Operators in C++

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.

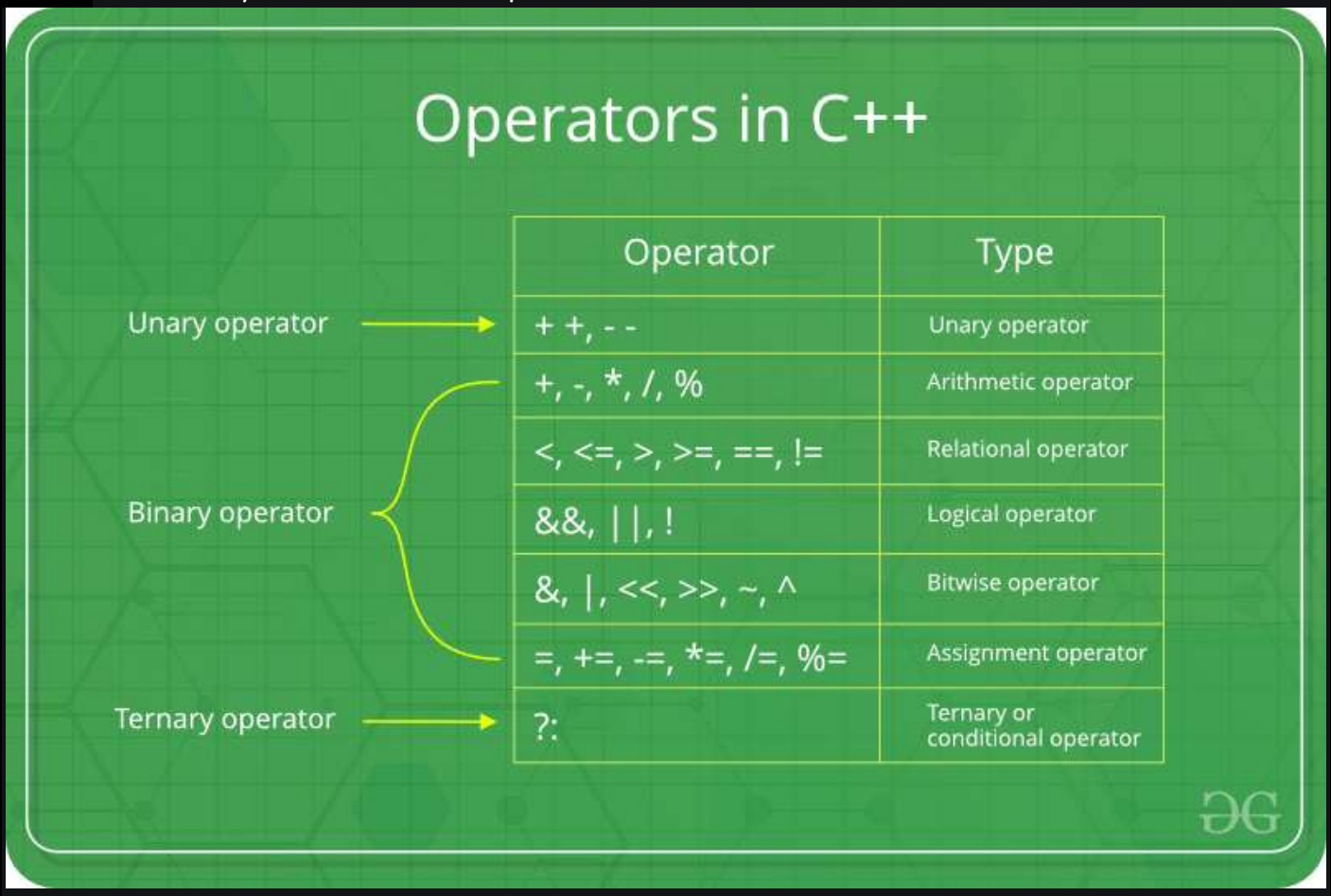
An operator operates the **operands**. For example,

```
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



1) Arithmetic Operators

These operators are used to perform arithmetic or mathematical operations on the operands. For example, '+' is used for addition, '-' is used for subtraction '*' is used for multiplication, etc.

Arithmetic Operators can be classified into 2 Types:

A) Unary Operators: These operators operate or work with a single operand. For example: Increment(++) and Decrement(–) Operators.

Name	Symbol	Description	Example
Increment Operator	++	Increases the integer value of the variable by one	int a = 5; a++; // returns 6
Decrement Operator	—	Decreases the integer value of the variable by one	int a = 5; a–; // returns 4

Example:
the description

Output

```
a++ is 10
++a is 12
b-- is 15
--b is 13
```

Time Complexity: O(1)
Auxiliary Space : O(1)

Note: *++a and* ***a++****, both are increment operators, however, both are slightly different. In* ***++a****, the value of the variable is incremented first and then It is used in the program. In* ***a++****, the value of the variable is assigned first and then It is incremented. Similarly happens for the decrement operator.*

B) Binary Operators: These operators operate or work with two operands. For example: Addition(+), Subtraction(-), etc.

Name	Symbol	Description	Example
Addition	+	Adds two operands	int a = 3, b = 6; int c = a+b; // c = 9
Subtraction	–	Subtracts second operand from the first	int a = 9, b = 6; int c = a-b; // c = 3
Multiplication	*	Multiplies two operands	int a = 3, b = 6;

Name	Symbol	Description	Example
			int c = a*b; // c = 18
Division	/	Divides first operand by the second operand	int a = 12, b = 6; int c = a/b; // c = 2
Modulo Operation	%	Returns the remainder an integer division	int a = 8, b = 6; int c = a%b; // c = 2

Note: *The Modulo operator(%) operator should only be used with integers.*

Example:

- C++

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

int main()
{
    int a = 8, b = 3;

    // Addition operator
    cout << "a + b = " << (a + b) << endl;

    // Subtraction operator
    cout << "a - b = " << (a - b) << endl;

    // Multiplication operator
    cout << "a * b = " << (a * b) << endl;

    // Division operator
    cout << "a / b = " << (a / b) << endl;
```

```
    // Modulo operator
    cout << "a % b = " << (a % b) << endl;


    return 0;
}
```

Output

```
a + b = 11
a - b = 5
a * b = 24
a / b = 2
a % b = 2
```

Time Complexity: O(1)
Auxiliary Space : O(1)

2) Relational Operators

These operators are used for the comparison of the values of two operands. For example, '>' checks if one operand is greater than the other operand or not, etc. The result returns a Boolean value, i.e., **true** or **false.**

Name	Symbol	Description	Example

Is Equal To	==	Checks if both operands are equal	int a = 3, b = 6;
			a==b;
			// returns false

Greater Than	>	Checks if first operand is greater than the second operand	int a = 3, b = 6;
			a>b;
			// returns false

Greater Than or Equal To	>=	Checks if first operand is greater than or equal to the second operand	int a = 3, b = 6; a>=b;

Name	Symbol	Description	Example
			// returns false

Less Than	<	Checks if first operand is lesser than the second operand	int a = 3, b = 6;
			a<b;
			// returns true

Less Than or Equal To	<=	Checks if first operand is lesser than or equal to the second operand	int a = 3, b = 6;
			a<=b;
			// returns true

Not Equal To	!=	Checks if both operands are not equal	int a = 3, b = 6;
			a!=b;
			// returns true

Example:

- C++

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

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

    // Equal to operator
    cout << "a == b is " << (a == b) << endl;

    // Greater than operator
```

```
    cout << "a > b is " << (a > b) << endl;


    // Greater than or Equal to operator
    cout << "a >= b is " << (a >= b) << endl;


    //  Lesser than operator
    cout << "a < b is " << (a < b) << endl;


    // Lesser than or Equal to operator
    cout << "a <= b is " << (a <= b) << endl;


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


    return 0;
}
```

Output

```
a == b is 0
a > b is 1
a >= b is 1
a < b is 0
a <= b is 0
a != b is 1
```

Time Complexity: O(1)
Auxiliary Space : O(1)

Here, **0** denotes **false** and **1** denotes **true**. To read more about this, please refer to the article – Relational Operators.

3) 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

Example:

- C++

```cpp
// 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
```

Time Complexity: O(1)
Auxiliary Space : O(1)

Here, **0** denotes **false** and **1** denotes **true**. To read more about this, please refer to the article – Logical Operators.

4) Bitwise Operators

These operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing.

Name	Symbol	Description	Example

Binary AND	&	Copies a bit to the evaluated result if it exists in both operands	int a = 2, b = 3;
			(a & b);
			//returns 2

Binary OR	|	Copies a bit to the evaluated result if it exists in any of the operand	int a = 2, b = 3;
			(a | b);
			//returns 3

Binary XOR	^	Copies the bit to the evaluated result if it is present in either of the operands but not both	int a = 2, b = 3;
			(a ^ b);
			//returns 1

Left Shift	<<	Shifts the value to left by the number of bits specified by the right operand.	int a = 2, b = 3;
			(a << 1);
			//returns 4

Name	Symbol	Description	Example
Right Shift	>>	Shifts the value to right by the number of bits specified by the right operand.	int a = 2, b = 3; (a >> 1); //returns 1
One's Complement	~	Changes binary digits 1 to 0 and 0 to 1	int b = 3; (~b); //returns -4

Note: *Only* ***char*** *and* ***int*** *data types can be used with Bitwise Operators.*

Example:

- C++

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

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

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

    // Binary OR operator
    cout << "a | b is " << (a | b) << endl;

    // Binary XOR operator
    cout << "a ^ b is " << (a ^ b) << endl;

    // Left Shift operator
    cout << "a<<1 is " << (a << 1) << endl;
```

```
    // Right Shift operator
    cout << "a>>1 is " << (a >> 1) << endl;

    // One's Complement operator
    cout << "~(a) is " << ~(a) << endl;

    return 0;
}
```

Output

```
a & b is 4
a | b is 6
a ^ b is 2
a<<1 is 12
a>>1 is 3
~(a) is -7
```

Time Complexity: O(1)
Auxiliary Space : O(1)
To read more about this, please refer to the article – Bitwise Operators.

5) Assignment Operators

These operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and the right side operand of the assignment operator is a value. The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.

Namemultiply	Symbol	Description	Example
Assignment Operator	=	Assigns the value on the right to the variable on the left	int a = 2; // a = 2
Add and Assignment Operator	+=	First adds the current value of the variable on left to the value on the right and then assigns the result to the variable on the left	int a = 2, b = 4; a+=b; // a = 6

Namemultiply	Symbol	Description	Example
Subtract and Assignment Operator	-=	First subtracts the value on the right from the current value of the variable on left and then assign the result to the variable on the left	int a = 2, b = 4; a-=b; // a = -2
Multiply and Assignment Operator	*=	First multiplies the current value of the variable on left to the value on the right and then assign the result to the variable on the left	int a = 2, b = 4; a*=b; // a = 8
Divide and Assignment Operator	/=	First divides the current value of the variable on left by the value on the right and then assign the result to the variable on the left	int a = 4, b = 2; a /=b; // a = 2

Example:

- C++

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

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

    // Assignment Operator
    cout << "a = " << a << endl;

    //  Add and Assignment Operator
    cout << "a += b is " << (a += b) << endl;

    // Subtract and Assignment Operator
    cout << "a -= b is " << (a -= b) << endl;
```

```
    //  Multiply and Assignment Operator
    cout << "a *= b is " << (a *= b) << endl;

    //  Divide and Assignment Operator
    cout << "a /= b is " << (a /= b) << endl;

    return 0;
}
```

Output

```
a = 6
a += b is 10
a -= b is 6
a *= b is 24
a /= b is 6
```

Time Complexity: O(1)
Auxiliary Space : O(1)

6) Ternary or Conditional Operators(?:)

This operator returns the value based on the condition.

```
Expression1? Expression2: Expression3
```

The ternary operator **?** determines the answer on the basis of the evaluation of **Expression1**. If it is **true**, then **Expression2** gets evaluated and is used as the answer for the expression. If **Expression1** is **false**, then **Expression3** gets evaluated and is used as the answer for the expression.
This operator takes **three operands**, therefore it is known as a Ternary Operator.
Example:

- C++

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

int main()
{
```

```
    int a = 3, b = 4;


    // Conditional Operator
    int result = (a < b) ? b : a;
    cout << "The greatest number is " << result << endl;


    return 0;
}
```

Output

```
The greatest number is 4
```

Time Complexity: O(1)
Auxiliary Space : O(1)

7) There are some other common operators available in C++ besides the operators discussed above. Following is a list of these operators discussed in detail:

A) sizeof Operator: This unary operator is used to compute the size of its operand or variable.

```
sizeof(char); // returns 1
```

B) Comma Operator(,): This binary operator (represented by the token) is used to evaluate its first operand and discards the result, it then evaluates the second operand and returns this value (and type). It is used to combine various expressions together.

```
int a = 6;
int b = (a+1, a-2, a+5); // b = 11
```

C) -> Operator: This operator is used to access the variables of classes or structures.

```
cout<<emp->first_name;
```

D) Cast Operator: This unary operator is used to convert one data type into another.

```
float a = 11.567;
int c = (int) a; // returns 11
```

E) Dot Operator(.): This operator is used to access members of structure variables or class objects in C++.

```
cout<<emp.first_name;
```

F) & Operator: This is a pointer operator and is used to represent the memory address of an operand.

G) * Operator: This is an Indirection Operator

- C++

```
// CPP Program to demonstrate the & and * Operators
```

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


int main()
{
    int a = 6;
    int* b;
    int c;
    //  & Operator
    b = &a;


    // * Operator
    c = *b;
    cout << " a = " << a << endl;
    cout << " b = " << b << endl;
    cout << " c = " << c << endl;


    return 0;
}
```

Output

```
a = 6
b = 0x7ffe8e8681bc
c = 6
```

H) << Operator: It is called the insertion operator. It is used with cout to print the output.
I) >> Operator: It is called the extraction operator. It is used with cin to get the input.

```
int a;
cin>>a;
cout<<a;
```

Time Complexity: O(1)
Auxiliary Space : O(1)

Operator Precedence Chart

Precedence	Operator	Description	Associativity
1.	()	Parentheses (function call)	left-to-right
	[]	Brackets (array subscript)	
	.	Member selection via object name	
	->	Member selection via a pointer	
	++/–	Postfix increment/decrement	
2.	++/–	Prefix increment/decrement	right-to-left
	+/-	Unary plus/minus	
	!~	Logical negation/bitwise complement	
	(type)	Cast (convert value to temporary value of type)	
	*	Dereference	
	&	Address (of operand)	
	sizeof	Determine size in bytes on this implementation	
3.	*,/,%	Multiplication/division/modulus	left-to-right
4.	+/-	Addition/subtraction	left-to-right
5.	<< , >>	Bitwise shift left, Bitwise shift right	left-to-right

Precedence	Operator	Description	Associativity
6.	< , <=	Relational less than/less than or equal to	left-to-right
	> , >=	Relational greater than/greater than or equal to	left-to-right
7.	== , !=	Relational is equal to/is not equal to	left-to-right
8.	&	Bitwise AND	left-to-right
9.	^	Bitwise exclusive OR	left-to-right
10.	|	Bitwise inclusive OR	left-to-right
11.	&&	Logical AND	left-to-right
12.	||	Logical OR	left-to-right
13.	?:	Ternary conditional	right-to-left
14.	=	Assignment	right-to-left
	+= , -=	Addition/subtraction assignment	
	*= , /=	Multiplication/division assignment	
	%= , &=	Modulus/bitwise AND assignment	
	^= , |=	Bitwise exclusive/inclusive OR assignment	
	<>=	Bitwise shift left/right assignment	

Precedence	Operator	Description	Associativity
15.	,	expression separator	left-to-right