# **WHAT ARE OPERATORS?**

The operators are the special type of functions that takes one or more parameters and gives new result. It is a symbol that tells the compiler to perform the mathematical and logical manipulations. The programming language like C or C++ is incomplete without the use of operators.

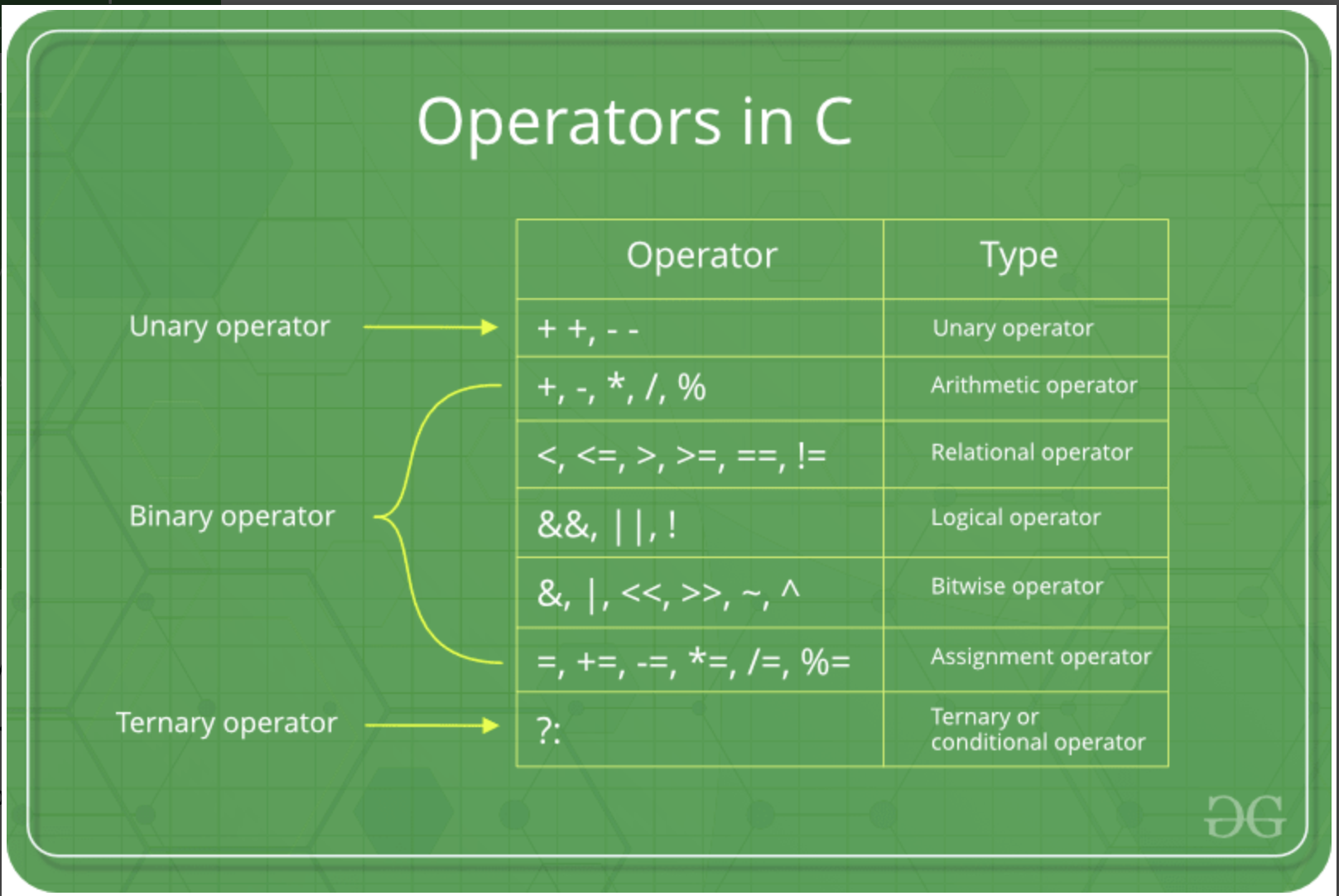
**Operators** are the foundation of any programming language. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands. For example, ‘+’ is an operator used for addition, as shown below:

c = a + b

Here, ‘+’ is the operator known as the addition operator and ‘a’ and ‘b’ are operands. The addition operator tells the compiler to add both of the operands ‘a’ and ‘b’.

***C/C++ has many built-in operators and can be classified into 6 types:***

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Other Operators



**1. Arithmetic Operators:**

These operators are used to perform arithmetic/mathematical operations on operands. Examples: (+, -, \*, /, %,++,–). Arithmetic operators are of two types:

**a) Unary Operators**: Operators that operate or work with a single operand are unary operators. For example: Increment(++) and Decrement(–) Operators

int val = 5;

++val; // 6

**b) Binary Operators**: Operators that operate or work with two operands are binary operators. For example: Addition(+), Subtraction(-), multiplication(\*), Division(/) operators

int a = 7;

int b = 2;

cout<<a+b; // 9

**2. Relational Operators:**

These are used for the comparison of the values of two operands. For example, checking if one operand is equal to the other operand or not, whether an operand is greater than the other operand or not, etc. Some of the relational operators are (==, >= , <= ).

int a = 3;

int b = 5;

a < b; // operator to check if a is smaller than b

**3. Logical** Operators**:**

Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a Boolean value either **true** or **false**.

For example, the **logical AND** represented as **‘&&’ operator in C or C++** returns true when both the conditions under consideration are satisfied. Otherwise, it returns false. Therefore, a && b returns true when both a and b are true (i.e. non-zero).

(4 != 5) && (4 < 5); // true

**4. Bitwise Operators:**

The Bitwise 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. For example, the **bitwise AND** represented as **& operator in C or C++** 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.

int a = 5, b = 9; // a = 5(00000101), b = 9(00001001)

cout << (a ^ b); uu // 00001100

cout <<(~a); // 11111010

**5. Assignment Operators:**

Assignment 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.

Different types of assignment operators are shown below:

**a. “=”:** This is the simplest assignment operator. This operator is used to assign the value on the right to the variable on the left.

For example:

a = 10;

b = 20;

ch = 'y';

**b. “+=”**: This operator is combination of ‘+’ and ‘=’ operators. This 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.

For example:

(a += b) can be written as (a = a + b)

If initially value stored in a is 5. Then (a += 6) = 11.

**c. “-=”**: This operator is a combination of ‘-‘ and ‘=’ operators. This operator first subtracts the value on the right from the current value of the variable on left and then assigns the result to the variable on the left.

For example:

(a -= b) can be written as (a = a - b)

If initially value stored in a is 8. Then (a -= 6) = 2.

**d. “\*=”**: This operator is a combination of ‘\*’ and ‘=’ operators. This operator first multiplies 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.

For example:

(a \*= b) can be written as (a = a \* b)

If initially, the value stored in a is 5. Then (a \*= 6) = 30.

**e. “/=”**: This operator is a combination of ‘/’ and ‘=’ operators. This operator first divides the current value of the variable on left by the value on the right and then assigns the result to the variable on the left.

For example:

(a /= b) can be written as (a = a / b)

If initially, the value stored in a is 6. Then (a /= 2) = 3.

**6. Other Operators**:

Apart from the above operators, there are some other operators available in C or C++ used to perform some specific tasks. Some of them are discussed here:

**a.** [**sizeof operator**](https://www.geeksforgeeks.org/sizeof-operator-c/):

* sizeof is much used in the C/C++ programming language.
* It is a compile-time unary operator which can be used to compute the size of its operand.
* The result of sizeof is of the unsigned integral type which is usually denoted by size\_t.
* Basically, the sizeof the operator is used to compute the size of the variable.

**b.** [**Comma Operator**](https://www.geeksforgeeks.org/comna-in-c-and-c/):

* The comma operator (represented by the token) is a binary operator that evaluates its first operand and discards the result, it then evaluates the second operand and returns this value (and type).
* The comma operator has the lowest precedence of any C operator.
* Comma acts as both operator and separator.

**c.** [**Conditional Operator**](https://www.geeksforgeeks.org/cc-ternary-operator-some-interesting-observations/):

* The conditional operator 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.
* We may replace the use of if..else statements with conditional operators.

**d.** [**dot (.)**](https://www.geeksforgeeks.org/dot-operator-in-c-c/) **and** [**arrow (->)**](https://www.geeksforgeeks.org/arrow-operator-in-c-c-with-examples/) **Operators:**

* Member operators are used to reference individual members of classes, structures, and unions.
* The dot operator is applied to the actual object.
* The arrow operator is used with a pointer to an object.

**e.** [**Cast Operator:**](https://www.geeksforgeeks.org/typecasting-in-c/)

* Casting operators convert one data type to another. For example, int(2.2000) would return 2.
* A cast is a special operator that forces one data type to be converted into another.
* The most general cast supported by most of the C++ compilers is as follows −  **[ (type) expression ]**.

**f.** [**&,\* Operator:**](https://www.geeksforgeeks.org/bitwise-operators-in-c-cpp/)

* Pointer operator & returns the address of a variable. For example &a; will give the actual address of the variable.
* Pointer operator \* is a pointer to a variable. For example \*var; will pointer to a variable var.

**EXERCISE**

1. int result = 4 + 5 \* 6 + 2;

cout << result;

1. int a = 5+7%2;

cout << a;

1. int x = 10,y;

y = x++;

cout << y;

1. int x = 10,y;

y = x++;

cout << x;

1. int x = 10;

x++;

cout << x;

1. int x = 10,y;

y = ++x;

cout << y;

1. int x = 10;

cout << ++x;

1. int x = 10;

cout << x++;

1. int x = 10,y;

y = x+x++;

cout << y;

1. int x = 10,y;

y = ++x + x++ + x;

cout << y;

1. int x = 10,y;

y = x++ + x + ++x;

cout << y;