电子工程与计算机科学系列

EECS

C++ 程序设计

翁惠玉 ◎编著

上海交通大學出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

电子工程与计算机科学系列 EECS

C++ 程序设计

翁惠玉 ◎编著

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

内容提要

本书以C/C++语言为环境，重点讲授程序设计的基本概念和方法，包括过程化的程序设计和面向对象的程序设计。对讲述的所有概念，均提供大量的例题和习题，使读者通过学习概念以及训练和实践，掌握程序设计的方法和过程，并具备良好的程序设计风格。本书内容丰富，结构清晰，采用以应用引出知识点的方法，让学生明确学习的目的，提高学习兴趣。

本书可作为高等院校计算机专业的教材，也可供从事计算机软件开发的科研人员参考。

图书在版编目(CIP)数据

C++程序设计/翁惠玉编著. —上海：上海交通大学出版社，2017

ISBN 978-7-313-13627-5

Ⅰ.①C… Ⅱ.①翁… Ⅲ.①C语言—程序设计—高等学校—教材

Ⅳ.①TP312

中国版本图书馆CIP数据核字(2015)第186135号

C++程序设计

编　　著：翁惠玉

出版发行：上海交通大学出版社　　地　　址：上海市番禺路951号

邮政编码：200030　　电　　话：021-64071208

出 版 人：郑益慧

印　　制：上海颛辉印刷厂　　经　　销：全国新华书店

开　　本：787mm×1092mm　1/16　　印　　张：17.25

字　　数：434千字

版　　次：2017年1月第1版　　印　　次：2017年1月第1次印刷

书　　号：ISBN 978-7-313-13627-5/TP

定　　价：42.00元

前　言

“程序设计”是计算机专业十分重要的一门课程，是实践性非常强的一门课程，也是一门非常有趣、让学生很有成就感的课程。但在教学过程中，很多学生的反应是：听懂了，但不会做，以至于最后丧失了兴趣。我认为主要的问题是我们的教学过程过分重视程序设计语言本身，强调理解语言的语法，而没有把思路放在如何解决问题上面，以至于让学生只认识语言但不能运用语言写程序。

本书是作者根据多年来在上海交通大学电信学院和致远学院讲授“程序设计”课程的经验，参考了近年来国内外主要的程序设计教材编写而成。本教材采用以程序设计方法为主，程序设计语言为辅的思想，以及以应用引出知识点的方法，让学生先了解学习的目的，提高他们的学习兴趣，最后使学生能利用学到的知识解决某一应用问题。

本教材以C++为教学语言，介绍了程序设计基本概念和方法。C++是业界非常流行的语言，它使用灵活，功能强大。既支持过程化的程序设计，又支持面向对象的程序设计，可以很好地体现程序设计的思想和方法。

“程序设计”是实践性很强的一门课程，必须大量读程序和写程序。本教材对每个知识点都给出了大量的例题，帮助读者理解和掌握知识的应用以及学习良好的程序设计风格。每一章都有一个小节总结可能出现的错误，每一章最后还给出了大量的习题。本教材的习题分为两类：简答题和程序设计题。简答题帮助理解本章的知识点；程序设计题让读者用本章学到的知识解决一些实际的问题。本教材的所有例题都在VC6.0中调试通过。

本书的内容可以分为三大部分：第1章到第7章为第一部分，主要介绍一些程序设计的基本概念、技术、良好的程序设计风格以及过程化程序设计。包括数据类型、控制结构、数据封装、过程封装以及各种常用的算法。第8章到第12章为第二部分，介绍面向对象的思想。包括如何设计及实现一个类，如何利用组合和继承实现代码的重用，如何利用多态性使程序更加灵活，如何利用抽象类制订一些工具的规范。第三部分是第13章，介绍了输入输出的处理。

由于作者水平有限，本书存在的不足之处敬请读者批评指正，以便于我们在今后的版本中进行改进。

目　录

1 绪论

在学习程序设计之前，让我们先来了解一下什么是程序设计、程序设计的过程以及如何学习程序设计。

1.1 程序设计概述

程序设计就是教会计算机如何去完成某一特定的任务，即设计出完成某个任务的正确的程序。学习程序设计就是学习当老师，你的学生就是计算机。老师上课前先要备课，然后再去上课，最后检查学生的学习情况是否达到了预期效果。对应于这三个阶段，程序设计也包括三个过程：第一步是算法设计；第二步是编码；第三步是编译与调试。

上课前首先要知道学生的知识背景，然后才能有的放矢地去教，学习程序设计首先也要了解计算机能做什么。备课是把所要教授的知识用学生能够理解的知识表达出来。算法设计是把解决问题的过程分解成一系列计算机能够完成的基本动作。上课是把备课的内容用某种学生能够理解的语言描述出来。给中国学生讲课，就把备课的内容用中文讲出来。如果给美国学生讲课，就把备课的内容用英语讲出来。编码阶段也是如此。如果你的计算机支持 C 语言，就把算法用 C 语言表示出来。如果支持 PASCAL 语言，就用 PASCAL 语言描述。算法中的每一步骤都能与程序设计语言的某个语句相对应。上完课后要检查教学的效果。如果没有达到预期的结果，需要检查备课或上课中哪个环节出了问题，修改这些问题，重新再试。同样，编码后要运行程序，检查程序的结果是否符合预期的效果，如果没有，则需要检查算法和程序代码，然后重新运行。

1.2 计算机的基本功能

程序员与计算机交流的工具是程序设计语言。因此在设计程序前，首先需要了解程序设计语言能够提供的功能。

程序设计语言的发展经过了 4 个阶段：机器语言、汇编语言、高级语言和非过程化语言。非过程化语言通常就是一些数据库操纵语言，如 SQL 语言。而程序设计所用的语言通常是高级语言，因此我们必须了解高级语言能提供什么功能。

机器语言是由计算机硬件识别并直接执行的语言。机器语言能够提供的功能是由计算机硬件设计所决定。不同的计算机由于硬件设计的不同，它们的机器语言也是不一样的。每条机器语言语句(指令)由一个二进制的比特串组成。机器语言之所以必须由 0 和 1 组成是因为计算机内部的电路都是开关电路。0 和 1 正好对应于开和关两种状态。

用机器语言书写的程序是二进制比特串，很难阅读和理解，容易出错。而且程序员在用机

器语言编程序时还必须了解机器的很多硬件细节。例如，有几类寄存器、每类寄存器有多少个、每个寄存器长度是多少等等。由于不同的计算机有不同的机器语言，一台计算机上的程序无法在另外一台不同类型的计算机上运行，这将引起大量的重复劳动。

由于机器语言提供的功能是直接由硬件实现的，所以机器语言能完成的功能非常简单，通常只有正整数的加法、比较运算、数据传送和执行过程控制等。要用机器语言写一个解决稍微复杂一些的程序都是非常困难的。

为了克服机器语言的缺点，人们采用了与机器指令意义相近的英文缩写作为助记符，于是在20世纪50年代出现了**汇编语言**。汇编语言是符号化的机器语言，即将机器语言的每条指令符号化，采用一些带有启发性的文字串，如ADD(加)、SUB(减)、MOV(传送)、LOAD(取)。

与机器语言相比，汇编语言的含义比较直观，使程序的编写更加方便，阅读和理解也更加容易。但计算机并不"认识"汇编语言，不能直接理解和执行汇编语言写的程序，必须将每一条汇编语言的指令翻译成机器语言的指令才能执行。为此，人们创造了一种称为**汇编程序**的程序，让它充当汇编语言的程序到机器语言程序的翻译。

汇编语言解决了机器语言的可读性的问题，但汇编语言的指令与机器语言的指令基本上是一一对应的，提供的基本功能与机器语言是一致的，都是一些非常简单的功能，用汇编语言写程序就像教小学生学微积分一样困难，只有很少的高手能编写机器语言和汇编语言的程序。

高级语言的出现是计算机程序设计语言的一大飞跃，使更多的人可以加入编程的队伍。FORTRAN、COBOL、BASIC、C++等都是高级语言。

高级语言是一种与机器的指令系统无关、表达形式更接近于科学计算的程序设计语言，从而更容易被人们掌握。程序员只要熟悉简单的几个英文单词、熟悉代数表达式以及规定的几个语句格式就可以方便地编写程序，同时不需要知道机器的硬件环境，所以其他领域的科研人员也可以编写自己需要的程序了。由于高级语言是独立于机器硬件环境的一种语言，在一台计算机上编写的程序可以在另一台计算机上运行，即具有较好的可移植性。

尽管每种高级语言都有自己的语法规则，但提供的功能基本类似。每种程序设计语言都允许在程序中直接写一些数字或字符串，这些称为**常量**。对于在写程序时没有确定的值，可以给它们一个代号，称为**变量**。高级语言事先做好了很多处理不同数据的工具，称为**数据类型**，如整型、实型和字符型等。每个工具实现了一种类型的数据处理。如整型解决了整数在计算机内的如何保存、如何实现整数的各种运算问题。当程序需要处理整数时，可以直接用整型这个工具。程序设计语言提供的类型越多，功能也越强。如果程序设计语言没有提供某种类型，则当程序要处理这种类型的信息时，程序员必须自己编程解决。例如，通常的程序设计语言都没有复数这个类型，如果某个程序要处理复数，程序员必须自己解决复数的存储和计算问题。高级语言提供了将一个常量或表达式计算结果与一个变量关联起来的功能，这称为**变量赋值**。也可以根据程序执行过程中的某些值执行不同的语句，这称为程序设计语言的**控制结构**。对于一些复杂的大问题，直接设计出完整的算法有一定的困难，通常采用将大问题分解成一系列小问题。在设计解决大问题的算法时可以假设这些小问题已经解决，直接调用解决小问题的程序即可。每个解决小问题的程序被称为一个**过程单元**。在程序设计语言中过程单元被称为函数、过程和子程序等。

如果解决某个问题用到的工具都是程序设计语言所提供的工具，如处理整数或实数的

运算，这些程序很容易实现。如果用到了一些程序设计语言不提供的工具，则非常困难，如要处理一首歌曲、一张图片或一些复数的数据。我们希望能有这样一个工具可以播放一首歌曲或编辑一首歌曲，这时我们可以创建一个工具，即创建一个新的数据类型，如歌曲类型、图片类型和复数类型。这就是**面向对象程序设计**。面向对象的程序设计提供了我们创建工具的功能。

1.3 算法设计

算法设计是程序设计的关键。算法设计是要设计一个使用计算机提供的基本功能（更确切地说，应该是程序设计语言提供的功能）来解决某一问题的方案。算法设计的难点在于计算机提供的基本功能非常简单，而人们要它完成的任务是非常复杂的。算法设计必须将复杂的工作分解成一个个简单的、计算机能够完成的基本动作。

解决问题的方案要成为一个算法，必须能用清楚的、明确的形式来表达，以使人们能够理解其中的每一个步骤，无二义性。算法中的每一个步骤必须有效，是计算机可以执行的操作。例如，若某一算法包含“用π的确切值与 r[①] 相乘”这样的操作，则这个方案就不是有效的，因为计算机无法算出π的确切值。而且，算法不能无休止地运行下去，必须在有限的时间内给出一个答案。综上所述，算法必须具备以下 3 个特点：

(1) 表述清楚、明确，无二义性。

(2) 有效性，即每一个步骤都切实可行。

(3) 有限性，即可在有限步骤后得到结果。

有些简单的问题，我们一下子就可以想到相应的算法，没有多大的麻烦就可写一个解决该问题的程序。例如，计算圆的面积和周长，因为程序设计语言提供了实数计算的工具。而当问题变得很复杂时，就需要更多的思考才能想出解决它的算法。学习程序设计的难点通常是你自己都不知道该如何去解决问题，当然也就无法设计出算法。这时，你无须气馁，因为计算机是一个需要终身学习的行业。如果要做一个金融方面的软件，你必须先学习金融；要做一个游戏，你必须先成为玩这个游戏的专家。学程序设计的初级阶段是将你知道的知识教给计算机，将你会解决的问题教会计算机。

与所要解决的问题一样，各种算法的复杂性也千差万别。大多数情况下，一个特定的问题可以有多个不同的解决方案（即算法），在编写程序之前需要考虑许多潜在的解决方案，最终选择一个合适的方案。

算法可以有不同的表示方法，常用的有自然语言、流程图、PAD 图和伪代码等。

自然语言是我们最习惯使用的语言，但用来表示算法中的某些操作，如分支和循环则比较啰嗦，而且往往不太严格，会有二义性的问题。除了一些非常简单的算法外，一般不用自然语言。

流程图是比较早期提出的一种算法表示方法，由美国国家标准化协会 ANSI 规定。流程图用不同的图形表示程序中的各种不同的标准操作。流程图用到的图形及含义如图 1-1 所示。

① 由于程序代码的特殊性，本文中出现的变量统一用正体表示。

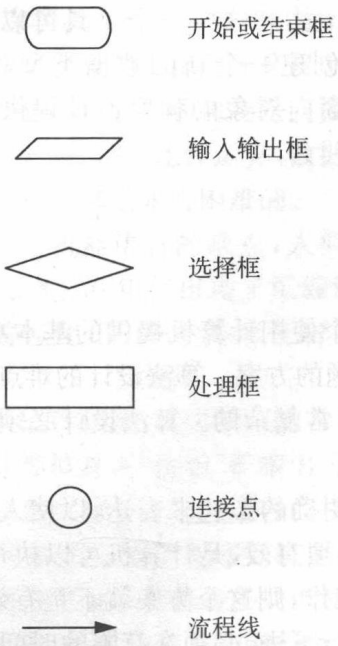

图 1-1　流程图符号

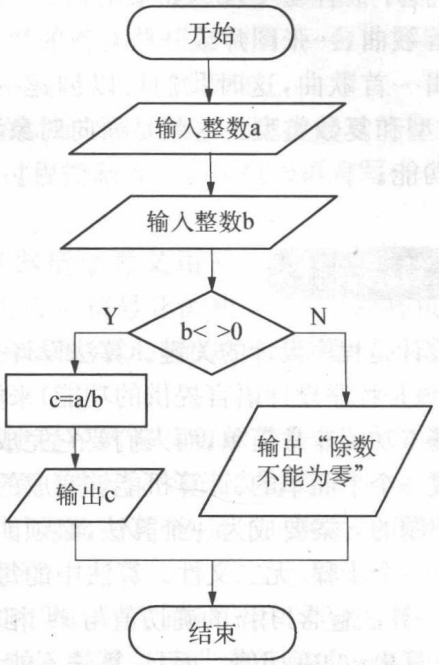

图 1-2　两个数相除算法的流程

例如，求两个数相除的流程图如图 1-2 所示。

用流程图表示算法直观清晰，清楚地表现出各个处理步骤之间的逻辑关系。但流程图对流程线的使用没有严格的限制，使用者可以随意使流程转来转去，使人很难理解算法的逻辑，难以保证程序的正确性。而且流程图占用的篇幅也较大。

随着结构化程序设计的出现，流程图被另一种称为 N-S 的图所代替。结构化程序设计规定程序只能由以下 3 种结构：顺序结构、分支结构和循环结构组成。N-S 图用 3 种基本的框表示 3 种结构，如图 1-3 所示。

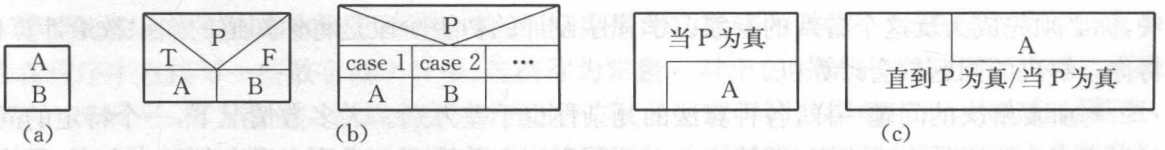

图 1-3　N-S 图的基本组件

(a) 顺序结构　(b) 分支结构　(c) 循环结构

既然程序可以由这些基本结构顺序组合而成，那么基本结构之间的流程线就不再需要了，全部的算法可以写在一个矩形框内。例如，求两个数相除的 N-S 图可表示为图 1-4 所示，判别整数 n 是否为素数的算法的 N-S 图如图 1-5 所示。

用流程图和 N-S 图表示算法直观易懂，但画起来太费事。另一种表示算法的工具称为伪代码。伪代码是介于自然语言和程序设计语言之间的一种表示方法。通常用程序设计语言中的控制结构表示算法的流程，用自然语言表示其中的一些操作。如果采用 C 语言的控制结构，则称为伪 C 代码。如果采用 PASCAL 语言的控制结构，则称为伪 PASCAL 代码。例如，判断整数 n 是否为素数的算法用伪 C 代码表示如下：

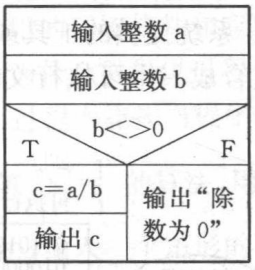

图 1-4 整数除法的 N-S 图表示

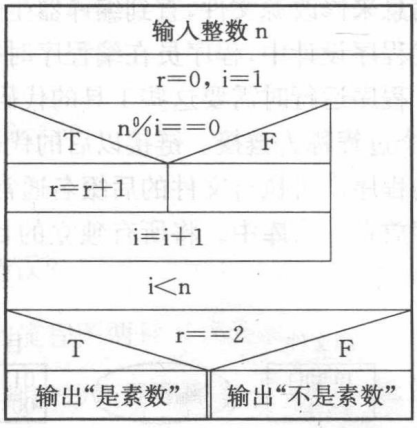

图 1-5 判断整数 n 是否为素数

```
输入 n
设 r = 0
for(i = 1; i <= n; ++i)
  if(n % i == 0)++r
if(r == 2)输出“n 是素数”
else 输出“n 不是素数”
```

伪代码书写格式比较自由，容易表达算法设计的思想，也容易转换成真正的代码。本书后面的章节中的算法都将采用伪代码的表示方法。

1.4 编码

编码是将算法用具体的程序设计语言的语句表达出来，所以我们必须学习一种程序设计语言。本书采用的是 C++语言。用程序设计语言描述的算法称为**程序**，存储在计算机中的程序称为**源文件**。C++的源文件名的后缀必须是“.cpp”。输入程序或修改程序内容的过程称为文件的**编辑**。各个计算机系统的编辑过程差异很大，不可能用一种统一的方式来描述，因此在编辑源文件之前，必须先熟悉所用的机器上的编辑方法。很多操作系统也提供一些综合编程的环境，如 VC2010 就是 Windows 系统提供的一个 C++的综合编程环境，为程序员提供了从源文件编辑到程序运行过程中的所有环节。

1.5 编译与链接

为了让高级语言编写的程序能够在不同的计算机系统上运行，首先必须将程序翻译成该计算机特有的机器语言。这个过程也因机器而异。例如，若为 Macintosh 机器写一个 C++的程序，这将需要运行一个特殊的程序，该程序将 C++语言写的程序转换成 Macintosh 的机器语言；如果在 IBM 的 PC 机上运行该程序，则需要使用另一个翻译程序。在高级语言和机器语言之间执行这种翻译任务的程序叫作**编译器**。经过编译器翻译得到的机器语言的程序称为**目标程序**。存储目标程序的文件称为目标文件，它的后缀名通常为“.obj”。

在编译过程中，编译器会找出源文件中的语法错误和词法错误。程序员可根据编译器输出的出错信息来修改源文件，直到编译器生成了正确的目标代码。

在现代程序设计中，程序员在编程序时往往会用到系统已经做好的工具或其他程序员做好的工具。程序运行时需要这些工具的代码。于是需要将目标文件和这些工具的目标文件放在一起，这个过程称为**链接**。链接以后的代码称为一个**可执行文件**。这是能直接在某台计算机上运行的程序。可执行文件的后缀名通常是".exe"。系统提供的工具或用户自己写的一些工具程序存放在一个**库**中。将所有独立的目标文件组合成一个可执行文件的过程如图 1-6 所示。

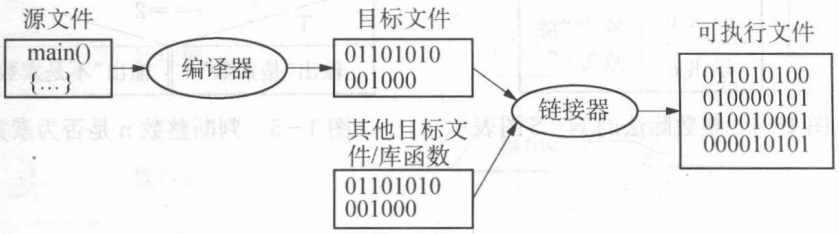

图 1-6　编译链接过程

1.6　程序调试和维护

可执行文件是一个可以运行的程序。在命令行界面下输入可执行文件名或在 Windows 界面下双击该文件的图标就可以运行该程序。可是开始时程序通常给出了不正确的结果或干脆异常终止，这是由于程序中存在一些逻辑错误。例如，算法设计有问题或有些特殊情况没有考虑。程序员称这种错误为 bug。找出并改正这种逻辑错误的过程称为**调试**(debug)，它是程序设计过程中一个重要的环节，也是程序设计中的一个难点。逻辑错误非常难以察觉。有时程序员非常确信程序的算法是正确的，但随后却发现它不能正确处理以前忽略了的一些情况；或者也许在程序的某个地方做了一个特殊的假定，但随后却忘记了；又或者可能犯了一个非常愚蠢的错误。调试一般需要运行程序，通过观察程序的阶段性结果来找出错误的位置和原因。

程序的调试及测试只能发现程序中的错误，而不能证明程序是正确的。因此，在程序的使用过程中可能会不断发现程序中的错误。在使用时发现错误并改正错误的过程称为程序的**维护**。

1.7　小结

本章主要介绍了程序设计所需的下列基础知识和基本概念：

- 什么是程序设计。
- 程序设计的过程及每个阶段的工作。
- 程序设计语言提供的基本功能。

1.8　习题

1. 了解并掌握一种 C++的综合编程环境，如 VC2010。

2. 所有的计算机能够执行的指令都是相同的吗?

3. 投入正式运行的程序就是完全正确的程序吗?

4. 为什么需要编译?为什么需要链接?

5. 调试的作用是什么?如何进行程序调试?

6. 为什么一个C++程序可以在不同的机器上运行?

7. 试列出一些常用的系统软件和应用软件。

8. 程序设计语言为什么要提供过程单元?

9. 为什么debug过程不能找出程序中的所有错误?

10. 设计一个计算 $\sum_{i=1}^{100}\frac{1}{i}$ 的算法,用N-S图和流程图两种方式表示。

11. 设计一个算法,输入一个矩形的两条边长,判断该矩形是不是正方形。用N-S图和流程图两种方式表示。

12. 设计一个算法,输入圆的半径,输出它的面积与周长。用N-S图和流程图两种方式表示。

13. 设计一个算法,计算下面函数的值。用N-S图和流程图两种方式表示。

$$y=\begin{cases}x & (x<1)\\ 2x-1 & (1\leqslant x<10)\\ 3x-11 & (x\geqslant 10)\end{cases}$$

14. 设计一个求解一元二次方程的算法。用N-S图和流程图两种方式表示。

15. 设计一个算法,判断输入的年份是否为闰年。用N-S图和流程图两种方式表示。

16. 设计一个算法计算出租车的车费。出租车收费标准为:3千米内收费12元;3~10千米之间,每千米收费2.4元;10千米以上,每千米收费3元。用N-S图和流程图两种方式表示。

2 程序的基本组成

一个程序就像是一篇实验指导书，让计算机按你的实验指导书一步步往下做，最终就能完成任务。实验指导书有实验指导书的格式，程序也有程序的格式。本章将从一些简单的程序出发，介绍程序的基本框架及组成程序的最基本的元素。

2.1 程序的基本结构

考虑一下设计一个求解一元二次方程的程序。如第 1 章所述，设计一个程序先要设计算法。如何教会计算机解一元二次方程？我们自己会有很多解一元二次方程的方法，例如，凑一个完全平方公式、平方差公式或用十字相乘法因式分解。但要教计算机，这些方法略显复杂。最简单的是用标准的公式 $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ 解一元二次方程。在众多的解一元二次方程的方法中，我们选择教计算机用标准公式求解，然后开始设计教学过程，即算法。这个算法很简单，它的 N-S 图描述如图 2-1 所示。根据这个算法得到的 C++程序如代码清单 2-1 所示。

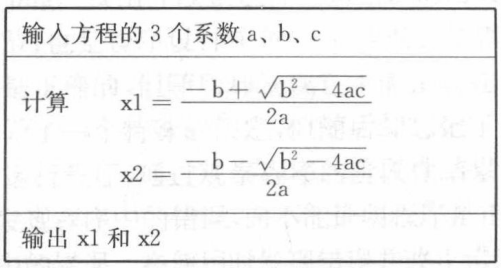

图 2-1 求解一元二次方程的算法

代码清单 2-1 求解一元二次方程

```
//文件名：2-1.cpp
//用标准公式求解一元二次方程
```
注释

```
#include <iostream>
#include <cmath>
```
编译预处理指令

```
using namespace std;
```
使用名字空间

```
主程序
int main()
{   double a, b, c, x1, x2, dlt;}   变量定义
    cout << "请输入方程的 3 个系数:" << endl;
    cin >> a >> b >> c;                          } 输入阶段
    dlt = b * b - 4 * a * c;
    x1 = (-b + sqrt(dlt)) / 2 / a;               } 计算阶段
    x2 = (-b -sqrt(dlt)) / 2 / a;
    cout << "x1 = " << x1 << "x2 = " << x2 << endl;}   输出阶段
    return 0;
}
```

一个C++程序由注释、编译预处理和主程序组成。主程序由一组函数组成。每个程序至少有一个函数,这个函数的名字为main。

2.1.1 注释

代码清单2-1最前面的2行是注释行。一般来说,每个程序都以一个专门的、从整体描述程序操作过程的注释开头,称为**程序注释**。它包括源程序文件的名称、作者和一些与程序操作有关的信息。程序注释还可以描述程序中特别复杂的部分,指出可能的使用者,给出如何改变程序行为的一些建议,等等。注释也可以出现在主程序中间,解释主程序中一些比较难理解的部分。在C++语言中,注释是从//开始到本行结束。在C++程序中也可以用C语言风格的注释,即在/*与*/之间所有的文字都是注释,可以是连续的几行。

注释是写给人看的,而不是写给计算机的。与程序执行的结果完全无关。它们是向其他程序员传递该程序的有关信息。C++语言编译器在编译程序时,注释被完全忽略。

初学者往往把握不好程序中哪些地方要添加注释,该添加什么内容的注释。你可以在写好一个程序后,把它放在一边,过几天再去读它。如果某些地方你自己一下子都不能理解,那么这个地方就需要添加注释。

因为注释并不是真正可执行的部分,所以很多程序员往往不愿意写,但注释对将来程序的维护非常重要。给程序添加注释是良好的编程风格。

2.1.2 编译预处理

C++的编译分成两个阶段:预编译和编译。先执行预编译,再执行编译。预编译阶段处理程序中的编译预处理指令,就是那些以#开头的指令。如代码清单2-1中的#include <iostream>。编译预处理命令有很多,常用的编译预处理指令主要是**库包含**,即include指令。其他的编译预处理命令将在用到时介绍。

库包含表示程序使用了某个库。**库**是实用工具的集合,这些实用工具是由其他程序员编写的,能够完成特定的功能。iostream是C++本身提供的标准输入/输出库。程序中所有数据的输入/输出都由该库提供的功能完成。本书的每个程序基本上都会用到这个库。cmath是数学函数库,包含了一些常用的科学计算函数的实现。由于求解一元二次方程时要用到求平方根,该函数包含在cmath库中,所以也要包含cmath库。库对于程序设计来说是十分重要的,当你开始编写一些较复杂的程序时,马上将会依赖一些重要的库。