

# OOPS(object oriented programming)

## 1.FEATURES OF C++ :

- It is middle level language.
- It supports principle of OOPS.
- It joins 3 separate programming tradition.
  - \*procedural programming tradition represented by c.
  - \*The object oriented language tradition represented in class .
  - \* Generic programming support.

## 2.COMPARISION B/W C AND C++ :

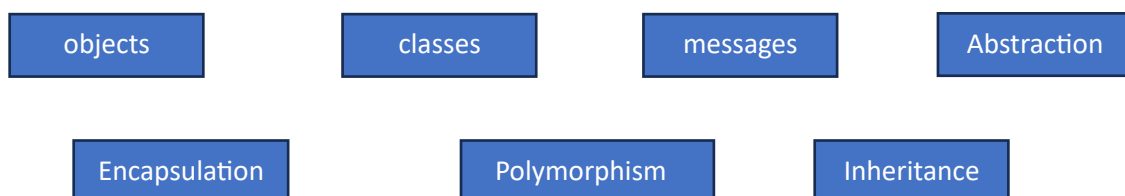
- C is the subset whereas C++ is superset of c language.
- C has 32 keywords whereas c++ has 52 keywords
- C++ program can used existing C software libraries.
- C follow top down approach of program and C++ follows bottom up approach of program.
- In c function and operator overloading is not support in c wheras C++ supports function and overloading.
- C does not support exception handling whereas C++ supports exception handling using try and catch.
- C only approach procedural oriented programming C++ is known as hybrid language as it support both pops and object oriented programming.

Object oriented programming is a high level computer programming language that implements objects and their associated procedures within the programming context to create software programs. The main aim of oop is to bind the data and the function that operate on them so that no other part of the code can access this data except this function and encourage reuse of these objects within same and other program.

There are two main concept used to manage the complexity :

- **MODULARITY** : Breaking a large system or program into smaller pieces until and unless each piece becomes simple enough to handle easily.
- **ABSTRACTION** : It means hiding the implementation detail in a module and providing a well defined interface through which the functionality of a module can be accessed by other module.

## Object oriented design and development :



## 3. APPLICATION OF OOPS :

- Real time system

- simulation and modeling
- object oriented data base
- AI and expert system
- Neural networks
- Parallel programming
- Decision support system
- Office automation system

## 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 instances of that class.

For example : Tree class that describes the feature of trees where each tree has branches etc., The tree class serve has abstract model for the concept of tree. Other examples are students , room , universities..

It is collection of similar type of objects. It is a template from which objects are created. It can have field methods , constructors etc.

Every class must have its own domain and each class obviously different from each other from some respect.. But an object may be member of different classes

## OBJECT :

Any entity in the system that can be defined as a set of properties and set of operation performed using entities property set is known as object. It has 2 main properties.

- State : the object encapsulated information about itself as attributes as field.
- Behaviour: The object can do something on the behalf of other object methods.

Object are created from a classes thus to create an object first we need to create an class. Once class is design we can create a number of objects in that class.

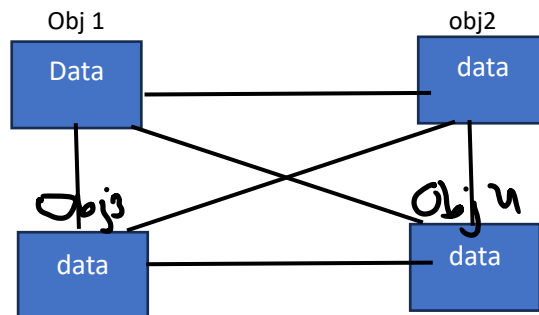
Ex: In a banking system ,a particular account is an example of object. Its state consists of attributes like owner , account number ,customer and behaviour consists of deposit , withdrawn.

when a class is defined no memory is allocated but it is instantiated (object is created) memory is allocated.

**PROBLEM SOLVING IN OOPS :** In real world problems are solved by the interaction among the objects. For example : If there is a problem in our car (which is an object) we take it to mechanic (an other object) to solve problem. Similarly when we create object that has method to solve some problem and we call that object by using (.) function .

## Encapsulation :

In oops, encapsulation is defined as **binding together the data and the functions** that manipulate the data and keeps both safe from interference and misuse.. Opps concept that help in data hiding . Data cannot be accessed by outside world , it means it is very secure from any kind threat which is must important aspect of feature.



From this given figure it is clear that data can only be accessed by the function within an object . The function of other object cannot access the data of other object to insure security .

## ABSTRTACTION :

Abstraction means displaying only essential information about data to the outside world and hiding the background details or implementation .

For ex: a student can have nature like polite , smart , tall , fair , name , roll\_ no. , marks etc. etc. but we are not interested in the attribute like fair, tall , smart because they have no importance in our contex .

1. **Abstraction using classes :**WE can implement abstraction in cpp using classes. Class help us to group the data member and membered function using available access specifiers. A class can decide which data will be visible to outside world and which not.
2. **Abstraction using header files** (math.h -> pow()) : Consider pow() method present in math.h of any member we simply call the pow() function which is present in math.h header file and pass the number as argument without knowing the algorithm according to which the function is actually calculating the power of number.
3. **Abstraction using access specifiers :** Access modifiers are the main pillars of implementation of abstraction in cpp. We can use it to enforce restrictions on class members .

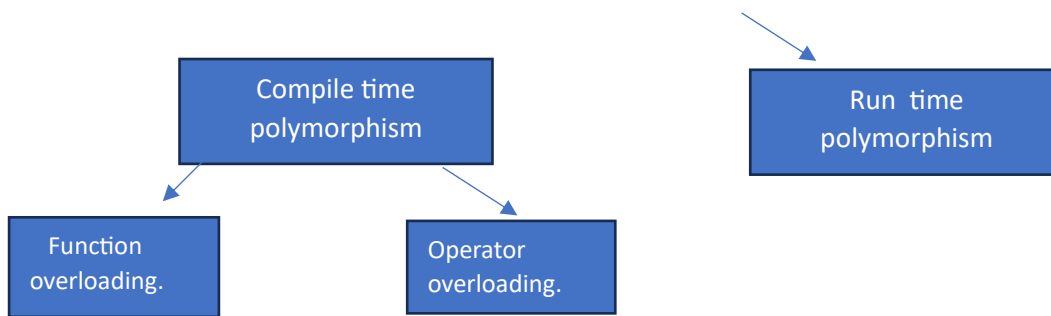
Advantages:

- Avoid code duplication and increase reusability and readability .
- Can change internal implementation of class independently.
- Helps to increase the security of an application program and only essential information provided to the user.
- Helps to avoid entering low level code.

## POLYMORPHISM :

In simple words , we can define polymorphism as the ability of a message to be displayed in more than one form.For example : a person is known by different name in his social world like he is a father , son , husband , student etc etc .

polymorphism



## A.) COMPILE TIME POLYMORPHISM :

**1.Function overloading** occurs when a drive class has a definition of one or more members of base class.

Function overloading is a feature in c++ where two or more function can have same name and different parameters.

```

#include<iostream>
using namespace std;

void print(int i){
cout << "this is for int " << i << endl;
}
void print(double k){
cout << "this is for double " << k;
}

int main(){
print(10);           // this is for int 10
print(20.22);        // this is for double 20.22
}
  
```

- Function name should be same
- Number of argument may be different
- Data type of arguments may be different
- Return type of a function may be different.

## HOW FUNCTION OVERLOADING IS RESOLVED :

First c++ tries to find the exact match.This is the case where actual argument exactly matches the one of the parameter type of function overloading.

If no exact matches are found , cpp tries to find the match through promotion.

Character , unsigned character and short is promoted to an integer.

Float is promoted to double.

If no problem is found then cpp tries to find the match through std conversion.

**2.OPERATOR OVERLOADING** : C++ have the ability to provide special meaning to the operator.

For ex.: we know “+” operator give sum . but I want “+” operator on call print “ Hello Ujjwal Mittal”

```

syntax :

return_type operator + (input) {
}
  
```

Assigning + operator as subtraction

```
void operator +(a,b){
    cout << "output" << a-b << endl;
}
```

```
#include <iostream>

class Marks {
private:
    int m1;
    int m2;
public:
    Marks(int m1Value, int m2Value) : m1(m1Value), m2(m2Value) {}


    // Overloading the > operator
    bool operator >( Marks& other) {
        return (m1 > other.m1) && (m2 > other.m2);
    }
};

int main() {
    Marks A(80, 90);
    Marks B(70, 85);

    if (A > B) {
        std::cout << "A is greater than B." << std::endl;
    } else {
        std::cout << "A is not greater than B." << std::endl;
    }

    return 0;
}
```

**B.) RUN TIME POLYMORPHISM** : also called **OVERRIDING** also know as (dynamic polymorphism)

It is implementing with the help of inheritance concept . **BASE CLASS**  **DERIVED CLASS.**

#### **RULES FOR RUN TIME POLYMORPHISM:**

Return type should be same.

Function name should be same .

Number of argument should be same .

Data type of argument should be same and inheritance is important.

```
class animal{
public :
    void speak(){
        cout << "speaking" << endl;
    }
};
```

```

class dog: public animal{
public:
void speak(){
    cout << "barking" << endl;
}
};

Int main(){

Animal tiger;
dog Dog;
Dog.speak();          // barking
}

Ho ye raha hai ki overriding mai jo class hai uska function call ker diya but agar
dog mai speak function hi nahi hota to animal ka speak function call hota.

```

#### INHERITANCE :

THE capability of a class to derive properties and characteristics from another class is called inheritance.

- Subclass
- Superclass
- Reusability

#### DYNAMIC BINDING:

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

**THIS POINTER**: Every object in c++ has access to its own address through an important pointer called this pointer.

Friend function doesnot have a “this” pointer because friends are not members of a class . only members function have this pointer.

#### CONSTRUCTORS:

A constructor in C++ is a special member function of a class that is automatically called when an object of the class is created. It is used to initialize the object's data members and perform any necessary setup. Constructors have the same name as the class and do not have a return type. They are used to ensure that objects of a class are in a valid state when they are created.

If we don't specify then cpp compiler generates a default constructor for us.

Compiler generates two constructor by itself (default and copy). But if any of the constructor is created by user, then default constructor will not be created by compiler.

- Default constructor: (No parameter passed)
- Parametrized constructor: (take the arguments )
- Copy constructor : it is used to initialize an object from another object .If we don't define a copy constructor then compiler automatically create it and it is public. Copy constructor always take argument as reference object.

```

#include <iostream>
using namespace std;

class student {
public:
    string name;

    int rollno;
    int city;

    student(string name,int rollno,int citycode){
        this->name = name;
        this -> rollno= rollno;
        this-> city = citycode;
    }

    void print(){
        cout << "name " << name << endl;
        cout << "rollno" << rollno << endl;
        cout << "city" << city << endl;
    }
};

int main()
{
    student a ('ugg',12,97);
    student b('t',15,52);
    a.print ();
    b.print ();
    return 0;
}

```

**NOTE : C++ COMPILER CREATES DEFAULT CONSTRUCTOR WHEN WE CREATED OUR OWN CONSTRUCTOR ??**

NO ... Jaise hi argumented constructor Banega to default constructor ki mritu ho Jaige and agar aab hum koi new class banege without giving an argument then it will give error. So once a argumented constructor is formed then its necessary to make new class with require arguments.

## DEEP COPY AND SHALLOW COPY

```

#include<iostream>
using namespace std;

class hero{
public :
    int health;
    int level;

    hero(int health){
        this -> health = health;
    }

    hero(int health , int level){
        this -> health = health;
        this -> level = level;
    }
}

```

```

// copy constructor
hero(hero& temp){
    cout << "inside the copy constructor" << endl;
    this -> health = temp.health;
    this -> level = temp.level;
}
void print(){
    cout << this -> health << " ";
    cout << this -> level << endl;
}

};
int main(){
    hero ramesh(50,60);
    ramesh.print(); // 50 60

    // making copy constructor of ramesh
    hero himesh(ramesh);
    himesh.print(); // 50 60
    cout << endl << endl;

    cout << "for ramesh " << endl;
    ramesh.health = 100;
    ramesh.print(); // 100 60
    cout << "for himesh" << endl;
    himesh.print(); // 50 60

}

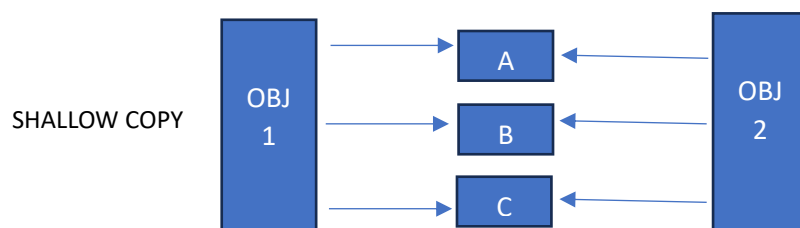
```

deep copy

HERE we see the concept of shallow copy and deep copy  
 There is shallow copy in default copy constructor in which there is 1 memory allocated where both the class are pointing means if there is change in object value of one class other automatically changed.

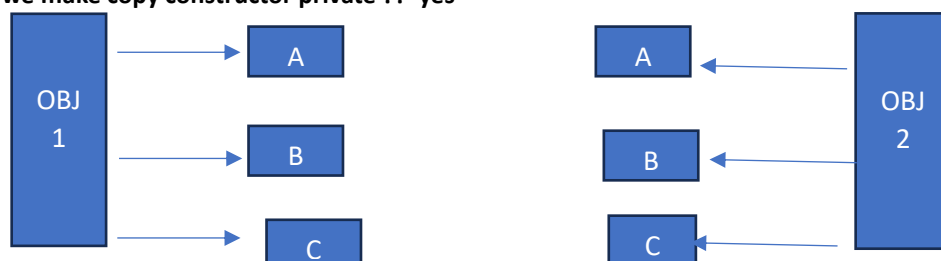
Whenever there is user defined copy constructor we make sure that pointers of copied object points to new memory .As in above example when himesh class copy Ramesh there is 2 memory allocated . changes in Ramesh memory doesnot effect himesh memory. This is deep copy.

Construction overloading can be done just like function overloading.



Deep copy : is POSSIBLE ONLY with user defined copy constructor . In user defined copy constructor , we make save that pointers at copied object points to new memory location .

Can we make copy constructor private ?? yes





## DEEP COPY

### 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.

### COPY ASSIGNMENT :

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

class hero{
public :
    int health;
    int level;

    hero(int health){
        this -> health = health;
    }

    hero(int health , int level){
        this -> health = health;
        this -> level = level;
    }

    // copy constructor
    hero(hero& temp){
        cout << "inside the copy constructor" << endl;
        this -> health = temp.health;
        this -> level = temp.level;
    }

    void print(){
        cout << this -> health << endl;
        cout << this -> level << endl;
    }
};

int main(){
    hero ramesh(50,60);

    // making copy constructor of ramesh
    hero himesh(ramesh);
    ramesh.health = 100;
    ramesh.print();           // 100 60
    himesh.print();           // 50 60

    // copy assignment
    cout << "copy assignment" << endl;
    himesh = ramesh;
    ramesh.print();           // 100 60
    himesh.print();           // 100 60

    // ramesh ke andar himesh ke saari values aa jaayegi and ramesh ki
    values gayab ho jaayegi.THIS is copy assignment in constructor.
```

Differentiation b/w function and constructor

1. **Name:** Constructors have the same name as the class, while functions have a distinct name.
2. **Return Type:** Constructors do not have a return type, whereas functions have a specified return type (including `void` if the function doesn't return a value).
3. **Invocation:** Constructors are automatically invoked when an object is created, without the need for an explicit call. Functions must be explicitly called to execute their code.
4. **Purpose:** Constructors are primarily used for initializing the state of objects, while functions are used for performing operations.

**DYNAMIC CONSTRUCTOR :** When allocation of memory is done dynamically using dynamic memory allocator “new” in constructor.

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

class hero{
public :
    int health;
    int level;

    hero(int health , int level){
        this -> health = health;
        this -> level = level;
    }

    void print(){
        cout << this -> health << endl;
        cout << this -> level << endl;
    }

};

int main(){
    // static
    hero baalver(2,4);

    baalver.print();    // 2 4

    // dynamic
    hero* shaktiman = new hero(8,100);
    shaktiman -> print();    // 8 100
    shaktiman->level = 10;
    shaktiman -> print();    // 8 10
}
```

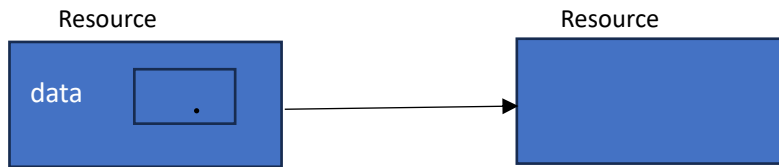
## Destructor in c++ :

Destructor is a member function which destructs or deletes an object. destructor don't have any argument and don't have any return type. Only one destructor is possible. and destructor cannot be static.

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

Actually destructor doesn't destroy object, it is the last function that is invoked before object destruction.

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.



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

class hero{
public :
    int health;
    int level;

    hero(int health , int level){
        this -> health = health;
        this -> level = level;
    }

    ~ hero(){
        cout << "inside destructor" << endl;
    }
};

int main(){
    // static
    hero baalver(2,4);
    // dynamic
    hero* shaktiman = new hero(8,100);
}

// output : inside destructor
For static allocated object default destructor is called whereas for dynamically
constructed object user defined constructor is called;
```

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 runs and it will call constructor of its parent class and after that execution of parent class constructor it will resume its construction execution.

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

### STATIC MEMBER:

Static member of a class are not associated with the objects of the class just like static variable once declared is allocated with memory that can't be changed every object to the same memory.

- Static member is independent of any object of class.
- Even called if no object of class exist.
- Also accessed using class name using scope operator.

- Access inside and outside of class.
- NEED: frequently used to store information that shared by all the classes.

## ACCESS MODIFIER:

\* data hiding is one of the important features of OOP which allows preventing the function of a program to access directly the internal representation of a class type.

\*The access restriction to the class member is specified by the labelled public ,private and protected sections within the class.

\*A class can have multiple public , private and protected labelled sections.

1.PUBLIC: public can be accessed by any class.

2. PRIVATE: can be accessed only by a function in a class (inaccessible outside the class).Only the class and friend function can access the private members.

3. PROTECTED : It is also inaccessible outside the class but can be accessed by subclass at that class.

**NOTE:** if we don't specify the access modifier inside the class then it by **default** will be **private** for the member

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

Access modifier	Access from own class	Access from derived class	Access from outside class
Public	yes	Yes	yes
Private	yes	No	no
Protected	yes	yes	no

There are two ways to define a membered function :

\*Inside class definition

\*outside class definition

To define the operator function outside class we will use the scope resolution operator (::) only with class name and function name.

LOCAL CLASS 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.

**Reference variable** : It provide and alternate name for previously defined variable.

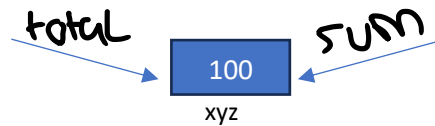
```
include<iostream>
using namespace std;
int main(){

int total = 100 ;
int &sum = total;

cout << total << endl;           //100
cout << sum << endl;             // 100
total+=5;
cout << "after value update" << endl;
cout << total << endl;           //105
```

```
cout << sum << endl;           // 105
}
```

Reference variable is an internal pointer. declaration variable preceded with "&" operator . reference variable must be initialized with during declaration.it can be initiated with already declared variable only.



**FUNCTION :** It is block of code performing a specific task. It has name arguments, return value, and return type. It is a way to achieve modularity .function are predefined and user defined . predefined function are in header files.

- Call by value : as the arguments are ordinary variable so there is change in function values and corresponding values remains the same.
- Call by reference : the addresses of actual argument are passed in the calling function and are copied into formal argument in it. This means that using these addresses we would have a access to actual argument and hence we would be able to manipulate items. Arguments are passed in form of pointer.

Formal arguments are : Ordinary variable , Pointer variable and reference variable .

STATIC MEMBERS IN C++ :

1. **STATIC MEMBER IN A FUNCTION :** When a variable is declared as static,space for it gets allocated for the life time for the program.(default initialized to 0) .Even if the function is called multiple times, the space for it is allocated once. This concept is taken from c language.
2. **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 being any object but the whole class.
  - ➔ There will be only one copy of static member variable for the whole classs.

**STATIC FUNCTION IN A CLASS :** Static member functions are allowed to access only the static data members or other static member function.

**BENEFIT OF FUNCTION :** easy to read , easy to modify , avoid rewriting of same code , easy to debug , better memory utilization .

**DEMERIT : FUNCTION IS TIME COSUMING** behaviour as every time a function is called it takes extra time in executing a series of instruction for task such as jumping to the function , saving registers , pushing arguments into the stack and returning to the calling function.

## INLINE FUNCTION :

- To eliminate cost of calls to small function , c++ propose a new feature called in line function .
- An inline function is function that is expanded in line when it is involved.
- Compiler replaces the function call with corresponding function code.
- It is a compiler optimization to increase efficiency .In line function is a request not a command so the compiler may ignore the request in some situation
  - ➔ Function containing loops , switches gate.
  - ➔ Function contain recursion.
  - ➔ The benefit of speed of inline function reduces as the function grow in size or high comlexity..

Inline return\_type function\_name(arguments) { //code }

```
// Example 1: An inline function for addition
inline int Add(int a, int b) {
    return a + b;
}

int result = Add(5, 3); // Compiler may replace this with result = 5 + 3;
```

ex 2 :

```
// Example 2: A function to square an integer
inline int Square(int x) {
    return x * x;
}

for (int i = 1; i <= 100; ++i) {
    int squared = Square(i); // Compiler may replace this with squared = i * i;
}
```

**STRUCTURE :** is a collection of dissimilar elements. For example :

Struct book -> title (string) -> page (int) -> price (float) -> publisher (string)

Keyword is **struct**. Accessing structure variable with **dot(.)** operator.

- A class can inherit from another class but structure does not.
- A data member of class can be protected but structure class not.

STRUCTURE	UNION
<ul style="list-style-type: none"><li>• keyword <b>struct</b> is used.</li><li>• In a structure all the data members are active at a time.</li><li>• Structure occupies higher memory space.</li><li>• Every member has own memory space.</li></ul>	<ul style="list-style-type: none"><li>• Keyword <b>union</b> is used.</li><li>• In union only the data member is active at a time.</li><li>• Union occupies lower memory space.</li><li>• All the member have same memory space.</li><li>• <b>Note</b> : the memory occupied by a union is large enough to hold the largest member of the union.</li></ul>

```
#include <iostream>
#include <cstring>

// Structure definition
struct Student {
    char name[50];
    int age;
    double gpa;
};

// Union definition
union Variant {
```

```

    int intValue;
    double doubleValue;
    char stringValue[50];
};

int main() {
    // Example with a structure
    Student student1;
    strcpy(student1.name, "Alice");
    student1.age = 20;
    student1.gpa = 3.75;

    cout << "Student Info:" << std::endl;
    cout << "Name: " << student1.name << std::endl;
    cout << "Age: " << student1.age << std::endl;
    cout << "GPA: " << student1.gpa << std::endl;

    // Example with a union
    Variant var;

    var.intValue = 42;
    std::cout << "Integer Value: " << var.intValue << std::endl;

    var.doubleValue = 3.14;
    std::cout << "Double Value: " << var.doubleValue << std::endl;

    strcpy(var.stringValue, "Hello");
    std::cout << "String Value: " << var.stringValue << std::endl;

    return 0;
}

```

Note the difference in memory usage between structures and unions. Structures use memory for each member individually, while unions use memory only for the largest member. In the union example, only one member can hold a value at a time, and changing the value of one member may affect the values of other members.

## FRIEND CLASS :

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

Ex: `class subclass : accessmode.baseclass{`  
`}`

```

#include<iostream>
using namespace std;

class Box{
private:
double width;
public:

```

```

friend void printwidth(Box box);
friend int doublewidth(Box box);
void setwidth(double wid);

};

void Box:: setwidth(double wid){
    width = wid;}

// ye wo function hai jo Box class mai friend declare hai to wahan ki width ko
access ker pa rahe

void printwidth(Box box){
    cout << box.width;
}

int doublewidth(Box box){
    return 2* box.width ;
}

int main(){

Box box;
box.setwidth(14);
printwidth(box);
cout << endl << doublewidth(box) << endl;

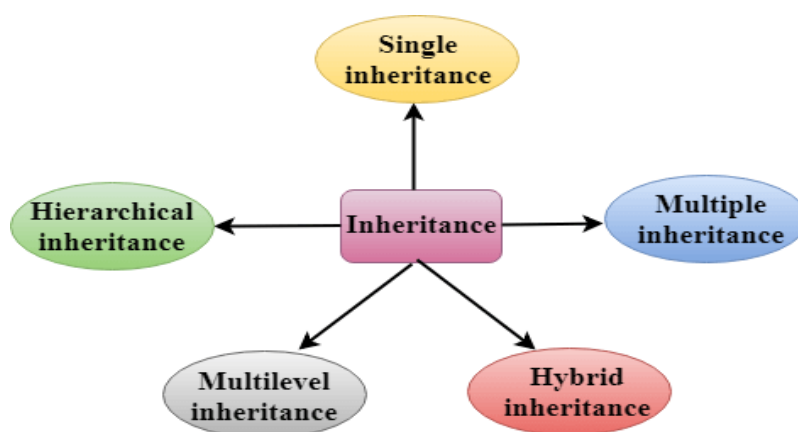
}

```

**INHERITANCE:** It is a process of inheriting properties and behaviour of existion class into a new class.

If B is subclass and visibility Mode is public

Then public member of A will be public in B , and protected will protected' If visibility mode is private the both protected and public member of A will be private member of B.



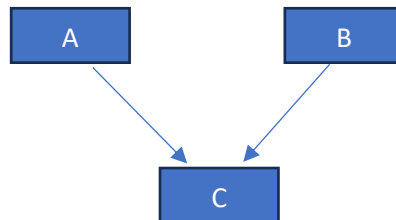


**SYNTAX :** Class Subclass : accessmode baseclass

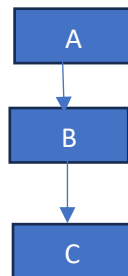
### 1.Single Inheritance



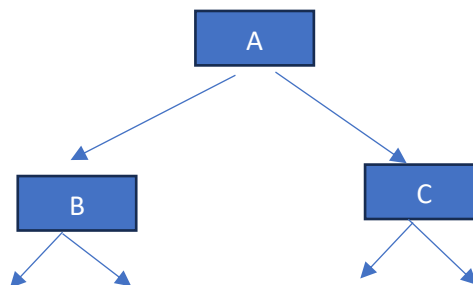
### 2.Multiple Inheritance :



### 3.Multi Level Inheritance :



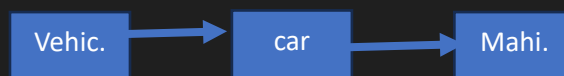
### 4.Hierarchical Inheritance :



### 5.Hybrid Inheritance : combination of one or more type.

```
// multi-Level inheritance
```

```
#include<iostream>
using namespace std;
class vehicles{
public:
    int max_speed;
    int No_tyres;
    char fuel;
};
class car : public vehicles{
public:
    char color;
    char seater;
```



```

};
class mahindra : public car{
public :
    double model;
};
int main(){

car balero;
cout << "ok" << balero.color;
cout << "bye " << balero.fuel;

class mahindra scorpio;
cout << scorpio.seater;

}

```

**INHERITANCE AMBIGUITY** : when one class is derived from two or more base class (multiplevel inheritance) and these two or more base class has function with same name and the derived class doesnot have that function(as otherwise there is fun. Overriding). Here compiler gets confused that with base class function I have to called and there is ambiguity.

Syntax : `derived_obj.base_class :: function_name();`

```

#include<iostream>
using namespace std;

// Base class A
class A {
public:
    void func() {
        cout << " I am in class A" << endl;
    }
};

// Base class B
class B {
public:

    void func() {
        cout << " I am in class B" << endl;
    }
};

// Derived class C
class C: public A, public B {

};

int main() {
    C obj;
    obj.func(); // give error as it get confused

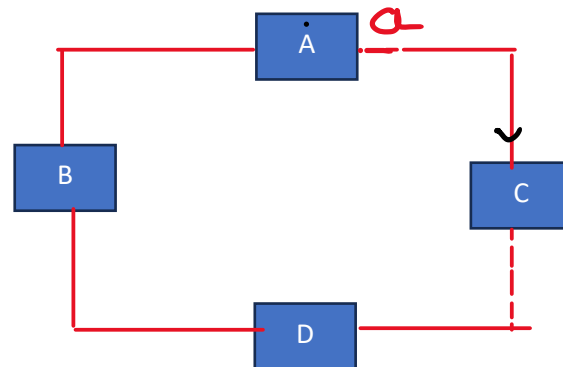
    // Calling function func() in class A
    obj.A::func();

    // Calling function func() in class B
    obj.B::func();

    return 0;
}

```

## VIRTUAL DERIVED CLASS



if class A has a fun/ data member and B and C both inheritate a from class A . So , when a derive class D inheritate both class B and C then D will have two a function one from B and One from C . Here we get the concept of virtual derived class. Here class B and C are set as virtual derived class from A . so when ever a derived class D made from both B and C then only once the function and data members are inheritate by the Class D of A.

```
/* student ->> test
   student ->> score
   test ->> result
   score ->> result
*/
#include<iostream>
using namespace std;

class student {
protected:
    int roll_no;
public:
    void set_number(int a){
        roll_no = a;
    }

    void print_number(void){
        cout << "your roll no." << roll_no << endl;
    }
};

class test : virtual public student{
protected:
    float math , physics;
public:
    void set_marks( float m1 , float m2){
        math = m1;
        physics = m2;
    }

    void print_marks(void){
        cout << "physics marks " << physics << endl;
        cout << "math marks" << math << endl;
    }
};
```

```

class sports : virtual public student{
protected :
float score;
public :
void set_score(float sc){
    score = sc;
}

void print_score(void){
    cout << "score" << score;
}
};

class result : public test , public sports{
private :
float result;
public:
void display(void){
    result = math + physics +score;
    cout << "total" << result << endl;
    print_number();
    print_score();
    print_marks();
}
};

int main(){
    result ujjwal;
    ujjwal.set_number(45);
    ujjwal.set_marks(90,95);
    ujjwal.set_score(20);
    ujjwal.display();
}

```

## Constructor in derived class

- If base class constructor does not have any argument we need not any constructor in derived class.
- But if there is one or more argument in base class constructor , derived class need to pass arguments to base class constructor.
- If both base and derived classes have constructor, base class constructor is executed first.
- In multiple inheritance

A →>> C

B ->>> C     to agar A pehle likha hai tu uska pehle then B ka otherwise B ka pehle

- In multilevel

A ->>> B ->>> C     order mai execute hoga pehle A ka then B ka and then C ka.

- Consteructor of virtual base classes are ivoked first then non virtual,

Syntax : when there are arguments in base as well as derived class

```
Derived(int a, int b, int c, int d) : Base2(b), Base1(a)
```

```

/*
Case1:
class B: public A{
    // Order of execution of constructor -> first A() then B()
};

Case2:
class A: public B, public C{
    // Order of execution of constructor -> B() then C() and A()
};

Case3:
class A: public B, virtual public C{
    // Order of execution of constructor -> C() then B() and A()
};
*/

#include<iostream>
using namespace std;
class Base1{
    int data1;
public:
    Base1(int i){
        data1 = i;
        cout<<"Base1 class constructor called"<<endl;
    }
    void printDataBase1(void){
        cout<<"The value of data1 is "<<data1<<endl;
    }
};

class Base2{
    int data2;

public:
    Base2(int i){
        data2 = i;
        cout << "Base2 class constructor called" << endl;
    }
    void printDataBase2(void){
        cout << "The value of data2 is " << data2 << endl;
    }
};

class Derived: public Base2, public Base1{
    int derived1, derived2;
public:
    Derived(int a, int b, int c, int d) : Base2(b), Base1(a)
    {
        derived1 = c;
        derived2 = d;
        cout<< "Derived class constructor called"<<endl;
    }
    void printDataDerived(void)
    {
        cout << "The value of derived1 is " << derived1 << endl;
        cout << "The value of derived2 is " << derived2 << endl;
    }
};

int main(){
    Derived harry(1, 2, 3, 4);
    harry.printDataBase1();
}

```

```

harry.printDataBase2();
harry.printDataDerived();
return 0;
}

```

```

Base2 class constructor called
Base1 class constructor called
Derived class constructor called
The value of data1 is 1
The value of data2 is 2
The value of derived1 is 3
The value of derived2 is 4

```

## VIRTUAL FUNCTION ()

A member function in base class which is declared with virtual keyword is virtual function.

Mainly dekh jarurat kya hai jab ham base class se pointer banate like

Base\_class\* p ;

P = & derived\_class and isko hume derived class mai point kara diya and dono mai display function pada hai to agar ise pointer ke through hum derived class function ko access karna chahte hai to base class ke function ko virtual banaa padega.

```

#include<iostream>
using namespace std;

class BaseClass{
public:
    int var_base=1;
    virtual void display(){
        cout<<"1 Dispalying Base class variable var_base "<<var_base<<endl;
    }
};

class DerivedClass : public BaseClass{
public:
    int var_derived=2;
    void display(){
        cout<<"2 Dispalying Base class variable var_base "<<var_base<<endl;
        cout<<"2 Dispalying Derived class variable var_derived
"<<var_derived<<endl;
    }
};

int main(){
    BaseClass * base_class_pointer;
    BaseClass obj_base;
    DerivedClass obj_derived;

    base_class_pointer = &obj_derived;
    base_class_pointer->display();
}

```

```
    return 0;
}
```

. In above example agar base class function virtual nahi hoga to hame output mai ye milta.

```
1 Displaying Base class variable var_base 1
```

### One more example

```
#include<iostream>

using namespace std;

class MHP{
protected:
    string title;
    float rate;
public:

    MHP(string title , float rate){
        this -> title = title;
        this -> rate = rate;
    }
    virtual void display(){
        cout << "inside base class" << endl;
    }
};

class wholesale : public MHP{
protected:
    int unit ;
public:
    wholesale(string title , float rate , int unit) : MHP( title ,rate){
        this -> unit = unit;
    }

    void display (){
        cout << "title of item " << title << endl;
        cout << "rate of item " << rate << endl;
        cout << " unit of item " << unit << endl;
    }
};

class retail : public MHP{
protected:
    string name;
public:
    retail(string title , float rate , string name): MHP(title,rate){
        this -> name = name;
    }

    void display(){
        cout << "title of item " << title << endl;
        cout << "rate of item " << rate << endl;
        cout << "customer name " << name << endl;
    }
};

int main(){

    retail r1( "pencil", 5 , "ujjwal");
    wholesale w1("notebook" , 50.86 , 60);

    // r1.display();
    // w1.display();
MHP* p[2];
```

```

p[0] = &r1;
p[1] = &w1;
cout << endl << endl;
p[0] -> display();
p[1] -> display();}

```

```

2 Displaying Base class variable var_base 1
2 Displaying Derived class variable var_derived 2

```

```

title of item pencil
rate of item 5
customer name ujjwal
title of item notebook
rate of item 50.86
unit of item 60
PS C:\Users\h5cd2\Desktop\coding\dsa\c++\k_OOPS\INHERITANCE>

```

Without virtual keyword . this is output

```

inside base class
inside base class
PS C:\Users\h5cd2\Desktop\coding\dsa

```

Base class member access specifier	Type of Inheritance		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

**EXCEPTION HANDLING :-** Exception are defined as the unexcepted or unwanted condition in the code while the run time and ways we define to handle these exception are called as exception handling.

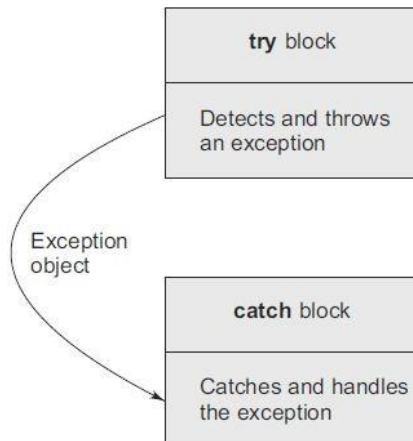
So when a program identify a error it is necessary to catch and deal with it effectively.

Mechanism :

- Find the problem.(Hit the problem).
- Inform that an error has occurred(Throw the exception)
- Receive the error information (catch the exception)
- Take the corrective actions (handle the exception.)

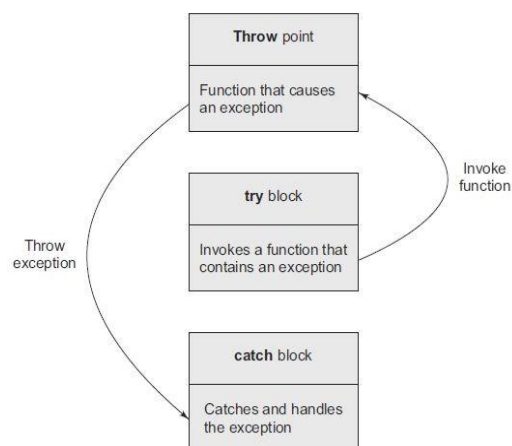


# EXCEPTION HANDLING



When there is a function too in our program then in the try block the function is invoked and argument are passed in the function after the operation if there is exception while operating it throws an exception to the catch block and it handles that exception.

## Throw Point



```
#include <iostream>
using namespace std;
int main()
{
    int a,b;
    cout << "Enter Values of a and b \n";
    cin >> a;
    cin >> b;
    int x = a-b;
    try
    {
        if(x != 0)
        {
            cout << "Result (a/x) = " << a/x << "\n";
        }
        else // There is an exception
        {
            throw(x); // Throws int object
        }
    }
    catch(int i) // Catches the exception
    {
        cout<<"Exception caught: DIVIDE BY ZERO\n";
    }
    cout << "END";

    return 0;
}
```

### First Run

```
Enter Values of a and b
20 15
Result (a/x) = 4
END
```

### Second Run

```
Enter Values of a and b
10 10
Exception caught: DIVIDE BY ZERO
END
```

```

void divide(int x, int y, int z)
{
    cout << "\nWe are inside the function \n";
    if((x-y) != 0)        // It is OK
    {
        int R = z/(x-y);
        cout << "Result = " << R << "\n";
    }
    else                  // There is a problem
    {
        throw(x-y);      // Throw point
    }
}

int main()
{
    try
    {
        cout << "We are inside the try block \n";
        divide(10,20,30);        // Invoke divide()

        divide(10,10,20);        // Invoke divide()
    }
    catch(int i)                // Catches the exception
    {
        cout << "Caught the exception \n";
    }
    return 0;
}

```

## FILE HANDLING :-

```

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

int main ()
{
    char data[100];

    // open a file in write mode.
    ofstream outfile;
    outfile.open("afile.dat");

    cout << "Writing to the file" << endl;
    cout << "Enter your name: ";
    cin.getline(data, 100);

    // write inputted data into the file.
    outfile << data << endl;

    cout << "Enter your age: ";
    cin >> data;
    cin.ignore();

    // again write inputted data into the file.
    outfile << data << endl;

    // close the opened file.
    outfile.close();

    // open a file in read mode.
    ifstream infile;
    infile.open("afile.dat");

    cout << "Reading from the file" << endl;
    infile >> data;

    // write the data at the screen.
    cout << data << endl;

    // again read the data from the file and display it.
    infile >> data;
    cout << data << endl;
    // close the opened file.
    infile.close();

    return 0;
}

```

```

}

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

class Person
{
public:
    char name[25];
    char address[25];

    void addPerson()
    {
        //
        char loop;
        //
        ofstream file;
        file.open("Test.txt",ios::out);
        //
        do
        {
            cout<<"Your Name is : "<<endl;
            cin>>name;
            cout<<"Your Address is : "<<endl;
            cin>>address;
            //
            file<<name<<"|";
            file<<address<<"\n";
            //
            cout<<"To Add More Persons Press 'y' else press 'n' : ";
            cin>>loop;
        }
        while(loop=='y');
        file.close();
    }

    void getAllPersons()
    {
        ifstream file;
        file.open("Test.txt",ios::in);
        //
        while(!file.eof())
        {
            file.getline(name,25,'|');
            cout<<name<<" - ";
            file.getline(address,25,'\n');
            cout<<address<<endl;
        }
    }
};

int main()
{
    Person p;
    //p.addPerson();
    p.getAllPersons();
    return 0;
}

```

tellg() : it is in ifstream . tellg() is used to get the current position of current character in the input stream .

tellp() : it is predefined function in the ofstream which is used to get the position of the character in the output stream .

seekg() : seekg is a predefined function which is used to move to get pointer to a desirable location using the reference point.

Seekp() : seekp() is a predefined function which is used to move the put pointer to desirable location using the reference point.

Syntax: **file\_pointer.seekp(number of bytes ,Reference point)**

**Example:** fout.seekp(10,ios::beg);

**Example :** fin.seekg(10,ios::beg);

## Storage class :-

C++ Storage Classes are used to describe the characteristics of a variable/function. It determines the lifetime, visibility, default value, and storage location which helps us to trace the existence of a particular variable during the runtime of a program. Storage class specifiers are used to specify the storage class for a variable.

### C++ Storage Class

Storage Class	Keyword	Lifetime	Visibility	Initial Value
Automatic	auto	Function Block	Local	Garbage
External	extern	Whole Program	Global	Zero
Static	static	Whole Program	Local	Zero
Register	register	Function Block	Local	Garbage
Mutable	mutable	Class	Local	Garbage
Thread Local	thread_local	whole thread	Local or Global	Garbage

The **auto storage class** is the default class of all the variables declared inside a block. The auto stands for automatic and all the local variables that are declared in a block automatically belong to this class.

The register storage class is just similar like the auto storage class the only difference is in the location of classes. These are stored in the register microprocessor which makes its access very fast.


The static keyword has a local scope with the initial value as zero. The static keyword preserves its existence till the program ends.

## DYNAMIC MEMORY ALLOCATION

S.No.	malloc()	calloc()
1.	malloc() is a function that creates one block of memory of a fixed size.	calloc() is a function that assigns a specified number of blocks of memory to a single variable.
2.	malloc() only takes one argument	calloc() takes two arguments.
3.	malloc() is faster than calloc.	calloc() is slower than malloc()
4.	malloc() has high time efficiency	calloc() has low time efficiency
5.	malloc() is used to indicate memory allocation	calloc() is used to indicate contiguous memory allocation
6.	Syntax : void* malloc(size_t size);	Syntax : void* calloc(size_t num, size_t size);
8.	malloc() does not initialize the memory to zero	calloc() initializes the memory to zero
9.	malloc() does not add any extra memory overhead	calloc() adds some extra memory overhead

### Malloc()

```
int* ptr = (int*) malloc ( 5* sizeof ( int ));
```

ptr =  → A large 20 bytes memory block is dynamically allocated to ptr


← 20 bytes of memory →

4 bytes



### Calloc()

```
int* ptr = (int*) calloc ( 5, sizeof ( int ));
```

ptr =  → 5 blocks of 4 bytes each is dynamically allocated to ptr

← 20 bytes of memory →

4 bytes

4b



1	In the static memory allocation, variables get allocated permanently, till the program executes or function call finishes.	In the Dynamic memory allocation, variables get allocated only if your program unit gets active.
2	Static Memory Allocation is done before program execution.	Dynamic Memory Allocation is done during program execution.
3	It uses <a href="#">stack</a> for managing the static allocation of memory	It uses <a href="#">heap</a> for managing the dynamic allocation of memory
4	It is less efficient	It is more efficient
5	In Static Memory Allocation, there is no memory re-usability	In Dynamic Memory Allocation, there is memory re-usability and memory can be freed when not required
6	In static memory allocation, once the memory is allocated, the memory size can not change.	In dynamic memory allocation, when memory is allocated the memory size can be changed.
7	In this memory allocation scheme, we cannot reuse the unused memory.	This allows reusing the memory. The user can allocate more memory when required. Also, the user can release the memory when the user needs it.
8	In this memory allocation scheme, execution is faster than dynamic memory allocation.	In this memory allocation scheme, execution is slower than static memory allocation.
9	In this memory is allocated at compile time.	In this memory is allocated at run time.
10	In this allocated memory remains from start to end of the program.	In this allocated memory can be released at any time during the program.
11	<b>Example:</b> This static memory allocation is generally used for <a href="#">array</a> .	<b>Example:</b> This dynamic memory allocation is generally used for <a href="#">linked list</a> .

