

Ein erneuter Versuch mit Copilot 22.05.2026

Ausgangspunkt von mir :

$$1 + 2 \ll 3 = 17$$

KI:

The expression is **true**:

$$1 + 2 \ll 3 = 17$$

Why?

Operator precedence matters.

1. + is evaluated **before** <<
2. So the expression becomes:
[(1 + 2) << 3]
3. Compute the addition:
[1 + 2 = 3]
4. Left-shift by 3:
[3 << 3 = 3 \cdot 2^3 = 3 \cdot 8 = 24]

Wait — that gives **24**, not 17.

So why do some languages give **17**?

If the expression is parsed as:

[1 + (2 << 3)] Then:

- (2 << 3 = 16)
- (1 + 16 = 17)

Final answer

- **Mathematically**, using standard operator precedence: **24**
- **If << has higher precedence than + in the language you're using**, the result is **17**

Most C-like languages (C, C++, Java, JavaScript) follow this rule:

Shift operators have lower precedence than addition.

So the correct result in those languages is **24**, not 17.

If you got **17**, the language you're using must treat << as **higher precedence** than +.

If you tell me the language, I can confirm exactly how it parses the expression.

SIRO: language C and Pascal

In **both C and Pascal**, the result of:

`1 + 2 << 3`

is **24**, not 17.

Here's the key point:

Operator precedence in C and Pascal

C language

In C, the precedence is:

1. + (addition)
2. << (shift)

So the expression is parsed as:

`(1 + 2) << 3`

Then:

- $1 + 2 = 3$
- $3 \ll 3 = 3 \times 2^3 = 3 \times 8 = 24$

Pascal

Pascal has the **same precedence rule**:

+ is evaluated **before** shl.

So:

`1 + 2 shl 3`

is also:

`(1 + 2) shl 3 = 3 shl 3 = 24`

✓ Final answer for both C and Pascal

The correct result is 24.

If you ever saw **17**, that means the expression was parsed as:

`1 + (2 << 3)`

—but **C and Pascal do NOT** parse it that way.

If you want, I can show you a small C or Pascal program that prints the result.