C 专家编程 1.9 节 20 页

有时候必须非常专注地阅读 ANSI C 标准才能找到某个问题的答案。一位销售工程师把下面这段代码作为测试例发给 Sun 的编译器小组。

```
1 foo(const char **p) { }
2
3 main(int argc, char **argv)
4 {
5         foo(argv);
6 }
```

这里引用实参过程，我们看作用实参给形参进行赋值

如果编译这段代码，编译器会发出一条警告信息：

```
line 5: warning: argument is incompatible with prototype
```

（第 5 行：警告：参数与原型不匹配）。

提交代码的工程师想知道为什么会产生这条警告信息，也想知道 ANSI C 标准的哪一部分讲述了这方面的内容。他认为，实参 char* s 与形参 const char *p 应该是相容的，标准库中所有的字符串处理函数都是这样的。那么，为什么实参 char **argv 与形参 const char **p 实际上不能相容呢？

看起来它们都是一个指针的指针，但为何不能利用实参对形参赋值呢？

答案是肯定的，它们并不相容。要回答这个问题颇费心机，如果研究一下获得这个答案的整个过程，会比仅仅知道结论更有意义。对这个问题的分析是由 Sun 的其中一位“语言律师”[1]进行的，其过程如下：

在 ANSI C 标准第 6.3.2.2 节中讲述约束条件的小节中有这么一句话：

每个实参都应该具有自己的类型，这样它的值就可以赋值给与它所对应的形参类型的对象（该对象的类型不能含有限定符）。

这就是说参数传递过程类似于赋值。

所以，除非一个 const char **类型的对象可以赋值给一个类型为 char **的值，否则肯定会产生一条诊断信息。要想知道这个赋值是否合法，就请回顾标准中有关简单赋值的部分，它位于第 6.3.16.1 节，描述了下列约束条件：

1.注意，这里重点强调“指针指向的类型”
2.即左边限定符要大于等于右边限定符数量

要使上述的赋值形式合法，必须满足下列条件之一：

两个操作数都是指向有限定符或无限定符的相容类型的指针，左边指针所指向的类型必须具有右边指针所指向类型的全部限定符。

正是这个条件，使得函数调用中实参 char*能够与形参 const char*匹配（在 C 标准库中，所有的字符串处理函数就是这样的）。它之所以合法，是因为在下面的代码中：

```
char *cp;
const char *ccp;
ccp = cp;
```

- 左操作数是一个指向有 const 限定符的 char 的指针。
- 右操作数是一个指向没有限定符的 char 的指针。
- char 类型与 char 类型是相容的，左操作数所指向的类型具有右操作数所指向类型的限定符（无），再加上自身的限定符(const)。

注意，反过来就不能进行赋值。如果不信，试试下面的代码：

```
cp = ccp;      /* 结果产生编译警告 */
```

标准第 6.3.16.1 节有没有说 char **实参与 const char *形参是相容的？没有。

标准第 6.1.2.5 节中讲述实例的部分声称：

const float *类型并不是一个有限定符的类型——它的类型是“指向一个具有 const 限定符的 float 类型的指针”，也就是说 const 限定符是修饰指针所指向的类型，而不是指针本身。

类似地，const char **也是一个没有限定符的指针类型。它的类型是“指向有 const 限定符的 char 类型的指针的指针”。

由于 char **和 const char **都是没有限定符的指针类型，但它们所指向的类型不一样（前者指向 char *，后者指向 const char *），因此它们是不相容的。因此，类型为 char**的实参与类型为 const char**的形参是不相容的，违反了标准第 6.3.2.2 节所规定的约束条件，编译器必然会产生一条诊断信息。

用这种方式理解这个要点有一定困难。可以用下面这个方法进行理解：

- 左操作数的类型是 FOO2，它是一个指向 FOO 的指针，而 FOO 是一个没有限定符的指针，它指向一个带有 const 限定符的 char 类型，而且……
- 右操作数的类型是 BAZ2，它是一个指向 BAZ 的指针，而 BAZ 是一个没有限定符的指针，它指向一个没有限定符的字符类型。

FOO 和 BAZ 所指向的类型是相容的，而且它们本身都没有限定符，所以符合标准的约束条件，两者之间进行赋值是合法的。但 FOO2 和 BAZ2 之间的关系又有不同，由于相容性是不能传递的，FOO 和 BAZ 所指向的类型相容并不表示 FOO2 和 BAZ2 所指向的类型也相容，所以虽然 FOO2 和 BAZ2 都没有限定符，但它们之间不能进行赋值。也就是说，它们都是不带限定符的指针，但它们所指向的对象是不相容的，所以它们之间不能进行赋值，也就不能分别作为函数的形参和实参。但是，这个约束条件很令人恼火，也很容易让用户混淆。所以，这种赋值方法目前在基于 Cfront 的 C++翻译器中是合法的（虽然这在将来可能会改变）。