Operators in C

An operator is a symbol that instructs the compiler to perform specific mathematical, relational, logical, or bitwise operations on one or more operands. Operators in C are categorized into several types based on their functionality:

1. **Arithmetic Operators**: Perform arithmetic operations such as addition, subtraction, multiplication, division, and modulus.
2. **Relational Operators**: Compare two values and return a Boolean result (true or false) based on whether the comparison is true or false.
3. **Logical Operators**: Perform logical operations on Boolean values (true or false). They are used to combine multiple conditions or negate a condition.
4. **Bitwise Operators**: Perform operations on individual bits of integer operands.
5. **Assignment Operators**: Assign a value to a variable and also perform a simple arithmetic operation at the same time.
6. **Increment and Decrement Operators**: Increase or decrease the value of an operand by one.
7. **Conditional Operator (Ternary Operator)**: Provides a compact way to evaluate a condition and choose one of two expressions.
8. **Comma Operator**: Evaluates multiple expressions from left to right and returns the value of the rightmost expression.

Here's a list of operators in C along with explanations and examples for each:

These operators cover a wide range of functionalities in C programming, from basic arithmetic and logical operations to more specialized bitwise manipulations and assignment operations.

**Arithmetic Operators**

1. **Addition +**
   * **Example**: int sum = 5 + 3;
   * **Explanation**: Adds two operands.
2. **Subtraction -**
   * **Example**: int difference = 8 - 2;
   * **Explanation**: Subtracts the second operand from the first.
3. **Multiplication \***
   * **Example**: int product = 4 \* 6;
   * **Explanation**: Multiplies two operands.
4. **Division /**
   * **Example**: float result = 10.0 / 3.0;
   * **Explanation**: Divides the first operand by the second.
5. **Modulus %**
   * **Example**: int remainder = 10 % 3;
   * **Explanation**: Computes the remainder after dividing the first operand by the second.

**Relational Operators**

1. **Equal to ==**
   * **Example**: if (a == b)
   * **Explanation**: Checks if two operands are equal.
2. **Not equal to !=**
   * **Example**: if (a != b)
   * **Explanation**: Checks if two operands are not equal.
3. **Greater than >**
   * **Example**: if (a > b)
   * **Explanation**: Checks if the left operand is greater than the right operand.
4. **Less than <**
   * **Example**: if (a < b)
   * **Explanation**: Checks if the left operand is less than the right operand.
5. **Greater than or equal to >=**
   * **Example**: if (a >= b)
   * **Explanation**: Checks if the left operand is greater than or equal to the right operand.
6. **Less than or equal to <=**
   * **Example**: if (a <= b)
   * **Explanation**: Checks if the left operand is less than or equal to the right operand.

**Logical Operators**

1. **Logical AND &&**
   * **Example**: if (x > 0 && y > 0)
   * **Explanation**: Returns true if both operands are true.
2. **Logical OR ||**
   * **Example**: if (x > 0 || y > 0)
   * **Explanation**: Returns true if at least one operand is true.
3. **Logical NOT !**
   * **Example**: if (!flag)
   * **Explanation**: Reverses the logical state of its operand.

**Bitwise Operators**

1. **Bitwise AND &**
   * **Example**: int result = a & b;
   * **Explanation**: Performs a bitwise AND operation on the operands.
2. **Bitwise OR |**
   * **Example**: int result = a | b;
   * **Explanation**: Performs a bitwise OR operation on the operands.
3. **Bitwise XOR ^**
   * **Example**: int result = a ^ b;
   * **Explanation**: Performs a bitwise XOR (exclusive OR) operation on the operands.
4. **Bitwise NOT ~**
   * **Example**: int result = ~a;
   * **Explanation**: Inverts all the bits of its operand.
5. **Left Shift <<**
   * **Example**: int result = a << 2;
   * **Explanation**: Shifts the bits of the left operand to the left by the number of positions specified by the right operand.
6. **Right Shift >>**
   * **Example**: int result = a >> 1;
   * **Explanation**: Shifts the bits of the left operand to the right by the number of positions specified by the right operand.

**Assignment Operators**

1. **Assignment =**
   * **Example**: a = 10;
   * **Explanation**: Assigns the value on the right to the variable on the left.
2. **Add and assign +=**
   * **Example**: a += 5;
   * **Explanation**: Adds the value on the right to the variable on the left and assigns the result to the variable on the left.
3. **Subtract and assign -=**
   * **Example**: a -= 3;
   * **Explanation**: Subtracts the value on the right from the variable on the left and assigns the result to the variable on the left.
4. **Multiply and assign \*=**
   * **Example**: a \*= 2;
   * **Explanation**: Multiplies the variable on the left by the value on the right and assigns the result to the variable on the left.
5. **Divide and assign /=**
   * **Example**: a /= 4;
   * **Explanation**: Divides the variable on the left by the value on the right and assigns the result to the variable on the left.
6. **Modulus and assign %=**
   * **Example**: a %= 3;
   * **Explanation**: Computes the modulus of the variable on the left by the value on the right and assigns the result to the variable on the left.

**Increment and Decrement Operators**

1. **Increment ++**
   * **Example**: a++;
   * **Explanation**: Increases the value of the operand by 1.
2. **Decrement --**
   * **Example**: a--;
   * **Explanation**: Decreases the value of the operand by 1.

**Conditional Operator**

1. **Conditional ?:**
   * **Example**: int max = (a > b) ? a : b;
   * **Explanation**: Evaluates a condition and returns one of two expressions based on whether the condition is true or false.

**Comma Operator**

1. **Comma ,**
   * **Example**: int x = (a++, b++);
   * **Explanation**: Evaluates both operands from left to right and returns the value of the right operand.

Practical Examples on Each One:

Here are the C programs with explanations and their respective outputs after execution:

### 1. Arithmetic Operators

#include <stdio.h>

int main() {

int a = 10, b = 4;

int sum = a + b;

int difference = a - b;

int product = a \* b;

float quotient = (float)a / b; // Casting to float for accurate division

int remainder = a % b;

printf("Sum: %d\n", sum); // Output: Sum: 14

printf("Difference: %d\n", difference); // Output: Difference: 6

printf("Product: %d\n", product); // Output: Product: 40

printf("Quotient: %.2f\n", quotient); // Output: Quotient: 2.50

printf("Remainder: %d\n", remainder); // Output: Remainder: 2

return 0;

}

**Explanation**:

* **Arithmetic Operators**:
  + +: Adds a and b.
  + -: Subtracts b from a.
  + \*: Multiplies a and b.
  + /: Divides a by b.
  + %: Computes the remainder of a divided by b.

**Output**:

Sum: 14

Difference: 6

Product: 40

Quotient: 2.50

Remainder: 2

### 2. Relational Operators

#include <stdio.h>

int main() {

int a = 5, b = 10;

if (a == b) {

printf("a is equal to b\n");

} else {

printf("a is not equal to b\n"); // Output: a is not equal to b

}

if (a != b) {

printf("a is not equal to b\n"); // Output: a is not equal to b

}

if (a > b) {

printf("a is greater than b\n"); // No output for this block

}

if (a < b) {

printf("a is less than b\n"); // Output: a is less than b

}

if (a >= b) {

printf("a is greater than or equal to b\n"); // No output for this block

}

if (a <= b) {

printf("a is less than or equal to b\n"); // Output: a is less than or equal to b

}

return 0;

}

**Explanation**:

* **Relational Operators**:
  + ==: Checks if a is equal to b.
  + !=: Checks if a is not equal to b.
  + >: Checks if a is greater than b.
  + <: Checks if a is less than b.
  + >=: Checks if a is greater than or equal to b.
  + <=: Checks if a is less than or equal to b.

**Output**:

a is not equal to b

a is not equal to b

a is less than b

a is less than or equal to b

### 3. Logical Operators

#include <stdio.h>

int main() {

int x = 5, y = 7;

if (x > 0 && y > 0) {

printf("Both x and y are positive\n"); // Output: Both x and y are positive

}

if (x > 0 || y > 0) {

printf("At least one of x or y is positive\n"); // Output: At least one of x or y is positive

}

if (!(x > 0)) {

printf("x is not positive\n"); // No output for this block

}

return 0;

}

**Explanation**:

* **Logical Operators**:
  + &&: Checks if both x and y are positive.
  + ||: Checks if at least one of x or y is positive.
  + !: Negates the condition; checks if x is not positive.

**Output**:

Both x and y are positive

At least one of x or y is positive

### 4. Bitwise Operators

#include <stdio.h>

int main() {

unsigned int a = 60; // 60 in binary: 0011 1100

unsigned int b = 13; // 13 in binary: 0000 1101

int result;

// Bitwise AND

result = a & b; // Result: 12 (0000 1100)

printf("a & b = %d\n", result); // Output: a & b = 12

// Bitwise OR

result = a | b; // Result: 61 (0011 1101)

printf("a | b = %d\n", result); // Output: a | b = 61

// Bitwise XOR

result = a ^ b; // Result: 49 (0011 0001)

printf("a ^ b = %d\n", result); // Output: a ^ b = 49

// Bitwise NOT

result = ~a; // Result: -61 (in 2's complement form)

printf("~a = %d\n", result); // Output: ~a = -61

// Left Shift

result = a << 2; // Result: 240 (1111 0000)

printf("a << 2 = %d\n", result); // Output: a << 2 = 240

// Right Shift

result = a >> 2; // Result: 15 (0000 1111)

printf("a >> 2 = %d\n", result); // Output: a >> 2 = 15

return 0;

}

**Explanation**:

* **Bitwise Operators**:
  + &: Performs bitwise AND operation.
  + |: Performs bitwise OR operation.
  + ^: Performs bitwise XOR (exclusive OR) operation.
  + ~: Performs bitwise NOT (one's complement) operation.
  + <<: Performs left shift operation.
  + >>: Performs right shift operation.

**Output**:

a & b = 12

a | b = 61

a ^ b = 49

~a = -61

a << 2 = 240

a >> 2 = 15

### 5. Assignment Operators

#include <stdio.h>

int main() {

int a = 10, b = 5, result;

result = a; // Simple assignment

printf("Result: %d\n", result); // Output: Result: 10

result += b; // Add and assign: result = result + b

printf("Result after addition: %d\n", result); // Output: Result after addition: 15

result -= b; // Subtract and assign: result = result - b

printf("Result after subtraction: %d\n", result); // Output: Result after subtraction: 10

result \*= b; // Multiply and assign: result = result \* b

printf("Result after multiplication: %d\n", result); // Output: Result after multiplication: 50

result /= b; // Divide and assign: result = result / b

printf("Result after division: %d\n", result); // Output: Result after division: 10

result %= b; // Modulus and assign: result = result % b

printf("Result after modulus: %d\n", result); // Output: Result after modulus: 0

return 0;

}

**Explanation**:

* **Assignment Operators**:
  + =: Assigns the value on the right to the variable on the left.
  + +=: Adds the value on the right to the variable on the left and assigns the result to the variable on the left.
  + -=: Subtracts the value on the right from the variable on the left and assigns the result to the variable on the left.
  + \*=: Multiplies the variable on the left by the value on the right and assigns the result to the variable on the left.
  + /=: Divides the variable on the left by the value on the right and assigns the result to the variable on the left.
  + %=: Computes the modulus of the variable on the left by the value on the right and assigns the result to the variable on the left.

**Output**:

Result: 10

Result after addition: 15

Result after subtraction: 10

Result after multiplication: 50

Result after division: 10

Result after modulus: 0

### 6. Increment and Decrement Operators

#include <stdio.h>

int main() {

int a = 5;

// Increment operator

a++;

printf("After increment, a = %d\n", a); // Output: After increment, a = 6

// Decrement operator

a--;

printf("After decrement, a = %d\n", a); // Output: After decrement, a = 5

return 0;

}

**Explanation**:

* **Increment and Decrement Operators**:
  + ++: Increments the value of the variable by 1.
  + --: Decrements the value of the variable by 1.

**Output**:

After increment, a = 6

After decrement, a = 5

### 7. Conditional Operator (Ternary Operator)

#include <stdio.h>

int main() {

int a = 10, b = 5;

int max;

// Ternary operator to find maximum

max = (a > b) ? a : b;

printf("Maximum between %d and %d is %d\n", a, b, max); // Output: Maximum between 10 and 5 is 10

return 0;

}

**Explanation**:

* **Conditional Operator (Ternary Operator)**:
  + condition ? true\_expression : false\_expression: Evaluates condition. If condition is true, evaluates and returns true\_expression; otherwise, evaluates and returns false\_expression.

**Output**:

Maximum between 10 and 5 is 10

### 8. Comma Operator

#include <stdio.h>

int main() {

int a = 5, b = 10, c;

// Comma operator to evaluate multiple expressions

c = (a++, b++, a + b);

printf("Result of (a++, b++, a + b): %d\n", c); // Output: Result of (a++, b++, a + b): 16

return 0;

}

**Explanation**:

* **Comma Operator**:
  + ,: Evaluates multiple expressions from left to right and returns the value of the rightmost expression.

**Output**:

Result of (a++, b++, a + b): 16