C语言中的序列点和副作用

iteye_17686 2012-10-16 08:57:00 112 收藏 2

文章标签： c/c++

C 语言中“术语副作用（side effect）是指对数据对象或者文件的修改。”例如，以下语句 var = 99;
的副作用是把 var 的值修改成 99。对表达式求值也可能产生副作用，例如：
se = 100
对这个表达式求值所产生的副作用就是 se 的值被修改成 100。
序列点（sequence point）是指程序运行中的一个特殊的时间点，在该点之前的所有副作用已经结束，并且后续的副作用还没发生。
C 语句结束标志——分号（;）是序列点。也就是说，C 语句中由赋值、自增或者自减等引起的副作用在分号之前必须结束。我们以后会说到一些包含序列点的运算符。任何完整表达式（full expression）运算结束的那个时间点也是序列点。所谓完整表达式，就是说这个表达式不是子表达式。而所谓的子表达式，则是指表达式中的表达式。例如：
f = ++e % 3
这整个表达式就是一个完整表达式。这个表达式中的 ++e、3 和 ++e % 3 都是它的子表达式。
有了序列点的概念，我们下面来分析一下一个很常见的错误：
int x = 1, y;
y = x++ + x++;
这里 y = x++ + x++ 是完整表达式，而 x++ 是它的子表达式。这个完整表达式运算结束的那一点是一个序列点，int x = 1, y; 中的 ; 也是一个序列点。也就是说，x++ + x++ 位于两个序列点之间。标准规定，在两个序列点之间，一个对象所保存的值最多只能被修改一次。但是我们清楚可以看到，上面这个例子中，x 的值在两个序列点之间被修改了两次。这显然是错误的！这段代码在不同的编译器上编译可能会导致 y 的值有所不同。比较常见的结果是 y 的值最后被修改为 2 或者 3。在此，我不打算就这个问题作更深入的分析，各位只要记住这是错误的，别这么用就可以了。有兴趣的话，可以看看以下列出的相关资料。

C 语言标准对副作用和序列点的定义如下：
Accessing a volatile object, modifying an object, modifying a file, or calling a function that does any of those operations are all side effects, which are changes in the state of the execution environment. Evaluation of an expression may produce side effects. At certain specified points in the execution sequence called sequence points, all side effects of previous evaluations shall be complete and no side effects of subsequent evaluations shall have taken place.
翻译如下：
访问易变对象，修改对象或文件，或者调用包含这些操作的函数都是副作用，它们都会改变执行环境的状态。计算表达式也会引起副作用。执行序列中某些特定的点被称为序列点。在序列点上，该点之前所有运算的副作用都应该结束，并且后继运算的副作用还没发生。

让我们来看看下面的代码：
int i=7; printf("%d\n", i++ * i++);
你认为会返回什么？56？no。正确答案是返回 49？很多人会问为什么？难道不该打印出56吗？在ccfaq中有非常详尽的解释，根本原因在于c中的序列点。
请注意，尽管后缀自加和后缀自减操作符 ++ 和 — 在输出其旧值之后才会执行运算，但这里的“之后”常常被误解。没有任何保证确保自增或自减会在输出变量原值之后和对表达式的其它部分进行计算之前立即进行。也不能保证变量的更新会在表达式 “完成” (按照 ANSI C 的术语, 在下一个”序列点”之前) 之前的某个时刻进行。本例中, 编译器选择使用变量的旧值相乘以后再对二者进行自增运算。只有到达一个序列点之后，自增运算才能保证真正被执行。
包含多个不确定的副作用的代码的行为总是被认为未定义。简单而言, “多个不确定副作用”是指在同一个表达式中使用导致同一对象修改两次或修改以后又被引用的自增，自减和赋值操作符的任何组合。这是一个粗略的定义。) 甚至都不要试图探究这些东西在你的编译器中是如何实现的 (这与许多 C 教科书上的弱智练习正好相反);正如 K&R 明智地指出，”如果你不知道它们在不同的机器上如何实现, 这样的无知可能恰恰会有助于保护你”。

那么，所谓的序列点是什么意思呢？
序列点是一个时间点(在整个表达式全部计算完毕之后或在 ||、 &&、 ? : 或逗号 运算符处, 或在函数调用之前)，此刻尘埃落定，所有的副作用都已确保结束。 ANSI/ISO C 标准这样描述：
在上一个和下一个序列点之间，一个对象所保存的值至多只能被表达式的计算修改一次。而且前一个值只能用于决定将要保存的值。
第二句话比较费解。它说在一个表达式中如果某个对象需要写入， 则在同一表达式中对该对象的访问应该只局限于直接用于计算将要写入的值。这条规则有效地限制了只有能确保在修改之前才访问变量的表达式为合法。
例如 i = i+1 合法，而 a[i] = i++ 则非法。为什么这样的代码：a[i] = i++; 不能工作？子表达式 i++ 有一个副作用 — 它会改变 i 的值 — 由于 i 在同一表达式的其它地方被引用，这会导致无定义的结果，无从判断该引用(左边的 a[i] 中)是旧值还是新值。那么，对于 a[i] = i++; 我们不知道 a[] 的哪一个分量会被改写，但 i 的确会增加 1，对吗？
不一定！如果一个表达式和程序变得未定义，则它的所有方面都会变成未定义。
为什么&& 和 || 运算符可以产生序列点呢？这些运算符在此处有一个特殊的例外：如果左边的子表达式决定最终结果 (即，真对于 || 和假对于 &&)，则右边的子表达式不会计算。因此，从左至右的计算可以确保，对逗号表达式也是如此。而且，所有这些运算符 (包括 ? :) 都会引入一个额外的内部序列点。