C语言实用之道

[美] Giulio Zambon 著
潘爱民 译

PRACTICAL C

清华大学出版社

C 语言实用之道

[美] Giulio Zambon 著
潘爱民 译

清华大学出版社
北 京

Giulio Zambon
Practical C
EISBN：978-1-4842-1768-9

北京市版权局著作权合同登记号 图字：01-2017-5758

图书在版编目(CIP)数据

C 语言实用之道 / (美)朱里奥·赞博(Giulio Zambon) 著；潘爱民 译. —北京：清华大学出版社，2018
书名原文：Practical C
ISBN 978-7-302-49904-6

Ⅰ. ①C… Ⅱ. ①朱… ②潘… Ⅲ. ①C 语言－程序设计 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字(2018)第 055459 号

责任编辑：王 军 李维杰
装帧设计：孔祥峰
责任校对：曹 阳
责任印制：李红英

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-62770175 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市金元印装有限公司
经 销：全国新华书店
开 本：170mm×240mm **印 张**：32.5 **字 数**：637 千字
版 次：2018 年 5 月第 1 版 **印 次**：2018 年 5 月第 1 次印刷
印 数：1～4000
定 价：98.00 元

产品编号：075900-01

中文版序

这是一本讲述C语言实践的书，作者以自身的实践和思考来展示C语言编程中的基础概念和典型使用场景。C语言本身简洁而又灵活，有强大的表达能力，几乎可以实现迄今为止所有能够想象到的计算能力。然而，越来越多的程序员在弃用C语言，改而学习更具生产效率的编程语言。很多人提出的一个问题是，学了C语言有什么用？从现实的角度，用C语言编写的、新的大型软件越来越少，但是，一些关键的软件往往离不开C语言，如图形引擎、网络协议等一些性能关键的模块，当然少不了像操作系统和驱动程序之类的最底层软件。因此，C语言在各种编程语言排行榜上始终排在前列。另外，C语言也适合一些“小而美”的程序，在本书中可以看到这样一些例子。以我个人之见，C语言是最贴近计算机工作原理的高级语言，并且Internet上有丰富的文档和代码积累，每一个对计算机工作原理有好奇心的IT从业人员都应该掌握C语言。

本书内容涵盖两大部分。第一部分介绍C语言编程中的基本概念和程序设计基础(前7章)，涉及变量、宏、结构、本地化、宽字符、整数和浮点数的表达形式等基本语言层面的概念和要点(第2章)，也包括迭代、递归、链表、栈、队列、异常等程序设计中广泛使用的设计元素(第3至第5章)，同时作者还完整剖析了两个实用案例：字符串(第6章)和动态数组(第7章)。第二部分是用C语言来完成特定领域中的开发示例，包括搜索(第8章)、排序(第9章)、数值积分(第10章)、嵌入式软件开发(第11章)、嵌入数据库功能(第12章)、嵌入Web服务器(第13章)以及游戏应用开发(第14章)。即使读者在实践中不需要涉猎如此广泛的应用范围，通过阅读这些章，也可以了解到C语言在这些领域中是如何被使用并发挥作用的。

这不是一本教科书，但是其内容非常适合学习C语言，并且作者的叙述风格也很有特色，他直接以第一人称和第二人称来讲解书中的内容，就好像在课堂上传授C语言的开发经验。本书在表达上有明显的口语化特点，相信在阅读时会有一种亲切感，学习也相对要轻松一些。然而，本书通过大量的例子和代码来讲解概念和技巧，这确保了本书内容的严谨性，并且不少代码非常有启发意义。此外，本书的代码又是成体系的，前后一致性比较好，如第6和第7章中讲到的字符串和动态数组组件，在后面的章节中也都用到了。从这个角度看，这本书又非常适合作为课程参考材料，教师在讲解了C语言基础知识以后，可以成体系地引入这本书中的内容。

我已经很多年没有接手翻译或著书的工作了，当王军编辑向我推荐这本书时，无论是内容，还是作者的经历，都引起我的共鸣。作为一名 C 程序员老兵，我看到每一个标准 C 函数都有一种亲切感。基于这样的心情，我答应王军编辑翻译这本书。经过一年来的努力，终于完成了翻译工作。原著中有一些笔误，我在翻译过程中修正了一些，但我相信，翻译本身难免也会引入新的错误，虽然经过两遍校对，但还会留有错误，请读者谅解。

潘爱民

2017 年 12 月于杭州

作者简介

Giulio Zambon 最初喜爱的是物理，但是三十年前他决定还是专注于软件开发，当时计算机是由晶体管和核心存储体构成的，程序还是打在卡上的，并且 FORTRAN 还只有算术 IF。多年来，他学习了很多种计算机语言，与各种操作系统打交道。他对电信和实时系统特别有兴趣，他曾经管理过好多个项目，都顺利地完成了。

在 Zambon 的职业生涯中，他去过五个不同国家的八个城市，曾任软件开发人员、系统顾问、过程改进经理、项目经理和首席运营官。自 2008 年初以来，他住在澳大利亚堪培拉以北几公里处的宁静的郊区，在这里他致力于他的许多兴趣，特别是编写软件来生成和解决数字难题。访问他的网站 http://zambon.com.au/，可以看到他撰写的论文和所著书籍的完整列表。

目　录

第 1 章

引　言

因为这是一本介绍C语言使用诀窍的书，所以这里不会有关于C语言的描述。不过，为了保证我们处在同一个频道上，有时候我会引入一些对于语言特性的简短描述。第2章将涵盖一些通常招致错误的C语言特性。

关于C语言的介绍，可以参考经典的Ivor Horton编著的《C语言入门经典(第5版)》，以及大量的关于这一主题的其他书籍。

我开发了本书中讲述的所有程序，使用gcc(GNU Compiler Collection) 4.8.4版本和Eclipse开发环境(4.5.0发布版)，在一台64位笔记本电脑上运行Linux-GNU Ubuntu 14.04 LTS版本。

C标准的当前版本是ISO/IEC 9899:2011，通常称为C11，它扩展了C标准的上一个版本(ISO/IEC 9899:1999，称为C99)。gcc的C编译器支持C99和C11。关于gcc选项中涉及C语言版本的完整列表，可以参考gcc.gnu.org/onlinedocs/gcc/C-Dialect-Options.html。

为了编译本书中的绝大多数代码，需要使用-std=c99选项，因为我使用了类似于Java的for循环格式，循环控制变量的定义包含在for语句中。例如：

```
for (int k = 0; k < N; k++)
```

以前版本的C语言要求在for语句之外定义控制变量，如下所示：

```
int k;
for (k = 0; k < N; k++)
```

1.1　编码风格

编码风格由几个方面构成。由于在本书的例子中使用了我的编码风格，因此通过理解这一编码风格，你可以更容易地理解我的代码。

1.1.1 缩进

看一下本书附带的源代码，可以注意到，语句结束处所有的右花括号都缩进排列，如下面的代码清单 1-1 所示。

代码清单1-1 作者的编码风格

```
1. void dar_list(Dar *stk) {
2.   if (stk == NULL) {
3.     printf("Nothing to list\n");
4.     }
5.   while (stk != NULL) {
6.     printf("%p %zu %zu\n", stk, stk->size, stk->n);
7.     stk = stk->down;
8.     }
9.   }
```

第 1、第 2 和第 5 行的左花括号紧跟在前面一行，而第 4、第 8 和第 9 行的右花括号缩进排列。这种风格算是与众不同，因为 Eclipse 和其他的开发环境默认并不认识这种风格，但它们对另外两种广为使用的风格(如代码清单 1-2 和代码清单 1-3 所示)却完全支持。

代码清单1-2 展开式的编码风格

```
1. void dar_list(Dar *stk)
2. {
3.   if (stk == NULL)
4.   {
5.     printf("Nothing to list\n");
6.   }
7.   while (stk != NULL)
8.   {
9.     printf("%p %zu %zu\n", stk, stk->size, stk->n);
10.    stk = stk->down;
11.  }
12. }
```

代码清单 1-2 中的展开式风格真的是展得太开了。除了需要更开阔的视野才能跟上代码流的逻辑以外，这种风格也给人造成了这样的印象：块语句相对前面的条件或迭代语句是独立的。例如，需要注意到第 3 行的 if 并非跟着一条简单语句和分号，这样可以强调只有当 if 条件为真的时候，此块语句才会被执行。这种风格也有一个概念性问题，在后面关于代码清单 1-3 的解释中再说明这一问题。很显然，如果把代码行数作为要达成的目标，那么你可能会喜欢这种风格！

代码清单1-3　紧凑型的非缩进编码风格

```
1. void dar_list(Dar *stk) {
2.  if (stk == NULL) {
3.    printf("Nothing to list\n");
4.  }
5.  while (stk != NULL) {
6.    printf("%p %zu %zu\n", stk, stk->size, stk->n);
7.    stk = stk->down;
8.  }
9. }
```

代码清单 1-3 中显示的风格可能最为常用。这种风格很不错，但是在我看来，它有两个问题：一个是概念性的，另一个是实用性的。

概念性的问题在于：用于分隔块语句的左右花括号属于块语句本身，块语句缩进了，那么为什么右花括号却没有跟着缩进(在展开式风格的情况下，左花括号也有同样的问题)？例如，代码清单 1-3 中第 8 行的右花括号属于从第 5 行开始的块语句，这一块语句包含 printf 和对 stk 的赋值。于是，第 8 行的右花括号应该放在 stk 的 s 的下面，而不是放在 while 的 w 的下面。

实用性的问题在于：通过不缩进右花括号，你可能获得了视觉上的纯净，我将之称为“石板效果”。为了演示什么是石板效果，我截取了代码清单 1-1 的屏幕快照，再加上阴影，形成了图 1-1。

```
void dar_list(Dar *stk) {
  if (stk == NULL) {
    printf("Nothing to list\n");
    }
  while (stk != NULL) {
    printf("%p %zu %zu\n", stk, stk->size, stk->n);
    stk = stk->down;
    }
  }
```

图 1-1　本书编码风格的石板效果

正如你所看到的，凡是依赖于 if 条件的所有内容都“挂”在 if 语句后面，凡是包含在 while 循环中的所有内容都挂在 while 语句后面。毫无疑问，这使得阅读代码更加容易。

关于多个 if 和 else，再多说几句。

我通常看到 if 和 else 串成这样：

```
if (condition 1) {
  ...
} else if (condition 2) {
  ...
} else {
```

```
  ...
}
```

这有可能从图形上看起来非常舒适和紧凑，但是，它并没有反映出这些 if 和 else 未被显示在同一层次上的事实(它们应该是同一层次的)。下面看看我如何处理这样的代码片段：

```
if (condition 1) {
  ...
  }
else if (condition 2) {
  ...
  }
else {
  ...
  }
```

1.1.2 命名和其他规范

本书包含了几个函数库，每个都由一个 C 文件和对应的头文件(例如 string.c 和 string.h)构成。每个库都被概括描述成少数几个标识性的字母(例如 str)，用这些字母作为所有导出的宏、变量和函数的前缀。宏常量(即不带参数的宏)用大写字母(比如 STR_LOG)表示，而像函数类型的宏，只有前缀是大写的(例如 STR_crash( ))。在导出的变量和函数的名称中，前缀部分用小写(例如 str_stack 和 str_list( ))。

绝大多数整型变量的名称以一个从 i 到 n 的字母开头，特别是如果这些名称很短的话。这是我最初使用计算机时养成的习惯：我学习的第一门计算机语言(40 多年前)是 FORTRAN，它自动把以这些字母开头的变量识别成 INTEGER 类型。我必须承认，我并不完全遵从这条规则来命名整型变量，但是，你在我的代码中绝对找不到以任何一个“整型”字母开头的非整型变量。很简单，我不会那么做。

如果看到一个带有多个函数的模块，你可能会注意到，这些函数以字母顺序排列。所以，无须搜索功能就可以立即找到函数。对于非导出的函数，为了做到无须关心它们在模块中的实际位置就可以引用它们，我在 C 文件的开始处声明这些函数。

同样，为便利起见，我在每个函数的最前面写一行注释，函数的名称位于右侧，如下所示：

```
//------------------------------------------- str_clean_up
```

当代码流要被中断的时候，也使用类似的规范，如下所示：

```
if (str == NULL) return;                                    //-->
```

通过将函数和退出点在右端单独标记出来，使得它们更易于被辨认出来。

这带来了另一个规范：每一代码行永远不要超过80个字符。这也是我早年编程经历留下来的另一个习惯，当时我在穿孔卡上键入FORTRAN程序，而穿孔卡只有80列。更进一步，精确而言，只有6至72列可被用于可执行代码(如果好奇的话，第1列用来标明是否注释，第2至第4列用于标记，第5列标明续行，第73至第80列是卡片编号)。为了使代码有更好的可读性，80列看起来是一个合理长度。

无论如何，在编程中重要的事情并不是采用了哪些规范，而在于是否始终如一地坚持规范。只要遵守规范并保持始终如一，就可以使代码更易于理解，更易于维护。

我认为，遵守规范的这种纪律和坚持，是优秀程序员的内在品质。在培训新程序员的时候，我甚至会检查：语句内的空白间隔是否在整个模块中保持一致，在任何一行的尾部有没有多余的空格，更不能使用制表符。现在，开发环境会自动剔除尾部的空格，但是“不使用制表符”这一规则仍然是有用的，例如当要把一部分代码粘贴到文档中时。

1.1.3 goto的使用

当我学会编程的时候，还没有块语句(block statement)。因此，实现块语句的唯一办法是使用goto。例如，如下代码结构：

```
if (condition) {
  // condition satisfied
  ...
  }
else {
  // condition not satisfied
  ...
  }
// whatever
...
```

在FORTRAN中的做法，类似如下(在20世纪70年代早期，FORTRAN严格大写，并且记住，从第一列开始的行是注释)：

```
    IF (condition) GOTO 1
C CONDITION NOT SATISFIED
    ...
    GOTO 2
```

```
C CONDITION SATISFIED
   1 ...
C WHATEVER
   2 ...
```

C 语言中等价于上述 FORTRAN 的代码如下：

```
if (condition) goto yes;

// Condition not satisfied.
...
goto done;

// Condition satisfied.
yes: ...

// whatever
done: ...
```

没有人会用这种方式来使用 C 语言，但是，在结构化语言中围绕着使用 goto 是不是一种禁忌，尚无定论(可能禁忌根本不存在，但我们就不跑题了)。例如，考虑下面的代码：

```
if (condition_1) {

  // Satisfied: 1.
  ...
  if (condition_2) {

    // Satisfied: 1 and 2.
    ...
    if (condition_3) {

      // satisfied: 1, 2, and 3.
      ...
      if (condition_4) {

        // satisfied: 1, 2, 3, and 4.
      ...
      // Here a big chunk of code happens to follow
      } // condition_4
    } // condition_3
  } // condition_2
} // condition_1
```

你要往右走多远？每多出一个 if，都把中间部分的大块代码往右移一点。你能用这种方式来处理 10 个 if 条件吗？我不会这么做。下面是使用 goto 的做法：

```
if (!condition_1) goto checks_done;          //-->

// Satisfied: 1
...
```

```
if (!condition_2) goto checks_done;          //-->

// Satisfied: 1 and 2
...
if (!condition_3) goto checks_done;          //-->

// Satisfied: 1, 2, and 3
...
if (!condition_4) goto checks_done;          //-->

// Satisfied: 1, 2, 3, and 4
...

checks_done:                                 //<--
...
// Here a big chunk of code happens to follow
```

这只是一个例子，用于说明在这样的场合下你可能会考虑使用 goto。然而，有可能它们并不恰当。我只是在说明我的观点：出于感情因素，而不是理性和实用的因素来拒绝使用一种有效的语言结构，这是错误的。

1.2　如何阅读本书

在本书中，当一章依赖于前面章节中介绍的信息时，可以找到对前面相关章节的引用。因此，你总是可以安全地跳过那些你当下认为没有帮助的章节。换句话说，可以聚焦在那些对于你当前正在开发的代码有帮助的章节上，而无须按顺序阅读本书。

第 2 章“微妙之 C”，讨论 C 语言中经常被误解的以及可能引入莫名其妙错误的那些特性。

第 3 章“迭代、递归和二叉树”，介绍递归技术和二叉树。

第 4 章“列表、栈和队列”，帮助你在表达项目集合时从多种可能的方法中进行选择。

第 5 章“异常处理”，告诉你如何捕捉运行时发生的问题，而不是简单地让程序崩溃。

第 6 章“字符串辅助功能”，讲述一种动态分配字符串的方法，而不是在编译时静态分配。

第 7 章“动态数组”，相对于第 6 章中讲述的针对字符串的一些函数，改编为可适用于通用的数组(毕竟，字符串只不过是以 null 结尾的字符数组而已)。

第 8 章“搜索”，讲述线性搜索和二分搜索，以及如何使用二叉搜索树。

第 9 章“排序”，介绍对一组无序项目进行排序的各种技术。

第 10 章“数值积分”，讲述在一条点画线的下面求面积以及在一个面的下面求体积的数值化方法。

第 11 章“嵌入式软件”，讨论在编写操纵硬件的实时软件时需要考虑的一些特殊事项。

第 12 章“数据库”，介绍如何在 C 语言中操作 SQL 数据库。

第 13 章“使用 Mongoose 开发 Web 服务器”，讲述如何在程序中嵌入一个 Web 服务器。

第 14 章“游戏应用：MathSearch”，讲述如何开发一个生成数字迷宫的程序。

附录 A 列出了本书用到的所有缩写，包括首字母缩写。

附录 B 概要摘录了用于控制数据库的 SQL 命令。