程序设计实践指导

朱明方　赵莼善　雷田玉　编著

清华大学出版社

(京)新登字 158 号

内 容 提 要

本书旨在帮助广大读者在程序设计的实践中掌握程序设计的基本知识,提高程序设计的能力。全书共分 7 章。第 1 至 4 章是程序设计的基础训练。第 1、2 章归纳了程序设计的基本要求和基本方法,第 3 章简要介绍了常用算法和数据结构,第 4 章总结了程序调试与测试的主要方法。第 5 至 7 章以电路和力学中的一些内容为背景,给出应用问题程序设计的思路和关键步骤。第 5 章重点归纳了自动建立电路方程的常用方法,第 6 章简要介绍了非线性电阻电路的求解问题,第 7 章结合力学中的一些例子介绍了一般工程领域中的程序设计问题。全书各章内容相对独立,并给出相应的例题和练习题,读者可以根据具体情况灵活选择学习与练习的内容。

本书可以作为大专院校非计算机专业的学生进行程序设计训练的教材,也可作为广大科技工作者的自学参考书。

图书在版编目(CIP)数据
程序设计实践指导/朱明方等编著 .- 北京: 清华大学出版社,1996
ISBN 7-302-02344-1

Ⅰ. 程… Ⅱ. 朱… Ⅲ. 程序设计-方法 Ⅳ. TP311.11

中国版本图书馆 CIP 数据核字 (96) 第 19392 号

出版者: 清华大学出版社 (北京清华大学校内,邮编 100084)
印刷者: 北京市清华园胶印厂
发行者: 新华书店总店北京科技发行所
开 本: 787× 1092 1/16 印张: 14.75 字数: 350 千字
版 次: 1997 年 1 月 第 1 版 1997 年 1 月 第 1 次印刷
书 号: ISBN 7-302-02344-1/TP・1165
印 数: 0001—6000
定 价: 15.60 元

前　言

实践是学习程序设计的重要环节，要掌握好程序设计的基本方法和技术，需要不断丰富理论知识，更需要不断积累实践经验。因此，在学习了程序设计的基本知识并进行了初步的上机练习之后，还需要进行大量的程序设计的训练。通过这些训练，可以深入理解和掌握程序设计的基本知识、基本方法和技术，并把它们与应用问题结合起来，逐步积累经验，最终达到能够用计算机解决实际问题的目的。本书正是为这种训练的需要而编写的。

本书是在教学中多年使用的《程序设计训练习题与指导》讲义的基础上修改、补充而成的，以具有程序设计初步知识的读者为对象，内容包括从程序设计基本要求到实际应用问题程序设计所涉及的基本知识、方法和一些关键步骤，以及针对各章不同的训练要求所给出的典型示例。

第 1 至 4 章是程序设计的基础训练。第 1, 2 章针对初步学过程序设计语言的读者，归纳了程序设计的基本要求和基本方法。第 3 章概要地介绍了常用算法和数据结构，这一部分的训练要求读者具有计算机算法和数据结构的基本知识。第 4 章总结程序调试与测试的主要方法，这是程序设计的重要环节，就训练而言读者更多地要解决程序调试问题，因此读者更要注重程序调试能力的培养和训练。

第 5 至 7 章是针对实际应用问题的程序设计训练。考虑到通用性，这部分以应用较广泛的电路、力学中的一些知识为背景。第 5 章围绕分析、求解线性电路的程序设计问题，在归纳总结有关电路与网络图论的一些基本概念的基础上，着重讨论了求解电路的关键步骤——自动建立电路方程的常用方法。对于方程组的求解问题，因为第 3 章中已有介绍，此章不作重点讨论。第 6 章简单介绍了非线性电阻电路的数值求解方法——牛顿-拉夫逊法，以及由此建立的友网络模型。第 7 章结合力学方面的一些应用例子，说明一般工程领域中计算与数据处理方面的程序设计问题。

全书各章都给出了进行训练所需要的练习题，练习中的有些问题可以用具体的数据进行验证，这些验证所需要的数据由教师指定，或由练习者自行选定。

本书第 1 至 4 章由朱明方编写，第 5, 6 章由赵莼善编写，其中练习题部分由其所在的基本电工教研组电路教学组编写，第 7 章由雷田玉编写，全书由朱明方定稿。

本书的编写与出版得到了清华大学电子工程系教务科的鼓励和支持，得到了清华大学出版社的大力帮助，在此表示衷心的感谢。

由于编写者的水平所限，书中难免存在一些问题和错误，恳请读者批评指正。

作者

1996 年 4 月

目　　录

第 1 章　程序设计的基本要求

1.1　程序设计的基本概念和一般步骤

1.1.1　什么是程序设计

说起程序设计,初学者往往想到就是用计算机语言写出一些语句。然而,对程序设计的这种理解却是非常片面、很不完整的。实际上,写出程序的语句,只是程序设计的一部分工作,而且就工作量而言是很小的一部分工作。在程序设计过程中,写出程序之前及其以后都有许多重要的工作必须做。那么,究竟什么是程序设计呢?对于这个问题可以从不同的角度予以回答。

简单地说,程序设计是将事先确定的解题步骤,用计算机指令或计算机所能理解的语言描述出来的过程。这里,我们应理解为确定解题步骤及用计算机语言描述这两个方面。为了用计算机语言把求解问题的过程描述出来,在确定解题步骤时,一方面要反映问题本身的计算要求,同时又要考虑方便计算机语言的描述。而计算要求又必须建立在正确的计算模型上,因此,程序设计工作事实上从拿到问题的时候就开始了。它包括分析问题、建立数学模型、确定算法与数据结构、程序编写、程序测试等一系列的工作,直至交出一个可用的程序。正因为如此,我们说程序设计是一项涉及到多方面内容的工作,是把复杂问题的求解转换为计算机能执行的简单操作的过程。为了完成这个过程,除了对实际问题进行抽象以外,还要设计适合于计算机求解的算法,有了正确的算法才能编写出解决问题的程序。另一方面,程序的处理对象是数据,每个数据都有一定的特性,而且数据之间还会有一定的联系,因而当处理的数据对象比较复杂时,我们必须仔细地分析数据以及它们之间的联系,把它们合理地组织起来,即要选择合适的数据结构。确定的数据结构不同,程序中采用的处理方法也不一样。这也就是说,算法与数据结构是人们用计算机求解问题时所要作的两种抽象。算法是从计算机的操作出发对解题过程的抽象,数据结构是从对被处理数据对象的组织出发对问题的抽象。这两种抽象互相依赖,互相补充,从而有效地降低问题的复杂性,达到解决问题的目的。这就是程序设计的实质性工作和真正的涵义。著名计算机科学家沃思(Wirth)曾对程序作过精辟的描述:“程序就是在数据的某些特定的表示方式和结构的基础上,对抽象算法的具体描述。”他对程序的这个定义亦即指出了程序设计的实质。

在程序设计中,除了算法和数据结构以外,程序设计方法也是关系程序设计成败和影响程序设计质量的重要因素。好的程序设计方法可以使设计者思路清楚,程序结构清晰,可以用较短的时间设计出正确的程序,并且所设计的程序容易验证其正确性,容易扩充,便于维护。显然,易验证、易扩充性等都是程序设计的目标。

为了提高程序设计的效率和质量,设计出满意的程序,选择使用各种优质的软件工具、改善程序设计的工作环境是非常重要的。这些工具包括算法描述的工具、计算机语言、

程序输入、调试、测试等工具软件。正如用先进的仪器设备可以生产出先进的产品，可以完成高、精、尖的科学实验一样，有了良好的环境和工具，就为设计高质量的程序提供了方便条件。

综上所述，程序设计不仅要设计算法，确定数据结构，还涉及到程序设计方法和设计工具的选择。因此，有人用以下的公式来描述程序设计：

程序设计= 算法+ 数据结构+ 方法+ 工具

这是对程序设计涵义的深刻揭示。

1.1.2 程序设计的一般步骤

如前所述，程序设计涉及到多方面的内容，它包括从拿到问题直至得到正确结果这样一个过程。这个过程可以分成若干步骤或者叫若干阶段，在程序设计中，设计者应该有计划、有条理地逐步进行。但是，有的初学者往往不重视一些必要的步骤，一拿到问题就想直接写出解题的程序。这是一种很不好的习惯和作风，这样是很难写出高质量的程序来的。就好像一个人写文章，在正式开始下笔之前未推敲提纲，没有对文章的详、略、开头、结尾等进行认真的思考和合理的安排，很难写出好文章一样。对于设计程序，这种“拿来就写”的做法，说明程序员没有掌握程序设计的一般步骤。

正如写作文有分析题意，选材，构思，列出粗提纲，列出细提纲，开始写作文这样一些规律性的步骤一样，设计程序也有一个常规的、通用的步骤。学习程序设计必须首先了解这些步骤，并且在不断的练习和实践中逐渐掌握它们。为了使初学者了解和掌握程序设计的基本规律和步骤，并具有良好的程序设计的作风，为设计大型的、高质量的程序打下基础，下面就一般规模的程序设计步骤作一些介绍。

程序设计的基本步骤如图 1.1 所示。

图 1.1 程序设计基本步骤

1. 明确所要解决的问题

这是拿到问题以后首先要做的工作，其目的是确切地搞清楚问题的性质、任务和要求，分析任务的规模及效益。这一步要确定解决问题的方针，决定解决问题的出发点和目的地。通过对问题的分析，明确所要达到的目的，在分析已知条件(包括问题中所给的条件和现有的计算机系统条件)的基础上，根据问题的性质确定解决问题的方针和对策。比如，根据条件决定要不要把问题分解；分析有哪些因素会影响处理的结果；在时间、空间和精度等因素发生矛盾的时候要决定如何取舍，等等。对任务、要求分析得越充分、越透彻，以后各步的工作就会越顺利。

2．分析问题，建立计算模型

对问题进行具体分析，找出内在的联系与规律，再经过一定的简化和抽象，用数学语言进行描述，建立起正确的计算模型。这是程序设计中十分重要的一步，通过这一步要把实际的物理问题转化为数学问题，要从实际问题中抽象出数据之间的关系。对于数值计算问题要建立起数学模型，对于非数值处理问题则借助于“数据结构”建立起数据模型。从本质上说，数据模型是某种数据关系的模式，因此，建立数据模型也就是建立数学模型。这里所说的计算模型是一个比数学模型更广泛的概念，在软件工程中还要用到其他的模型，但就初学者来说，主要是建立数学模型的问题。

解决科学计算方面的问题，一般不需要程序员自己去建立数学模型，而是用已有的基本模型去完成本问题的映射。设计者的主要任务是把实际问题归结到某种基本模型上，还要善于在不同的情况下利用不同的基本模型。已有的各种公式、定律、定理等都是基本的数学模型。例如，一个电路网络的求解，经过分析，可以抽象为解联立方程组，这就是电路网络问题的数学模型。当然这个数学模型的建立是比较容易的，这是因为在《电路原理》中已经给我们总结了可循的规律。

如果不是纯计算问题，尤其在非数值领域，程序员有时要自己去建立数学模型或数据模型，这就需要设计者付出更多的努力来解决计算模型的问题。为说明这一点，我们举一个实例。

猎人带着狗、兔和白菜要乘小船渡河，但因船太小，一次只能带一样。因为狗能吃兔，兔要吃菜，所以狗和兔，兔和菜都不能在无人监视的情况下留在一起。现要找出既能安全渡河，又使往返监送次数最少的方案。

这个问题要设计程序来解决，关键是要找到将其转换为数学问题的思路，即建立起计算模型。用人工解决这个问题的时候，可以凭着狗、兔、白菜这些概念以及它们之间不相容的情况，找出求解该问题的方案。比如，可以通过图 1.2 所示的①～⑦7 步来完成从本岸

图 1.2　猎人渡河过程

到对岸的运送。图中,“本岸”表示开始渡河以前,猎人、狗、兔和白菜所在的一岸;“对岸”表示猎人带着狗、兔和白菜最终要到达的河岸,即“目的”河岸。

要设计程序(系统不支持汉字的情况下)来解决上述问题,则要以不同的“标志”分别代表狗、兔、白菜,为方便起见,可以取 1、2、3 分别代表狗、兔、白菜。这样取数的目的,是要把狗、兔、菜两两在一起时,相容与不相容的情况区分开。可以看出,当两数之差为 1 时为不相容情况,两数之差为 2 时则是表示相容的情况。显然在程序中是容易判别它们的。这就是实际问题向数学问题的转化,也是建立该问题的计算模型的过程。具体实现时,程序中可以用两个数组分别代表本岸和对岸,用一个变量代表小船,1、2、3 分别代表狗、兔、和白菜。本岸数组中的数 1、2、3 通过小船变量传送给对岸数组,对岸数组中的数也会送回给本岸数组(当到达对岸的两者不相容时,要将其中之一送回本岸)。在每次从数组中取出或向数组存入的时候,要检查数组中的数是否相容,若不相容则试取另一个数,或交换一个数,使留在本岸或对岸数组中的数总是相容或者只有一个。

这个例子中,对问题的抽象和向数学问题的转化都是比较简单的,但通过它可以说明非数值计算问题如何通过设计程序来解决。

3. 确定数据结构,选择或设计算法

这一步要确定计算机解决所给问题的方法与步骤。对于数值计算问题,因为通常情况下数据量不是特别大,所以,主要是解决采用何种计算方法及在计算机上实现的算法的问题。对于非数值处理问题,往往数据量比较大,数据的组织合理与否,对计算机处理的效率影响很大,因此,必须很好地考虑和安排数据的组织形式,即确定数据结构。同时,针对所采用的数据结构采取相应的计算机算法。前已述及,程序设计的根本问题是解决算法与数据结构的问题,可见这一步的重要性。

计算机算法是一个专门的研究课题,人们已总结出许多经验和规律,有许多问题,特别是数值计算问题已有一些现成的算法可以选用。但也有一些问题其算法没有“现成的模式”可以照搬,而需要自己去设计,设计算法有时并非易事。在《计算机算法》课程中讨论了一些常用的算法和一些基本的规律,但要真正掌握它,还要经过大量的实践。

数据结构要研究和讨论的问题也很多,对基本数据结构的掌握和应用也要以大量的实践为基础。

第 3 章中的练习正是为了加强以上两方面的训练。

4. 根据算法画出流程图

这一步是对算法进行具体描述;并用选定的程序设计语言,根据流程图编写程序。画出流程图的过程是把算法具体化的过程,也是使算法向程序靠近的过程。根据逐步细化的方法,画流程图是从粗到细分为不同层次的,当流程图细到能与程序语句对应时,就可以进行程序编写了。当然,对每一步画出的流程图都必须进行正确性检查,避免程序中的逻辑错误。这里需要说明一点:为了方便结构化程序设计(关于结构化程序设计方法,将在第 2 章中专门练习),程序流程图要画成结构化流程图。在开始学习程序设计时就学习画结构化流程图,对学会结构化程序设计是非常重要的。

在画出流程图以后,根据程序流程图编写程序,这个过程称为编码或代码设计。这一过程是将流程图中的各基本结构转换为程序语言的语句描述。在编写程序时,设计者应该正确、合理地使用语言的语句和语法规则,这就要求设计者要熟练掌握所用的语言。为了提高程序设计的质量,增强程序的易读性,还应该使程序设计风格化。关于程序的风格将在第 2 章中讨论和练习。

5. 程序的调试与测试

一个程序编写完以后,往往不能保证完全正确,而需要经过反复的检查和修改,这一步骤叫做程序的调试。

程序的调试是程序设计的一个重要环节,程序调试的工作量和所花费的时间,往往要比编写程序大得多,而且它具有很强的技术性和经验性。因此它也是学习程序设计时需要予以足够重视的基本技能。每个程序的功能和复杂程度不同,其调试的难易也不一样,但就一般情况而言,程序调试总需要一些基本的步骤。人们已总结出一些常用的程序调试方法。

为了验证程序的正确性还要对程序的结构和功能进行测试。程序的测试与程序调试有密切的联系,但又有区别,它也有常用的方法。对于初学者来说,因为程序的规模小,也比较简单,程序的调试与测试是不易区分的,但从学习程序设计的要求出发,还是应该了解并掌握程序测试的基本方法。关于程序调试和测试的问题将在第 4 章中专门讨论。

6. 正式运行程序

在程序调试和测试完成以后,就可以正式运行程序了。对于小程序来说,这一步只要送入正式数据,得到必要的运算结果就达到目的了。但对比较大的程序而言,程序在正式运行后还有维护的问题。在一个大型程序交付运行之后,有可能会发现在设计过程中无法预测的潜在错误,还可能为满足新的要求要对程序进行改进,也还可能需对原来的程序增加新的功能。总之,还有可能要对程序进行修改和扩充,这就是程序的维护工作。

7. 编制程序资料

一个好的程序其使用往往不是一次性的,为了充分发挥程序的作用,比如,为了他人能方便地使用程序,为了所设计的程序能更好地与其他程序“接口”,同时也为了有利于程序的维护,必须重视程序资料的编制工作。许多初学者由于练习的程序比较小,问题比较简单,因而不体会程序资料的重要性,不注意这一步的练习。这是需要纠正的一种“错觉”,必须提醒初学者注意。

编制程序资料包括下列内容:

(1) 程序内容摘要,程序功能概述;

(2) 主要设计思想或算法;

(3) 关于程序逻辑结构的详细叙述;

(4) 主要符号说明;

(5) 输入记录及形式;

(6) 输出报告及格式;

(7) 程序试算时的输入、输出示例;

(8) 程序流程图;

(9) 源程序清单及必要的注释和说明;

(10) 程序使用说明和注意事项。

以上所介绍的 7 个步骤是程序设计的一般过程,但对每个具体问题来说,突出的重点是不一样的。有的问题已有现成的典型算法,不必自己花时间去建立;有的问题层次结构非常明显,程序结构也比较简单;而有的问题则可能建立计算模型比较困难;有的问题可能在程序调试中遇到的问题比较多。这些都需要程序设计者认真、仔细地分析,抓住重点解决问题。

1.2 程序的质量

凡是学过程序设计的人都会深切地体会到,要做到得心应手地进行程序设计,必须遵循程序设计的基本准则,同时还要经过大量的基本训练。这是学习程序设计的基本要求,也是设计一个好的程序的基础。只有在掌握程序设计的基本准则的前提下,通过大量的训练,不断的实践,才能掌握程序设计的一些"要领",才会体会到其中的一些"奥妙"。否则,即便是看起来很简单的问题,也可能设计出"不好"的程序。

比如有这样一个简单的问题:输入 A, B, C 三个数,要求按从大到小的顺序把它们打印出来。

对于这样一个问题,有的初学者编写出以下的 FORTRAN 语言程序:

```
1     READ(*,*) A,B,C
      IF (A.GE.B.AND.B.GE.C) GOTO 20
      IF (A.GE.C.AND.C.GE.B) GOTO 40
      IF (B.GE.A.AND.A.GE.C) GOTO 60
      IF (B.GE.C.AND.C.GE.A) GOTO 80
      IF (C.GE.B.AND.B.GE.A) GOTO 100
      IF (C.GE.A.AND.A.GE.B) GOTO 120
20    WRITE(*,*) A,B,C
      GOTO 1
40    WRITE(*,*) A,C,B
      GOTO 1
60    WRITE(*,*) B,A,C
      GOTO 1
80    WRITE(*,*) B,C,A
      GOTO 1
100   WRITE(*,*) C,B,A
      GOTO 1
120   WRITE(*,*) C,A,B
```

```
      GOTO 1
      END
```

显然,这个程序的运行结果是正确的。但是,大家都会觉得太噜嗦了,效率太低了,以至于编写者自己也感到不满意。

相反,对于上述问题,有的人用以下的 FORTRAN 语言程序来实现:

```
1     READ(*,*) A,B,C
      P= MAX(A,B,C)
      Q= MIN(A,B,C)
      R= A+ B+ C- P- Q
      WRITE(*,*) P,R,Q
      GOTO 1
      END
```

很明显,这个程序不但运行结果是正确的,而且语言表达精练,计算机上执行的效率也高。相比之下,人们都会喜欢后一个程序。这两个程序都是正确的,但质量上却有很大的差别。

通常,编制一个解决一般问题的程序,要做到程序运行正确,即使对一个初步掌握程序设计方法而缺乏训练的人来说,也是可以做到的。但这只是程序设计的最起码的要求,仅能满足这个要求的程序是远远不够的。而要做一个能够设计高质量程序的优秀程序员就不是一件轻而易举的事了。那么,如何评价一个程序的质量呢? 应该说,不同的情况、不同的问题评价一个程序质量的标准是不大一样的,但一般情况下,可以用以下基本准则来评价一个程序的好坏,即: 正确性、可靠性、简明性、有效性、可维护性、适应性。

1.2.1 程序的正确性与可靠性

正确性就是要求程序是正确的。这是对程序设计的基本要求,是判定程序质量的最基本的准则,不能得到正确的运行结果的程序当然是无效的。对程序的逻辑正确性予以证明是比较麻烦的事情,对复杂程序正确性的证明更是困难。因此,通常可以用测试的方法来检验程序的正确性。对于小程序来说,由于逻辑结构比较简单,正确性的验证相对简单。关于程序的测试将在第 4 章中讨论。

可靠性则要求程序在多次反复使用的过程中保证程序的正确性。可以通过反复测试,特别是设置一些临界状态来验证程序运行的情况,以检验其可靠性,这也是程序测试的内容。

1.2.2 程序的简明性

一个好的程序应该简明易读,这就是程序的简明性。

简明性要求程序结构清晰、易读好懂,更不可人为地增加程序的复杂性。程序的简明性要求是为保证程序的易读、易懂而提出的,但从教学角度而言也是培养严谨的科学作风的一个重要环节,它是衡量程序员素质的重要的标志之一。因此,它应该贯穿程序设计学习的全过程,并作为基本训练的主要内容之一。

程序的简明性与所用的程序设计语言的表达能力有关,此外,程序设计的风格是非常重要的因素。只有做到程序设计风格化,才能使设计的程序结构清晰、易读易懂,具有简明性。程序设计风格化是人们在程序设计工作中总结出来的经验。一个无风格的程序,往往难读懂,查错和修改也比较困难,因而对软件的生产和推广使用都是非常不利的。下面举一个小例子说明程序风格对易读性的影响。

例如有这样一个问题:求 $1+\frac{1}{1\times 2}+\frac{1}{2\times 3}+\cdots+\frac{1}{N(N+1)}$ 的和,要求累加至最后一项的值小于 10^{-3} 为止,但最多累加到第 20 项。

对该问题有的人写出如下程序:

```
      Y= 1.0
      N= 0
10    N= N+ 1
      T= 1.0/(N* (N+ 2))
      Y= Y+ T
      IF (N.LT. 19) GOTO 30
      GOTO 40
30    IF (T.GE.1E- 3) GOTO 10
40    WRITE(* ,* )Y,T,N
      END
```

这个程序本身并不长,但用了三个语句标号和三个 GOTO 语句,使人感到思路不清晰。这就好比平时走路,出发点与目的地本来相距很近,但由于绕着圈子走,所以让人感到路途遥远,对出发点和目的地的位置关系发生错觉。如果说因为这个程序很小,这种感觉还不明显的话,那么对于再大一些的程序这一点就会暴露得非常明显。造成这个问题的原因是编程者对程序的结构没有仔细考虑和认真安排,也就是不注意程序设计的风格化。

与此相反,注意程序设计风格化,就可以克服类似上述程序这样的毛病,可以使程序结构清晰,逻辑关系简明,从而使程序具有较好的易读性。这样不仅可以减少程序出错的因素,有利于迅速查出错误,便于算法交流,还可以使程序具有易理解性。易理解性与易读性是密切相关的,一个易读性好的程序,因为语言表达精炼,意思简明扼要,通常会具有较好的易理解性。例如,对上面求累加和的程序,我们可以稍加修改,使程序变得非常清晰、简单:

```
      Y= 1.0
      N= 0
10    N= N+ 1
      T= 1.0/(N* (N+ 2))
      Y= Y+ T
      IF (N.LT. 19.AND.T.GE.1E- 3) GOTO 10
      WRITE(* ,* )Y,T,N
      END
```

很明显,这个程序由于结构清楚,比前一个程序的易读性和易理解性要好得多。

除此以外,风格化的程序设计还可以使程序具有易修改性。这一点也是非常重要的。因为任何软件的研制和开发都有一个逐步完善,不断发展扩充的过程。同时,情况的变化和技术的发展也必然会对软件的功能和性能提出新的要求。这就是说,软件的可修改性和可扩充性标志了软件的生命力,只有可修改性和可扩充性好的软件才有发展前途和应用前途。

程序的风格是程序的效果的反映,是程序质量的重要标志。在学习程序设计的时候必须十分注意这方面的学习和训练。

实现程序设计风格化有很多具体的方法和措施,例如模块化程序设计方法、结构化程序设计方法以及程序资料文件化等等。关于这些方法将在第 2 章中专门讨论和练习。

1.2.3 程序的有效性

节省存储空间,减少运行时间,计算结果达到精度要求,这是程序的有效性要求。

在保证易读性、易理解性、易修改性和易扩展性的前提下,使程序的运行速度尽可能快,占用的存储空间尽量少,是程序设计的要求之一。尽管当今计算机技术的发展使运行速度和存储空间都比计算机发展的初期有了极大的提高,对程序的时间效率与空间效率的要求已经不是第一要求,但程序的时空效率仍然还是程序设计的重要目标。

例如,为了逆置一维数组中的元素,有人写出以下 FORTRAN 子程序:

```
      SUBROUTINE INVARY(A, B, N)
      DIMENSION A(N), B(N)
      DO 100 I= 1, N
        B(N- I+ 1)= A(I)
100   CONTINUE
      DO 200 I= 1, N
        A(I)= B(I)
200   CONTINUE
      END
```

这个程序用了一个中间数组 B,而数组 B 所占的存储空间与被操作的数组 A 是一样大小的,也就是说,为了实现对 A 数组中元素的逆置,需要有两倍于数组 A 的存储空间。

同样是这个问题,有的人写出以下 FORTRAN 子程序:

```
      SUBROUTINE INVARY1 (A,N)
      DIMENSION A(N)
      DO 100 I= 1, INT(N/2)
        B= A(I)
        A(I)= A(N- I+ 1)
        A(N- I+ 1)= B
100   CONTINUE
      END
```

这个程序的运算量与前一个程序是相同量级的,但执行过程中只需要一个中间变量 B 作

为辅助单元,所需要的存储空间基本上就是 A 数组的存储空间。

从这个简单的问题可以看出,在程序设计中程序的空间效率问题必须予以重视,前一个程序由于设计者不注意程序的效率,所占用的存储空间就大,而后一个程序存储空间的利用率就大大提高。

又如,有以下两个程序段,它们实现相同的计算。

程序段 1:

```
        ┆
      DO 200 I= 1,20
        DO 100 J= 1,3
          ISUM= (I+ 1)* * 2* (J+ 2)+ ISUM
100     CONTINUE
200   CONTINUE
        ┆
```

程序段 2:

```
        ┆
      DO 200 J= 1,3
        K= J+ 2
        DO 100 I= 1,20
          ISUM= K* (I+ 1)* * 2+ ISUM
100     CONTINUE
200   CONTINUE
        ┆
```

这两段程序虽然功能相同,但由于循环嵌套设计不同,其执行效率差别很大,程序段 1 执行的语句数要比程序段 2 多出二分之一以上。由此可以看出好的时间和空间效率对程序设计的重要性。

显然,一个程序如果处理问题或计算的精度达不到要求,也不能达到预期的目的,当然也就很难说程序是有效的。因此,在软件、硬件环境允许的前提下,所设计的程序应在处理精度上达到要求,这也是保证程序有效性的一个方面。

1.2.4 程序的可维护性

程序结构要模块化,且各个模块的功能应专一化,这不仅是增强程序可读性的需要,而且是程序可维护性的要求。

一个较大的程序,特别是一个大的软件系统的设计花费了大量的人力物力,付出的代价是昂贵的,因此总希望所开发的软件有很强的生命力,有很长的使用期。但一个软件使用的时间越长,可维护性对维护费用的影响就越大,当然软件使用得越广泛,可维护性要求也越显得重要。

软件的维护包括下述内容。首先是对使用过程中发现的原有程序中的一些小错误进行修正,好的程序和软件不会因修改而导致新错误的产生。其次,一个软件系统使用的时

间越长,使用面越宽,在环境、条件、要求等方面的变化就越大。为了适应这些变化就可能要对原有的软件作修改,一个好的软件,应该是方便这种适应性修改的。再者,由于技术的发展及使用中对系统认识的深化,往往要对原有软件做些完善性工作,而且会不止一次地进行这种完善。因此,作为一个有应用前景的软件,应该给不断的补充和完善打下好的基础,提供良好的条件。

为了使所设计的软件达到上述要求,提出了程序设计模块化和局部化的问题。程序结构模块化就是把整个程序要实现的功能划分为几个子功能,每一个子功能由一个程序模块来完成。局部化则要求各模块功能专一,且具有良好的独立性。模块独立就是把模块内部的信息包封起来,只留出一些必要的与外部通信的"口子"(即模块间的接口)。这样的程序模块就好比是一些不同颜色、不同形状的积木块,合理、巧妙地利用这些积木块可以盖出各种各样的小房子。在程序中则可以根据需要,灵活调用所设计的模块,而程序模块之间只通过接口联系。这样设计出来的程序,层次结构分明,模块之间联系清晰,相互之间的牵制和影响小,从而使程序具有良好的可维护性。有关模块化设计的问题在第 2 章将详细讨论。

1.2.5 程序的适应性

一个软件要能够很好地推广使用,具有生命力,必须要有好的适应性。

如前所述,计算机硬件的迅速发展,技术的不断更新,必然会对软件提出新的要求。要使所开发的软件具有强盛的生命力,就要求在环境、条件有所变化的情况下,软件仅作很小的修改,甚至不作修改仍能够继续使用。同时,由于计算机的种类、型号日益繁多,一个好的软件的推广、移植就有更重要的意义,而在这种情况下软件的推广、移植的工作难度很大,这就对软件的适应性提出了更高的要求。

软件的适应性要求可以从两方面采取措施予以提高。其一是软件开发时应尽量采用标准化、规范化的语言,这样就给软件的推广、移植提供了有利条件。比如高级语言程序就要比汇编语言程序的可移植性好,这是因为汇编语言与计算机本身的特点联系紧密,各计算机系统的汇编语言差异甚大,因而把汇编语言程序从一种计算机系统移植到其它计算机系统上,或推广到其它计算机上使用时就很困难。而高级语言不仅提供了抽象的语义而掩蔽了实现细节,同时高级语言的标准化、规范化相对地说比较好,因此移植和推广高级语言的软件就相对比较容易。又如在支持面向对象的程序设计语言中,对象具有更高层次的抽象,与具体机器的特征离得更远,信息的隐蔽性也更好,因此,软件的移植也就容易,适应性当然也好。其二是尽量提高程序的通用性。一个没有通用性的程序所能发挥的作用是非常有限的,它不能为大家所共享,这显然不能满足适应性要求,与当今普遍要求资源共享的形势也不相适应。也许有人会认为,强调程序的通用性只有在大的程序系统中才有意义,在编制小程序时要求通用性是小题大做,多此一举。这种看法或想法是不对的。因为一种良好的习惯与作风的培养,都是有一个过程的,只有在开始学习程序设计时就努力按要求去做才会达到目的。因此,通用性的要求应贯穿在小程序的练习中。比如,在设计一个子程序时,就应该认真考虑如何使子程序的调用灵活、方便,怎样才能使子程序具有更好的通用性等等。