

88 Argument Evaluation Order in C++

本节讲的是是一些C++的提示、技巧之类的东西，下一节就是移动语义了！

1. 参数求值顺序

```
void Function(int a, int b, int c) {}  
int a = 2;  
Function(GetResult(), ++a, a--);
```

这样有众多参数的函数中，参数求值的顺序是什么？

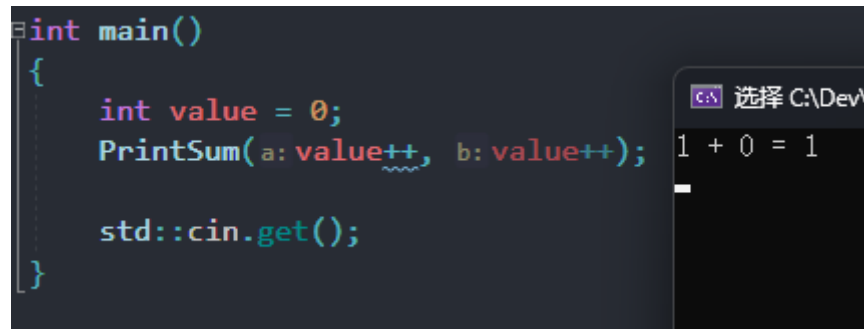
Cherno认为如果你不知道这个答案是正常且合理的，因为它涉及到对编译器如何编译代码的深入了解，你并不会在日常编程中考虑到这个“普遍的”问题。

用一个例子来考虑这个问题，请给出下面示例中PrintSum函数的运行结果：

```
void PrintSum(int a, int b)  
{  
    std::cout << a << " + " << b << " = " << (a + b) << std::endl;  
}  
  
int main()  
{  
    int value = 0;  
    PrintSum(value++, value++);  
  
    std::cin.get();  
}
```

value++是一个后增的操作符，意思是初始值应该先传入再递增，所以我认为答案是1,0

但这个问题的答案应该是“未定义行为”（*undefined behavior*），C++标准并没有真正定义这种情况下应该发生什么。这里的“未定义行为”就是说它会根据便一起的不同而变化，完全依赖于C++编译器将代码转换成机器码的实际实现。



The screenshot shows a C++ IDE with the following code in the main function:

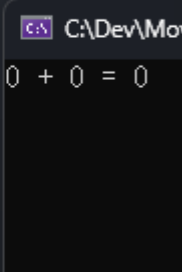
```
int main()  
{  
    int value = 0;  
    PrintSum(a: value++, b: value++);  
  
    std::cin.get();  
}
```

The output window shows the result of the function call: `1 + 0 = 1`. The output is displayed in a window titled "选择 C:\Dev\".

Debug模式

```
int main()
{
    int value = 0;
    PrintSum(a: value++, b: value++);

    std::cin.get();
}
```



Release模式

改成前增：

```
PrintSum(++value, ++value);

// Release: 2 + 2 = 4
// Debug   : 2 + 2 = 4
```

所以Release模式下的后增运算到底发生了什么呢？

原因是这种情况下，编译器实际上被允许并行地计算出这些是什么，不必先按照指定的顺序对这些参数求值，然后查看它们是什么（有点像`constant folding`，也就是预先计算常量表达式的结果，如：`int a = 1 × 2`，不会在`runtime`计算，而是`compile-time`优化为`int a = 2`）

这在C++17标准以前是可以的，因为C++17标准中加入了一个新规则：后缀表达式必须在别的表达式之前被计算所以运行结果会发生变化：

```
PrintSum(value++, value++);

// C++ 14 :
// Release: 0 + 0 = 0
// Debug   : 1 + 0 = 1

// C++ 17 :
// Release: 1 + 0 = 1
// Debug   : 1 + 0 = 1
```

不过这里的顺序仍然没有定义

用gcc和C++20标准运行一下：

← → ↻ wandbox.org

Wandbox

Language

C++

Compiler

gcc 12.2.0

Options

☒ Warnings

☐ Optimization

☐ Verbose

Boost 1.81.0

C++2a(GNU)

no pedantic

Raw compiler options

```
1 #include <iostream>
2
3 void PrintSum(int a, int b)
4 {
5     std::cout << a << " + " << b << " = " << (a + b) << std::endl;
6 }
7
8 int main()
9 {
10     int value = 0;
11     PrintSum(value++, value++);
12
13     std::cin.get();
14 }
15
16
```

\$ g++ prog.cc -Wall -Wextra -I/opt/wandbox/boost-1.81.0-gcc-12.2.0/include -std=gnu++2a

Run Share

```
prog.cc: In function 'int main()':
prog.cc:11:32: warning: operation on 'value' may be undefined [-Wsequence-point]
11 |     PrintSum(value++, value++);
    |               ~~~~~^~
prog.cc:11:32: warning: operation on 'value' may be undefined [-Wsequence-point]
1 + 0 = 1
```

Exit Code: 0

截图看出相比于VS的MSVC编译器，这里gcc至少告诉我们这个value操作可能是未定义的，也得到了结果

而clang的编译结果是相反的：

Compiler
clang 12.0.1

Options
☒ Warnings
☐ Optimization
☐ Verbose
Boost 1.78.0
C++2a(GNU)
no pedantic
Raw compiler options

```
1 #include <iostream>
2
3 void PrintSum(int a, int b)
4 {
5     std::cout << a << " + " << b << " = " << (a + b) << std::endl;
6 }
7
8 int main()
9 {
10     int value = 0;
11     PrintSum(value++, value++);
12
13     std::cin.get();
14 }
15
16
```

\$ clang++ prog.cc -Wall -Wextra -I/opt/wandbox/boost-1.78.0-clang-12.0.1/include -std=gnu++2a

RunShare

prog.cc:11:16: warning: multiple unsequenced modifications to 'value' [-Wunsequenced]
 PrintSum(value++, value++);
 ^ ^
1 warning generated.
0 + 1 = 1

所以如果你在工作面试或者编程测试中遇到了这个问题，正确答案是“未定义”，因为C++实际上没有提供一个规范，一个定义，来说明在这种情况下应该发生什么，形参或实参应该按照什么顺序求值。但如果你提到C++17后不能同时计算就会加分了。但是这个顺序没有在规范中定义，这意味着你在技术上无法知道计算顺序是什么。