Problem Solving Through programming in C

Course Code:ONL1001

Relational, logical Operators

Ms. Shubhra dwivedi
Assistant Professor
School - SCOPE
VIT-AP Amaravati
Shubhra.d@vitap.ac.in

Relational operators

- Relational operators are used for comparison of two values to understand the type of relationship a pair of number shares.
- The result of comparisons is what is called a boolean: a value TRUE or FALSE. FALSE is represented by 0; anything else is TRUE.
- The relational operators are:
  - < (less than)
  - <= (less than or equal to)
  - > (greater than)
  - >= (greater than or equal to)
  - == (equal to)
  - != (not equal to)

The table shows description of relational operators and also shows the working of each operators :

Operator	Name	Description	Result
==	Equal to	Checks if the values of two operands are equal or not	x==5 returns true
!=	Not equal to	Checks if the values of two operands are not equal or not	x!=5 returns false
<	Less than	Checks if the value of left operand is less than the value of right operand	x < 8 returns true
>	Greater than	Checks if the value of left operand is greater than the value of right operand	x > 8 returns false
<=	Less than equal to	Checks if the value of left operand is less than or equal to the value of right operand	x <=5 returns true
>=	Greater than equal to	Checks if the value of left operand is greater than or equal to the value of right operand	x>=5 returns true

Comparison Operators

Math Symbol	English	C Notation	C Sample	Math Equivalent
=	equal to	==	`x + 7 == 2*y`	$x + 7 = 2y$
≠	not equal to	!=	`ans != 'n'`	ans ≠ 'n'
<	less than	<	`count < m + 3`	count < m + 3
≤	less than or equal to	<=	`time <= limit`	time ≤ limit
>	greater than	>	`time > limit`	time > limit
≥	greater than or equal to	>=	`age >= 21`	age ≥ 21

```
#include <stdio.h>
int main()
{
   int a = 5, b = 6;
   printf("\n%d", a == b);
   printf("\n%d", a != b);
   printf("\n%d", a > b);
   printf("\n%d", a < b);
   printf("\n%d", a >= b);
   printf("\n%d", a <= b);
   return 0;
}
```

Output

```
0
1
0
1
0
1
```

When the expression was true, we got 1 and when false, 0.

```
#include <stdio.h>

int main()
{
   int a = 10, b = 4;

   // greater than example
   if (a > b)
      printf("a is greater than b\n");
   else
      printf("a is less than or equal to b\n");

   // greater than equal to
   if (a >= b)
      printf("a is greater than or equal to b\n");
   else
      printf("a is lesser than b\n");

   // less than example
   if (a < b)
      printf("a is less than b\n");
   else
      printf("a is greater than or equal to b\n");
```

```
// lesser than equal to
    if (a <= b)
        printf("a is lesser than or equal to b\n");
    else
        printf("a is greater than b\n");

    // equal to
    if (a == b)
        printf("a is equal to b\n");
    else
        printf("a and b are not equal\n");

    // not equal to
    if (a != b)
        printf("a is not equal to b\n");
    else
        printf("a is equal b\n");

    return 0;
}
```

Output:

a is greater than b
a is greater than or equal to b
a is greater than or equal to b
a is greater than b
a and b are not equal
a is not equal to b

Logical operators

(also called Boolean operators)

- These have Boolean operands and the result is also a Boolean.
- The basic Boolean operators are:
  - && (logical AND)
  - || (logical OR)
  - ! (logical NOT)

They are used to combine two or more conditions/constraints or to complement the evaluation of the original condition under consideration. They are described below:

1.Logical AND operator: The '&&' operator returns true when both the conditions under consideration are satisfied. Otherwise it returns false. For example, a && b returns true when both a and b are true (i.e. non-zero).

2.Logical OR operator: The '||' operator returns true even if one (or both) of the conditions under consideration is satisfied. Otherwise it returns false. For example, a || b returns true if one of a or b or both are true (i.e. non-zero). Of course, it returns true when both a and b are true.

3.Logical NOT operator: The '!' operator returns true the condition in consideration is not satisfied. Otherwise it returns false. For example, !a returns true if a is false, i.e. when a=0.

Truth Tables

AND

Exp_1	*Exp_2*	*Exp_1 && Exp_2*
true	*true*	*true*
true	*false*	*false*
false	*true*	*false*
false	*false*	*false*

OR

Exp_1	*Exp_2*	*Exp_1 || Exp_2*
true	*true*	*true*
true	*false*	*true*
false	*true*	*true*
false	*false*	*false*

NOT

Exp	*!(Exp)*
true	*false*
false	*true*

For example

Logical Operators		
Operator	Description	Example
&&	AND	x=6 y=3 x<10 && y>1 Return True
||	OR	x=6 y=3 x==5 || y==5 Return False
!	NOT	x=6 y=3 !(x==y) Return True

Eg:

if(age>55 **&&** sal<1000)

if(number<0 || number>100)

```
#include <stdio.h>
int main()
{
   int a = 5, b = 0;
   printf("\n%d", a && b);
   printf("\n%d", a || b);
   printf("\n%d", !a);
   printf("\n%d", !b);
   return 0;
}
```

```
Output
0
1
0
1
```

Since a is non-zero but b is zero, so AND between them will be false (or 0). As only one of them is true (or non-zero). But with OR, since anyone of them (i.e. a) is non-zero, so, a||b is true (or 1). In this example, since the value of 'a' is non-zero, therefore it is true. So, !a makes it false. The case with !b is the opposite.

```
// C program to demonstrate working of logical operators

#include <stdio.h>

int main()
{
    int a = 10, b = 4, c = 10, d = 20;

    // logical operators

    // logical AND example
    if (a > b && c == d)
        printf("a is greater than b AND c is equal to d\n");
    else
        printf("AND condition not satisfied\n");
```

```
 // logical OR example
    if (a > b || c == d)
        printf("a is greater than b OR c is equal to d\n");
    else
        printf("Neither a is greater than b nor c is equal to d\n");

    // logical NOT example
    if (!a)
        printf("a is zero\n");
    else
        printf("a is not zero");

    return 0;
}
```

Output:

AND condition not satisfied

a is greater than b OR c is equal to d

a is not zero