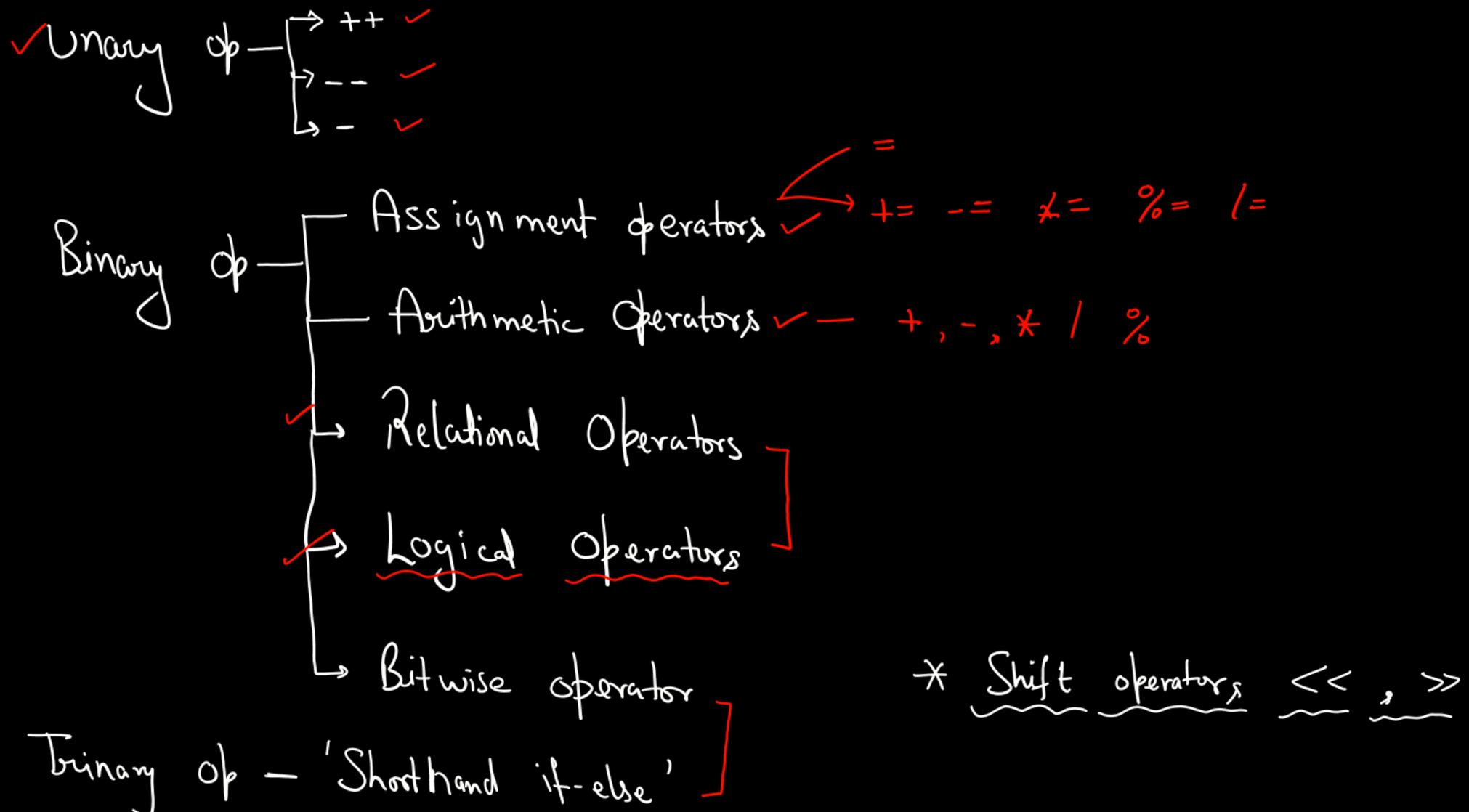
# Lecture - 09

# Programming in C

## [Operators - 03]


Unary op
++
--
-
Binary op
Assignment operators
=
+= -= *= %= /=
Arithmetic Operators
+, -, * / %
Relational Operators
Logical Operators
Bitwise operator
Trinary op – 'Shorthand if-else'
* Shift operators << , >>

## Relational Operator: → Compare two operands → Return Boolean result

↳ Comparision operator

→ It check for the relationship between two values

↳ Equality & Inequalities

**Equality**

- equal to (=)
- Not equal to (≠)

**Inequalities**

- Strict
  - <
  - >
- Slack
  - <=
  - >=

Relational operator → two Results ← True (1), False (0)

↳ True / False

| Relational op | Result |
|---|---|
| 2 = 3 | No (False) |
| 2 < 3 | Yes (True) |
| 3 ≠ 10 | Yes (True) |
| 3 > 5 | No (False) |
| 5 < 10 | Yes (True) |
| 3 ≤ 3 | Yes (True) |
| 3 ≥ 3 | Yes (True) |

a) is equals to (==)

↳ Return True (1) if both quantities are equal.

Ex: 3 == 3 ⇓ True (1)

```
int x = 17;
int y = 15;
printf("%d", x == y);
```
⇒ 0

("%d" ⇓ int; x == y ⇓ False = 0)

```
int x = 10;
printf("%d", x == 10);
```
⇓ True (1)

b) is not equals to operator (!=) → ! ← not

↳ ≠ ⇝ !=

Return True is both values are unequal

Ex: 3 != 5 ⇓ True (1)

```
int y = 45;
printf("%d", y != 45)
```
⇓ False (0)

```
int m = 0;
int n = 1;
printf("%d", m != n)
```
⇓ True (1)

relationalop.c

relationalop.c > main()

```c
# include <stdio.h>

int main(){
    printf("%d \n", 10 == 10); //True
    printf("%d\n", 10 != 10); // False
    printf("%d\n", 10 == 10.0);  // True
    printf("%d\n", 10 == 10.1); // false
    printf("%d \n", 'A' == 65); //True
    int m = 24;
    float n = 24.0f;
    printf("%d", m == n); //True

    return 0;
}
```

```
1
0
1
0
1
1
PS>
```

c) is less than (<)

↳ Return True (1) if left operand is less than Right operand.

Eg `printf("%d", 10 < 100);`

`%d` ⇓ 1

`10 < 100` → True

```
int x = 100;
int y = 100.00;
printf("%d", x < y);
```

`%d` ⇓ 0

`x < y` → False

d) is greater than (>):

↳ Return True if left operand is greater than Right operand.

↳ 1

Eg `printf("%d", 10 > 100);`

`%d` ⇓ 0

`10 > 100` → False

`printf("%d", 10 > 5.012)`

`%d` ⇓ 1

`10 > 5.012` → True

relationalop.c

relationalop.c > main()

```c
# include <stdio.h>

int main(){
    printf("%d \n", 10<10); // false
    printf("%d \n", 10>10); // false
    printf("%d \n", 19.524 > 19.52400);
    printf("%d \n", 10<20); //True
    printf("%d \n", 20>10); //True

```

```
0
0
0
1
1
PS>
```

e) is lessthan or equals to ( ≤ → <= )

↳ Retun true (1) if left operand is less than or equals to the Right operand

printf ('%d', 10 <= 10); → 1 (True)

printf ("%d", 10 <= 9); → 0 (False)

f) is greater than or equals to ( ≥ → >= )

↳ Return True if left operand is greater than or equals to Right operand

↳ 1

Eg printf("%d", 10 >= 10);  printf("%d", 20 >= 10)

1  True  1  True

relationalop.c

relationalop.c > main()

```c
# include <stdio.h>

int main(){
    printf("%d \n", 10<=10); //True
    printf("%d \n", 10 >= 10); //True
    printf("%d \n", 15>=10); //True
    printf("%d \n", 20>=30); //False
```

```
1
1
1
0
PS>
```

# Logical Operator :

↳ works with Binary input & generate Binary output

⇒ It provides the logic in the program for decision making.

& ← address of

&& ← logical AND

AND ⇒ The output is high (1) when all inputs are high.

↳ true

↳ Binary < True / false

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

&& ← AND

```
printf("%d", (10 < 20) AND (9 != 10));
```

&&

%d ⇓ 1

(10 < 20) → True 1

AND && → 1 AND 1 → 1

(9 != 10) → True 1

printf ( "%d", (10==10) && (2<3) && (3!=3));

0 ⇓

(10==10) → 1 && (2<3) → 1 && (3!=3) → 0 ← low

relationalop.c | logicalop.c ×

logicalop.c > main()

```c
#include<stdio.h>

int main(){
    printf("%d \n", (10 == 10) && (5<6)); //True (1)
    printf("%d \n", 10!=10 && 15==15); //False
    return 0;
}
```

```
1
0
PS>
```

## Logical OR operator (||) : Return True if any one input is high (1) otherwise false

True → 1

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

a || b ← a OR b

OR – या / या

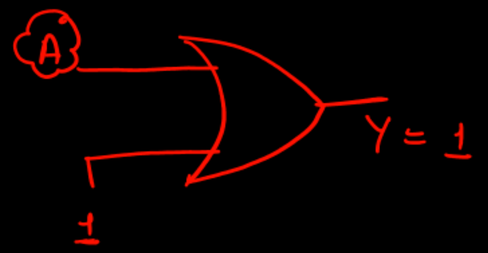
```
printf ("%d", (1 != 0) || (10 == 12) || (15 > 10));
```

True || Skip

any 1 statement = True  O/P = True

* True OR something = True
  ↳ Disabled OR gate

$[1 + x = 1]$ ⇒

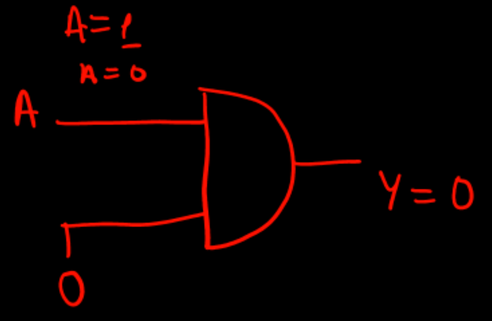



```
printf ("%d", (16 != 16) && (10 == 10), (2 < 3));
```

False (0) && Skip

* False AND Something = False
  ↳ Disabled AND gate

$[0 \cdot x = 0]$




```
printf ("%d", 1 && 1 && 1);
```
⇓ 1

1 && 1 && 1 → 1

```
printf ("%d", 0 && 1 && 1 && 1);
```
⇓ 0

0 && Skip → false

## NOT (Logical)

↳ Return True (1) if Statement is false

↳ !

```
printf ("%d", !(10==10))
```

!(10==10) → !True (1) ⇒ !True = false

!1 = 0

$$\begin{bmatrix} !1 = 0 \\ !0 = 1 \end{bmatrix}$$

* Inverse the logic

* Except 'Zero' Every integer is true

↳ Number

0 → false

relationalop.c | logicalop.c

logicalop.c > main()

```c
#include<stdio.h>

int main(){
    printf("%d\n", !(10==5)); //True
    printf("%d\n", !(10==10)); //False
    printf("%d\n", !1);
    printf("%d\n", !0);
    printf("%d\n", !5); // 0 ke alava, sab true hai
    printf("%d \n", !(-5));
    // printf("%d", (10==12) || (12!=15) || (1.0 == 1));

    // printf("%d \n", (10 == 10) && (5<6)); //True (1)
    // printf("%d \n", 10!=10 && 15==15); //False
    return 0;
}
```

```
1
0
0
1
0
0
PS>
```