C/C++ 语言中表达式的求值

裘宗燕
北京大学数学学院信息科学系
本文基本内容发表于《编程高手》杂志 2004 年第 12 期

经常可以在一些讨论组里看到下面的提问：“谁知道下面 C 语句给 n 赋什么值？”

```
m = 1; n = m+++m++;
```

最近有位不相识的朋友发 email 给我，问为什么在某个 C++系统里，下面表达式打印出两个 4，而不是 4 和 5：

```
a = 4; cout << a++ << a;
```

C++ 不是规定 << 操作左结合吗？是 C++ 书上写错了，还是这个系统的实现有问题？

要弄清这些，需要理解的一个问题是：如果程序里某处修改了一个变量（通过赋值、增量/减量操作等），什么时候从该变量能够取到新值？有人可能说，“这算什么问题！我修改了变量，再从这个变量取值，取到的当然是修改后的值!”其实事情并不这么简单。

C/C++ 语言是“基于表达式的语言”，所有计算（包括赋值）都在表达式里完成。“x = 1;”就是表达式“x = 1”后加表示语句结束的分号。要弄清程序的意义，首先要理解表达式的意义，也就是：1）表达式所确定的计算过程；2）它对环境（可以把环境看作当时可用的所有变量）的影响。如果一个表达式（或子表达式）只计算出值而不改变环境，我们就说它是引用透明的，这种表达式早算晚算对其他计算没有影响（不改变计算的环境。当然，它的值可能受到其他计算的影响）。如果一个表达式不仅算出一个值，还修改了环境，就说这个表达式有副作用（因为它多做了额外的事）。a++ 就是有副作用的表达式。这些说法也适用于其他语言里的类似问题。

现在问题变成：如果 C/C++ 程序里的某个表达式（部分）有副作用，这种副作用何时才能实际体现到使用中？为使问题更清楚，我们假定程序里有代码片段“...a[i]++ ... a[j] ...”，假定当时 i 与 j 的值恰好相等（a[i] 和 a[j] 正好引用同一数组元素）；假定 a[i]++ 确实在 a[j] 之前计算；再假定其间没有其他修改 a[i] 的动作。在这些假定下，a[i]++ 对 a[i] 的修改能反映到 a[j] 的求值中吗？注意：由于 i 与 j 相等的问题无法静态判定，在目标代码里，这两个数组元素访问（对内存的访问）必然通过两段独立代码完成。现代计算机的计算都在寄存器里做，问题现在变成：在取 a[j] 值的代码执行之前，a[i] 更新的值是否已经被（从寄存器）保存到内存？如果了解语言在这方面的规定，这个问题的答案就清楚了。

程序语言通常都规定了执行中变量修改的最晚实现时刻（称为顺序点、序点或执行点）。程序执行中存在一系列顺序点（时刻），语言保证一旦执行到达一个顺序点，在此之前发生的所有修改（副作用）都必须实现（必须反应到随后对同一存储位置的访问中），在此之后的所有修改都还没有发生。在顺序点之间则没有任何保证。对 C/C++ 语言这类允许表达式有副作用的语言，顺序点的概念特别重要。

现在上面问题的回答已经很清楚了：如果在 a[i]++ 和 a[j] 之间存在一个顺序点，那么就能保证 a[j] 将取得修改之后的值；否则就不能保证。

C/C++语言定义（语言的参考手册）明确定义了顺序点的概念。顺序点位于：

1. 每个完整表达式结束时。完整表达式包括变量初始化表达式，表达式语句，return 语句的表达式，以及条件、循环和 switch 语句的控制表达式（for 头部有三个控制表达式）；
2. 运算符 &&、||、?: 和逗号运算符的第一个运算对象计算之后；
3. 函数调用中对所有实际参数和函数名表达式（需要调用的函数也可能通过表达式描述）的求值完成之后（进入函数体之前）。

假设时刻t_i和t_{i+1}是前后相继的两个顺序点，到了t_{i+1}，任何C/C++ 系统（VC、BC等都是C/C++系统）都必须实现t_i之后发生的所有副作用。当然它们也可以不等到时刻t_{i+1}，完全可以选择在时段 $[t, t_{i+1}]$ 之间的任何时刻实现在此期间出现的副作用，因为C/C++ 语言允许这些选择。

前面讨论中假定了 a[i]++ 在 a[i] 之前做。在一个程序片段里 a[i]++ 究竟是否先做，还与它所在的表达式确定的计算过程有关。我们都熟悉 C/C++ 语言有关优先级、结合性和括号的规定，而出现多个运算对象时的计算顺序却常常被人们忽略。看下面例子：

```
(a + b) * (c + d)        fun(a++, b, a+5)
```

这里“*”的两个运算对象中哪个先算？fun 及其三个参数按什么顺序计算？对第一个表达式，采用任何计算顺序都没关系，因为其中的子表达式都是引用透明的。第二个例子里的实参表达式出现了副作用，计算顺序就非常重要了。少数语言明确规定了运算对象的计算顺序（Java 规定从左到右），C/C++ 则有意不予规定，既没有规定大多数二元运算的两个对象的计算顺序（除了&&、|| 和 ,），也没有规定函数参数和被调函数的计算顺序。在计算第二个表达式时，首先按照某种顺序算 fun、a++、b 和 a+5，之后是顺序点，而后进入函数执行。

不少书籍在这些问题上有错（包括一些很流行的书）。例如说 C/C++ 先算左边（或右边），或者说某个 C/C++ 系统先计算某一边。这些说法都是错误的！一个 C/C++ 系统可以永远先算左边或永远先算右边，也可以有时先算左边有时先算右边，或在同一表达式里有时先算左边有时先算右边。不同系统可能采用不同的顺序（因为都符合语言标准）；同一系统的不同版本完全可以采用不同方式；同一版本在不同优化方式下，在不同位置都可能采用不同顺序。因为这些做法都符合语言规范。在这里还要注意顺序点的问题：即使某一边的表达式先算了，其副作用也可能没有反映到内存，因此对另一边的计算没有影响。

回到前面的例子：“谁知道下面 C 语句给 n 赋什么值？”

```
m = 1; n = m++ +m++;
```

正确回答是：不知道！语言没有规定它应该算出什么，结果完全依赖具体系统在具体上下文中的具体处理。其中牵涉到运算对象的求值顺序和变量修改的实现时刻问题。对于：

```
cout << a++ << a;
```

我们知道它是

```
(cout.operator <<(a++)).operator << (a);
```

的简写。先看外层函数调用，这里需要算出所用函数（由加下划线的一段得到），还需要计算 a 的值。语言没有规定哪个先算。如果真的先算函数，这一计算中出现了另一次函数调用，在被调函数体执行前有一个顺序点，那时 a++的副作用就会实现。如果是先算参数，求出 a 的值 4，而后计算函数时的副作用当然不会改变它（这种情况下输出两个 4）。当然，这些只

是假设，实际应该说的是：这种东西根本不该写，讨论其效果没有意义。

有人可能说，为什么人们设计 C/C++时不把顺序规定清楚，免去这些麻烦？C/C++ 语言的做法完全是有意而为，其目的就是允许编译器采用任何求值顺序，使编译器在优化中可以根据需要调整实现表达式求值的指令序列，以得到效率更高的代码。像 Java 那样严格规定表达式的求值顺序和效果，不仅限制了语言的实现方式，还要求更频繁的内存访问（以实现副作用），这些可能带来可观的效率损失。应该说，在这个问题上，C/C++和 Java 的选择都贯彻了它们各自的设计原则，各有所获（C/C++ 潜在的效率，Java 更清晰的程序行为），当然也都有所失。还应该指出，大部分程序设计语言实际上都采用了类似 C/C++的规定。

讨论了这么多，应该得到什么结论呢？C/C++ 语言的规定告诉我们，任何依赖于特定计算顺序、依赖于在顺序点之间实现修改效果的表达式，其结果都没有保证。程序设计中应该贯彻的规则是：如果在任何“完整表达式”（形成一段由顺序点结束的计算）里存在对同一“变量”的多个引用，那么表达式里就不应该出现对这一“变量”的副作用。否则就不能保证得到预期结果。注意：这里的问题不是在某个系统里试一试的问题，因为我们不可能试验所有可能的表达式组合形式以及所有可能的上下文。这里讨论的是语言，而不是某个实现。总而言之，绝不要写这种表达式，否则我们或早或晚会某种环境中遇到麻烦。

后记：去年参加一个学术会议，看到有同行写文章讨论某个 C 系统里表达式究竟按什么顺序求值，并总结出一些“规律”。从讨论中了解到某“程序员水平考试”出了这类题目。这使我感到很不安。今年给一个教师学习班讲课，发现许多专业课教师也对这一基本问题也不甚明了，更觉得问题确实严重。因此整理出这篇短文供大家参考。

后后记：4 年多过去了，许多新的和老的教科书仍然在不厌其烦地讨论在 C 语言里原本并无意义的问题（如本文所指出的）。希望学习和使用 C 语言的人不要陷入其中。——2009.2