高校计算机等级考试辅导丛书

（非计算机专业）

PASCAL 语言（二级）

罗 凡 杜燕平 编著

上海科学技术出版社

高校计算机等级考试辅导丛书

（非计算机专业）

PASCAL 语言（二级）

丛书主编　杭必政
丛书副主编　王春森　吴永明　乔沛荣
编　　著　罗　凡　杜燕平
审　　阅　王春森　杭必政

上海科学技术出版社

内 容 提 要

本书旨在帮助大学生参加PASCAL语言等级考试。主要内容包括PASCAL语言的知识要点、编程基本方法、常用算法、习题与答案及1994年考题等。知识要点着重于知识的归纳、整理和分析，编程基本方法与常用算法着重于提高结构化程序设计能力和复习常用算法，习题与答案着重于检查对基本内容、方法的掌握情况。本书重点突出，编写紧凑。

本书可作为高等学校计算机等级考试的辅导材料，也可作为学习PASCAL语言的参考书。

高校计算机等级考试辅导丛书
（非计算机专业）
PASCAL 语言（二 级）
丛书主编 杭必政
丛书副主编 王春森 吴永明 乔沛荣
编 著 罗 凡 杜燕平
审 阅 王春森 杭必政
上海科学技术出版社出版、发行
（上海瑞金二路450号）
新华书店上海发行所经销 常熟市印刷六厂印刷
开本 787×1092 1/16 印张 7.25 字数 167,000
1995年9月第1版 1995年9月第1次印刷
印数 1-4,000
ISBN 7-5323-3975-0/TP·57
定价：10.80元

序

计算机的发展推动了社会经济、文化和军事等方面的飞跃发展，而经济的发展又对计算机的发展提出了更高的要求。我国许多地方早就把计算机列为重点发展的高新技术支柱产业。高校中95%以上的非计算机专业大学生是我国未来计算机应用的最重要的生力军。非计算机专业大学生计算机基础教育的状况直接影响着我国今后的计算机应用水准。计算机应用知识和应用能力已成为现代大学生知识结构中的重要组成部分。目前各高校学生争选计算机课程，渴求增加计算机知识和增强计算机应用能力，他们已领悟到在信息时代里，没有计算机应用知识等于是新时代的文盲。

四年前上海市高教局率先决定建立上海高校非计算机专业大学生计算机应用知识和应用能力等级考试制度，顺应了计算机发展的浪潮。在这以后，北京、陕西、四川、江苏和浙江等全国多数省市已先后开展或准备开展非计算机专业大学生的这种等级考试。从上海和全国有关省市已经进行的这种等级考试来看，等级考试在目前确实促进了课程体制的改革，实验设备的添置和更新，促进了教材和教学手段的改革。

为了帮助非计算机专业大学生进一步提高计算机编程、设计等应用能力，也为了具体指导大学生如何更好地应试等级考试，我们从上海高校中请来了长期在第一线从事计算机基础教学，且有较丰富教学经验的教师编写了这套丛书。

编者以上海等地区的《考纲》为依据，着重从知识要点、试题分析、习题和解答等方面来组织编写这套《高校计算机等级考试辅导丛书》。我们相信该套丛书对广大学生会有所裨益。丛书包括：《计算机基础》(一级)、《C语言》(二级)、《FORTRAN语言》(二级)、《PASCAL语言》(二级)、《TRUE BASIC程序设计》(二级)、《硬件》(三级)和《软件》(三级)等七本书。该丛书由杭必政任主编，王春森、吴永明、乔沛荣任副主编。史济民、高传善、章鲁、杭必政和王春森分别审阅了有关书稿。

上海科学技术出版社大力支持本套丛书的出版，上海市有关高校领导在本书的编写过程中也给予了积极关心和支持，在此一并表示感谢。由于时间紧迫，书中定有不足和错误之处，敬请读者批评和指正！

主编

1995年4月

前　言

随着计算机的日益普及，对其编程、设计等应用能力的要求也愈来愈高。本书提供给有PASCAL语言初步知识的读者，作复习和提高用参考书。

复习PASCAL语言程序设计，首先要注意掌握语法体系和语法细节，其次要掌握结构化程序设计的正确步骤和程序设计的一些常用概念和技巧，最后，应该复习算法和数据结构。准备参加考试的读者要了解以往的考试内容，分析考题类型。

本书以此为线索展开，全书内容包括三个部分：第一部分为PASCAL语言知识要点，共有两章。内容有基本语法和注意事项：包括数据类型、运算、操作、语句、过程与函数等。第二部分为PASCAL语言的编程基本方法与常用算法。也有两章，介绍了编程的基本方法和程序设计技巧及常用算法。第三部分为习题与答案，提供读者一些典型的练习题及参考答案。

限于作者水平有限，书中难免有错误和不足之处，敬请读者指正。

编者 1995.5.

目　　录

第一章　数据类型

第二章　运算、操作、语句与函数、过程

第三章　编程基本方法

第四章　常用算法

第五章　习题与答案

第一章　数 据 类 型

第一节　基本数据类型

一、常量及其定义

常量是程序运行过程中其值不允许改变的量。

1. 常量定义一般形式

CONST　常量标识符＝常量；

2. 使用中常见错误

(1) 整数越界

整数值域应为　－32768≤N≤32767

(2) 指数表示错误

要注意比例因子 E 之前之后必须有常数，指数必须是整数，E 前后不允许有空格。例如，下列指数是错误的：

.E，3.2E　8，E11，－12.7E5.3

(3) 单引号出现在字符串内部

若单引号也是字符串一部分时，则应用双写。

二、变量及其说明

变量是在程序运行过程中其值可以改变的量。

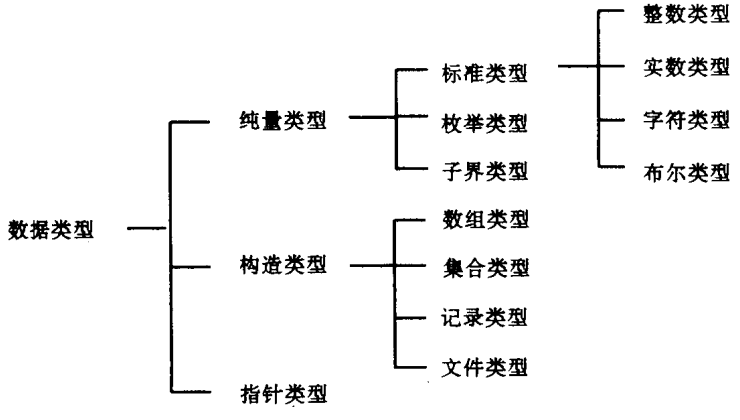
1. 变量说明形式

VAR　标识符：　类型；

如：VAR

```
i,j,k : integer;
x,y : real;
```

PASCAL 中的变量类型如下所示：



2. 标准类型

integer　　整型

real　　实型

char　　字符型

boolean　　布尔型

3. 易出错处

(1) 重复说明一个变量。

例：VAR a,b:integer;

b:real;

(2) 用保留字作变量名。

例：VAR CONST:integer;

(3) 使用未定义过的变量。

第二节　类型定义和纯量类型

一、类型定义

一般形式

```
TYPE
  t1=ty1;
  t2=ty2;
   ...
  tn=tyn;
```

其中　ti(i=1,2,3,...,n)是一个标识符，称为类型名；

tyi 是一个类型构造或是前面已定义过的类型名。

二、枚举类型

枚举类型是程序员用枚举的方法指定一个集合值，每一个值以标识符来代表，其中的标识符是该类型的常量。

1. 定义

```
TYPE
  weekdays=( Sun,Mon,Tue,Wed,Thu,Fri,Sat);
  color=(red,orange,yellow,green,blue,while,black);
  sex=(male,female);
  operators=(plus,minus,times,divide);
```

类型定义后就可以用此类型说明变量：

```
VAR
  today,yesterday,tomorrow : weekdays;
   ink,paint : color;
```

```
student : sex;
```

枚举类型的变量可以用作循环的控制变量和 CASE 语句的选择器表达式。

例：
```
FOR tomorrow:=Sun TO Sat DO 语句;
 CASE student OF
  male : 语句 1
 female : 语句 2
 END;
```

2. 易出错处

(1) 枚举类型变量的运算

枚举类型的变量只能进行赋值和比较两种运算，不能进行算术和布尔运算；其中比较(关系)运算是根据枚举标识符在类型定义时的顺序进行比较的，第一个标识符的序数为0，以后依次递增，比较结果为布尔值。

例：
```
TYPE weekday=(mon,tue,wed,thu,fri,sat,sun);
 VAR date1,date2:weekday;
```

下述语句是合法的：
```
date1:=wed;
date2:=fri;
IF date1<date2 THEN <语句>;
WHILE date1=mon DO <语句>;
```

而语句 date1=date1+date2; 是非法的。

(2) 枚举类型变量的取值

枚举类型的值不能直接读入和写出。

例：
```
……
date:=tue;
write(date); 结果将是错误的。
……
```

三、子界类型

1. 类型定义

```
TYPE  标识符=常量 1..常量 2;
```

其中　常量 1 称为下界；

　　常量 2 称为上界。

上下界常量的类型要相同，它们的类型称为该子界类型的基类型。

例：
```
TYPE
 year=1988..2000;
```

基类型是整型。

定义后，在变量说明部分就可以说明这种类型的变量。

例：
```
VAR
 y1,y2:year;
```

也可将类型定义与变量说明合并。

```
VAR
  y1,y2:1988..2000
```

适用于子界类型的基类型的所有运算符都适用于该子界类型，使用过程中注意子界类型的变量不要越界。

2. 易出错处

(1)基类型的使用

基类型必须是顺序类型如整型，布尔型 ，字符型或枚举类型以及它们的子界，若是枚举类型必须在子界类型之前定义。

(2)取值范围

所定义的子界类型，其取值范围必须在它所属的基类型的取值范围之内。

如下例引用就是错误的：

```
TYPE
  lenmonth=28..31;
 VAR l:lenmonth;
.....
   l=35;
```

因 35 超出了子界范围，故此语句的引用是错误的。

第三节　构 造 类 型

一、数组

数组是具有固定数目的相同类型的元素按一定的顺序排列而成。

1. 数组类型的定义

```
TYPE
  标识符=ARRAY[下标类型] OF 成份类型
```

其中下标类型可以是顺序类型。

例:

```
TYPE
    srg=0..100;
    week=(sun,mon,tus,wen,sat,fri,sat);
    ary1=ARRAY[srg] OF integer;
    ary2=ARRAY[week] OF real;
    ary3=ARRAY[1975..1995] of char;
```

数组变量的说明：

```
VAR
  a : ary1;
 x,y: ary2;
 f : ary3;
```

一般，只允许通过数组变量名与相应下标访问数组变量中的一个元素。但当两个数组变量的类型同一时，允许以变量名整体地互相赋值。所谓类型同一是指使用同一个类型标识符或类型标识符不同但有形为 T=P 的类型定义。

例：TYPE

```
    arraytype=ARRAY[1..10] OF real;
  VAR
    a,b:arraytype;
  BEGIN
    FOR i:=1 to 10 do
      read(b[i]);
    a:=b;
```

2. 易出错处

(1) 数组定义部分

定义时，下标标识符中不允许出现变量。

```
  VAR
    a: ARRAY[0..N] OF integer;
```

其中，若 N 不是已定义的常量，则这样的说明是错误的。因为 PASCAL 语言的数组长度是不可变的。

(2) 下标类型选择错误

下标类型不允许是实型。

如下定义是错误的：

```
CONST n=15.6;
  VAR a:ARRAY[1..n] OF real;
```

二、集合

集合是相同类型对象的聚集。构成集合的每一个对象称为集合的元素。元素的类型称为集合的基类型。

1. 定义

```
TYPE
集合类型 = SET  OF  基类型;
```

其中基类型是顺序类型。

例：

```
TYPE
  lettersets=SET OF 'A'..'Z';
VAR
  ls1,ls2:lettersets;
```

2. 易出错处

(1)基类型选择错

基类型只允许是顺序类型。

如下定义是正确的：

```
TYPE
  lettersets=SET OF 'A'..'Z';
  coloursets=SET OF (red,yellow,blue);
  agesets=SET OF 0..120;
```

而如下定义就错了：

```
TYPE
  salarysets=SET OF 300.58..1500.98;
```

因实数不是顺序类型。

(2) 运算符选择错

用五种关系运算符=,<>,<=,>=和 IN 等测试集合成员关系。

集合运算有：

设 A,B 是两个同类型的集合变量

赋值：A:=[ ];

并：A+B 结果是 A 和 B 中的所有元素组成的集合；

交：A*B 结果是 A 和 B 中公共元素组成的集合；

差：A−B 结果是在 A 中但不在 B 中的元素组成的集合。

三、记录

1. 记录类型定义与变量说明

例：

```
TYPE
    date =RECORD
            year:integer;
            month:(Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,
                Sep,Oct,Nov,Dec);
            day:1..31
            END;
  VAR
            day:date;
```

2. 记录变量

引用形式：记录变量.域标识符

例：

```
day.year:=1960;
day.month:=Jul;
  day.day:=21;
```

如某个域是字符型数组，当通过 read 语句给域赋值，则应组织一个循环语句来实现。

例：

```
TYPE
   student=RECORD
            number :integer;
            name:PACKED ARRAY[1..20] OF char;
```

```
        score:real;
        END;
VAR stu:student;
可用  FOR  i:=1  TO  20  DO
        read(stu.name[i]);
```

注意必须赋满20个字符，其中空格也作字符处理。

3. 易出错处

(1) 记录变量的赋值

对一个记录赋值，要对每个域逐个进行，不能用 read 语句整体读入；同一类型的记录成分允许互相赋值。当一个记录的所有成分有值时，允许用记录变量名将此记录值整体赋给另一记录变量。

例：VAR

```
    day1,day:date;
```

则整体赋值结果等价于分别赋值

```
                    day1.year:=day.year;
day1:=day <=>       day1.month:=day.month;
                    day1.day:=day.day;
```

除能整体赋值外，记录变量不能作为任何运算符的对象进行运算。

(2) 记录变量用于函数和过程

记录变量可以作为过程或函数的参数，但不能用记录类型作函数的结果。

第四节　动态数据结构

一、指针类型和动态变量

访问一个内存变量通常有两种方法。一种是通过名字访问，另一种是通过地址访问。指针是通过地址访问变量的一种数据类型。

由于动态数据结构的变量必须在程序执行过程中动态生成，所以不能预先说明，无法预先给这些变量起好名字，访问时也无法通过名字进行，由此只能用指针得到它的地址，然后间接访问它们。

定义：TYPE　指针类型标识符＝↑类型标识符；

如：TYPE

```
        inpoter=↑integer;
```

说明：

```
VAR
        p:inpoter;
```

生成一个动态变量同时返回动态变量的地址用过程语句 new(p)，用 P ↑表示动态变量。消去 p 所指的动态变量，使 p 的值变为 NIL 用 dispose(p)；

二、向量和线性链表

1. 线性表

一个线性表是有限元素的一个有序集合，线性表可表示为

$$(K1,K2,\cdots,Ki-1,Ki,Ki+1,\cdots,Kn)$$

其中 K1,K2,…,Kn(n>0)称为表元素，当 1<i<n 时，表元 Ki −1称为表元 Ki 的前导表元，而表元 Ki+1称为 Ki 的后继表元。当 n=0时，表为空。

线性表的物理存储方式有顺序，散列，链接等。它具有下列性质：

(1) 一致性：表中每个元素的数据类型和长度相同。

(2) 有序性：表中各个元素之间的次序不能任意改变。

线性表常用操作：

(1) 查找：在线性表中查找满足给定特性的元素。

(2) 插入：在线性表中插入一个新的元素到指定位置。

(3) 删除：在线性表中删除满足给定特性的元素。

2. 向量

向量(我们通常称数组)是一种特殊的线性表，它的分量又可以是一个线性表。对于给定的域 D，我们给出向量的递归定义：

(1) 一维数组是一个线性表，它的每个元素属于域 D。

(2) n(n>1) 维数组也是一个线性表，其中每个元素都具有一个相同上、下界的数组。例如，三维数组 P，每维的界长分别为 5,3,3，则称它为 5 * 3 * 3 维数组。

由于数组是连续地存放在存储空间，所以插入和删除运算时，为了保持运算结果仍然是顺序存储，非常麻烦，特别是当插入操作超过了预先分配的存储区域，需要临时扩大是相当困难的。

3. 链表

以链接的方式存储可以克服上述缺点，适合于插入和删除频繁，存储空间大小不能预先确定的线性表。链表有：单向链表，双向链表，循环表等，必须采用动态存储管理的一系列技术，才能灵活地进行插入和删除，充分地利用存储空间。

动态结构可用带指针的记录作为其元素来实现。

例：我们可以利用下面的 node 类型的结点来生成一个链表(见图1-1)：

```
TYPE
  link= ↑node;
  node=RECORD
         value:integer;
         next:link
         END;
VAR
  pr:link;
```

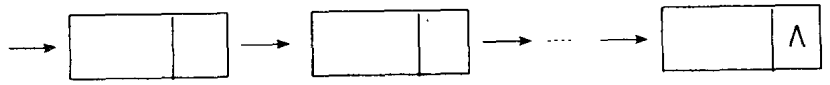


图1-1 链表

其中 pr 指向首结点，称首指针，最后一个结点的指针域为 NIL，表示指针为空，也可用"∧"表示。

为了方便插入、删除操作，有时在链表首部增加一个特殊表元称"哨兵"(guard)。

链表常用操作：

(1) 查找：在链表中检查出满足给定特性的结点。

(2) 插入：在链表中的给定位置后插入一个新结点。

例：在 u 指针所指结点后面插入 v 指针所指结点：

```
PROCEDURE  insert(VAR u,v:link);
        BEGIN
            v↑.link:=u↑.link;
            u↑.link:=v
         END;
```

(3) 删除：在链表中删除满足给定特性的结点。

例：在以 pr 为首指针的链表中删除 v 指针所指的结点：

```
PROCEDURE  delete(VAR pr,v:link)
        VAR  u:link;
         BEGIN
           IF    v=pr
           THEN  writeln('error')     {删去哨兵则报错}
           ELSE
           ELSE
               BEGIN
                 u:=pr;
                  WHILE  u↑.link<>NIL  AND  u↑.link<>v  DO
                 u:=u↑.link;
                  IF  u↑.link=NIL
                  THEN  error  {链表中析所指表元}
                  ELSE  u↑.link:=v↑.link
               END;
        END;
```

(4) 遍历：对链表中的每个成员依次访问一遍。

例：对于上述链表，我们可以在遍历时输出：

```
p:=pr;
WHILE  p <> NIL  DO
   BEGIN
      writeln(pr↑.value);
       p:=p↑.next
   END.
```

以下是两种特殊的链表。

(1) 循环链表(见图1-2)。

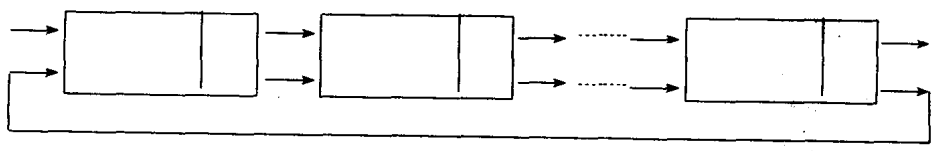


图1-2　循环表

(2) 双向循环链表(见图1-3)。

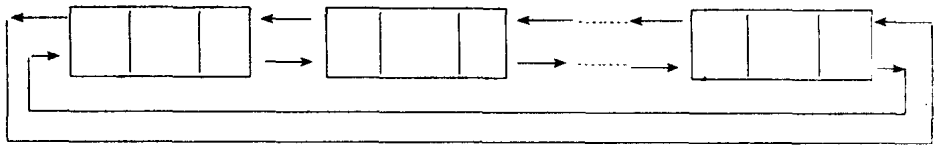


图1-3　双向循环表

链表操作常见错误：

(1) 使用没有定义过的指针。

(2) 指针赋值运算次序颠倒使链表脱钩。

例要删除指针 p 所指结点的后继结点：

```
TYPE
  nodeptr= ↑nodetype;
  nodetype=RECORD
             info:integer;
             next:nodeptr
           END;
  ......
PROCEDURE  delafter(p:nodeptr);
  VAR
    q:nodeptr;
  BEGIN
    IF p=NIL
      THEN
        writeln('void deletion')
      ELSE
        IF  p↑.next=NIL
          THEN
            writeln('void deletion')
          ELSE
            BEGIN
              p↑.next:=q↑.next; 因 q 不指向任何结点,故此语句造成链表脱钩
              q:=p↑.next;
              dispose(q);
            END;
END.
```

(3) 指针变量与普通变量互相赋值，一起参加运算。

例：

```
VAR
  p:↑integer;
  a,x:integer;
BEGIN
  new(p);
  p↑:=3;
  a=5;
  x=a+p;      此语句出错
  ......
```