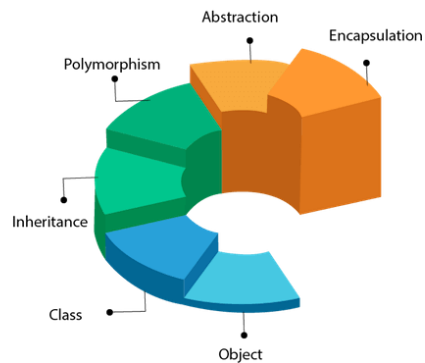


OOP IN C++

OOPs (Object-Oriented Programming System)

The main of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except this function



Class: It is a user defined data type, which hold it's own data members functions, which can be accessed and used by creating an instance of that class.

Object: It is a component of a program that knows how to perform certain actions and how to interact with other elements of the program. Objects are the basic units of object-oriented programming. A simple example of an object would be a person.

Encapsulation: In OOP, encapsulation is defined as binding together the data and the functions that manipulates them.

Abstraction: Abstraction means displaying only essential information and hiding the details.

- Abstraction using classes
- Abstraction using header files (math.h → pow())

Polymorphism: In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

- Operator overloading
- Function overloading

`int sum (10, 20, 30)`
`int sum (10, 20)`

Inheritance: The capability of a class to derive properties and characteristics from another class is called Inheritance.

☞ Subclass

☞ Superclass

☞ Reusability

Dynamic Binding: The dynamic binding, the code to be executed in response to function call is decided at runtime.

Constructor: A constructor in C++ is a special 'MEMBER FUNCTION' having the same name as that of its class which is used to initialize some valid values to the data members of an object. It is executed automatically whenever an object of a class is created.

- It has same name as class itself.
- Constructor don't have return type.

Types:

1. Default Constructor (No Parameter Passed)
2. Parameterized Constructor
3. Copy Constructor

Destructor: A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a call to delete . A destructor has the same name as the class, preceded by a tilde (~).

- Derived class destructor will be invoked first, then the base class destructor will be invoked.

Access Modifier:

- **Public=** It can be accessed by any class.
- **Private=** It can be accessed only by a function in a class (inaccessible outside of a class).
- **Protected=** It is also inaccessible outside but can be accessed by subclass at that class.

Note: If we do not specify any access modifier inside the class then by default the access modifier for the member will be private.

Friend Class: A friend class can be access private and protected members of other class in which it is declared as friend.

Ex: Friend class A:

Inheritance:

Class subclass : accessmode baseclass

1. Single Inheritance: A → B

2. Multiple Inheritance: A B
└─┬─┘
 ↓
 C

3. Multilevel Inheritance:

A
↓
B
↓
C

4. Hierarchical Inheritance:

A
├──┬──┘
 ↓ ↓
 B C
 ├──┬──┘ ├──┬──┘
 ↓ ↓ ↓ ↓

4. Hybrid Inheritance: Combination of one or more type.

Polymorphism: Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance. Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks.

Types:

- **Compile-time polymorphism:** The function to be invoked is decided at the compile time only. It is achieved using a **function or operator overloading**.
- **Runtime polymorphism:** The compiler resolves the object at run time and then it decides which function call should be associated with that object. It is also known as dynamic or late binding polymorphism. This type of polymorphism is executed through virtual functions and function overriding.

Advantages of Data Abstraction:

- Avoid code duplication and increases reusability
- Can change internal implementation of class independently.

Structure VS Class: Most important difference is security. A structure is not secure and can not hide it's member function and variable while class is secure and can hide it's programming and designing details.

Local class in C++ : A class declared inside a function become local to that function and is called local class. All the methods of local class must be defined inside the class only.

Virtual Function & Runtime Polymorphism:

A virtual function is a member function which is declared within a base class and redefined (overridden) by derived class. Function are declared with virtual keyword in base class.

Exception handling in C++ :

Try: represent a block of code that can throw an exception.

Catch: represent a block of code that get executed when error is thrown.

Throw: used to throw an exception.

- There is a special catch block → catch(...)
- It catches all types of error.

Inline Function:

- Inline is a request, not command.
- It is function that is expanded in line when it is called. When the infinite function is called, whole code get inserted or substituted at the point of inline function call.

```
inline return_type fun();  
{  
    -----  
}
```

Function overloading: It is a feature in C++ where two or more functions can have same name but different parameters.

```
void print (int i)  
{  
    cout << "Here is the integer " << i <<  
    } endl;  
  
void print (float i)  
{  
    cout << "Here is the float " << i <<  
    } endl;  
  
int main()  
{  
    print (10);  
    print (10.12);  
    return 0;  
}
```

Difference between C and C++

C

C++

1. C supports procedural programming.	1. C++ is known as hybrid language because it supports both procedural and object oriented programming.
2. As C does not supports the OOP's concept, so it has no supports for polymorphism, encapsulation and inheritance.	2. C++ supports for polymorphism, encapsulation and inheritance as it is an OOP language.
3. C is a subject of C++.	3. C++ is superset of C.
4. C contains 32 keywords.	4. C++ has 52 keywords (public, private, protected, try, catch, throw...)
5. C is a function driven language.	5. C++ is an object driven language.
6. Function and Operator overriding is not supported in C.	6. C++ supports function & operator overriding.
7. C does not support exception handling.	7. C++ supports exception handling using try and catch.

- **Structure** is a collection of dissimilar elements.
- **Static member in C++ :**
 - **Static variable in function:** When a variable is declared as static, space for it get allocated for the lifetime of the program (default initialized to 0). Even if the function is called multiple times, the space for it is allocated once.
 - **Static variable in class:**
 - Declared inside the class body.
 - Also known as class member variable.
 - Static variable doesn't belong to any object, but to the whole class.
 - They must be defined outside the class.
 - There will be only one copy of static member variable for whole class.

Example:

```
Class Account // initialized outside class
{
    Private:
        int balance;
        static float roi;
    Public:
        void setbalance ( int b )
        {
            balance = b;
        }
};

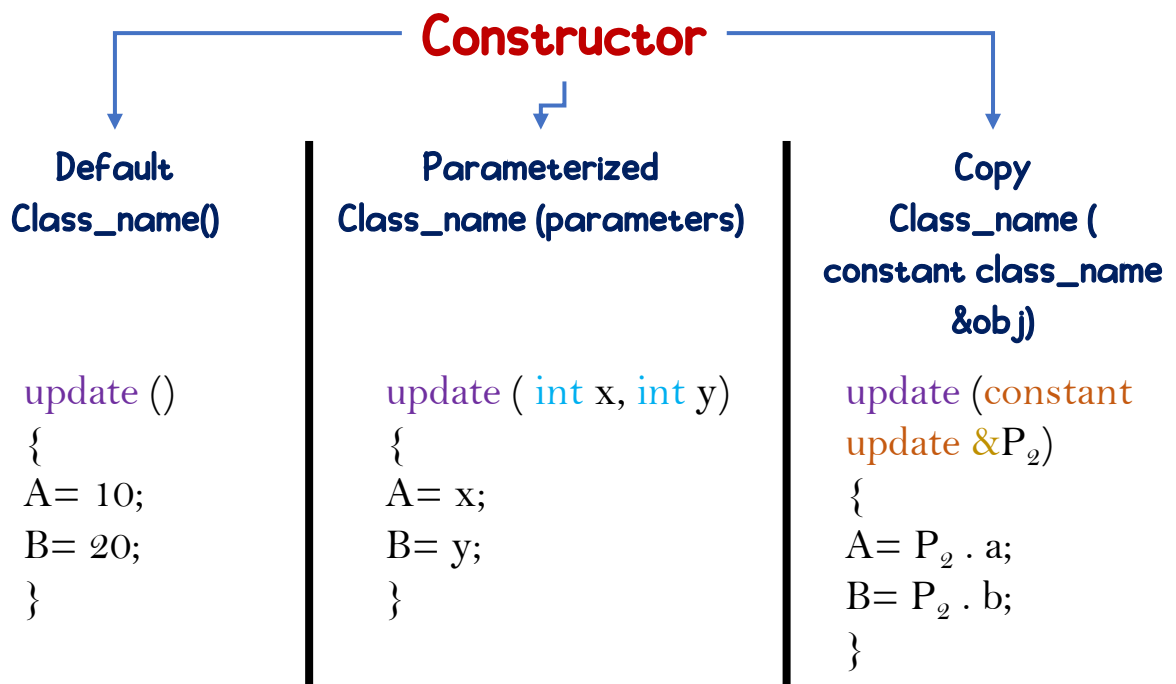
Float Account :: roi = 3.5 F;
void main()
{
    Account + Q1;
}

*Object can also be declared as
static:
    static
    Account+Q1;
```

- **Static function in a class:** static member functions are allowed to access only the static data members or other static member functions

Constructors:

- Constructor is an special member function of the class. It is automatically invoked when an object is created.
- It has no return type.
- Constructor has the same name as class itself.
- If we do not specify, then C++ compiler generates a constructor for us.



Compiler generates two constructors by itself:

- ☐ Default Constructor
- ☐ Copy Constructor

But if any of the constructor is created by user, then default constructor will not be created by compiler.

Constructor Overloading can be done just like function overloading.

Deep Copy is possible only with user defined constructors. In user defined copy constructor, we make sure that pointers of copied object points to new memory location.

Can we make copy constructor private? Yes.

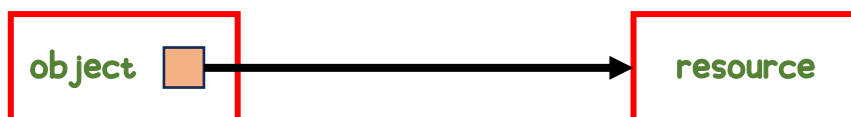
Why arguments to copy constructor must be passed as a reference?

Because if we passed value, then it would make to call copy constructor which become non-terminating.



Destructor

- Destructor is a member function which destructs or delete an object.
- Destructor don't take any argument and don't have any return type.
- Only one destructor is possible.
- Destructor can not be static.
- Actually destructor doesn't destroy object, it is the last function that invoked before object destroy.



Destructor is used, so that before deletion of object, we can free space allocated for this resource. B/C if object gets deletion then space allocated for object will be free but resource doesn't.

Operator Overloading:

C++ have the ability to provide special meaning to the operator.

Ex:

Class Complex

```
{...
    Complex operator + ( Complex & C1)
    {
        Complex res ;
        res . a = C1 . a ;
        res . b = C2 . b ;
    }
}

int main ()
{
    C = C1 + C2
}
```

As '+' can't add complex numbers directly, we can define a function with name +, but we need to write operator keyword before it. So, we can use all operator like this.

Friend Class

A friend class access the private and protected members of other class in which it is declared as friend.

There can be friend class and friend function.

Ex:

Class Complex

```
{
    Private:
        double width;
    Public:
        friend void Printwidth(Complex complex);
        void Setwidth(double wid);
}

void Complex :: Setwidth(double wid)
{ width = wid ; }

void Printwidth(Complex complex)
{ cout << complex.width; }

int main()
{ Complex complex;
  complex. Setwidth(14);
  Printwidth(complex);
return 0;
}
```


Inheritance

It is a process of inheriting properties and behavior of existing class into a new class.

```
Class Base_class  
{...  
};
```

```
Ex: Class car  
{...  
};
```

```
Class der_class : // visibility_mode  
Base_class
```

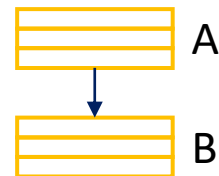
```
{...  
};
```

```
Ex: Class Sports_car: Public Car  
{...  
};
```

Types of Inheritance :

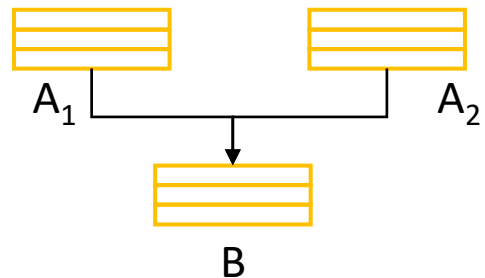
Single Inheritance:

```
Class B : Public A  
{  
};
```



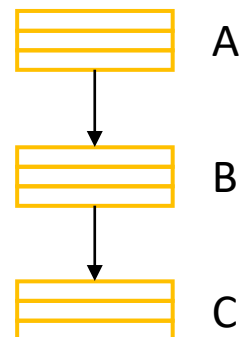
Multiple Inheritance:

```
Class A1  Class A2  
{          {  
};         };  
Class B : Public A1 , Public A2  
{  
};
```



Multilevel Inheritance:

```
Class B : Public A  
{  
};  
Class C : Public B  
{  
};
```



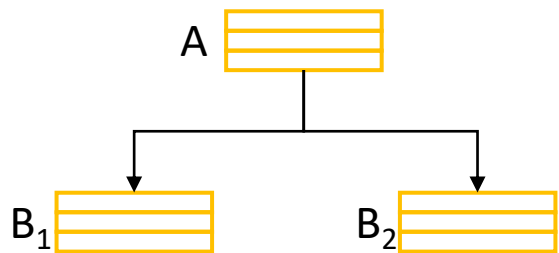
Hierarchical Inheritance :

Class B_1 : Public A

```
{  
};
```

Class B_2 : Public A

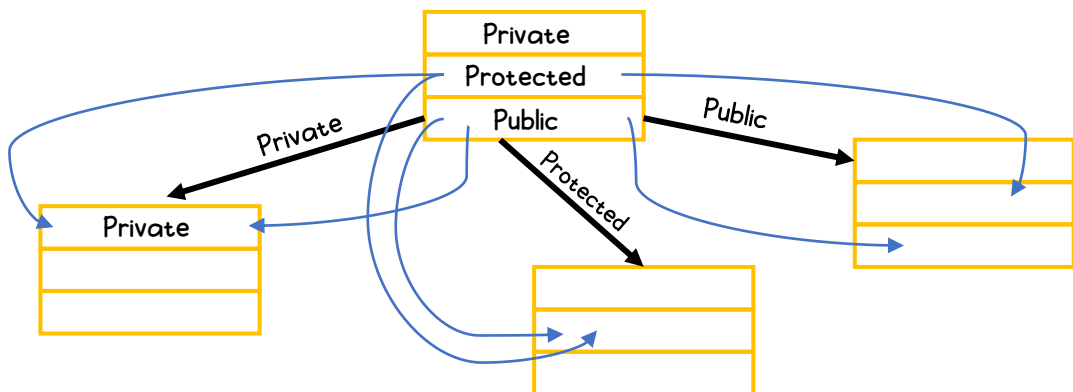
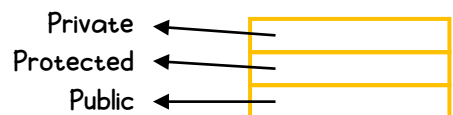
```
{  
};
```



Visibility mode

A – base class

B – sub class



If B is subclass and visibility mode is Public,

Class A : **Public** B
{
}

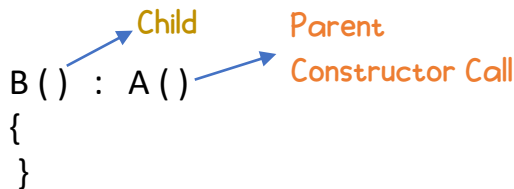
then public members of A will be Public in B , and protected will protected.

If visibility mode is private then both protected and public member of A will be private member of B.

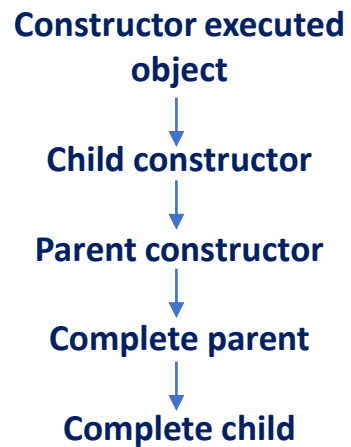
→ “Is a Relationship” is always implemented as a public inheritance.

Constructor and Destructor in Inheritance:

The first child class constructor will run during the creation of the object of the child class, but as soon as the object is created, the child class constructor will run, and it will call the constructor of its parent class. After the execution of the parent class constructor, it will resume its constructor execution.



While in case of destructor first child destructor executed, then parent destructor executed.



this Pointer

Every object in C++ has access to its own address through an important pointer called this pointer.

Friend function doesn't have a this pointer, b/c friends are not member of a class.

Only member function have this pointer.

```
Class Box
{
    Private:
        int l, b, h;

    Public:
        void set( int l, int b, int h )
        {
            this → l = l;
            this → b = b;
            this → h = h;
        }
};

int main()
{
    --- Box b;
    b . set ( 5, 10, 4);
    return 0;
}
```

Method Overriding (achieved at run time)

It is redefinition of base class function in it's derived class, with same return type and same parameters.

While method overloading is achieved at compile time.

Ex:

```
Class Car
{
    private:
        int gear;

    public:
        void change_gear( int gear )
        {
            gear ++ ;
        }
};

Class SportsCar : public Car
{
    public:
        void change_gear ( int gear )
        {
            if ( gear > 5 ) {
                gear ++;
            }
            Car :: change_gear ( gear );    // call the base class
function
        }
};

int main()
{
    SportsCar sc ;
    sc . Change_gear ( 4 ) ;
    return 0;
}
```

“Function of sports car will be called.”

While calling change_gear(), first it check if any function with this name exists in a calling class, otherwise it goes to base class.

Useful: Like we have change_gear() for all except one car which have unique method of gearchange.

Virtual Function

A virtual function is a member function which is declared with a 'virtual keyword' in the base class and redeclared (overridden) in a derived class. When you refer to a object of derived class using pointer to a base class, you can call a virtual function of that object and execute the derived class's version of the function.

- They are used to achieve Run-time Polymorphism.
- Virtual function cannot be static and also cannot be friend function of another class

Compile-time (Early binding) vs Run-time (late binding)

```
Class Base
{
    public:
        virtual void print ()
        {
            cout << "This is base print" << endl ;
        }
        void show()
        {
            cout << "Base show function" << endl ;
        }
};

Class Derived : public Base
{
    public:
        void print() override
        {
            cout << " Derived print " << endl;
        }
        void show()
        {
            cout << " Derived show function " << endl;
        }
};

int main()
{
    base *bptr ;           // Use a pointer to achieve polymorphism
    Derived der ;

    bptr = & der ;
    bptr → print ();       // Run-time
    bptr → show ();        // Compile-time
    return 0;
}
```

Output: Derived print // Late Binding
 Base show function // Early Binding

As during compiler time bptr behavior judge on the bases of which class it belong, so bptr represent base class.

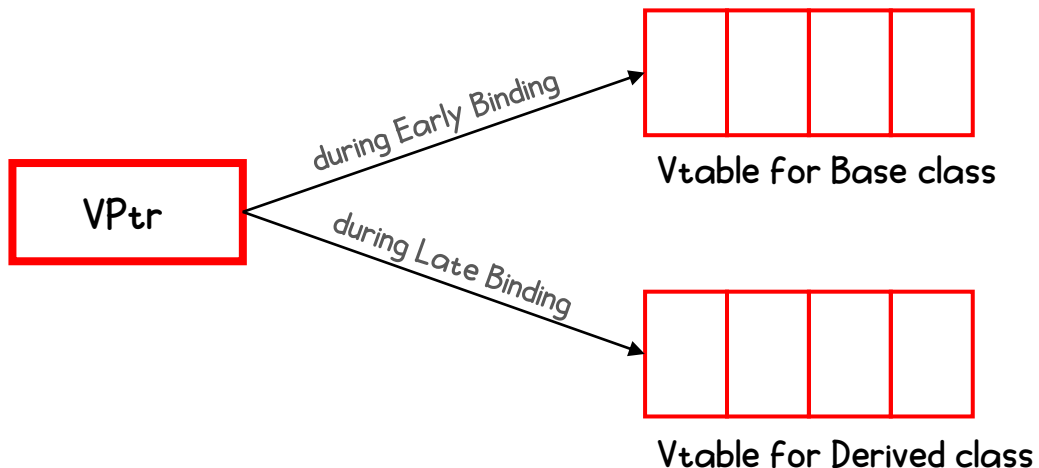
If function is not virtual then it will allow binding at compiler-time and print function of base class will get bounded b/c bptr represents base class.

But at runtime bptr points to the object of class derived, so it will bind function of derived at runtime.

Working of Virtual Function (VTable & VPtr)

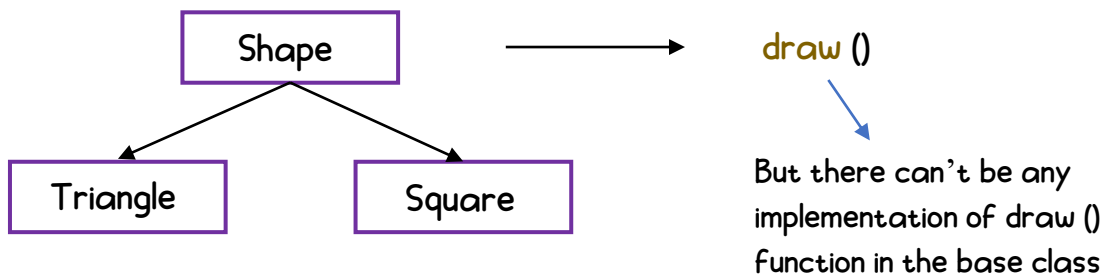
If a class contains virtual function then compiler itself does two things:

1. A virtual pointer (VPtr) is created every time an object is created for that class, which contains a virtual function.
2. Irrespective of whether an object is created or not, a static array of pointers called VTable is created, where each cell point to each virtual function is created in the base class and the derived class.



Pure Virtual Function and Abstract Class

Sometimes implementation of all function cannot be provided in the base class. Such a class is called Abstract Class.




A pure virtual function in C++ is a virtual function for which we don't have any implementation, we only declare it.

// Abstract Class

```
Class Test
{
    Public :
        virtual void fun () = 0;
}
```

Pure virtual function



1. A class is abstract if it has at least one pure virtual function. We can not declare object of abstract class. Ex: Test t; // will show error
2. We can have pointer or reference of abstract class.
3. We can access the other functions except virtual by object of it's derived class.
4. If we don't override the pure virtual function in derived class then it become abstract.
5. An abstract class can have constructors. (Read from GFG)

Template in C++ :

```
template < class x > check a, xb
{
    if ( a > b ){
        return a;
    }else {
        return b;
    }
}
```

It just help in data type. So that we can write generic function that can be used for different data type.

Dynamic Constructor

When allocation of memory is done dynamically using dynamic memory allocator 'new' in constructor.

```
Class Geeks
{
    Public :
        void fun ()
        {
            char *P = new char ( b );
        }
};

int main ( )
{
    Geeks g = Geeks ( );
}
```