[toc]

# 算术运算符

| 运算符 | 功能 |
| --- | --- |
| + （单目） | 正 |
| - （单目） | 负 |
| * （双目） | 乘法 |
| / | 除法 |
| % | 取模 |
| + （双目） | 加法 |
| - （双目） | 减法 |

node: "单目与双目运算符" 单目运算符（又称一元运算符）指被操作对象只有一个的运算符，而双目运算符（又称二元运算符）的被操作对象有两个。例如 `1 + 2` 中加号就是双目运算符，它有 `1` 和 `2` 两个被操作数。此外 C++ 中还有唯一的一个三目运算符 `?:` 。

算术运算符中有两个单目运算符（正、负）以及五个双目运算符（乘法、除法、取模、加法、减法），其中单目运算符的优先级最高。

其中取模运算符 `%` 意为计算两个整数相除得到的余数，即求余数。

而 `-` 为双目运算符时做减法运算符，如 `2-1` ；为单目运算符时做负值运算符，如 `-1` 。

使用方法如下

```
op=x-y*z
```

得到的 `op` 的运算值遵循数学中加减乘除的优先规律，首先进行优先级高的运算，同优先级自左向右运算，括号提高优先级。

## 算术运算中的类型转换

对于双目算术运算符，当参与运算的两个变量类型相同时，不发生 类型转换 ，运算结果将会用参与运算的变量的类型容纳，否则会发生类型转换，以使两个变量的类型一致。

转换的规则如下：

- 先将 `char` ， `bool` ， `short` 等类型提升至 `int` （或 `unsigned int` ，取决于原类型的符号性）类型；
- 若存在一个变量类型为 `long double` ，会将另一变量转换为 `long double` 类型；
- 否则，若存在一个变量类型为 `double` ，会将另一变量转换为 `double` 类型；
- 否则，若存在一个变量类型为 `float` ，会将另一变量转换为 `float` 类型；
- 否则（即参与运算的两个变量均为整数类型）：
  - 若两个变量符号性一致，则将位宽较小的类型转换为位宽较大的类型；
  - 否则，若无符号变量的位宽不小于带符号变量的位宽，则将带符号数转换为无符号数对应的类型；

    - 否则，若带符号操作数的类型能表示无符号操作数类型的所有值，则将无符号操作数转换为带符号操作数对应的类型；
    - 否则，将带符号数转换为相对应的无符号类型。

例如，对于一个整型（`int`）变量 $x$ 和另一个双精度浮点型（`double`）类型变量 $y$：

- `x/3` 的结果将会是整型；
- `x/3.0` 的结果将会是双精度浮点型；
- `x/y` 的结果将会是双精度浮点型；
- `x*1/3` 的结果将会是整型；
- `x*1.0/3` 的结果将会是双精度浮点型；

## 位运算符

| 运算符 | 功能 |
| --- | --- |
| `~` | 逐位非 |
| `&`（双目） | 逐位与 |
| `\|` | 逐位或 |
| `^` | 逐位异或 |
| `<<` | 逐位左移 |
| `>>` | 逐位右移 |

note: 位运算符的优先级低于普通的算数运算符。

## 自增/自减 运算符

有时我们需要让变量进行增加 1（自增）或者减少 1（自减），这时自增运算符 `++` 和自减运算符 `--` 就派上用场了。

自增/自减运算符可放在变量前或变量后面，在变量前称为前缀，在变量后称为后缀，单独使用时前缀后缀无需特别区别，如果需要用到表达式的值则需注意，具体可看下面的例子。详细情况可参考 引用 介绍的例子部分。

```
i = 100;

op1 = i++;  // op1 = 100, 先 op1 = i, 然后 i = i + 1

i = 100;

op2 = ++i;  // op2 = 101, 先 i = i + 1, 然后赋值 op2

i = 100;

op3 = i--;  // op3 = 100, 先赋值 op3, 然后 i = i - 1

i = 100;
```

```
op4 = --i;  // op4 = 99, 先 i = i - 1, 然后赋值 op4
```

note: `--i--` 未定义行为: 在语言规则中未定义, 根据不同的编译器输出结果可能不同.

## 复合赋值运算符

复合赋值运算符实际上是表达式的缩写形式。

`op=op+2` 可写为 `op+=2`

`op=op-2` 可写为 `op-=2`

`op=op*2` 可写为 `op*=2`

## 比较运算符

| 运算符 | 功能 |
| --- | --- |
| `>` | 大于 |
| `>=` | 大于等于 |
| `<` | 小于 |
| `<=` | 小于等于 |
| `==` | 等于 |
| `!=` | 不等于 |

其中特别需要注意的是要将等于运算符 `==` 和赋值运算符 `=` 区分开来，这在判断语句中尤为重要。

`if(op=1)` 与 `if(op==1)` 看起来类似，但实际功能却相差甚远。第一条语句是在对 op 进行赋值，若赋值为非 0 时为真值，表达式的条件始终是满足的，无法达到判断的作用；而第二条语句才是对 `op` 的值进行判断。

## 逻辑运算符

| 运算符 | 功能 |
| --- | --- |
| `&&` | 逻辑与 |
| `\|\|` | 逻辑或 |
| `!` | 逻辑非 |

```
Result = op1 && op2;  // 当 op1 与 op2 都为真时则 Result 为真

Result = op1 || op2;  // 当 op1 或 op2 其中一个为真时则 Result 为真

Result = !op1;  // 当 op1 为假时则 Result 为真
```

## 逗号运算符

逗号运算符可将多个表达式分隔开来，被分隔开的表达式按从左至右的顺序依次计算，整个表达式的值是最后的表达式的值。逗号表达式的优先级在所有运算符中的优先级是 **最低** 的。

```
exp1, exp2, exp3;  // 最后的值为 exp3 的运算结果。

Result = 1 + 2, 3 + 4, 5 + 6;
//得到 Result 的值为 3 而不是 11，因为赋值运算符 "="
//的优先级比逗号运算符高，先进行了赋值运算才进行逗号运算。

Result = (1 + 2, 3 + 4, 5 + 6);

// 若要让 Result 的值得到逗号运算的结果则应将整个表达式用括号提高优先级，此时
// Result 的值才为 11。
```

## 成员访问运算符

| 运算符 | 功能 |
| --- | --- |
| `[]` | 数组下标 |
| `.` | 对象成员 |
| `&` （单目） | 取地址/获取引用 |
| `*` （单目） | 间接寻址/解引用 |
| `->` | 指针成员 |

这些运算符用来访问对象的成员或者内存，除了最后一个运算符外上述运算符都可被重载。与 `&` ， `*` 和 `->` 相关的内容请阅读 指针 和 引用 教程。这里还省略了两个很少用到的运算符 `.*` 和 `->*` ，其具体用法可以参见 C++ 语言手册 。

```
auto result1 = v[1];  // 获取v中下标为2的对象
auto result2 = p.q;   // 获取p对象的q成员
auto result3 = p -> q;  // 获取p指针指向的对象的q成员，等价于 (*p).q
auto result4 = &v;      // 获取指向v的指针
auto result5 = *v;      // 获取v指针指向的对象
```

## C++ 运算符优先级总表

来自 百度百科 ，有修改。

| 运算符 | 描述 | 例子 | 可重载性 |
| --- | --- | --- | --- |
| **第一级别** | | | |

| 运算符 | 描述 | 例子 | 可重载性 |
| --- | --- | --- | --- |
| :: | 作用域解析符 | Class::age = 2; | 不可重载 |
| **第二级别** | | | |
| () | 函数调用 | isdigit('1') | 可重载 |
| () | 成员初始化 | c_tor(int x, int y) : _x(x), _y(y*10){}; | 可重载 |
| [] | 数组数据获取 | array[4] = 2; | 可重载 |
| -> | 指针型成员调用 | ptr->age = 34; | 可重载 |
| . | 对象型成员调用 | obj.age = 34; | 不可重载 |
| ++ | 后自增运算符 | for (int i = 0; i < 10; i++) cout << i; | 可重载 |
| -- | 后自减运算符 | for (int i = 10; i > 0; i--) cout << i; | 可重载 |
| **第三级别** （从右向左结合） | | | |
| ! | 逻辑取反 | if( !done ) … | 可重载 |
| ~ | 按位取反 | flags = ~flags; | 可重载 |
| ++ | 前自增运算符 | for (i = 0; i < 10; ++i) cout << i; | 可重载 |
| -- | 前自减运算符 | for (i = 10; i > 0; --i) cout << i; | 可重载 |
| - | 负号 | int i = -1; | 可重载 |
| + | 正号 | int i = +1; | 可重载 |
| * | 指针取值 | int data = *intPtr; | 可重载 |
| & | 值取指针 | int *intPtr = &data; | 可重载 |

| 运算符 | 描述 | 例子 | 可重载性 |
| --- | --- | --- | --- |
| new | 动态元素内存分配 | long *pVar = new long; MyClass *ptr = new MyClass(args); | 可重载 |
| new [] | 动态数组内存分配 | long *array = new long[n]; | 可重载 |
| delete | 动态析构元素内存 | delete pVar; | 可重载 |
| delete [] | 动态析构数组内存 | delete [] array; | 可重载 |
| (type) | 强制类型转换 | int i = (int) floatNum; | 可重载 |
| sizeof | 返回类型内存 | int size = sizeof floatNum; int size = sizeof(float); | 不可重载 |
| **第四级别** | | | |
| ->* | 类指针成员引用 | ptr->*var = 24; | 可重载 |
| .* | 类对象成员引用 | obj.*var = 24; | 不可重载 |
| **第五级别** | | | |
| * | 乘法 | int i = 2 * 4; | 可重载 |
| / | 除法 | float f = 10.0 / 3.0; | 可重载 |
| % | 取余数（模运算） | int rem = 4 % 3; | 可重载 |
| **第六级别** | | | |
| + | 加法 | int i = 2 + 3; | 可重载 |
| - | 减法 | int i = 5 - 1; | 可重载 |
| **第七级别** | | | |
| << | 位左移 | int flags = 33 << 1; | 可重载 |
| >> | 位右移 | int flags = 33 >> 1; | 可重载 |

| 运算符 | 描述 | 例子 | 可重载性 |
| --- | --- | --- | --- |
| **第八级别** | | | |
| < | 小于 | if( i < 42 ) ... | 可重载 |
| <= | 小于等于 | if( i <= 42 ) ... | 可重载 |
| > | 大于 | if( i > 42 ) ... | 可重载 |
| >= | 大于等于 | if( i >= 42 ) ... | 可重载 |
| **第九级别** | | | |
| == | 等于 | if( i == 42 ) ... | 可重载 |
| != | 不等于 | if( i != 42 ) ... | 可重载 |
| **第十级别** | | | |
| & | 位与运算 | flags = flags & 42; | 可重载 |
| **第十一级别** | | | |
| ^ | 位异或运算 | flags = flags ^ 42; | 可重载 |
| **第十二级别** | | | |
| \| | 位或运算 | flags = flags \| 42; | 可重载 |
| **第十三级别** | | | |
| && | 逻辑与运算 | if (conditionA && conditionB) ... | 可重载 |
| **第十四级别** | | | |
| \|\| | 逻辑或运算 | if (conditionA \|\| conditionB) ... | 可重载 |
| **第十五级别** （从右向左结合） | | | |
| ? : | 条件运算符 | int i = a > b ? a : b; | 不可重载 |

| 运算符 | 描述 | 例子 | 可重载性 |
| --- | --- | --- | --- |
| = | 赋值 | int a = b; | 可重载 |
| += | 加赋值运算 | a += 3; | 可重载 |
| -= | 减赋值运算 | b -= 4; | 可重载 |
| *= | 乘赋值运算 | a *= 5; | 可重载 |
| /= | 除赋值运算 | a /= 2; | 可重载 |
| %= | 模赋值运算 | a %= 3; | 可重载 |
| &= | 位与赋值运算 | flags &= new_flags; | 可重载 |
| ^= | 位异或赋值运算 | flags ^= new_flags; | 可重载 |
| \|= | 位或赋值运算 | flags \|= new_flags; | 可重载 |
| <<= | 位左移赋值运算 | flags <<= 2; | 可重载 |
| >>= | 位右移赋值运算 | flags >>= 2; | 可重载 |
| **第十六级别**（从右向左结合） | | | |
| throw | 异常抛出 | throw EClass("Message"); | 不可重载 |
| **第十七级别** | | | |
| , | 逗号分隔符 | for (i = 0, j = 0; i < 10; i++, j++) ... | 可重载 |

note: 说多不多, 说少不少, 如果忘了, 就加括号!