Copilot

In **C programming**, **bitwise operators** allow low-level manipulation of data stored in a computer's memory. These operators work at the **bit level**, enabling you to perform operations on individual bits within data. Let's explore the main bitwise operators in C:

1. **Bitwise AND (`&`)**: Takes two numbers as operands and performs an AND operation on each bit. The result is 1 only if both corresponding bits are 1.
2. **Bitwise OR (`|`)**: Also takes two numbers as operands and performs an OR operation on each bit. The result is 1 if either of the two bits is 1.
3. **Bitwise XOR (`^`)**: Takes two numbers as operands and performs an XOR (exclusive OR) operation on each bit. The result is 1 if the two bits are different.
4. **Left Shift (`<<`)**: Shifts the bits of the first operand to the left by a specified number of places (determined by the second operand).
5. **Right Shift (`>>`)**: Shifts the bits of the first operand to the right by a specified number of places.
6. **Bitwise NOT (`~`)**: Takes one number and inverts all its bits.

Here's an example program demonstrating the use of these operators:

```c
#include <stdio.h>

int main() {
    unsigned char a = 5, b = 9;

    printf("a = %d, b = %d\n", a, b);
    printf("a & b = %d\n", a & b);
    printf("a | b = %d\n", a | b);
    printf("a ^ b = %d\n", a ^ b);
    printf("~a = %d\n", ~a);
    printf("b << 1 = %d\n", b << 1);
    printf("b >> 1 = %d\n", b >> 1);

    return 0;
}
```

Output:

```
a = 5, b = 9
a & b = 1
a | b = 13
a ^ b = 12
~a = 250
b << 1 = 18
b >> 1 = 4
```

Remember that bitwise operators are essential for low-level operations and can be powerful tools in certain scenarios. However, use them carefully and consider their behavior with negative numbers and integer size limits.