OOPs in C++

The main aim 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 types, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.

Object : When a class is defined no memory is allocated but when it is instantiated (i.e., object is created) memory is allocated.

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 clas is called inheritance.

- SubClass
- SuperClass
- Reusability

Dynamic Binding : In dynamic binding, the code to be executed in response to function call is decided at run time.

Constructors : A constructor is a member function o a class which initializes objects of a class. In C+ constructor is automatically called when the object creates.

It has same name as class itself.

Constructor don't have a return type.

1. Default Constructor (No parameter passed)
2. Parametrized Constructor
3. Copy Constructor

Destructor in C++ : Derived class destructor will be invoked first, then the base class destructo will be invoked.

Access Modifier : Public - can be accessed by any class

Private :- can be accessed only by a function in a class (inaccessible outside the class).

Protected :- It is also inaccessible outside the class but can be accessed by subclass of that class.

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

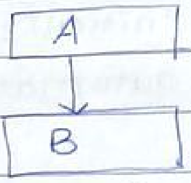
Friend class : A friend class can access private and protected members of other class in which it is declared as friend.

Ex-: friend class B ;

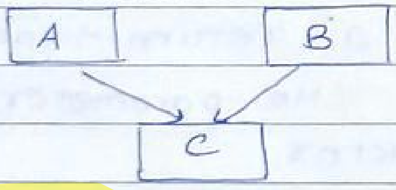
- Inheritance

```
class subclass : acessmode baseclass
{
    ==== .
}
```

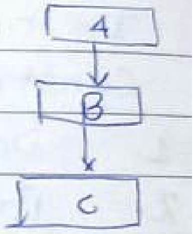
1. Single inheritance



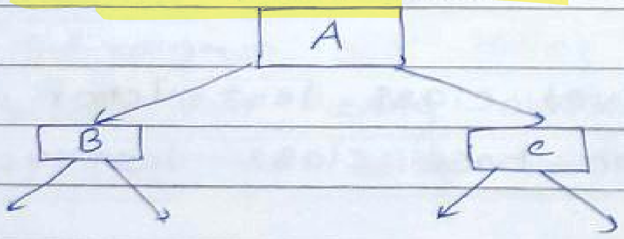
2. Multiple inheritance



3. Multilevel



4. Heerarchical inheritance



5. Hybrid

Combination of one or more type.

- Polymorphism

→ Compile time Poly ↗ Operator overloading
↘ Function overloading.

→ Run time Poly
↳ Function overriding occurs when a derive class has a definition of one or more members of base class.

Advantages of Data Abstraction

- Avoid code duplication and inc. reusability.
- can change internal implementation of class independently.

Structure Vs Class : Most important difference is security.
A structure is not secure and cannot hide its member function and variable while class is secure and can hide its programming & designing details.

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

Virtual function and Runtime Polymorphism :
A virtual function is a member function which is declared within a base class and redefined (overriden) by derived class.
functions 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 cath 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 called. When the inline function is called, Whole code get inserted or substituted at the point of i function call.

```
inline return-type fun( )
{
}
```

• Function Overloading is a feature in C++ When two or more functions can have same name b different parameters.

```
void print (int i)
{     cout << "Here is int " << i << endl;
}

void print (float i)
{     cout << " Here is float" << i << endl;
}

int main
{     print(10);
      print (10.12);
}
```

Differences b/w C and C++

C	C++
1. C supports procedural prog.	• C++ is known as hybrid language, because it support both procedural and object oriented programming.
2. As C does not support the oops concept so it has no support for polymorphism, encapsulation and inheritance	• C++ has support for polymorphism, encapsulation and inheritance as it is an oops language.
3. C is a subset of C++	• C++ is superset of C
4. C contains 32 keywords	• C++ contain 52 keywords (public, private, protected, try, catch, throw...)
5. C is a function driven language	• C++ is an object driven language.
6. Function and operator overloading is not support in C.	• C++ supports function & operator overloading.
7. C does not support exception handling	• C++ ~~doesnot~~ supports exception handing using try and catch

- Structure is a Collection of dissimilar elements

- Static Members in C++

- Static variable in a Function : When a variable is declared as static, space for it gets allocated for the lifetime of the program. (defaut initialized to 0) Even if the function is called multiple times, the space for it is allocated once.

- Static variable in a class :
  → Declared inside the class body.

→ Also known as class member variable.

→ They must be defined outside the class.

→ Static variable doesn't belong to any object, but to the whole class.

→ There will be only one copy of static member variable for the whole class.

Ex:
```
class Account
{
        private :
            int balance;
            static ~~int~~ float roi ;
        public :
            void setBalance(int b)
            {       balance = b ;    }
};
// intialised outside class
        float Account :: roi = 3.5f ;
        void main
        {
            Account a1;
        }
```

• Object can also be declared as static.

Static Account a1;

• Static function in a Class

Static member functions are allowed to access only the static data members or other static member functions.

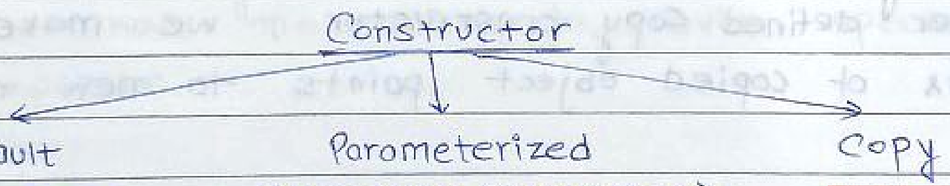
• Constructors :

→ Constructors is an special member function of the class. It is automatically invoked when an object is created.

→ It has no return type.

→ Constructor has same name as class itself.

→ If we do not specify, then C++ compiler generates a default constructor for us.



Class_name();	class_name(parameters)	class_name (const class_name &obj)

update()
{
a=10;
b=20;
}
	update(int x, int y)
	{
	a = x;
	b=y;
	}
		update(const update &p2)
		{
		a=p2.a;
		b= p2.b;
		}

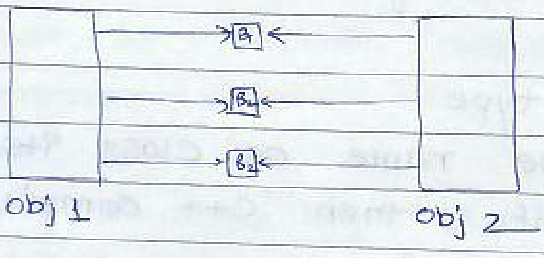
Compiler generates two constructor by itself.

1. Default Constructor
2. Copy Constructor

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

Construction overloading can be done just like function overloading.

Default (Compiler's) Copy constructor can done only shallow copy.



(Points to same memory location)

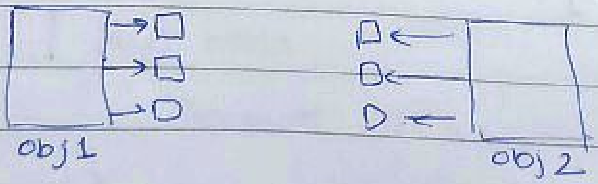
Deep Copy is possible only with user defined constructors

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 argument to copy constructor must be passed as a reference?

Because if we pass value, then it would made to call copy constructor which becomes non-terminating



Deep Copy

- Destructor

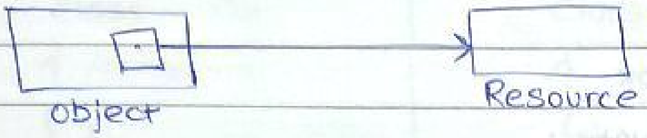
→ Destructor is a member function which destructs or deletes an object.

→ Destructor don't take any argument and don't have any return type.

→ only one destructor is possible

→ Destructor cannot 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 obj we can free space allocated for this resource. B/c if obj gets deleted then space allocated for obj will be free but resource doesn't.

- Operator Overloading

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

```
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 no's directly . So we can define a function with name + but we need write operator keyword before it. So, we can use ~~a~~ all operator ~~like~~ this.

Friend Class

A friend class can 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 Box
            {      private :
                      double width ;
                   public :
                   friend void printWidth(Box box);
                          void setWidth( double Wid);
            }


        void Box : : setWidth (Box double Wid)
        {             width = Wid;      }

        void printWidth (Box box)
        {          Cout << box.width ;    }

        int main()
        {          Box box ;
                   box. setWidth ( 14);
                   PrintWidth(box);
        }
```

Inheritance

It is a process of inherting properties and behaviour of existing class into a new class.

```
class Base_class
{
};
```

```
class der_class : ~~bas~~ Visibility_Mode Base class
{
};
```

Ex:

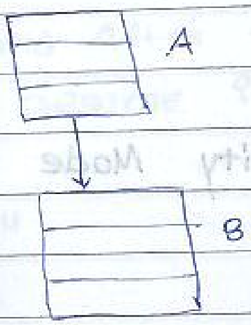
```
class Car
{
};
```

```
Class Sports_Car : public Car
{
};
```

Types of Inheritance :

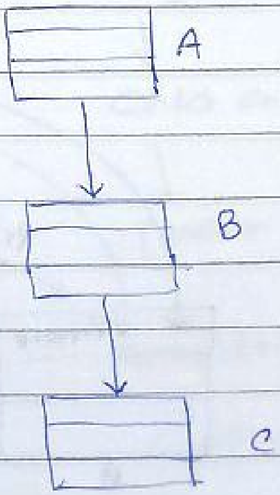
a). Single Inheritance :

```
class B : public A
{

};
```

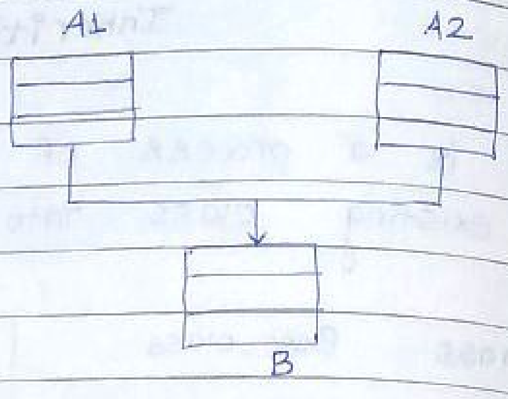


b). Multilevel Inheritance :

```
Class B : public A
{
};
class C : public B
{
};
```

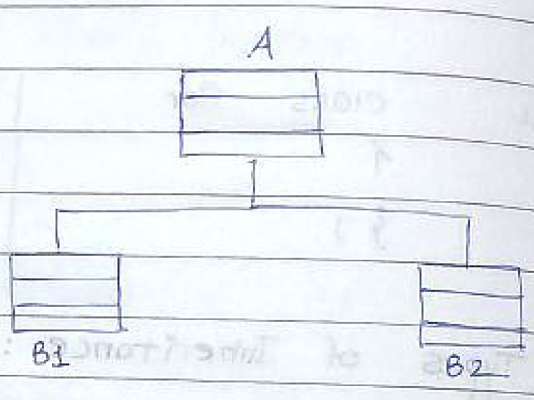


c). Multiple Inheritance



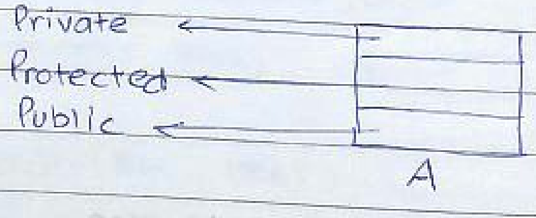
```
class A1          class A2
{                 {
};                };

  class B : public A1, public A2
  {
  };
```

d). Heirarchial Inheritance



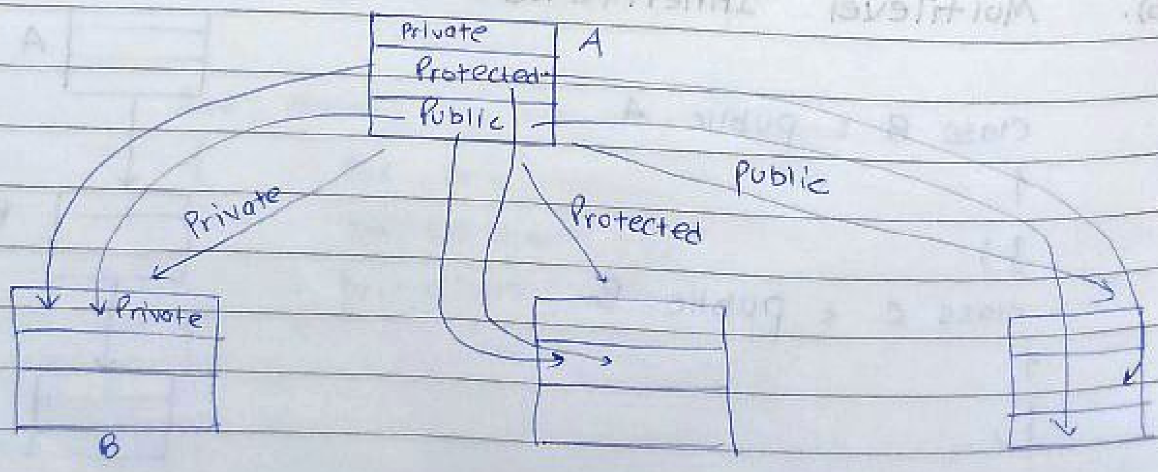
```
class B1 : public A
{
};
class B2 : public A
{
};
```

→ Visibility Mode :



A – base class

B- Sub Class



If B is Sub Class and visibility Mode is public.

```
class A : public B
{
};
```

then public member will be public in B, and protecte
of A
will protected.

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

- is a Relationship is always implemented as a public inheritance.

- Constructor and Destructor in Inheritance

First child class constructor will run during creation of object of Child class, but as soon as obj is created child class constructor run and it will call constructor of it's parent class and after the execu of parent class constructor it will resume it construc execution.

Child → parent constructor call

```
B() : A()
{
}
```

constructor exec,

obj
↓
Child const
↓
Parent Const
↓
complete para
↓
complete ch

while in case of destructor,
first child destructor exec,
then parent desc. 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 members 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);
}
```

Method Over Riding
(achieved at run time)

It is the ~~a~~ redefinition of base class function in its derived class, with same return type and same parameters.

while method Overloading is achieved at Compile time.

Ex:

```
Class Car
{       private:
             int gearno;
        public:
             void change_gear(int gear)
             {
                  gear++;
             }
}

Class SportsCar : public Car
{
        void change_gear(int gear)
        { if (gear>5)
              gear++;
        }
}

int main
{       SportsCar sc;
        sc.change_gear(4);
}
```

function of sports Car Class will be called. first it check if any fun while calling change_gear(), with this name exist in ~~bs sat~~ 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 frie 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 fun" << endl;
            }
}
class derived
{       public:
            void print()
            { cout << " derived Print" <<endl; }
            Void show()
            { cout << " derived show fun" << endl; }
}
```

```
int  main ()
{
        base  *bptr ;
        derived   der ;
        bptr  =   &der ;

     bptr -> print() ;          // Run time
     bptr -> show() ;           // Compile time
}
```

```
output :    derived print                // Late Binding
            ~~This~~ Base show fun       // Early binding
```

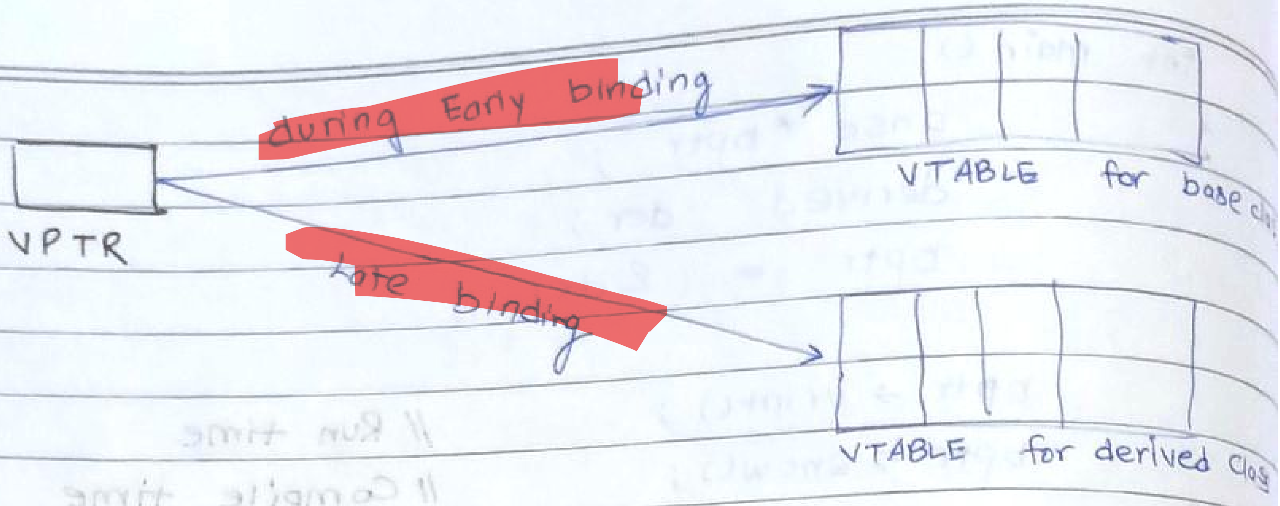
As during compiler time bptr , behaviour judged on the bases of which class it belong , so bptr represent base class .

If the function is not virtual then it will allow binding at compile time and print fun of base class will get binded b/c bptr represent base class . But at run time bptr points to the object of class derived , so it will ~~fu~~ bind function of derived at run time .

Working of Virtual Function (VTable & VPtr)

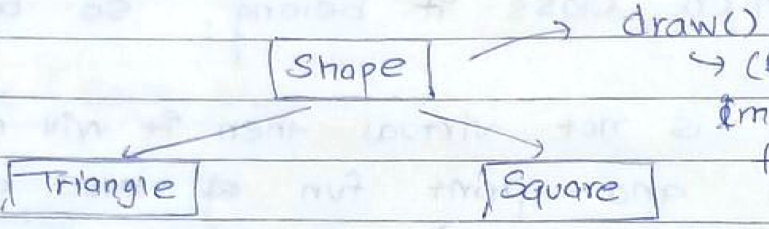
If a class contains virtual function then compiler itself does two things :

1. A virtual pointer (VPTR) is created every time obj is created for that class which contains virtual function.
2. Irrespective of object is created or not, static array of pointer called VTABLE where each cell point to each virtual function is created, in base class and derived class.



Pure Virtual Function and abstract Class

Sometimes implementation of all function cannot be pro in the base class. Such a class is called abstract clas



(But there can't be any implementation of draw fun in base 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 cannot 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 its derived class.

4. ~~If~~ we dont override the pure virtual function in derived class then it becomes abstract.

5. An abstract class can have constructors.
(Read from GfG)

Template in C++

```
template <class X> ~~int~~ X check (~~int~~ X a, X b)
{
        if(a>b)
            return a;
        else return b;
}
```

It doe 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() { p= new char[6]; }
}
int main()
        geeks q = new geeks();
```

Virtual Destructor

Deleting a derived class object using a pointer to a base class that has a non-virtual destructor resul in undefined behaviour, i.e, descrutor of base class runs only ,

Nested Class

A nested class is a member and as such has the same access rights as any other member. The member of enclosing class have no such access to ~~enclosing~~ nested class, members

```
class Enclosing
{      ~~public~~ private:
                  int x;
       public:
            class Nested
            {     int y;
                  void fun(int a) .        }  Run properly
                  { x = a; }
            }

       void fun1(int b)   // Error (b/c it doesn't
       {  y = b; }             have access to y)
}
```