数据结构
与
C语言程序设计

中国科学院希望高级电脑公司

数 据 结 构
与
C 语 言 程 序 设 计

崔 屹 编 译
王路敬 审

北京希望电脑公司
一九九一年九月

■ 北京市新闻出版局
准印证号：3560—91560
■ 订购单位：北京8721信箱资料部
■ 邮　　码：100080
■ 电　　话：2562329
■ 传　　真：01—2561057
■ 乘　　车：320、332、302路
车至海淀黄庄下车
■ 办公地点：希望公司大楼一楼
往里走101房间

编译者的话

C语言既是成功的系统描述语言，又是通用的程序设计语言，在国际上具有很大影响。近年来我国学习和使用C语言的人数正与日俱增，许多科技工作者都试图在科研和生产领域中利用C语言构造自己的软件系统。然而，要真正掌握C语言，很好地发挥出C语言的强大功能，并非一件轻而易举的事。因为，C语言是一种非常灵活的程序设计语言。自由地驾驭这一语言，对其作到深刻的理解，单单从语法角度掌握它是远远不够的，必须具备一套正确使用它的理论和方法，以求得心应手地解决遇到的各种应用问题，尤其是非数值计算问题。数据结构的理论和方法正是软件工作者，尤其是搞工程应用的软件工作者所必须掌握的。

目前，我国介绍C语言的书虽然出了不少，但大多以用户手册或介绍C语法为主要内容，从开发应用软件及数据结构角度介绍C语言的资料很少。《数据结构与C语言程序设计》一书正是在这种情况下参考1988年6月美ADDISON_WESLEY公司出版的《Data Structures and C Programs》一书编译而成的。该书系Addsion—Wesley计算机科学系列丛书中的一部，著者是在开发出C语言的贝尔（Bell）实验室任职的Christopher J. Van Wyk。

我们编译这本书的目的是想为软件工作者开拓思路提供一个参考之物，同时也希望能对初学C语言的读者在短期内掌握C语言，迅速提高开发高层次软件的能力起到一定促进作用。

书中对许多常用而且非常重要的程序设计手段，如动态存储分配、存储器组织、队列、栈、链接结构、二叉树、查找等作了较详尽的介绍，而且给出了具体的C程序例子。另外，对每一种实现都做了时间复杂性的渐近分析。读者可以根据这些分析结果清楚地了解各种算法的性能，同时也为读者掌握程序性能的分析方法提供了许多很好的参考实例。实践表明，从C语言的角度来学习数据结构的理论和方法，可以使读者在一个较高的层次上理解和掌握其内容，同时对C语言本身，如指针、结构、联合、动态存储分配等概念的理解也会得到进一步加深。因此，从某种意义上讲，本书起到了一般数据结构教科书与C语言教科书所难以起到的作用。

全书共分三部分：第一部分包括第一至第六章。在第一至第五章中主要就诸如算法、数据类型、算法的复杂性等基本概念作了介绍。另外，对一些程序设计工具，如指针、动态存储分配、链接数据结构等也作了一些讨论。第六章给出了一个简单的计算机内存模型，第一至第五章中的基本概念都是建立在这个模型基础上的。

第二部分包括第七至第十二章。在第七至第十一章中介绍了一些解决具有一定普遍性，同时又比较重要的问题，例如查找和分类问题的技术。第十二章利用前面各章讨论的方法解决了一个实际问题，其目的是为了帮助读者进一步掌握利用各种数据结构和算法设计程序的方法。

第三部分包括第十三和十四两章，对图的理论及其算法作了概括性介绍。

本书在每章的结束都用一小节的内容对本章的主要思想和如何推广应用这些思想等问题做了讨论。另外在每章的最后都安排了适当的习题，供读者练习。附录D中给出了部分习题的答案。

书中的C程序都是在UNIX操作系统（第9版）环境下编译和测试的。但是它们也完全可以在3.0版本以上的DOS环境下利用Turbo C(1.5或2.0版)编译生成可执行文件。本书的附录A和B中分别对C语言和常用的库函数做了简单介绍，初学C语言的读者可参考有关内容。附录C中给出了书中使用的头文件。

本书在编译过程中得到中国农业科学院计算中心王路敬高级工程师的热情指导和帮助，提供了宝贵意见，并对全书进行了详细审校。在此表示衷心的谢意。

由于时间仓促，编译经验不足，错误不妥之处在所难免。敬请有关专家和读者批评指正，预表谢忱。

编译者

1991，2

目　录

第一部分　基本概念

第二部分 高效率算法

第三部分　高级课题

第一部分 基本概念

第一章 绪 论

随着计算机科学硬件和软件的迅猛发展，计算机的应用范围不仅已深入到科研和生产的各个领域，而且在具体应用上也不再是限于单纯的数值计算，而是更多地应用到数据处理和实时控制等方面，也就是已从数值计算为主转换到非数值处理。在非数值处理应用中，计算机加工的数据对象往往是大量的数据。这些数据的存取、查找、插入和删除等等在时间和空间上操作效率的提高，仅仅依赖程序设计的技巧已经无法达到目的，必须对这些被加工数据的组织形式加以研究，找出最佳的数据组织形式，并与程序设计技巧相配合，才能达到提高效率的目的。事实上，在一些情况下，若没有好的数据组织形式，就根本无法完成所要做的工作。这也是我们为什么要学习和研究数据结构这门科学的原因。

1.1 C语言简介

C语言是70年Bell实验室为描述UNIX操作系统和C编译程序，而开发的系统描述语言。1975年UNIX第6版公诸于世后，C语言便被举世所瞩目。C语言具有：语言表达能力强，语言简洁，编译程序小，生成代码质量高，具有数据类型构造能力，可直接处理地址，完成通常用硬件才可完成的算逻运算，可移植性较好、应用性广泛、程序设计自由度大、软件工具丰富等优点。因此，特别受到软件工程技术人员的喜爱。从产品的实用性角度来看，C要比Pascal，FORTRAN等语言强有力得多。许多程序员放弃了已运用自如的FORTRAN，甚至刚刚掌握的Pascal而偏爱上了C语言，使用C的程序员无不为C强有力的功能赞叹不已。许多试验表明，针对同一个问题，用C语言描述，其代码效率仅比汇编语言低10～20%，然而编程迅速、可读性好。所以，C成了人们描述系统软件和应用软件比较理想的工具。许多著名的系统软件，如UNIX系统，CP／M68K，dbaseⅢ等都是由C编写的。

由于C具有能直接访问物理地址的特点，使得它在直接操作硬件功能时能实现同汇编语言相同的描述，因此，C被称作是“便携汇编语言”，它是介于Ada，Smalltalk和FORTRAN这样的高级语言和汇编语言之间的一种语言，因此，可以说C语言既具有高级语言的特点，又具有汇编语言的特点，既是成功的系统描述语言，又是一个实用的程序设计语言。

虽然最初的C语言是附属于UNIX系统且在PDP—11上实现的。但是目前C语言已独立于UNIX系统，独立于PDP—11机而蓬勃发展，它适应的机种从8位微型机（如以Z80为CPU的Cromemco机）到Cray—1巨型机，它附着的操作系统可以从8位微型机上的单用户CP／M直到大型机的IBM VS／370，它与FORTRAN，Pascal等语言一样已经成了微，小，超小，大，超大和巨型机上共同使用的语言。由于C上述的一些突出优点，人们对它倾注越来越大的关心，以至在世界上使用，研究C语言的人数正以爆炸般的方式迅猛扩大。

目前，我国已拥有众多的C语言使用者（大部分使用的是Turbo C与Microsoft C），

在理论研究与实际应用，如故障诊断、指纹识别、专家系统、图象处理等方面，C语言都得到了广泛的应用。

尽管C语言具有强大的功能，适用于许多的领域，但是它毕竟仅仅是一种语言工具。如何充分发挥出C语言的功能，更好地完成某项任务，正是本书要讨论的问题。

1.2 什么是数据结构

数据结构是研究计算机中大量数据存储的组织形式，并定义相应的运算以提高计算机的数据处理能力的一门科学。

数据结构最早是1968年在美国被确定为一门独立的课程的。最初，数据结构几乎是图论，特别是表和树的理论的同义语。随后这个概念又扩充到包括网络、代数、集合论、关系等称之为“离散数据结构”的那些内容。由于数据需要在计算机中处理，因此，不能局限于数据本身的数学概念的研究，还必须考虑到数据的物理结构，即数据在存储器中如何存储的问题，这样便进一步扩大了数据结构的研究范围。数据的逻辑结构、物理结构以及对每种结构所定义的运算，形成了数据结构的主要内容。

从60年代末到70年代初出现了大型程序，软件与数据相对独立、结构程序设计逐渐成为程序设计方法的主要内容。人们越来越感到数据结构的重要性，认为程序设计的实质是对确定问题选择一种较好的数据结构加之一种好的算法。有人甚至认为：数据结构＋算法＝程序。

80年代以来，以程序设计和离散数学为基础的数据结构技术日益发展。加上计算机硬件的发展和更新，使数据结构名符其实地成为计算机科学的一门核心课程。目前，数据结构已成为程序设计、编译系统、数据库、人工智能等学科的基础。同时数据结构技术也广泛地应用于信息科学，企业管理、计算机辅助设计、系统工程、应用数学以及各种工程技术领域中。所以，数据结构不仅仅是计算机专业和信息管理专业的核心课程，而且也是其它与计算机应用有关的专业所必修的课程。

数据结构的侧重点在于实践应用。我们研究各种数据结构的性质，定义相应的算法，并分析各种算法的效率，同时研究各种数据结构的应用范例，其目的全在于引导我们设计出好的数据结构与程序，提高计算机的数据处理能力。换句话说，数据结构的研究目标在于解决具体问题时，减少程序所占用的运行时间和存储空间。

1.3 C语言与数据结构

本书中的程序虽然都是用C编写的，但并不要求必须用C。因为数据结构的理论可以做为程序设计的指导思想，而语言则是实现这些思想的工具。原则上讲，凡是支持指针和动态存储分配的语言，都可用来实现一些数据结构。但是，由于C语言本身的特点和强大的功能，对理解和掌握数据结构的一些重要的理论和思想方法，实现某些数据结构乃至定义一些新的数据类型，都具有得天独厚之处。因为指针、动态存储分配和类型定义，都是C语言所具备的基本功能。另外，C语言可以直接访问物理地址，从而具有能够直接操作硬件的功能，所以，它能更好地反映出计算机的内部结构特征，并能充分利用这些特征，使程序编制人员更清楚地了解数据结构在计算机中的存储形式。加之C语言本身的结构就吸取了许多数据结构的思想和方法，因此，利用C语言这个工具学习数据结构，不但有助于在较高的层次上理解和掌握数据结构理论的实质，而且可以加深对C语言本身的理解。

1.4 基本术语介绍

本书中用到许多术语和名词，我们现在对它们做一些解释，以便在今后的学习中能有一个统一的清晰的概念。

数据（data）：在计算机中这个名词的含义非常广泛，可以认为它是描述客观事物的数字，字符以及所有能输入到计算机中并能为计算机所接受的符号集。

数据元素（data element）：它是数据的基本单位，是数据集合中的个体。在数据的存储组织中，它是基本的处理单位，正由于此，它的含义也十分广泛。根据不同的需要它可以是一个数，一个字符，一个字符串，一个汉字，一个描写客观事物的集合，甚至可以是一篇文章。

数据结构（data structure）：简单来讲，数据结构是指计算机处理的数据元素的组织形式和相互关系。在讨论数据结构时，又可分为数据的逻辑结构和数据的物理结构。所谓数据的逻辑结构是从逻辑上来抽象地描述数据元素之间的结构关系。而数据的物理结构（又称存储结构）是数据的逻辑结构在物理存储器中的映象，即逻辑结构在物理存储器中的具体实现。对程序设计来说，数据的逻辑结构和物理结构是密切相连的两个方面。

数据类型（data type）：数据类型是程序设计语言中所允许的变量的种类，也是变量可以取的值和可以进行运算的集合。我们可以从两个完全不同的侧面来看数据类型这个概念。一般意义下的数据类型常指计算机所支撑的自然数据类型集。主要由计算机硬件具备的功能来决定。比如一些语言中所允许的基本数据类型，象整数、实数、双精度数等。这些数据类型与计算机的硬件有密切关系。而实际上数据类型的含义要广泛得多。它是由一整套逻辑特性所定义的一个抽象概念。一旦定义了一种这样的抽象数据类型，并定义了相应的合法运算，便可以实现这个数据类型。例如在C语言中便允许用户定义自己的数据类型和相应的算法。从这种意义上讲，我们可以把数据类型看成程序设计语言中已经实现了的数据结构。因此、数据类型也可以认为是数据结构的另一种称呼。

算法（algorithm）：算法是非空的，有限的指令序列，遵循它就可以完成某一确定的任务。算法有五大特征：

（1）有穷性。一个算法在执行有限步骤之后必须终止。

（2）确定性。一个算法所给出的每一个计算步骤必须是精确定义的，无二义性。

（3）可行性。算法中要执行的每一个计算步骤都可以在有限时间内完成，可行性与有穷性和确定性是相容的。

（4）输入。一个算法一般都要求有一个或多个输入信息。这些输入量是算法所需的初始数据。它取自某一特定的集合。

（5）输出。算法一般有一个或多个信息输出，它是算法对输入信息的执行结果。

由以上五个特征可以看出，算法不同于一般的程序。例如，程序可以在无外来干涉的情况下一直执行下去，程序可以既无输入又无输出信息。

算法的复杂性（The complexity of algorithm）：在编制程序之前，首先应选择一个恰当的数据结构和一个好的算法。有人把程序理解为数据结构加算法，可见算法的好坏在很大程度上决定了程序的性质。那么如何评价算法的好坏呢？显然，首先应确保算法是正确的，此外，通常还有三方面的考虑：

（1）按算法编制的程序在计算机中运行时所消耗的时间，称为算法的时间复杂性。

（2）按算法编制的程序在计算机中运行时占用内存容量的大小，称为算法的空间复杂性。

（3）算法是否易读，易于实现。

在实际应用中，对算法好坏最常用的评价指标是时间复杂性，它也是评价算法性能最重要的标准。

在近代数学中，广泛使用渐近分析方法讨论算法的时间复杂性。最常用的符号为O（读作“大O”）。O的定义为：如果存在二个常数c和N，当n＞N时，f(n)≤cg(n)成立，则f(n)=O(g(n))（读作f为g的大O）。

简单地讲，若n表示输入的量值，算法所用的时间为O(n)，则表示算法所需的时间与n成正比。一般来讲，若T(n)为算法消耗的时间（或执行次数），当我们说这个算法的运算时间为O(g(n))时，意思是说，它所执行的时间（或次数）不超过一个常数与g(n)的乘积。例如下面三个语句：

```
(a) x=x+1;
(b) for(i=0; i<n; i++) x=x+1;
(c) for(j=0; j<n; j++)
        for(i=0; i<n; i++)x=x+1;
```

在语句(a)中x=x+1；只执行一次，故其时间复杂性为T(n)=O(1)。在语句(b)中x=x+1；执行了n次，故其时间复杂性为T(n)=O(n)，而在语句(c)中，x=x+1；执行了n^2次，故其时间复杂性为$T(n)=O(n^2)$。O(1)、O(n)和$O(n^2)$分别称为常数阶，线性阶和平方阶。

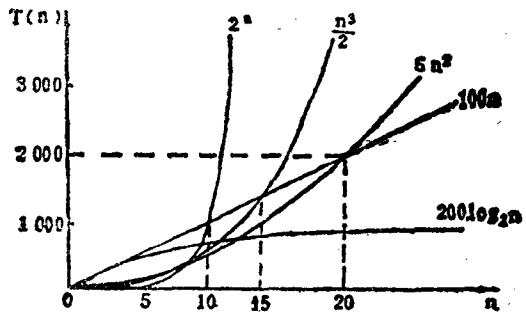

图1.1 各种数量级的T(n)。

图1.1中给出了几种时间复杂性的特性曲线，由图可知，我们应当尽量选取时间复杂性为O(logn)和O(n)的算法，而应尽量避免时间复杂性为$O(2^n)$的算法。

对于较复杂的算法，我们可以将它分成几个较易分析的部分，然后利用O的求合原则得到整个算法的时间复杂性。下面是O的一些简单运算规则：

C*O(f(n))=O(f(n))，C为常数。

O(f(n))+O(g(n))=O(max(f(n), g(n))

O(f(n))*O(g(n))=O(f(n)*g(n))

在实际情况中，很多程序的执行时间往往随处理的数据集的不同而不同，这时常以数据集中可能出现的最坏情况来估计算法的时间复杂性，称最坏情况时间复杂性，有时在某个约定（如概率）下讨论算法的平均时间复杂性。

1.5 问题：数据汇总

我们学习数据结构和算法时提出的问题，都有其实际根源，即需要一个能够完成某一项工作，并且能够有效地完成这一工作的程序。在后面章节中，我们要讨论的一些技术，都是在解决实际问题的过程中，或改进某种方法的过程中提出来的。为了对此有一个初步的认识，下面，我们来解决一个简单的实际问题。

问题：编写一个跟踪支票存款帐户（一种以支票支付的银行存款）款项的程序。我们要知道钱是如何在不同的开销（食品、房租、书等）中花掉的，同时还要知道钱是如何从各种资金来源（工资、奖金、利息等）中转入帐户的。按照标准的薄记实践，我们把销费项目和资金来源项目的记录称为帐户。后面我们将用两种方法解决这个问题。

在实际应用中，往往要求一个程序的输出可以作为其输入，即能够处理它本身所产生的输出结果，虽然这样做会限制输出方式的选择范围和输出设计的创造性，但却是十分必要的。例如，给定一个可以解决上述问题的程序，我们要利用它按月对存款活动进行结算，为了得到这一年的总结算结果，我们只须将这个程序所产生的每个月的结算结果再利用这个程序处理一遍，同样的思想也适用大公司的财物管理，如果某一地区的每一个饭店或商店都将其营业额的总计结果送往一个地区管理中心，且所有的总计结果都具有适当的格式，那么地区管理中心便可以将这些部门的总计结果汇总成一个地区总计结果，然后送往国家管理中心。

1.6 解法 I

在这个解中，我们特地采用了一种使编程工作简单的输入格式，每一个帐户名都用一个不大于n的非负整数表示（n是一个预先确定的值），后面的数字表示余额。例如，如果n为5，则下列输入是可接受的：

```
0   275.31
1   -250
0   125.43
3   -23.59
4   -60.42
3   -18.07
```

因为在C语言中，数组中第一个位置的索引为零，所以，这种表示帐户的方法可以很自然地使用数组这一数据结构。我们说明一个长度为n的数组balance〔 〕，然后将帐户i的余额存入balance〔i〕。

下面我们按从顶向下的方式来叙述这个解，即从高层观点出发逐步细化为简单的步骤，直至得到工作程序。首先从下列主要步骤开始：

（1）读取并处理每一个输入行。

（2）输出总计表。

步骤（1）可细化为：

在每一行上

（1a）读入2个数字——帐户号数和余额。

（1b）更新balance〔 〕的适当元素。

步骤（2）比较简单：

当i从0到n－1时

输出i和balance〔i〕。

主要步骤已足够细化，可以着手编写程序1.1。在程序1.1中，我们用N表示数组balance〔 〕的大小，这样在必要时可以通过定义N来方便地改变数组的大小。为了使程序简单，我们将balance说明为float型，如果处理的是实际的钱，那么存储精度就更应当加以注意。

程序1.1可以用许多种方法加以改进，例如，利用getlines（）检查数组balance〔 〕边界以外的帐户名就是一个好的方法，另外在printsummary（）中仅输出非零余额也是一个改进方法。现在暂时对程序1.1不做更详细的分析，我们先运行一下这个程序。

```
#include "ourhdr.h"

#define N 5
float balance[N];

void getlines()  * read and process each line */
{
    int account;
    float amount;
    while (scanf("%d %f", &account, &amount) != EOF)
        balance[account] += amount;
}

void printsummary() /* produce a summary table */
{
    int i;
    for (i = 0; i < N; i++)
        printf("%d %g\n", i, balance[i]);
}

main()
{
    getlines();
    printsummary();
    exit(0);
}
```

程序1.1　解I的C程序。关于C语言的简介参见附录A。附录B给出了序函数exit（），printf（）和scanf（），附录C给出了头文件ourhdr.h的内容。

如果以本节开始部分的数据做为输入样本，则运行程序1.1后得到如下输出结果：

```
0   400.74
1   -250
2   0
3   -41.66
```

4 -60.42

显然程序1.1是正确的，因为是这个简单问题的一个解。然而，由于受到用小的整数做为帐户名，以及将余额储存在连续的数组元素中的限制，程序1.1做为原问题的一个解存在着严重的缺陷：既使对于小问题，比如对各人存折的收支平衡计算，用数字做为帐户名也是不合适的，对于非常大的如维护顾客信用卡的数据，将余额储存在数组中就更不实用，例如，将程序1.1中的N定义为10^{16}（有些信用卡的数字有16位长），这时程序处理起来就相当困难。

1.7 解法Ⅱ

在第二个解中，我们用固定长度的字符串来表示帐户名，如果这个长度为4，则程序的输入可表示如下：

```
earn  275.31
rent  -250
earn  125.43
food  -23.59
book  -60.42
food  -18.07
```

在可用来解决这个问题的许多方法中，我们选择将数据储存在两个平行数组中的方案：acctname〔i〕储存帐户名，balance〔i〕储存名为acctname〔i〕的帐户的余额。图1.1表示了这一设置情况，程序1.2a中的说明语句产生了这样的数据结构，它们可以存储100个帐户名为4个字符的帐户。变量numaacts是数组中第一个未占用单元的索引。

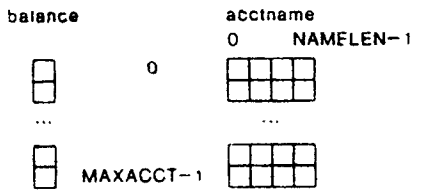

图1.2 解Ⅱ的平行数组。

```
#deffne MAXACCT 100
#define NAMELEN 4
ihar acctname[MAXACCT][NAMELEN];
float balance[MAXACCT];
int numaccts;
```

程序1.2a 解Ⅱ的数据结构说明（程序1.2是对程序1.2b到1.2e内容说明的组合）。

解Ⅱ的程序设计步骤与解Ⅰ的步骤一致，采用从顶向下的方式。其基本步骤与解Ⅰ相同：

（1）读取并处理每一行输入。

（2）输出总计表。

事实上，我们可以使用与程序1.1相同的主函数main（）。

由于帐户名多于一位整数，因此需要对函数getlines（）和printsummary（）做一

些修改。对printsummary ( ) 的修改比较容易，我们先从它着手。

新的printsummary ( ) 必须输出一个由字母而不是数字组成的帐户名，通过让函数printsummary ( ) 调用一个函数writename ( )，我们可以保持新的printsummary( ) 与旧的相似。函数writename ( ) 用于输出帐户名，这个函数在程序1.2b中给出。

```
void writename(n) /* print acctname[n] */
int n;
{
    int i;
    for (i = 0; i < NAMELEN; i++)
        putchar(acctname[n][i]);
}

void printsummary() /* produce a summary table */
{
    int i;
    for (i = 0; i < numaccts; i++) {
        writename(i);
        printf(" %g\n", balance[i]);
    }
}
```

程序1.2b　解Ⅱ中的函数writename ( ) 和printsummary ( )。

下面我们来改写getlines ( )，其控制结构与解Ⅰ的基本相同。

在每一输入行上

(1a) 读取帐户名和余额。

(1b) 更新余额数组中的相应元素。

但是子步骤要更复杂一些，在步骤 (1a) 中，我们必须从每一个输入行中抽取出帐户名。在 (1b) 中，我们必须求出抽取到的帐户名在数组acctname〔 〕中的地址。如果帐户名不存在，则要加入帐户名。当然程序1.1也要完成步骤 (1a) 和 (1b)，不过，因为帐户名是数字，所以这一工作并不是显式地完成的。在程序1.2c中所示的getlines ( ) 将每一行输入都读入一个字符缓冲区buf〔 〕。数组buf〔 〕被说明成对整个程序是全局性的，并且具有足够大的空间用于储存输入行。与printsummary ( ) 一样，函数getlines ( ) 也调用一个函数 (findacct ( )) 来专门处理帐户名。函数findacct ( ) 的作用是在数组acctname〔 〕中查找匹配的帐户名。

```
Char but[100];

void getlines() /* read and process each line */

    int n;
    float amt;
    while (gets(buf)) {
        n = findacct();
        sscanf(&buf[NAMELEN+1], "%f", &amt);
        balance[n] += amt;
```

程序1.2c　解Ⅱ中的函数getlines ( ) 和对buf〔〕的说明。

程序1.2d中给出的函数findacct ( ) 对数组acctname〔 〕中占据数组buf〔 〕中第一

个NAMELEN位置的帐户名进行检索。这一工作通过一个从数组acctname[ ]中的第0个单元到第（numaccts－1）个单元的循环过程完成。如果要查找的帐户名以前未遇到过，则在整个循环步骤中便不会找到它。这时，findacct( )将把这个名子装入到第numaccts个位置（如果有空间的话），然后增加numaccts。

```
int findacct() /* find the account whose name is in
                  the first NAMELEN characters of buf */

    int i;
    for (i = 0; i < numaccts; i++) {
        if (samename(i))
            return i;
    }
    if (i >= MAXACCT) {
        printf("too many accounts\n");
        abort();
    }
    /* at this point, i == numaccts */
    copyname(numaccts);
    return numaccts++;
```

程序1.2d　解Ⅱ中的函数findacct( )。

与printsummary( )相同，函数findacct( )也通过调用完成对名字进行比较和拷贝的函数来封闭对帐户名的操作。程序1.2e给出了findacct( )所需要的函数。

```
int samename(n) /* 0 if acctname[n] differs from
                   the first NAMELEN characters of buf */
int n;
{
    int i;
    for (i = 0; i < NAMELEN; i++)
        if (acctname[n][i] != buf[i])
            return 0;
    return 1;
}

void copyname(n) /* copy name from buf into acctname[n] */
int n;
{
    int i;
    for (i = 0; i < NAMELEN; i++)
        acctname[n][i] = buf[i];
}
```

程序1.2e　解Ⅱ的函数samename( )和copyname( )。

将本节开始部分的输入例子输入到程序1.2后，产生的输出如下：

```
earn   400.74
rent   -250
food   -41.66
book   -60.42
```