<<你必须知道的495个C语言问题>>

图书基本信息

书名：<<你必须知道的495个C语言问题>>

13位ISBN编号：9787115194329

10位ISBN编号：7115194327

出版时间：2009-2

出版时间：人民邮电出版社

作者：Steve Summit

页数：260

译者：孙云,朱群英

前言

1979年的某段时间，我听到很多人在谈论C这个当时还挺新的语言和那本刚刚推出的书。
我买了一本BrianKemighan和DenisRitchie写的ThecProgrammingLanguage（也称K&R），但它在我的书架上空等了好一阵子，因为当时我并不急着需要它（况且我那时候还是一个余暇无多的大一新生）。
后来证明这本书买得很幸运，因为当我最后拿起它以后，就再也没有放下了：从那以后，我就一直在用C语言编程。

1983年我结识了新闻组net．lang．c，这（以及它的后继者comp．1ang．c）是一个绝佳的地方，你可以学习C语言的方方面面，发现别人关于C语言的各种疑问，认识到你可能根本还没有掌握关于C语言的一切。
C语言尽管表面上很简单，但也还有一些并不显而易见的方面，有些问题不断有人问起。
本书根据我从1990年5月开始在comp．1ang．c上发布的常见问题（FAQ）列表收集了这样的一些问题，并提供了答案。

然而我得声明，这本书并不是对C语言的批评或诽谤。
用户在使用时遇到困难，很容易迁怒于语言（或其他任何工具）或者要求正确设计的工具“应该”防止用户的误用。
因此看到书中提及的各种误用以后，很容易将这样的书看作试图显示C语言的先天不足的长篇控诉。
这实在是远悖我的本意。

如果我不认为C语言是一门伟大的语言，或者没有在这种语言的编程中获得那么多的乐趣，那我永远也学不到足够的关于C语言的知识来写出本书，而且也不会试图写出本书来让别人更爱用r语言。
我很喜欢C语言，我教c的课并花时间参与网上讨论的原因之一，就是希望发现这门语言（或者说编程本身）在哪些方面比较难学，让人不易高效地编程。
本书展示了我认识到的部分内容，这些问题毫无疑问就是人们遇到麻烦最多的，而答案则经过多年的反复修正，就是为了消除人们的麻烦。

如果这些答案中有任何错误，那么读者一定会遇到麻烦。
尽管审稿人和我都尽力去除所有的错误，但从一部手稿中根除最后一个错误，就跟从程序中去掉最后一个bug一样困难。
通过出版社转交或发往我的E．mail地址的任何修正和建议我都感激不尽。
同时我也对任何错误的第一个发现者按惯例提供$1．00的报酬。
如果你能够访问因特网，你可以在问题20．47提到的ftp和lhttp网址中找到一份勘误表（和错误发现者的积分表）。

<<你必须知道的495个C语言问题>>

内容概要

本书以问答的形式组织内容，讨论了学习或使用C语言的过程中经常遇到的一些问题。书中列出了C用户经常问的400多个经典问题，涵盖了初始化、数组、指针、字符串、内存分配、库函数、C预处理器等各个方面的主题，并分别给出了解答，而且结合代码示例阐明要点。

本书结构清晰，讲解透彻，是各高校相关专业C语言课程很好的教学参考书，也是各层次C程序员的优秀实践指南。

作者简介

Steve Summit，著名的C语言专家。
Usenet C FAQ的创始人和维护者，有近30年的C编程经验。
毕业于麻省理工学院。
他曾在华盛顿大学教授C语言课程多年。
除本书外，他还与人合著了C Unleashed一书。

书籍目录

3.3 对于代码int i=3; i=i++; 不同编译器给出不同的i值，有的为3，有的为4，哪个是正确的？
3.4 有这样一个巧妙的表达式：a^= b^= a^= b; 它不需要临时变量就可以交换a和b的值。
3.5 可否用显式括号来强制执行我所需要的计算顺序并控制相关的副作用？
就算括号不行，操作符优先级是否能够控制计算顺序呢？
3.6 可是&&和||操作符呢？
我看到过类似while（（c = getchar（））！= EOF && c != \n）的代码3.7 是否可以安全地认为，一旦&&和||左边的表达式已经决定了整个表达式的结果，则右边的表达式不会被求值？
3.8 为什么表达式printf（"%d %d"， f1（）， f2（））; 先调用了f2？
我觉得逗号表达式应该确保从左到右的求值顺序。
3.9 怎样才能理解复杂表达式并避免写出未定义的表达式？
“ 序列点 ”是什么？
3.10 在a[i] = i++;中，如果不关心a[]的哪一个分量会被写入，这段代码就没有问题，i也的确会增加1，对吗？
3.11 人们总是说i=i++的行为是未定义的。
可我刚刚在一个ANSI编译器上尝试过，其结果正如我所期望的。
3.12 我不想学习那些复杂的规则，怎样才能避免这些未定义的求值顺序问题呢？
其他的表达式问题3.13 ++i和i++有什么区别？
3.14 如果我不使用表达式的值，那我应该用i++还是++i来做自增呢？
3.15 我要检查一个数是不是在另外两个数之间，为什么if（a b c）不行？
3.16 为什么如下的代码不对？
int a=1000， b=1000; long int c=a * b;3.17 为什么下面的代码总是给出0？
double degC， degF; degC= 5.0 / 9 *（degF - 32）;3.18 需要根据条件把一个复杂的表达式赋给两个变量中的一个。
可以用下面这样的代码吗？
（（condition） ？
a ： b）= complicated_expression;3.19 我有些代码包含这样的表达式。
a ？
b=c ： d 有些编译器可以接受，有些却不能。
为什么？
保护规则3.20 “ semantics of ‘ ’ change in ANSI C ”的警告是什么意思？
3.21 “ 无符号保护 ”和“ 值保护 ”规则的区别在哪里？
第4章 指针基本的指针应用4.1 指针到底有什么好处？
4.2 我想声明一个指针并为它分配一些空间，但却不行。
这些代码有什么问题呢？
char *p; *p =malloc（10）;4.3 *p++自增p还是p所指向的变量？
指针操作4.4 我用指针操作int数组的时候遇到了麻烦。
4.5 我有一个char *型指针碰巧指向一些int型变量，我想跳过它们。
为什么（（int *）p）++; 这样的代码不行？
4.6 为什么不能对void *指针进行算术操作？
4.7 我有些解析外部结构的代码，但是它却崩溃了，显示出了“ unaligned access ”（未对齐的访问）的信息。
这是什么意思？
作为函数参数的指针4.8 我有个函数，它应该接受并初始化一个指针：void f（int *ip）{ static int dummy = 5; ip = &dummy;}但是当我如下调用时：int *ip; f（ip）; 调用者的指针没有任何变化。
4.9 能否用void ** 通用指针作为参数，使函数模拟按引用传递参数？
484.10 我有一个函数extern intf（int *）;，它接受指向int型的指针。
我怎样用引用方式传入一个常数？

调用f（&5）;似乎不行。
4.11 C语言可以“ 按引用传参 ”吗？
其他指针问题4.12 我看到了用指针调用函数的不同语法形式。
到底怎么回事？
4.13 通用指针类型是什么？
当我把函数指针赋向void *类型的时候，编译通不过。
4.14 怎样在整型和指针之间进行转换？
能否暂时把整数放入指针变量中，或者相反？
4.15 我怎样把一个int变量转换为char *型？
我试了类型转换，但是不行。
第5章 空指针空指针和空指针常量5.1 臭名昭著的空指针到底是什么？
5.2 怎样在程序里获得一个空指针？
5.3 用缩写的指针比较“ if（p） ”检查空指针是否有效？
如果空指针的内部表达不是0会怎样？
NULL 宏5.4 NULL是什么，它是怎么定义的？
5.5 在使用非零位模式作为空指针的内部表示的机器上，NULL 是如何定义的？
5.6 如果NULL定义成#define NULL（（char *）0），不就可以向函数传入不加转换的NULL 了吗？
5.7 我的编译器提供的头文件中定义的NULL为0L。
为什么？
5.8 NULL可以合法地用作函数指针吗？
5.9 如果NULL和0作为空指针常量是等价的，那我到底该用哪一个呢？
5.10 但是如果NULL的值改变了，比如在使用非零内部空指针的机器上，用NULL（而不是0）不是更好吗？
5.11 我曾经使用过一个编译器，不使用NULL就不能编译。
5.12 我用预处理宏#define Nullptr（type）（type *）0帮助创建正确类型的空指针。
回顾5.13 这有点奇怪：NULL可以确保是0，但空（null）指针却不一定？
5.14 为什么有那么多关于空指针的疑惑？
为什么这些问题如此频繁地出现？
5.15 有没有什么简单点儿的办法理解所有这些与空指针有关的东西呢？
5.16 考虑到有关空指针的所有这些困惑，要求它们的内部表示都必须为0不是更简单吗？
5.17 说真的，真有机器用非零空指针吗，或者不同类型用不同的表示？
地址0上到底有什么？
5.18 运行时的整数值0转换为指针以后一定是空指针吗？
5.19 如何访问位于机器地址0处的中断向量？
如果我将指针值设为0，编译器可能会自动将它转换为非零的空指针内部表示。
5.20 运行时的“ null pointer assignment ”错误是什么意思？
应该怎样捕捉它？
第6章 数组和指针数组和指针的基本关系6.1 我在一个源文件中定义了char a[6]，在另一个源文件中声明了extern char *a。
为什么不行？
6.2 可是我听说char a[]和char *a是等价的。
是这样的吗？
6.3 那么，在C语言中“ 指针和数组等价 ”到底是什么意思？
6.4 既然它们这么不同，那为什么作为函数形参的数组和指针声明可以互换呢？
数组不能被赋值6.5 为什么不能这样向数组赋值？
extern char *getpass（）; char str[10]; str=getpass（"Enter password：");6.6 既然不能向数组赋值，那这段代码为什么可以呢？

int f（char str[]）{ if（str[0] == \0） str="none";}6.7 如果你不能给它赋值，那么数组如何能成为左值呢？
回顾6.8 现实地讲，数组和指针的区别是什么？
6.9 有人跟我讲，数组不过是常指针。
这样讲准确吗？
6.10 我还是很困惑。
到底指针是一种数组，还是数组是一种指针？
6.11 我看到一些“ 搞笑 ”的代码，包含5["abcdef"]这样的“ 表达式 ”。
这为什么是合法的C语言表达式呢？
数组的指针6.12 既然数组引用会退化为指针，如果array是数组，那么array和&array又有什么区别呢？
6.13 如何声明一个数组的指针？
动态数组分配6.14 如何在运行时设定数组的大小？
怎样才能避免固定大小的数组？
6.15 我如何声明大小和传入的数组一样的局部数组？
6.16 如何动态分配多维数组？
6.17 有个很好的窍门，如果我这样写：int realarray[10]; int *array = &realarray[-1]; 我就可以把“ array ”当作下标从1 开始的数组。
函数和多维数组6.18 当我向一个接受指针的指针的函数传入二维数组的时候，编译器报错了。
6.19 我怎样编写接受编译时宽度未知的二维数组的函数？
6.20 我怎样在函数参数传递时混用静态和动态多维数组？
数组的大小6.21 当数组是函数的参数时，为什么sizeof不能正确报告数组的大小？
6.22 如何在一个文件中判断声明为extern的数组的大小（例如，数组定义和大小在另一个文件中）？
sizeof操作符似乎不行。
6.23 sizeof返回的大小是以字节计算的，怎样才能判断数组中有多少个元素呢？
第7章 内存分配第8章　字符和字符串第9章　布尔表达式和变量第10章　C预处理器第11章　ANSI/ISO标准C第12章　标准输入输出库第13章　库函数第14章　浮点运算第15章　可变参数列表第16　章 奇怪的问题第17章　风格第18章　工具和资源第19章　系统依赖第20章　杂项术语表参考文献

章节摘录

第1章　声明和初始化　　C语言的声明语法本身实际上就是一种小的编程语言。
一个声明包含如下几个部分（但是并非都必不可少）：存储类型、基本类型、类型限定词和最终的声明符（也可能包含初始化列表）。
每个声明符不仅声明一个新的标识符，同时也表明标识符是数组、指针、函数还是其他任意的复杂组合。
基本的思想是让声明符模仿标识符的最终用法。
（问题1.21将会更加详细地讨论这种“声明模仿使用”的关系！
）　　基本类型　　让一些程序员惊奇的是，尽管C语言是一种相当低级的语言，但它的类型体系仍然略显抽象。
语言本身并没有精确定义基本类型的大小和表示法。
　　问：我该如何决定使用哪种整数类型？
　　答：如果可能用到很大的数值（大于32 767或小于一32 767），就使用lon9型。
否则，如果空间很重要（例如有很大的数组或很多的结构），就使用short型。
除此之外，就用int型。
如果定义明确的溢出特征很重要而负值无关紧要，或者希望在操作二进制位和字节时避免符号扩展的问题，请使用对应的unsigned类型。
（但是，在表达式中混用有符号和无符号值的时候，要特别注意。
参见问题3.21。
）　　尽管字符类型（尤其是unsigned char型）可以当成“小”整数使用，但这样做有时候很麻烦，不值得。
编译器需要生成额外的代码来进行char型和int型之间的转换（导致目标代码量增大），而且不可预知的符号扩展也会带来一堆麻烦。
（使用unsigned char会有所帮助。
类似的问题参见问题12.1。
）　　在决定使用float型还是double型时也有类似的空间／时间权衡。
（很多编译器在表达式求值的时候仍然把所有的float型转换为double型进行运算）。
但如果一个变量的地址确定且必须为特定的类型时，以上规则就不再适用。
　　很多时候，人们错误地认为C语言类型的大小都有精确的定义。
事实上，能够确保的只有如下几点：

媒体关注与评论

“本书是Summit以及C FAQ在线列表的许多参与者多年心血的结晶，是C语言界最为珍贵的财富之一。
我向所有C语言程序员推荐本书。
”. ——Francis Glassborow，著名C／C++专家，ACCU(C／C++用户协会)前主席 “本书清晰地阐明了Kernighan与Ritchie的The C Programming Language一书中许多简略的地方，而且精彩地总结了C语言编程实践，强烈推荐!” ——Yechiel M．Kimchi，以色列理工学院

编辑推荐

全球C语言程序员集体智慧的结晶　　Amazon全五星图书　　权威解答495个最常遇到的C语言问题　　C是一门简洁精妙的语言，掌握基本语法容易，真正能够自如运用，就不那么简单了。
你难免会遇到各种各样的问题，有些可能让你百思不得其解，甚至翻遍图书馆，也找不到问题的答案。
《你必须知道的495个C语言问题》的出版填补了这一空白。
书中内容是世界各地的C语言用户多年来在新闻组comp.1ang.c中讨论的成果。
作者在网络版CFAQ列表的基础上进行了大幅度的扩充和丰富，结合代码示例，权威而且详细深入地解答了实际学习和工作中最常遇到的495个C语言问题，涵盖了初始化、数组、指针、字符串、内存分配、库函数、C预处理器等各个方面的主题。
许多知识点的阐述都是其他资料中所没有的，弥足珍贵。
涵盖C99标准　　“本书是Summit以及C FAQ在线列表的许多参与者多年心血的结晶，是C语言界最为珍贵的财富之一。
我向所有C语言程序员推荐本书。
”.　　——Francis Glassborow，著名C／C++专家，ACCU（C／C++用户协会）前主席　　“本书清晰地阐明了Kernighan与Ritchie的The C Programming Language一书中许多简略的地方，而且精彩地总结了C语言编程实践，强烈推荐！
”　　——Yechiel M.Kimchi，以色列理工学院

版权说明