🟥 → more important note and Week Lecture / homework

🟪 → Data Structure

🩷 → links

🟩 → Heading

🟫 → Subheading or important point

IMPORTANT POINTS

→ Focus on work on skill and build networks those work in companies and can talk to HR for you

→ Internship- do if you want to learn or convert it as PPO

→ Rough sols. and dry run are very important

→ Do documentation to promote and mail everything you discuss with HR or team

→ Think twice, Code once

→ To clarify about any code you are confused, use cout statements everywhere to know what is going on in the code

→ Code all approaches you can think of and can find & understand from google

→ Revise all incorrect & skipped questions in quizes regularly

→ Watch sol. only after attempting the question

IMPORTANT C++ NOTES

- → Making global variable is BAD PRACTICE

- → To increase range of int / long long , you can use unsigned int / long long

- → % → heavy operator
  $\hookrightarrow$ so try to use it less

  Use bitwise operators instead of this if possible

Thought process to solve a problem-

W1-L1

→ Understand a problem

→ input values

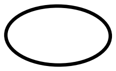
→ find approach

$$\begin{tikzcd}
\boxed{\text{Problem}} \arrow[r] & \boxed{\text{Sol. on head}} \arrow[r] & \text{rough sol.} \arrow[r] \arrow[d] & \boxed{\text{Code}} \arrow[r] & \boxed{\text{Compiler}} \arrow[r] & \boxed{\text{low level lang.}} \\
 & & \text{very imp} & & &
\end{tikzcd}$$

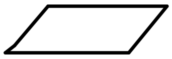
Algorithm- Sequence of steps

Flowchart- Graphical representation of algo

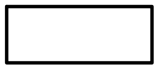
Components-



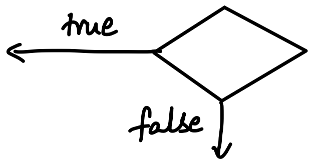
terminator for start /end



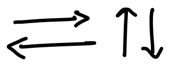
for input /output read /write



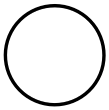
computation/ process / declaration



decision making block
takes condition



flow



Connector
takes function

Pseudo Code- Generic way of writing algo

Dry Run → **Very Important** to understand any topic

W1-L2

IDE- Replit, VS-Code

```
# include <iostream>
using namespace std;
int main () {
    cout << "Namaste Bharat";
}
```

include <iostream> → preheader file contains implementation of identifiers

namespace std → using standard namespace implementation of cout choosing from multiple types of namespace

namespace → region where scope of identifier is defined

cout << → used to print on console/ standard display

"Namaste Bharat" → string

; → to end any statement

```
cout << endl;
cout << '\n';
```

→ for next line

int a ; ⟶ a is an integer $\boxed{gv} \rightarrow$ garbage value

cin>> a ; ⟶ $\boxed{7}$

ex- 7

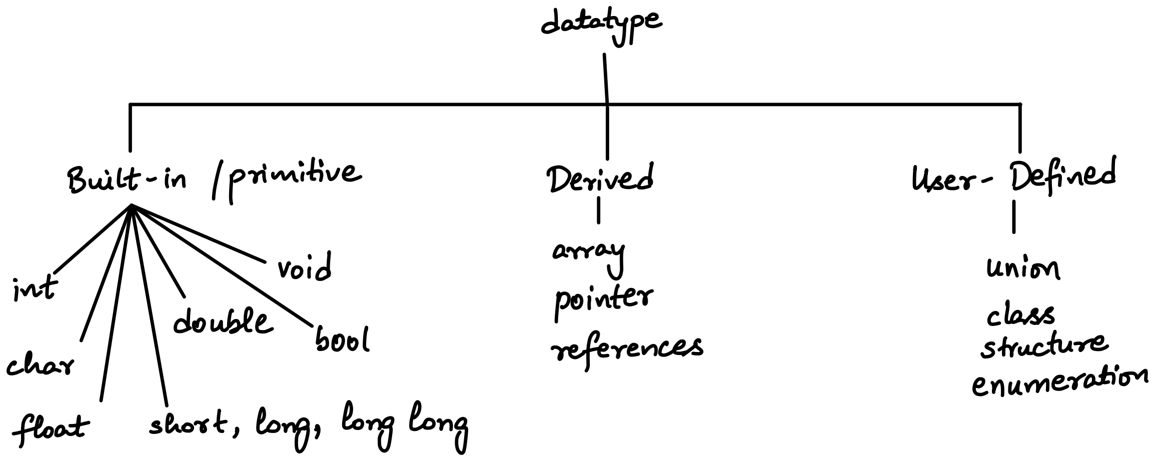
Variables

named memory location

$$\begin{tikzcd} \text{int} \arrow[d] & \text{a} \arrow[d] & \text{= 5 ;} \arrow[r, bend right] & \text{value} \\ \text{datatype} & \text{variable} & & \end{tikzcd}$$

Datatypes

type of data



int - 4 byte - 32 bits in memory

$\hookrightarrow -2^{31}$ to $2^{31}-1$ in signed int

$\hookrightarrow 0$ to $2^{32}-1$ in unsigned int

char - 1 byte - 8 bits in memory

$\hookrightarrow 2^{8}$ different chars.

ASCII

char maps with numerical ASCII value

char ⟷ ASCII value → store in memory

bool → 1 byte → 8 bits

true - 1
false - 0

└→ because minimum addressable memory is 1 byte
We cannot address 1 bit in memory

float → 4 byte → 32 bits

double → 8 byte → 64 bits

long long → 8 byte → 64 bits

short → 2 byte → 16 bits

long → 4 byte → 32 bits

How data is stored

int a=5

└→ 32 bits $\underbrace{0 \ldots 00}_{29 \text{ bits}}101$

+ve no $\longrightarrow$ 0 ______

-ve no. $\longrightarrow$ 1 ______

Most significant bits

How -ve number is stored in memory

In 2's complement form

$\hookrightarrow$ 1's complement +1

$\hookrightarrow$ reverse all bits

int a = -7

7 $\longrightarrow$ 0 . . . . 0 0 1 1 1 } 32 bits

ignore -ve sign
find binary equivalant

1's (7) $\longrightarrow$ 1 . . . . 1 1 0 0 0

find 2's complement

2's (7) $\longrightarrow$ 1 . . . . 1 1 0 0 1

$\hookrightarrow$ this is how -7 will be stored

How to read -ve no. present in memory

$\hookrightarrow$ take 2's complement

1 . . . . 1 1 0 0 1

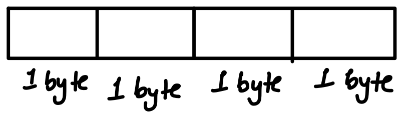
$\hookrightarrow$ 1's complement $\longrightarrow$ 0 . . . 0 0 1 1 0

2's complement $\longrightarrow$ 0 . . . 0 0 1 1 1

$\hookrightarrow$ + 7

$\boxed{-7}$

Interesting problem



how computer know these are 4 chars or a single integer

$\hookrightarrow$ Using datatype

$\hookrightarrow$ tell 2 things

- $\to$ type of data used
- $\to$ space used in memory

Signed vs Unsigned

Unsigned $\to$ 0, +ve

Signed $\downarrow$ +ve, -ve, 0

$\hookrightarrow$ by default

int – 4 byte – 32 bits in memory

$\hookrightarrow$ total no. of combinations – 2^{32}

signed int	unsigned int	
-2^{31} to $2^{31}-1$	0 to $2^{32}-1$	} range
10...01 011...1	0......0 1.....1	} in memory

$\hookrightarrow$ by using 2's complement of 2^{31}

General Formula

n bits in memory

$\hookrightarrow$ total no. of combinations $- 2^n$

$$\begin{tikzcd}
 & & & \text{total no. of combinations} \arrow[dll] \arrow[drr] & & & \\
 & \text{signed int} \arrow[d] & & & & \text{unsigned int} \arrow[d] & \\
-2^{n-1} \arrow[d, bend right] & \text{to} & 2^{n-1}-1 \arrow[d, bend left] & & 0 \arrow[d] & \text{to} \quad 2^{n}-1 \arrow[d, bend left] & \} \text{ range} \\
\underbrace{10\ldots01}_{n \text{ bits}} & & 011\ldots1 & & 0\ldots\ldots0 & \underbrace{1\ldots.1}_{n \text{ bits}} & \} \text{ in memory}
\end{tikzcd}$$

Typecasting

$\hookrightarrow$ convert one type of data to another

implicit type casting

ex- `char ch= 97;`

`cout << ch;` $\longrightarrow$ $\textcircled{a}$

explicit typecasting

ex - `char ch =(char) 97;`

`cout << ch;` $\longrightarrow$ $\textcircled{a}$

overflow ex -

`char ch = 9999;`

`cout<< ch;`

$$9999 \xrightarrow[\text{binary conversion}]{} 100\ 111\ \underbrace{0000\ 1111}_{\text{stores only last 8 bits}}$$

so ch stores 00001111 in memory

$$\begin{tikzcd}
00001111 \arrow[d] \\
7 \arrow[d] \\
\text{acc. to ASCII table}
\end{tikzcd}$$

Operators -

Arithmatic Operator

└→ +, -, *, /, %

$$\begin{tikzcd}
\text{int op int} \arrow[r] & \text{int}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{float op int} \arrow[dr, bend left] & \\
\text{int op float} \arrow[r, bend right] & \text{float} \\
\text{float op float} \arrow[ur, bend right] &
\end{tikzcd}$$

$$\begin{tikzcd}
\text{double op int} \arrow[dr, bend left] & \\
\text{int op double} \arrow[r, bend right] & \text{double} \\
\text{double op double} \arrow[ur, bend right] &
\end{tikzcd}$$

3 → int
3.0 → float / double not int

Relational Operator

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

these are different things

Output - 0 or 1

$$\begin{tikzcd}
0 \arrow[d] & 1 \arrow[d] \\
\text{false} & \text{true}
\end{tikzcd}$$

Assignment Operators

=

Logical Operators

└→ when you have multiple conditions

&&	→ and	→ true if both are true
||	→ or	→ true if any one is true
!	→ not	→ negate the result

Output - 0 or 1

0 ↙ false

1 ↘ true

(cond1 && cond 2 && cond 3)

└→ if cond 1 is false

compiler will not check further

as ans will already false

Conditions

W1-L3

```
if (cond.){
      execute
}
```

if

```
if(cond){
      execute 1
}
else {
      execute 2
}
```

if-else

```
if(cond1)
      execute 1
else if(cond2)
      execute 2
```

if- else if

```
if (cond1)
    execute 1
else {
        if () { }
        else () { }
}
```

nested if - else

```
if (cond 1)
        execute 1
else if ( cond 2)
        execute 2
else if ( . . .
else
        execute n
```

if - else if - else

Loops

└→ to do something repeatedly

for - loop

initialization   condition   updation

```
for( int i=0 ;  i<5 ; i=i+1 ){
        cout << "Love" ;          ——→  executing body
}
```

flow -

$$\begin{tikzcd}
\text{initialization} \arrow[r] & \text{condition} \arrow[r, "\text{false}"] \arrow[d, "\text{true}"] & \text{exit} \\
& \text{executing body} \arrow[d] & \\
& \text{updation} \arrow[uu, bend left] &
\end{tikzcd}$$

initialization
condition
updation

$\Bigg\}$ none is mandatory
one or multiple can be added

patterns -

Generally 2 loops $\longrightarrow$

```
outer loop(){          → for rows
    inner loop(){      → for cols
    }
    cout << endl;
}
```

→ a op = b $\longrightarrow$ a = a op b

op → +, -, *, %, /

cin in if( )

```
int num;
if(cin >> num){
    cout << "hello";
}
else {
    cout << "hi";
}
```

it will not give error

output -
hello

for all values of num
↓
0, +ve, -ve

cout in if( )

```
int num = 0;
if( cout << num << endl){
    cout << "hello";
}
else {
    cout << "hi";
}
```

it will not give error

output -
0
hello

for all values of num
↓
0, +ve, -ve

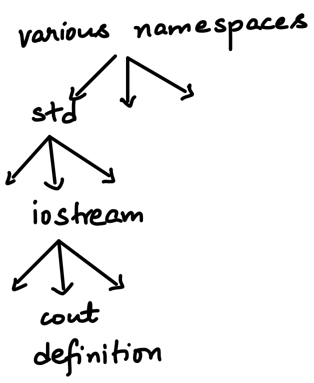
W1-L4

HLL - High level language
↳ human readable and user friendly

C++, C - Middle Level language

namespace ⟶ to avoid collision
↓
multiple definitions of a single keyword

heirarchy
↳ various namespaces
↓
std namespace
↓
iostream preheader file
↓
keyword definition



```
float f= 2.0+ 100;
cout << f;
```

⟶ output -

$\underbrace{\text{102 or 102.0}}_{\text{compiler dependent}}$

```
float f= 2.7;
int n= 157;
int diff= n-f;
cout << diff;
```

output -
154

explanation -
n-f = 157 - 2.7 = 154.3
int diff = n-f
diff = 154

ternary operator -

W1-HW

↳ syntax

variable = (condition) ? expression2 : expression3

(condition)? variable = expression 2 : variable = expression3 patterns

W2-L1

↳ how to think

- finding formula for rows and cols

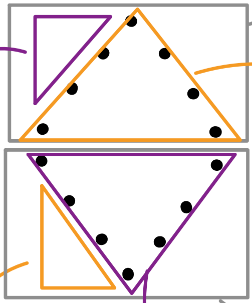
n=5

row	stars		n-1	no. of times loop runs
0	0		-1	0
1	0	→ formula - 0 to < n-1	0	0
2	1		1	1
3	2		2	2
4	3		3	3

→ as condition fails (0 < -1)

- to do anything n times
  - ↳ for(i=0 ; i<n; i++){ }

- break the complex patterns



0 to < n-(i+1)

space

0	→	3	4-1
1	→	2	4-2
2	→	1	4-3
3	→	0	4-4

0 to < 2i+1

chars

0	→	1	1
1	→	3	2(1)+1
2	→	5	2(2)+1
3	→	7	2(3)+1

0 to < i

space

0	→	0	i
1	→	1	i
2	→	2	i
3	→	3	i

0 to < 2n-2i+1

chars

0	→	7	8-1	8-1
1	→	5	8-3	8-(2(1)+1)
2	→	3	8-5	8-(2(2)+1)
3	→	1	8-7	8-(2(3)+1)

Bitwise Operators

W2-L2

↳ use on bit level

And	(a & b)	1 if both bits are 1
Or	(a | b)	1 if any or both bits are 1
not	(~ a)	negate the result
xor	(a ^ b)	same values ⟶ 0
		diff. values ⟶ 1

~5

↳ 5 ⟶ 0...0101

~5 ⟶ 1...1010

↳ how compiler read this

↳ 2's complement

0....0101 ⟶ 1's complement

0...0110 ⟶ 2's complement

$\boxed{-6}$

So ~5 = 6

Left and right shift operators

<<

shift all bits to left

* by 2 (not in every case)

↳ if MSB is 1 and 2nd MSB is 0

>>

shift all bits to right

/ by 2 (not in every case)

↳ if MSB and 2nd MSB is 1

$a = a << b$ a left shifts, b times $\longrightarrow$ result $\longrightarrow a \times 2^b$

$a = a >> b$ a right shifts, b times $\longrightarrow$ result $\longrightarrow \dfrac{a}{2^b}$

b cant be -ve

↳ in case of -ve

↳ gives gv

a=5;

a= a<<1; a= 10

a=5;

a= a<<2; a=20

in left shift → filled with 0

in right shift → filled with 0 and 1

0 → in +ve no.

1 → in -ve no.

right shift in -ve number

-ve no. in memory → 1 . . . . .

↓ right shift

1 1 . . . .

↳ signed bit is used to fill the vacant bit

left shift in number where MSB is 1 and 2nd MSB is 0

no. → 1 0 . . → -ve no.

left shift ↳ 0 . . . .

↳ +ve no.

Pre- Post → Increament / Decreament Operator

pre- increament
↳ ++a
 ↳ first increament by 1, then use

post - increament
↳ a++
 ↳ first use then increament by 1

pre- decreament
↳ --a
 ↳ first decreament by 1, then use

post - decreament
↳ a--
 ↳ first use then decreament by 1

```
int a= 5;
cout << (++a) * (++a);
```

output -
49
↳ due to operator precendence

break and continue

break ⟶ exit from that loop

continue ⟶ skip that iteration

Variable Scoping -

```
int g= 25;                ——————>  global variable
int main (){
    int a;                ——————>  declaration
    int b= 5;             ——————>  initialization
    b = 10;               ——————>  updation
    //int b= 15;          ——————>  redefinition is not allowed
    int c= 7;
    g= 30;
    cout << g;            ——————>  30
    if ( true){
        int b=15;
        cout << b;        ——————>  15
        cout << c;        ——————>  7
        g=50;
        cout << g;        ——————>  50
    }
    cout << a;            ——————>  gv
    cout << b;            ——————>  10
    cout << c;            ——————>  7
    cout << g;            ——————>  50
}
```

Making global variable is very BAD PRACTICE

Operator Precedence

- → order of priority of operator
- → no need to remember
  - use brackets properly

Switch Case

can also have nested switch case

```
switch (expression) {

    case value1 :
        executing body 1
        break;
    case value2 :
        executing body 2
        break;
    :
    case value n :
        executing body n
        break;

    default :
        executing body
}
```

not mandatory

without break
↳ all below executing body will also execute

→ continue cannot be used in switch case
  ↳ can only use in loops

Function -

W2-L3

- program linked with well defined task
- why
  - reusable
  - readable
- without
  - bulky
  - lengthy
  - buggy if mistake in any place

syntax -

```
return type  functionname ( input  parameters){
        function  executing  body
}
```

void → empty / no value

```
int  main(){

      return  0;
}
```

returns 0 to Operating System

0 is used as means of successful execution

→ a cpp file cant have more than 1 main functions

Function Call Stack

function call ⟷ function invoke

Stack
└→ Last In First Out

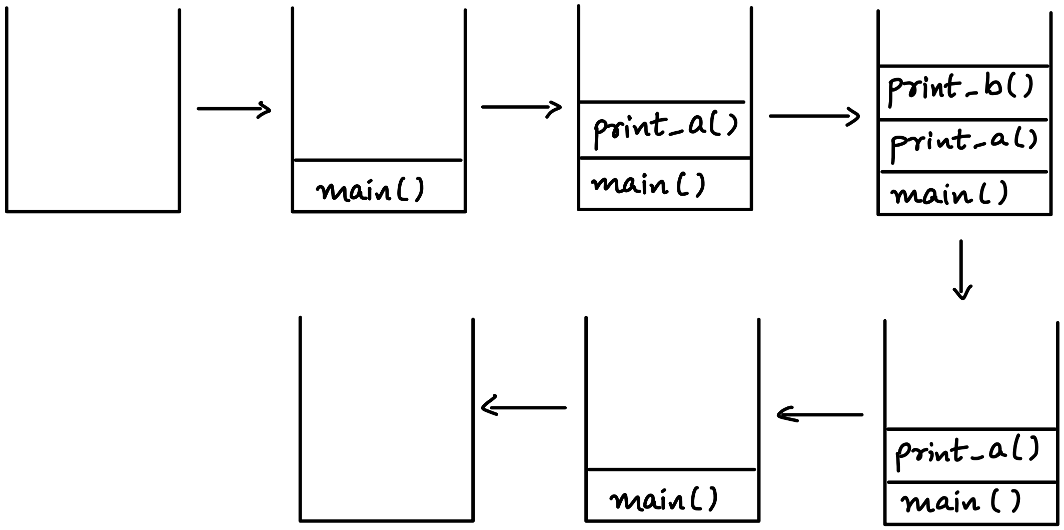
→ tells
- → what functions
- → which function calls which
- → local variables of function
- → return type of function

ex -

```
int main(){
    int a=5;
    print_a(a)
    return 0;
}
```

```
void print_a(int a){
    cout<< a;
    int b=3;
    print_b(b);
}
```

```
void print_b(int b)
{
    cout<< b;
}
```



Pass by value

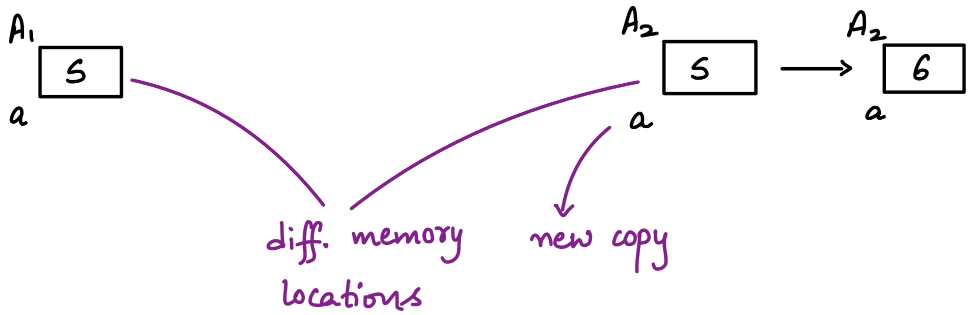
↳ a copy will be created of variables

```
int main() {
    int a=5;
    printNumber(a);
    cout<< a;
}
```

↳ argument

parameter

```
void printNumber(int a){
    cout << a;
    a++;
    cout <<a;
}
```



Address Of Operator &

```
int n=5;
cout << &n;
```

⟶ output - address of n

```
int main(){
    int a= 2;
    int b= 3;
    int sum= add(a,b);
    cout<< sum;
    return 0;
}
```

OS

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

7
sum
7
result

Function Order

Order 1

function declaration and definition

```
int add (int a, int b){
    return a+b;
}
int main (){
    int a= 3;
    int b= 5;
    int sum= add (a,b);
    cout << sum;
    return 0;
}
```

Order 2

function declaration

```
int add (int a , int b);

int main (){
    int a= 3;
    int b= 5;
    int sum= add (a,b)
    cout << sum;
    return 0;
}
```

function definition

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

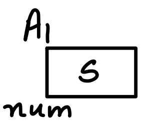
% operator ⟶ heavy operator

↳ so try to use it less

W2-L4

$$\begin{tikzcd} \text{BTS} \arrow[r, bend left] & \text{Behind The Scenes} \end{tikzcd}$$

```
int num=5;
```



Symbol table

$$\begin{tikzcd}
\text{int} \arrow[d] \arrow[dd, bend right] & \text{num} \arrow[d] \arrow[r, leftrightarrow] & A_1 \arrow[d] \\
\text{datatype} & \text{variable name} & \text{variable memory location} \\
\text{variable mapping with address is stored} & &
\end{tikzcd}$$

cout <<num;

$$\begin{tikzcd}
\text{symbol table} \arrow[r, "\text{go to}"] & \underset{\text{(address)}}{A_1} \arrow[r, "\text{print}"] & \underset{\text{(value)}}{5}
\end{tikzcd}$$

go to symbol table

while loop

```
int i=0;                 → initialization
while (i<5){             → condition
    cout<< i <<endl; }
    i++;             }   → body
}                            ↘ updatation
```

flow

$$\begin{tikzcd}
\text{initialization} \arrow[r] & \text{condition} \arrow[r, "\text{false}"] \arrow[d, "\text{true}"] & \text{exit} \\
& \text{body (containing updatation)} \arrow[u, bend left] &
\end{tikzcd}$$

left and right shift operators

```
int a= 2;
 a<<1 ;        ⟶ no change
 cout<< a;     ⟶ 2
 a= a<<1;      ⟶ change ⟶ left shift by 1
 cout <<a;     ⟶ 4
```

right shift in -ve no.

↳ link in links.txt in repo