卢雄远　蒋代梅

标准-PASCAL程序设计学习与实践

中央广播电视大学出版社

标准PASCAL程序设计

学习与实践

卢雄远　蒋代梅

中央广播电视大学出版社

标准 PASCAL 程序设计

学习与实践

卢雄远　蒋代梅

*

中央广播电视大学出版社出版
新华书店北京发行所发行
解放军七二二六厂印装

*

开本 787×1092　1/16 印张 11　千字 275
1987 年 1 月第 1 版　1987 年 6 月第 1 次印刷
印数　1—16000
书号 15300·62　定价 1.85 元

前　言

本书是为配合中央电大 Pascal 语言课程而编写的辅导教材。它可以作为这门课程所用数材《标准 Pascal 程序设计》(丘玉圃教授编著，本书中简称为课本)的补充教材。本书共分三大部分。第一部分按照电大讲课顺序(与课本第一版顺序稍有差别)进一步阐述了每一章的重点、难点；第二部分介绍了上机调试运行 Pascal 程序的方法，这一部分内容课本中没有涉及到；第三部分给出了课本中全部习题参考答案(注意，习题内容和顺序与课本第一版稍有不同)。

本书第一部分由卢雄远同志编写，第二、三部分由蒋代梅同志编写。由于时间仓促，难免有错误或不足之处，欢迎广大读者批评指正。

作　者

目　录

第一部分　Pascal 语言学习指南

绪论　Pascal——一种全新的程序设计语言

一、学习 Pascal 语言的意义

对于已经掌握某些传统的高级程序设计语言，如：FORTRAN, ALGOL, BASIC 或者 COBOL 等等的读者来说，学习 Pascal 语言的意义，决不仅仅是单纯地多掌握一种程序设计语言工具。他将会发现，他接触到一种全新的、现代的程序设计思想，它从根本上否定了传统程序设计思想的某些基本观念。对于他原来所熟悉的经验和方法而言，无异是一场"地震"，原有的经验都不得不以新的观点重新检验。这种新的观念就是所谓的"结构程序设计"观念。而对于初次学习程序设计语言的读者来说，他应当感到庆幸，他刚入门就能直接学习新的、现代的程序设计方法，避免了重复历史走过的弯路。

对于所有的读者来说，学习 Pascal 语言，都是学习"结构程序设计方法"的一个自然的途径。

二、什么是"结构程序设计"

"结构程序设计"概念是随着计算机软件的发展而被提出的，它的出现是程序设计方法论的一场革命。

在计算机软件发展的早期阶段，计算机程序的质量标准主要是程序占用的存储空间和运行的速度　程序的正确性被视为是当然的。六十年代，随着大型软件的不断发展，程序的正确性，软件系统的可靠性、越来越难以保证，一些大型软件的研制工作，由于内部的错误和缺陷往往一再拖延，难以按时完成。这就是当时所谓的"软件危机"。因此，正确性、可靠性已变成评价程序的第一位的标准。此外，对程序易修改、易扩充、易移植性能的要求也日益增长。而程序占用空间及运行速度的问题，由于计算机硬件技术的高速发展，相对而言已比较容易解决。六十年代末，提出了"软件工程"的概念，它涉及程序设计方法论，工具，组织和规范各方面的问题，核心就是荷兰计算机科学家，图林奖获得者 E. W. Dijkstra 提出的"结构程序设计"思想。

"结构程序设计"就是抛弃传统的手工艺式精雕细琢的方法而寻求以工业化方式可靠地生产软件的方法论。简言之，就是为了使程序具有合理结构，以便保证和验证其正确性而规定的一套进行程序设计方法。用结构程序设计的方法设计出来的程序称为结构化程序，而反映了结构程序设计的要求和限制，便于用来书写结构化程序的程序设计语言就称为结构程序设计语言。用这种语言书写的程序易于保证正确性。以上的叙述，与其说是一种定义，不如说更象是一种要求。现将它的具体含义归结为以下几点：

1. 基于自顶向下、逐步求精的设计方法

从实际问题出发，首先用对问题而言比较自然的、某种适当抽象的算法描述和数据结构描述来表达对问题的解法。然后对其成份进一步分析、分解，则可对每个成分得到下一层抽象的算法和数据描述，……，如此逐步分解、精细化，直至这种描述能为所使用的计算机接受为止。

举例说明，一个矩阵和向量相乘的算法可以这样来逐步实现。

首先，问题叙述本身就是一种自然语言的描述。“矩阵”、“向量”是数据结构描述，“相乘”是算法描述。这里所用语言实际上已不完全是自然语言，而是专门化的数学语言，是高度抽象了的。形式化描述为：

(1) $y:=A\cdot X$

其中，A 为 $m\times n$ 阶矩阵，X 为 n 维向量，y 为 m 维向量，“:=”表示右端向量赋值给左端结果向量。“·”表示矩阵和向量相乘。进一步分解：

(2) $$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} := \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

换一种数学描述，可表示为：

$$y_i := \sum_{j=1}^{n} a_{ij} \cdot x_j \qquad (i=1,2\cdots m)$$

这里的数据结构描述 y_i, x_j, a_{ij} 都已换成带下标的标量，算法描述：=，Σ，·都是用于标量的了。至此，除了算法描述“Σ”还需要进一步具体化以外，其他描述都可容易地翻译为计算机上配备的 Pascal 编译程序能够识别的描述。下面进一步分解，改写为：

```
(3) for i:=1 to m do      {逐个处理 yi}
      begin
        y[i]:=0;          {先给 yi 赋初值 0}
        for j:=1 to n do  {aij 与 xj 逐对相乘后向 yi 累加}
          y[i]:=y[i]+a[i,j]*x[j]
      end
```

这里，标量 x_j, y_i, a_{ij} 已改用 Pascal 语言中相应的数组元素来描述，算法描述也改为相应的 Pascal 语句和算符。至此程序已能为计算机上的 Pascal 编译程序接受。当然，要实际运行，还要按照 Pascal 语言的规定，把必要的说明部分及输入，输出部分补齐。

以上所述自顶向下，逐步求精方法的核心是对要处理的问题分层考虑。首先从大处着眼，选择适当的数据和算法描述表达问题的解法，从宏观上考虑该解法是否正确可行，而先不去考虑细节。经分析认为可行，才进一步把这种粗线条的算法描述和相应的数据描述分别分解为下一层的、具体化一些的描述来考虑，这样一层层地逐步求精，每一步正确的抽象保证了最终算法的正确。有时分析到某一层细节和总的考虑有冲突，需要修改上层的解法，那就要从所涉及的层次重新向下求精。

可以说，这种从大处着眼，逐步求精的步骤并不新奇，通常人们分析处理问题时就是这样作的。从这个意义上说，结构程序设计方法只不过是“回到常识”。正因为这种方法符合人们的思维规律，所以使人感到条理清晰、自然，从而大大减少了设计中的错误。

2. 在程序结构方面的特点

(1) 限制"转"语句。无节制地使用"转"语句导致程序结构错综复杂，难以阅读、验证和修改，容易造成错误。在结构程序设计中完全可以只使用几种基本控制结构：顺序、分支、循环来构造任何程序。这些基本控制结构都是单入口、单出口的，其中包含了几种标准形式的"转"。实质上就是用几种标准化的"转"取代任意的转语句来构造程序。这样就使程序形式上的静态结构与运行时的动态结构能较好地对应，程序结构清晰，易读、易改、易于进行正确性验证。对于结构程序设计来说，"转"语句不是必要的。

(2) 作为逐步求精的结果，程序结构是分层的，依赖关系基本上是"半序的"。这不仅使程序清晰，而且使因修改部分程序而产生的影响局部化，从而使程序具有易修改的特点。

(3) 模块化结构，每个模块的数据结构及在其上施行的算法自成一体，相对独立，模块之间的联系较简单，因而便于分别编写、调试和相互组合、联接。

3. 程序员组织

与程序结构相对应，也是分层的。由水平较高、经验丰富的软件人员任最高层的主程序员，负责总体和与下层各程序员之间的协调。

按照结构程序设计方法生产的程序，结构清晰，逻辑自然，易读易改，便于进行程序正确性验证。现代软件工程组织生产的软件产品，出错率已可达每 10000 行程序含错不多于 1 个。

三、结构程序设计语言——Pascal

1970 年由瑞士 N. Wirth 教授提出的 Pascal 程序设计语言是第一个实现并得到普遍承认的结构程序设计语言。较好地反映了结构程序设计思想的特点。具有丰富的数据类型和控制结构，简明，易读，易于实现和移植，特别适用于教学、算法描述和书写顺序型系统软件。Pasca 还是一种自编译语言，可以采用特殊的自展方法实现，较易实现，可靠性高。

Pascal 语言对其后的程序设计语言的发展有着深远的影响。尽管从今天的观点看来，Pascal 也有一些不足之处，例如没有引入模块概念，缺少进行并发程序设计的手段，文件处理功能较差等等，这是历史的局限所致。可以说，Pascal 语言是程序设计语言发展史上一个重要的里程碑，至今仍是国际上应用最广泛的算法语言之一。

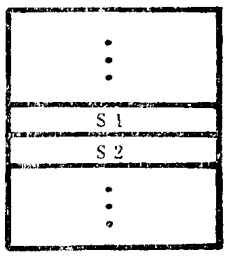
四、结构化框图

在程序设计实践中，人们常用由方框和带箭头的直线构成的框图来表示程序结构，这是一种直观、有效的工具。但它是不符合结构程序设计原则的。因为其中可以随意指向的、带箭头的线，容易诱使人们无节制地使用转语句；这种方法还容易使人在整体构想完成之前先写出大量程序细节，尔后又因不忍割爱而以整体迁就细节。这些显然都是和结构程序设计原则直接冲突的。因此在课本中没有采用这种传统的框图。

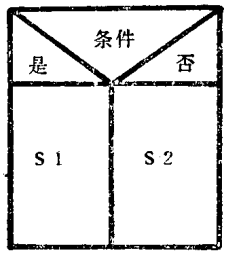
为了保留框图这一方便、直观的工具，有人提出了结构化的框图方法。这种框图兼有直观和结构化的特点，难以用来表示非结构化的设想。目前这种框图有许多种形式，建议习惯使用框图的读者在学习 Pascal 语言的过程中采用下面介绍的框图形式。这种框图是由一些矩形框相联或嵌套组成。一个矩形框表示一段程序或一个语句。图 1·1(a)表示顺序结构，对应于 Pascal 语句…s1; s2…。图 1.1(b) 表示二分支结构，对应于 Pascal 语句 **if 条件 then s1 else s2**。图 1.1(c)表示多分支结构，对应于 Pascal 语句

```
case 情形 of
v 1: s 1;
v 2: s 2;
  ...
vn: sn
    end
```

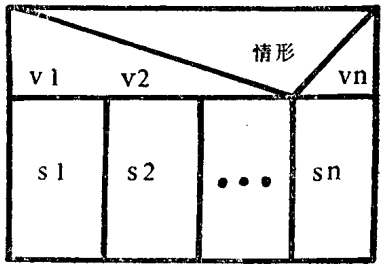
图 1.1(d)是先判断条件循环，对应于 Pascal 中的 **while** 和 **for** 语句。图 1.1(e)是后判断条件循环，对应于 Pascal 中的 **repeat** 语句。图 1.1(f)中的端圆框表示调用过程(或引用函数)，相当于 Pascal 中的过程语句(或函数引用)，而图 1.1(g)表示一个过程(或函数)的具体实现。



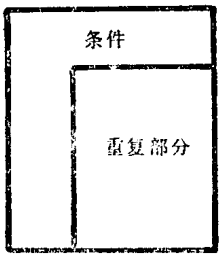
(a) 顺序结构



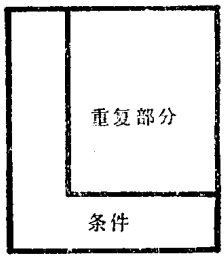
(b) 二分支结构



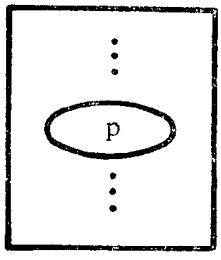
(c) 多分支结构



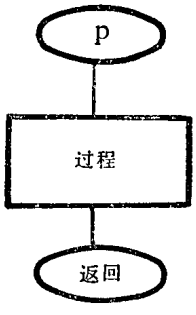
(d) 先判断条件循环



(e) 后判断条件循环



(f) 过程调用



(g) 过程(函数)

图 1.1

第一章　Pascal 语言概述

本章主要从宏观上介绍 Pascal 语言的基本组成部分和特点，并具体介绍了描述算法和数据的基本“构件”。

一、Pascal 程序的结构和语法图

课本在这一章首先通过一个实例来介绍 Pascal 程序的结构，接着使用语法图这一工具给出其严格的形式定义。

书中对实例的叙述过程，就是应用“自顶向下，逐步求精”的结构程序设计方法的一个简单范例。在以后各章节的实例介绍中也都着意反映了这种结构化的设计方法，读者应注意学习、体验并逐步掌握这一方法。

从例中可以看出，Pascal 程序的书写格式比较自由，没有 Fortran 语言那样严格的规定。读者可能已经注意到，程序格式是类似阶梯形的。建议读者仿效例中这种格式，即按照程序的层次，同层的定义、说明、语句从同一列开始，互相对齐，下层的逐级向右退进若干列开始，并适当以文字注释。这样的程序清晰易读。

语法图是课本中用来定义 Pascal 程序语法成份的形式结构的工具。它严格、简明而直观。语法图中左端的词是被定义的 Pascal 程序语法成分，一般称为非终极符号，它不得以该形式直接出现在正规的 Pascal 程序中。图中用带箭头的线和框表示被定义的成分的形式上的结构组成。其中方框中的词也是非终极符号，是需要进一步定义的语法成分，端圆框和圆框内的称为终极符号，它是直接出现在 Pascal 程序中的合法符号，不必再定义。读者应该掌握语法图这一方便的工具。

语法规则还可以用其他方式表示，课本后的附录中就采用了 Backus—Naur 范式来描述 Pascal 语言的语法。

根据语法图，Pascal 程序形式上由“程序首部”后接“程序体”构成。（国家标准中“程序体”改称为“分程序”）

“程序首部”由终极符号 program 开头，它标志“程序”由此开始；后接的“标识符”是用户自己定义的程序的名字，在实际程序中该处出现的必须是按照“标识符”的语法定义组成的、具体的名字，如例中的 Contchars；其后接着的、用括号括起来的“标识符”，是程序参数，一般是实际的文件变量名，程序通过它访问外部文件（即与外部文件之间进行输入或输出数据传递）。程序参数多于一个时，其间用“,”号分隔。最常用的程序参数是 input 和 output，表示程序要访问标准输入、输出文件变量。如果程序不访问外部文件，则文件参数和括号这部分可以省略。

“程序体”则由“标号说明部分”、“常量定义部分”、“类型定义部分”、“变量说明部分”、“过程与函数说明部分”和“语句部分”依次连接构成。注意：这些语法成分的这种形式上的顺序和结构是必须严格遵循，不能随心所欲的。其中的任一成份都可以空缺，但不能增加或改变顺序。“语句部分”是程序的执行部分，其他各“部分”则是程序的说明部分。

上述语法成分中的非终极符号可进一步定义，直至全部由终极符号组成为止。这些定义的全体就构成了 Pascal 语言的形式语法规则。任何 Pascal 程序必须在形式上满足这些规则才是合法的、能够被 Pascal 编译程序接受的。

二、Pascal 语言的词法记号和分隔符

任何程序设计语言在形式上都是由最基本的素材——字符组成的。Pascal 语言中使用的字符可分三类：字母(26 个)，数字(10 个)，其他字符(20 个)。大多数计算机的字符集都包含了这三类字符。

任一程序设计语言都有它自己的一套有特定意义的符号，用以构造程序。Pascal 语言中的这种符号称为“特定符号”，它们是由上述的“字母”和“其他字符”构造而成的，又可分为字符号和非字特定符号。这是 Pascal 语言最基本的语法单元，称为词法记号。特定符号、字母和数字的总和就是终极符号全体。

还有几种词法记号，它们是：

标识符　是由字母开头的字母、数字序列。用来标识常量、变量、类型、过程、函数、程序和记录中的域。在词法上，标识符长度不限(在具体机器上实现时也许有限制)，不得与字符号拼法相同。字符号是保留给系统专用的保留字。

指示字　即 forward，用于过程或函数说明中。是由系统定义的标识符，不是保留字。

标号　是数字序列。用于标志程序点，与转语句配合使用。标号显示的整数值限于从 0 到 9999。

数　采用十进制。注意，Pascal 的数的表示形式中，小数点“.”若出现，其前后都必须有数字；实数表示中的“E”若出现，其前必须有数字，其后必须接整数。数用于表示相应类型的变量的值。对于在具体计算机上实现的 Pascal 语言来说，数值的范围都有一定的限制。

字符串　由一对撇号“'”括起来的字符序列构成，用于表示字符型或串类型变量的值。注意：这个字符序列中允许出现所用的计算机字符集中的任意字符，而不限于 Pascal 规定的三类字符。为了能在字符串中表示撇号而不引起二义性，规定字符串中的撇号用两个连续的撇号表示。数和字符串统称为常量。

任何由标识符、字符号、标号和无正负号数组成的两个在程序中相邻的记号之间必须至少有一个分隔符，以免产生二义性。而分隔符除了不得在它们之中每一个以及非字特定符号的内部出现之外，可以出现在程序中其他任何地方。分隔符包括空格符、行分隔符(换行符)和注解。它们除用作分隔记号外，没有任何语法意义。

注解由一对花括号和它括起来的、其中不包含右花括号的任意字符和行分隔符的序列构成。可用来对程序进行解释性说明。

除非是出现在字符串中，Pascal 语言不区分大、小写字母。

三、Pascal 程序中的数据结构和算法

任何程序实质上就是对特定的数据结构和表示方式施行的抽象算法的具体表达。

1. 数据类型

传统的程序设计语言提供给用户的基本上是几种固定的数据结构，用类型标识符来标识。用户只能使用这些由系统定义的类型标识符来说明自己使用的变量，至多能对数组变量的类

型结构自行作出某些有限的选择。

在这方面，能向用户提供自行定义各种数据类型的灵活手段，从而使用户能够使用丰富多样的数据类型，正是 Pascal 语言的一大特色。用户可以利用 Pascal 语言特有的“类型定义”定义自己需要的数据类型，并自选适当的类型标识符来标识它。用户定义的类型标识符可以象系统给出的类型标识符一样用来在“变量说明”中定义变量。这使 Pascal 具有很强的描述各种数据结构的能力。

Pascal 语言中的数据类型可分为简单类型，构造类型和指针类型三类。其中前两类中又包括了如下所示的几类：

- 简单类型
  - 实数类型
  - 顺序类型
    - 整数类型
    - 布尔类型
    - 字符类型
    - 枚举类型
    - 子域类型
- 构造类型
  - 数组类型
  - 记录类型
  - 集合类型
  - 文件类型

任何构造都是由基本的“构件”单元组合而成的。Pascal 数据类型的基本“构件”就是由系统给出的，统称为“需求类型”的实数类型、整数类型、布尔类型和字符类型。它们由系统提供的需求类型标识符表示。本章详细介绍了这些类型。用户可通过“类型定义”的手段，用这些基本“构件”构造出上述的其他类型，还可组合出更为复杂的类型。值得特别指出的是：枚举类型是一种特殊的，由用户指定的标识符集合构成的顺序类型。

2. 变量

数据的一种形式是常量，常量的类型由其形式即可确定。程序中的数据更多的是表现为变量。

Pascal 中的变量称为“变量存取”。使用前一般都要由用户进行类型说明。其中一部分是在变量说明部分或过程(函数)说明中用值参数指明、变量参数指明直接说明，这种变量称为整体变量。说明时可象别的语言那样用类型标识符(系统定义的或用户定义的)说明，有时也可用新类型(既非需求类型亦未在类型定义部分定义过的类型)来说明。另一部分则是随着某整体变量的说明而间接地被说明的，其中称为成分变量的有两种：数组的成分——下标变量，它随着数组变量的说明而被说明。记录的成分——域命名符同样随着记录变量的说明而被说明。缓冲变量则随着它所属的文件变量的说明而被说明，而且它是用该文件变量名后跟一个“↑”来表示的。这些都不需另行说明。还有一种称为标识变量，它是在程序语句部分，在程序运行时由需求过程 new 作用于某个指针变量而产生的，它的类型和表示形式都是依赖于该指针变量的(详见以后章节)。

3. 算法“构件”

与数据结构相似，Pascal 程序中的算法也有它的基本“构件”，也是必须由系统提供的。它们是在一些数据结构上定义的运算(如四则运算、逻辑运算，关系运算、集合运算等)赋值以及一些系统定义的需求过程和需求函数。这些算法单元都是和它们处理的数据结构紧密相联，各自有具体的使用范围和规则的。在课本中和相应的数据结构一起介绍。

用户使用 Pascal 语言提供的丰富多样的各种语句，对上述的基本算法“构件”进行有机地组合，来表达自己设计的算法，这就是程序。Pascal 语言的特点决定它适于表达结构化的程序。

几点注意：

(1) Pascal 的表达式中，运算符优先级顺序稍有特殊：它将算术运算符、逻辑运算符、集合运算符统一考虑：逻辑非优先级最高，以下依次为乘法运算符、加法运算符。关系运算符最低。同级运算按书写顺序从左向右计算。多层括号，先计算最内层。Pascal 中未定义乘幂运算符。

(2) 参加运算的对象原则上应有相同类型，但也有例外，如在算术运算中允许混合运算，整数类型值可以赋值给实数类型变量，读语句可向实数类型变量中读入整数等等，课本上在具体介绍中都有详尽的叙述，并在第四章§3中，对各种类型的数据对象之间的这种相容关系的规则作了概括。

(3) 关系表达式只能由一个关系运算符连接两个运算对象构成。数学上合法的关系式 A≤B<C 在 Pascal 程序中只能表示为(A<=B)AND(B<C)。类似地，赋值号也只能连接两个对象，形式为 A:=B:=C 的式子在 Pascal 程序中是非法的。

除了需求过程(函数)和需求类型之外，系统还定义了一些常量、变量和标识符。其中一部分是必须的，用户无法自行定义的，还有一些是为方便用户而定义的。用户可直接使用。它们都以标识符的形式出现，但不是保留字。建议用户不要定义和它们形式相同的标识符，那样有害无益。这些标识符计有：

常量 3 个：maxint, false, true

类型标识符 5 个：real, integer, char, boolean, text

正文文卷变量 2 个：input, output

指示字标识符：forward

需求过程 13 个，函数 17 个(略)。

四、注意

(1) Pascal 语言提供了一些便利，以增加程序的易读性：不限制标识符的长度，因而可选用有一定含义的标识符来表示变量、常量或过程(函数)；可在程序中插入必要的说明作为注解，读者应充分利用它。

(2) Pascal 比较严格地遵循先定义后使用的原则，对于标识符和标号都这样要求，只有极个别的例外。

(3) 语言课是实用课程，只有通过反复实践才能真正理解和掌握这一语言工具。因此争取尽早上机，多做练习是明智的。

五、习题

1. 基本习题，第一章全部习题。

2. 思考题

设 R 为整型变量，ch 为字符型变量，若在程序中出现语句

readln(R,ch); writeln(R,ch);

则该程序运行时，能否从终端正确读入任意的整数和字符并输出它们？为什么？如何解决？试用整数 3 和字符“5”试验。

第二章　控制语句

课本第一章中介绍了一些程序的例子，这些程序的执行部分都是顺序结构的。即程序运行时，其执行部分的语句都是按照书写的顺序依次执行的。其中接触了赋值语句和过程语句中的某些需求过程。仅有以上这些初步的工具是无法写出有实用价值的程序的。本章介绍的几种控制语句是 Pascal 语言中构造各种算法的重要工具。控制语句包括条件语句、重复性语句和转语句。下面仅介绍重点和应注意问题。

一、条件语句

条件语句用来表示分支结构，包括 **if** 语句和 **case** 语句两种。

1. **if** 语句——二分支情况

(1) **then** 和 **else** 后面皆可为任何语句，因而可嵌套使用条件语句，构成复杂的分支。

(2) 按照"**else** 与前面最近的、尚未配对的 **then** 配对"的原则，简单可靠地杜绝了二义性问题。

2. **case** 语句——多分支情况

多分支情况也可以用嵌套的 **if** 语句来表示。但 pascal 语言提供的 **case** 语句使程序更为清晰易读。**case** 语句的情况表元素中的语句可以是任何一个语句，因而也可以嵌套使用条件语句。

注意:

(1) 情况下标(即 **case** 语句中的 **case** 后面出现的表达式)的类型只能是顺序类型。

(2) 全部情况常量表必须包括情况下标所有可能的取值，否则执行时可能出现错误。这点有时不太方便，但可通过和 **if** 语句组合使用，来避免情况下标取情况常量表之外的值。有的非标准的 pascal 语言在 **case** 语句中扩充了 **else** 子句，这是更完善的解决办法。

二、重复性语句

为使用方便及程序结构清晰，提供了三种形式的重复性语句。它们还可以嵌套使用，构成多重循环结构。

1. **repeat** 语句和 **while** 语句

在 **repeat** 语句中，从 **repeat** 到 **until** 之间的语句序列为重复部分，至少执行一次。判断条件满足则终止重复。

在 **while** 语句中 **do** 后的语句为重复部分，可能一次也不执行。判断条件不满足时中止重复。

注意: (1) 这两种语句中的判断条件一般和重复部分有关，以使得循环能够中止。(2)判断条件中若使用了实型数据，则要注意到计算误差的影响，以保证其正确的逻辑功能。

2. **for** 语句

for 语句是程序中较常见的，适用于循环次数预知，步长为 1 (或者说控制变量依次取紧接的顺序类型值)的情形。课本上这部分内容很详尽，其中特别要注意的是:

(1) 控制变量必须是顺序类型的，而且必须在最紧包含该 **for** 语句的程序体的变量说明中被说明。

(2) 在 **for** 语句的循环体内，不允许直接或间接对其控制变量进行赋值。控制变量的初值和终值在进入循环体之前，由对应的初值和终值表达式确定，在循环体执行期间也不得改变。

(3) **for** 语句正常执行完毕后，其控制变量的值无定义，不得引用。若是利用转语句从循环体内转出(非正常结束)，则其值可以引用。

以上所述的分支和重复结构中出现的每个语句，指的都是单个语句。如果要在该处实现的功能必须用多个语句才能表达，就必须用由 **begin** 和 **end** 组成的"语句括号"将这些语句构成的序列括起来，相邻语句之间用";"号分隔，组成一个"复合语句"。

三、转语句

任何语句前都可以附加上一个标号，中间用冒号":"连接。该标号必须在包含它的程序体的标号说明部分先行说明。

转语句和标号配合使用，表示控制下一步应转去执行附有该标号的语句。对转语句的使用有下面的限制:

不允许用转语句从构造语句(包括前面介绍的复合语句、条件语句、重复性语句和将在第八章介绍的 **with** 语句) 外部转入其内部；也不允许用转语句从过程说明或函数说明外部转入其内部。

按照结构程序设计的观点，不赞成使用转语句，特别是任意的转语句。实际上，在 Pascal 中，转语句的引入不是必须的。使用前面介绍的条件语句、重复性语句等控制结构来组合基本的算法"构件"就能够编制出任何程序。这些单入口，单出口的基本控制结构是按照结构程序设计的原则设置的，能够避免使用任意的转语句带来的弊病。然而，"不使用转语句"的原则不是绝对的，在某些特殊情况下，也许使用有限制的转语句，程序结构更为清晰。总之，"最终原则"就是编制出结构化的程序。

有时，使用转语句只是为了把控制转向程序中某个程序点，而不需要附加任何操作。例如，要把控制从某个复合语句内部某处转向其中语句序列的末尾，这时就要使用一个带有标号的"空语句"。空语句不包含任何符号，也不表示任何动作。空语句不一定和标号一起出现。程序中任何一个应该出现语句的位置，如果空着，即可视为该处存在着一个空语句。例如:

```
if x<0 then else x:=0
```

或者

```
begin x:=1 ; y:=3;end
```

其中都存在一个空语句。

四、习题

1. 基本习题 5,6,7,8,9,12,16,19,20。

2. 思考题

课本 38 页描述 **for** 语句的执行过程，用一个等价的复合语句表示。形式比较繁琐，似乎可以简化，比如，用一个赋值语句给循环变量赋初值，后接一个 **while** 语句来表示。课本上的这种形式有什么含义？(提示: 反映了 **for** 语句的某种限制)。

第三章　过程与函数

在第一章里我们学习了由系统定义的一些需求过程和需求函数。引用一个需求过程，就是对某些数据进行某些特定的操作，而引用一个需求函数则还带回一个值来参加表达式的运算。当我们要设计一个程序来处理某个任务时，通常可以把该任务分成若干子任务，若某个子任务比较大，则可进一步划分。每个子任务就是对某些数据完成某些特定的操作。用户可用 Pascal 语言中提供的过程说明来定义一个过程，它具有处理这项子任务的功能。通过过程语句引用该过程即执行该子任务。如果某个子任务的最终目的是求出一个值，则可将它定义为一个函数。这样得到的程序结构清晰、易读、易修改。倘若某项子任务是要多次执行的，那么定义为过程(或函数)就更有价值。 Pascal 中的过程说明和函数说明也是用结构化的方法来构造算法的一个重要工具，有利于程序结构的模块化。遗憾的是由于历史的局限，Pascal 还不具备现代的模块手段。例如，过程和函数没有分离编译的功能(有的非标准的 Pascal 已对此作了扩充)；由于可使用全程量，使过程(和函数)与外部的某些联系成为隐藏的、不清晰的，从而损害了它的独立性等等。模块概念是七十年代中期发展起来的，在当今程序设计中已成为一种强有力的手段。

由于函数和过程基本上是相似的，所以这里着重介绍过程，对函数则仅指出其特殊之处。此外也不重复课本对过程(和函数)说明及过程语句的形式语法的叙述，只说明两点：

过程说明的结构和程序的结构是非常相似的，只是过程首部与程序首部稍有不同。过程说明象是一个“子”程序。且可进一步嵌套定义。这种形式上的统一与结构程序设计方法是相呼应的。

程序是通过出现在语句部分的过程语句来引用过程的。引用时先将过程语句实在参数表中的每个实在参数，和其过程说明的形式参数表中对应的形式参数结合，即所谓虚实结合，然后控制转去执行该过程说明的语句部分，完成后，返回调用点。

下面介绍本章的两个中心内容：

一、形参与实参结合问题

过程(或函数)首部形式参数表中的形式参数和过程语句(或函数引用)实在参数表中的实在参数是按照它们在表中的位置一一对应地结合的，和标识符的形式没有关系。

在 Pascal 中，形式参数与实在参数结合的方式有两种：

1. 在执行过程语句时，相当于先将实在参数的值赋给对应形式参数，然后执行过程体。这种形式参数称为值参数。值参数对应的实参可以是表达式，且要求与过程值参数指明的类型赋值相容。

2. 变量参数是这样的形式参数：在执行过程语句而进入对应的过程体时，体中对该形式参数的访问实际上就是对它对应实参的访问，这样就可通过这个实参带回结果。因此变量参数对应的实参只能是同类型的变量存取，不得为表达式或常量。特别要注意的是：记录变体部分的标志域和紧缩类型变量的成分不能作为变量参数的实在参数。

注意:

(1) 前面所说两种形参类型指明都只能用类型标识符表示。理由见第四章关于类型相同的定义。

(2) 形参与实参结合关系都是在执行过程体之前确定的，且在过程体的整个活动期间都不再改变。故下标变量作为变量参数的实参，与形参结合的始终是进入过程体之前时的下标值所确定的那个下标变量。同样，由某指针型变量表示的标识变量(动态变量) 作为变量形参的实参时，也始终是开始时该指针所指的标识变量与对应变量形参结合。

二、标识符的作用域

由于标号在作用域问题上和标识符面临的问题是相同的，所以，下面若无特别声明，对标志符所作的讨论可视为同样适用于标号。

标识符必须先定义，后使用。由此引起两个自然的问题：在何处定义？以该定义确定的含义的使用范围是什么？这就是标识符的作用域问题。这个范围就是该标识符的作用域。Pascal中关于标识符作用域的规则，简明而有效地解决了标识符的定义和使用中产生的种种问题。

标识符(和标号)在程序中的出现，可分为定义性出现和使用性出现两种情况。定义性出现指的是：标号说明中的标号；常量定义和类型定义中"="号左端；变量说明中":"号左端；变量说明和类型定义中枚举类型的值(标识符)；过程和函数说明首部中的过程标识符或函数标识符以及形式参数标识符。程序中标识符定义性出现的位置，称为该标识符的定义点。标识符在程序中的其他出现都是使用性出现。

域标识符的作用域问题在第八章讨论。

Pascal 规定：标识符的作用域从它的定义点起直到最紧包含该定义点(即包含定义点且是最内层) 的程序体区域的末尾。如果在这个区域内的过程说明或函数说明中又定义了同名的标识符，则该标识符在这个内层的新的作用域内就有了新的含义，即服从内层的定义。而在外层原来作用域扣除内层新作用域所余的区域中，仍保持原来的意义。在此情况下，该标识符在外层定义下的作用域就是这个"剩余区域"。

前面提到的"程序体区域"(或称分程序区域)包括程序体和与它相连的参数表两部分。程序体区域的划分反映了程序中的层次概念。参数表和程序体构成区域的"内部"，对应的过程标识符或函数标识符被看作是"外层"的。过程体和函数体都是程序体。

特别指出：类型标识符的作用域不包括该类型定义中"="号右端部分（称为类型表记符)。此处该类型标识符是无定义的，不得引用，除非它紧跟在"↑"后出现。(见指针类型)。

在同一程序体中，除过程或函数说明的形参以及相应体中的标识符外，任一标识符只能被定义一次。也就是说，在同层程序体区域中，不能把多个对象定义为同名的标识符，或称不能重复定义，但在不同层可以定义同名的标识符。作用域冲突时则按上述原则解决。

标识符的使用性出现只有在它的作用域内才是合法的。

例：编程序，使能读入 3 个数，求和，从小到大排序，并记录交换次数。(程序见第 13 页)。

程序中用虚线画出了各层程序体区域。最外层程序体区域定义的标识符 w,a,b,c,sum, sort 的作用域从其定义点起直到程序结尾。但由于其内层定义了一些同名标识符，所以标识符 a,b,c 的作用域不包括与过程 sum 相关的程序体区域，它们在那里有新的、局部于那个区域的含义，对于外层含义的 a 还要扣除与过程 swap 相关的程序体中从 a 的新定义点开始的作用域。标识符 w 则在 sort 过程的程序体中有新的作用域，也要扣除。标识符 x 在