微计算机丛书

IBM PC UCSD Pascal 程序设计

〔美〕S·V·波拉克 著
高勇秀 译 刘春年 校

电子工业出版社

IBM PC UCSD Pascal 程序设计

〔美〕S·V·波拉克 著
高勇秀 译
刘春年 校

電子工業出版社

内 容 简 介

IBM PC有多种操作系统，本书是讨论在UCSD P—系统支持下的Pascal语言程序设计的基本方法。本书反映了结构程序设计是一种合理的程序开发途径，并把 Pascal 语言作为支持这种结构的最理想的工具。

本书适用IBM PC的用户、大专院校师生、及有关培训班学员。

Programming the IBM Personal Computer: UCSD Pascal
Seymour V. Pollack
CBS College Publishing
1983

IBM PC UCSD Pascal程序设计
[美] S·V·波拉克 著
高勇秀 译
刘春年 校
责任编辑 王惠民

电子工业出版社出版（北京万寿路）
新华书店北京发行所发行 各地新华书店经售
北京通县曙光印刷厂印刷

开本：787×1092 1/16 印张：14.75 字数357千字
1987年8月第1版 1987年8月第1次印刷
印数：8,000册 定价3.20元
统一书号：15290·550
ISBN 7-5053-0054—7/TN33

译 者 的 话

这是一本关于 IBM PC (个人计算机) UCSD Pascal 程序设计的书。

大家知道，学习一种计算机语言的根本目的是学会程序设计，而 Pascal 正是作为新一代程序设计语言提出的。本书反映了结构程序设计是一种合理的程序开发途径，并把Pascal语言作为支持这种结构的最理想的工具。

IBM PC 有多种操作系统，本书讨论的是在p—系统环境下进行操作的。学习和掌握了本系统的特性和操作之后，就能够容易地对付各种各样的操作环境。

为提高和培养利用计算机解决问题的能力，进一步推广微型计算机的应用，本人翻译了这本书。

北京工业大学计算机系刘春年博士对译文作了校订，并提供了宝贵意见，在此谨表谢意。

由于译者水平有限，不妥之处，谨请读者批评指正。

译者　　1985·8

目　录

前　言

就某种意义来讲，Pascal 在 IBM PC 上的实现可以看成是正在进行的两项革新的组合。第一项革新是 Pascal，这是第一个基于把程序设计理解为规则化过程的编程语言。在这之前的一些语言，虽然也包含了某些为结构程序设计提供的特性，但这些特征能否实现，与其说是设计的问题不如说是环境的问题。这些语言大多数在结构程序设计概念出现之前就已经形成，所以，只能在保持原来语法规则的基础上，对这些语言进行扩充。

相反，Pascal 语言没有这些传统的限制，它明确地反映出结构程序设计的原则。实际上，Pascal 的一个主要设计目标就是促进使用那些与编制清楚可靠程序有关的结构和技法。即采用结构程序设计语言的特性，并且设置一些要求，使得 Pascal 不能不用结构程序设计。

从这方面看，把 Pascal 作为新一代程序设计语言的起点是有益的。本书反映了这种观点，认为结构程序设计是一种合理的程序开发方法，并且把 Pascal 提出来作为支持这一方法的一种较为理想的工具。在此训练阶段，无需维护结构程序设计而反对先前某些特定方法。

第二项革新更引人注意。从七十年代中期起，个人计算机系统就引人注目地进入了市场，IBM PC的出现使个人计算机本来已经迅速增长的情况戏剧性地变为爆炸性的增长。这一市场变化的原因是多种多样的，但主要是 IBM公司在计算机行业中不可否认的影响所起的作用。对于许多个人和团体来说，这个声誉卓著的公司重视而且积极进入个人计算机领域这件事本身，就被看作是对个人计算机所做的保证和对它的重要性的承认。因而，这种功能很强的多用途计算机系统，在一个高速发展的运动中起了重要的作用，这个运动的最终目标是要使计算机的使用和程序设计就象读书和写字一样成为人人都会的技能。

IBM PC 具有操作很方便的特性，若与 Pascal 设计所固有的便于训练的优点相结合，就会使它成为培养良好的程序设计习惯和相对容易地获得必要的程序设计技巧的有力工具。

P-系统是一个对学习和使用 Pascal 特别有效的操作环境，它的引入更加强了这一吸引人的结合。在这一环境中，可以编写，修改，存储，取用和执行程序，而不必在用户和系统之间进行那些麻烦复杂的交互操作。因此，当学生学习用明确而有序的方法编写程序时，还能得到一些附加的收益，即学到一个交互式操作系统的性质和有效地使用。尽管这些附加的操作技术几乎是偶然地获得的，但却是很有用的基础知识，它使学生比较容易地对付他们在将来可能碰到的各种各样的操作环境。

把以上因素融合起来，就形成了 IBM PC 的 UCSD Pascal。本书包含了有关 P-系统的足够信息，不需要附加其它的文件，就可有效地应用。尽管如此，我们还是期望学生在他们工作之前或工作初期，能较好地掌握屏幕编辑程序的知识，这方面的一些建议将在第二章最后给出。

本书是为那些从未学过 Pascal 和 IBM PC 的学生准备的。书中列举了广泛的实例，使之既适宜用作教科书也可用作自学教材。如果学生已经用过其它的程序设计语言(任何系统)，就可以跳过第一章。对于那些熟悉微计算机基本系统的学生，只须把第二章的内容浏览一下

就可转到P-系统或 IBM PC 部分。

本书不打算讲述全部的 UCSD Pascal。为了集中讨论主要的程序设计原理和实践，略去了一些太专门化的特性。掌握了本书提供的基础知识之后，程序员只要了解附加特性的运算性质就不难自己去应用它们。有关方面的资料可参阅 IBM PC 手册UCSD Pascal/A Language Definition。

本书中许多以打印方式出现的程序例，也可象磁盘上的P-系统文本文件一样使用。这就为学生提供了编译这些程序并在任何所希望的操作条件下执行它们的机会。即使不考虑规模和花销的限制，从课文本身是不能深入了解程序行为及其设计优缺点的。增加的这个程序盘，还为学生（或教师）提供了进一步的教学上的便利，它可方便地修改程序并观察修改后对程序行为的影响。由于很多程序都设计为交互式的，我们可立即看到这种修改的结果，从而使所希望学到的概念得到进一步的加强。除了这些程序之外，这个附加磁盘还包含了正文中一些问题的测试数据。这样，学生就不必每人都去准备自己的数据，从而可把精力集中到问题本身。此外，通常在解决数据量很大的问题时会发生的障碍也不存在了。

我要感谢CBS学院从事出版工作的布伦特·哈里森先生和保罗·贝克先生、华盛顿大学计算机设备部的汤姆·巴哥尼吉先生及 Cobb/Dunlop 版机构有价值的建议和帮助。特别要感谢华盛顿大学计算机科学系主任杰罗姆 R. Cox, Jr. 先生在设置新课程时的创造性及甘愿从事这种工作的精神，最后还要感谢赛得·玛克和塞维·赖克先生等。

S·V·波拉克

第一章　绪　论

本书将讨论 Pascal 程序设计语言的特性和在 IBM　PC 上用 Pascal 解决问题的技术。与其它大多数语言不同，Pascal 这个名字不是缩写，而是以法国十七世纪伟大的数学家 Blaise Pascal 命名的。通常人们认为是他成功地创造了第一个机械加法器。尽管没有直接的历史联系，他研制的装置(图 1.1)被认为是作为今天的数字电子计算机前身的算术机器的长期发展的开端。

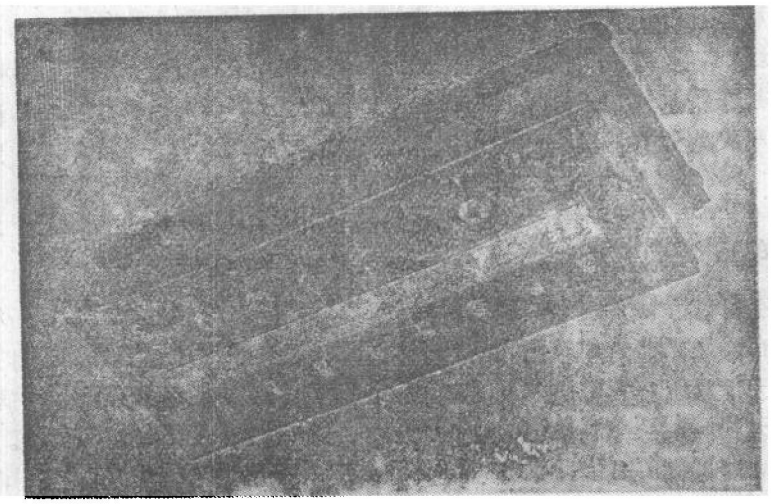

图1.1　机械计算器

Pascal 程序设计语言是 Niklaus Wirth 教授在 1971 年提出来的。尽管当时已有数百种程序语言，但 Pascal 一开始就表现出它不是一般的语言，因为它明显地支持把程序设计作为系统的、有规则的活动的观点。早期的主要程序设计语言代表了50年代初对程序设计的看法和态度。因此这些语言重点是适于某些特殊种类计算的描述。除少数例外情况，这些语言只是偶然注意到计算的组织和它们运算的数据。

这些年来，程序设计过程已成为广泛的研究课题。其结果是人们认识到了把程序作为生产产品的重要性。虽然这种产品不是有形的东西（我们不把程序设想为能用桶装起来的东西），但它却享有一个物质产品的许多属性：它由若干针对特定目的而设计的部件（称为模块）所组成。而且，这些模块之间又必须按预定的方式工作，才能使总装起来的程序正确有效地发挥功能。程序设计过程同样适用在设计开发任何其它产品时所采取的规则的而又整体化的方法。目前仍在发展着的结构式设计和结构式编程就都起源于这种认识。

由上所述，Pascal 语言之所以重要，是由于它是把程序的设计和组织作为中心问题的第一个主要的程序设计语言。为它所设置的一些主要特性，都是为了便于描述其良好的结构。借助于一些对语句排列的要求，Pascal 进一步使用户较容易地采用良好的结构。结果我们得

到了一个强有力的工具，掌握它大大有助于应用计算机去有效地解决问题。

第一节　系统地解决问题

计算机不是万能的。严格地讲，计算机什么也不"知道"。它只是人们设计的一种能按照规定的方式传送信息，并能完成信息指定的基本操作的一种电路。而我们所要做的是把这些操作进行从头至尾的组合，以得到有用的结果。因此，当我们用计算机解决问题时，要我们自己确定如何去解决问题，计算机只是按着我们的命令去执行。所以程序员在使用计算机前，就要知道如何解决问题。解决问题的方法称为算法，是预先做出的，并把它表示成一系列步骤，每进一步就更加接近最后的结果。我们还要把算法写成程序送入计算机。上述的一系列步骤可用程序设计语言这个精确的词汇来描述。

在写程序之前，首先要确定算法，清楚地描述这个算法，使得它能简单对应地转换成正确的程序。这是在解决问题一开始就要做的事，而不是到写程序时才做。对于问题求解虽无严格的、万无一失的处方，但是我们可以给出一些思考方法，它们将有助于顺利地解决问题。

一、问题的识别

倘若问题本身就弄错了，人们将白费时间、财力和精力去寻找全面的精巧的解。常识告诉我们，只要在解题之前先把问题弄清楚就可避免这种浪费，但类似情况仍时有发生。最近，在某城市开设一家医院，它的病人与其他医院接收的进行相同手术或治疗的病人相比，出院时间要晚一两天。问题被想当然地认为是大夫在手术后将病人留院时间太长。于是该医院当病人康复后就立即催其出院。结果毫无成效。最后，真正的问题所在被发现了：原来病人在结账之前不能出院，而该医院的会计部门比别的医院要多花一、两天准备账单。

鉴于此类情况，很多单位作为制度规定，在着手解决一个问题之前，必须对问题本身作出精确的、书面的陈述。

二、找出合适的解法

当已经确切地知道问题是什么，下一步就是寻找一个有效的方法去解决它。这一点说起来容易做起来难。因为不存在一个由问题到解的普遍适用的一步一步的过程。事实上，仅仅知道了问题是什么并不能总是保证它有解。由于这些未知因素，所以这一步通常是整个过程中最困难的一步。大量创造性的工作和技巧都集中在这一步。

问题的解取决于算法的形式。我们可以进一步定义算法为满足下列要求的一系列步骤：

1　一个算法必须是有限的，即它能够最终停下来。例如，要调配一种特殊色调的蓝色颜料，我们建议如下的解决方法：

· 取一罐底色颜料和一罐蓝色颜料。

· 打开这两罐颜料。

· 把蓝色颜料加到底色颜料中，直到色调适宜为止。

这个过程看起来似乎不错，但可能是无限的。例如，底色本身可能就太蓝了，不符合规

定的色调要求。可是过程指示是往底色里再加蓝色，其结果是蓝色越来越深，结果无解。因为计算机本身不能执行判断，要靠我们提供一个专门的机构，以保证在任何条件下运行的算法都可停下来。对上述过程可做如下修改：

· 取底色，白色和蓝色的颜料各一罐。

· 打开三罐颜料。

· IF （如）底色太蓝

THEN （则）

加白色颜料，每次加一点，直到颜色合适为止。

ELSE （否则）

加蓝色颜料，每次加一点，直到颜色合适为止。

我们可以想象在某种情况下这个修改过程仍是无限的，但这里不作进一步讨论了。要点在于必须有办法使过程成功地终止。

2．算法要精确，即每一步都要精确描述，使得任何执行者都能够完成这个步骤。例如，上例就不够精确，我们本应该指明加蓝色或白色颜料的数量，（"加一点"是多少？）也必须说明合适的颜色是什么样子，对于那些色感不灵敏的人来说，给他的指令应该严格到用数量表示，如"正确的色调是在色度计上介于 407.5 和 423.3 之间的颜色"。如把这种算法转换成计算机程序，精确性的要求意味着每一步都必须按照可在计算机上运行的要求进行描述。

3．算法应是通用的，即过程不应限于只能解决某一具体问题的某一具体情况。相反，它应对各种情况都能给出满意的解决。例如，上述调制颜料混合物的过程可解决调制所要求的蓝色色调的问题。我们可以把它扩充为调制别种蓝色，甚至处理其它颜色的各种色调。

在很多实例中，可以给出解决一个问题的不同方法。例如，假定在长途旅行中，汽车突然颠簸，方向盘也不那么好使了，而路面并无异常，停车检查，发现是一个轮胎漏气。在讨论可能的解之前，我们必须弄清问题是什么。这个例子并不难识别，那就是我们的旅程被中断了要尽快恢复。问题提出之后，我们可以考虑种种可能的解决办法。例如：

（1）抛弃汽车，继续步行。

（2）另买一辆汽车，继续前进。

（3）给轮胎打气后继续前进。

（4）就近找到服务站，请修理工换掉轮胎。

（5）向过路汽车打招呼，请求把我们带到目的地。

（6）向过路汽车打招呼，请求支援轮胎。

（7）给家里打电话，来人换上轮胎。

（8）自己换轮胎，并继续前进。

首先我们把明显不适宜的解抛弃掉，从而缩小选择的范围。解(2)和解(3)就属于这一类。解(5)和解(6)大概也不难排除掉。其后，要找最合适的解就有点难了，因为这与环境有关。例如，若我们离目的地已不远并且时间很紧迫，解(1)可能就是最好的。

三、问题的分解

当我们遇到一个问题后，往往不可能一下子全部解决它。问题多半是很复杂的，往往难

以同时掌握它的全部细节。这时通常把它分解成许多相关的子问题的集合。每个子问题都小到可以作为一个完整的容易处理的实体。这一分解过程使得对每一个小问题都可以很容易地得到解。结果我们就完整而准确地知道了每一小块要做什么以及它怎样与相关的小块连接。然后可以着手分别为每一个部分求解。

为了说明这个问题，再回到那个开不动的汽车的例子。假如我们有一个备用的轮胎，此时最好的解决办法就是换掉漏气的轮胎。如果我们更仔细地研究这个解，就可以分出几个步骤，每一步体现了一个独立的小问题：

(1) 从车尾的行李箱里取出备用轮胎及必要的工具。

(2) 取下漏气的轮胎。

(3) 安装备用的轮胎。

(4) 把在第(2)步取下的轮胎以及工具装回车尾行李箱内。

随着观察的愈来愈细，每一步所需的行动也就越来越清楚。例如，我们对第(2)步再加细察就会发现取下漏气轮胎的工作又必须有一些准备工作。

如(2)取下漏气的轮胎：

- 取下漏气轮胎的毂盖及其固定螺帽。
- 把千斤顶放在合适的位置。
- 安装千斤顶。
- 用千斤顶顶起轮子，使之离开地面。
- 移走轮子。

如果必要还应把这些子问题依次再分解，直到把每步操作化为一系列明确的简单步骤为止。

用计算机解决问题，也有同样的过程。其最终结果是一个程序，但目前我们不知道该程序的任何细节，而是从分解过程中了解如何将该程序分成程序片断以及每个片断要做些什么。

四、各部分细节的标识

在这个阶段我们实际上还不知道在分解问题的过程中标识出的各个片断如何完成它们的任务。事实上，我们也可能根本不知道是否能解决各个片断的问题。而分解过程的根本目的就在于更容易地逐个解决这些子问题。一旦划定了这些片断，我们就能确定每一个问题性质。结果是每一片断将被完整地定义：它要执行什么任务，为执行这些任务需要些什么，及究竟如何执行这些任务。

下面这个简单的例子可用以说明分解的思想：假定我们的问题是要生产唱机。对该问题分解的结果表明需要一些部件，其中之一是转盘（我们经研究认为旋转唱片加上静态音臂的设计比让唱针在静止的唱片的音槽里转圈好）。唱盘上应能放得下一定大小的唱片，并有一定的转速。此时，我们可以处理如下这些问题：唱盘的厚度、加工的材料、整个表面的重量分布及其它一些特性。这些特性的确定就为唱盘生产提供了足够的信息。对于其它部件，也有这样的过程。在用计算机解决问题时，这一阶段就为计算机程序的各个部分产生了完整的规范说明。这些部分称为程序模块或简称为模块，在以后的章节里我们将在这个意义上使用该术语。注意迄今我们还没有写出任何程序。下一步就要做这件事。

五、编码

现在我们已经能够把问题的求解表达为一系列程序语句以便让计算机去执行。如果在前面的各步骤已经做了很好的工作，这一步就类似于把一种"语言"翻译成另一种。写出这种程序模块称为编码。重要的是应认识到在这一步我们并不是仍在寻找解决问题的方法，因为前面各个步骤已做好了这项工作，已经知道了所要做的事和如何去做，也知道了要告诉计算机些什么。当我们编写代码时，只不过是把我们所描述的意图，从计算机不能理解的形式转换成可理解的形式。

Pascal 本身的组织使得编码过程相当方便，特别是当编码前已经有了一组条理化的规范说明时。下面还将通过对程序模块的清晰、无二义性地描述，来充分利用 Pascal 的便利之处。有关这方面的讨论将从本章第三节开始。

六、模块的测试

用 Pascal 语句描述模块以后，并不能保证程序就是正确的。即使进行了最仔细的分解和分析，也仍然不能保证我们不会忽视某个细节或不犯错误。因此模块编码以后，必须进行系统地检验测试，查错改错，才能完善。由于整个问题是被分解成小的问题来解决的，因此我们可以对这些小的部分分别进行编制和调试。这就意味着当用计算机解决很大问题时，可以由几个人分工完成。这里我们再次体会到编制程序和开发其他复杂产品之间在概念上的类似性。

七、组合

系统开发过程的最后一步，就是要把各个片断组合在一起，看总体是否能正确工作。由于每个模块都已经检验过，组合过程主要是处理各个片断之间的连接。要让程序员找到装配各部分而需要调整的区域。

当第一次把这些片断连在一起时，期望完全适合是不现实的。但这种有条理的方法确实降低了出错的可能性。许多单位已经组织一套机构来推行某种系统地开发计算机程序的方法。

第二节　计算机和程序

在程序产生有用的结果之前，必须先将它装入计算机，然后通过程序指令来操纵机器的运行。本节将简单讨论这一过程。

一、计算机结构概述

计算机结构这一术语是用来表征一个计算系统的功能组织特性的。我们注意的焦点是主要部件做什么，它们又是如何连接的。对一具体结构的考察，可按不同的级别进行。对于低级的分析，是从各个电路来考虑计算机的功能特性。这种考察使我们深入了解信息是如何从计算机内的一个地方移动到另一个地方以及计算是如何完成的。在比较高级的分析中，具体

电路是"不可见"的，而把重点放在比较大的部件上。所感兴趣的是完成主要操作的方式和支持这些操作的信息传送方式。

计算机靠指令执行操作，每条指令都完成一特定操作，例如，两数相加，比较两数或把信息从一个地方转移到另一个地方。多数机器是一次执行一条指令，但越来越多的计算机能同时完成几个操作（称为多重处理机）。

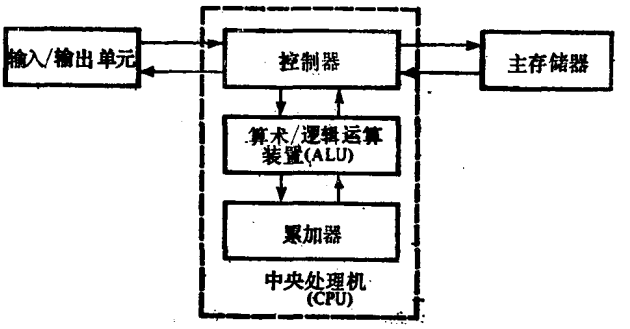

图1.2 数字计算机的基本功能部件

IBM PC系统的功能图见图1.2。注意，图中的方块并不一定对应于单独的物理部件。IBM PC一般组装成四个部件：中央处理机（包含磁盘驱动器）；键盘；屏幕显示器和打印机。

1．中央处理机

中央处理机是IBM PC的心脏，芯片是Intel 8088。系统的大多数计算都是由它完成的，而处理机外部所要完成的功能也是由中央处理机指导的。处理机所执行的指令，及这些指令操作的数据都存储在主存储器内。

主存储器分成独立的单元（字节），每个单元能存放一个单独的字符信息，即一个字母、数字或标点符号等。每个字节都用一个独特的地址表示它的物理位置。这就为存储，查寻及转移信息提供了依据。这些操作都和地址有关。例如，我们可以这样来描述一个典型的活动："拷贝当前存储在784地址的信息，拷贝的结果存在1077地址，同时也抹掉了该地址原来的内容"。Pascal提供了薄记服务，因而程序员可不去考虑这些细节。

当处理机执行指令时，真正的计算是由称为算术/逻辑运算装置(ALU)的电路完成的。这部分配备了具有存储功能的累加器。当需要用数据项进行运算时，就把该数据从存储器取到累加器进行运算。其结果可送到存储器指定的位置，也可保留在累加器做进一步运算。

控制器操纵全过程。为了执行程序，控制器必须从主存储器逐条取出程序指令。指令被检验以确定指定操作的类型，而控制器只是激发作业所要求的那部分ALU。当完成这一操作后，控制器就又执行下一条指令。这样，可以把处理机的主存储器和ALU看作辅助部分，它们根据控制器的要求提供指令、数据或其它计算服务。

2．外部设备

IBM PC有几种外部设备，用来与中央处理机传送信息。各种设备具有不同的数据形式、能力和操作速度。有些（如键盘）只能向处理机发送信息（输入设备）；另外一些(如，

监视器或打印机）只用来接收处理机信息（输出设备）；还有另外一些(磁盘或磁带),可在两个方向上用作信息转换器(输入/输出设备)。

有很多外部设备本身就有相当的计算能力，这是因为它们有自己的处理机，可控制自己的一些操作。但由于它们是由中央处理机的控制器的信号起动的，因而，这些操作仍然是辅助性的。

二、机器语言

IBM PC的机器指令是作为处理机的一部分设计的。因此，如果一个程序要能正确执行，那么它的指令一定要取自处理机所能"识别"的指令系统(即机器语言)。

机器语言是基本结构，其指令是用1和0的串(即二进制串)表示的。每条机器语言指令都指出了象简单加法或乘法这样的基本操作。(有些指令可完成更广泛的操作)。因此，任何用户所感兴趣的算法一般都得表示为一长串的机器指令。显然，用这种方法写程序是很令人沮丧的。幸运的是，象Pascal这样的语言可以省去这种繁琐的工作。

三、UCSD Pascal和机器语言

尽管计算设备本身的发展惊人，但机器语言却总是变化不大。指令仍须以二进制串的形式提交给控制器。为了减轻程序员的负担，不是从机器上想办法，而是开发越来越方便的程序设计语言，使程序员和计算机系统之间的交流按人所熟悉的方式进行。

Pascal这样的程序设计语言称为高级语言，是因为它们使程序员能够把复杂的处理活动当作一个操作来描述。程序员可以认为他所用的计算机就是直接处理高级语言的。协助程序员工作的是编译程序，它是一个翻译程序，能使程序员不必涉及机器语言的细节。UCSD Pascal编译程序体现了完整的Pascal结构规则。编译程序可检验原始程序(称为源程序)并确定所要求的计算。这样的分析结果，产生了用一台虚构的机器(称为P-机器)的语言写的等价的指令序列。这种语言称为P-代码。这些指令传送到P-机器的解释器，该解释器调用一系列机器语言指令执行P-代码程序。这虽然说起来复杂(似乎从Pascal直接到机器语言的翻译更为理想)，实际上使用假想的P-机器简化了把程序员的原始Pascal语句转换成等价的计算过程的工作。通常，每个Pascal语句形成几个比较简单的p-代码指令。其关系描绘在图1.3。

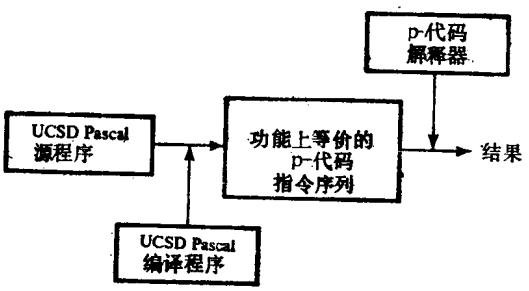

图1.3 UCSD Pascal程序的制备

第三节　算法的系统描述

把Pascal程序转换成p-代码表示形式，而最后是由机器语言指令执行，这一切不需要人的干预。首先，程序员必须按以前的一些规范说明形成正确的Pascal语句序列。这件事并非十分可怕。我们如果能保证规范说明尽可能清楚地描述程序应做的事，用它来指导程序编制就有很大的帮助。

系统地表达规范说明的方法有很多种。有一种通用的方法是伪码表示法。伪码概念的基本思想，是用模仿最后程序的形式来描述算法的规范说明。它把计算和其它一些处理操作都写成叙述句子而不是准确的程序语句。描述中使用的词汇和短语都没有严格的定义。我们讨论的重点是放在这些描述所出现的结构框架上。下面这一基本思想虽然简单，却是很重要的，因为它是结构程序设计的基础：

> "某些处理操作可作为所有程序的基本构造块。因此，可以把任一系列的计算完全描述成这些构造块的组合。若程序只用这些部分构成，就认为是结构良好的程序。这是因为这种程序清晰，易懂，也比那些不遵循这些结构概念的程序少出错。"

伪码为每一种程序构造块的表述规定了标准的方法。这样就比较容易写出结构良好的算法说明。Pascal的最大特点之一即是它的某些语句直接与这些构造块相对应。因此，从伪码到Pascal的转换是非常简单的。

伪码有多种版本。但差别仅是在结构成分的格式上有所不同。本书所选用的版本是最常用的一种；从一种格式改变成另一种格式，只需要几分钟的时间。我们将通过学习各个基本结构成分及其表达形式来熟悉伪码表示。

一、序列

最简单的操作是一个一个地依次执行一系列任务。从概念上讲，有多少任务或这些任务多么复杂都是没关系的。要点在于序列具有开始和结尾。当执行一个序列时，我们从开始部分启动，在结尾部分结束。

一个序列的伪码表示，可简单地由下列动作表组成：

```
Do this.
Then do this.
Then do this.
   .....
Finally, do this.
```

例如，设有某人交回租用的汽车，出租汽车的公司要计算帐单。计费是根据每天固定的租金，每英里固定的租金，保险费，汽车交回时装满油箱的汽油来计算的，并将总数打一个折扣。所需的数据包括汽车的标识号，租车者姓名，出租天数，行驶英里数，汽油费及折扣。此过程的伪码序列可表示如下。我们用"read"表示获取计算中所用数据的过程(即使信息进入处理机)，而"Write"表示显示或从处理机向外界输送数据的过程：

```
Read car i.d., name, no. of days, miles, gas
   charge, discount.
Compute daily charge.
Compute mileage charge.
Add daily charge, mileage charge, gas charge,
   and insurance fee to obtain total charge.
Compute net charge by applying discount rate
   to total charge.
Write car i.d., name, individual charges, total
   charge, discount, and net charge.
```