```
            (t=a[i];a[i]=a[i+1];a[i+1]=t;)
        printf("the sorted numbers :\n");
        for(i=0;i<10;i++)
          printf("%d ",a[i]);
        printf("\n");
        return 0;
    }
```

运行结果：

```
input 10 numbers :
34 67 90 43 124 87 65 99 132 26

the sorted numbers :
26 34 43 65 67 87 90 99 124 132
```

程序分析：程序中实现起泡法排序算法的主要是第 10～13 行。请仔细分析嵌套的 for 语句。当执行外循环第 1 次循环时，j=0，然后执行第 1 次内循环，此时 i=0，在 if 语句中将 a[i]和 a[i+1]比较，就是将 a[0]和 a[1]比较。执行第 2 次内循环时，i=1，a[i]和 a[i+1]比较，就是将 a[1]和 a[2]比较……执行最后一次内循环时，i=8，a[i]和 a[i+1]比较，就是将 a[8]和 a[9]比较。这时第 1 趟过程完成了。

当执行第 2 次外循环时，j=1，开始第 2 趟过程。内循环继续的条件是 i<9－j，由于 j=1，因此相当于 i<8，即 i 由 0 变到 7，要执行内循环 8 次。其余类推。

说明：通过此例，着重学习有关排序的算法。排序的算法有多种，本例介绍的是起泡法，常用的还有选择法、希尔法等。本章的习题 2 是要求用选择法排序，其程序在《C 程序设计(第五版)学习辅导》一书第 6 章，建议读者尽可能参考一下。希望读者不要满足于教材中的内容，要善于扩展知识，善于思考，善于比较，善于归纳提高。

重要的是了解和掌握解题思路，学会分析问题，建立算法，以及如何利用 C 语言编程的技巧。

6.2　怎样定义和引用二维数组

前面已提到，有的问题需要用二维数组来处理。例如，有 3 个小分队，每队有 6 名队员，要把这些队员的工资用数组保存起来以备查。这就需要用到二维数组，见图 6.6。如果建立一个数组 pay，它应当是二维的，第一维用来表示第几分队，第二维用来表示第几个队员。例如用 $pay_{2,3}$ 表示 2 分队第 3 名队员的工资，它的值是 1725。

	队员 1	队员 2	队员 3	队员 4	队员 5	队员 6
1 分队	2456	1847	1243	1600	2346	2757
2 分队	3045	2018	1725	2020	2458	1436
3 分队	1427	1175	1046	1976	1477	2018

图　6.6

二维数组常称为矩阵(matrix)。把二维数组写成行(row)和列(column)的排列形式,可以有助于形象化地理解二维数组的逻辑结构。

6.2.1 怎样定义二维数组

怎样定义二维数组呢?其基本概念与方法和一维数组相似。如:

```
float pay[3][6];
```

以上定义了一个 float 型的二维数组,第 1 维有 3 个元素,第 2 维有 6 个元素。每一维的长度分别用一对方括号括起来。

二维数组定义的一般形式为

类型说明符 数组名[常量表达式][常量表达式];

例如:

```
float a[3][4],b[5][10];
```

定义 a 为 3×4(3 行 4 列)的数组,b 为 5×10(5 行 10 列)的数组。注意,不能写成

```
float a[3,4],b[5,10];                    //在一对方括号内写两个下标,错误
```

C 语言对二维数组采用这样的定义方式,使得二维数组可被看作一种特殊的一维数组:它的元素又是一个一维数组。例如,可以把 a 看作一个一维数组,它有 3 个元素:

```
a[0],a[1],a[2]
```

每个元素又是一个包含 4 个元素的一维数组,见图 6.7。

```
a[0]  ----  a[0][0]  a[0][1]  a[0][2]  a[0][3]
a[1]  ----  a[1][0]  a[1][1]  a[1][2]  a[1][3]
a[2]  ----  a[2][0]  a[2][1]  a[2][2]  a[2][3]
```

图 6.7

可以把 a[0],a[1],a[2]看作 3 个一维数组的名字。上面定义的二维数组可以理解为定义了 3 个一维数组,即相当于

```
float a[0][4],a[1][4],a[2][4];
```

此处把 a[0],a[1],a[2]看作一维数组名。C 语言的这种处理方法在数组初始化和用指针表示时显得很方便,这在以后会体会到。

C 语言中,二维数组中元素排列的顺序是按行存放的,即在内存中先顺序存放第 1 行的元素,接着再存放第 2 行的元素。图 6.8 表示对 a[3][4]数组存放的顺序。

假设数组 a 存放在从 2000 字节开始的一段内存单元中,一个元素占 4 个字节,前 16 个字节(2000～2015)存放序号为 0 的行中的 4 个元素,接着的 16 个字节(2016～2031)存放序号为 1 的行中的 4 个元素,余类推,如图 6.9 所示。

注意:用矩阵形式(如 3 行 4 列形式)表示二维数组,是逻辑上的概念,能形象地表示出行列关系。而在内存中,各元素是连续存放的,不是二维的,是线性的。这点务请明确。

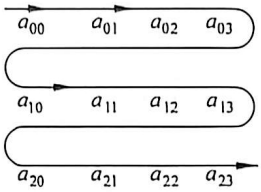

图 6.8

图 6.9

C语言还允许使用多维数组。有了二维数组的基础，再掌握多维数组是不困难的。例如，定义三维数组的方法如下：

```
float a[2][3][4];                    //定义三维数组 a,它有 2 页,3 行,4 列
```

多维数组元素在内存中的排列顺序为：第1维的下标变化最慢，最右边的下标变化最快。例如，上述三维数组的元素排列顺序为

```
a[0][0][0]→a[0][0][1]→a[0][0][2]→a[0][0][3]→a[0][1][0]→a[0][1][1]→a[0][1][2]→
a[0][1][3]→a[0][2][0]→a[0][2][1]→a[0][2][2]→a[0][2][3]→a[1][0][0]→a[1][0][1]→
a[1][0][2]→a[1][0][3]→a[1][1][0]→a[1][1][1]→a[1][1][2]→a[1][1][3]→a[1][2][0]→
a[1][2][1]→a[1][2][2]→a[1][2][3]
```

6.2.2 怎样引用二维数组的元素

二维数组元素的表示形式为

数组名[下标][下标]

例如，a[2][3]表示a数组中序号为2的行中序号为3的列的元素。下标应是整型表达式，如a[2－1][2＊2－1]。不要写成a[2,3]、a[2－1,2＊2－1]形式。

数组元素可以出现在表达式中，也可以被赋值，例如：

```
b[1][2]=a[2][3]/2
```

注意：在引用数组元素时，下标值应在已定义的数组大小的范围内。在这个问题上常出现错误。例如：

```
int a[3][4];                         //定义 a 为 3×4 的二维数组
  ⋮
a[3][4]=3;                           //不存在 a[3][4]元素
```

按以上的定义，数组a可用的“行下标”的范围为0～2，“列下标”的范围为0～3。用a[3][4]表示元素显然超过了数组的范围。

注意：请读者严格区分在定义数组时用的a[3][4]和引用元素时的a[3][4]的区别。

前者用a[3][4]来定义数组的维数和各维的大小，后者a[3][4]中的3和4是数组元素的下标值，a[3][4]代表行序号为3、列序号为4的元素（行序号和列序号均从0起算）。

6.2.3 二维数组的初始化

可以用“初始化列表”对二维数组初始化。

（1）分行给二维数组赋初值。例如：

```
int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

这种赋初值方法比较直观，把第1个花括号内的数据给第1行的元素，第2个花括号内的数据赋给第2行的元素……即按行赋初值。

（2）可以将所有数据写在一个花括号内，按数组元素在内存中的排列顺序对各元素赋初值。例如：

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

效果与前相同。但以第(1)种方法为好，一行对一行，界限清楚。用第(2)种方法如果数据多，则会写成一大片，容易遗漏，也不易检查。

（3）可以对部分元素赋初值。例如：

```
int a[3][4]={{1},{5},{9}};
```

它的作用是只对各行第1列（即序号为0的列）的元素赋初值，其余元素值自动为0。赋初值后数组各元素为

```
1    0    0    0
5    0    0    0
9    0    0    0
```

也可以对各行中的某一元素赋初值，例如：

```
int a[3][4]={{1},{0,6},{0,0,11}};
```

初始化后的数组元素如下：

```
1    0    0    0
0    6    0    0
0    0    11   0
```

这种方法对非0元素少时比较方便，不必将所有的0都写出来，只须输入少量数据。

也可以只对某几行元素赋初值：

```
int a[3][4]={{1},{5,6}};
```

数组元素为

```
1    0    0    0
5    6    0    0
0    0    0    0
```

第 3 行不赋初值。

也可以对第 2 行不赋初值，例如：

```
int a[3][4]={{1},{ },{9}};
```

(4) 如果对全部元素都赋初值(即提供全部初始数据)，则定义数组时对第 1 维的长度可以不指定，但第 2 维的长度不能省。例如：

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

与下面的定义等价：

```
int a[][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

系统会根据数据总个数和第 2 维的长度算出第 1 维的长度。数组一共有 12 个元素，每行 4 列，显然可以确定行数为 3。

在定义时也可以只对部分元素赋初值而省略第 1 维的长度，但应分行赋初值。例如：

```
int a[][4]={{0,0,3},{ },{0,10}};
```

这样的写法，能通知编译系统；数组共有 3 行。数组各元素为

```
0    0    3    0
0    0    0    0
0    10   0    0
```

从本节的介绍中可以看到：C 语言在定义数组和表示数组元素时采用 a[][]这种两个方括号的方式，对数组初始化时十分有用，它使概念清楚，使用方便，不易出错。

6.2.4　二维数组程序举例

【例 6.4】 将一个二维数组行和列的元素互换，存到另一个二维数组中。例如：

$$a=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}\qquad b=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$$

解题思路： 可以定义两个数组：数组 a 为 2 行 3 列，存放指定的 6 个数。数组 b 为 3 行 2 列，开始时未赋值。只要将 a 数组中的元素 a[i][j]存放到 b 数组中的 b[j][i]元素中即可。用嵌套的 for 循环即可完成此任务。

编写程序：

```
#include <stdio.h>
int main()
  {
    int a[2][3]={{1,2,3},{4,5,6}};
    int b[3][2],i,j;
    printf("array a:\n");
    for(i=0;i<=1;i++)                          //处理 a 数组中的一行中各元素
      {
       for (j=0;j<=2;j++)                      //处理 a 数组中某一列中各元素
```

```
            printf("%5d",a[i][j]);                    //输出a数组的一个元素
            b[j][i]=a[i][j];                          //将a数组元素的值赋给b数组相应元素

        printf("\n");

    printf("array b:\n");                             //输出b数组各元素
    for(i=0;i<=2;i++)                                 //处理b数组中一行中各元素

        for(j=0;j<=1;j++)                             //处理b数组中一列中各元素
            printf("%5d",b[i][j]);                    //输出b数组的一个元素
        printf("\n");

    return 0;
```

运行结果:

```
array a:
    1    2    3
    4    5    6
array b:
    1    4
    2    5
    3    6
```

【例 6.5】 有一个 3×4 的矩阵,要求编程序求出其中值最大的那个元素的值,以及其所在的行号和列号。

解题思路:先思考一下在打擂台时怎样确定最后的优胜者。先找出任一人站在台上,第 2 人上去与之比武,胜者留在台上。再上去第 3 人,与台上的人(即刚才的得胜者)比武,胜者留台上,败者下台。以后每一个人都是与当时留在台上的人比武。直到所有人都上台比过为止,最后留在台上的就是冠军。这就是"打擂台算法"。

解本题也是用"打擂台算法"。先让 a[0][0]作"擂主",把它的值赋给变量 max,max 用来存放当前已知的最大值,在开始时还未进行比较,把最前面的元素暂时认为是当前值最大的。然后让下一个元素 a[0][1]与 max 比较,如果 a[0][1]>max,则表示 a[0][1]是已经比过的数据中值最大的,把它的值赋给 max,取代了 max 的原值。以后依此处理,值大的赋给 max。直到全部比完后,max 就是最大的值。

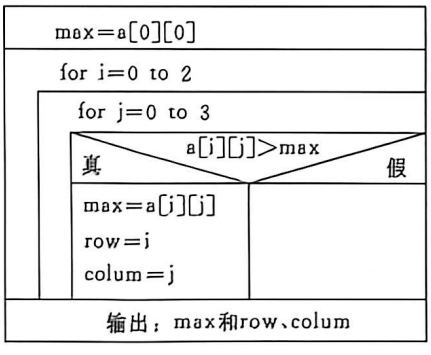

图 6.10

按此思路画出 N-S 图,见图 6.10。

编写程序:

根据流程图很容易写出程序:

```
#include <stdio.h>
int main()
  {int i,j,row=0,colum=0,max;
```