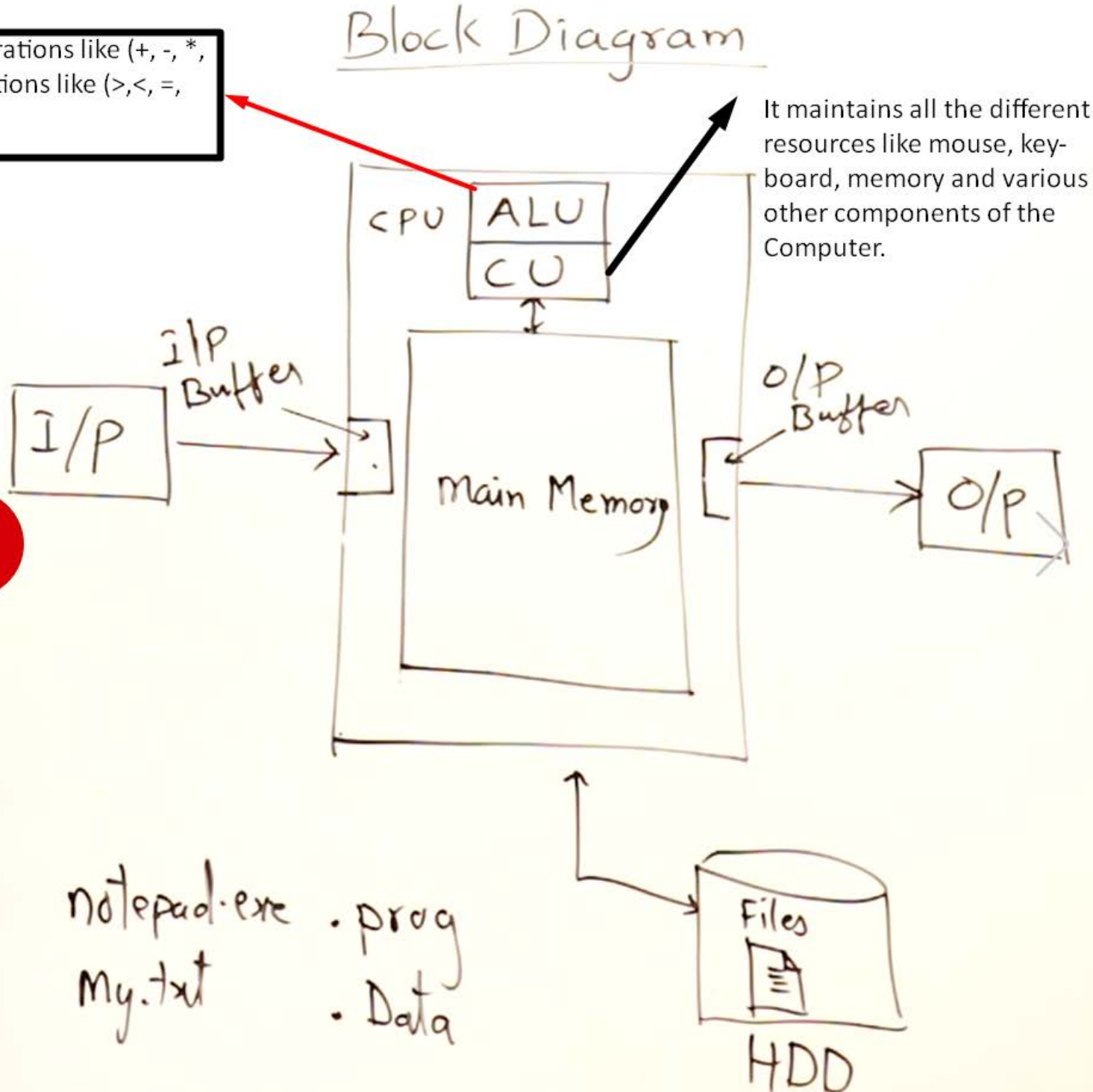
Block Diagram

ALU helps in arithmetic operations like (+, -, *, / , %) and also logical operations like (>,<, =, AND, OR, NOT).

It maintains all the different resources like mouse, key-board, memory and various other components of the Computer.



How Computers Works

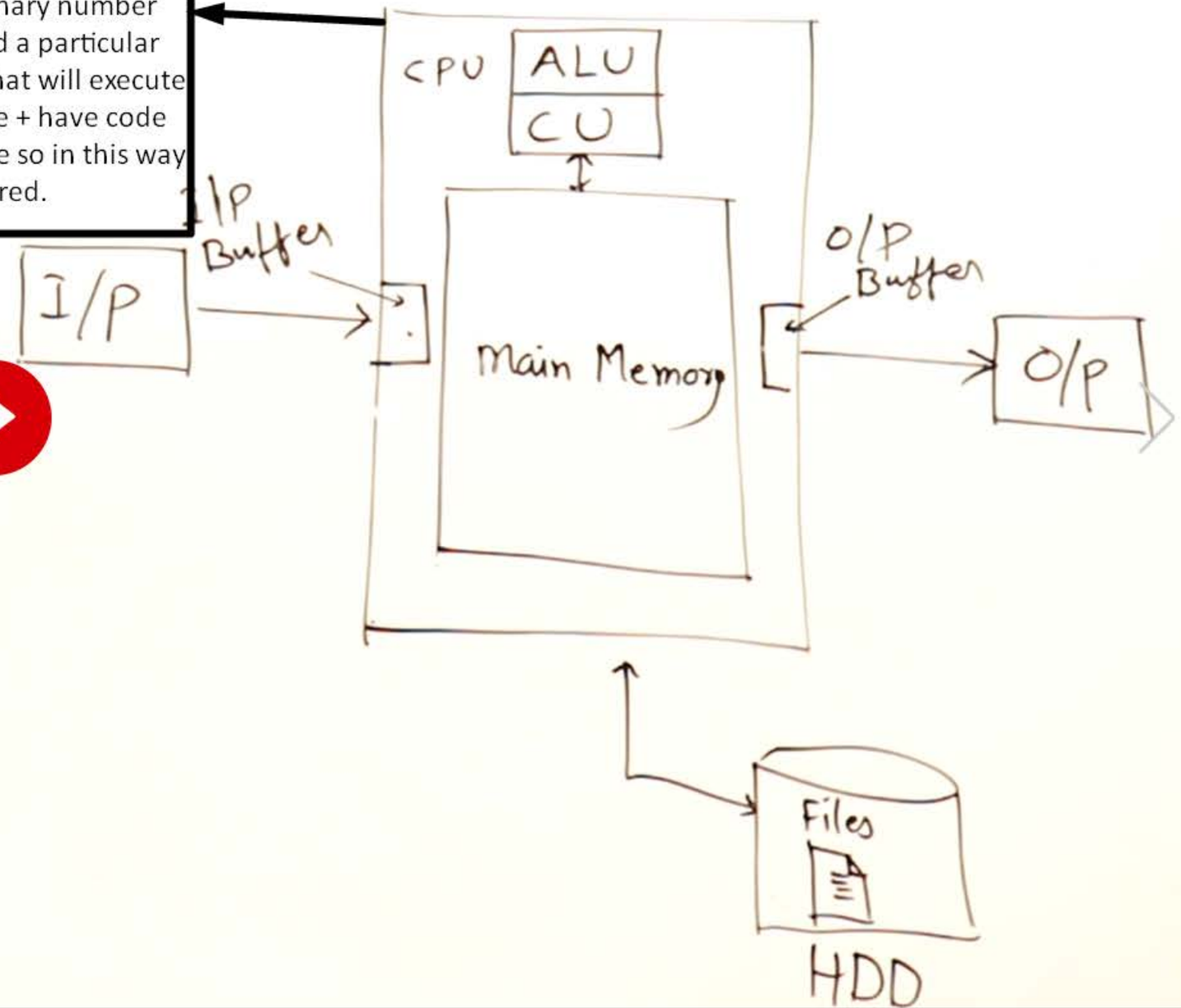
0, 1, 2, - - - 9

0, 1

CPU works upon only two numbers 0 1. These number are called binary number. And for each binary number we already have defined a particular arranged form of code that will execute that instruction. Suppose + have code 0101110101 just suppose so in this way we have information stored.

0 — 0
1 — 1
2 — 10
3 — 11
4 — 100
5 — 101
.
.
.

Block Diagram



Number System

Binary = {0, 1} 10

Octal = {0, 1, 2, 3, 4, 5, 6, 7}

Decimal = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} 10

HexaDecimal = {0, 1, 2, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

Decimal	Binary	octal	HexaDecima
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

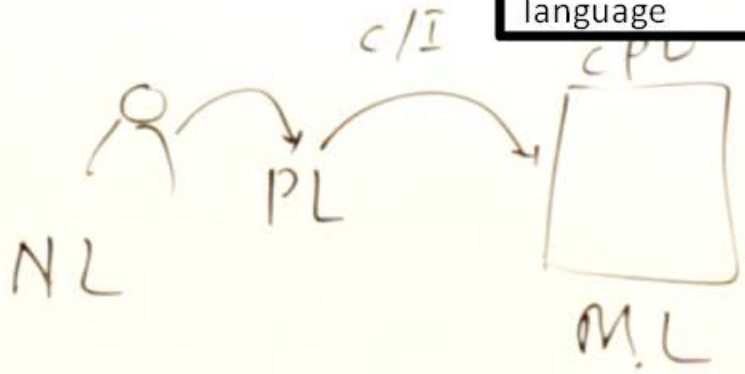
Number system

What is a Program

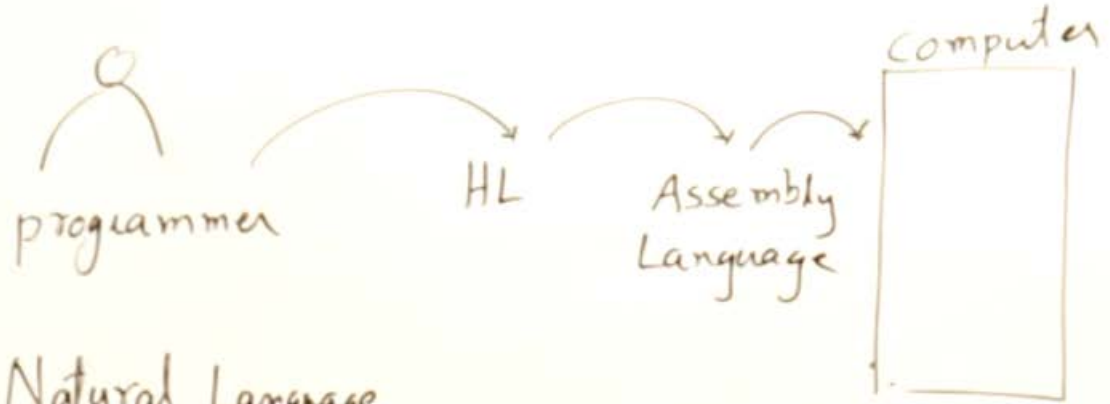
A program is a set of useful data and meaningful instructions that can produce a useful output as per the requirement. No ehn a user want to write program we donot know machine language that is known to computer and also computer donot now our natural langugae so so there is an intermediate between 2 that is programming language and now we write things in programming a pl and then (CI) compiler and interpreter converts it to machine language

List → 10, 15, 9, 6, 8, 4, 12, 16, 7, 5

- Data
- Instruction



Low-Level vs High-Level Language



Natural Language

Machine Language

Low-level

ML

AL

High-Level

C

C++

Java

Python

C#

VB

VC++

10110111000

Binary

Assembly langugae is devloped very first after Machine langugae erlier things are done in machine language for a small period of time but then need asked for new that can be easy to learn and easy to teach and there comes our assembly langugae but this is still tough and called low level language and only used while designing operating system.

Highlevel langugae actually high isliye kha hai ki jo jitna human ke karob hua wo utna high Insani soch wohi wlai bat hai and there are in today we have a number of high level language. Also these laguage can be of three type on is compiler based other is interpretor based and third one is hybrid.

Compiler vs Interpreter

Both of them check for the errors in the program before converting it to machine language. If any it will tell you.

1. check Errors

Then both of them convert the code into machine langauge.

2. Convert into MC

3. Execution

Compiler have no plan for executing the code. While Interpretor do both the task.

Compiler vs Interpreter

C++

First.cpp

```
int add(int a,int b)
{
   int c;
   c=a+b;
   return c;
}
int main()
{
   int x,y,z;
   x=10;
   y=15;
   z=add(x,y);
   cout<<z;
}
```

compiler

First.exe

add

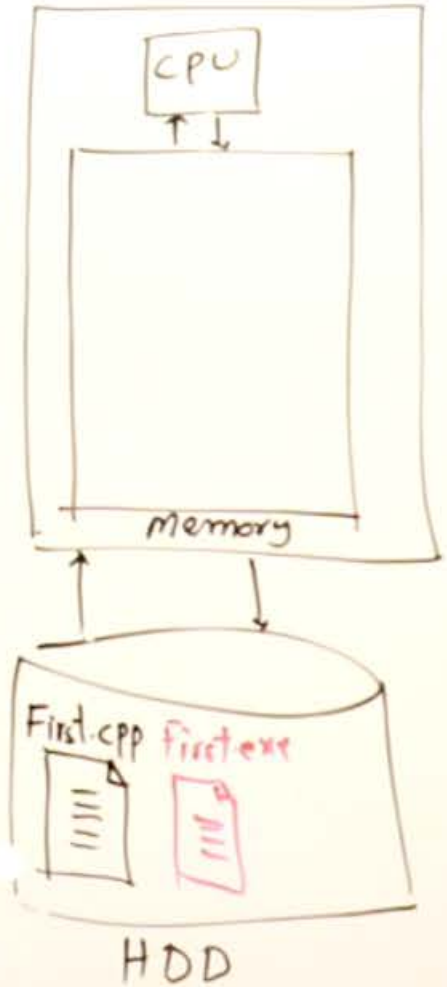
main

1. In compiler if there is no error conversion of code from programming language to machine language for each of the function like here two functions are converted and both seprate.

2. Now after sucessful compilation it produces a executable .exe file and we can use it to run our program.

3. If compiled once then there is no need to compile agian unless and until you made any kind of change in your program.

4. C++ is an example of this.



```
int add(int a,int b)
{
  int c;
  c=a+b;
  return c;
}

main()
{
  int x,y,z;
  x=10;
  y=15;
  z=add(x,y);
  cout<<z;
}
```

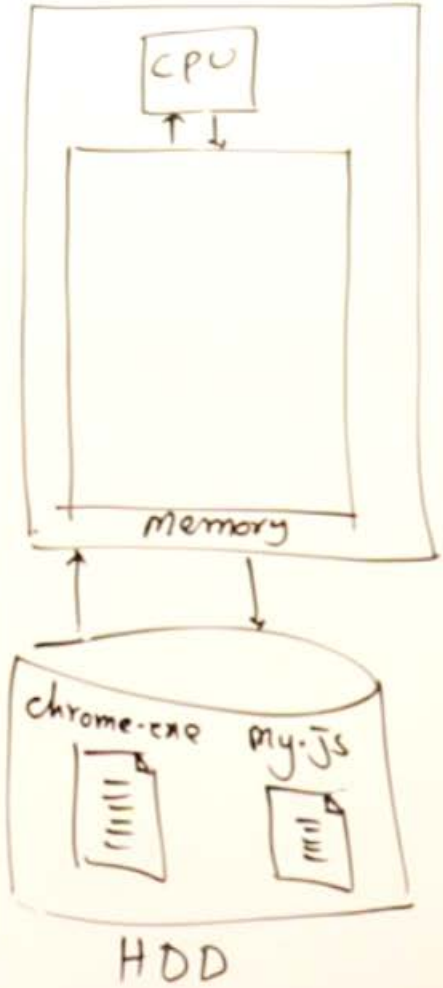
Javascript

```
function add(var x,var y)
{
  var z;
  z=x+y;
  return z;
}
```

1. Interpretor order of running a program is check error then convert that line and then execute that same line and then move to next line.

2. Now this is slow since every time we need to compile for running. It works as javascrpt used to design the webpages. So like we write a program in javascript and executed it with the help of say chrome.exe then in chrome it will execute first line no error it will execute 1st then move to second and so on.

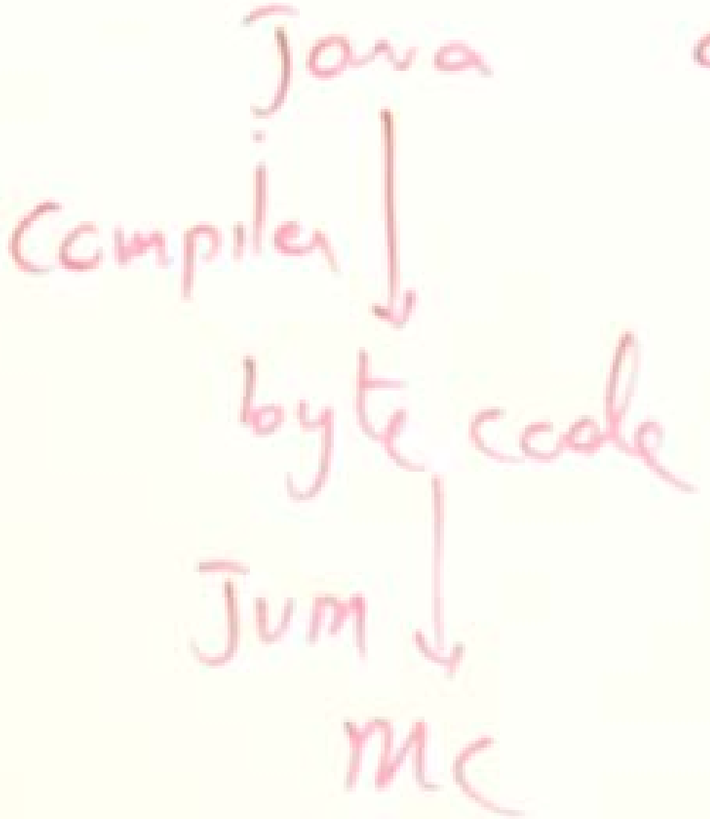
3. This is but easy a bit reason being ki we can point out our error easily that in which line exactly the error is like jha tk hua hai execute wha tk to thik hi hai aage uske jo hai wo thik se dekhlo.



Interpretor and compiler

Suppose you want to cook a german food you have recipie but donot know german so for that now one of your friend know german well so in that case he can help you in two ways.
One is ki he will say send me file and Then he will send you a copletely translated file and also he will check ki yar mere dost ke sath kisi ne mzak na kiya ho kuch harmful hua isme to mai translate nhi karoonga btadoonga ki ye poisonius hai ya whatever. Now if nothing wrong in recipie then he will send it to you the translated copy of that recipie and now the time you get this recipie you donot need your friend anymore for this purpose. This is what compiler do.

Now in case man lo ki tumne us dost ko kha ki ghar aa ja ek recipie hai bna ke khilaoonga bas recipie ko tu sath mai translate kario to he will say okay and jab mile tum dono to kha chl ho ja suru usne kha chl mashroom nikal aab mirch nikal and one by one and in case bich mai likha hua ho ki aab jhar nikal to khega ye khtrnak hai bhai rhne de this is what interpretor do.



1. Hybrid code so phle isme compiler hai ek jo ki start karega compile karna and if no error then it will produce a byte code.

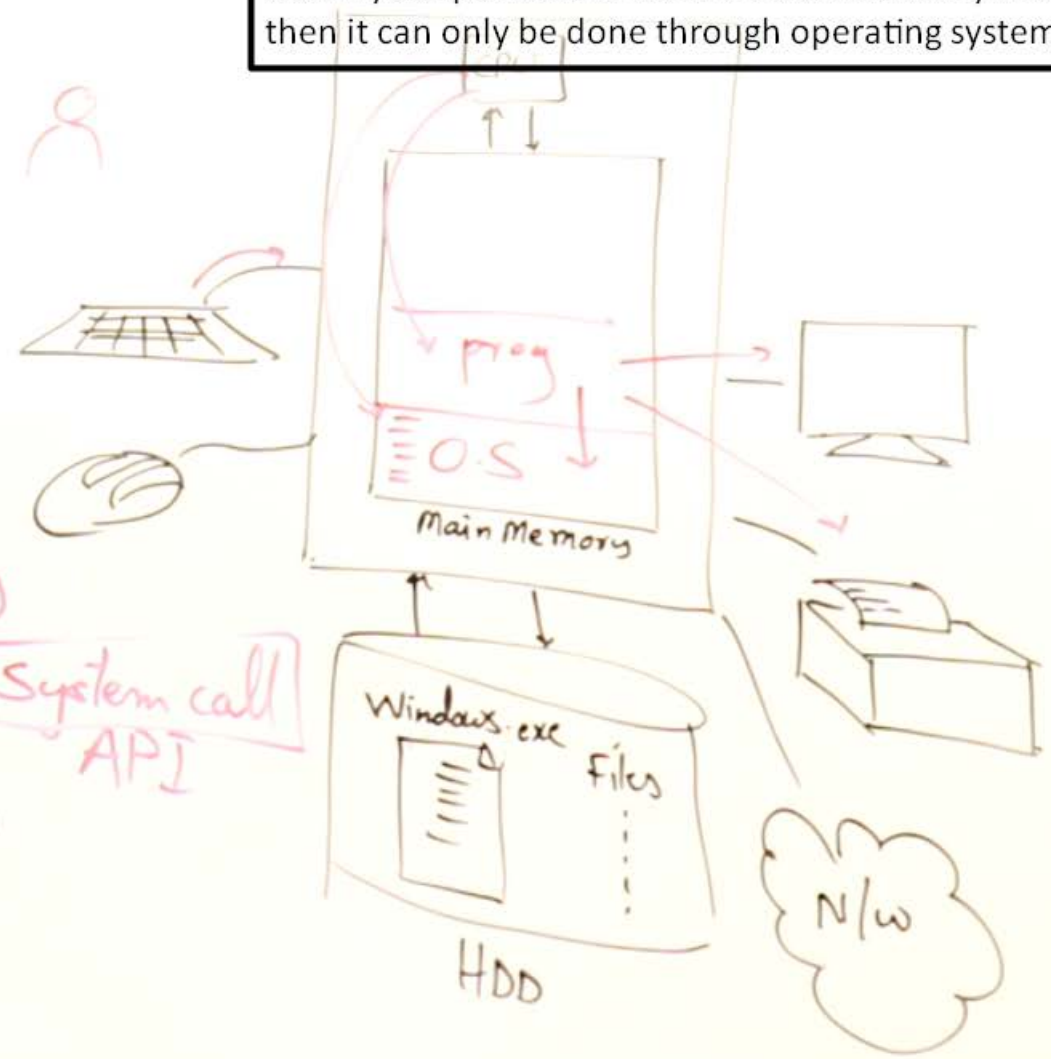
2. Now aab jab compile hoke byte code aa gya hai then after that ek interpreter like java ke case mai hai JVM (java virtual machine) to this will now work as interpretor and then check convert and execute the byte code.

3. Java and c# are a few example of this.

Operating System

Operating system is a master program that get executed very first when you open your device it gets loaded to main memory and then for any work you as a user want to be done it will be executed only via os. Also if you cpu need to acess a attached utility like lcd printer netwrok then it can only be done through operating system.

1. Window
2. Linux
3. Mac osx
4. Android
5. ios

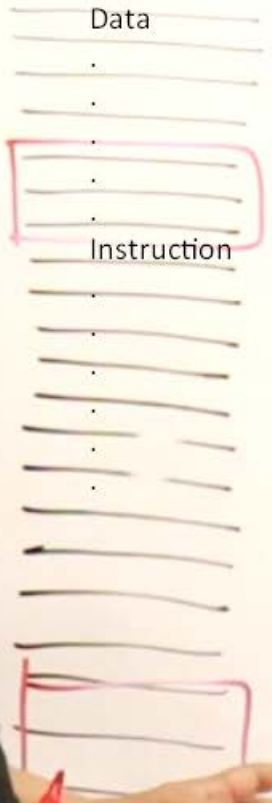


So in this khichri bna do data bhi yhi instruction bhi aur like ek add ko do bar karna pde to dono bar likho.

Programming Paradigm

Monolithic / Basic

program



procedural / Modular

C-Language

In this thora better hai ki function like koi repeting part hai to usko likhna nhi parega

```
function1( )
{
   ≡
}
function2( )
{
   ≡
}
   ⋮
main( )
{
   function1( );
   function2( );
   function3( );
   function4( );
}
```

Modular

This is better then previous ki chlo isme data type agr ek hi trh ka hai to 10 bar usko initialize nhi karna parega

```
structure Info
{
   data1;
   data2;
   data3;
}

function1(Info)
{
   ≡
}

function2(Info)
{
   ≡
}
   ⋮
main()
{
   struct Info i;
   function1(i);
   function2(i);
   ⋮
}
```

Object-oriented

This is the most used and Ham yhi use kar rhe hai like java c++ mai ye hi method hai ki ek class bna ke all the data and function are all together assembled in one place. Hence best method of programming is object oriented programming.

```
class Info
{
   data1;
   data2;
   data3;
   function1( )
   { ≡ }
   function2( )
   { ≡ }
   ⋮
};

main( )
{
   Info i;
   i.function1( );
   i.function2( );
   ⋮
}
```

Algorithm

A set of instruction in lay man words that we can use to solve a problem it is called algorithm. You can use any language to write this algorithm bas the thing is ki apko smjh mai aani chahiye meaning that is enough. This is generally written as pseudo code but have no meaning ki agr syntax mai glti hoo gai to kya hoga.

Program is also a well defined collection of data and instructions band have some useful meaning to solve a problem but it must need to be in a format and must obey the rules laid down for the programming language that is predefined like for c++ has its way of programming Java have it's own.

Algorithm ⟶ program

↑

Psuedocode

C++

Syntax

1. Algorithm
2. Psuedocode
3. Program

Algo example

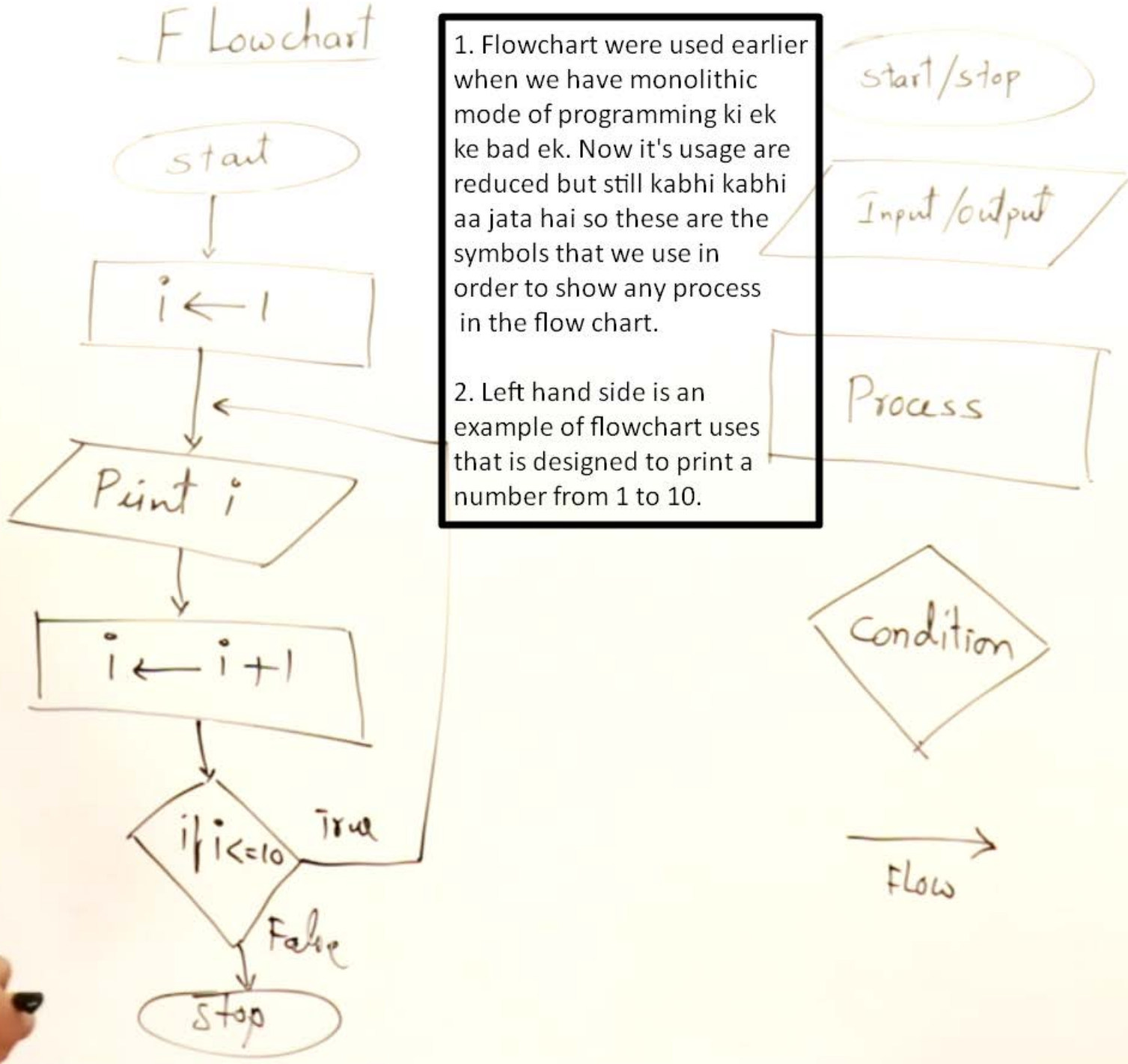
Algorithm

```
Algorithm Average (List, n)
{
    sum := 0;
    for each element x in List do
    Begin
        sum := sum + x;
    End
    avg := sum/n;
    return avg;
}
```

Program example not complete program only a function to understand

```
float Average (int L[], int n)
{
    float sum=0.0;
    for(i=0; i<n; i++)
    {
        sum = sum + L[i];
    }
    float avg=sum/n;
    return avg;
}
```

Mtlb kuch bhilikho bracket hi ho jaroori nhi can be anything bas tumhe aur jiske liye hai likha use smjh aana chahiye.


F Lowchart
start
i ← 1
Print i
i ← i + 1
if i<=10
True
False
Stop
1. Flowchart were used earlier when we have monolithic mode of programming ki ek ke bad ek. Now it's usage are reduced but still kabhi kabhi aa jata hai so these are the symbols that we use in order to show any process in the flow chart.
2. Left hand side is an example of flowchart uses that is designed to print a number from 1 to 10.
start/stop
Input/output
Process
Condition
Flow

First.cpp

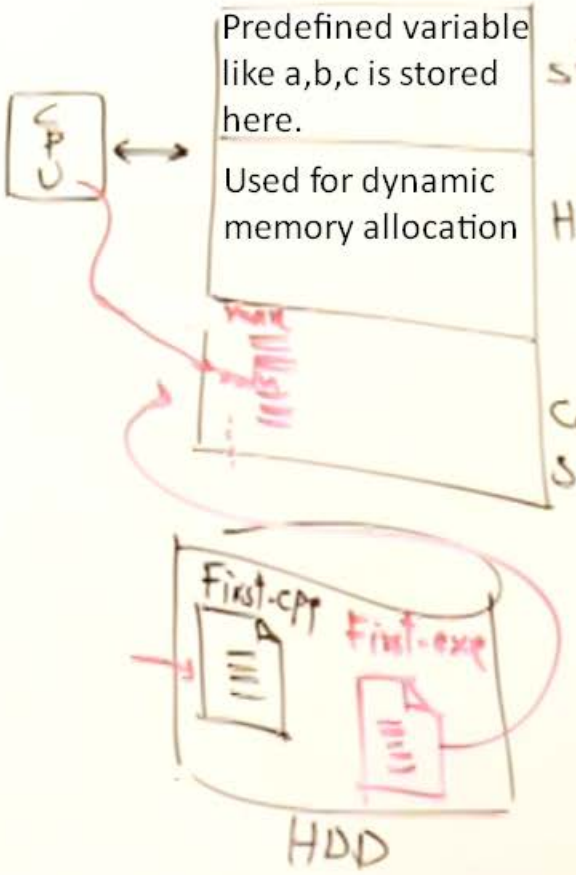
Steps for program Development & Execution

```
#include<iostream>

int max(int x, int y)
{
    if (x>y)
        return x;
    else
        return y;
}

int main()
{
    int a=10, b=5, c;
    c=max(a,b);
    std::cout<<c;
    return 0;
}
```

Order of execution a program in C++

1. Editing
2. Compiling
3. Linking Library
4. Loading
5. Execution

3. Library gives you acess to a lot of things that are needed to be accessed but it is not good to do it again like pow cos log etc. These can accessed through header files via proper library.



4.loading is ki aab compile wgera ho gya hai to main memoy mai load karna parega tki sab data type aur function ka apna space allocate kar ske.

11:28

First Program

First-cpp

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

int main()
{

    cout<<"Hello world";


    return 0;
}
```

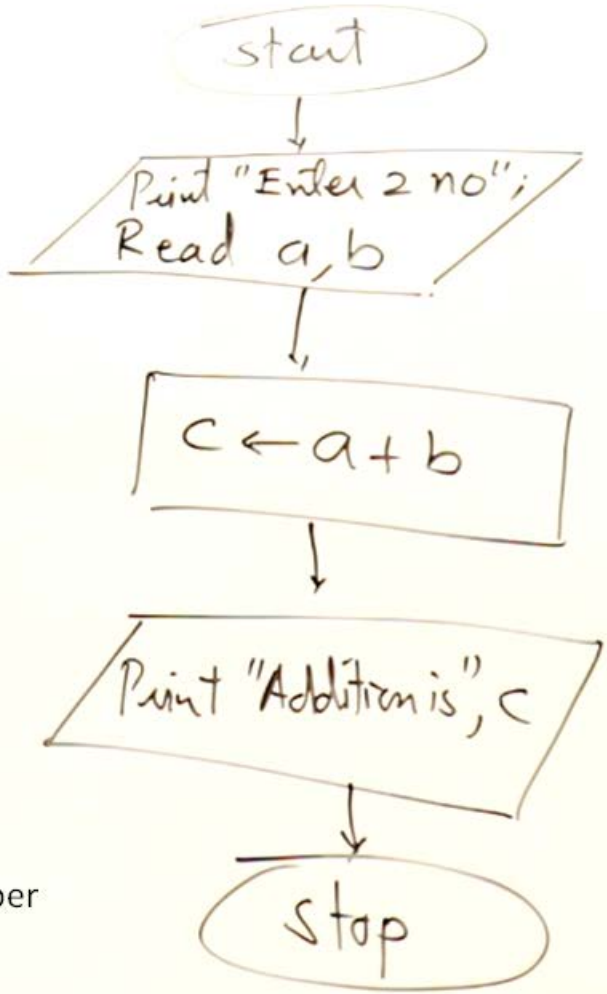
Console output

How to write a Program

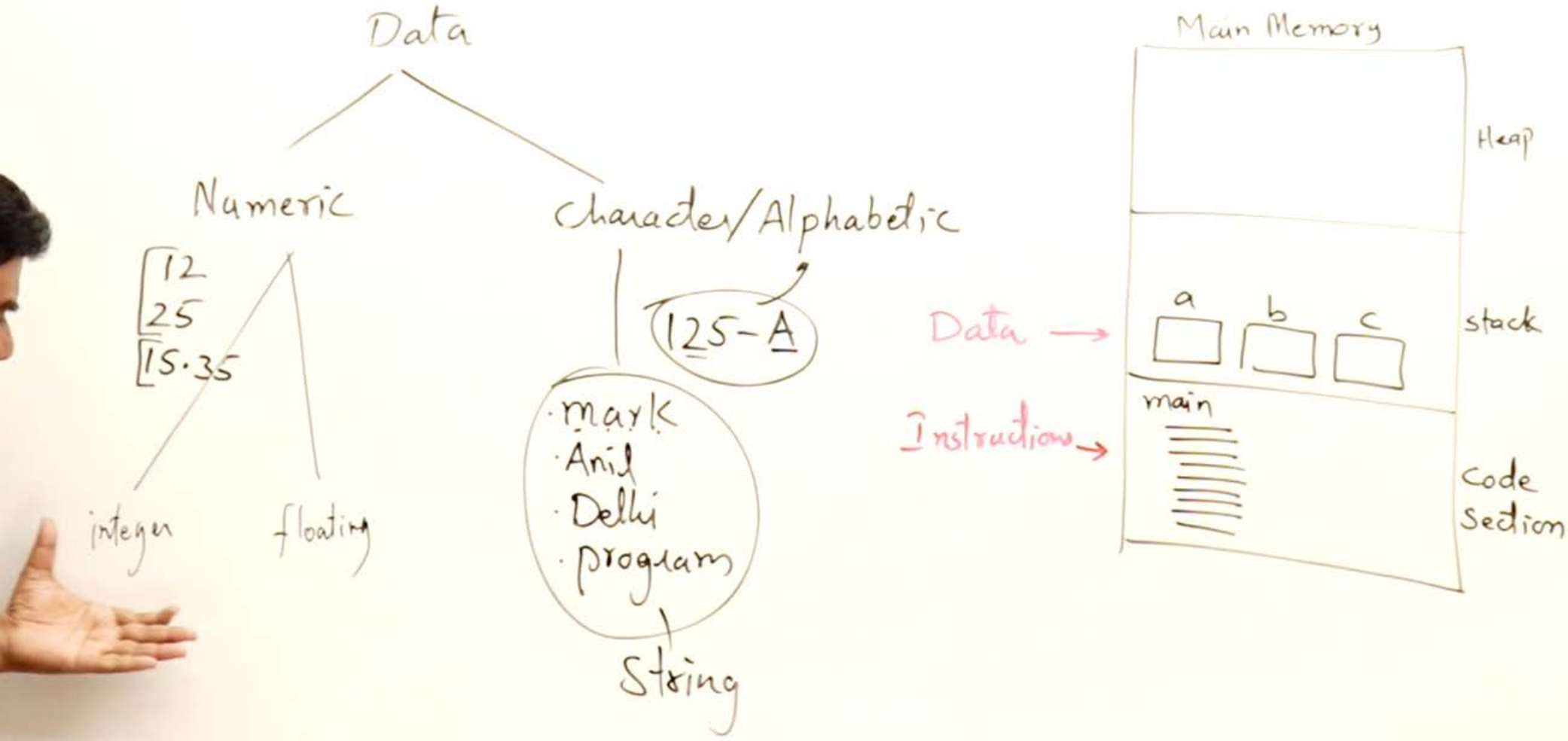
```
Algorith Add
Begin
    Print "Enter 2 no"
    Read a, b
    c ← a + b
    Print "Addition is", c
end.
```

Example here to add two number

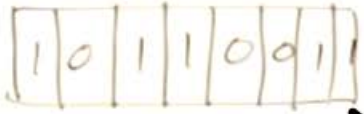
Flowchart
Adding 2 Numbers



Why Data Types



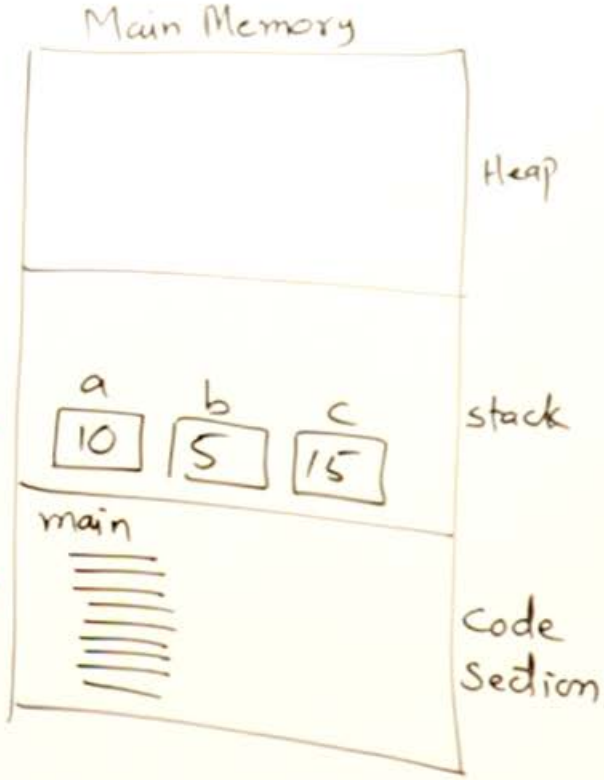
1 byte



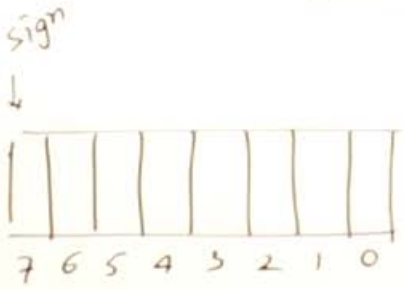
0 – 255

How data is stored in memory look for each byte

```
int main()
{
    int a, b, c;

    a = 10;
    b = 5;
    c = a + b;
    .
    .
    .
```



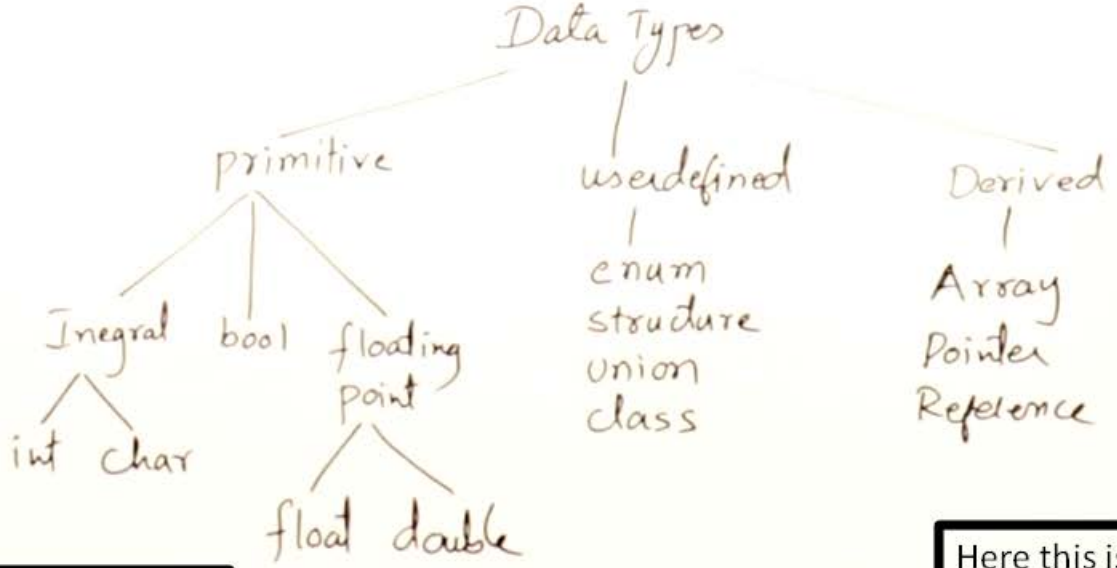
Data Types and Variables



$2^7 = 128$

0 – 127

-128 to 127



ASCII

A – 65	a – 97	0 – 48
B – 66	b – 98	1 – 49
⋮	⋮	⋮
Z – 90	z – 122	9 – 57

Data Type	size	Range
int	2 or 4	-32768 to 32767
float	4	-3.4×10^{-38} to 3.4×10^{38}
double	8	-1.7×10^{-308} to 1.7×10^{308}
char	1	-128 to 127
bool	undefined	true / false

Here this is for 2 byte but you may find for 4 byte

Important and good to remeber the points that are marked with a arrow.

Modifiers

- unsigned
- long

Modifiers are used to increase the range as per requirement for instance if we need to store only positive number then we can modify int to unsigned int and it will increase the range of int.
Same you can do with char.
Also another modifier is long that can be used with int and double only.

unsigned int

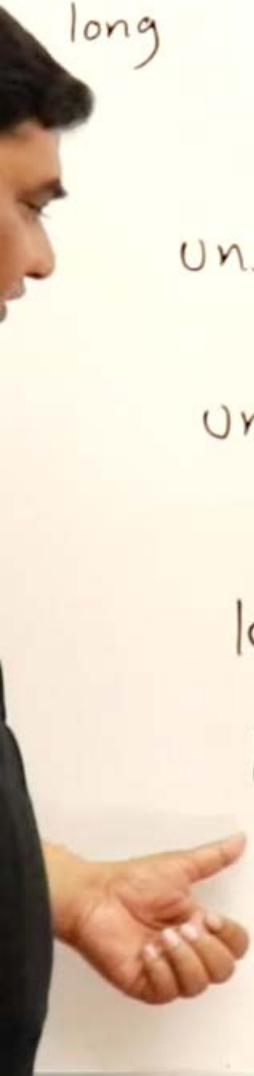
0 – 65535

unsigned char

0 – 255

long int — 4 bytes
8 bytes

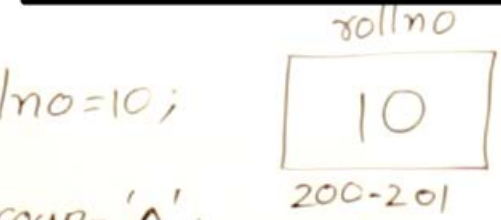
long double — 10 bytes



Data Types and Variables

```
int main()
{
    int rollno=10;
    char group='A';
    //Initialisation of char should be in ' '
    float price=12.75f;
//It is good to write a f at end of float
otherwise it is treated may be as double
```

Variable are storage house to store our data




```
✓ int x1;
✗ int 1x;
✓ int rollno;
✗ int roll no;
✓ int roll_no;
✓ int rollNo;
✓ int RollNo;
```

Valid and invalid variable name
Also it is advised that you use meaningful variable name it will be good for you only to understand better

$f(x) = 3x^2 + 4x$

Operators & Expressions

Arithmetic operators are used for add(+), subtract (-), multiply(*), Divide(/), and reminder(%)modulo

Arithmetic — +, -, *, /, %

Used to check the given condition between two of the available options like if this is true or not if true return 1 or true and if not valid then return 0 or false.

Relational — <, <=, >, >=, ==

It consisits of some logics like some things can run on logic if both switch are on simmilar like that.

Logical — &&, !!, !

Bitwise Operators

Need any two variable to operate very first convert them to binary and then for every bit perform and operation on each bit and then you will get a binary value and also the output produced will be in decimal that is binary equivalent of this binary number same is for OR operator. and XOR operator.

& and Both true is true

| or Any true is whole true

^ x-OR Dono opposite to true

~ not

Simple words mai khe to har bit ko ek ek left shift sirf most significant bit jo hai last wla sign ka wo nhi affect karega First one is left shift operator (<<) and next one is right shift operator (>>) in simple words if I say if a value is passed through an (variable << i) it means the value of that int will multiply by 2^i times. and for right shift the value will be divided by 2^i times and actually written as eg;

<<

>>

```
int a=16;
cout<<(a<<2); // It will produce 64 that is 16*(2^2)=64
cout>>(a>>1); it will produce 8 that is 16/(2^1)=8
```

Two's complement method use karta hai ye phle to sabko binary mai lao phir jo hai usko palto phir again sabka not leke dono ka binary addition karo that will be the answer

Logical — &&, ||, !

Bitwise — &, |, ~, ^

Increment/Decrement — ++, --

Assignment — =

Increment operator is used to increadse and decrease value two types are there one is post and other pre post means use and change and pre means change and use.

Used for assigning the value to the variable or any data assigns right value to left variable always. not as math that a=b means b=a;

Precedence order top is highest prior and bototom is least

Operator	Assumed Precedence
( )	3
*, /, %	2
+, −	1

$x = a + b * c - d / e$

3 1 4 2

$x = (a + b) * (c - d) / e$

Law of assosiativity it means if two operator of same precdence are there then operate in order form left to right like in above * and / both are here hence first see which comes first so we have * so it will be performed first that is b*c and later we will have d/e in next step.

Now in order to increase the precedence of a operator we use bracket so that it's precdence is increased that we can see in second expression above.

Compound Assignment

In one word previous value jo hai variable ki wo ek operand hua aur = ke aage jo likha hai wo dusra operand now jo bhi operation hai needed perform it and store in that variable itself.

`+=`	It is same as a=a+(some thing)
`-=`	Equivalent to a = a - something
`*=`	a = a * something
`/=`	a = a / something
`%=`	a = a % something

`&=`

`|=`

`<<=`

`>>=`

Same inka bhi hai wese hi

.
.
.

pre inc

```
++x;
```

post inc

```
x++;
```

pre dec

```
--x;
```

Post dec

```
x--;
```

Overflow

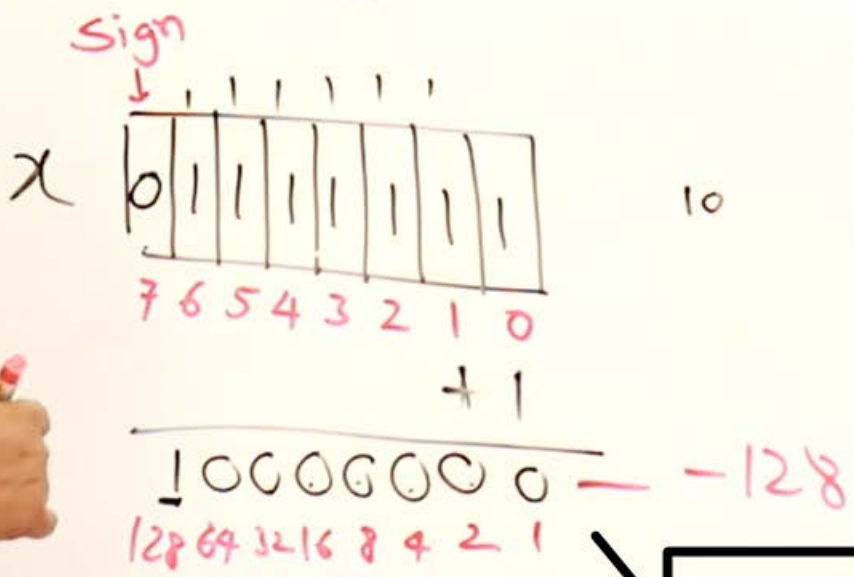
Range of char
-128 to 127

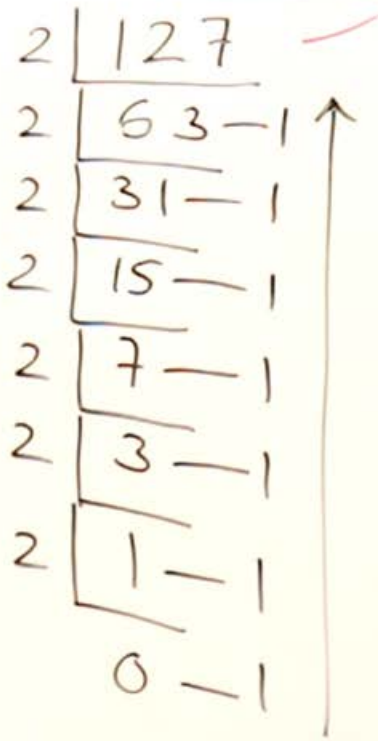
char x=127

++x;

cout<<(int)x;

-128





When all the data in range of a data type is excedded than it goes back to the the element that is in the starting of it. Simple words mai khe to char ka range hai -128 to 127 now apne kha char x=127 and then ++x; so it will move now to x = -128. So it means when you reach it is overflow condition that is exceeding the available upper range of data and in any programing language when this overflow is attained the variable again have begining value.

This is how it is done in any langugae.

Enum and Typedef

User defined data type. But must be made from some exisiting data type.

Departments

CS — 1
ECE — 2
IT — 3
Civil — 4

Menu

File
New — 0
Open — 1
Save — 2
close — 3

Days

Mon — 0
Tue — 1
Wed — 2
⋮

Card Shapes

club — 0
Spade — 1
Diamond — 2
Hearts — 3

Feedback

poor
satisfactory
good
excellent

```
enum dept {cs=1,ece,it,civil}
          1  2   3  4
int main()
{
  dept d;
  d=cs;
  ⋮
```

Now when you want ki 0 se na ho ke 1 ya koi aur value se suru ho to you can make and then use like this.

```
enum day {mon,tue,wed,thu,fri,sat,sun};
         0   1   2   3   4   5   6
          ↑ data type
int main()
{
  day d;
  d=mon;
  d=fri;
  d=0;
```

Now we have like we made day our new user defined data type by default it will have value that are from 0 to end and then finally we can use it in our main function but correct method to use it is shown here in this.

```
if(d==mon)
```

enum day {mon=1, tue, wed=5, thur, fri, sat=9, sun}

You can also brak the count by using the above method.

Enum and Typedef

User defined data type to make program more readable

```
typedef int marks;
typedef int rollno;

int main()
{
    marks m1,m2,m2;
    rollno r1,r2,r3;
}
```

```
int main()
{
    int m1,m2,m3,r1,r2,r3;
}
```

Here we first told ki bhaiya hm typedef kar rhe hai aur in short khe to int type ka hi ek hmsakal bna rhe hai uski sari property int ki hogi to aab int marks hai to marks likh ke int type ka variable to create kar hi skte hai aur sath mai ye bhi fayda ki aage jake bhuloge nhi ki ye wla int kiske liye hai ya rkha tha.

Bad habbit hai koi kaise bhi kar do variable likh diya kuch bhi aur do sal bad aaye to yar ye kya kiya tha kis variable ke liye hai ye to uski problem so to overcome this we have option to create a user defined datatype that is basically an alias for predefined data type see on left for more,

Conditional Statement

Relational

<

<=

>

>=

==

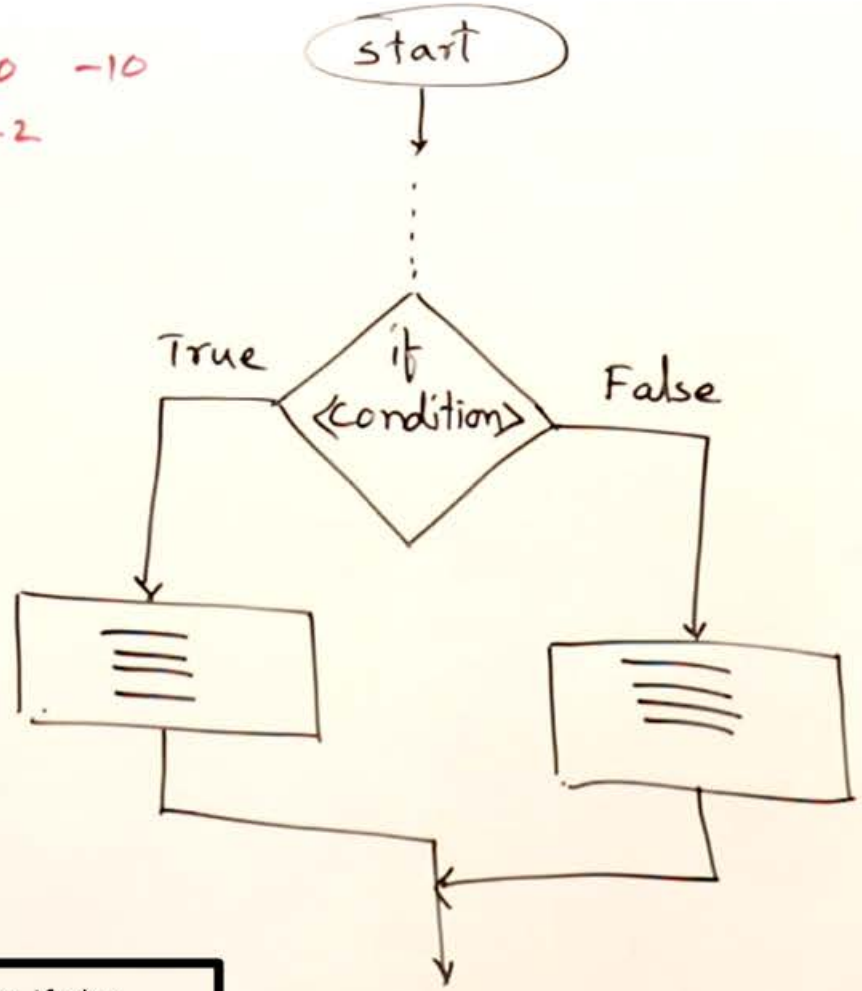
!=

false — 0

true — 1

10 -10 -2

```
if (<condition>)
{
    ===
}
else
{
    ===
}
```



Also we do use logical operators apart from these relational operators. That we talk in some other section.

Any Integer value is true for an if else condition but actually it returns 1 for true and always 0 for false.

Conditional Statements

- If and else is used for writing conditional statement
- If condition is true then if block is executed
- If condition is false then else block is executed
- 0 - means false
- 1- means true or non0zero value is also true

If can be nested inside if as well as else statement
Nesting of is is also written as else-if ladder

Program to Demonstrate Simple Conditional Statement

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

int main()
{
    int roll;
    cout<<"Enter your Roll No."<<endl;
    cin>>roll;
    if(roll>0)
    {
        cout<<"Valid Roll No."<<endl;
    }
    else
    {
        cout<<"Invalid Roll No."<<endl;
    }
    return 0;
}
```

Logical Operators

$x < y$ $x < z$

You read about relational operator already now logical are here these are actually used in case if we need to check condition with two expression in same conditional box as shown in above in between that we can use these realtaional operators.

Logical Operators

&& AND

|| OR

! NOT

Relational Operators

<

<=

>

>=

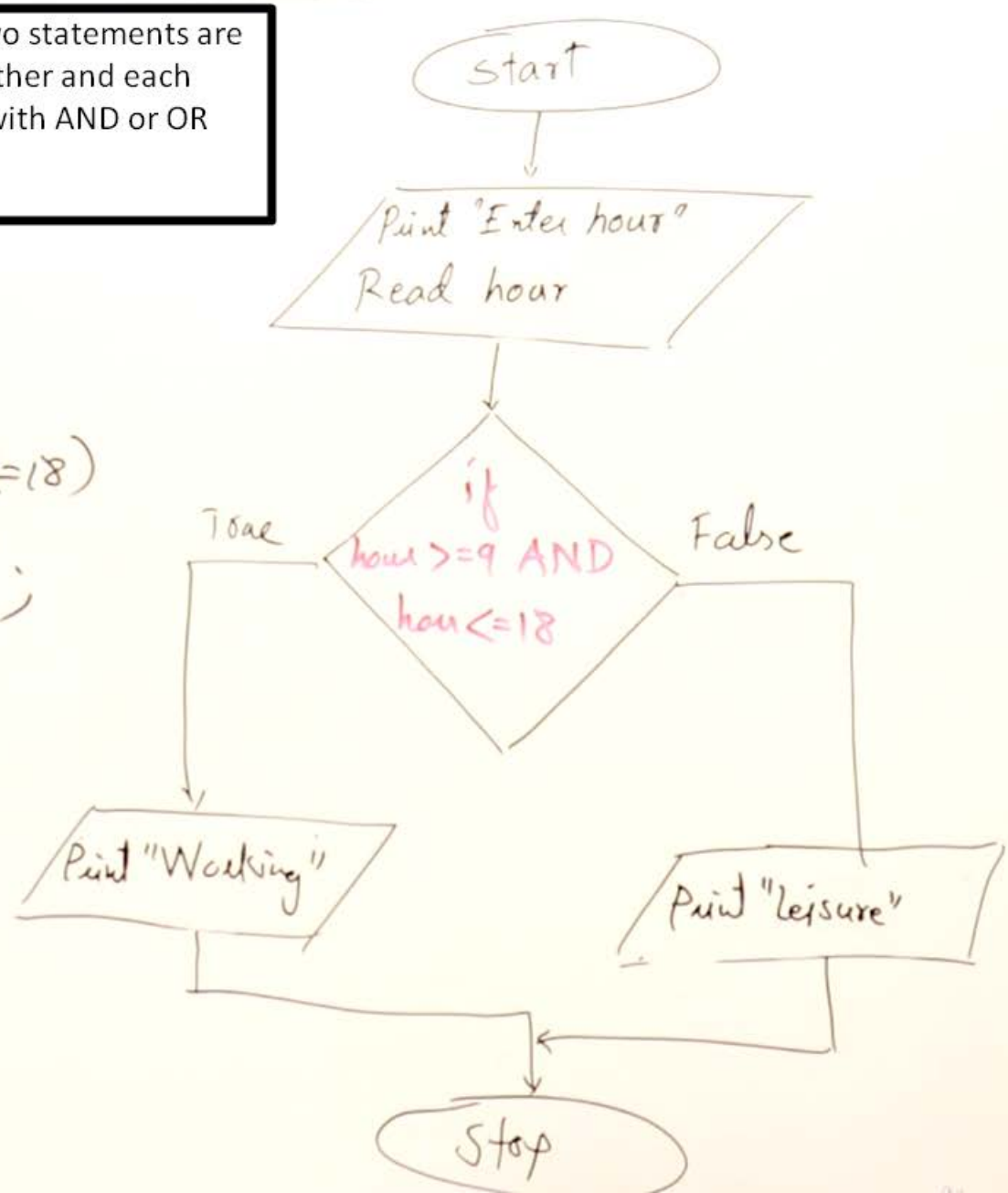
==

!=

Compound Conditional Statement

More than two statements are clubbed together and each are atached with AND or OR symbols.

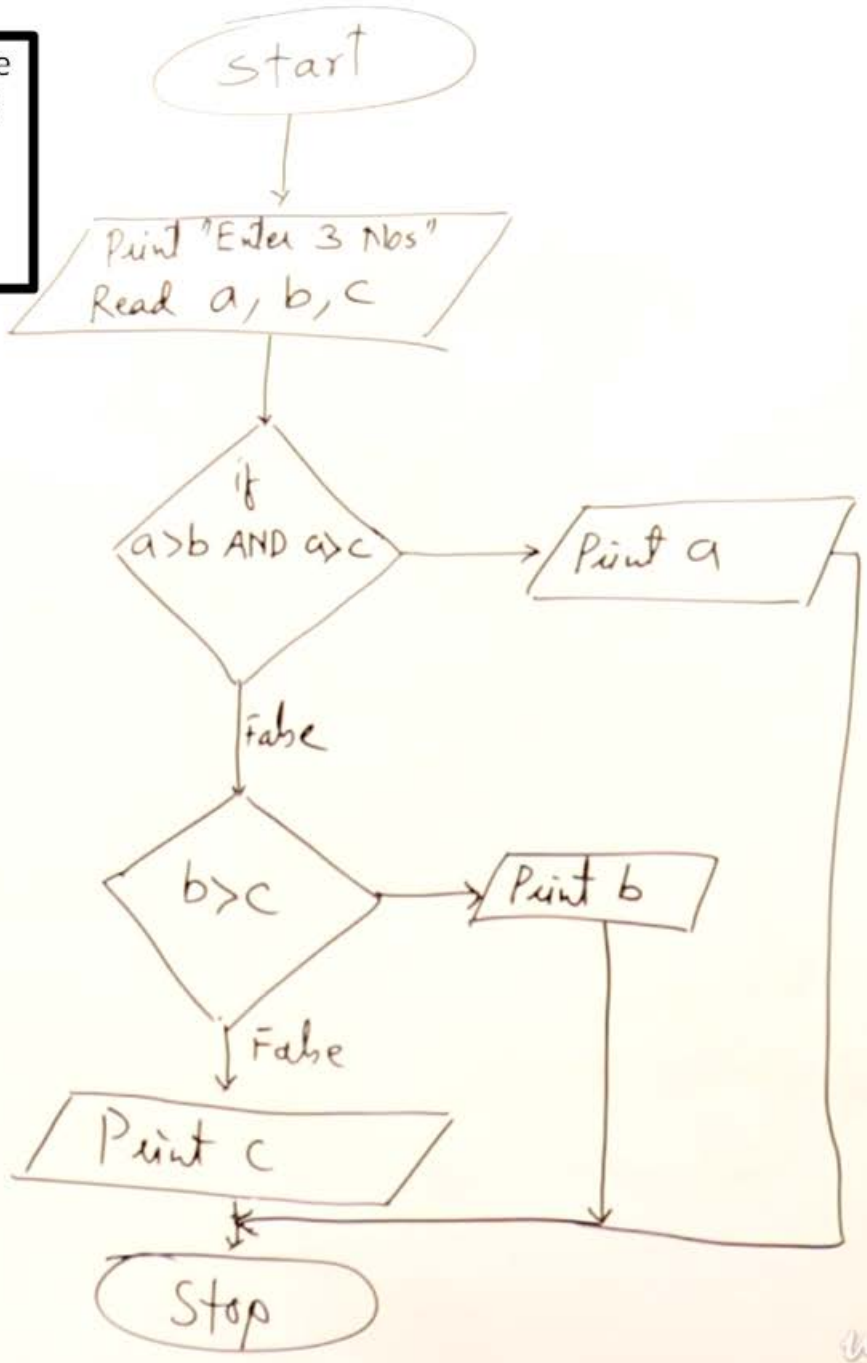
```
int main()
{
    int hour;
    cout<<"Enter hours";
    cin>>hour;
    if(hour>=9 && hour<=18)
    {
        cout<<"working";
    }
    else
        cout<<"Leisure";
    }
    return 0;
}
```



Nested if

If in an if block we have more new if statement and in that if as well we have if else statements then that is called Nested if condition.

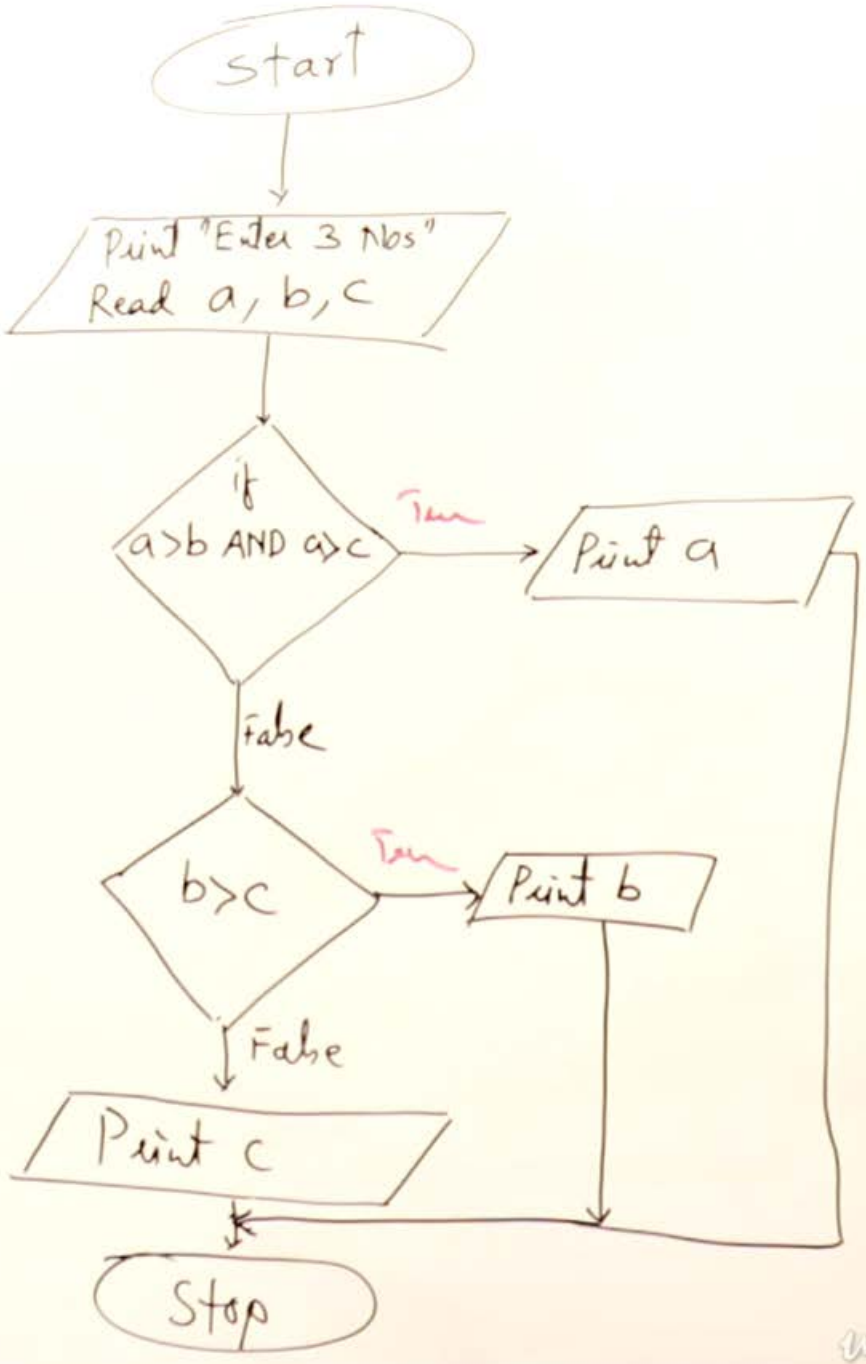
```
if (<condition>)
{
    if (<condition>)
    {
        =
    }
    else
    {
        =
    }
}
else
{
    if (<condition>)
    {
    }
    else
    {
        =
    }
}
```



Nested if

Max of 3 Numbers

```
int main()
{
  int a, b, c;
  cout<<"Enter 3 nos";
  cin>>a>>b>>c;
  if(a>b && a>c)
  {
    cout<<a;
  }
  else if(b>c)
    cout<<b;
  else
    cout<<c;
}
```



Nature of Root

```
int main()
{
    float a,b,c,d,r1,r2;

    cout<<"Enter a, b and c";
    cin>>a>>b>>c;

    d=b*b-4*a*c;

    if(d==0)
    {
        cout<<"roots are real and equal";
        cout<<endl<<(-b/(2*a));

    }
    else if(d>0)
    {
        cout<<"roots are real and unequal";
        cout<<endl<<(-b+sqrt(d))/(2*a);
        cout<<endl<<
    }
    else
        cout<<"Imaginary";
```

Quadratic Equation

$$ax^2+bx+c=0$$

$$roots = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

$\frac{-b+0}{2a}$ $\frac{-b-0}{2a}$

discriminant

$$d = b^2-4ac$$

if $d=0$, real and equal

if $d>0$, real and unequal

if $d<0$, imaginary

else if Ladder

```
int main()
{
    int day;
    cout<<"Enter day number";
    cin>>day;
    if(day==1)
    {
        cout<<"Mon"<<endl;
    }
    else if(day==2)
    {
        cout<<"Tue";
    }
    else if(day==3)
    {
        cout<<"wed";
    }
    .....
    else cout<<"invalid day";
}
```

Else if ladder

```
if(<condition>)
{
    ——
}
else if(<condition>)
{
    ——
}
else if(<condition>)
{
    ——
}
else
{
    ——
}
```

Short circuit

In this case if first condition is false then it won't go to next one hence Short circuit condition.

```
if ( a > b && a > c )
     3   5
     F
```

Logical

&&

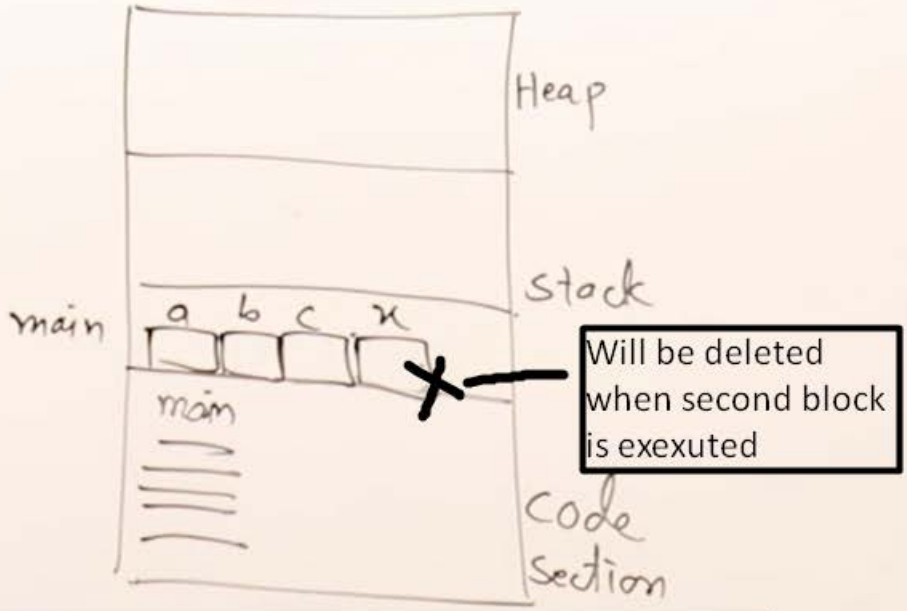
||

```
if ( a > b || a > c )
     5   2
     T
```

Simmilar is in this case if condition 1 is true then it will not move ahead to check for condition 2 . This condition is again called short circuit.

Dynamic Declaration

```
int main()
{
    int a,b,c,x;

    if( )
    {
        int m;
    }

    if( )
    {
        int x;
    }
}
```



In some case we need a variable for a particular time aur nor earlier to it nor beyond this we need it so here it goes decleare it in block where you need like if first condition here is true it will create m and then when clock is over m will be removed same for x also not only this for loop mai int i wo isliye good hai ki bhaiya bas is block tk hua use age nhi par ye feature c nhi deta hence c++ is good with this dynamic allocation

Dynamic Declaration

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

int main()
{
    int a=10,b=5;

    if(true)
    {
        int c=a+b;
        cout<<c<<endl;
    }

    {
    int d=a-b;
    if(true)
    {
        cout<<d<<endl;
        }
    }

    if(int e=a*b)
    {
        cout<<e<<endl;
    }
}
```

Switch case

FILE

New - 1

Open - 2

Save - 3

Save As - 4

int / char

↓

```
switch (expr)
{
    case 1: ____
            break;
    case 2: ____
            break;
    :
    default: ____
}
```

Only int or char can be at place of expression.

break is good to have otherwise next case to it automatically be executed unless break nhi mila tb tk.

also default hona achi bat hai ki kuch nhi to kha jaye.

And isko use kar skte ho aap jaise menu driven progrm bnana ho tb ke liye.

Switch case

```
int main()
{
    int day;
    cout<<"Enter day no";
    cin>>day;
    switch(day)
    {
        case 1: cout<<"Mon";
                break;
        case 2: cout<<"Tue";
                break;
                .
                .
                .
        default: cout<<"Invalid day";
    }
}
```

Switch case example is this.

One more thing that in genral that you may can use nested switch statement like ki ek switch mai hi dusra switch and so on.

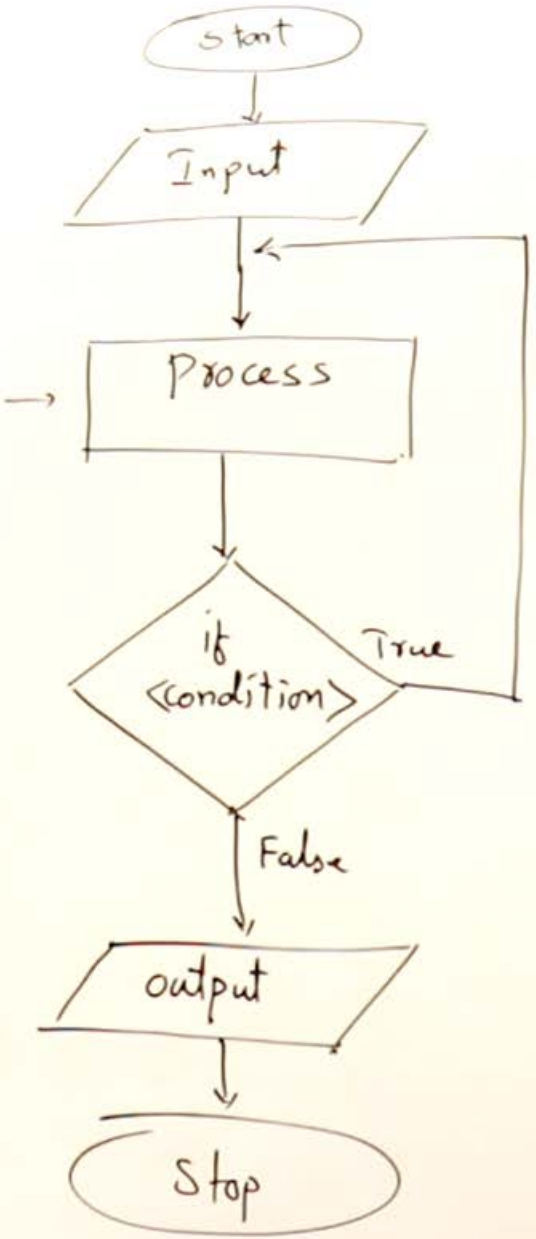
Switch

- Switch is a branch and control statement
- Switch can have 0 or more cases
- Each case is defined with a label
- Depending on the value of expression in switch corresponding case black is executed
- Is a case block is not available then default block is executed
- Default block is optional
- Every case block must terminate with break
- If breaks are not mentioned then cases will fall thru
- Switch is used for menu-driven programs

Loops / Repeating Statements / Iterative Statements

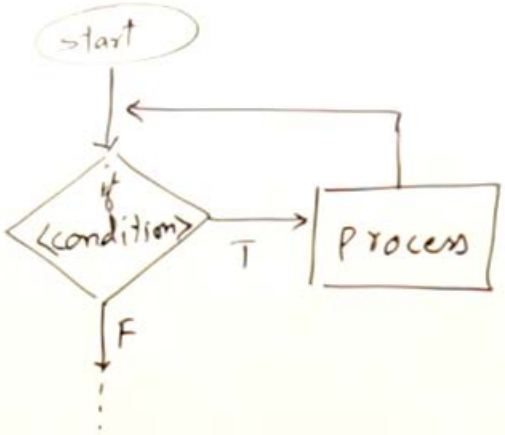
1. while
2. do while
3. for
4. for each

When we repeat the same statement until a condition is true than this is done using loops in programming language.



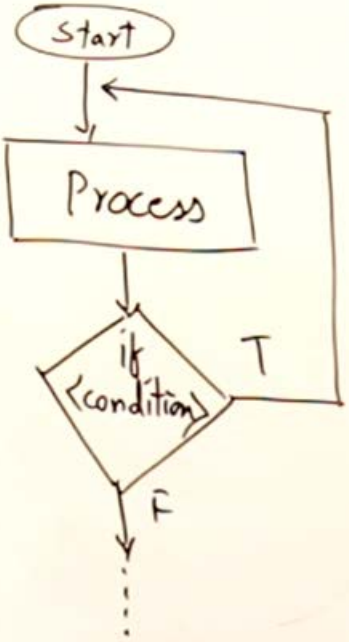
```
while(<condition>)
{
    process;
}
```

Check first and then implement




```
do
{
    process;
}while(<condition>);
```

First implement statement once then Check condition and then implement if true.



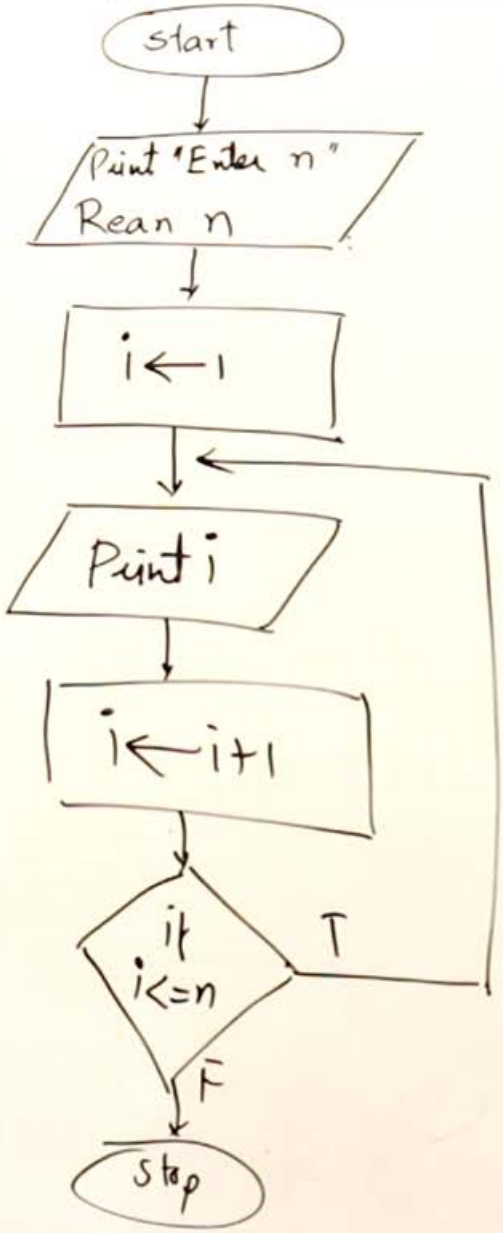
Loops / Repeating Statements / Iterative Statements

Example of do while and same is for while loop.

```
int main( )
{
   int n, i=1;
   cout<<"Enter n";
   cin>>n;
   do
   {
      cout<<i<<endl;
      i++;
   }while(i<=n);
   ---
}
```

n=10, i=1

Print i	i←i+1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10
10	11

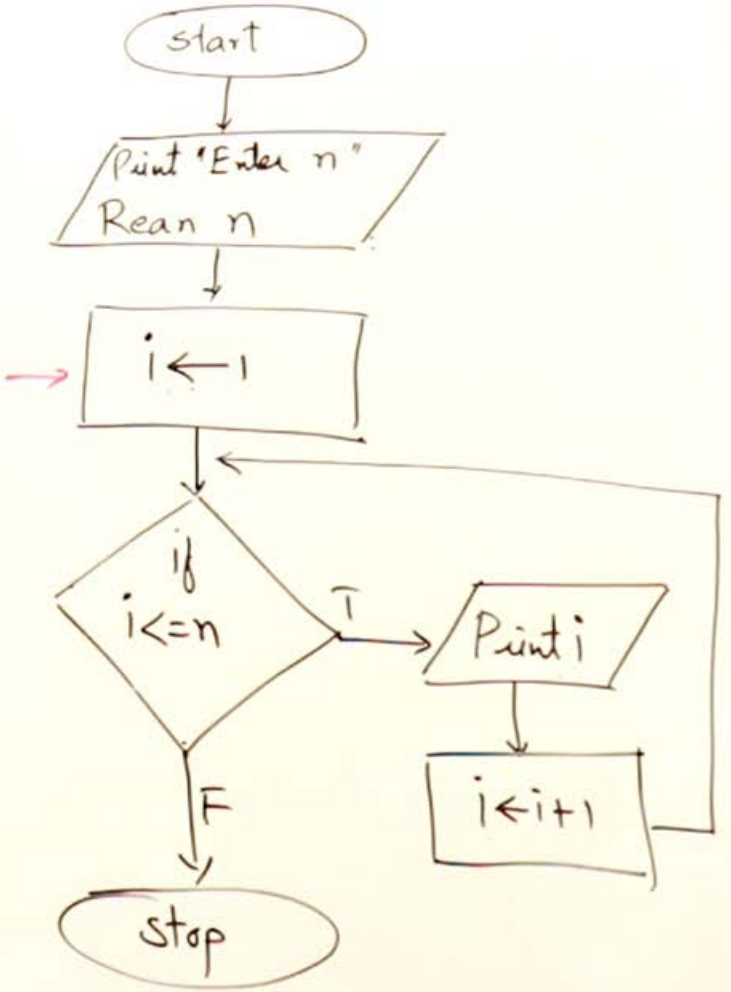


Loops

- Loops are iterative statements
- Block of statements are repeatedly executed as long as condition is true
- Condition given in loop must become false after some finite iterations otherwise its a infinite loop
- Values used in condition must update inside the body of finite loop
- Four types of loops
-
- pre-tested loop while()
- post-tested loop do..while()
- counter controlled loop for()
- for each loop for Collections for()

Loops / Repeating Statements / Iterative Statements

```
for(initialization; condition; updation)
{
    procesr. - (3) (6) (9) (12)
}

int main( )
{
    int n, i;
    cout<<"Enter n";
    cin>>n;
    for(i=1; i<=n; i++)
    {
        cout<<i<<endl;
    }
}
```



Counter controlled loop that is based on counter and actually more user friendly loop as compared to while or do while and commonly and widely used in genral.

Program to Test all Loops

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

int main()
{
    int a=0;
    while(a<10)
    {
        cout<<"a "<<a;
        a++;
    }

    int b=0;
    do
    {
        cout<<"b "<<b;
        b++;
    }while(b<10)

    ;for(int i=0;i<10;i++)
    {
        cout<<"i "<<i;
    }

    return 0;
}
```

Program for Infinite Loop

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

int main()
{
    int i=0;
    ;for(;;)
    {
        cout<<" Hello";
        i++;
    }
}
```

Perfect Number

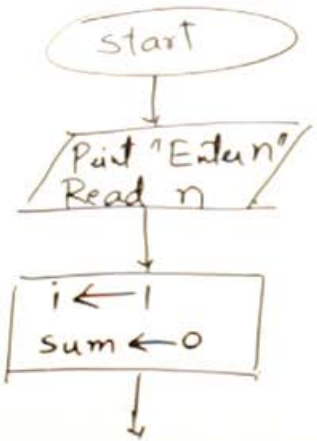
n=6

1+2+3+6=12

2*6=12

n=8 2*8=16

sum=15 15



i	if(n%i==0)	sum=sum+i
1	8%1==0	0+1=1
2	8%2==0	1+2=3
3	8%3==2	
4	8%4==0	3+4=7
5	8%5==3	
6	8%6==2	
7	8%7==	
8	8%8==0	7+8=15

;i++)

+i;

«sum;

Let's say the sum of factors of a number is S.
And if sum of all factors of a number is equal to twice the number that is S=2*number then this is called perfect number.
Example is 6 is a perfect number while 8 is not a perfect number.

Arrays

- Array is a collection of similar data elements under one name, each element is accessible using its index
- Memory for array is allocated contagiously
- For-each loop is used for accessing array
- N-dimension arrays are supported by C++
- Two-Dimensional Arrays are sued for Matrices
- Array can be created in Stack or Heap Section of memory

Arrays

scalar $x = 5$

vector/List $A = (5, 8, 3, 9, 7, 4, 8, 6, 3, 2)$

0 1 2 3 4 5 6 7 8 9

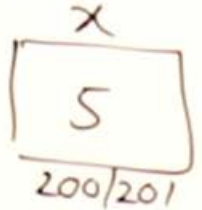
$A_0\ A_1\ A_2\ A_3\ A_4$ - -

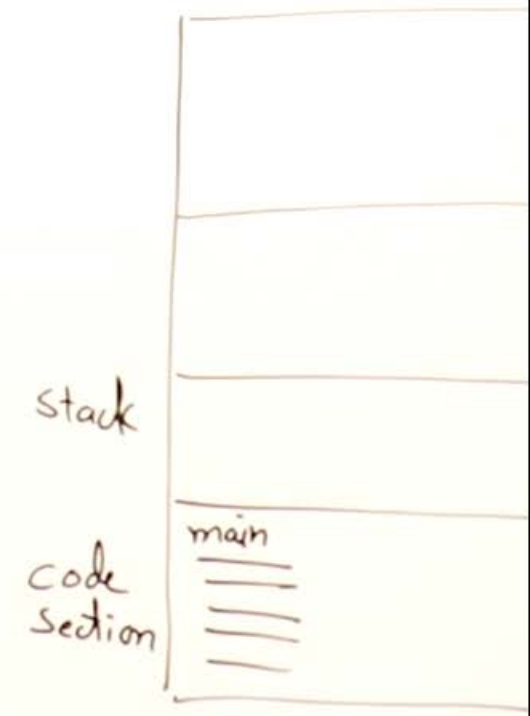
```
cout<<A[3];
cout<<A[8];
cout<<A;
```

x_1, x_2, x_3, x_4

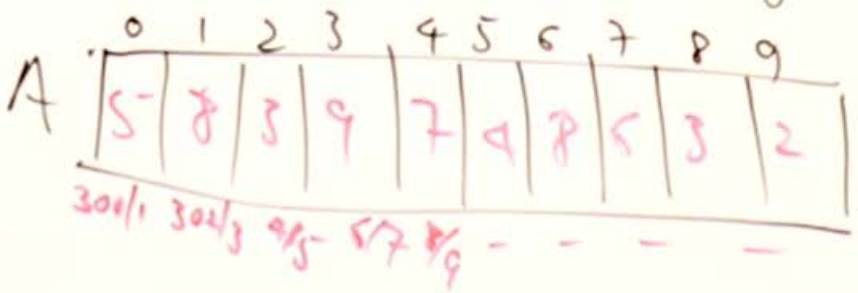
This does not print whole array.

```
int x=5;
```

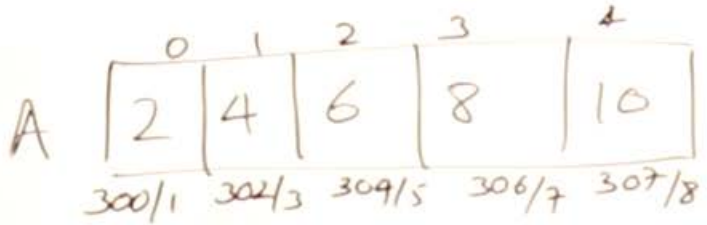





```
int A[10]={5,8,3,9,7,4,8,5,3,2};
```

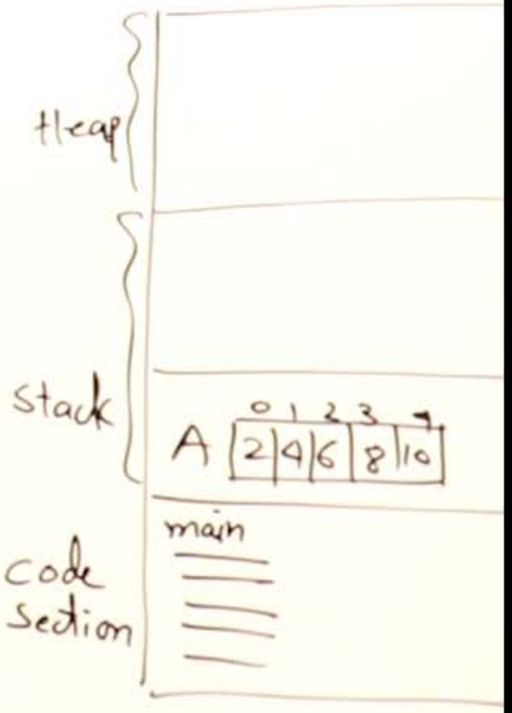


Arrays



i	A[i]
0	A[0] = 2
1	A[1] = 4
2	A[2] = 6
	1
1	1
1	

```
int main()
{
    int A[5]={2,4,6,8,10};

    for(int i=0; i<5 ;i++)
    {
        cout<<A[i]<<endl;
    }
```



Arrays

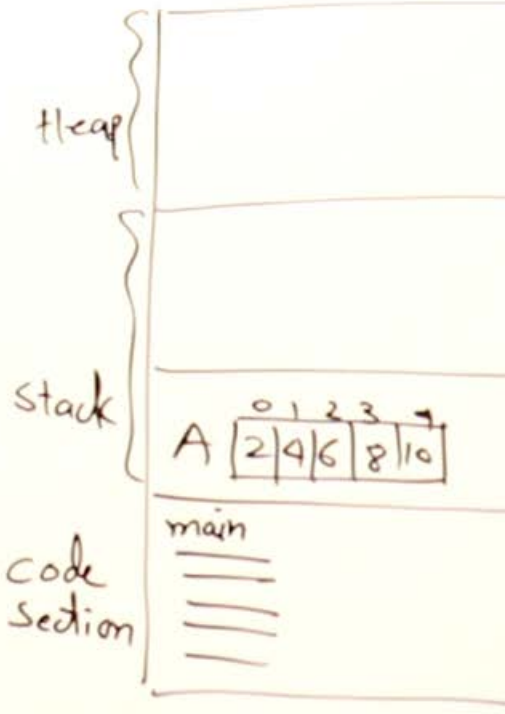
```
int A[5]={2,4,6,8,10};
float B[5]={1.1,2.4,3.7,6.2,9.5};
char C[5]={'A','B','C','D','E'};
```

```
int A[5]={2,4};
```

	0	1	2	3	4
A	2	4	0	0	0

```
int B[ ]={1,2,4,6}
```

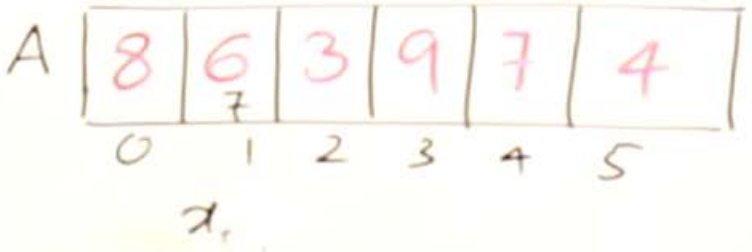
	0	1	2	3
B	1	2	4	6



For each Loop

It works upon collection of object either array, vector, etc.

n=6



This is read as "for each x in A".

In case you donot lnow about data type of collection A us auto keyword it will automatically detect data type in case you know data type then you can mention it as well.

```
for(auto x : A)
{
    cout<<++x;
}
```

Any change on x is not producing change in real values to change real value must right in for each loop as follow

for(auto x : A)

It will change the actual value as well.

```
for(int i=0; i<6; i++)
{
    cout<<A[i];
}
```

Linear Search

```
int main()
{
   int A[10], n=10, i;

   cout<<"Enter numbers";
   for(i=0;i<n;i++)
   {
      cin>>A[i];
   }
   cout<<"Enter key";
   cin>>key;
   for(i=0;i<n;i++)
   {
      if(key==A[i])
      {
         cout<<"found at"<<i;
         return 0;
      }
   }
   cout<<"NotFound";
}
```

n=10

A	6	11	25	8	15	7	12	20	9	14
	0	1	2	3	4	5	6	7	8	9

Key=12 successful
6

Key=35 unsuccessful

Finding an key in an array is called searching

Linear search works for unsorted array.
And works in linear order that it means have time complexity o(n) for worst case and best case it is o(1).

Binary Search

```
int main()
{
    int A[10]={6,8,13,17,20,22,25,28,30,35};
    int l=0, h=9, key, mid;

    cout<<"Enter key";
    cin>>key;
    while( l<=h )
    {
        mid=(l+h)/2;
        if(key==A[mid])
        {
            cout<<"Found at"<<mid;
            return 0;
        }
        else if(key<A[mid]) h=mid-1;
        else l=mid+1;
    }
    cout<<"Not Found";
```

n=10 l=0 h=9

	0	1	2	3	4	5	6	7	8	9
A	6	8	13	17	20	22	25	28	30	35

h l

Time = O(log n)

Key = 25
27

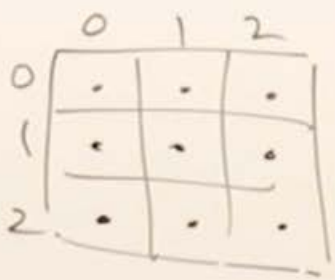
l	h	mid = ⌊(l+h)/2⌋
0	9	(0+9)/2 = 4
5	9	(5+9)/2 = 7
5	6	(5+6)/2 = 5
6	6	(6+6)/2 = 6

Nested for Loops

It helps to generate the dimensions of 2-Dimensional array i acts as row while j acts as column.

i	j
0	0
0	1
0	2
0	3 X
1	0
1	1
1	2
1	3 X
2	0
2	1
2	2
2	3 X
3	

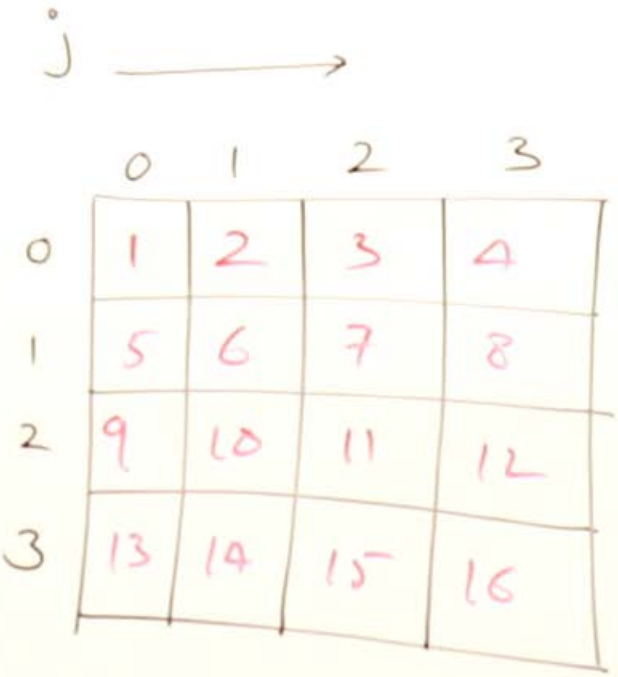
X




```
for(int i=0; i<3; i++)
{
    for(int j=0; j<3; j++)
    {
        cout<<i<<j<<endl;
    }
}
```

Patterns

```
count=1;
for(i=0; i<4; i++)
{
    for(j=0; j<4; j++)
    {
        cout<<count<<" ";
    }
    cout<<endl;
}
```

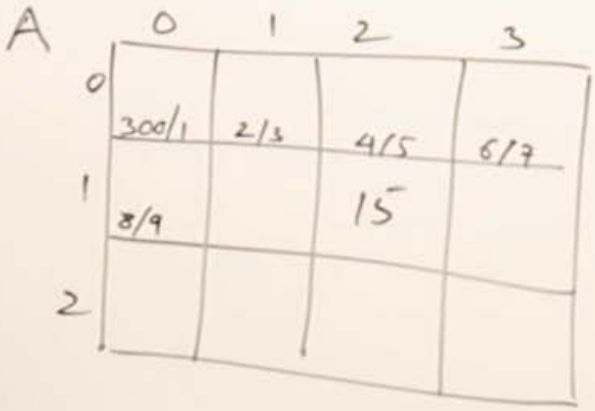


1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

2D Arrays

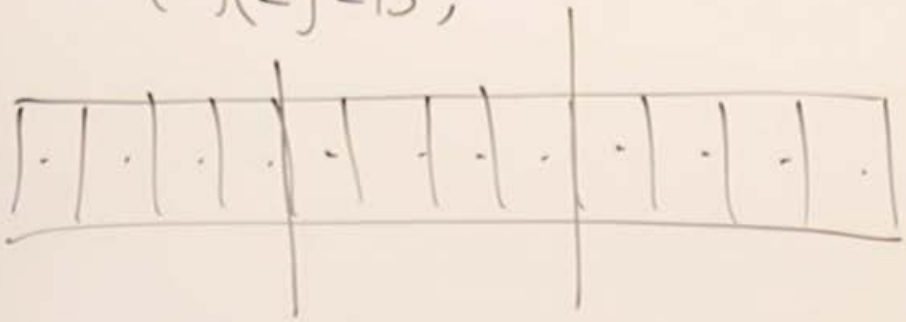
It is a type of array that we need generally at the time when we need to manage data and it have subdata in it.

```
int A[3][4];
```




```
A[1][2]=15;
```

A =



In memory multidimensional array is also stored as 1-D array very first first row as continue block than second and so on. This is shown tight there in picture.

2D Arrays

Initialisation methods

```
int A[2][3]={{2,5,9}, {6,9,15}};
```

A	0	1	2
0	2	5	9
1	6	9	15

```
int A[2][3]={2,5,9,6,9,15};
```

This will fill the array as well in the same manner as shown above.

```
int A[2][3]={2,5};
```

In this first two are filled with the data entered rest will be 0.

Accessing elements of 2-D matrix

```
int main()
{
    int A[2][3]={{2,4,6},{3,6,9}};

    for(int i=0; i<2; i++)
    {
        for(int j=0; j<3; j++)
        {
            cout<<A[i][j];
        }
        cout<<endl;
    }

}
```

Pointers

Pointers are used to allocate memory location to store adress of a variable and using a pointer we can acess the adress where a variable is stored and also view the stored data at that place via derefrencing

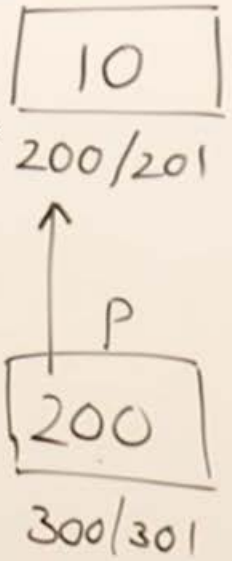
declaration `int *p;`

Initialization `p=&x;`

dereferencing `cout<<*p;`

```
int x=10; //Data variable x
```

⟶ `int *p; //Address variable`



`cout<<x;` — 10

`cout<<&x;` — 200

`cout<<p;` — 200

`cout<<&p;` — 300

⟶ `cout<<*p;` — 10

Pointer

- Pointer is address variable
- It can store the address of data
- Pointer are used for accessing heap memory
- 5 Arithmetic operations are allowed pointer
- p++ - move pointer to next element
- p - - move pointer to previous element
- p+k gives address of kth element form pointer location to right
- p-k gives address of kth element from pointer location to left
- q-p gives number of elements between 2 pointers p and q
- Pointers can be of many levels
- Double pointer is used for accessing 2D arrays

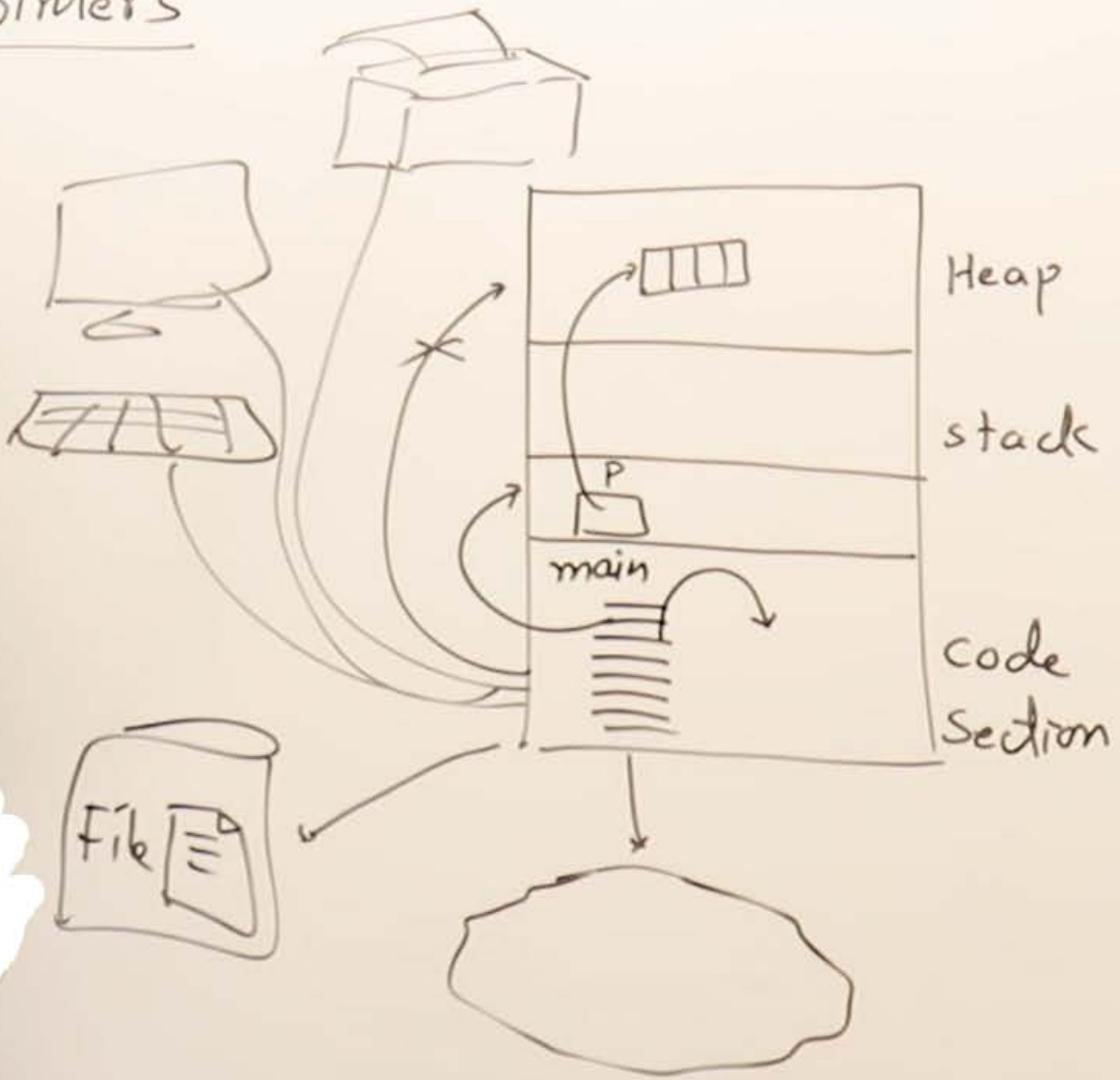
Program to Demonstrate Pointer Syntax

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

int main()
{
    int a=10;
    int *p=&a;

    cout<<a<<endl;
    cout<<&a<<endl;
    cout<<p<<endl;
    cout<<*p<<endl;

    return 0;
}
```

Why Pointers

```
ma inc )
{
}
```

Actually program can access only code section memory and stack memory it cannot access anyhow Heap memory or some other resources like file network data keyboard resource mouse printer and various more. These can not be accessed directly via program. To access these we need to use pointers. For sile access we have file pointers. And more of different types.



Heap

Heap data are always created using "new" kryword in c++ and is not removed unless and untill we delete it using "delete" keyword. As we say of stack stack variable are only available within given domain. But heap must need to be delete otherwise data leak is possible.

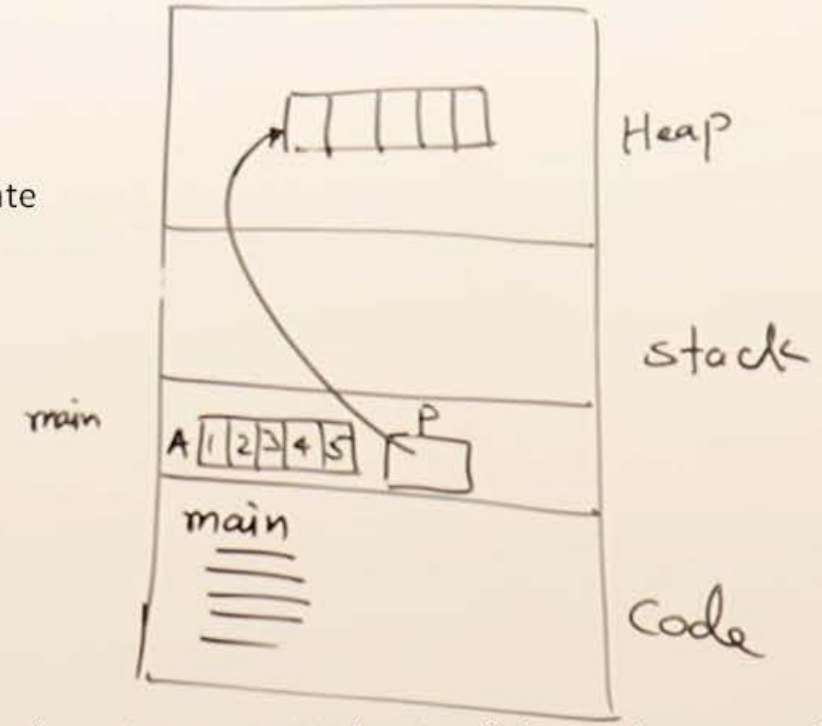
```
main()
{
```

Stack → `int A[5]={1,2,3,4,5};`

Heap →
```
int *p;
p=new int[5];
```

Like this we can create some data in heap.

→ `int *p=new int[5];`




```
:
:
```

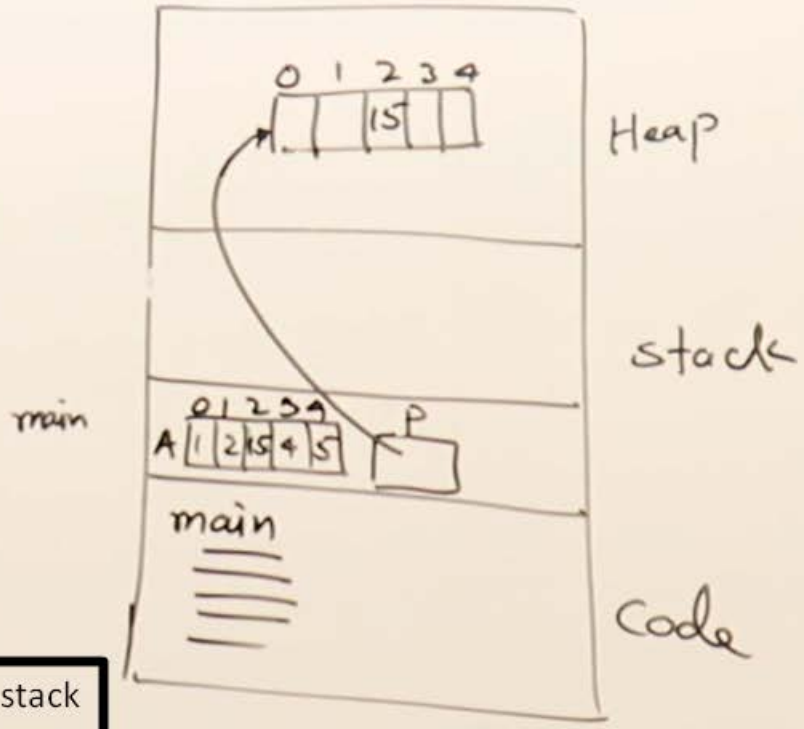
~~*~~`p=NULL;` /* If before deletion we assigned p pointer as NULL then it will loose it's access to heap data and then it won't be deleted */

```
delete []p;
```

`p=NULL;` /* If a pointer points to no where it is then good that we say it is pointing to null */

Heap

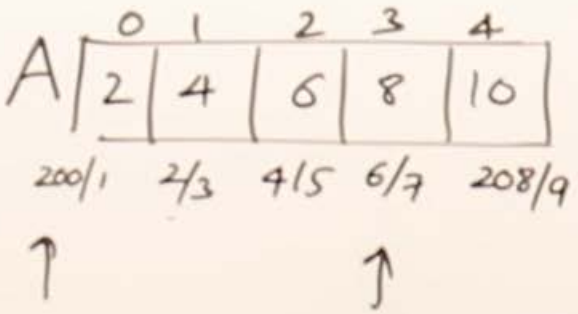
```
main()
{
    int A[5]={1,2,3,4,5};
    int *p;
    p=new int[5];
    A[2]=15;
    P[2]=15;
```



Yes so as you can acess array in stack same way you can access for array p the way of accessing is shown here.

Pointer Arithmetic

```
int A[5]={2,4,6,8,10};
int *p=A; int *q=&A[3]
```



1. `p++;` //Move pointer to next location
2. `p--;` // Move pointer to one location back
3. `p=p+2` //Move pointer to ahead 2 location
4. `p=p-2` //Move pointer to back 2 location
5. `d=p-q;` /*It will say how many elements are lieing in between two given pointers also if p-q is -ve thi means q is ahead of p and vice-versa.*/

-3

$206-200=\frac{6}{2}=3$

$200-206=\frac{-6}{2}=-3$

Problems using Pointers

Pointers are very dangerous if a careless programer uses them it will lead to various problem and all of them are listed here.

1. uninitialized Ptr
2. Memory Leak
3. Dangling pointer

```
int x=10;
int *p;
① p=&x;
② p=(int*)0x5638;
③ p=new int[5];
cout<<*p;
```

There are three way to initialise pointer all are done here 1 and 3 are very common 2 is used rarely and generally used while designing operating system.

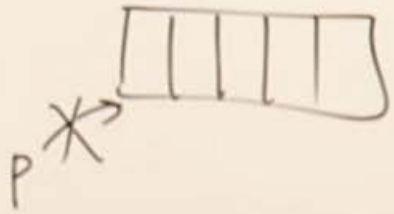
/* In place of this if you donot put [ ] and size it will allocate you memory for one int and give it's address to you */

Next two memory leak and Dangling pointer are in next slide

Problems using Pointers

- uninitialized Ptr
- Memory Leak
- Dangling pointer

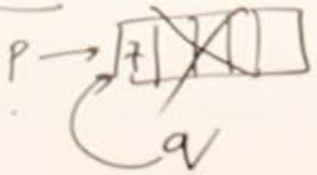
```
int *p= new int[5];
        .
        .
        .
delete [] p;
p=NULL;
p=0;
p=nullptr;
```



You can select any method to free our pointer from previous pointing location but nullptr is suggfested to be good as it move this pointer to a default 0 location untill not used.

So here we must delete allocated memory before delocating our pointer from it otherwise kya hoga ki aab p pointer khi aur NULL pe hai aur memory mai created array hai hi jo ki ek time ke bad pura memory bhar dega aur space hi na bache iske alawa ye bhi hia ki if anyhow that data is accessed to wo bhi missuse ho skta haii.

Problems using Pointers



1. uninitialized Ptr
2. Memory Leak
3. Dangling pointer

In this case what happened is ki memory allocate hui initialise bhi hui par phir delete ho gai aur aab access nhi hai aur delete kaise hui to ye bgl wla example dekho.

```
void main()
{
    int *p=new int[5];
    :
    :
    fun(p);
    cout<<*p;
}
```

```
void fun(int *q)
{
    =
    =
    =
    delete []q;
}
```

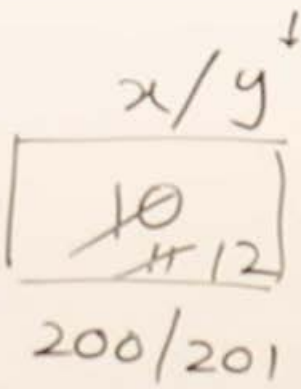
So here p pointer is been passed to function fun now wha q tha reference to dono same memory pe point kar rhe the par aab q ne to kam kar ke sara delete kar diya aur p ke pas kuch bacha nhi hence it is dangling pointer.

it is like the to pta nhi kha kho gya.

Reference

One of the powerful tool of c++ only that allow to have reference so this is actually giving a nick name or creating an alias for a variable that already exist. It is must that with creation of alias you must initialise it the same time.

```
main( )
{
    int x=10;
    int &y = x; /* So here l-value location value is assigned to y. And this made the y an alias of x. */
    x++;
    y++;
    cout<<x; ---- 12
    cout<<y; ---- 12
```



So here in this program no new y variable is created. Both of them are on same point and data at this location can be accessed using any of the one x or y.

One more thing if y became alias for one variable then it is final it cannot be done ki aab &y=a kar diya. There is no such rule in reference. Also reference consume no memory at all.

Reference

- Reference is a Alias of variable
- It must be initialised when declared
- It doesn't take any memory
- It cannot be modified to refer other variable
- Syntax for reference declaration is
- Int &y=x;

Pointer to a function

```
void display()
{
    cout<<"Hello";
}

int main()
{
    void (*fp)();      // decl
    fp=display;        // init
    (*fp)();           // call
}
```

So in short ham ise direct bhi kar skte hai ki sidhe likha display(); main mai aur print ho gya par what if we want to use it through pointer.

So for that we have following prototype ki phle same order mai pointer decleare as function ka signature likhte ho if any parameter is there mention that too in bracket And then intialise it to function and finally call it as done here.

/* This means ki hm ek function type ka pointer bna rhe hai par haa yha ( ) mai function ke pointer ko likhna must hai.*/

// This will point fp pointer to dispay ka address.

// This will give a call to the function.

Pointer to a function

```
int max(int x,int y)
{
   return x>y?x:y;
}
```

```
int min(int x,int y)
{
   return x<y?x:y;
}
```

```
int main()
{
    int (*fp)(int,int);
    fp=max;
    (*fp)(10,5);
    fp=min;
    (*fp)(10,5);
}
```

max is called

min is called

So here we have two functions and these are max and min we actually created one pointer and now we used it in two ways but phle haa btana pra ki yha point karo wha karo.

Actually jab kabhi bhi hm polymorphism ko implement karte hai to isi tarike se hota hai ki agr same number of argumnts hai to kha tumhe konse wle function ko call karna hai.

So this concept is actually used there but ye hme implement karna nhi hota wo apne aap hota hai here hme idea mila par ki kaise hota hai.

Functions

return_Type Function_name(parameter List)

{ o/p

atmost 1 value

void

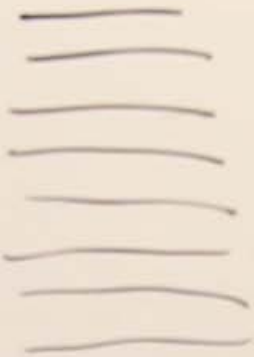
i/p

0 or more

}

void main()

{



}

So actually monolithic programming ke approach mai ye tha ki we were only using same section main aur usi mai sab chawal daal sabji ek sath to aab new method ki chote chote hisse mai likho aur use call kar lo function mai.

For that function is useful and above is complete sugnature of a function decleration to statement of the function.

Functions

return_Type Function_name(parameter List)

```
void display( )
{
   cout<<" Hello";
}
```

```
void main( )
{
1 ____
2 display( );
3 ____
4 ____
}
```

This is how it is called.

But one more thing not necessary but acha hai ki you donot use cin and cout that is interactions with user through function all interaction must be done through main function and only operation is advised to be done through function If used then called bad function.

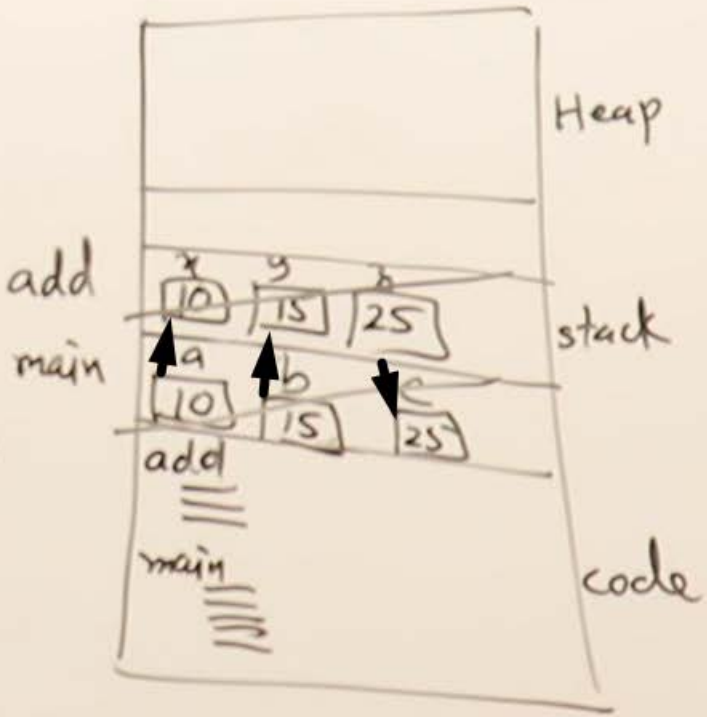
It is same as ki main is owner of the company now all important iputs and output from user client he should take not one of his employee that is a function in this case.

So in memory very first main ke liye jgh allocate hui then jab command mai add ka call aaya to uske variable allocate hue aur jb job done then memory released. After returning or performing the task.

Functions

return_Type Function_name(parameter List)

```
int add(int x, int y)
{
    int z;
    z=x+y;
    return z;
}

void main()
{
    int a=10,b=15,c;
    c=add(a,b);
    cout<<"sum is "<<c;
}
```



Note: Heap memory is not dellocated itself as it is done for stack for heap must delete it using delete operator.

Functions

- Function is a module which performs a specific task
- Functions are called by name
- Rules for giving function name is same as variable name
- Function can take 0 or more parameters
- Function can return single value
- Void function don't return any value
- Default return type is int

Example of Simple Function

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

void display()
{
    cout<<"Hello";
}

int main()
{
    display();
    return 0;
}
```

Example of Function with Arguments

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

float add(float x,float y)
{
    float z;
    z=x+y;
    return z;
}

int main()
{
    float x=2.3,y=7.9,z;
    z=add(x,y);
    cout<<z<<endl;
    return 0;
}
```

Example of function to find Maximum of 3 number

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

int maxim(int a,int b,int c)
{
    if(a>b && a>c)
    return a;

    else if(b>c)
    return b;

    else
    return c;
}

int main()
{
    int a,b,c,d;
    cout<<"Enter three no.s ";
    cin>>a>>b>>c;

    d=maxim(a,b,c);
    cout<<"Maximum is "<<d<<endl;

    return 0;
}
```

Function Overloading

```
int add(int x,int y)
{
    return x+y;
}
int add(int x,int y,int z)
{
    return x+y+z;
}
float add(float x,float y)
{
   return x+y;
}
void main()
{
    int a=10,b=5,c,d;
    c=add(a,b);
    d=add(a,b,c);
    int i=2.5f,j=3.5f,k;
    k=add(i,j);
}
```

Same function name can be used for multiple purpose they can be diffrentiated on the basics of either number of arguments or type of arguments. **(Data type)**

This is very uiseful as it allows the programmer to donot remeber multiple names for same work;
He can use same add name for float type as well as int type or double as well.

Function Overloading

int max(int, int)

float max(float, float)

int max(int, int, int)

X float max(int, int)

These are list of some of the valid as well as invalid function definition remeber return type is never taken into consideration while implementing function overloading.

Function Overloading

- If More than one functions can have same name, but different parameter list, then they are overloaded functions
- Return the is not considered in overloading
- Function overloading is used for achieving compile time polymorphism

Program to Demonstrate Function Overloading using Sum function

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

int sum(int a,int b)
{
    return a+b;
}

float sum(float a,float b)
{
    return a+b;
}

int sum(int a,int b,int c)
{
    return a,b,c;
}

int main()
{
    cout<<sum(10,5)<<endl;
    cout<<sum(12.5f,3.4f)<<endl;
    cout<<sum(10,20,3)<<endl;

    return 0;
}
```

Function Template

I must say awesome concept Like one way to hai right side ka that is ki multiple program likh ke kiya hai to apart from this we have left side mai ek hi templete function bna liya aur usi ko use kar skte hai different data types ke liye.

```
template <class T>
T   max(T x, T y)
{
    if(x>y)
        return x;
    else
        return y;
}

main()
{
    int c=max(10,5);
    float d=max(10.5f, 6.9f);
}
```

```
int max(int x, int y)
{
    if(x>y)
        return x;
    else
        return y;
}

float max(float x, float y)
{
    if(x>y)
        return x;
    else
        return y;
}
```

Function Template

- Function template are used for defining generic functions
- They work for multiple datatypes
- Datatype is decided based on the type of value passed
- Datatype is a template variable
- Function can have multiple template variables

Example of Function Template

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

template<class T>
T maxim(T a,T b)
{
    return a>b?a:b;
}

int main()
{
    cout<<maxim(12,14)<<endl;
    cout<<maxim(2.3,1.4)<<endl;
    cout<<maxim(2.3f,5.6f)<<endl;

    return 0;

}
```

Default Arguments

```
int add(int x, int y, int z=0)
{
    return x+y+z;
}
int fun(int a, int b, int c, int d)
main()
{
    int c=add(2,5);
    c=add(2,5,8);
    c=add(2,5,0);
}
```

optional

```
int add(int x, int y)
{
    return x+y;
}
int add(int x, int y, int z)
{
    return x+y+z;
}
```

So here in left side we have two function so we need to define two function sepreately though they bith do the same job.

So c++ provided solution to that with the concept of default arguments it allows us to assign some default value to a variable in case it is not passed while function call.

One more thing is this that we must assign must default value from right most variable to left. esa nhi ki phle ko default kar do aur baki sab aage wle ko chor diya you must follow order ki sabse right most se kiya and then move.

Default Arguments

- Parameters of a function can have default values
- If a parameter is default then , passing its value is options
- Function with default argument can be called with variable number of argument
- Default values to parameters must be given from right side parameter
- Default arguments are much useful in constructors
- Default arguments are useful for defining overloaded functions

Example of Default Arguments

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

int sum(int a,int b,int c=0)
{
    return a+b+c;
}

int main()
{
    cout<<sum(10,20,3)<<endl;

    return 0;
}
```

Parameter Passing Methods

In c++ we have 3 methods to pass argument to a function. These are all listed below.
It is same as that where we say call by value; call by address; call by reference. Donot get confused in pass by and call by they are same.

Also one thing c language do not support pass by reference.

1. Pass by value
2. Pass by Address
3. Pass by Reference.

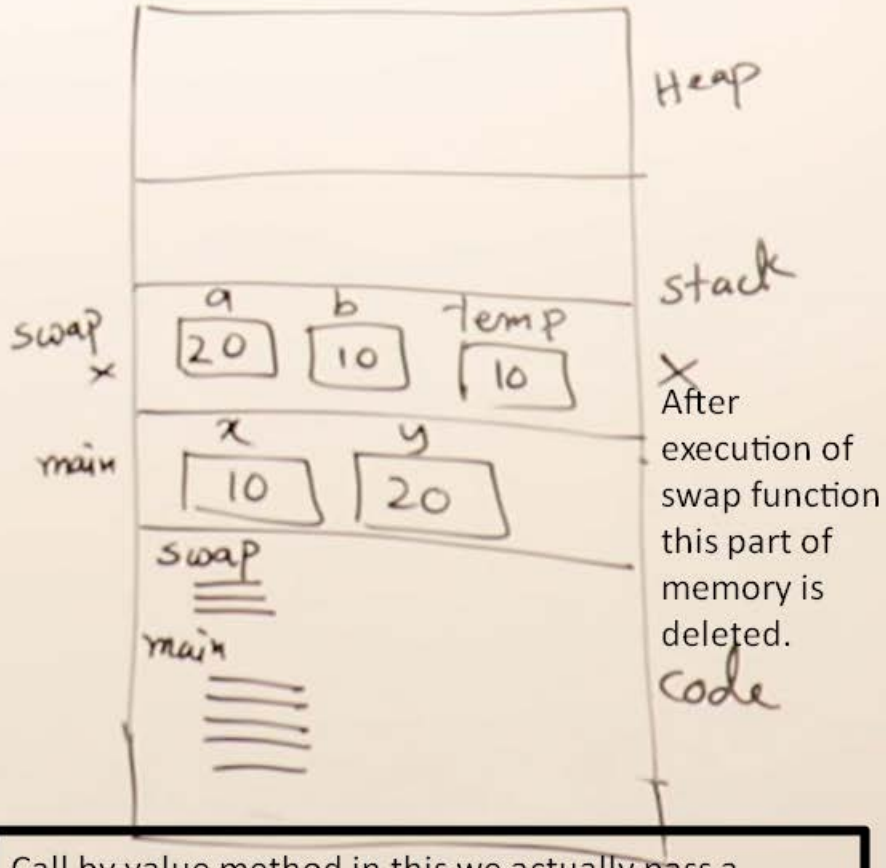
Call by value method example

```
void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

int main()
{
    int x=10, y=20;
    swap(x, y);
    cout<<x<<" "<<y;
}
```

formal

actual

10 20



After execution of swap function this part of memory is deleted.

Call by value method in this we actually pass a value to the function and it is copied in the stack created for function and any change in the passed value by function won't get reflected back in main part. This is used when we generally need only to perform an operation and retuen value don't need to update the value in main part.

Parameter Passing Methods

Three parameter passing methods are supported by C++

Pass-By-Value : values of Actual parameters are passed to formal parameters. Actual parameters cannot be modified by function

Pass-By-Address: Address of Actual Parameters are passed to a function, formal parameters must be pointers. Function can indirectly access actual parameters.

Pass-By-Reference: Actual parameters are passed as reference to formal parameters, function can modify actual parameters.

Program for Call by Value

- Value of actual parameters are copied in formal parameters
- If any changes done to formal parameters in function, they will not modify actual parameters

```
Void swap(int a, int b)
{
        int temp;
        temp=a;
        a=b;
        b=temp;
}

Int main()
{
        int x=10, y=20;

        swap(x,y);
        cout<<x<<y;
}
```

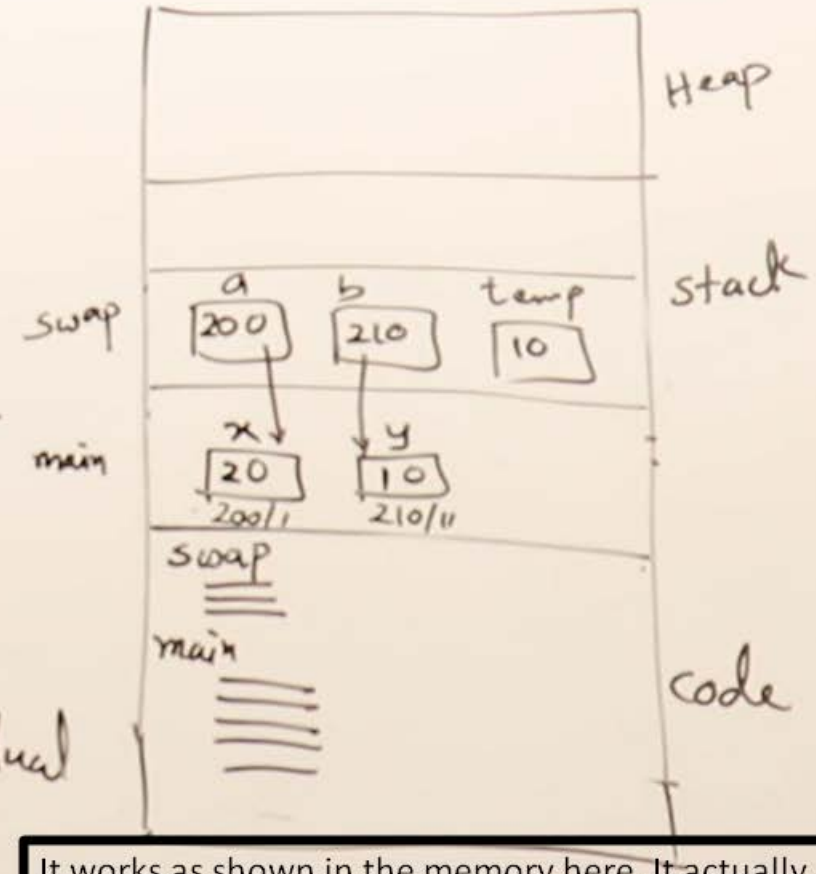
Call by address method

```
void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    int x=10, y=20;
    swap(&x,&y);
    cout<<x<<" "<<y;
}
```

formal

actual

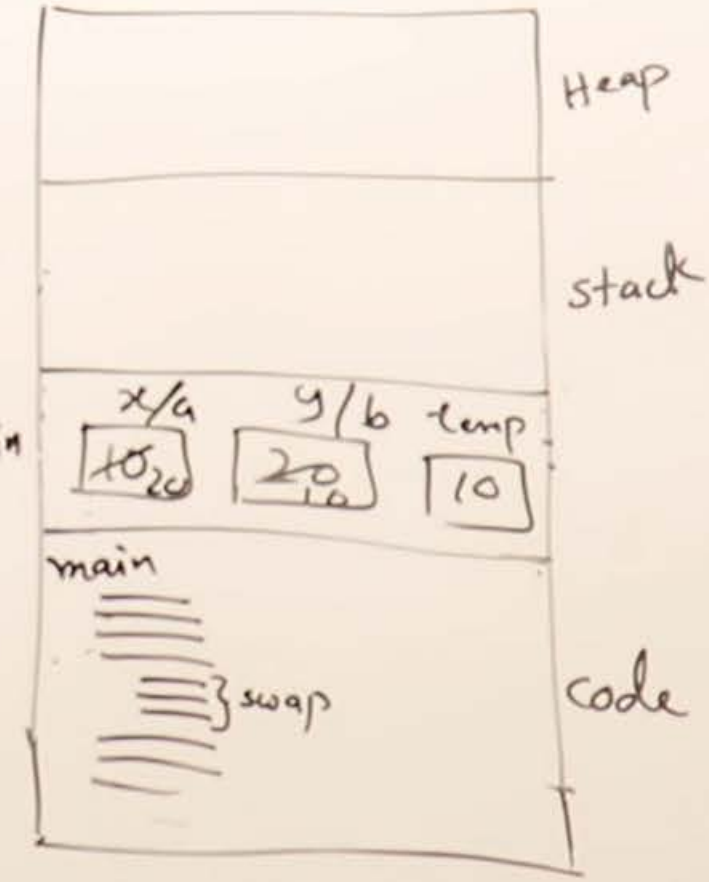
20 10



It works as shown in the memory here. It actually works same as pointer way ki int a =10; int *p=a; now ye to hai ki ek function dusre ke data ko access nhi kar skta par haa hm to adress pass kar rhe hai aur usko dereference kar ke value modify kar rhe hai so we can do this. And this is what pass by address is.

Call by reference

```
void swap(int&a, int&b)
{                          ↑ formal
    int temp;
    temp = a;
    a = b;
    b = temp;
}

int main()
{
    int x=10, y=20;
    swap( x, y); ← actual
    cout<<x<<" "<<y;
}
```



So here we have 3 points:

1. Yes as you know ek function dusre ke variable ko access nhi kar skta to yha kya hua jo ki alias ke liye same main mai access hi nhi balki temp bhi main ke stack mai hi aaya.

2. It actually donot create any memory for itself in stack it copies the function code on main itself as you can see in memory diagram as well.

3. Ab kya hai ki jha jha bhi swap call hoga main mai wo swap ka code copy kar lega while compiling it is same as inline function.

4. Haa with this we will have a problem ki memory jada consume hogi and also ye program warning signs bhi deta hai agr is trh ke function mai koi complex code generally loop ko agr implement kar rhe ho to.

So in short ye acha bhi hai bura bhi it depends agr chota ek do line ka code function mai to call by reference kar lo nhi to aur bhi hai method as per need wo kar lo implement.

Call by Address

- Address pf actual parameters are passed.
- Formal parameters must be pointers
- Formal parameters ca indirectly access actual parameters.
- Changes done using formal parameters will reflect in actual parameters

```
Void swap(int *x, int *y)
{
        int temp;
        temp=*x;
        *x=*y;
        *y=temp;
}

Int main()
{
        int a=10, b=20;

        swap(&a,&b);
        cout<<a<<b;
}
```

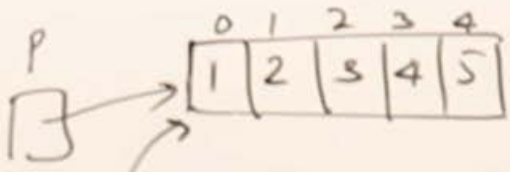
Call by Reference

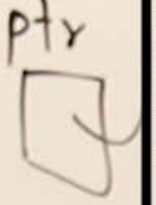
- Actual parameters are passed as reference
- Formal parameters can directly access actual parameters
- Function call is converted into inline function, if not possible it will become call by address
- Reference don't take extra memory
- Syntax is same as Call by Value except, formal parameters are reference

```
Void swap(int &a, int &b)
{
        int temp;
        temp=a;
        a=b;
        b=temp;
}

Int main()
{
        int x=10, y=20;

        swap(x,y);
        cout<<x<<y;
}
```

Return by Address

```
int * fun(int size)
{
    int *p = new int[size];
    for(int i=0; i<size; i++)
        P[i] = i+1;
    return P;
}
main()
{
    int *ptr = fun(5);
}
```





So here we have a function that can also have a return type as address and that is it's return type is of a pointer type.

use of this is ki manlo kabhi heap mai koi data create kiya through function and heap to apne aap delete hoga nhi as it is done in stack ke case mai ki function gyab memory gyab.

To now agar main mai hmne acess return kiya jo us heap ka address to uska data wha se accessible hoga.

Return by Address

- A function can return address of memory
- It should not return address of local variables, which will be disposed after function ends
- It can return address of memory allocated in heap

```
Int * fun(int n)
{
        int *p=new int[n];
        for(int i=0;i<n;i++)
                p[i]=i+1;
        return p;
}
Int main()
{
        int *ptr=fun(5);

        for(int i=0;i<5;i++)
              cout << ptr [ i ] ;
}
```

Return by Reference

```
int & fun(int &a)
{
    cout<<a;   — 10
    return a;
}

main()
{
    int x=10;
    fun(x)=25;
    cout<<x;   — 25
}
```

x/a
10/25

The amazing concept ki aab tk to tha ki yar r-value side mai hi function call hota hai but yha phli bar l-value mai function call hua aab ye kar kya rha hai ki function hi reference bn gya hai x to in short fun(x)=x mtlb fun(x) alias ho gya hai aab x ka.

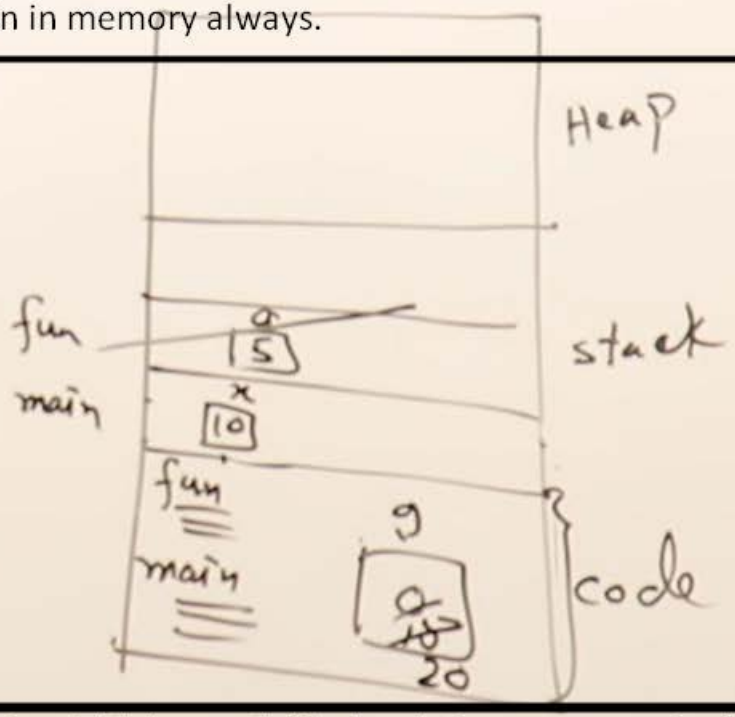
// This will store 25 in x also will execute fun(x)

// This will print 25

Global vs Local Variables

```
int g=0;        15+5=20
                +1
void fun( )
{
    int a=5;
    g=g+a;
    cout<<g;  - 20
}
```

// Global variable -> this is created in code section itself and can be acessed from each and every part of program provided that function donot defined the same variable as a local variable. This remain in memory always.



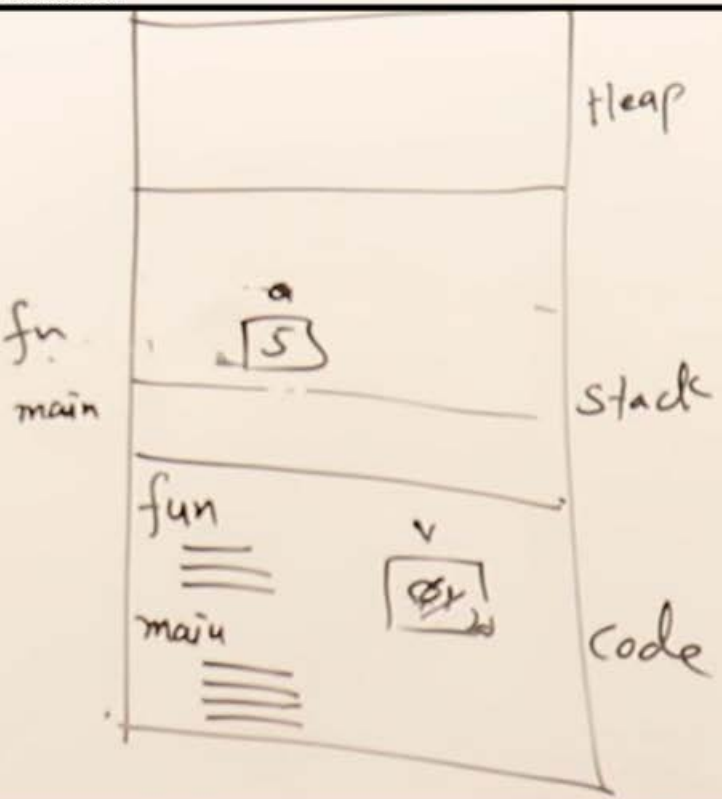

```
void main()
{   int x=10;
    g=15;
    fun();
    g++;
    cout<<g;  - 21
}
```

Local variable -> This is a variable that is been accessed only within the function domian in which it is defined and also this variable gets out of memory when the execution of that specific function is over. As you can see for int a of function fun after completion of execution it will go out of memory.

Static Variables

It is same as global variable ki ye memory mai ek bar create hoga aur jab tk program hai tb tk wo whi rhega. Bhle hi wo kisi particular function ka part hai par still wo code section ke memory mai hi apne apko create karega if static keyword use kiya ghy hai use declare karte wkt to.

```
void fun( )
{
    static int v=0;

    int a=5;
    v++;
    cout<<a<<" "<<v;
}

main()
{
    fun(); —— 5 1
    fun(); —— 5 2
    fun(); —— 5 3
}
```



Par haa the point is ki bhle hi ye code section mai bna hai global ki trh par access apne defined domain mai hi hoga. Bas ye hai ki bhle hi fun out of stack jaye har bar after execution par ye int v bna rhega throughout program run and keep stored data with it like ek bar update kar ke 1 aaya to 1 bna hai abhi bhi.

Static variables

- They have local scope but remain in memory thru out the execution of program
- They are created in code section
- They are history-sensitive

```
Void fun()
{
        static int v=0;

        int a=10;
        v++:
        cout<<a<<" "<<v;
}

Int main()
{
        fun();
        fun();
        fun();
}
```

Introduction to OOPS

Modular Programming

Bank

openAcc( )
deposit( )
withdraw( )
checkbal( )
applyloan( )
.
.

This is modular programming wla approach ki bank wlo ke liye bna rhe hai program aur phir har kam ke liye alg alg function ho then in that case hai ye par acha way nhi hi hai ki sare data ko ek jgh rkh ske ye isme nhi hai possible.

Ki ek object ka hai to whi pe rhega sab store.

Object-oriented programming

Govt

Electric
newconn( )
close( )
Billpay( )

Water
.
.

Education
.
.

Transport
.
.

object

Bank
deposit( )
withdraw( )
.
.

This is new approach ki wk hi class bnaya govt uska sub class bnaya water electricity and other now every consumer is an object of this class.

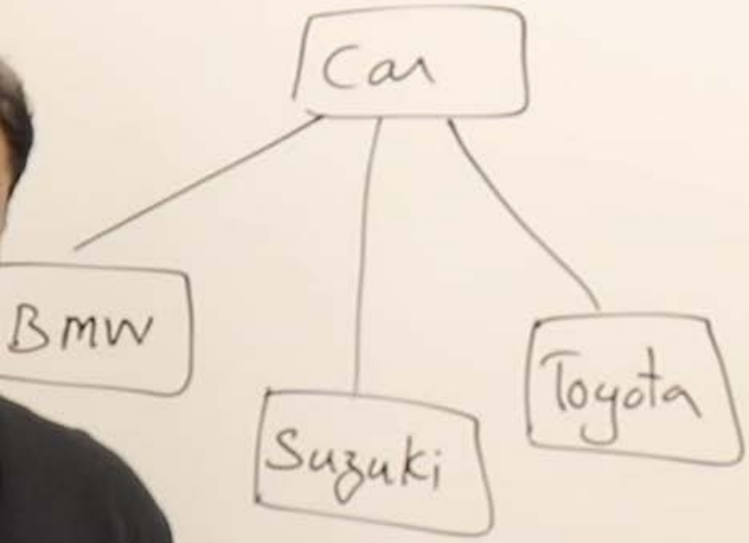
Principles of object-orientation

PART - 1

```
class Car
{
  private:
    1. Data
  public:
    2. functions()
```

1. Abstraction — Internal woking user ko kyo hi btana
2. Encapsulation — Sab ko ek tokre mai rkh do
   - Data Hiding — Chupa ke rkho isliye nhi ki koi bhut bda secret hai bas we want ki miss handling na ho us data ka
3. Inheritance
4. Polymorphism

Principles of object-oriented



1. Abstraction
2. Encapsulation
   - Data Hiding
3. Inheritance
4. Polymorphism

This inheritance is ki ek ki property utha lo bar bar like aab car kisine bnaya usne uski property kuch de di like seat drive accelerate and bla bla aab kya hai ki hundai bnaye ya tyota sab ye base property to legi hi to wo base property inheritance se aai baki hyundai apni kuch extra dalna chahe dal le.

Like base class ka function hai drive par hm chahte hai ki overwrite kare aur aab multiple way hai ki like agr nitro button bhi dekhega on ke time pe input mai bmw wla
suzuki saftey button check karna chahta hai to they can do it.

DATA IS PROPERTY AND FUNCTION IS BEHAVIOUR IN A CLASS

Class is like a blue print like it is a classification for example building hmne kha ki ese ese bnana hai aab wo ready kr ke diya aab real mai jo bhi building bn ke aaya us blueprint ke according that is object to that class.

So this is actual software example of class. This rectangle is complete blueprint and below that in main function we defined objects of class rectangle that actaully have all the property of rectangle.

Class vs Object

you

my

mvv x1

amri

```
class Rectangle
{
    float length;
    float breadth;

    float area()
    float perimeter()
    float diagonal()
    :
    :
};
main()
{
    Rectangle r1, r2, r3;
}
```

objects

Object-Oriented Programming

Features of OOPS

- Abstraction
- Data Hiding
- Inheritance
- Polymorphism

Classes

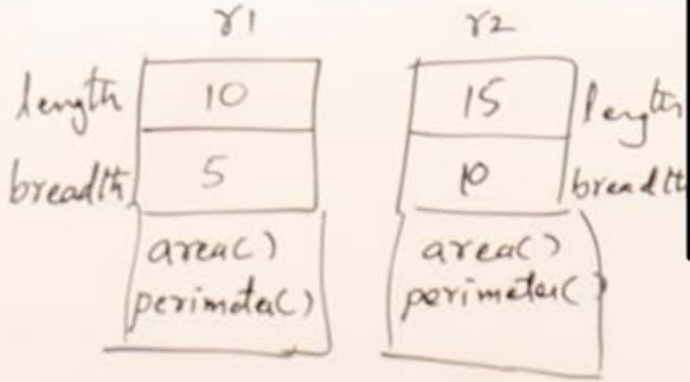
- Class is a blue print of an object
- Class in a group of objects
- Class is a design of object
-
- Many object can be created from same class
- Object consumes memory equal to sum of sizes of all data members
- Member functions don't occupy memory
- Member functions are called depending on object
- . Dot operator is used for accessing members of object
- Memory allocated for object is also called as instance

writing a class

```
class Rectangle
{   public:
    int length;  — 2
    int breadth; — 2
                   4 byte

    int area( )
    {
        return length * breadth;
    }
    int perimeter()
    {
        return 2*(length+breadth);
    }
};
```

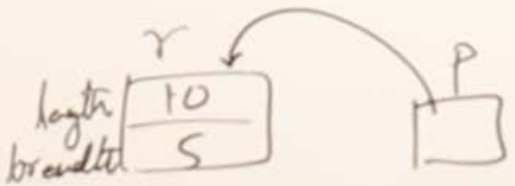
By default every thing is private in a class _you can not acess any thing in this if it is private so make it public to acess in your program.

In memory it will create one 4 byte memory space when a object of user defined data type rectangle is created.




```
void main()
{
    Rectangle r1, r2;

    r1.length=10;
    r1.breadth=5;
→   cout<<r1.area(); — 50
    r2.length=15; r2.breadth=10;
→   cout<<r2.area(); — 150
```

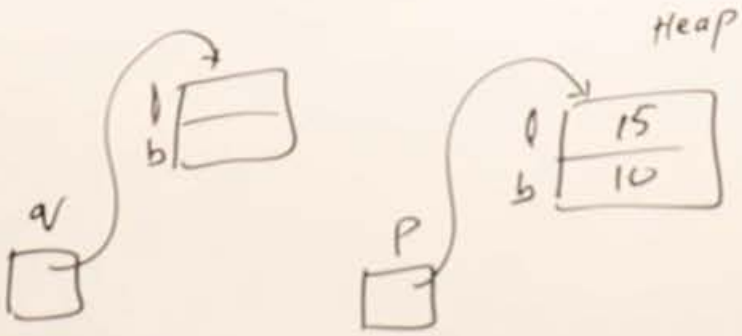
Pointer to Object




```
int main()
{
    Rectangle r;
    Rectangle *p;
    p=&r;
    r.length=10; //don't execute comment hai ye
    p->length=10; // So instead of writting *p we can directly derefernce here using arrow.
    p->breadth=5;
    cout<<p->area();
```

```
class Rectangle
{
    public:
        int length;
        int breadth;

        int area()
        {
            return length*breadth;
        }
        int perimeter()
        {
            return 2*(length+breadth);
        }
};
```

Pointer to Object




```
class Rectangle
{
    public:
        int length;
        int breadth;

        int area()
        {
            return length * breadth;
        }
        int perimeter()
        {
            return 2*(length + breadth);
        }
};
```

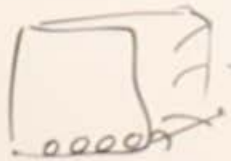
Two mthd to create object

```
Rectangle r;        // Will create in stack
Rectangle *p=new Rectangle();
```

Will create in heap

```
void main()
{
    Rectangle *p;
    P=new Rectangle;
    Rectangle *q=new Rectangle();
    {
        P->length=15;
        P->breadth=10;
        cout<<P->area();
    };
```

Data Hiding



Telivision ka example ki wha bas wire hoti jo ki data hua to misshandle ho skta tha so sidhe button do jo ki function kare in par.

```
void main()
{
    Rectangle r;
    r.length=10;
    r.breadth=-5;
    cout<<r.area(); ---- -50
```

```
class Rectangle
{
  public:
    int length;
    int breadth;

    int area()
    {
      return length*breadth;
    }
    int perimeter()
    {
      return 2*(length+breadth);
    }
};
```

Data hiding ka mtlb hi hai ki misshandling se bchana hai so for that ye nhi ki chhupa lo koi security threat bilkul nhi. exampple se dekho like yha pe hmne breadth ko -5 kar diya ab kya hai ki we know length and breadth can not be negative. So now isse bchne ka hai ki length aur breadth ko agr set kiya hota particular function ya constructor ke madad se jha kuch condition hm bta skte the to in that case wo misshandle nhi hota. This is the philosphy of data hiding.

Data Hiding

r

length | garbage
breadth | garbage

```
void main()
{
    Rectangle r;
```

✗ r.length = 10;

✗ r.breadth = -5;

✗ cout << r.length;

cout << r.area(); : garbage

```
class Rectangle
{
  private:
    int length;
    int breadth;
  public:
    int area()
    {
      return length * breadth;
    }
    int perimeter()
    {
      return 2*(length + breadth);
    }
};
```

Now private kiya to acess hi nhi hai length amd breadth ka aur garbage aa rhi hai.
to iska soln is get (display) function and set (intialise) function but the main is through constructor.

Data Hiding

Property function

Accessor – getXXX read data

Mutator setXXX Write data

r

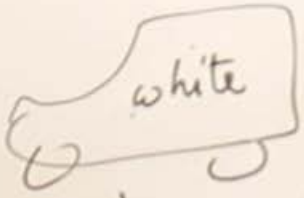
length	10
breadth	-5

```
void main()
{
    Rectangle r;
    r.setLength(10);
    r.setBreadth(-5);
    cout<<r.area();   ——  0
    cout<<"Length is "<<r.getLength();

}
```

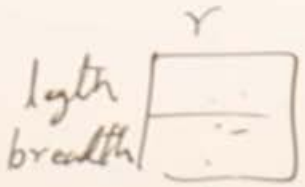
10 x 0

```
class Rectangle
{
  private:
      int length;
      int breadth;
public:
    void setLength(int l)
    {
        if(l>=0)
            length=l;
        else length=0;
    }
    void setBreadth(int b)
    {
        if(b>=0)
            breadth=b;
        else
            breadth=0;
    }
    int getLength()
    {
      return length;
    }
    int getBreadth()
    {
      return breadth;
    }
};
```

Constructors



color – white



setLength()
setBreadth()

```
void main()
{
    Rectangle r(10,5)
    r.setLength(10);
    r.setBreadth(5);
```

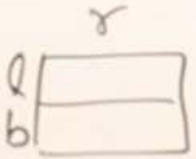
So bat whi hai ki yar manlo koi car laye aab ye to hoga nhi na ki car kaye aur uspe uski property for example colour tuhe ghar lane ke bad set karna ho. tum wha uska behaviour manage karna chahoge na ki property that is basically data member.
Same rectangle wla ki agr dukan se kharida hai rectangle to kya kharidne ke bad uska length aur breadth define karoge nhi na wo to time of buying hi ho jana chahiye so to solve this we have constructor and it is actually used not get or set function.

```
class Rectangle
{
  private:
    int length;
    int breadth;
  public:
    void setLength(int l)
    {
      if(l>=0)
        length=l;
      else length=0;
    }
    void setBreadth(int b)
    {
      if(b>=0)
        breadth=b;
      else
        breadth=0;
    }
    int getLength()
    {
      return length;
    }
    int getBreadth()
    {
      return breadth;
    }
};
```

Constructors

```
class Rectangle
{
    private:
        int length;
        int breadth;
    public:
```

Rectangle r;



1. Default constructor

2. Non-parameterized "

3. Parameterized "

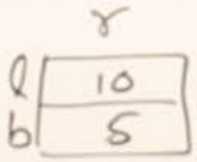
4. Copy constructor

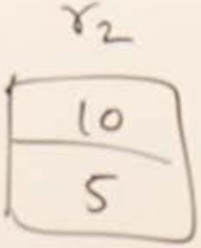
Sometime this is called built in constructor. Aur mtlb jo ki aab constructor nhi hai phir bhi memory to dila rha hai na compiler that is what built in constructor.

Now jab upar wle ko built in kho to niche wle ko default se kh dete hai sabdo se se jada bhwna smjho.

Constructors

```
Rectangle r;
Rectangle r();
Rectangle r(10,5);
Rectangle r2(r);
```





so here now need to ho gya ki constructor chahiye to aab tin trh ka kyo to uska logic ye hai ki aab manlo pen lena hai tumhe to dukan gye ek bar kha bhaiya pen dena usne koi bhi pen diya tumne le liya aur aa gye now dusra case hai ki tumne kha ki bhaiya blue pen ball point mai dena to tumne parameter btaya pen ka yha to wo hua ek aur aab man lo final mai tum ek purana pen tha wo leke gye bole ki bhaiya ye pen tha yhi de do to wo hua copy constructor jaisa.

```
class Rectangle
{
  private:
    int length;
    int breadth;
public:
  Rectangle( )          //Non parametrised constructor.
  {
    length=0;
    breadth=0;
  }
  Rectangle(int l,int b)
  {
    setLength(l);       //Parametrised constructor
    setBreadth(b);
  }
  Rectangle(Rectangle &rect)
  {
    length=rect.length; //copy constructor
    breadth=rect.breadth;
  }
};
```

And here the concept of constructor overloading is also implemented.

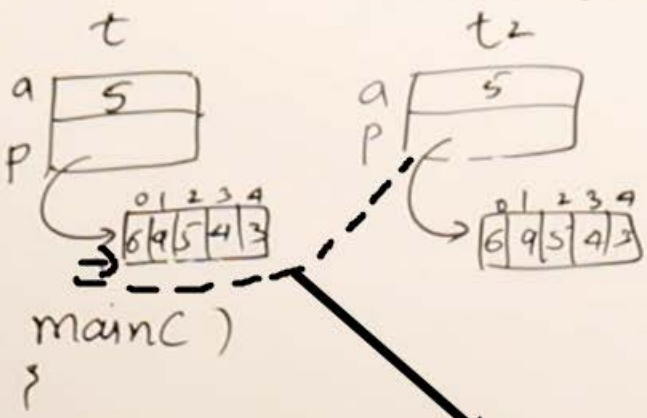
Constructors

```
class Rectangle
{
    private:
        int length;
        int breadth;
    public:
        Rectangle( )
        {
            length=0;
            breadth=0;
        }
        Rectangle (int l=0, int b=0)
        {
            setLength(l);
            setBreadth(b);
        }
        Rectangle (Rectangle &rect)
        {
            length=rect.length;
            breadth=rect.breadth;
        }
};
```

```
Rectayle  r(10,5);
"         r(10);
"         r( );
```

So here it is same function mai default parameter pass and so we don't need the first constructor.

Deep Copy Constructor




```
class Test
{
    int a;
    int *p;

    Test(int x)        // x = 5
    {
        a = x;
        p = new int[a];
    }

    Test(Test &t)
    {
        a = t.a;
        // p = t.p;
        p = new int[a];
    }
};
```

Deep copy constructor mtlb ki copy constructor hi hai par jab dynamic mai memory allocate kiya to hme chahiye to tha do memory slot par hmne ek ko hi copy kiya aur alias bna ke chor diya aab isse jiska alias bnaya wo bhi manupulate ho skta hai aur useful at the end nhi rhega so while programming if you take care of these than i.e; Deeo copy constructor.

So man lo ki hmne copy constructor ko call kiya aur ye aa gya aab problem kya hai ki yar t object khega bhai maine to bas tujhe apna reference diya tha tu to mujpe hi aa gya to is se bchne ka dhyan dena hai to aap cutted line jo likha hai usko na likh ke niche wla likhte ho. To alg se dynamic mai create hoga memory and that is good. So now apne phir dekha ki yar aab kar to diya par value bhi thi t ke array mai to use bhi copy kro for loop lga ke par ese hi sidhe nhi karna.

Destructor

Used to deallocate memory.

Destructor ek hi ho skta hai bas multiple destructor wla concept bilkul nhi hota.

Destructor mai multiple statement execute kar skte ho

```
class Test
{
    int *p;
    ifstream fis;

    Test()
    {
        p=new int[10];
        fis.open("my.txt");
    }

    ~Test()
    {
        delete []p;
        fis.close();
    }
};
```

Destructor in inheritance

```
int main()
{
    Derived d;
    .
    .
    .
}
```

Base const- - -
Derived cont- -
— Derived des- - -
— Base dest- -

```
class Base
{
  public:
    Base()
    {
      cout<<"Base constructor"<<endl;
    }
   ~Base()
    {
      cout<<"Base destructor"<<endl;
    }
};

class Derived :: public Base
{
  public:
    Derived()
    {
      cout<<"Derived Constructor"<<endl;
    }
   ~Derived()
    {
      cout<<"Derived Destructor"<<endl;
    }
};
```

Whenever we are deallocating object of a class then in that case first Derived class destructor is called first then of the base class bottom to top approach.

Virtual Destructor

```
int main()
{
    Base *p=new Derived();
    :
    delete p;
}
```

× Derived Der---.
– Base Destructor.

```
class Base
{
  public:
    Base()
    {
      cout<<"Base constructor"<<endl;
    }
  virtual ~Base()
    {
      cout<<"Base destructor"<<endl;
    }
};
```

```
class Derived :: public Base
{
  public:
    Derived()
    {
      cout<<"Derived Constructor"<<endl;
    }
   ~Derived()
    {
      cout<<"Derived Destructor"<<endl;
    }
};
```

So we have pointer of base type to compiler ko to yhi lgega ki base ka pointer hai to wo bas base ka object smjhega par hmne to derived ka object bnaya hai. ab heap mai dynamic create kiya hai to delete to karna chahiye derived ko bhi but ye call bas base ka destructor karega hence to overcome this issue make the destructor of base calss be a virtual destructor tb jake ye dono ko order mai call karega as needed and expected.

Types of functions in a class

```
class Rectangle
{
    private:
        int length;
        int breadth;
    public:
        Rectangle();                          // Constructor
        Rectangle(int l, int b);              // Constructor
        Rectangle(Rectangle &r);              // Constructor
        void setLength(int l);                // Mutator
        void setBreadth(int b);               // Mutator
        int getLength();                      // Accessor
        int getBreadth();                     // Accessor
        int area();                           // Facilitators
        int perimeter();                      // Facilitators
        int isSquare();                       // enquiry
        ~Rectangle();                         // Destructor
};
```

This is the way to write a class you may define all the functions here itself but the good practise is to define class like this and then later define them outside the class using scope resolution operator.

Pointer to an Object

- A pointer of type class can be created
- A pointer can point on existing object
- A new object can be created in heap using pointer
- Arrow operator is used for accessing members of an object using pointer

Data Hiding

- Data members of a class class are usually declared as Private or Protected,
- They can be accessed only inside the class and child classes
- Data finding protects data from mishandling

Constructors

- Constructor is a member function of a class
- It will have same name as class name
- It will not have return type
- Its should be public
- It can be declared as private also in some cases
- It is called when object is created
- It is used for initialising an object
- It can be overloaded
- If its not defined then class will have a default constructor
- Constructor can take default arguments

Types of constructors

- Non-argument constructor
- Parameterised constructor
- Copy constructor

All types of Member Functions

- Constructors - called when object is created
- Accessors - used for knowing the value of data members
- Mutators - used for changing value of data member
- Facilitator - actual functions of class
- Enquiry - used for checking if an object satisfies some condition
- Destructor - used for releasing resources used by object

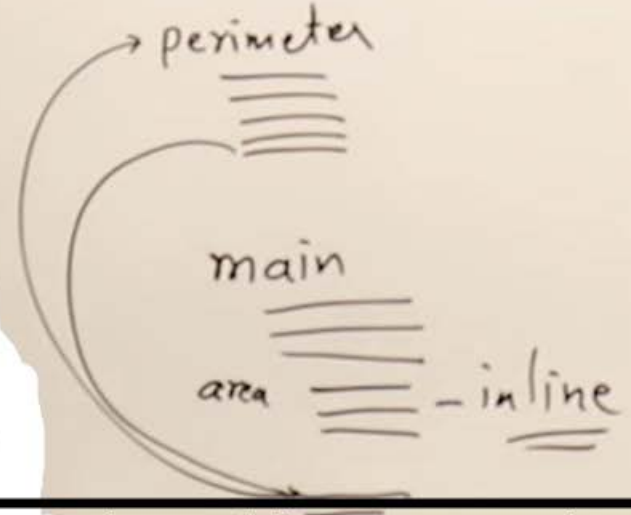
Class Rectangle
Write all functions for rectangle

```
class Rectangle
{
private:
    int length;
    int breadth;
public:
    Rectangle();
    Rectangle(int l,int b);
    Rectangle(Rectangle &r);
    int getLength(){return length;}
    int getBreadth(){return breadth;}
    void setLength(int l);
    void setBreadth(int b);
    int area();
    int perimeter();
    bool isSquare();
    ~Rectangle();
};
int main()
{
    Rectangle r1(10,10);
    cout<<"Area "<<r1.area()<<endl;
    if(r1.isSquare())
        cout<<"Yes"<<endl;
}

Rectangle::Rectangle()
{
    length=1;
    breadth=1;
}
Rectangle::Rectangle(int l,int b)
{
    length=l;
    breadth=b;
}
Rectangle::Rectangle(Rectangle &r)
{
    length=r.length;
    breadth=r.breadth;
}
void Rectangle::setLength(int l)
{
    length=l;
}
void Rectangle::setBreadth(int b)
{
    breadth=b;
}
int Rectangle::area()
{
    return length*breadth;
}
int Rectangle::perimeter()
{
    return 2*(length+breadth);
}
bool Rectangle::isSquare()
{
    return length==breadth;
}
Rectangle::~Rectangle()
```

{cout<<"Rectangle Destroyed";}

Scope Resolution Operator

```
void main()
{
    Rectangle r(10,5);
    cout<<r.area();
    cout<<r.perimeter();
}
```



Now here area became inline so you cannot have complex logic in that function otherwise it will not be a good practise. Now in other case when we defined it outside class using scope resolution in that case we have seprate section for that function and if we call it the program control jumps to that place and get excuted and hence complex logics can also be implemented.

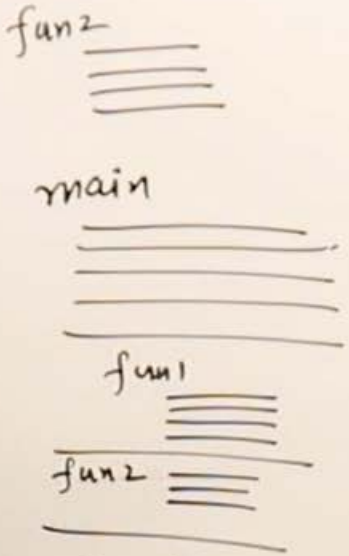
This tells that this fuction or entity is defined within the scope of given class.

```
class Rectangle
{
    private:
        int length;
        int breadth;
    public:
        int area()
        {
            return length*breadth;
        }
        int perimeter();
};
int Rectangle::Perimeter()
{
    return 2*(length+breadth);
}
```

So if a function is defined within class then actaully it automatically become inline.

inline function

Class ke andr koi function ka decleration aur definition kiya to by default inline ho gya no seprate block will be generated for all inline function. Now one more way to make a function inline that you write inline keyword before that fucntion jaroori nhi class ka hi function ho normal function bhi.




```
int main()
{
    Test t;
    t.fun1();
    t.fun2();
}
```

```
class Test
{
  public:
    void fun1()
    {
        cout<<"Inline";
    }
inline void fun2();
};

void Test::fun2()
{
    cout<<"non-inline";
}
```

Yha phle inline nhi likha tha to memory mai fun2 ka alg se hi block tha par jaise hi inline likha wo niche main mai hi ghul mil gya.

Complex function ke liye inline advisable nhi hai issse code ki lenegth bewjh badhti hai bas haa time kam karna ho execution ka ki yar function ko yhi le aao jump bar bar na kare chota sa hi chij hai us function ko execute karna to inline kar liya.

Learn > Learn > main.cpp > Demo

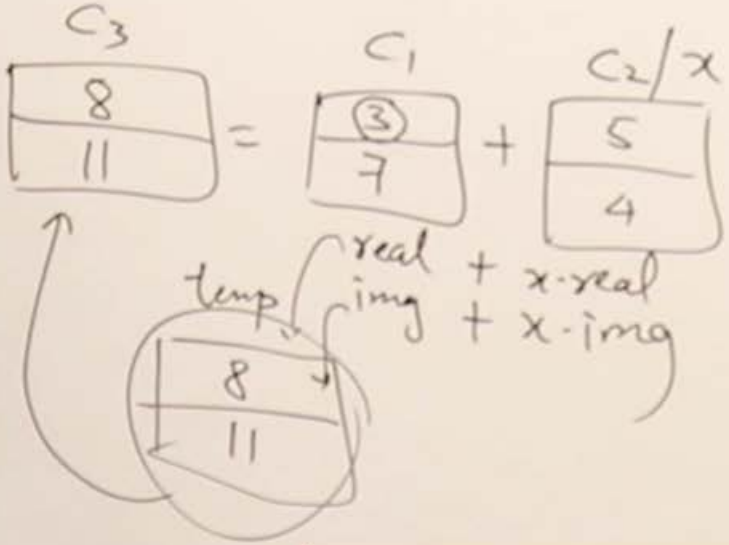
Struct vs class

```cpp
#include <iostream>

using namespace std;

struct Demo
{
    int x;
    int y;

    void Display()
    {
        cout<<x<<" "<<y<<endl;
    }
};

int main()
{
    Demo d;
    d.x=10;
    d.y=20;
    d.Display();

}
```

This can be done when structure is used beacuse by default sab public hai struct mai it also allow you to have function in c++ same thing class phir different kaise hai to class mai by default sab chij public hoga.

Operator Overloading

```
main()
{
    Complex C1(3,7)
    Complex C2(5,4)
    Complex C3;

    C3 = C1.add(C2);
```



So this is the method that we actually use to do ki add nam ka function define kiya aur wo add ka khel kar diya. That is purana tareeka concept is actually in next page

```
Class Complex
{
  private:
    int real;
    int img;
  public:
    Complex(int r=0, int i=0)
    {
      real=r;
      img=i;
    }
Complex add(Complex x)
{
    Complex temp;
    temp.real=real+x.real;
    temp.img=img+x.img;
    return temp;
}
```

Operator Overloading

```
main()
{
    Complex C1(3,7)
    Complex C2(5,4)
    Complex C3;

    C3 = C1.add(C2);

    C3 = C1 + C2
```

Now we used keyword operator and then told the compiler ki konsa wla operator ko hm overload karne ja rhe hai and phir usko as shown above we used. to execute the result.

```
class Complex
{
  private:
    int real;
    int img;
  public:
    Complex(int r=0, int i=0)
    {
      real = r;
      img = i;
    }

Complex operator+(complex x)
{
    Complex temp;
    temp.real = real + x.real;
    temp.img = img + x.img;
    return temp;
}
```

Operator overloading

- Operators can be overloaded on our classes
- We can define operator for our own classes
- Operators can be overloaded using member functions or friend functions
- Global functions can also access private and protected members of an object if they are declared as friend inside a class

```
class Complex
{
private:
    int real;
    int img;
public:
    Complex(int r=0,int i=0)
    {
        real=r;
        img=i;
    }
    void display()
    {
        cout<<real<<"+i"<<img<<endl;
    }

    Complex operator+(Complex c)
    {
    Complex temp;
    temp.real=real+c1.real;
    temp.img=img+c1.img;
    return temp;
    }


  };


int main()
{
    Complex c1(5,3),c2(10,5),c3;

    c3=c1+c2;

    c3.display(); }
```

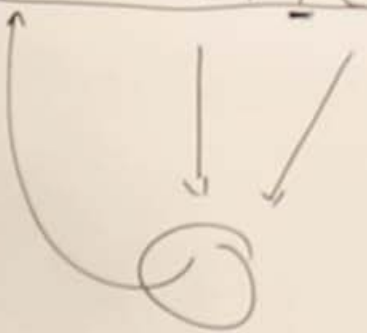
Operator Overloading

Friend function ki help se operator overloading

```
main()
{
    Complex c1(3,7)
    Complex c2(5, 4)
    Complex c3;
```

```
class Complex
{
    private:
        int real;
        int img;

    public:
        friend Complex operator+(Complex c1, complex c2)
        };
```

C3 = C1 + C2



```
Complex operator +(Complex c1, Complex c2)
{
    Complex t;
    t.real = c1.real + c2.real;
    t.img = c1.img + c2.img;
    return t;
}
```

Operator Overloading using Friend Functions

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

class Complex
{
private:
    int real;
    int img;
public:
    Complex(int r=0,int i=0)
    {
        real=r;
        img=i;
    }
    void display()
    {
        cout<<real<<"+i"<<img<<endl;
    }

friend Complex operator+(Complex c1,Complex c2);
  };

Complex operator+(Complex c1,Complex c2)
{
    Complex temp;
    temp.real=c1.real+c2.real;
    temp.img=c1.img+c2.img;
    return temp;
}

int main()
{
    Complex c1(5,3),c2(10,5),c3;

    c3=c1+c2;

    c3.display();
}
```

Operator Overloading

```
main( )
{
    Complex  c1(3,7)

    int  x=10;

    cout << x;

    c1.display( );
```

3+i7

```
class Complex
{
    private:
        int real;
        int img;

    public:

void  display( )
{
    cout<<real<<"+i"+img;
}
```

In general if we need to display this we use an function in this case display( ) but there must be a way ki << se hi sidhe wo output jo hme chahiye jis fprm mai wo de de. Will see this in next image now.

Operator Overloading

These are done not same as for arithmetic operator. It is implemented through friend function. The reason behind this is here as we see in the signature of function we are using ostream as well as complex in input. Hence, we can not have this function within class. we must use and make this function as friend function to make this work.

```
main()
{
    Complex c1(3,7)
    display();
    cout<<c1;
}
```

```
class Complex
{
    private:
        int real;
        int img;
    public:
        friend ostream & operator<<(ostream &o, Complex &c1)
        };

ostream & operator<<(ostream &o, complex &c1)
{
    o<<c1.real<<"+i"<<c1.img;
    return o;
}
```

3+i7

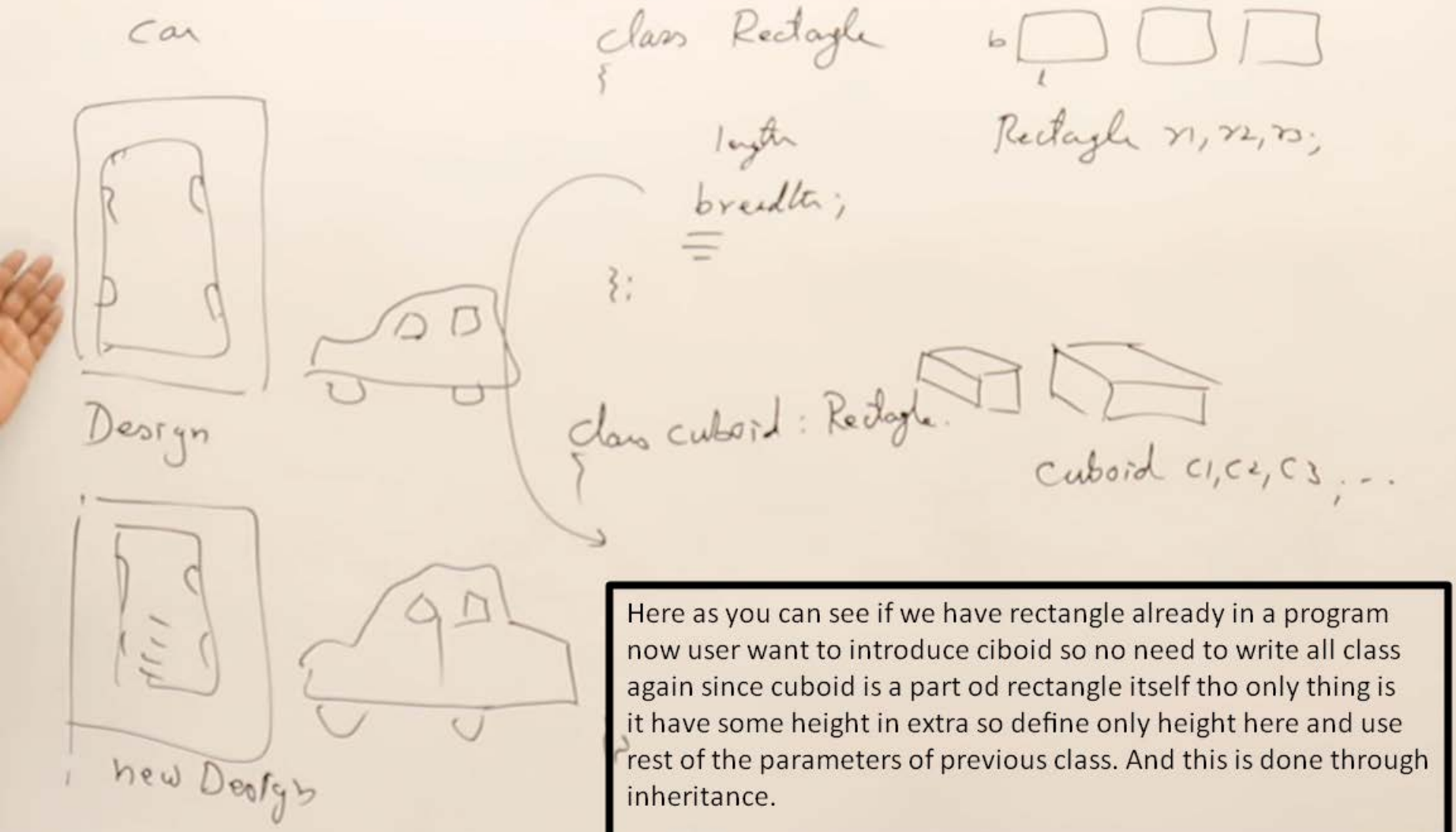
Stream operator overloading

Input stream cin >> (extraction) operator can be overloaded upon a class
output stream cout << (insertion) operator can be overloaded upon a class

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

class Complex
{
private:
    int real;
    int img;
public:
    Complex(int r=0,int i=0)
    {
        real=r;
        img=i;
    }

friend ostream & operator<<(ostream &out,Complex &c);
  };

ostream & operator<<(ostream &out,Complex &c)
{
    out<<c.real<<"+i"<<c.img<<endl;
    return out;
}

int main()
{
    Complex c(10,5);
    cout<<c<<endl;

    operator<<(cout,c);
}
```

Inheritance

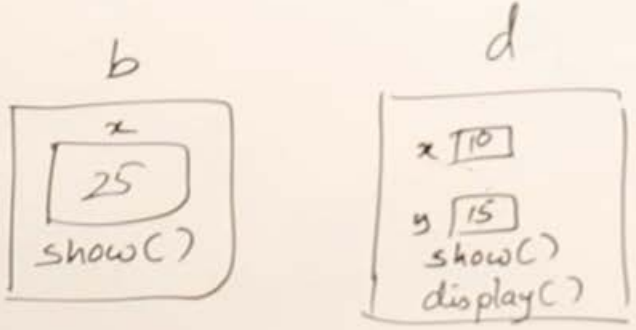
Using a predefined class to define some new class in which new class have all the features of previous class. That is called inheritance.



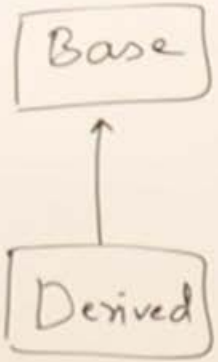
Here as you can see if we have rectangle already in a program now user want to introduce ciboid so no need to write all class again since cuboid is a part od rectangle itself tho only thing is it have some height in extra so define only height here and use rest of the parameters of previous class. And this is done through inheritance.

Inheritance

Example and memory representation of the objects of the data members and functions derived in a class from a base class.




```
class Base
{
    public:
    → int x;
    → void show()
      {
        cout<<x;
      }
};
```




```
int main()
{
    Base b;
    b.x=25;
    b.show(); —— 25
```

```
    Derived d;
    d.x=10;
    d.y=15;
    d.show(); ———— 10
    d.display(); —— 10 15
```

```
class Derived :public Base
{
    public:
        int y;
        void display()
        {
            cout<<x<<" "<<y;
        }
};
```

Inheritance

- It is a process of acquiring features of an existing class into a new class
- It is used for achieving reusability
- features of base class will be available in derived class

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

class Base
{
public:
    int a;
    void display()
    {
        cout<<"Display of Base "<<a<<endl;
    }
};

class Derived:public Base
{
public:
    void show()
    {
        cout<<"Show of Derived"<<endl;
    }
};

int main()
{
    Derived d;
    d.a=100;
    d.display();
    d.show();
}
```

Constructors in Inheritance

```
int main()
{
    Derived d;
}
```

```
Default of Base
Default of Derived
```

So understadnd this concept with a story tumhe ek lakdi ka table chahiye tumne dukandar ko kha bas table ready kar do to you not told him any dimension to phirr wo apne general ke according jo default pe bnata ahi wo usi hisan se bna dega. But iske liye use kya kya karna hoga to iska answer hai ki use wood class ko btana hoga phle ki tum default ke hisAB SE lakdi bhej do aab uske bad wo call ho gya to next to it apna wla default call karega. to order of execution of constructor yhi hai. Now aab kya hai ki in case derived mai kuch parametre pass bhi kar diya ki bhaiya fla fla dimension ka bna do par in that case as well lakdi wla default hi call krega base yni wood class ka kyoki uska to rule hai ki pesa wood pe mai utna hi karoonga kharch usse bn paye to thik nhi to dekh lo. To aab if wood bhi bdhana hai to uska mthd dekhenge next slide mai.

```
class Base
{
  public:
    Base()
    {
      cout<<"Default of Base"<<endl;
    }
    Base(int x)
    {
      cout<<"Param of Base"<<x<<endl;
    }
};

class Derived : public Base
{
  public:
    Derived()
    {
      cout<<"Default of Derived";
    }
    Derived(int a)
    {
      cout<<"Param of Derived"<<a;
    }
};
```

Constructors in Inheritance

```
int main()
{
    Derived d(10);
}
```

Default of Base

Param of Derived 10

```
class Base
{
  public:
    Base()
    {
      cout<<"Default of Base"<<endl;
    }
    Base(int x)
    {
      cout<<"Param of Base"<<x<<endl;
    }
};

class Derived : public Base
{
  public:
    Derived()
    {
      cout<<"Default of Derived";
    }
    Derived(int a)
    {
      cout<<"Param of Derived"<<a;
    }
};
```

Ye whi dusra wla part pichle statement ka ki bhaiya same dam ka bnao par ye dimension rkhna to default of base yni wood will be executed aur table maker ka parametrised will be called.

Constructors in Inheritance

```
int main()
{
    Derived d(20,10);
}
```

Param of Base 20

Param of Derived 10

```
Derived(int x, int a) : Base(x)
{
    cout<<"Param of Derived"<<a;
};
```

```
class Base
{
  public:
    Base()
    {
      cout<<"Default of Base"<<endl;
    }
    Base(int x)
    {
      cout<<"Param of Base"<<x<<endl;
    }
};
```

```
class Derived : public Base
{
  public:
    Derived()
    {
      cout<<"Default of Derived";
    }
    Derived(int a)
    {
      cout<<"Param of Derived"<<a;
    }
```

Now this is the proper way ki hme derived class mai hi iss syntax ke according ek constructor bnana hoga and theta will call the parametrised constructor of both the base class and derived class mtlb wood bhi mere according aur aab table bhi mere according. and again execution wese hi hoga phle base ka and then derived ka jaise ki upr mai show kiya bhi gya hai.

Constructors in inheritance

- Constructors of base class is executed first then the constructor of derived class is executed.
- By default, non-parameterised constructor of base class is executed.
- parameterised constructor of base class must be called from derived class constructor

Explain using base and derived class

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

class Base
{
public:
    Base(){cout<<"Non-param Base"<<endl;}
    Base(int x){cout<<"Param of Base "<<x<<endl;}
};
class Derived:public Base
{
public:
    Derived(){cout<<"Non-Param Derived"<<endl;}
    Derived(int y){cout<<"Param of Derived "<<y<<endl;}

    Derived(int x,int y):Base(x)
    {
        cout<<"Param of Derived "<<y<<endl;

    }
};

int main()
{
    Derived d(5,10);
}
```

isA vs hasA

hasA

```
class Table
{
    Rectangle top;
```

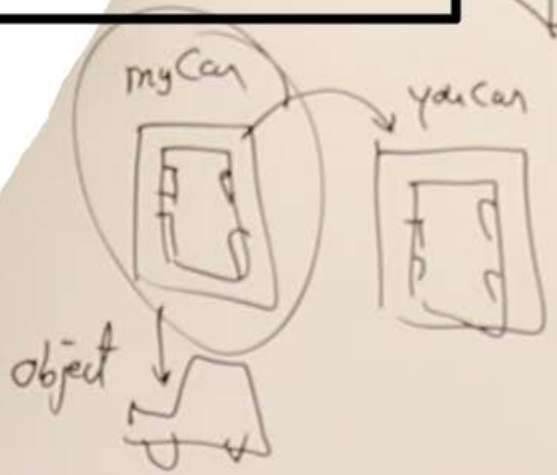
```
class Rectangle
{
    private: —
    protected: —
    public: —
}
```

isA

```
class Cuboid : public Rectangle
{
```

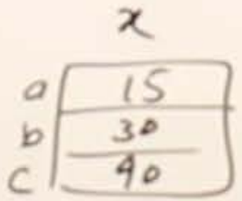
So when we define a class object within a class in thayt case we say that class in this case that is table so table has a rectangle

Now in the case when we have derived class or say inheritance is implemented in that case we can say that derived class is a base class that in this case we can say cuboid is a rectangle.



Access Specifiers

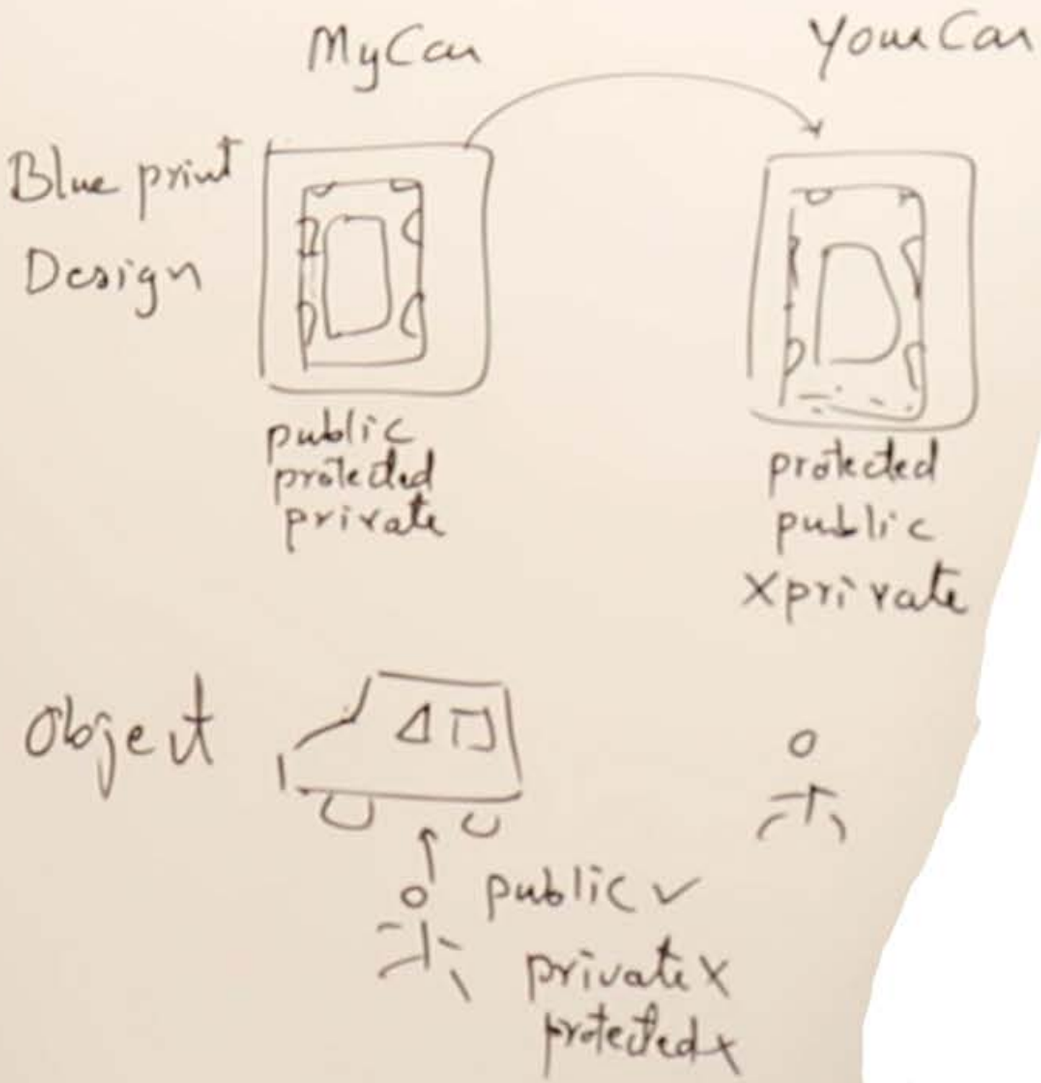
```
int main()
{
    Base x;

    x.a=15;   ✗
    x.b=30;   ✗
    x.c=90;   ✓
```




```
class Base
{
  private:
    int a;
  protected:
    int b;
  public:
    int c;
    void funBase()
    {
      a=10;
      b=20;
      c=30;
    }
};

class Derived : Base
{
  public:
    funDerived()
    {
      a=1;   ✗
      b=2;   ✓
      c=3;   ✓
    }
};
```

Access specifiers are the keywords that basically defines the scope of a particular data member or data function.
The acess specifiers are:
private - they are accessible only within class.
protected - they can be accessed outside as well but with some conditions applied.
public - They are accesssible throughout the program everywhere whereever object related to that class is called

Access Specifiers



Lets try to understand this access specifier story with an example:

To man lo ki meri ek car ki company hai I created mycar and usme maine apna define kar diya private, protected and public members now later on maine naya design bnaya aab socha purana kya krna mujhe to that time some local company contacted ki yar apna model hme de de hm usi ke according bna ke bech lenge to maine kha chl thik hai to maine use derive karne ka permission de diya to now as he derived to on that case he can access only protected and public data members mtlb ki maine kha ki tum sab to bdl lo but kuch design maine protected rkhe hai use tumhe as it is hi use krna parega.

now third to this is ki ek car mycar company ki yha jo bhi buyer hai wo bas public member ko access kar skta hai. maine user ko ye permission bhi nhi di ki wo protected ko modify kar ske. now par jb mujhse kisi ne derive ki permission li to miane kha chlo protected bhi kar lo modify but private jo ai doon whi karna hai tumhe.

Access Specifiers

	private	protected	public
inside class	✓	✓	✓
inside Derived class	✗	✓	✓
On object	✗	✗	✓

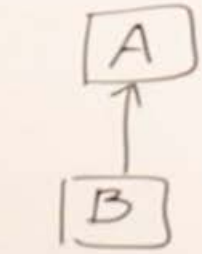
So in short yhi hmne discuss kiya hai pichle page pe to is table se whi bat pta chl rhi hai cross means not accessible and tick means accessible.

Access Specifiers

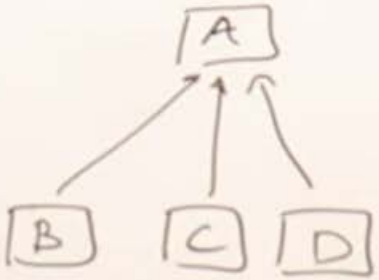
- Private - Accessible only inside a class
- Protected - Accessible inside a class and inside derived classes
- Public - accessible inside class, inside derived class and upon object

```
class Base
{
private: int a;
protected: int b;
public: int c;

    void funBase()
    {
        a=10;
        b=5;
        c=15;
    }
};
class Derived:public Base
{
public:
    void funDerived()
    {
        a=10;
        b=5;
        c=15;
    }

};

int main()
{
    Base b;
    b.a=10;
    b.b=5;
    b.c=20;
}
```

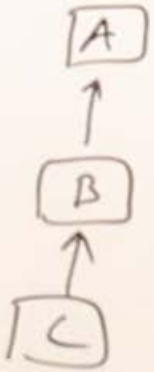
Types of Inheritance



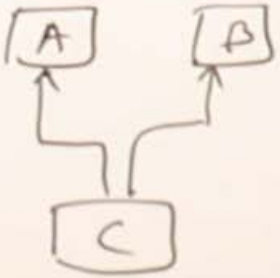
Simple/single



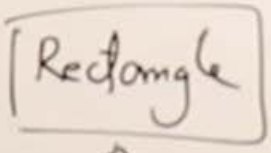
Hierarchical

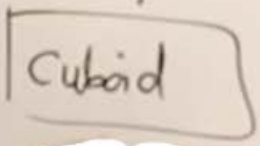


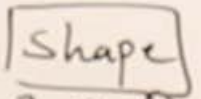
Multilevel

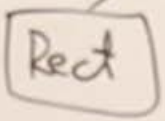


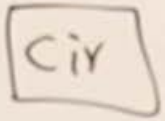
Multiple

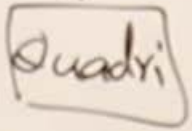


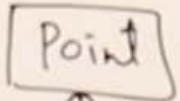


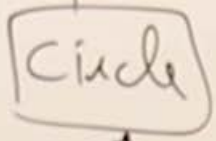


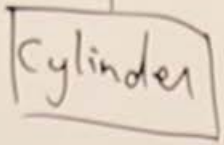


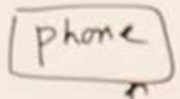


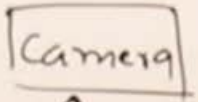


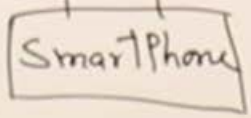


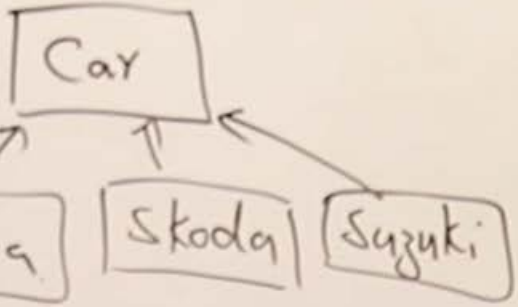








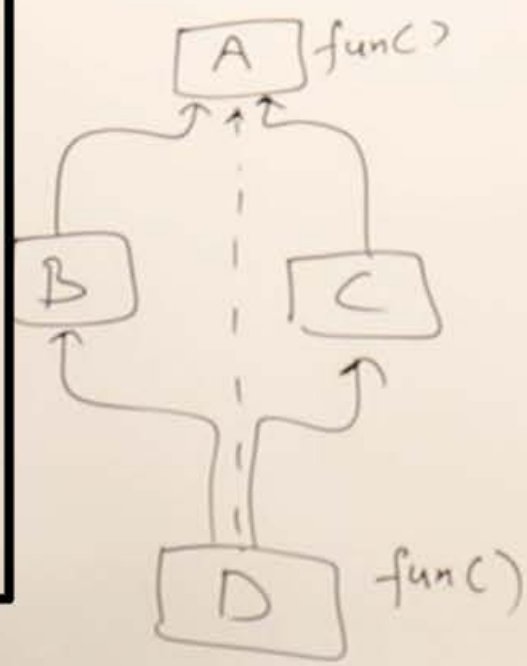




Sab to thik hai par ek bat aur ki aap multiple inheritance bas yha c++ mai dekh skte ho aur jgho par ye feature allowed nhi hai

Types of Inheritance

In c++ we are allowed to have hybrid mode of inheritance mtlb ki hm jaise yha dekh skte hai ki A se heirarchy mode wla derive kiya do class B aur class C now isse multiple use karte hue D derive kiya. Now suppose one thing ki fun ( ) function hai koi in class A to ab it is available in D obviously but it can be accessed through two path A -> B -> D or A -> C -> D and this can lead to ambiguity And hence to overcome this make class B and class C by deriving class through virtual keyword. It will help but how will se later.



Multipath

```
class A
{
=
};

class B: virtual public A
{
};

class C: virtual public A
{
};

class D: public B, public C
{


};
```

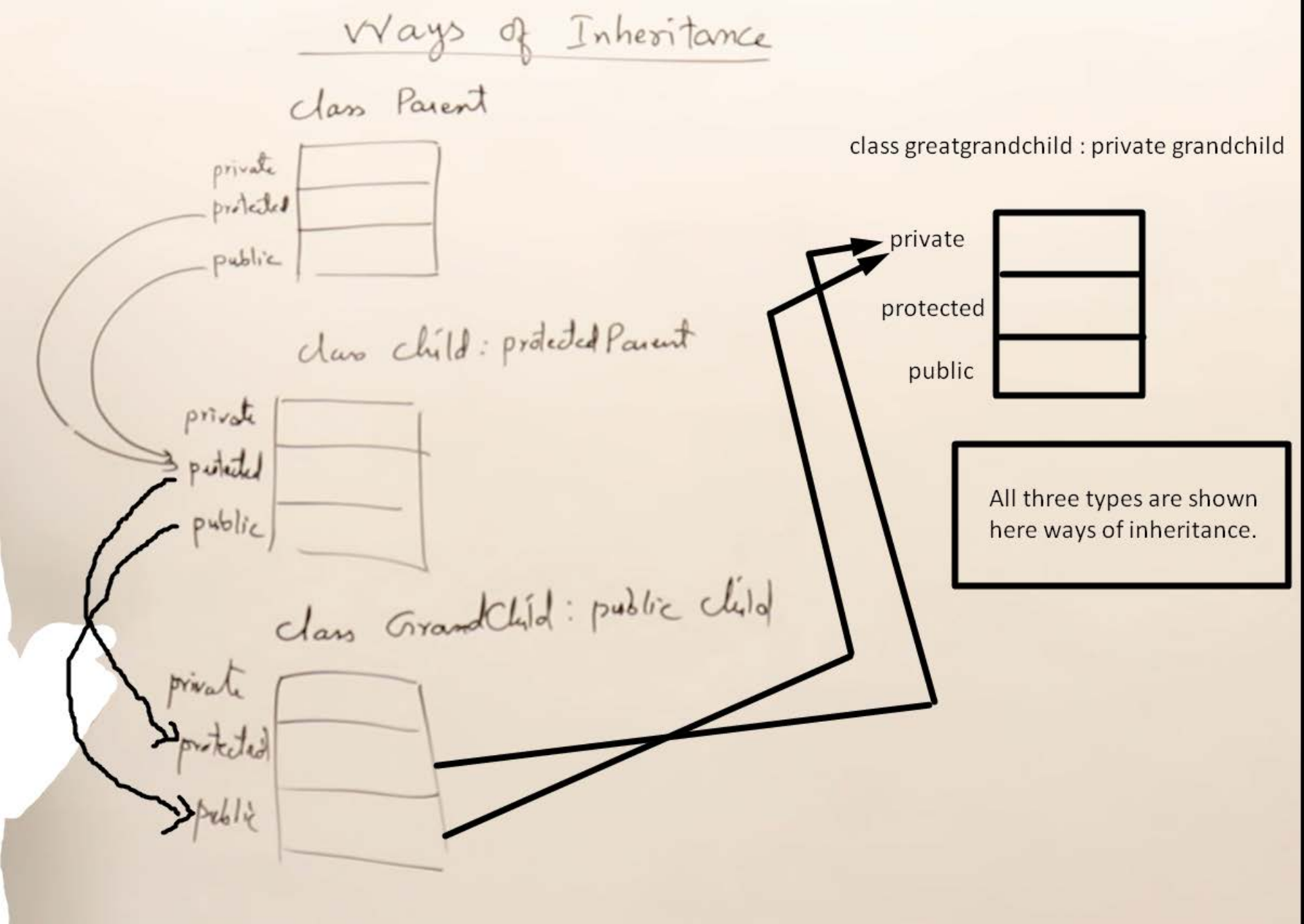
Ways of Inheritance

```
class Base
{
    private : int a;
    protected : int b;
    public : int c;
};

class Derived : privately Base
{
  private:
    void fun()
    {
      a=10;
      b=5;
      c=3;
    }
};
```

So we can inherit in three ways:
1. private
2. protected
3. public

If not mentioned any thing then by default it is private mode of inheritance.

Ways of Inheritance



Ways of inheritance

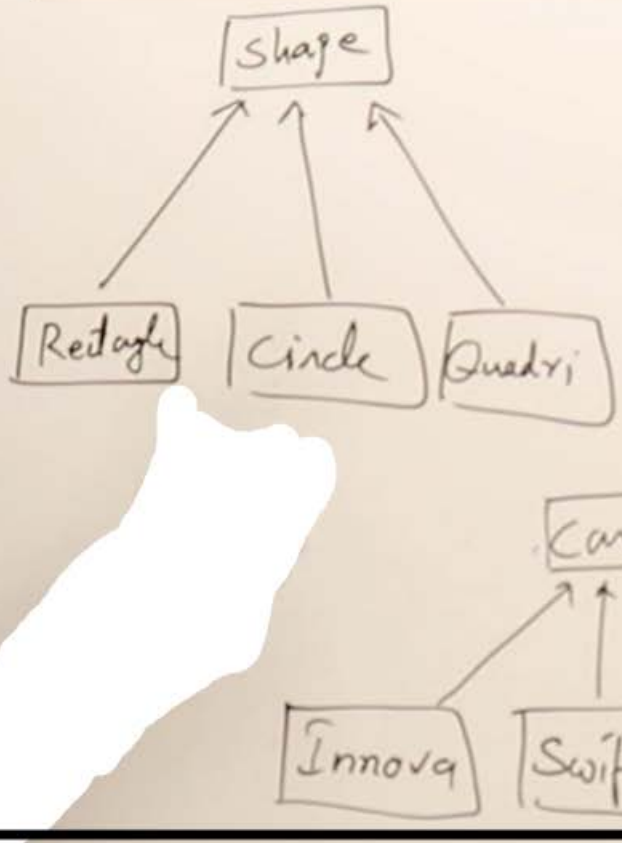
A class can be inherited in flowing ways

Publicly	-	All members of base will have same accessibility in derived class
Protectedly	-	All members of base will become protected in derived class
Privately	-	All members of base will become private in derived class

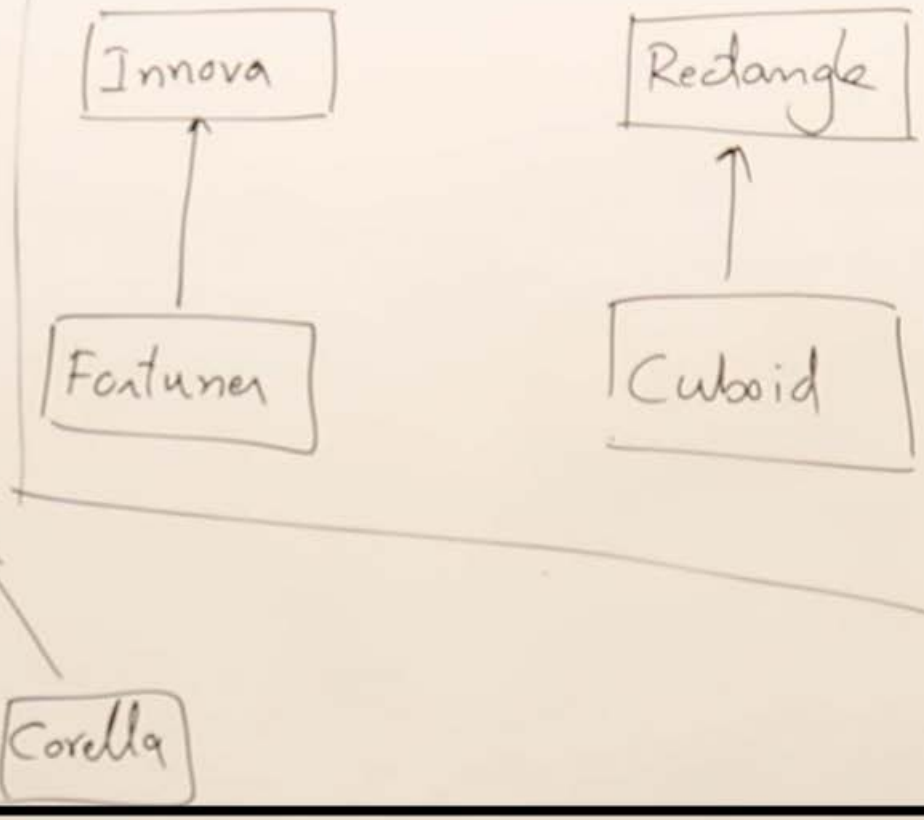
```
class Parent
{
private: int a;
protected: int b;
public: int c;

    void funParent()
    {
        a=10;
        b=5;
        c=15;
    }
};
class Child: private Parent
{
private:

protected:

public:

    void funChild()
    {
        //a=10;
        b=5;
        c=15;
    }

};
class GrandChild : public Child
{
public:
    void funGrandChild()
    {
        //a=10;
        //b=5;
        //c=20;
    }
};
int main()
{
   // Child c;
    //c.a=10;
    //c.b=5;
    //c.c=20;
}
```

Generalization vs. Specialization

These are example of generalisation



These are example of specialisation

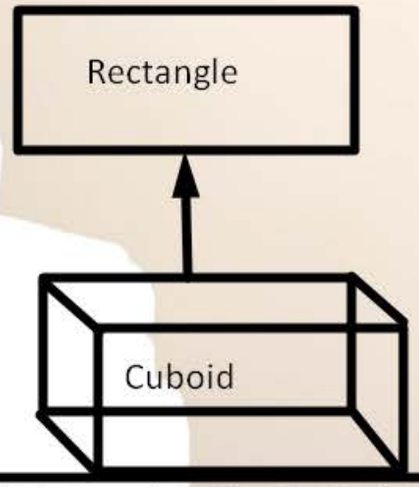


Generalisation means bottom to top approach that is ki base class have nothing to give to child class par kuch classes hme ese dikhe jiske bad hme lga ki nhi iska koi common name hona chahiye to hmne ek common base clas kah diya. And generalisation can give us the idea of polymorphism. In this case bse class in general have no real presence ki shape mujhe dikha skte ho kya kya hai nhi car dikha skte ho nhi. So that is the way.

Now in specialisation we have ki top to bottom approach mtlb ye hai ki phle jo base class hai wo bhi real world mai exist to karta hai aab later on we have thought of ki kuch naya mosify karte hai like innova toyota company ki car thi now company decided ki same seat capacity ke sath naya model nikalte hai kuch extra feature dalenge usme to for that purpose we have this approach.

So in short we have generalisation gives an idea polymorphism and specialisation gives us an idea of actual inheritance that is sharing and passing of some pre-existing features.

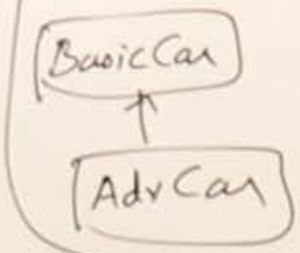
Base class Pointer Derived class Object



So aab hmne phle ek pointer bnaya rectangle type ka aur usse ek cuboid type ke class pe point kar diya. now in short hmne ye kha hai ki cuboid jo hai wo ek rectangle hai kyoki pointer to base class ka hi liya na so we said that ki cuboid object is a rectangle and ye shi bhi hai koi glt bat bhi nhi hai we saw this before as well is a and has a wle concept mai. Now hmne kha koi cuboid type ka object bnaya hai hmne par pointer to kewal rectangle ka hai mtlb ki access kewal rectangle ka function ho skta hai so using P we can only access objects of rectangle class.

```
int main()
{
    Base *P;
    P=new Derived();
    P->fun1();
    P->fun2();
    P->fun3();
    P->fun4();   //Not allowed
    P->fun5();   //Not allowed
```

```
class Base
{
public:
    void fun1();
    void fun2();
    void fun3();
};

class Derived : public Base
{
public:
    void fun4();
    void fun5();
};
```




```
BasicCar *P;
P=AdvCar();
```

One more thing
Derived *P;
P= new Base ();
ye valid nhi hoga and ye hai mtlb iska ki aap bete ka object bna rhe ho but property use jab de rhe ho to bas baap ki aur aab chahte ho ki beta apna bhi gun show kare that is impossible.

Base class Pointer pointing to derived class object

- Base class pointer can point on derived class object
- But only those functions which are in base class, can be called
- If derived class is having overrides functions they will not be called unless base class functions are declared as virtual
- Derived class pointer cannot point on base class object

Example 1

```
class Base
{
public:
    void fun1()
    {
        cout<<“fun1 of Base "<<endl;
    }
};
```

Example 2

```
class Derived: public Base
{
public:
    void fun2()
    {
        cout<<“fun2 of Derived"<<endl;
    }
};

class Rectangle
{
public:
    void area()
    {
        cout<<“Area of Rectangle"<<endl;
    }
};

class Cuboid: public Rectangle
{
public:
    void volume()
    {
        cout<<“Volume of Cuboid"<<endl;
    }
};
```

Example 3

```
class BasicCar
{
public:
    void start()
    {
        cout<<"Car started"<<endl;
    }
};

class AdvanceCar: public BasicCar
{
public:
    void playMusic()
    {
        cout<<"Music Playing"<<endl;
    }
};
```

Function Overriding

```
Parent P;
P.display(); — Display of parent

Child c;
c.display(); — Display of child
```

```
class Parent
{
  public:
    void display()
    {
      cout<<"Display of parent";
    }
};
class child : public Parent
{
  public:
    void display()
    {
      cout<<"Display of child";
    }
};
```

The concept is ki manlo feature hai mycar class mai ki window up down aur kisine yourcar bnai to maine tu feature meri use kar le apna modify kar lo par you thought ki yar window mai na ye mai key system lga doon an to acha rhega so you re modified predefined function of window open and close. And this redefining is called function over-riding.

Function Overriding

- Redefining a function of base class in derived class
- Function overriding is used for achieving runtime polymorphism
- Prototype of a overrides function must be exactly same as base class function

```
class Base
{
public:
    void fun()
    {
        cout<<“fun of Base"<<endl;
    }
};

class Derived: public Base
{
public:
    void fun()
    {
        cout<<“fun of Derived"<<endl;
    }
};

int main()
{
          Derived d;
          d.fun( );
}
```

Calling Overrided Method

```
int main()
{
    Base *p=new Derived();
    p->fun();
    Derived d;
    d.fun();
}
```

```
class Base
{
  public:
  virtual void fun()
    {
      cout<<"fun of Base";
    }
};
class Derived : public Base
{
  public:
    void fun()
    {
      cout<<"fun of Derived";
    }
};
```

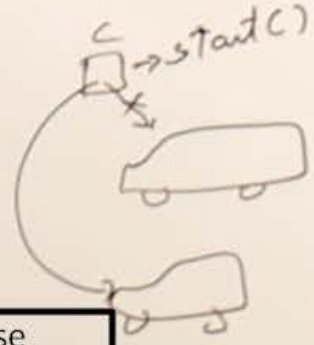
So shi bat hai hme phle lga bhi tha jab base class pointer wla kar rhe the ki yar object to derived ka hai na to karne do na derived ke function ko call but ye c++ mai direct tha nhi. Par to correct the things if you have done overriding and want to use overrided of derived then in that case make the function in base class being virtual. To when ever it is called we call the function of derived class if virtual is used.

Virtual Functions

- Virtual functions are used for achieving polymorphism
- Base class can have virtual functions
- Virtual functions can be overrides in derived class
- Pure virtual functions must be overrides by derived class

```
class BasicCar
{
public:
    virtual void start(){cout<<"BasicCar started"<<endl;}
};

class AdvanceCar: public BasicCar
{
public:
    void start(){cout<<"AdvanceCar Started"<<endl;}
};

int main()
{
    BasicCar *p=new AdvanceCar();
    p->start();
}
```

Polymorphism




```
main()
{
    Car *C = new Innova();
→   C->start();   — "Innova started"
    C = new Swift();
→   C->start();   — "swift started"
```

```
class Car
{
  public:
  virtual void start()=0;
  virtual void stop()=0;   ← pure virtual function
};
```


```
class Swift : public Car
{
  public:
    void start()
    {
      cout << "Swift started";
    }
    Void stop()
    {
      cout << "swift stopped";
    }
};
```

```
class Innova : public Car
{
  public:
    void start()
    {
      cout << "Innova started";
    }
    void stop()
    {
      cout << "Innova stopped";
    }
};
```

So bat kya hui ki we can use generalisation wla concept ki car hai to kuch nhi par hmne kh diya ki class hai. Now hmne kya kiya ki car ke function ko bas signature mai define to kar diya par hmara intension ye ktai nhi ki hm usko use kare hm chahte jhai ki innova ka jab call ho to innova ka start call ho aur jab swift ko call kare to swift ka start call ho. to piche jo virtual bna ke kiya tha whi ek way hai now iske alawa ye bhi hai point ki perfect virtual function wo hi hia jo ki definitely override kiya jaye apne derived class mai aur base class mai use hm define nhi karte pura. hm as shown in this esa hi karte hai.

Polymorphism

- Same name different actions
- Runtime Polymorphism is achieved using function overriding
- Virtual functions are abstract functions of base class
- Derived class must override virtual function
- Base class pointer pointing to derived class object and a override function is called

Summary: class car is defined, then sub classes override, then base class method made as virtual the pure virtual

```
class Car
{
public:
    virtual void start()=0;
};
class Innova:public Car
{
public:
    void start(){cout<<"Innova Started"<<endl;}
};
class Swift:public Car
{
public:
    void start(){cout<<"Swift Started"<<endl;}
};

int main()
{
    //Car c;
    Car *p=new Innova();
    p->start();
    p=new Swift();
    p->start();

}
```

Abstract classes

If a class contain an pure virtual function within it then that class is called abstract class.

Abstract ⟶

```
class Base
{
    public:
        void func()
        {
            cout<<"Base fun1";
        }
    virtual void fun2()=0;
};
```

✗ `Base b;`

✓
```
Base *p=new Derived();
p->fun2();
```

So if any class have a pure virtual function then creating the object of that class is not possible at all it will show you error im the program if you try to make the object of such class.

But you can create the pointer of that class and cam access the function of base as well as overrided function in the derived class.

```
class Derived : public Base
{
    public:
        void fun2()
        {
            cout<<"Derived fun2";
        }
};
```

Abstract classes

interface

Base
All concrete

↓

reusability

Base
some concrete
Some pure virtual

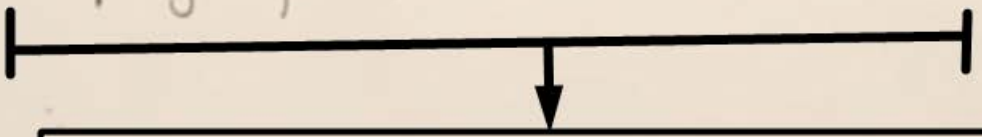
↓

reusability
polymorphism

Base
All pure virtual

↓

polymorphism



These two are called abstract class. and concrete function means reusablitity hai us function ka ki like car make sab car make lets say make hui to make () sab mai hm chahte hai ki reuse ho but start() sb apna apna khe to wo car class that is base class to usme use pure virtual function bna diya hmne.

Abstract class

- Class having pure virtual function is a abstract class
- Abstract class can have concrete also.
- Object of abstract class cannot be created
- Derived class can must override pure virtual function, otherwise it will also become a abstract class.
- Pointer of abstract class can be created
- Pointer of abstract class can hold object of derived class
- Abstract classes are used for achieving polymorphism
-
- Base class can be
- Concrete
- Abstract with some concrete and some pure virtual functions
- All virtual functions

Explain using base and derived class

```
using namespace std;

class Base
{
public:
    virtual void fun1()=0;
    virtual void fun2()=0;
};
class Derived :public Base
{
public:
    void fun1()
    {
        cout<<"fun1 of Derived"<<endl;

    }
    void fun2()
    {
        cout<<"fun2 of Derived"<<endl;
    }

};
int main()
{
    Derived d;
    d.fun1();
    d.fun2();

}
```

Friend Functions

```
class Test
{
    private:
        int a;
    protected:
        int b;
    public:
        int c;
friend void fun();
};

void fun()
{
    Test t;
    t.a=15;
    t.b=b;
    t.c=5;
}
```

To object to bas public data members ko access kar skta hai but we want ki ek function mai uska object hm bna rhe hai to wha pe uska all data member private public protected sab access kar ske to iske liye hme kya karna hoga ki class mai isi function ko define karo as shown bas uska prototype likhna hai aur kuch nhi iske bad bahar jake us function ko define kar do pura. There object of this class can acess all the data member private or protected or public.

Friend class

```
class your;
class My
{
    private:
        int a=10;
    friend your;
};
```

so same whi bat hai friend fucntion ki hi trh bas the difference here is ki hme agr class ko friend bnana hai to use ek bar define phle karna hoga like yha kiya hai pura class nhi define karna bas signature define kar do aur phir tum friend keyword ka use karte he bna lo use friend class and later define karo pura class to there you can use all the data members of my class in your class.

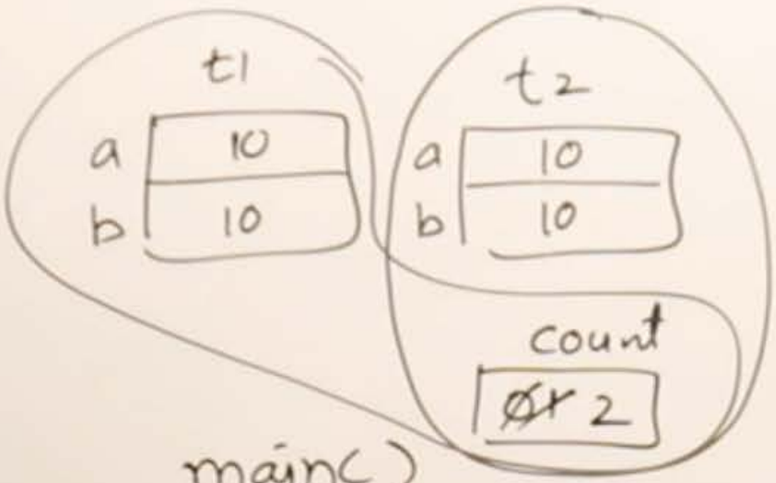
```
class your
{
    public:
        My m;
        void fun()
        {
            cout<<m.a;
        }
};
```

Friend functions and classes

- Friend functions are global functions
- They can access member of a class upon their objects
- A class can be declared as friend on another class
- All the functions of friend class can access private and protected members of other class

```
class Your;

class My
{
private:int a;
protected: int b;
public: int c;
    friend Your;
};
class Your
{
public:
    My m;
    void fun()
    {
        m.a=10;
        m.b=10;
        m.c=10;
    }

};
int main()
{

}
```

Static Members




```
main()
{
    Test t1;
    Test t2;
    cout<<t1.count; —2
    cout<<t2.count; —2
    cout<<Test::count; —2
```

```
class Test
{
  private:
      int a;
      int b;

  public:
      static int count;

      Test()
      {
          a=10;
          b=10;
          count++;
      }
};
```

→ `int Test::count=0;`

Static data members are created once in a program and that same is used throughout the whole program so here as you can see in the example if we have created the two objects of Test then count becomes 2 and for every case t1.count or t2.count or even with scope resolution all will have same value of count that is 2 here.

Static members are generally used for three purpose:
1. Counter
2. can be used as a shared memory like tum age dekhoge roll no wla example.
3. can provide information about the class like aage hi ek dekhoge example hai isme car price wla wo static price kar ke bna diya wo information de rha hai.

One more thing is that we are supposed to decleare all the static variable as shown here globally. But we decleare it within scope of Test class and also we have to initialise it there.

Static Members

```
class Test
{
  private:
     int a;
     int b;

  public:
     static int count;

     Test()
     {
        a=10;
        b=10;
        count++;
     }
     static int getcount()
     {
        a++;
        return count;
     }
};

int Test::count=0;
```

```
main()
{
   cout<<Test::getCount();-0
   Test t1;
   cout<<t1.getcount();
```

//Ways of aceessing static member function.

Static function are decleared this way.

The one important thing is we can only access static data members in static function non static data members can never be accessed within static function.

Static Members

- Static data members are members of a class
- Only one instance of static members is created and shared by all objects
- They can be accessed directly using class name
-
- Static members functions are functions of a class, they can be called using class name, without creating object of a class.
- They can access only static data members of a class, they cannot access non-static members of a class.

```
class Test
{
public:
    int a;
    static int count;

    Test()
    {
        a=10;
        count++;
    }
    static int getCount()
    {
        return count;
    }

};
int Test::count=0;

int main()
{
    Test t1,t2;
    cout<<Test::getCount()<<endl;
    cout<<t1.getCount()<<endl;
}
```

Static Members

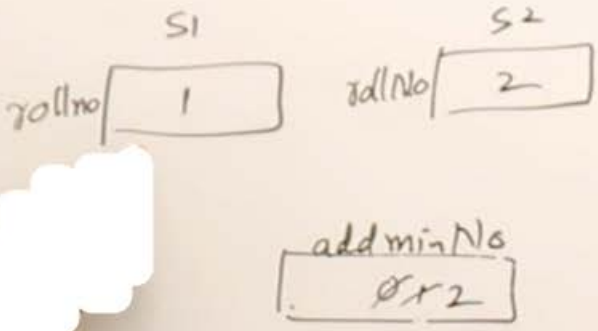
Static function can access all static data members of the class no matter even it is private public or protected.

Now real world mai smjho ese onr thing here ki you look for a car and price is fixed of it. But you want ki yar mujhe price btaoge ki nhi even if I donot bought the car so it can done as follow.

```
main()
{
// Accessing get price without creating object
cout<<Innova::getPrice();
Innova my;
cout<<my.getPrice();
// Accessing get price by creating object
```

```
class Innova
{
    public:

    static int price;

    Innova()
    {
      =
    }

    static int getPrice()
    {
        return price;
    }

};

int Car::price=20;
```

Static Members




```
class Student
{
    public:
    int rollno;
    static int addminNo;

    Student()
    {
        addminNo++;
        rollno=addminNo;
    }
};

int Student::addminNO=0;

main()
{
    Student s1;
    Student s2;
```

So one more real world use ki manlo hm admission le rhe hai to kya kiya ki admissionno ka static bnaya aur jitne bhi object create ho rhe hai aage wo us hisab se admissionno ko update kar rhe hai and every time update hokar wo to ek hi jgh rhta hai par hm rollno mai use har bar stire bhi akr lete hai isse ka fayda hai ki rollno hme mil gya hme dekhna nhi parega ki yar kha tha last jitna bhi create karo wo uske according hi vary kar ke store karta jata hai.

Example of Static

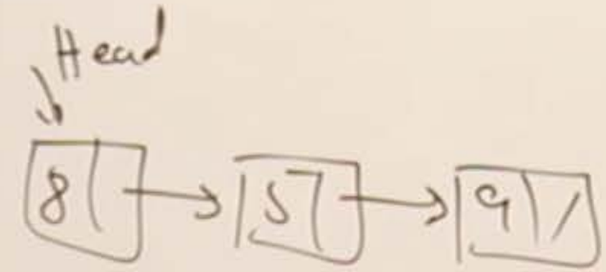
```
#include <iostream>
using namespace std;

class Student
{
public:
    int roll;
    string name;
    static int addNo;
    Student(string n)
    {
        addNo++;
        roll=addNo;
        name=n;
    }
    void display()
    {
        cout<<"Name "<<name<<endl<<"Roll "<<roll<<endl;
    }
};

int Student::addNo=0;

int main()
{
    Student s1("John");
    Student s2("Ravi");
    Student s3("Khan");
    Student s4("Khan");
    Student s5("Khan");
    Student s6("Khan");


    s1.display();
    s6.display();
    cout<<"Number Admission "<<Student::addNo<<endl;
}
```

Inner/Nested classes

We acan have clas within a class. Now the features of this is in this inner class we can access only the static data members of outter class.

However, the outter class can access all the data members of inner class that is puclic provided the object of inner class is created in the outter class.

```
class LinkedList
{
    class Node
    {
        int data;
        Node *next;
    };
    Node *Head;
    =
    =
    =
    =
};
```



This concept helps in creation of a data type that is lnnked lise you can see the way how it is used here above.

```
class Outer
{
public:
    ① int a=10;
    ② static int b;

    void func()
    {
        i.show();
        cout<<i.x;
    }

    class Inner
    {
    public:
        int x=25;
        void show()
        {
            cout<<b;
        }
    };
    ③ Inner i;
};
int outer::b=20;
```

Exception Handling

Problem may arise in the program at run time like phle do jo error hai wo to tuimhari glti hai programmer ki you not worked properly but at run time like apne kya kiya int dalna tha string dal diya or internet connection ke bina program nhi chlega aur apne kya kiya ki connection nhi diya to ye run time pe error dega and that is the exceptions at which program will not run. But hme ye nhi karna chahiye ki as a proggramer ki usne glt dala to uski problem hmne shi kiya na we must give him some feedback jisse wo user apni galti ko asani se smjh ske aur use program mai implement kare. This is the concept of exception handling.

Error

Programmer ki galti

1. Syntax Error — Compiler → Program likhna to sikh le yar
2. Logical Error — Debugger → Conncept pta nhi aur implement kar diya
   - Like accending order mai lgana tha par kuch hua kuch nhi to apke logic glt hai

User ki galti

→ 3. Runtime Exception
   1. bad input
   2. problem with Resources

Exception Handling

```
int main()
{
    int a=10, b=0, c;

    try
    {
        if(b==0)
            throw 101;
        c=a/b;
        cout<<c;
    }
    catch(int e)
    {
        cout<<"Division by zero"<<"erro code"<<e;
    }
    cout<<"Bye";
}
```

try and catch block somewhat same as if else if any error in try then will jump to catch block but used try and catch is used in exception handling. So now how it is used in the program that:

1. we can write a particular code in try block and also mention a specific condition that may cause error.
2. If that codition is found then we throw that there is an error via throw keyword. aab throw kw sath 101 likho ya 501 no matter it is just the error code type java mai ye jarurat nhi parti hai wese. Wo apne aap throw kar deta hai.
3. Now throw kiya apne jo bhi kiya hai throw wo as value pass hogi catch block mai aur user ko khega ki tumne ye glti to kar di hai dhyan do tki he become aware ki aage se use kya nhi karna hai.

Exception Handling

```
int main()
{
    int a=10, b=0, c;

    try
    {
1.      c=division(a,b);
x       cout<<c;
    }
    catch(int e)
    {
2.      cout<<"Division by zero"<<"erro code"<<e;
    }
3.  Cout<<"Bye";
```

```
int division(int x, int y)
{
    if(y==0)
        throw 1;
    return x/y;
}
```

So actually try and catch ki jgh if else se bhi to km ho jata to ye laye hi kyo to actual mai kya hai ki when you execute a program via function and program is supposed to return some values then in that case if an error is encountered it will throw some error and as programer your code must be ready to actch that error.

So this is why we actutally have try and catch actually even more than this is in this concept. Will learn as per you use it later.

Exception handling

- Exceptions are Runtime errors
- Try and catch blacks are used for handling exceptions
- If exceptions are not handled then program may crash
- Exceptions must give a message to the user, giving correct reason on cause of exception
-
- A try block can have Multiple catch blocks
- Catch-All can catch all exception
- Catch-All must be a last block
- If classes in inheritance are used in catch block then child class should come first

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

int division(int a,int b)throw(int)
{
    if(b==0)
        throw 1;
    return a/b;
}

int main()
{
    int x=10,y=2,z;

    try
    {
        z=division(x,y);
        cout<<z<<endl;
    }
    catch(int e)
    {
        cout<<"Division by zero "<<e<<endl;
    }
    cout<<"Bye"<<endl;
}
```

Exception Handling

```
class MyException : public exception
{

};
int division(int x, int y) throw( )
{
    if(y==0)
        throw 1;
    return x/y;
}
```

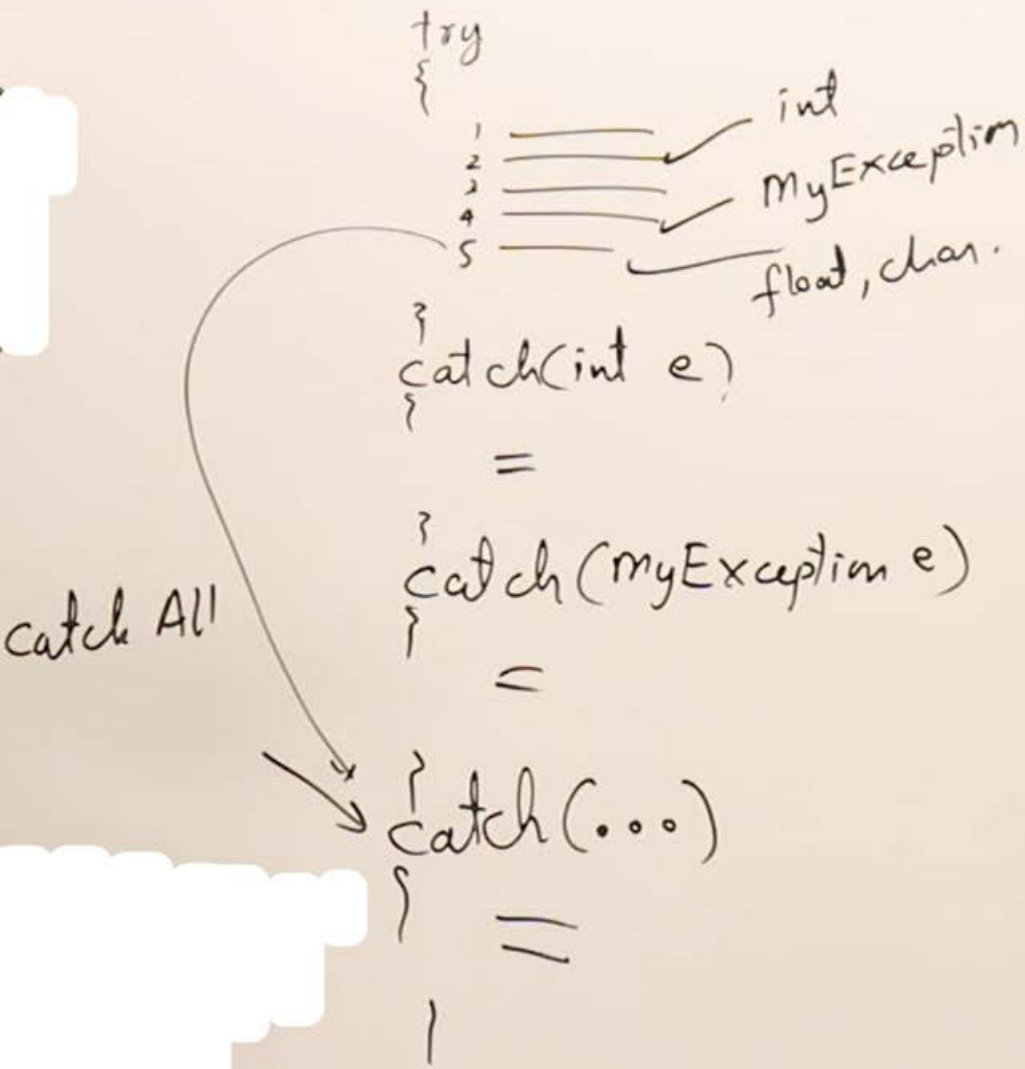
A function can throw any data type in exception. Like a class my exception that is derrived publically from a predefined class of c++ that is exception.

You can write with the function if it throw some or exception or not but it is totally optional ki likhna hai to likho nhi to nhi likho aur likhna chaho to kaise likho as shown in this code.

Ab wha ( ) jo hai usme wo btana prega ki kya throw karne wle hai like here 1 throw ho rha hai to mention as throw (int) and same for other types for class as well if you are throwing class.

Now one last thing if you wrote throw ( ) then it means ki apke function mai koi throw nhi hai and you can not throw anything in that case.

Exception Handling



We can have multiple catch statements. It is actually practical as well ki error code ye aaya to ye msg show hona chahiye ye aaya to ye.
To aab as shown here ye akr skte ho ho multiple error handling option try block mai jo bhi throw hoga uske according catch execute ho jayega.

Now ek bat ye ki sabse last wla catch jo hai usme hai (...) mtlb iska ki it can handle all the exception thrown out of try block. To in case apne int ka aur class type ka catch define kar diya par apne flaot char ak to kiya hi nhi to wo sab handle ye (...) wla block karega.

Aur wk aur bat ye (...) wka hmesa last mai hi aayega agr if you write first catch as catch (...) to sare whi chle jayenge and phir apka onjective jo hai ki is error mai ye is mai wo aaye wo nhi pura ho skta hai.

Exception Handling

Nesting of try and catch block can also be done as shown here.

```
try
{
    1 ——
    2 ——
    try
    {
        3 ——
        4 ——
    }
    catch(——)
    {
    }
}
catch( )
{
}
```

Exception Handling

```
class MyException1
{
};
class MyException2 : public MyException1
{
};
```

```
try
{
    ≡≡≡
}
catch(MyException2 e)
{
  —
}
catch(MyException1 e)
{
  —
}
```

Now in this apne kya kiya ki class ko derive kiya hai aab apne phir try mai error throw kiya to yad rkho hmesa phle catch child class ka define hota hai then uske bad parent calss ka kar skte ho catch define if needed.

Program to Demonstrate user defined Exception

```
#include<iostream>
using namespace std;
class StackOverFlow:exception{};
class StackUnderFlow:exception{};
class Stack
{
private:
    int *stk;
    int top=-1;
    int size;
public:
    Stack(int sz)
    {
        size=sz;
        stk=new int[size];
    }
    void push(int x)
    {
        if(top==size-1)
            throw StackOverFlow();
        top++;
        stk[top]=x;
    }
    int pop()
    {
        if(top==-1)
            throw StackUnderFlow();
        return stk[top--];
    }
};
int main()
{
    Stack s(5);
```

```
    s.push(2);
    s.push(3);
    s.push(4);
    s.push(10);
    s.push(9);
    s.push(8);

}
```

Templates

Template is a genearlised approach we make for one and use for all as you can see here we have templete to find larger number then now after writing this you can comapre anything like float double even char as they are also comparable. You can pass anything as long they are comparable it will give you the the result that is maximum of two

```
template <class T>
T maximum(T x, T y)
{
    return x>y?x:y;
}

maximum(10,15);
"  (12.5,9.5);
```

Templates

```
template<class T, class R>

void add(T x, R y)
{
    cout<< x + y;
}
```

int → T, double → R

You can make templete even for multiple data types all in one just make classes different and here you are to achieve the objective.

```
add(10, 12.9);
```

Templates

```
template <class T>
class stack
{
  private:
    T  s[10];
    int top;
  public:
    void push(T x);
    T  pop();
};
```

```
template<class T>
void stack<T>::push(T x)
{
  ==
}
```

```
template<class T>
T stack<T>::pop()
{
  ==
}
```

```
stack<int> S;
stack<float> S2;
```

Template class in this like hme stack data type bnana hai par as per need alg alg data ka jarurat prta hai like hme stack float ka bhi int ka bhi string ka bhi jaroorat par skt ahai to good way is to create template. Now this is done as shown here above also if you donot decleare the function inside the class and you decided to make it outside then in that case we always need to mention templete<class T> and also follow the same syntax as shown here in this code.

Now finally to create the object of that stck you have to write the intialisation as shown.

Template functions and classes are generic classes

They can work for any type of data
Classes can support different type of data

```
template<class T>
class Stack
{
private:
    T *stk;
    int top;
    int size;
public:
    Stack(int sz)
    {
        size=sz;
        top=-1;
        stk=new T[size];
    }
    void push(T x);
    T pop();
};

template<class T>
void Stack<T>::push(T x)
{
    if(top==size-1)
        cout<<"Stack is Full";
    else
    {
        top++;
        stk[top]=x;
    }
}

template<class T>
T Stack<T>::pop()
{
    T x=0;
    if(top==-1)
        cout<<"Stack is Empty"<<endl;
    else
    {
        x=stk[top];
        top--;
    }
    return x;
}

int main()
{
    Stack<float> s(10);
    s.push(10);
    s.push(23);
    s.push(33);
}
```

Constant

Define is also a method to define some variable value to a fixed thing globally and ithis value can also never be changed throughout the program. But this is advised that if you are making something globally constant than only use this otherwise for local go ahead with constant keyword.

```
#define x 10

int main()
{
  const int x=10;
  x++;
  cout<<x;
```

x

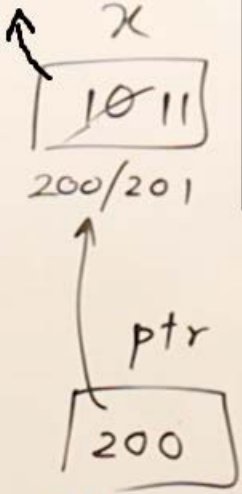
10

200/201

If you defined any thing as constant than in that case no chance throughout the program that we can further udate it. Like here ++x can never be executed.

Constant

```
#define x 10

int main()
{
    int x = 10;
    // This is pointer to a constant.
    const int *ptr = &x;
    // This can also be writeen as "int const *ptr=&x" same meaning of both
    X ++*ptr; // will never execute this and while run will show error.
    cout<<x; ——— 10

    cout<< *ptr ; // will print 10
```



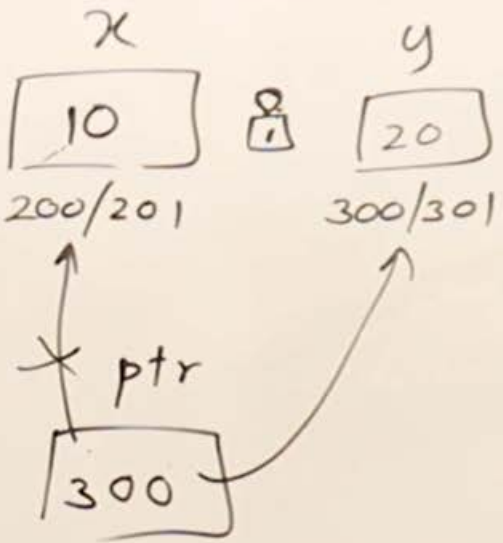
No change allowed x will remain 10

A pointer is constant for a given data type then in that case we can never update the value stored in that address where this pointer is pointing. As here you can not modify the value of x at all.

One more thing defining anyhting constant via define will not consume memory at all and also that will not be a part of compiler. Wo defining preprocessor ke part pe hoga and isse compiler ka lena dena nhi.

Constant

```
#define x 10

int main()
{
    int x=10;
int const *ptr=&x;
    int y=20;
    ptr=&y;
X ++(*ptr);
✓ ++y;
```

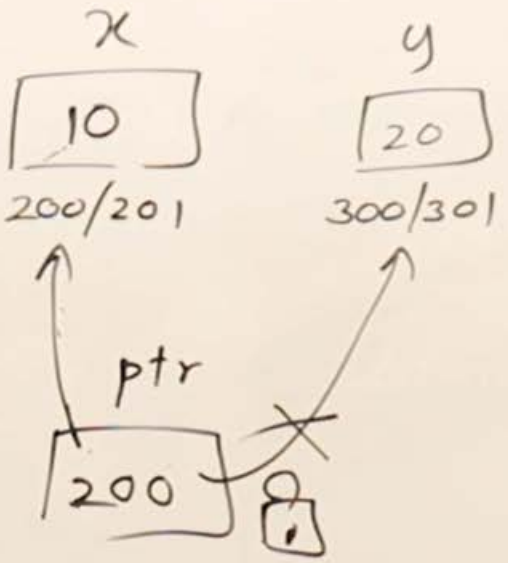


You can also use this method of decleration will work same as before in all aspects here this slide is added to add one more concept that is ki you can change the pointer pointing on another variable one thing to keep in miond ki wha bhi tum value update nhi kar skte bas will show you error.

Haa tum value update karna hi chahte ho to sidehe ++y kar lo wo update kar dega but through ++(*ptr) can not be done.

Constant

```
#define x 10

int main()
{
    int x=10;
int * const ptr=&x;
    int y=20;
→ ptr=&y;
    ++(*ptr);
```

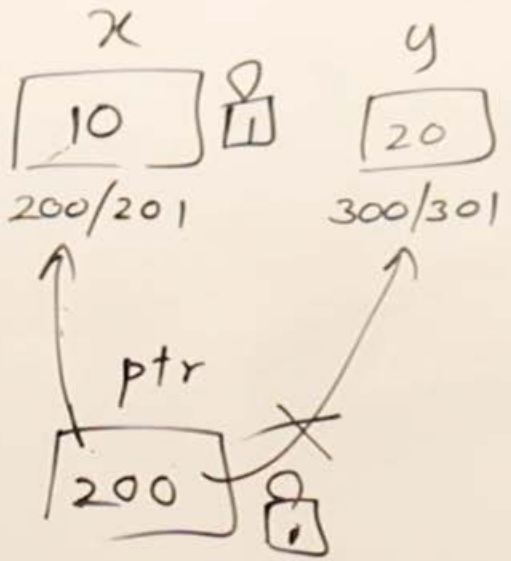


Now in this ptr is constant it means ki whatever memory location it is poinitng now in decleration can not be updated.

But yes here ++(*ptr) can be executed and it will update the value also.

Constant

```
#define x 10

int main()
{
    int x=10;
cont int * const ptr=&x;
    int y=20;
    ptr=&y;
    ++(*ptr);
```




```
x ++(*ptr)
ptr=&y;
```

In this type of initialisation we can not either update the data through ++(*ptr) and also we can not change the pointer to another memory location both will show error.

Constant

```
class Demo
{
  public:
     int x=10;
     int y=20;
     void Display() const
     {
      x x++;
        cout<<x<<" "<<y<<endl;
     }
};
int main()
{
    Demo d;
    d.Display();
}
```

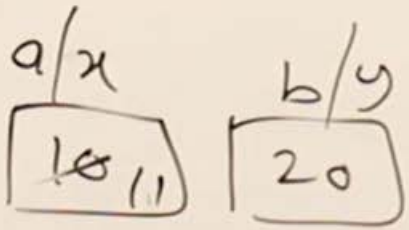
So if you are sure ki ye function koi bhi hai mera aur wo kuch bhi mere variable ko glti se bhi update nhi karega to tum function ke end mai const lga do as shown here it will show error agr phir koi bhi value update ho rhi hogi us function ke andr.

as you can see hm chahte hai bas display to usme update ka kya kam par ho gai glti se update to uska tarika hai ki const likh diya end mai to wo error show karr ke bta dega ki apne glti se update kar diya hai malik

—— 11 20

Constant

```
void fun(const int &x, int &y)
{
    x++;
    cout<<x<<" "<<y;        // 20
}

int main()
{
    int a=10, b=20;
    fun(a, b);
}
```



So now here is ki apne call by reference se copy kar liya aur function ne update bhi kar diya value par aap chahte nhi ki wo is value ko update kare to aap us parameter ko const karke pass karo ki kuch bhi ho is variable ko update mat karna.

Preprocessor Directives / Macros

define pi 3.1425 jah bhi pi dikhega wo use 3.1425 se replace kar dega turant

define SQR(x) (x * x) Jha khi SQR(x) dikha wo use turant x*x se replace

define MSG(x) #x Jha khi bhi MSG(x) dikha to jo bhi x mai hai usse replace

"x"

define c cout //mtlb ye hua ki jha khi bhi isne dekha c wo use replace kar dega cout se

```
int main()
{
```

cout << ~~SQR(5)~~;
5*5

cout << ~~MSG(Hello)~~;
"Hello"

define ka mtlb hi hai ki ek bar bta diya aage jha khi bhi dekho tum is string ko use sidhe replace kar do us string ke aage jo define karte wkt likha tha us se.

c << pi; -> will print 3.14 that is replace pi with 3.14 and c with cout.

Preprocessor Directives/Macros

```
#ifndef
    #define PI 3.1425
#endif
```

Now this is a kind of condtiotion ki agr apne ise phle define kiya hai to ye dubara nhi karega define isi ko and if nhi kiya to define kar dega

PreProcessor Directives

- They are instructions to compiler
- They are processed before compilation
- They are used for defining symbolic constant
- They are used for defining functions
- They also support conditional definition

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

#define max(x,y) (x>y?x:y)
#define msg(x) #x

#ifndef PI
    #define PI 3.1425
#endif
int main()
{
    cout<<PI;
    cout<<max(10,12)<<endl;
    cout<<msg(hello)<<endl;
}
```

Namespaces

```
namespace First
{
    void fun()
    {
        cout << "First";
    }
}

namespace second
{
    void fun()
    {
        cout << "second";
    }
}

using namespace First;
int main()
{
    fun();
    second::fun();
}
```

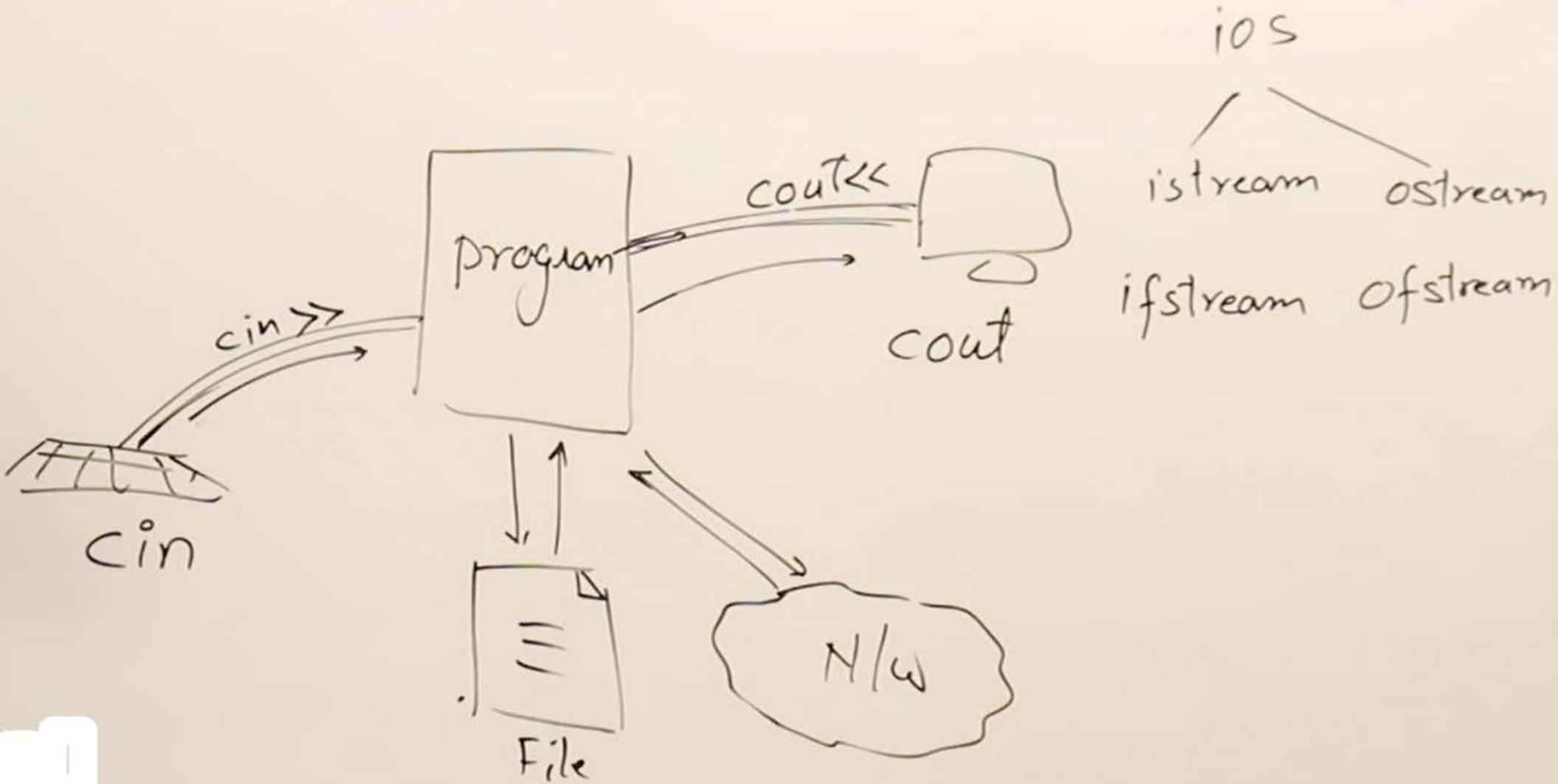
Kabhi kabhi ye hota hai ki you need fucntion with same name and then apne call kar diya to pta kaise chle ki kisko call karna hai to uske liye choice hai ki aap namespace aur koi name dal ko us chij ko ek mai define kar do not only function usme fucntion object classs sab bna skte ho same nam ka jha bhi conflict ho aur phir jha bhi use karna hai to wo use name aur sath mai name :: scope reolution ke sath use kar lo to wo usko call karega.

Ab ek aur bat ki aapne using namespace First jasie yha kar diya to bydefault wo nameconflict jha bhi hal wo usme first wla hi clock use karega and if you want ki nhi second aab yha use karna hai to second:: mention karke use karo.

Streams

I/O

To handle data in and out in a program either from keyboard or form a network or sending a data to screen or file or network we have streams in c++. It allows us to access and operate over data for input or output purpose.



File Handling

```
#include<fstream>
```

header file that contains input and outputstream class that will be writting data in your file.

```
int main()
{
    ofstream outfile("My.txt");
    outfile<<"Hello"<<endl;
    outfile<<25<<endl;
    outfile.close();
}
```

My.txt

Hello
25

Two modes are there to open file

1. ios::app -> this will continue into file mtlb phle ka delete nhi karega.

2. ios::trunc -> This will delete all the data of the file and start from beginning. By defuault ye hi hota hai

Always close file if work is done otherwise data loss may appear.

File Handling

How to read data from a file. code for this.

```
#include<fstream>

int main()
{
    ifstream infile;
    infile.open("My-Txt");
    string str;
    int x;
    infile >>str;
    infile >> x;
    cout<<str<<" "<<x;
    if( infile.eof()) cout<<"end of file reached";
    infile.close();
}
```

My-Txt

```
Hello
25
```

one more code that if you want to check weather the file you opened is opened properly or not you can do it by two syntax.

1. if(!infile) it will return true and it means file is not opened.
2. if(infile.is_open()) it will return true if file is open and false if not open.

Streams

- I/O Streams are used for input and output data to and from the program
- C++ provides class for accessing input output operations
- Iostream is a base class for all classes
- Istream is for input
- Cin is the object of istream
- ostream is for output
- Cout is an object of ostream
-
- ifstream is a file input stream
- ofstream is a file output stream

Writing in a File

```
int main()
{
    ofstream of("Test.txt",ios::trunc);
    of<<"John"<<endl;
    of<<25<<endl;
    of<<"CS"<<endl;

    of.close();

}
```

Reading from File

```
int main()
{
    ifstream ifs("Test.txt");
    string name;
    int roll;
    string branch;

    ifs>>name>>roll>>branch;
    cout<<name<<endl<<roll<<endl<<branch<<endl;
    ifs.close();

}
```

Serialization

- Serialization is a process of string and retrieving state of an object
- Class must have a default constructor

```
class Student
{
private:
    string name;
    int roll;
    string branch;
public:
    Student(){}
    Student(string n,int r,string b)
    {
        name=n;
        roll=r;
        branch=b;
    }
    friend ofstream & operator<<(ofstream &ofs,Student s);
    friend ifstream & operator>>(ifstream &ifs,Student &s);
    friend ostream & operator<<(ostream &os,Student &s);
};
ofstream & operator<<(ofstream &ofs,Student s)
{
    ofs<<s.name<<endl;
    ofs<<s.roll<<endl;
    ofs<<s.branch<<endl;
    return ofs;
}
ifstream & operator>>(ifstream &ifs,Student &s)
{
    ifs>>s.name;
    ifs>>s.roll;
    ifs>>s.branch;
    return ifs;
}
ostream & operator<<(ostream &os,Student &s)
{
    os<<"Name "<<s.name<<endl;
    os<<"Roll "<<s.roll<<endl;
    os<<"Branch "<<s.branch<<endl;
    return os;
}
int main()
{
    ofstream ofs("Test.txt");
    Student s1("John",10,"CS");
    ofs<<s1;
    ofs.close();
    Student s2;
    ifstream ifs("Test.txt");
    ifs>>s1;
    cout<<s1;

}
```

File Handling

mode ios::binary
read( ) operation to read
write( ) operation to write

Text

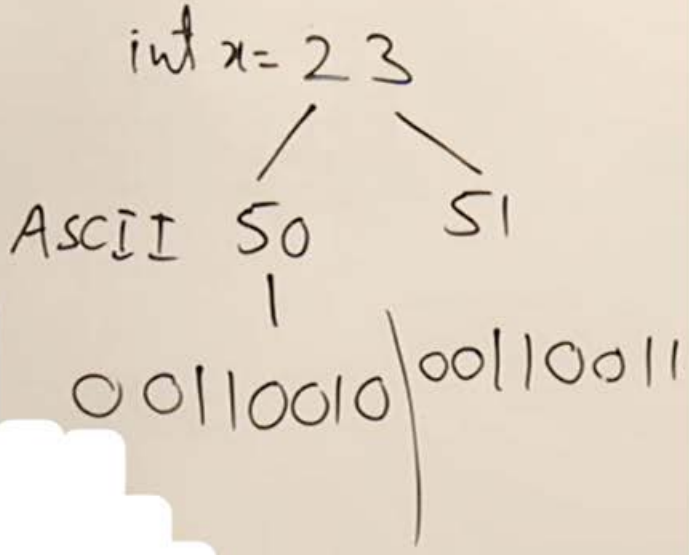
00110010
00110011

Binary

0000000000010111

My.txt

Hello
25



This is answer to your previous question asal mai you must know ki keyboard ke har input ke liye ek ascii code hota hai and phir wo usi form mai machine ko btaya jata hai. even apne text mai store kiya hai 23 to in that case you have two things 2 and other is 3 2 is represented by 50 in asciii and 50 is something 00110010 to har ek string ka 8 mit mai conversion hota hai aur phir wo computer mai store hota hai. Ab text file kya karta hai ki ye har bar 8 bit ke set ko read karta hai when we open and uska jo text equivalent hai wo bta deta hai to ye isliye 23 ko 23 mai dekh skte hai par aab binary mai kya hai ki wo 16 bit mai store karta hai sab and if not that long data to most significant bits ko 00000000 kar deta hai aab tum kya karte the ki binary ko text mai kholne ki kosis to text to apna hai rule ki 8 bits ko padhunga uska jo bhi ascii mai ho uske equivalent symbol de dunga to ye apko wi symbol de deta tha to ye kuch question marks ya emoji type ka symbol hota hai to hme whi dikhta hai aur tum sochte the ki binary hai to 0100 wla form kyo nhi hai bhaiya to hoga kaise tumne khla text mai hai jiska rule hi hai ki 8 bit padho aur uska equivalent smbol bharo.
Now notable thing binary mode is fast and occupy less space whi text to har symbol ke liye apna 8 bit lega hi ye iska hai ki 23 ko store karna hai to 23 ka binary nikalo aur kar lo store.

Manipulators

These are used to format the output endl ho gya next line ke liye same way mai hmare pas hai different formators. like for integer data types hex hai to ye jo bhi data ahi int mai if written as below to wo uska hexadeximal form dega. same for fixed it will fix the value of float koi change nhi kar skta isme wo 125.731 hi aayega. ws is used for white space. and many more.

```
cout << endl;
cout << "\n";

cout << hex << 163;
                A3
cout << fixed << 125-731;
cout << set(10) << "Hello";
cout << 10 << ws << 20;
```

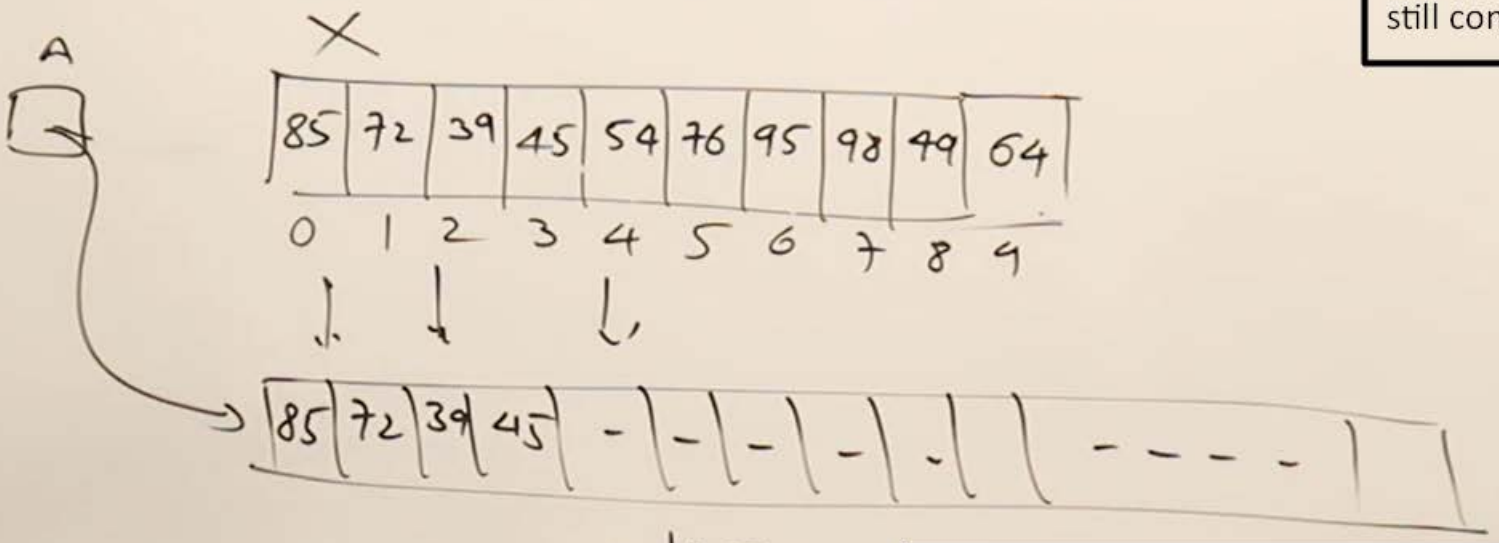
integer	other
hex	set( )
oct	left
dec	right
Float	ws
fixed	
scientific	

STL

Standard template library

Data Structure

int A[10]; → This will create memory in stack and if we need more to kuch nhi kiya ja skta. to overcome heap is thora bhut useful but still complex.



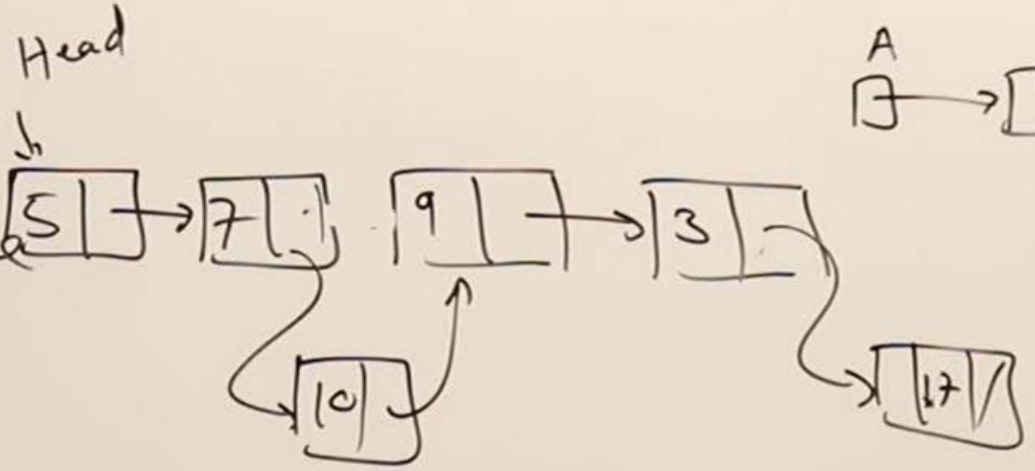
int *A = new int[10]; → It will create memory for 10 int type values in heap dynamically.

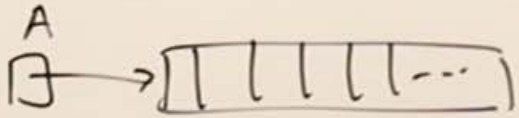
A = new int[20]; → Now te problem is ki yar kbhi hme jada jarurat par jaye size ki to iska bhi upaye hai ki hm A ko dynamically hi ek naye array pe point kar de aur purane ki value ko isme copy kar ke purana delete kar de.

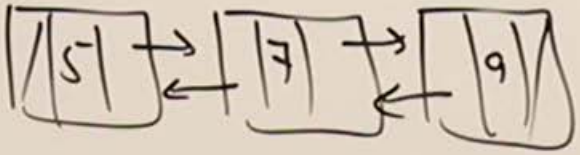
STL

Data Structure

stack
Queue
Deque
Priority_Queue
Map
Set







With orevious two data type these stack, queue, dqueue, priority queue, map, set linked list or doubly linked list these are some more types of data strucuture that we can use in our program as per need.

Now STL standdard template library have already some predefined data structures in some header files library and we can include these library called standard template libraray in our program to use that data structure in the program.

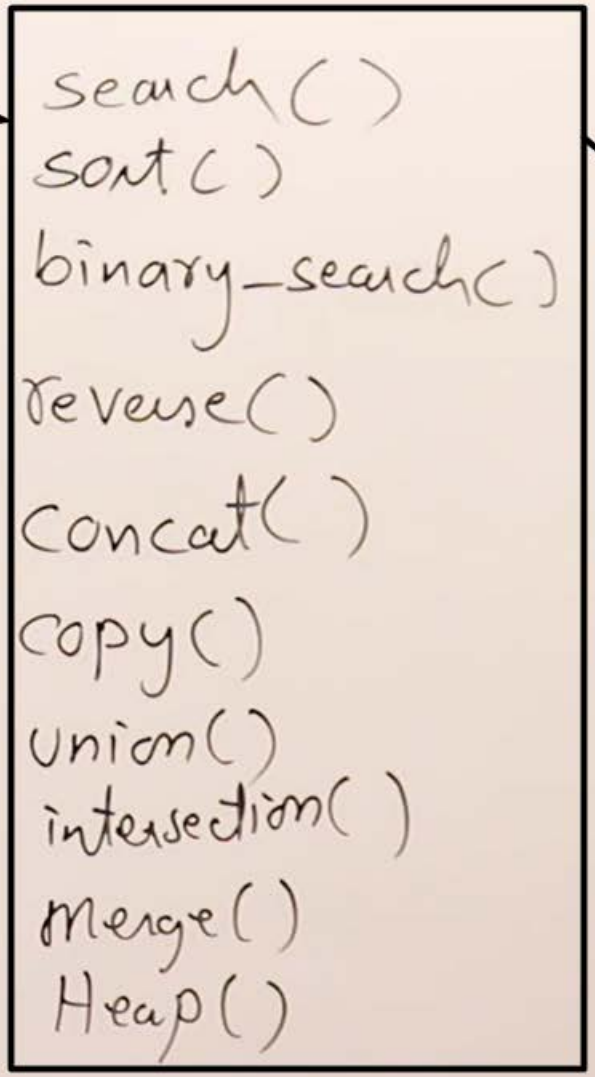
STL

In stl we have algo, container and iterartors. Algo are some predefined functions that can perform a few task if required library is included and also the proper function to implement that in the program.

Algorithms

Containers

Iterators.



List of a few algorithm that we may use in our program and there are many more to it explore them.

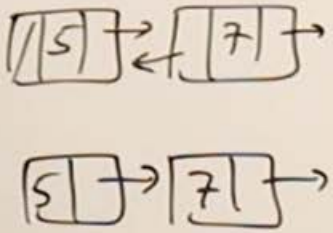
Containers are the thing like data structure jisme aap vector bhi aab pdhoge and you can access these containers with the help of iterators.

STL

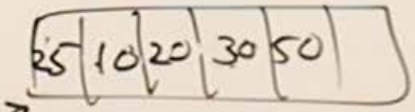
Template

By default vector ka size 16 hota hai

- vector
- list
- forward_list
- deque
- priority_queue
- stack
- set
- Multiset



push_front()
pop_front()
push_back()
pop_back()
insert()
remove()
size()
empty()
front()
back()



1. Vector -> kind of array bas it is dynamic ki size and insertion can be done using functions no need to worry for arranging. Isme sare function mentioned chlte hai bas push_front(), pop_front(), [redacted] inhe chor ke.
2. list -> doubly linked list that is aage ka bhi pta hai piche ka bhi and ye sare operation jo yha likhe hai isme ho skte hai.
3. forward _lise -> singly linked list again aage ki trf hi bas dekhna hai to ye better hai list ke comparision mai kyoki size iska kam hoga doubly ke comparision mai.
4. deque -> a kind of array hi hai par isme one thing is ki hme allowed hai insertion aage se ho ya piche se both way possible. and also 2,3,4 inme sare function yha jo likhe hai chlte hai.

STL

Template

- vector
- list
- forward_list
- deque.
- priority_queue
- stack LIFO
- set ——— unique
- Multiset

5. priority_queue -> Max hep is this and behave as this only and as in max heap jo max hota hai wo top pe aa jata hai same yha hai wo aab is max pe aap niche likhe sare operation kar skte ho and again jo max hai wo phir se top pe aa jayega. can do all mentiond operations here.

6. stack -> works on lifo principle and can perfrom all the below operations mentioned here.

push( )
pop( )
empty( )
size( )

5.1. queue -> ye normal queue hai jiska priciple hai kam karne ka FIFO. It is same way work as priority key but the thing is ki isme duplicate key allowed hai aur ye FIFO pe chlta hai. Par priority_queue wo max heap ke concept pe chlta hai and hence max on top and no duplicate allowed.

7. set -> this is like math wla set you can store data in it and important thing is ki duplicate nhi ho skta koi agr hoga to wo ise nhi lega and same operation ke sath kam hoga uspe bhi jo upr likhe hai.
8. multiset -> same as set sare operations allowed hai bas ek difference hai ki isme duplicates are also allowed.

STL

Template.

Note: ye dhyan mai rkhna ki ye sab template hai mtlb it can be used with any data type like vector of int float double string kuch bhi ho skta hai you only need to write them with proper syntax ya say iterators jo abhi dekhenge hm.

- vector
- list
- forward_list
- deque.
- priority_queue
- Stack LIFO

Hash Table.

Map / MultiMap

<key; value> pair

key value

1 "John"

2 "Ajay"

3 "Khan"

set ——— unique

Multiset ———

9. map -> it is hash table type chij ki key ke correspond mai value hai ya python ka dictonary kah lo. again isme hai ek key aur value and no key can be duplicated.
10. -> same as above bas difference is ki aapko allowed hai duplicate key rkhna par haa same key value pair nhi hona chahiye like 1 "john" and dubara 1"john" hi kar do this is not allowed 1 "john" hai to 1"som" ye hona hi chaiye same nhi chlega.

STL

```
#include <vector>

main()
{
    vector<int> v={10,20,40,90};
    v.push_back(25);
    v.push_back(70);
    vector<int>::iterator itr;
    for(itr=v.begin(); itr!=v.end(); itr++)
        cout<<*itr;
```

A few more functions that work with all the container it actually helps in retriving and working up on the stored data.
begin( ) -> will give you to be at starting of the stored data in that container
end ( ) -> will let you to be at the end of the the data stored in container.
rbegin() ->agr reverse direction mai traverse karna hai to ye last ko phla bna ke dega apko.
rend() -> isme apko jo actual begin hai wo reversed end ban ke milega to aap data pe reverse direction mai bhi work kar skte ho

// here iterator hai ek iski madad se ap data ko via this iterator object le skte ho aur can work upon that data.

Same way aap jha jha vector hai wha dqueue likho wo us type ka data structure ban jayega also library mai dqueue same way list ho ya forward_list replace only vector with these will work for you as function are of same name. even you can work with set as well but set mai insert() se data jata hai to wha wo function use karna hoga. Later to but iterator same way work karta hai aur even you can show using for each loop like for (int x:v) esa kuch kar ke.

```
using namespace std;

float fun()
{
    return 2.34f;
}
int main()
{
    auto y=fun();
    double d=12.3;
    int i=9;
    auto x=2*d+i;
    cout<<x;
}
```

auto -> this allows user to get out of the hectic job ki data type konsa hai konsa doon data type no worry write auto compiler will take care of rest.

// To yha kya hua ki hme pta nhi tha ki koi function konsa value return karta hai to we said ki jo bhi kare hm to auto kh rhe hai data type ko jo bhi type usi ko tum smjho

// auto keyword allows you ki tum data ka type mat likho compiler apne aap pta kar lega

```cpp
#include <iostream>

using namespace std;

int main()
{
    int x=10;
    float y=90.5;

    decltype(y) z=12.3;
}
```

decltype(variable_name) -> using this manlo ki hme ek naya variable bnana hai par us naye wle ka data type kisi purane variable ke jaiasa hi hona chahiye to simply say decltype(purane_wale_variable_ka_nam) aur naya jo variable bna na hai wo likh do ye kam kar dega apka

```cpp
#include <iostream>

using namespace std;
class Parent
{
    virtual void show() final
    {

    }

};

class Child:Parent
{
    void show()
    {

    }
```

Agr chahte ho ki aapek function apne base class mai bnaya aur phir aage use koi override na kar ske to use final keyword with the given syntax. and also to make any function final we need to make that function virtual first.

Declaration of 'show' overrides a 'final' function

Lambd Expressions

way of writing a function jb koi nam nhi dena ho function ko man kiya sidhe likh dete hai to ye way hai proper ways to write and execute them are shown below.

```
[capture_list](parameter_list) -> returntype { body };
```

if bracket mai value bhi pass ki hai to mtlb ki apne function define kiya aur call bhi kiya hai whi.
is wle ko dekfo to is funciion ka koi nam nhi hai sidhe parameter list aur phir body retturn type c++ apne aap dekh leta hai.

```
main()
{
auto f=[ ]( ){ cout<<"Hello"; };
```

To isme kya kiya hai ki hmne function unnamed bnaya par auto kar ke use ek variable mai store kar diya aur jab kkbhi bhi jarurat lgi use call kar liya as shown down. yha dhyan rkhna ki end mai bracket nhi likha hai agr likha hota mtlb call bhi kiya gya hai as you can see in next two.

```
[ ](int x,int y){ cout<<"sum:"<<x+y;}(10,5);
int x=[ ](int x,int y){ return x+y;}(10,5);
```

isme same phle aur dusre ka mix hai mtlb ki call bhi kiya par store krwa liya us result ko x mai now x = 15 hai use karlo jha karna hai again yha return type mention nhi kiya hai mtlb ki c++ ko khud return type pta chl jayega.

```
f();
int a = [ ] (int x, int y) -> int { return x - y }(10, 5);
}
```

ye pura rule wla hai ki hmne koi nam bhi nhi diya parameter list bhi btai phir return type is lamda function ka btaya aur final mai body likhi ki karta kya hai aur phir ( ) mai do value pass kar call bhi kar liya and here a mai store hogya 5.

Lambd Expressions

```
[capture_list](parameter_list)->returntype{ body };
```

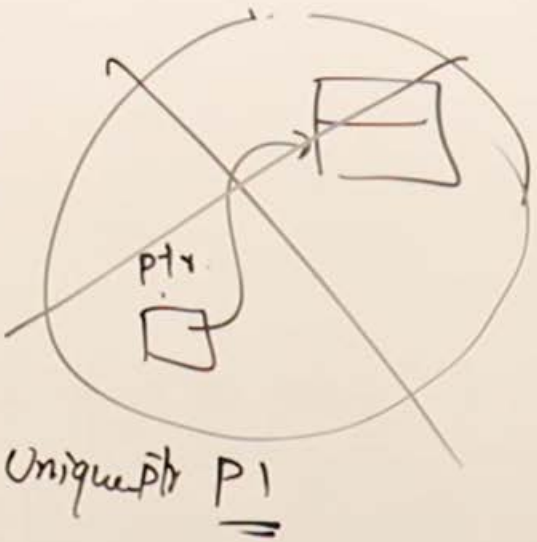
```
main()
{
    int a=10;
    int b=5;
    [ & ]( ){cout<<++a<<" "<<++b;}( );
    [a, b]( ){ cout<<++a<<" " << ++b;}( );
}
```

if you write _in capture list then This will allow you to use all the variable in the scope to be used in this lamda expression. and even you can update them.

if you write the container list like this than you can never do any change in a or b like as you did here ++a or ++b with this mode of passing you can only access value of a and b. If you want to update them as well for that case you must write container list as follow: [a, b] () {cout<<++a<<++b;}();

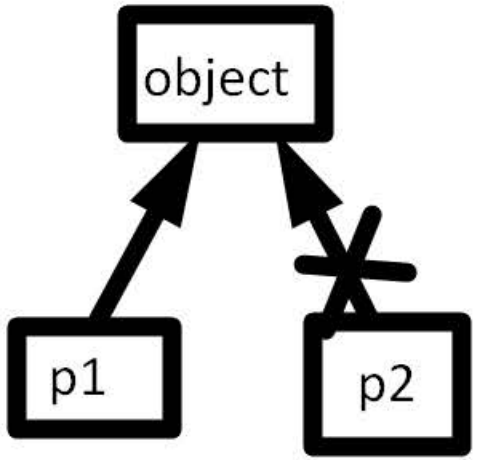
Smart Pointers

```
fun()
{
    unique_ptr<Rectangle> p1(new Rectangle(10,5));
    cout << p1->area();
    cout << p1->perimeter();
}
main()
{
    while(1)
    {
        fun();
    }
}
```



so actually problem with normal pointer ki we have to delete it aur nhi kiya to memory leak wla case aa skta hai to overcome this c++ give us some smart pointer jo ki out of scope agr object hua to apne ap hi us object ki memory ko delete kar denge. and syntax to use it is shown above in this program ans as fun get's executed it will delete this rectangle object and hence delete pointer p1

unique pointer

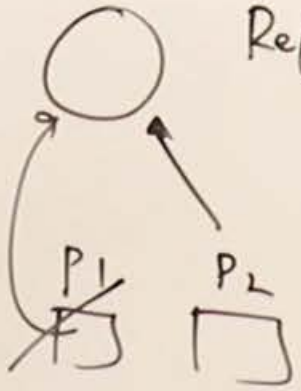


if a pointer is pointing on an object than abother pointer can not point over the same object provided that p1 is an unique_pointer.

Smart Pointers

pointers are basically used to acess the data that are outside the program like in heap or say the data that is dynamic we use pointer to acess them. in general if I say among three unique_pointer, shared_pointer and weak_pointer the useful one is unique_pointer baki rest bhi hai depends upon use of thier.

shared_ptr



If two of the pointer is pointing on a same object then it is possible that wither it is a weak_pointer or it is a shared_pointer the difference in the two is weak_pointer donot keep the the record that how many pointers are pointing over that object while in shared_pointer we have a ref_counter that holds the number of pointers pointing to that object and also we can find number of such pointer using use_count();

Ellipsis

- Used for defining variable argument functions
- … is used as symbol of ellipsis
- Printf and scanf of C language are example of ellipsis

```
int sum(int n,...)
{
    va_list list;
    va_start(list,n);

    int x;
    int s=0;
    for(int i=0;i<n;i++)
    {
        x=va_arg(list,int);
        s+=x;
    }
    return s;
}

int main()
{
    cout<<sum(3,10,20,30)<<endl;
    cout<<sum(5,1,2,3,4,5)<<endl;
}
```