Operators in C Language

By

Prof. Muhammad Iqbal Bhat

Department of Higher Education,

Government Degree College Beerwah

Operators in C language

Operators are symbols that perform operations on operands in a programming language

C language has various types of operators, including arithmetic, relational, logical, assignment, and bitwise operators

Operators allow programmers to manipulate data and perform different computations in C programs

Understanding the different types of operators and their functionality is essential for writing effective C code

In the following slides, we will delve into each type of operator in C language, their syntax, examples, and common mistakes to avoid.

Operators

- Arithmetic Operators.
- Increment and Decrement Operators.
- Relational Operators.
- Logical Operators.
- Bitwise Operators.
- Assignment Operators.
- Conditional Operator.
- Special Operators

Arithmetic Operators

Arithmetic operators perform basic mathematical operations on operands in C language.

- Addition (+): Adds two operands together and returns the sum
- Subtraction (-): Subtracts one operand from another and returns the difference
- Multiplication (*): Multiplies two operands and returns the product
- Division (/): Divides one operand by another and returns the quotient
- Modulus (%): Divides one operand by another and returns the remainder

Arithmetic Operators- Example

```
int a = 5; // declare and initialize variable a with value 5
int b = 3; // declare and initialize variable b with value 3
int sum = a + b; // add a and b, store the result in sum
int difference = a - b; // subtract b from a, store the result in
difference
int product = a * b; // multiply a and b, store the result in product
int quotient = a / b; // divide a by b, store the result in quotient
int remainder = a % b; // divide a by b and get the remainder, store the
result in remainder
```

Increment/Decrement Operators

Increment and decrement	Increment and decrement operators are used to increase or decrease the value of a variable by a fixed amount
Increment	Increment (++): Increases the value of a variable by 1
Decrement	Decrement (--): Decreases the value of a variable by 1
Prefix	Prefix increment: ++operand (e.g., ++x)
Prefix	Prefix decrement: --operand (e.g., --x)
Postfix	Postfix increment: operand++ (e.g., x++)
Postfix	Postfix decrement: operand-- (e.g., x--)

Increment/Decrement Operators- Example

```
int x = 5; // declare and initialize variable x with value 5

int y = 10; // declare and initialize variable y with value 10


// Prefix increment

int result1 = ++x; // increment x by 1 and store the result in
result1, x is now 6


// Prefix decrement

int result2 = --y; // decrement y by 1 and store the result in
result2, y is now 9


int a = 3; // declare and initialize variable a with value 3

int b = 8; // declare and initialize variable b with value 8


// Postfix increment

int result3 = a++; // store the current value of a in result3, then
increment a by 1, a is now 4


// Postfix decrement

int result4 = b--; // store the current value of b in result4, then
decrement b by 1, b is now 7
```

Relational Operators:

Relational operators compare two operands and return a boolean value (true or false) based on the comparison result

- Greater than (>): Checks if the value of operand1 is greater than the value of operand2
- Less than (<): Checks if the value of operand1 is less than the value of operand2
- Greater than or equal to (>=): Checks if the value of operand1 is greater than or equal to the value of operand2
- Less than or equal to (<=): Checks if the value of operand1 is less than or equal to the value of operand2
- Equal to (==): Checks if the value of operand1 is equal to the value of operand2
- Not equal to (!=): Checks if the value of operand1 is not equal to the value of operand2

Relational Operators-Example

```
int a = 5;
int b = 3;
bool result1 = a > b;
bool result2 = a < b;
bool result3 = a >= b;
bool result4 = a <= b;
bool result5 = a == b;
bool result6 = a != b;
```

Logical Operators:

Logical operators are used to perform logical operations on boolean values (true or false)

- AND (&&): Returns true if both operands are true, false otherwise
- OR (||): Returns true if at least one operand is true, false otherwise
- NOT (!): Returns the opposite of the operand's value

Logical Operators-Example

```
int x = 1;
int y = 0;

// AND operator
int result1 = x && y;

// OR operator
int result2 = x || y;

// NOT operator
int result3 = !x;
```

Assignment Operators:

Assignment operators are used to assign a value to a variable in C language.

- Assignment (=): Assigns a value to a variable
- Compound assignment operators (e.g., +=, -=, *=, /=): Performs an operation and assigns the result to a variable in a single step

Assignment Operators-Example

```
int x = 10;

// Assignment operator
x = 20;

// Compound assignment operator
x += 5;
```

Bitwise Operators

Bitwise operators are used to perform operations on individual bits of integer values in C language.

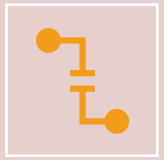
- Bitwise AND (&): Performs bitwise AND operation on each pair of corresponding bits
- Bitwise OR (|): Performs bitwise OR operation on each pair of corresponding bits
- Bitwise XOR (^): Performs bitwise XOR operation on each pair of corresponding bits
- Bitwise NOT (~): Inverts the bits of the operand
- Left shift (<<): Shifts the bits of the operand to the left by a specified number of positions
- Right shift (>>): Shifts the bits of the operand to the right by a specified number of positions

Bitwise Operators-Example

```
unsigned int x = 5; // declare and initialize unsigned integer variable x with value 5 (binary: 0101)

unsigned int y = 3; // declare and initialize unsigned integer variable y with value 3 (binary: 0011)

// Bitwise AND

unsigned int result1 = x & y; // perform bitwise AND operation on x and y, store the result in result1 (binary: 0001)

// Bitwise OR

unsigned int result2 = x | y; // perform bitwise OR operation on x and y, store the result in result2 (binary: 0111)

// Bitwise XOR

unsigned int result3 = x ^ y; // perform bitwise XOR operation on x and y, store the result in result3 (binary: 0110)

// Bitwise NOT

unsigned int result4 = ~x; // perform bitwise NOT operation on x, store the result in result4 (binary: 1111 1111 1111 1100)

// Left shift

unsigned int result5 = x << 2; // shift the bits of x to the left by 2 positions, store the result in result5 (binary: 10100)

// Right shift

unsigned int result6 = x >> 1; // shift the bits of x to the right by 1 position, store the result in result6 (binary: 0010)
```

Operator Precedence

Operator precedence determines the order in which operators are evaluated in an expression.



C language follows a specific set of rules for operator precedence, which dictates the order in which operators are evaluated.

Operator Precedence

Precedence	Operator(s)	Example Expression	Evaluation Result
1	() [] -> .	a = (b + c) * d	a = (10 + 20) * 30
2	++ --	a = b++ + c	a = 10 + 20
3	+ -	a = b + c - d	a = 30 - 5
4	* / %	a = b * c / d	a = 25 / 5
5	+ -	a = b + c - d	a = 5
6	< <= > >=	a = b < c	a = 1 (True)
7	== !=	a = b == c	a = 0 (False)
8	&&	a = b && c	a = 1 (True)
9	||	a = b || c	a = 1 (True)
10	= += -= *= /= %=	a = b + c * d	a = 300

Program Examples

1. Calculate the area of a rectangle

```
#include <stdio.h>

int main() {
    int length, width;
    printf("Enter length of the rectangle: ");
    scanf("%d", &length);
    printf("Enter width of the rectangle: ");
    scanf("%d", &width);

    int area = length * width;
    printf("Area of the rectangle: %d\n", area);

    return 0;
}
```

2. Calculate the sum and average of three numbers:

```c
#include <stdio.h>

int main() {
    int num1, num2, num3;
    printf("Enter three numbers separated by spaces: ");
    scanf("%d %d %d", &num1, &num2, &num3);

    int sum = num1 + num2 + num3;
    float average = (float)sum / 3;

    printf("Sum: %d\n", sum);
    printf("Average: %.2f\n", average);

    return 0;
}
```

3. Check if a number is even or odd:

```
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num % 2 == 0) {
        printf("%d is even.\n", num);
    } else {
        printf("%d is odd.\n", num);
    }

    return 0;
}
```

4. Swap two numbers without using a temporary variable

```
#include <stdio.h>

int main() {
    int num1, num2;
    printf("Enter two numbers separated by spaces: ");
    scanf("%d %d", &num1, &num2);

    printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);

    num1 = num1 + num2;
    num2 = num1 - num2;
    num1 = num1 - num2;

    printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);

    return 0;
}
```

5. Check if a year is a leap year or not

```c
#include <stdio.h>

int main() {
    int year;
    printf("Enter a year: ");
    scanf("%d", &year);

    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
        printf("%d is a leap year.\n", year);
    } else {
        printf("%d is not a leap year.\n", year);
    }

    return 0;
}
```

6. Convert temperature from Celsius to Fahrenheit

```c
#include <stdio.h>

int main() {
    float celsius;
    printf("Enter temperature in Celsius: ");
    scanf("%f", &celsius);

    float fahrenheit = (celsius * 9/5) + 32;

    printf("Temperature in Fahrenheit: %.2f\n", fahrenheit);

    return 0;
}
```

7. Calculating Simple Interest:

```
#include <stdio.h>

int main() {
    float principal = 1000;
    float rate = 0.05;
    float time = 2;

    float interest = principal * rate * time;

    printf("Principal: $%.2f\n", principal);
    printf("Rate: %.2f\n", rate);
    printf("Time: %.2f years\n", time);
    printf("Simple Interest: $%.2f\n", interest);

    return 0;
}
```

8. Calculate the sum of digits of a given number:

```
#include <stdio.h>

int main() {
    int num;
    printf("Enter a positive integer: ");
    scanf("%d", &num);

    int sum = 0;
    while (num > 0) {
        sum += num % 10;
        num /= 10;
    }

    printf("Sum of digits: %d\n", sum);

    return 0;
}
```

Questions?