**5.Operators in C**

C language supports a rich set of built-in operators. An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation. Operators are used in programs to manipulate data and variables.

 Thus [operators in C](https://www.javatpoint.com/c-operators) is a special symbol used to perform the functions. The data items on which the operators are applied are known as operands. Operators are applied between the operands. Depending on the number of operands, operators are classified as follows:

**1.Unary Operator**

A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)\*.

# DIFFERENCE****BETWEEN PRE/POST INCREMENT & DECREMENT OPERATORS IN C:****

|  |  |
| --- | --- |
| **Operator** |  **Operator/Description** |
| Pre increment operator (++i) | value of i is incremented before assigning it to the variable i |
| Post increment operator (i++) | value of i is incremented after assigning it to the variable i |
| Pre decrement operator (–i) | value of i is decremented before assigning it to the variable i |
| Post decrement operator (i–) | value of i is decremented after assigning it to variable i |

**Binary Operator**

The binary operator is an operator applied between two operands. The following is the list of the binary operators:

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment Operator
* Conditional Operator
* Special Operator

**1. Arithmetic Operators**

It includes basic arithmetic operations like addition, subtraction, multiplication, division, modulus operations, increment, and decrement.

The Arithmetic Operators in C include:

1. **+ (Addition)** – This operator is used to add two operands.
2. **– (Subtraction)** – Subtract two operands.
3. **\* (Multiplication)** – Multiply two operands.
4. **/ (Division)** – Divide two operands and gives the quotient as the answer.
5. **% (Modulus operation)** – Find the remains of two integers and gives the remainder after the division.
6. **++ (Increment)** – Used to increment an operand.
7. **— (Decrement)** – Used to decrement an operand.

**2. Relational Operators**

It is used to compare two numbers by checking whether they are equal or not, less than, less than or equal to, greater than, greater than or equal to.

1. **==** **(Equal to)**– This operator is used to check if both operands are equal.
2. **!=**  **(Not equal to)**– Can check if both operands are not equal.
3. **>** **(Greater than)**– Can check if the first operand is greater than the second.
4. **< (Less than)-**Can check if the first operand is lesser than the second.
5. **>=**  **(Greater than equal to)**– Check if the first operand is greater than or equal to the second.
6. **<= (Less than equal to)**– Check if the first operand is lesser than or equal to the second

**3. Logical Operators**

It refers to the boolean values which can be expressed as:

* Binary logical operations, which involves two variables: AND and OR
* Unary logical operation: NOT

Logical Operators in C Includes –

1. **&& (AND)** – It is used to check if both the operands are true.
2. **|| (OR)** – These operators are used to check if at least one of the operand is true.
3. **! (NOT)** – Used to check if the operand is false

If the logical statement is satisfied (it is true), then the program will return the value 1, otherwise, if the relational statement is not satisfied (it is false), the program will return the value 0.

**4. Assignment Operators**

It is used to assign a particular value to a variable.

1. **=  (Assignment)-** Used to assign a value from right side operand to left side operand.
2. **+= (Addition Assignment)-**To store the sum of both the operands to the left side operand.
3. **-= (Subtraction Assignment) –**To store the difference of both the operands to the left side operand.
4. **\*=** **(Multiplication Assignment)** – To store the product of both the operands to the left side operand.
5. **/= (Division Assignment)** – To store the division of both the operands to the left side operand.
6. **%= (Remainder Assignment)** – To store the remainder of both the operands to the left side operand.

**5. Bitwise Operators**

It is based on the principle of performing operations bit by bit which is based on boolean algebra. It increases the processing speed and hence the efficiency of the program.

The Bitwise [Operators](https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B) in C Includes –

1. **& (Bitwise AND)** – Converts the value of both the operands into binary form and performs AND operation bit by bit.
2. **| (Bitwise OR) –**Converts the value of both the operands into binary form and performs OR operation bit by bit.
3. **^ (Bitwise exclusive OR)** – Converts the value of both the operands into binary form and performs EXCLUSIVE OR operation bit by bit.
4. **~ (One’s complement operator)**: Converts the operand into its complementary form.
5. **<< – Left shift**
6. **>> – Right shift**

**6.Conditional operator**

The conditional operators in C language are known by two more names

1. **Ternary Operator**
2. **? : Operator**

It is actually the if condition that we use in C language decision making, but using conditional operator, we turn the if condition statement into a short and simple operator.

The syntax of a conditional operator is :

expression 1 ? expression 2: expression 3

**Explanation:**

* The question mark **"?"** in the syntax represents the **if** part.
* The first expression (expression 1) generally returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)
* If (expression 1) returns true then the expression on the left side of **" : "** i.e (expression 2) is executed.
* If (expression 1) returns false then the expression on the right side of **" : "** i.e (expression 3) is executed.

**7.Special operator**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| sizeof | Returns the size of an variable | **sizeof(x)** return size of the variable x  |
|  & | Returns the address of an variable | **&x ;** return address of the Variable **x** |
|  \* | Pointer to a variable | **\*x ;** will be pointer to a variable **x** |

**sizeof** – It returns the memory occupied by the particular data type of the operand

**& (Pointer)** – It refers to the address (memory location) in which the operand is stored.

**\*** **(Pointer)** –  It is a pointer operator