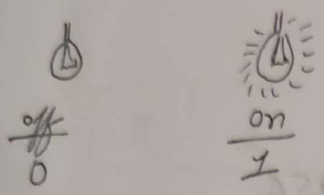
Fundamentals

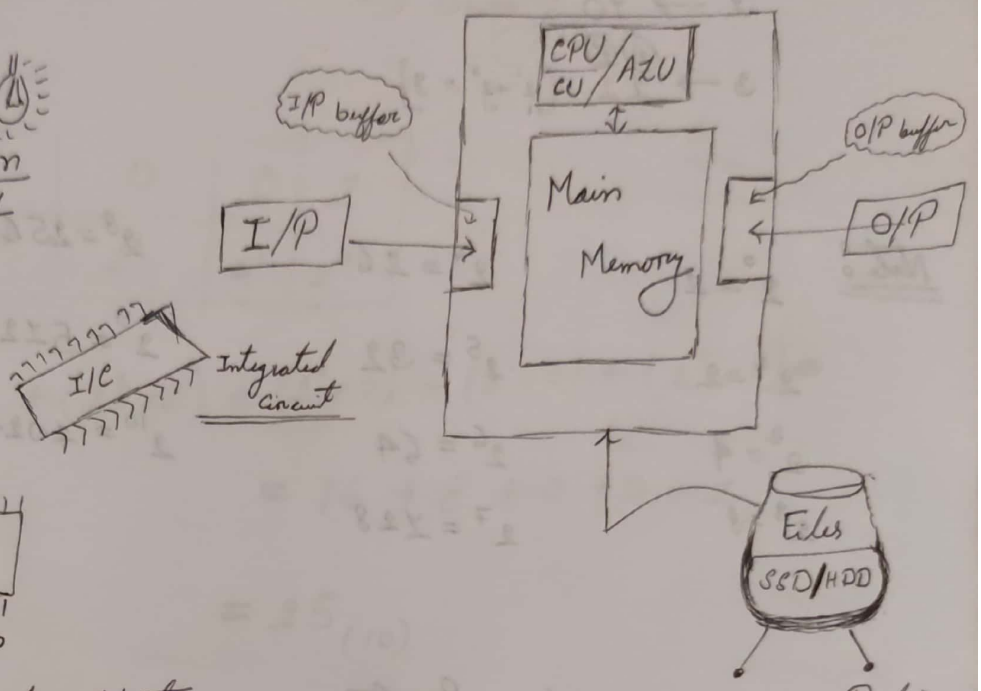
How Computers think?

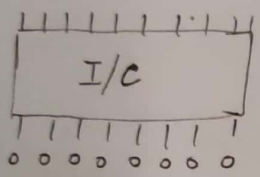
Low → 0

High → 1

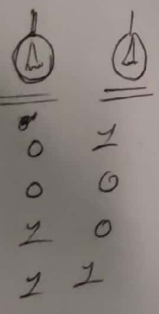


Block Diagram

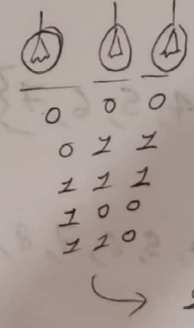




8-bit = 1 byte



$\hookrightarrow 2^2$ outcomes



$\hookrightarrow 2^3$ outcomes

8-bits

$\Rightarrow$ Total number of outcomes $= 2^8$

Decimal $\{0, 1, 2 \ldots\ldots 9\}$

Binary

$1 \rightarrow 01$

$2 \rightarrow 10$ (2^1)

$3 \rightarrow 11$ $(2^1, 2^0)$ $\{2^1 + 2^0 = 3\}$

$4 \rightarrow 100$

Note:

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

$2^7 = 128$

$2^8 = 256$

$2^9 = 512$

$2^{10} = 1024$

Number Systems

${}_2$Binary $= \{0, 1\}$

${}_8$Octal $= \{0, 1, 2, 3, 4, 5, 6, 7\}$

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

${}_{16}$Hexadecimal $= \{0, \ldots\ldots\ldots; 9, A, B, C, D, E, F\}$

$= 11001_{(2)}$

2 | 25
2 | 12 — 1
2 | 6 — 0
2 | 3 — 0
2 | 1 — 1
0 → 1

Decimal
↳ Binary

1	0	0	1
2^3	2^2	2^1	2^0

$(2^4 \times 1) + (2^3 \times 1) + (2^2 \times 0) + (2^1 \times 0) + (2^0 \times 1)$

$= 16 + 8 + 0 + 0 + 1$

$= 25_{(10)}$

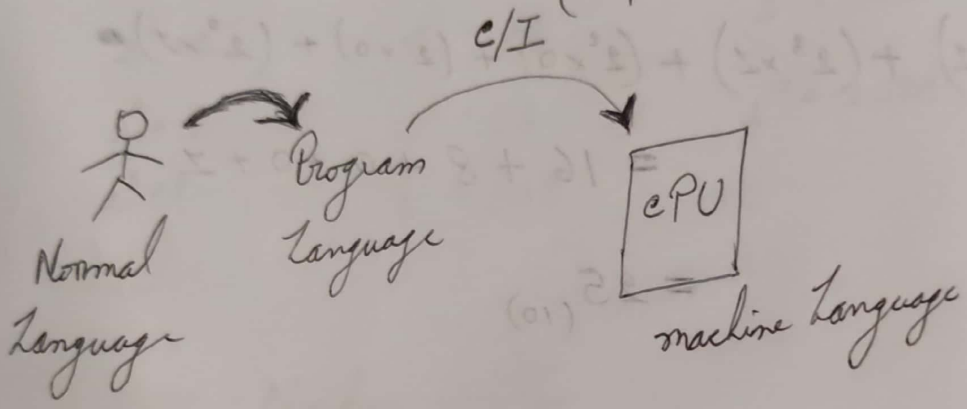
↳ Decimal

What is a Program?

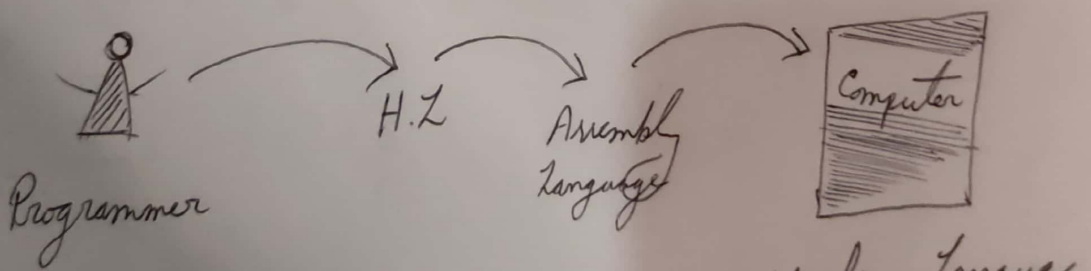
> A Computer is a programable computational device.

Data

Instructions



Low-Level v/s High-Level Language



> Machine Language

> 00110100

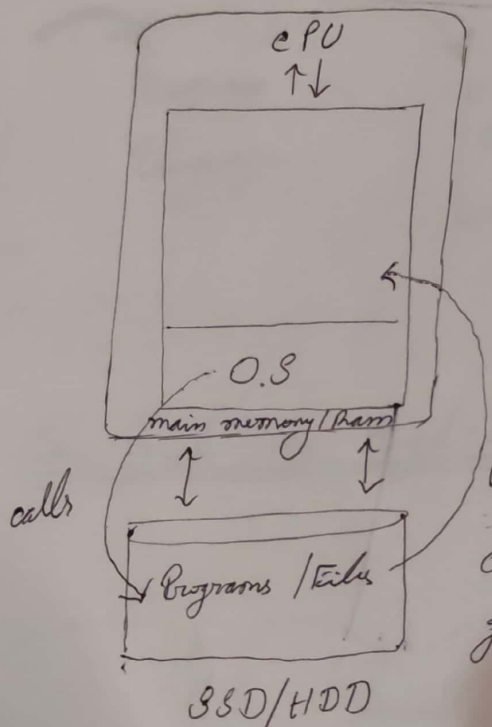
> Binary

Operating System

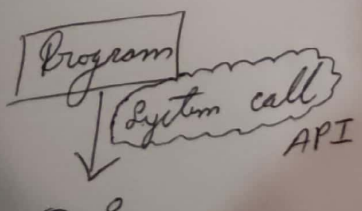
- O.S is a master program which will take all the things under control & provide management.

→ As soon as the system starts – O.S starts with it.

→ O.S gets programs executed ↓



→ To get access to Hardware a Program should call O.S to get access



• O.S manages all the resources → Software + hardware of an operating System & Programs uses the resources with the help of O.S.

Programming Paradigm / Styles

1. Monolithic Programming
2. Modular / Procedural programming
3. Object Oriented programming
4. Aspect oriented programming / Component assembly "

Monolithic
Basic

C-language

Procedural / Modular

```
Func 1 ()
{ ---
  ---
  ---
}
func 2()
{ ---
  ---
  ---
}
:
main()
{ func 1()
  func 2()
  func 3()
}
```

Modular
structural Info

```
{ Data 1
  Data 2
  Data 3
}
func 1(info)
{ ---
  ---
  ---
}
func 2(info)
{ ---
  ---
  ---
}
:
main()
{ struct Info i;
  func 1 (i)
  func 2 (i),
}
```

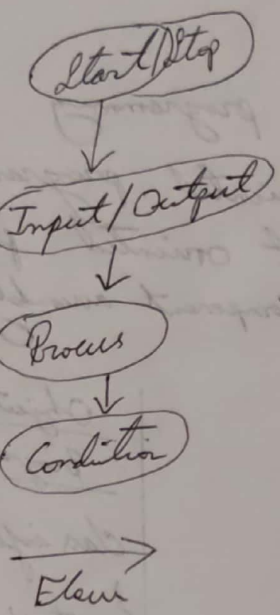
Object Oriented Programming

```
class info
{ data 1();
  data 2();
  data 3();
  func 1()
  { ---
    ---
    ---
  }
  func 2()
  { ---
    ---
    ---
  }
};
main()
{ info i;
  i. func 1();
  i. func 2();
  :
}
```

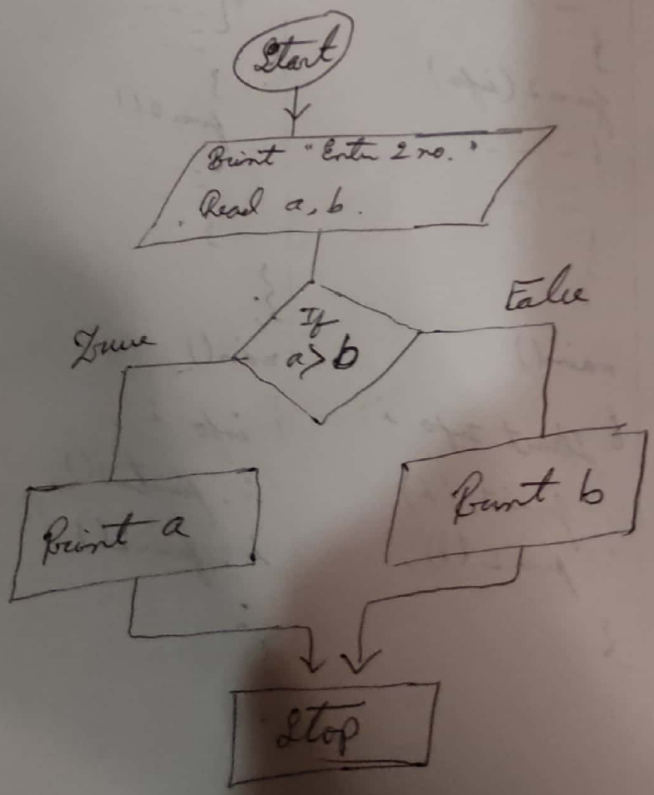
Syntax = Programing Grammar

Flow Charts

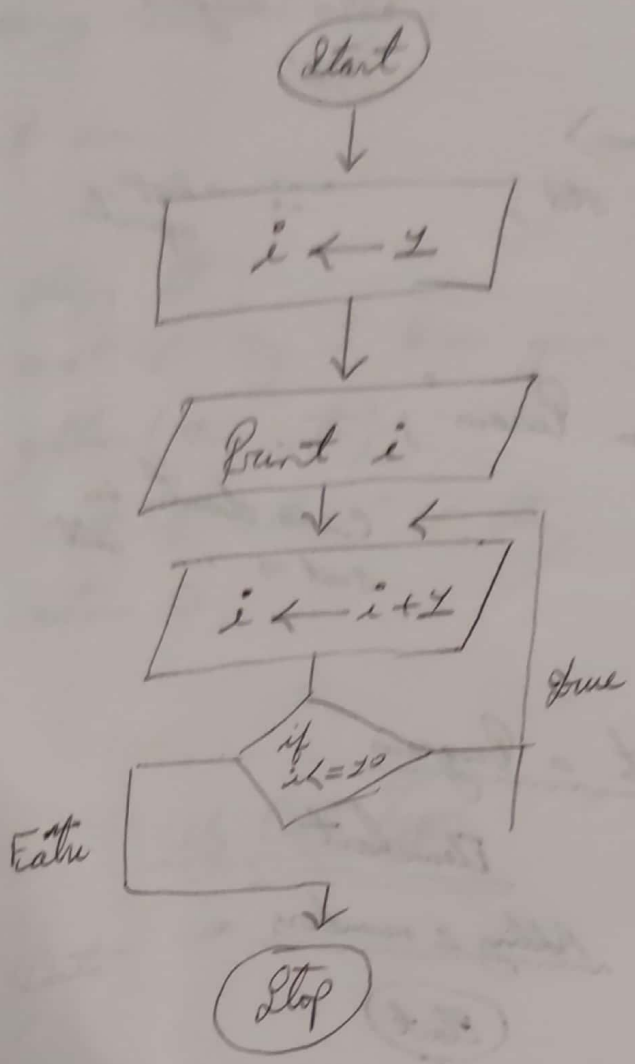
Showing the flow / step of a program is going.



Input → Process → Output



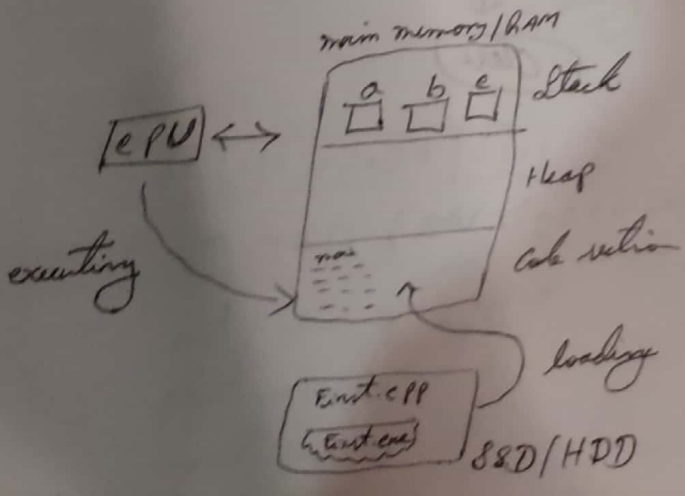
1] Print no. from 1 to 10



22

Steps involved in Development & Execution

1. Editing (writing the code)
2. Compiling (Exe.cpp → Exe.exe)
3. Linking (Libraries)
4. Loading
5. Executing



First Program

"<<" insertion operator
"=" assigned
">>"

First.cpp

```
#include <iostream>
using namespace std;
int main()
{
  cout << "Hellow Param";
  return 0;
}
```

:: → Scope operator

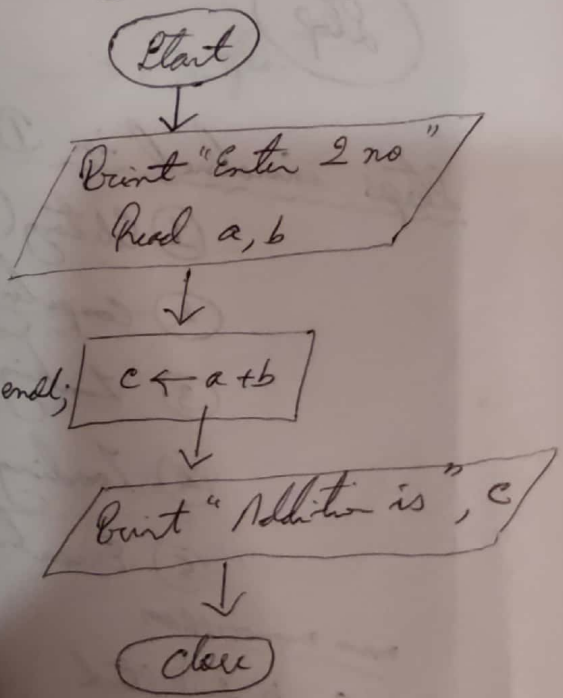
cin → character in
cout → " out.

How to write a Program

```
#include <iostream>
using namespace std;
int main()
{   int a, b, c;
  cout << "Enter 2 no.";
  cin >> a >> b;
    c = a+b
cout << "ADDition is" << c << endl;
  return 0;
}
```

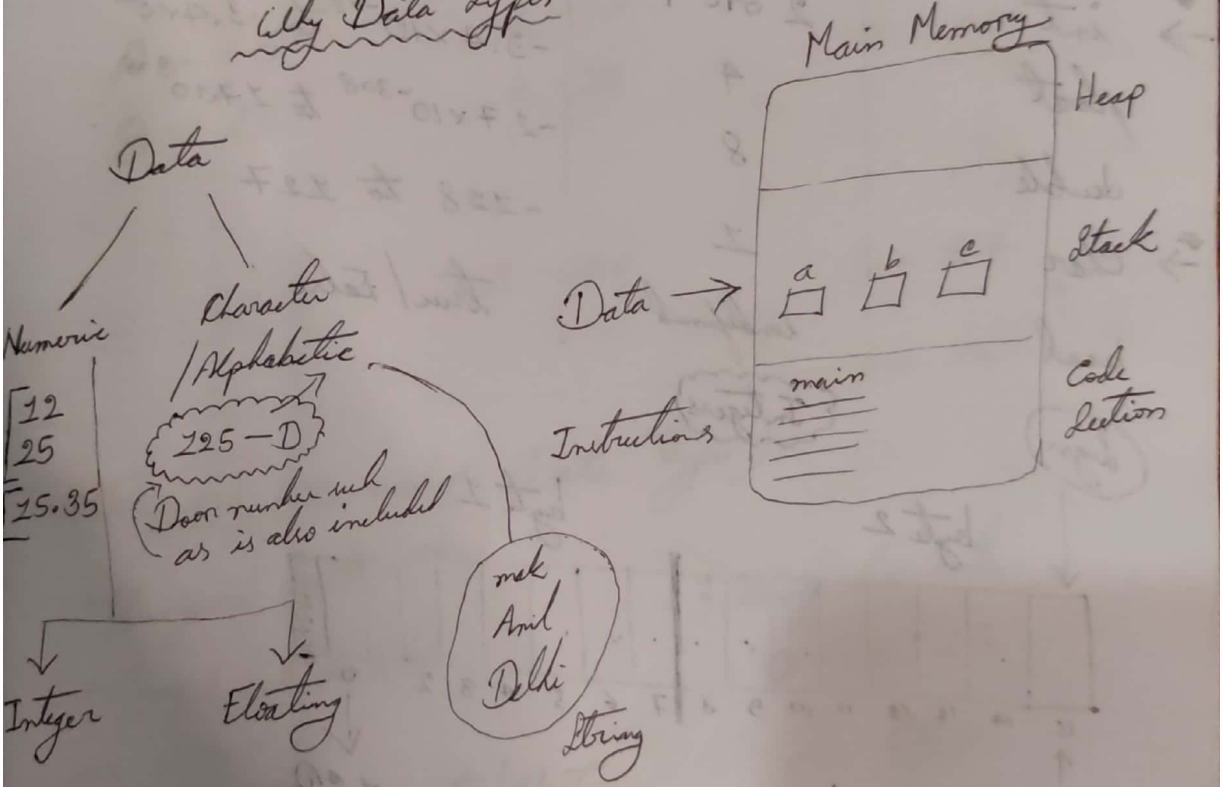
Flowchart

Adding 2 numbers



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

int main()
{
    string str;
    cout << "May I know your name? ";
    getline(cin, str);
    cout << "Hello " << str;
    return 0;
}
```

My Data Types



1 byte | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |

0 - 255 (total $2^8 = 256$ combinations)

int ⟶ byte?

float ⟶ ?

char ⟶ ?

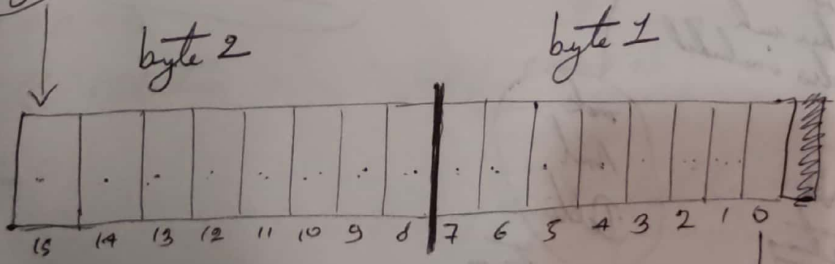
Data Types & Variables

- Primitive
  - Integral
    - Bool
    - Char
  - Floating Point
    - Float
    - Double
- User Defined
  - Enum
  - Structure
  - Union
  - class
- Derived
  - Array
  - Pointer
  - Reference

Data type	Size (Bytes)	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.

Integers

Sign



MSB [Most Significant Bit]

1 (-ve)

0 (+ve)

LSB [Least Significant Bit]

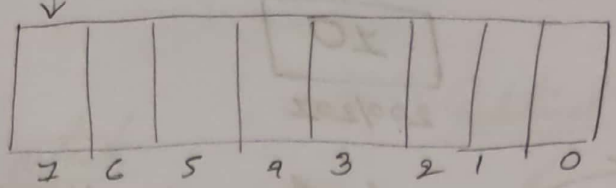
no. of possibilities $(2^{15} = 32768)$ ← Next 15 remaining

• M.S.B declares sign of the number (+ve/-ve).

→ Character (1 byte)

-2^{n-1} to $+2^{n-1}-1$

Sign (+ve/-ve)



$2^7 = 128$

(0 - 127)

-128 to 127

ASCII (American Standard Code for Information Interchange)

A - 65

B - 66

⋮

Z - 90

a - 97

b - 98

⋮

z - 122

0 - 48

1 - 49

⋮

9 - 57

Modifiers (To modify Data types)

- Unsigned (int, char)
- Long (int, double)

long int - 4 bytes / 8 bytes

long double - 20 bytes

1] Unsigned int

0 - 65535

1] Unsigned char

0 - 255

25. Variables:

```
int main()
{
    int rollno;
    rollno = 10;
}
```

rollno.
10
200/201

int rollno; ← Declaration of a variable
(int = type, rollno = variable)

rollno = 10; ← Initialization of variable.

or do both;

####

```
int main()
{
    int rollno = 10;
    char group = 'A';
    float price = 12.75f;
```

Rules:

int x1; ✓
int 1x; ✗

int rollno; ✓
int roll no; ✗
int roll_no; ✓
int rollNo ✓
int RollNo ✓

Operators & Expressions

Arithmetic - "+" "-" "*" "/" "%"

+
-
* artith
/
% mod

Relational

<, <=, >, >=, ==

Logical

&&, ||, !

Bitwise

&, |, ~, ^

Increment/Decrement

++, --

Assignment

=

Eg:

```
int a, b, c;
a = 10;
b = 5;
```

① c = a + b;
(10 + 5)

② c = a - b;
(10 - 5)

③ c = a * b;
(10 × 5)

④ c = a / b;
(10/5)

⑤ c = a % b
(13 % b)

int a=13, b=5, c;

5)13(2
10
3

c = a/b
(13/5)

or

int

do to get actual value we should use float

eg:

```
int a=13, b=5;
float c;
c=(float) a/b;
```

(2.6)

We will get answer in "Decimal."

```
c = a%b
```

```
float a=13.5, b=4.1, c;
c=a/b;
c=a%b
```

"%" operator cannot be in float.

27

Expressions & Operator Precedence:

$x = a + \underset{③}{b} * \underset{①}{c} - \underset{④}{d} / \underset{②}{e}$

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

- Area of $\Delta = \frac{1}{2} bh$
- Perimeter of Rect. $= 2(l+b)$
- Sum of n-terms $= S = \frac{n(n+1)}{2}$
- n^{th} term of A.P. series

$$t = a + (n-1)d$$

$$\downarrow$$

$$t = a + (n-1) * d$$

(X) $a = 1/2$

cause 1 & 2 both are integers

$a = (b*h)/2$

$a = 0.5 * (b*h)$

$P = 2 * (l + b)$

(#include <cmath>)

- root of quad. eq^n

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

$$\hookrightarrow x = \frac{(-b + \underset{\text{split}}{\wedge}(b*b - 4*a*c))}{(2*a)}$$

- Displacement: $S = \frac{v^2 - u^2}{2a}$

$$\hookrightarrow S = \frac{(v*v - u*u)}{(2*a)}$$

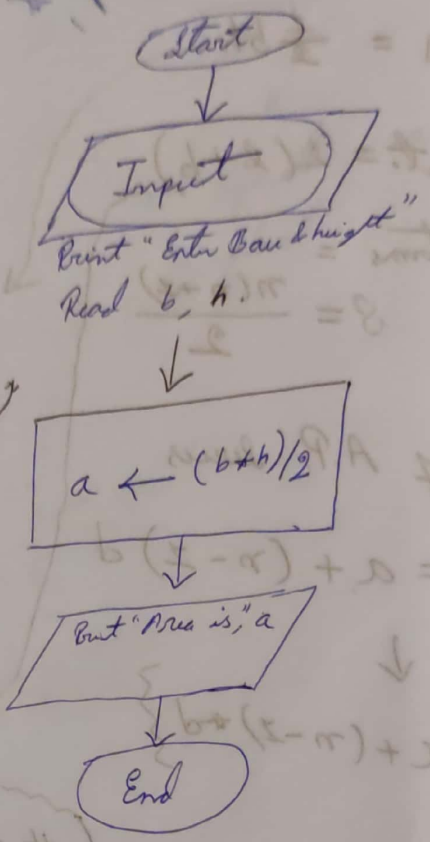
$$\hookrightarrow s = (Pow(v,2) - Pow(u,2)) / (2*a)$$

Program for Area of Triangle.

(Area = (b*h)/2)

Area — output (O/P); (b*h)/2 — input (I/P)

- Input
- Process
- Output



Algorithm

Begin
Print "Enter Base & height"
Read b, h.
a ← (b*h)/2
Print "Area is", a
End.

Program

```
# include <iostream>
using namespace std;
int main()
{ float b, h, a;
Cout << "Enter Base & Height";
Cin >> b >> h;
a = (b*h)/2;
Cout << "Area is : " << a;
}
```

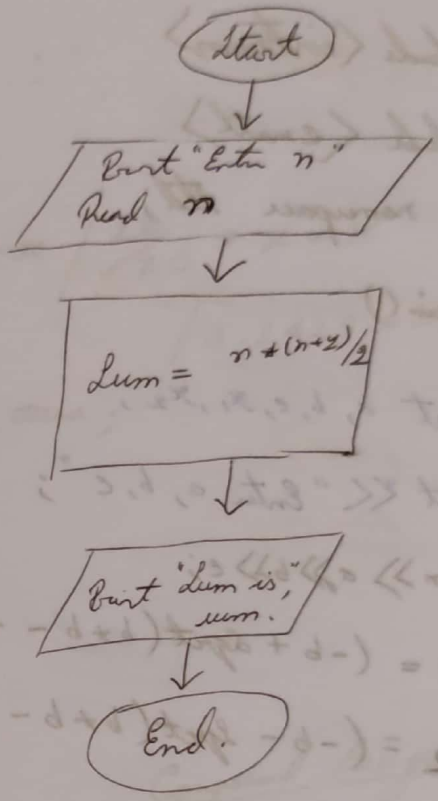
29. Finding the sum of first n Natural No.'s :

$$\underset{O/P}{Sum} = \underset{I/P}{n*(n+1)/2}$$

$$= 1 + 2 + 3 + \ldots\ldots (n-1) + n$$

Program

```
#include <iostream>
using namespace std;
int main().
{
    float n; sum;
    Cout << "Enter n ";
    cin >> n;
    Sum = n*(n+1)/2;
    Cout << Sum;
    return 0;
}
```

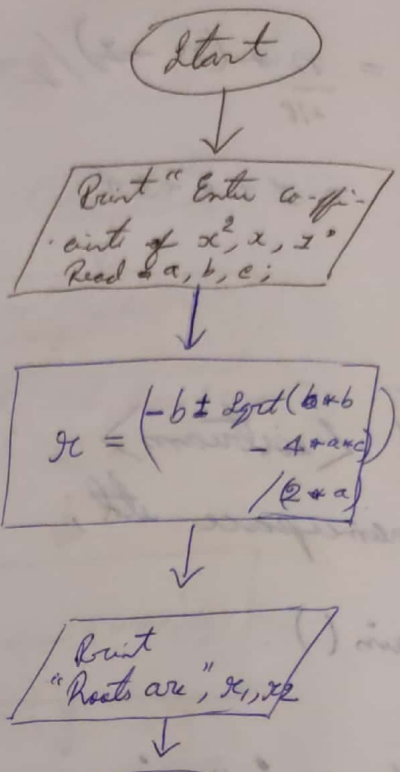


30.

Finding roots of quad eqn

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

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




```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    float a, b, c, x1, x2;
    cout << "Enter a, b, c";
    cin >> a >> b >> c;
    x1 = (-b + sqrt(b*b - 4*a*c))/(2*a)
    x2 = (-b - sqrt(b*b - 4*a*c))/(2*a)
    cout << "The roots x1 & x2 : " << x1 << "  " << x2;
    return 0;
}
```

31:

Area of Circle

```
#include <iostream>
using namespace std;
int main()
{
    float r, area;
    cout << "Enter the radius : ";
    cin >> r;
    area = ((float)(22/7)) * r * r / 22/7.0 * r * r
                                  / 3.1425 * r * r
    cout << "The area is : " << area;
    return 0;
}
```

32

```
int a=10 , b=5, c=15;
int Sum = 5;
[Sum = Sum+a+b+c;]
```

Sum = Sum + a
15 5 10

Sum = Sum + b
20 15 5

Sum = Sum + c
35 20 15

```
Sum += a;
Sum += b;
Sum += c;
```

Compound Assignment

+=
-=
*=
/=
%=

&=
|=
<<=
>>=

Sum is added with a & the result is stored at Sum.

∴ Prod = Prod * a * b * c

⇒ Prod *= a * b * c

33

Demo

```
# include <iostream>
using namespace std;
int main()
{
    int sum = 10, x = 5;
    sum += x;          ——→ (sum = sum + x;)
    cout << sum;
    return 0;
}
```

Output: 15

34.

int x

New,

eg:

34. # Increment Decrement Operator

Pre inc
++x;

Post inc
x++

Post dec
x--

Pre dec
--x

```
int x=5, y;
```

✗ x++;

✗ ++x;

means adding "1" to x in both cases.

But we don't use it.

x=6
y=6

y = ++x;
(2) (1)
5+1=6

Note:
First increament & then assign it to y.

Now,

y = x++;
(1) (2)

y=5
x=6

Note:
First assign & then increment.

eg:

```
int x=5, y=20, z;
```

z = x++ * y
(1)
(50) (5) × (20)

then x is incremented
∴ x=6.

Again,

z = ++x * y
(5+1)=6 × (10)
60

first increment → x=6
then multiply by (10)

```
#include <iostream>
using namespace std;
int main()
{
    int i=5, j;
    cout << i++ << " " << ++i;
    return 0;
}
```

i++ → 6 (i becomes 6 after printing)

++i → 6+1 = 7

Output:
5 7

```
{ int i=5, j;
  j = i++;
  cout << i << " " << j;
  return 0;
}
```

Output: 6 5

```
{ int i=5, j;
  j = ++i
  cout << i << " " << j;
```

Output: 6 6

Note: Multiple unsequenced increment operator should not be used with 1 variable.

```
{ int i=5, j;
  j = 2 * ++i + 2 * i++;
```

++i ① → 12

i++ → 12 ✗

cout <<

It will run but will depend on the compiler, how it will run.

36. ++

Overflow

Range

char x = 127 → -128 to 127

Sign

	1	1	1	1	1	1	1	1
x	0	1	1	1	1	1	1	1
	7	6	5	4	3	2	1	0

Dec → Binary

2 = 10

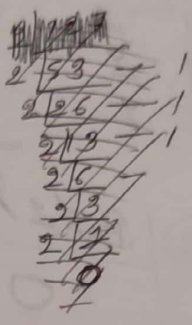
0 → +ve

1 → -ve

+1

1 0 0 0 0 0 0 0 — (-128)

128 64 32 16 8 4 2 1



2 | 127
2 | 63 — 1
2 | 31 — 1
2 | 15 — 1
2 | 7 — 1
2 | 3 — 1
2 | 1 — 1
0 — 1

If a number is overflown → it moves in a cyclic way.

Eg: 127 (+1) → -128 ✓

```
#include <iostream>
using namespace std;
int main()
{
  char x=127;
  x++;
  cout << (int)x;
  return 0;
}
```

output: -128

38. **Bitwise Operator**

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
0	1	0
1	0	0
1	1	1

Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
0	1	1
1	0	1
1	1	1

Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
0	1	1
1	0	1
1	1	0

&	and
|	or
^	x-or
~	not
<<	
>>	

2|7

Eg: int x = 11, y = 7, 3;

x = 0 0 0 0 1 0 1 1
y = 0 0 0 0 0 1 1 1

x & y = 0 0 0 0 0 0 1 1
x | y = 0 0 0 0 1 1 1 1
x ^ y = 0 0 0 0 1 1 0 0

Scanned with CamScanner

char x=5, y;

x → 00000101

y=~x → 11111010

↑ -ve number.

↓

11111010

00000101

+1

= 00000110

-(2²+2¹) = -6 ✓

→ (1) Take compliment

(2) Find Binary - Decimal

(3) add 1 to the number

eg: -(5+1) ✓

YT: "-ve" numbers in binary (Watch if you forgot)

Debopam Clock watch later

Or, (we Two's Compliment)

→ -256+128+64+32+16+8+0+2+0

= -6 ✓

40 Enum & Typedef

Enum : Enumeration Datatype
(User Defined 1)

Departments	Menu Enb	Days	Card, Colors / Shapes	Feedback
CS - 1	New - 0	Mon - 0	Clubs - 1	
Ece - 2	Open - 1	Tue - 1	Spade - 2	
ISE - 3	Save - 2	Wed - 2	Diam - 3	
CIVIL - 4	Close - 3	⋮		

Note: In programming if there are limited set of words very commonly used, we can assign codes for each of it & use it to save time.
(numeric values can represent those words)

```
const int CS = 1
      int ECE = 2
const  "
⋮
```

by default ↓ 0, 1, 2, 3, 4, 5, 6

```
enum day { mon, tue, wed, thu, fri, sat, sun }
     ↑
user defined datatype

int main()
{
    day d;
    d = mon;
    d = fri;
}
```

if (d == mon)

Xd = 0

enum dept { cs=1, ece, it, civil=6, ele }
2 3 7 1

Type def:

```
typedef int marks;
typedef int rollno;
int main()
{
    marks m1, m2, m3;
    rollno r1, r2, r3;
```

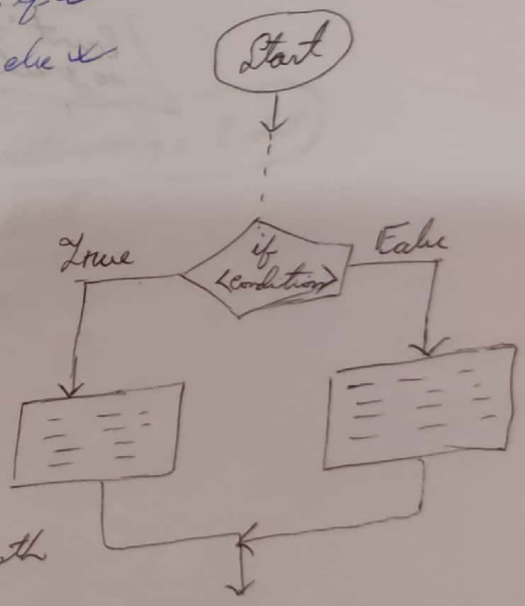
```
int main()
{
    int m1, m2, m3, r1, r2;
```

← to improve readability of a program.

Conditional Statements

True → if
False → else

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



True - 1 (or any nth integer)
False - 0

Relational

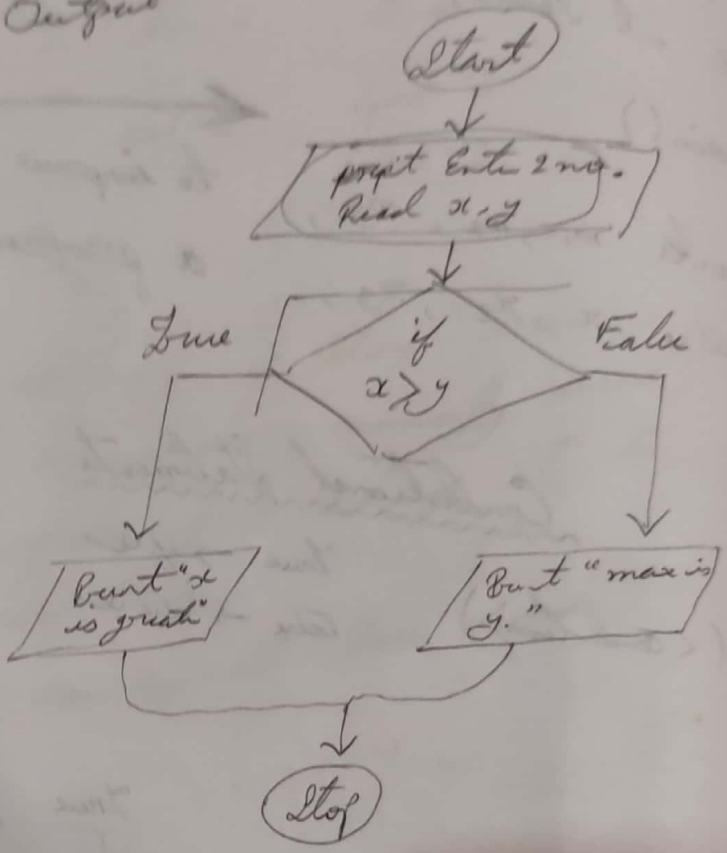
<
<=
>
>=
==
!=

a = 10
b = 15
a < b → true (1)
a == b → false (0)

1. Find maximum of 2 No.s

2 No.s → x (i/p) y (i/p)

- Input
- Process
- Output



42 Compound Conditional Statements

x<3

x>y

T-shirt	Cap
T	T
T	F
F	F
F	T

T-shirt && Cap

T
F
F
F

Logical Operators

&&	AND
||	OR
!	NOT

Relational Operators

<
>
<=
>=
!=

T-shirt || Cap

T-shirt	Cap
T	T
T	F
F	T
F	F

T
T
T
F

(He is not wearing a T-shirt)

(He is wearing a T-shirt)

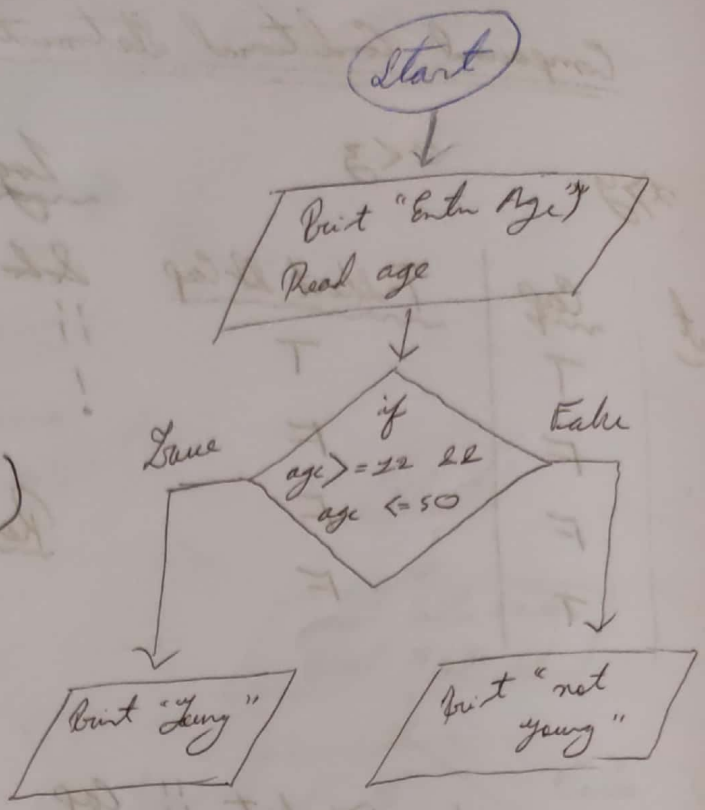
! T-shirt

T-shirt
T
F

F
T

```
int main()
{
  int age
```

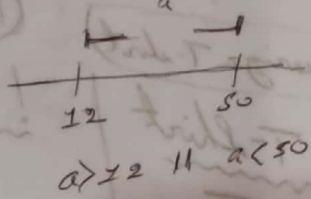
(Lec in V.S code)



AND
&&

OR
||

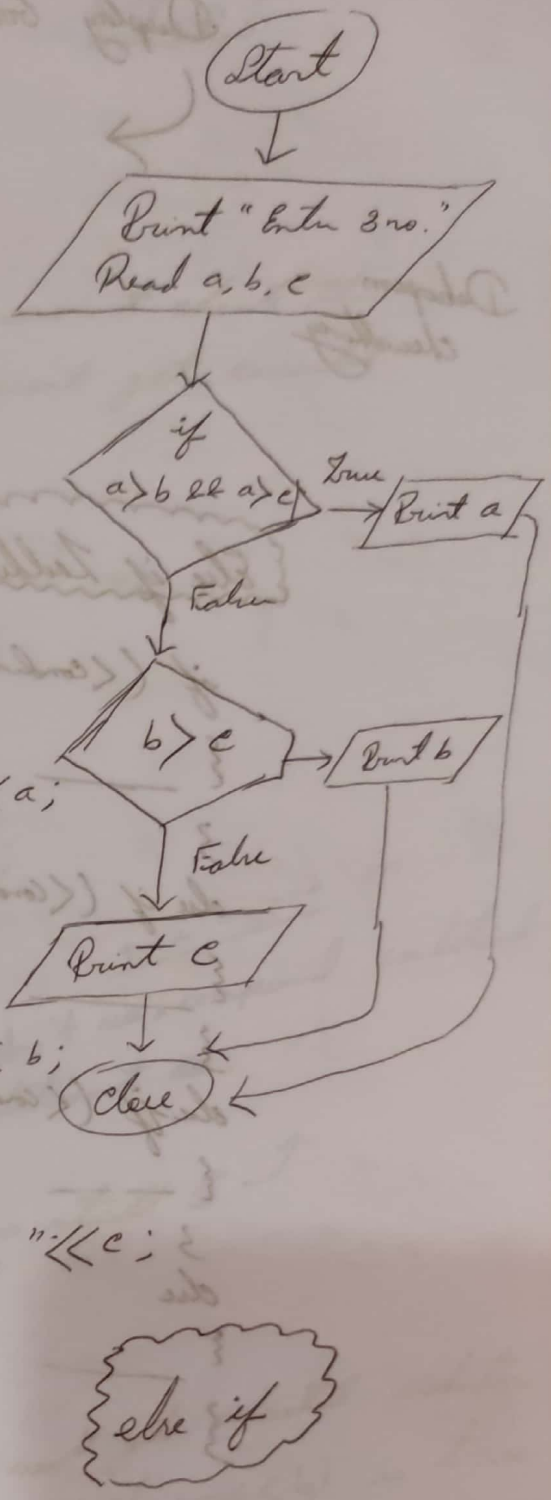
NOT
!



Nested if

Max of 3 numbers

```
int main()
{
    int a, b, c;
    cout << "Enter 3 no.";
    cin >> a >> b >> c;
    if ( a>b && a>c)
    { cout << "The greatest no" << a;
    }
    else if ( b>c)
    { cout << "The greatest no is" << b;
    }
    else
    { cout << "The greatest no. is " << c;
    }
    return 0;
}
```



else if

Display Grades

A → Avg >= 60

B → [redacted]
35 <= Avg < 60

C → Avg < 35

Debopam Chowdhury

Else if Ladder

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

58

Imp

Short Circuit

```
int a, b, i;
if (a>b && a>c)
```

a>b → F → whole if condition F → it won't check this, it won't get executed

```
if (a>b || a>c)
```

a>b → T → whole if condition T

So,

```
int a=15, b=7, i=5;
if (a<b && ++i<b)
cout<<"i";
else
cout<<i;
```

a<b → F

++i or --i in compound conditional statement → So; Don't use.

here i is still 5

again,

```
if (a>b && ++i<b)
```

a>b → T → It will get executed whether the whole (++i<b) is true or not.

So i = 5+1 = 6

~~###~~

```
int a=15, b=5, c=9;
if(a>b && ++c>a)
{
  cout << "Yaa \n";
}
cout << c;
```

(a>b is T; ++c will get executed)

Output
c = 10 {9+1}

```
if(a>b || ++c>a)
```

(a>b is T; ++c won't get executed)

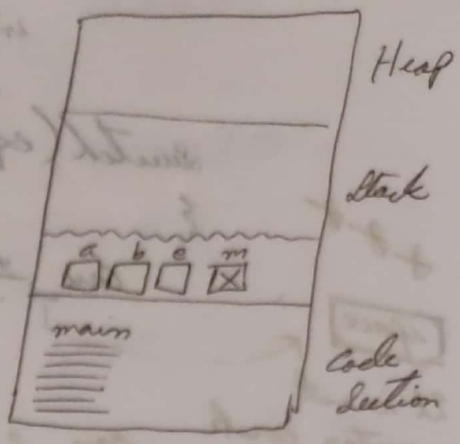
Output
c = 9

Note:
Never use "Increment Operator" in the second portion of a compound condition statement

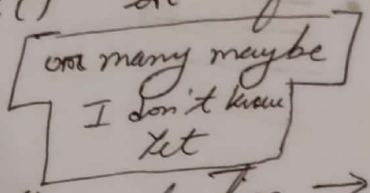
```
if(++i>b && b<c)   X
if(b<c && ++i>b)   X
```

60. # Dynamic Declaration

```
int main()
{
   int a, b, c;
   if(int m = 2, m<b)
   {
      ---------
   }
```



Note:
Here, if you declare a variable within the "if()" or "for()" section; after



completion of the function → the variable is removed from the memory.

eg:

```
for (int i=0, i<=0, i++)
{
   ---
   ---
   ---
}
cout << i;
```

(X) won't be executed because "i" is removed from the memory.

62

Switch Case

int / char

switch (expr)

{

case 1: _____

break;

case 2: _____

_____

break;

default: _____

_____

_____

}

Used in [Menu Options]

Eg:

File

New - 1

open - 2

save - 3

save as - 4

exit - 5

Eg:

Laptop

9) Switch (1026)
↳2
{
Case 1: cout<< "One";
" 2: " <<"two";
" 3: " <<"three";
}

Output: twothree

10 ↳ $2^4\ 2^3\ 2^2\ 2^1\ 2^0$
↑ ↑ ↑ ↑ ↑
0 1 0 1 0
↳ (1) 0 0 1 1 0
0 0 0 1 0
= 2

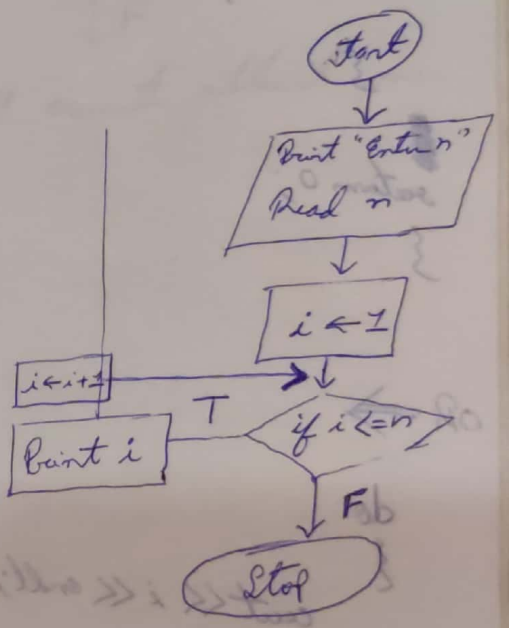
Loops

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

① While

n = 20 , i = 1

Print i	i ← i+2
1	2
2	3
3	4
4	5
5	6
6	7
7	8
9	10
10	11 → (False)



Stop

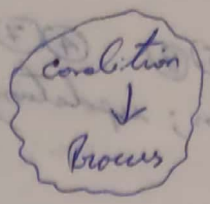
Program

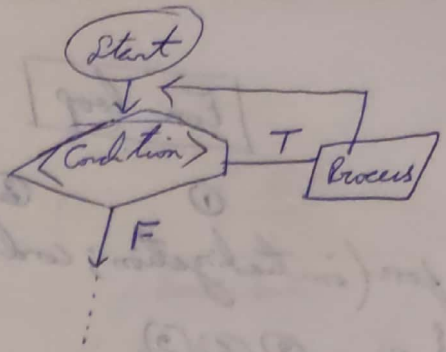
```
int main()
{
    int n, i=1;
    cout << "Enter n";
    cin >> n;
    while (i<=n)
    {
        cout << i << endl;
        ++i;
    }
    return 0;
}
```

OR

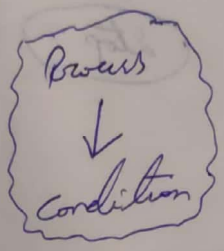
```
do
{
    cout << i << endl;
    ++i
} while (i<=n);
```

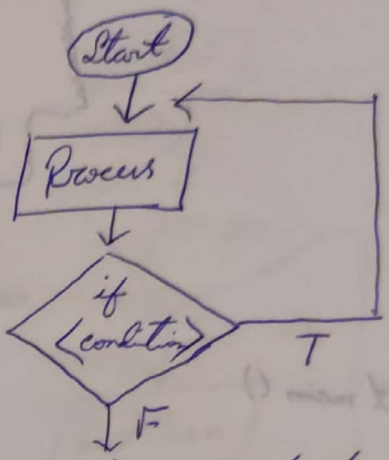
- while (<condition>)
{
process ;
}





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





∴ the process will execute atleast once

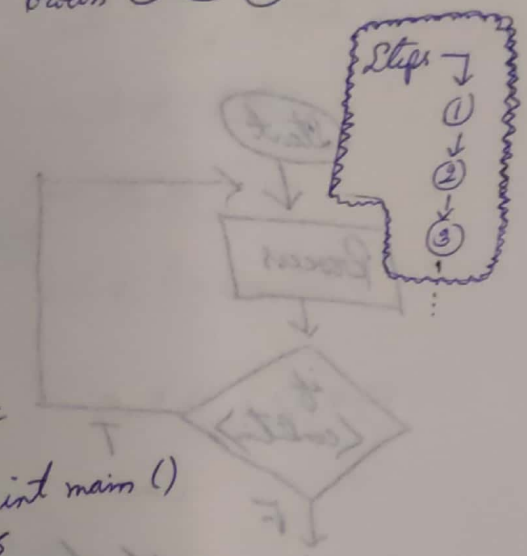
For loop

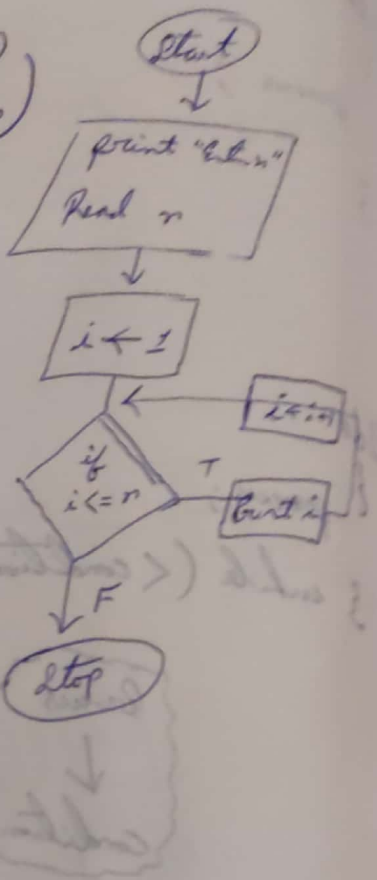
① ②⑤⑧ ④⑦

for (initialization ; condition ; updation)

{ Process ③ ⑥ ⑨

}






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

  }
return 0;
}
```

Multiplication table

n = 6

6 x 1 = 6 ← i * n

6 x 2 = 12

6 x 3 = 18

6 x 4 = 24

6 x 5 = 30

6 x 6 = 36

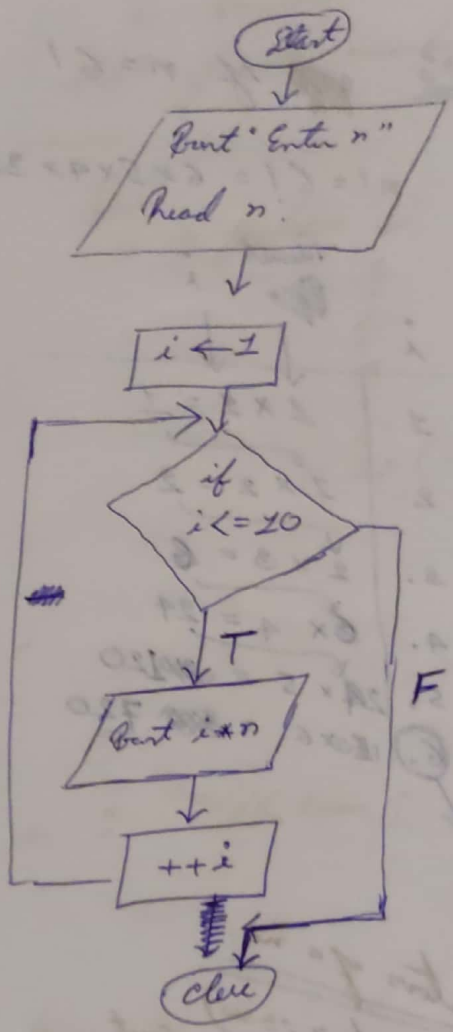
6 x 7 = 42

6 x 8 = 48

6 x 9 = 54

6 x 10 = 60

n ↑ ↖ i



8] using for loop :

→ Sum of first "n" natural no. →

*** (See in Laptop)

T i = 1 ; Sum = 0

i	Sum = Sum + i
1	0 + 1 = 1
2	1 + 2 = 3
3	3 + 3 = 6
4	6 + 4 = 10

↓ Sum ↓ i

8) n!

Ans: If n = 6!

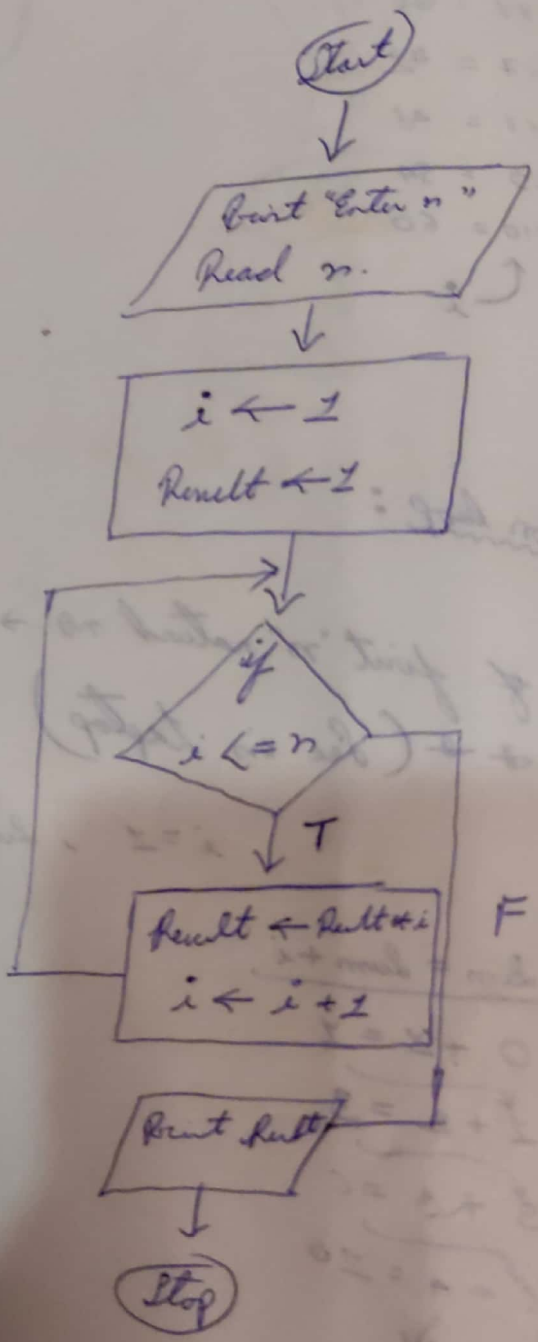
n! = 6! = 6 × 5 × 4 × 3 × 2 × 1

i	Result = Result × i
1	1 × 1 = 1
2	1 × 2 = 2
3	2 × 3 = 6
4	6 × 4 = 24
5	24 × 5 = 120
6	120 × 6 = 720

n = 6
i = 1
Result = 1

Factors of a no:

i	if (n % i == 0)	cout << i << all
1	4 % 1 == 0	1
2	4 % 2 == 0	2
3	4 % 3 == 0	X
4	4 % 4 == 0	4



Perfect Number

[Sum of factors is double than that of the number]

```
int main()
{
  int n; i, sum=0
  cout<< "Enter n";
  cin >> n;
  for (i= 1; i <= 0; i++)
  {
      if (n% i ==0)
      {
          sum = sum +i ;
      }
  }
  cout<< "Sum of the factors is :  "<< sum;
  return 0;
}
```

Eg:
6 → factors ↓

1 + 2 + 3 + 6 = 12 (6×2)

Initially

(n = 4)   (sum = 0) (i = 1)

i	{n%i==0}	sum = sum + i
1	4%1 == 0	0 + 1
2	4%2 == 0	1+2 = 3
3	4% 3 == 0	×
4	4%4 != 0	3+4 = 7

∴ 7 ≠ (4×2)

4 is not a perfect no.

[Now we can write a code for which we can infer the result of this blue print. only for YT — Don't read. Nothing it is fine the I can work all of the things out.]

Prime Number

count = 0

i	if (n % i == 0)	++count
1	4 % 1 == 0	1
2	4 % 2 == 0	2
3	4 % 3 == 0	—
4	4 % 4 == 0	4

Code (Laptop)

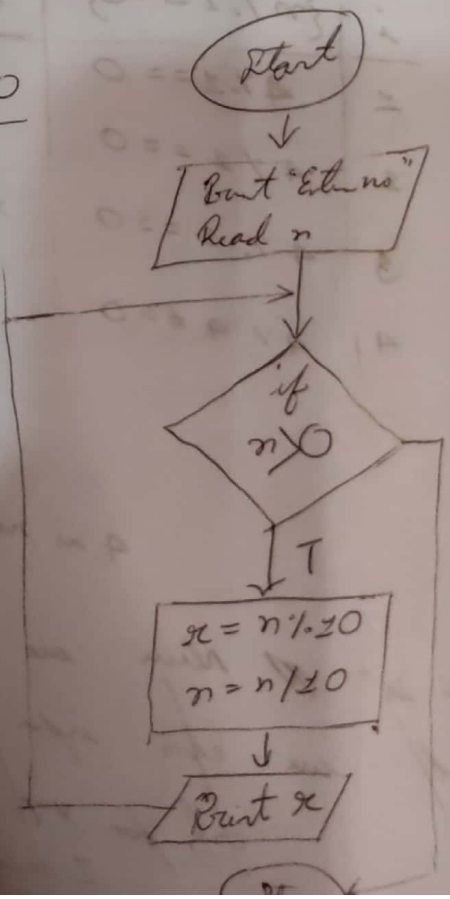
Display Digits of a number

→ n = 1724

1724 % 10 — 4

1724/10 — 172 { ∵ n = int }

n	r = n % 10	n = n/10
1724	4	172
172	2	17
17	7	1
1	1	0
0	—	—



✓ We can use both for() & while() loop.

(8) Armstrong Number

eg: 153

153 → Digits

$1^3 + 5^3 + 3^3$

$= 1 + 125 + 27 = 153$

If sum of cubes of the digits of Armstrong no. a number = number itself.
is known as Armstrong number.

sum = 0

n	r = n % 10	n/10	sum += r^3
153	3	15	$0 + 3^3 = 3^3$
15	5	1	$3^3 + 5^3 = 152$
1	1	0	$152 + 1^3 = 153$
0			↓ number.

Reverse

153

153 % 10 = 3 × 100 = 300

[rev = 0]

n	r = n%10	n = n/10	rev = rev * 10 + r
153	3	15	0 × 10 + 3 = 3
15	5	1	3 × 10 + 5 = 35
1	1	0	35 × 10 + 1 = 351

GCD

(Greatest Common Divisor)

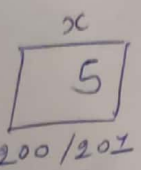
Array

Scaler $x = 5$

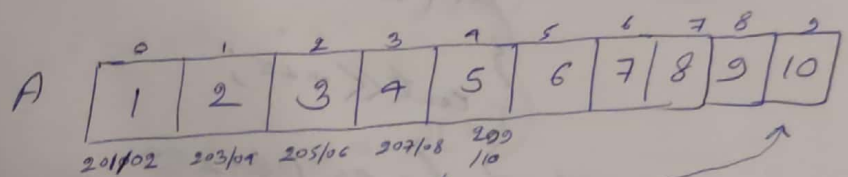
Vector/List $A = (5, 8, 3, 9, 7, 4, 8, 6, 3, 2)$

$A_0\ A_1\ A_2\ A_3\ A_4$

```
int x = 5;
```




```
int A[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }
```

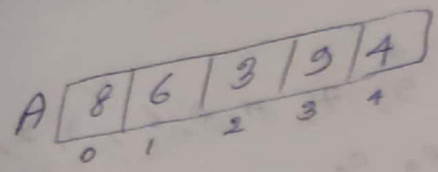



```
cout << A[3];  → 4
cout << A[8];  → 10
```

```
int main()
{
    int A[5] = { 2, 4, 6, 8, 10 };
    for ( int i=0; i<5; ++i)
    {
        cout << A[i] << endl;
    }
return 0;
}
```

i	A[i]
0	A[0] = 2
1	A[1] = 4
2	A[2] = 6
3	A[3] = 8
4	A[4] = 10

For each loop



n=5

```
for( int x : A)
{
   cout << x << endl;
}
```

→ x will get the copy of value.

if ↓

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

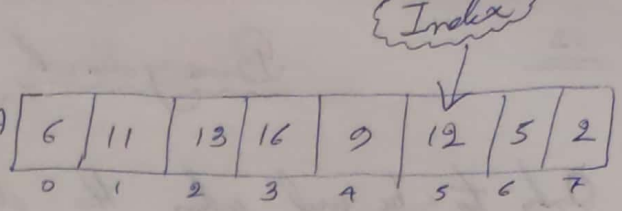
value as (8+1=9, 6+1=7)

It will print the but the original value within the array remains unchanged.

To change the value we can write reference here →

```
for(int &x:A)
{
  - - - -
}
```

91 Linear Search →




```
int main()
{
    int A[20], n=20, key;
    cout << "Enter the numbers : ";
    for (int i=0; i<n; i++)
    {
        cin >> A[i];
    }
    cout << "Enter key ";
    cin >> key;
    for (int i=0; i<n, ++i)
    {
        if ( key == A[i])
        { cout << "The position is : " << i;
          cout << " at ";
          exit(0);
        }
    }
    cout << " Not found ";
    return 0;
}
```

input
key = 12
→ 5
index

input
key = 35 unsuccessful

32

Binary Search

Only to be used when the elements in an Array are sorted.
(arranged in ascending / descending order)

$n = 10, \quad l = 0, \quad h = 9$

	0	1	2	3	4	5	6	7	8	9
A	6	8	13	15	17	19	21	23	25	27

Now,
let; key = 27

l	h	mid $\left[\frac{l+h}{2}\right]$
0	9	$\frac{0+9}{2} = 4$ → number is greater than 4 (17)
4+1 = 5	9	$\frac{5+9}{2} = 7$ → < than 7 (23)
5	(7-1) = 6	$\frac{5+6}{2} = 5$ → > than 5 (19)
6	6	$\frac{6+6}{2} = 6$ → Here is the number = 27.

Index

Here,

Binary Search is faster because

→ order is log(n)

& it scans the array more efficiently than Linear search (one by one)

Code

int main() → Lie in

Nested - For loops

Rows →

Columns ←

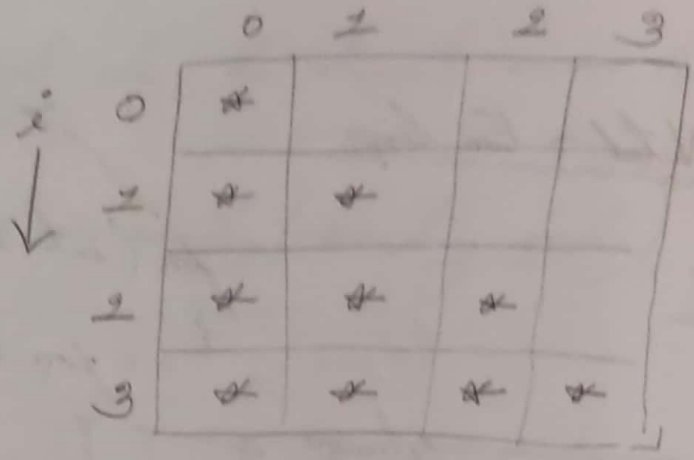
```
for (int i=0; i<3; ++i)
{
    for (int j=0; j<3; ++j)
    {
        cout << i << j << "\n";
    }
    endl;
}
```

output

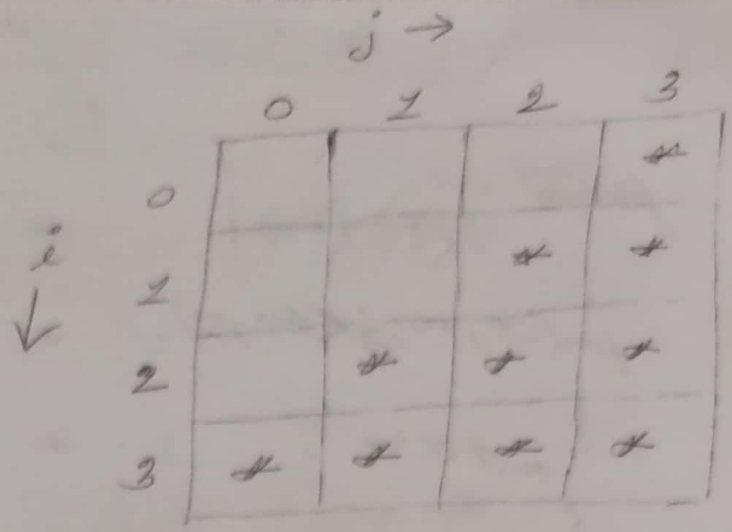
i	j
0	0
0	1
0	2
	3 × won't execute
1	0
1	1
1	2
	3 × won't execute
2	0
2	1
2	2
	3 × "
3	— × " .

Patterns

(1)



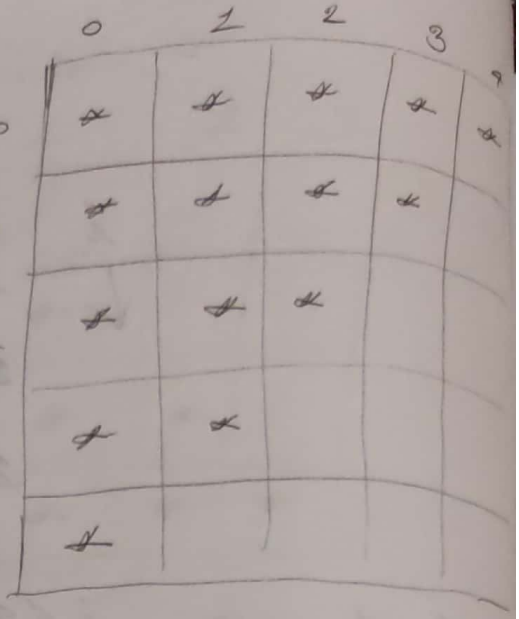

```
for(int i=0; i<4; i++)
{
    for(int j=0; j<4; j++)
    {
        if(j<=i)
        {cout<<"* ";}
        else
        break;
    }
    cout<<endl;
}
```




```
for (int i=0;  i<4, i++)
{  for (int j=0; j<4, j++)
   {   if ((i+j)>=(4-1))
       { cout<< "* ";
       }
       else
       { cout<< "  ";}
   }
   cout<<endl;
}
```

(3)

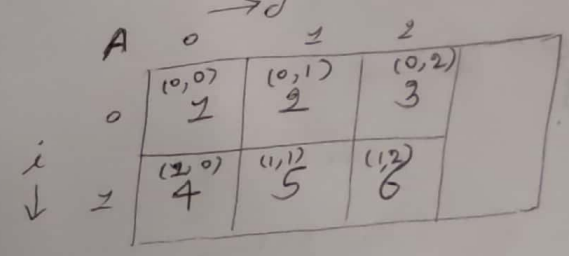
```
for(int i=0;
    i<5 ; i++){
for(int j=0; j<5-i; j++)
  { cout<<"* ";
  }
cout<<endl;
}
```



100

2D Arrays

```
int A[2][3] = { {1, 2, 3}, {4, 5, 6} }
```



We can also fill like →

```
int A[2][3] = {1, 2, 3, 4, 5, 6};
              row 1   row 2
```

or) = {1, 2} // rest will fill up as zero.

Note: To access the elements →

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

i, j: co-ordinates of the Array element

> 2D arrays are usually used to perform matrix operations. like →

Adding, Subtracting or multiplying two matrixes.

+++

Watch the Super Interesting Demo →

Pointers

Declaration int *p;

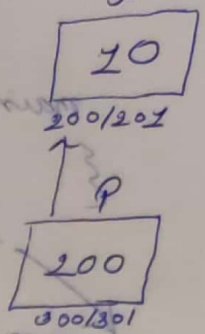
Initialization p = &x;

Dereferencing cout << *p;

int x = 10;

→ int *p;

→ p = &x;



cout << x; — 10

cout << &x; — 200

cout << p; — 200

cout << &p; — 300

cout << *p; — 10

Location of x in database

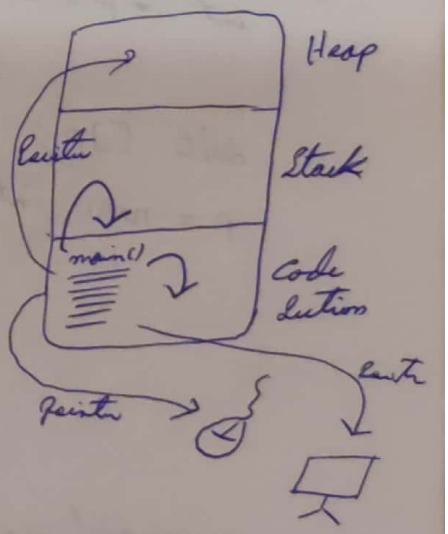
Pointer (shows per" of x)

shows per" of p

Dereferencing the data stored in x ; pointed out by p.

Why? Pointers?

A Program can only access code-section & stack memory. But in order to get access to Hardwares & Heap memory we need to use pointers.

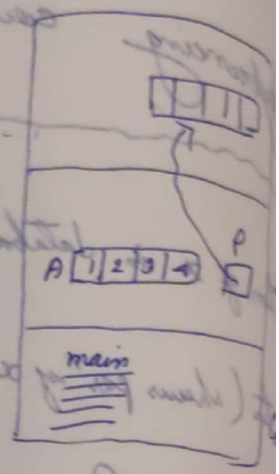


Note: Cin / Cout internally use pointers to interact with the devices.

1] How to create file in Heap?

```
main()
{
    int A[5] = {1, 2, 3, 4, 5};
```

It creates file in Stack memory




```
    int *p;
    p = new int[5];
```

It creates file in Heap memory

109

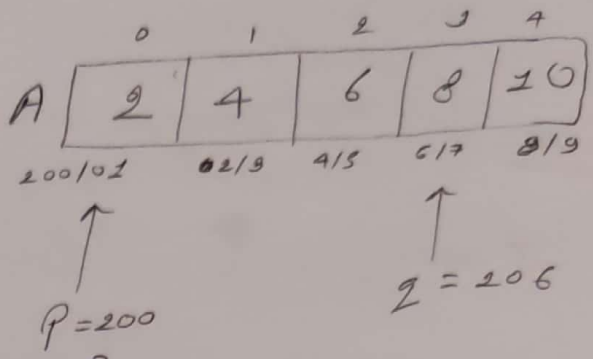
Dynamic Allocation

```
int main()
{
    int *p = new int[20];
    .
    .
    .
    delete []p;
    p = new int[40];
```

- We can change the size of Array in program (make pointer point to a new array which is not possible in stack memory.

Pointer Arithmetic

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



1. `p++;` — move to or behind the next element in Array.
2. `p--;`
3. `p = p + 2` { move 2 elements }
4. `p = p - 2` { move behind 2 " }
5. `d = p - q (-3)` → -ve then p is behind q.
   `d = q - p (+3)` → +ve then q is ahead of p.

Strings

1. Using char Array.
2. class string.

```
int main()
{
    char s[] = "Hello";
    cout << s << endl;
    return 0;
}
```

→ Hello

```
{
    char s[] = {'H', 'e', 'l', 'l', 'o', '\0', 'P', 'P'};
    cout << s << endl;
    return 0;
}
```

→ Hello
(string will stop after null character '\0')

```
{
    char s[] = {65, 66, 67, 68};
    cout << s << endl;
    return 0;
}
```

→ ABCDá!!-
→ because I didn't use '\0' at the end.

++ Always keep it in mind.

Instead of char[] to show string, we can use / should use string class.

116

```
int main(),
{
  char S[100];
  cout<<" Enter Your Name: ";
  cin.get (S, 100);
  cout << " Welcome << S << endl;
  return 0;
}
```

or, we can use cin.getline

→ Debojyoti Chowdhury

out: Welcome " " .

Note: Do not use cin.get

Use cin.getline. ✓

Solution to problem

```
int main(){

  char s1[100];
       s2[100];
  cout << "Enter your name:";
  cin.get(s1,100);
  cout << "Welcome" << s1 << endl;

  cin.ignore();

  cout << "Enter name again: ";
  cin.get(s2,100);
  cout << "Bye" << endl;

  return 0;
}
```

If we don't use it; the function cin.get cannot read Enter; it takes Enter as '\n'; & it is got by the 2nd get function. So, it shows →

```
Welcome Debajam Chowdhury
Enter name again: Bye.
```

When we don't use cin.ignore() ↘

Does not ask for value second time. ←

O/P

Note:

It is better to use → getline.

String Functions

library → <string.h>/<cstring>.

```
#include <string.h>
int main(){
    char s[50] = "Hello";
    cout << strlen(s) << endl;
```

→ Gives the length of string.

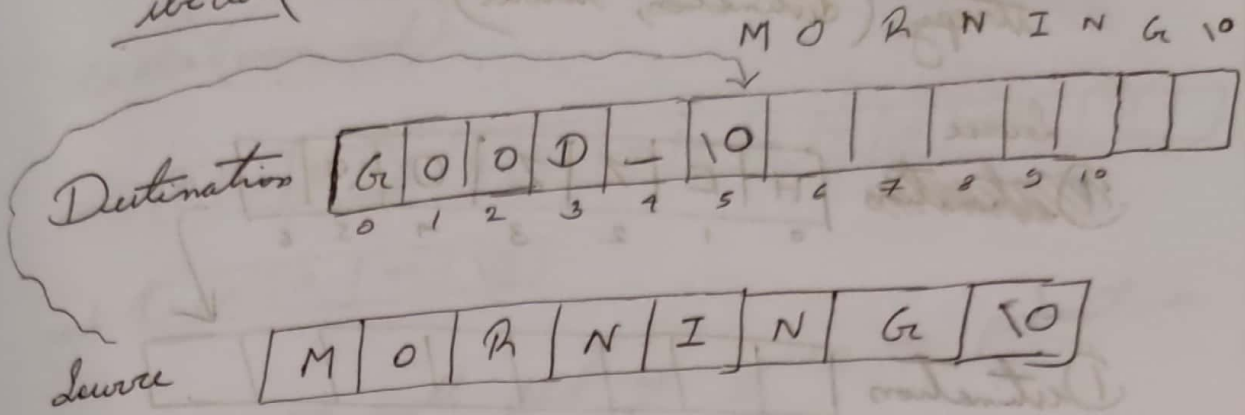
O/P:
5.

Using a pointer

```
{
    char *S;
    cout << "Enter a string: ";
    cin.getline(S, 100);
    cout << "Length → " << strlen(S) << endl;
    return 0;
}
```

String Concatination

strcat(destination, source);



⇒ To join two strings.

1. strncat(destination, source, length)
   → of the source to be added

Eg

```
{ char s1[] = "Good ";
  char s2[] = "Morning.";
  strcat(s1, s2)
  cout << s1 << endl;
  return 0;
}
```

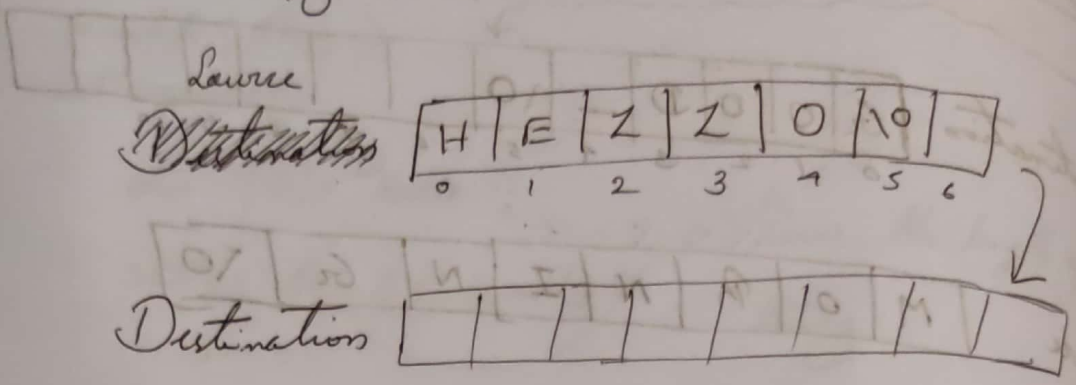
O/P:
Good Morning

```
strncat(s1, s2, 3)
```

O/P:
Good Mor

String Copy

strcpy (destination, source)



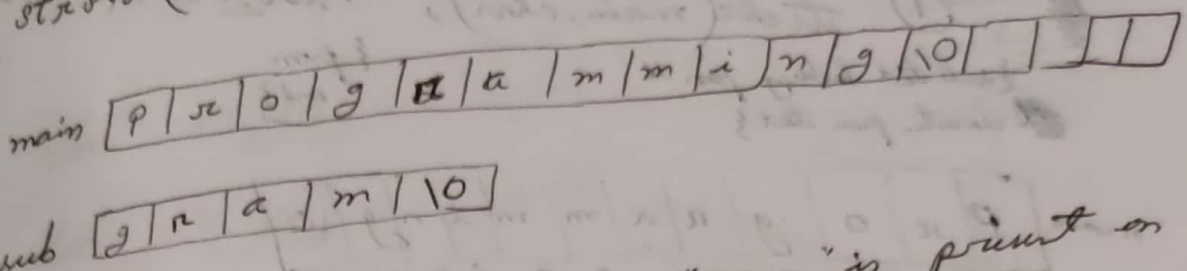
- To copy one string to another

strncpy (destination, source, length.

Similar to string concatination.

☑ Sub-string & Compare — 228

strstr(main, sub)



NOTE: It will find if "gram" is present in the "main" string.

If it is → It will print "gram" + the rest of the string.

⇒ If "sub" is not found in the main, it gives a runtime error. to avoid that → we can use:

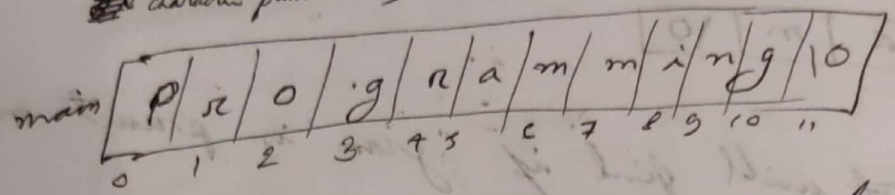
```
if (strstr(main, sub) =! NULL)
{ cout << strstr(main, sub); }
else
{ cout << "the function is null"; }
// "not found
```

• This is one way to find out the occurrence of a substring inside a main string.

"char" in a String

(1) strchr(main, char);
{will start looking from the character from start}

(2) strrchr(main, char);
{from end}



(1) It will find out if char (eg "r") present or "main". If it does it will print from start to end →

(2) ~~print from end to start~~

start to check from right hand side & print the out →

String to Integer / float:

String

∴ Decimal number system.

```
char s1[10] = "235";
char s2[10] = "54.78";

long int x = strtol(s1, NULL, 10);
float y = strtof(s2, NULL);
```

########

String to Token:

```
char s1[20] = "x=3;y=4;z=5";

token = strtok(s1, ";")

while(token != 0)
{
    cout << token << endl;
    token = strtok(NULL, ";");

}
return 0;
```

This will make (token = "y=4;z=5") run in the next step of the loop.

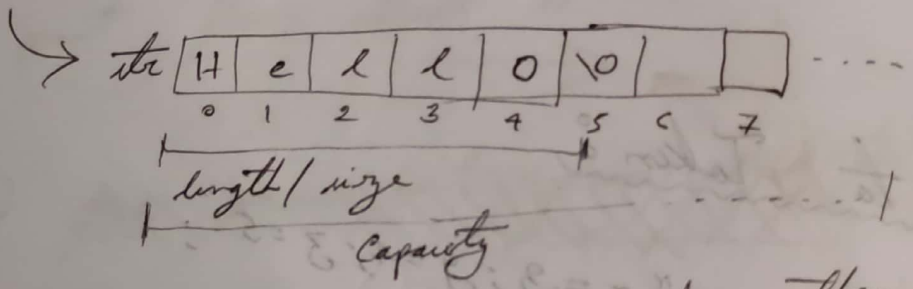
Class String

#include <string> ⟶ Library

```
string str;
cin >> str;
getline (cin, str);
```

~~Eg:~~

```
string str = "Hello";
```



→ The array created internally is larger than the element needed automatically :)

Demo

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

int main()
{
    string str;
    cout << "Enter Your Name : ";
    getline(cin, str);
    cout << "Hello " << str << endl;
    return 0;
}
```

Some string (class) Functions :

```
s.length()
s.size()
s.capacity()
s.max_size()
s.resize(50)      → changing capacity.
s.clear()   / s.erase()
s.empty()
```

More functions

- Adding at the end of a string

```
s.append()
```

eg:

```
string s = "Hello ";
s.append("World");
cout<<s<<endl;
```

→ Hello World.

- s.capacity()

length() < capacity()

- s.insert(3, "area", 3);

  - 3 → Index at which it needs to get inserted
  - "area" → object
  - 3 → number of characters to be inserted (area)

] s.replace (4, 3, "Dada")

Index of action

number of characters to be replaced.

] s.push_back ("z") → Hello z

use single quotes

] s.pop_back ( ) → pop out the last character

] s1.swap (s2)

s1 = "AAA"

s2 = "BBB"

]

s.copy (a, s.length())

string s → will get copied to a[20].

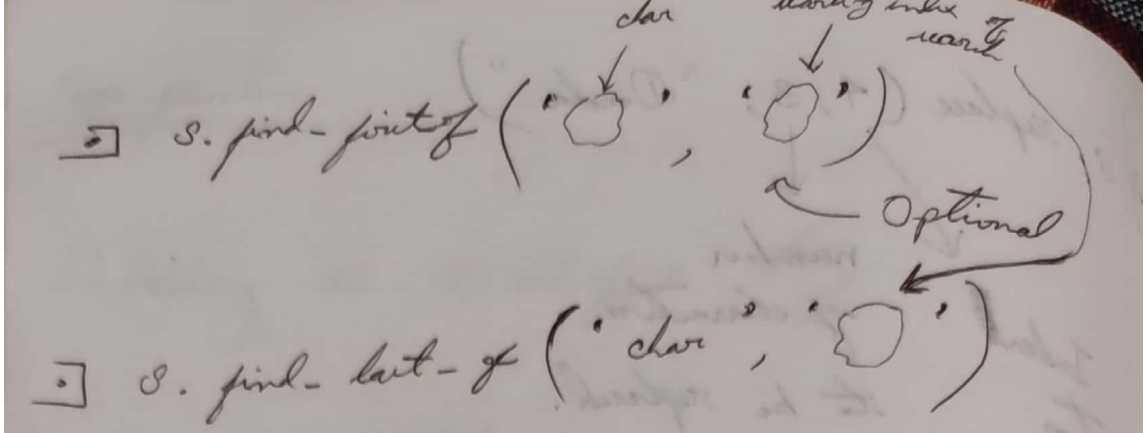
no of characters to be copied.

] s.find ("str") or ('char')

Note:
For string " " / char ' '.

s.rfind (" ")

↳ will start searching in reverse.



- S. find_first_of ('', '')
- S. find_last_of ('char', '')

Note:

If we put a string ("le") there → it will start looking for the individual characters / if you wanna find the word/string, use the previous functions.

- S. substr (starting index, no. of characters to be added)
- S. compare (str) - ve 0 +ve
  not required.

String Operators

s.at(i) — index

- s.at() — similar to s[i] but we can use it as an input operator as well
- s.front()
- s.back()
- []
- + (concatenation)
- = (Assignment)

String Iterator

string : iterator

- begin()
- end()
- reverse_iterator
- rbegin()
- rend()

Hello Everyone
0 1 2 3 4 5 6 7 8 9 10 11 12 13

A B C D E D C B A
0 1 2 3 4 5 6 7 8

// String Palindrome in the coolest way →

```
{
    char  s1[50];
    int i, length;
    int flag = 0;
    cout<< " Enter the String:";
    cin>> s1;
    length =strlen(s1);
    for(i=0; i<(length/2); i++)
    {
      if( s1[i] != s1[length-i-1])
      { flag =1;
        break;
      }
    }
    if(flag=1)
    { cout << " Not palindrome " << endl;
    }
    else { cout<< " Palindrome " << endl; }
```

Functions

Function-name (parameter list)

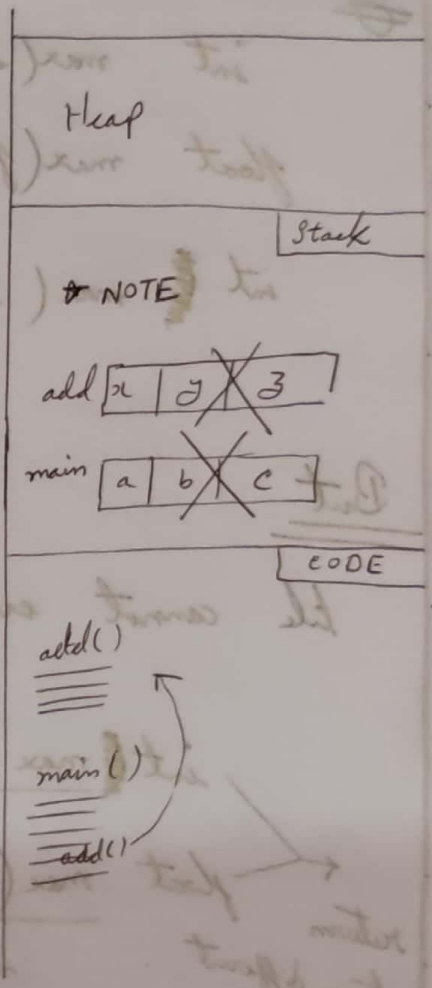
return-type ↓ i/o

parameter list ↓ 0 to n

arg arg

Eg: 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 << c << endl;
}
```



NOTE:

① function returns maximum 1 value.

② void() doesn't returns anything.

③ * After func() call, the memory in stack is automatically deleted.
is finished

Function Overloading

- C++ allows us to use multiple functions under same name — if 2
  1. Arguments are different / no of arguments
  2. " data types are different

Eg:

```
int   max(int, int)
float max(float, float)
int   max(int, int, int)
```

But

We cannot over-ride the return datatype

```
int   max(int, int)
float max(int, int)
```

→ Same

return type different

✗ don't do

Template

When no. of arguments & body of two or more functions are same & the only difference is Data-type; then we can just use template & it will adapt accordingly to the datatype.

Eg:

```
template <class T>
T maximum (T a, T b)
{
    return a>b? a : b;
}
int main()
{
    float/int x, y;
    cin>>x>>y;
    cout<< maximum(x,y) <<endl;
}
```

Default Arguments

Similar fun" with different no. of arguments.

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

So → to combine them we can assign default value to the arguments.

```
int add(int x, int y, int z=0)
{
    return x+y+z;
}
```

Then run both →

```
add(5,10)
add(5,10,15)
```

NOTE: Always start from right hand side & don't leave any arguments in blw.

```
int add(int a, int b, int c=0, int d=0)
```

here if
b → non-assigned
a = 0
} ✗ invalid

Parameter Passing Address

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

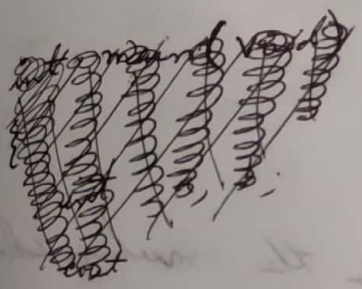
pointer variables.

call

actual parameters

```
swap(&a, &b)
```

I/P address



Pass by Reference

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

call

```
swap(x, y)
```

NOTES →

Reference in C++

When a variable is declared as reference, it becomes an alternative name for an existing variable. it is declared by putting "&" before the variable name.

Eg:

```
int &a = x;
```

(&a: reference variable)

###

In pass by reference method,

when function is called

→ the instructions are copied in the machine code like this

```
main()
{
    ]  Swap function
}
```

Thus,

there is no function call in reality.

NOTE

"Temp" is deleted after the swap chunk

the ... is executed.

Usually when function is called, a seperate piece of Machine code is generated along with seperate activation memory.
& Deleted after it's function is executed

But,
in this method, no seperate memory or code is created.

When to Use

- When we want to change the actual parameters.

NOT to use

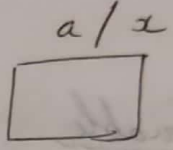
- Not to use comlex logic inside inline function (it will increase the size of machine code) by repeatation copying.

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;
}
int main()
{
    int *ptr = fun(5);
    for(int i=0; i<5; i++)
    {
        cout << *(p+i) << endl;
    }
}
```

Return by Reference

```
int & fun( int &x)
{
    return x;
}
int main()
{
    int a=5;
    fun(a) = 25;
    cout<< a<< endl;
}
```



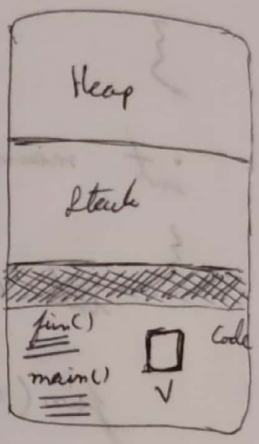
function has become reference variable of a

→ 25 (value got changed)

⊡ How to access global variable inside of main() function? →

```
cout << ::x ;
```

↑ use double colon to access global variable inside main function.



Static Variable

- It acts just like a Global variable with limited scope.
- It is created in the code section during ~~booting~~ execution (only once)
- & it stays there as it is.

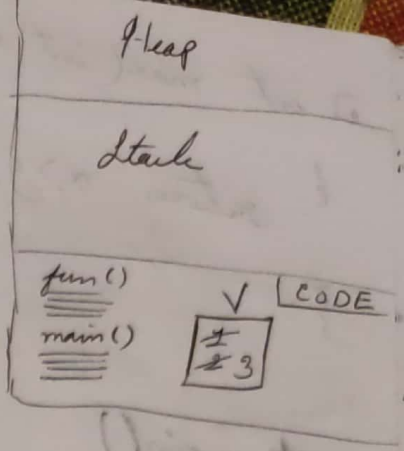
Eg

```
void fun()
{
    static int v=0;
    int a=5;
    v++;
    cout<<a<<" "<<v<<endl;
}
```

→ Executed only one to create a memory on code section.

Remains throughout the program.

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



Recursive Functions

When a function calls itself is recursive function.

(More in Data Structures)

Function Pointer → pointer assigned to a function

```
void display()
{
    cout<< "Hello";
}
int main()
{   void (*fp)();   —— Declaration
    fp = display;   —— Initialization
    (*fp)();        —— call
}
```

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

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

int main()
{
    int (*fp)(int, int);  — Declaration

    fp = max;
    (*fp)(20, 5);  —— max is called

    fp = min;
    (*fp)(20,5);  —— min "

}
```

Modular

Bank

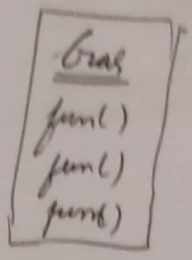
Deposit()

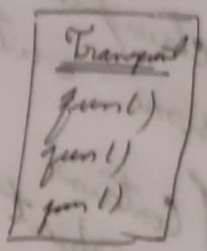
Withdraw()

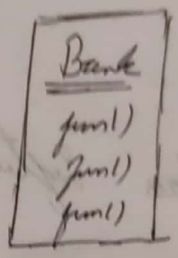
Check Bal()

OOPS

Grant







Principles of OOPS

Data

Functions()

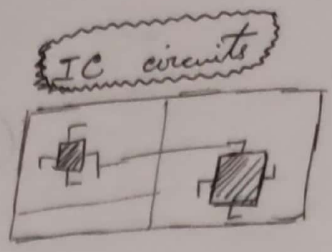
Note: Functions() perform operations on Data.

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

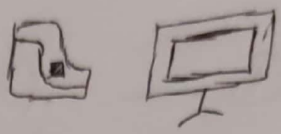
Class vs Object

Categorization or classification or some criteria.

(Contains Object)



Class



Objects

```
class Software
{
    Data;      ——— property
    Function;  ——— behaviour
};
```

Example

```
Class Rectangle
{   float length;
    float breadth;
    float area();
    float perimeter();
    float diagonal();
    :
    :
};
main()
{ Rectangle rc1, rc2, rc3;
}
```

rc1, rc2, rc3 — Objects

Inheriting a class

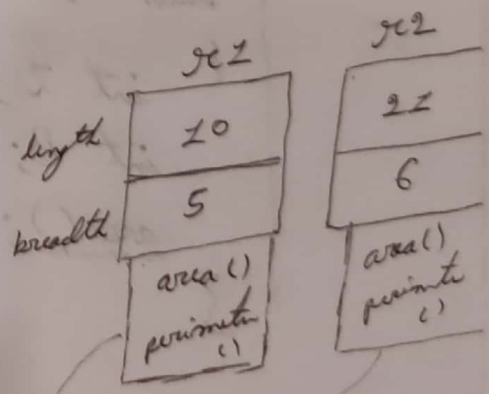
```
class Rectangle
{
public:
    int length;
    int breadth;

    int area()
    { return length * breadth; }

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

int main()
{
    Rectangle r1, r2;

    r1.length = 10;
    r1.breadth = 20;

    cout << r1.area() << endl;
}
```

Data

Functions

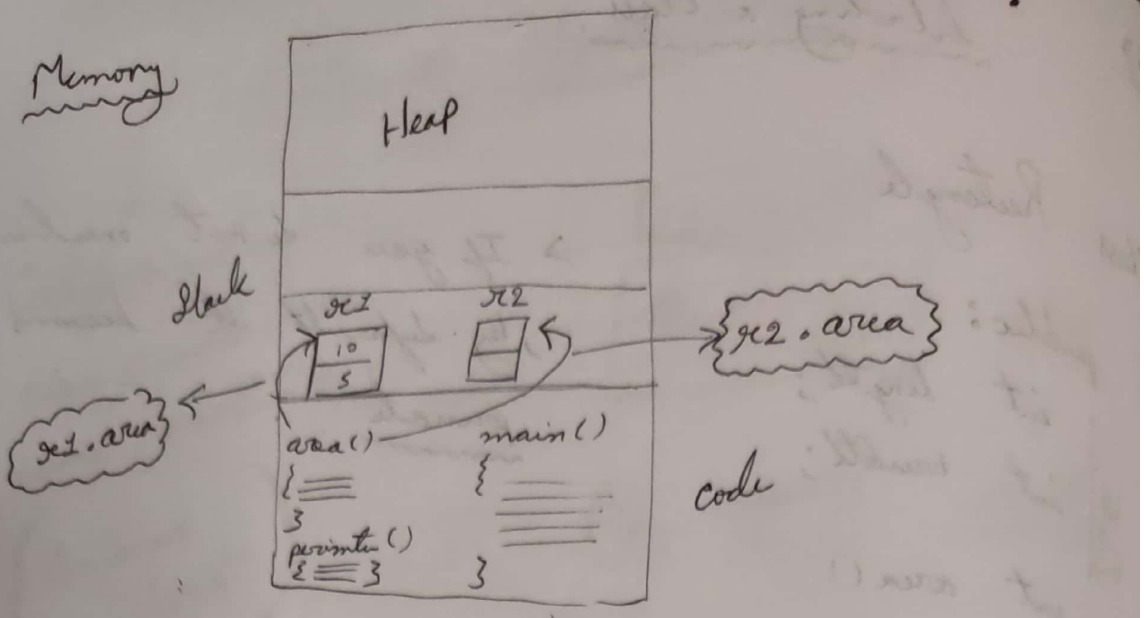
→ If you do not mention, by default it becomes private.

→ becomes user defined data type (or Table with boxes)



It does not take any memory.

Temporary memory created & deleted when f" is called.



NOTES

- Data hiding is done for avoiding Data mishandling not for security

- Data is accessible only within class
- Only the functions of a class, knows that what processing should be done on data.

```
class rect
{
    int l;
    int B;
    area();
}
```

class "fun" only can access the Data.

Encapsulation vs Abstraction

- A class is used for both encapsulation as well as abstraction.

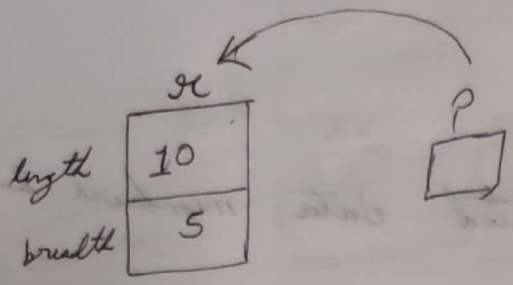
→ Encapsulation:
Means combining related data members & functions together.

→ Abstraction:
Hiding data & showing required functions
(achieved by "private:" & "public:")

Structure vs class

→ Data public by default (structure) | Data private by default (class)

Pointer to Object



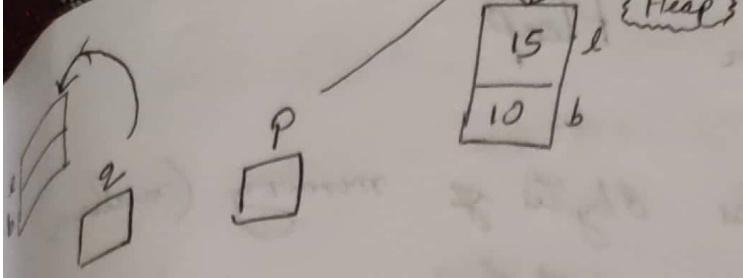

```
int main()
{ Rectangle r;  // variable
  Rectangle *p;  // pointer
  p = &r;  // Assignment
  // r.length = 10;
```

⟶ dot(.) operator is used for accessing an object using variable name.

```
  p -> length = 20;
```

⟶ Arrow(->) by using pointers

```
  p -> breadth = 5;
cout << p -> area(); << endl;
}
```




```
class Rectangle
{ public:
    int l;
    int b;
    int area();
};
```

```
void main()
{ Rectangle *p;
  p = new Rectangle;
  or, Rectangle *q = new Rectangle();
  p -> length = 15;
  p -> breadth = 20;
  cout << p -> area();
}
```

Creating Object in Stack vs Heap

Stack

```
Rectangle r;
Access: r.area();
```

Heap

```
Rectangle *p = new Rectangle();
Access: r -> area();
```

Stack or Heap

- Every pointer takes 8 bytes of memory (modern) irrespective of the datatype.

Philosophy behind data handling / Hiding

→ Lecture 163 ←

- Data member → private
- Functions → public

Data Hiding

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

class Rectangle
{
    private:
        int length, breadth;

    public:
        void setLB( ) // set length & breadth
        { int l, b;
        jump:
            cout << "Enter length & Breadth : " << endl;
```

```
cin>> Z >> B;
if (Z >= 0 && B >= 0)
{
    length = Z;
    breadth = B;
}
else
{ cout << " Length & Breadth cannot be
                -ve " << endl;
    goto jump;
}                    // Loop to get
                        valid input
} // setZB
int getLength()
{ return length;
}
int getBreadth()
{ return breadth;
}
int area()
{ return (length * breadth);
}
};
main()
{ Rectangle rc;
  setZB();
} cout << "Length: " << rc.getLength() << ..... ;
```

Property functions

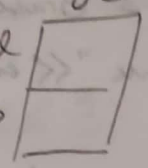
Accessors — get xxx

Mutators — set xxx

Constructors

- According to convention, whenever an object is created, the data should have a default value.

Rectangle r;




```
class Rectangle
{
private:
    int l, b;
public:
```

Here, there is a build in function that commands the allocation of memory (r) — default constructor

1. Default constructors.
2. Parameterised "
3. Non - " "
4. Copy - "

} UDC (user defined constructors)

→ P.T.O

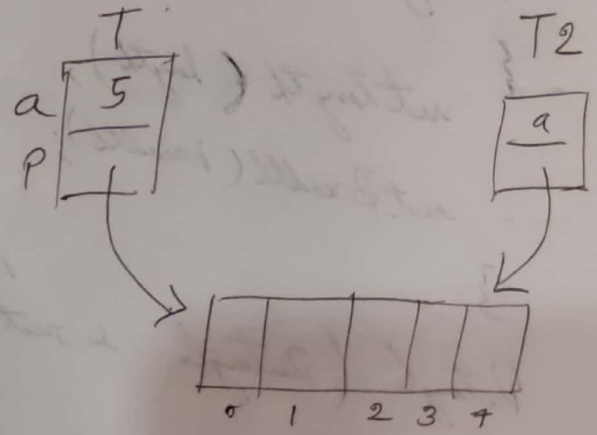
- Rectangle r; → Default contructor
- Rectangle r(); → (0, 0) no-parameter
- Rectangle r(10, 5); → (10, 5) parameter
- Rectangle r2(r); → copy contructor

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

→ Reference variable so that no extra memory is used / created.

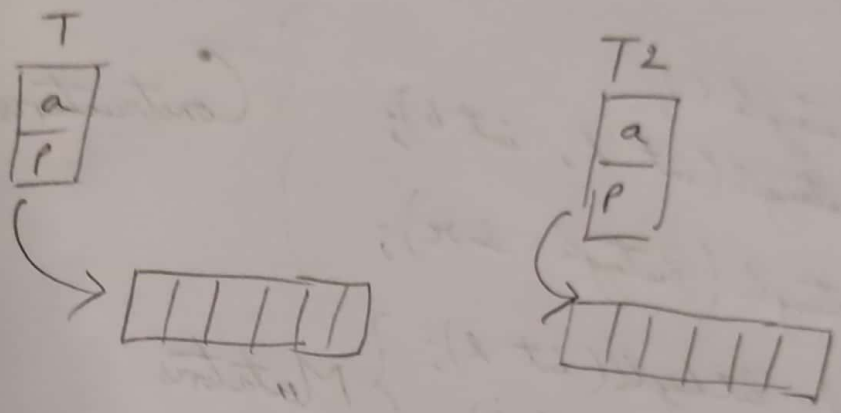
Deep Copy Constructor

```
class Test
{ public:
    int a;
    int *p;
    Test (int x)
    { a = x;
      p = new int [a];
    }
    Test (Test &m)
    {
        a = m.a;
        p = m.p;
    }

main()
{   Test T(5);
    Test T2(T);
}
```



- The problem is both of them are pointing at the same array, instead of making a new array.

```
Test (Test &m);
{  a = m.a;
   p = new int[a]; // new array is created
}
```



∴ Everything is copied
[Deep Copy Contructor]

≡ Remember to delete * if you use Dynamic Allocation (Heap memory)

Types of function in a class

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

  public:
     Rectangle ();
     Rectangle (int l, int b);
     Rectangle (Rectangle &rc);

     void SetLength (int l);
     void SetBreadth (int b);

     int getLength ();
     int getBreadth ();

     int area ();
     int perimeter ();

     bool isSquare ();

     ~Rectangle ();
};
```

Constructors

Mutators

Accessors

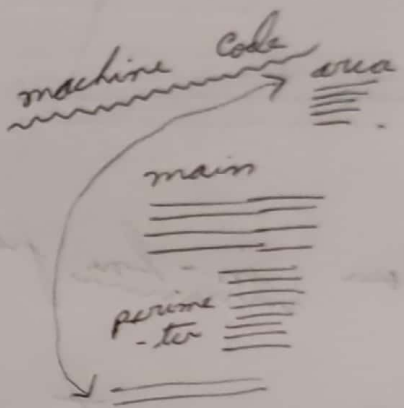
Facilitators

Inspector / Enquiry

Destructor

Scope Resolution

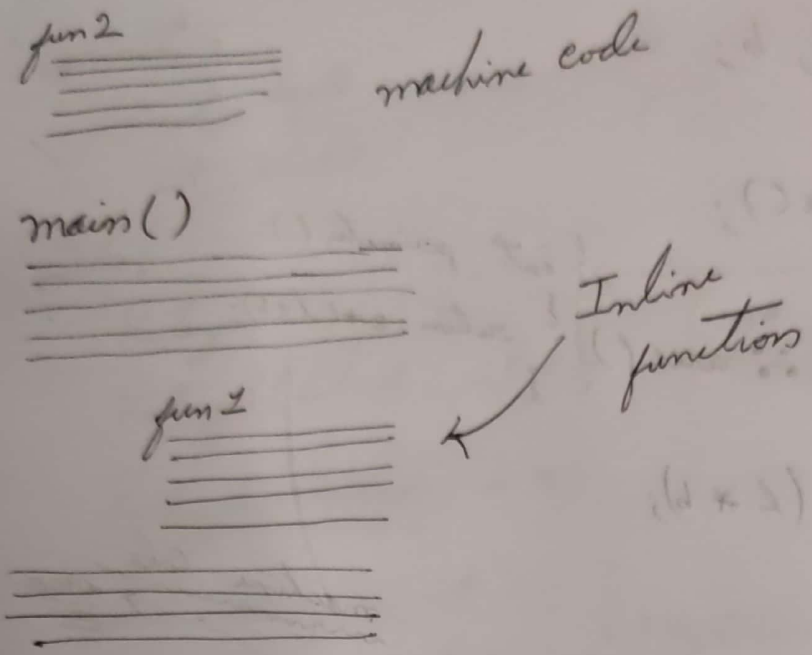
```
class Rectangle
{ private:
    int l, b;

  public:
    int area();
    int perimeter()
    { return 2*(l+b);
    }
};
int Rectangle :: area()
{   return (l * b);
}

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



∴ Fn using scope resolution have their machine code outside the main function & generated when called.

NOTE: Inbuilt functions outside the class, saves memory & processing power.

Inline Functions



How in-line functions are make →

- If we write a function inside class (In-line)
- " " " " " outside " (non-inline)

- While declaring function within class → declare inline.

```
class Rect
{
  inline void setLB();
};
void setLB()
{
}
```

This Pointer

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

  public:
    Rectangle (int length, int breadth)
    { this -> length = length;
      this -> breadth = breadth;
    }
```

→ By using {this ->} operator we can avoid name problem.
It always points to class Data (in private section)