C++ Operators

programiz.com/cpp-programming/operators

Join our newsletter for the latest updates.

In this tutorial, we will learn about the different types of operators in C++ with the help of examples. In programming, an operator is a symbol that operates on a value or a variable.

Operators are symbols that perform operations on variables and values. For example, `+` is an operator used for addition, while `-` is an operator used for subtraction.

Operators in C++ can be classified into 6 types:

1. C++ Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on variables and data. For example,

```
a + b;
```

Here, the `+` operator is used to add two variables *a* and *b*. Similarly there are various other arithmetic operators in C++.

Operator	Operation
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`%`	Modulo Operation (Remainder after division)

Example 1: Arithmetic Operators

```
#include <iostream>
using namespace std;

int main() {
    int a, b;
    a = 7;
    b = 2;

    // printing the sum of a and b
    cout << "a + b = " << (a + b) << endl;

    // printing the difference of a and b
    cout << "a - b = " << (a - b) << endl;

    // printing the product of a and b
    cout << "a * b = " << (a * b) << endl;

    // printing the division of a by b
    cout << "a / b = " << (a / b) << endl;

    // printing the modulo of a by b
    cout << "a % b = " << (a % b) << endl;

    return 0;
}
```

Output

```
a + b = 9
a - b = 5
a * b = 14
a / b = 3
a % b = 1
```

Here, the operators `+`, `-` and `*` compute addition, subtraction, and multiplication respectively as we might have expected.

/ Division Operator

Note the operation `(a / b)` in our program. The `/` operator is the division operator.

As we can see from the above example, if an integer is divided by another integer, we will get the quotient. However, if either divisor or dividend is a floating-point number, we will get the result in decimals.

```
In C++,

7/2 is 3
7.0 / 2 is 3.5
7 / 2.0 is 3.5
7.0 / 2.0 is 3.5
```

% Modulo Operator

The modulo operator `%` computes the remainder. When `a = 9` is divided by `b = 4`, the remainder is **1**.

Note: The `%` operator can only be used with integers.

Increment and Decrement Operators

C++ also provides increment and decrement operators: `++` and `--` respectively.

- `++` increases the value of the operand by **1**
- `--` decreases it by **1**

For example,

```
int num = 5;

// increment operator
++num;  // 6
```

Here, the code `++num;` increases the value of *num* by **1**.

Example 2: Increment and Decrement Operators

```
// Working of increment and decrement operators

#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 100, result_a, result_b;

    // incrementing a by 1 and storing the result in result_a
    result_a = ++a;
    cout << "result_a = " << result_a << endl;


    // decrementing b by 1 and storing the result in result_b
    result_b = --b;
    cout << "result_b = " << result_b << endl;

    return 0;
}
```

Output

```
result_a = 11
result_b = 99
```

In the above program, we have used the `++` and `--` operators as **prefixes (++a and --b)**. However, we can also use these operators as **postfix (a++ and b--)**.

To learn more, visit increment and decrement operators.

2. C++ Assignment Operators

In C++, assignment operators are used to assign values to variables. For example,

```
// assign 5 to a
a = 5;
```

Here, we have assigned a value of `5` to the variable *a*.

Operator	Example	Equivalent to
`=`	`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;`

Example 3: Assignment Operators

```
#include <iostream>
using namespace std;

int main() {
    int a, b;

    // 2 is assigned to a
    a = 2;

    // 7 is assigned to b
    b = 7;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "\nAfter a += b;" << endl;

    // assigning the sum of a and b to a
    a += b;  // a = a +b
    cout << "a = " << a << endl;

    return 0;
}
```

Output

```
a = 2
b = 7

After a += b;
a = 9
```

3. C++ Relational Operators

A relational operator is used to check the relationship between two operands. For example,

```
// checks if a is greater than b
a > b;
```

Here, `>` is a relational operator. It checks if *a* is greater than *b* or not.

If the relation is **true**, it returns **1** whereas if the relation is **false**, it returns **0**.

Operator	Meaning	Example
`==`	Is Equal To	`3 == 5` gives us **false**
`!=`	Not Equal To	`3 != 5` gives us **true**
`>`	Greater Than	`3 > 5` gives us **false**
`<`	Less Than	`3 < 5` gives us **true**
`>=`	Greater Than or Equal To	`3 >= 5` give us **false**
`<=`	Less Than or Equal To	`3 <= 5` gives us **true**

Example 4: Relational Operators

```
#include <iostream>
using namespace std;

int main() {
    int a, b;
    a = 3;
    b = 5;
    bool result;

    result = (a == b);   // false
    cout << "3 == 5 is " << result << endl;

    result = (a != b);  // true
    cout << "3 != 5 is " << result << endl;

    result = a > b;   // false
    cout << "3 > 5 is " << result << endl;

    result = a < b;   // true
    cout << "3 < 5 is " << result << endl;

    result = a >= b;  // false
    cout << "3 >= 5 is " << result << endl;

    result = a <= b;  // true
    cout << "3 <= 5 is " << result << endl;

    return 0;
}
```

Output

```
3 == 5 is 0
3 != 5 is 1
3 > 5 is 0
3 < 5 is 1
3 >= 5 is 0
3 <= 5 is 1
```

Note: Relational operators are used in decision-making and loops.

4. C++ Logical Operators

Logical operators are used to check whether an expression is **true** or **false**. If the expression is **true**, it returns **1** whereas if the expression is **false**, it returns **0**.

Operator	Example	Meaning
`&&`	expression1 **&&** expression2	Logical AND. True only if all the operands are true.
`||`	expression1 **||** expression2	Logical OR. True if at least one of the operands is true.

!	!expression	Logical NOT. True only if the operand is false.

In C++, logical operators are commonly used in decision making. To further understand the logical operators, let's see the following examples,

```
Suppose,
a = 5
b = 8

Then,

(a > 3) && (b > 5) evaluates to true
(a > 3)  && (b < 5) evaluates to false

(a > 3) || (b > 5) evaluates to true
(a > 3) || (b < 5) evaluates to true
(a < 3) || (b < 5) evaluates to false

!(a < 3) evaluates to true
!(a > 3) evaluates to false
```

Example 5: Logical Operators

```
#include <iostream>
using namespace std;

int main() {
    bool result;

    result = (3 != 5) && (3 < 5);     // true
    cout << "(3 != 5) && (3 < 5) is " << result << endl;

    result = (3 == 5) && (3 < 5);    // false
    cout << "(3 == 5) && (3 < 5) is " << result << endl;

    result = (3 == 5) && (3 > 5);    // false
    cout << "(3 == 5) && (3 > 5) is " << result << endl;

    result = (3 != 5) || (3 < 5);    // true
    cout << "(3 != 5) || (3 < 5) is " << result << endl;

    result = (3 != 5) || (3 > 5);    // true
    cout << "(3 != 5) || (3 > 5) is " << result << endl;

    result = (3 == 5) || (3 > 5);    // false
    cout << "(3 == 5) || (3 > 5) is " << result << endl;

    result = !(5 == 2);    // true
    cout << "!(5 == 2) is " << result << endl;

    result = !(5 == 5);    // false
    cout << "!(5 == 5) is " << result << endl;

    return 0;
}
```

Output

```
(3 != 5) && (3 < 5) is 1
(3 == 5) && (3 < 5) is 0
(3 == 5) && (3 > 5) is 0
(3 != 5) || (3 < 5) is 1
(3 != 5) || (3 > 5) is 1
(3 == 5) || (3 > 5) is 0
!(5 == 2) is 1
!(5 == 5) is 0
```

Explanation of logical operator program

- `(3 != 5) && (3 < 5)` evaluates to **1** because both operands `(3 != 5)` and `(3 < 5)` are **1** (true).
- `(3 == 5) && (3 < 5)` evaluates to **0** because the operand `(3 == 5)` is **0** (false).
- `(3 == 5) && (3 > 5)` evaluates to **0** because both operands `(3 == 5)` and `(3 > 5)` are **0** (false).
- `(3 != 5) || (3 < 5)` evaluates to **1** because both operands `(3 != 5)` and `(3 < 5)` are **1** (true).
- `(3 != 5) || (3 > 5)` evaluates to **1** because the operand `(3 != 5)` is **1** (true).
- `(3 == 5) || (3 > 5)` evaluates to **0** because both operands `(3 == 5)` and `(3 > 5)` are **0** (false).
- `!(5 == 2)` evaluates to **1** because the operand `(5 == 2)` is **0** (false).
- `!(5 == 5)` evaluates to **0** because the operand `(5 == 5)` is **1** (true).

5. C++ Bitwise Operators

In C++, bitwise operators are used to perform operations on individual bits. They can only be used alongside `char` and `int` data types.

Operator	Description
`&`	Binary AND
`|`	Binary OR
`^`	Binary XOR
`~`	Binary One's Complement
`<<`	Binary Shift Left
`>>`	Binary Shift Right

To learn more, visit C++ bitwise operators.

6. Other C++ Operators

Here's a list of some other common operators available in C++. We will learn about them in later tutorials.

Operator	Description	Example
`sizeof`	returns the size of data type	`sizeof(int); // 4`
`?:`	returns value based on the condition	`string result = (5 > 0) ? "even" : "odd"; // "even"`
`&`	represents memory address of the operand	`&num; // address of num`
`.`	accesses members of struct variables or class objects	`s1.marks = 92;`
`->`	used with pointers to access the class or struct variables	`ptr->marks = 92;`
`<<`	prints the output value	`cout << 5;`
`>>`	gets the input value	`cin >> num;`