免费提供电子教案

高等院校规划教材
计算机科学与技术系列

C/C++程序设计技术

陈卫卫　王庆瑞　编著

机械工业出版社
CHINA MACHINE PRESS

高等院校规划教材·计算机科学与技术系列

C/C++程序设计技术

陈卫卫　王庆瑞　编著

机械工业出版社

本书是用于学习C/C++语言程序设计技术的教科书。全书共7章，主要内容包括：C语言基础，分支和循环，构造类型，函数，指针，类和对象的概念和设计方法，Visual C++ 6.0上机操作方法，以及用附录形式给出的ASCII码表、常用库函数表、部分习题参考答案等。

本书以基本语法规则为线索，通过126个例题和460多道习题（连同一题多问的小题，共700多道），向读者传授程序设计技术。

本书可作为普通高校计算机科学与技术专业语言课程教材，也可作为广大电脑爱好者学习程序设计方法的参考书。

图书在版编目（CIP）数据

C/C++程序设计技术/陈卫卫，王庆瑞编著. —北京：机械工业出版社，2008.6
（高等院校规划教材·计算机科学与技术系列）
ISBN 978-7-111-24339-7
Ⅰ.C… Ⅱ.①陈… ②王… Ⅲ.C语言-程序设计-高等学校-教材
Ⅳ.TP312
中国版本图书馆CIP数据核字（2008）第088105号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）
责任编辑：唐德凯
责任印制：杨 曦
三河市宏达印刷有限公司印刷

2008年7月第1版·第1次
184mm×260mm·21.25 印张·521千字
0001-5000册
标准书号：ISBN 978-7-111-24339-7
定价：34.00元

凡购本书，如有缺页，倒页，脱页，由本社发行部调换
销售服务热线电话：（010）68326294
购书热线电话：（010）88379639 88379641 88379643
编辑热线电话：（010）88379753 88379739
封面无防伪标均为盗版

出版说明

计算机技术的发展极大地促进了现代科学技术的发展，明显地加快了社会发展的进程。因此，各国都非常重视计算机教育。

近年来，随着我国信息化建设的全面推进和高等教育的蓬勃发展，高等院校的计算机教育模式也在不断改革，计算机学科的课程体系和教学内容趋于更加科学和合理，计算机教材建设逐渐成熟。在“十五”期间，机械工业出版社组织出版了大量计算机教材，包括“21世纪高等院校计算机教材系列”、“21世纪重点大学规划教材”、“高等院校计算机科学与技术‘十五’规划教材”、“21世纪高等院校应用型规划教材”等，均取得了可喜成果，其中多个品种的教材被评为国家级、省部级的精品教材。

为了进一步满足计算机教育的需求，机械工业出版社策划开发了“高等院校规划教材”。这套教材是在总结我社以往计算机教材出版经验的基础上策划的，同时借鉴了其他出版社同类教材的优点，对我社已有的计算机教材资源进行整合，旨在大幅提高教材质量。我们邀请多所高校的计算机专家、教师及教务部门针对此次计算机教材建设进行了充分的研讨，达成了许多共识，并由此形成了“高等院校规划教材”的体系架构与编写原则，以保证本套教材与各高等院校的办学层次、学科设置和人才培养模式等相匹配，满足其计算机教学的需要。

本套教材包括计算机科学与技术、软件工程、网络工程、信息管理与信息系统、计算机应用技术以及计算机基础教育等系列。其中，计算机科学与技术系列、软件工程系列、网络工程系列和信息管理与信息系统系列是针对高校相应专业方向的课程设置而组织编写的，体系完整，讲解透彻；计算机应用技术系列是针对计算机应用类课程而组织编写的，着重培养学生利用计算机技术解决实际问题的能力；计算机基础教育系列是为大学公共基础课层面的计算机基础教学而设计的，采用通俗易懂的方法讲解计算机的基础理论、常用技术及应用。

本套教材的内容源自致力于教学与科研一线的骨干教师与资深专家的实践经验和研究成果，融合了先进的教学理念，涵盖了计算机领域的核心理论和最新的应用技术，真正在教材体系、内容和方法上做到了创新。同时本套教材根据实际需要配有电子教案、实验指导或多媒体光盘等教学资源，实现了教材的“立体化”建设。本套教材将随着计算机技术的进步和计算机应用领域的扩展而及时改版，并及时吸纳新兴课程和特色课程的教材。我们将努力把这套教材打造成为国家级或省部级精品教材，为高等院校的计算机教育提供更好的服务。

对于本套教材的组织出版工作，希望计算机教育界的专家和老师能提出宝贵的意见和建议。衷心感谢计算机教育工作者和广大读者的支持与帮助！

机械工业出版社

前　言

作者根据多年的语言教学经历，总结出这样的经验：要学好用好语言，必须把握好“记、读、仿、练、操”5个环节。

记，是指学生先要粗记基本语法和程序框架，然后再通过上机练习，在理解中加强记忆和巩固，以起到事半功倍的功效。尤其是对那些内容繁杂的、比较抽象和“绕人”的内容，如输入/输出格式、函数参数传递、指针等不能盲目地死记。

读，就是熟读教材中的示例程序，细心体会设计方法和技巧。大多数例题都有一定的代表性和渐近性，需要熟读。

仿，就是在第二步“熟读”的基础上，多模仿示例，编写那些与示例内容相近、结构相近的程序，逐步“仿造”出“好”程序来。

练，就是多做练习题，特别要独立完成程序跟踪和程序填空题。程序跟踪是用人脑模拟电脑跟踪程序的执行，对巩固语法规则很有帮助。完成程序填空题，需要弄清程序的功能和大体结构，根据上下文，“猜出”应该填写的语法成份，这对提高学生的程序思维能力大有好处。

操，即上机操作。在纸上编写的程序是“静止”的“死”程序，只有上机操作，才能让程序“动”起来，“活”起来，从而逐步学会如何在调试过程中，找出程序中的语法错误和逻辑错误。只有学会了在机器上编程并调试，才算“真正的”学会了编程。

作者正是按照如何在教学中紧扣上述的5点打造本书的，并将这一教学理念融入在内容的编排、取舍和组织，示例程序的选配，以及习题的设置，甚至参考答案的挑选等各个方面。本书不仅是作者长期教学经验的结晶，也是作者几本先期出版的相关教材内容的整合。本书的主要特色如下。

1. 全书以程序设计方法为主，将语言的语法概念作为支撑程序设计的工具，而不是单纯拿语法“说事”。重点章节，如分支、循环、数组、结构、函数、指针都用单独一节介绍程序设计示例，连同习题的示例程序，一起作为正文内容的补充和延伸。

每个例题和习题都经过精心挑选，对题中的程序进行了精心设计，使其具有良好的程序结构和程序设计风格。有的习题先后多次出现，以体现不同的解法（如是否用数组，是否用函数，是否递归，是否用指针，是否用文件等）。通过这些范例，展示如何用语法去描述问题和求解问题，借此向读者传授程序设计方法和技术。

2. 适当地弱化语法概念，缩减单纯语法所占篇幅。大量与语法概念有关的基本概念渗透在示例和习题中，为教师留下较大的教学空间，以突出重点，强调语言的应用。

3. 将相关内容融合在一起，既体现出共性，压缩了篇幅，也便于归纳和总结。例如，运算符和表达式、构造类型、参数传递方式等。

4. 以C语言为主，将C++“好用的”语法成分（如行注释、cin、cout和传引用等）穿插其中，使学生从一开始就养成编写结构良好程序的习惯。特别是“传引用”的参数传递方式，对“净化程序”和增强可读性起到“不可代替的”作用。

5. 习题量大，题型多（包括一般概念题、选择题、改错题、程序填空题、程序跟踪题、程序设计题等），有层次，而且自成体系，能够起到对教学内容“消化、细化、深化”的作

用，并选定一些对巩固课堂知识具有典型作用的编程题作为上机练习。本书在附录部分还给出部分习题的参考答案。

6. 简略地介绍了 C ++ 的面向对象程序设计机制，使学生了解面向对象程序设计的基本原理，并为以后掌握面向对象的程序设计方法打下基础。

在全书的 7 章中，第 3、4、5、6 章是最基本也是最重要的内容，第 1 章和第 2 章中的上机环境用法是辅助内容，第 7 章是引申的提高内容，标有“ * ”的章节为选讲（或略讲）内容。

本书所有例题和习题中的程序、程序段都在 Visual C ++ 6.0 环境下测试通过。

本书可作为普通高校计算机科学与技术专业语言课程教材和教学参考书，也可作为广大电脑爱好者学习程序设计方法的参考书。

欢迎广大读者对本书提出宝贵意见。本书提供的电子教案可在网站上免费下载，网址为 www. cmp. edu. com。作者可向选用本书的教师提供全部习题答案，作者的邮箱是 lvkeyi@126. com。

在此，对曾经为本书的编写和出版给与帮助、支持和鼓励的所有人士表示感谢。

作　者

目　录

第1章 基础知识

1.1 程序设计语言的发展和分类

众所周知,计算机是在程序的控制下自动工作的,要让计算机完成某项任务,必须为其设计相应的计算机程序。编写计算机程序(简称编程)必须使用程序设计语言。程序设计语言则是人和机器都能"懂得"(理解)的一种语言,是人与计算机交流,并指挥计算机工作的工具。

由于计算机中直接参与计算的部件——运算器和控制器等,都是由逻辑电路构成的,而逻辑部件只"认识"0和1,所以程序的最终形式都是由0和1组成的二进制代码形式(指令序列)。这种二进制代码形式的语言称为机器语言。

早在计算机诞生之初,人们就是用机器语言编程的。但是,这种在计算机看来十分明了的机器语言程序,在人看来却是一部"天书"。后来,人们又将3个二进制位合并在一起,这就形成了八进制,再后来,为了与字节对应,又将4个二进制位合并在一起,就变成了十六进制。将机器语言程序写成八进制或十六进制形式,要比二进制形式"好看"多了。

不管二进制、八进制,还是十六进制,用数字表示程序都不直观,不仅专业性极强,且非常难读难用,编程工作效率低,且极易出错。好在当初计算机应用面很窄,编程工作量不大,矛盾并不十分突出。

随着计算机应用面不断地扩大,程序需求量大增,编程工作量也越来越大,人们便产生了用符号(通常选用英文字词的缩写)代表机器指令(称为硬指令)的想法,设计出汇编语言(Assemble Language,又称符号语言)。比如,用ADD表示加法指令,用SUB表示减法指令等,要比形如"00111011"表示某条指令直观得多。人们将汇编语言编写的程序(称汇编源程序)送入计算机,再由计算机中的汇编程序将源程序自动翻译成计算机能够直接执行的二进制程序(目标程序,可执行程序)。

汇编程序(Assembler,又称汇编器)是专门用来将汇编源程序翻译成机器指令程序的软件。当然,它也是人们事先编写好,并安装在计算机系统中供反复使用的。一台计算机配上了汇编程序就相当于人们"教会"计算机认识汇编语言了。

汇编程序把人容易理解的汇编源程序转变成了计算机可直接执行的目标程序。再后来,人们又设计出反汇编程序,它能将机器语言程序反过来翻译成汇编语言程序。通过反汇编,人们就可以读懂安装在计算机中的可执行程序。

使用汇编语言减轻了人们不少的编程工作量,但是,汇编语言仍然十分原始,一条汇编语句(也称汇编指令)对应一条机器指令,易读性仍然很差。编制一个程序,哪怕只是用来完成简单计算任务的程序,通常需要成百上千条汇编指令。不仅编程效率低,程序不易调试,而且容易出错。更为麻烦的是,这种语言是完全按照计算机硬件设计的,不同种类的计算机都有自己特有的机器语言和汇编语言,一种类型的机器无法识别另一种类型机器的机器语言,所以,汇编源程序缺乏可移植性。

人们把机器语言和汇编语言归属为低级语言。

汇编语言的出现具有划时代的意义,它启发人们,可以设计更好用的语言,只需要通过翻译器,将新语言的源程序翻译成机器语言程序。于是,人们期待的,脱离计算机机种的高级程序设计语言(以下简称高级语言)便陆续被设计出来。

第一个高级语言是20世纪50年代中期出现的FORTRAN(FORmula TRANslator)语言,它的取名就意味着它所起的角色(翻译器)。该语言特别适用于编写科学计算(即纯数值计算)的程序,直到今天,仍然在发挥着作用,可见其生命力之强大。

后来人们又相继设计出用于商业事务处理的COBOL语言(Common Business Oriented Language),适合于算法描述的算法语言ALGOL(ALGOrithmic Language),面向初学者的BASIC语言(Beginner's All-purpose Symbolic Instruction Code),PL/1语言(Programming Language/1),以及Niklaus Wirth根据结构化程序设计思想设计的PASCAL语言(Philips Automatic Sequence CALculator,同时也为了纪念最早实用计算器的发明者、数学家Blaise Pascal而命名)等,这些语言在程序设计语言的发展历程中,都起到十分重要的作用。

高级语言引入了变量、数组、分支、循环、子程序,以及接近数学语言的表达式等语法成分,用接近英语口语的语句描述处理步骤(例如,if …then … else …),不仅容易理解和记忆,而且一条语句的处理能力相当于几条、几十条,甚至几百条汇编指令,大大地提高了编程效率。

高级语言有两种形式:一种是编译型的,另一种是解释型的。

用编译型的高级语言编写的源程序(源程序也称源代码)经过编译程序(Compiler,又称编译器)对其编译,产生出机器语言程序(称目标程序),再将一个或几个目标程序与标准库函数程序连接起来,最终构成一个完整的可执行程序。FORTRAN、PASCAL和C等语言都属于编译型的。

BASIC是典型的解释型高级语言,采用解释的方法去执行源程序中的语句。通过解释程序(Interpreter,又称解释器)对源程序中的语句边解释边执行,而不产生目标程序文件。目前最流行的网络编程语言Java也属于解释型高级语言。

编译和解释的最大区别是:前者得到一个完整的机器语言程序,执行时可以脱离翻译环境,所以运行速度快;后者则不能脱离解释器单独执行,因而执行速度慢。

另外,高级语言又可以分为两大类,一类是面向过程的,另一类是面向对象的。凡提供面向对象程序设计手段的语言都属于面向对象的,否则,属于面向过程的。

早期的高级语言(如PASCAL、ALGOL和C等)都是面向过程的(不提供面向对象程序设计手段)。这类语言基于数据类型定义,和处理数据的过程(或函数)定义。数据和处理数据的过程(或函数)分离。数据在其作用域内,可被任何过程(或函数)访问,使数据缺乏保护机制。

面向对象的语言提供的机制支持面向对象的程序设计方法,它以类(实际上是类的对象)作为基本单位,类中含有被处理的数据定义和处理数据的代码(即操作语句)定义。用面向对象的方法设计的程序中,没有游离于类之外的对类中数据的处理操作,不能绕过类访问类中的数据,也就是说,在类的外部只能通过类访问和处理类中的数据。外部只知有类,但不知(也不必知道)类是如何处理的。由于类把数据和处理这些数据的代码严密地封装在一起,所以对外完全屏蔽类内部的处理细节,对类中的数据起到了很好的保护作用,易于保证数据的完整

性和一致性(详见第7章)。

20世纪60年代开发的Simula 67语言提出了对象和类的概念,被认为是面向对象语言的鼻祖。20世纪70年代出现的ADA语言(为纪念第一位有文字记载的女程序员Augusta Ada Lovelace而命名)支持抽象数据类型,是基于对象的语言。但由于它并不全面支持继承,所以仍算不得真正面向对象。再后来的Simulatalk语言和Java语言进一步丰富了面向对象的概念,将信息隐藏得更加严密,用向对象(和对象之间)发送信息的方式进行程序设计。同样,C++语言也支持类和对象,所以,也属于面向对象的程序设计语言。

另外,为了满足不同行业、不同人群的需要,人们还设计出各种各样的专用语言(比如,数控语言、各种辅助设计语言、数据库操作语言),以及专门用于软件开发的各种开发工具(有的也属于编程语言)等,这里不能一一列举。

总之,语言的发展促进了编程技术的发展,而编程技术的发展和编程要求同样也激励着编程语言的发展。易学、好用、功能强大,对使用者所应具备的计算机专业知识要求不高,是未来语言的发展方向。

1.2 程序的基本结构和流程

1.2.1 程序的基本结构

程序是计算机能够执行的指令(或语句)序列。一个完整的程序,总是由一个主程序和若干个(也可没有)子程序组成。不同的语言对子程序的叫法有所不同,有的叫过程,有的叫函数,有的叫子例程。为叙述方便,下面将主程序和子程序统称为程序块。

程序结构分为物理结构和逻辑结构两个方面。

物理结构指程序外在形式,即程序块与程序块之间的组织关系。主要有两种方式,一种是模块式结构,一种是层次式结构(或称嵌套式结构),不同的语言,其源程序的物理结构不同。

对于模块式结构,主程序和各个子程序自成一块,块与块并列排放,排列次序可能有关,也可能无关。但块与块不能嵌套,即一个程序块中不能包含另一个程序块。FORTRAN、BASIC和C语言都属于模块式结构。

对于层次式结构,主程序中可以嵌套定义若干个一级子程序;一级子程序中,又可以嵌套定义若干个下一级子程序……同级子程序块并列排放。主、子程序一层层地嵌套排列,主程序处于最外层。ALGOL和PASCAL语言就属于层次式结构。

图1-1为程序物理结构示意图。其中,图1-1a是模块式,图1-1b是层次式。

程序的逻辑结构指的是同一程序块内语句的执行次序,以及块与块的执行次序。

同一程序块内的语句总是顺序排列的,执行时,也总是从第一条语句开始一条一条地依次执行。但是,当执行一条控制语句(比如,条件语句、转移语句、循环语句)时,就会改变语句的执行次序(改变程序的流程方向)。

有四种不同的流程结构:顺序结构、分支结构、循环结构和子程序调用结构。其中,前三种属于块内结构,第四种是块间的调用关系。下面分别用流程图描述这四种结构。

处在顺序结构中的语句,执行时,严格按照语句的先后次序逐条执行,其流程图如图1-2所示(箭头表示流程方向),即执行次序是:语句1,语句2,语句3。

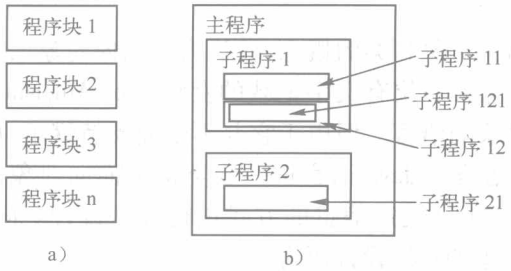

图 1-1　程序物理结构示意图

a）模块式结构　b）层次式结构

图 1-2　顺序结构流程图

分支结构有二分支和多分支两种，参见图 1-3。

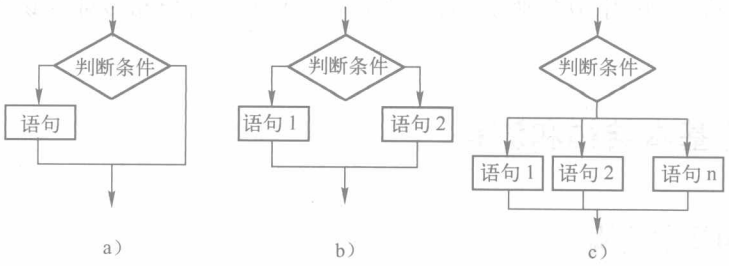

图 1-3　分支结构流程图

a）二分支结构之一　b）二分支结构之二　c）多分支结构

分支结构的前部通常有一条用于测试判断条件的语句，当程序执行到该语句时，就对事先设置在程序中的判断条件进行测试，根据测试结果，从两个或多个分支中选择一个分支执行（其他分支本次不被执行）。例如，C 语言中的 if 语句和 switch 语句（分别属于二分支结构和多分支结构）。

循环结构也称重复结构。处在循环结构中的语句称为循环体，在循环控制条件的控制之下，循环体可以反复执行。循环结构用于反复处理相同或相似的计算步骤。

循环控制条件（带有测试条件）通常设在循环结构的前部（当型循环）或后部（直到型循环），如图 1-4 所示。每当准备执行循环体时，都要测试循环控制条件，根据测试结果，确定是再次执行循环体，还是退出循环结构。

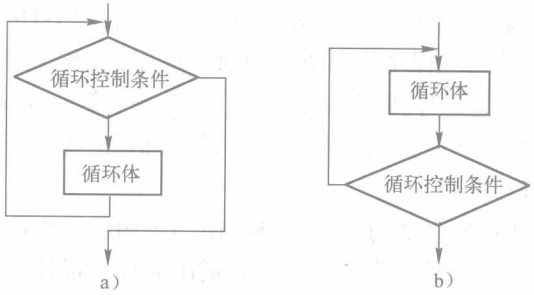

图 1-4　循环结构流程图

a）当型循环结构　b）直到型循环结构

执行当型循环结构时，先测试控制条件，然后根据测试结果，确定是否再次（包括第一次）执行循环体，其意是当控制条件满足时就执行一次循环体（如此反复）。

执行直到型循环结构时，先执行一次循环体，然后测试循环控制条件，确定是否再次执行循环体。

顺序、分支和循环三种结构可以穿插出现在同一程序块中，而分支结构、循环结构中也可以含有顺序结构、分支结构和循环结构，就是说，三种结构可以相互叠加，相互嵌套，形成复杂的程序结构。

子程序调用决定程序块之间的执行次序。

无论模块式结构，还是层次式结构，总是从主程序的第一条语句开始执行，当执行完主程序的最后（指逻辑上的最后）一条语句时，程序便正常终止。其间，若执行到子程序调用语句，则控制（流程）从主调程序的调用点转移到被调程序，开始执行被调用程序。执行完被调程序后，返回到主调程序的调用点，继续执行主调程序中调用语句下面的语句，如图 1-5 所示。

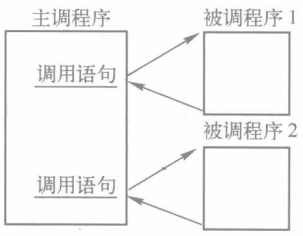

图 1-5　子程序调用流程图

主程序可以调用子程序，子程序可以调用本身（递归调用）或别的子程序。无论子程序还是主程序，都不能调用主程序。

对于模块式结构，各子程序都是并列的，平等的，它们可以相互调用。

对于层次式结构，外层程序可以调用内层程序，内层程序不能调用外层程序；并列的同层程序可以相互调用。

程序调用可以是嵌套的，也可以是递归的。

程序块 A 调用程序块 B，程序块 B 又调用程序块 C，从而形成嵌套调用。程序调用可以层层嵌套，有时可达几十层或更多层。

一个程序，直接或间接调用自身，就形成递归调用。处于递归调用“圈”内的程序都属于递归程序。由于主程序不能被任何程序调用，所以主程序不能是递归的，只有子程序才可能是递归的。

1.2.2　程序设计的基本步骤

归纳起来，设计一个程序通常需要经过确定算法（即求解问题的方法）、编程实现、程序调试、投入试运行、日常维护和升级优化等多个阶段，而且，其间可能还需要几经反复。

进一步说，在为求解某问题编制程序时，首先要确定求解问题的算法，再根据算法中描述的求解步骤，选择适当的程序设计语言，按照算法写成程序（编程实现），然后将程序上机编译并调试，生成可执行文件，将生成的可执行文件投入试运行。在试运行阶段，注意搜集用户意见，找出程序的不足之处或隐蔽性错误，对程序进行修改、维护和优化，再投入运行……如此周而复始，直到满意为止。

下面通过一个示例，对确定算法、编程实现及程序优化等步骤进行简单介绍，而程序调试方法将在 1.3.4 小节介绍，其他步骤从略。

【例 1-1】　将一个正整数 n 分解成质因子连乘形式。

例如,16、17 和 18 的质因子分解式为:

16 = 2 × 2 × 2 × 2
17 = 17
18 = 2 × 3 × 3

求解这个问题时,大多数人会想到,用不超过 n 的每个质数 x 试除 n,如果能除尽(余数等于 0),x 就是 n 的一个质因子,这样可以一个个地求出 n 的所有质因子。

当然,为了有一个可循的章法,最好从小到大一个个找出 n 的质因子 x。因为 2 是最小的质数,所以,开始时设 x = 2。

按照这个思想,写出算法如下:

第 1 步,x = 2;
第 2 步,如果 x 大于 n,则算法结束,否则继续下一步;
第 3 步,如果 x 是质数,而且 n/x 所得余数等于 0,则(否则执行第 4 步):
  第 3-1 步,输出 n 的一个质因子 x;
  第 3-2 步,用 n/x 所得的商取代原来的 n;
  第 3-3 步,返回第 3 步(循环处理);
第 4 步,增大 x 的值;
第 5 步,返回第 2 步(再进行循环处理)。

算法只是程序的雏形,编程时,必须不折不扣地将算法各步骤写成准确的程序语句或程序段。

下面介绍如何用 C/C ++ 语言实现上述算法,也就是,如何将算法的每一步变换成 C/C ++ 语言的语句或程序段。

算法的第 1 步容易实现(本身就是一条语句),即“x = 2;”。

算法的第 2 步与第 5 步合起来构成循环,可写成:

```
while(x <= n)
{
  循环体是第 3 步与第 4 步的处理步骤;
}
```

下面分析一下如何实现第 4 步“使 x 的值增大”。

如果要求 x 必须是质数,那么,第一次(x 的初值)x = 2,下一次 x = 3,再以后,x = 5,x = 7,x = 11……。x 值的变化并无简单规律,实现较困难。

为了便于实现,第 4 步可让 x 的值每次加 1。可以写成语句:“x = x + 1;”,“x += 1;”或“x ++;”。这种改动不会影响执行结果。

再考虑第 3 步的实现方法。第 3 步中的条件“如果 x 是质数”可以忽略(请思考为什么)。条件“n/x 所得余数等于 0”很容易用取余运算 % 实现。其余语句都不难实现。

整个第 3 步可以写成:

```
if(n%x==0) {cout << x << '*';n=n/x;}
```

于是,合起来,完成此任务的程序段为:

```
x=2;
```

```
while(x <= n)
 {
  if(n%x ==0){cout << x << '*';n = n/x;}
  else x ++;
 }
```

如果要求不高,只要配上必要的变量定义等语法成分,和输入数据的语句,就是一个完整的程序了。

但是,经过分析,对这段程序还有优化的余地,可以减少循环执行次数,加快处理速度。

首先,根据数学知识,没有必要让 x 从 2 一直变到 n,而只需要从 2 变到$\sqrt{n}$(请思考为什么)就行了。这样,循环语句"while(x <= n)"可改为"while(x * x <= n)",循环次数大为减少(从 n 次减为$\sqrt{n}$ 次)。

完整的程序如下:

```
#include <iostream.h>
void main()
{ int x,n;
  cout <<"请输入正整数 n 的值 n = ";
  cin >>n;
  cout <<n<< '=';
  x = 2;
  while(x * x <= n)
  {
  if(n%x ==0){cout << x << '*';n = n/x;}
  else x ++;
  }
  cout <<n<<endl;
}
```

实际上,循环次数还可以再减少一半。考虑语句"else x ++;",每次只将 x 的值加 1(速度太慢),第一次执行该语句时是必要的(x 的值从 2 变到 3),但其后,x 每次至少加 2 才有效。

如果将"else x ++;"改为"else if(x ==2)x ++;else x +=2;"看似减少了循环次数,但是,由于该语句出现在循环体中,每次执行循环体,都要判断"x 是否等于 2",循环次数并未减少。

改进的方法是,在进入循环之前,先提取(输出)n 的所有"2"因子,然后将 x 的初值定为 3,再进入循环时,x 的值每次加 2。

按照这一思想,写出的程序(比较完善)如下:

```
#include <iostream.h>
void main()
{ int x,n;
  cout <<"请输入正整数 n 的值 n = ";
  cin >>n;
  cout <<n<< '=';
```