参考《C 专家编程》 3.3 节 63 页

理解 C 语言声明的优先级规则

A 声明从它的名字开始读取，然后按照优先级顺序依次读取。

B 优先级从高到低依次是：

B. 1 声明中被括号括起来的那部分

B. 2 后缀操作符：

括号（）表示这是一个函数，而

方括号[]表示这是一个数组。

B. 3 前缀操作符：星号*表示“指向...的指针”。

C 如果 const 和（或）volatile 关键字的后面紧跟类型说明符（如 int, long 等），那么它作用于类型说明符。在其他情况下，const 和（或）volatile 关键字作用于它左边紧邻的指针星号。

注意运算符优先级：
1.[]高于*，先数组后指针；
2.函数 () 高于*，先函数后指针。

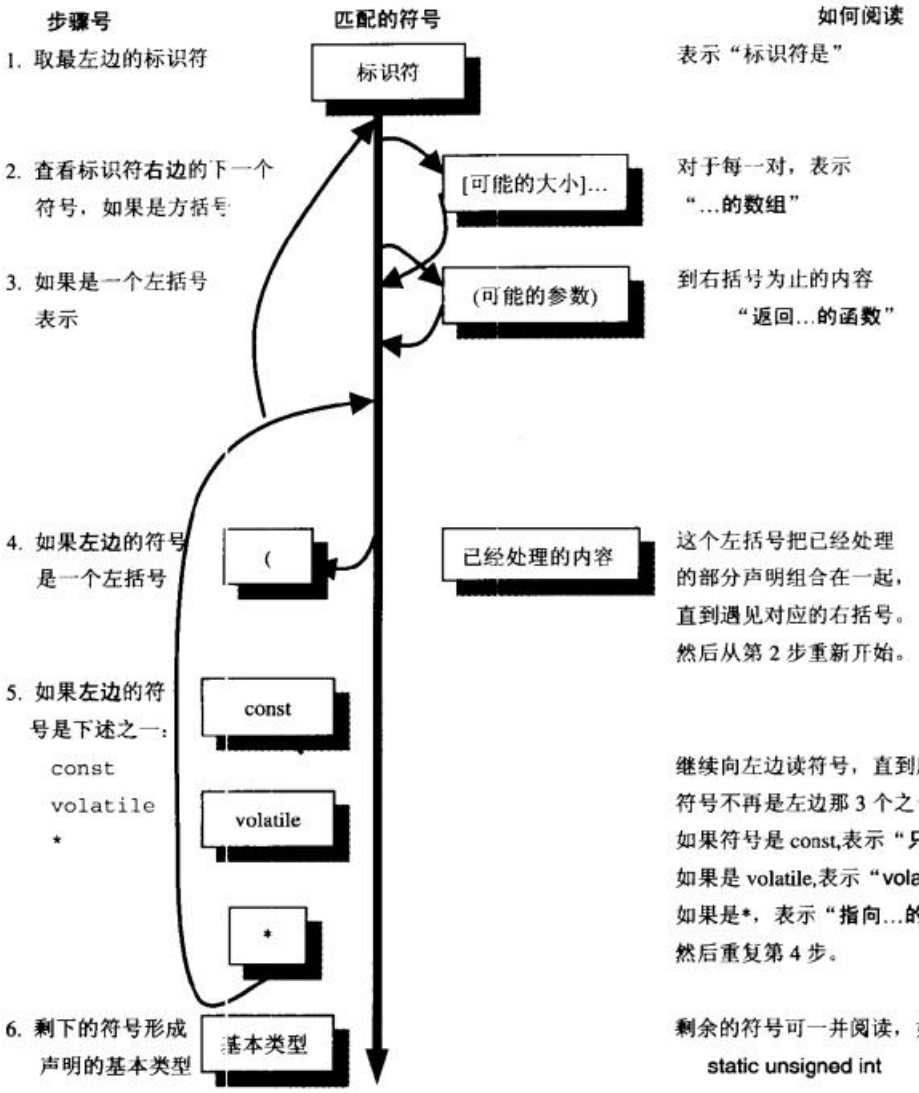
用优先级规则分析 C 语言声明一例：

```
char * const *(*next)();
```

表 3-3 用优先级规则解决一个声明

适用规则	解 释
A	首先，看变量名“next”，并注意到它直接被括号所括住
B.1	所以先把括号里的东西作为一个整体，得出“next 是一个指向...的指针”
B	然后考虑括号外面的东西，在星号前缀和括号后缀之间作出选择
B.2	B.2 规则告诉我们优先级较高的是右边的函数括号，所以得出“next 是一个函数指针，指向一个返回...的函数”
B.3	然后，处理前缀“*”，得出指针所指的内容
C	最后，把“char * const”解释为指向字符的常量指针

把上述分析结果加以概括，这个声明表示“next 是一个指针，它指向一个函数，该函数返回另一个指针，该指针指向一个类型为 char 的常量指针”，大功告成。优先级规则浓缩了所有的规则，如果你更喜欢看上去直观一些的方法，请看图 3-1。



C 语言中的声明读起来并没有固定的方向，一会儿从左读到右，一会儿又从右读到左，真不知该用一个怎样的词来描述这个情况。一开始，我们从左边开始向右寻找，直到找到第一个标识符。当声明中的某个符号与图中所示匹配时，便把它从声明中处理掉，以后不再考虑。在具体的每一步骤上，我们首先查看右边的符号，然后再看左边。

当所有的符号都被处理完毕后，便宣告大功告成。

让我们试一些例子，用图中所示方法来分析声明。假如我们想知道本章开头所举的那个代码例子的意思：

```
char * const *(*next)();
```

在分析这个声明时，需要逐渐把已经处理过的片段“去掉”，这样便能知道还需要分析多少内容。再次提醒，记住 const 表示“只读”，并不能因为它的意思是常量就认为它表示的就是常量。

表 3-4　　分析一个 C 语言声明的步骤

剩余的声明 （从最左边的标识符开始）	所采取的下一步骤	结　果
char * const *(**next**) ( );	第 1 步	表示“next 是...”
char * const *(*　　) ( );	第 2、3 步	不匹配，转到下一步，表示“next 是...”
char * const *(*　　) ( );	第 4 步	不匹配，转到下一步
char * const *(*　　) ( );	第 5 步	与星号匹配，表示“指向...的指针”，转第 4 步
char * const *(　　) ( );	第 4 步	“（”和“）”匹配，转到第 2 步
char * const *　　( );	第 2 步	不匹配，转到下一步。
char * const *　　( );	第 3 步	表示“返回...的函数”
char * const *　　;	第 4 步	匹配，转到下一步
char * const *　　;	第 5 步	表示“指向...的指针”
char * **const**　　;	第 5 步	表示“只读的...”
char *　　;	第 5 步	表示“指向...的的指针”
char　　;	第 6 步	表示“char”

拼在一起，读作：

“next 是一个指向函数的指针，该函数返回另一个指针，该指针指向一个只读的指向 char 的指针”，大功告成。

现在让我们试一个更复杂的例子。

```
char *(* c[10])(int **p);
```

请按照上面那个例子的步骤进行分析。具体步骤在本章的最后给出，可以自己先试一下，然后对照一下答案。