软件基础实验指导

徐士良　　朱明方

清华大学出版社

目　　录

第一章　绪论 …… 1

第二章　基本数据结构 …… 6

实验 2.1　线性表的插入与删除 …… 6

实验 2.2　迷宫问题 …… 8

实验 2.3　求解约瑟夫问题 …… 12

实验 2.4　多项式的表示与相加 …… 13

实验 2.5　稀疏矩阵的表示与相乘 …… 15

实验 2.6　二叉树 …… 17

实验 2.7　递归函数的计算 …… 19

实验 2.8　无括号的简单表达式的处理 …… 21

实验 2.9　有括号的简单表达式的处理 …… 23

实验 2.10　拓扑分类 …… 25

实验 2.11　求解皇后问题 …… 28

实验 2.12　最短距离问题 …… 30

实验练习 …… 32

第三章　查找与排序 …… 33

实验 3.1　寻找最大项与次大项 …… 33

实验 3.2　二叉排序树与查找 …… 35

实验 3.3　排序方法的比较 …… 37

实验 3.4　线性链表的插入排序 …… 39

实验 3.5　字符串匹配的 KMP 算法 …… 40

实验 3.6　线性 Hash 表与随机 Hash 表 …… 43

实验 3.7　指标 Hash 表 …… 44

实验练习 …… 45

第四章　操作系统 …… 47

实验 4.1　MS-DOS 操作系统 …… 47

实验 4.2　模拟文件系统 …… 49

实验 4.3　有效地址与物理地址的映射 …… 51

实验 4.4　存储分配方法比较 …… 54

实验 4.5　SPOOLing 技术 …… 56

实验 4.6　二级文件目录的管理 …… 59

实验 4.7　生产者-消费者问题 …… 62

实验练习 …… 65

第五章　数据库系统 …… 67
实验 5.1　数据库文件的建立和最简单的使用 …… 67
实验 5.2　数据库的数据修改和报表输出 …… 69
实验 5.3　数据的转移、数据结构的修改和多重数据库操作 …… 71
实验 5.4　设计一个简单的应用程序——测定命令的执行时间 …… 75
实验 5.5　建立一个学生成绩库，并设计一个简单的管理程序 …… 77
实验 5.6　关系运算的应用 …… 79
实验 5.7　一种自动生成程序的方法 …… 80
实验 5.8　书目检索 …… 82
实验练习 …… 85
第六章　解释原理与编译技术 …… 86
实验 6.1　解释程序的模拟 …… 86
实验 6.2　算术常数的识别和翻译 …… 89
实验 6.3　优先矩阵法的编译过程 …… 90
实验 6.4　递归子程序法处理表达式 …… 91
实验 6.5　模拟编辑程序 …… 93
附录 1-1　dBASE II（V 2.3 D）命令一览表 …… 99
附录 1-2　dBASE II SET 命令表 …… 103
附录 2　dBASE II 全屏幕编辑控制键一览表 …… 105
附录 3　出错信息及纠正方法 …… 108
附录 4-1　dBASE III 命令表 …… 112
附录 4-2　dBASE III SET 命令表 …… 118
附录 5　dBASE III 函数一览表 …… 122
附录 6　dBASE III 全屏幕编辑控制键一览表 …… 126
附录 7　dBASE III 与 dBASE II 主要差别比较表 …… 128
参考书目 …… 129

第一章 绪 论

软件基础课程涉及到的内容直接与计算机在各个领域中的应用密切相关。要掌握好与应用有关的计算机软件基础知识，并熟悉其应用背景，必须重视实验这一重要环节。

做一个实验如同解决一个实际问题，必须弄清以下几个问题。

一、实验的目的

实验作为教学的一个重要环节，其目的在于更深入地理解和掌握课程教学中的有关基本内容，在解决实际问题时应用课程中所讲授的基础技术，以提高分析问题、解决问题的能力。一般来说，解决一个实际问题，可以有多种方案，而各种方案中所采用的技术又各不相同。实验也是解决一个实际问题，由于受到教学要求、进度以及实验条件的约束，必须对每一个实验制定目的明确又切实可行的目标。在这个意义上，实验又不同于一般的解决某一个实际问题。例如，为了求解一个问题，既可以采用分支选择语句编制程序，又可以用循环语句编制程序，为了结合关于分支选择结构这一教学内容，达到熟练使用分支选择语句的目的，实验中就需采用分支选择语句。对于同一问题，如果为了比较这两种方法，就要求同时用两种方法编制程序，以便进行比较。因此，当我们做一个实验的时候，必须明确实验的目的，以保证达到课程所指定的基本要求。在第二章数据结构的各实验中，求解一个问题完全可以采用各种数据结构，但作为实验，必须根据实验的目的来制定解决问题的方案。

综上所述，做实验不同于就事论事地解决一个实际问题。实验的目的，通常有以下几种类型。

1. 指定采用某种技术或数据结构，以便通过问题的解决，掌握这种技术或数据结构，以达到基本的教学要求。

2. 求解一个问题可以用多种方案，各方案采用不同的技术或数据结构。实验中要对各种方案进行比较，以便掌握多种技术或数据结构或它们的综合应用。

3. 通过实验综合应用有关的技术或数据结构，并选择最合理、有效的方案。

总之，实验目的是指导实验的总的原则，只有明确目的，才能使实验更有的放矢，以便达到预期的教学要求。

二、实验的内容

实验目的是通过解决一个具体问题来达到的，因此，了解具体的实验内容是做好实验的基础。通常，实验内容将告诉你做什么、完成什么任务或求解什么问题。在了解实验内容的基础上，更重要的是要分析所要解决的问题，要根据实验的目的着重考虑以下几个问题。

1. 为了解决这个问题，可能要用到哪些数据结构，而其中哪些是根据实验目的的要求必须采用的，哪些是可以选择的？

2. 为了提高算法或程序的效率、易读性、可靠性等，需要采用哪些技术？

3. 为了解决这个问题，还要同时解决哪些辅助性的问题？它们与主问题之间的关系是什么？应采用什么样的程序结构将它们有机地组织起来？

4. 采用什么样的输入方式与输出方式最方便、直观？

在考虑以上这些问题时，首先要采用符合实验要求的数据结构或技术，然后再考虑选择其它的数据结构或技术。这些问题决定之后，需要对解决这个问题作出完整的描述，它包括两个方面：一是对解题的步骤给出描述，即写出算法；二是对算法中所采用的数据结构和其它技术给出应有的说明，即写出文档。

从分析问题到最后确定算法这样一个设计过程是一件很细致的工作，应尽量减少失误。在本实验指导书中，对每一个实验给出了比较简略的说明，有的还给出了算法，但这只起一个抛砖引玉的作用，或者说只是给读者一个启发，更详细的设计应由读者自己完成。因此，我们希望读者在解决一个实际问题或做某个实验的时候，始终要认真考虑“给定的算法是否很好”、“还有没有更好的算法”等问题，这样才能收到事半功倍的效果，得到满意的结果。

三、算法和程序是否正确

要得到理想的实验结果，算法的正确性分析(当然算法还有其他方面的分析)是很重要的。因为设计一个正确的算法是得到一个正确程序，直至得到正确结果的前提。算法的正确性分析与程序的调试有着密切的联系，有些原则和方法可能是互相适用的。由于实际的算法是一系列指令(甚至是计算机程序)，因此，为了分析算法的正确性，必须要说明算法中的每一个步骤确实做了我们所希望做的工作。显然，为了从这个方面来说明算法是正确的，首先必须建立一个精确的命题，这个命题用来说明在给定某些输入以后，经过执行算法中的每一个步骤将要产生什么结果，然后再来证明这个命题。下面我们用一个简单的例子来说明这种算法正确性的严格证明。

我们的问题是：用顺序搜索法在一个一维数组中寻找元素值为 X 的下标。我们所采用的方法是将 X 依次分别与数组中的每一个元素进行比较，直到找到了一次匹配或者该数组被检查完为止。如果在数组中没有 X，则以 0 作为其答案。其算法如下。

算法：顺序搜索法

输入：L(1:n), X

输出：j

```
1. j ← 1
2. WHILE (j ≤ n) and (L(j) ≠ X) DO
3. j ← j + 1
4. IF j > n THEN j ← 0
```

上面提到，在证明这个算法的正确性以前，我们要建立一个精确的命题来说明这个算法做了什么样的工作。表面看来，这件工作是很简单的，因为根据题意，如果 L 是一个具有 n 个元素的数组，而且数组中有一个元素等于 X，那么算法将 j 置成数组中元素值等于 X 的下标而结束；如果数组中没有等于 X 的元素，则这个算法将 j 置成 0 而结束。但这样的一种叙述有两个缺点：一是它没有说明当 X 在数组中多次出现时的结果；二是没有指出算法在 n 为何值时工作，如果假设 n 是非负的，当数组为空即 $n = 0$ 时，算法执行的情

况也不太清楚。由此，我们构造一个更精确的命题如下：

给定一个具有 $n(n \geqslant 0)$ 个元素的数组 L，并且给定 X，当 X 在 L 之中时，顺序搜索法将 j 置成 X 在 L 中第一次出现处的下标而结束，否则算法将 j 置成 0 而结束。

为了便于证明，我们将这个命题改写，使之对于算法执行时应该成立的一些条件给出明确的结论。

对于 $1 \leqslant k \leqslant n+1$，如果算法第 k 次地执行上述算法中的第 2 行中的测试，则下列条件成立：

(1) $j = k$，而且 $1 \leqslant i < k$，$L(i) \neq X$

(2) 若 $k \leqslant n$，而且 $L(k) = X$，则算法将在执行第 2 行中的测试和第 4 行以后结束，此时 j 仍然等于 k。

(3) 若 $k = n+1$，则算法将在执行了第 2 行中的测试和第 4 行以后结束，此时 $j = 0$。

采用数学归纳法证明如下：

首先假设 $k = 1$。

从算法的第 1 行可得 $j = k$，且条件 (1) 的第二部分被认为是满足的。

如果 $1 \leqslant n$ 和 $L(1) = X$，则算法中第 2 行的测试不成立，算法进行到第 4 行，这时因为 $j = k \leqslant n$，所以 j 不改变，条件 (2) 满足。

如果 $k = n+1$ (显然，此时 $n = 0$)，则 $j = n+1$，以致算法第 2 行测试不成立，进行到第 4 行，这时 j 被置成 0，条件 (3) 满足。

然后假设对于某个 $k < n+1$，以上三条件成立。

最后考虑 $k+1$ 的情形。

由于条件 (1) 在 k 时成立，而在算法第 $k+1$ 次回到第 2 行之前又执行了一次第 3 行，所以 $j = k+1$；另外，由于条件 (1) 在 k 时成立，说明对于 $1 \leqslant i < k$，$L(i) \neq X$，又由于条件 (2) 对于 k 成立，说明 $L(k) \neq X$，否则，算法已经结束，不会出现 $k+1$ 次，所以对于 $1 \leqslant i < k+1$，$L(i) \neq X$。总之，对于 $k+1$ 次地执行第 2 行的测试，条件 (1) 成立。

如果 $k+1 \leqslant n$ 和 $L(k+1) = X$，则第 2 行的测试不成立，算法进行第 4 行，此时由于 $j = k+1 \leqslant n$，所以 j 不改变，条件 (2) 满足。

如果 $k+1 = n+1$ (显然 $n = k$)，则 $j = n+1$，以致于第 2 行测试不成立，算法进行到第 4 行，此时由于 $j > n$，j 被置成 0，条件 (3) 满足。

由此看出，对于 $k+1$ 也满足上述三条件。

命题得证。

由上述被证明的结论可以看出，算法中的第 2 行测试最多被执行 $n+1$ 次，而且，当且仅当执行了 $n+1$ 次后，才输出 $j = 0$，此时，由条件 (1)，对于 $1 \leqslant i \leqslant n$，$L(i) \neq X$，即 X 不在数组 L 之中；当且仅当 $L(k) = X$ 时，才输出 $j = k$，此时由条件 (1)，对 $1 \leqslant i < k$，$L(i) \neq X$，因此，k 是 X 在数组 L 中第一次出现的下标。由此可知，上述顺序搜索算法是正确的。

显然，这个证明是令人乏味的。我们之所以举这个例子进行算法正确性的严格证明，

一方面是让读者了解一下如何进行算法正确性的严格证明，另一方面也是为了告诉读者，算法的正确性是可以证明的。为了证明算法的正确性，必须进行上例所示的这类工作，但在实际上却很少这样做，因为对于数据结构和控制结构稍微复杂一些的算法程序，要证明它的正确性是一件令人望而生畏的工作。通常，如果一个算法相当短，又比较简单，那我们就采用一些很不正规的、而且是难以描述的方法，以使我们自己确信算法中的每一步确实做了我们希望它做的工作。有时索性用少量的实例来人工模拟这个算法，以检验它的正确性。对于稍微复杂一点的算法，我们可以将它分解，分别验证其中每一个独立部分的算法段。

四、实验是否顺利及结果是否正确合理

实验的最后一个步骤是将预先编制好的程序输入到计算机，然后根据实验要求以及设计的初始数据运行这个程序，最后得到结果。程序调试过程中往往会出现错误，可能是某种数据结构(包括运算)有错，也可能是程序的细节有错，还可能是对计算机的使用操作有错。不管是什么类型的错误，通过实验发现这些错误，并改正这些错误，总结经验教训，将是非常有益的。此外，对结果感到不满意的现象也是经常会发生的，或者是结果本身不合理，或者是结果的输出方式不理想，通过总结，可以进一步修改算法或程序。因此，做完实验后认真写好实验报告，对提高实验技能与解决问题的能力以及达到预定的教学要求是必要的。

读者通过程序设计语言的学习及其实际应用的过程，对于程序设计技术已经有了一个初步的认识和体会。在此特别要提出的是，关于程序设计的风格问题。读者在做实验的过程中要给予充分的注意，这些在软件基础教材或其它有关资料中都有比较详细的介绍，在此不再赘述。

在本书中，我们约定了一种算法描述语言，读者也可以很方便地用自己所熟悉的语言将算法改写成程序。下面简单介绍一下这种算法描述语言的主要细节，读者在阅读本书中的各算法时可以逐步体会它。

1. 标识符

与一般程序设计语言相类似，标识符由字母开头的字母数字串所构成。

2. 运算符

算术运算符有：+、-、*、/；

关系运算符有：=、≠、<、>、<=、>=；

逻辑运算符有：and、or、not。

在表达式中出现的其它一些运算符读者也是不难理解的。

3. 赋值语句

有关赋值的语句有以下三种形式：

$a \leftarrow e$　表示将表达式 e 的值赋给变量 a；

$a \leftarrow b \leftarrow e$　表示将表达式 e 的值同时赋给变量 a 和 b；

$a \rightleftharpoons b$　表示将变量 a 和 b 的值交换。

4. 选择分支结构

选择分支结构的语句有如下形式：

IF C THEN S

或 IF C THEN S_1

ELSE S_2

其中C为条件，S、S_1 与 S_2 为单一的语句。如果 S、S_1、S_2 为复合语句，则要用一对花括弧即{}将它们括起来。即有如下形式：

```
IF C THEN
    {语句 1
        ⋮
     语句 n
    }
ELSE
    {语句1
        ⋮
     语句 m
    }
```

这些语句与程序设计语言中的 IF 语句基本一致，其含义不难理解。

5. 循环结构

循环结构的语句有两种。

一种是 FOR 循环语句，形式为

FOR I = t_1 TO t_2 [BY t_3] DO S

其中 I 为循环控制变量，t_1、t_2 和 t_3 分别表示循环的初值、终值和步长，当 $t_3 = 1$ 时，用方括弧括起来的部分可省略。上述语句中，S 为单一的语句。如果 S 为复合语句，则要用括弧括起来，即

```
FOR I = t1 TO t2 [BY t3] DO
    {语句1
        ⋮
     语句 n
    }
```

另一种是 WHILE 循环语句，形式为

WHILE C DO S

其中C为条件，S 的意义与 FOR 语句相同。在这种循环中，若条件C满足（即其值为真）则执行 S，然后再判断条件C是否满足，直到条件C不满足时退出循环，顺序执行下一语句。

6. 其它

在我们所约定的算法描述语言中，还有可能出现其它语句，而这些语句在程序设计语言中经常遇到，理解不会有困难，例如：INPUT，READ，OUTPUT，RETURN，CALL 等等。有时为了使算法描述得更确切，也偶而用叙述性的语言作为一个语句或一个表达式。

另外，为了节省篇幅，经常将几个语句书写在同一行上，用分号(即;)将它们彼此隔开。

第二章　基本数据结构

为解决给定的问题而设计的程序，其有效性、清晰性和复杂性与在程序中用到的数据的组织方式(即所选取的数据结构)有着密切的关系。本章各实验中提出的问题，可以说只要掌握一种计算机语言就能解决，而不必考虑数据的组织方式。但读者在阅读各实验中的方法说明并做完实验之后就会发现，利用恰当的数据结构，对提高程序的效率是至关重要的。

实验所采用的方法可以是多种多样的，读者在阅读本章所介绍的这些方法说明之前应思考一下：如果采用一般的方法，这个问题应如何解决？根据你所熟悉的数据结构，这个问题又应如何解决？然后再阅读书中给出的方法说明进行比较与分析。我们的目的不单是解决问题本身，更注重通过解决问题掌握各种数据结构在程序设计中的应用。

本章所安排的 12 个基本实验以及后面的实验练习，涉及到数据结构的多个方面，但也不是全部。这些方面主要包括线性表的运算、数组、栈、队列及其链式存储结构在程序设计中的应用。数组本身是一种数据结构的形式。在一般的程序设计语言中，数组的各元素是顺序存储在计算机内存中的，它是一种随机存取的结构。因此，在我们的所有实验中，差不多都用一维数组来模拟计算机的内存空间，并且用多列二维数组来模拟链式分配时所用的可利用空间，这样，二维数组中的每一行就表示了一个结点，每一列分别表示值域与指针域。有时，我们也将值域和指针域分开，用多个一维数组来表示。

实验 2.1　线性表的插入与删除

一、实验目的

掌握线性表在顺序分配下的插入与删除运算，并掌握一种产生随机数的方法。

二、实验内容

1. 产生 1000 个 0 至 999 之间的随机整数，并依产生的次序存入一个数据文件中。

2. 编制一个程序，依次实现以下功能。

(1) 定义一个有序(非递减)线性表，其最大容量为 1000，初始时为空。

(2) 从由 1 产生的数据文件中依次取前 N 个随机整数，陆续插入到此线性表中，并要求在每次插入后保持线性表的有序性。最后将此有序线性表打印输出。

(3) 在由 (2) 产生的线性表中，依在 1 中产生的次序逐个将元素删除，直至表空为止。

3. 以 $N=100$ 及 $N=400$ 分别运行 2 的程序，并比较它们的运行时间。

三、实验步骤和要求

1. 事先编制好实验内容中 1、2 的程序(可参考本实验中的方法说明)，并调试通过。

2. 运行 1 的程序，生成 1000 个 0 至 999 之间的随机整数的数据文件，并打印输出此

数据文件。

3. 以 $N=100$ 运行 2 的程序，并记下运行时间。

4. 以 $N=400$ 运行 2 的程序，并记下运行时间。

5. 整理程序清单及所有结果，写出实验报告。

四、方法说明

1. 随机整数的产生

产生随机数的方法有很多，下面只介绍一种方法。

设 $m=2^{16}$，初值 $y_0=0$，则递推公式

$$y_i = \mathrm{mod}\ (2053y_{i-1}+13849, m)$$

产生 0 至 65535 之间的随机整数。如要产生 0 至 999 之间的随机整数，只需作运算

$$x_i = \mathrm{INT}\ (1000y_i/m)$$

其中 INT 是取整函数。

2. 线性表的插入与删除

在本实验中，线性表是动态增长的。插入一个新元素后，为了使线性表仍保持有序性，必须要找到新元素插入的位置。实际上这是一个插入排序的问题。

为了要将新元素插入到一个有序的线性表中，可以从原有序线性表的最后一个元素开始，往前逐个与新元素比较，并将大于新元素的所有元素都往后移动一个位置，直到找到新元素应插入的位置为止。显然，插入一个新元素后，表的长度也增加了 1。现假设用一个数组 $L(1:m)$ 来存储线性表，其中 m 为最大容量(在本实验中为 $m=1000$)；用一个变量 n 表示线性表的长度(在本实验中，其初值为 $n=0$)。则可以得到将新元素插入到有序线性表的算法如下。

输入：数组 L(1:m)，有序线性表 L(1:n)，需插入的新元素 b。其中 $n<m$。

输出：插入 b 后的有序线性表 L(1:n)。

```
IF n = m THEN OUTPUT "overflow"
ELSE
  {IF n = 0 THEN L(n + 1) ← b
   ELSE
    {i ← n
     WHILE (i > 0) and (L(i) > b) DO {L(i + 1) ← L(i); i ← i − 1}
     L(i + 1) ← b
    }
   n ← n + 1
  }
RETURN
```

要在原线性表中删除一个元素 b(在本实验中，保证 b 在线性表中)，且仍保持线性表的顺序存储结构，可以从线性表的第一个元素开始，往后逐个与新元素比较，直到发现一个元素与新元素相等。然后从当前位置的下一位置开始，将后面所有元素都往前移动一个位置，直到线性表的最后一个元素为止。显然，删除一个元素后，线性表的长度减小了

1。其算法如下。

输入：线性表 L(1:n)(其中 n 为线性表的长度)，删除的元素 b(一定在线性表中)。

输出：删除 b 后的线性表 L(1:n)。

```
IF n = 0 THEN OUTPUT "underflow"
ELSE
  {i ← 1
   WHILE (i ≤ n) and (L(i) ≠ b) DO i ← i + 1
   IF i > n THEN OUTPUT "not this element"
   ELSE
      {IF i < n THEN
          FOR j = i TO n - 1 DO L(j) ← L(j + 1)
       n ← n - 1
      }
  }
RETURN
```

在上述算法中，从线性表的第一个元素开始寻找要删除的元素 *b*，实际上，我们还可以从线性表的最后一个元素开始寻找 *b*，其算法留给读者自行考虑。

实验 2.2 迷 宫 问 题

一、实验目的

巩固数组与栈的基本概念，掌握栈在程序设计中的具体应用。

通过对求解迷宫问题的分析，进一步了解程序设计的基本方法，提高解决实际问题的能力。

二、实验内容

今有一个迷宫，用一个二维数组表示，其元素值只有两个：0 与 1。其中以值 0 表示通路，以值 1 表示阻塞。在此数组的左上方有一入口，右下方有一出口，如图 2.1 所示。问从入口到出口有无通路？若有，则指出其中一条通路的路径(即从入口开始，每一步所到的位置及下一步的方向)。

三、实验步骤和要求

1. 仔细预习本实验中对于求解迷宫问题的方法说明，并根据实验要求事先编制好程序。

2. 上机调试你所编写的程序，并最后输出结果。要求输出结果包括以下两项：

输出迷宫数组，其中通路上的“0”用另一数字（例如“8”）替换；

依次输出通路上的各位置坐标及下一步的方向 (i, j, v)。

3. 实验做完后，整理源程序清单和结果，并对实验中遇到的问题进行分析和说明，必要时写出自己的体会。

4. 若采用异于本实验中说明的方法，在实验报告中必须作详细说明(包括方法概述、

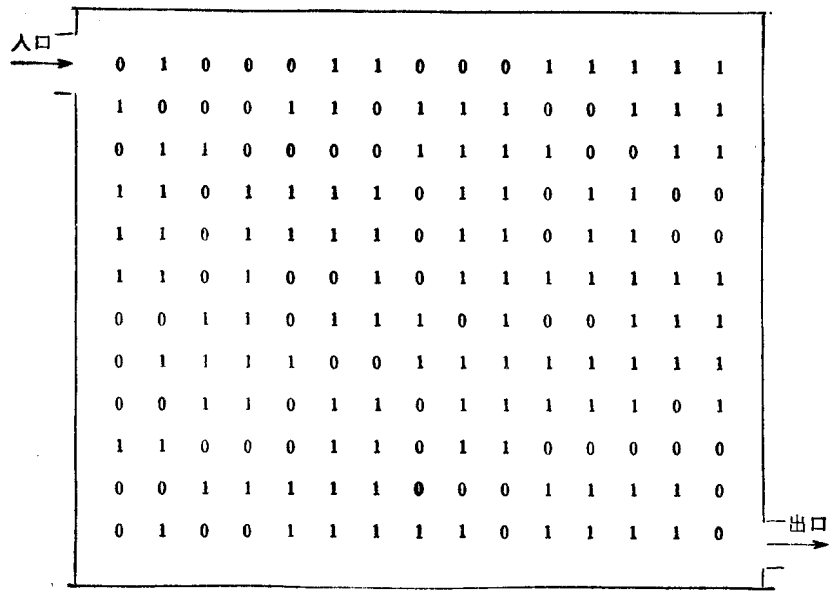

图 2.1　迷宫例

所采用的数据结构、算法的描述)。

四、方法说明

这是一个耗子走迷宫的古典问题。现在我们借助计算机来解决这个问题。显然，在找到一条正确路径之前可能要走许多错误的路径，但利用计算机可以记下它走过的所有路径，以免重入歧路。

为了方便，我们将此迷宫问题作如下说明。

(1) 设二维数组 maze (i,j) $(1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n)$ 表示迷宫，并设在 (1, 1) 处进入迷宫 (maze (1,1) = 0)，在 (m,n) 处走出迷宫。迷宫中的每一位置用它所在的行和列的坐标 (i,j) 来表示。

(2) 对于迷宫中的每个位置 (i,j) 处，可能移动的路线可以有八个方向，我们用指南针上八个方向的名字作为这八个方向的名称。从正东起沿顺时针方向的顺序为：E、SE、S、SW、W、NW、N、NE，如图 2.2 所示。

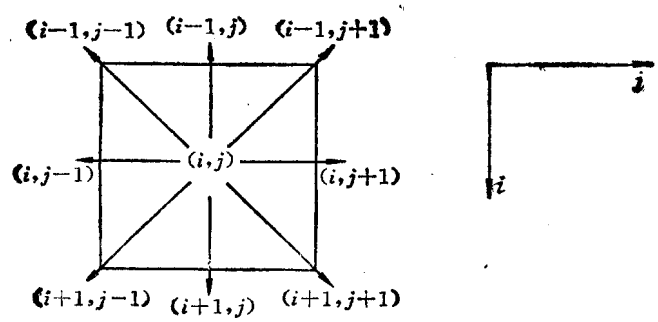

图 2.2　方向示意图

为了简化计算，我们用一个二维数组 move 表示这八个方向上坐标的增量，并把这八个方向从正东起按顺时针方向编上序号(如表 2.1 所示)，则 move (v, 1) 表示第 v 个方向上 i 的增量，move (v, 2) 表示第 v 个方向上 j 的增量。例如，从 (i, j) 出发，沿着东南 (SE) 方向到达下一位置的坐标 (g,h) 为

$$g = i + \text{move}\ (2,1) = i + 1$$
$$h = j + \text{move}\ (2,2) = j + 1$$

再如，从 (i,j) 出发，沿着正西 (W) 方向到达下一位置的坐标 (g,h) 为

$$g = i + \text{move}\ (5,1) = i$$
$$h = j + \text{move}\ (5,2) = j - 1$$

表 2.1 方向增量表

	E	SE	S	SW	W	NW	N	NE
v	1	2	3	4	5	6	7	8
move (v,1)	0	1	1	1	0	−1	−1	−1
move (v,2)	1	1	0	−1	−1	−1	0	1

(3) 当处于迷宫边缘时，它的下一个位置不再有八种可能，甚至只有 3 种可能。所以，为了避免时时控制检测边界状态，我们将二维数组 maze (1:m, 1:n) 扩充为 maze (0:m + 1, 0:n + 1)，且令第 0 行和第 0 列、第 m + 1 行和第 n + 1 列的值均为 1。

(4) 计算机解迷宫，用的是瞎子走路的办法，一步一试探。在每一步开始时，都从正东方向起，沿顺时针方向检测。当探到某个方向上下一位置的值为 0 时，就沿此方向走一步，当这一步四周剩下的七个方向上的值均为 1 时，则退回一步重新检测下一方向。因此，为了能够退到刚走过的位置，并继续检测下一方向，必须记下所走过的路径和方向。我们用一个栈来记下所走过的路径和方向，即用 stack (top, 1) 记横坐标，stack (top, 2) 记纵坐标，stack (top, 3) 记方向。其中 top 为栈顶指针，它反映了所走路径的步数。同时，为了避免走重复的路径，再用一个二维数组 mark，其所有元素的初值为 0，当到达某位置 (i,j) 时，就设 mark (i,j) = 1。

由以上对迷宫解法的说明，我们可以粗略地得出解迷宫的步骤：

(1) 从 (1, 1) 处进入迷宫，并向正东方向试探。

(2) 检测下一方向 (g,h)。若 (g,h) = (m,n)，且 maze (m, n) = 0，则说明已到达迷宫出口，输出所有走过的路径。若 maze (m,n) ≠ 0，说明迷宫无解。

(3) 若 (g,h) ≠ (m,n)，但 (g,h) 方位能走通(即 maze (g,h) = 0)，且为第一次经过(即 mark (g,h) = 0)，则将所走的这一步的位置及方向推入栈中，并从 (g,h) 出发，再向东试探下一步(必须注意，还要记下所经过的这一位置，即令 mark (g, h) = 1)。若上述条件不成立，仍在原 (i,j) 方位上换一个方向进行试探。

(4) 若 (i,j) 方位周围的七个方向都阻塞或都曾经到达过，则需从栈中退出一步，然后换个方向再作试探。若此时已退到迷宫入口，即栈空，则无路可退，说明迷宫走不

通。

从以上的说明中可以归纳出解迷宫的算法如下。

```
输入: maze (0:m + 1, 0:n + 1), move (1:8,1:2)
输出: stack
FOR i = 1 TO m DO
FOR j = 1 TO n DO mark (i,j) ← 0
(i,j,v) ← (1,1,1); mark (1,1) ← 1
top ← 0
WHILE (top ≠ 0) or (v ≠ 8) DO
  {g ← i + move (v,1); h ← j + move (v,2)
   IF ((g,h) = (m,n)) and (maze (m,n) = 0) THEN
    {stack (top + 1,1) ← m
     stack (top + 1,2) ← n
     stack (top + 1,3) ← 1
     top ← top + 1
     OUTPUT (stack); RETURN
    }
   IF (maze (g,h) = 0) and (mark (g,h) = 0) THEN
    {mark (g,h) ← 1; top ← top + 1
     stack (top,1) ← i
     stack (top,2) ← j
     stack (top,3) ← v
      (i,j,v) ← (g,h,1)
    }
   ELSE
    {IF v < 8 THEN v ← v + 1
      ELSE
        {WHILE (stack (top,3) = 8) and (top > 0) DO top ← top — 1
         IF top > 0 THEN
            {i ← stack (top,1)
             j ← stack (top,2)
             v ← stack (top,3)
             top ← top — 1
            }
        }
    }
  }
OUTPUT "无通路"
```

RETURN

实验 2.3　求解约瑟夫问题

一、实验目的

用循环队列的链式结构求解约瑟夫（Josephu）问题。

二、实验内容

设有 n 个人围坐在圆桌周围，从某个位置开始编号为 1，2，3，…，n，编号为 1 的位置上的人从 1 开始报数，数到 m 的人便出列；下一个人（第 $m+1$ 个）又从 1 开始报数，数到 m 的人便是第二个出列的人。如此重复下去，直到最后一个人出列为止，于是便得到一个出列的顺序。例如，当 $n=8$，$m=4$ 时，若从第一个位置数起，则出列的次序为 4、8、5、2、1、3、7、6。

这个问题称为约瑟夫问题。

三、实验步骤和要求

1. 先对于一般的 n 和 m 编写好程序，并调试通过。

2. 以 $n=30$，$m=7$ 作为输入，运行你的程序。

3. 以 $n=7$，$m=30$ 作为输入，再运行你的程序。

4. 整理程序清单和两次运行的结果，写好实验报告。如果你采用异于本方法说明中的数据结构和算法，则应加以详细说明。

四、方法说明

设以自然数 1，2，3，…，n 为元素构成一个循环队列，并用一个数组 A(1:n) 存放此队列中各元素的直接后继，其中数组元素 A(i) 表示整数 i 的下一个整数。显然，在初始时，此数组的各元素为

$$\begin{cases} A(i)=i+1, & i=1,2,\cdots,n-1 \\ A(n)=1 \end{cases}$$

以后由于不断地有元素从这个队列中出来，这个数组中的元素值也在不断地变化。即当有元素出列后，某些整数 i 的下一个数就不一定是 $i+1$ 了。

一般来说，假设当前要出列的整数为 k，它的前一个数为 l，则有 A(l) = k，且 k 的下一个数为 A(k)，即

$$\cdots, l, k, A(k), \cdots$$

当 k 出列以后，将变为

$$\cdots, l, A(k), \cdots$$

显然，此时 l 的下一个数不是 k，而是 A(k)。由此看出，为了使整数 k 出列，只需要做以下两件工作：

(1) 将出列的整数 k 依次存入另一数组 B；

(2) 将元素 A(k) 的值赋给 A(l)。

下一步的问题是如何模拟报数，即如何确定每次该出列的数。根据上面的分析可知，当整数 k 出列后，下一轮第一个报数者应是 A(k)，我们将它重新赋给 k，当它报完数

后，又将它赋给 l，且又将下一个报数者 A(k) 赋给 k，以此类推，当报过 $m-1$ 次后，k 便是应该出列者。由此可知，这个报数的过程可以用语句

FOR j = 1 TO m − 1 DO {l ← k; k ← A(k)}

来模拟，其中 k 的初值为上一次出列者的直接后继。

综上所述，如果用数组 B(1:n) 依次存放每次的出列者，则约瑟夫问题的求解算法如下。

```
输入: A(1:n), m
输出: B(1:n)
FOR i = 1 TO n DO A(i) ← i + 1
A(n) ← 1
k ← 1
FOR i = 1 TO n DO
    {FOR j = 1 TO m − 1 DO {l ← k; k ← A(k)}
     B(i) ← k; A(l) ← A(k); k ← A(k)
    }
OUTPUT (B)
RETURN
```

实验 2.4　多项式的表示与相加

一、实验目的

掌握多项式的链式存储结构及其两个多项式的相加，并进一步熟练线性链表的插入运算。

二、实验内容

设两个多项式为

$$A_m(x) = a_m x^{e_m} + a_{m-1}x^{e_{m-1}} + \cdots + a_1 x^{e_1}$$

$$B_n(x) = b_n x^{e'_n} + b_{n-1}x^{e'_{n-1}} + \cdots + b_1 x^{e'_1}$$

其中 a_i 与 b_i 均不为 0，且

$$e_m > e_{m-1} > \cdots > e_1 \geqslant 0$$

$$e'_n > e'_{n-1} > \cdots > e'_1 \geqslant 0$$

编写一个程序，实现以下功能：

(1) 用链式结构存放多项式 $A_m(x)$ 与 $B_n(x)$；

(2) 在不增加结点的前提下，作两个多项式的加法，最后输出和多项式。

三、实验步骤和要求

1. 根据方法说明中介绍的数据结构及处理的方法编制程序。在程序中对于两个原多项式以及和多项式都要以某种合适的方式打印输出。

2. 运行程序，从键盘输入两多项式

$$A_7(x) = 3x^9 + 4x^8 - 2x^6 + 5x^5 + 8x^2 + 9x + 1$$

$$B_6(x) = -2x^8 - 3x^7 + 7x^6 + 4x^3 - 9x + 6$$

3. 整理程序与结果，写好实验报告。

4. 若有可能，自行确定两个多项式 $A_m(x)$ 和 $B_n(x)$ 作为输入，再运行一次这个程序。

四、方法说明

1. 多项式的表示

多项式用线性链表表示，链表中的每一个结点表示多项式中的一项，两个数据域分别存放系数(称为系数域，用 COEF 表示)和指数(称为指数域，记为 EXP)，其指针域的链接是按照指数从大到小的次序。即多项式 $A_m(x)$ 与 $B_n(x)$ 分别用图 2.3 所示的两链表表示。其中 HA、HB 分别为它们的头指针。

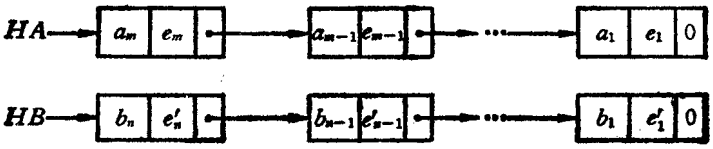

图 2.3　多项式的链接表

为了模拟存储空间，可以开辟一个具有足够容量(以多项式的一般规模而定)的三列二维数组 L(1:*t*,1:3)，其中第一列存放系数，第二列存放指数，第三列存放链接指针。显然，此数组的一行就表示了链接表中的一个结点。

2. 多项式的相加

当两个多项式相加时，除两个原多项式的链式结构需要存储空间外，其和多项式也用一个链式结构的存储空间。考虑到存储空间有限，和多项式的存储空间可以覆盖 $A_m(x)$ 和 $B_n(x)$ 的存储空间。

有了这种存储结构，两个一元多项式的相加就很简单了，其过程如下。

设 HA 为多项式 $A_m(x)$ 的头指针，HB 为多项式 $B_n(x)$ 的头指针，HC 为和多项式的头指针。在作相加运算之前，和多项式的链表为空(即 $HC = 0$)。然后设置指针 i 和 j (初始时 $i = HA$，$j = HB$)，以便于从 HA 和 HB 所指结点(即数组 L 的某两行)开始检测，对每一次检测结果作如下处理：

若两项指数相等，则系数相加，且两多项式的指针均进一。并判断其结果是否为 0，若为 0，则继续检测；否则将系数相加结果存放在多项式 $A_m(x)$ 的本次所检测结点的系数域中，并将此结点链接到和多项式上，再继续检测。

若多项式 $A_m(x)$ 当前项的指数小于 $B_n(x)$ 中当前项的指数，则多项式 $B_n(x)$ 的指针进一，且将 $B_n(x)$ 中本次所检测的结点链接到和多项式上，再继续检测。否则，多项式 $A_m(x)$ 的指针进一，且将 $A_m(x)$ 中本次所检测的结点链接到和多项式上，再继续检测。

若发现某个多项式已无后继，则将另一个多项式的剩余部分链接到和多项式上，过程结束。或者两个多项式同时无后继，过程结束。