05/04/2025

① Lec 3 - Basics of Programming (III)

→ Operators

- Symbols which are Used to perform operation.

Types:

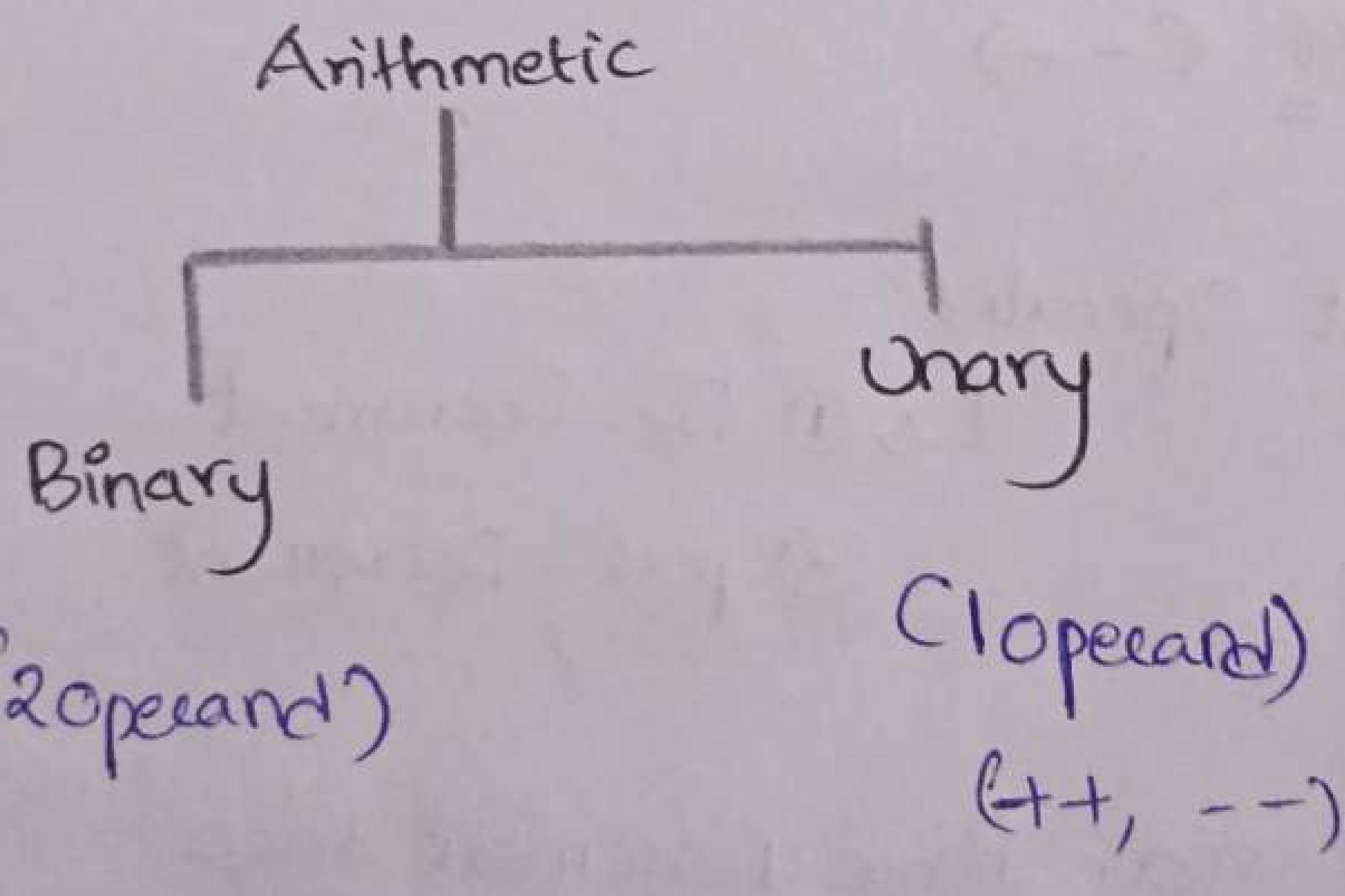
Arithmetic Operator: +, -, *, /, %, ++, --

Relational operator: ==, !=, >, <, >=, <=

Assignment Operator: =

Logical Operator: &&, ||, !

Bit-wise operator: ↷ Learn this from (LearnC++) - &, |, ∧, V, >>, <<



(++) - Increment Operator

↳ ① Pre-Increment

② Post-Increment

① Pre-Increment   a. Pehle Increment Karo

(++a)   b. Fir Use Karo

Ex int a=5;

cout << ++a;

↳ ① $a = \cancel{5}6$ $\underset{a}{\boxed{\cancel{5}}}^{6}$

② print 6

② Post-Increment   a. Pehle Use Karo

b. Fir Increment Karo

(a++)

Ex: int a=6;

cout << a++;

→ ① print 6

② $a = \cancel{6}7$ $\underset{a}{\boxed{\cancel{6}}}^{7}$

→ **Decrement Operator:** (--)

(--) - Decrement Opeeator.

↳ ① Pre-Decrement

② post-Decrement.

① Pre-Decrement   a. Pehle Decrement Karo

(--a)   b. Fir Use Karo

Ex: int a=7;

cout << --a; ↗ ① $a = \cancel{7}6$ $\underset{a}{\boxed{\cancel{7}}}^{6}$

② print 6

② Post-Decrement a. Pehle print karo

(a--) b. Fir Use karo.

Ex: int a=8;

cout<<a--; → ① print 8

② $a = \cancel{8}$ ⑦

→ Homework: 20questions on Unary operator solve(using ai generate questions)

Q) int a=5; → $a = \boxed{5}$

cout<< (++a); → ① $a = \boxed{\cancel{5}}$ 6 ② print 6

cout<< a; → ③ 6

cout<< (a++); → ④ print 6 , $a = \boxed{\cancel{6}}$ 7

cout<< a; → ⑤ print 7

cout<< (--a); → ⑥ $a = \boxed{\cancel{7}}$ 6 print 6

cout<< a; → ⑦ 6

cout<< (a--); → ⑧ print 6 , $a = \boxed{\cancel{6}}$ 5

cout<< a; → ⑨ 5

<u>output:</u>

6 6 6 7 6 6 6 5

Q) Homework:

Int val= (++a) (a++) + (--a)* (a--) ;

Cout<< val;

Q) Is there any operator like +++ a+++b

Ans ① Not poss

② Caunter Case

It runs on Compiler like

$\underbrace{(a++)}_{\text{Unary}} \underbrace{+}_{\text{binary}} b$

Q) why we are using brackets for Increment operator in operation

val= (a++) * (++a)

why cant a++ * •++a?

Ans:• operator precedence & Associativety makes operation Easily

• Acc to BODMAS rule the operation makes Easily acc to rules.

Q) a=5, b=7 (--a)*(++b) *5

here print means use

a=~~5~~4 b=~~7~~8

print(4) * print(8) *5

=> 4*8*5 = 180

Q) a=5, b=6 (--a)*(b++)

print → use

a=~~5~~4 b=6

print(4) * 6 = 24

Q) a=5 (--a)*(a--)

Ambiguous behaviour.

→ Conditionals

- Basic if block
- if else block
- if -else if - else block
- Nested if else
- switch case
- Ternary operator.

→ Basic if block

Syntax:
```
if (condition)
{
   =
}
```

Ex①:
```
if (score >1000)
{
  cout<< "Yelo macbook";
}
```

Ex②
```
if (age>=18)
{
  cout<< "Yelo license";
}
```

Ex③
```
if (age>=18 &&
          score>1000)
{
  cout<< "Bike";
}
```

Q) Difference b/w a&b a&&b

a&b → Bitwise AND operator

a&&b → Logical AND

① 2&3

```
  2-0--- 010
  3-0----011
  ----------
  0---010 = 2
```

② 2&&3 =>

$$\underset{1}{2} \&\& \underset{1}{3}$$

1&&1 = ① → True.

①

② If-else block

```
Syntax: if(condition)
        {
            ≡
        }
        else
        {

        }
```

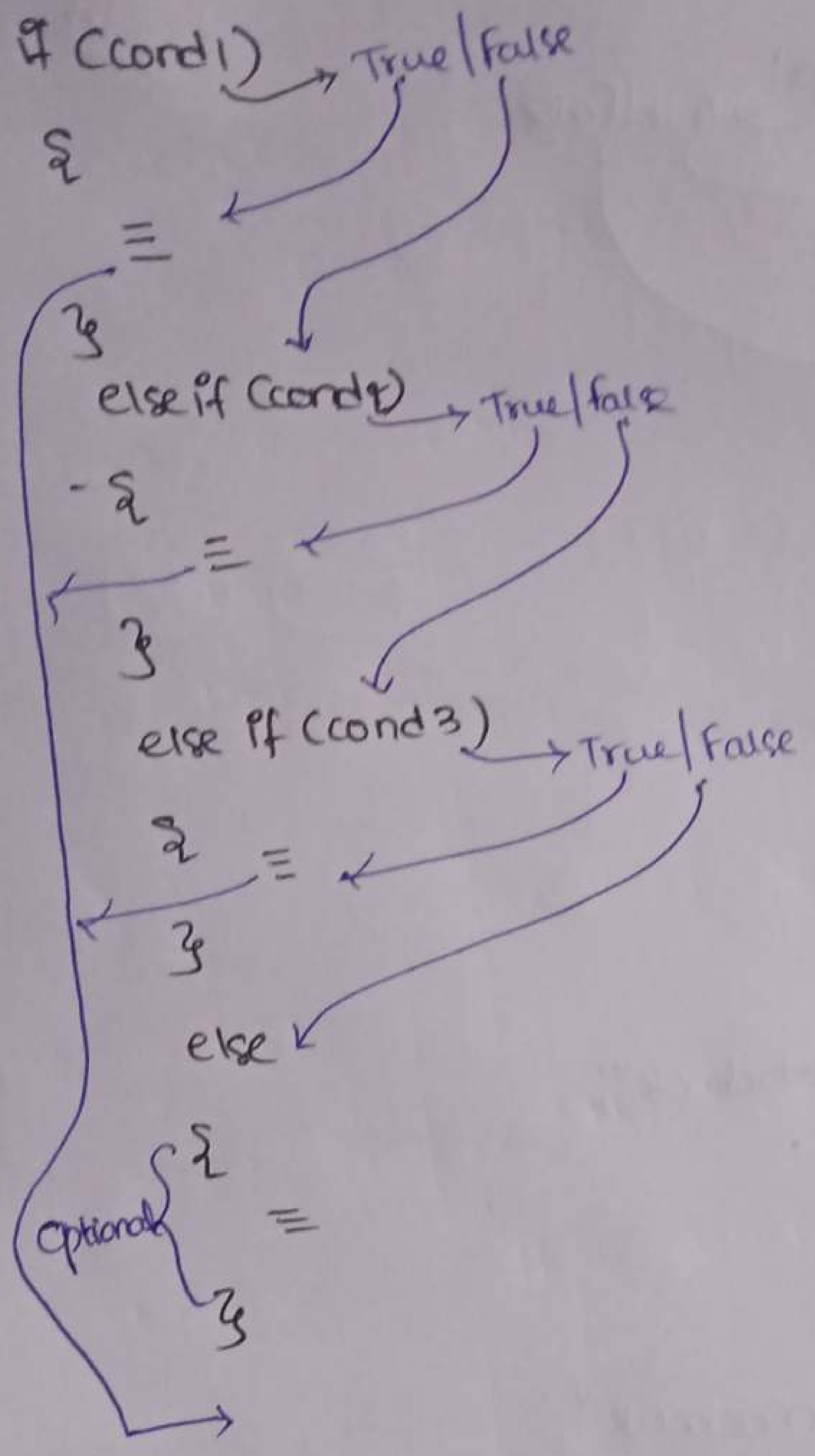
Ex①

```
if(score>1000)
{
   Cout<< "Yelo Macbook";
}
else
{
   Cout<< "No macbook";
}
```

③ if-else if ladder

```
Syntax: if (cond 1)
        {
            ≡
        }
```

Syntax:



```
if (cond1) → True|False
{
   ≡
}
else if (cond2) → True|false
{
   ≡
}
else if (cond3) → True|False
{
   ≡
}
else
{
   ≡
}
```

Ex:

```
if (peecentage >90)
{
   cout<<  "A";
}
else if (peecentage >80 && peecentage <90)
{
      cout << "B";
}
```

char + in →

| Explicit Type casting

```
else if (percentage >70 && percentage <80)
 {
     cout << "C";

  }
  else
  {
       cout << "fail";

   }
```

→ Short Circuit :

```
              True       True        if false go out of loop
 if ( C1 && C2 && C3 && C4 )
 {
      cout << "love";

 }
```

Ex: Rest- 1chef Vacancy

Chef1 - ✗

Chef2 → ✗

chef3 → ✓

Chef4 } no need

chef5 } to check.

→ Nested if-else.

```
Syntax: if ( )
         {
           if ( )
           {
              if ( )
              {
              }
           }
         }
```

Nested if

```
if ( )
{
    if ( )
    {
    }
    else
    {
    }
```

Nested if else

→ **Ternary Operator:**

- Easy Understandable if we have conditionals (if, ..) knowledge
- It works like if-else conditions

Syntax:

```
                 false
Condition ? logic : logic
        true
```

Ex① int age = 15;   int value = (age>18)? 50 : 100

15>18 ✗
false

return 100 for value.

Ex② int age = 21;

(age>18)? cout<< "Hello" : cout<< "no hello";

21>18 ✓   Hello will be printed.

→ **Switch case:**

Syntax: switch (expression)

```
{
  case 0:
  case 1:
  case 2:
(optional) default:
}
```

Ex:

```
Switch(index) {

   Case 1:
        Cout<< "Monday";

    Cout 2:
         Cout << "Tuesday";

     Cout 3:
           Cout << "wednesday";

      default:
             Cout<< "sunday";

    }
```

→ Loops:

For loop

while loop } discussed in learn C++

do while loop }

for each loop

Syntax:

```
for (initialization; condition, opdation)
{
     =

}
```

Q)
```
for (int i=1; i<=5; i=i+1)
{
  cout<<i;
}
```

→ Can also be written as (i+=1 or i++ or ++i)

DryRun:

```
for (int i=1; i<=5; i=i+1)
{
  cout <<i;
}
```

i=1 → 1<=5 ↙ 1✓ → i=i+1 = 1+1 = ②

i=2 → 2<=5 ↙ 2✓ → i=i+1 = 2+1 = ③

i=3 → 3<=5 ↙ 3✓ → i=i+1 = 3+1 = ④

i=4 → 4<=5 ↙ 4✓ → i=i+1 = 4+1 = ⑤

i=5 → 5<=5 ↙ 5✓ → i=5+1 ↓ ⑥

i=6 → 6<=5✗

Output:

1 2 3 4 5

Q) int n=4

```
for (int i=0; i<=4;i=i+1)
{
    cout << "Love";
}
```

```
int n = 4 ~~0~~;

for (int i=0; i<=4;i++)
{
  cout << "Love";
}
```

i=0 → 0<=4
↓
Love → i=i+1
=0+1
=1

i=1

i=1 → 1<=4
↓
Love → i=i+1
=1+1
=2

i=2 → 2<=4
↓
Love → i=i+1
=2+1 = $\boxed{3}$

i=3 → 3<=4
↓
Love → i=i+1
=3+1
= $\boxed{4}$

i=4 → 4<=4
↓
Love → i=i+1
=4+1
= $\boxed{5}$

i=5 → 5<=4 ✗

n=3

Q) for (int i=n; i>=0;i=i-1)

```
{
  cout << "i";
}
```

char + in ... type Casting

n=8

Q)
```
for(int i=0; i<=8; i=i+2)
{
  cout<<i<<" ";
}
```

<u>Dry Run:</u>

i=0 → 0<=8

↓

0<=8 ✓ → i=0+2

=2

↓

print 0

i=2 → 2<=8

↓

2<=8 ✓ → i=2+2

=4

↓

print 2

i=4 → 4<=8 ✓

↓

4 → i=i+2

=4+2

=6

i=6 → 6<=8

↓

print 6 → i=i+2

= 6+2

=8

i=8 → 8<=8

↓

print 8 → i=i+2

=8+2

=10

10<=8

✗

Q)
```
int n=10
for (int i=1; i<=n; i++)
{
  cout << (2*i) <<endl;
}
```

Dry Run:

```
i=1 → 1<= 10            | i=2 → 2<=10              ....   i= 10 → 10<=10
  ↙                     |   ↙                               ↙
2*i ✓ → i=i+1           | 2*2 → i= i+1                   2*10 → i= i+1
↓ 1      =1+1           |  ↓      =2+1                    ↓      =10+1
②        =2             |  ④      = 3                     20     =11
```

O/p: 2 4 6 8 10 12 14 16,18 20