程序设计语言结构

董 浩
安淑芝 编

哈尔滨船舶工程学院出版社

程序设计语言结构

董 浩 安淑芝 编

哈尔滨船舶工程学院出版社

(黑)新登字第9号

内 容 简 介

本书共分九章，主要对程序设计语言中的变量、存贮管理、函数与过程调用等问题进行了讨论，并从语法和语义结合的角度，介绍了 FORTRAN、PASCAL、C 语言、COBOL、LISP、prolog 等常用程序设计语言的数据类型与结构特点，以便读者对各种语言的特点和使用有进一步的了解，并能较容易地自学其它程序设计语言。

本书可作为高等院校工科高年级大学生或研究生的教材，也可作为使用计算机的各种工程技术人员的参考用书。

程序设计语言结构

董 浩 安淑芝 编

•

哈尔滨船舶工程学院出版社出版
新华书店首都发行所发行
绥棱县印刷厂印刷

•

开本787×1092 1/16 印张12.75 字数 286千字
1992年10月第1版 1992年10月第1次印刷
印数：1—5000册
ISBN 7-81007-203-X/TP·10
定价：3.35元

前　言

当前，几乎所有的非计算机专业的理工科大学生都要学习计算机课程，而在大多数情况下，所谓的计算机课程实际就是一种程序设计语言，如BASIC程序设计。在计算机技术高速发展而且广泛普及应用的今天，这显然是不够的。就从使用计算机的角度而言，至少还应进一步了解一些关于语言和操作系统的知识，才可以比较自如地和计算机打交道。对于有些专业，还可以学一些文字编辑、数据库以至硬件方面的知识。

本书是作为非计算机专业计算机程序设计课程的一门后续课程而编写的教材。目的是使学生在学习了一种程序设计语言之后，能对计算机程序设计语言的工作原理有一些进一步的了解，以便更好地使用学过的语言。同时，对现在较流行的若干种程序设计语言有一个更广泛的了解，以便在不同的应用环境下能较为合理地选择使用不同的语言。有了这样的知识背景，这门课程企图达到的第三个目的就是使学生今后在学习一种新的程序设计语言时，具有较强的自学能力。

本书的前三章是属于语言方面的一些基本知识。对于只学过BASIC语言的学生，学起来会感到吃力一些，但对于学过其它语言的学生，接受前三章的内容问题是不大的。本书的后六章从语法和语义的角度介绍了几种不同风格和特点的程序设计语言。在课堂上当然不一定都要讲，而且还要看上机练习的条件。一般讲三至四种语言，并适当上机练习，就可以收到较好的效果。

本书也可供计算机专业但不是这个专业方向的学生选择或参考。

本书第一、二、三章由董浩编写，其余部分由安淑芝编写。哈尔滨工业大学李光汉教授详细地审阅了全书，并提出了许多宝贵意见。在编写过程中，得到了惯性导航及仪器教材编审小组的大力支持和具体帮助。在此一并表示衷心感谢。

由于编者水平有限，时间仓促，书中难免存在错误和不当之处，恳请读者提出宝贵意见。

编　者

目　录

第一章 基本概念

§1 引 言

现在，学习和使用计算机的已经远远不只是计算机专业的人员了。各行各业的人员都开始接触、使用计算机。在大学里，几乎所有理工科专业的学生都要学习、使用计算机，甚至一部分文科专业的学生也要学一点。不同的人，他们具有不同的基础，带着不同的要求与目标，当然学习的深度、广度也不同。对于那些需要在计算机上开发自己的软件，或需要用计算机完成自己特定的数值计算或信息处理任务的人，学习一种或几种程序设计语言应该是起码的要求。

当一个初学者坐在计算机面前开始和计算机对话时，他也许会开始理解他所面对的绝不是一台在硬件含意上的计算机，而是一台硬件和软件结合在一起的计算机系统。每一套这样的系统一般都为用户提供若干种可供选择使用的程序设计语言。开始接触计算机程序设计语言的人会有两个突出的感觉：一是死，语法的规定非常死，一个标点符号都不能错；二是繁，每个语句有几种不同的用法，都要用心记住，并作过练习，学到最后一条语句，笔记也快记了一大本了。然而，这种感觉只是初学阶段如此。当你用熟了一种语言之后，就会发现那些语句其实是很简单的，要比我们生活中的语言简单得多。如果再进一步，你学习了不只一种程序设计语言，而且进行一下比较的话，你会发现它们之间实际是有很多共同之处的，而对它们互相区别的特点也就会有更深入的了解。

对自己所编程序的某一条语句在运行时会产生一些什么样的效果，或者说计算机是如何执行这一条语句的，只有使用汇编语言的人才能最详尽地了解。这是因为汇编语言的语句和计算机的最基本的操作是一一对应的。使用汇编语言的人也必须了解计算机的硬件结构及其工作原理。对于使用高级程序设计语言的人，则不需要如此。但是，如果能对计算机执行语句的过程有一个基本的了解，对正确地掌握和使用程序设计语言无疑是有帮助的。

在这本书的前一部分，我们力图一般地而不是就某种具体的语言来讲解一些原理性的问题，在后一部分则介绍几种常用的程序设计语言。这种介绍既不同于初学者的教科书，也不想代替该语言的使用手册，而是想就每一种语言的特点给读者留下一个概貌。在本书前一部分的基础上，相信再加上一本使用手册，读者就会相当成功地使用所介绍的语言。

§2 计算机模型

为了解释计算机是如何按照我们所编写的程序工作的，我们这里所建立的计算机模

作环境指针，记作 EP，它指向程序的运行环境，即程序可以访问的那一部分存贮单元，以后我们会看到，程序的环境在运行中是可以改变的；第三项是主机中堆栈的状态。虽然当主机执行完程序的一条语句之后，在大多数情况下，堆栈中是空的，但堆栈作为主机自有的存贮手段有时发挥着特殊的作用。

到目前为止，我们的计算机模型仅仅是这样一些简单的概念。在下一节的例子中，这些概念也许会具体化、形象化一些。

§3 程序框图

我们用程序框图来描述无论是哪一种语言所编写的计算机程序。由于程序框图的表示方法非常直观，在很多情况下，几乎不用解释就可以理解其含意，所以，下面我们先用一个例子来说明。

设在国际象棋的棋盘上已知王后的位置，现在要用计算机来求出王后所能控制的那些位置。棋盘用一个8×8的网格来表示，王后的位置用M、N 两个坐标值表示。在计算机的输出网格上，用字母“Q”表示王后的位置，用符号“*”表示王后所控制的位置，符号“-”表示王后不能控制的位置，如图 1.2 所示。王后所能控制的位置包括与王后处于同一横行、同一竖行和同一斜行上的所有位置。

现在，我们采用自上而下逐步进行任务分解的方法来设计所要求的程序。在最高层次上看这个程序，如图 1.3 所示。

I \ J	1	2	3	4	5	6	7	8
1	-	*	-	*	-	-	-	-
2	*	*	*	-	-	-	-	-
3	*	Q	*	*	*	*	*	*
4	*	*	*	-	-	-	-	-
5	-	*	-	*	-	-	-	-
6	-	*	-	-	*	-	-	-
7	-	*	-	-	-	*	-	-
8	-	*	-	-	-	-	*	*

图 1.2 棋盘网格图

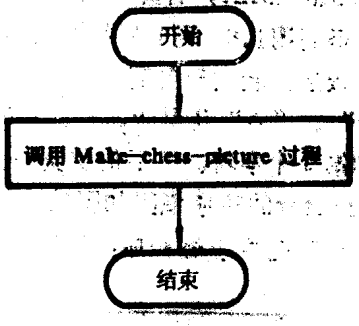

图 1.3 TOP主程序框图

图1.3中的过程Make-chess-picture 假设要反复地进行，每次输入一组坐标 M、N，输出一幅所要求的棋盘网格。但如果数据M、N不满足1≤M≤8,1≤N≤8 的条件，这种反复将会被终止。由此过程Make-chess-picture可以进一步分解为图1.4所示。

图1.4中的框1表示输入数据M、N。框2为过程 Picture-with-ok-data，它又可以进一步分解为图1.5所示。在对数据 M、N 进行合理性检验之后，应该计算并输出一幅

棋盘格，并输出一个空行，以和下一幅棋盘格拉开间隔，这个过程叫做 Picture-and-a-blank-line。

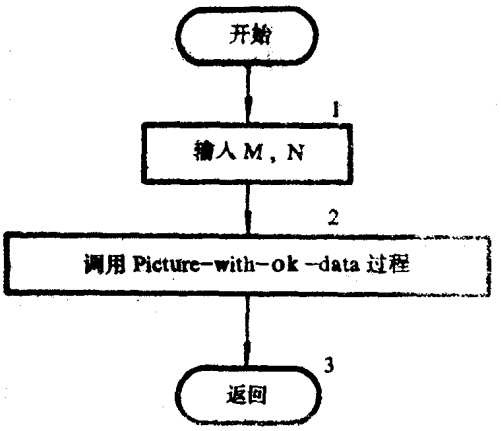

图 1.4 Make-ckess-picture 程序框图

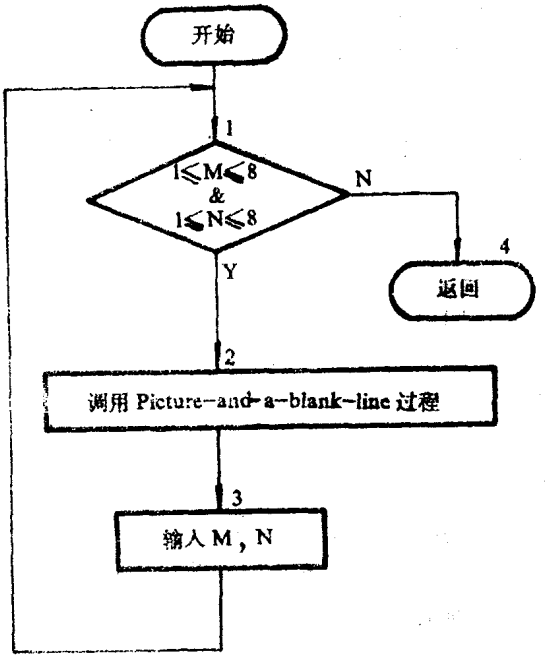

图 1.5 picture-with-ok-data 程序框图

下面就要考虑如何输出棋盘格的问题了。最容易想到的办法就是一行一行地输出。我们可以设立一个变量叫CHESS，它是由8个字符组成的数组。每次计算好一行的值以后，就通过CHESS输出，输出8次即可完成一个棋盘网格。如果把输出一行的任务叫作One-Line，计算一行中每一个位置是否被王后控制的任务叫作 One-character，则可得到图1.6、图1.7两个框图。图1.6中框3表示输出一个空行。

最后，我们已经到达了任务分解的最后一步，即计算某一个具体位置上应输出“Q”、“*”还是“-”。这时，需要计算的位置是在第I行、第J列，对应 CHESS 里的第 J 个字符。王后的位置是在第 M 行、第 N 列。这个计算当然并不复杂。One-character的程序框图如图1.8所示。

为了完成上述程序，这里我们把它的标识符表列出，如表1.1所示。

表 1.1 标 识 符 表

标 识 符	说 明
M	整 数
N	整 数
I	整 数
J	整 数
CHESS	由8个单个字符组成的数组

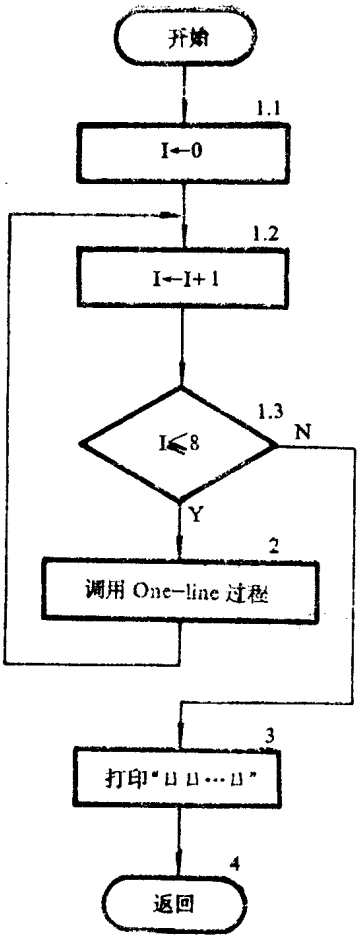

图 1.6 Picture-and-blank-line 程序框图

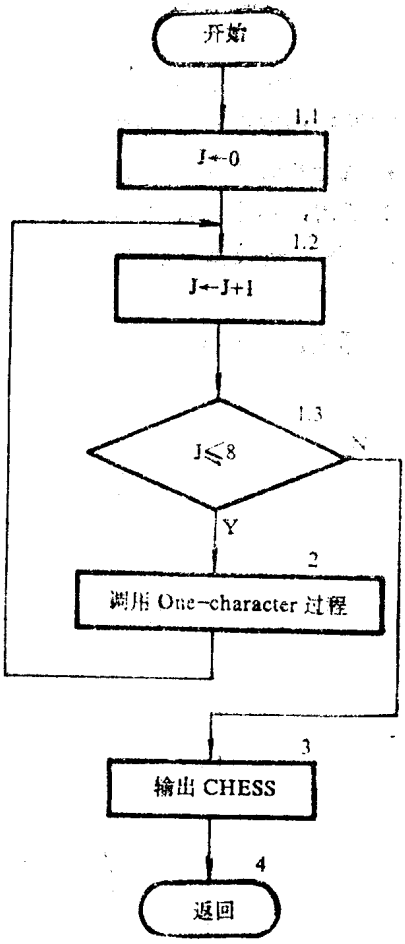

图 1.7 One-line程序框图

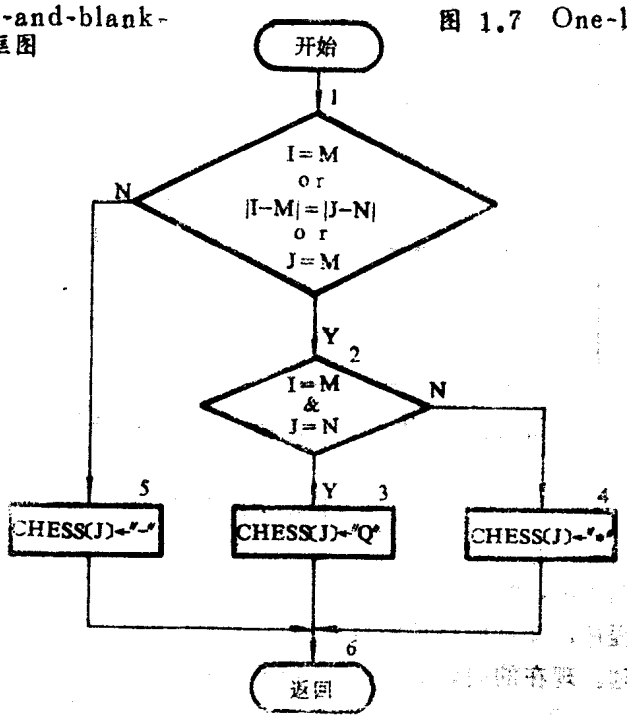

图 1.8 One-character程序框图

在上一节讲到计算机模型的时候，我们已经介绍了程序运行环境和主机状态的概念。如果在程序运行中间的某一点上，我们把各存贮单元中存贮的值和主机的状态描述出来，就好象计算机运行到一半突然停止运行而保持状态不变一样，我们就得到所谓的瞬时状态图，以下即简称为状态图。图1.9和图1.10分别画出了程序开始运行前和读入M、N的第一组数据之后的状态图。图中的MC表示主机，MC向左的箭头即为指令指针，指向下一步要执行的语句。环境指针在这里是隐含的，因为目前这个例子中只有一个运行环境。因堆栈目前暂不起作用，故省略了。

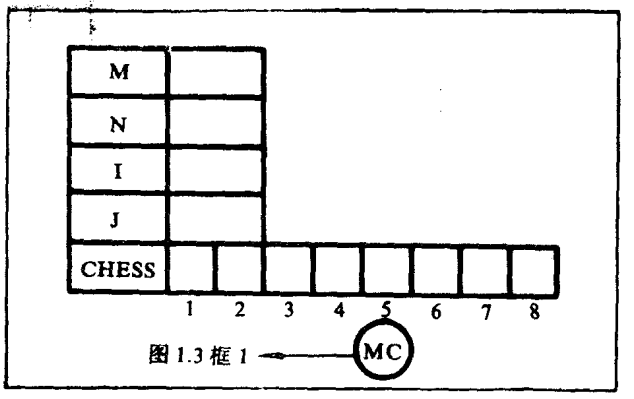

图 1.9　程序运行前的状态图

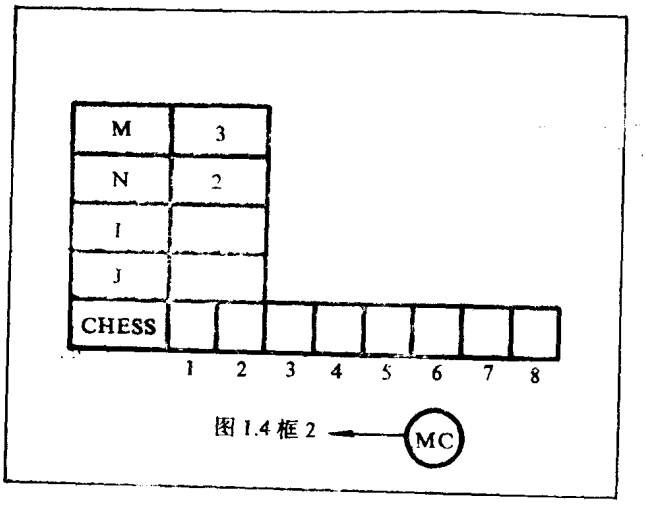

图 1.10　读入第一组M、N后的状态图

§ 4　过程和环境

在上一节的例子中，我们假设读者已经熟悉过程子程序的概念和用法，因而，完全可以看懂所编的程序，知道程序是按照怎样的顺序执行的。这一节也不想讨论过程子程序本身的工作原理。现在的问题是，当一个程序被分解为若干过程子程序时，如何理解程序运行的环境，当一个子程序被调用或从子程序返回时，运行环境是否变化和如何变化。

一种可能的情况是，整个程序，包括所有用到的子程序，都使用同一个完整的标识符表，就如同上一节中所举的例子那样，并且所有变量的存贮单元都是在程序运行之前由主机一起分配的。在这种情况下，所有的程序或子程序都是在同一个存贮环境中运行，而且，这个运行环境一直存在于整个程序运行的始终。到目前为止，我们所介绍的概念和所举的例子都是和这种处理方法相吻合的。这种处理方法比较容易理解，也容易实现。BASIC和COBOL语言就是使用这种方法建立运行环境的。

另一种可能的情况是与此相反的，即主程序和每一个子程序都分别具有各自不同的运行环境，这就意味着：

(1) 每一个子程序必须有一个独立的标识符表；

(2) 当主机在某一个环境中运行时，它无法直接访问另外环境中的变量；

(3) 每一次调用子程序或从子程序返回时，都伴随着从一个运行环境改变到另一个环境的变化。

广泛使用的FORTRAN语言就属于这种情况。在程序运行之前，主机根据各个子程序的标识符表分配存贮单元，即在建立主程序的运行环境的同时，也为每一个子程序分别建立起各自的运行环境。

以上介绍的两种情况的例子都是在程序运行之前分配存贮单元的，在程序运行的过程当中，变量和存贮单元的对应关系将保持不变，直到程序全部运行完了。因此，这种分配存贮器的方法又叫静态存贮分配。还有一些程序设计语言，如PASCAL、LISP，虽然它们也属于上面介绍的第二种情况，即每个子程序有自己单独的运行环境，但它们在程序运行时，子程序的存贮单元不再预先进行分配。也就是说，每当一个子程序被调用时，主机都要根据它的标识符表临时为它建立起一个运行环境；当从这个子程序中返回时，它的运行环境将被取消，被分配的存贮单元也将重新处于等待分配的“机动”状态。如果在一个子程序的运行中间，这同一个子程序再一次被调用（递归调用）时，主机将按照它的标识符表再一次为它分配内存，即建立起同一个子程序的第二个运行环境。这种处理方法叫作动态存贮分配。

下面，我们还是用上一节的例子来看一下动态存贮分配的情况。为写起来方便，用如下的缩写来代替对应的过程名称：

MCP：Make-chess-picture；

POKD：Picture-with-ok-data；

PBL：Picture-and-a-blank-line；

Line：One-Line；

One-char：One-character。

各部分程序的标识符表列于表1.2。

表 1.2 程序标识符表

表 1.2.1 主程序标识符表

标识符	说明

表1..2.2 MCP标识符表

标识符	说明
M, N	整数

表 1.2.3 POKD标识符表

标识符	说明
M	MCP中的M
N	MCP中的N

表 1.2.4 PBL标识符表

标识符	说明
I	整数

表 1.2.5 Line标识符表

标识符	说明
J	整数
CHESS	由8个单个字符组成的数组

表 1.2.6 One-char标识符表

标识符	说明
J	Line中的J
M	MCP中的M
N	MCP中的N
I	PBL中的I
CHESS	Line中的CHESS

从表1.2中可以看到，每一个过程具有一个单独的标识符表。在主程序中，因为它不需要访问任何变量，因而标识符表中是空的，也就是说，在进入主程序时，主机不需要为它分配内存。这里需要说明的是，在主程序的运行环境中，虽然不需要为它分配存贮单元，但此时主机必须有办法知道存在一个过程子程序，叫作 Make-chess-picture，并在调用它时转到它的入口。类似的情况我们就不再另外说明了，因为这对我们要讨论的问题并无什么影响。在其它的标识符表中，列出了在每一个过程子程序运行时需要访问的变量的名称。当主机进入MCP过程时，根据它的标识符表，主机为它建立的运行环境中将包括两个存贮单元，分别存放整数M和N。当主机继续执行程序，要进入POKD过程时，碰到一个需要特别注意的问题。在POKD过程的标识符表中，也列出了两个变量M和N。但是从程序的内容上显然可以看出，这里的M和MCP过程中的M是同一个变量，这里的N和MCP过程中的N也是同一变量。假如在POKD的标识符表中，M、N也被说明为整数，主机再一次地为它们分配存贮单元作为POKD过程的运行环境的话，程序的运行结果显然是不正确的。正因为如此，在POKD的标识符表中，必须明确地说明这里的M、N和MCP过程中的M、N变量之间的关系，而不能用整数类型加以说明。这样，当主机进入POKD过程时，在POKD过程的运行环境中，将保存着分别指向MCP运行环境中变量M、N的指针（地址），而不是存贮M、N的值的存贮单元。这种情况在下面的状态图中将会看得更清楚。

图1.11 所示的状态图是程序运行到One-char过程子程序中的一个状态。此时，One-char已是第二次被调用了（注意到J的值为2），而主机即将执行图1.8程序中框2的操作。从这个状态图中可以看出：

(1) 每一个过程子程序分别具有自己的独立的运行环境。在每一个运行环境的状态图的外面，我们用一个框线把它围了起来，并标明了这是哪一个过程、第几次调用所建立的运行环境。例如，One-char^{2}是表示One-char过程的运行环境，上标2则表示这个

环境是第二次调用时建立的。

(2) 在每一个过程子程序的运行环境中，有一个自动分配的存贮单元，称为返回标志（retlab），存放着从该过程返回原调用程序时应回到的环境和下一步操作，即EP、IP指针对。

(3) 主机MC正在 One-char[2] 环境中运行。主机所画的位置就表示了主机的环境指针。

(4) 当主机需要访问别的环境中的变量时，例如要访问变量M，都是通过One-char环境中对应的指针间接实现的（在One-char环境中并不真正存放着M的值）。

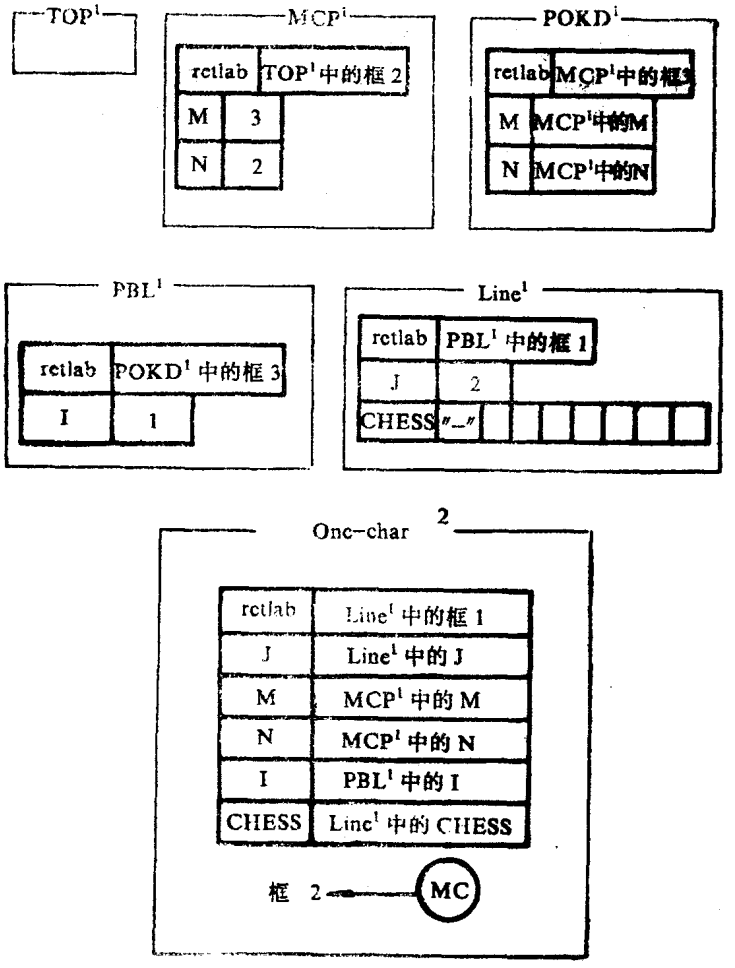

图 1.11 运行到One-char过程子程序中的状态图

§ 5 全局变量与局部变量

在上一节的例子中，我们看到在每一个过程子程序的标识符表中实际上存在着两种情况。一种情况是，在标识符表中列出了变量的名称和该变量的类型（从而说明了该变

量对应的存贮单元的容量）。当主机进入过程子程序时，对于这样说明的变量将按照标识符表的说明对变量分配相应的内存。这种变量相对于对其进行说明的子程序来说就称为局部变量。例如，整形变量M、N对过程MCP来说就是局部变量。同样，整型量I对PBL过程、字符串CHESS对Line过程都是局部变量。另一种情况是，变量的类型已在别的过程中进行了说明，并在相应的运行环境中分配了内存，在本过程的标识符表中，只是说明了这个变量被定义的过程和相应的变量名称，如POKD过程中对M、N的说明，这种变量对本过程就称为全局变量。同样的，M、N、I、J、CHESS对过程Onechar来说都是全局变量。

一般地说，如果变量A是在某一过程Q的执行过程中被建立起来的，然而，在另一过程P的执行过程中，主机仍可以访问变量A，则称变量A为过程P的全局变量，为过程Q的局部变量。显然，全局变量和局部变量都是相对的。对某一个过程来说是局部变量，对另一个过程来说可能是全局变量；反之，对一个过程来说是全局变量，对另一个过程（或主程序）来说就是局部变量。事实上，全局变量是把信息或数据从一个过程传递到另一个过程的最直接了当的办法。

以上，我们介绍的计算机模型、程序框图、过程和环境以及全局变量和局部变量，是我们理解程序语言的工作原理的几个最基本的概念。基于这些概念，下面我们重点介绍过程之间的信息传递问题。

第二章　过程之间的数据传递

§1　形式参数与实际参数

在上一章中，我们已经提到全局变量是把信息或数据从一个过程传递到另一个过程的最直接了当的方法。这样就产生了一个问题，即如何在过程之间传递或共享有关的数据或信息。

毫无疑问，全局变量是可以实现这一目标的。从上一章的例子中，我们可以这样理解：所谓全局变量是指在某一过程中使用的一些变量，它们与外层程序中的对应变量使用相同的名称和存贮单元。这里所说的外层程序是可以调用该过程的另一个过程或主程序，这个外层程序的运行环境就是该过程的调用环境。

虽然全局变量可以实现过程间的数据传递，但是，当我们设计一个过程子程序的时候，必须预先知道在调用环境中这些变量的名称，从而才能在过程子程序中作为全局变量使用这些变量。而且，当这些变量的名称一旦确定下来之后，就不能再改变，这样就限制了一个过程子程序在不同的调用环境中被调用。

参数传递方法就可以解决上述的问题。使用这种方法，当我们设计一个过程子程序的时候，并不需要知道调用环境中变量的名称，而是用假想的名称，称为形式参数，来占据相应的位置。在调用这个过程的时候，由外层程序提供的需要传递给该过程的数据或信息，称为实际参数，将逐个对应地替代到相应的位置上，去完成过程子程序的操作。这样，就解决了使用全局变量传递数据所存在的问题，使子程序的设计和调用更为灵活。在实际使用中，不光是变量可以作为实际参数，在有些条件下，表达式也可以作为实际参数。

现在，我们还是来看上一章举过的棋盘格的例子。在打印出的棋盘格中，我们所选择的打印符号“*”和“—”实际上与解题的过程及运算的结果是没有什么关系的。我们可以用两个变量U和V来代表所打印的字符，并在每一次计算整个棋盘格之前输入给程序。在进入过程One-char时，使用参数传递方法把这两个字符的信息提供给One-char过程。这样就得到如图2.1所示的程序框图（此程序在上章的程序的基础上对一些过程进行了合并）。

在图2.1中我们看到，过程One-char中用了二个形式参数X、Y，并分别在框4、框5中用这二个形式参数参加了操作。而在过程Make-picture-and-line（简写为MPL）中，使用实际参数U、V对过程One-char进行调用。这就决定了在One-char的实际运行中，所有X的位置将由U来代替，所有Y的位置将由V来代替，这种对应关系是按次序排列来决定的。

表2.1示出了过程One-char的标识符表。为了对参数和变量分别加以说明，在以下的标识符表均增加了类别一项。