Object-Oriented Programming.

OOPs is a Programming technique where everything revolves around objects.

Objects :- is an Entity which have properties and Behaviour.

Why ? → Readibility, more Relatable to Real life applications.

Characteristics →

1.) Class, 2.) Object 3.) Encapsulation 4.) Abstraction
5.) Polymorphism 6.) Inheritance 7.) Dynamic Binding.
8.) Message Passing.

Class → 1.) A class is a user-defined datatype that has data-members and member functions.

→ Data Members are the data variables and member functions are the functions used to manipulate these variables together.
→ Member functions :- Define the properties and behaviour of the objects in a class.

07 | FRIDAY
JANUARY 2-0-2-2

Week-02 (007-358)

January 2022

S	M	T	W	T	F	S	S	M	T	W	T	F	S
						1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30	31					

#Class vs Structure.

Class:- is somehow similar to structure except for two small differences. The most important of which is hiding implementation details.

A structure by default doesn't hide its implementation details from whoever uses it in code.

A class by default hides all it's implementation details and therefore will prevent programmer from using it.

→ Class is normally used for data abstraction and further inheritance

→ Structure is normally used for Grouping of Data.

Access Modifiers

Access modifiers or Access Specifiers in a class are used to assign the accessibility to the class-members i.e, they set some restrictions on the class members so that they can't be directly accessed by the outside functions.

Types of Access Modifiers :-

i) **Public** :- Can be accessed from anywhere including functions outside the class.

ii) **Private** :- Can be accessed by member functions or inside the class ~~only~~ only.

iii) **Protected** :- Same as private but member function and friend functions are allowed to access the members of class.

Sunday 09

Syntax :-

```
class Cars {public: // Public modifier
    int no;
    int speed

    private:
    int registration_key;
};
```

10 | MONDAY
JANUARY 2-0-2-2

Week-03 (010-355)

January 2022

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Dynamic Allocation in CPP.

Dynamic memory Allocation in C++ is a mechanism that allows you to allocate memory for variable at run-time, rather at compile time.

Stack Memory:-

→ Used for storage of function call frames and local variable.

→ Memory is allocated and deallocated automatically (In LIFO) order.

→ Size of Stack is fixed and determined at Compile time and is relatively small compared to heap.

→ Data Stored in Stack is Automatically deallocated when a function exits.

```
Void fun()
{ int x = 69; } // x is a local var stored in stack

int main()
{
    Void fun();
    // memory for fun automatically gets deallocated
}  when fun() exits.
```

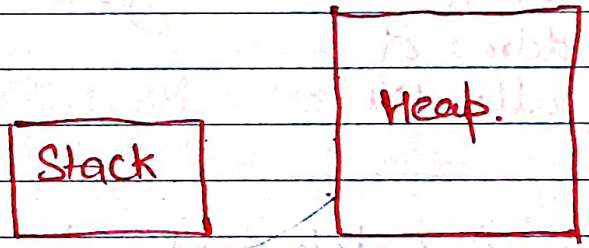
Heap - Memory :-

→ The Heap is a region of memory used for dynamic memory allocation (where you can allocate or deallocate memory as needed in run-time).

→ Memory on Heap is managed manually.

→ Size of Heap is much larger than of Stack and it can grow or shrink according to need.

→ Data stored in Heap persists until manually deallocated, which can lead to memory leaks if not managed properly.



Dynamic Allocation :-

In CPP we use 'new' keyword to allocate memory from Heap. it returns the address (starting) of memory allocated in Heap!

'And we know a guy who's expert in storing addresses!
Pointers. Pointers. Pointers!!!

```
int * p = new int;
```

int *p = dynamic

```
int *p = new int; // Allocate
*p = 69; // Assigning / modification
delete p; // Deallocation.
```

In case of Arrays! Suppose an Array of type int. require 'n' size of elements.

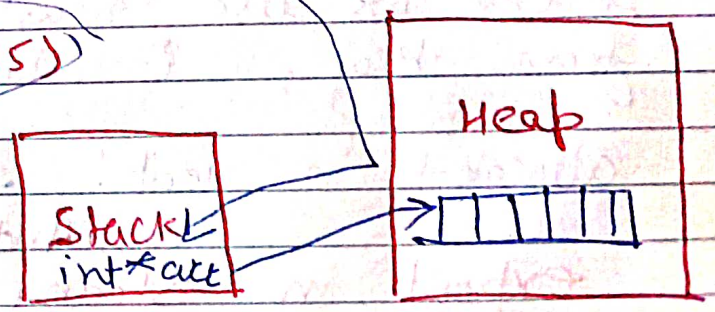
```
int * arr = new int(n)
```

- arr: pointer assigned with starting address of memory allocated
- new: keyword.
- int(n): int datatype x n blocks.

There's A catch baby!

```
int * arr = new int(5)
```

↳ it is stored in Stack and is basically pointing to starting address of memory allocated in heap



Dynamic - Allocation of Objects.

Suppose a Class named Car :-

```
Class Car
{ public :
    int speed;
    int mileage;

  void increasespeed(n)
    { speed += n; }

};

int main ( )
{
  // Static declaration of object.
  Car Suzuki;
  Suzuki.speed = 160;
  suzuki.increaseSpeed(10);
  // Dynamic Declaration

  Car* Buggati = new Car;
  (*Buggati).speed = 260;
  (*Buggati).increasespeed(50);
};
```

Note :- "(*Buggati).speed" and "Buggati -> speed" are same things we can use it like this also :- Buggati -> increaseSpeed(50);

this → Pointer !

This is a pointer that points to current object.

in previous Program :-
we had a member-function

```
Void increase_Speed ( int speed )
{
    speed += speed;
}
```

→ instead of n we choose speed as parameter name.

↳ how to ↲
What speed this two are refrencing to (to parameterized speed as it's in the closest in scope).

Now how to access 'that Speed variable in class'

(*this). speed can be used in this kinda cases!

this → Speed !

```
Void increase_Speed ( int speed )
{
    this → Speed += speed;
}
```

Constructor :-

→ Constructor is a member-function of a class, whose name is same as the class name.

→ It is a Special type of member function that is used to initialize the data members for an object of a class automatically. when an object of same class is created

→ It doesn't have a return type.

Types of Constructor :-

1.) Default Constructor :- Constructors which takes no arguement.

2.) Parameterized Constructors :-
It is possible to pass arguements to Constructors. To create a Parameterized Constructer simply add parameters to it.

!!! if we don't specify ~~the~~ a Constructer. C++ Compiler automatically generates a default Constructer for object (no parameters has empty

Sunday 16

3.) Copy Constructor :-

A Copy Constructor is a member-function that initializes an object using another object of the Same Class :-
Basically it assigns the properties of one object to another object.

→ Copy Constructor - takes a reference to an object of same class as an arguement.

→ By reference or else it'll be an infinite loop!

```
Car ( Car & obj ) {
    this -> max_speed = obj -> max_speed
}
```

Types of Copy Constructor :-

1.) User-Defined 2.) Default.

★ Can Constructor be made Private ?

Yes, and the use case is if you don't want to make direct objects of the class.

Destructor :-

Destructer is an instance member function that is invoked automatically, whenever an object is going to be destroyed. Meaning a destructer is the last function that is going to be called before an object is destroyed.

Syntax:

Inside-Class

```
~ class_name() {
    // Instructions
}
```

Outside-Class.

```
classname :: ~ class_name() {
    // Instructions.
}
```

Properties of Destructer.

→ Destructer function is automatically invoked when objects are destroyed.

→ It cannot be declared static or const.

→ The destructer doesn't have arguements.

→ It has no return type, not even void.

→ An object of class with a Destructer cannot become a member of the union.

→ It should be declared in public.

→ Programmer can't access the address of the destructor.

When is Destructor Called ?

→ When the function ends.

→ the program ends.

→ a block containing local variables ends.

→ a delete operator is called.

Call Destructors Explicitly :-

Syntax

```
Object_name.~class_name()
```

~~Can a Destructor be~~

If we don't write our own destructor in Class, the compiler creates a default Constructor for us. The default Destructor works fine unless we have dynamically allocated memory or pointer in class.

Virtual Destructor :-

→ Virtual in CPP :- is used to achieve Run-time polymorphism.

★ # Can we make a Virtual Constructor?

→ No, Constructer can't be virtual, because when a Constructer of a class is executed there is no virtual table in the memory. means no virtual pointer is defined yet.

Virtual Destructor ? :-

Yes, its important to handle proper destruction of derived class.

4 - Pillars of OOPS

1.) Encapsulation
2.) Inheritance
3.) Polymorphism
4.) Abstraction.

→ Sab-kuch hi Abstraction hai!

→ Encapsulation :- Wrapping up of data and information in a single-unit

it basically means binding the data and functions that manipulates them together

Two important Properties of Encapsulation :-

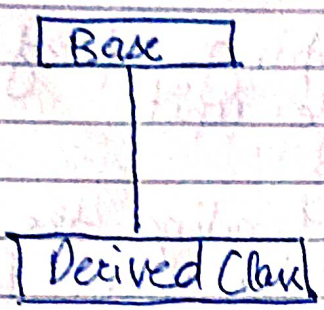
1.) Data Protection :- Encapsulation protects the internal state of an object by keeping it's data members private.

2.) Information Hiding :- Encapsulation hides the internal implementation details of a class from external code.

Sunday 23

Inheritance!

Inheritance is a feature or process in which, new classes are created from existing classes. The new class created is called "derived class" or "child class", and the existing class is known as "base class" or "parent class".



Private members. Can't be inherit!

Syntax:-

```
Class Derived-Base-Class-name : public Base-Class.
{
    // modification etc etc.
};
```

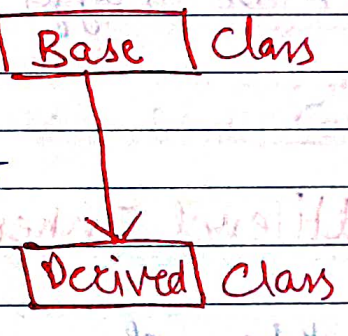
Basically we can use the properties and data of Base class in a new class without actually modifying it!

Base Class Member Access Specifier	Type of Inheritance		
	Public	Protected	Private
Public	Public	Protected	Private.
Protected	Protected	Protected	Private
Private.	N.A	N.A	N.A.

Types of Inheritance :-

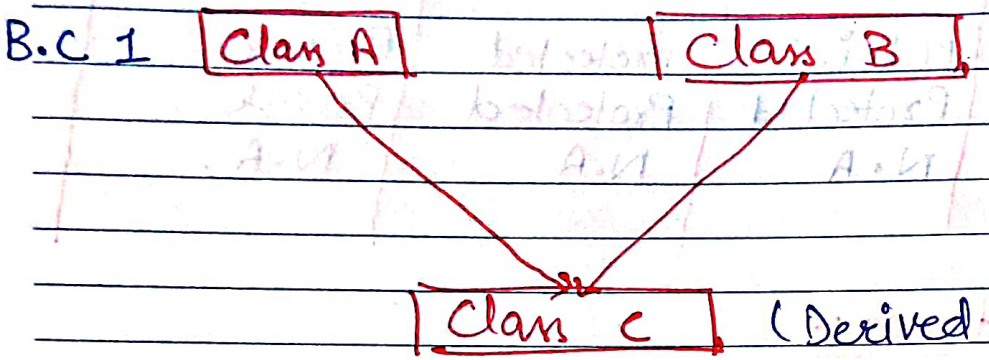
1.) Single Inheritance
2) Multi-level Inheritance
3.) Multiple Inheritance
4.) Hierarchical Inheritance
5.) Hybrid Inheritance.

✱ Single Inheritance :-



A class is allowed to inherit from only one class i.e one subclass is inherited by one base class only.

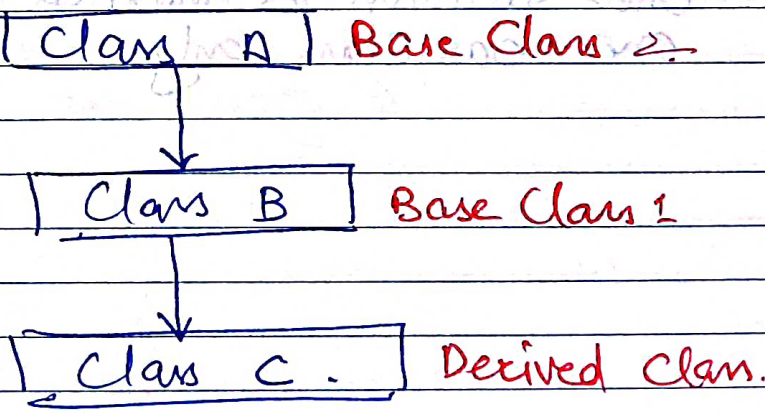
2.) Multiple Inheritance :- Multiple Inheritance is a feature of C++, where a class can inherit from more than one class.



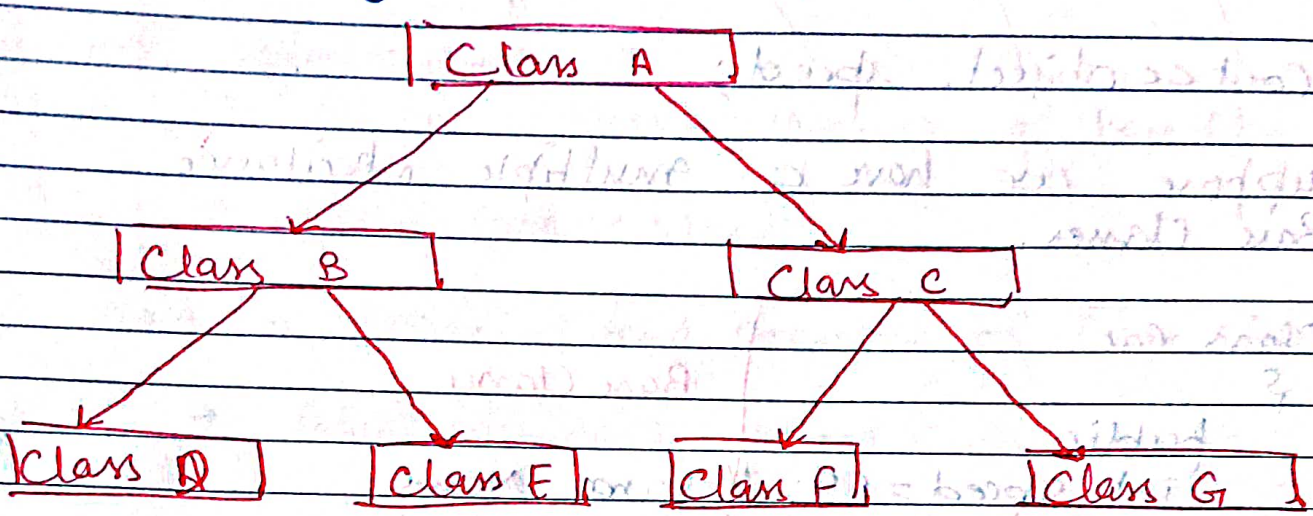

```
class Derived_class_name : Access_modifier class 1, class 2
{
    // Do stuffs!
};
```

3.) Multilevel Inheritance :-

→ In this type of Inheritance, a derived class is created from another Derive Class!



4.) Hierarchical Inheritance :- In this type of Inheritance more than one subclass is inherited from a single base class i.e. more than one derived class is created from a single base class.



Scope Resolution (Fixing Diamond Problem).

Suppose we have one Base Class Car.

```
Class Car {
    int speed = 69;
};
```

and one Derived Class.

```
class Mahindra : public Car
{ public:
    int Speed = 699;
};
```

Now, when I'll try to access the speed member.

```
Mahindra object;
Cout << object.speed;
```

Suppose we have a multiple inheritance Base Classes.

```
Class Car
{
    public
    int speed = 69; // Same Name
};
```

Base Classes.

```
class Mahindra
{ public:
    int Speed = 699; // Same Name.
};
```

```
class Scorpio : public Car, public Mahindra
{

}
```

And if we'll toy to accers the 'speed' or any same name functions it'll throws an error.

→ Scorpio Abhi; // objection creation.

→ ~~Abhi.speed;~~

To Accers speed member of Base Class Car.

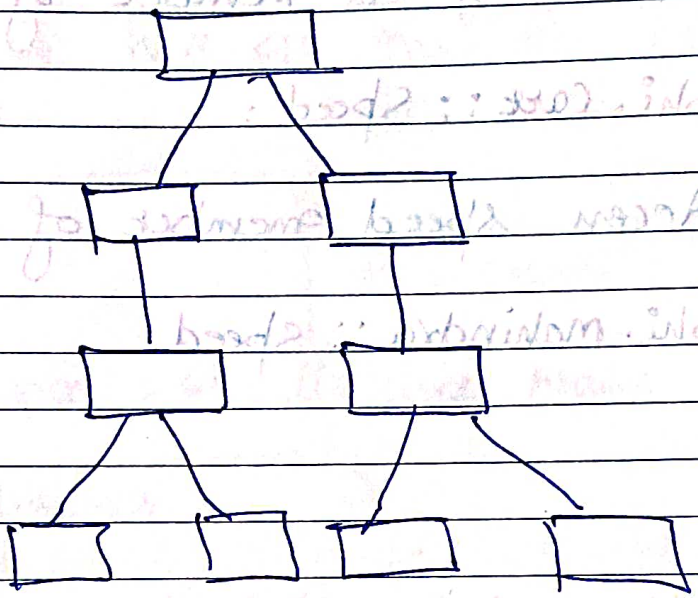
→ Abhi.Car::Speed;

and to Accers speed member of Base Class Mahindra

→ Abhi.mahindra::speed.

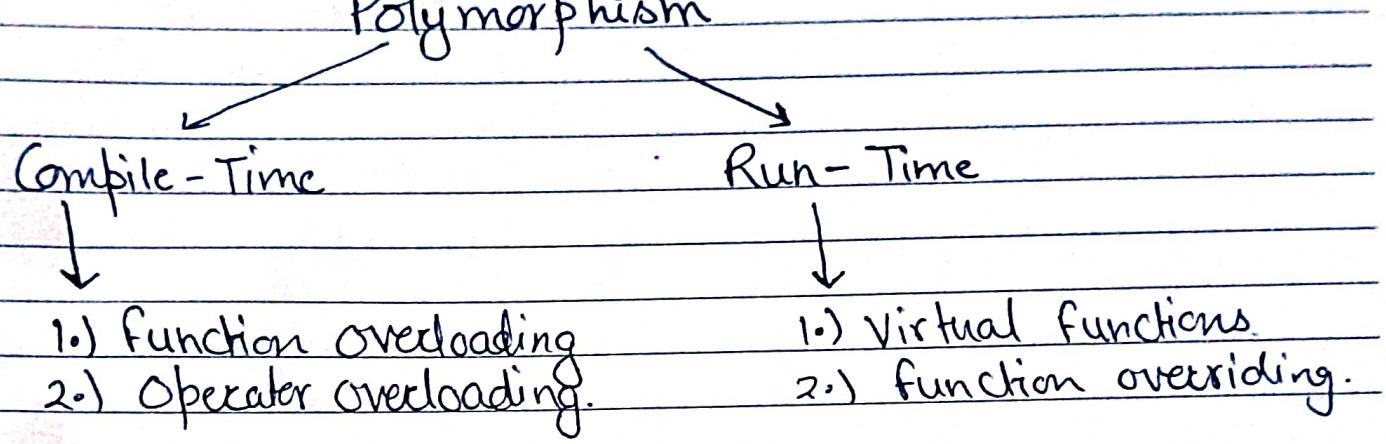
Hybrid Inheritance:-

Hybrid Inheritance is implemented by combining more than one type of inheritance. It's basically mixture of all other types of Inheritance.



Polymorphism :-

Polymorphism basically means ~~many into one~~ existing into many forms.



★ Function overloading :-
Same function with different parameters can be used to perform different tasks and this property is called function overloading.
Basically,

Function Overloading is a feature of OOPs where two or more functions can have the same name.

If multiple functions having same-name but parameters of the function should be different is known as Function Overloading.

Function over-riding.

Function over-riding in C++ is termed as the redefination of base class function in its derived class with the same sign (return type and parameters.

Friend- Keyword.

→ Friend is a keyword in C++ that is used to share the information of a class that was previously hidden.

→ For Example :-

Suppose class A have private data member int x.

```
class A {
  private:
    int x;
};
```

```
class B {
  private: cout
    int x;
};
```

Basically friend keyword is used to accen the private members of a class.

We can use

Sunday 06

friend classname class_name;

Friend function :

friend void fun ( ) ;

Const - keyword

1.) The Const - keyword is used to declare that a variable function, or object is immutable i.e it's value can't be changed after initialisation.

```
Const int x = 5;
x = 10;
```
→ this line will throw an error.

2.) We can declare a function as Const which means that it doesn't modify the state of the object it's called on.

3.) Compiler may be able to store Const variables in read-only memory, which can result in faster-access time.

Two types of variables

Lvalue :- Variable having memory locations.

Rvalue :- Variables don't have memory locations.

11 FRIDAY
FEBRUARY 2-0-2-2 Week-07 (042-323)

February 2022
S M T W T F S S M T W T F S
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

Non-Const Data and Const pointer :-

Const keyword (after (*) asterik), then it's Constant pointer with variable data.

int* Const(a)= new int(69) → Value.
↳ New keyword
↳ Datatype

datatype keyword Variable name

Constant Data with Non Const pointer.

→ Const keyword before (*) asterik :-

Const int * y = new int(69);

Constant Data and Constant Pointer :-

Const int * Const a = new int(10);

keyword before & after Const

You may be disappointed if you fail, but you are doomed if you don't try. – Beverly Sills

Const - functions :-

Const Member functions don't allow changing the data values of the data-members of their class.

Any attempt to change Const objects will result in run-time error.

```
void get_x() const
{
    return x
}
```

→ this lines makes the function immutable.

↳ Can What if we do wanna Change the value of x despite being in an immutable function:!

→ We can use 'mutable' keyword for doing so

Sunday 13

```
mutable int x;
```

↳ basically it means we can change the value of x even after in an im-mutable functions.

Initialization List :-

Initialization list is used in initializing the data-members for a class. The list of members initialized is indicated with a constructor as a comma-separated list followed by a colon.

```
Class abc {
    int x, y, z;
    abc (.) : x(5), y(6), z(4) { };
};
```

↳ basic syntax of an Initialization list.

it's basically a Constructer with a new / special style for initialisation of data members.

MACROS :-

In C-PP, macros are preprocesor directives that allow you to define constants, functions or code snippets that can be used throughout your code.

↳ Defined using #Define Directive.

→ Evaluated by the procesor before the code is compiled.

→ Macros can be used for a variety of purposes, such as defining constants or creating short-hand for commonly used expresions.

Types of Macros:-

(Basically you can define a snippet of code that is which is going to be used in multiple times throughout the program).

1.) Object like Macro :- Simple identifier replaced by code fragments. It is called object-like because it looks like an object in code that uses it, popularly used to replace a symbolic name/ ~~property~~ variable represented as constant.

```
#Define PI 3.14
```

2. # Chain Macros :-

Macros inside macros are called chain macros.

In Chain macros Parent macro is expanded first.

```
#Define Square(x) (x * x);
#Define Double-square(y) Square(y) * Square(y)
```

3. # Multi-line Macros :-

```
#Define ABC 1. \
            2. \
            3.
```

To create a multiline macro you have to use \ newline.

4.) # Function like Macros :-

```
#Define AREA(r) (PI * r * r);
```

Advantages of Macro :-

1.) Increased Productivity 2.) Customization
3. Consistency 4.) Efficiency.
5. Ease of Use.

Disadvantages.

1.) Security risks 2.) Limited functionality
3.) Maintainence 4.) Dependence.
5.) Compatibility issues.

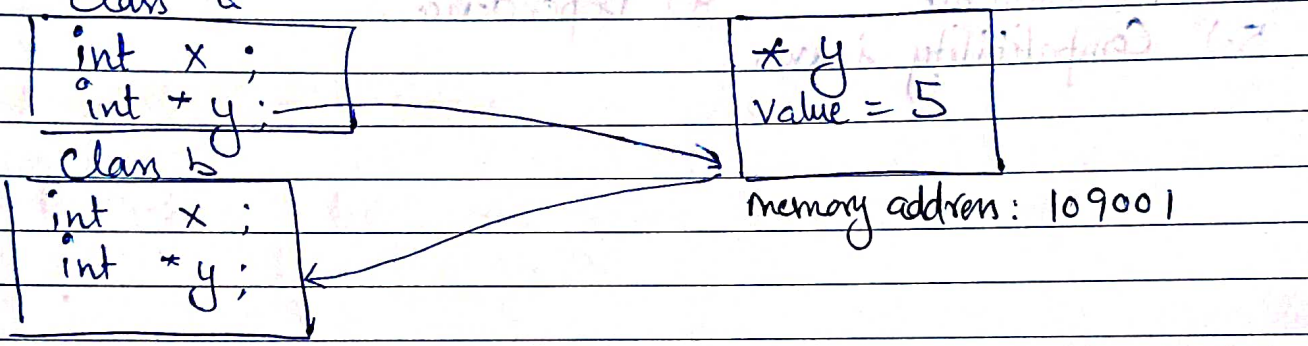
February 2022

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

Shallow - Copy vs Deep Copy.

Shallow Copy : Basically means copying the data only without ~~thinki~~ taking memory allocations and stuffs into considerations.

Deep Copy :- In a Deep Copy, the values of memory allocation are set differently for the object.



In shallow copy the data mem, the memory address of *y will be copied to object b from object a.

and if any change made into that memory location of [pointer *y] then it'll effect the pointer of class b as well.

But in Deep Copy, A new memory will be allocated, so that any changes made in class a shouldn't affect class B.

Shallow Copy :-

1.) When we create a copy of object by copying data of all member variables as it is, then it's called Shallow Copy.

2.) A shallow copy of an object copies all of the member field values.

3.) In Shallow copy, two objects are not independent.

4.) It also creates the copy of the dynamically allocated objects.

Deep Copy :-

1.) When we copy data along with the values that reside outside the object, then it's called Deep Copy.

2.) ~~A shallow copy of an object copies all of the member field values.~~ Deep Copy is performed by implementing our own copy constructor.

3.) It copies all fields, and makes copy of dynamically allocated memory pointed to by the fields.

4.) If we don't create the deep-copy in a rightful way then the copy will point to original with disastrous consequences.

Local Variable :-

→ Written inside a function
→ Accessible inside that function scope only.
→ Scoped.

Global Variable :-

→ Written inside a function
→ ~~Accessible inside the function scope only.~~
→ Accessible to All functions (copy)

→ If both local and global variables are present in a scope, local one will be given priority.
And still if you want to access global variable simply use "::var-name".

i.e ::var-name;

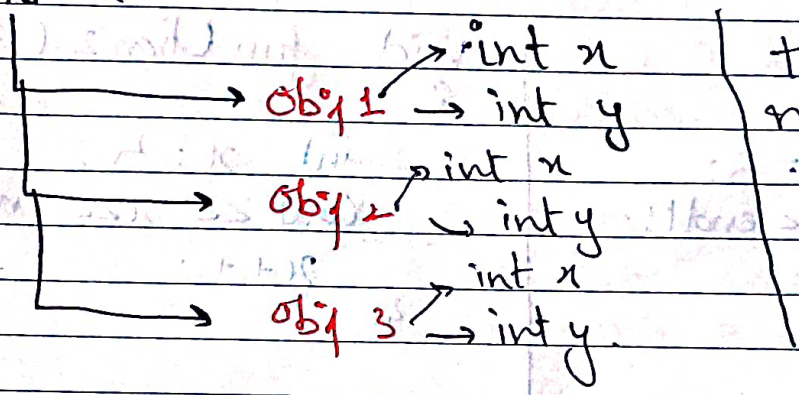
Static- keyword in Class :-

Static - Data - member :- That variable is going to share memory with all of the class instances.

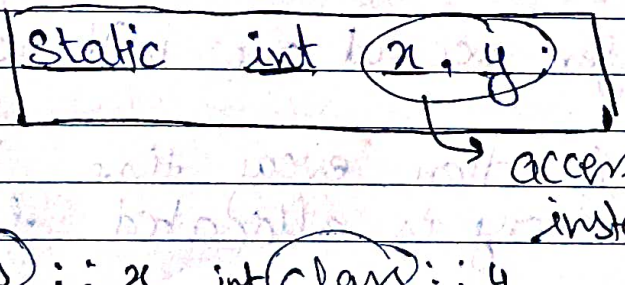
→ Static Member function :- There is no instance of that class being passed into the method.

See, in default

We have classes and there are instances of classes, which in default doesn't share the same data members / memory :



If you want to use same variables (inter-connected variables) we can use static keyword for the same.



int Class :: x ; int Class :: y

Class name →

Static keyword have different meanings when used with different types:-

#Static Variable in a function :-

When a variable is declared static inside a function, space for it gets allocated for the lifetime of the program. Even if the function is called multiple times

Space for static variables is allocated only once and the value of the variable in previous call gets carried throughout the next call.

```
void fun()
{
  static int x = 5;
  cout << x << endl;
  x++
}
```

```
void function2()
{
  int x = 5;
  cout << x << endl;
  x++;
}
```

if we call the same function 5 times

output for first one will be :- 5, 6, 7, 8, 9

whereas output for second one will be :- 5, 5, 5, 5, 5

Cause in second function every time it's getting called new memory is allocated for variable x.

Static Variable in Class :-

As the variables declared as static are initialized only once as they're allocated space in separate static storage so, the static variables in a class are shared by the objects.

→ There can't be multiple copies of the same static variables for different objects. Also because of this reason static variables can't be initialized using Constructers.

```
Class abc {
  public:
  static int x;

};
```

```
int abc :: x = 1;
// We have to initialise it outside
  the class.

Now every instance of class
will have accen to this same
veriable x.
```

Static functions in a class :- Static member functions are allowed to access only the static data members or other static member functions, they can't access the non-static data members or other static member functions.

```
Static void print ()
{

}
```

```
abc :: print ();
```

Abstraction in C++. (Implementation hiding)

→ Delivering only essential information to the outer-world while masking the background details.

→ It's a design and programming method that separates the interface from the implementation.

→ Example:- To Drive a Car u only need to know the driving procen and not the mechanics of Car engine.

Abstraction in Header - files.

→ Function's implementation is hidden in header files.

→ We could use the same program without knowing its inside working.

→ E.g Sort(). for example is used to sort an array, a list or a collection of items.
We know it will sort the Container but we don't know which sorting algorithm it uses to sort that Container.

Abstraction using Classes.

→ Grouping data-members and member functions into classes using access specifiers.

→ A class can choose which data members are visible to the outside world and which are hidden. (can use getters & setters).

★ # What is an Abstract Class ?

→ Classes that contain atleast one pure virtual function, this classes can't be instantiated.

→ It has came from the idea of Abstraction.

Pure Virtual Class function

```
virtual void fun() = 0;
```

Sunday 27

28 MONDAY
FEBRUARY 2-0-2-2 Week-10 (059-306)

February 2022
S M T W T F S S M T W T F S
1 2 3 4 5
6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28

In-line Functions :-

1. An inline function is a regular function that is defined by the inline keyword.

2. The code for an inline function is inserted directly into the code of the calling function by compiler while compiling which can result in faster execution and less overhead compared to regular function calls.

3. Instead of calling function the statements of functions are ~~pasted~~ pasted in calling function

4. Used with small-sized functions, so that the executables are small (handled automatically by compiler optimisation levels).