An operator is a symbol that operates on a value or a variable. For example: + is an operator to perform addition.

C has a wide range of operators to perform various operations.

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

**Arithmetic operators**

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

| **Operator** | **Meaning of Operator** |
| --- | --- |
| + | addition or unary plus |
| - | subtraction or unary minus |
| \* | multiplication |
| / | division |
| % | remainder after division (modulo division) |

**Assignment Operators**

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

| **Operator** | | **Example** | | **Same as** |
| --- | --- | --- | --- | --- |
| = |  | a = b |  | a = b |
| += |  | a += b |  | a = a+b |
| -= |  | a -= b |  | a = a-b |
| \*= |  | a \*= b |  | a = a\*b |
| /= |  | a /= b |  | a = a/b |
| %= |  | a %= b |  | a = a%b |

### Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in [decision making](https://www.programiz.com/c-programming/c-if-else-statement) and [loops](https://www.programiz.com/c-programming/c-for-loop).

| **Operator** | **Meaning of Operator** | **Example** |
| --- | --- | --- |
| == | Equal to | 5 == 3 is evaluated to 0 |
| > | Greater than | 5 > 3 is evaluated to 1 |
| < | Less than | 5 < 3 is evaluated to 0 |
| != | Not equal to | 5 != 3 is evaluated to 1 |
| >= | Greater than or equal to | 5 >= 3 is evaluated to 1 |
| <= | Less than or equal to | 5 <= 3 is evaluated to 0 |

### Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in [decision making in C programming](https://www.programiz.com/c-programming/c-if-else-statement).

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| && | Logical AND. True only if all operands are true | If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0. |
| || | Logical OR. True only if either one operand is true | If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1. |
| ! | Logical NOT. True only if the operand is 0 | If c = 5 then, expression !(c==5) equals to 0. |

1. // Working of logical operators
2. #include <stdio.h>
3. int main()
4. {
5. int a = 5, b = 5, c = 10, result;
6. result = (a == b) && (c > b);
7. printf("(a == b) && (c > b) is %d \n", result);
8. result = (a == b) && (c < b);
9. printf("(a == b) && (c < b) is %d \n", result);
10. result = (a == b) || (c < b);
11. printf("(a == b) || (c < b) is %d \n", result);
12. result = (a != b) || (c < b);
13. printf("(a != b) || (c < b) is %d \n", result);
14. result = !(a != b);
15. printf("!(a == b) is %d \n", result);
16. result = !(a == b);
17. printf("!(a == b) is %d \n", result);
18. return 0;
19. }

Output

(a == b) && (c > b) is 1

(a == b) && (c < b) is 0

(a == b) || (c < b) is 1

(a != b) || (c < b) is 0

!(a != b) is 1

!(a == b) is 0

**Explanation of logical operator program**

* (a == b) && (c > 5) evaluates to 1 because both operands (a == b) and (c > b) is 1 (true).
* (a == b) && (c < b) evaluates to 0 because operand (c < b) is 0 (false).
* (a == b) || (c < b) evaluates to 1 because (a = b) is 1 (true).
* (a != b) || (c < b) evaluates to 0 because both operand (a != b) and (c < b) are 0 (false).
* !(a != b) evaluates to 1 because operand (a != b) is 0 (false). Hence, !(a != b) is 1 (true).
* !(a == b) evaluates to 0 because (a == b) is 1 (true). Hence, !(a == b) is 0 (false).

### Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

| **Operators** | **Meaning of operators** |
| --- | --- |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |
| ~ | Bitwise complement |
| << | Shift left |
| >> | Shift right |

### C Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

| **Operators** | **Meaning of operators** |
| --- | --- |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |
| ~ | Bitwise complement |
| << | Shift left |
| >> | Shift right |

## C Increment and Decrement Operators

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

### Example 2: Increment and Decrement Operators

1. // Working of increment and decrement operators
2. #include <stdio.h>
3. int main()
4. {
5. int a = 10, b = 100;
6. float c = 10.5, d = 100.5;
7. printf("++a = %d \n", ++a);
8. printf("--b = %d \n", --b);
9. printf("++c = %f \n", ++c);
10. printf("--d = %f \n", --d);
11. return 0;
12. }

**Output**

++a = 11

--b = 99

++c = 11.500000

++d = 99.500000

Here, the operators ++ and -- are used as prefixes. These two operators can also be used as postfixes like a++ and a--. Visit this page to learn more about how [increment and decrement operators work when used as postfix](https://www.programiz.com/article/increment-decrement-operator-difference-prefix-postfix).