

程序设计

(修订版)

梁晋清 施振夏 陆菊康 编

Pascal
PASCAL

上海交通大学出版社

程 序 设 计

(修订版)

梁晋清 施振夏 陆菊康

上海交通大学出版社

(沪)新登字 205 号

内 容 简 介

本书自始至终以自顶向下,逐步细化的方法描述了程序设计的进程(经过)。根据结构程序设计的原则完整地讨论了 PASCAL 语言。全书共分 15 章,前 14 章结合 PASCAL 语言讨论了程序设计的一些基本概念,以及通过挑选过的程序例题,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。

由于本书概念清晰,重点突出,层次分明,深入浅出,强调程序设计的基本概念和程序设计风格,以及由于 PASCAL 语言所显示出的与日俱增的优越性,本书适宜作为高等院校计算机专业学生的程序设计教科书。同时,也可作为应用计算机的科研、工程技术人员的参考书。

程 序 设 计 (修 订 版)

出版:上海交通大学出版社

(上海市华山路1954号·200030) 字数: 524000

发行:新华书店上海发行所

版次: 1992 年 12 月 第 1 版

印刷:常熟市印刷二厂

印次: 1992 年 12 月 第 1 次

开本: 787×1092(毫米) 1/16

印数: 1—3100

印张: 21.25

科目: 283—338

ISBN 7-313-01130-X/TP·3 定 价: 5.80 元

修 订 版 序 言

程序设计一书第一版出版已有多年,曾三次印刷,适应了国内多所院校的有关教学需要。该书通过介绍 PASCAL 语言,强调良好的程序设计风格和清晰的程序结构,使学生在开始进入计算机领域时得到良好的训练,因而是一本受到广泛欢迎的教材。

程序设计修订本对第一版作了如下的更新:

- 由于计算机领域新概念和新技术的发展,更新了一些描述。
- 增加了第 14 章一个综合实例。该实例是对一个 400 行自顶向下的 PASCAL 编译源程序的分析,它是一个结构程序设计的范例。第一版中的第 14 章改为修订版中的第 15 章。
- 对程序实例和排版作了一些改善。

在本书的出版过程中,得到了许多同志的大力支持,在长期的教育实践中,陈刚、陈毛狗、曹明涛等同志对本书提出了宝贵的意见,作者在此一并表示感谢。

编 者

1992 年 5 月

序 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 70 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他语言都优越。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受益匪浅的。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 14 章,前 13 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 14 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1章 程序设计的基本概念	1
1.1 程序员理解的计算机	1
1.2 程序设计	2
1.3 高级程序设计语言	3
1.4 PASCAL 语言	4
1.5 语言的实现	5
第2章 表示法和程序的基本结构	6
2.1 巴科斯范式和语法图	6
2.2 PASCAL 字汇	8
2.3 数	9
2.4 标识符	10
2.5 字符串	12
2.6 空格、行结束符和注解	12
2.7 程序的基本结构	13
习题	15
第3章 数据的类型、定义和说明	16
3.1 数据的类型	16
3.2 数据说明和定义	25
3.3 标识符定义和说明的唯一性和有序性	28
习题	28
第4章 语句、表达式和赋值	30
4.1 语句	30
4.2 表达式	30
4.3 赋值语句	34
习题	35
第5章 数据的简单输入和输出	37
5.1 输入和输出	37
5.2 PASCAL 的简单输入	38
5.3 PASCAL 的简单输出	39
5.4 程序实例(计算到达时间)	42
习题	44
第6章 基本的构造语句	45
6.1 复合语句	45

6.2 重复性语句	46
6.3 条件语句	58
习题	67
第7章 过程和函数	70
7.1 过程的概念	70
7.2 分程序结构和标识符的作用域	74
7.3 参数	80
7.4 函数	90
7.5 过程和函数作为参数	96
7.6 递归	98
习题	105
第8章 结构程序设计和转(goto)语句	109
8.1 结构程序设计	109
8.2 转(goto)语句	111
第9章 数组	116
9.1 数组的概念	116
9.2 二维数组	131
9.3 对数组的整体操作	134
9.4 程序实例(计算发工资时应付的最合理的元数)	136
9.5 紧缩数组	139
9.6 字符串	141
9.7 程序实例(构造索引表)	142
9.8 其他构造类型	146
习题	146
第10章 记录	150
10.1 记录的概念	150
10.2 记录变量的访问	152
10.3 开域(with)语句	153
10.4 紧缩记录	156
10.5 程序实例(更新足球联赛表)	156
10.6 带变体的记录	162
10.7 程序实例(对正文行长的加工)	166
习题	173
第11章 集合	175
11.1 集合的概念	175
11.2 集合的构造和运算	177
11.3 程序实例	181
习题	197
第12章 文件	198

12.1	文件的概念	198
12.2	文件处理的标准过程(文件的生成—写与文件的使用—读)	199
12.3	程序实例(更新顺序文件)	203
12.4	程序实例(排序一个文件)	207
12.5	文本文件	214
12.6	程序实例(文本编辑程序)	217
	习题	224
第 13 章	指针	225
13.1	指针的概念	225
13.2	链表	230
13.3	树	235
13.4	程序实例(构造索引表)	244
	习题	250
第 14 章	一个综合实例	252
14.1	PL/0 语言及其编译程序	252
14.2	PL/0 编译程序中的编译部分	258
14.3	PL/0 编译程序中的解释部分	283
14.4	PL/0 程序清单	290
第 15 章	程序正确性证明初步	303
15.1	程序测试与程序验证	303
15.2	程序的终止性	307
15.3	Hoare 证明规则	309
15.4	用证明规则证明程序的正确性	311
15.5	最弱前置条件 WP	313
15.6	用 WP 证明程序的完全正确性	315
15.7	机械程序验证系统	316
15.8	小结	317
	习题	318
附录一	ASCII 字符代码	321
附录二	PASCAL 语法图	322
附录三	运算符一览表	328
附录四	标准函数的自变量类型和结果类型	329
参考文献		330

第 1 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),人们统称它为程序。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,一条指令指明机器能执行的一个基本动作,一张指令表指明这台机器能执行的全部基本动作。尽管一台计算机的结构相当复杂,但一个基本动作的内容是简单的,而且这些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,在自动工作的计算机系统中,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,实现人们难能在短时间内完成的任务。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生产、设计和科研领域,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:执行指令表指明的动作。

(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、纸上,并在人的控制下一步一步地执行。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,一般是近 1MB (百万字节),较先进的机器可达若干 MB。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,以致仅有主存的机器不能运行。

为克服主存的这些缺点,计算机必须配置外存,如磁盘、磁带等。它们的特点是:首先,能持久地保存信息。一批数据和程序都能一直保存下去,不管这个程序是否已执行结束,以及计算机是否停止运行。其次,外存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)、鼠标器等。常用的输出设备是打印机、屏幕显示等。

1.2 程序设计

研制计算机程序是一项创造性的脑力劳动。首先要理解和分析用计算机求解的问题,对于一个大型的程序要写出各种要求级别的设计说明书,对于小型程序也要对解题有一个总体的设计,然后用逐步细化(或逐步求精,或自顶向下)的方法设计算法,即根据设计的要求,把一个要解的问题划分为一组比较简单的子问题(这种划分称为问题的细化),如果这些子问题都解决了,那么整个要解的问题也就解决了。如果子问题还是相当复杂,还要进一步细化,把每个复杂的子问题再划分为相应的一组更为简单的子问题,逐个解决。如此反复深入,直到整个问题的解决。这里所谓问题的解决是指使用程序设计语言表达解题的步骤,换句话说,就是计算机的程序设计。在开始划分时,只限于抽象地表达解题步骤,用单纯的文字描述,或以文字为主、程序设计语言为辅的混合形式表达解题步骤。每一步细化,都使得解题步骤更加详尽,有更多的部分采用程序设计语言具体表达,更加接近完整的程序。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,整个解题步骤完全用程序设计语言表达,程序就设计完成了。这种逐步细化的程序设计方法体现了结构程序设计的原则,其突出优点是使得程序设计人员面临的复杂性局限于细化过程及其子问题之中,使设计成的程序逻辑结构自然清晰,错误较少,即使出错也容易发现和改正。

早期的程序设计仅仅是使用某种程序设计语言来表示数据和算法的一种编码技术,那是极其初步的和粗浅的。随着计算机系统的迅速发展,计算机的功能愈来愈强,要计算机求解的问题愈来愈复杂,程序设计已远非编码技术,新的程序设计方法层出不穷,诸如软件工程、人工智能、面向对象等技术,已为程序设计领域开辟了崭新的局面。但逐步细化的方法作为一种基础技术,至少广泛适用于规模较小和层次较低的程序设计场合,所以是初学程序设计的人们必

须首先掌握的技术,使他们摆脱那种无章可循、随心所欲地设计程序的工作方式,而习惯于使用逐步细化的方法设计合理而经济的数据结构和正确而有效的算法。

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,直至对输入的数据得到正确的结果为止。这一过程称为调试(testing)。一个程序所能处理的合理数据的数据极大,对所有这些数据都进行调试,既不可能,也不必,所以即使是通过调试的程序,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。第15章的内容就是有关的预备知识。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和增删,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。我们统称它们为低级语言。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曾经熟悉的机器,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”,进行仔细的出错检查。

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,例如就像写代数式 $A \times B$ 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可能的出错的检查。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。前者称为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,写成的程序客观上又是可执行的。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。

如图 1-1 所示:

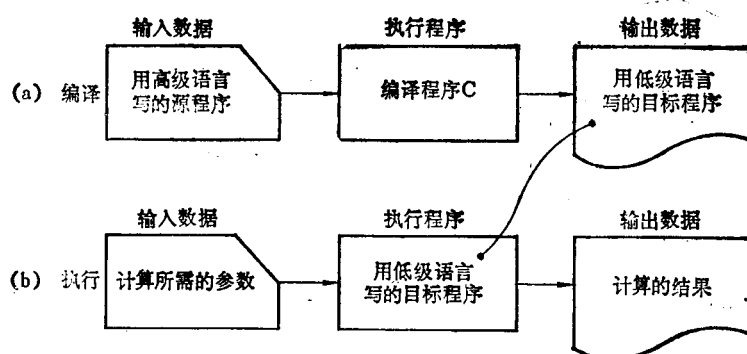


图 1-1 编译和执行一个高级语言写的程序

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 $(4/3) \cdot 3.1416 \cdot R \cdot R$ 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, 1983 年国际标准化组织 (ISO) 发表了 ISO 7185-1983 关于 PASCAL 程序设计语言的国际标准文本, 它是 PASCAL 的严格定义, 因而是采用 PASCAL 设计程序以及实现 PASCAL 编译程序的根据。

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。

20年的实践表明,和其他任何一种高级程序设计语言相比,PASCAL语言作为程序设计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学生学习计算机程序设计的第一程序设计语言。本书采用PASCAL作为背景语言,讨论程序设计尤为合适。此外,它还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。这一切都足以说明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,像PASCAL这样的高级语言本身以及用该语言编制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,那么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)。一方面,实现有时扩充或不提供语言规定的某些功能,有时要求程序接受语言没有规定的实现的额外限制;另一方面,语言有意将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,除了要遵守语言规定的要求外,还要遵守具体实现的特殊要求。

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2章 表示法和程序的基本结构

2.1 巴科斯范式 and 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 );  
    sum := first + second;  
    writeln( sum )  
  end.
```

(2-1)

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)，简称为 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，；，分程序，以及·，该·是(2-2)式中用“”括起来的·，最后的·是表示定义结束。(2-2)式表示了任何一个 PASCAL 程序都是由上述四个部分组成。

(2-1)式是一个程序，它也必须遵重规则(2-2)，第一行的全部符号构成程序首部和；，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。例如在(2-2)式中的程序、程序首部和分程序都是语法实体，还要用语法规则进一步定义它们。用“”括起来的字符·是在程序正文中书写出来的实际符号，它是表示程序的结束的符号，不必再用别的语法规则定义它，这种语言符号称为终结符。

BNF 是用来描述语法规则的，它是 J.W. Backus 在 1959 年首次发表。它采用元语言来描述程序设计语言的语法规则(公式)。因而克服了以往用自然语言描述程序设计语言语法规则混淆不清的情况。卅多年来，元语言记号有所发展。扩展的元语言符号见表 2-1，其含义如下：

表 2-1 元语言符号

元 符 号	意 义
=	定义为
	或者
·	定义结束
[]	例：[x]，表示 x 的 0 次或 1 次出现
{ }	例 {x}，表示 x 的 0 次或多次出现
()	组合，例 (x y)，不是 x 就是 y
"xyz"	终结符 xyz
元标识符〔注〕	非终结符

在这个表示法中，记号 = [[] { } () 及点·都用作元语言记号，但终结符也包括它们，为避免混淆，终结符必须用一对双引号“ ”括起来。例如定义标识符的语法规则是：

$$\text{标识符} = \text{字母} \{ (\text{字母} | \text{数字}) \} \quad (2-3)$$

标识符是程序中用到的量的名字，它是字母开头的字母数字序列。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，任选不只是二者择一，可以是多个成分的择一。例如字母可以是 A—Z 之间的 26 个字母之一，数字可以是 0—9 之间的 10 个数字之一。其语法规则是：

$$\begin{aligned} \text{字母} = & \text{"a"} | \text{"b"} | \text{"c"} | \text{"d"} | \text{"e"} | \text{"f"} | \text{"g"} | \text{"h"} | \text{"i"} | \text{"j"} | \text{"k"} | \text{"l"} | \text{"m"} | \text{"n"} | \text{"o"} | \text{"p"} | \text{"q"} | \\ & \text{"r"} | \text{"s"} | \text{"t"} | \text{"u"} | \text{"v"} | \text{"w"} | \text{"x"} | \text{"y"} | \text{"z"} . \end{aligned} \quad (2-4)$$

$$\text{数字} = \text{"0"} | \text{"1"} | \text{"2"} | \text{"3"} | \text{"4"} | \text{"5"} | \text{"6"} | \text{"7"} | \text{"8"} | \text{"9"} .$$

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。

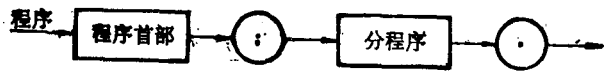


图 2-1 程序的语法图

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。

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：把这程序的某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，则这程序的某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；

〔注〕 元标识符以汉字表示。

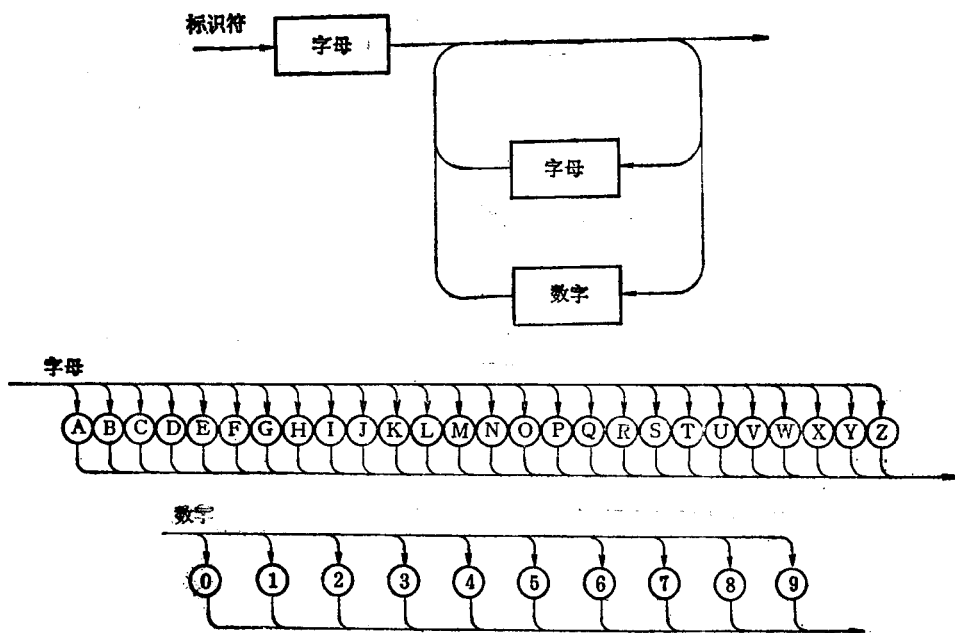


图2-2 标识符、字母和数字的语法图

对于语法实体，以这程序的某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，其中有两条可任选的路径。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4ABC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、数字和特定符号组成。字母、数字和特定符号这三类基本符号都是PASCAL语言的字汇，所有这些字汇组成了一张字汇表。一个程序就是由此表中的字汇按照语法规则构造而成的。

PASCAL语言中的数字是10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16进制数，数字还应该包括从A—F六个字母。要特别注意字母O和数0的区别。

PASCAL语言中，定义字母、数字的语法规则见式(2-4)定义特定符号的语法规则是：

特定符号 = “+” | “-” | “*” | “/” | “=” | “<” | “>” | “[” | “]” | “.” | “,” | “:” | “;” | “↑” | “(” | “)” | “(” | “)” | “<” | “>” | “<=” | “>=” | “:=” | “..” | 字符符号。

字符符号 = “and” | “array” | “begin” | “case” | “const” | “div” | “do” | “down” | “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”. (2-5)

读者注意:特定符号中的<>、>=、<=、:=、..是由两个字符连起来作为一个不可分割的符号使用。另外,字符符号是由字母拼成的字,整个字作为不可分割的符号使用,它保留了语言规定的含义,不允许程序员改变它的含义,作别的使用,为醒目起见,本书中的程序例子用黑体字表示字符符号。

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。

正负号 = “+” | “-”。

无符号整数 = 数字序列。

数字序列 = 数字 {数字}。

(2-5)

下面是合法的整数例子:

1992
+ 128000
- 50

下面是不合法的整数例子:

128,000 整数不应有非数字字符,。

- 6.0 整数不应有非数字字符。。

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3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