程序设计语言

编译原理

中国人民解放军国防科学技术大学　陈火旺
复　旦　大　学　钱家骅　编
上　海　交　通　大　学　孙永强

国防工业出版社

程序设计语言

编 译 原 理

中国人民解放军
国防科学技术大学　陈火旺
复 旦 大 学　钱家骅　编
上海交通大学　孙永强

国防工业出版社

内 容 简 介

本书旨在介绍编译程序构造的一般原理和基本实现方法，内容包括词法分析、语法分析、中间代码产生、优化和目标代码产生。作为一本原理性的书，重点在于介绍基本的理论和方法，不拘泥于具体的实现细节。本书既注意了最经典、最广泛应用的编译技术，又反映了七十年代以来的一些最重要的研究成果。在词法、语法分析方面特别注重分析器的自动产生；在翻译方面突出了语法制导翻译方法；在优化方面强调全局数据流分析。全书的组织注意了前后连贯，循序渐进，各章之后并附有习题。本书可作为高等（理、工）院校计算机科学（或工程）专业的教材，或作为教师、研究生、高年级学生或软件工程技术人员的参考书。

程序设计语言

编译原理

中国人民解放军国防科学技术大学　陈火旺

复　旦　大　学　钱家骅　编

上海交通大学　孙永强

*

国防工业出版社 出版

新华书店北京发行所发行　　各地新华书店经售

国防工业出版社印刷厂印装

*

787×1092 1/16　印张20 3/4　483千字

1980年11月第一版　1980年11月第一次印刷　印数：0,001—8,200册

统一书号：15034·2104　　定价：2.15元

前　言

本书系高等院校工科电子类计算机科学与工程专业统编教材之一。

本书旨在介绍程序设计语言编译程序构造的一般原理和基本实现方法，内容包括词法分析、语法分析、中间代码产生、优化和目标代码产生等五大部分。作为一本原理性教科书，我们着重介绍编译的基本理论和方法，对于实现的具体细节未予详述。本书既注意了最经典、最广泛应用的基本编译技术，如算符优先法和递归子程序法，又力求反映七十年代以来的一些最重要的新成果，如 LR 分析法和全局数据流分析。为了沟通硬件和软件的联系，本书最后一章专门介绍了高级语言计算机系统结构。

对于硬件专业的编译原理课来说，删去所有打星号的章节后，本书可作为 60 学时课程的教科书。对于软件专业而言，本书可作为 80～100 学时的教程。但在学这门课之前，学生必须预修过计算引论(程序设计方法)和高级语言（FORTRAN、ALGOL 或 PASCAL），并且最好具有数据结构和离散数学方面的基本知识。

在本书的引论中，我们概要地介绍编译程序的功能和结构。第一章的内容对于读者应是熟知的，这部分材料主要是对高级语言若干基本概念的回顾和复习。第二章是全书的基础，学生除了掌握词法分析器的基本构造方法外，还应该了解正规式和有限自动机的基本理论，它们和二、三、四各章的关系都很密切。第三章讲文法和基本语法分析器构造，这是现今所有讲授编译方法的教科书大多具备的部分。第四章介绍 LR 分析法和语法分析器的自动构造。第五章介绍各种中间代码及产生方法，着重讨论语法制导翻译产生四元式的过程。第六章讨论符号表组织。第七章讲存贮分配问题。第八章介绍出错诊察处理方法。第九、十两章讨论局部优化、循环优化和全局数据流分析问题。第十一章介绍目标代码产生过程。最后，第十二章讨论高级语言计算机结构。

本书的体系是参照引论后的参考文献〔1〕和〔2〕的结构建立的，有些章节的基本内容也是从〔1〕移植过来的。大多数章节之后附有习题和参考文献。

本书的前七章由中国人民解放军国防科学技术大学陈火旺编写，九至十一章由复旦大学钱家骅编写，八和十二章由上海交通大学孙永强编写。

本书的主审单位是华中工学院软件教研室，主要审稿人是陶葆兰同志。

在本书编写过程中，我们曾得到国防科学技术大学计算机研究所软件教研室、复旦大学软件教研室和上海交通大学计算机系的许多教师、研究生和学生的大力支持和帮助，特别是肖泽志同志做了大量的工作，在此谨向他们表示感谢。

由于我们学识浅薄，编写时间仓促，谬误之处在所难免，恳切希望读者不吝赐教。

编　者

1979. 12

目　录

引　论

0.1　什么叫编译程序

翻译程序是这样一个程序，它能够把某一种语言程序（称为**源语言程序**）改造成另一种语言程序（称为**目标语言程序**）。如果源语言是诸如 FORTRAN、ALGOL 或 COBOL 这样的所谓“高级语言”，而目标语言是诸如汇编语言或机器语言这样的“低级语言”，那么，这样的一个翻译程序就称为**编译程序**。

执行一个高级语言程序大体上可分为**两步**：第一步，把高级语言源程序编译成低级语言目标程序；第二步，运行所得的目标程序。

本课程的目的是介绍设计和构造编译程序的基本思想和基本方法。编译的理论与技术是廿多年来计算机科学中发展得最迅速的一个分支。现在已经初步形成了一套比较系统化的理论与方法，这些理论与方法指导人们如何设计和构造编译程序。由于本课程只是一门引论性的课程，因此，不能指望这份教材能够包括这个领域的全部内容。但我们将力求介绍一些较新的材料。

0.2　编译过程概述

编译程序的典型工作过程是：输入源程序，对它实行加工处理，最后输出目标程序。由于整个加工处理过程是非常复杂的，因此，宜于把整个过程看成是由一系列不同**阶段**所组成的。每个阶段完成一部分特定任务。整个过程可看成是如图 0.1 所示的一条“生产线”。编译程序的结构基本上是仿照这个工作流程而设计的。

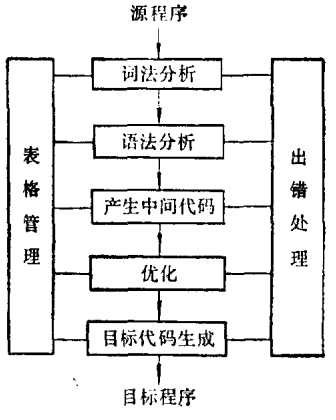


图 0.1　编译过程的各阶段

0.2.1　编译的基本阶段

第一阶段，**词法分析**。执行词法分析的程序叫**扫描器**。词法分析的任务是：对构成源程序的字符串进行扫描（从左到右）和分解，识别出一个一个的**单词**（称为单词符号或**符号**），如基本字（begin、if、for 等）、标识符、常数、算符和界符（标点符号、左右括号，等等）。例如，对 FORTRAN 中的语句

DO　150　I=1, 100

扫描器应逐一识别出基本字‘DO’、标号‘150’、标识符‘I’、等号‘=’、整型常数‘1’、逗点‘,’、整型常数‘100’，等等。这些单词是组成语句的基本符号。作为扫描器的输出，每个基本符号通常用两部分表示：一是**种别**，二是符号自身的**内部形式**。种别是指各类符号的某种整数编码。例如，假定“标识符”这一类符号编为第 5 种，那么，扫描器在识别

出标识符 ABD 之后的输出将表示成'5'和这个标识符自身的“内部形式”。内部形式指字符串的一种编码结构，这种结构是随不同的目标语言（具体的机器系统）而不同的。

第二阶段，**语法分析**。执行语法分析的程序叫**分析器**。语法分析的任务是：根据语法规则，把扫描器所提供的单词符号串分析成各类语法范畴。譬如，由单词符号组成短语，由短语组成语句，由语句组成程序。通过分析，确定整个的单词符号串是否构成一个语法上正确的程序。例如，当接受到符号串

$$(A+B)*C$$

时，分析器则应识别出这是一个表达式。

第三阶段，**中间代码产生**。执行中间代码产生的程序叫**中间代码产生器**。它的任务是：按照语法分析器所识别出的语法范畴（如表达式）产生相应的**中间指令**。中间指令的形式常因各种不同要求而有很大的不同。例如，国产 441B-Ⅲ机 FORTRAN 编译程序所产生的中间指令是一种非常接近于机器指令形式的所谓“半目标指令”；而 151 机的FORTRAN 编译程序则产生一种称为“间接三元式”的中间指令。为了满足代码优化阶段的需要，许多编译程序采用了诸如“间接三元式”、“四元式”或“逆波兰记号”之类的中间代码语言。例如，下面的赋值句

$$X=(A+B)*C$$

可被翻成如下的一串四元式：

算　符	左操作数	右操作数	结　果
+	A	B	T_1
*	T_1	C	T_2
:=	T_2	—	X

其中第一个四元式意味着 $T_1:=A+B$；第二个四元式意味着 $T_2:=T_1*C$；第三个四元式指把 T_2 的值赋给变量 X。T_1 和 T_2 是编译程序引进的临时工作变量。

第四阶段，**代码优化**。优化的任务在于对前阶段所产生的中间代码进行加工变换，以便最后所得的目标程序能运行得更快更省（省内存）。也就是说，优化的任务在于期望获得较高效的目标程序。代码优化的主要方面有：循环优化、公共子式的提取、算符归约以及联系具体计算机系统的特殊优化。

最后阶段，**目标代码生成**。这个阶段的任务是：按照中间代码（或经优化处理之后）和各种表格所记存的有关源程序的信息，确定各类数据的存贮空间的位置，选择合适的代码指令，分配寄存器等等，直到最后形成可直接（或几乎可直接）在机器上运行的目标程序。这部分工作大多和具体的计算机硬件有关，因此，如何产生出足以充分发挥硬件效率的目标程序是一件很不容易的事情。

0.2.2 表格管理和出错处理

编译程序在工作过程中需要保持一系列表格，以登记源程序中各种名字的有关性质和状态。理论上说，用一张统一的符号表也就可以了。但为了便于查找和更改，许多编译程序按不同种类的名字分立了几种不同的表格。如常数表 CD、标号表 LD、过程表 PD，以

及一张用于登记变量名和数组名的一般符号表 SD，等等。各表格每一项的结构大致象下列格式：

名字 (NAME)	信息 (INFORMATION)

其中信息栏通常还划分成若干子栏，分别记录名字的不同属性和状态。

作为一个例子，让我们来看一看 151-FORTRAN 编译程序所设置的各种表格的基本结构。例如对于下述的程序段

```
      SUBROUTINE  INCSWAP (M, N)
10    K = M + 1
      M = N + 4
      N = K
      RETURN
      END
```

经编译的头三段工作后，将产生各种表格和四元式序列●，如符号表 SD（表 0.1）、常数表 CD（表 0.2）、过程表PD（表 0.3）、标号表 LD（表 0.4）和四元式序列表（表 0.5）。注意：在四元式中，实际上我们不是直接写上操作数的名字，而是记上它们在有关表格中的位置（入口数）。在标号表和过程表的信息栏里记上了相应四元式的地址（编号）。

表0.1　符号表 SD

	NAME	INFORMATION
1	M	哑。整变量
2	N	哑。整变量
3	K	整变量

表0.2　常数表CD

	值(VALUE)
1	1
2	4

表0.3　过程表PD

NAME	INFORMATION	
INCSWAP	二元子程序	k

表0.4　标号表LD

NAME	INFORMATION
10	k + 1

表0.5　四元式序列表

	算　符	左操作数	右操作数	结　果
k	subr			
k + 1	+	SD_1	CD_1	T_1
k + 2	: =	T_1	—	SD_3
k + 3	+	SD_2	CD_2	T_2
k + 4	: =	T_2	—	SD_1
k + 5	: =	SD_3	—	SD_2
k + 6	return			

在前面所说的编译过程五阶段的每一阶段中，编译程序都要不断查阅引证或更新修改各符号表的内容。所有这些工作通常是由一组专门的程序来完成的。这组程序称为**表格管**

● 151-FORTRAN用间接三元式作中间代码。

理程序。它的作用就像一位文书一样，抄抄写写，专做薄记工作。

出错处理是编译程序的另一个重要组成部分。编译的各阶段均有能力发现源程序中的这样或那样的错误。不论哪一阶段发现了什么错误，编译程序均应把出错的性质和地点报告给用户，而且还应把出错的影响限制在尽可能小的范围里，使得其余部分能够继续编译下去，以便能进一步发现其它的可能错误。出错处理程序（或称错误诊察程序）就是为此目的而设计的一组专门程序。

“表格管理”和“出错处理”这两部分程序和各阶段程序之间是相互作用的。

0.2.3 遍

编译程序的结构是十分复杂的，而且体积也很大。由于受到具体机器主存容量的限制，往往把编译的几个不同阶段的工作组合成**遍**（pass）。每遍的工作由从外存（如磁盘、磁鼓或磁带等）上获得前一遍的工作结果开始（对于第一遍而言，从外存上获得源程序）。完成它所含的有关阶段程序的工作之后，再把结果记录于外存之中。每遍所含的诸阶段工作往往是穿插进行的，施行控制的是每遍有关控制程序。当一遍工作完之后，它所占用的存贮空间大部分被释放。下一遍进入后，即可使用几乎全部的存贮空间。至于一个编译程序究竟应分成几遍，以及如何划分的问题，这是和所面临的具体机器的内、外存设备有关的，因此难于统一划定。遍数多一点有个好处，即整个编译程序的逻辑结构可能清晰一点。但遍数多势必增加输入/输出所消耗的时间。因此，在主存可能的前提下，一般还是遍数尽可能少一点为好。

0.3 编译程序的生成

以前人们构造编译程序大多是用机器语言或汇编语言作工具的。为了充分发挥各种不同硬件系统的效率，为了满足各种不同的具体要求，现在许多人仍然采用这种工具来构造编译程序。但是，越来越多的人倾向于使用高级语言作工具来构造编译程序。因为，这样可以节省大量的程序设计时间，而且所构造出来的编译程序也易于阅读、修改和移植。

现在人们已经建立了多种编制部分编译程序或整个编译程序的有效工具。有些能用于自动产生扫描器，有些可用于自动产生语法分析器，有些甚至可用来自动产生整个的编译程序。这些构造编译程序的工具有：**编译程序-编译程序、编译程序产生器、翻译程序书写系统**等，它们是按照对源语言和目标语言（或机器）的形式描述（作为输入数据）而自动产生编译程序的。

近年来，有些人主张采用“自编译方式”产生编译程序。意思是，先对语言的核心部分构造一个小小的编译程序（可用手编实现），再以它为工具构造一个能够编译更多语言成分的较大编译程序。如此扩展下去，就像滚雪球一样，越滚越大，最后形成人们所期望的整个编译程序。这种通过一系列自展途径而形成编译程序的过程叫做**自编译**过程。

现在，有些编译程序是通过“移植”而得的。即把某一机器上的编译程序移植到另一机器上。这需要寻找某种适当的“中间语言”。但是，由于建立通用中间语言实际上办不到，因此，移植也只能在几种机种之间进行。

对于自编译和移植有兴趣的读者，请参阅有关专门论著。但对于自动产生器，本课程

将把它作为一个重要课题来讨论。

0.4 学习构造编译程序

要在某一台机器上为某种语言构造一个编译程序，必须掌握下述三方面的内容：

1．源语言　对被编译的源语言（如 ALGOL 或 FORTRAN），要深刻理解其结构（**语法**）和含义（**语义**）。

2．目标语言　假定目标语言是机器指令，那么，就必须搞清楚硬件的系统功能和操作系统的功能。

3．编译方法　虽然把一种语言程序翻成另一种语言程序的方法很多，但必须准确地掌握某些基本方法。

本课程是讲编译方法的，并且主要是讨论 ALGOL 和 FORTRAN 之类语言的翻译技术。尽管我们假定读者对这些语言已有一定的基本知识，但为了有所衔接，在第一章，我们仍将复习一下这些语言的基本概念。

在本门课中，我们并不假定以某一特定机器作为目标机器。当需要涉及目标机器指令时，我们将采用一些人所共知的假想指令。因此，在学习这门课之前，读者必须具有计算机基础程序设计的知识。

由于编译程序是一个极其复杂的系统，故在讨论时，我们只好把它肢解开来，一部分一部分地进行研究。因此，在学习过程中应注意前后联系，切忌用静止的、孤立的观点看待问题。

本书中所引用的具体算法大多是用文字描述的，有些是用类似 PASCAL语言表示的。所有这些算法都是原理性和解释性的，而且大多是不完备的（忽略某些次要因素或尚未学到的成分）。因此，并不意味着这些算法可以直接照抄使用。

在着手构造一个编译程序时，需要预先考虑种种具体因素〔诸如，系统功能要求（这种要求常常是多方面的）、硬件设备、软件工具等等〕，特别是，必须估量所有这些因素对编译程序构造的影响。虽然这些都是工程实现时应予考虑的细节，但因篇幅所限，不可能涉及太多。

后面，在复习高级语言的基本概念之后，我们将按照 0.2 节中所说的编译过程的各基本阶段，逐步介绍编译程序的构造方法和技术。

参 考 文 献

〔1〕 Aho, A. V. and Ullman, J. D., Principles of Compiler Design, Addison-Wesley, 1977.

〔2〕 格里斯，数字计算机的编译程序构造，科学出版社，1976。

第一章　高级程序语言概述

本章概述高级程序语言的结构和某些主要的共同特征。

高级程序语言是用来交流算法和计算机实现这两重目的的。现在已有数百种高级语言，它们在应用上各有不同的侧重面。例如，FORTRAN 宜于数值计算，COBOL 便于事务处理，而 SNOBOL 则更利于字符串加工（符号处理）。与机器语言或汇编语言相比，高级语言有许多明显的优点。

高级语言一般较接近于数学语言和自然语言，因此比较直观、自然和易于理解。高级语言程序比较易读、易写，易于交流、出版和存档。由于易于理解，故有利于防止程序出错和便于验证其正确性，一旦发现错误也易于修改。使用高级语言编写程序比用机器语言或汇编语言所费的劳动代价要低得多。例如用 FORTRAN 语言写程序的效率比用机器语言要高十倍以上。由于高级语言一般都是独立于机器的，且多数有标准文本，因此，只要按标准办事，同一程序也可在许多不同的机器上执行。当环境改变时，用户无需针对具体机器重新编制程序。

在软件进入到工程化时期的今天，一个好的程序语言应为用户提供较好的模块设计（或层次设计）的手段，使得人们可采用“分而治之”的办法构造出结构清晰、层次分明、易读、易理解且易于保证正确性的程序。

1.1　程序语言的定义

一个程序语言是一个记号系统。如同自然语言一样，程序语言也是由**语法**和**语义**两方面定义的。

1.1.1　语言的词法和语法结构

任何语言程序都可看成是一定字符集（称为**字母表**）上的一字符串（有限序列）。但是，什么样的字符串才算是一个**合式**的程序呢？所谓一个语言的**语法**是指这样的一组规则，用它可以形成和产生一个合式的程序。这些规则的一部分称为**词法规则**，另一部分称为**语法规则**（或**产生规则**）。

例如，字符串 0.5＊X1＋C，通常被看成是常数 0.5、标识符X1和C，以及算符＊和＋所组成的一个表达式。其中常数`0.5′，标识符`X1′和`C′，算符`＊′和`＋′称为语言的**单词符号**，而表达式`0.5＊X1＋C′称为语言的一个**语法范畴**。

语言的单词符号是由词法规则所确定的。词法规则规定了字母表中哪样的字符串是一个单词符号。

一个程序语言只使用一个有限字符集作为字母表。例如 FORTRAN 的字母表中含有 26 个大写字母 A，B，C，…，X，Y，Z；10 个数字 0，1，…，9；以及 11 个其它字符：空白，＋，－，＊，／，＝，（，），逗点，圆点和 $。ALGOL 60 的字母表是相

当大的。它除了一般的大小体印刷字符之外，还把基本字（如 **begin**, **real**, **procedure** 等等）也都看成是字母表中的单个字符。尽管 ALGOL 参考语言中把基本字都看成是字符，但在实现中，仍可把它们看成是由通常的印刷字符组合而成的。于是，在实现时它实际上只使用了一个小得多的字母表。

单词符号是语言中具有独立意义的最基本结构。一般程序语言中的单词符号包括有常数（如 0.5）、标识符（如 X1）、基本字（如 INTEGER）、算符（如 * *）和界符（如逗点、分号等）。

语言的语法规则规定了如何从单词符号组成更大的结构（即语法范畴），如表达式、语句、分程序、过程、程序等等。例如，单词符号串 0.5 * X1 + C，在不同的语言中它可能代表不同的表达式。如在 ALGOL 中它代表 (0.5 * X1) + C，而在 APL 中它则代表 0.5 * (X1 + C)。因为这两种语言的语法规则不同。

语法范畴比单词符号具有更丰富的意义。如何理解和定义它们的意义即是语言的语义问题。

某些程序语言要求程序的书写服从一定的格式，如 FORTRAN，所有语句都必需写在输入卡片的一定位置上。这种要求增加了词法分析的复杂性。现在多数语言倾向于使用自由格式书写法，容许程序员随自己的意愿编排程序格式。这既便于阅读，又可以回避因书写格式不正确而造成的错误。

空白字符是另一个值得注意的问题。有些语言规定，空白字符除了在文字常数中的出现之外，在别的任何地方的出现都是没有意义的。在这种情况下，空白字符可用于编排程序格式，但增加了词法分析的麻烦。在某些语言中，空白字符用作间隔符。它们的出现决定了单词符号的划分。

如何描述一个程序语言的词法规则和语法规则呢？词法规则的描述是比较容易的。有限自动机是一种描述词法规则的很好的理论。描述语法规则一般是很不容易的。但就现今多数的程序语言来说，上下文无关文法仍是一种可取的有效工具。在本书中，有限自动机和上下文无关文法是我们讨论词法分析和语法分析的主要理论基础。

1.1.2 语义

一旦有了一个合式的程序时，我们又如何阐明它的意义（或功能）呢？这就是所谓语义问题。例如，许多高级语言（如 ALGOL 和 PASCAL）都具有如下形式的语句，

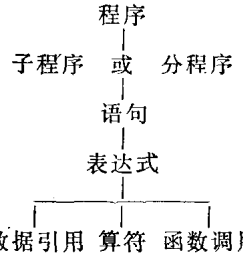
$$\textbf{for}\ \ i := E_1\ \ \textbf{step}\ \ E_2\ \ \textbf{until}\ \ E_3\ \ \textbf{do}\ \ S$$

但其含义各有不同。对于编译来说，只有了解程序的语义，我们才知道应把它翻译成什么样的目标指令代码。

所谓一个语言的**语义**是指这样的一组规则，使用它可以定义一个程序的意义。这些规则称为**语义规则**。阐明语义要比阐明语法难得多。现在还没有一种公认的形式系统，借助于它可以自动地构造出正确的编译程序。在本书中我们仍沿用一些周知的术语来阐明语义。下面我们将用这些基本术语来综述现今程序语言的某些重要概念。

一个程序语言的基本功能是描述**数据**和对数据的**运算**。所谓一个**程序**，从本质上来说是描述一定数据的处理过程。在现今的程序语言中，一个程序大体上可视为下面所示的层

次结构：



自上而下看上列层次结构：顶端是程序本身，它是一个完整的执行单位。一个程序通常是由若干个**子程序**或**分程序**组成的，它们常常含有自己的数据（**局部名**）。子程序或分程序是由**语句**组成的。而组成语句的成分则是各种类型的**表达式**。表达式是描述数据运算的基本结构，它通常含有**数据引用**、**算符**和**函数调用**。

自下而上看上列层次结构：我们希望通过对下层成分的理解来掌握上层成分，从而掌握整个程序。我们将从下层成分开始，综述程序语言各层次的结构和意义。

程序语言的每个组成成分都有（抽象的）**逻辑**和计算机**实现**两方面的意义。当从数学上考虑每个组成成分时，我们注重它的逻辑意义。当从计算机这个角度来看时，我们注重它在机内的表示和实现的可能性与效率。例如，一个表示实数的名字，从逻辑上说，可以看成是一个变量或一个可用于保存一实数的场所；从计算机实现上说，可看成是一个或若干个相继的存贮单元，这些单元的每位都有特殊的解释（如符号位、阶码和尾数），它们能表示一个一定大小和精度的数值。

1.2 初等类型数据

一个程序语言必须提供一定的初等类型数据成分，并定义对于这些数据成分的运算。有些语言还提供了由初等数据构造复杂数据的手段。不同的语言含有不同的初等数据成分。常见的初等数据类型有：

1．**数值数据** 如整数、实数、复数以及这些类型的双长（或多倍长）精度数。

2．**逻辑数据** 多数语言有逻辑型（布尔型）数据，有些甚至有位串型数据。对它们可施行逻辑运算（∧、∨、¬等等）。

3．**字符数据** 有些语言容许有字符型或字符串型的数据，这对于符号处理是必须的。

4．**指示器** 指示器是这样一种类型的数据，它们的值是指另一些数据。尽管语法上可能不尽相同，但一般的意义是，假定P是一个指示器，P：＝**addr**(X) 意味着P将指向X，或者说，P的值将是变量X的地址。有些语言中用P↑表示指示器P的内容。在这种情况下，如令P↑：＝0.3，则意味着X的值为0.3。

1.2.1 标识符和名字

程序语言所涉及的对象不外是数据、函数和过程。对于每个这种对象，程序员通常都用一个能反映它的本质的、有助于记忆的**名字**来表示和称呼它。例如，常常可以看到人们用WEIGHT来表示一个代表重量的实型数据，用INNERPRODUCT表示一个求内积的过程。在程序语言中各种名字都是用**标识符**表示的。所谓标识符系指由字母或数字组成的

但以字母为开头的一个字符串。

虽然名字和标识符在形式上往往难于区分，但这两个概念是有本质区别的。例如，对于'PI'，我们有时说它是一个名字，有时又说它是一个标识符。标识符是一个没有意义的字符序列，但名字却有明确的意义和属性。作为标识符的PI，无非是两个字母的并置，但作为名字PI，常常被用来代表圆周率。在高级语言中常有“局部名”、“全局名”之称，但却没有“局部标识符”、“全局标识符”之说。

用计算机术语来说，每个名字可看成是代表一个抽象的存贮单元，这个单元可含一位、一字节、一字或相继的许多个字。而这个单元的内容则认为是此名字（在某一时刻）的**值**。名字的值就是它所表示的一个具体对象。仅把名字看成代表一定的存贮单元还是不够的，我们还必须同时指出它的**属性**。如果不指出名字的属性，它的值就无法理解。例如，设一个名字代表一个32位的存贮单元，如果不指明属性，那么我们就不知道此单元的内容代表什么，即不知道其内容是一个整数、一个实数还是一个布尔值。名字的属性通常是由说明语句给出的。

有些名字似乎没有通常意义的值，例如过程名就是如此。但我们可以设想过程名具有某种代表输入-输出关系的“值”。

注意，在许多程序语言中，同一标识符在程序中的不同地点（如不同分程序）可用来代表不同的名字。在程序运行时，同一个名字在不同的时间也可能代表不同的存贮单元（在递归的情形下）。反之，同一个存贮单元也可能有好几个不同的名字(如FORTRAN中出现在EQUIVALENCE和COMMON语句里的名字)。

1.2.2 名字的属性和说明

一个名字的属性包括**类型**和**作用域**。名字的类型决定了它能具有什么样的值，值在计算机内的表示方式，以及对它能施加什么运算。名字的作用域规定了它的值的存在范围。例如，一个ALGOL名的作用域是那个包含此名的说明的最小分程序（或过程），只有当这个分程序（或过程）运行时此名字才有对应的存贮单元。

在多数的程序语言中，名字的性质是用说明句明确规定的。例如在ALGOL中，说明句

```
real  X, Y
```

规定了名字X，Y代表实型（简单）变量。即，X和Y各对应有一个标准长度的存贮单元，其内容是浮点数，可对它们进行各种算术运算。

在某些语言中，名字的性质有时容许是隐约定的。例如在FORTRAN中，对未经说明句明显说明的名字，凡以I，J，…，N为首者均认为是代表整型的，否则为实型的。

某些语言既没有说明句也没有隐约定，如APL就是这样。在这种语言中，同一标识符在某一行中可能代表一个整型变量，而在另一行中则代表一个实型数组。因此，名字的性质只能在程序运行时“动态”地确定。也就是，“走到哪里，是什么，算什么”。

如果一个名字的性质是通过说明句或隐约定规则而定义的，则称这种性质是“静态”确定的。如果名字的性质只有在程序运行时才能知道，则称这种性质是“动态”确定的（我们以后常用“静态”和“动态”两词。凡编译时可以确定的东西称为“静态”的；凡必须推迟到

程序运行时才能确定的东西称为“动态”的)。

对于具有静态性质的名字，编译时应对它们引用的合法性进行检查。例如，假定I是整型变量，X是实型变量，混合加法运算I＋X在FORTRAN中是不允许的●，而在ALGOL中则是认可的，但必须预先产生把I转换成实型量的代码。

对于具有动态性质的名字，应在程序运行时收集和确定它们的性质，并进行必要的类型转换。名字的动态性质对于用户来说是方便的，但对计算机实现来说则其效率甚低。

1.3 数据结构

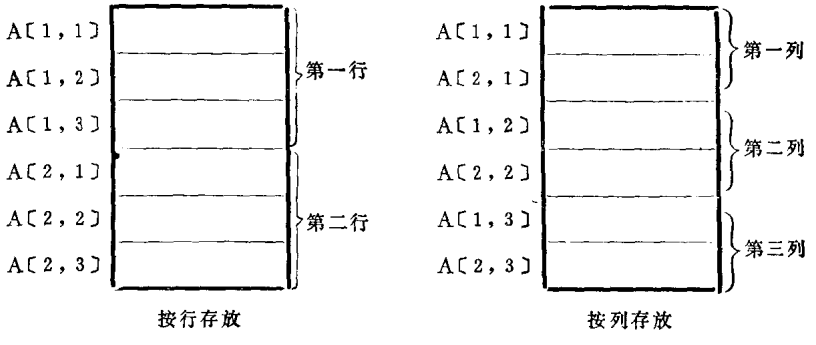
许多程序语言提供了一种可从初级数据定义复杂（高级）数据的手段。在本节，我们将概述几种常见的定义方式。

1.3.1 数组

从逻辑上说，一个**数组**是由同一类型数据所组成的某种n维矩形结构。沿着每一维的距离称为一个**下标**。每维的下标只能在该维的上、下限之内变动。数组的每个元素是矩形结构中的一个点，它的位置可通过给出每维的下标来确定。在FORTRAN中最多只容许三维的数组，并且每维的下限都是1。在其它语言中，数组的维数和上、下限可以是任意的（当然，在计算机实现时必须加以具体的限制)。

数组的每个元素(也称**下标变量**)是由数组名连同各维的下标值命名的，如A〔$i_1, i_2, \cdots, i_n$〕。根据数组的类型，每个数组元素在计算机中占有同样大小的存贮空间。如果一个数组所需的存贮空间的大小在编译时就已知道，则称此数组是一个**确定**数组；否则，称为**可变**数组。

数组的存贮表示有多种形式，最简单的一种是把整个数组**按行**（或**按列**）存放在一片连续存贮区中。例如，若A是一个2×3的二维数组，每个元素占一个机器字，那么，所谓按行和按列的存贮方式分别如下所示：



一般而言，假定对一个n维数组附上一个n位数码管显示器，每个管代表一个下标，每管显示的值在相应维的下限与上限之间变动。所谓按行存放意味着，当从数组的第一个元素开始扫描整个数组时，越是后面的下标（数码管）变化得越快。按列存放意味着越是前面的下标变化得越快。

有些程序语言，如FORTRAN，要求以列为序存放数组；另一些，如PL/1，要求以

● 指ANSI FORTRAN 66而不是现今的FORTRAN 77，下同。