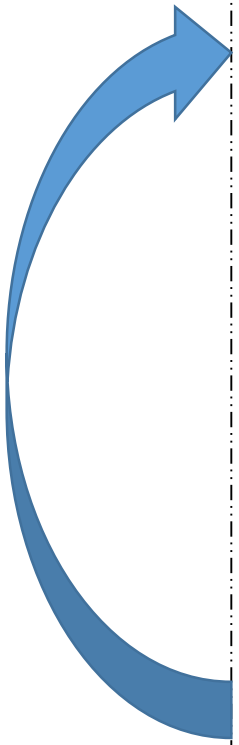
C++ FRIEND FUNCTION & INHERITANCE

By P.P.Krishnaraj

FRIEND FUNCTION

- In encapsulation and data hiding: a non member function has no permission to access private data of the class.
- The private data members of the class are accessed only from member function of the same class. Any non-member function cannot access the private data of the class.
- C++ allows a mechanism in which a non-member function can access the private data members of the class i.e by declaring a non-member function friend of the class whose private data has to be accessed.



```
class DCA
{
int roll;
friend void disp(void);
};
void disp(void)
{
----;
----;
}
int main()
{
disp();
}
```

RULES

- Not a member of any class therefore no need for scope resolution operator.

~~void sum::putdata(void)~~ **void disp(void)**

- Friend function can be called like normal function, no need of object.(since it is not a member of any class)

~~a.disp();~~ **disp();**

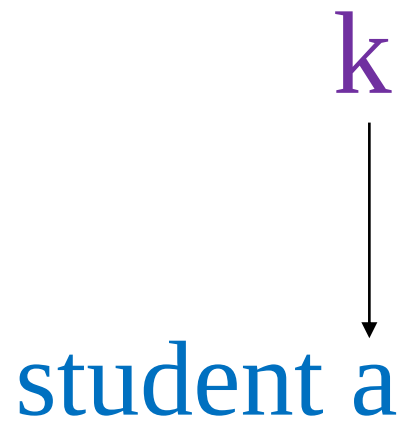
- Definition also like normal function.(no need to mention the name of the class).
- Arguments assed while calling friend function should be the object of the class.

Q) Write a program to access private data using non-member function. Use friend function.

```
class student
{
int mark1;
public :
void get_marks()
{
cout<<“enter two values”;
cin>>mark1;
}
friend void show_mark(student);
};

void show_mark(student a)
{
cout<<“mark of student =“<<a.mark1;
}
```

```
void main()
{
student k;
k.get_marks();
show_mark(k);  //call to friend f(n)
return 0;
}
```



Q) Write a program to declare friend function in two classes. Calculate the sum of integers of both the classes using friend sum() function.

```
#include<iostream.h>
#include<conio.h>
class first;
class second
{
int s;
public :
void get_value()
{
cout<<“enter a number”;
cin>>s;
}
friend void sum(second,first);
};
```

```
class first
{
int f;
public :
void get_value()
cout<<“enter a number”;
cin>>f;
}
friend void sum(second,first);
};
void sum(second d,first t)
{
cout<<“sum of numbers”<<t.f+d.s;
}
```

```
void main()
{
clrscr();
first a;
second b;
a.get_value();
b.get_value();
sum(b,a);
}
```

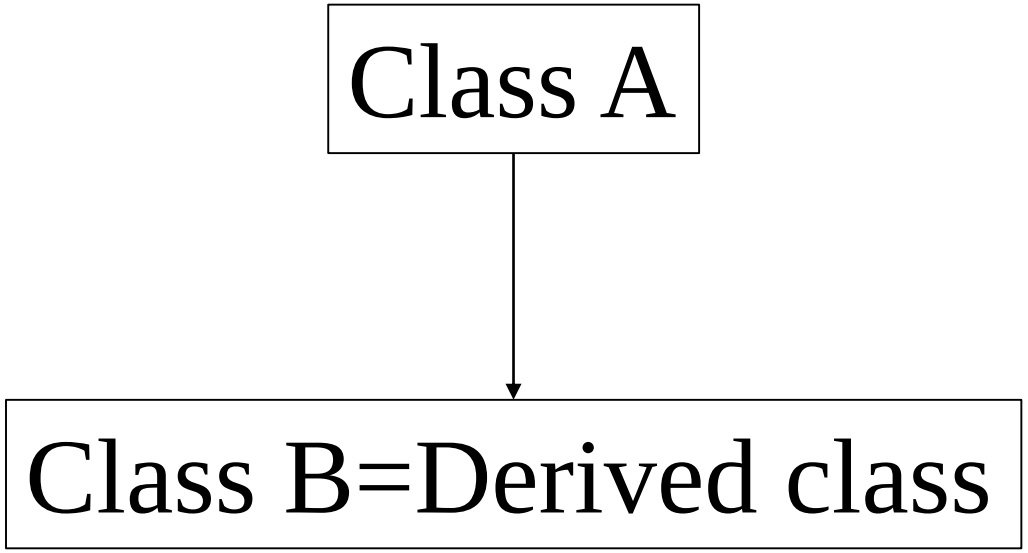
Q) Write a program to exchange values between two classes using friend functions. (assignment)

Q) Write a program to declare three classes. Declare integer array as data member in each class. perform addition of two data member arrays into array of third class. Use friend function.(assignment)

INHERITANCE

(Concept)

- Existing class are main components of inheritance
- New classes can be derived from existing class
- The properties of existing class are simply extended to new class
- The new class created using such methods are called as derived class and the existing classes are known as base classes. The relationship between new class and existing class are known as kind of relationship.
- The programmer can define new member variables and functions in derived class but the base class remain unchanged.
- The object of derived class can access members of base class as well as derived class. On the other hand the base class cannot access members of derived class. The base classes do not know about their sub classes.



class Name of the derived class : Access specifier Name of the base class
{
-----;
-----;
}
Eg: class B : public A

```
class B : public A
{
//members of class B
};

class B : private A
{
//members of class B
};

class B : protected A
{
//members of class B
};
```

(1)PUBLIC INHERITANCE

When a class is derived publically, all the public members of the base class can be accessed by directly in the derived class where as in private derivation, an object of derived class has no permission to access even the public members of the base class directly.

Write a program to derive a class publically from base class. Declare the base class with its member under public section.

```
#include<iostream.h>
#include<conio.h>
class A      //base class
{
public :
int X;
};
class B : public A
{
public :
int Y;
};
```

```
void main()
{
clrscr();
B b;
b.X=20;//access to base class member
b.y=30;//access to derived class member
cout<<“member of A:”<<b.X;
cout<<“member of B :”<<b.y;
}
```

Write a program to derive a class publically from base class. Declare the base class member under private section.

```
#include<iostream.h>
#include<conio.h>
/*public derivation*/
class A          //base class
{
private :
int X;
public :
A()
{ x=20; }
void show_x()
{
cout<<“”x=”<<x;
}
};
```

```
class B : public A //derived class
{
public :
int Y;
B()
{ y=30; }
void show()
{
show_x();
cout<<“y=“<<y;
}
};
```

continued…

```
void main()
{
clrscr();
B b;
b.show_x();
}
```

Output:

X=20

Y=30

Explanation: Class B is derived publically from base class A. The private members of the base class can be accessed by using public member functions of the base class.

The object b invokes the member function show() of the derived class. The function show() invokes show_x() function of base class.

The object b can access member function defined in both base class and derived class.

i.e

b.show() //invoke member function of the derived class.

b.show_x() // invokes member function of base class.

(2) PRIVATE INHERITANCE

The object of privately derived class cannot access the public members of base class under public section.

Q.) Write a program to derive a class privately. Declare the member of base class under public section.

```
#include<iostream.h>
#include<conio.h>
class A
{
public :
int X;
};
class B : private A
{
public :
int Y;
B()
{
X=20;
Y=40;
}
void show()
{
cout<<“X=“<<X;
cout<<“Y=“<<Y;
}
};
void main()
{
clrscr();
B b;
b.show();
}
```

OUTPUT

X=20

Y=40

Explanation: class B is a derived class from class A. The member variable X is a public member of base class. The object b of derived class cannot access variable x directly.

b.X// cannot access

Member function of derived class can access members of base class. i.e the function show() does the same.

Protected inheritance

Why??

The member functions of derived class cannot access the private member variables of base class. The private members of base class can be accessed using public member functions of same class. This approach makes a program lengthy. To overcome the problem associated with private data another access specifier called protected was introduced.

Write a program to declare protected data in base class. Access data of base class declared under protected section using member functions of derived class.

```
#include<iostream.h>
#include<conio.h>
class A
{
protected :
int x;
};
class B : private A
{
int y;
public :
B()
{   X=30;
    Y=40;  }
void show()
{
cout<<“X=“<<X;
cout<<“Y=“<<Y;
}};
void main()
{
clrscr();
B b;
b.show();
}
```

OUTPUT

X=30

Y=40

Types of inheritance

The process of inheritance depends on (1)number of base classes (i.e program can use one or more base class to derive a single class)

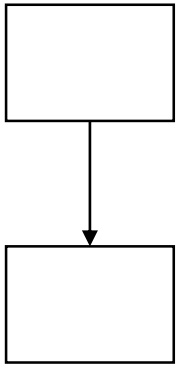
(2)Nested derivation (the derived class can be used as base class and new class can be derived from it.

Different types of inheritance are:

1. Single
2. Multiple
3. Hierarchical
4. multilevel
5. Hybrid
6. multipath

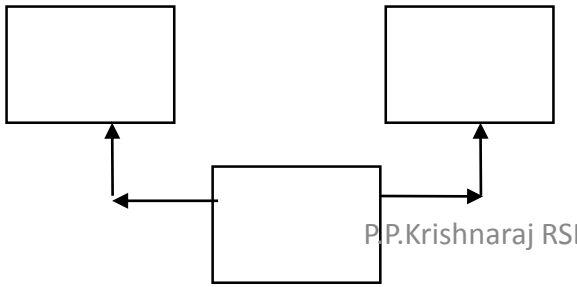
Single inheritance

When only one base class is used for derivation of a class and derived class is not used as base class, such type of inheritance between one base and derived class is known as single inheritance.



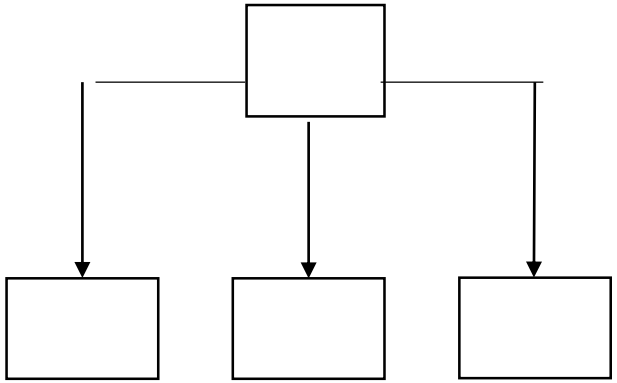
Multiple inheritance

When two or more base classes is used for derivation of a class, it is called multiple inheritance.



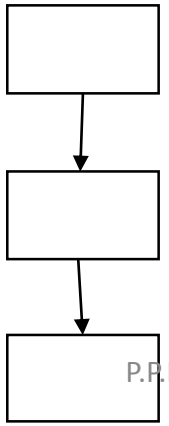
Hierarchical inheritance

When a single base class is used for derivation of two or more classes.



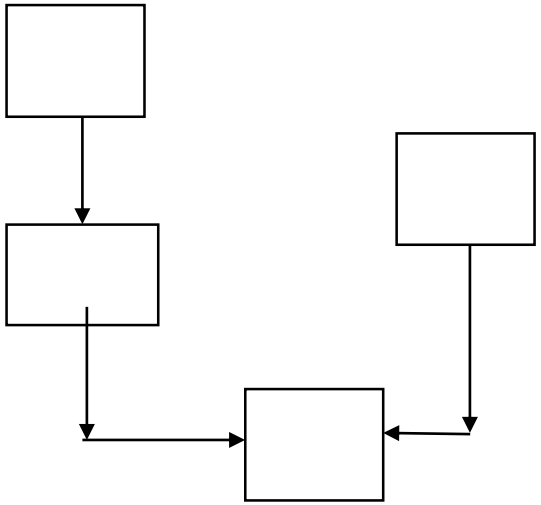
Multilevel inheritance

When a derived class is derived from another derived class i.e, derived class acts as base class.



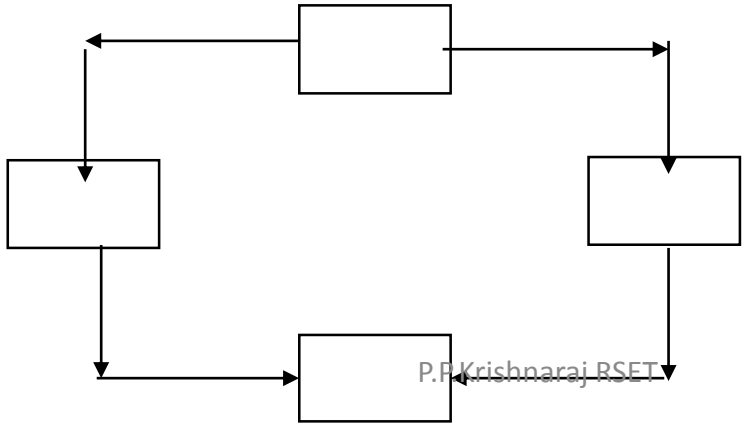
Multilevel inheritance

When a class is derived from another class i.e derived class acts as base class such type of inheritance is called multilevel inheritance.



Hybrid inheritance

Combination of one or more type of inheritance is called as hybrid inheritance.



CONSTRUCTORS

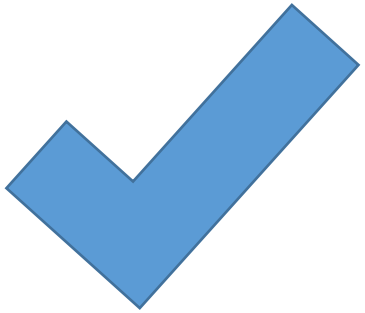
Normally we invoke member function using object and also data members are initialized through objects.

C++ provides a pair of in built special(???) member functions called constructor and destructor.

Job???

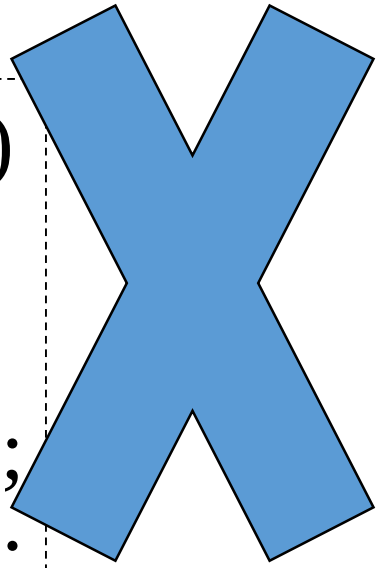
```
class A
{
public :
int X,Y;
A()
{
X=20;
Y=30;
}
}
```

```
int main()
{
A a;
}
```



```
Class A
{
public :
int X,Y;
}
```

```
int main()
{
A obj;
obj.X=20;
obj.Y=30;
}
```

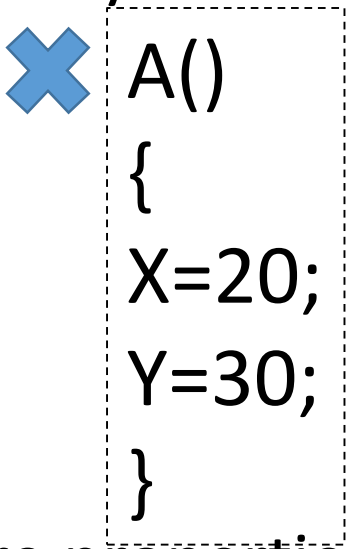


Note: when we create object for class, when the object A a; is initialized, the constructor is invoked.
i.e *automatically invoked*

Definition: A **constructor** is a special member function whose task is to initialize the objects of that class. It is special because its name is the same as the class name

RULES FOR CONSTRUCTOR

1) Constructor has the same name as that of the class it belongs.
2) Constructor does not return any value.i.e neither return value nor void



1) In inheritance concept where properties of one class is inherited by another. Constructor is not inherited.
2) Constructor is executed when an object is declared.
3) Constructor can have default and can be overloaded.
4) The constructor without arguments is called as default constructor.

1) Default constructor

If we are not using any arguments in a constructor, then it is called as default constructor.

```
class test
{
public :
test()
{
----;
----;
}
}
```

Since no argument is passed
this is called as default constructor
(this is used to initialise a value)

Simple program to understand default constructor

```
#include<iostream.h>
#include<conio.h>
class test
{
int a,b;
test()  //constructor declared
{
a=0;
b=0;
void disp(void)
{
cout<<“value of a :”<<a;
cout<<“value of b :”<<b;
}
int main()
{
test t;        //constructor automatically invoked
t.disp();          //function call using object
getch();
}
```

/*no need of separate function of initialising the value of variables*/

OUTPUT

value of a : 0

value of b :0

★ 2) Parameterized constructor

```
class test
{
int a,b;
public :
test(int X,int Y)     //parameterised constructor,
                        some argument is passed.
{
a=X;
b=Y;
}
```

NOTE: now if we want to use constructor but value initialisation to be done by programmer, we define another constructor called as parameterised constructor.

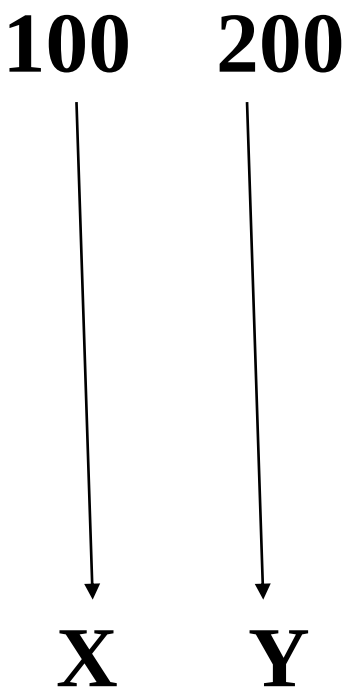
Simple program for parameterised constructor

```
#include<iostream.h>
#include<conio.h>
class test
{
int a,b;
public :
test(int X,int Y)
{
a=X;
b=Y;
}
void disp(void)
{
cout<<“value of a”<<a;
cout<<“value of b”<<b;
}};
int main();
{
test t(100,200); //NOTE: when we create object then
                        itself we have to pass the argument
t.disp();
getch();
}
```

OUTPUT

Value of a 100

Value of b 200



3) Copy constructor

- using copy constructor it is possible for a programmer to declare and initialize one object using reference of another object.
- when ever a constructor is called a copy of an object is created.
- all copy constructor requires one argument, with reference to an object of that class.

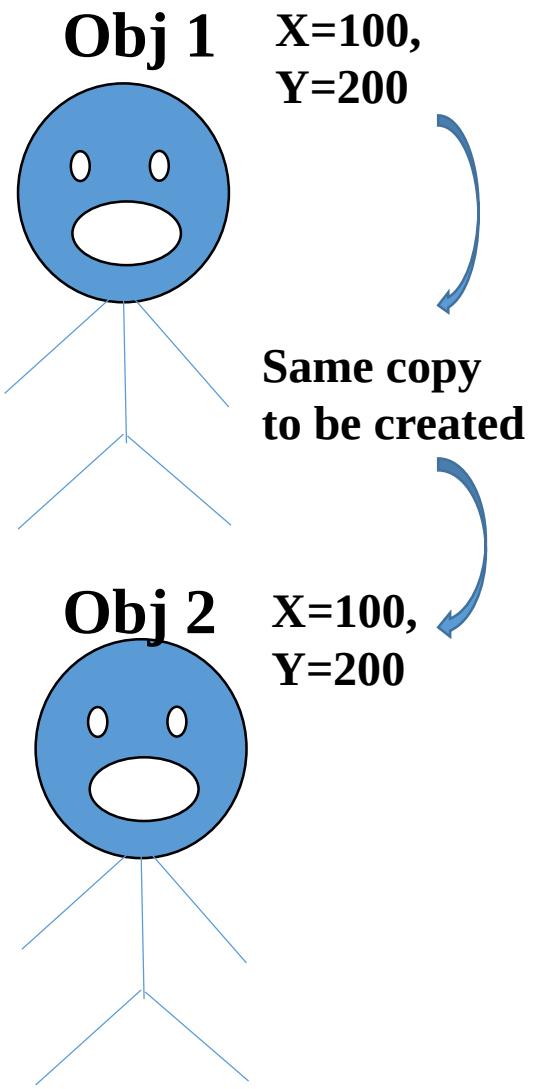
Syntax:

```
Class_name (const class_name &object)
{
------;
------;
}
```

Const y?????
We are copying the object values, no changes are needed.

```
class_name
{
int x,y;
Parameterised constructor
{
X=20;
Y=30;
}
copy constructor syntax
{
----;
}
```

```
int main()
{
Obj 1(100,200);
Obj 2(OBJ 1);
}
```



```
#include<iostream.h>
#include<conio.h>
class test
{
int code, price;
public :
test(int c,int p)//parameterised constr.
{
code=c;
price=p;
}
test(const test &t1) //copy constr.
{
code=t.code;
price=t.price;
}
void disp()     //for displaying
{
cout<<"code :"<<code;
cout<<: price :<<price;
}
int main()
{
test t1(101,100);
test t2(t1);       //copy constructor called
/*object t2 is created member values are
initialised through t1 object*/

cout<<"t1 object";
t1.disp();
cout<<"t2 object";
t2.disp();
}
```

★ Constructor overloading

Constructor overloading definition: a class contains more than one constructor which are defined with same name as the class but contains different number of arguments. depending on number of arguments the compiler executes appropriate constructor.

```
#include<iostream.h>
#include<conio.h>
Class test
{
int a,b;
public :
test()    //(1)default
{
a=0;
b=0;
}
test(int X)
//2.parameterised
{
a=b=X;
}
test(int X,int Y)
//3.parameterised
{
a=x;
b=y;
}
void disp()
{  cout<<“value of a”<<a;
   cout<<“value of b”<<b;
}
int main()
{
test A;
test B(100);
test(100,200);
cout<<“object A”;
A.disp();
cout<<“object B”;
B.disp();
cout<<“object C”;
C.disp();
}
```

OUTPUT
OBJECT A
A=0
B=0
OBJECT B
A=100
B=100
OBJECT C
A=100
B=200