# C++ Operator Precedence

The following table lists the precedence and associativity of C++ operators. Operators are listed top to bottom, in descending precedence.

| Precedence | Operator | Description | Associativity |
|---|---|---|---|
| **1** | `::` | Scope resolution | Left-to-right |
| **2** | `++ --`<br>`()`<br>`[]`<br>`.`<br>`->` | Suffix/postfix increment and decrement<br>Function call<br>Array subscripting<br>Element selection by reference<br>Element selection through pointer | |
| **3** | `++ --`<br>`+ -`<br>`! ~`<br>`(type)`<br>`*`<br>`&`<br>`sizeof`<br>`new, new[]`<br>`delete, delete[]` | Prefix increment and decrement<br>Unary plus and minus<br>Logical NOT and bitwise NOT<br>Type cast<br>Indirection (dereference)<br>Address-of<br>Size-of<br>Dynamic memory allocation<br>Dynamic memory deallocation | Right-to-left |
| **4** | `.* ->*` | Pointer to member | Left-to-right |
| **5** | `* / %` | Multiplication, division, and remainder | |
| **6** | `+ -` | Addition and subtraction | |
| **7** | `<< >>` | Bitwise left shift and right shift | |
| **8** | `< <=`<br>`> >=` | For relational operators < and ≤ respectively<br>For relational operators > and ≥ respectively | |
| **9** | `== !=` | For relational = and ≠ respectively | |
| **10** | `&` | Bitwise AND | |
| **11** | `^` | Bitwise XOR (exclusive or) | |
| **12** | `\|` | Bitwise OR (inclusive or) | |
| **13** | `&&` | Logical AND | |
| **14** | `\|\|` | Logical OR | |
| **15** | `?:`<br>`=`<br>`+= -=`<br>`*= /= %=`<br>`<<= >>=`<br>`&= ^= \|=` | Ternary conditional<br>Direct assignment (provided by default for C++ classes)<br>Assignment by sum and difference<br>Assignment by product, quotient, and remainder<br>Assignment by bitwise left shift and right shift<br>Assignment by bitwise AND, XOR, and OR | Right-to-left |
| **16** | `throw` | Throw operator (for exceptions) | |
| **17** | `,` | Comma | Left-to-right |

When parsing an expression, an operator which is listed on some row will be bound tighter (as if by parentheses) to its arguments than any operator that is listed on a row further below it. For example, the expressions `std::cout<<a&b` and `*p++` are parsed as `(std::cout<<a)&b` and `*(p++)`, and not as `std::cout<<(a&b)` or `(*p)++`.

Operators that are in the same cell (there may be several rows of operators listed in a cell) are evaluated with the same precedence, in the given direction. For example, the expression `a=b=c` is parsed as `a=(b=c)`, and not as `(a=b)=c` because of right-to-left associativity.

An operator's precedence is unaffected by overloading.

## Notes

The standard itself doesn't specify precedence levels. They are derived from the grammar.

`const_cast`, `static_cast`, `dynamic_cast`, `reinterpret_cast` and `typeid` are not included since they are never ambiguous.

Some of the operators have alternate spellings (e.g., `and` for `&&`, `or` for `||`, `not` for `!`, etc.).

## See also

Order of evaluation of operator arguments at run time.

| Common operators | | | | | | |
|---|---|---|---|---|---|---|
| assignment | increment decrement | arithmetic | logical | comparison | member access | other |
| `a = b`<br>`a = rvalue`<br>`a += b`<br>`a -= b`<br>`a *= b`<br>`a /= b`<br>`a %= b`<br>`a &= b`<br>`a \|= b`<br>`a ^= b`<br>`a <<= b`<br>`a >>= b` | `++a`<br>`--a`<br>`a++`<br>`a--` | `+a`<br>`-a`<br>`a + b`<br>`a - b`<br>`a * b`<br>`a / b`<br>`a % b`<br>`~a`<br>`a & b`<br>`a \| b`<br>`a ^ b`<br>`a << b`<br>`a >> b` | `!a`<br>`a && b`<br>`a \|\| b` | `a == b`<br>`a != b`<br>`a < b`<br>`a > b`<br>`a <= b`<br>`a >= b` | `a[b]`<br>`*a`<br>`&a`<br>`a->b`<br>`a.b`<br>`a->*b`<br>`a.*b` | `a(...)`<br>`a, b`<br>`(type) a`<br>`? :` |
| **Special operators** | | | | | | |
| `static_cast` converts one type to another compatible type<br>`dynamic_cast` converts virtual base class to derived class<br>`const_cast` converts type to compatible type with different cv qualifiers<br>`reinterpret_cast` converts type to incompatible type<br>`new` allocates memory<br>`delete` deallocates memory<br>`sizeof` queries the size of a type<br>`sizeof...` queries the size of a parameter pack (since C++11)<br>`typeid` queries the type information of a type<br>`noexcept` checks if an expression can throw an exception (since C++11)<br>`alignof` queries alignment requirements of a type (since C++11) | | | | | | |