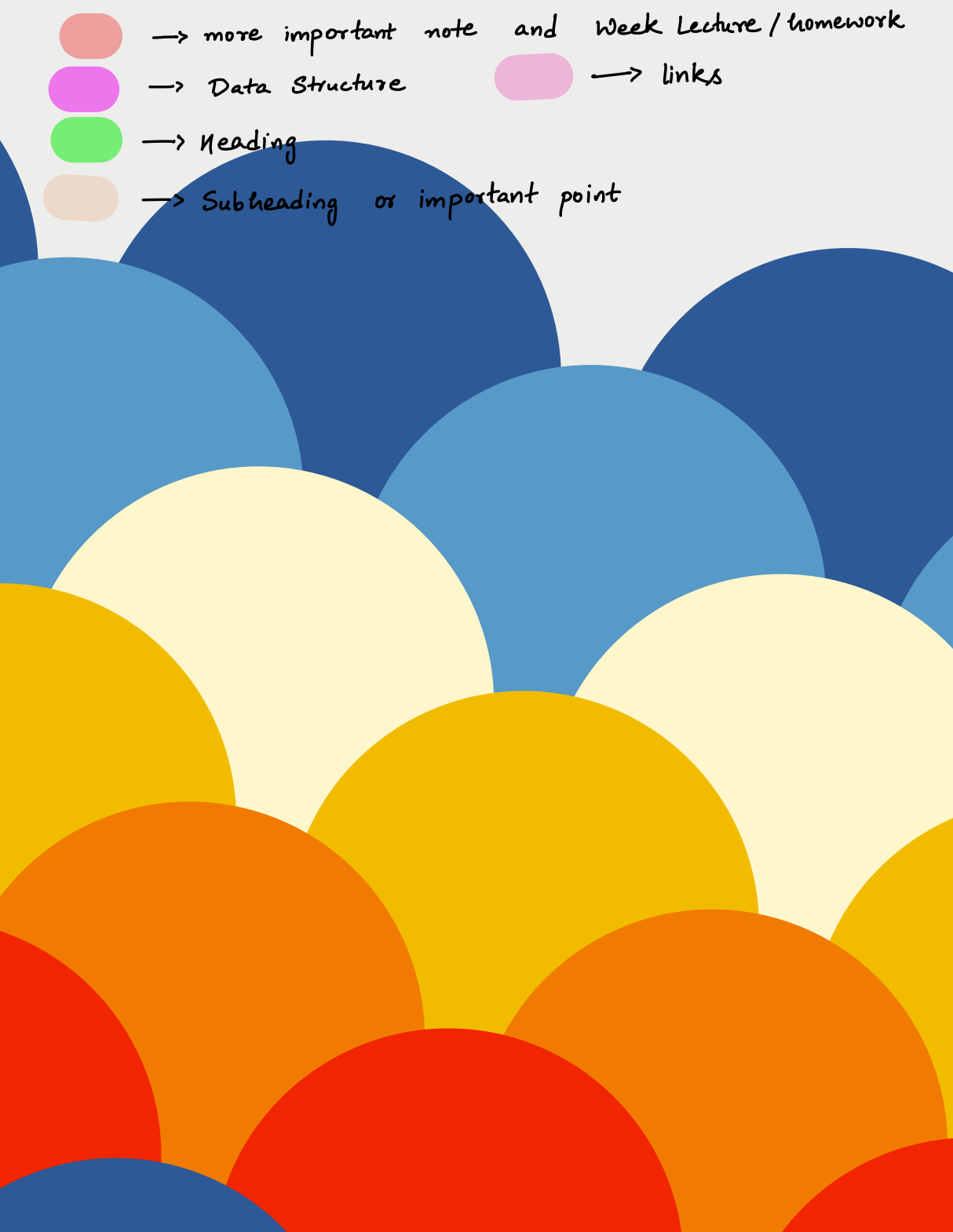

→ more important note and Week Lecture / homework
→ Data Structure
→ links
→ Heading
→ Subheading or important point

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 dismss with HR or team
- Think twice, Code once
- To clarify about any code you are confused, use cout statements every where 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
  - → so try to use it less

  Use bitwise operators instead of this if possible

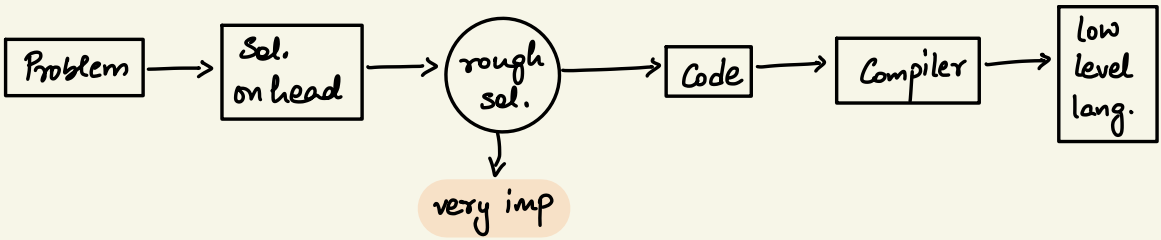
W1-L1

Thought process to solve a problem-

→ Understand a problem

→ input values

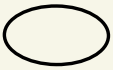
→ find approach

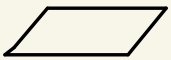


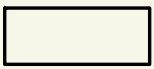
Algorithm- Sequence of steps

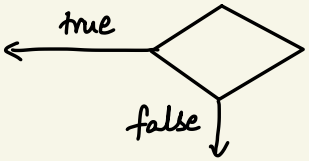
Flowchart- Graphical representation of algo

Components-

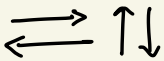

terminator for start /end

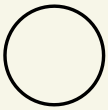

for input /output read /write


computation / process / declaration



decision making block takes condition


flow



Connector
takes function

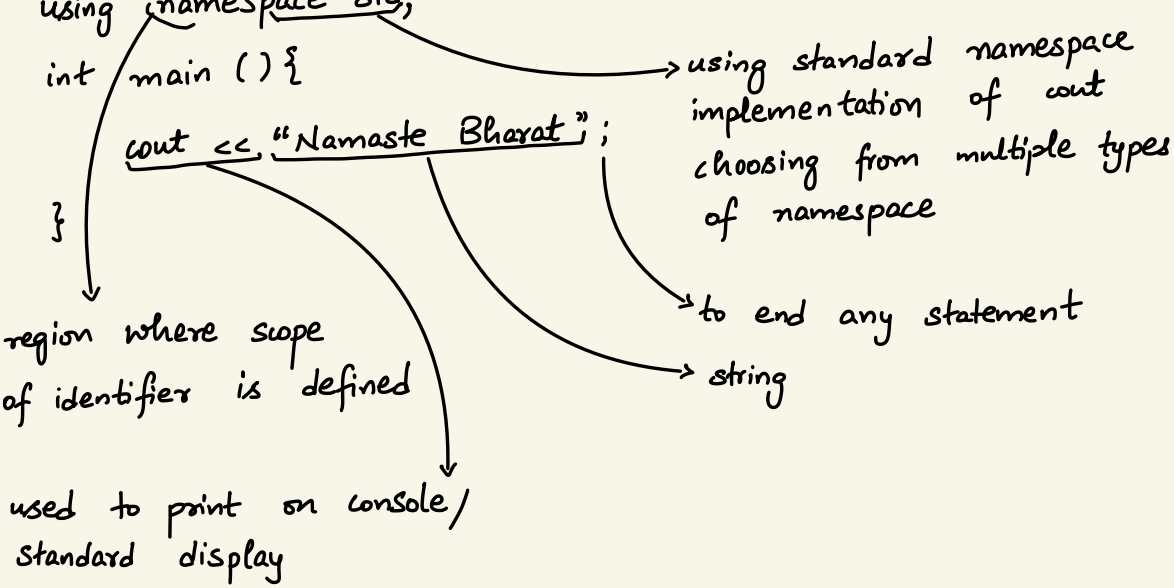
Pseudo Code- Generic way of writing algo

Dry Run → Very important to understand any topic

W1-L2

IDE- Replit, VS-Code

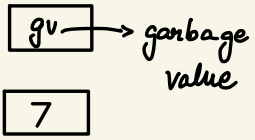
```
# include <iostream>
using namespace std;
int main (){
    cout << "Namaste Bharat";
}
```




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

→ for next line

int a ; ⟶ a is an integer

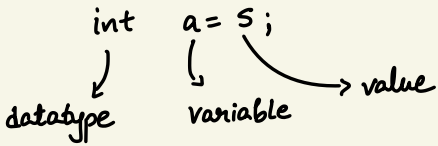


cin >> a ; ⟶

ex- 7

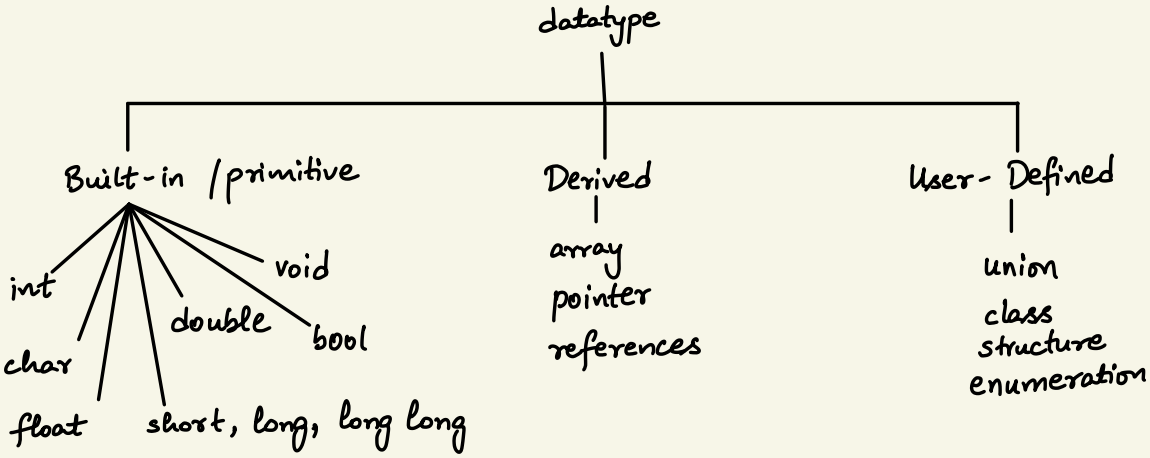
Variables

named memory location



Datatypes

type of data



int - 4 byte - 32 bits in memory

- ⟶ -2^{31} to $2^{31}-1$ in signed int
- ⟶ 0 to $2^{32}-1$ in unsigned int

char - 1 byte - 8 bits in memory

- ⟶ 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

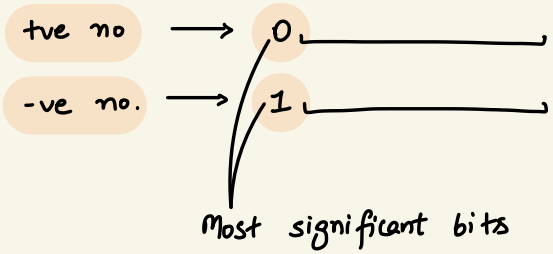
long long → 8 byte → 64 bits

short → 2 byte → 16 bits

long → 4 byte → 32 bits

How data is stored

int a = 5
↳ 32 bits 0...00101
29 bits



How -ve number is stored in memory

In 2's complement form

→ 1's complement +1

→ reverse all bits

int a = -7

7 → 0.....00111 } 32 bits

ignore -ve sign

find binary equivalant

1's (7) → 1.....11000

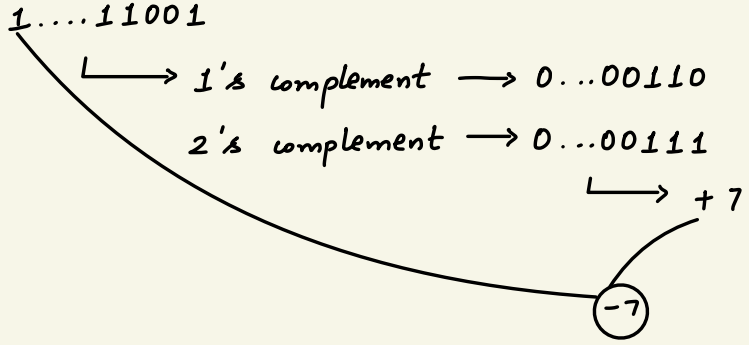
find 2's complement

2's (7) → 1.....11001

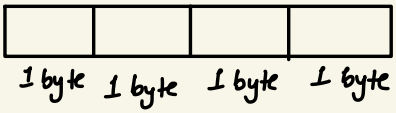
→ this is how -7 will be stored

How to read -ve no. present in memory

→ take 2's complement



Interesting problem



how computer know these are 4 chars or a single integer

↳ Using datatype

↳ tell 2 things

- → type of data used
- → space used in memory

Signed vs Unsigned

↳ 0, +ve

+ve, -ve, 0

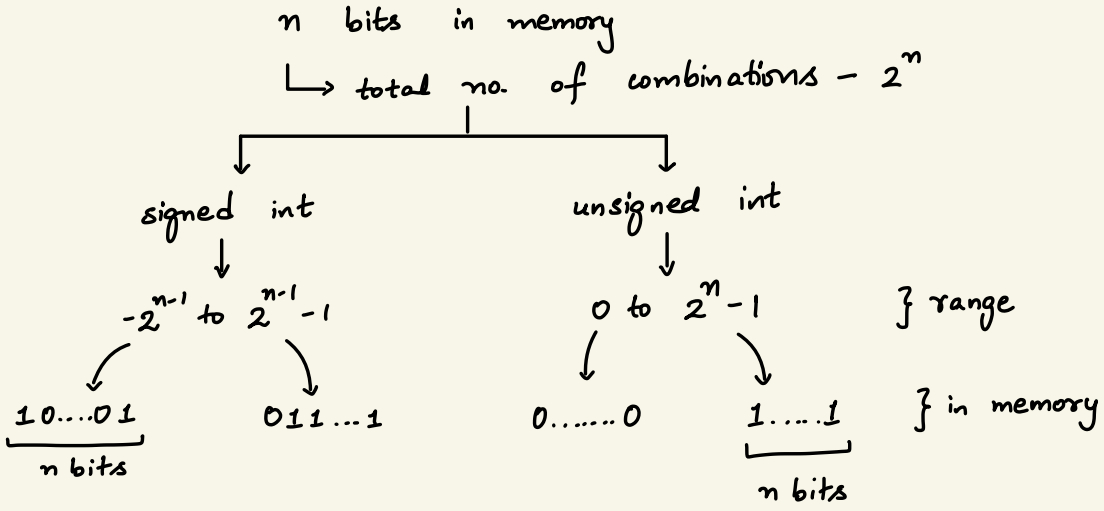
↳ by default

int - 4 byte - 32 bits in memory

↳ total no. of combinations - 2^{32}

- signed int
  - -2^{31} to $2^{31}-1$ } range
    - 10...01 (by using 2's complement of 2^{31})
    - 011...1 } in memory
- unsigned int
  - 0 to $2^{32}-1$ } range
    - 0.....0 } in memory
    - 1.....1

General Formula



Typecasting

→ convert one type of data to another

implicit typecasting

```
ex- char ch= 97;
    cout << ch;
```
→ (a)

explicit typecasting

```
ex- char ch = (char) 97;
    cout << ch;
```
→ (a)

overflow ex-

```
char ch = 9999;
cout<< ch;
```

9999 → (binary conversion) 100 111 0000 1111

stores only last 8 bits

so ch stores 00001111 in memory

↓

7

↓

acc. to ASCII table

Operators -

Arithmatic Operator

↳ +, -, *, /, %

int op int → int

float op int → float

int op float → float

float op float → float

double op int → double

int op double → double

double op double → double

3 → int

3.0 → float / double not int

Relational Operator

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

Output - 0 or 1

0 → false

1 → true

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

false ← 0 , 1 → true

(cond1 && cond2 && cond3)

↳ if cond1 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 (cond 1)
    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

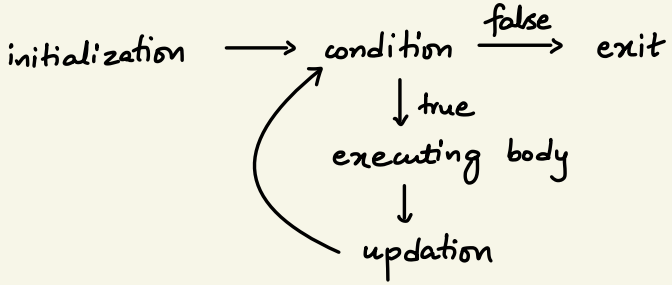
↳ to do something repeatedly

for - loop

initialization → int i=0 ; condition → i<5 ; updation → i=i+1

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

flow -



initialization
condition
updation

} none is mandatory
one or multiple can be added

patterns -

Generally 2 loops ⟶

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

→ a op= b ⟶ 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

HLL - High level language

↳ human readable and user friendly

W1-L4

C++, C - Middle Level language

namespace ⟶ to avoid collision

↓

multiple definitions of a single keyword

heirarchy

↳ various namespaces

↓

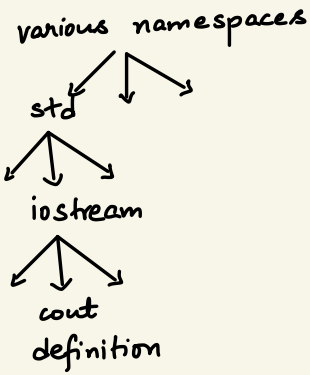
std namespace

↓

iostream preheader file

↓

keyword definition




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

⟶ output -

102 or 102.0

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 -

↳ syntax

W1-HW

variable = (condition) ? expression2 : expression3

(condition) ? variable = expression2 : variable = expression3

patterns

W2-L1

↳ how to think

- finding formula for rows and cols

n=5

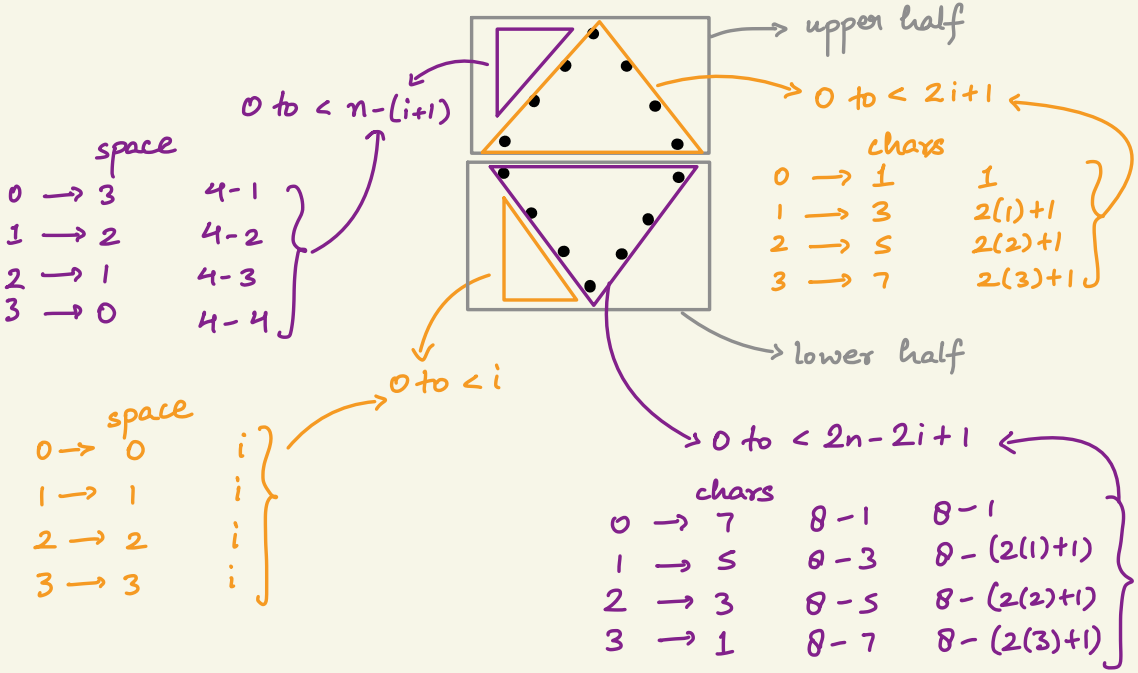
row	stars
0	0
1	0
2	1
3	2
4	3

→ formula- 0 to < n-1

n-1	no. of times loop runs
-1	0
0	0
1	1
2	2
3	3

as condition fails (0<-1)

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



W2-L2

Bitwise Operators

↳ 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

(-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 -ve numbers)

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

a >> b a right shifts, b times → result → $\frac{a}{2^b}$

b cant be -ve

↳ in case of -ve

↳ gives gv

```
a=5;
a<<1;        a=10

a=5;
a<<2;        a=20
```

right shift in -ve number

-ve no. in memory → 1 .....

↓ right shift

01....

↳ very large +ve number

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

no. → 10. . → -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

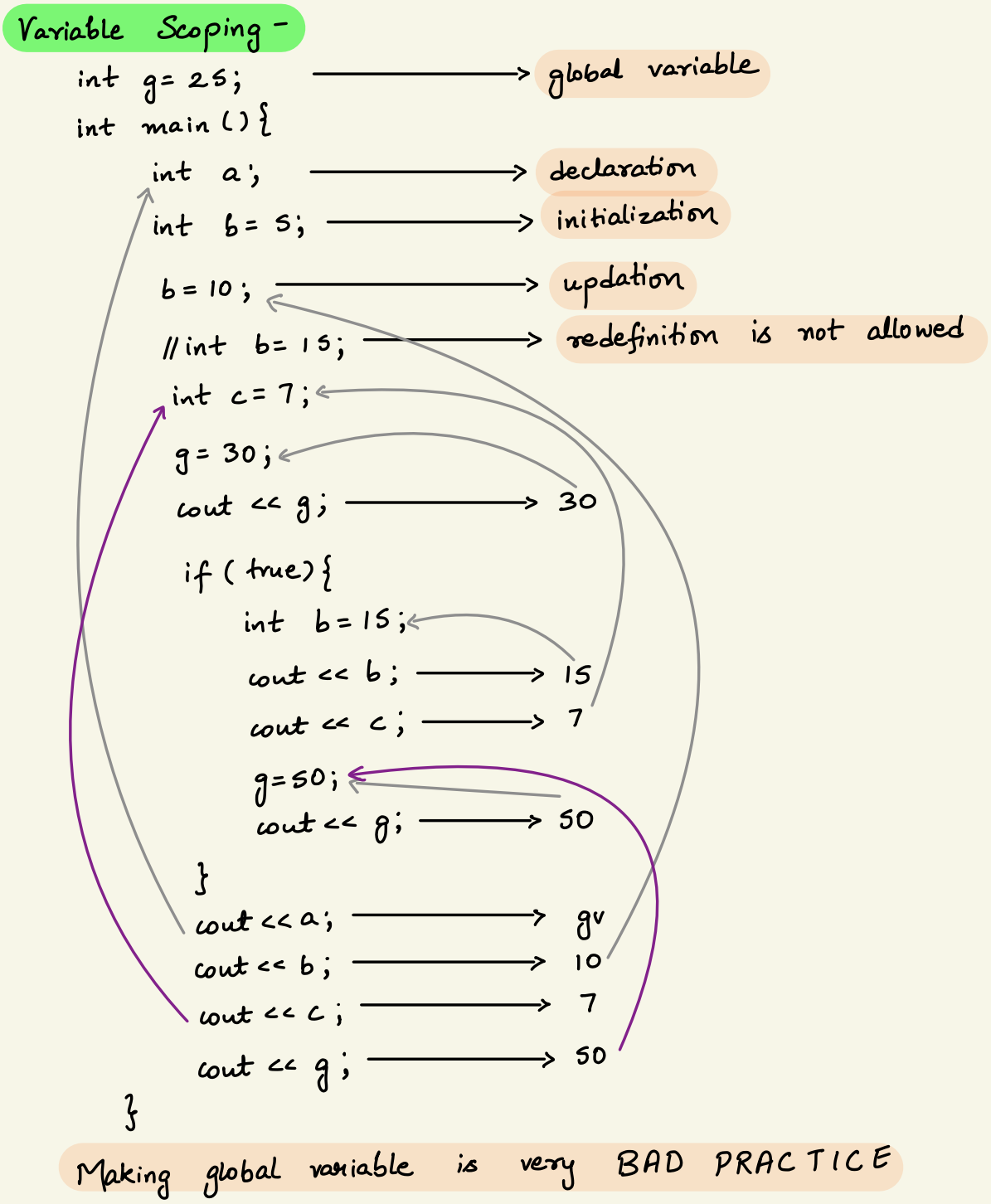
break and continue

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

W2-L3

Function -

- 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;
}
```

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

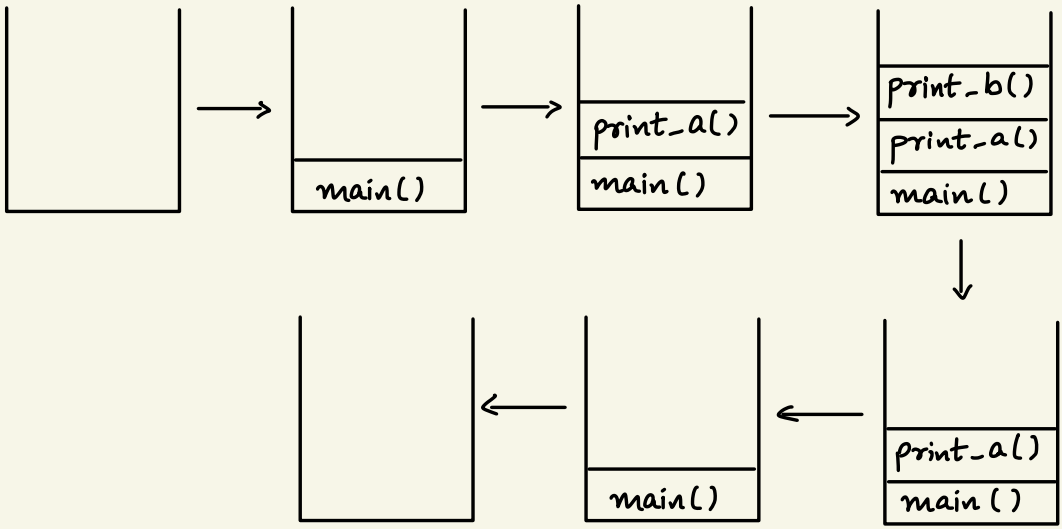
tells
- what functions
- which function calls which
- local variables of function
- return type of function

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

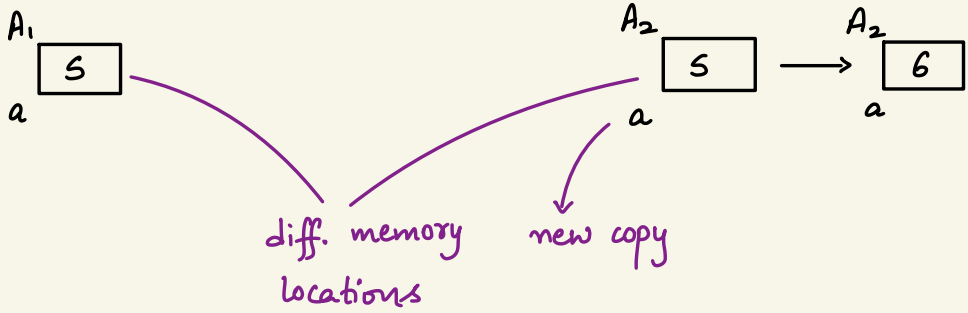
↳ a copy will be created of variables

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

argument

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

parameter



Address Of Operator &

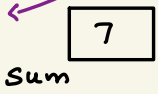
```
int n = 5;
cout << &n;
```

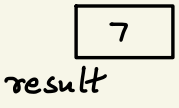
⟶ output - address of n

```
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;
}
```





Function Order

Order 1

```
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;
}
```

function declaration and definition

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

function definition

% operator ⟶ heavy operator

↳ so try to use it less