PASCAL 程序设计

〔加〕P. 格罗戈诺 著

PASCAL 程序设计

〔加〕P. 格罗戈诺 著

李三立 林定基 译

科学出版社

1985

内 容 简 介

PASCAL 语言是按结构程序设计原则设计的一种语言，可用于编写系统软件、应用软件以及进行科学计算和教学等。它已成为目前最流行的几种程序设计语言之一。

本书共分十章。第一章为计算机程序设计的基本概念。第二、五、六、七、八等章阐述 PASCAL 语言的各种数据结构及其用法。第三、四章介绍 PASCAL 的各种语句。第九章讨论几个较深入的课题。最后在第十章中介绍实用程序的编写、验证和调试的技术。附录部分提供了用 PASCAL 语言进行程序设计所必需的资料。

本书可供计算机科学系统软件设计人员、数值计算工作者、科研人员、工程技术人员和科学管理人员阅读，也可作为理工科高等院校的程序设计语言的教材。

Peter Grogono

PROGRAMMING IN PASCAL

Addison-Wesley, 1979

PASCAL 程序设计

〔加〕P. 格罗戈诺 著

李三立 林定基 译

责任编辑 李淑兰 刘晓融

科学出版社出版

北京朝阳门内大街 137 号

中国科学院印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1982 年 2 月第 一 版 开本：850×1168 1/32

1985 年 4 月第三次印刷 印张：11 7/8

印数：20,801—42,300 字数：311,000

统一书号：15031·388

本社书号：2455·15—8

定价：3.35 元

前　言

计算机程序设计语言 PASCAL，是最早用条理清楚的方式，体现出 Edsger Dijkstra 和 C. A. R. Hoare 所定义的结构程序设计概念的一种语言．它本身就是程序设计语言发展过程中的里程碑．PASCAL 是苏黎世 Eidgenossische 工科大学的 Niklaus Wirth 提出的，它是从 ALGOL 60 语言发展出来的，但其功能更强，使用更为方便．现在 PASCAL 已被广泛采用，成为一种实施有效的很有用的语言以及优良的教学手段．

本书目的是为愿意用 PASCAL 写程序的读者所编写的．它并不使用任何其它程序设计语言的知识，因此它很适合作为一个入门的课程．它还可以和离散结构的教科书结合在一起，在更为详细的'程序设计原理'的课程中使用．

第一和第十章讨论计算机程序设计的一般原理．第十章自然比第一章内容深些，但这也不必等到最后再读，其中大部分内容是无需 PASCAL 高级知识就可以学懂的．如果把它和本书其余几章一起学也是很有益处的．第二、五、六、七和八章讨论 PASCAL 中数据管理和结构，第三和四章介绍 PASCAL 语句．这样的处理方法是有系统的，它省略了语言中的一些较为深奥的特点．这些特点在第九章中做了整理．如果读者已经学过一种程序设计语言，则将会觉得前面几章很容易看．然而，在 PASCAL 中，基本概念的定义要比其它一些语言更为精确，所以读者最好不要跳过这些章节．

用高级语言，如 PASCAL，而不是用低级的汇编语言来做程序设计，其理由有两个：首先，用高级语言编写程序较为容易．其次，也是更重要的，用高级语言写成的程序更为易读和易懂．职业程序设计员花费很多时间去修改他们自己或别人写的程序，如果

他们没有完全弄懂这些程序，那他们修改时的意图实际上会毁坏了这些程序. 因此，要学会如何去读和去修改程序，这和如何学会写程序是同样的重要. 虽然本书中的程序，大部分较短，较为简单，但它们不是片段的，而是完整的可以工作的程序. 读了这些程序，你可以获得对这些程序的初步了解，但为了真正掌握它，你应该去改进这些程序. 本书中有很多练习，这些练习指出如何在给定例题的基础上进一步提高的途径，做了这些练习，有助于更快和更为顺利地学会 PASCAL. 其它的练习，大多是要求你写出独创的程序. 这些规定的程序并不是轻而易举的，它需要做相当大量的工作. 如果花一个月的时间，去写一个一定规模然而是正确的程序，这比花费同样的时间去写若干个不现实的或者不能正式运行的程序更有教育意义. 同样，作为一个工作小组的成员去学习程序设计，比作为个人去学习也更为有教益，而第六、七、八和十章的一些练习，其分量之大是足够一个工作小组来承担解决的.

在用 ALGOL 系列语言写的程序中，一般习惯是把保留用字印得突出一些. 在排版的材料中，保留用字是用小写黑体字，而标识符是用斜体字. 在打字的材料中，这是不实际的. 而另一种通常的代用方法(即在字体下面划横线)又不美观. 因此本书中所用的记号是和普通方法不同的. 我选用大写字体印保留用字，而小写字体印标识符，结果程序是整齐而易读的.

在准备写此书时，我得到很多帮助和鼓励. 我特别感谢 H. F. Ledgard 和 D. C. Oppen，以及 Concordia 大学计算机科学系很多成员所给予我的指导，建议和指正. 我还感谢 S. H. Nelson，他耐心地阅读并编辑很多手稿. 我也感谢 R. Seppanen，他打印了最后的定稿. 我对书中还存在的错误承担责任. 最后，我对 Concordia 大学的 Cyber 172 计算机的"勤奋努力"表示欣赏，这台机器用 Urs Amman 的 PASCAL 编译程序，验证了例题的程序.

目　　录

第一章　程序设计概念 …… 1
1.1　程序 …… 1
1.2　结构 …… 2
1.3　PASCAL 的初步介绍 …… 5
1.4　编译和执行 …… 8
1.5　表示法和例子 …… 12
第二章　数据，表达式和赋值 …… 19
2.1　标识符 …… 19
2.2　文字和常数 …… 22
2.3　数据 …… 25
2.4　整数类型 …… 28
2.5　实数类型 …… 33
2.6　布尔类型 …… 37
2.7　字符类型 …… 41
2.8　程序结构 …… 44
练习 …… 51
第三章　判定和重复 …… 55
3.1　*IF* 语句(条件语句) …… 55
3.2　*REPEAT* 语句(重复语句) …… 62
3.3　*WHILE* 语句(当语句) …… 66
3.4　*FOR* 语句(循环语句) …… 73
练习 …… 77
第四章　过程和函数 …… 80
4.1　写过程 …… 80
4.2　函数 …… 95
4.3　递归 …… 99
4.4　非局部变量和副作用 …… 116

4.5 伪随机数 …… 117
练习 …… 121
第五章 变量类型 ……125
5.1 标量 …… 125
5.2 子界 …… 128
5.3 集合 …… 130
5.4 *CASE* 语句(分情形选择语句) …… 139
5.5 重新分析程序 calculator …… 142
练习 …… 149
第六章 结构类型 ……152
6.1 数组 …… 152
6.2 记录 …… 173
练习 …… 193
第七章 文件 ……198
7.1 顺序文件 …… 199
7.2 文本文件 …… 203
7.3 输入和输出 …… 207
7.4 举例 …… 214
7.5 子文件结构 …… 227
练习 …… 228
第八章 动态数据结构 ……230
8.1 指针 …… 230
8.2 链接表 …… 234
8.3 举例：离散事件模拟 …… 247
8.4 树 …… 257
练习 …… 264
第九章 高级课题 ……268
9.1 *GOTO* 语句(转语句) …… 268
9.2 过程和函数作为参数 …… 272
9.3 存贮分配 …… 278
练习 …… 283
第十章 程序设计 ……284

10.1 程序研制 …… 285
10.2 测试和检验 …… 289
10.3 调试 …… 301
10.4 举例：交叉索引生成程序 …… 303
10.5 PASCAL 的评价 …… 318
练习 …… 320
深入一步的阅读资料 ……322
附录A PASCAL 用语表 ……332
A.1 保留字 …… 332
A.2 标识符 …… 232
A.3 标点符号 …… 333
附录B PASCAL 语法 ……335
附录C PASCAL 的一种实现方案……345
C.1 标准类型 …… 345
C.2 算术运算 …… 347
C.3 标准过程和函数 …… 347
C.4 输入和输出 …… 348
C.5 文件 …… 349
C.6 分段文件 …… 350
C.7 外部过程 …… 352
C.8 编译程序任选项 …… 354
附录D 程序标准 ……356
D.1 程序描述 …… 356
D.2 注释 …… 357
D.3 说明 …… 357
D.4 布局格式 …… 358
D.5 可移植性 …… 360
D.6 自动格式化 …… 362
索引 ……363

第一章　程序设计概念

1.1 程　　序

程序是一个指令的序列．一份食谱，乐谱以及毛线编织图案都是程序．在这个意义上说，程序在计算机发明以前早就存在了．然而，计算机的程序比起其它类型的程序要更长而且更为复杂，因而写起来需要更加小心和精确．在我们更深入观察计算机程序以前，我们将定义一些普通的术语，并且考虑程序设计系统中所共有的一些特点．

程序，需要有个作者去写它，也需要有个处理器去做这些指令．做这些指令，叫做执行或者运行程序，而运行程序称为处理．执行一份食谱，这叫做烹调，而厨师就是处理器．一份乐谱是为了演奏一段音乐所需要的一组指令，而演奏家则是处理器．程序有以下一些共同的特点：

（1）指令是顺序地执行的．除非已有明确说明，否则就从第一条指令开始，依次执行每条指令，直到结束为止．这种一般的模式在某些明确规定的方式下可以变更，譬如当你要重复某一段音乐的时候，就不是顺序执行的．

（2）处理过程会得到一个结果．这个结果可以是一顿饭，或者是音乐的音响．如果这个程序是一个计算机程序，那么这个结果的形式往往是打印或显示符号的输出．

（3）程序是对某些对象操作的．指令'磨豆蔻'就表示要把一些豆蔻磨碎．而计算机程序操作的对象是数据．

（4）有时在指令的前面，先要有指令操作对象的说明．这一点对食谱也是如此，在食谱前面经常有所需配料的列表．在很多程序设计语言中，程序员在写他的程序以前，必须先说明所用数据

的属性.

(5) 有时指令需要处理器做出判定. '假如你用新鲜西红柿，要剥皮，并在放洋葱之前放入；但假如你用的是罐头西红柿，则在最后加入'. 在这种情况下，指令的作者并不知道在某一特定场合下处理器要做什么，但他可以确定一个准则，而处理器将根据这一准则做出判定.

(6) 可能需要不止一次地执行一条指令或一组指令. 这在编织毛线和钩织花边时是经常需要的，因为这些工作本身就是重复的过程. 当重复一条指令时，必须规定重复的次数. 这可以直接给出需要重复的次数('编织十行')，或者确定一个取决于过程状态的准则('编织到这一行结束为止'). 这两种重复的方式在计算机程序中都经常发生. 因为现代计算机可以每秒执行一百万条以上的指令，一个没有重复的程序运行时间不会超过几分之一秒.

(7) 程序本身是一个静态的整体，但执行这些指令的过程是动态的. 我们不要把厨师和食谱，或者把钢琴家和乐谱混淆起来，同样重要的是不要把处理器和程序混淆起来.

以上这些特点是所有各种程序共有的，包括为计算机写的程序. 我们看到，程序本质上是程序的作者与处理器用来通话的手段. 通话就需要一种语言，虽然诸如英语这种自然语言经常用作非正式的指令，但大部分程序设计任务需要一种特殊的语言. 甚至食谱也要使用自然语言的一种专门化的术语，而音乐家，舞蹈设计师和毛线编织者也已发明完全独特的语言，并用这些语言来传送他们的指令.

1.2 结　构

最早的计算机程序只不过是一些计算机能直接执行的原始指令的列表. 随着时间的推移，写出了更为复杂的程序，而这些列表变得难以管理. 其原因是它们缺乏结构性. 对于机器来说，执行包括有几千条指令的列表是没有问题的，因为机器只是机械地执

行每条指令而不去管它们的意义或结果. 但对于程序员来说，要去弄懂成千条看不出显著差别的指令的列表，却是一个难以解决的问题. 程序设计语言的历史，在很大程度上可以认为是如何给这些指令的原始列表赋予一定结构性的历史.

结构性首先表现在表达式的计算中. 假如我们必须编写一个计算机程序来计算面积的数值，其中

面积 $= 3.1415926535 \times 5^2$

在早期，程序设计员要写如下一系列的指令：

```
输入  3.1415926535
乘    5
乘    5
存贮面积值
```

程序员必须注意正确地解释表达式. 举例说，表达式

$$2 \times 3 + 4$$

相应于这个程序

```
输入  2
乘    3
加    4
```

但表达式

$$2 \times (3 + 4)$$

则相应于以下程序

```
输入  3
加    4
乘    2
```

程序员理解到，从符号表达式翻译成机器指令的列表是机器所能执行的一种机械操作，更进一步说，计算机是做这种事的很合适的机器. 此后，程序设计员就用普通代数形式来写表达式.

其次一个需要结构化的是数据. 计算机具有几千字的存贮

器，而一个字可以包含一个数，一组字符，或者一条指令．程序设计员可以决定，在存贮器第 300 到 399 的单元中存放一组一百个击球手的平均得分数，而以后他可能忘记了这个决定，而把其它一些东西存放在那里．但如使用具有数据结构化的语言，我们就可以写成

```
averages: ARRAY [1..100] OF integer;
```

并知道，它将保留存贮器的一个区域不再作其它任何使用．

当数据结构化的问题至少部分地被解决后，人们又把注意力转向改进指令本身的结构化．可以证明所有计算机程序都能用四种基本的结构来表达．这四种结构是：**顺序**，**判定**，重复结构或**循环**，以及**过程**．顺序是一条接着一条执行的一组指令．判定是一种结构，它使程序的动作可以受到数据的影响．很多语言用 '*IF*' 这个字来引入判定的结构，用它可以写出如下指令

```
IF  x≥0
  THEN  y:=x
  ELSE  y:=-x
```

循环结构用来多次地执行一条或者一个序列的指令．虽然每次循环执行时，指令是相同的，但它们操作的数据却并不相同．例如，重复下列指令

```
add 1 to x
```

一百次，其结果是 100 加 x．我们必须很小心地注明循环中的指令执行了多少次．如果我们假设起初 $x=0$ 和 $y>0$，则程序

```
repeat
  add 1 to x
until  x²>y
```

将会给予 x 以其平方超过 y 的最小整数值．这个程序是安全的，因为我们可以担保一定能找到这样一个值．而程序

```
repeat
  add 1 to x
until  x²=y
```

是不安全的．譬如说，如果 $y=5$，则条件 $x^2=y$ 永远不能成立．理论上这个程序将永远在循环中转不出来，但在实际的计算机中，

程序将运行到 x 足够大，以至于 x^2 不能在机器上表示时，将会停止.

过程使我们可以用单条指令来代替一组指令. 过程经常在烹调书中使用. 一本好的烹调书会提供例如做奶油调味汁的过程，然后在每份需要这种奶油调味汁的食谱中引用这个过程. 在计算机程序设计中使用过程，不仅使程序写起来较短，而且较为容易，更重要的是，它给程序一个层次分明的结构. 过程的概念很重要，如果没有它，就不能写出有用的计算机程序.

程序设计语言 PASCAL 应用所有这些结构化的技术. 它们结合在一种简单而优美的形式中，从而使 PASCAL 成为一种强有力的、然而却是易学易用的语言.

1.3 PASCAL 的初步介绍

在本节我们将学习若干个非常简单的 PASCAL 程序，通过这些程序来看一下前节所描述的结构在具体的程序设计语言中是如何实现的. 本节中的三个程序是完整的可以工作的程序. 这些程序可以在计算机中运行.

```
PROGRAM squarerootoftwo (output);
  BEGIN
    write (sqrt (2))
  END.
```

这个程序如由计算机执行，将打印出

1.4142135624

它近似为 2 的平方根值，其中第三行

```
write (sqrt (2))
```

是程序 *squarerootoftwo* 的核心. *write* 是一个过程，其结果是把 *sqrt* (2) 的值打印出来，*sqrt* (2) 是它的自变量. *sqrt* 是一个标准 PASCAL 函数. *sqrt* (2) 的值为 $\sqrt{2}$，其精度受计算机的限制.

程序的第一行包含有 *PROGRAM* 这个字. 所有 PASCAL 程

序的第一个字都用这个字. 它的后面是 *squarerootoftwo*，这是程序的名字. 程序的名字是由程序员选择的，一个好的程序员会试图选择一个能反映出程序功能的名字. 在名字后面，我们要确定程序及其与外界环境的关系. *output* 这个字指出，这程序要产生某些结果. 在现代计算机中运行程序的外界环境通常就是操作系统，在这个例子中的操作系统的作用是接受程序产生的结果，并把它们发送到一个打印机或者一个终端.

最后，注意 *BEGIN* 和 *END* 这两个字，顾名思义，这两个字表示程序的开始和结束.

程序 *squarerootoftwo* 不是很有用的. 因为我们如要知道$\sqrt{2}$的值，只要查一下书就可以了. 下面的程序是程序 *squarerootoftwo* 的一种改进的形式：

```
PROGRAM squareroot (input, output);
  VAR
    x: real;
  BEGIN
    read (x);
    write (sqrt (x))
  END.
```

这个程序阐明了**顺序结构**. 这里有两条指令 *read* (*x*) 和 *write* (*sqrt* (*x*)). 这两条指令将按照所写的次序一条接着一条地执行. 而 *BEGIN* 和 *END* 两个字如同是把指令序列括起来的括弧. 这个程序中只包含一个这样的序列，因此只有一对 *BEGIN-END*. 较为复杂的程序包含很多指令序列，每个序列都要用 *BEGIN* 和 *END* 括起来.

程序 *squareroot* 从外界环境接受数据. 第一行包含有 *input* 这个字，它表示程序将请求数据. 而过程 *read* 则具体实行这个请求. *read* 指令的作用是从输入设备获得一个数值，并把这个值赋给 *x*. 输入设备是什么，程序并不知道. 我们可以认为，这个数值是用卡片穿孔，或者是在终端键盘上打字输入的.

程序 *squareroot* 引入的第三个新的概念是**数据**. 这里我们称

为 x 的对象是一个实数变量，这是在说明中宣布的

```
VAR
  x: real;
```

这个说明规定了变量 x 的属性是实数类型。

程序 *squareroot* 仍然不是一个很好的计算机程序例子。假如 x 为负，则无法求 *sqrt* (x)，然而并没有办法防止用户输入一个负数。此外，计算好平方根以后程序就停止了，要计算另一个值时必须重新开始。这个缺陷在以下程序 *squareroots* 中可以弥补。

```
PROGRAM squareroots (input, output);
  VAR
    x: real;
  BEGIN
    REPEAT
      read (x);
      IF x≥0
        THEN write (sqrt(x))
        ELSE write ('argument error')
    UNTIL x = 0
  END.
```

这个程序已包含有**判定**和**重复**的结构。读了一个 x 的值以后，处理器必须在两种作用过程之间选择一种。假如 x 的值是非负的，则 $\sqrt{x}$ 的值必须打印出来；假如 x 的值为负，则要打印出如下信息

```
argument error
```

此外，*REPEAT* 语句和与它相配的 *UNTIL* 语句将确保程序继续运行，一直到读出一个零值为止。

在这个程序中，虽然处理器是顺序执行指令的，但它们执行的次序，已经与它写的次序并不相同。在写出一个平方根以后，处理器将测试 x 的值。假如 x 是非零，处理器将重新执行 *read* 指令。这个过程只是当条件

$$x = 0$$

为真时才终止。我们必须区分静态程序与执行程序的动态过程之

间的差别. 当我们随便说: '程序做某某事', 这实际上是把上述两者之间的差别混淆了. 记住,上面一句话只是下句话的简缩语,即'处理器在执行程序时做某某事'. 一个程序是一个文本; 而一个处理器是一部机器,它执行包含在文本中的指令.

虽然程序 *squareroots* 的使用范围相当局限,但它是一个完整的程序. 它的终止条件(遇到某数,其值为零)并不是令人满意的,在本书中我们以后会看到更为完美的终止程序的办法.

1.4 编译和执行

当程序设计语言的结构程度愈来愈高时,计算机程序显得更便于人们编写和分析. 与此同时,把程序翻译成计算机能够执行的简单指令的任务,也显得更为艰巨. 因此我们希望能找到一个与任一程序设计语言相联系的程序,它的作用是把该程序设计语言写成的程序翻译成机器指令. 这种程序叫做该语言的**编译程序**. 把 PASCAL 程序翻译成机器指令的程序,叫做 **PASCAL 编译程序**,本书中我们将经常用到这个程序.

假如我们写的程序一直是正确的,而且使用的是一个很好的编译程序,这样,在实际上我们可以把编译程序和计算机看成是一部机器,它好象是能够直接去执行程序. 也就是说,我们可以完全忽略其中翻译的过程. 编译程序和计算机这种方式的组合,有时称为是一个**虚拟机器**. 现在有很多很好的 PASCAL 编译程序,因此可以把一个 PASCAL 编译程序和一台合适的计算机看成是一台 PASCAL 机器.

PASCAL 机器如图 1.1 所示,在这里和以后的图中,圆圈表示静态的整体,如程序; 而矩形表示处理器. 线条表示送到处理器或者从处理器接收的数据流,其方向用箭头标明. 图 1.1 表示 PASCAL 机器需要有一个源程序,即任何 PASCAL 程序,以及若干输**入数据**. 当 PASCAL 机器运行时,它产生**输出数据**. 如果我们假设源程序就是程序 *squareroots*, 而输入数据是:

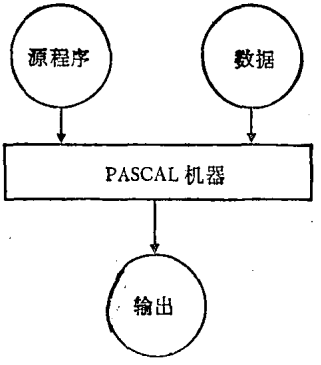

图 1.1 PASCAL 机器

```
2
3
4
0
```

则当我们启动 PASCAL 机器，它将运行很短片刻，并在输出文件中打印出如下结果：

```
1.41421356
1.73205088
2.00000000
0
```

如同我们所见，虚拟机器是一台计算机和一个编译程序，如图 1.2 所示．现在处理器就是计算机本身，它使用了两次．在第一次使用中，程序是 PASCAL 编译程序，而输入数据就是源程序．其结果是把源程序翻译成机器指令，这叫做**目标程序**．于是计算机再使用一次，而把目标程序作为它的程序，我们的数据作为输入数据．其结果是我们所要求的输出数据．

我们无需注意 PASCAL 机器的内部结构，但要知道出现错误的可能性．这里有三种严重的程序设计错误：

（1）在翻译过程中查出的错误．例如，写了 *BEGIN* 以后而漏写了与其对应的 *END*．这叫做**编译时的错误**．