

Lecture - 09

Programming in C

[Operators - 03]

✓ Unary op — $\begin{cases} \rightarrow ++ \\ \rightarrow -- \\ \rightarrow - \end{cases}$

Binary op — $\begin{cases} \text{Assignment operators} \rightarrow =, +=, -=, *=, /=, \% = \\ \text{Arithmetic Operators} \rightarrow +, -, *, /, \% \\ \text{Relational Operators} \\ \text{Logical Operators} \\ \text{Bitwise operator} \end{cases}$

Binary op — 'Shorthand if-else'

* Shift operators <<, >>

Relational Operator: \rightarrow Compare two operands \rightarrow Return Boolean result
 \hookrightarrow Comparison operator
 \hookrightarrow True / False

\rightarrow It check for the relationship between two values
 \hookrightarrow Equality & Inequality

Equality

\hookrightarrow equal to ($=$)

\hookrightarrow Not equal to (\neq)

Inequality

Strict

$\hookrightarrow <$
 $\hookrightarrow >$

Slack

$\hookrightarrow < =$
 $\hookrightarrow > =$

Relational operator \rightarrow two Results \leftarrow True (1)
False (0)

Relational op \rightarrow Result

$2 = 3 \Rightarrow$ No (False)

$2 < 3 \Rightarrow$ Yes (True)

$3 \neq 10 \Rightarrow$ Yes (True)

$3 > 5 \Rightarrow$ No (False)

$5 < 10 =$ Yes (True)

$3 \leq 3 =$ Yes (True)

$3 \geq 3 =$ Yes (True)

a) is equals to (==)

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

Ex $3 = 3$

True (1)

$$\text{int } x = 17;$$

```
int y = 15;
```

`printf('%d', x == y);` \Rightarrow 0

\downarrow \uparrow
int false = 0

```
int x = 10;
```

```
printf('%d', x == 10);
```

↓
True (1)

b) is not equals to operator ($!=$) \rightarrow $!$ \leftarrow not

$$\hookrightarrow \not\sim \rightarrow ! =$$

Return True is both values are unequal

Ex 31 = 5

True (1)

```
int y = 45;
```

```
printf ("%d", y != 45)
```

↓
False (0)

```
int m = 0;
```

```
int n = 1;
```

printf("%d", m != n)

↓

True (1)

C relationalop.c > 📁 main()

```
1  # include <stdio.h>
2
3  int main(){
4      printf("%d \n", 10 == 10); //True
5      printf("%d\n", 10 != 10); // False
6      printf("%d\n", 10 == 10.0); // True
7      printf("%d\n", 10 == 10.1); // false
8      printf("%d \n", 'A' == 65); //True
9      int m = 24;
10     float n = 24.0f;
11     printf("%d", m == n); //True
12
13     return 0;
14 }
```

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

c) is less than (<)

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

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

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

d) is greater than (>):

↳ Return True if left operand is greater than Right operand.
 ↓
 1

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

`printf("%d", 10 > 5.012)`
 ↓ ↓
 1 True



The screenshot shows a code editor window with a tab labeled "relationalop.c". The code is a C program that demonstrates the use of relational operators. It includes the standard input/output header and defines a main function. Inside the main function, five printf statements are used to test various relational expressions. The expressions are: 10 < 10, 10 > 10, 19.524 > 19.52400, 10 < 20, and 20 > 10. The comments next to each statement indicate the expected boolean result: false, false, true, true, and true. The code is as follows:

```
1 #include <stdio.h>
2
3 int main(){
4     printf("%d \n", 10<10); // false
5     printf("%d \n", 10>10); // false
6     printf("%d \n", 19.524 > 19.52400);
7     printf("%d \n", 10<20); //True
8     printf("%d \n", 20>10); //True
9 }
```

On the right side of the editor, there is a vertical toolbar with icons for running, stepping through, and other debugging actions. Below the toolbar, there is a small window showing the output of the program, which displays the results of the relational expressions as 0 or 1.

e) is less than or equals to ($\leq \rightarrow \leq =$)

`printf('%d', 10 <= 10);`
 ↓
 1
 True

`printf("%d", 10 <= 9);`
 ↓
 0
 False

f} is greater than or equals to ($\geq \rightarrow \geq$)

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

Eg $\text{printf}(\text{"\%d", } \underbrace{10}_{\substack{\uparrow \\ 1}} \geq \underbrace{10}_{\text{True}});$ $\text{printf}(\text{"\%d", } \underbrace{20}_{\substack{\uparrow \\ 1}} \geq \underbrace{10}_{\text{True}});$

C relationalop.c ×

▶ ▾ □ ... >_

C relationalop.c > main()

```
1  # include <stdio.h>
2
3  int main(){
4      printf("%d \n", 10<=10); //True
5      printf("%d \n", 10 >= 10); //True
6      printf("%d \n", 15>=10); //True
7      printf("%d \n", 20>=30); //False
8
```

○ 1
1
1
0
PS>

Logical Operator:

↳ works with Binary input & generate Binary output

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

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

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Binary $\begin{cases} \text{True} \\ \text{False} \end{cases}$

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

↓
1

True 1 True 1

1 AND 1
1

$\&$ ← address of
 $\&\&$ ← logical AND

$\&\&$ ← AND

printf ("%d", $\underbrace{(10 == 10)}_1 \&\& \underbrace{(2 < 3)}_1 \&\& \underbrace{(3 != 3)}_0$);

0 1 1 0 ← low

```

C relationalop.c  C logicalop.c x
C logicalop.c > main()
1  #include<stdio.h>
2
3  int main(){
4      printf("%d \n", (10 == 10) && (5<6)); //True (1)
5      printf("%d \n", 10!=10 && 15==15); //False
6      return 0;
7  }

```

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

↘ 1

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

$a || b$
↖ a OR b

OR - 3 वा 4

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

True or Skip

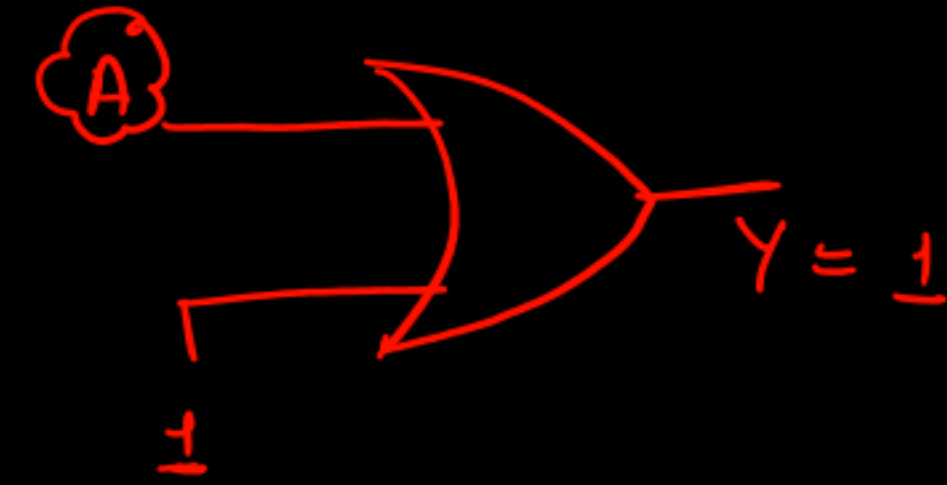
Any 1 statement = True

O/P = True

* True or something = True

↳ Disabled or gate

$$[1 + x = 1] \Rightarrow$$



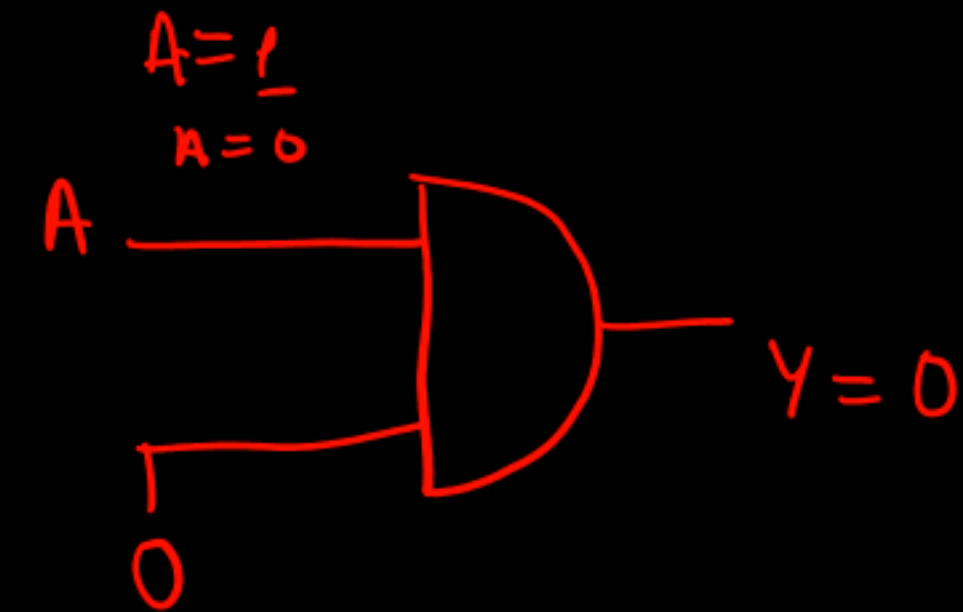
```
printf("%d", (16 != 16) && (10 == 10), (2 < 3));
```

False (0) AND Skip

* False AND something = False

↳ Disabled AND gate

$$[0 \cdot x = 0]$$



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

1

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

0 false Skip

NOT (Logical)

↳ Return True (1) If Statement is false
↳ !

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

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

! True (1) =

! True = false
! 1 = 0

* Inverse the logic

✓ Except 'Zero' Every integer
is true
↳ Number

0 → False

```
C relationalop.c  C logicalop.c x
C logicalop.c > main()
1  #include<stdio.h>
2
3  int main(){
4      printf("%d\n", !(10==5)); //True
5      printf("%d\n", !(10==10)); //False
6      printf("%d\n", !1);
7      printf("%d\n", !0);
8      printf("%d\n", !5); // 0 ke alava, sab true hai
9      printf("%d \n", !(-5));
10     // printf("%d", (10==12) || (12!=15) || (1.0 == 1));
11
12     // printf("%d \n", (10 == 10) && (5<6)); //True (1)
13     // printf("%d \n", 10!=10 && 15==15); //False
14     return 0;
15 }
```