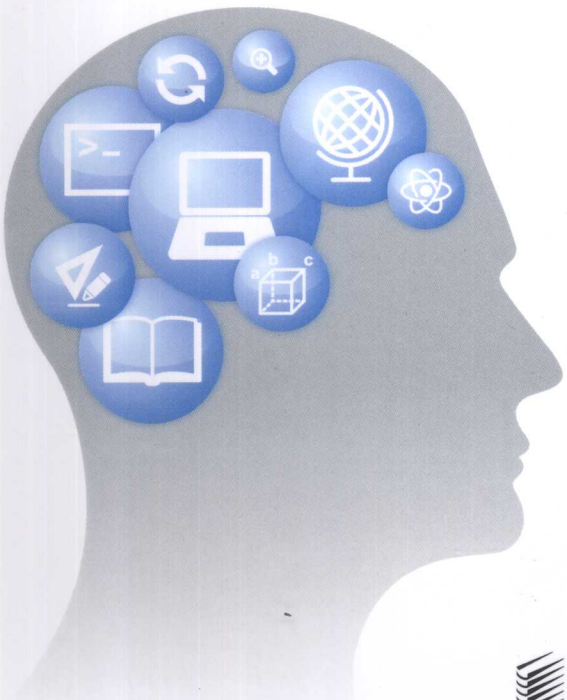
像计算机科学家一样思考C++

[美] Allen B. Downey 著
黄鑫 夏思雨 译

How To Think
Like A Computer Scientist
C++ Version

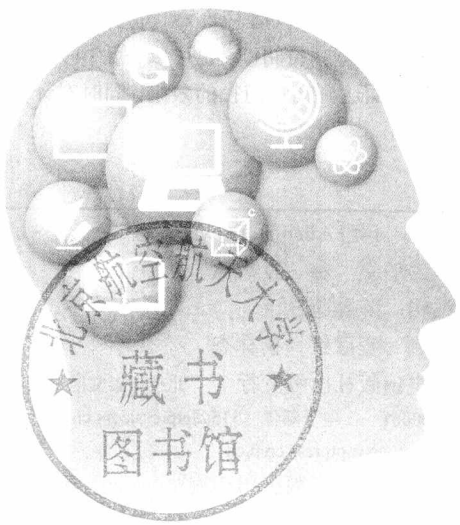
语言本身并不重要，解决问题的创新方法才是王道
学会“像计算机科学家一样思考”



人民邮电出版社
POSTS & TELECOM PRESS

像计算机科学家一样思考C++

[美] Allen B. Downey 著
黄鑫 夏思雨 译



人民邮电出版社
北京

图书在版编目（CIP）数据

像计算机科学家一样思考C++ / （美）唐尼(Downey, A. B.）著 ；黄鑫，夏思雨译. -- 北京 ：人民邮电出版社，2013.6
ISBN 978-7-115-31280-8

Ⅰ. ①像… Ⅱ. ①唐… ②黄… ③夏… Ⅲ. ①C语言－程序设计 Ⅳ. ①TP31

中国版本图书馆CIP数据核字(2013)第060703号

内 容 提 要

本书作者基于自己在美国各所大学和学院讲授计算机程序设计课程的经验，开创了“像计算机科学家一样思考（How to Think Like a Computer Scientist）”的教学理念和方法。本书正是基于这样的方法，用全新的角度、丰富的实例全面讲解了C++语言。

全书共15章。第1章介绍了编程的基本知识，即什么是编程以及如何编程。第2章到第9章介绍了C++的基本元素与基本语法，包括变量、类型、函数、迭代、字符串、结构体等等；第10章到第15章介绍了C++的高级功能，包括Vectors、成员函数、类和不变式、文件输入输出等。

本书适合C++的初学者和初级程序员阅读，也可以作为相关专业或培训的教程使用。通过学习本书，读者不仅可以在C++方面达到初窥门径的效果，同时对计算机编程这门技艺也会有一个全面而科学的认识。

◆ 著 [美] Allen B. Downey
译 黄 鑫 夏思雨
责任编辑 陈冀康
责任印制 程彦红 杨林杰

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：787×1000 1/16
印张：14.25
字数：265千字 2013年6月第1版
印数：1－3 000册 2013年6月河北第1次印刷

著作权合同登记号 图字：01-2012-5792号

定价：39.00元

读者服务热线：(010)67132689 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版权声明

How To Think Like A Computer Scientist, C++ Version, First Edition, by Allen B. Downey

ISBN-13: 978-1441419057

作者简介

Allen B. Downey 是美国 Olin 工程学院的计算机科学副教授。他曾经在 Wellesley College、Colby College 和 U.C. Berkeley 教授计算机科学课程。他在 MIT 获得学士和硕士学位，并且从 U.C. Berkeley 获得计算机科学博士学位。Allen 基于自己教授计算机程序设计课程的经验，开创了“像计算机科学家一样思考（How to Think Like a Computer Scientist）”的教学理念和方法，并由此编写了几本程序设计语言的图书。其中，《Think Python》、《Think Complexity》由 O’Reilly 出版；《Think Java》、《Think C++》也广受关注和好评。

译者简介

黄鑫，毕业于西安交通大学。多年软件开发经验，对设计大型分布式系统有独到的见解。对将更多的开源项目引入 Windows 平台有浓厚的兴趣。目前致力于推广持续交付这一实践。希望通过自己的努力，让技术改变世界。微博：http://weibo.com/dummyone

夏思雨，1987 年 8 月出生于陕西西安。2009 年毕业于华中科技大学，2012 年毕业于北京邮电大学。现就职于思特沃克软件技术（西安）有限公司，从事软件开发。微博：http://weibo.com/evainexia

目录

第 1 章　编程方式 …… 1
1.1　什么是编程语言 …… 2
1.2　什么是程序 …… 4
1.3　什么是调试 …… 5
1.3.1　编译时错误 …… 5
1.3.2　运行时错误 …… 6
1.3.3　逻辑和语义错误 …… 6
1.3.4　实验调试 …… 6
1.4　形式语言和自然语言 …… 7
1.5　第一个程序 …… 9
1.6　术语 …… 12
第 2 章　变量和类型 …… 14
2.1　输出更多 …… 14
2.2　值 …… 15
2.3　变量 …… 16
2.4　赋值 …… 17
2.5　输出变量 …… 19
2.6　关键字 …… 20
2.7　运算符 …… 20
2.8　计算顺序 …… 22
2.9　字符类型的运算符 …… 22
2.10　组合 …… 23
2.11　术语 …… 24
第 3 章　函数 …… 26
3.1　浮点数 …… 26
3.2　从 double 转换为 int …… 28
3.3　数学函数 …… 28
3.4　复合表达式 …… 30
3.5　添加新的函数 …… 30
3.6　定义和用法 …… 33

3.7 多函数程序 ……34
3.8 形参和实参 ……35
3.9 形参和局部变量 ……36
3.10 多参数函数 ……37
3.11 带返回值的函数 ……38
3.12 术语 ……39
第 4 章 条件和递归 ……40
4.1 模运算符 ……40
4.2 条件执行 ……40
4.3 选择执行 ……41
4.4 链式条件 ……42
4.5 嵌套条件 ……43
4.6 return 语句 ……44
4.7 递归 ……44
4.8 无限递归 ……47
4.9 递归函数的调用栈图 ……47
4.10 术语 ……48
第 5 章 带返回值的函数 ……49
5.1 返回值 ……49
5.2 程序开发 ……52
5.3 复合用法 ……54
5.4 重载 ……55
5.5 布尔值 ……57
5.6 布尔型变量 ……57
5.7 逻辑操作符 ……58
5.8 布尔函数 ……59
5.9 main 函数返回值 ……60
5.10 多重递归 ……61
5.11 信心的跳跃 ……64
5.12 更多的例子 ……64
5.13 术语 ……66
第 6 章 迭代 ……67
6.1 多次赋值 ……67
6.2 迭代 ……68
6.3 while 语句 ……69

6.4 表格 ····· 71
6.5 二维表 ····· 74
6.6 封装和广义化 ····· 74
6.7 函数 ····· 76
6.8 更多封装 ····· 77
6.9 局部变量 ····· 77
6.10 更多广义化 ····· 78
6.11 术语 ····· 80
第 7 章 字符串和其他 ····· 82
7.1 字符串容器 ····· 82
7.2 apstring 变量 ····· 83
7.3 字符串中的字符 ····· 83
7.4 长度 ····· 84
7.5 遍历 ····· 85
7.6 运行时错误 ····· 86
7.7 find 函数 ····· 86
7.8 自定义 find 函数 ····· 87
7.9 循环和计数 ····· 88
7.10 递增和递减操作符 ····· 88
7.11 字符串拼接 ····· 89
7.12 apstring 的可变性 ····· 91
7.13 apstrings 的可比较性 ····· 91
7.14 字符分类 ····· 92
7.15 其他 apstring 函数 ····· 93
7.16 术语 ····· 93
第 8 章 结构体 ····· 95
8.1 复合值 ····· 95
8.2 Point 对象 ····· 95
8.3 访问实例变量 ····· 97
8.4 操作结构体 ····· 98
8.5 将结构体作为参数 ····· 99
8.6 值传递 ····· 99
8.7 引用传递 ····· 100
8.8 矩形 ····· 102
8.9 返回结构体类型 ····· 104

8.10 将其他类型按引用传递 ……104
8.11 获取用户输入 ……105
8.12 术语 ……108
第 9 章 更多结构体 ……109
9.1 Time ……109
9.2 printTime ……110
9.3 对象函数 ……110
9.4 纯函数 ……111
9.5 const 参数 ……113
9.6 修改器 ……114
9.7 填写函数 ……115
9.8 哪个最好 ……115
9.9 增量式开发 VS 规划 ……116
9.10 普遍化 ……117
9.11 算法 ……118
9.12 术语 ……119
第 10 章 vector ……120
10.1 访问元素 ……121
10.2 复制 vector ……123
10.3 for 循环 ……123
10.4 vector 的长度 ……124
10.5 随机数 ……125
10.6 统计 ……126
10.7 随机数的 vector ……127
10.8 计数 ……128
10.9 检查其他值 ……129
10.10 直方图 ……131
10.11 单次遍历的解决方案 ……132
10.12 随机种子 ……132
10.13 术语 ……133
第 11 章 成员函数 ……135
11.1 对象和方法 ……135
11.2 print ……136
11.3 隐式变量访问 ……138
11.4 另一个例子 ……139

11.5 第三个例子 ····· 140
11.6 更复杂的例子 ····· 140
11.7 结构体 ····· 141
11.8 初始化还是构造 ····· 142
11.9 最后一个例子 ····· 143
11.10 头文件 ····· 144
11.11 术语 ····· 147
第 12 章 包含对象的 vector ····· 149
12.1 复合形式 ····· 149
12.2 Card 对象 ····· 149
12.3 printCard 函数 ····· 151
12.4 equals 函数 ····· 154
12.5 isGreater 函数 ····· 155
12.6 包含 Card 对象的 vector ····· 157
12.7 printDeck 函数 ····· 159
12.8 搜索 ····· 160
12.9 二分查找 ····· 161
12.10 vector 和子 vector ····· 164
12.11 术语 ····· 166
第 13 章 向量对象 ····· 167
13.1 枚举类型 ····· 167
13.2 switch 语句 ····· 169
13.3 Deck ····· 170
13.4 另一个构造函数 ····· 172
13.5 Deck 成员函数 ····· 172
13.6 洗牌 ····· 174
13.7 排序 ····· 175
13.8 subdeck ····· 176
13.9 洗牌和处理 ····· 177
13.10 合并排序 ····· 177
13.11 术语 ····· 180
第 14 章 类和不变式 ····· 181
14.1 私有数据和私有类 ····· 181
14.2 什么是类 ····· 182
14.3 复数 ····· 183

14.4 访问器函数 ……187
14.5 输出……188
14.6 支持复数运算的函数 ……189
14.7 支持复数运算的其他函数 ……190
14.8 不变式 ……191
14.9 先验条件 ……192
14.10 私有函数 ……195
14.11 术语……196
第 15 章 文件输入/输出和 apmatrix ……197
15.1 流……197
15.2 文件输入 ……198
15.3 文件输出 ……200
15.4 输入解析 ……200
15.5 数字解析 ……202
15.6 Set 数据结构 ……203
15.7 apmatrix ……207
15.8 距离矩阵 ……209
15.9 合适的距离矩阵 ……210
15.10 术语……212
附录 A AP 类的快速参考 ……213

第1章

编程方式

本书旨在教会你如何像计算机专家一样思考。我喜欢计算机专家的思考方式，因为他们综合了数学、工程和自然科学的最佳特性。计算机专家像数学家一样，运用形式语言来表达思想（尤其是计算指令）；又像工程师一样进行设计，将组件装配到系统里并对可替换的部件进行评估权衡；还像自然科学家一样，观察复杂系统的行为，形成假设并通过实验来证明预测。

解决问题是一个计算机专家应该具备的最重要的单一技能。该技能包括明确表述问题的能力，有创意地思考解决方案以及清楚准确地表述解决方案。人们后来发现，学习编程的过程是练习解决问题技巧的一个相当好的机会。这就是为什么本章叫做“编程方式”。

同时，本书的另一目的是帮助你准备计算机科学 AP 考试[1]。尽管我们可能并没有直接实现这一目标。比如，本书并没有很多类似 AP 考试题的练习。但从另一个角度说，如果你完全理解了本书中的概念和 C++编程的细节，你就可以在考试中有一

[1] 编者注：AP 考试全称 AdvancedPlacement，是美国大学预修课程。由美国大学理事会主持，AP 成绩不但可以抵扣成功申请美国大学的同学入学后相应课程的学分，而且 AP 成绩也是美国各大学录取学生的重要依据。

个良好的表现。

1.1 什么是编程语言

你即将学习的编程语言是 C++。自 1998 年起的 AP 考试都以 C++为基础。在这之前，采用的是 Pascal。C++和 Pascal 都是**高级编程语言**，你可能听说过的其他高级语言有 Java、C 和 FORTRAN。

你可能从“高级编程语言”这个名字中得知还有低级编程语言。低级编程语言一般指的是机器语言或者汇编语言。一般来说，计算机只能执行用低级语言编写的程序。因此，高级语言编写的程序需要先转换成低级语言再执行。高级语言的一个小缺点就是这一转换过程需要耗费一些时间。

但是，高级语言具有巨大的优势。首先，用高级语言编程要容易得多，这意味着该程序的编程时间较短，简明易读，正确性较高。其次，高级语言具有可移植的优势。这意味着用高级语言编写的程序只要经过略微的修改就可以在不同的计算机操作系统上运行。而用低级语言编写的程序只能在某一种计算机系统上运行，若要在另一种系统上运行，则需要重新编写代码。

鉴于这些优势，几乎所有的程序都是用高级语言编写。低级语言只应用在少数特殊场景中。

有两种将高级语言翻译成低级语言的方式：**解释**或者**编译**。解释器就是一个读取高级程序并执行的程序。实际上，解释器逐行翻译程序，交替读取代码行及执行命令，如图 1-1 所示。

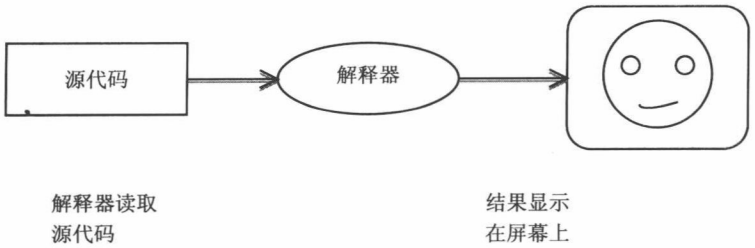


图 1-1

编译器则会在执行命令前，一次性地将全部高级程序代码翻译成机器语言。通常可以将编译程序作为一个单独的步骤，稍后再执行编译后的代码。在这种情况下，高级程序称为**源代码**；编译后的程序称为**目标代码**或者**可执行代码**。

以下面这种情况为例，假设你用 C++编写程序。你可能选择一个文本编辑器来编写程序（文本编辑器就是一个简单的文字处理器）。当程序编写完成时，可以将它保存为 program.cpp。program 是你自己命名的文件名，后缀.cpp 则表示文件为 C++源代码。

然后，根据编程环境，可以关闭文本编辑器，运行编译器。编译器会读取源代码，编译源代码并创建一个包含目标代码的新文件 program.o，或者可执行文件 program.exe，如图 1-2 所示。

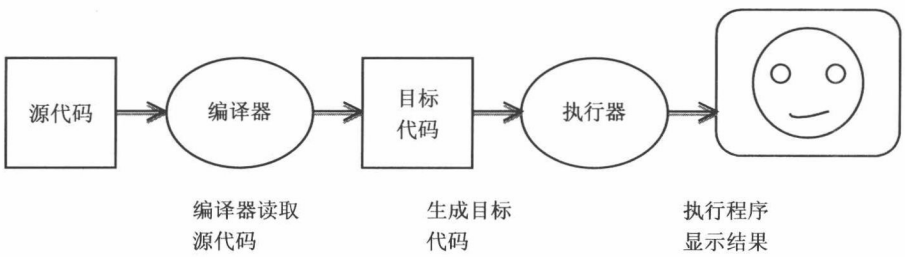


图 1-2

下一个步骤就是运行程序，这一步需要程序的执行器。程序的执行器需要加载可运

行程序（从硬盘复制到内存）并让计算机开始执行程序。

尽管这一过程看起来有点复杂，但是好消息是在绝大多数的编程环境（有时称为开发环境）中，这些步骤已经能够自动执行。一般来说，只需要编写一段程序，输入一条命令就可以完成编译和运行过程。另一方面，了解程序执行过程中有哪些步骤在后台运行是很有用的，这样在出错的时候你可以很快发现问题所在。

1.2 什么是程序

程序就是详细说明如何进行一次计算的一个指令序列。该计算可能是数学计算，比如，解方程组或者计算多项式的根；也可能是符号计算，比如，在文件中搜索和替换文本或者编译一个程序（够奇怪了）。

不同编程语言中的指令（命令或者描述）看起来都不一样，但是每种语言都有一些基本的功能。

输入：从键盘或者其他设备读取数据和文件。

输出：向显示器或者其他设备输入数据，或将数据写入文件。

数学计算：完成基本数学运算，如，加法和乘法等。

测试：检查特定条件并按适当序列执行指令。

复现：在有一定可变性下重复执行某些动作。

不管你相信与否，这几乎是一个程序所有的功能。你所使用过的每个程序，不管多复杂，都是由或多或少类似这样的功能组成的。因此，描述程序的一个方法就是将

大而复杂的任务划分成尽可能小的子任务，直到这些小的子任务可以用这些基本功能中的某一个完成。

1.3 什么是调试

编程本身是一个复杂的过程，并且由人类而不是机器完成，所以经常会发生一些错误。由于一些奇怪的原因，程序中的错误称为 bug，而追踪定位 bug 并且将其修正的过程则称为调试（Debug）。

程序中发生的错误有不同的种类，知道如何分辨不同的错误可以更快速地定位 bug 的位置。

1.3.1 编译时错误

编译器只能编译语法正确的程序，否则会导致编译过程失败，无法运行程序。**语法**指的是程序结构以及与该结构相关的规则。

以英语语法为例，一个句子必须以大写字母开头，句号结尾。诸如“this sentence contains a syntax error.”和“So does this one”这样的两个句子都包含语法错误。

对大多数读者来说，少量语法错误并不是什么大问题。这就是为什么我们可以毫无障碍地阅读 E.E.卡明斯的诗歌。

但是编译器并不是如此的宽容。如果你的程序中出现一处语法错误，编译器就会输出错误消息并且退出，而你就无法再运行自己的程序。

更糟糕的是，C++中具有比英语更多的语法规则，并且大多数时候你从编译器得到的错误消息都没有太大帮助。在你刚开始学习编程的时候，你很可能会花费大量的