Operators in C

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C language is rich in built-in operators and provides the following types of operators:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment and decrement operators
6. Conditional operators
7. Misc Operators

1. **Arithmetic operator:** These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus.

Operator	Description	Example	Result
`+`	Adds two operands	`a + b`	Sum of `a` and `b`
`-`	Subtracts the second operand from the first	`a - b`	Difference of `a` and `b`
`*`	Multiplies two operands	`a * b`	Product of `a` and `b`
`/`	Divides the first operand by the second	`a / b`	Quotient of `a` and `b`
`%`	Returns the remainder of the division of two operands	`a % b`	Remainder of `a` divided by `b`

Program in C for Arithmetic Operators

```c
#include <stdio.h>

int main() {
    int a = 10;
    int b = 3;
    int sum, difference, product, quotient, remainder;

    sum = a + b;
    difference = a - b;
    product = a * b;
    quotient = a / b;
    remainder = a % b;

    printf("a = %d, b = %d\n", a, b);
    printf("Sum: %d + %d = %d\n", a, b, sum);
    printf("Difference: %d - %d = %d\n", a, b, difference);
    printf("Product: %d * %d = %d\n", a, b, product);
    printf("Quotient: %d / %d = %d\n", a, b, quotient);
    printf("Remainder: %d %% %d = %d\n", a, b, remainder);

    return 0;
}
```

Output:

```makefile
a = 10, b = 3
Sum: 10 + 3 = 13
Difference: 10 - 3 = 7
Product: 10 * 3 = 30
Quotient: 10 / 3 = 3
Remainder: 10 % 3 = 1
```

2. Relational Operators: These operators are used to compare the value of two variables.

Relational Operators

Operator	Description	Example	Result
`==`	Checks if two operands are equal	`a == b`	True if `a` is equal to `b`, else false
`!=`	Checks if two operands are not equal	`a != b`	True if `a` is not equal to `b`, else false
`>`	Checks if the first operand is greater than the second	`a > b`	True if `a` is greater than `b`, else false
`<`	Checks if the first operand is less than the second	`a < b`	True if `a` is less than `b`, else false
`>=`	Checks if the first operand is greater than or equal to the second	`a >= b`	True if `a` is greater than or equal to `b`, else false
`<=`	Checks if the first operand is less than or equal to the second	`a <= b`	True if `a` is less than or equal to `b`, else false

Program in C for Relational Operators

```c
#include <stdio.h>

int main() {
    int a, b;

    // Reading two integers from the user
    printf("Enter the first integer: ");
    scanf("%d", &a);
    printf("Enter the second integer: ");
    scanf("%d", &b);

    // Checking if the two operands are equal using ==
    if (a == b) {
        printf("%d is equal to %d\n", a, b);
    } else {
        printf("%d is not equal to %d\n", a, b);
    }

    // Checking if the two operands are not equal using !=
    if (a != b) {
        printf("%d is not equal to %d\n", a, b);
    } else {
        printf("%d is equal to %d\n", a, b);
    }

    // Checking if the first operand is greater than the second using >
    if (a > b) {
        printf("%d is greater than %d\n", a, b);
    } else {
        printf("%d is not greater than %d\n", a, b);
    }

    return 0;
}
```

Output:

```
Enter the first integer: 10
Enter the second integer: 20
10 is not equal to 20
10 is not equal to 20
10 is not greater than 20
```

3. Logical Operators: These operators are used to perform logical operations on the given two variables.

Logical AND (&&):
- Usage: a && b
- Description: Evaluates to true if both a and b are true.
- Example: If a is true and b is true, then a && b is true. Otherwise, it is false.

Logical OR (||):
- Usage: a || b
- Description: Evaluates to true if at least one of a or b is true.
- Example: If a is true or b is true, then a || b is true. If both are false, then a || b is false.

Logical NOT (!)
- Usage: !a
- Description: Evaluates to true if a is false.
- Example: If a is false, then !a is true. If a is true, then !a is false.

Program in C using Logical Operators

c Copy code

```c
#include <stdio.h>

int main() {
    int a, b, c;

    // Reading three integers from the user
    printf("Enter the first integer: ");
    scanf("%d", &a);
    printf("Enter the second integer: ");
    scanf("%d", &b);
    printf("Enter the third integer: ");
    scanf("%d", &c);

    // Checking if all three numbers are positive using &&
    if (a > 0 && b > 0 && c > 0) {
        printf("All three numbers are positive.\n");
    } else {
        printf("Not all three numbers are positive.\n");
    }

    // Checking if at least one number is positive using ||
    if (a > 0 || b > 0 || c > 0) {
        printf("At least one number is positive.\n");
    } else {
        printf("None of the numbers are positive.\n");
    }

    // Checking if none of the numbers are zero using !
    if (!(a == 0 || b == 0 || c == 0)) {
        printf("None of the numbers are zero.\n");
    } else {
        printf("At least one number is zero.\n");
    }

    return 0;
}
```

Output:

```
sql                                                    Copy code

Enter the first integer: 10
Enter the second integer: -5
Enter the third integer: 0
Not all three numbers are positive.
At least one number is positive.
At least one number is zero.
```

Explanation:

Input: The program prompts the user to enter three integers using scanf.

Logical AND (&&):

- The program uses the && operator to check if all three integers are positive.
- If all are positive, it prints a message stating that all three numbers are positive.
- Otherwise, it prints that not all three numbers are positive.

Logical OR (||):

- The program uses the || operator to check if at least one of the integers is positive.
- If at least one is positive, it prints a message stating that at least one number is positive.
- Otherwise, it prints that none of the numbers are positive.

Logical NOT (!):

- The program uses the ! operator to check if none of the integers are zero.
- It achieves this by using the || operator within the ! operator.
- If none of the integers are zero, it prints a message stating that none of the numbers are zero.
- Otherwise, it prints that at least one number is zero.

4. Assignment Operators: In C programs, values for the variables are assigned using assignment operators.

Assignment Operators in C

Operator	Name	Description	Example	Equivalent to
=	Assignment	Assigns the right-hand operand to the left-hand operand	a = b	-
+=	Add and Assign	Adds the right operand to the left operand and assigns the result to the left operand	a += b	a = a + b
-=	Subtract and Assign	Subtracts the right operand from the left operand and assigns the result to the left operand	a -= b	a = a - b
*=	Multiply and Assign	Multiplies the left operand by the right operand and assigns the result to the left operand	a *= b	a = a * b
/=	Divide and Assign	Divides the left operand by the right operand and assigns the result to the left operand	a /= b	a = a / b
%=	Modulus and Assign	Takes the modulus using two operands and assigns the result to the left operand	a %= b	a = a % b

C Program Using += and -= Assignment Operators

```c
#include <stdio.h>

int main() {
    int a, b;

    // Reading two integers from the user
    printf("Enter the first integer (a): ");
    scanf("%d", &a);
    printf("Enter the second integer (b): ");
    scanf("%d", &b);

    // Display initial values
    printf("Initial values: a = %d, b = %d\n", a, b);

    // Add and Assign
    a += b;
    printf("After a += b: a = %d\n", a);

    // Resetting a to initial value
    a -= b;  // This is to bring a back to its original value before the next operation

    // Subtract and Assign
    a -= b;
    printf("After a -= b: a = %d\n", a);

    return 0;
}
```

Explanation:
Reading User Input: The program prompts the user to enter two integers a and b using scanf.

Initial Values: The initial values of a and b are displayed.

Add and Assign (+=):

- a += b: Adds b to a and assigns the result to a.
- The updated value of a is displayed.

Resetting a:

- a is brought back to its initial value by subtracting b from the result of a += b.
- This ensures that a is restored to its original value before the next operation.

Subtract and Assign (-=):

- a -= b: Subtracts b from a and assigns the result to a.
- The updated value of a is displayed.

Output:

```less
Enter the first integer (a): 10
Enter the second integer (b): 5
Initial values: a = 10, b = 5
After a += b: a = 15
After a -= b: a = 5
```

Increment and Decrement Operator

In C, ++ and – are called increment and decrement operators respectively. Both of these operators are unary operators, i.e, used on single operand. ++ adds 1 to operand and – subtracts 1 to operand respectively. For example:
Let a=5 and b=10

- a++; //a becomes 6
- a--; //a becomes 5
- ++a; //a becomes 6
- --a; //a becomes 5

C Program Using Increment and Decrement Operators

```c
#include <stdio.h>

int main() {
    int a;

    // Reading an integer from the user
    printf("Enter an integer: ");
    scanf("%d", &a);

    // Initial value
    printf("Initial value of a: %d\n", a);

    // Post-increment
    printf("Value after a++ (post-increment): %d\n", a++);
    printf("Value of a after post-increment operation: %d\n", a);

    // Pre-increment
    printf("Value after ++a (pre-increment): %d\n", ++a);

    // Post-decrement
    printf("Value after a-- (post-decrement): %d\n", a--);
    printf("Value of a after post-decrement operation: %d\n", a);

    // Pre-decrement
    printf("Value after --a (pre-decrement): %d\n", --a);

    return 0;
}
```

Explanation:

Reading User Input: The program prompts the user to enter an integer a using scanf.

Initial Value: The initial value of a is displayed.

Post-Increment (a++):

- The value of a is displayed before it is incremented by one.

- The value of a is displayed again after the post-increment operation to show the updated value.

Pre-Increment (++a):

- The value of a is incremented by one and then the updated value is displayed.

Post-Decrement (a--):

- The value of a is displayed before it is decremented by one.
- The value of a is displayed again after the post-decrement operation to show the updated value.

Pre-Decrement (--a):

- The value of a is decremented by one and then the updated value is displayed.

Output:

```mathematica
Enter an integer: 10
Initial value of a: 10
Value after a++ (post-increment): 10
Value of a after post-increment operation: 11
Value after ++a (pre-increment): 12
Value after a-- (post-decrement): 12
Value of a after post-decrement operation: 11
Value after --a (pre-decrement): 10
```

Conditional Operators (? :)

Conditional operators are used in decision making in C programming, i.e, executes different statements according to test condition whether it is either true or false.

Syntax of conditional operators;

conditional_expression?expression1:expression2

- condition: The expression that is evaluated first. If this condition is true (non-zero), expression1 is evaluated and its value is returned.
- expression1: The expression that is evaluated and returned if the condition is true.

- expression2: The expression that is evaluated and returned if the condition is false.

Example

```c
int a = 5, b = 10;
int max;

max = (a > b) ? a : b;
printf("The maximum value is: %d\n", max);
```

In the example above, the condition a > b is evaluated:

- If a > b is true, max is assigned the value of a.
- If a > b is false, max is assigned the value of b.

C Program Using Conditional Operators

```c
#include <stdio.h>

int main() {
    int a, b, max, min;

    // Reading two integers from the user
    printf("Enter the first integer (a): ");
    scanf("%d", &a);
    printf("Enter the second integer (b): ");
    scanf("%d", &b);

    // Using conditional operator to find the maximum value
    max = (a > b) ? a : b;
    printf("The maximum value is: %d\n", max);

    // Using conditional operator to find the minimum value
    min = (a < b) ? a : b;
    printf("The minimum value is: %d\n", min);

    // Using conditional operator to check if a number is even or odd
    printf("Enter an integer to check even or odd: ");
    int num;
    scanf("%d", &num);
    printf("%d is %s\n", num, (num % 2 == 0) ? "even" : "odd");

    return 0;
}
```

Explanation

Reading User Input: The program prompts the user to enter two integers a and b using scanf.

Finding the Maximum Value:

- The ternary operator (a > b) ? a : b is used to determine the maximum value between a and b.
- If a is greater than b, max is assigned the value of a; otherwise, max is assigned the value of b.
- The maximum value is then printed.

Finding the Minimum Value:

- The ternary operator (a < b) ? a : b is used to determine the minimum value between a and b.
- If a is less than b, min is assigned the value of a; otherwise, min is assigned the value of b.
- The minimum value is then printed.

Checking Even or Odd:

- The program reads an integer num from the user.
- The ternary operator (num % 2 == 0) ? "even" : "odd" is used to check if num is even or odd.
- If num % 2 == 0 is true, "even" is printed; otherwise, "odd" is printed.

Output

vbnet Copy code

```
Enter the first integer (a): 10
Enter the second integer (b): 5
The maximum value is: 10
The minimum value is: 5
Enter an integer to check even or odd: 7
7 is odd
```

Miscellaneous Operators in C

C provides several operators that do not fit neatly into other categories but are still widely used. These miscellaneous operators include:

- sizeof Operator
- Comma Operator

Using `sizeof` with Data Types:

```c
#include <stdio.h>

int main() {
    printf("Size of int: %lu bytes\n", sizeof(int));
    printf("Size of float: %lu bytes\n", sizeof(float));
    printf("Size of char: %lu bytes\n", sizeof(char));
    printf("Size of double: %lu bytes\n", sizeof(double));
    return 0;
}
```

Using sizeof with Data Types:

- The sizeof operator returns the size of the specified data type.
- This can be useful to determine how much memory different data types occupy.

```plaintext
Size of int: 4 bytes
Size of float: 4 bytes
Size of char: 1 byte
Size of double: 8 bytes
```

Comma Operator in C

The comma operator in C is used to separate two or more expressions, ensuring that each of the expressions is evaluated, and the result of the last expression is returned.

It's a sequence point, which means all side effects of the previous expression are completed before the next expression is evaluated.

Syntax of comma operator:

expression1, expression2, expression3, ..., expression

Example of Comma Operator

```c
#include <stdio.h>

int main() {
    int a, b;
    a = 1, b = 2; // Using comma operator to separate expressions
    printf("a = %d, b = %d\n", a, b);

    int result = (a = 3, b = 4, a + b); // Evaluating multiple expressions
    printf("Result: %d\n", result); // Result is the value of the last expression (a + b)

    return 0;
}
```

Explanation

Variable Initialization and Assignment: a = 1, b = 2; initializes a to 1 and b to 2.

First printf Statement:

- printf("a = %d, b = %d\n", a, b); prints the values of a and b.
- At this point, a is 1 and b is 2.

Comma Operator and Assignment:

- int result = (a = 3, b = 4, a + b);
- This statement performs multiple assignments and a final addition:
- a = 3: Assigns 3 to a.
- b = 4: Assigns 4 to b.
- a + b: Adds a and b (3 + 4) and assigns the result (7) to result.

Second printf Statement:

- printf("Result: %d\n", result); prints the value of result.
- The value of result is 7.

```plaintext
a = 1, b = 2
Result: 7
```