

10余年开发经验的资深C语言专家全面从C语法和C11标准两大方面深入探讨编写高质量C代码的技巧、禁忌和最佳实践

Writing Solid Code
125 Suggestions to Improve Your C Program

编写高质量代码

改善C程序代码的 125个建议

马伟 著



机械工业出版社
China Machine Press

Writing Solid Code
125 Suggestions to Improve Your C Program

编写高质量代码

改善C程序代码的 125个建议

马伟 著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

编写高质量代码：改善 C 程序代码的 125 个建议 / 马伟著. —北京：机械工业出版社，2016.1

(Effective 系列丛书)

ISBN 978-7-111-52434-2

I. 编… II. 马… III. C 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字 (2015) 第 313259 号

编写高质量代码：改善 C 程序代码的 125 个建议

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：姜 影 高婧雅

责任校对：殷 虹

印 刷：三河市宏图印务有限公司印刷

版 次：2016 年 1 月第 1 版第 1 次印刷

开 本：186mm×240mm 1/16

印 张：29.25

书 号：ISBN 978-7-111-52434-2

定 价：89.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88379426 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

为什么写作本书

众所周知，C 语言是一门既具有高级语言特点，又有汇编语言特点的通用计算机编程语言，无论是操作系统（如 Microsoft Windows、Mac OS X、Linux 和 UNIX 等）、嵌入式系统与普通应用软件，还是目前流行的移动智能设备开发，随处都可以看见它依然矫健的身影。它能够轻松地应用于各类层次的开发中，从设备驱动程序和操作系统组件到大规模应用程序，它都能够很好地胜任。毋庸置疑，它是二十几年来使用最为广泛、生命力最强的编程语言，它的设计思想也影响了众多后来的编程语言，例如 C++、Objective-C、Java、C# 等。

尽管 C 语言有着悠久的历史和广泛的使用场景，但它依旧让大部分计算机编程人员望而生畏，相信绝大多数读者也还停留在“入门者”这个阶段。所谓“入门者”指的是已经可以简单使用 C 语言编写普通应用程序，但是却不明白如何编写高质量代码的人。面对这样的实际情况，在准备编写本书之前，一连串的问题深深地映入笔者的脑海：到底什么样的编程书籍才能够帮助“入门者”快速进阶？面对市面上众多的优秀 C 语言编程书籍，编写本书的价值何在？怎样的内容才能够与众不同？

带着这一连串的问题，笔者开始回顾自己这些年的开发生涯，发现如下几类问题经常困扰“入门者”：

- ❑ 基础数据类型问题：如数据取值范围、整数溢出与回绕、浮点数精度、数据类型转换的范围检查等。
- ❑ 数组与指针问题：指针与地址、野指针、空（null）指针、NULL 指针、void 指针、多级指针、指针函数与函数指针，以及数组越界与缓冲区溢出等。
- ❑ 内存管理问题：内存分配、内存释放、内存越界与内存泄漏等。
- ❑ 字符与字符串问题：串拷贝与内存拷贝，内存重叠与溢出，字符串查找等。

□ 高效设计问题：表达式设计、算法设计与函数设计，内联函数与宏的取舍等。

□ 其他杂项问题：信号处理、文件系统、断言与异常处理、内嵌汇编的使用等。

如果你同样也苦于处理这些问题，或者对这些问题模棱两可，那么本书正是为你所准备的。本书为普遍存在于初级与中级开发者脑海中的那些问题给出了经验性的解决方案。全书分为 15 章，通过 125 个建议深度剖析 C 语言程序设计中的常见性问题，并给出经验性的解决方案。除此之外，为了使读者能够尽量做到“知其所以然”，本书重点阐述了一些尖锐的问题，如 IEEE 754 浮点数、指针与数组、越界与溢出等问题。当然，这些经验和心得的积累并非我一人之力，“我只不过是站在巨人的肩膀上而已”。因此，在撰写本书的过程中也参考了大量的资料，如 www.securecoding.cert.org 的《SEI CERT C Coding Standard》、ISO/IEC 9899:1990、ISO/IEC 9899:1999 与 ISO/IEC 9899:201x 标准文档等。

如何阅读本书

本书适合那些有一定 C 语言基础并希望快速提升程序设计能力的初级与中级程序员。因此，本书并不会阐述 C 语言中的一些基础概念，而是将 C 语言编程过程中可能遇到的疑问或者障碍进行一一列举与剖析，并给出了经验性解决方案与建议。

如果你是一位有一定 C 语言编程基础的初中级读者，本书就是为你量身打造的。你可以逐章进行系统性学习，并结合我们提供的源码动手实践，巩固所学的知识。书中的大多数建议实战性很强，要完全理解其中的奥妙，请果断地放弃 `printf` 函数，多调试一下程序，编程高手都是调试出来的；如果你是一位编程经验非常丰富的高级读者，那么可以将书中的大部分经验与自己的一些经验进行融合，从而获得更多提高与升华。

资源及勘误

通常情况下，一个问题的解决方案往往不止一种，你可能会不同意本书中的一些观点，甚至强烈反对。同时，尽管笔者在本书的写作过程中非常认真与努力，但由于水平有限，书中难免存在错误和不足之处，恳请批评指正。如果你对本书有什么意见、问题或想法，欢迎使用下面的邮箱通知笔者，笔者将不胜感激。当然，也可以通过微信（sc-mawei）与笔者取得联系，共同进行技术交流。

Email: madengwei@hotmail.com

特别鸣谢

最后，要感谢那些所有帮助过笔者的人，没有他们的帮助与付出，这本书很难顺利完

成。尤其要感谢下面这些人：

首先，机械工业出版社的杨福川与姜影为本书的整体策划、审阅和出版做了大量的工作，与他们的合作是非常愉快的。同时，由于写作过程漫长，难免令笔者情绪波动，是他们给了我一如既往的支持与鼓励，当我想要放弃的时候，是他们的敦促让我对写作时刻保持着热情，坚持完成本书。也正因为他们对本书的不断要求，才使得本书的结构更加系统化，内容更加深刻，语言更加简单易懂。

其次，要感谢家人的支持。为了编写本书，笔者投入了大量的时间和精力，牺牲了许多可以陪家人的周末和节假日。

最后，要感谢那些曾经为本书的编写提过意见的朋友，感谢他们对本书的默默支持。

马 伟

目 录 *Contents*

前 言

第 1 章 数据, 程序设计之根本	1
建议 1: 认识 ANSI C	1
建议 2: 防止整数类型产生回绕与溢出	6
建议 2-1: char 类型变量的值应该限制在 signed char 与 unsigned char 的交集范围内	11
建议 2-2: 使用显式声明为 signed char 或 unsigned char 的类型来执行算术运算	11
建议 2-3: 使用 rsize_t 或 size_t 类型来表示一个对象所占用空间的整数值单位	13
建议 2-4: 禁止把 size_t 类型和它所代表的真实类型混用	16
建议 2-5: 小心使用无符号类型带来的陷阱	16
建议 2-6: 防止无符号整数回绕	19
建议 2-7: 防止有符号整数溢出	24
建议 3: 尽量少使用浮点类型	28
建议 3-1: 了解 IEEE 754 浮点数	29
建议 3-2: 避免使用浮点数进行精确计算	39
建议 3-3: 使用分数来精确表达浮点数	43
建议 3-4: 避免直接在浮点数中使用“=”操作符做相等判断	47
建议 3-5: 避免使用浮点数作为循环计数器	50
建议 3-6: 尽量将浮点运算中的整数转换为浮点数	51
建议 4: 数据类型转换必须做范围检查	52
建议 4-1: 整数转换为新类型时必须做范围检查	53
建议 4-2: 浮点数转换为新类型时必须做范围检查	56

建议 5: 使用有严格定义的数据类型	57
建议 6: 使用 typedef 来定义类型的新别名	61
建议 6-1: 掌握 typedef 的 4 种应用形式	61
建议 6-2: 小心使用 typedef 带来的陷阱	65
建议 6-3: typedef 不同于 #define	65
建议 7: 变量声明应该力求简洁	66
建议 7-1: 尽量不要在一个声明中声明超过一个的变量	67
建议 7-2: 避免在嵌套的代码块之间使用相同的变量名	68
建议 8: 正确地选择变量的存储类型	68
建议 8-1: 定义局部变量时应该省略 auto 关键字	69
建议 8-2: 慎用 extern 声明外部变量	70
建议 8-3: 不要混淆 static 变量的作用	72
建议 8-4: 尽量少使用 register 变量	75
建议 9: 尽量不要在可重入函数中使用静态 (或全局) 变量	76
建议 10: 尽量少使用全局变量	78
建议 11: 尽量使用 const 声明值不会改变的变量	78
 第 2 章 保持严谨的程序设计, 一切从表达式开始做起	81
建议 12: 尽量减少使用除法运算与求模运算	81
建议 12-1: 用倒数相乘来实现除法运算	82
建议 12-2: 使用牛顿迭代法求除数的倒数	84
建议 12-3: 用减法运算来实现整数除法运算	86
建议 12-4: 用移位运算实现乘除法运算	86
建议 12-5: 尽量将浮点除法转化为相应的整数除法运算	87
建议 13: 保证除法和求模运算不会导致除零错误	87
建议 14: 适当地使用位操作来提高计算效率	88
建议 14-1: 尽量避免对未知的有符号数执行位操作	89
建议 14-2: 在右移中合理地选择 0 或符号位来填充空出的位	90
建议 14-3: 移位的数量必须大于等于 0 且小于操作数的位数	90
建议 14-4: 尽量避免在同一个数据上执行位操作与算术运算	91
建议 15: 避免操作符混淆	92

建议 15-1: 避免 “=” 与 “==” 混淆	92
建议 15-2: 避免 “ ” 与 “ ” 混淆	94
建议 15-3: 避免 “&” 与 “&&” 混淆	95
建议 16: 表达式的设计应该兼顾效率与可读性	95
建议 16-1: 尽量使用复合赋值运算符	95
建议 16-2: 尽量避免编写多用途的、太复杂的复合表达式	97
建议 16-3: 尽量避免在表达式中使用默认的优先级	98
 第 3 章 程序控制语句应该保持简洁高效	101
建议 17: if 语句应该尽量保持简洁, 减少嵌套的层数	101
建议 17-1: 先处理正常情况, 再处理异常情况	101
建议 17-2: 避免 “悬挂” 的 else	102
建议 17-3: 避免在 if/else 语句后面添加分号 “;”	105
建议 17-4: 对深层嵌套的 if 语句进行重构	106
建议 18: 谨慎 0 值比较	108
建议 18-1: 避免布尔型与 0 或 1 进行比较	108
建议 18-2: 整型变量应该直接与 0 进行比较	109
建议 18-3: 避免浮点变量用 “==” 或 “!=” 与 0 进行比较	109
建议 18-4: 指针变量应该用 “==” 或 “!=” 与 NULL 进行比较	111
建议 19: 避免使用嵌套的 “?:”	111
建议 20: 正确使用 for 循环	114
建议 20-1: 尽量使循环控制变量的取值采用半开半闭区间写法	114
建议 20-2: 尽量使循环体内工作量达到最小化	115
建议 20-3: 避免在循环体内修改循环变量	115
建议 20-4: 尽量使逻辑判断语句置于循环语句外层	116
建议 20-5: 尽量将多重循环中最长的循环放在最内层, 最短的循环放在最外层	117
建议 20-6: 尽量将循环嵌套控制在 3 层以内	117
建议 21: 适当地使用并行代码来优化 for 循环	117
建议 22: 谨慎使用 do/while 与 while 循环	118
建议 22-1: 无限循环优先选用 for(;;), 而不是 while(1)	118
建议 22-2: 优先使用 for 循环替代 do/while 与 while 循环	119

建议 23: 正确地使用 switch 语句	120
建议 23-1: 不要忘记在 case 语句的结尾添加 break 语句	120
建议 23-2: 不要忘记在 switch 语句的结尾添加 default 语句	122
建议 23-3: 不要为了使用 case 语句而刻意构造一个变量	122
建议 23-4: 尽量将长的 switch 语句转换为嵌套的 switch 语句	123
建议 24: 选择合理的 case 语句排序方法	124
建议 24-1: 尽量按照字母或数字顺序来排列各条 case 语句	124
建议 24-2: 尽量将情况正常的 case 语句排在最前面	125
建议 24-3: 尽量根据发生频率来排列各条 case 语句	125
建议 25: 尽量避免使用 goto 语句	125
建议 26: 区别 continue 与 break 语句	127
 第 4 章 函数同样需要保持简洁高效	 129
建议 27: 理解函数声明	129
建议 28: 理解函数原型	131
建议 29: 尽量使函数的功能单一	132
建议 30: 避免把没有关联的语句放在一个函数中	135
建议 31: 函数的抽象级别应该在同一层次	136
建议 32: 尽可能为简单功能编写函数	137
建议 33: 避免多段代码重复做同一件事情	138
建议 34: 尽量避免编写不可重入函数	140
建议 34-1: 避免在函数中使用 static 局部变量	140
建议 34-2: 避免函数返回指向静态数据的指针	140
建议 34-3: 避免调用任何不可重入函数	142
建议 34-4: 对于全局变量, 应通过互斥信号量 (即 P、V 操作) 或者中断机制等方法 来保证函数的线程安全	143
建议 34-5: 理解可重入函数与线程安全函数之间的关系	144
建议 35: 尽量避免设计多参数函数	145
建议 35-1: 没有参数的函数必须使用 void 填充	145
建议 35-2: 尽量避免在非调度函数中使用控制参数	147
建议 35-3: 避免将函数的参数作为工作变量	148

建议 35-4: 使用 <code>const</code> 防止指针类型的输入参数在函数体内被意外修改	149
建议 36: 没有返回值的函数应声明为 <code>void</code> 类型	149
建议 37: 确保函数体的“入口”与“出口”安全性	150
建议 37-1: 尽量在函数体入口处对参数做有效性检查	150
建议 37-2: 尽量在函数体出口处对 <code>return</code> 语句做安全性检查	151
建议 38: 在调用函数时, 必须对返回值进行判断, 同时对错误的返回值还要有 相应的错误处理	152
建议 39: 尽量减少函数本身或者函数间的递归调用	153
建议 40: 尽量使用 <code>inline</code> 内联函数来替代 <code>#define</code> 宏	154

第 5 章 不会使用指针的程序员是不合格的

建议 41: 理解指针变量的存储实质	157
建议 42: 指针变量必须初始化	162
建议 43: 区别 “ <code>int *p = NULL</code> ” 和 “ <code>*p = NULL</code> ”	163
建议 44: 理解空 (<code>null</code>) 指针与 <code>NULL</code> 指针	164
建议 44-1: 区别空 (<code>null</code>) 指针与 <code>NULL</code> 指针的概念	164
建议 44-2: 用 <code>NULL</code> 指针终止对递归数据结构的间接引用	166
建议 44-3: 用 <code>NULL</code> 指针作函数调用失败时的返回值	169
建议 44-4: 用 <code>NULL</code> 指针作警戒值	170
建议 44-5: 避免对 <code>NULL</code> 指针进行解引用	170
建议 45: 谨慎使用 <code>void</code> 指针	171
建议 45-1: 避免对 <code>void</code> 指针进行算术操作	172
建议 45-2: 如果函数的参数可以是任意类型指针, 应该将其参数声明为 <code>void *</code>	173
建议 46: 避免使用指针的长度确定它所指向类型的长度	175
建议 47: 避免把指针转换为对齐要求更严格的指针类型	176
建议 48: 避免将一种类型的操作符应用于另一种不兼容的类型	177
建议 49: 谨慎指针与整数之间的转换	180
建议 50: 区别 “ <code>const int *p</code> ” 与 “ <code>int *const p</code> ”	180
建议 51: 深入理解函数参数的传递方式	183
建议 51-1: 理解函数参数的传递过程	183
建议 51-2: 掌握函数的参数传递方式	188

建议 51-3: 如果函数的参数是指针, 避免用该指针去申请动态内存	191
建议 51-4: 尽量避免使用可变参数	195
第 6 章 数组并非指针	199
建议 52: 理解数组的存储实质	199
建议 52-1: 理解数组的存储布局	199
建议 52-2: 理解 &a[0] 和 &a 的区别	203
建议 52-3: 理解数组名 a 作为右值和左值的区别	203
建议 53: 避免数组越界	204
建议 53-1: 尽量显式地指定数组的边界	207
建议 53-2: 对数组做越界检查, 确保索引值位于合法的范围之内	209
建议 53-3: 获取数组的长度时不要对指针应用 sizeof 操作符	210
建议 54: 数组并非指针	213
建议 55: 理解数组与指针的可交换性	217
建议 56: 禁止将一个指向非数组对象的指针加上或减去一个整数	219
建议 57: 禁止对两个并不指向同一个数组的指针进行相减或比较	220
建议 58: 若结果值并不引用合法的数组元素, 不要将指针加上或减去一个整数	220
建议 59: 细说缓冲区溢出	220
建议 60: 区别指针数组和数组指针	226
建议 61: 深入理解数组参数	227
第 7 章 结构、位域和枚举	231
建议 62: 结构体的设计要遵循简单、单一原则	231
建议 62-1: 尽量使结构体的功能单一	232
建议 62-2: 尽量减小结构体间关系的复杂度	234
建议 62-3: 尽量使结构体中元素的个数适中	235
建议 62-4: 合理划分与改进结构体以提高空间效率	236
建议 63: 合理利用结构体内存对齐原理来提高程序效率	237
建议 64: 结构体的长度不一定等于各个成员的长度之和	249
建议 65: 避免在结构体之间执行逐字节比较	250
建议 66: 谨慎使用位域	251

建议 67: 谨慎使用枚举	252
建议 68: 禁止在位域成员上调用 <code>offsetof</code> 宏	254
建议 69: 深入理解结构体数组和结构体指针	255
第 8 章 字符与字符串	260
建议 70: 不要忽视字符串的 <code>null ('\\0')</code> 结尾符	260
建议 70-1: 正确认识字符数组和字符串	261
建议 70-2: 字符数组必须能够同时容纳字符数据和 <code>null</code> 结尾符	262
建议 70-3: 谨慎字符数组的初始化	263
建议 71: 尽量使用 <code>const</code> 指针来引用字符串常量	264
建议 72: 区别 <code>strlen</code> 函数与 <code>sizeof</code> 运算符	264
建议 73: 在使用不受限制的字符串函数时, 必须保证结果字符串不会溢出内存	265
建议 73-1: 避免字符串拷贝发生溢出	266
建议 73-2: 区别串拷贝 <code>strcpy</code> 与内存拷贝 <code>memcpy</code>	270
建议 73-3: 避免 <code>strcpy</code> 与 <code>memcpy</code> 函数内存重叠	273
建议 73-4: 区别字符串比较与内存比较	278
建议 73-5: 避免 <code>strcat</code> 函数发生内存重叠与溢出	283
建议 74: 谨慎 <code>strtok</code> 函数的不可重入性	287
建议 75: 掌握字符串查找技术	292
建议 75-1: 使用 <code>strchr</code> 与 <code>strrchr</code> 函数查找单个字符	292
建议 75-2: 使用 <code>strpbrk</code> 函数查找多个字符	293
建议 75-3: 使用 <code>strstr</code> 函数查找一个子串	294
建议 75-4: 区别 <code>strspn</code> 与 <code>strcspn</code> 函数	295
第 9 章 文件系统	298
建议 76: 谨慎使用 <code>printf</code> 和 <code>scanf</code> 函数	299
建议 77: 谨慎文件打开操作	308
建议 77-1: 正确指定 <code>fopen</code> 的 <code>mode</code> 参数	309
建议 77-2: 必须检查 <code>fopen</code> 函数的返回值	310
建议 77-3: 尽量避免重复打开已经被打开的文件	311

建议 77-4: 区别 <code>fopen</code> 与 <code>fopen_s</code> 函数	312
建议 77-5: 区别 <code>fopen</code> 与 <code>freopen</code> 函数	313
建议 78: 文件操作完成后必须关闭	313
建议 79: 正确理解 EOF 宏	314
建议 80: 尽量使用 <code>fEOF</code> 和 <code>ferror</code> 检测文件结束和错误	316
建议 81: 尽量使用 <code>fgetc</code> 替换 <code>getc</code> 函数	319
建议 82: 尽量使用 <code>fputs</code> 替换 <code>puts</code> 函数	321
建议 83: 合理选择单个字符读写函数	323
建议 84: 区别格式化读写函数	324
建议 84-1: 区别 <code>printf/scanf</code> 、 <code>fprintf/fscanf</code> 和 <code>sprintf/sscanf</code>	325
建议 84-2: 尽量使用 <code>snprintf</code> 替代 <code>sprintf</code> 函数	327
建议 84-3: 区别 <code>vprintf/vscanf</code> 、 <code>vfprintf/vfscanf</code> 、 <code>vsprintf/vsscanf</code> 和 <code>vsnprintf</code>	328
建议 85: 尽量使用 <code>fread</code> 与 <code>fwrite</code> 函数来读写二进制文件	330
建议 86: 尽量使用 <code>fseek</code> 替换 <code>rewind</code> 函数	332
建议 87: 尽量使用 <code>setvbuf</code> 替换 <code>setbuf</code> 函数	334
建议 88: 谨慎 <code>remove</code> 函数删除已打开的文件	336
建议 89: 谨慎 <code>rename</code> 函数重命名已经存在的文件	337
 第 10 章 预处理器	 338
建议 90: 谨慎宏定义	338
建议 90-1: 在使用宏定义表达式时必须使用完备的括号	339
建议 90-2: 尽量消除宏的副作用	340
建议 90-3: 避免使用宏创建一种“新语言”	342
建议 91: 合理地选择函数与宏	343
建议 92: 尽量使用内联函数代替宏	345
建议 93: 掌握预定义宏	350
建议 94: 谨慎使用“ <code>#include</code> ”	353
建议 94-1: 区别“ <code>#include <filename.h></code> ”与“ <code>#include "filename.h"</code> ”	354
建议 94-2: 必须保证头文件名称的唯一性	355
建议 95: 掌握条件编译指令	355

建议 95-1: 使用 “#ifndef/#define/#endif” 防止头文件被重复引用	355
建议 95-2: 使用条件编译指令实现源代码的部分编译	357
建议 95-3: 妙用 “defined”	358
建议 96: 尽量避免在一个函数块中单独使用 “#define” 或 “#undef”	359
第 11 章 断言与异常处理	361
建议 97: 谨慎使用断言	361
建议 97-1: 尽量利用断言来提高代码的可测试性	362
建议 97-2: 尽量在函数中使用断言来检查参数的合法性	367
建议 97-3: 避免在断言表达式中使用改变环境的语句	368
建议 97-4: 避免使用断言去检查程序错误	369
建议 97-5: 尽量在防错性程序设计中 使用断言来进行错误报警	370
建议 97-6: 用断言保证没有定义的特性或功能不被使用	372
建议 97-7: 谨慎使用断言对程序开发环境中的假设进行检查	373
建议 98: 谨慎使用 <code>errno</code>	374
建议 98-1: 调用 <code>errno</code> 之前必须先将其清零	375
建议 98-2: 避免重定义 <code>errno</code>	377
建议 98-3: 避免使用 <code>errno</code> 检查文件流错误	379
建议 99: 谨慎使用函数的返回值来标志函数是否执行成功	380
建议 100: 尽量避免使用 <code>goto</code> 进行出错跳转	380
建议 101: 尽量避免使用 <code>setjmp</code> 与 <code>longjmp</code> 组合	381
第 12 章 内存管理	384
建议 102: 浅谈程序的内存结构	384
建议 103: 浅谈堆和栈	389
建议 104: 避免错误分配内存	396
建议 104-1: 对内存分配函数的返回值必须进行检查	397
建议 104-2: 内存资源的分配与释放应该限定在同一模块或者同一抽象层内进行	398
建议 104-3: 必须对内存分配函数的返回指针进行强制类型转换	400
建议 104-4: 确保指针指向一块合法的内存	401
建议 104-5: 确保为对象分配足够的内存空间	402

建议 104-6: 禁止执行零长度的内存分配	405
建议 104-7: 避免大型的堆栈分配	405
建议 104-8: 避免内存分配成功, 但并未初始化	407
建议 105: 确保安全释放内存	407
建议 105-1: malloc 等内存分配函数与 free 必须配对使用	407
建议 105-2: 在 free 之后必须为指针赋一个新值	409
建议 106: 避免内存越界	411
建议 106-1: 避免数组越界	412
建议 106-2: 避免 sprintf、vsprintf、strcpy、strcat 与 gets 越界	413
建议 106-3: 避免 memcpy 与 memset 函数长度越界	413
建议 106-4: 避免忽略字符串最后的 '\0' 字符而导致的越界	413
建议 107: 避免内存泄漏	415
建议 108: 避免 calloc 参数相乘的值超过 size_t 表示的范围	417
第 13 章 信号处理	418
建议 109: 理解信号	418
建议 110: 尽量使用 sigaction 替代 signal	423
建议 111: 避免在信号处理函数内部访问或修改共享对象	428
建议 112: 避免以递归方式调用 raise 函数	429
第 14 章 了解 C11 标准	432
建议 113: 谨慎使用 _Generic	433
建议 114: 尽量使用 gets_s 替换 gets 函数	436
建议 115: 尽量使用带边界检查的字符串操作函数	436
建议 116: 了解 C11 多线程编程	438
建议 117: 使用静态断言 _Static_assert 执行编译时检查	442
建议 118: 使用 _Noreturn 标识不返回值的函数	442
第 15 章 保持良好的设计	443
建议 119: 避免错误地变量初始化	443
建议 120: 谨慎使用内联函数	444

建议 121: 避免在函数内定义占用内存很大的局部变量	445
建议 122: 谨慎设计函数参数的顺序和个数	446
建议 123: 谨慎使用标准函数库	447
建议 124: 避免不必要的函数调用	447
建议 125: 谨慎程序中嵌入汇编代码	448