程序设计

施振夏　梁晋清　编

上海交通大学出版社

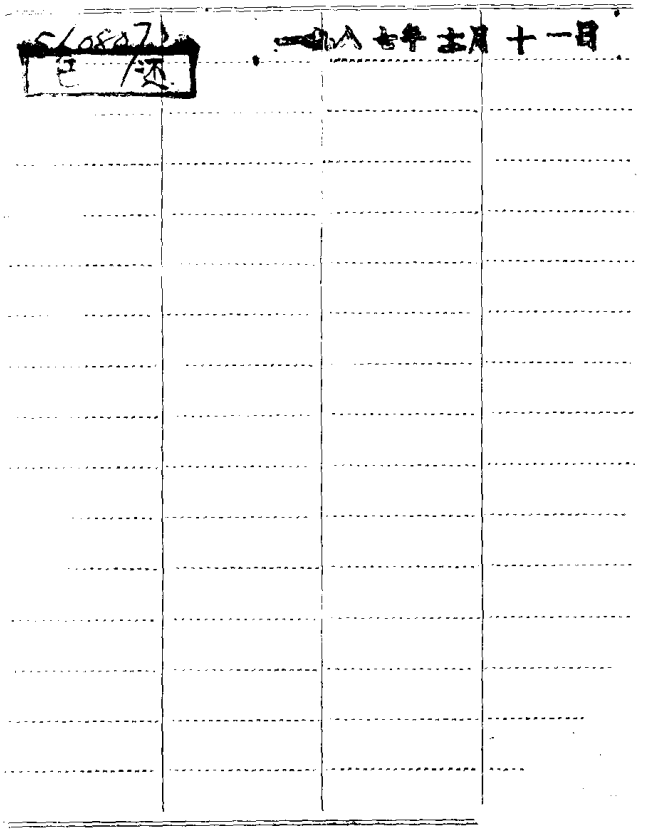
程 序 设 计

施振夏　梁晋清　著

上海交通大学出版社

期　限　表

请于下列日期前将书还回



北京卡片商店 1001

程序设计

☆

上海交通大学出版社出版

（淮海中路1984弄19号）

新华书店上海发行所发行

常熟文化印刷厂排版印装

开本 787×1092 毫米　1/16　印张 17.5　字数 435000

1985 年 12 月第 1 版　　1986 年 2 月第一次印刷

印数 1—10,000

统一书号：15324·160　　科技书目：125—180

定价：3.30 元

前　言

计算机的程序设计是一项创造性的智力活动。《程序设计》课程应该是讲解设计和构成程序的进程(经过)。本书是程序设计的基础性的教科书。

自从七十年代以来，人们开发了结构程序设计的原则，来指导程序设计。本书的主要目的是使读者初步学会根据结构程序设计的原则，逐步细化地研制程序，从而培养读者具有良好的程序设计风格，以便在今后的程序设计实践中，有助于设计出易读、易调和易维护的程序。此外，由于成功的程序设计语言能导致使用它的人们设计出成功的程序，我们认为选择PASCAL语言作为本书的程序设计语言比其它语言都优越。因而，本书的另一个目的是教会读者使用PASCAL语言，把PASCAL语言作为第一程序设计语言将是得益匪浅的。这两个目的是相辅相成的。只有学习应用结构程序设计的原则，才能更深入地掌握PASCAL语言的实质；只有以PASCAL语言作为工具，才能更清晰地体现结构程序设计原则。

本书自始至终以自顶向下，逐步细化的方法描述程序设计进程。全书共分十四章，前十三章结合PASCAL语言，讨论了程序设计的一些基本概念，以及通过挑选的程序例题示范了程序设计的进程。最后一章（程序正确性证明初步）介绍了目前程序设计方法学中的一些基本概念，它是读者进一步深造的阶梯。

本书已在上海交通大学、上海科技大学等院校试用多次，深受学生欢迎。

我们感谢孙永强教授对编写本书的鼓励。感谢宋国新同志对编写本书第十四章的帮助。我们还感谢许多支持和帮助本书出版的同志们。

由于我们学识水平有限，工作经验不足，书中错误和缺点在所难免，敬请读者批评指正。

编　者

1984年9月

目　录

第一章　程序设计的基本概念

学习计算机的程序设计，首先要求了解计算机的特性、计算机程序的特性以及用来表达程序的程序设计语言的特性。本章通过对这些特性的描述，概括了程序设计的最基本概念，对于初学程序设计的人来说，这些内容是以后各章的预备知识。

1.1　程序员理解的计算机

在程序员看来，计算机是一台能可靠地高速地执行很长指令序列的机器。一条指令对应于计算机执行某一基本动作。指令序列是用计算机解一个特定问题时必需执行的动作序列(或解题步骤)，人们通称它为程序。用计算机解题的过程是：计算机的用户把解题的程序装入计算机，计算机执行程序，即输入数据(解这个题目所需要的信息)，对它们进行处理(动作、操作、加工或运算)，最后输出结果。这里，动作和动作的对象（动作所处理的数据）是两个最基本的概念。

用计算机解题有两个特点：

第一是“可靠”。每一台具体的计算机都规定一张指令表，它是全部指令的列表，它指出这台机器能执行的全部基本动作。尽管计算机的结构是如何的复杂，但一个基本动作的内容是简单的，这种动作的数量是有限的。计算机能可靠地执行人们设计的程序，即不多不少地执行了指令序列所规定的动作，即使程序长到几万甚至几十万条指令，计算机也能绝对地服从人的指挥，一点也不逾越人的意志，完成规定的动作。就这点讲，在计算机自动工作的系统中，人是决定的因素，而计算机本身却是一个可靠的“奴隶”。

第二是“高速”。一台计算机每秒能执行上百万次甚至更多的运算。一道普通的程序设计习题，计算机能在瞬间完成，人们要算几个月的工作，计算机能在很短时间内完成，实现人本来难以做到的任务。就这点讲，计算机是一个高速运算的工具。

计算机的可靠和高速这两个特点，使得它能广泛地应用于工业、国防和科研领域，成为我国社会主义四化建设不可缺少的自动的机器。

程序员理解的计算机是由处理器（运算器）、存贮器以及输入和输出设备三大部件组成。

(a) 处理器：执行指令表规定的动作。

(b) 存贮器：存贮处理器动作所需的信息。说信息存放在存贮器中，其实是用存贮器单元内 0/1 状态的组合来表示这些信息的编码。存贮器分为主存（内存）和外存(后备或次级存贮器)两类。

主存中任一单元的内容能被处理器高速存取，存取速度能与处理器的高速动作相匹配。主存除存放各种数据外，还存放着程序本身，程序也以某种形式编码存放在主存中。电子计算机与在它问世以前的任何计算器(例如算盘、手摇计算器等)的根本区别在于计算器的解题步骤总是放在机器之外的某处，例如人脑中，或纸上，并在人的控制下一步一步执行。而

计算机是把程序的编码存放在主存里，也就是存放在处理器能直接取得到的地方，机器就能以与电子运动相关的速度执行人规定的动作序列。这是计算机能高速运算的最根本原因。数据与程序共享内存的思想是 J.V.Neumann (1903—1957) 提出的。因而人们称他为现代计算机之父。主存的缺点是：首先，它不能持久地保存信息，计算机停止运行之后，甚至程序执行完毕之后，主存中的程序和数据也就消失了；其次，它的可利用空间有限，一般是几十到几百 KB(千字节)，较先进的机器可达几 MB(百万字节)，当一个程序很大时，或数据量很大时，程序连同它的数据甚至会超过整个主存的空间，以致不能运行。

为克服主存的这些缺点，计算机必须配置外存，如磁带、磁盘等。它们的特点是：首先，能持久地保存信息。一批数据或一个程序能一直保存下去，不管这个程序是否已执行结束，或计算机是否停止运行。其次，它的容量比主存大得多，一片磁盘的容量一般是用 MB 来度量。外存的缺点是：处理器访问存贮在外存中信息的速度比主存要慢得多。

这两种存贮器相辅相成，对计算机系统来说，都是不可缺少的。

(c) 输入和输出设备：机器要知道人需要处理的数据，人要知道机器运算的结果。输入设备是用来把计算机外部世界的信息传递给计算机主存，输出设备是把计算机主存里的信息传递给外部世界。简而言之，它们是实现人机通讯的工具。常用的输入设备是终端键盘(输入按键字符的信息)或光电机(输入纸带上的穿孔信息)等。常用的输出设备是打印机或屏幕显示，它们分别将正文和图形打印在纸上或显示在屏幕上。

1.2 程序设计

研制计算机程序是一种创造性的脑力劳动。首先，要理解和分析用计算机求解的问题，对于一个大的程序要写出设计说明书。然后用逐步细化(或自顶向下)的方法设计算法：根据说明书的要求，把要解的问题划分成较为简单的子问题的序列，如果这些子问题都解决了，那末整个要解的问题也就解决了。如果子问题还是相当复杂，还可以进一步细化，把这个子问题再划分成更为简单的子问题的序列，再加以解决。如此深入下去，直到整个问题的解决。在开始划分时，只限于抽象地表达问题的解，可以用单纯的文字或以文字为主、程序设计语言为辅，混合表达解题的步骤。每细化一步，都使得解题步骤更加详尽，更加接近最后的解，并有更多的部分用程序设计语言来表达。随着算法的细化，数据的表示也从抽象到具体被细化和构成，直到最后，整个解题步骤用程序设计语言表达，程序就设计完成了。这种逐步细化方法的好处是使得程序员面临着的复杂性局限在当前细化步的子问题之中，使设计的程序结构清晰，错误较少，即使出错也容易发现和改正。[注]

开始时认为程序设计仅仅是使用某种程序设计语言来表示数据与算法的编码技术，那是极其片面的。由于计算机的功能愈来愈强，使用愈来愈灵活以及要计算机求解的问题愈来愈复杂，人们才认识到程序设计远非编码技术。程序设计的主要困难在于寻找出一种合理经济的数据结构和正确有效的算法。编码固然需要做到正确无误，但毕竟还是容易做到的。在设计程序时我们应把主要精力集中在基本算法的构思上。

[注] 程序设计的另一种方法是自底向上。这种方法是先着手设计一些独立的子程序系统，再建立这些子程序系统之间的联系，成为软件系统整体。在设计大型软件时，往往采用这种方法(也要结合使用自顶向下的方法)。当高级设计人员对软件系统的整体结构还不是十分清楚，且对未来必将来临的扩充难能充分预见时，自底向上是一种十分适用的方法。

设计好的程序必须上机运行，输入一组经过挑选的数据，看能否产生预期的结果。即使是一个有经验的程序员设计的程序也难免有错。如果发现错误，程序员还要反复检查和改正他的程序，再上机，直至输入的数据得到正确的结果为止。这一过程称为调试(testing)。一个程序有许许多多组合理的数据，对所有这些数据都进行调试，既不可能，也不必要，所以即使是经调试的程序，也只能说对选定的数据是正确的，决不能通过这种方法证明这个程序对任何合理的数据都是正确的。Dijkstra 有一句名言："程序的调试只能够用来指出程序存在错误，但决不能证明不存在错误"。要做到这一点有待于程序正确性证明技术的进展。一个设计好的程序，经反复调试，即可投入使用。但由于用户需求和环境的变化以及程序本身可能存在的错误，程序在使用的过程中还需要不断修改、增删，逐步改进和完善，这项工作称为维护。

1.3 高级程序设计语言

用计算机机器指令编写的程序称为机器语言程序，程序员不愿意用这种难以记忆的0/1数字符号的组合来设计程序，这实在是太麻烦而又辛苦的工作。尽管如此，机器却仅仅能够识别和执行这种语言的程序。汇编语言虽然能帮助程序员减轻记忆的负担，但一条汇编指令对应于一条机器指令，也只能表达一个"低级"的机器动作，所以汇编语言和机器语言相差无几，它们都具有明显的机器特性。我们通称它们为低级语言。低级语言的缺陷在计算机发展的初期就显示出来：

(a) 由于不同的机器有不同的指令表，所以同一程序在不同的机器上不能通用。如要使用别的机器上的程序，必须用这台机器的指令改写，造成智力浪费。

(b) 程序不合乎人类的自然习惯。例如用机器语言写一个乘法子程序，要用存、取、移位、加、减和判别等十几条指令组成。复杂的程序更是难以设计，就连阅读、交流都十分困难，甚至熟悉某机器的程序员如果有一段时间不接触他的机器，也难以理解他自己写的程序。

(c) 低级语言程序很难做到在程序中"处处设防"，进行仔细的安全检查。

由于以上的缺陷，人们期望有一种高级语言，它能在多种计算机上通用；能以很自然的方式用它编写程序，例如就象写代数式 $A \times B$ 一样写出 A 乘以 B 的程序；还有它能自动帮助用户完成许多安全检查。这种想法促使高级程序设计语言的发展。

我们设想有一台虚拟的计算机，用它的指令(高级语言)编写的程序是较为通用、习惯和安全的。另外，我们还有一台能执行指令表动作的物理的计算机。前者称P机，它是合乎人的需要，但并非实物；后者称为M机，它是客观上可执行的，但使用不便。现在我们要设法把这两种机器联系起来，取长补短，以便使用，即把用P机上高级语言写的(源)程序翻译成M机上低级语言写的(目标)程序，执行这个目标程序等效于执行原来用P机高级语言写的源程序。若能如此，人们可以利用高级语言方便地编写程序，写成的程序观客上又是可执行的。要人来完成上述的翻译工作是极其麻烦的，计算机工作者很自然地想到写一个翻译程序，叫计算机M翻译，我们称这个翻译的程序为编译程序(compiler)C。

运行一个高级语言写的程序分两个阶段：

(a) 编译：编译程序C把高级语言写的源程序翻译成低级语言写的目标程序。

(b) 执行：执行目标程序，输入数据，输出结果。

如下图所示：

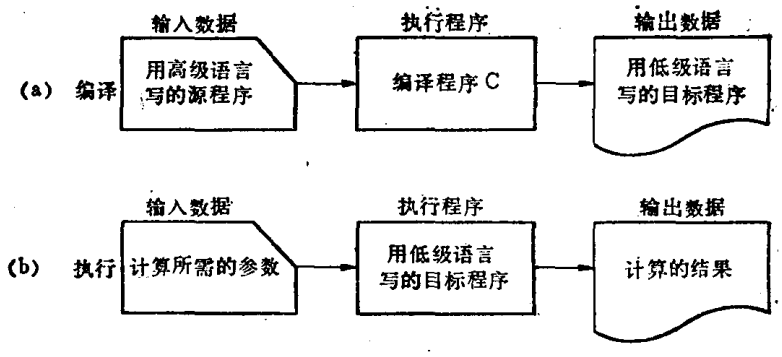


图 1-1 编译和执行一个高级语言写的程序

程序的可能错误有三种：

(a) 编译时的错误：程序员用一种高级语言编写程序时，如果存在着表示法方面的错误，则编译程序能发现这种错误，并打印出出错的信息。例如只有 **begin** 没有 **end** 的漏写。

(b) 运行时的错误：编译成功后在执行目标程序时还可能发现一些语言规则不允许的错误，例如数值的越界、除数为零等。

(c) 逻辑错误：一个程序在编译时没有出现编译时的错误，在执行时也没有出现运行时的错误，它还可能得不到正确的结果。这是由于在算法的设计中或程序的表达中存在逻辑的错误。例如错用 (4/3)*3.1416*R*R 来计算球的体积。

调试要求消除上述各种错误。

1.4 PASCAL 语言

PASCAL 语言是 Niklaus wirth 教授于 1969 年在瑞士苏黎世联邦工学院 (ETH) 设计成功的，1970 年在 CDC-6000 机上首次实现了 PASCAL 语言的编译，1971 年发表了 PASCAL 语言用户手册，1974 年又发表了修订报告[注 1]，1980 年国际标准化组织 (ISO) 发表了 ISO(DP/7185) 关于 PASCAL 的建议草案[注 2]。

本语言的取名是为了纪念法国数学家和哲学家 Blaise Pascal(1623—1662)，他实现了世界上第一台机械式的加法器。他在欧美闻名的却是二项式系数组成的三角形，常称为 PASCAL 三角形。其实，中国古代数学家杨辉，比 Pascal 至少早 400 年就发现了这种三角形，文字记载于 1261 年宋朝的《详解九章算术》。

PASCAL 语言是在高级语言 ALGOL 60 的基础上发展起来的，但其功能比 ALGOL 60 强得多。PASCAL 语言的设计目标是：

(a) 该语言能清晰地展示程序设计中算法和数据的自然结构，因而使程序设计教学成为逻辑的系统的训练。

(b) 该语言能在现有计算机上可靠有效地实现。

[注 1] 见参考文献[1]，本书中提到的"PASCAL 报告"是指该修订报告，本书中提到的"标准 PASCAL"是指该手册中规定的语言。

[注 2] 本书中提到的草案即指参考文献[2]。

十多年的实践表明，和其它任何一种高级程序设计语言相比，PASCAL 语言作为教学语言具有突出的优点，因而已成为国内外许多高等院校计算机科学系的第一程序设计语言。它还愈来愈广泛地用于书写系统程序和应用程序。目前几乎在每一种类型的计算机上都配置有 PASCAL 语言的编译程序。这一切都足以证明 N.Wirth 是成功地达到并超过了他原来的设计目标。

1.5 语言的实现

原则上，象 PASCAL 这样的高级语言本身以及用该语言写的程序是能在各种实际的计算机上通用的。换句话说，它们是独立于机器的。一台计算机只要配置 PASCAL 编译程序，那末它就能运行用该语言编制的程序。但实际上设计出来的编译程序都是各不相同的。为特定语言和特定计算机设计的编译程序称为该语言的一个实现 (implementation)。有时实现扩充或删除了语言规定的某些功能，有时实现要求程序接受语言没有规定的额外限制，有时语言的某些特性留待实现确定。由于这些原因，程序员要在一个具体的实现上运行 PASCAL 程序时，除了要遵守语言规则的要求外，还要遵守那个实现的特殊要求。

一个具体的实现还包括一整套的错误检查，一旦发现有错，即报告错误的位置、性质等信息。

由于各种实现不尽相同，本书中讨论的程序尽量不涉及那些与实现有关的特性。

第二章　表示法和程序的基本结构

2.1　巴科斯范式和语法图

一篇用英语书写的文章是由一系列的字母、数字、标点符号和空格等字符组成。同样，一个 PASCAL 程序，它的正文也总是由字母、数字和语言规定的特殊符号组成。广义地说，它是一个字符序列。下面是一个用 PASCAL 语言编写的简单程序：

```
program add( input, output );
  var first, second, sum : integer;
  begin
    read ( first, second );                    (2-1)
    sum := first + second;
    writeln( sum )
  end.
```

一种自然语言都有它的一整套的语言规则，每一种计算机程序设计语言也有它一整套的语言规则，而且比自然语言规定得更加严格。用这种语言写的任何计算机程序都不允许违反这些规则，哪怕是一点点也不行。一个程序是否正确，首先要判定它是否遵守这些规则。根据这些规则，程序员才能使用语言编制程序，而且根据这些规则，程序员才能阅读和理解别人设计的程序。

语言规则由语法(syntax)规则和语义(semantic) 规则两部分组成。语法规则确定了字符和符号如何构成一个合法的程序。语义规则确定了程序的含义。通常，语法规则是用严格的形式化方法来描述的，而语义规则目前还只能用非形式化的方法来描述。这里我们介绍两种描述语法规则的方法，一种是巴科斯范式(Backus Naur Form)，简写为 BNF，本书中常简称它为语法规则；另一种是语法图。例如一个 PASCAL 程序的语法规则可用下面的 BNF 表示：

〈程序〉::=〈程序首部〉〈分程序〉·　　(2-2)

这一规则的读法是：一个程序定义为程序首部，紧接分程序，紧接·。它表示任何一个 PASCAL程序都是由上述三个部分组成。

(2-1) 式是一个程序，它也必须遵从规则(2-2)，第一行的全部符号构成程序首部。此后，从 **var** 到 **end** 的全部符号构成分程序，最后是·，表示程序结束，对应于(2-2)式中最后的·。

构成语法的成分有两种：语法实体(又称非终结符)和终结符。括在尖括号里的内容称为语法实体，例如在(2-2)式中的程序、程序首部和分程序，还要用语法规则进一步定义它。字符·本身就是在程序正文中书写出来的实际符号，在这里，它是表示程序结束的符号，不必再用别的语法规则定义它，这些语言符号称为终结符。

BNF 是用来描述语法规则的。描述语法规则的 BNF 符号称为元语言符号。这些符号并不是在程序中出现的 PASCAL 语言符号，不能混淆。元语言符号有：::=，| 和 {}。::=

等价于“定义为”；|表示它的两边的语法成分是可以任选的，它们是或的关系；括在{}中的语法成分可以重复 0 次或几次。

例如标识符的语法规则是：

〈标识符〉 ::=〈字母〉{〈字母〉|〈数字〉} (2-3)

标识符是程序中用到的量的名字，它是字母带头的字母数字序列，换句话说，标识符是一个字母（这时 {} 内的成分为空）或一个字母后面跟几个字母或数字（这时{} 内的成分重复几次）。|表示在一个标识符中，除带头的字母外，后面可以任选字母或数字，任选不止是二者择一，可以是多个成分的择一。例如字母可以是 A 到 Z 之间的 26 个字母之一，数字可以是 0 到 9 之间的 10 个数字之一。其语法规则是：

〈字母〉 ::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

〈数字〉 ::=0|1|2|3|4|5|6|7|8|9 (2-4)

语法规则还可以用语法图表示。一个语法实体的语法图是该语法实体的语法规则的图解。例如(2-2)式用 BNF 表示的程序可以用图 2-1 的语法图表示。

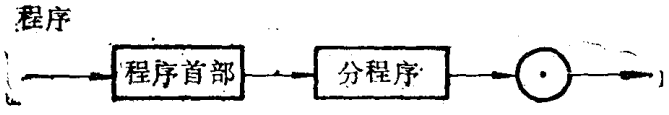


图 2-1 程序的语法图

方框中的内容是语法实体，圆(或圆角框)中的内容是终结符。又如规则(2-3)和(2-4)式的语法图如图 2-2 所示。

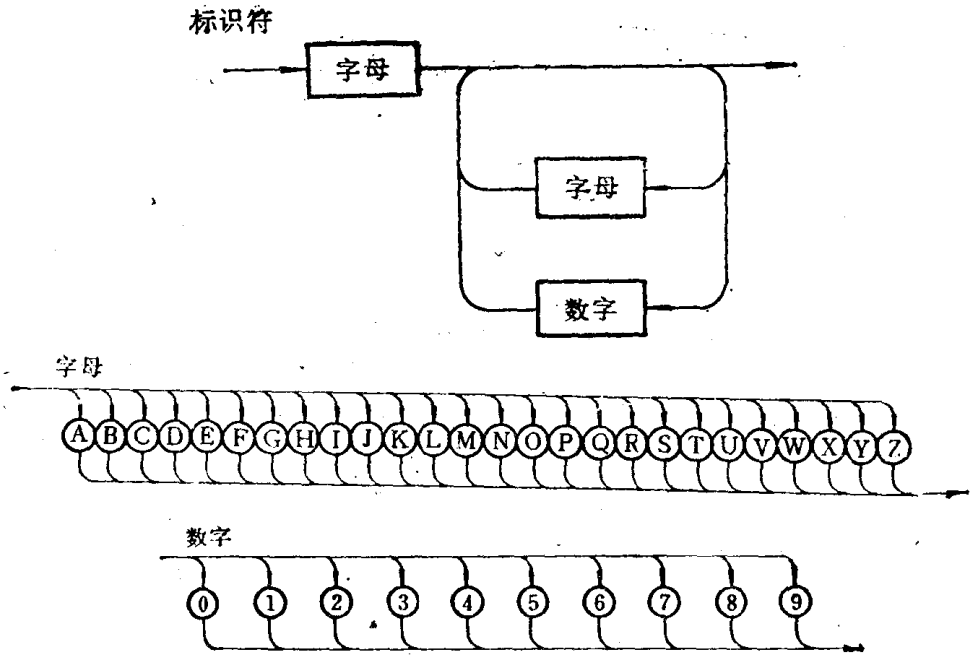


图 2 2 标识符、字母和数字的语法图

利用语法图检查程序的某一部分在语法上是否正确的方法是：把这程序部分与相应语法实体的语法图对照，如果从入口到出口能沿某一路径(分叉路径可任选)走通，则这程序部分在语法上是正确的。所谓走通是指：对于终结符，语法图上和程序中的对应符号相同；对于语法实体，以这程序部分中的某一部分符号再转去对照另一相应语法图，也能从入口到出

口走通。语法上正确的程序就是能走通图 2-1 的语法图。

在图 2-2 上可以看到一个标识符必须由字母带头，后面是一个可以重复 0 次或几次的部分，其中有 2 条可任选的路径。另外在字母和数字的语法图上分别有**26**条和 10 条任选的路径。由此可见，语法图也能明显而精确地表示语法规则。

不论从(2-3)、(2-4)式还是图 2-2 都可以阐明 sum 是一个合法的标识符，而 1ABC 就不是合法的标识符。

我们在全书中讨论语言的一种特性时，总是写出有关的 BNF 规则。但有时往往只涉及到一条规则定义的一部分内容，这时还不能理解有关规则的完整定义，余下来的部分要待以后遇到时再继续讨论，那时才能全面地理解一条规则。

在编制或阅读程序时，如遇到语言的语法疑难问题，可在附录 2 中查阅语法图。

2.2 PASCAL 字汇

前一节讨论过，一个 PASCAL 程序与一篇英语正文相似，是由字母、数字和特殊符号组成。字母、数字和特殊符号这三类基本符号都是 PASCAL 语言的字汇，所有这些字汇组成一张字汇表。一个程序就是由此表中的字汇按照语法规则构造而成的。

按照标准 PASCAL 的规定，一个字母可以是大写的或小写的。不过 (2-4)式或图 2-2 表示的字母只是大写的。有些计算机及其外部设备配有大、小写字母的字符，而有些却仅仅配有大写字母的字符，因而 PASCAL 实现相应地有能同时使用大、小写字母的和仅能使用大写字母的。大、小写字母的作用是否一样，要由具体实现规定。本书中书写程序有两种形式，一种形式是同时使用大、小写字母，用于书写书中的程序例子。如把小写字母改成大写，在只有大写字母的实现上同样能执行这些程序。另一种形式是字母全部大写，它们是在一个具体实现上由机器打印的 PASCAL 程序。

PASCAL语言中的数字是十个阿拉伯数字之一。有些具体的实现还扩充了标准PASCAL的功能，增加非十进制数以及对它们的运算。例如，扩充了十六进制数，数字还应该包括从A到F六个字母。要特别注意字母 O 和数 0 的区别。

表 2-1 特殊符号表

单符	+ - * / . , ; : ' =			
	<> ( ) [ ] { } ↑			
双符	<>>= <= := ..			
保留字	and	array	begin	case
	const	div	do	downto
	else	end	file	for
	function	goto	if	in
	label	mod	nil	not
	of	or	packed	procedure
	program	record	repeat	set
	then	to	type	until
	var	while	with	

PASCAL 语言还规定了大量的特殊符号在程序中作为运算符、分隔符以及其它目的使用。全部 PASCAL 特殊符号在表 2-1 中列出。特殊符号又分为三种：

(a) 单符：单个字符的符号。

(b) 双符：两个字符连起来作为一个不可分割的符号使用，如：〈〉，〉=，·· 等。

(c) 保留字：由字母拼成的字。整个字作为一个不可分割的符号使用，称为保留字(或字分隔符或字符号)。保留字保留有语言规定的含义，不允许程序员改变它的含义，作别的使用。为了醒目起见，本书中的程序例子用黑体字表示保留字。

2.3 数

PASCAL 的数有两种：整数和实数，用十进制表示。它是动作的一种对象，用于数值运算。本节阐明程序中数的表示法。

整数是一个数字序列，它的前面可以带或不带正负号。整数的语法规则是：

〈整数〉::=〈无符号整数〉
　　|〈正负号〉〈无符号整数〉　　(2-5)
〈正负号〉::= + | −
〈无符号整数〉::=〈数字〉{〈数字〉}

下面是合法的整数例子：

0
1983
+128000
−50

下面是不合法的整数例子：

128,000　整数不应有非数字符，。
−6.0　整数不应有非数字符·。

虽然 PASCAL 并没有定义这个数字序列的最大长度，即允许任何大小的整数，但事实上每一具体的 PASCAL 实现对整数都规定一个允许范围(详见第三章)。

如果一个数超过了整数的允许范围，或者一个数带有小数点，则可用实数来表示。实数有两种表示法：

(a) 一般表示法：把实数写成带小数点的数，小数点前后至少要有一个数字，实数前面可以带或不带正负号。下面是合法的实数例子：

0.0
3.1415926
−84.2
+100.001

下面是不合法的实数例子：

0.　小数点后面没有数字。
.693　小数点前面没有数字。
2,456.81　实数不应有非数字符，。