



# Programming in C++

## Operators in C++



Dr. Zeyad Safaa Younus Saffawi

# Operators in C++

- **Operators** are **special symbols** used to **perform operations on variables and values**.
  They are mainly divided into different types, such as:

  - **Arithmetic Operators**
  - **Relational Operators**
  - **Logical Operators**
  - **Bitwise Operators**
  - **Assignment Operators**

# Operators in C++

**1. Arithmetic Operators** are used to perform basic mathematical calculations like addition, subtraction, multiplication, and division.

| Operator | Meaning | Example | Result |
|---|---|---|---|
| + | Addition | x + y | Sum of x and y |
| - | Subtraction | x - y | Difference between x and y |
| * | Multiplication | x * y | Product of x and y |
| / | Division | x / y | Quotient of x divided by y |
| % | Modulus | x % y | Remainder of x divided by y |
| ++ | Increment by 1 | x++ or ++x | Increases value of x by 1 |
| - - | Decrement by 1 | x - - or - - x | Decreases value of x by 1 |

# Operators in C++

- **Difference Between Pre and Post Increment**

| Form | Description | Example |
|---|---|---|
| + +x | **Pre-increment: Increases x before using it in an expression** | **If x=3, then y=++x → x=4, y=4** |
| x+ + | **Post-increment: Uses x first, then increases it after** | **If x=3, then y=x++ → x=4, y=3** |

- The same logic applies to - -x (pre-decrement) and x- - (post-decrement).

# Operators in C++

Example: The program performs **basic arithmetic operations**. It demonstrates the difference between post-increment (x++) and pre-decrement (--y).

```
#include <iostream>
using namespace std;
int main()
{
   int x = 5, y = 2;
   cout << "Addition: " << x + y << endl;
   cout << "Subtraction: " << x - y << endl;
   cout << "Multiplication: " << x * y << endl;
   cout << "Division: " << x / y << endl;
   cout << "Remainder: " << x % y << endl;
   cout << "\n Post Increment (x++): " << x++ << endl;  // Uses x= 5, then x becomes 6
   cout << "After increment, x = " << x << endl;
   cout << "Pre Decrement (--y): " << --y << endl;    // Decreases y =1 before using it
   return 0;
}
```

# Operators in C++

**2. Relational Operators are u**sed for decision making (The result is **Boolean,** true (1) or false(0)). For example, (**compare two values** (numbers, characters, or variables)).

| Operator | Meaning | Example | Result |
|---|---|---|---|
| = = | Equal to | a = = b | True if a equals b |
| ! = | Not equal to | a != b | True if a and b are different |
| < | Less than | a < b | True if a is smaller than b |
| < = | Less than or equal to | a <= b | True if a is smaller or equal to b |
| > | Greater than | a > b | True if a is larger than b |
| > = | Greater than or equal to | a >= b | True if a is larger or equal to b |

# Operators in C++

## Example: **Relational Operators**

```
#include <iostream>
using namespace std;
int main ()
{
   int x = 15,    y = 6;
   int z;
   z = (x = = y);                          // 15 == 6 → false → 0
   cout << "x = = y : " << z << endl;
   z = (x != y);                           // 15 != 6 → true → 1
   cout << "x != y : " << z << endl;
   z = (x < y);                            // 15 < 6 → false → 0
   cout << "x < y  : " << z << endl;
   z = (x <= y);                           // 15 <= 6 → false → 0
   cout << "x <= y : " << z << endl;
   z = (x > y);                            // 15 > 6 → true → 1
   cout << "x > y  : " << z << endl;
   z = (x >= y);                           // 15 >= 6 → true → 1
   cout << "x >= y : " << z << endl;
   z = ('p' > 'b');              // character comparison using ASCII values → true → 1
   cout << "'p' > 'b' : " << z << endl;
   return 0;
}
```

# Operators in C++

**3. Logical Operators** are used to **combine or reverse Boolean expressions**. They are mainly used in **decision-making statements** like if, while, etc.

| Operator | Name | Action / Meaning |
|---|---|---|
| && | AND | True if both conditions are true |
| \|\| | OR | True if any conditions are true |
| ! | NOT | Reverses the boolean value (True → False, False → True) |

**Examples:**

**AND (&&)**

```
int a = 5, b = 10;
if (a > 0 && b > 5) {
   cout << "Both conditions are true";
}
```

**OR (||)**

```
int a = 5, b = 2;
if (a > 0 || b > 5) {
    cout << "At least one condition is true";
}
```

**NOT (!)**

```
bool status = false;
if (!status) {
   cout << "Status is false";
}
```

- **Notes:**
  - Use && when **all conditions** must be true.
  - Use || when **any condition** can be true.
  - Use ! to **reverse** a condition.

# Operators in C++

**4. Bitwise operators** work on the **individual bits** (0s and 1s) of integers. They do operations **bit by bit**. They are mostly used in **low-level programming**, **hardware control**, **encryption**, and **performance optimization**.

| A | B | A & B (AND) | A \| B (OR) | A ^ B (XOR) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | **1** | **1** |
| 1 | 0 | 0 | **1** | **1** |
| 1 | 1 | **1** | **1** | 0 |

# Operators in C++

**Bitwise operators**

| A | ~A (Not) |
|---|---|
| 0 | **1** |
| 1 | **0** |

| Operation | Bit Effect | Example |
|---|---|---|
| A << 1 | Move bits left, fill with **0** | 00000110 → 00001100 |
| A >> 1 | Move bits right, fill depends on type | 00000110 → 00000011 |

# Operators in C++

- **Bitwise operators: example when a = 6 and b = 3**
- **a = 6** → binary: **00000110**
- **b = 3** → binary: **00000011**

**AND**
**00000110**
**& 00000011**
**----------**
**00000010 = 2**

**OR**
**00000110**
**| 00000011**
**----------**
**00000111 = 7**

**XOR**
**00000110**
**^ 00000011**
**----------**
**00000101 = 5**

**NOT**
**~ 00000110**
**= 11111001 (depends on sign & system)**

**Left Shift ( << )**
**00000110 << 1**
**= 00001100 = 12**

**Right Shift ( >> )**
**00000110 >> 1**
**= 00000011 = 3**

# Operators in C++

- **Bitwise operators: example**

```
#include <iostream>
using namespace std;
int main()
{
unsigned int a = 60; // 60 = 0011 1100
unsigned int b = 13; // 13 = 0000 1101
int c = 0;
c = a & b;                  // result is : 12 = 0000 1100
cout << "Value of c is : " << c << endl ;
c = a | b;                  // result is : 61 = 0011 1101
cout << "Value of c is: " << c << endl ;
c = a ^ b;                  // result is : 49 = 0011 0001
cout << "Value of c is: " << c << endl ;
c = ~a;                     // result is : -61 = 1100 0011
cout << "Value of c is: " << c << endl ;
c = a << 2;                 // result is : 240 = 1111 0000
cout << "Value of c is: " << c << endl ;
c = a >> 2;                 // result is : 15 = 0000 1111
cout << "Value of c is: " <<c << endl ;
return 0;
}
```

# Operators in C++

**5. Assignment Operators** are **short forms** that combine an **arithmetic operation** with an **assignment (=)** in one step.

- The **assignment operator (=)** is used to **assign a value or the result of an expression** to a variable.
- It copies the value on the **right-hand side (RHS)** into the variable on the **left-hand side (LHS)**. For example: int x = 7; // 'x' is initialized with 7

- Instead of writing both separately (like x = x + 5), you can write it more simply (x += 5).

| **Operator** | Long Form (Full Expression) | Short Form (Compound) | Description |
|---|---|---|---|
| **+ =** | num1 = num1 + 5 | num1 += 5 | Adds 5 to num1 |
| **- =** | count = count - 3 | count -= 3 | Subtracts 3 from count |
| *** =** | price = price * 2 | price *= 2 | Multiplies price by 2 |
| **/ =** | total = total / 4 | total /= 4 | Divides total by 4 |
| **% =** | remainder = remainder % 3 | remainder %= 3 | Stores the remainder of remainder / 3 |

# Operators in C++

- **Example of Compound Assignment Operators**

```
#include <iostream>
using namespace std;
int main()
{
   int num1 = 10, count = 25, price = 6, total = 40, remainder = 14;
   num1 += 5;                  // num1 = num1 + 5 → num1 = 15
   count -= 3;                 // count = count - 3 → count = 22
   price *= 2;                 // price = price * 2 → price = 12
   total /= 4;                 // total = total / 4 → total = 10
   remainder %= 3;      // remainder = remainder % 3 → remainder = 14 % 3 = 2
   cout << "After + =  , num1 = " << num1 << endl;
   cout << "After - =  , count = " << count << endl;
   cout << "After * =  , price = " << price << endl;
   cout << "After / =  , total = " << total << endl;
   cout << "After % =  , remainder = " << remainder << endl;
   return 0;
}
```

# Operators in C++

- **Precedence of Operators in C++** determines the **order in which operations are performed** in an expression. Operators with **higher precedence** are evaluated first.

| Priority | Operator | Example | Explanation / Result |
|---|---|---|---|
| 1 | ( ) | int a = (2 + 3) * 4; | Parentheses evaluated first → (2+3)=5 → 5*4=20 |
| 2 | ! , ++ , - - (prefix) | int b = 5; int c = ++b; | ++b increments before use → b=6, c=6 |
| 3 | * , / , % | int d = 10 * 2 / 5; | Multiplication/division/modulus → 10*2=20, 20/5=4 |
| 4 | + , - | int e = 5 + 3 - 2; | Addition and subtraction → 5+3=8, 8-2=6 |
| 5 | < , <= , > , >= | bool f = 10 > 5; | Comparison → true (1) |
| 6 | == , != | bool g = (a == b); | Equality/Inequality → false (0) |
| 7 | & | bitwise and y=a & b | 0&0→0 0&1→0 1&0→0 1&1→1 |
| 8 | ^ | bitwise xor y=a ^ b | 0^0→0 0^1→1 1^0→1 1^1→0 |
| 9 | \| | bitwise or y a \| b | 0\|0→0 0\|1→1 1\|0→1 1\|1→1 |
| 10 | && | bool h = (a > b) && (b < c); | Logical AND → true if both true |
| 11 | \|\| | bool h = (a > b) \|\| (b < c); | Logical OR → true if any true |
| 12 | ? : | int j = (a > b) ? a : b; | Ternary operator → chooses value based on condition |
| 13 | + = , - = , * = , / = , = | a += 5; | Assignment operators evaluated after arithmetic |
| 14 | ++ , -- (postfix) | int k = b++; | Post-increment → use value first, then increment → k=6, b=7 |

# Operators in C++

- **Examples of <u>Operators Precedence</u>**

```
int   p = 5,   q = 3, r;
r = (p + q) * 2;                    // Parentheses first: (5+3)=8 → 8*2=16
int    s = 4;
int   t = ++ s;                     // Prefix increment: s=5, t=5
int    u = 10,    v = 2;
int w = u * v / 5;                  // Multiplication/Division first: 10*2=20, then 20/5=4
bool check = (p > q) && (q < r);     // Logical AND:  true && true → true
bool status = (p == q) || (r > s);   // Logical OR: false || true → true
```

# Operators in C++

- **Comparison and Logical Operators**

```
int m = 7, n = 10, result;
result = (m > n) && (n > 5);          // 0 && 1 → 0
cout << result << endl;               // Output: 0

result = (m < n) || (n < 5);          // 1 || 0 → 1
cout << result << endl;               // Output: 1

result = !(m == n);                   // !(0) → 1
cout << result << endl;               // Output: 1
```

# References


- Gaddis, T. (2014). *Starting out with C++: From control structures through objects* (8th ed.). Pearson.
- Soulié, J. (2007, April 24). C++ language tutorial. cplusplus.com.
- Tutorials Point. (n.d.). *Learn C++ programming language*. Tutorials Point.