What is an Operand?

An operand is a value on which any operator works. For example, when we say **4+5**, here, numbers **4** and **5** are operands whereas **+** is an operator.

Different operators work with different numbers of operands like the + operator requires two operands or values.

C Arithmetic Operators

The C language supports all the basic arithmetic operators such as **addition**, **subtraction**, **multiplication**, **division**, etc.

The following table shows all the basic arithmetic operators along with their descriptions.

| **Operator** | **Description** | **Example**  **(where a and b are variables with some integer value)** |
| --- | --- | --- |
| + | adds two operands (values) | a+b |
| - | subtract second operands from first | a-b |
| \* | multiply two operands | a\*b |
| / | divide numerator by the denominator, i.e. divide the operand on the left side with the operand on the right side | a/b |
| % | This is the **modulus operato**r, it returns the remainder of the division of two operands as the result | a%b |
| ++ | This is the **Increment operator** - increases integer value by one. This operator needs only a **single operand**. | a++ or ++a |
| -- | This is the **Decrement operator** - decreases integer value by one. This operator needs only a **single operand**. | --b or b-- |

Prog 1:

#include <stdio.h>

int main() {

int a = 50, b = 23, result;

// addition

result = a+b;

printf("Addition of a & b = %d \n",result);

// subtraction

result = a-b;

printf("Subtraction of a & b = %d \n",result);

// multiplication

result = a\*b;

printf("Multiplication of a & b = %d \n",result);

// division

result = a/b;

printf("Division of a & b = %d \n",result);

return 0;

}

### Example: Using Modulus Operator (%)

The modulus operator returns the remainder value after the division of the provided values.

#include <stdio.h>

int main() {

int a = 23, b = 20, result;

// Using Modulus operator

result = a%b;

printf("result = %d",result);

return 0;

}

### Example: Using Increment and Decrement Operators

The increment operator is used to increase the value of any numeric value by 1, whereas the decrement operator is used to decrease the value of any numeric value by 1.

#include <stdio.h>

int main()

{

int a = 10, b = 20, c, d;

/\* Using increment operator \*/

printf("Incrementing value of a = %d \n", ++a);

/\* Using decrement operator \*/

printf("Decrementing value of b = %d \n", --b);

// first print value of a, then decrement a

printf("Decrementing value of a = %d \n", a--);

printf("Value of a = %d \n", a);

// first print value of b, then increment b

printf("Incrementing value of b = %d \n", b++);

printf("Value of b = %d \n", b);

return 0;

}

C Relational operators

The relational operators (or **comparison** operators) are used to check the relationship between two operands. It checks whether two operands are **equal** or **not equal** or **less than** or **greater than**, etc.

It returns **1** if the relationship checks **pass**, otherwise, it returns **0**.

For example, if we have **two numbers 14 and 7**, if we say **14 is greater than 7**, this is **true**, hence this check will **return 1** as the result with relationship operators. On the other hand, if we say **14 is less than 7**, this is **false**, hence it will **return 0**.

The following table shows all relational operators supported in the C language.

| **Operator** | **Description** | **Example**  **(a and b, where a = 10 and b = 11)** |
| --- | --- | --- |
| == | Check if two operands are equal | a == b, returns 0 |
| != | Check if two operands are not equal. | a != b, returns 1 because a is not equal to b |
| > | Check if the operand on the left is greater than the operand on the right | a > b, returns 0 |
| < | Check operand on the left is smaller than the right operand | a < b, returns 1 |
| >= | check left operand is greater than or equal to the right operand | a >= b, returns 0 |
| <= | Check if the operand on left is smaller than or equal to the right operand | a <= b, returns 1 |

#include <stdio.h>

int main() {

int a = 10, b = 20, result;

// Equal

result = (a==b);

printf("(a == b) = %d \n",result);

// less than

result = (a<b);

printf("(a < b) = %d \n",result);

// greater than

result = (a>b);

printf("(a > b) = %d \n",result);

// less than equal to

result = (a<=b);

printf("(a <= b) = %d \n",result);

return 0;

}

# Logical Operators in C

Logical operators are used to perform logical operations of given expressions (relational expressions) or variables.

There are three logical operators available in C.

|| (Logical OR) operator

If one of the operands or expressions is true, it will return 1.

If all of them are false, it will return 0.

| **A** | **B** | **A || B** | **Example** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | (5 > 10) || (5 < 4)         Both expressions are false. so, logical OR output will be 0 |
| 0 | 1 | 1 | (10 > 20) || (10 < 20)    First expression is false and second one is true. so, logical OR output will be 1 |
| 1 | 0 | 1 | (10 < 20) || (10 > 100)   First expression is true and second one is false. so, logical OR output will be 1 |
| 1 | 1 | 1 | (10 < 20) || (10 < 100)   Both expressions are true. so, logical OR output will be 1 |

&& (Logical AND) operator

If both left and right operands or expressions are true, it will return true. Otherwise, it will return false.

Note, non-zero value operands are considered as true.

| **A** | **B** | **A && B** | **Example** |
| --- | --- | --- | --- |
| 0 | 0 | 0 | (5 > 10) && (5 < 4)         Both expressions are false. so, logical AND output will be 0 |
| 0 | 1 | 0 | (10 > 20) && (10 < 20)    First expression is false and second one is true. so, logical AND output will be 0 |
| 1 | 0 | 0 | (10 < 20) && (10 > 100)   First expression is true and second one is false. so, logical AND output will be 0 |
| 1 | 1 | 1 | (10 < 20) && (10 < 100)   Both expressions are true. so, logical AND output will be 1 |

! (Logical NOT) operator

Logical NOT operator is used to inverse the current decision. Say, if current state is true, Logical NOT (!) operator will make it false.

| **A** | **!A** | **Example** |
| --- | --- | --- |
| 0 | 1 | !(100 < 10)   100 is greater than 10. So, it will return false. !(false) ==> true |
| 1 | 0 | !(10 < 100)   10 is less than 100. So, it will return true. !(true) ==> false |

Sample Program

Example

#include<stdio.h>

**int** **main**()

{

**int** a = **5**, b = **10** , ret;

ret = ( (a <= b) || (a != b) );

// 5 <= 10 ==> true. 5 != 10 ==> true. So, 1 || 1 will return 1.

printf("Return value of above expression is %d**\n**",ret);

ret = ( ( a < b) && (a == b ) );

// 5 < 10 ==>true. 5 == 10 ==> false. So, 1 && 0 will return 0;

printf("Return value of above expression is %d**\n**",ret);

ret = ! ( ( a < b) && (a == b ) );

/\*we have used the same expression here.

And its result was 0.

So !0 will be 1.\*/

printf("Return value of above expression is %d**\n**",ret);

**return** **0**;

}

C Assignment Operators

The sssignment operators are used to assign value to a variable. For example, if we want to assign a value **10** to a variable x then we can do this by using the assignment operator like: x = 10; Here, = (equal to) operator is used to assign the value.

In the C language, the = (equal to) operator is **used for assignment** however it has several other variants such as +=, -= to combine two operations in a single statement.

You can see all the assignment operators in the table given below.

| **Operator** | **Description** | **Example**  **(a and b are two variables, with where a=10 and b=5)** |
| --- | --- | --- |
| = | assigns values from right side operand to left side operand | a=b, a gets value 5 |
| += | adds right operand to the left operand and assign the result to left operand | a+=b, is same as a=a+b, value of a becomes 15 |
| -= | subtracts right operand from the left operand and assign the result to left operand | a-=b, is same as a=a-b, value of a becomes 5 |
| \*= | mutiply left operand with the right operand and assign the result to left operand | a\*=b, is same as a=a\*b, value of a becomes 50 |
| /= | divides left operand with the right operand and assign the result to left operand | a/=b, is same as a=a/b, value of a becomes 2 |
| %= | calculate modulus using two operands and assign the result to left operand | a%=b, is same as a=a%b, value of a becomes 0 |

#include <stdio.h>

int main() {

int a = 10;

// Assign

int result = a;

printf("result = %d \n",result);

// += operator

result += a;

printf("result = %d \n",result);

// -= operator

result -= a;

printf("result = %d \n",result);

// \*= operator

result \*= a;

printf("result = %d \n",result);

return 0;

}

## C Ternary Operator (?)

The ternary operator, also known as the conditional operators in the C language can be used for statements of the form if-then-else.

The **basic syntax** for using ternary operator is:

(Expression1)? Expression2 : Expression3;

Copy

Here is how it works:

* The question mark ? in the syntax represents the if part.
* The first expression (expression 1) returns either **true** or **false**, based on which it is decided whether (expression 2) will be executed or (expression 3)
* If (expression 1) returns **true** then the (expression 2) is executed.
* If (expression 1) returns **false** then the expression on the right side of : i.e (expression 3) is executed.

### Example: Using Ternary Operator

#include <stdio.h>

int main() {

int a = 20, b = 20, result;

/\* Using ternary operator

- If a == b then store a+b in result

- otherwise store a-b in result

\*/

result = (a==b)?(a+b):(a-b);

printf("result = %d",result);

return 0;

}