print(10\*4/6+3-1%2)

\* and / same priority

%

+ and - same priority

Bitwise operators

print(10&4+3)

+ has more priority then &

10&7=2

print(7+2&4+3&9)

9&7&9=1

EXPLANATION:

Let's break down each of the expressions and discuss their results:

1. `print(10\*4/6+3-1%2)`

- The expression inside the parentheses is evaluated as follows:

- `10 \* 4 = 40`

- `40 / 6 = 6.66667` (in C, dividing integers yields an integer, so it's truncated to 6)

- `1 % 2 = 1`

- `6 + 3 - 1 = 8`

- The final result is 8.

2. `print(10&4+3)`

- The expression inside the parentheses is evaluated as follows:

- `4 + 3 = 7`

- `10 & 7 = 2`

- The final result is 2.

- The `&` operator performs a bitwise AND operation.

3. `print(7+2&4+3&9)`

- The expression inside the parentheses is evaluated as follows:

- `7+2 = 9`

- `4+3 = 7`

- `9 & 7 & 9 = 1`

- The final result is 1.

- The `&` operator performs bitwise AND operations, which are applied after addition.

Notes:

- In C, `\*` and `/` have the same priority, and they are evaluated from left to right.

- `%` (modulus) has higher precedence than `\*` and `/`.

- `+` and `-` have the same priority, and they are also evaluated from left to right.

- `&` (bitwise AND) and other bitwise operators have lower precedence than arithmetic operators, so they are performed after addition and subtraction.

It's essential to be aware of operator precedence and the order of evaluation in C to write correct and predictable expressions. Parentheses can be used to change the order of evaluation when necessary.

PRIORITY OF BITWISE OPEARTORS:

Bitwise operators in C have the following priority from highest to lowest:

1. `~` (Bitwise NOT)

2. `<<` (Bitwise Left Shift) and `>>` (Bitwise Right Shift)

3. `&` (Bitwise AND)

4. `^` (Bitwise XOR)

5. `|` (Bitwise OR)

Let's go through each of these operators and provide examples to justify their priorities:

1. `~` (Bitwise NOT):

- The bitwise NOT operator inverts each bit of the operand.

- Example: `~5` would result in `-6` because `5` in binary is `00000101`, and `~5` would be `11111010`, which is `-6` in decimal.

2. `<<` (Bitwise Left Shift) and `>>` (Bitwise Right Shift):

- These operators shift the bits of the left operand to the left or right by the number of positions specified by the right operand.

- Example: `10 << 2` would result in `40` because it shifts the binary representation of `10` (`00001010`) two positions to the left, resulting in `00101000`, which is `40` in decimal.

Tricks: 5>>1 = 2

0 1 0 1 == 5

0 0 1 0 == 2

5>>1==5/2 ans 2

10<<3==10\*2^3=ans = 80

3. `&` (Bitwise AND):

- The bitwise AND operator performs a bitwise AND operation between the bits of two operands.

- Example: `12 & 7` would result in `4` because it performs a bitwise AND between `1100` (binary for `12`) and `0111` (binary for `7`), resulting in `0100`, which is `4` in decimal.

4. `^` (Bitwise XOR):

- The bitwise XOR operator performs a bitwise XOR operation between the bits of two operands.

- Example: `9 ^ 3` would result in `10` because it performs a bitwise XOR between `1001` (binary for `9`) and `0011` (binary for `3`), resulting in `1010`, which is `10` in decimal.

Note:

Even 1’s : 0

Odd 1’s : 1

4^6^5

0 1 0 0

0 1 1 0

0 1 0 1

Ans after exor: 0 1 1 1

Xor of number itself is zero

Xor of number with zero is number itself

5. `|` (Bitwise OR):

- The bitwise OR operator performs a bitwise OR operation between the bits of two operands.

- Example: `5 | 3` would result in `7` because it performs a bitwise OR between `0101` (binary for `5`) and `0011` (binary for `3`), resulting in `0111`, which is `7` in decimal.

These priorities ensure that bitwise NOT is applied first, followed by bitwise shifts, then bitwise AND, followed by bitwise XOR, and finally bitwise OR. Parentheses can be used to change the order of evaluation when necessary, just like with arithmetic operators. Understanding bitwise operators and their priorities is crucial when working with low-level bit manipulation in C.

Code:

print(6|3&9+6)

print(~9+3&6)

output:

7

2

In C, the `+` operator indeed has a higher priority than the `~` (bitwise NOT) operator. Let's break down your code and explain the output:

1. `print(6|3&9+6)`

- The expression inside the parentheses is evaluated as follows:

- `9 + 6 = 15`

- `3 & 15 = 3` (bitwise AND)

- `6 | 3 = 7` (bitwise OR)

- The final result is 7.

2. `print(~9+3&6)`

1st it add 9+3 = 13 then perform ~ operation

- The expression inside the parentheses is evaluated as follows:

- `~13` inverts each bit of the number 13, resulting in -14 in two's complement representation (assuming a typical 32-bit integer).

- `-14 & 6 = 2` (bitwise AND)

- The final result is 2.

Notes:

- The `+` operator has higher precedence than the `~` operator, so the addition is performed before the bitwise NOT operation in the first expression.

- In C, bitwise NOT (`~`) inverts each bit of an integer. So, `~13` turns all the `1` bits to `0` and vice versa, resulting in -14 in two's complement representation.

- Bitwise AND (`&`) combines two numbers bitwise, where a bit in the result is set if the corresponding bits in both operands are set.

- Bitwise OR (`|`) combines two numbers bitwise, where a bit in the result is set if at least one of the corresponding bits in the operands is set.

- Understanding operator precedence is essential for correctly interpreting expressions in C. When in doubt, you can use parentheses to explicitly specify the order of evaluation.