Operators in C

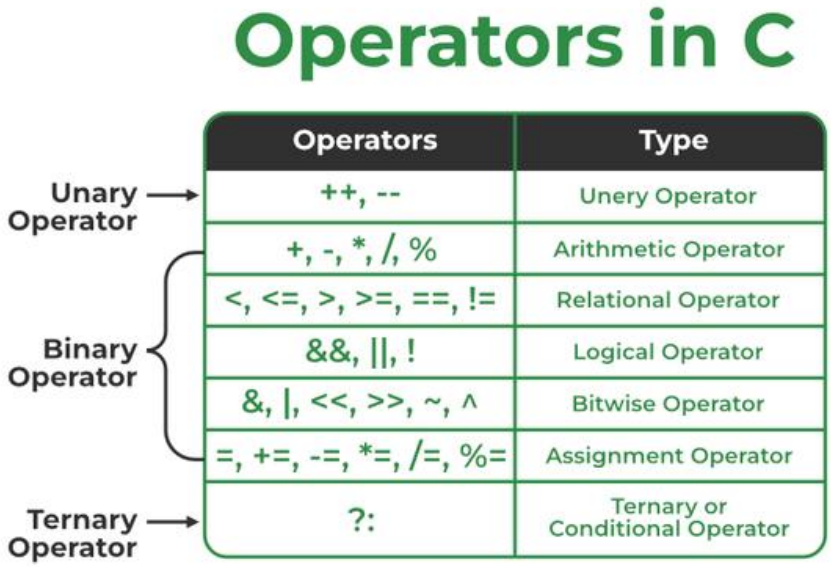
C Operators are symbols that represent operations to be performed on one or more operands. C provides a wide range of operators, which can be classified into different categories based on their functionality. Operators are used for performing operations on variables and values.

Operators can be defined as the symbols that help us to perform specific mathematical, relational, bitwise, conditional, or 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'. The functionality of the C programming language is incomplete without the use of operators.



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
```

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;

a+b; // 9

Ternary Operators:

when we use an operator on three variables or operands, it is known as a Ternary Operator. We can represent it using **? :** . Thus, it is also known as a conditional operator.

Types of Operators in C

Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Example 1: Arithmetic Operators

```
// Working of arithmetic operators
#include <stdio.h>
int main()
{
   int a = 9,b = 4, c;
   c = a+b;
   printf("a+b = %d \n",c);
```

```
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);
    return 0;
}
```

Increment and Decrement Operators

C programming has two operators increment `++` and decrement `--` to change the value of an operand (constant or variable) by 1.

Increment `++` increases the value by 1 whereas decrement `--` decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

Example 2: Increment and Decrement Operators

```
// Working of increment and decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;
    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);
    return 0;
}
```

C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Example 3: Assignment Operators

```
// Working of assignment operators
#include <stdio.h>
int main()
{
   int a = 5, c;
   c = a;      // c is 5
   printf("c = %d\n", c);
   c += a;     // c is 10
   printf("c = %d\n", c);
   c -= a;     // c is 5
   printf("c = %d\n", c);
   c *= a;     // c is 25
   printf("c = %d\n", c);
   c /= a;     // c is 5
   printf("c = %d\n", c);
   c %= a;     // c = 0
```

```
    printf("c = %d\n", c);

    return 0;
}
```

C Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	`5 == 3` is evaluated to 0
>	Greater than	`5 > 3` is evaluated to 1
<	Less than	`5 < 3` is evaluated to 0
!=	Not equal to	`5 != 3` is evaluated to 1
>=	Greater than or equal to	`5 >= 3` is evaluated to 1
<=	Less than or equal to	`5 <= 3` is evaluated to 0

Example 4: Relational Operators

```
// Working of relational operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;
    printf("%d == %d is %d \n", a, b, a == b);
    printf("%d == %d is %d \n", a, c, a == c);
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
```

```
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);
    return 0;
}
```

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

Example 5: Logical Operators

```
// Working of logical operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;
    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);
    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) is %d \n", result);
    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);
```

```
    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);
    result = !(a != b);
    printf("!(a != b) is %d \n", result);
    result = !(a == b);
    printf("!(a == b) is %d \n", result);
    return 0;
}
```

C Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Bitwise AND Operator &

The output of bitwise AND is **1** if the corresponding bits of two operands is **1**. If either bit of an operand is **0**, the result of corresponding bit is evaluated to **0**.

In C Programming, the bitwise AND operator is denoted by &.

Let us suppose the bitwise AND operation of two integers **12** and **25**.

Example

```
#include <stdio.h>
int main() {
    int a = 12, b = 25;
```

```
    printf("Output = %d", a & b);
    return 0;
}
```

Bitwise OR Operator |

The output of bitwise OR is **1** if at least one corresponding bit of two operands is **1**. In C Programming, bitwise OR operator is denoted by |.

Example

```
#include <stdio.h>
int main() {
    int a = 12, b = 25;
    printf("Output = %d", a | b);
    return 0;
}
```

Bitwise XOR (exclusive OR) Operator ^

The result of bitwise XOR operator is **1** if the corresponding bits of two operands are opposite. It is denoted by ^.

Example

```
#include <stdio.h>
int main() {
    int a = 12, b = 25;
    printf("Output = %d", a ^ b);
    return 0;
}
```

Bitwise Complement Operator ~

Bitwise complement operator is a unary operator (works on only one operand). It changes **1** to **0** and **0** to **1**. It is denoted by ~.

Example

```
#include <stdio.h>
int main() {
    printf("Output = %d\n", ~35);
    printf("Output = %d\n", ~-12);
    return 0;
}
```

Shift Operators in C programming

There are two shift operators in C programming:

- Right shift operator
- Left shift operator.

Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by `>>`.

Left Shift Operator

Left shift operator shifts all bits towards left by a certain number of specified bits. The bit positions that have been vacated by the left shift operator are filled with **0**. The symbol of the left shift operator is `<<`.

Example #5: Shift Operators

```
#include <stdio.h>
int main() {
   int num=212, i;
   for (i = 0; i <= 2; ++i) {
      printf("Right shift by %d: %d\n", i, num >> i);
   }
   printf("\n");
   for (i = 0; i <= 2; ++i) {
      printf("Left shift by %d: %d\n", i, num << i);
   }
   return 0;
}
```

Conditional or Ternary Operator (?:)

The **conditional operator in C** is kind of similar to the if-else statement as it follows the same algorithm as of if-else statement but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible. It is also known as the **ternary operator in C** as it operates on three operands.

Syntax

The conditional operator can be in the form

variable = Expression1 **?** *Expression2* **:** *Expression3;*

Or the syntax can also be in this form

variable = (condition) **?** *Expression2* **:** *Expression3*;

Since the Conditional Operator '?:' takes three operands to work, hence they are also called **ternary operators**.

```
// C program to find largest among two

// numbers using ternary operator

#include <stdio.h>

int main()

{

    int m = 5, n = 4;

    (m > n) ? printf("m is greater than n that is %d > %d",

                    m, n)

    : printf("n is greater than m that is %d > %d",

                        n, m);

    return 0;

}
```

Special Operators

Comma Operator

Comma operators are used to link related expressions together. For example:

Int a,c=5,d;

The sizeof operator

The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

Example: sizeof Operator

```
#include <stdio.h>
int main()
{
   int a;
   float b;
   double c;
   char d;
   printf("Size of int=%lu bytes\n",sizeof(a));
   printf("Size of float=%lu bytes\n",sizeof(b));
   printf("Size of double=%lu bytes\n",sizeof(c));
   printf("Size of char=%lu byte\n",sizeof(d));

   return 0;
}
```

C Operators With Example

```
// C Program to Demonstrate the working concept of
// Operators
#include <stdio.h>

int main()
{

        int a = 10, b = 5;
        // Arithmetic operators
        printf("Following are the Arithmetic operators in C\n");
        printf("The value of a + b is %d\n", a + b);
        printf("The value of a - b is %d\n", a - b);

        printf("The value of a * b is %d\n", a * b);
        printf("The value of a / b is %d\n", a / b);
        printf("The value of a % b is %d\n", a % b);
        printf("The value of a++ is %d\n", a++);

        printf("The value of a-- is %d\n", a--);
        printf("The value of ++a is %d\n", ++a);


        printf("The value of --a is %d\n", --a);
        printf(
                "\nFollowing are the comparison operators in C\n");
        printf("The value of a == b is %d\n", (a == b));
```

```
        printf("The value of a != b is %d\n", (a != b));
        printf("The value of a >= b is %d\n", (a >= b));
        printf("The value of a <= b is %d\n", (a <= b));
        printf("The value of a > b is %d\n", (a > b));
        printf("The value of a < b is %d\n", (a < b));
        printf("\nFollowing are the logical operators in C\n");
        printf("The value of this logical and operator ((a==b) "
               "&& (a<b)) is:%d\n",
               ((a == b) && (a < b)));
        printf("The value of this logical or operator ((a==b) "
               "|| (a<b)) is:%d\n",
               ((a == b) || (a < b)));
        printf("The value of this logical not operator "
               "(!(a==b)) is:%d\n",
               (!(a == b)));

        return 0;
}
```