PASCAL语言及其程序设计

胡学联　潘金贵　编著
郑国梁　钱士钧　主审

南京大学出版社

PASCAL 語言
及其程序设计

胡学联　潘金贵　编著
郑国梁　钱士钧　主审

南京大学出版社
1987.南京

PASCAL语言及其程序设计

胡学联　潘金贵　编著
郑国梁　钱士钧　主审

*

南京大学出版社出版
（南京大学校内）
合肥市东方印刷厂印刷
江苏省新华书店发行　各地新华书店经销

*

开本：787×1092　1/16　印张：15　字数：346千
1987年8月第1版　1987年8月第1次印刷
印数：1—7,000

ISBN 7-305-00125-2/TP·10

统一书号：15336·030　定价：4.80元

内 容 简 介

本书较为全面地介绍了PASCAL语言，并详细讨论了自顶向下逐步求精的程序开发技术。全书分三个部分论述。第一部分介绍PASCAL语言的基本数据类型、几乎所有的语句、过程与函数以及自顶向下的逐步求精程序设计方法。第二部分讨论PASCAL中实用的数据类型。第三部分是其余深一些的内容。

本书叙述浅显易懂，示例丰富，习题难易适中，给出的几个实例研究能较好地供初学者学习、模仿。

本书适合用作函授教材，也可供参加计算机自学考试的人员作为教材或参考书。同时也可作为高等院校程序设计课程的教材或教学参考书。

前　言

有经验的程序人员大多见过这种情况：以某种语言编写的程序反而带有另一种语言的风格。那么，程序人员学习的第一个程序语言对其设计能力的影响是否象一个人的家乡话对其思维方式的影响那么大？如果承认有一定的影响，那么，拿什么语言作为程序人员的第一语言比较合适呢？软件工程心理学认为，程序设计是一种人的行为。还有人说程序设计是一种智力训练。如果真是这样，那么又应当怎样学习程序设计呢？人们自然会问，取PASCAL语言作为初学者的第一语言是否可行呢？以PASCAL语言为工具讨论程序设计又是否切实有效呢？回答是肯定的。这导致我们向读者——什么程序设计语言也未曾学过的读者——推荐、介绍PASCAL语言。

因此，本书目的有二：第一，打算用作以程序设计语言PASCAL为基础的计算机程序设计的初级教程。我们假定读者对计算机尚不甚了解，而只有一般的数学知识。本书强调程序设计原理，良好的程序设计风格及程序开发的系统方法。因而，本书对那些最终想用别的语言来编写程序的人也是很有用的。从程序设计的角度来看，我们的主要目的是教会读者如何去编写好程序。

第二个目的是提供一个PASCAL导引。本书对初学者采取循序渐进的方式加以引导，以便使读者能由浅入深的逐步掌握PASCAL语言的基本内容。

PASCAL语言是以法国数学家和哲学家Plaise Pascal（1623—1662）的名字命名的，以纪念他发明了第一台机械计算器。该语言是由瑞士的著名计算机科学家威尔特（N.Wirth）教授于一九七一年设计出来的。他的目标是设计一个有效的语言，这种语言能使程序设计的教学变成一种系统的训练；使用这种语言，许多程序设计技术都能既科学又经济地表达出来。

PASCAL语言简单、易学，写出来的程序简明、清晰、直观、易读，也便于查找和纠正错误，适用范围广泛，尤其适用于编写较大型的程序。因此，它一经问世，便引起人们极大的兴趣并倍受推崇，很快便被教育界、科技界、企业界接受使用。现有的计算机系统，大型、中小型、微型（如长城0520，TRS—80、IBM PC等），无不配有此语言。PASCAL以后的较有影响的语言，如Modula—2、Ada、XCY等，多是建立在它的基础之上的。

本书较为自然地划分为三个部分。第一部分共九章。详细地介绍了PASCAL语言中的基本数据类型，几乎所有的语句以及过程与函数，并讨论了基本的程序设计方法学。目的在于一开始就教会读者用一种系统的方法去编写和调试程序，并且能解一些基本的问题。第二部分共五章，主要介绍PASCAL中的常用数据类型，掌握了这一部分，便可以用PASCAL语言解比较复杂的问题。第三部分介绍了PASCAL的其余内容，初学者可以在第一次阅读时略去，以后再来逐步掌握它们。书中选学部分，均用*号标记。但

标有*号的章节中的内容，在随后的未标*号的章节中可能也用到了。

获得程序设计的系统方法，较好地掌握一种语言，最好的途径可能是有意识地模仿。但"创作从模仿始，模仿不是创作"。在模仿的基础上，努力发挥个人优势（如知识、智力、客观条件等），就能够设计出好程序，就能够比较得心应手地解决许多实际问题。解决问题一要方法，二要工具，二者不可偏废。因此本书着意避免那种只讲语言（工具）不讲程序设计方法，或只讲程序设计方法，而不和一个具体的支持这些方法的语言相结合的弊病。

书中所给例子，大多在IBM—PC机上使用IBM PASCAL验证过，练习分为难、易两种，前者也用*号标记。我们建议读者把你编写的程序都拿到计算机上去运行。为此，附录D介绍了IBM PC机上的PC—DOS支持下的PASCAL编译程序的使用，读者可以按所列步骤上机调试、运行所编的程序。附录E中还介绍了PASCAL S语言的使用。PASCAL S也是N.Wirth教授设计的，它是PASCAL的一个子集。该语言目前已在IBM—PC机上实现。对于初学者来说，先学习掌握PASCAL的子集，再进一步掌握完全的PASCAL，似乎也是一个可取的方法。

值此机会，特别感谢我们的老师徐家福教授长期以来对《程序设计》教学的谆谆教导。在本书编写过程中，南京大学郑国梁、钱士钧副教授自始至终给予了热情支持和具体指导，并且亲自审改了全部书稿，在此谨致深切谢意。作者还要感谢安徽大学程锦松高级工程师、安徽微型机函授大学钱洲胜副校长的大力支持和鼓励。张琪美、刘为民、周世兴、潘繁、王小琳等同志帮助做了大量抄写工作。作者对他们的帮助表示衷心感谢。

由于我们水平所限，加之时间仓促，书中不当、欠妥、谬误之处在所难免，恳请广大读者批评指正，以便再版时修改，日臻完善。

编著者

1987年6月

目　录

第一部分

第二部分

第三部分

第一章 总 论

1.1 引 言

人类从用手指头、绳结、筹码开始，相继创造了算盘，计算尺，手摇的和电动的计算机等进行加、减、乘、除等运算的计算工具。本世纪四十年代中期，出现了世界上第一台电子数字计算机ENIAC起，计算的能力得到了不断发展。现在，人们已使用电子计算机实现了银行结帐、财政报告、旅馆管理、旅游登记、办公事务自动化。工业上，用计算机控制机械工具和化学设备。科学家用它分析实验数据，医生用它产生“横断面”X光图形。心理学家用它模拟心理过程。载人或无人的空间探索没有计算机的帮助几乎是不可能完成的。人们还能在计算机上玩某些游戏，下棋等。计算机在军事上的使用则早就有了一个相当长的历史。

三十多年前，计算科学是一无所有，而现在则形成了作为世界大工业之一的一个产业，其发展趋势仍是迅速的。价廉物美的集成电路的发展，微型机的大量出现使得计算机不仅集团所有而且可以私人所有，从而把“阳春白雪”的时代，变为“下里巴人”的天下。它渐渐渗入人们的家庭生活，将成为更高级的“家用电器”。

关于计算机这一概念，实际上也应予以修正，它不再仅仅是计算的机器了。

计算机在许多方面的能力超过个人，这是毋庸置疑的。我们对计算机能力的较为恰当的评价是：计算机是一个愚人，除非告诉它怎么干，否则它什么也不会；虽然它是一个愚人，但一旦告诉它怎么干，它便能准确地、迅速地按规定执行。随着人工智能科学技术的发展，今后新一代的计算机，不再是一个愚人，而将具有“智能”了。

1.2 计算机硬件与软件

早期的计算机，它的高大的金属框架占据了好几个大房间。这些金属框架上装上了成千上万个真空管、热水银槽以及闪光灯配电板。这个庞然大物象小五金商的仓库一样，它很受人喜爱，以致当时的计算机工程师们幽默地把他们的这一发明创造称作“硬件”。现今一台功能强得多的计算机已经可以容易地放入公文皮包里了，但其工作原理仍是相同的。

每一台数字计算机的硬件包括一个处理器、一个存贮器和一些分门别类的外部设备。处理器是真正执行运算的部件。它包括一个指挥操作的控制部件以及一个算术运算部件，后者相当于一只电子计算器，但速度快得多。为了利用这种速度，处理器必须能够高速地存取数据。为处理器快速存取保留数据是计算机存贮器的任务。计算机的外部设备或输入输出设备就相当于计算器的数字键和显示部件。数据通过外部设备输给计算

机的存贮器，经过加工之后又通过外部设备输出去。图1.1示意了这种结构。

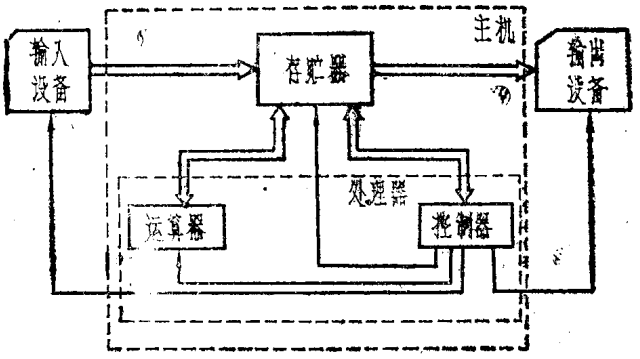

图1.1　计算机硬件示意图

注：图中双道实线表示数据传输路径，单道实线表示控制信号。虚线框内分别为主机和处理器。

但对于计算机用户来说，我们看到的只其是整体，如图1.2所示。

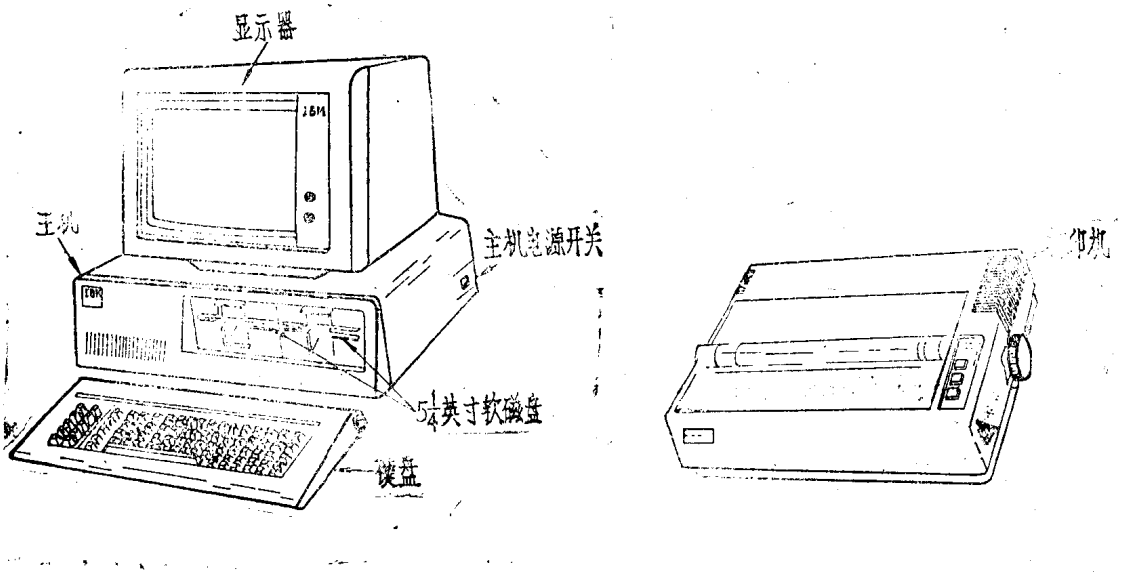

图1.2　IBM—PC　计算机系统

每种计算机，都能提供一组基本功能，说到底就是，它只能识别和执行一些基本的命令，比如两数相加、减，从输入设备上读一个数等。我们称这些基本功能为指令。不同类型的计算机有不同的指令系统。虽然，计算机所提供的基本功能是有限的，但人们可以利用这些基本指令的适当组合来完成一系列的工作。完成某一特定工作的指令序列称作程序。计算机硬件(如图1.1所示)的重要任务就是快速、准确地执行人们编写的程序。

与此相应地，在计算机系统中所有实用的程序以及有关的文件资料组成了该机的软

件（Software）。创造软件这个词是为了强调程序和硬件一样重要。这个词和硬件有效地形成了对照。一台计算机总是由硬件和软件两大部分组成的，硬件是由磁的、机械的、线路的部件构成的可见实体。软件则是依附硬件而存在的，它不会用旧（只会过时）。计算机系统的硬件一经确定后是不轻易更改的；而软件则需不断开发经常处于改变的状态中。可以形象地说：硬件是躯体，软件是灵魂。

没有硬件，便履行不了基本功能，那末软件也就失去了效用；但仅有硬件，没有软件，计算机也不能发挥它的潜在能力。比方说，某先生拥有一架钢琴（硬件，每个琴键好比一条指令），但他不会乐谱（软件），因而就不能弹奏出优美动听的乐曲，而只能发出噪音！

为了使计算机能很好地完成所给任务，提高机器的利用效率，使所有资源（诸如处理器、存贮器、外部设备及各种软件）协调一致，使机器有条不紊地工作，同时也为了能更方便地使用计算机，就必须有一套承担协调和管理职能的程序系统——**操作系统**。概括说来，一方面，操作系统是用户与计算机的接口（界面），用户通过操作系统使用计算机；另一方面，操作系统是计算机系统中（硬件与软件）的控制中心。操作系统本身也是软件，我们称其为**系统软件**。

前面，我们已经说过，计算机只能识别机器指令（代码），因此，最初的计算机是用机器代码编写程序的，亦即用数字形式直接给出机器指令。然而，其缺陷很快被人们识别到了：

（a）由于机器指令是最原始的，这引起了机器代码程序的设计既冗长又易出错。

（b）同样的原因，对于机器代码程序的理解和修改是困难的。

（c）编写程序是费时、高代价的工作。如果能在计算机之间交换程序，那便是一种很大的节约。但是机器代码程序是局限于某一类型的计算机且在另一台不同类型的机器上是不能运行的。

为了克服上述缺点，并使得没有受过专门训练的人都能使用计算机，就得把计算机语言改造得直观一些，使得用起来方便一些。

最早产生的是（符号）汇编语言。它并不直观，但较易于记忆。它没有很好地解决使用机器语言的缺陷，因此人们又在探求更好的语言，这便导致了**高级语言**的出现。本书介绍的PASCAL语言是这些高级语言中较为理想的一种。

有了高级语言之后，人们就可以用它来解实际问题。但是一个用高级语言书写的程序是不能由计算机硬件直接执行的，必须把用高级语言写的程序翻译成等价的机器指令程序。翻译工作是由一个计算机程序十分正确地完成的，这个程序叫**编译程序**——它也是系统软件。

上述两项工作——为机器配操作系统和语言编译程序——实际上拓广了计算机的能力，由此可以得到一个语言计算机。我们以配PASCAL编译程序的机器为例给出一个图示（见图1.3）。我们称这个拓广了的计算机为PASCAL语言计算机。

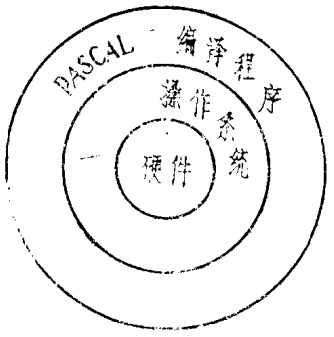

图1.3 一台PASCAL语言计算机

1.3 软件工程：目标和原理

如果计算机至今只是算算题目，则计算机就不会成为一门科学。我们希望计算机“无所不能”，希望它在许多应用领域发挥巨大作用。我们通常设计出来的程序实际往往很大，很复杂，有的几千行，有的几万行甚至几十万行。因此常称为**软件系统**。设计这样的软件系统需要大量的集体的智能活动。理论上，技术上，管理上都有很大的难度。实践表明，用50年代计算机程序设计的工匠技巧来应付这种大型软件系统的设计，已不再有效，而且会带来一系列严重问题。失败的例子比比皆是。软件发生了危机。

一九六八年前后在Garmisch召开的计算机科学家会议上第一次广泛承认软件危机的存在，并且提出了“软件工程”的概念。人们认为，要得到廉价的软件，它是可靠的，而且能在实际机器上有效地工作，就必须建立和应用牢固的**工程准则**来达到。用修建乡村小溪木桥的方法和工具，去建造南京长江大桥显然是根本不可能的。这里，**方法、工具**（还有管理）便是问题的关键。它们是软件工程的支柱，也是软件设计人员要致力解决的主要问题。计算机的专家发现，软件危机的解决，主要出路之一在于应用一整套由高级语言支持的现代软件设计方法学，这种高级语言鼓励并且强迫实施方法学中规定的那些原则。PASCAL语言在许多方面满足了这种要求（但不完全！）。

1.3.1 软件工程的目标

软件设计的一个最显然的目标是程序的执行结果要满足用户需求。软件工程总的目标是可修改性、有效性、可靠性、可理解性。

可修改性

软件系统的用户可能会随时提出一系列的修改要求，即用户需求发生变化，或者一个软件系统在较早的开发阶段中隐藏的错误被发现之后，必须修改。如果这种修改会牵一发而动全身，则是十分危险的。很有可能改了一处又错了十处。因此，一个可修改的软件系统应该可以控制变化，即某些部分或方面保持全然不变，只要改变其它部分或方面，即能期望得到新结果。

为了有效的修改系统，必须尊重已有的设计结构。如果不考虑原先的设计而修改软件，就会破坏软件的原有的逻辑结构。几次修改之后，原先的设计面貌全非，系统的修改会变得更加困难。如果系统是可修改的，就应当允许进行修改而不会增加原先系统的复杂性。

有效性

有效性的目标是指软件系统以最佳方式使用可用资源的情况下运行。资源可分为两类：时间资源和空间资源。简单地说，前者指执行速度，后者指占用机器内存和外部设

备。有时必须权衡利弊，或以牺牲时间换取空间，或以牺牲空间换取时间。在许多情况下，同时有效地使用两种资源是不可能的，这时要有一种折衷处理的办法。

在开发软件系统的过程中，要注重宏观上的有效性，不应在一开始就把主要精力集中在有关许多有效性的细节上面。

可靠性

对于不用人干预的、长时间运行的计算机系统，可靠性是一个很重要的目标。如果这样的系统控制很重要的目标，例如核电力工厂或宇宙飞船系统，则失败的代价太高了，所以不允许有失败的情况发生。可靠性必须既要防止因概念、设计和结构不完善造成的失效，又能挽回因操作不当等所造成的失效。

可靠性必须存在于整个软件设计中。只能在设计一开始就保证可靠性；无法在结束时附加上它。完美无缺的可靠性是不存在的，问题在于出现故障，可靠的系统应能把故障的影响减至最小，或系统本身降级使用。

可理解性

软件工程的最后一个目标是可理解性，也许它在帮助我们管理软件系统复杂性方面是最关键的。可理解性是特定的问题与对应的解之间的桥梁。换言之，一个可理解的系统必须直接反映问题的本来面目。问题的解必须有清晰的结构。在软件开发中把握住这一点，对于构造可修改的、有效的和可靠的系统来说是基本的。

一个有层次的软件系统将有助于可理解性。在最低层，有适当编程风格的软件应当是可读的，在较高层，能够容易地分解出解中的数据结构（对象）和算法（操作）。我们将看到，可理解性很大程度上依赖于作为表示解的工具的程序设计语言。

一个成功的系统，往往很难同时完全达到上述四个目标。当发生冲突时，首先是把可靠性放在第一位，而有时或许要以牺牲效率为代价来换取易理解性。具体情况要具体分析。

上述目标似乎不高，但是，过去的经验告诉我们，要真正做到却是相当困难的。原因何在呢？基本原因就是要求软件设计人员（包括程序人员）具备多种素质，另一原因是客观上应当有一套供设计人员使用的软件设计方法学及基于这种方法学的工具。本书感兴趣的是后面一个问题。

1.3.2 软件工程的原理

上一小节讨论的目标实际上对任何软件系统都是合理的，当然它们也是程序设计所应追求的目标。问题是，我们不能一方面承认这种目标，另一方面则使用无规则的开发方法来试图达到这些目标。相反，设计软件时，必须遵循一系列简单的支持这些目标的软件工程原理，它们是：

- 分解
- 抽象
- 信息隐蔽
- 模块化
- 局部化

•一致性
•完整性
•可验证性

我们简要讨论如何应用这些原理达到软件工程的那些目标。

分解

应付复杂问题的重要手段是“分解”，即分而治之。日常生活中的例子如：一个大学可以分成若干个系，一个系又可以分成几个教研室等等。分解的含义是将一个复杂的问题分割成若干小的、较容易解决的部分，然后分别进行处理。分解后的各个部分应相对独立，以使复杂的问题简化成若干独立的小问题。随意的分解则有可能使问题更加复杂化。在此原则上当然可以允许有各种不同的分解方法。分解原理有助于可修改性，可理解性。

抽象和信息隐蔽

抽象和信息隐蔽可以看成是同一问题的两个方面，它们也是应付复杂问题的重要手段。抽象是突出“做什么”，信息隐蔽是将“怎么做”隐藏起来，这样便能使问题得到简化、易于理解（增强了可理解性）。当对一个问题进行逐层分解时，也就体现了抽象。具体说来，抽象的目的是确定基本内容，隐蔽的目的则是使影响系统其它部分的某些细节变得不可访问。抽象不仅是对算法而言，对数据也可以进行抽象。这两条原理有助于软件工程诸目标的实现。

模块化和局部化

帮助我们控制软件系统复杂性的另一个原理是模块化。人们分解一个问题时，实际上就是在生成一些模块。

在最一般的意义上，模块可以是子程序（函数或过程）。这种模块常称为功能模块（一个模块完成某一特定的功能）。然而，按信息隐蔽原理构造的模块不妨称为现代模块。

局部化原理要求一个模块是充分独立的，从而使得“组合式”模块成为可能。

模块化和局部化原理直接支持可修改性、可靠性和可理解性。

一致性、完整性和可验证性

一致性的原理直接支持可理解性的目标。简单说来，一致性指各模块使用一致的符号并且没有不必要的差异。其中包括统一化、标准化。一致性大多由好的编程风格得到。完整性在于保证不丢失任何重要成分。一个问题分解成若干部分之后，从总体上看应是完整的。

可验证性原理指出分解系统时必须使它能被容易的测试，从而也使得系统可以修改。完整性和可验证性难以实施，大多数情况下要借助一些特定的工具。

总的说来，这些原理在于确保软件工程目标的达到。换言之，为了达到软件工程的目标，各种软件开发方法应当遵循一组基本的原理，下面一节，我们就来简要介绍一些典型的程序设计方法学。

1．4 程序设计方法学

有哪些程序设计方法学能实现软件工程原理呢？基本上有四大类提供一致标准的设计方法学：

（1）自顶向下的结构化设计

（2）结构数据设计

（3）Parnas分解方法

（4）面向对象的设计

本书主要讨论和使用第（1）种方法，并且也简要提到了其余三种方法，但未进一步研究它们的使用。

本节先概略地介绍一下这几种方法学。

自顶向下的结构化设计建议分解系统时，逐步降低问题的抽象级。即在解的较高层，定义抽象算法（做什么），解的较低层给出实现高层动作的基本操作。

结构数据设计的设计方法学在数据处理类型的应用中较为有效。使用这种技术时，首先定义数据结构，然后才构造基于数据结构的操作子程序。

对于Parnas分解方法，分解时要使组成解的各个模块都隐蔽一个设计。使用这种方法，能够明显地抓住决定设计决策的层次，得到软件设计的结构。如果以后要修改解法，它能比较容易地限制修改的影响范围。

面向对象的设计方法学，简单说即是，首先在求解的一个给定层次确定抽象的对象（数据等），定义对它的适当操作，然后对每一个对象开发一个隐蔽起实现的模块。它充分认识到对象作为主体（而不是动作为主体）的重要性。

1．5 程序设计语言

为了建立计算机的解，只有方法学是不充分的，还必须要有合适的工具，特别是要有程序设计语言这样的工具来表达和执行设计。最流行的一些语言可列出如下：

FORTRAN Ⅰ	
ALGOL 58	
FLOWMATIC	
IPL V	
FORTRAN Ⅱ	子例程，独立编译
ALGOL 60	分程序结构，数据类型
COBOL	数据描述，文件处理
LISP	表处理，指针
PL/1	FORTRAN+ALGOL+COBOL
ALGOL 68	ALGOL 60扩展了的后继
PASCAL	ALGOL 60的简单化了的后继