Operators in C programming language

Operators in C programming language are symbols that perform specific operations on variables and expressions. There are different types of operators in C, each with its own purpose.

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and decrement operators

Arithmetic operators

Arithmetic operators are used to perform mathematical operations on variables, expressions and constants in C. There are five basic arithmetic operators in C.

- Addition

  The addition operator (+) adds two operands together.
  For example, `5 + 2 = 7`

- Subtraction

  The subtraction operator (-) subtracts the second operand from the first.
  For example, `5 - 2 = 3`

- Multiplication

  The multiplication operator (*) multiplies two operands together.
  For example, `5 * 2 = 10`

- Division

  The division operator (/) divides the first operand by the second.
  For example, `4 / 2 = 2`

- Modulus

  The modulus operator (%) returns the remainder of an integer division.
  For example, `5 % 2 = 1`

Here are some examples of arithmetic operators in C programming language.

```
int a = 5;
int b = 2;
int c = 4;

// Addition
```

```
int d = a + b; // d = 7

// Subtraction
int e = a - b; // e = 3

// Multiplication
int f = a * b; // f = 10

// Modulus
int i = a % b; // i = 1


// Division
int g = c / b; // g = 2
int h = a / b; // h = 2
```

If one integer is divided by another integer, the answer will be an integer value. This is because integer division in C is truncation, which means that the decimal part of the result is discarded. For example, the expression `5 / 2` evaluates to `2`, because the decimal part of the result, `0.5` is discarded.

If you want to get the decimal part of the result, you can convert one of the integers to `float` or `double` data type using a cast operator.

Cast operator
The cast operator in C temporarily transforms a variable into a different data type.

Here is an example of how to use integer division in C with cast operator.

```
int a = 5;
int b = 2;

int c = a / b; // c = 2
float d = float(a) / b; // d = 2.5
```

Relational operators

Relational operators in C are used to compare two values and determine if they are equal, not equal, greater than, less than, greater than or equal to, or less than or equal to.

There are six relational operators in C

Symbol	*Operator*
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>	Greater than or equal to

`<=`	Less than or equal to

The relational operators are used in expressions to compare two values. The result of a relational expression is a Boolean value, which can be either `true` or `false`.

For example, the following expression compares the values of the variables `a` and `b`.

```
a == b
```

If the values of `a` and `b` are equal, the expression evaluates to `true`. Otherwise, the expression evaluates to `false`.

Here are some examples.

```
int x = 8;
int y = 2;
```

`x + y == 10` evaluates to true

`x - y == 5` evaluates to false

`x * y > 10` evaluates to true

`x < y` evaluates to false

Logical operators

Logical operators in C programming language are used to combine two or more conditions. A condition is an expression that evaluates to a Boolean value, which is either `true` or `false`.

There are 3 major logical operators in the C language.

Symbol	*Operator*
`&&`	Logical AND
`||`	Logical OR
`!`	Logical NOT

Here are some examples how logical operators yield true or false.

- Logical AND (&&)

The logical AND operator yields true if and only if both of its operands are true.
For example, the expression `5 < 10 && 10 > 5` yields true because both operands are true.

- Logical OR (||)

The logical OR operator yields true if at least one of its operands is true.
For example, the expression `5 < 10 || 10 > 5` yields true because at least one operand is true.

- Logical NOT (!)

The logical NOT operator yields the opposite of its operand.
For example, the expression `!(5 < 10)` yields false because the operand is true.

Assignment operators

Assignment operators in C programming language are used to assign the value of an expression to a variable.

The most commonly used assignment operator is the `=` operator. For example, the following code assigns the value of 5 to the variable `a`

```
int a = 5;
```

The assignment operator can also be used with arithmetic operators.
For example, the following code increases the value of `a` by `2` and then assign new value (`7`) to the variable `a`

```
int a = 5;
a += 2;  //this is same as a = a + 2
```

The following table summarizes the most commonly used assignment operators in C.

Symbol	*Operator*
=	Simple assignment
+=	Increment assignment
-=	Decrement assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment

Increment and decrement operators

Increment and decrement operators in C programming language are used to increase or decrease the value of a variable by 1. The increment operator is `++` and the decrement operator is `--`.

The increment operator can be used in two ways: as a prefix operator or as a postfix operator.

When used as a prefix operator, the increment operator is placed before the variable, and the value of the variable is incremented before the expression is evaluated. For example, the following code increments the value of the variable `a` by 1 and then prints the value of `a`:

```
int a = 5;
printf("The value of a is: %d\n", ++a); // The value of a is: 6
```

When used as a postfix operator, the increment operator is placed after the variable, and the value of the variable is incremented after the expression is evaluated. For example, the following code prints the value of the variable `a`, increments the value of `a` by 1, and then prints the value of `a` again:

```
int a = 5;

printf("The value of a is: %d\n", a++); // The value of a is: 5
printf("The value of a is: %d\n", a);   // The value of a is: 6
```

The decrement operator works in a similar way to the increment operator. When used as a prefix operator, the decrement operator decrements the value of the variable before the expression is evaluated. When used as a postfix operator, the decrement operator decrements the value of the variable after the expression is evaluated.

Precedence of operators

Precedence of operators in C determines which operator is executed first if there is more than one operator in an expression. In C, operators with higher precedence are evaluated first.

Order	*Operator/s*	*Associativity*
1	( )	Left to right
2	!	Right to left
3	* / %	Left to right
4	+ -	Left to right
5	< <= > >=	Left to right
6	== !=	Left to right
7	=	Right to left

Arithmetic operators have higher precedence than relational operators, and relational operators have higher precedence than logical operators. This means that the expression `a + b < c` will be evaluated as `(a + b) < c` because the addition operator has higher precedence than the relational operator.

If there are operators with the same precedence, the associativity of the operators determines the order of evaluation. Associativity specifies how operators of equal precedence are grouped together. In C, the associativity of most operators is left-to-right. For example, the expression `a + b - c` is evaluated as `(a + b) - c`