Operators

In C programming, operators are used to perform operations on variables and values. Below are the various types of operators in C:

1. **Arithmetic Operators**

These operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.

- **Addition (+)**: Adds two operands.
  int a = 5, b = 3;
  int result = a + b; // result = 8

- **Subtraction (-)**: Subtracts the second operand from the first.
  int a = 5, b = 3;
  int result = a - b; // result = 2

- **Multiplication (*)**: Multiplies two operands.
  int a = 5, b = 3;
  int result = a * b; // result = 15

- **Division (/)**: Divides the first operand by the second.
  int a = 6, b = 3;
  int result = a / b; // result = 2

- **Modulus (%)**: Returns the remainder when the first operand is divided by the second.
  int a = 7, b = 3;
  int result = a % b; // result = 1

2. **Relational Operators**

These operators are used to compare two values and return either true (1) or false (0).

- **Equal to (==)**: Checks if two operands are equal.
  int a = 5, b = 5;
  int result = (a == b); // result = 1 (true)

- **Not equal to (!=)**: Checks if two operands are not equal.
  int a = 5, b = 3;
  int result = (a != b); // result = 1 (true)

- **Greater than (>)**: Checks if the first operand is greater than the second.
  int a = 5, b = 3;
  int result = (a > b); // result = 1 (true)

- **Less than (<)**: Checks if the first operand is less than the second.
  int a = 5, b = 3;
  int result = (a < b); // result = 0 (false)

- **Greater than or equal to (>=)**: Checks if the first operand is greater than or equal to the second.
  int a = 5, b = 5;
  int result = (a >= b); // result = 1 (true)

- **Less than or equal to (<=)**: Checks if the first operand is less than or equal to the second.
  int a = 3, b = 5;
  int result = (a <= b); // result = 1 (true)

3. **Logical Operators**

These operators are used to perform logical operations on expressions.

- **Logical AND (&&)**: Returns true if both operands are true.
  int a = 1, b = 1;
  int result = (a && b); // result = 1 (true)

- **Logical OR (||)**: Returns true if at least one operand is true.
  int a = 1, b = 0;
  int result = (a || b); // result = 1 (true)

- **Logical NOT (!)**: Reverses the logical state of its operand.
  int a = 1;
  int result = !a; // result = 0 (false)

4. **Assignment Operators**

These operators are used to assign values to variables.

- **Simple Assignment (=)**: Assigns the value of the right operand to the left operand.
  int a;
  a = 5; // a = 5

- **Add and Assign (+=)**: Adds the right operand to the left operand and assigns the result to the left operand.
  int a = 5;
  a += 3; // a = 8

- **Subtract and Assign (-=)**: Subtracts the right operand from the left operand and assigns the result to the left operand.
  int a = 5;
  a -= 3; // a = 2

- **Multiply and Assign (*=)**: Multiplies the left operand by the right operand and assigns the result to the left operand.
  int a = 5;
  a *= 3; // a = 15

- **Divide and Assign (/=)**: Divides the left operand by the right operand and assigns the result to the left operand.
  int a = 6;
  a /= 3; // a = 2

- **Modulus and Assign (%=)**: Takes the modulus of the left operand by the right operand and assigns the result to the left operand.
  int a = 7;
  a %= 3; // a = 1

5. **Bitwise Operators**

These operators perform operations on bits.

- **AND (&)**: Performs bitwise AND.
  int a = 5, b = 3;
  int result = a & b; // result = 1

- **OR (|)**: Performs bitwise OR.
  int a = 5, b = 3;
  int result = a | b; // result = 7

- **XOR (^)**: Performs bitwise XOR.
  int a = 5, b = 3;
  int result = a ^ b; // result = 6

- **NOT (~)**: Performs bitwise NOT.
  int a = 5;
  int result = ~a; // result = -6

- **Shift Left (<<)**: Shifts bits to the left.
  int a = 5;
  int result = a << 1; // result = 10

- **Shift Right (>>)**: Shifts bits to the right.
  int a = 5;
  int result = a >> 1; // result = 2

6. **Increment and Decrement Operators**

These operators are used to increase or decrease a variable by 1.

- **Increment (++)**: Increases the value of the operand by 1.
  int a = 5;
  a++; // a = 6

- **Decrement (--)**: Decreases the value of the operand by 1.
  int a = 5;
  a--; // a = 4

7. **Conditional (Ternary) Operator**

The conditional operator is a shorthand for if-else statements.

- **Syntax**: condition ? expr1 : expr2;
  - If the condition is true, expr1 is executed; otherwise, expr2 is executed.

  int a = 5, b = 3;
  int result = (a > b) ? a : b; // result = 5

8. **Sizeof Operator**

The sizeof operator is used to get the size (in bytes) of a data type or variable.

- **Syntax**: sizeof(datatype/variable);

  int a = 5;
  int size = sizeof(a); // size = 4 (on a 32-bit system)

These are the basic operators used in C programming. Each operator serves a different purpose and helps in performing a wide range of tasks efficiently.

Example Program

```
#include <stdio.h>
int main() {
   // Variable initialization
   int a = 10, b = 5, result;
   // Arithmetic operators
   result = a + b; // Addition
   printf("a + b = %d\n", result);
   result = a - b; // Subtraction
   printf("a - b = %d\n", result);
   result = a * b; // Multiplication
   printf("a * b = %d\n", result);
   result = a / b; // Division
   printf("a / b = %d\n", result);
   result = a % b; // Modulus
   printf("a %% b = %d\n", result);
```

```
    // Relational operators
    printf("a == b: %d\n", a == b); // Equal to
    printf("a != b: %d\n", a != b); // Not equal to
    printf("a > b: %d\n", a > b);   // Greater than
    printf("a < b: %d\n", a < b);   // Less than
    // Logical operators
    printf("a && b: %d\n", a && b); // AND
    printf("a || b: %d\n", a || b); // OR
    printf("!a: %d\n", !a);         // NOT
    // Assignment operators
    a += 5; // Add and assign
    printf("a += 5: %d\n", a);

    a -= 3; // Subtract and assign
    printf("a -= 3: %d\n", a);
    a *= 2; // Multiply and assign
    printf("a *= 2: %d\n", a);
    a /= 2; // Divide and assign
    printf("a /= 2: %d\n", a);
    a %= 3; // Modulus and assign
    printf("a %%= 3: %d\n", a);
    // Bitwise operators
    printf("a & b: %d\n", a & b);   // Bitwise AND
    printf("a | b: %d\n", a | b);   // Bitwise OR
    printf("a ^ b: %d\n", a ^ b);   // Bitwise XOR
    printf("~a: %d\n", ~a);         // Bitwise NOT
    printf("a << 1: %d\n", a << 1); // Shift left
    printf("a >> 1: %d\n", a >> 1); // Shift right
    // Increment and Decrement operators
    a++; // Increment
    printf("a++: %d\n", a);
    b--; // Decrement
    printf("b--: %d\n", b);
    // Conditional (Ternary) operator
    result = (a > b) ? a : b; // Ternary operator
    printf("Ternary operator result: %d\n", result);
    // Sizeof operator
    printf("Size of a: %lu bytes\n", sizeof(a));  // Size of int in bytes
    return 0;
}
```

Output

```
a + b = 15
a - b = 5
a * b = 50
```

```
a / b = 2
a % b = 0
a == b: 0
a != b: 1
a > b: 1
a < b: 0
a && b: 1
a || b: 1
!a: 0
a += 5: 15
a -= 3: 12
a *= 2: 24
a /= 2: 12
a %= 3: 0
a & b: 0
a | b: 5
a ^ b: 5
~a: -1
a << 1: 0
a >> 1: 0
a++: 1
b--: 4
Ternary operator result: 5
Size of a: 4 bytes
```