Logical Operators

An **operator** is a symbol that operates on a value to perform specific mathematical or logical computations. They form the foundation of any programming language. In C++, we have built-in operators to provide the required functionality.

An operator operates the **operands**. For example,

```
int c = a + b;
```

Here, '+' is the addition operator. 'a' and 'b' are the operands that are being 'added'.

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

1. Arithmetic Operators
2. Relational Operators
3. **Logical Operators**
4. Bitwise Operators
5. Assignment Operators
6. Ternary or Conditional Operators

Logical Operators

These operators are used to combine two or more conditions or constraints or to complement the evaluation of the original condition in consideration. The result returns a Boolean value, i.e., **true** or **false**.

Name	Symbol	Description	Example

Logical AND	&&	Returns true only if all the operands are true or non-zero	int a = 3, b = 6;
			a&&b;
			// returns true

Logical OR	||	Returns true if either of the operands is true or non-zero	int a = 3, b = 6;
			a||b;
			// returns true

Logical NOT	!	Returns true if the operand is false or zero	int a = 3;
			!a;
			// returns false

```
// CPP Program to demonstrate the Logical Operators
#include <iostream>
using namespace std;

int main()
{
    int a = 6, b = 4;

    // Logical AND operator
    cout << "a && b is " << (a && b) << endl;

    // Logical OR operator
    cout << "a ! b is " << (a > b) << endl;

    // Logical NOT operator
    cout << "!b is " << (!b) << endl;

    return 0;
}
```

Output

```
a && b is 1
a ! b is 1
!b is 0
```

Here, **0** denotes **false** and **1** denotes **true**.

Short-Circuiting in Logical Operators:

- In the case of **logical AND**, the second operand is not evaluated if the first operand is false. For example, program 1 below doesn't print "GeeksQuiz" as the first operand of logical AND itself is false.

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

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

```
    bool res = ((a == b) && cout << "GeeksQuiz");
    return 0;
}
```

Output

- But the below program prints “GeeksQuiz” as the first operand of logical AND is true.

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

int main()
{
    int a = 10, b = 4;
    bool res = ((a != b) && cout << "GeeksQuiz");
    return 0;
}
```

Output

`GeeksQuiz`

- In the case of **logical OR**, the second operand is not evaluated if the first operand is true. For example, program 1 below doesn’t print “GeeksQuiz” as the first operand of logical OR itself is true.

```
#include <iostream>
using namespace std;
int main()
{
    int a = 10, b = 4;
    bool res = ((a != b) || cout << "GeeksQuiz");
    return 0;
}
```

Output

- But the below program prints “GeeksQuiz” as the first operand of logical OR is false.

```
#include <iostream>
using namespace std;
int main()
{
    int a = 10, b = 4;
    bool res = ((a == b) || cout << "GeeksQuiz");
    return 0;
}
```

Output

`GeeksQuiz`

There is also a **three-way comparison operator** in C++ called <=>, which is sometimes called the **"spaceship operator."** This operator compares two values and returns:

- 0 if the values are equal
- 1 if the first value is greater than the second value
- 1 if the first value is less than the second value

This operator was introduced in **C++20** and can be used as an alternative to chaining together multiple comparisons using the <, ==, and > operators.

```
int result = a <=> b;
if (result == 0) {
    // a and b are equal}
    else if (result == 1) {
    // a is greater than b}
    else if (result == -1) {
    // a is less than b
}
```