Data Structures & Algorithms

PRANIT
MODI

Data Structures & Algorithms

⟹ Problem Solving

Problem
- Understand
- Given Values
- Approach of solving

↓

Program Creation

Problem ⟶ Solution

Rough Soln ⟶ flowchart / pseudocode

↓ Code

Program in High Level Language / src code

↓ Convert

Machine Understandable format

Flowchart

↳ diagramatic representation of an approach

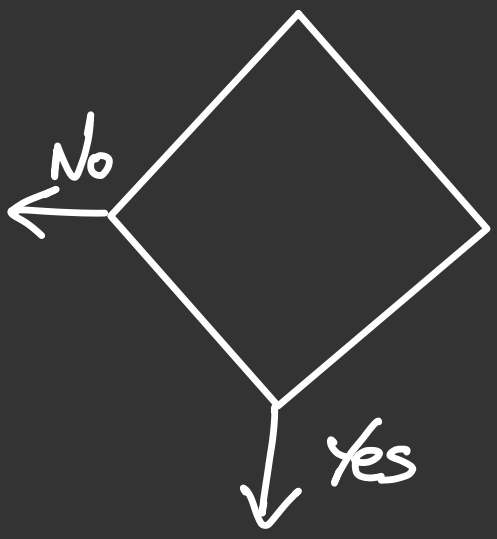
Components of flowchart

→ Terminator

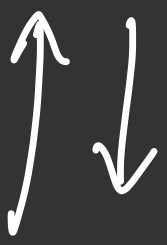
↳ Start / End

→ input / output

→ process



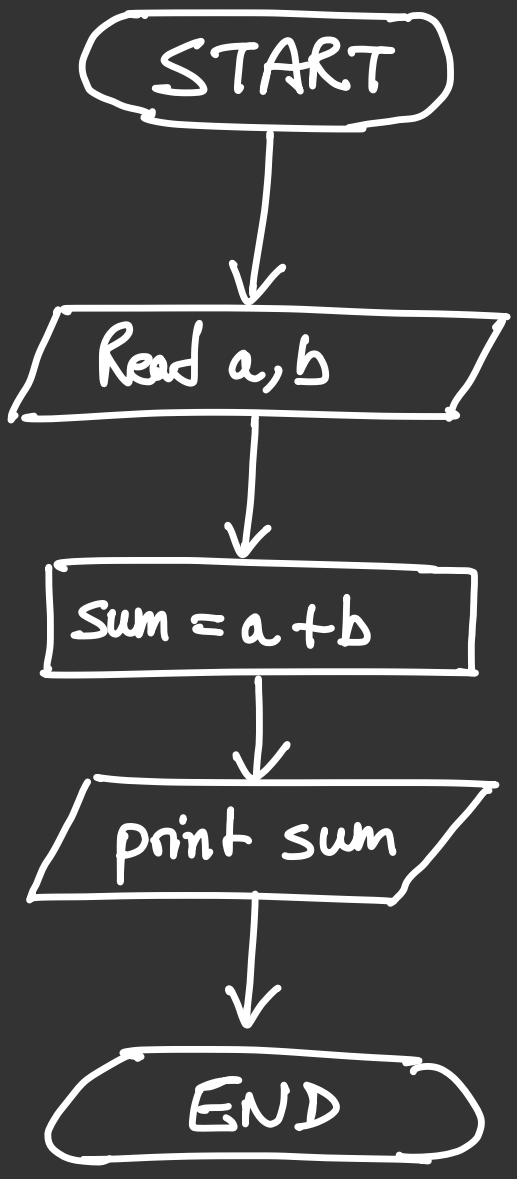
→ Decision Making



Arrows
(to showcase the flow and connect components)

Connector

eg: Sum of two numbers given a, b



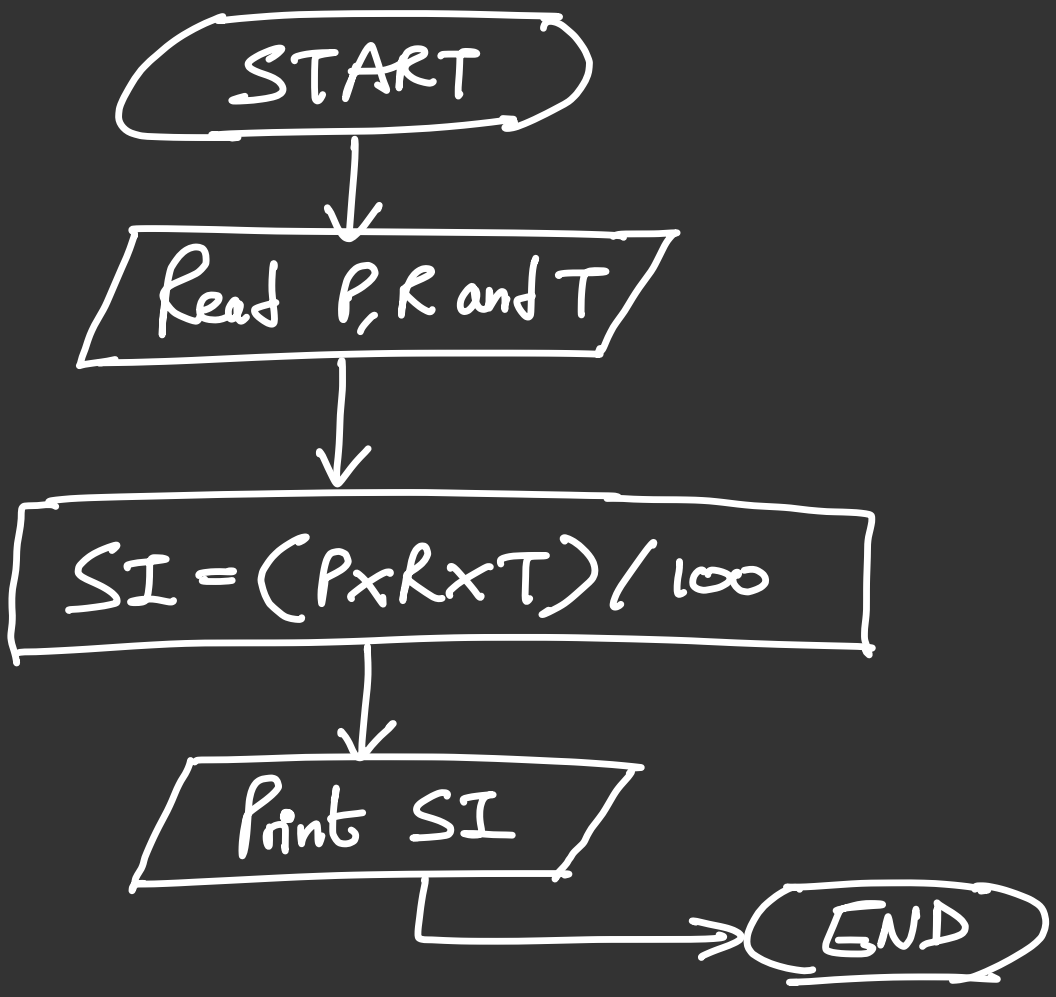
Pseudo Code

- way of representing logic
- generic irrespective of programming language

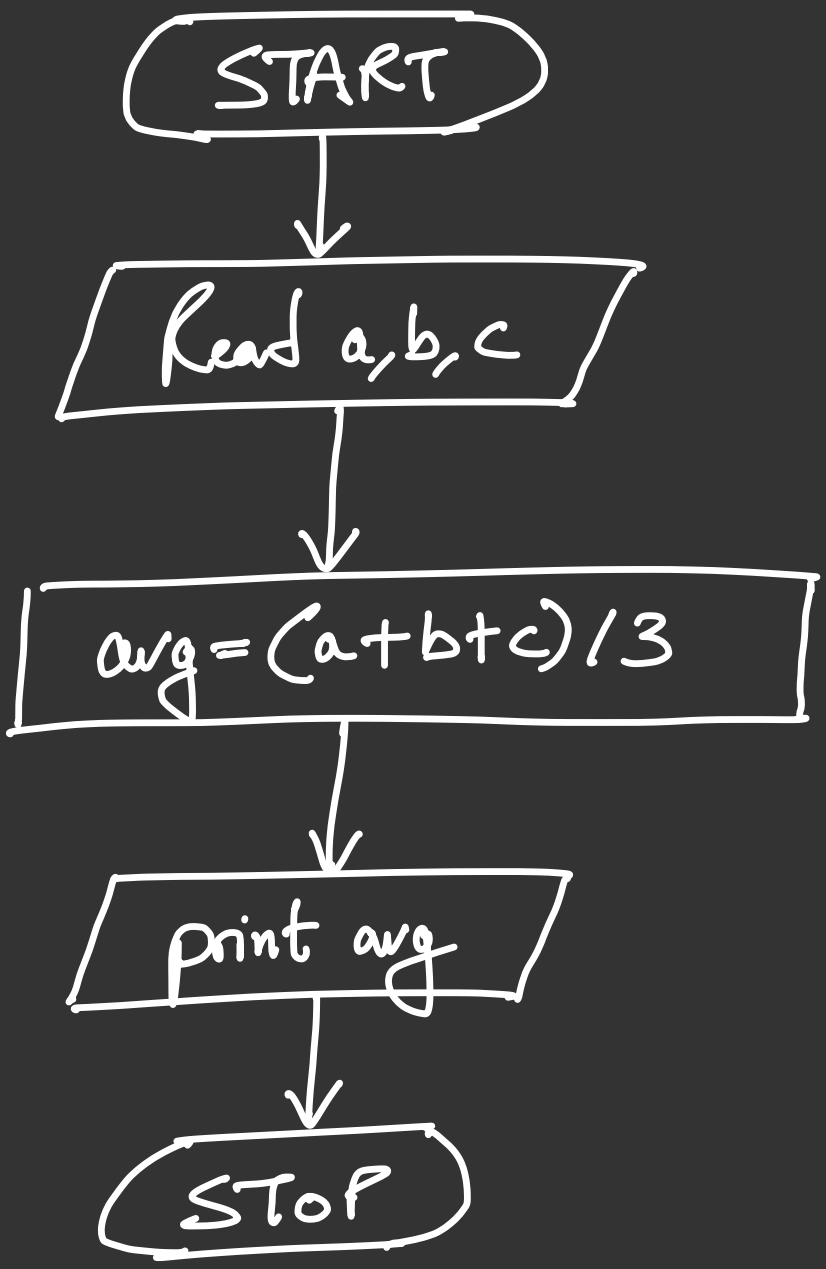
eg: Sum of two numbers

1. Read a, b
2. sum = a + b
3. print sum

Question : Flow chart for Simple Interest



eg: Average of three numbers

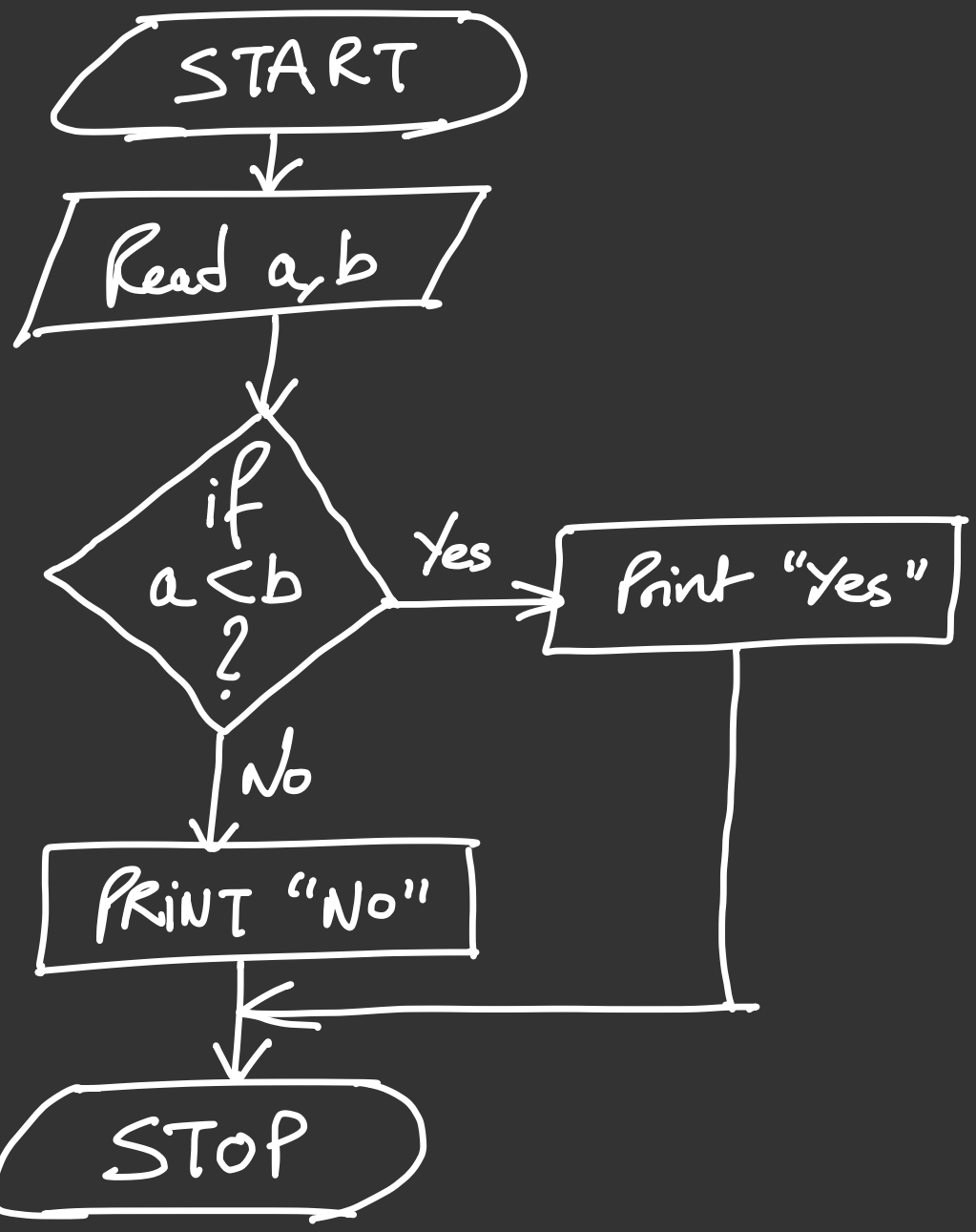


→ Read a,b,c

→ avg=(a+b+c)/3

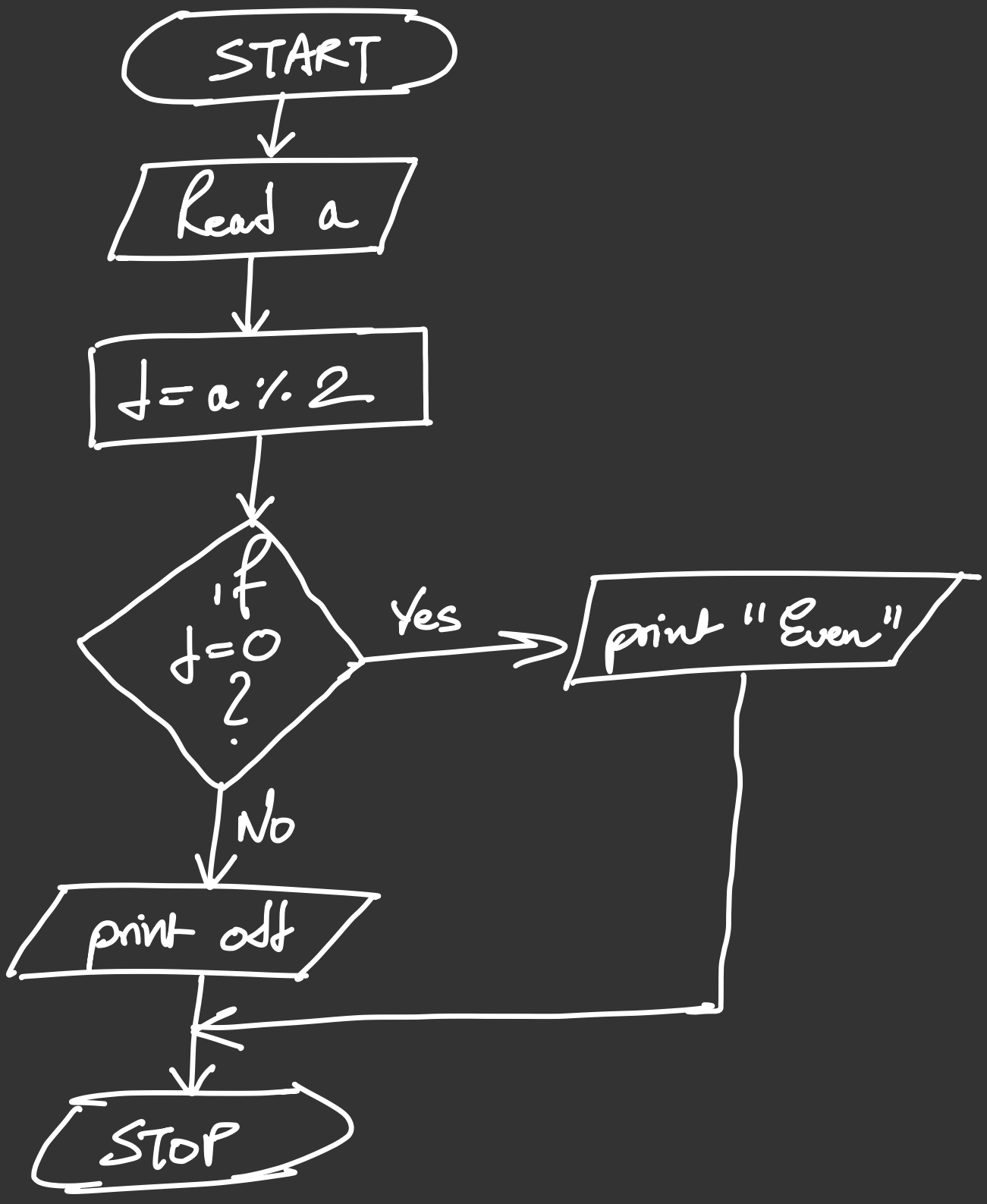
→ print avg

Question: Check if a<b ?

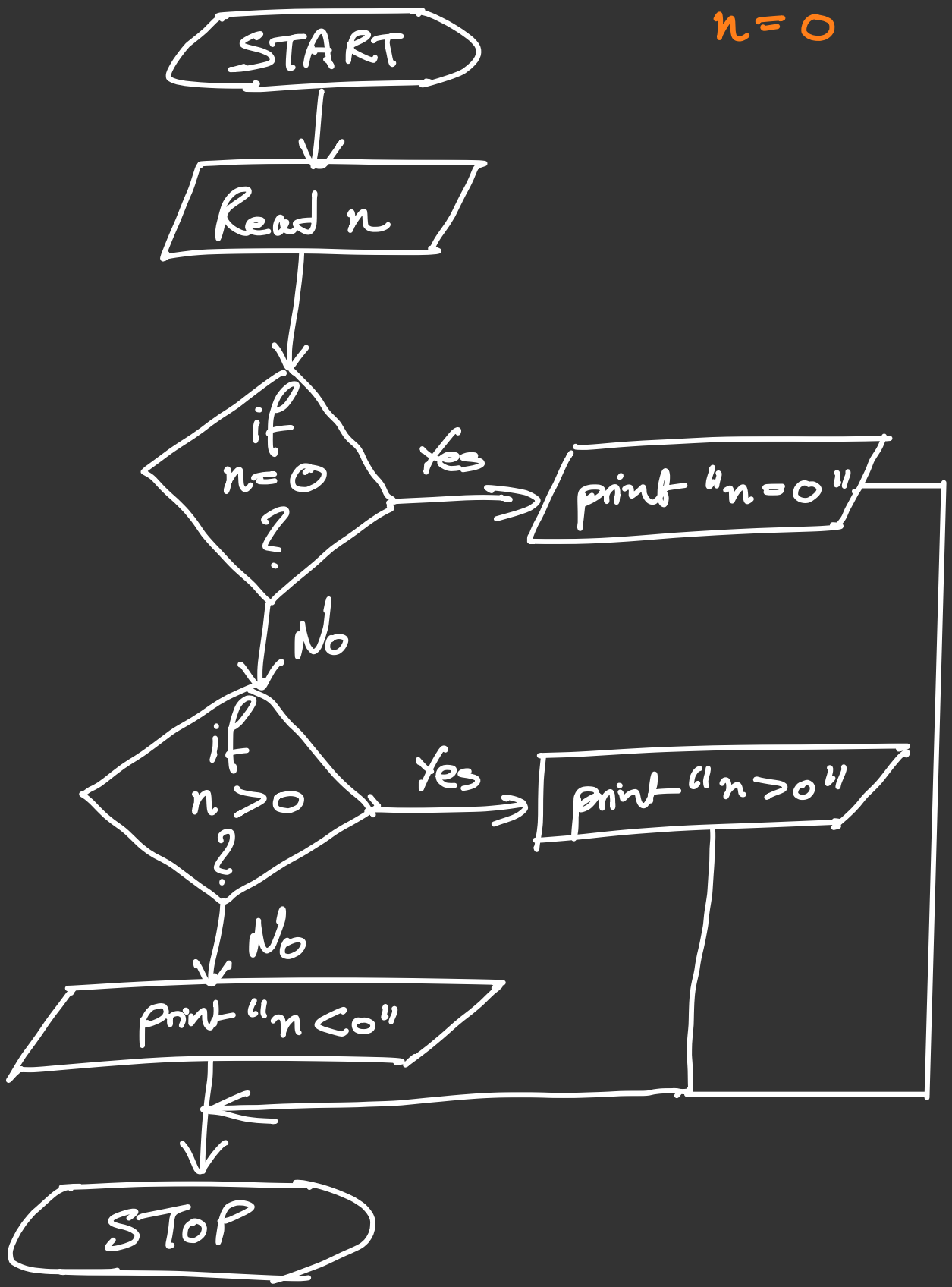



```
Read a,b
if a<b
    Print "Yes"
else
    Print "No"
```

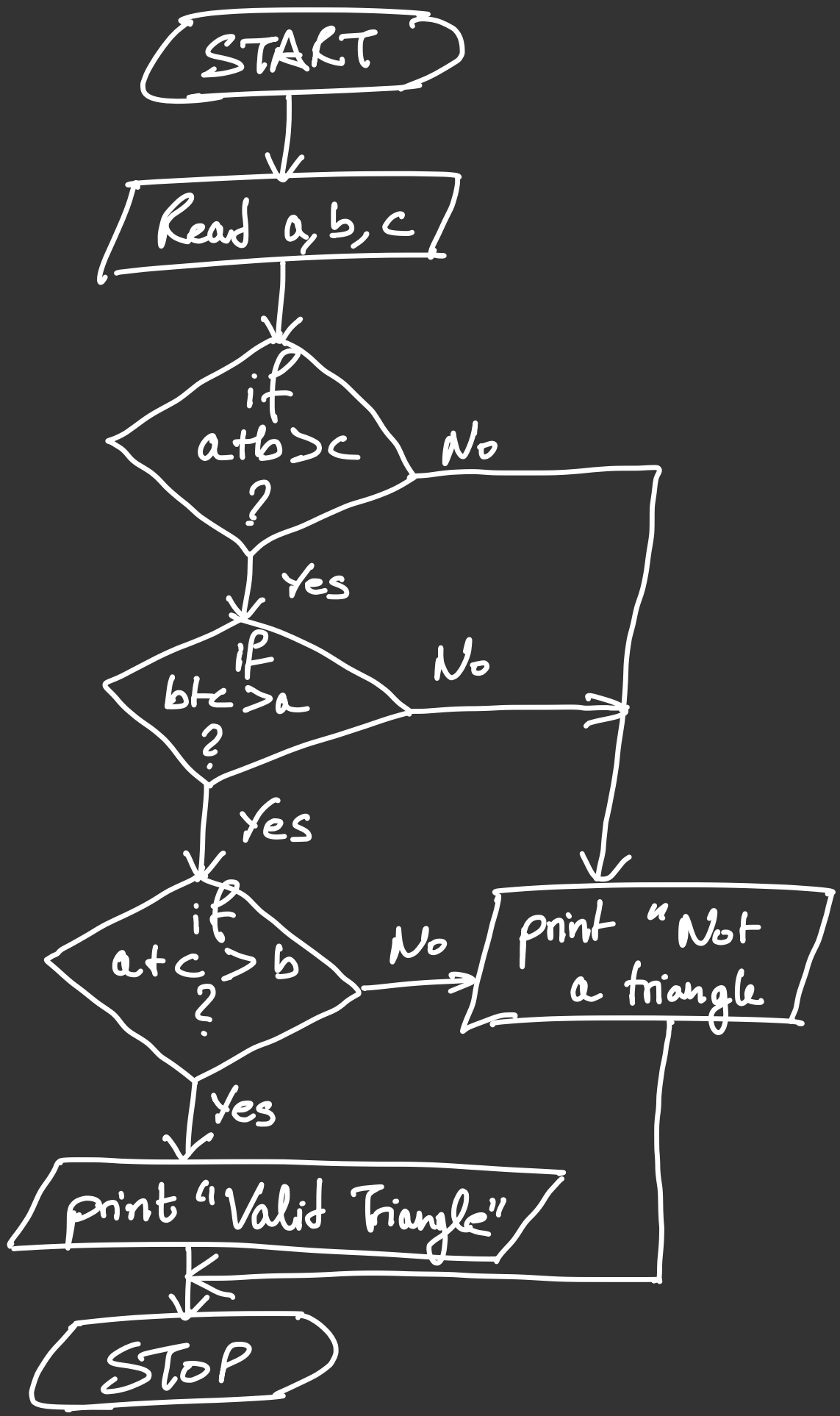
Question: Check if a number is even or odd



Question : Check if the number is
$n > 0$
$n < 0$
$n = 0$

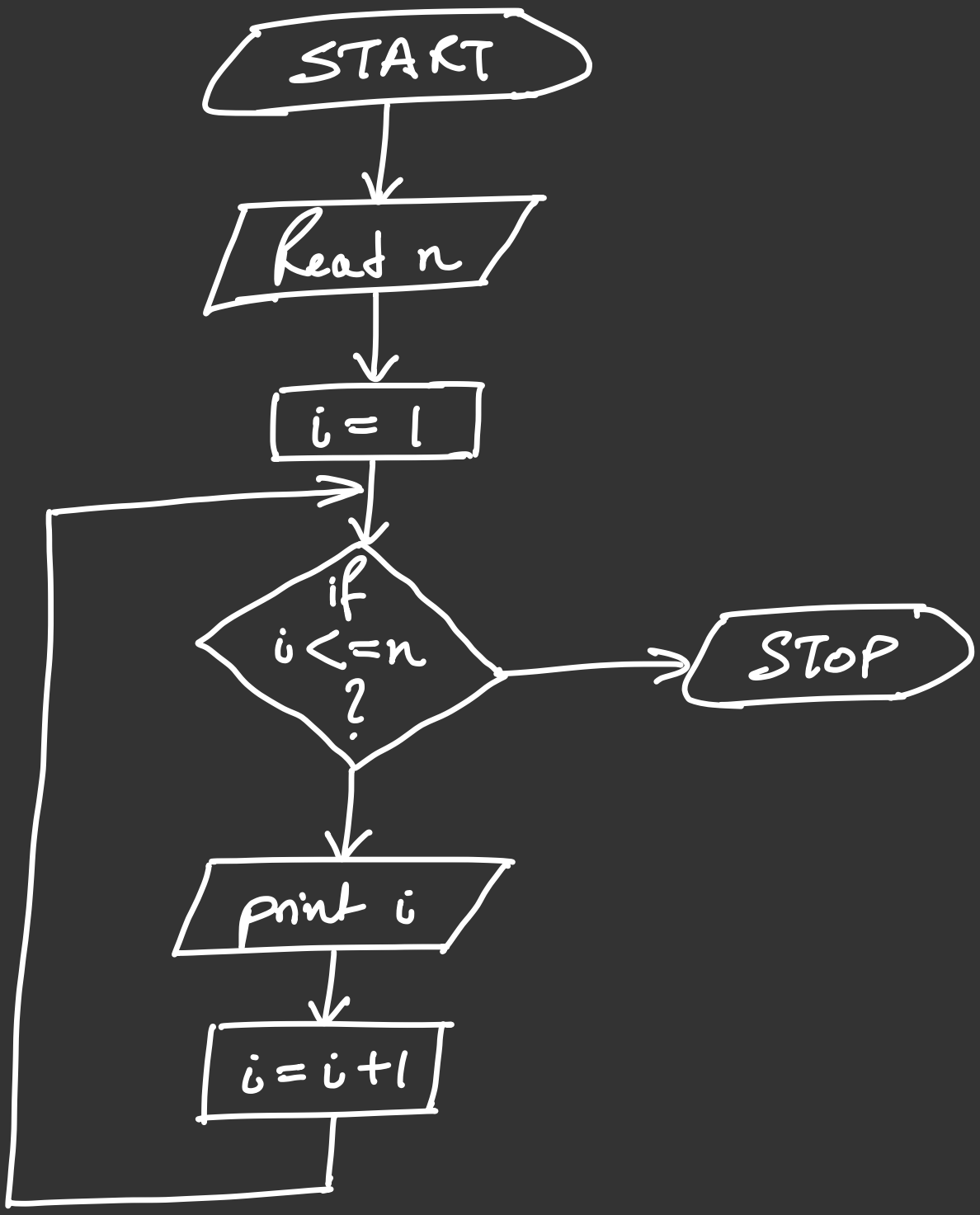


Question: Sum of two sides greater than the third → Check for Validity

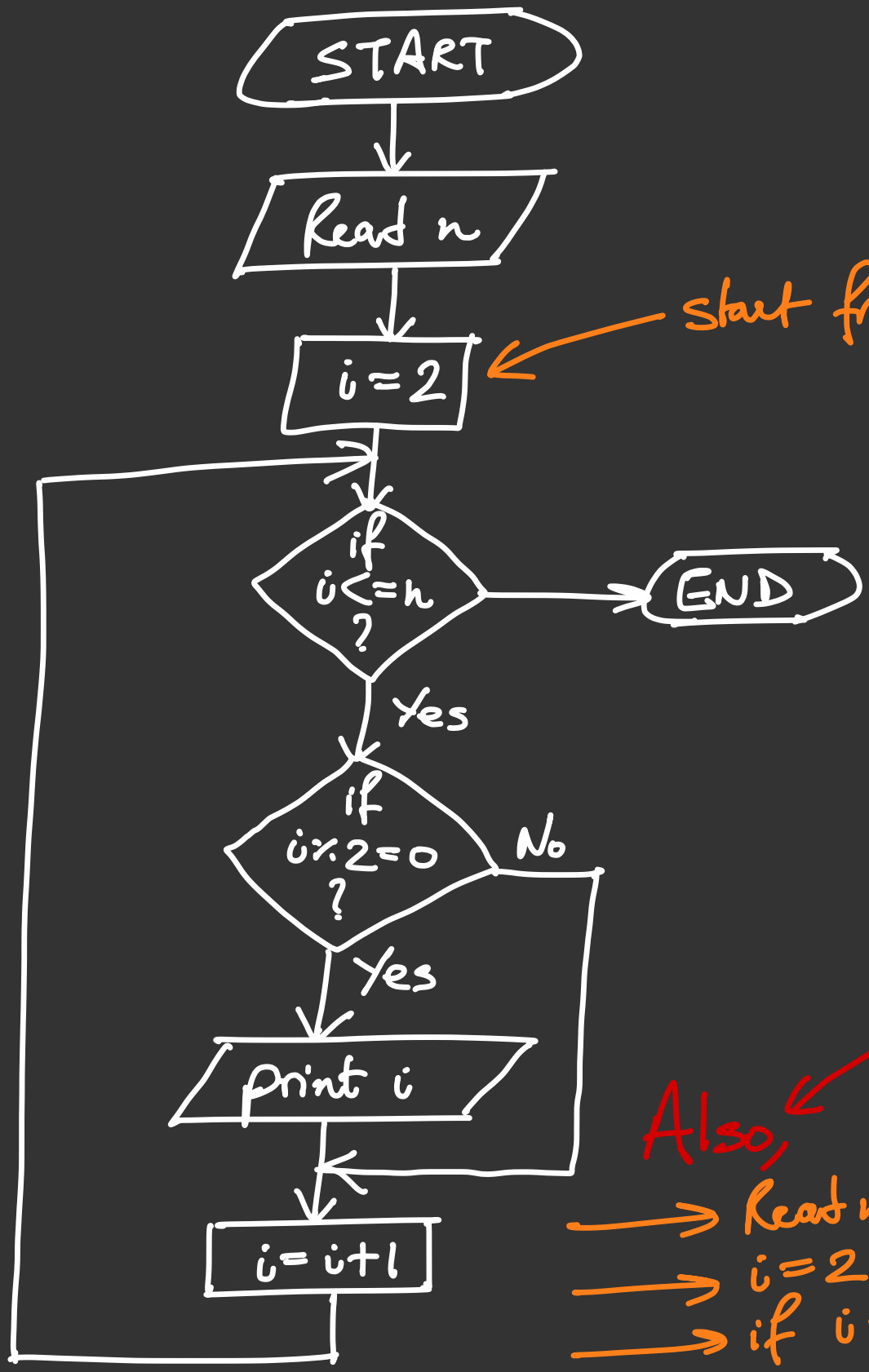


Loops

Question : Print 1 to N



Question: Print all even numbers from 1 to n.



Also,

n is exclusive

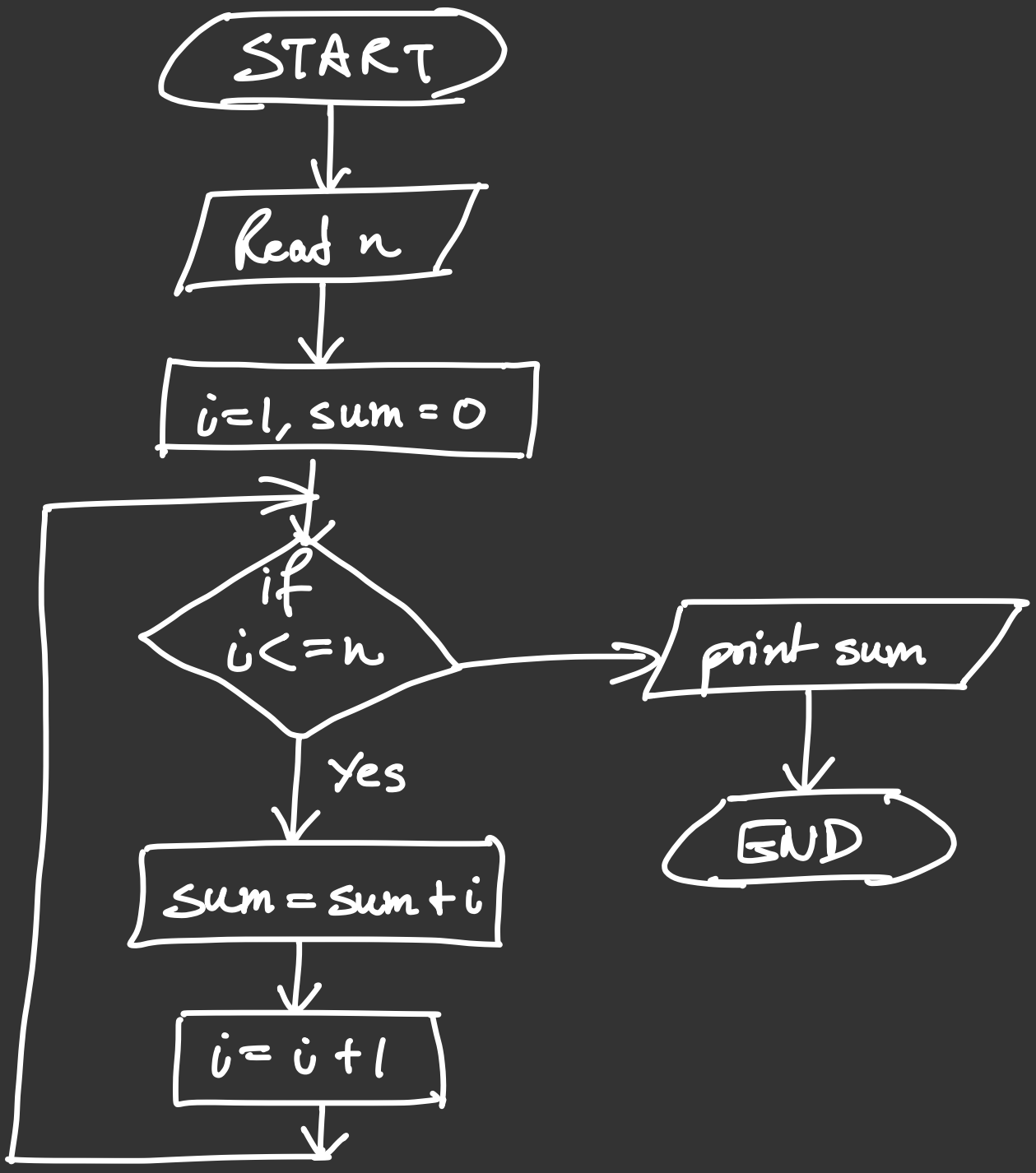
→ Read n

→ i=2

→ if i<n
print i
i = i+2

→ else Stop

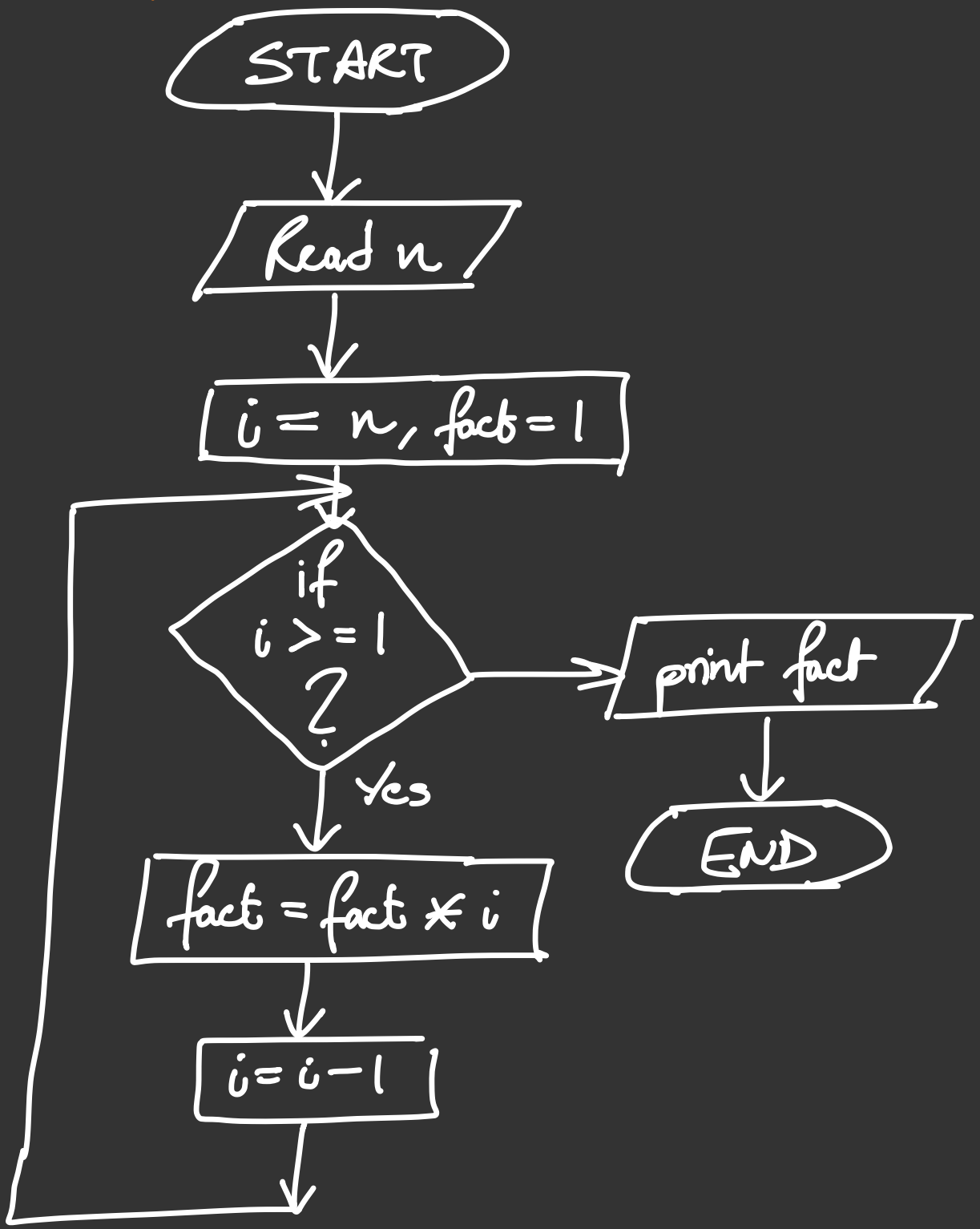
Question : find sum 1 to n (Inclusive)



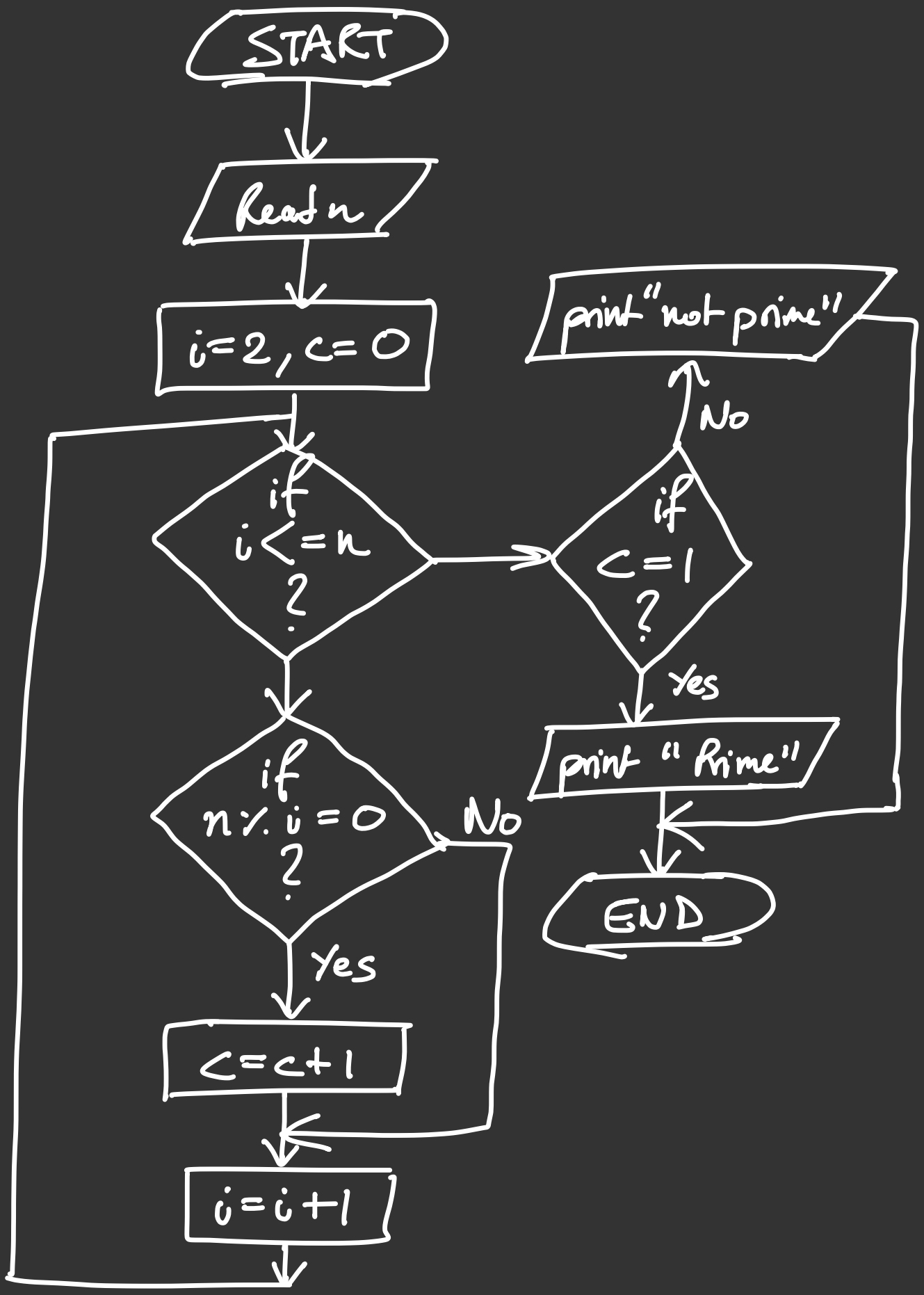
Question: factorial of n

eg: n = 5

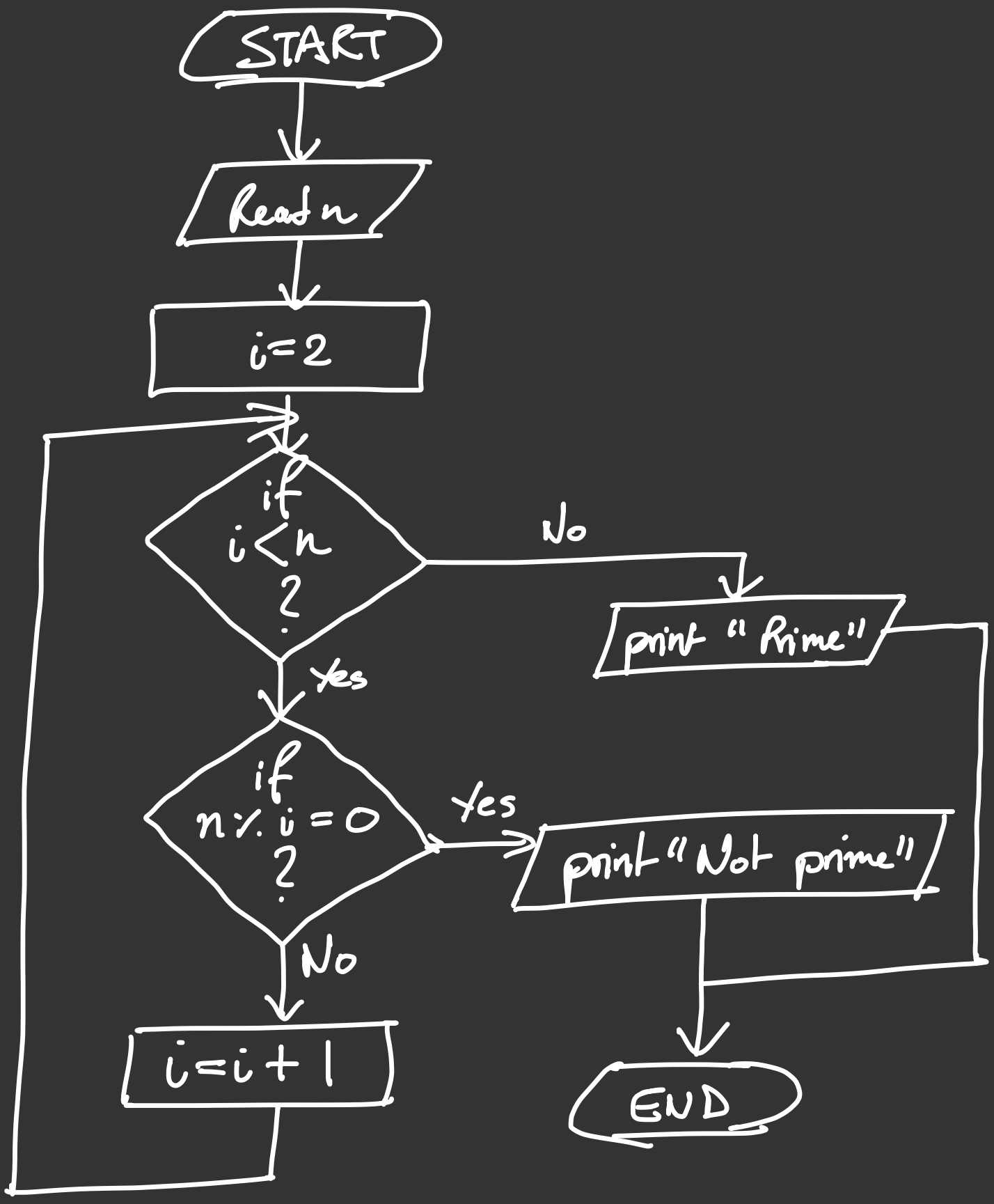
fact = 5 x 4 x 3 x 2 x 1



Question : Check for prime



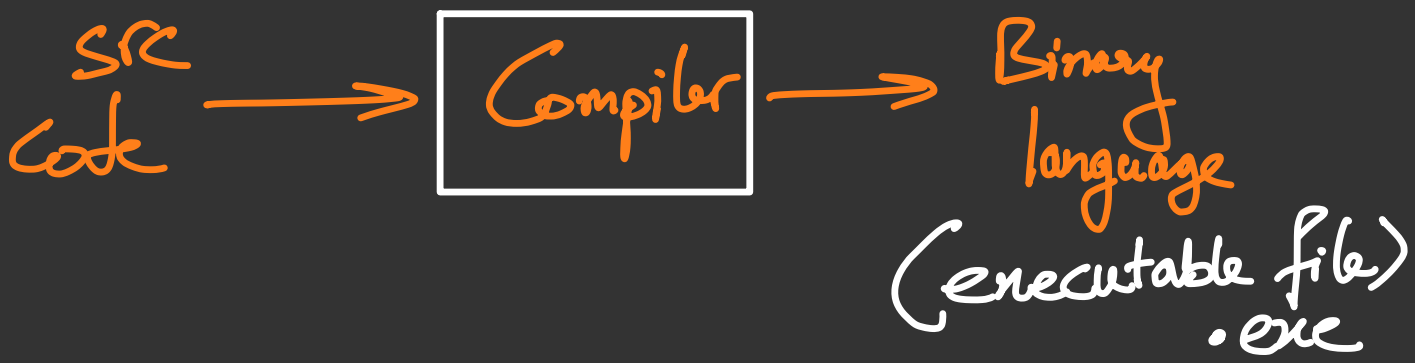
Better Logic →



Programming Languages ?

What we write
- source code
- programs

What computer understands
- 0 / 1
- Binary Language



→ a way to communicate with a computer.
eg: C, C++, Python, R, Java

→ Every language must be written following some rules called Syntax of that language

Lecture 2:

Source code → Compiler → Machine executable code (Binary Language)

Compiler

- Translation from src code to binary code
- find errors

IDE → Integrated Development Environments

Programs in C++

START → int main()
{
≡≡≡
}

Printing → cout << "Namaste Dunia"
<< endl

Default Code:

```
#include <iostream>
using namespace std;
int main()
{
    ____
    ____
    ____
    ____
}
```

→ 'iostream' is a file which includes cout

→ We want to use the namespace which is std, C++ has multiple namespaces.

→ for output
→ to display / write.

endl → takes cursor to the new line.
→ Other way, '\n'

☆ cout

eg: cout << "Namaste Duniya \n";
cout << "Namaste Duniya" << '\n';
cout << "Namaste Duniya" << endl;

Semicolon (;) → to end the code line

Data Types and Variables

Integer: int → 4 bytes = 32 bits

```
int a = 5;
```

- a — variable name
- datatype — integer
- memory needed = 4 bytes

Character: ch → 1 byte = 8 bits

- within single inverted commas.

```
char ch = 'a';
ch ≠ 'ab'
```

Boolean: bool

- True, 1
- False, 0

```
bool b = 1;
```

→ 1 bit

→ While printing, if
- true → 1 is printed
- false → 0

Float : float ⟶ 8 Bytes = 64 bits

float f = 1.2;

Double : double ⟶ 8 Bytes

double d = 1.23;

Variable Naming Rules

⟶ Dont start with numbers

⟶ You can use numbers, alphabets, underscore

⟶ You can start naming with alphabets or underscore.

☆ sizeof()

↳ returns the size of the variable

eg: int size = sizeof(a); a ⟶ int

Thus, size = 4

How Data is Stored?

```
int a = 8;
```

$8_{(10)} \longrightarrow 1000_{(2)}$

↓

4 bits

but memory allocated = 32 bits

8 bits | 8 bits | 8 bits | 8 bits

↳ how stored in memory

* for negative number?

ASCII Chart

space = 32

A - Z = 65 to 90

a - z = 97 to 122

0 - 9 = 48 to 57

fullstop (.) = 46

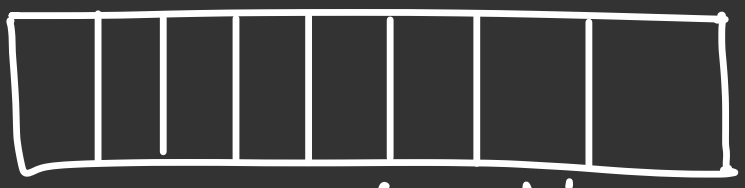
Comma (,) = 44

Colon (:) = 58

Semicolon (;) = 59

In Memory,

Character's ASCII value is converted to binary and that is stored in Memory.



8 bits = 1 byte

* Datatype helps us to identify that the given bytes in the memory are what.

Type Casting

→ char to int

```
int a = 'a';
cout << a << endl; #97
```

→ int to char

```
char ch = 98;
cout << ch << endl; #b
```

What if we wants to type cast from int to char but the value is too large to be stored in char?

int → 4 byte = 32 bits

$2^{32} - 1$ = max

0 = min

char → 1 byte = 8 bits

$2^8 - 1$ = max

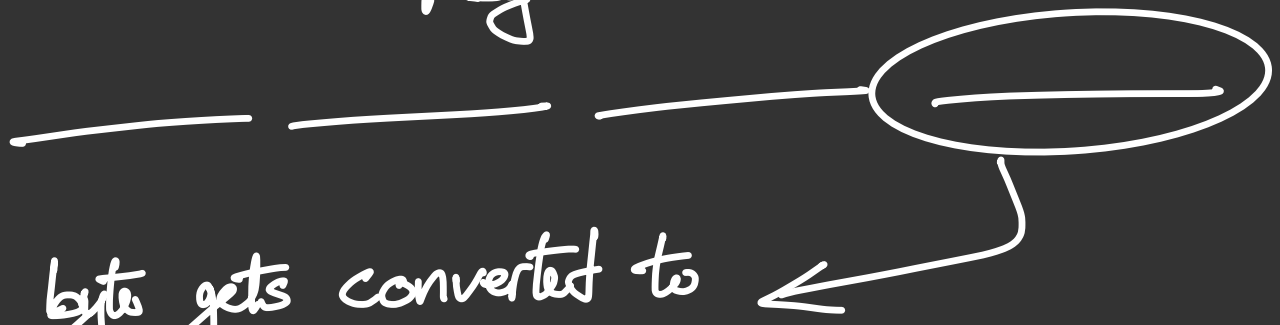
0 = min

eg:

char ch = 123456;

↓

4 bytes



last byte gets converted to char, which is coming to be 64 in this case.

Thus, 64 ⟶ @

Cout << ch << endl; #@

How negative numbers are stored?

First Bit

↳ 0 - +ve

↳ 1 - -ve

eg: -5

Storage of negative 5

1. Ignore the -ve sign
2. Convert into binary
3. Take 2's complement and store it

5 → 00000000 00000000 00000000 00000101

1's compliment

→ 11111111 11111111 11111111 11111010

2's compliments

→ add 1 to 1's compliments

11111111 11111111 11111111 11111011

This will be stored

To display the stored value →

① Take 2's compliment of the stored number

1 1111111 11111111 11111111 11111011

↳ this tells us negative value

for 2's compliment, first do 1's compliment

→ 00000000 00000000 00000000 00000100

2's compliment

→ 00000000 00000000 00000000 00000101

↳ 5

& negative sign above

⟹ −5

Storage to Integers

(i) Negative

→ Ignore the -ve sign
→ Convert the number to binary
→ Take 2's compliment
  ↳ 1's compliment
  ↳ Add one in the end

(ii) Positive

→ Convert it into Binary
→ Stop

✱ Thus, while displaying a stored value
→ if first bit is 1_2 → negative
→ if first bit is 0_2 → positive

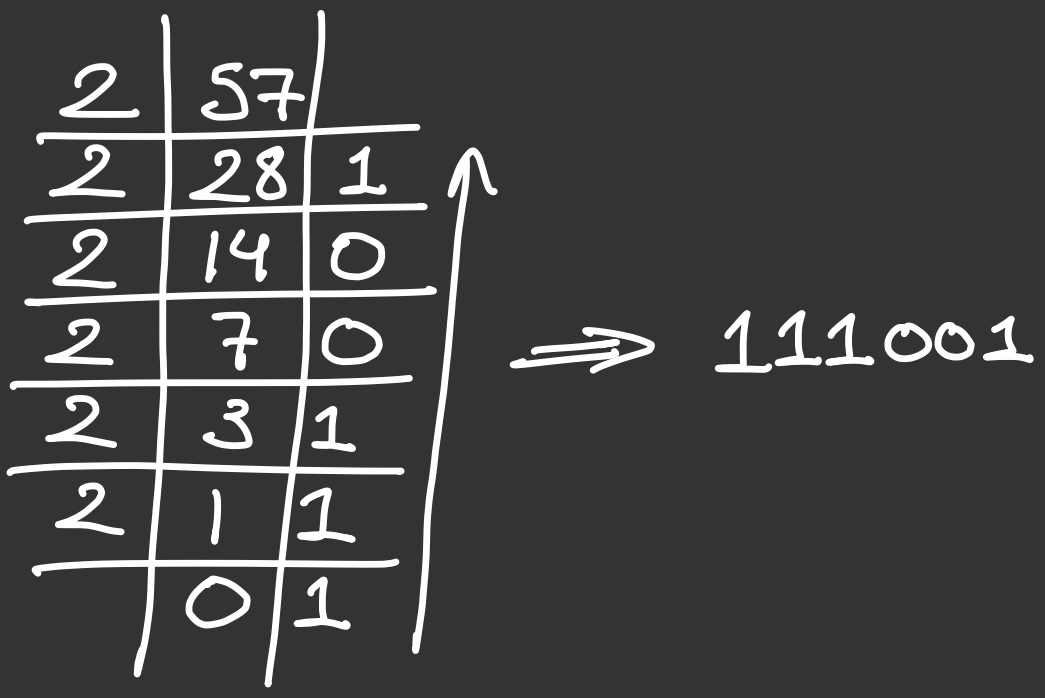
Converting Decimal / Integer to Binary

eg: 57

Method 1:

0	0	1	1	1	0	0	1
128	64	32	16	8	4	2	1

Method 2:



* Why 2's compliment ?

Integer range is
↳ $-(2^{31}-1)$ to $2^{31}-1$

But, in here
→ 0 and -0 both are counted

So, to treat both of them as one value → 2's Compliment

New Range : $-(2^{31})$ to $2^{31}-1$

* Unsigned Values

↳ Range : 0 to $2^{31}-1$

eg: unsigned int i = 112; ✓
unsigned int i = -112;
cout << i << endl; # big value

→ it will store as negative value that is in 2's compliment, but Compiler will treat it as unsigned value thus converts the whole thing to its equivalent positive decimal number.

Operators

int / int = int

double / int = double

float / int = float

eg:

```
int a = 5/2; //2
double d = 5/2; //2
double d1 = 5/2.0; //2.5
```

Relational Operators

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

To check equality

↳ a == b

Logical Operator

&& — AND

|| — OR

! ⟶ True to False

↳ False to true

eg:

```
int a=21;
int b=0;
cout << !a << endl; //0
cout << !b << endl; //1
```

☆ ! of anything other than zero is 0.

Lecture 3:

Conditional & Loops

If Statements

```
if( )
{
    ...
}
```

→ if the condition is true, then these code statements will be executed.

If-Else

```
if (condition)
{
    ...
}
else
{
    ...
}
```

→ if condition is true

→ if condition is false

☆ cin >> n

↳ for inputs

Question: (number_positive.cpp)
Get an inputs from the user and check if the number is positive or not?

☆ if cin >> a >> b;

↳ runs for 1 enter 2 enter ("\n")
↳ runs for 1 tab 2 ("\t")
↳ runs for 1 space 2 ("_")

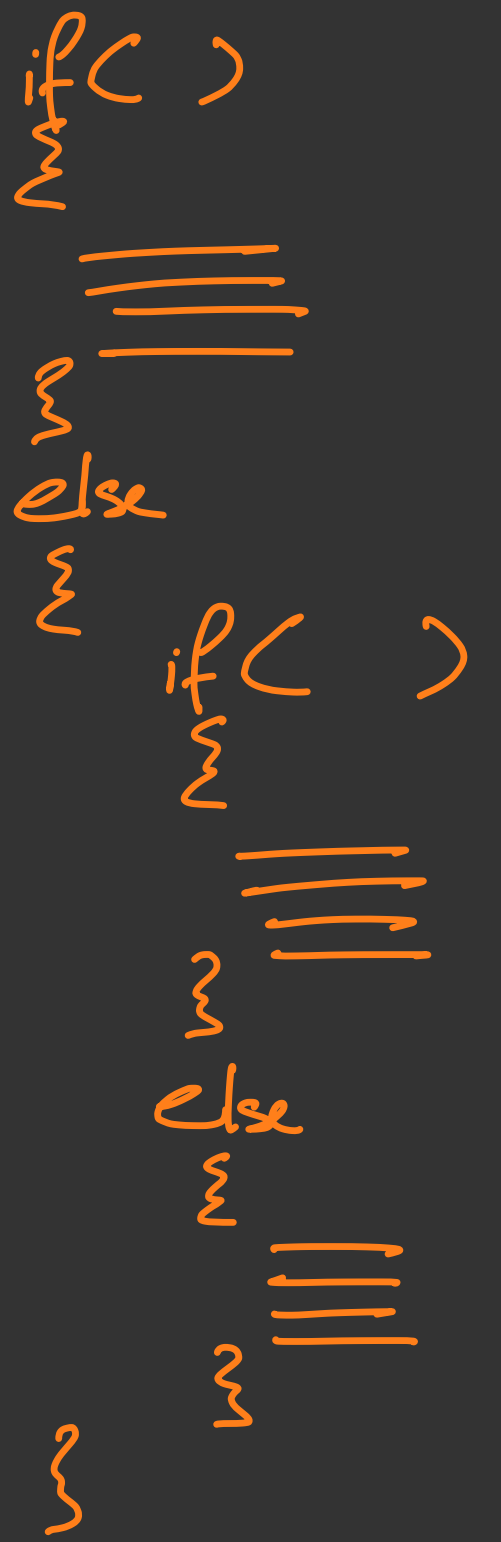
☆ cin.get()

eg: int a;
a = cin.get();

↳ It takes the character
So, for 1 ⟶ ASCII value of 1 = 49

Thus, 49 gets stored in a.

Nested if-else



Question: (which_character.cpp)

char ch;

if 'A' to 'Z' ⟶ upper case

if 'a' to 'z' ⟶ lower case

if '0' to '9' ⟶ numbers

While-loop

```
while (condition)
{
    ...
}
```

→ till the condition is true, these codes will be executed multiple times.

Question:

1. Print numbers from 1 to n.
   print_1_to_n.cpp
2. Sum of numbers from 1 to n.
   Sum_1_to_n.cpp

3. Whether a number is prime or not

prime_no_or_not.cpp

PATTERNS

1. 

→ pattern1.cpp

2. Same pattern as above, but user gives input of rows and columns.

↳ pattern1_5.cpp

3.
```
1 1 1  → 1st row
2 2 2  → 2nd column
3 3 3  → 3rd row
```

user → 3(n)

↳ pattern2.cpp

 → first while for rows

↳ second while insite first while for columns of each row.

Lecture 4:

⚠ ⟹ After seeing the pattern, first observe ki how rows and colums are affecting the pattern.

(4)

```
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
```

user → n

→ pattern3.cpp

(5)

```
1 2 3 4
1 2 3
1 2
1
```

user → 4(n)

→ patten3_5.cpp

(6)

```
3 2 1
3 2 1
3 2 1
```

user → 3(n)

→ pattern3_75.cpp

(7)

```
1 2 3
4 5 6
7 8 9
```

user → 3 (n)

→ pattern4.cpp

(8)

```
*
* *
* * *
* * * *
```

user → 4 (n)

→ pattern5.cpp

(9)

```
1
2 3
4 5 6
7 8 9 10
```

→ pattern6.cpp

(10)

```
1
2 3
3 4 5
4 5 6 7
```

→ pattern7.cpp

(11)

```
1
2 1
3 2 1
4 3 2 1
```

(12)

A
B B
C C C
D D D D

n → 4

→ pattern9.cpp

★ char ch = 'A' + 1;
↳ 'B'

29:06

(13)

A B C
D E F
G H I

→ practice > test_pattern4.cpp

(14)

A B C
B C D
C D E

(15)

A
B C
D E F
G H I J

16

D

C D

B C D

A B C D

D is not the end goal → starts from such a character that it reaches D.

→ practice>
test_pattern5.cpp

17

18

19

practice>
test_pattern6.cpp

(20)

```
1 1 1 1
 2 2 2
  3 3
   4
```

(21)

```
   1
  2 2
 3 3 3
4 4 4 4
```

(22)

```
1 2 3 4
 2 3 4
  3 4
   4
```

(23)

```
   1
  2 3
 4 5 6
7 8 9 10
```

(24)

```
_ _ _ 1
_ _ 2 1 2
_ 3 2 1 2 3
4 3 2 1 2 3 4
```

25

```
    1
  2 1
3 2 1
4 3 2 1
```

26 n=5

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	5	4	3	2	1
2	1	2	3	4	✗	✗	4	3	2	1
3	1	2	3	✗	✗	✗	✗	3	2	1
4	1	2	✗	✗	✗	✗	✗	✗	2	1
5	1	✗	✗	✗	✗	✗	✗	✗	✗	1

→ pattern_dabangg.cpp

Lecture 5:

Bitwise Operator 2 → they work at bit level

AND → &

OR → |

NOT → ~

XOR → ^

AND

&& → logical operator

& → bitwise operator

eg: a = 2, b = 3

a & b → 2 = 10

3 = 11

(1·1)(0·1) → 10 → 2

$z = a \mid b$

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

OR

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

eg: a = 3, b = 6

```
  11
 100
-----
 111  → 7
-----
```

NOT

x	y
1	0
0	1

eg: a = 2

→ int = 4 bytes = 32 bits

→ 00000000 ___ 0010

~a → 11111111 ____ 1101

→ negative

for printing ~a

→ 1's compliment

00000000 ____ 0010

→ 2's compliment

00000000 ____ 0010

+1

00000000 ____ 0011

→ 3

Since, the number was negative = **-3**

XOR

a	b	z
0	0	0
0	1	1
1	0	1
1	1	0

→ 1 for odd no. of 1's

→ 0 for even no. of 1's

eg: a=2 ⟶ 010

b=4 ⟶ 100

110 ⟶ a^b

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

8	1000
9	1001
10	1010 (A)
11	1011 (B)
12	1100 (C)
13	1101 (D)
14	1110 (E)
15	1111 (F)

eg: a = 5 → 101
b = 3 → 011

→ bitwise_op.cpp

⇒ a & b →

```
  101
  011
  ---
  001   → 1
```

⇒ a | b →

```
  101
  011
  ---
  111   → 7
```

⇒ ~a →

000___0101

↓

111___1010

000___0101

+1

```
000___0110  → -6
```

⇒ ~b →

0000___011

↓

1111___100

0000___011

+1

```
000___100  → -4
```

⟹ a^b ⟶ 101
011
110 ⟶ (6)

Left & Right Operator

⟹ Left Shift

eg: 5<<1 5x2 = 10

5 ⟶ 000_ ___ 101
↓ 1
000_ __1010 ⟶ (10)

eg: 3<<2 3x2x2 = 12

3 ⟶ 000_ ___ 011
↓ 2
000_ __ 1100 ⟶ (12)

✗ x2 ✓ ⟹ not everytime, but most of the time

✗ for big numbers, this might create negative numbers

eg: 0100_ ___ 0010
↓
10000_ ___ 0100
↳ negative number

⟹ Right Shift

eg: 5 >> 2 $\quad 5/2 = 2/2 = 1$

5 ⟶ 0000 ___ 0101

0000 ___ 0001 ⟵ 2

⟶ (1)

* for positive (+)ve ⟶ padding is always with zero.

⟶ ie if new bits need to be added, they will be 0 only.

for (−)ve

⟶ padding ⟶ compiler dependant

eg:

17 >> 1 ⟶ 8 } ÷2

17 >> 2 ⟶ 4

19 << 1 ⟶ 38 } x2

21 << 2 ⟶ 84

⇒ practice > test3.cp

int a = -5

2 → 5 = 101

000 -- 000101

→ 111 -- 111010

+1

111 -- 111011

int b = 6

2 → 000 -- 000110

1111 -- 111011

0000 -- 000110

a & b = 0000 -- 000010

→ (2)

a | b = 1111 -- 111111

→ while printing treated as negative number

→ 0000 -- 000000

+1

0000 -- 000001

add the negative sign → (-1)

$a \wedge b = 1111\ldots1101$

↳ while printing treated as negative number

→ $0000\ldots0010$

$+1$

$0000\ldots0011$

↳ -3

$a = 1111\ldots1101$

$\sim a = 0000\ldots0010$

↳ 4

$b = 0000\ldots0110$

$\sim b = 1111\ldots1001$

↳ $0000\ldots0110$

$+1$

$0000\ldots0111$

↳ -7

a = 111--111011

a<<1 = 111___110110

↳ 000__001001

+1

000__001010

↳ -10

a<<2 = 111___101100

↳ 000__010011

+1

000___010100

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

$= 4 + 16 = 20$

⟶ -20

b>>1 = 000____000011

↳ 3

Increments and Decrements operator

Increments

- i++; → post-increment
- ++i; → pre-increments

Decrements

- i--; → post-decrements
- --i; → pre-decrements

i = i + 1 ⟶ i += 1;

i = i - 1 ⟶ i -= 1;

eg: i = 8;

a = (i++) + (--i) + i + (--i);

cout << i << endl;

cout << a << endl;

8 9 8 7

8 + 8 + 8 + 7 = 31

eg: if(a) → true : a ≠ 0
→ false : a = 0

3 ≡

```
#include <iostream>
using namespace std;
int main()
{
    int a = 1;
    int b = 2;

    if(a-- > 0 && ++b > 2 ){
    cout << "Stage1 - Inside If ";
    } else{
    cout << "Stage2 - Inside else ";
    }
    cout << a << " " << b << endl;
}
```

a = 1 ~~0~~
b = 2 ~~3~~

→ Stage1 - Inside If
0 3

```
#include <iostream>
using namespace std;
int main()
{
        int number = 3; 4
        cout << (25 * (++number) );
}
```

→ 100

```
#include <iostream>
using namespace std;
int main()
{
    int a = 1;
    int b = a++;
    int c = ++a;
    cout << b ;
    cout << c;
}
```

a = 1 ~~2~~ 3
b = 1
c = 3

→ 1
3

For Loop → has iterations

Syntax:

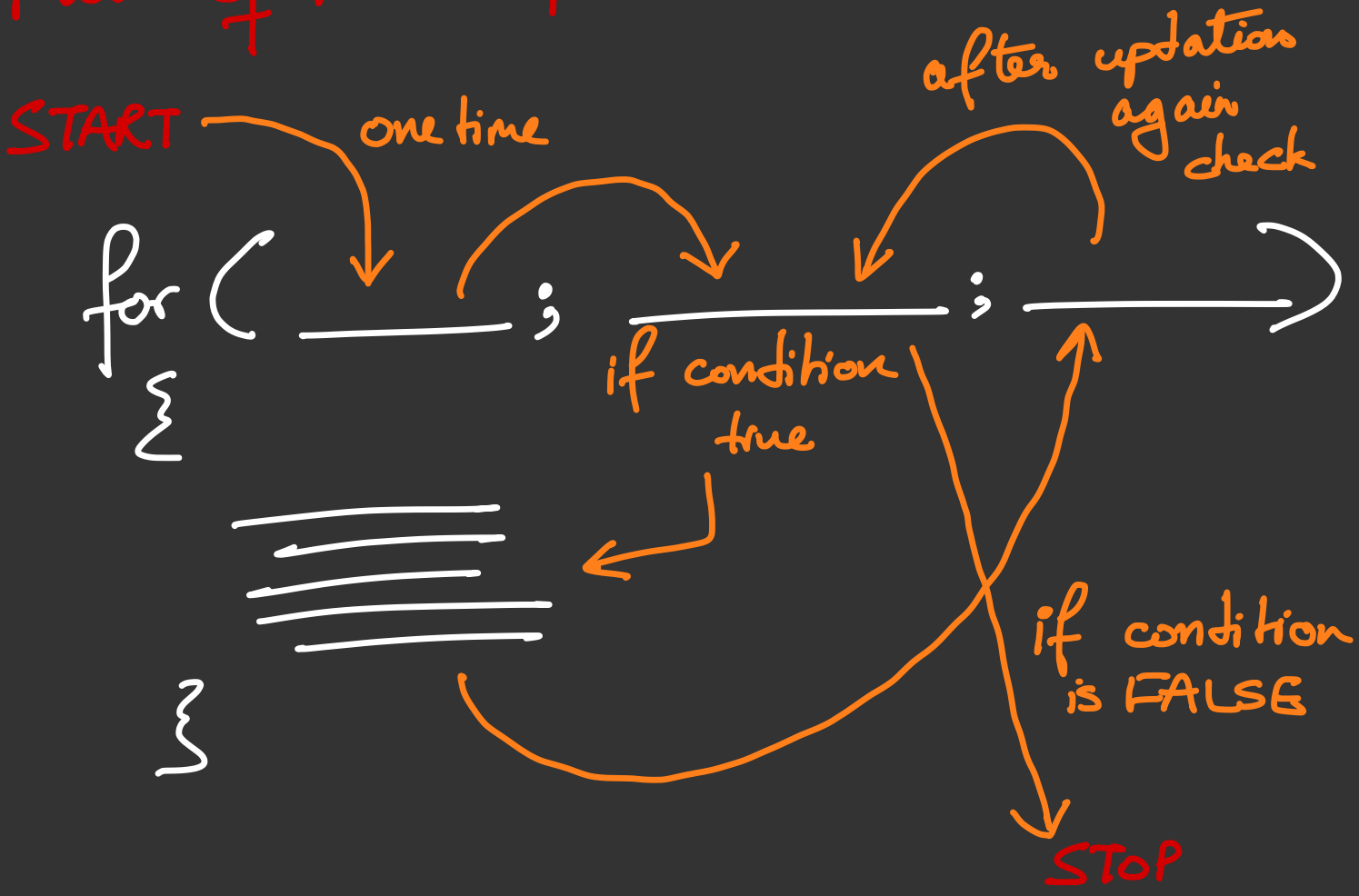
```
for(initiation; condition; incre/decrements)
{
    ||||||
}
```

break; → gets you out of the current loop

Question:
Print 1 to n using non-parameterized for loop.

→ for_loop_no_parameter.cpp

Flow of For Loop



Question:

Fibonnaci Series

↳ 0, 1, 1, 2, 3, 5, 8, 13

⟶ fibonnaci_series.cpp

Prime Numer or not

↳ prime_or_no.cpp

Continue; → to skip all the code below this statement for current iteration.

Output Questions:

```
#include<iostream>
using namespace std;
int main() {

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

Two times increments

i= 0 1 2 3 4 5 6

0 2 4

```cpp
#include<iostream>
using namespace std;
int main() {

    for(int i = 0; i<=15; i += 2 ) {
        cout<< i << " ";

        if( i&1 ){
            continue;
        }

        i++;
    }
}
```

[odd number & 1] = 1

i = 0 ~~1 2 3 4 5 6 7 8 9 10 11 12 13 14 15~~ 17

0 & 1 → 0

3 & 1 → 0011
0001
0001 → 1

5 & 1 → 0101
0001
0001 → 1

7 & 1 → 1

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Outputs:

0 3 5 7 9 11 13 15

```cpp
#include<iostream>
using namespace std;
int main() {

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

```
0 0
0 1
0 2
0 3
0 4
0 5
1 1
1 2
1 3
1 4
1 5
2 2
2 3
2 4
2 5
3 3
3 4
3 5
```

```
4 4
4 5
5 5
```

2 → i cannot be 5

```cpp
#include<iostream>
using namespace std;
int main() {

    for(int i=0;i<5;i++) {
        for(int j=i;j<=5;j++) {
            if(i+j == 10) {
                break;
            }
            cout<<i << " " << j <<endl;
        }
    }
}
```

0 0
0 1
0 2
0 3
0 4
0 5
1 1
1 2
1 3
1 4
1 5
2 2
2 3
2 4
2 5
3 3
3 4
3 5

4 4
4 5

Scope of Variables

```
int main()
{
    int a=3

    if(true)
    {
        cout<<a;
        ==========
        int b=3;
    }

    int a=1;

    cout<<b;
}
```

→ this will work

→ will give error "redefination of a"

→ will show error "use of undeclared identifier 'b'"

* If a variable is defined inside a block, then its presence is limited to inside that block only. But cannot define again in same block.

```
int a = 2;
int b;
b = 3;
```

2 → Defining

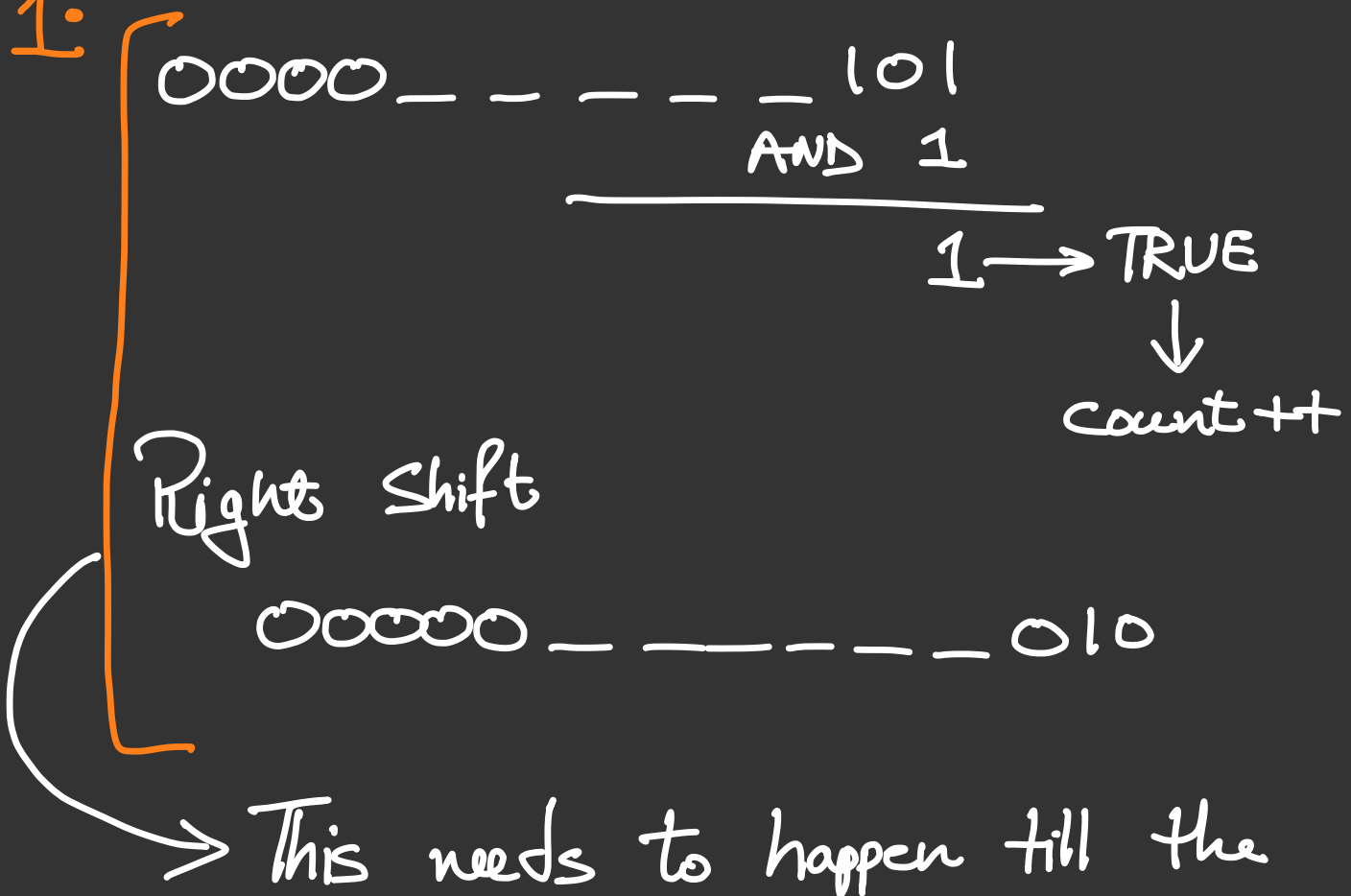
2 → Assigning

Operator Precedence

→ use brackets

Question: Number of 1's (LEET CODE)

1:



→ This needs to happen till the number becomes zero
ie. no 1's left

2.

2	5	1
2	2	0
2	1	1
	0	

2 → just keep dividing by 2 and keep changing n and counting 1's.

Thus,

Right Shift is equivalent to division by 2.

$$n >> 1 == n/2$$

eg:

7 = 0111

7/2 = 3 → 0111 ↓ RS → 0011

3 = 0011

3/2 = 1 → 0011 RS → 0001

1 = 0001

Lecture 6

Binary and Decimal Number System

Decimal to Binary

Method 1 ⟶ decimal_to_binary_method1.cpp

n = 10

2	10	0
2	5	1
2	2	0
2	1	1
	0	

$10_{(10)} \longrightarrow 1010_{(2)}$

Steps

- ⟶ divide by 2
- ⟶ store remainder
- ⟶ repeat step 2 until n != 0
- ⟶ reverse the remainders

⭐⭐⭐ If $n \& 1 = 1$ → n is odd

else if $n \& 1 = 0$ → n is even

Thus, for last bit, just AND 1,
if result is 1 → then bit was 1
else bit was 0.

Method 2 → decimal_to_binary_method2.cpp

binary = xxxxx = n

→ for last digit (n&1)

→ n = n>>1; = Right Shift

How to store these bits?

101 → 1
01
101

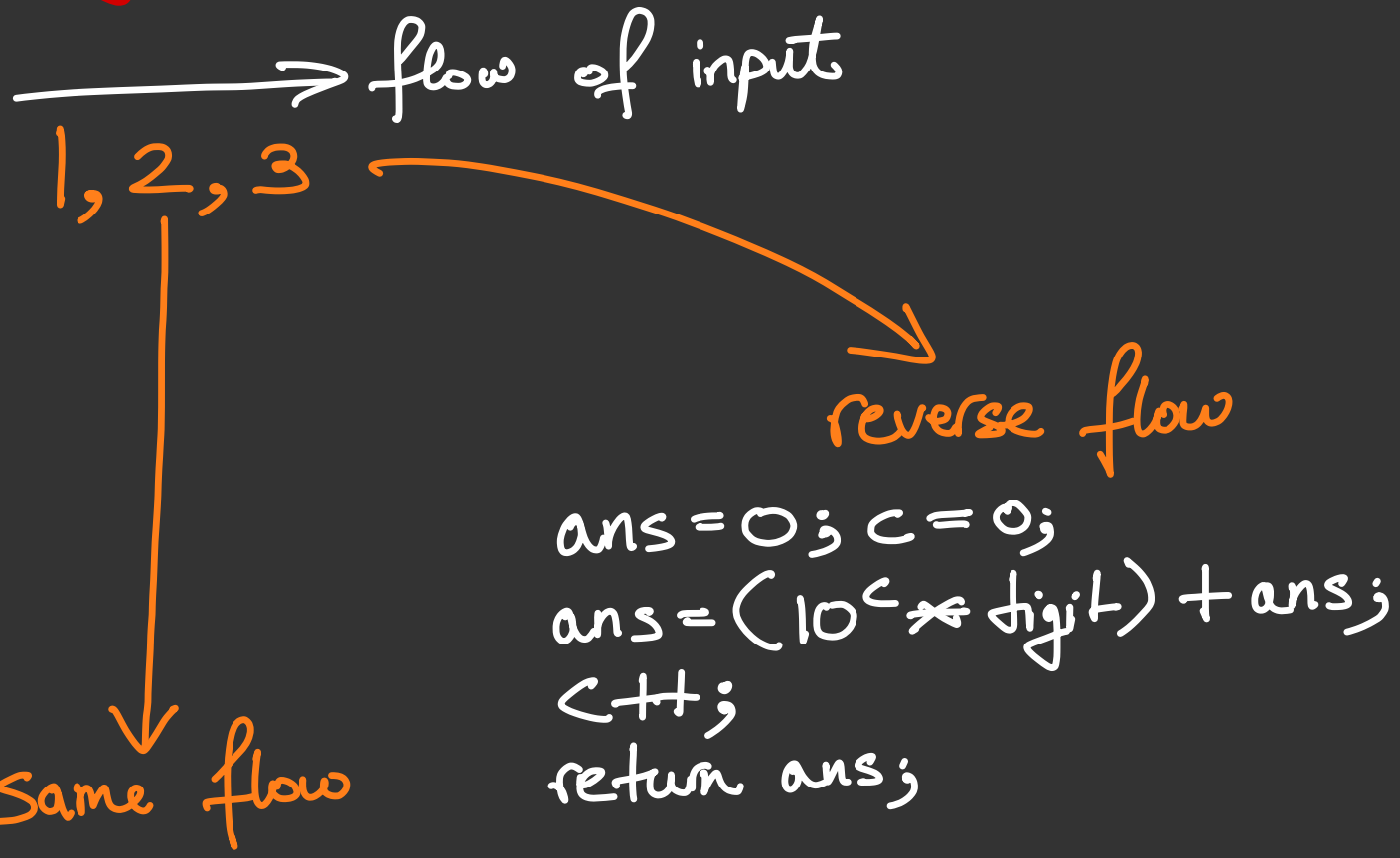
⇒ res = 0; count = 0;

res = (10^count x digit) + res;

count ++;

pow() should be used as double variable.
If needed typecast it to int.

Digits



Negative number to binary form

↳ neg_decimal_to_binary.cpp

n = -6

1. Ignore -ve sign
2. Take 2's compliment
   - ⟶ 1's compliment
   - ⟶ +1

Negative Binary to Decimal ☆

1. Check if first bit = 1, means number is binary
2. Take 2's compliments of the number

Binary to Decimal

↳ binary_to_decimal.cpp

eg: 101

1 0 1

$2^2 + 2^1 + 2^0$

$\Rightarrow (1 \times 2^2) + (1 \times 2^0)$

$\Rightarrow 4 + 1 \Rightarrow 5_{(10)}$

☆ n = 101011

Last digit can be taken out by two ways ⟶

① int d = n % 10;
n = n / 10;

② int d = n & 1;
n = n >> 1;

✗ if the inputs (binary) taken as int, cannot use this.

This, works will last bit of the decimal number.

Lecture 7

INT_MAX
INT_MIN
} → represents the extremes of integer datas type

⚠ Compliment Base 10 Integer

Given = 5

↳ 0000 --- 0101

Want = 0000 --- 0010

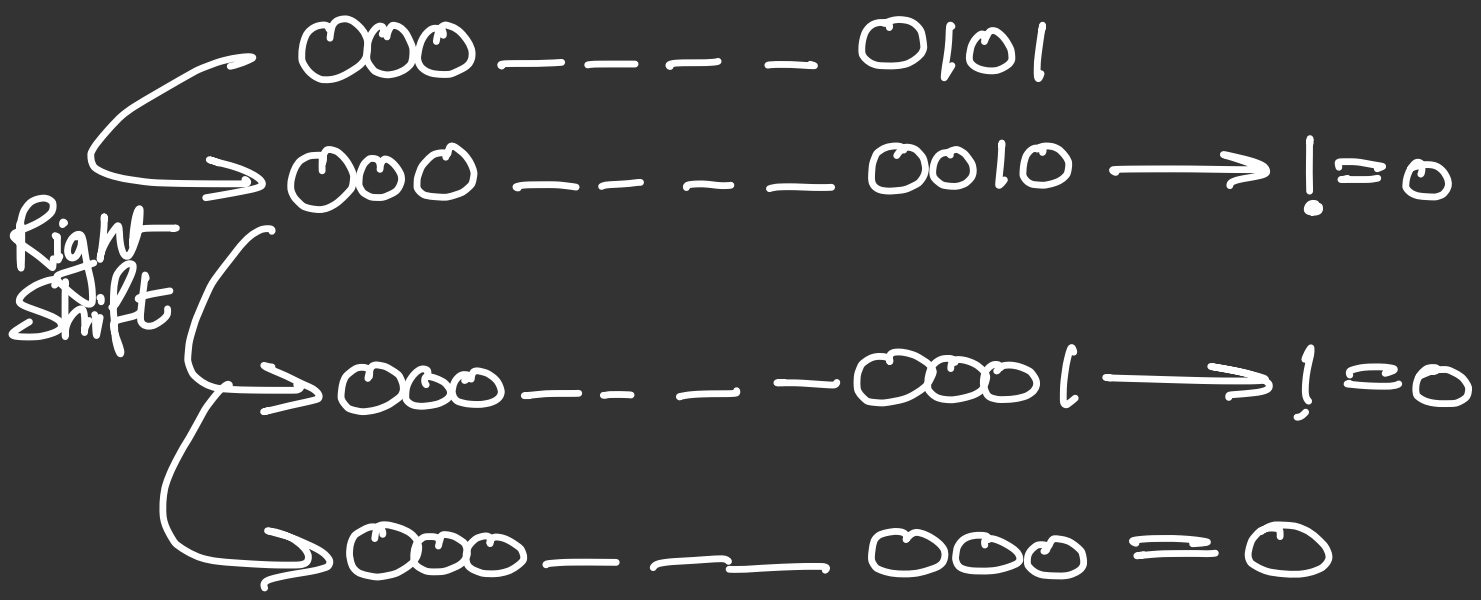
↳ 2

Mask 1 = ~n = 1111 --- 1010

Mask 2 = & 0000 --- 0111

0000 --- 0010

↳ 2




```
int n1 = ~n;
int mask=0;
while(n!=0)
{
    n=n>>1;
    mask = mask<<1;
    mask = mask | 1;
}
int c = n1 & mask;
cout << " Compliment is: " << c << endl;
```

Right Shift

Left Shift

At the end we will get 1 at the last

LeetCode Question - 231

Power of 2

→ power_of_2.cpp

Method 1:

n → number

```
d=0; c=0;
while (d<=n)
{
    d= pow(2,c);
    if(d==n)
        cout<<"Yes";
        return 0;

    c++;
}
cout<<"No"
```

Method 2:

		SetBits
0	0000	
$2^0 = 1$	0001	1
$2^1 = 2$	0010	1
3	0011	
$2^2 = 4$	0100	1
5	0101	
6	0110	
7	0111	
$2^3 = 8$	1000	1
⋮		
$2^4 = 16$	10000	1

Set Bits = no. of 1 bits in the binary of the integer

★ for all 2^x = no. of set bits = 1

no. of 1's = 1

Code:

```
n ⟶ number
count = 0;
while(n != 0)
{
    if(n & 1)
        count++;
    n = n >> 1;
}
```

Checking for last bit if it is 1.

Right Shift

Lecture 8

Switch – Case

→ int (or) char

```
switch (expression)
{
    case  :  ========
             break;
    case  :  ========
             break;
    default :  ======
}
```

- ☆ Default not mandatory
- ☆ If break not written after a case, it check for other cases as well

Thus, if no breaks throughouts, defaults will always be printed

Nested switch also allowed

* If else iff's can be converted to switch case and vice-versa.

* exit(0);

↳ terminates the program

break;

↳ helps you get outs of switch case or any loop.

* continue; → cannot be used with switch case.

Calculator program

a, b

operator = *, /, +, − } perform using switch case.

→ mini_calculator.cpp

Homework Question

Total Amount = ₹ x → ₹100 Note, ₹50 note, ₹20 note, ₹1 note

→ using switch case.

→ money-split.cpp

★ Use switch(1)

{

case 1:

case 2:

case 3:

}

→ all the cases will be executed

Functions

→ to stop bulkyness of the code (reduce LOC)

→ reuseability of code

Syntax:

```
return_type FunctionName (argument)
{
    ========

    return(—);
}
```

return_type → void

→ any other data type

Power a^b

↳ power_a_b.cpp

is Even?

↳ even_odd.cpp

Any number

$n \& 1 = 1$

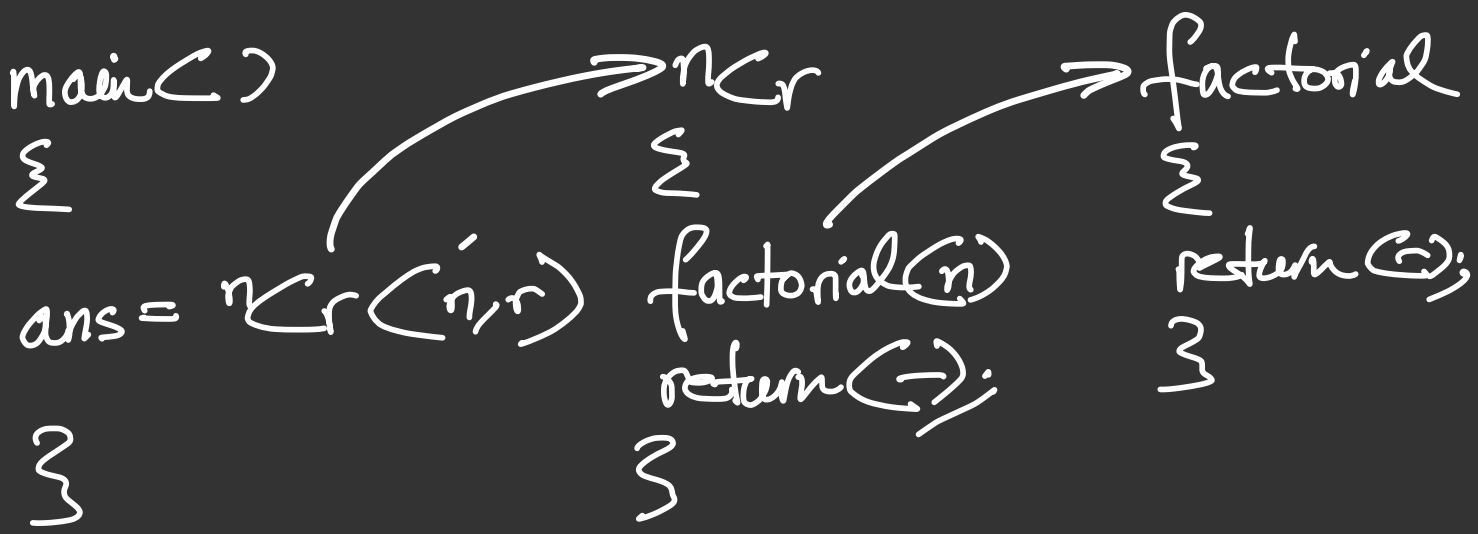
then, n = odd number

$nCr = \frac{n!}{r! * (n-r)!}$

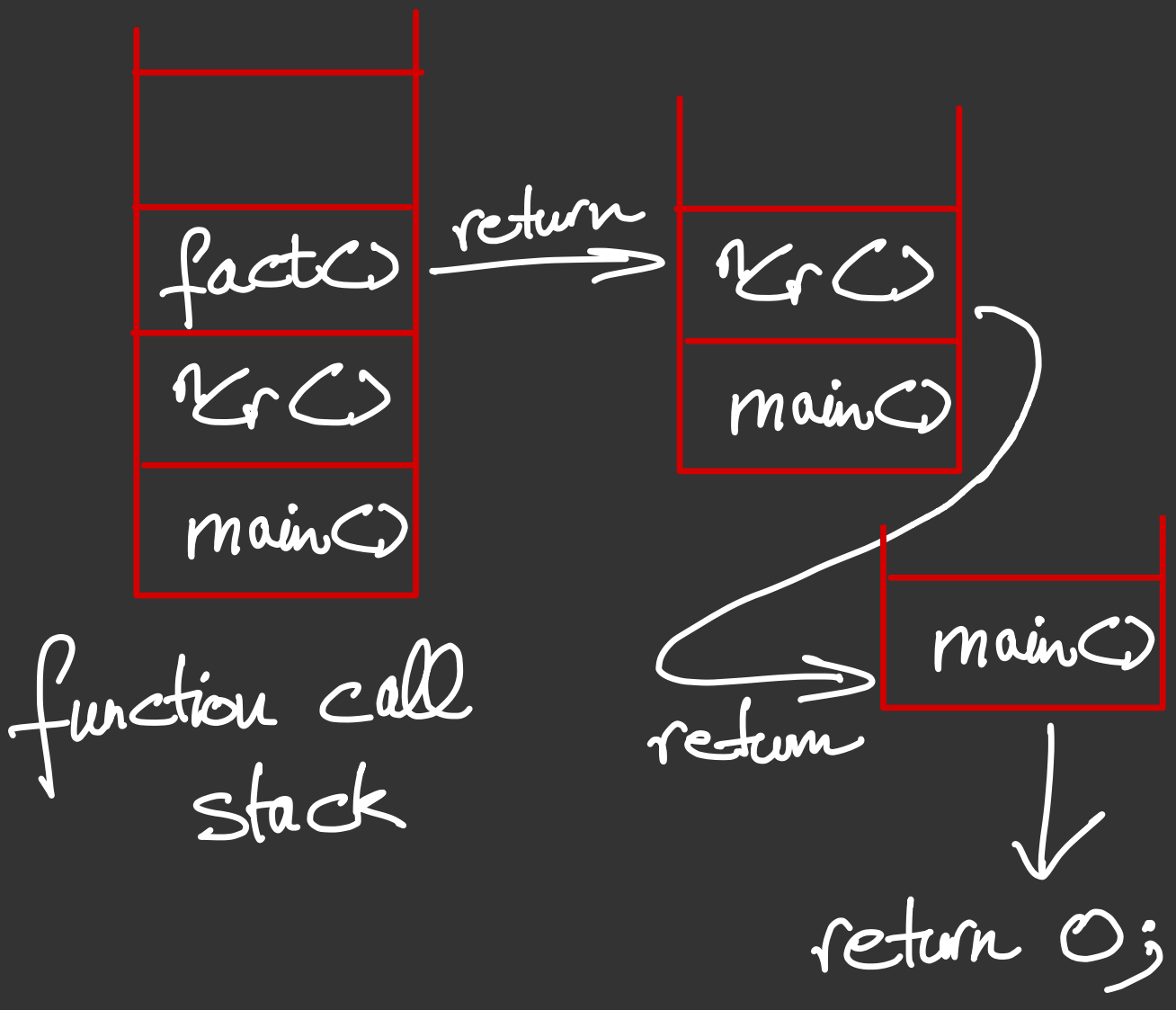
↳ nCr.cpp

☆ In case of void

↳ return; can be used without any value



Function Call Stack



function call stack

Pass by value

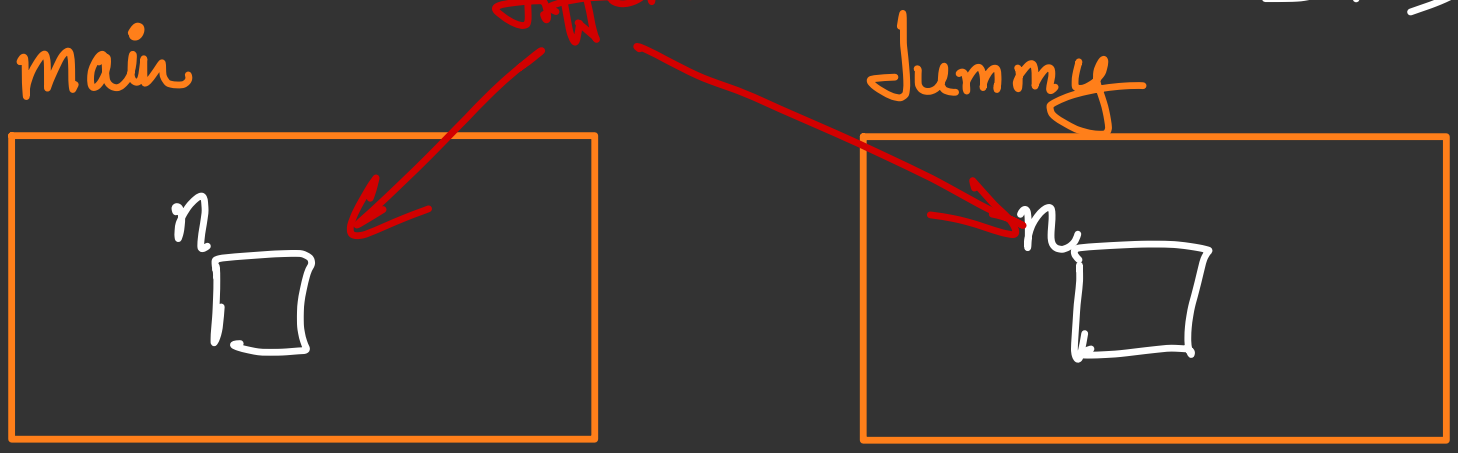
Example:

```
main()
{
   n=15;
   dummy(n);
   cout << n;
}
```

```
dummy(int n)
{
   n++;
   cout << n;
}
```

Outputs → 16
15

Because (its just that there name is same)



* Variable 'n' in dummy is diff. and 'n' in main() is different.

Its just while calling dummy the value of 'n' in main is assigned to variable 'n' in dummy.

Outputs Questions

```
1  void update(int a) {
2      a = a / 2;
3  }
4  int main() {
5      int a = 10;
6      update(a);
7      cout << a << endl;
8  }
9
```

10

```
1   int update(int a){
2       a -= 5;
3       return a;
4   }
5
6   int main() {
7       int a = 15;
8       update(a);
9       cout << a << endl;
10  }
11
```

error

* functions returns a value and not assigned to a variable.

```
1   int update(int a){
2       int ans = a * a;
3       return ans;
4   }
5
6   int main() {
7       int a = 14;
8       a = update(a);
9       cout << a << endl;
10  }
11
```

196

Homework Question

1. AP = 3*n + 7
2. a & b ⟶ total no. of set bits

a = 2 ⟶ 10
b = 3 ⟶ 11
no. of 1's ⟶ 3

3. Fibonnaci

nth elements

0, 1, 1, 2, 3, 5, 8 - -

⟶ homework.cpp