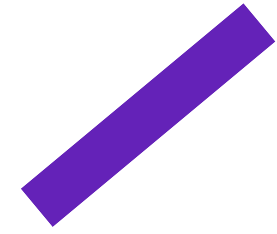




Operators in C++

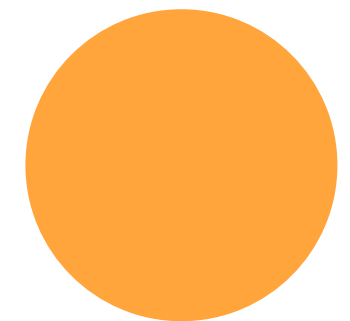
Let's crack operators !

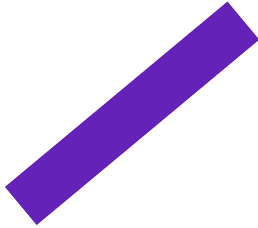




C++ Operators

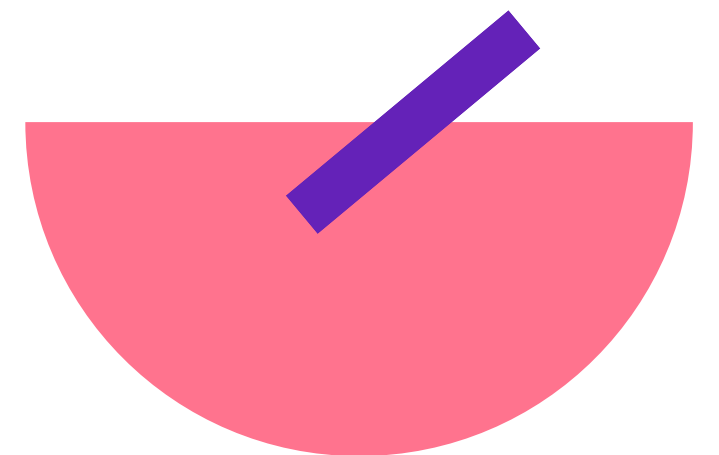
Operators are used to perform operations on variables and values.

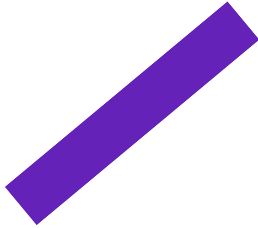




Types of Operators

1. Arithmetic (+, -, /, *, %)
2. Unary(-, !, ++, --, ~)
3. Relational (==, !=, >, <, >=, <=)
4. Logical(&&, ||, !)
5. Bitwise(&, ||, ~, ^, <<, >>)
6. Assignment(+=, =, -=, *=, /=, %=)
7. Misc (sizeof, ' ', ? : , &)




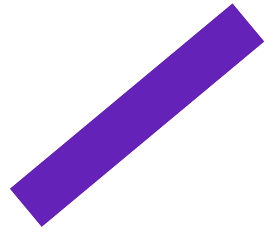


Arithmetic Operators

1. Arithmetic Operators:

Arithmetic operators are used for performing common mathematical operations, i.e., Addition, Subtraction, Multiplication, and division. The basic arithmetic operators in C++ are given below:

- **Addition Operator (+):** It is used to add two numbers.
 - **Subtraction Operator (-):** It is used to subtract two numbers.
 - **Multiplication Operator (*):** It is used to multiply two numbers.
 - **Division Operator (/):** It is used to divide two numbers. Mind that it gives the quotient.
 - **Modulus Operator (%):** It is used to retrieve the remainder from the division operation.
- 

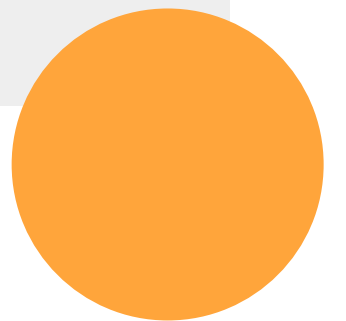


Unary Operators

2. Unary Operators:

Unary Operators are the types of operators that require only one operand. They form various operations on single operands, such as incrementing or decrementing the value by one, negating an expression, or inverting a boolean's value. Let's understand the various unary operators with an example.

(i) Unary minus operator (-): This operator can be used to convert a negative value into a positive value and vice-versa.



(ii) **Unary NOT Operator (!):** This operator is used to convert “true” to “false” and vice versa.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int a = 10, b = 1;
    cout << "!(a < b) = " << !(a < b) << endl;
    cout << "(a < b) = " << (a < b);
    return 0;
}
```

Output:

```
!(a < b) = 1
(a < b) = 0
```

(iii) **Increment Operator (++):** This operator is used to increment the value by 1. There are two types of increment operators

1. **Post-increment operator:** Post increment operator is used to increment the variable's value after it has been evaluated for use in the expression.
2. **Pre-increment operator:** Pre increment operator is used to increment the variable's value before it's evaluated in the expression.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int num = 10;
    cout << "Post increment = " << num++;
    // first print 10, then number is increment to 11
    cout << "Pre increment = " << ++num;
    // num was 11, incremented to 12 and print
    return 0;
}
```

Output:

```
Post increment = 10
Pre increment = 12
```

1. Post-decrement operator: Post decrement operator is used to decrement the value of the variable after it has been evaluated for use in the expression.
2. Pre-decrement operator: Pre decrement operator is used to decrement the value of the variable before it's evaluated in the expression.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int num = 10;
    cout << "Post increment = " << num--;
    // first print 10, then number is decrement to 9
    cout << "Pre increment = " << --num;
    // num was 9, decremented to 8 and print
    return 0;
}
```

Output:

Post decrement = 10

Pre decrement = 8

v) Bitwise Complement (~): This operator is used to return the one's complement representation of the input value.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int num1 = 6;
    int num2 = -2;
    // Performing bitwise complement
    cout << num1 << "'s bitwise complement = " << ~num1;
    cout << num2 << "'s bitwise complement = " << ~num2;
    return 0;
}
```

Output:

6's bitwise complement = -7

-2's bitwise complement = 1

Relational Operators

3. Relational Operators:

The Relational operators are used to check the relationship between two operands. This operator is also called a comparison operator because it is used to make a comparison between two operands. The result of these operators is always boolean value. These operators are used in if statements and loops. There are many types of relational operators, which are given below:

i) **Equal to operator (==):** This operator is used to check whether the two operands are equal or not. If they are equal, it returns true(1); otherwise it returns false(0).

Example:

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

int main() {
    int a = 10;
    int b = 20;
    int c = 20;
    cout << (a == b) << endl;
    cout << (b == c);
    // Check two operands are equal or not if they equal return true,
    // else return false
    return 0;
}
```

Output:

0
1

ii) **Not Equal to operator (!=):** This operator is used to check whether the two operands are equal or not. It returns true(1) if the left operand is not equal to the right operand; otherwise, it returns false(0).

Example:

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

int main() {
    int a = 10;
    int b = 20;
    int c = 20;
    cout << (a != b) << endl;
    cout << (b != c);
    // Returns true if the left operand is not equal to the right operand
    return 0;
}
```

Output:

1
0

iii) **Greater than operator (>):** This operator is used to check whether the first operand is greater than the second operand or not. It returns true(1) if the first operand is greater than the second operand and false(0) if not.

Example:

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

int main() {
    int a = 10;
    int b = 20;
    cout << (a > b) << endl;
    cout << (b > a);
    // Returns true if left operand is greater then right operand
    return 0;
}
```

Output:

0
1

iv) **Greater than equal to the operator (>=):** This operator is used to check whether the first operand is greater than or equal to the second operand or not. It returns true(1) if the first operand is greater than or equal to the second operand; otherwise, it returns false(0).

Example:

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

int main() {
    int a = 10;
    int b = 8;
    cout << (a >= b);
    // It returns true because the first operand is greater than the second
    return 0;
}
```

Output:

1

v) **Less than operator (<):** This operator is used to check whether the first operand is less than the second operand or not. It returns true(1) if the first operand is less than the second operand else returns false(0).

Example:

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

int main() {
    int a = 10;
    int b = 15;
    cout << (a < b);
    // It returns true because the first operand is smaller than the second
    return 0;
}
```

Output:

1

vi) **Less than or equal to operator (<=):** This operator is used to check whether the first operand is less than or equal to the second operand or not. It returns true(1) if the first operand is less than or equal to the second operand; else, return false(0).

Example:

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

int main() {
    int a = 10;
    int b = 5;
    cout << (a <= b);
    // It returns false because the first operand is
    //not less than or equal to the second operand.
    return 0;
}
```

Output:

0



Logical Operators

These operators are used to perform logical operations such as OR, AND, and NOT operations. It operates on two boolean values, which return true or false as a result.

There are three types of Logical Operators in C++:

i) Logical AND operator (&&): This operator returns true(1), if both the conditions are true else returns false(0).

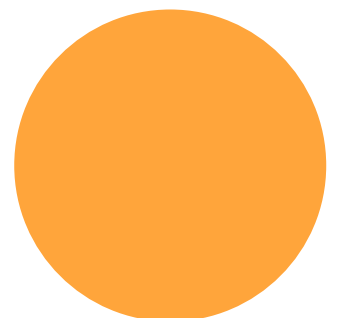
Example:

```
#include<iostream>
using namespace std;
int main() {
    int a = 10;
    int b = 20;
    int c = 30;
    cout << ((b > a) && (c > b)) << endl; // true
    cout << ((b > a) && (c < b)); // false
    return 0;
}
```

Output:

1
0

ii) Logical OR operator (||): This operator returns true(1) if any one of the conditions is true.



```

#include<iostream>
using namespace std;
int main() {
    int a = 10;
    int b = 20;
    int c = 30;
    cout << ((b > a) || (c < b)); // true
    cout << ((b < a) || (c < b)); // false
    return 0;
}
Output:
1
0

```

iii) Logical NOT operator (!): This operator is used to reverse the operand's value. If the operand's value is true, it returns false(0), and if the value of the operand is false, it returns true(1)

```

#include<iostream>
using namespace std;
int main() {
    int a = 10;
    int b = 20;
    cout << (! (a == b)); // true
    cout << (! (b > a)); // false
    return 0;
}
Output:
1
0

```

Bitwise Operators

5. Bitwise Operators:

The Bitwise operators are used to perform bit manipulation on numbers. There are various types of Bit operators that are used in C++.

i) **Bitwise AND operator (&):** It takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1. Mind that the commutative property holds true here.

That is,

$$1 \& 1 = 1$$

$$1 \& 0 = 0$$

Example:

```
#include<iostream>
using namespace std;
int main() {
    int a = 6; // Binary representation of 6 is 0110
    int b = 7; // Binary representation of 7 is 0111
    cout << "a & b = " << (a & b); //0110 & 0111 = 0110 = 6
    return 0;
}
```

Output:

a & b = 6

ii) **Bitwise OR operator (|):** It takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1. Mind that the commutative property holds true here.

That is,

$$1 | 1 = 1$$

$$1 | 0 = 1$$

$$0 | 0 = 0$$

Example:

```
#include<iostream>
using namespace std;
int main() {
    int a = 6; // Binary representation of 6 is 0110
    int b = 7; // Binary representation of 7 is 0111
    cout << "a | b = " << (a | b); //0110 | 0111 = 0111 = 7
    return 0;
}
```

Output:

a | b = 7

iii) **Bitwise NOT operator (~):** It takes one number and inverts all bits of it.

That is,

$$\sim 1 = 0$$

$$\sim 0 = 1$$

Example:

```
#include<iostream>
using namespace std;
int main() {
    // Binary representation of 6 is 0000000000000000000000000000110 (size of int is 32)
    int a = 6;

    // ~6 = 1111111111111111111111111111001 = -7 (-ve nos. are stored in 2's complement)
    cout << "~a = " << (~a);
    return 0;
}
```

Output:

~a = -7

iv) **Bitwise XOR operator (^):** It takes two numbers as operands and does XOR on every bit of two numbers.

The result of XOR is 1 if the two bits are different. Mind that the commutative property holds true here.

That is,

$$1 \wedge 1 = 0$$

$$1 \wedge 0 = 1$$

$$0 \wedge 0 = 0$$

Example:

```
#include<iostream>
using namespace std;
int main() {
    int a = 6; // Binary representation of 6 is 0110
    int b = 7; // Binary representation of 7 is 0111
    cout << "a ^ b = " << (a ^ b); // 0110 ^ 0111 = 0001 = 1
    return 0;
}
```

Output:

a ^ b = 1

v) **Left shift operator (<<):** It takes two numbers, the left shift operator shifts the bits of the first operand, the second operand decides the number of places to shift.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int a = 8; // Binary representation of 8 is 1 0 0 0
    cout << "a << 2 = " << (a << 2);
    // Left shift means appending numbers of 0's to the right.
    // 1 0 0 0 0 0 = 32
    return 0;
}
Output:
a << 2 = 32
```

vi) **Right shift operator (>>):** It takes two numbers; the right shift operator shifts the bits of the first operand, the second operand decides the number of places to shift.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int a = 8; // Binary representation of 8 is 1 0 0 0
    cout << "a >> 2 = " << (a >> 2);
    // Right shift means remove numbers of 0's from right 1 0 = 2
    return 0;
}
Output:
a >> 2 = 2
```



Assignment Operators

6. Assignment Operators:

The Assignment operators are used to assign a value to the variable. In C++, we can use many assignment operators. These are explained below:

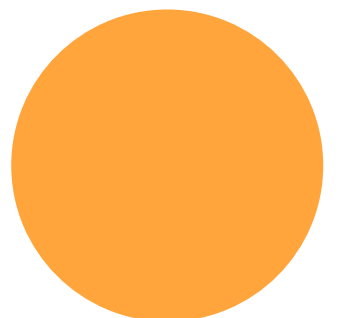
i) **+=**: This assignment operator is used to add the left operand with the right operand and then assign it to a variable on the left.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int num = 10;
    num += 20; // num = num + 20;
    cout << num;
    return 0;
}
```

Output:

30



ii) -=: This assignment operator is used to subtract the left operand with the right operand and then assign it to a variable on the left.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int num = 20;
    num -= 10; // num = num - 10;
    cout << num;
    return 0;
}
```

Output:

10

iii) *=: This assignment operator is used to multiply the left operand with the right operand and then assign it to a variable on the left.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int num = 10;
    num *= 5; // num = num * 5;
    cout << num;
    return 0;
}
```

Output:

50

iv) /=: This assignment operator is used to divide the left operand with the right operand and then assign it to a variable on the left.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int num = 10;
    num /= 2; // num = num / 2;
    cout << num;
    return 0;
}
```

Output:

5

v) %=: This assignment operator is used to modulo the left operand with the right operand and then assign it to a variable on the left.

Example:

```
#include<iostream>
using namespace std;
int main() {
    int num = 19;
    num %= 5; // num = num % 5;
    cout << num;
    return 0;
}
```

Output:

4



Misc Operators

7. Misc Operators:

Apart from the above operators, some other operators are available in C++ to perform some specific tasks. They are explained below:

i) **sizeof operator:** This operator determines a variable's size. sizeof operator can also be used to determine the size of a data type.

Example:

```
#include <iostream>
using namespace std;
int main() {
    int a;
    cout << "Size of int : " << sizeof(int) << "\n";
    cout << "Size of char : " << sizeof(char) << "\n";
    cout << "Size of float : " << sizeof(float) << "\n";
    cout << "Size of double : " << sizeof(double) << "\n";
    cout << "Size of a : " << sizeof(a) << "\n";
    return 0;
}
```

Output:

```
Size of int : 4
Size of char : 1
Size of float : 4
Size of double : 8
Size of a : 4
```


ii) **Comma operator(,):** It is a binary operator that evaluates its first operand and discards the result. It then evaluates the second operand and returns this value.

Example:

```
#include <iostream>
using namespace std;
int main() {
    int x, y;
    y = 100;
    x = (y + 10, 99 + y);
    cout << "With brackets value of x :" << x << endl;
    x = y + 10, 99 + y;
    cout << "Without brackets value of x :" << x;
    return 0;
}
```

Output:

```
With brackets value of x :199
Without brackets value of x :110
```



iii) Conditional Operator(?:) or ternary operators: It is of the form

```
Expression1 ? Expression2 : Expression3
```

Here, Expression1 is the condition to be evaluated. If the condition(Expression1) is True, then we will execute and return the result of Expression2; otherwise, if the condition(Expression1) is false, then we will execute and return the result of Expression3. Since it takes three operands to work, hence they are also called ternary operators.

Example:

```
#include <iostream>
using namespace std;
int main() {
    int a = 1, b;
    b = (a < 10) ? 2 : 5; //As a is less than 10 hence b=2
    cout << "Value of b: " << b << endl;
    return 0;
}
```

Output:

Value of b: 2

iv) Pointer Operator:

a)&: It refers to the address (memory location) in which the operand is stored.

b)*: It is a pointer operator.

Example:

```
#include <iostream>
using namespace std;
int main() {
    int a = 1, * b; //Here b is a pointer operator of int type
    b = & a;
    cout << "Address of variable a: " << b << endl;
    cout << "Address of variable b: " << & b;
    return 0;
}
```

//Here, answers may vary.

Output:

Address of variable a: 0x7ffe85c2db74

Address of variable b: 0x7ffe85c2db78