

8.10 给出下面代码的输出：

```
int[][] array = {{ {1, 2}, {3, 4}}, {{5, 6}, {7, 8}}};  
System.out.println(array[0][0][0]);  
System.out.println(array[1][1][1]);
```

本章小结

1. 可以使用二维数组来存储表格。
2. 可以使用以下语法来声明二维数组变量：

元素类型 [] [] 数组变量

3. 可以使用以下语法来创建二维数组变量：

`new 元素类型 [行的个数][列的个数]`

4. 使用下面的语法表示二维数组中的每个元素：

数组变量 [行下标][列下标]

5. 可以使用数组初始化语法来创建和初始化二维数组：

元素类型 [] [] 数组变量 = {{ 某行的值 }, ..., { 某行的值 }}

6. 可以使用数组的数组构成多维数组。例如：一个三维数组变量可以声明为“元素类型 [] [] [] 数组变量”，并使用“`new 元素类型 [size1][size2][size3]`”来创建三维数组。

测试题

在线回答本章节的测试题，位于 www.cs.armstrong.edu/liang/intro10e/quiz.html。

编程练习题

- *8.1 (求矩阵中各列数字的和) 编写一个方法，求整数矩阵中特定列的所有元素的和，使用下面的方法头：

```
public static double sumColumn(double[][] m, int columnIndex)
```

编写一个测试程序，读取一个 3×4 的矩阵，然后显示每列元素的和。下面是一个运行示例：

```
Enter a 3-by-4 matrix row by row:  
1.5 2 3 4 ↵ Enter  
5.5 6 7 8 ↵ Enter  
9.5 1 3 1 ↵ Enter  
Sum of the elements at column 0 is 16.5  
Sum of the elements at column 1 is 9.0  
Sum of the elements at column 2 is 13.0  
Sum of the elements at column 3 is 13.0
```

- *8.2 (求矩阵主对角线元素的和) 编写一个方法，求 $n \times n$ 的 double 类型矩阵中主对角线上所有数字的和，使用下面的方法头：

```
public static double sumMajorDiagonal(double[][] m)
```

编写一个测试程序，读取一个 4×4 的矩阵，然后显示它的主对角线上的所有元素的和。下面是一个运行示例：

```
Enter a 4-by-4 matrix row by row:  
1 2 3 4.0 ↵ Enter  
5 6.5 7 8 ↵ Enter  
9 10 11 12 ↵ Enter
```

13 14 15 16 ↵Enter

Sum of the elements in the major diagonal is 34.5

*8.3 (按考分对学生排序) 重写程序清单 8-2, 按照正确答案个数的升序显示学生。

**8.4 (计算每个雇员每周工作的小时数) 假定所有雇员每周工作的时间存储在一个二维数组中。每行将一个雇员 7 天的工作时间记录在 7 列中。例如：右面显示的数组存储了 8 个雇员的工作时间。编写一个程序，按照总工时降序的方式显示雇员和他们的总工时。

8.5 (代数方面：两个矩阵相加) 编写两个矩阵相加的方法。
方法头如下：

```
public static double[][] addMatrix(double[][] a, double[][] b)
```

为了能够进行相加，两个矩阵必须具有相同的维数，并且元素具有相同或兼容的数据类型。假设 c 表示最终的矩阵，每个元素 c_{ij} 就是 $a_{ij} + b_{ij}$ 。例如，对于两个 3×3 的矩阵 a 和 b，c 就有：

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{13} + b_{13} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} \\ a_{31} + b_{31} & a_{32} + b_{32} & a_{33} + b_{33} \end{pmatrix}$$

编写一个测试程序，提示用户输入两个 3×3 的矩阵，然后显示它们的和。下面是一个运行示例：

Enter matrix1: 1 2 3 4 5 6 7 8 9 ↵Enter
 Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 ↵Enter
 The matrices are added as follows
 1.0 2.0 3.0 0.0 2.0 4.0 1.0 4.0 7.0
 4.0 5.0 6.0 + 1.0 4.5 2.2 = 5.0 9.5 8.2
 7.0 8.0 9.0 1.1 4.3 5.2 8.1 12.3 14.2

**8.6 (代数方面：两个矩阵相乘) 编写两个矩阵相乘的方法。方法头如下：

```
public static double[][] multiplyMatrix(double[][] a, double[][] b)
```

为了使矩阵 a 能够和矩阵 b 相乘，矩阵 a 的列数必须与矩阵 b 的行数相同，并且两个矩阵的元素要具有相同或兼容的数据类型。假设矩阵 c 是相乘的结果，而 a 的列数是 n，那么每个元素 c_{ij} 就是 $a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \dots + a_{in} \times b_{nj}$ 。例如，对于两个 3×3 的矩阵 a 和 b，c 就有：

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

这里的 $c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j}$ 。

编写一个测试程序，提示用户输入两个 3×3 的矩阵，然后显示它们的乘积。下面是一个运行示例：

Enter matrix1: 1 2 3 4 5 6 7 8 9 ↵Enter
 Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 ↵Enter
 The multiplication of the matrices is
 1 2 3 0 2.0 4.0 5.3 23.9 24
 4 5 6 * 1 4.5 2.2 = 11.6 56.3 58.2
 7 8 9 1.1 4.3 5.2 17.9 88.7 92.4

*8.7 (距离最近的两个点) 程序清单 8-3 给出找到二维空间中距离最近的两个点的程序。修改该程序，让程序能够找出在三维空间上距离最近的两个点。使用一个二维数组表示这些点。使用下面的点来测试这个程序：

```
double[][] points = {{-1, 0, 3}, {-1, -1, -1}, {4, 1, 1},
{2, 0.5, 9}, {3.5, 2, -1}, {3, 1.5, 3}, {-1.5, 4, 2},
{5.5, 4, -0.5}};
```

计算两个点 (x_1, y_1, z_1) 和 (x_2, y_2, z_2) 之间距离的公式是 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$ 。

**8.8 (所有最近的点对) 修改程序清单 8-3，找出所有具有相同最小距离的点对。下面是一个运行示例：

```
Enter the number of points: 8 [Enter]
Enter 8 points: 0 0 1 1 -1 -1 2 2 -2 -2 -3 -3 -4 -4 5 5 [Enter]
The closest two points are (0.0, 0.0) and (1.0, 1.0)
The closest two points are (0.0, 0.0) and (-1.0, -1.0)
The closest two points are (1.0, 1.0) and (2.0, 2.0)
The closest two points are (-1.0, -1.0) and (-2.0, -2.0)
The closest two points are (-2.0, -2.0) and (-3.0, -3.0)
The closest two points are (-3.0, -3.0) and (-4.0, -4.0)
Their distance is 1.4142135623730951
```

***8.9 (游戏：井字游戏) 在井字游戏中，两个玩家使用各自的标志（一方用 X 则另一方就用 O），轮流填写 3×3 的网格中的某个空格。当一个玩家在网格的水平方向、垂直方向或者对角线方向上出现了三个相同的 X 或三个相同的 O 时，游戏结束，该玩家获胜。平局（没有赢家）是指当网格中所有的空格都被填满时没有任何一方的玩家获胜的情况。创建一个玩井字游戏的程序。

程序提示两个玩家可以选择 X 和 O 作为他们的标志。当输入一个标志时，程序在控制台上重新显示棋盘，然后确定游戏的状态（是获胜、平局还是继续）。下面是一个运行示例：

```
| | | |
-----
| | | |
-----
| | | |
```

```
Enter a row (0, 1, or 2) for player X: 1 [Enter]
Enter a column (0, 1, or 2) for player X: 1 [Enter]
```

```
| | | |
-----
| | X | |
-----
| | | |
```

```
Enter a row (0, 1, or 2) for player O: 1 [Enter]
Enter a column (0, 1, or 2) for player O: 2 [Enter]
```

```
| | | |
-----
| | X | O |
-----
| | | |
```

```
Enter a row (0, 1, or 2) for player X:
```

.	.	.
X		
0	X	0

```
X player won
```

- *8.10 (最大的行和列) 编写一个程序，在一个 4×4 的矩阵中随机填入 0 和 1，打印该矩阵，找到第一个具有最多 1 的行和列。下面是一个程序的运行示例：

```
0011  
0011  
1101  
1010  
The largest row index: 2  
The largest column index: 2
```

- **8.11 (游戏：九个正面和背面) 一个 3×3 的矩阵中放置了 9 个硬币，这些硬币有些面向上，有些面向下。可以使用 3×3 的矩阵中的 0 (正面) 或 1 (反面) 表示硬币的状态。下面是一些例子：

```
0 0 0    1 0 1    1 1 0    1 0 1    1 0 0  
0 1 0    0 0 1    1 0 0    1 1 0    1 1 1  
0 0 0    1 0 0    0 0 1    1 0 0    1 1 0
```

每个状态都可以使用一个二进制数表示。例如，前面的矩阵对应到数字：

```
000010000 101001100 110100001 101110100 100111110
```

总共会有 512 种可能性。所以，可以使用十进制数 0, 1, 2, 3, ..., 511 来表示这个矩阵的所有状态。编写一个程序，提示用户输入一个在 0 到 511 之间的数字，然后显示用字符 H 和 T 表示的对应的矩阵。下面是一个运行示例：

```
Enter a number between 0 and 511: 7 [Enter]  
H H H  
H H H  
T T T
```

用户输入 7，它代表的是 000000111。因为 0 代表 H 而 1 代表 T，所以输出正确。

- **8.12 (财务应用程序：计算税款) 使用数组重写程序清单 3-5。每个登记者的身份都有六种税率。每种税率都应用在某个特定范围内的可征税收入。例如：对一个可征税税率为 400 000 美元的单身登记者来讲，8350 美元的税率是 10%，8350 ~ 33 950 之间的税率是 15%，33 950 ~ 82 250 之间的税率是 25%，82 250 ~ 171 550 之间的税率是 28%，171 550 ~ 372 550 之间的税率是 33%，而 372 950 ~ 400 000 之间的税率是 36%。这六种税率对所有的登记身份都是一样的，可以用下面的数组来表示：

```
double[] rates = {0.10, 0.15, 0.25, 0.28, 0.33, 0.35};
```

所有登记者身份的每个税率括号都可以用一个二维数组表示，如下所示：

```
int[][] brackets = {  
    {8350, 33950, 82250, 171550, 372950}, // Single filer  
    {16700, 67900, 137050, 20885, 372950}, // Married jointly  
                                              // -or qualifying widow(er)  
    {8350, 33950, 68525, 104425, 186475}, // Married separately  
    {11950, 45500, 117450, 190200, 372950} // Head of household  
};
```

假设单身身份的登记者的可征税收入是 400 000 美元，该税收可以如下计算：

```
tax = brackets[0][0] * rates[0] +
(brackets[0][1] - brackets[0][0]) * rates[1] +
(brackets[0][2] - brackets[0][1]) * rates[2] +
(brackets[0][3] - brackets[0][2]) * rates[3] +
(brackets[0][4] - brackets[0][3]) * rates[4] +
(400000 - brackets[0][4]) * rates[5]
```

*8.13 (定位最大的元素) 编写下面的方法，返回二维数组中最大元素的位置。

```
public static int[] locateLargest(double[][] a)
```

返回值是包含两个元素的一维数组。这两个元素表示二维数组中最大元素的行下标和列下标。编写一个测试程序，提示用户输入一个二维数组，然后显示这个数组中最大元素的位置。下面是一个运行示例：

```
Enter the number of rows and columns of the array: 3 4 ↵ Enter
Enter the array:
23.5 35 2 10 ↵ Enter
4.5 3 45 3.5 ↵ Enter
35 44 5.5 9.6 ↵ Enter
The location of the largest element is at (1, 2)
```

**8.14 (探索矩阵) 编写程序，提示用户输入一个方阵的长度，随机地在矩阵中填入 0 和 1，打印这个矩阵，然后找出整行、整列或者对角线都是 0 或 1 的行、列和对角线。下面是这个程序的一个运行示例：

```
Enter the size for the matrix: 4 ↵ Enter
0111
0000
0100
1111
All 0s on row 1
All 1s on row 3
No same numbers on a column
No same numbers on the major diagonal
No same numbers on the sub-diagonal
```

*8.15 (几何：在一条直线上吗？) 编程练习题 6.39 给出一个方法，用于测试三点是否在一条直线上。编写下面的方法，检测 points 数组中所有的点是否都在同一条直线上。

```
public static boolean sameLine(double[][] points)
```

编写一个程序，提示用户输入 5 个点，并且显示它们是否在同一直线上。下面是一个运行示例：

```
Enter five points: 3.4 2 6.5 9.5 2.3 2.3 5.5 5 -5 4 ↵ Enter
The five points are not on the same line
```

```
Enter five points: 1 1 2 2 3 3 4 4 5 5 ↵ Enter
The five points are on the same line
```

*8.16 (对二维数组排序) 编写一个方法，使用下面的方法头对二维数组排序：

```
public static void sort(int m[][])
```

这个方法首先按行排序，然后按列排序。

例如：数组 {{4, 2}, {1, 7}, {4, 5}, {1, 2}, {1, 1}, {4, 1}} 将被排序为 {{1, 1}, {1, 2}, {1, 7}, {4, 1}, {4, 2}, {4, 5}}。

***8.17 (金融风暴) 银行会互相借钱。在经济艰难时期, 如果一个银行倒闭, 它就不能偿还贷款。一个银行的总资产是它当前的余款减去它欠其他银行的贷款。图 8-8 就是五个银行的状况图。每个银行的当前余额分别是 2500 万美元、1 亿 2500 万美元、1 亿 7500 万美元、7500 万美元和 1 亿 8100 万美元。从节点 1 到节点 2 的方向的边表示银行 1 借给银行 2 共计 4 千万美元。

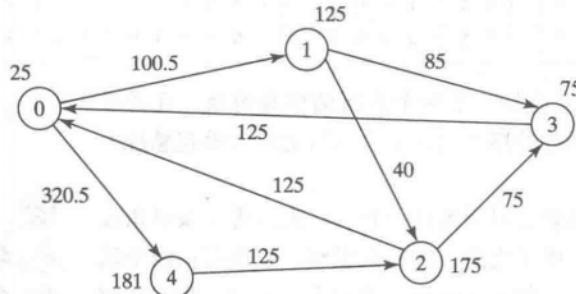


图 8-8 银行之间互相借款

如果银行的总资产在某个限定范围以下, 那么这个银行就是不安全的。它借的钱就不能返还给借贷方, 而且这个借贷方也不能将这个贷款算入它的总资产。因此, 如果借贷方总资产在限定范围以下, 那么它也不安全。编写程序, 找出所有不安全的银行。程序如下读取输入。它首先读取两个整数 n 和 $limit$, 这里的 n 表示银行个数, 而 $limit$ 表示要保持银行安全的最小总资产。然后, 程序会读取描述 n 个银行的 n 行信息, 银行的 id 从 0 到 $n-1$ 。每一行的第一个数字都是银行的余额, 第二个数字表明从该银行借款的银行, 其余的就是两个数字构成的数据对。每对都描述一个借款方。每一对数字的第一个数就是借款方的 id, 第二个数就是所借的钱数。例如, 在图 8-8 中五个银行的输入如下所示 (注意: $limit$ 是 201):

```
5 201
25 2 1 100.5 4 320.5
125 2 2 40 3 85
175 2 0 125 3 75
75 1 0 125
181 1 2 125
```

银行 3 的总资产是 $75+125$, 这个数字是在 201 以下的。所以, 银行 3 是不安全的。在银行 3 变得不安全之后, 银行 1 的总资产也降为 $125+40$ 。所以, 银行 1 也不安全。程序的输出应该是:

Unsafe banks are 3 1

提示: 使用一个二维数组 `borrowers` 来表示贷款。`borrowers[i][j]` 表明银行 i 贷款给银行 j 的贷款额。一旦银行 j 变得不安全, 那么 `borrowers[i][j]` 就应该设置为 0。

*8.18 (打乱行) 编写一个方法, 使用下面的方法头打乱一个二维整型数组的行:

```
public static void shuffle(int[][] m)
```

编写一个测试程序, 打乱下面的矩阵:

```
int[][] m = {{1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}};
```

**8.19 (模式识别: 连续的四个相等的数) 编写下面的方法, 测试一个二维数组是否有四个连续的数字具有相同的值, 这四个数可以是水平方向的、垂直方向的或者对角线方向的。

```
public static boolean isConsecutiveFour(int[][] values)
```

编写一个测试程序, 提示用户输入一个二维数组的行数、列数以及数组中的值。如果这个数组有四个连续的数字具有相同的值, 就显示 `true`; 否则, 显示 `false`。下面是结果为 `true` 的一些例子:

0	1	0	3	1	6	1
0	1	6	8	6	0	1
5	6	2	1	8	2	9
6	5	6	1	1	9	1
1	3	6	1	4	0	7
3	3	3	3	4	0	7

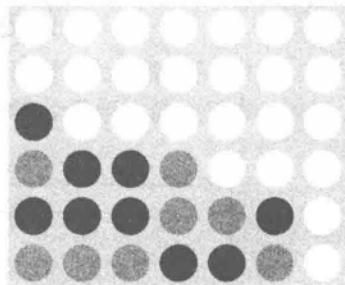
0	1	0	3	1	6	1
0	1	6	8	6	0	1
5	5	2	1	8	2	9
6	5	6	1	1	9	1
1	5	6	1	4	0	7
3	5	3	3	4	0	7

0	1	0	3	1	6	1
0	1	6	8	6	0	1
5	6	2	1	6	2	9
6	5	6	6	1	9	1
1	3	6	1	4	0	7
3	6	3	3	4	0	7

0	1	0	3	1	6	1
0	1	6	8	6	0	1
9	6	2	1	8	2	9
6	9	6	1	1	9	1
1	3	9	1	4	0	7
3	3	3	9	4	0	7

***8.20 (游戏：四子连) 四子连是一个两个人玩的棋盘游戏，在游戏中，玩家轮流将有颜色的棋子放在一个六行七列的垂直悬挂的网格中，如下所示。

这个游戏的目的是在对手实现一行、一列或者一条对角线上有四个相同颜色的棋子之前，你能先做到。程序提示两个玩家交替地下红子 Red 或黄子 Yellow。当放下一子时，程序在控制台重新显示这个棋盘，然后确定游戏的状态（赢、平局还是继续）。下面是一个运行示例：



```

| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Drop a red disk at column (0-6): 0 ↵Enter

| | | | | |
| | | | | |
| | | | | |
|R| | | | |
| | | | | |

Drop a yellow disk at column (0-6): 3 ↵Enter

| | | | | | |
| | | | | |
| | | | | |
|R| | Y | | |
| | | Y | R | Y | |

The yellow player won

```

*8.21 (中心城市) 给定一组城市，中心城市是和所有其他城市之间具有最短距离的城市。编写一个程序，提示用户输入城市的数目以及城市的位置（坐标），找到中心城市以及和所有其他城市的总

距离。

```
Enter the number of cities: 5 [Enter]
Enter the coordinates of the cities:
2.5 5 5.1 3 1 9 5.4 54 5.5 2.1 [Enter]
The central city is at (2.5, 5.0)
The total distance to all other cities is 60.81
```

*8.22 (偶数个 1) 编写一个程序，产生一个 6×6 的填写了 0 和 1 的二维矩阵，显示该矩阵，检测是否每行以及每列中有偶数个 1。

*8.23 (游戏：找到翻转的单元格) 假设给定一个填满 0 和 1 的 6×6 矩阵。所有的行和列都有偶数个 1。让用户翻转一个单元（即从 1 翻成 0 或者从 0 翻成 1），编写一个程序找到哪个单元格被翻转了。程序应该提示用户输入一个 6×6 的填满 0 和 1 的矩阵，并且找到第一个 r 行以及第一个 c 列具有偶数个 1 的特征是不符合的（即 1 的数目不是偶数），则该翻转的单元格位于 (r, c) 。下面是一个运行示例：

```
Enter a 6-by-6 matrix row by row:
1 1 1 0 1 1 [Enter]
1 1 1 1 0 0 [Enter]
0 1 0 1 1 1 [Enter]
1 1 1 1 1 1 [Enter]
0 1 1 1 1 0 [Enter]
1 0 0 0 0 1 [Enter]
The flipped cell is at (0, 1)
```

*8.24 (检测数独的解决方案) 程序清单 8-4 通过检测棋盘上的每个数字是否是有效的，从而检测一个解决方案是否是有效的。重写该程序，通过检测是否每行、每列以及每个小的方盒中具有数字 1 到 9 来检测解决方案的有效性。

*8.25 (马尔科夫矩阵) 一个 $n \times n$ 的矩阵被称为一个正马尔科夫矩阵，当且仅当每个元素都是正数，并且每列的元素的和为 1。编写下面的方法来检测一个矩阵是否是一个马尔科夫矩阵。

```
public static boolean isMarkovMatrix(double[][] m)
```

编写一个测试程序，提示用户输入一个 3×3 的 double 值的矩阵，测试它是否是一个马尔科夫矩阵。下面是一个运行示例：

```
Enter a 3-by-3 matrix row by row:
0.15 0.875 0.375 [Enter]
0.55 0.005 0.225 [Enter]
0.30 0.12 0.4 [Enter]
It is a Markov matrix
```

```
Enter a 3-by-3 matrix row by row:
0.95 -0.875 0.375 [Enter]
0.65 0.005 0.225 [Enter]
0.30 0.22 -0.4 [Enter]
It is not a Markov matrix
```

*8.26 (行排序) 用下面的方法实现一个二维数组中的行排序。返回一个新的数组，并且原数组保持不变。

```
public static double[][] sortRows(double[][] m)
```

编写一个测试程序，提示用户输入一个 3×3 的 double 类型值的矩阵，显示一个新的每行排好序的矩阵。下面是一个运行示例。

```
Enter a 3-by-3 matrix row by row:
0.15 0.875 0.375 ↵Enter
0.55 0.005 0.225 ↵Enter
0.30 0.12 0.4 ↵Enter
```

```
The row-sorted array is
0.15 0.375 0.875
0.005 0.225 0.55
0.12 0.30 0.4
```

- *8.27 (列排序) 用下面的方法实现一个二维数组中的列排序。返回一个新的数组，并且原数组保持不变。

```
public static double[][] sortColumns(double[][] m)
```

编写一个测试程序，提示用户输入一个 3×3 的double类型值的矩阵，显示一个新的每列排好序的矩阵。下面是一个运行示例。

```
Enter a 3-by-3 matrix row by row:
0.15 0.875 0.375 ↵Enter
0.55 0.005 0.225 ↵Enter
0.30 0.12 0.4 ↵Enter
```

```
The column-sorted array is
0.15 0.0050 0.225
0.3 0.12 0.375
0.55 0.875 0.4
```

- 8.28 (严格相同的数组) 如果两个二维数组m1和m2相应的元素是相等的话，则认为它们是严格相同的。编写一个方法，如果m1和m2是严格相同的话，返回true。使用下面的方法头：

```
public static boolean equals(int[][] m1, int[][] m2)
```

编写一个测试程序，提示用户输入两个 3×3 的整数数组，显示两个矩阵是否是严格相同的。下面是一个运行示例。

```
Enter list1: 51 22 25 6 1 4 24 54 6 ↵Enter
Enter list2: 51 22 25 6 1 4 24 54 6 ↵Enter
The two arrays are strictly identical
```

```
Enter list1: 51 25 22 6 1 4 24 54 6 ↵Enter
Enter list2: 51 22 25 6 1 4 24 54 6 ↵Enter
The two arrays are not strictly identical
```

- 8.29 (相同的数组) 如果两个二维数组m1和m2具有相同的内容，则它们是相同的。编写一个方法，如果m1和m2是相同的话，返回true。使用下面的方法头：

```
public static boolean equals(int[][] m1, int[][] m2)
```

编写一个测试程序，提示用户输入两个 3×3 的整数数组，显示两个矩阵是否是相同的。下面是一个运行示例。

```
Enter list1: 51 25 22 6 1 4 24 54 6 ↵Enter
Enter list2: 51 22 25 6 1 4 24 54 6 ↵Enter
The two arrays are identical
```

```
Enter list1: 51 5 22 6 1 4 24 54 6 ↵Enter
Enter list2: 51 22 25 6 1 4 24 54 6 ↵Enter
The two arrays are not identical
```

*8.30 (代数: 解答线性方程) 编写一个方法, 解答下面的 2×2 线性方程组系统:

$$\begin{aligned} a_{00}x + a_{01}y &= b_0 \\ a_{10}x + a_{11}y &= b_1 \end{aligned} \quad x = \frac{b_0a_{11} - b_1a_{01}}{a_{00}a_{11} - a_{01}a_{10}} \quad y = \frac{b_1a_{00} - b_0a_{10}}{a_{00}a_{11} - a_{01}a_{10}}$$

方法头为:

```
public static double[] linearEquation(double[][] a, double[] b)
```

如果 $a_{00}a_{11} - a_{01}a_{10}$ 为 0, 方法返回 `null`。编写一个测试程序, 提示用户输入 a_{00} 、 a_{01} 、 a_{10} 、 a_{11} 、 b_0 以及 b_1 , 并且显示结果。如果 $a_{00}a_{11} - a_{01}a_{10}$ 为 0, 报告“方程无解”。运行示例和编程练习题 3.3 的类似。

*8.31 (几何: 交点) 编写一个方法, 返回两条直线的交点。两条直线的交点可以使用编程练习题 3.25 中显示的公式找到。假设 (x_1, y_1) 和 (x_2, y_2) 是直线 1 上的两点, 而 (x_3, y_3) 和 (x_4, y_4) 位于直线 2 上。方法头是:

```
public static double[] getIntersectingPoint(double[][] points)
```

这些点保存在一个 4×2 的二维矩阵 `points` 中, 其中 (`points[0][0]`, `points[0][1]`) 代表 (x_1, y_1) 。方法返回交点, 或者如果两条线平行的话就返回 `null`。编写一个程序, 提示用户输入四个点, 并且显示交点。运行示例参见编程练习题 3.25。

*8.32 (几何: 三角形面积) 编写一个方法, 使用下面的方法头, 返回一个三角形的面积:

```
public static double getTriangleArea(double[][] points)
```

点保存在一个 3×2 的二维矩阵 `points` 中, 其中 (`points[0][0]`, `points[0][1]`) 代表 (x_1, y_1) 。三角形面积的计算可以使用编程练习题 2.19 中的公式。如果三个点在一条直线上, 方法返回 0。编写一个程序, 提示用户输入三角形的三个点, 然后显示三角形的面积。下面是一个运行示例。

```
Enter x1, y1, x2, y2, x3, y3: 2.5 2 5 -1.0 4.0 2.0 ↵Enter
The area of the triangle is 2.25
```

```
Enter x1, y1, x2, y2, x3, y3: 2 2 4.5 4.5 6 6 ↵Enter
The three points are on the same line
```

*8.33 (几何: 多边形的子面积) 一个具有四个顶点的凸多边形被分为四个三角形, 如图 8-9 所示。

编写一个程序, 提示用户输入四个顶点的坐标, 然后以升序显示四个三角形的面积。下面是一个运行示例。

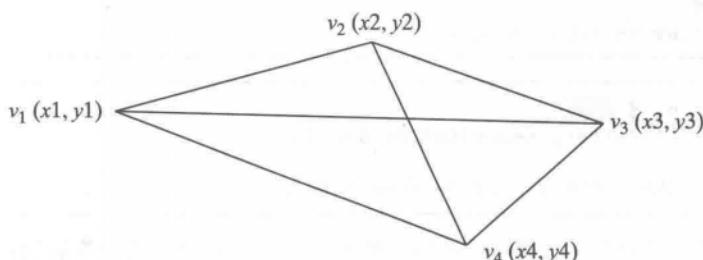


图 8-9 一个具有四个顶点的多边形被四个顶点所限定

```
Enter x1, y1, x2, y2, x3, y3, x4, y4:  
-2.5 2 4 4 3 -2 -2 -3.5 ↵ Enter  
The areas are 6.17 7.96 8.08 10.42
```

- *8.34 (几何: 最右下角的点) 在计算几何中经常需要从一个点集中找到最右下角的点。编写以下方法, 从一个点的集合中返回最右下角的点。

```
public static double[]  
getRightmostLowestPoint(double[][] points)
```

编写一个测试程序, 提示用户输入 6 个点的坐标, 然后显示最右下角的点。下面是一个运行示例。

```
Enter 6 points: 1.5 2.5 -3 4.5 5.6 -7 6.5 -7 8 1 10 2.5 ↵ Enter  
The rightmost lowest point is (6.5, -7.0)
```

- **8.35 (最大块) 给定一个元素为 0 或者 1 的方阵, 编写一个程序, 找到一个元素都为 1 的最大的子方阵。程序提示用户输入矩阵的行数。然后显示最大的子方阵的第一个元素, 以及该子方阵中的行数。下面是一个运行示例。

```
Enter the number of rows in the matrix: 5 ↵ Enter  
Enter the matrix row by row:  
1 0 1 0 1 ↵ Enter  
1 1 1 0 1 ↵ Enter  
1 0 1 1 1 ↵ Enter  
1 0 1 1 1 ↵ Enter  
1 0 1 1 1 ↵ Enter  
  
The maximum square submatrix is at (2, 2) with size 3
```

程序需要实现和使用下面的方法来找到最大的子方阵:

```
public static int[] findLargestBlock(int[][] m)
```

返回值是一个包含三个值的数组。前面两个值是子方阵中的行和列的下标, 第 3 个值是子方阵中的行数。

- **8.36 (拉丁正方形) 拉丁正方形是一个 $n \times n$ 的数组, 由 n 个不同的拉丁字母填充, 每个拉丁字母恰好只在每行和每列中出现一次。编写一个程序, 提示用户输入数字 n 以及字符数组, 如示例输出所示, 检测该输出数组是否是一个拉丁正方形。字符是从 A 开始的前面 n 个字符。

```
Enter number n: 4 ↵ Enter  
Enter 4 rows of letters separated by spaces:  
A B C D ↵ Enter  
B A D C ↵ Enter  
C D B A ↵ Enter  
D C A B ↵ Enter  
The input array is a Latin square
```

```
Enter number n: 3 ↵ Enter  
Enter 3 rows of letters separated by spaces:  
A F D ↵ Enter  
Wrong input: the letters must be from A to C
```

- **8.37 (猜测首府) 编写一个程序, 重复提示用户输入一个州的首府。当接收到用户输入后, 程序报告答案是否正确。假设 50 个州以及它们的首府保存在一个二维数组中, 如图 8-10 所示。程序提

示用户回答所有州的首府，并且显示所有正确回答的数目（忽略英文字母的大小写）。

Alabama	Montgomery
Alaska	Juneau
Arizona	Phoenix
...	...
...	...

图 8-10 一个保存了州以及它们的首府的二维数组

下面是一个运行示例。

```
What is the capital of Alabama? Montogomery ↵Enter
The correct answer should be Montgomery
What is the capital of Alaska? Juneau ↵Enter
Your answer is correct
What is the capital of Arizona? ...
...
The correct count is 35
```