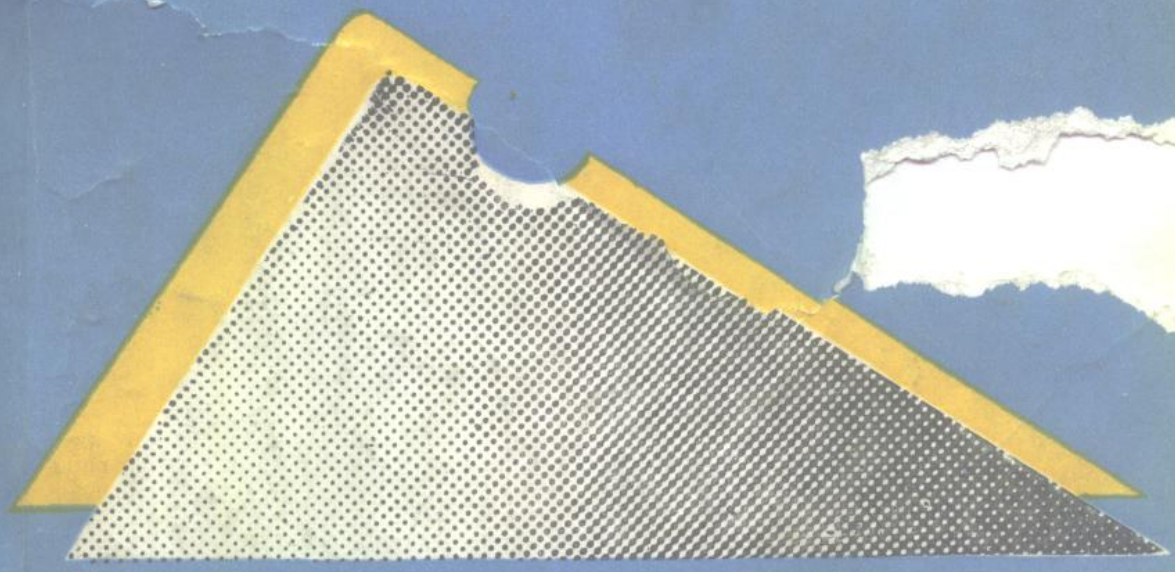
高等教育电教教材

计算机软件技术基础

杨德元　主编



高等教育出版社

高等教育电教教材

计算机软件技术基础

杨德元　主编

高等教育出版社

内 容 提 要

本书是为“计算机应用软件人员电视培训班”编写的教材之一．内容包括 Pascal 入门、实用数据结构、系统实用程序、语言编译和操作系统等几部分。书中含有大量例题，特别是具有相当数量的结合应用软件人员水平考试的题例分析．书中还附有大量习题，供读者练习，以提高编写程序的能力．本书注重基础训练和内容实用性．

本书配有教学录像带，录像带由高等教育出版社出版．

该书适合于应用软件人员、各类程序员、高等学校非计算机专业的本科生、研究生以及教师选作教材或参考书．

高等教育电教教材

计算机软件技术基础

杨德元 主编

*

高等教育出版社出版

新华书店北京发行所发行

北京印刷一厂印装

*

开本 787×1092 1/16 印张 31.75 字数 710 000

1988 年 12 月第 1 版 1988 年 12 月第 1 次印刷

印数 0001— 16 220

ISBN7-04-001961-2/TP·39

定价 7.10 元

前　言

随着微电子和通信技术的飞速发展，计算机在我国的应用已经深入到各个领域．特别是近几年来，由于计算机的广泛应用，大大地促进了各个领域生产和科技水平的提高，加速了我国社会信息化的进程．

由于应用领域的扩大和深入，对软件产品品种和复杂性的需求，远远超过了人们的设想．目前应用开发远远落后于应用需求的矛盾正日益尖锐．供需之间的矛盾，主要表现为应用软件开发的大量积压．一种是可见积压，即已经提出了开发任务，但需要等待以年计的时日之后，才有可能得到开发；一种是更为广泛的，尚未提出的不可见积压．而今硬件技术急速发展，硬件成本继续下降，微型和小型计算机如同雨后春笋，计算机网络更将是星罗棋布．这一切更加促进了开发与需求之间矛盾的加剧．

矛盾的来源主要在两个方面：一是应用软件开发的生产率低，二是广大应用领域中不仅缺乏应用软件开发人员，而且现有软件人员的素质还有待进一步提高．

解决这一矛盾的根本措施在于扩大应用软件的开发队伍和提高应用软件人员的素质．本书即是为提高应用软件人员的素质而编写的．

“计算机软件技术基础”的主要目的在于帮助各类在职人员为掌握程序设计技术和进行应用软件开发奠定必要的软件基础．软件基础具有广泛而丰富的内容，本书主要从程序设计人员的基本要求出发，注重基础，强调实用．内容包括实用数据结构，系统实用程序，高级语言与编译，以及操作系统等．为了便于讲述数据结构，增加了Pascal语言一章，这一章并不全面介绍Pascal语言，主要是讲述为数据结构描述算法所必要的语言成分以及展示结构程序设计技术的基本思想．

编者相信，广大读者从本书繁多的内容中，如能以数据结构为主攻目标，抓好程序设计的基本方法，掌握系统软件的特征，了解高级语言的共性，就完全可能通过知识的掌握和技能的训练，达到素质提高的目的．这里必须强调指出：一定要上机实践，即不仅要完成习题，还必须坐在计算机终端前，动手、实践，甚至成为一个计算机迷．

本书是由许多同志根据多年的教学实践分别编写的．因时间紧迫，缺点和不足在所难免，敬请读者批评指教．其中第一章由米宁编写，米宁和严蔚敏共同编写第二章至第七章，田金兰编写第八章至第十三章，张素琴编写第十四章至第十七章，金千方编写第十八章至第二十五章．参加编写工作的还有孙鸿昌，吕映芝和傅兴纲等同志．全书由杨德元进行了统一修改．

本书由杨冬青同志审阅了第一至第七章，吴家勋同志审阅第八章至第十三章和第十八章至第二十五章，蒋维杜同志审阅第十四章至第十七章，蔡大用教授阅读了第六章有关稀疏矩阵存储方法的内容，并提出了中肯的意见在此谨向审阅本书的各位专家致以衷心的感谢．

本书编写之初曾得到高等教育出版社邀请的多位顾问和专家的指导，特别要感谢金兰教授和胡正家教授对本书编写的指导思想和方法所提出的许多宝贵意见。

编　者

1988.6 于清华大学

目　录

第一部分　Pascal入门

第二部分　实用数据结构

第三部分　系统实用程序

第一部分 Pascal入门

第一章 Pascal程序设计简介

Pascal 语言是由N. 沃思教授于 1971 年研制出来的一种高级程序设计语言. 它是系统地体现结构程序设计思想的第一种语言，是程序设计语言发展史上的一个里程碑. 目前这个语言已经得到极为广泛的使用，尤其在程序设计教学方面，几乎已经成为标准的教学语言. 它简单易学，控制结构和数据结构都十分丰富，而且功能很强(如递归特性等). 用它书写的程序易懂易维护，转化为其他语言的程序也不困难.

本书不是一本专门讨论 Pascal 语言的教材，只是希望利用这种语言中提供的概念和手段讲解数据结构，因而只能利用有限篇幅重点地介绍这种语言中的最常用和最重要的部分. 本章介绍最基本的部分，以后章节中还要根据需要再引入一些概念和用法. 已经熟悉 Pascal 语言的读者可以跳过本章，或只读其中的程序设计例子. 有关 Pascal语言的问题可参阅文献[1]～[7].

为了全书统一起见，本章中的程序和程序段落一律按"算法"编号. 下一章详细介绍算法概念.

1.1 Pascal 语言概述

Pascal 语言是一种**高级语言**. 高级语言能够比较直接地表达算法，因此程序比较容易编写，程序设计活动的主要内容集中在问题分析与算法设计方面，最后用高级语言写出的程序也比较容易读懂.

高级语言对于人来说比较方便，而机器却不能直接执行用高级语言书写的程序，必须经过翻译. 通常有两种翻译方式. 一种是边执行边翻译，即需要执行某条语句时先将它翻译成机器语言，然后执行它，此后重复上述过程，直至结束. 这种方式称为**解释**. 其优点是程序修改后再运行比较方便，执行翻译任务的系统程序也比较小；缺点是执行速度较慢. 多数机器上配备的 BASIC 语言都采取解释的方式运行. 另一种方式是执行前将整个程序都翻译成机器语言程序，然后再执行. 这种方式称为**编译**. 其优缺点恰好与解释方式相反. 对于同样的程序，编译执行与解释执行的速度通常相差 10 倍以上. Pascal 程序一般采取编译执行的方式.

用 Pascal 语言书写的符号形式的程序称为**源程序**. 一个新编好的 Pascal 源程序先要从终端键入机器中，再调用编译程序对它进行编译，然后才能执行. 但编译一次以后，相应的机器语言程序就保留在机器中了(这种程序称为**目标程序**)，以后可以运行任意多次而不必每次执行前

都编译. 如何键入源程序,以及如何编译和运行等具体操作方法因机器而异,通常是通过一些操作系统命令实现的.

通过大量的程序实例展示语言的特性和用法,可能是学习一种新的语言的捷径. 本章主要采取这种方式.

例 1.1 "打招呼"程序.

算法 1.1 打招呼

```
program  greetings(output);
begin
   write('Hello!')
end.
```

这是一个最简单的 Pascal 程序. 这个程序在机器上运行以后,只在终端屏幕上显示"Hello!"便执行结束. 程序的第一行是首部,第二至四行是程序体. 首部必须以**program** 开头,然后是一个由程序员为此程序起的名字,程序名前要放一个或多个空格符,以便与**program** 区分开. 括号部分是一个程序参量表,其中列出了此程序中用到的文件名. 程序在执行中将向这些文件中输出数据或从中读入数据. 在这个例子中,程序要向屏幕输出数据,屏幕也是一个文件,它的名字是 output;键盘是另一个文件,名为 input. 这两个文件名是预先约定好的,不能更换. 程序体中含有一系列语句,用 **begin** 和 **end** 包起来. 整个程序以"."结束.

在这个程序中,只有体内的"写语句"是可执行的语句,即有实际对应的执行动作,其余部分只起某种说明作用.

Pascal 语言是一种**自由格式**的语言. 上面的程序写成四行并适当地空格,只是为了便于人们阅读. 如果把上面的程序按如下的格式键入机器,程序同样能够编译和执行,执行结果也完全相同:

```
program␣greetings(output);begin␣write('Hello!')␣end.
```

或者

```
program
greetings(output)␣;␣
begin␣␣␣write('Hello!')␣␣␣end
.
```

在上面的两种格式中,"␣"表示一个空格符. 每行的行尾相当于有一个空格符. 第一种格式中的空格符数量是最少的,在每一个空格符前增加任意多个空格符都可以,但各个字内不能加空格符,否则就变为两个字了. 当然,单引号内也不能随便加空格符. 如果在叹号前加一个空格符,则运行后就会显示"Hello␣!",此程序变成了另一个程序.

例 1.2 求圆的面积的程序.

算法 1.2 圆的面积

```
program  area(input,output);
```

```
const  pi=3.1416;
var  r,a:real;
begin
    writeln('Enter the radius of a circle please.');
    read(r);
    if r<0 then write('Arguement error.')
    else begin
        a:=r*r*pi;{计算面积}
        write('Area:␣',a)
    end {else}
end.
```

这个程序比上一个程序复杂一些. 由于需要从键盘上读数据，程序参量表中增加了 input. 程序首部与程序体之间增加了两个说明部分：前者为常量说明部分，引入标识符 pi 表示圆周率；后者为变量说明部分，列出了程序中将要用到的变量，并说明它们都是实数类型(real)的变量. 程序中用到的一切常量标识符和变量标识符都必须在这两个部分中说明.

写语句的另一种形式是 writeln，它与 write 的不同之处在于当执行完输出数据的工作之后，将光标移到下一行开始.

程序体中共有三条语句，即 writeln，read 和 if，语句之间由分号隔开. if 语句比较复杂，它是一个分支语句，内含一个条件和两个语句. 当条件成立时执行第一个语句(而不执行第二个语句)，否则只执行第二个语句. 第二个语句是一个由 **begin** 和 **end** 包起来的语句组，这样的语句称为**复合语句**. 复合语句中的第一个语句是一个**赋值语句**，它的作用是计算":="右边的**表达式**的值，并将此值赋给":="左边的变量. ":="称为**赋值号**. 如果右边只有一个变量，则将此变量的值赋给左边的变量，因为变量也是一种表达式，常量亦然(但常量是不准改变的量，不能出现在赋值号左边). 不难看出，程序体就是一个复合语句.

花括号括起来的部分(包括这对花括号)称为**注释**，用来解释某个语句或语句段的功能，其目的是增加程序的可读性. 程序编译时，注释统统被编译程序滤掉，因而注解不影响程序的功能. 任何可以加空格的地方都可以加注释.

这个程序运行以后，屏幕上显示

Enter the radius of a circle please.

即要求用户键入一个圆的半径. 若用户键入 2 (实际上键入"2 RETURN"，用回车键标志输入数据结束，以后的叙述中均忽略这个细节之处)，等待输入数据的 read 语句才能执行完毕，此时变量 r 中的值为2. 随后程序显示以下计算结果并执行结束：

Area:␣␣1.25664E+01

这里，结果是用科学表示法给出的，即 1.25664×10^{1}.

一个 Pascal 程序包含两类符号. 一类是语言自身规定的有特定意义的符号，例如乘号“*”，逗号“,”，括号“(”，赋值号“:=”等，或者是**保留字**，如 **program**, **begin**, **if**, **then**, **else** 等等. 保留字在本书中用黑体印刷(图除外)，键入计算机时用小写或大写字母键均可. 另一类符号是**标识符**. 有些标识符是标准标识符，编译程序认识它们，因而不用定义就可以使用，如 real, read, writeln 等，其他标识符是由程序设计者任意选定的，如 area, pi, r 等等. 一个合法的标识符是一个以字母开头，由字母和数字组成的字符串，其中不能含有其他字符. 在选择标识符时应注意不能与保留字和标准标识符相重(参阅附录 B)，应该以易于记忆为原则. 由于通常的编译程序只识别标识符的前 6～8 个字符，最好使不同的标识符保持前 6 个字符不相同.

在上面的程序中，也可以去掉常量说明部分，并将程序体中的 pi 替换或 3.1415. 在语句中出现的这种直接写出的常量称为**直接量**. 实际上，写语句中由单引号引起来的字符串(包括单引号在内) 也是直接量. 通过引入常量标识符而代替直接量的用法有时有很大好处. 假设一个大程序中有多处要用到 π 的值，当需要将 π 的值改为 3.14159265 时，引入常量标识符的程序只要改一处，而直接量用法的程序中则要改多处，一旦某处漏改，就隐含了一个错误.

Pascal 是一种结构程序语言，大结构内套着小结构，每个结构只有一个入口和一个出口，编写程序时一般不用 **goto** 转移语句. 阅读这样的程序时很容易弄清每个结构完成了怎样一件事情. 这与 FORTRAN 和BASIC语言的早期版本很不同.

从上述例子中虽然可以得到对 Pascal 程序的形象化印象，但还无从得知怎样才是一个严格地合乎语法的 Pascal 程序. 一个程序中只要有一处不合语法，即使根本不影响理解，编译程序也会拒绝执行编译. 一种描述语法的手段是**语法图**. 附录 A 给出了一个 Pascal 语言源程序的完整语法图. 在语法图中，圆圈或椭圆中的符号或符号串是应该原原本本地出现在程序中的符号，称为**终结符**；方框中的名字表示一个需要进一步解释的语法对象，这个对象的语法由另一个图表示，可以在上下文中找到，称为**非终结符**. 每个在此语法图上“走得通”的程序都是一个合法的 Pascal 程序.

例如，对于例1.1中的程序，由于希望它是一个合乎语法的程序，所以先在程序语法图上“走”，即进入第(1)子图. **program** 得到匹配之后遇到 greetings，希望它合乎标识符的语法，于是进入第(15)子图，在含“字母”的圈上转了 7 圈之后结束，回到子图(1)，随后匹配“(”，识别标识符 output 同前，匹配“)”和“;”继而进入子图(2)识别“块”，进入最后一支，匹配 **begin** 之后转去识别一个语句，进入子图(3)中“过程标识符”一支，随后识别(过程)标识符 write、“(”，继而进入子图(8)、(9)、(10) 不转圈，(11)仅走第一支，最后进入子图(13)，走最后一支，在含字符的环中转 5 圈半后顺序退出子图(13)、(11)、(10)、(9)、(8)，回到子图(3)中转走处，识别“)”后回到子图(2)，识别 **end**，最后回到子图(1)，识别“.”并结束. 因此，例1.1中的程序是一个合乎语法的 Pascal 程序(简称**合法**的程序). 例 1.2 中的程序合法与否可由读者验证. 学会使用语法图是十分重要的.

以后各节的讨论中经常出现“存取”一词，它在硬件基础课程中已经讨论得较充分. 对变量的**存取**，指的是向一个变量中存入新的值或只是将其中内容取出来使用而不改变其内容.

1.2 标准数据类型

计算机处理的数据是多种多样的，不同种类的数据在存储和处理方式上都很不相同. 例如，将两个字符相加是没有意义的，因而有必要对数据分门别类. 在程序中，数据主要由变量表示和存储，变量可取值的范围(即可取值的集合)在变量的特性描述中起着重要的作用. 值集合与在其上规定的一组操作合起来称为(**数据**)**类型**.

编写程序时每引入一个新变量都要先说明它的类型，这样做有很多好处. 了解一个变量的取值范围就可以了解它的大概用途，对于程序的可读性非常重要. 编译程序也是根据变量的取值范围为它分配存储空间的，例如一个字符型变量的取值范围就是附录C中列出的128个字符，一个字节可以表示256种不同状态，因而分配一个字节就足够了，而对于整型变量就必须分配一个完整的机器字(4～6个字节)才能满足要求，此外，编译程序还可以根据变量的类型检查出一些程序的设计错误(如字符相加). 每个表达式都有一个确定的类型.

Pascal语言预先规定了四种最基本的数据类型，即整型、实型、布尔型和字符型，类型名分别为integer, real, boolean和char. 它们都是有序的数据类型，即任一类型中的任意两个值都可以比较大小，故也称为预先定义好的纯量型.

1.2.1 整型

整型所表示的是整数，例如3798, 0, −56等. 数学中的整数有无穷多个，然而每个计算机的表示能力都是有限的. 实际上，每个具体的计算机系统都规定了整型变量取值范围的上下界，不同的计算机系统所规定的界限一般是不同的. Pascal语言中规定了一个标准常量标识符maxint，用来刻画这对界限，整型变量的取值范围是−maxint至maxint(字长为32位的机器上maxint $=2^{31}-1=2147483647\approx 2\times 10^9$).

下面列出了Pascal语言中一些整型直接量的写法：

2096　　(不能写成"2,096")

−2

+30

0

整型量(包括直接量、常量和变量)之间可以有加、减、乘、整除和求模(即求整除的余数)等运算，得到整型结果. 它们的操作符分别为+, −, *, **div**, **mod**. 其中后两个操作符定义如下：

$$\mathbf{a\ div\ b}=\begin{cases}\left\lfloor\dfrac{a}{b}\right\rfloor, & \text{当}\dfrac{a}{b}\geqslant 0\text{ 时}\\ \left\lceil\dfrac{a}{b}\right\rceil, & \text{当}\dfrac{a}{b}<0\text{ 时}\end{cases}\qquad(1\text{-}1)$$

$$\mathbf{a\ mod\ b}=a-((a\ \mathbf{div}\ b)*b)\qquad(1\text{-}2)$$

其中(1-1)式含有"下取整"和"上取整"两对数学符号. 例如：

89 **div** 10＝8

89 **div** (－10)＝－8

89 **mod** 10＝9

运算结果的绝对值不得超过 maxint，否则发生运行时错误(溢出).

整型表达式的打印格式有以下约定. 设整型变量 n 含有－173，印出时需要 4 个字符位置. 标准过程 write 中的格式写法与打印结果如下所示:

write(n:6)　印　␣␣－173　因 6＞4，右对齐.

write(2*n＋1:2)　印　－347　因 2＜4，左对齐.

如果冒号与其后的打印区宽度省去不写，则按编译程序自定的宽度印(可能是 1):

write(－173) 印 ␣␣␣␣␣␣－173

1.2.2 实型

科学计算中遇到的数值大到天体质量，小到原子直径，变化范围相当大，Pascal 语言中设立了实数数据类型，用来表示这类数据.

实数在计算机中的表示方法与整数不同. 如果用 32 位的一个机器字存放一个实数，小数 19.5 是按下述方式存放的①: 因为

$$19.5=(10011.1)_2=(0.100111\times10^{101})_2$$

即$\left(\frac{19.5}{2^5}\right)\times2^5$的二进制表示形式. 用第一个字节存指数，后三个存尾数(即小数点后的部分)得到

00000101 01001110 00000000 00000000

其中，前两个字节的第一位分别表示指数和尾数的正负号. 当用于表示一个实数的二进制位数一定时，所能表示实数的总数目不会超过所有不同的状态数(例如 3 位可得 8 种不同状态). 因此，表示范围越大，所能表示的数值就分布得越稀疏，或者说，有效数字位数就越少，精度就越低. 为了简单起见，考虑用一个字节存储一个实数的情况，在某种表示方法下所能表示的值如表 1.1 所示(表中只列出了部分正值，负值可由正值改变符号而得到).

1.1　单字节表示实数的情形

112	96	80	64	56	48	40	32	28	24	20	16	14	12	10	8
7	6	5	4	$\frac{7}{2}$	3	$\frac{5}{2}$	2	$\frac{7}{4}$	$\frac{3}{2}$	$\frac{5}{4}$	1	$\frac{7}{8}$	$\frac{3}{4}$	$\frac{5}{8}$	$\frac{1}{2}$
$\frac{7}{16}$	$\frac{3}{8}$	$\frac{5}{16}$	$\frac{1}{4}$	$\frac{7}{32}$	$\frac{3}{16}$	$\frac{5}{32}$	$\frac{1}{8}$	$\frac{7}{64}$	$\frac{3}{32}$	$\frac{5}{64}$	$\frac{1}{16}$	…	$\frac{7}{2^{10}}$	$\frac{6}{2^{10}}$	$\frac{5}{2^{10}}$

由此可见，计算机中所能表示的实数只是数学中实数的一个很小的子集. 大部分有理数和所有无理数都不能精确地表示. 为了扩大表示范围和提高精度，表示实数的位数一般不少于 32 位. 一个典型的大、中型计算机可以提供$\pm10^{\pm50}$的表示范围和 10 位十进制有效数字的精

① 仅仅在原理上是这样.

度，足以满足一般计算的要求.

实型直接量可以用常规形式写，也可以用科学记数法，例如：

+200

200.0　　(不能写成“200.”)

−0.7500004　(不能写成“.7500004”)

−3.27E−5　(即 -3.27×10^{-5})

1E−50　　(即 10^{-50}，不能写成E−50)

最后一个实数在某些机器上只能用0近似地表示；直接量1000000.00000005可能与直接量1E6具有相同的内部表示. 一般地说，比较两个实数是否相等是没有意义的，应代之以判别两实数差的绝对值是否小于某个小正数.

实型量之间可以进行加、减、乘、除等运算，得到实型结果. 它们的操作符分别为+，−，*，/. 运算时如果运算结果超出了机器表示范围，就会因溢出错误而终止运行.

实型表达式的打印格式更复杂一些，一般的形式为write(表达式:打印区宽度:精度). 举例如下：

write(3.14159:5:2)　印 ␣3.14

write(3.14159:5:1)　印 ␣␣3.1

write(3.14159:5:3)　印 3.142 自动舍入

write(3.14159:9)　印 ␣␣3.1E+00 无精度则按科学记数法印

write(−0.007:9)　印 ␣−7.0E−03

write(−0.007:8:3)　印 ␣␣−0.007

write(−0.007)　印 ␣␣−7.0E−03

最后一种情况按编译程序自定的宽度印.

由标准过程read向一个变量中读入实数时，键入的字符串序列必须符合实型直接量的语法.

另外两个常用的标准函数是取整函数和舍入函数，它们都将实型值变为整型值：

trunc(4.9)=4
trunc(−4.9)=−4　} 向零截断，舍小数部分

round(4.51)=5
round(−4.51)=−5
round(4.49)=4　} 四舍五入，取最接近的整数，只视小数点后一位而定

1.2.3 布尔型

布尔型变量只能取真或假两个值之一，这两个值分别由标准常量标识符true和false表示，也就是直接量的写法.

布尔数据类型中的顺序关系是false在前，true在后. Pascal提供了一个求序号标准函数

ord，用它求序号时，ord(false)＝0，ord(true)＝1．两值还可以比较大小：false＜true.

与布尔型量有关的基本运算有“与”，“或”和“非”，分别由 **and**，**or** 和**not** 表示．由这三个布尔操作符组合布尔量得到的布尔型表达式，一般表达了一个条件．如果说明了 3 个布尔量：

```
var a,b,c:boolean;i:integer;
```

则布尔表达式

```
not((i≤0) or (i>100)) or c
```

与布尔表达式

```
(0<i) and (i≤100) or c
```

所表达的条件是完全相同的，即“0＜i≤100 或者 c 成立，”但在 Pascal 语言中不能写成

```
(0<i≤100) or c
```

表 1.2 给出了三个基本运算的定义．

表 1.2　布尔运算功能表

	a	b	a and b	a or b	not a
取值	false	false	false	false	true
	false	true	false	true	true
	true	false	false	true	false
	true	true	true	true	false

三个运算的优先级不同，由高到低顺序为 **not**，**and** 和 **or**，就像算术运算中“先乘除后加减”的规则一样．括号可以用来改变运算顺序．因此上面的表达式中若 **not** 后不加括号，则

```
not(i≤0)or(i>100)or c
=(0<i)or(i>100)or c
=(0<i)or c
```

与加括号时的意义完全不同．

下面是一些对布尔变量赋值的例子：

```
a:=false
b:=(i>0)
c:=c or (0<i) and (i≤100)
```

布尔值不能通过 read 从键盘直接读入，但可以由 write 直接印出，例如

```
write(a:6)　印　␣false
```

1.2.4　字符型

字符数据类型的取值范围由具体机器上选用的字符集而定，最常见的字符集是 ASCII 字符集，如附录C 所示．每个字符都有一个次序号，所有字符可以按次序号由小到大排成一列，字符之间的次序关系(即大小关系)由其次序号而定．直接量的写法是用单引号将字符括起来．求序号标准函数可以用于求一个字符的次序号．如果 ord('{')＝123，ord('}')＝125，则'{'＜'}'．

字符集一定含有 10 个数字和 26 个大写拉丁字母，且满足：

(1) '0'<'1'<…<'9'，且次序号连续；

(2) 'A'<'B'<…<'Z'；

(3) 若含小写字母，则'a'<'b'<…<'z'

如果说明

```
var ch:char; i:integer;
```

条件

```
(ord('0')≤ord(ch)) and (ord (ch)≤ord('9'))
```

与

```
('0'≤ch) and (ch≤'9')
```

是完全等价的，同表达了"ch 含有一个数字字符"这一条件．当此条件成立时，就可以将此字符转化为与该数字等值的整型数值：

```
i:=ord(ch)-ord('0')
```

标准函数 chr 是 ord 的逆函数，可以把一个序号(整型量)转化为字符．若 0≤i≤9，则

```
ch:=chr(ord('0')+i)
```

又把一个整数还原成字符，例如当 i=3 时，ch='3'．

一般地说，ord('0')≠0．在 ASCII 字符集中，ord('0')=48．此外，用直接量表示单引号时写成

''''

但当程序执行到 read(ch)时，键入一个单引号就可输入该字符．字符输出的例子如下：

```
write('''':7)     印  '''''''
write('␣')        印  ␣
```

1.2.5 小结

Pascal 语言具有自动类型转换的功能．整型量与实型量相加，或实型量与整型量相加，结果(或相应的表达式)就是实型的，实际上，在相加之前整型量被隐含地转化为实型量．减法和乘法操作亦然．整型量与实型量之间也可以进行比较．表 1.3 总结了各操作符的结果类型．**not** 只有一个操作数，称为单目操作符．

Pascal 语言规定了一些标准函数．这些函数可以直接使用，不必说明或定义．自变量为标准类型的函数如表 1.4 所示，其中的每个函数都只有一个自变量，每个自变量是一个表达式（直接量和变量是表达式的特例）．对于一个具体机器上配备的 Pascal 编译程序来说，通常还扩充了一些其他的函数或过程到 Pascal 库中来，例如随机数产生函数，终止程序执行过程等，使用时应查阅机器手册．但含有这些非标准函数或过程的程序的可移植性不好，即在另一台机器上不能直接运行，而必须修改非标准用法的部分．