11.2-二维数组

**二维数组：**

顾名思义，具有两个下标的数组

**声明：**

数据类型 数组名[行数][列数] （={元素初值}）

int a[2][3]; //具有“两行三列”的二维数组，编译器分配6个int型变量的存储空间

可以想象成一个“矩阵”

**存储方式：**

相邻元素间的地址是连续的

int a[2][3],i,j;

for(i=0;i<2;i++)

{

for(j=0;j<3;j++)

{

cout<<"a["<<i<<"]["<<j<<"]的地址为:"<<&a[i][j]<<endl;

}

}

（int型变量大小为4个字节）

地址分配原则：

（1）所有的维度都是从小到大依次分配;

（2）先分配排列最“后”的维度——低维

（3）相邻元素间地址仍然是连续的（递增）

**初始化：**

1.按照元素（地址存储）的顺序进行依次赋值：

（1）全部明确赋初识值：

int a[2][3]={1，2，3，4，5，6}

（2）部分赋初识值

int a[2][3]={1,2,3,4} //a[1][1]==a[1][2]==0

（3）省略最高维的初始化：

int a[][3]={1，2，3，4，5，6} //最高维数=元素数/低维数

2.将二维数组看成“最高维数”个一维数组依次赋值：

（1）a[0][3]==a[1][2]==a[1][3]==0

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

int i,j;

for(i=0;i<3;i++)

{

for(j=0;j<4;j++)

{

cout<<"a["<<i<<"]["<<j<<"]的值为:"<<a[i][j]<<endl;

}

}

将a[3][4]看成一个具有三个元素（高维数）的一维“数组”；

每个“数组”的元素都是一个包含四个元素（低维数）的一维数组。

（2）省略最高维：

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

最高维是3

**访问：**

访问二维数组的元素必须同时指明每一个维度的下标值

根据第一个下标得到该“行”元素的**首地址**

根据第二个下标得到该元素对于首地址的**偏移量**

首地址：

数组的首地址为a[0][0]的地址；

第i行元素的首地址为a[i][0]的地址。

每个元素都是一个int型的变量

代码示例1：

编写代码，完成矩阵转置；同时判断该矩阵是不是对称矩阵。

对称矩阵：转置矩阵与原矩阵完全相等

输入：

第一行输入两个整数m,n，分别代表矩阵的行和列(1<m,n<=10);

接下来按照行数和列数按照矩阵的排列输入每个元素

输出：

先按照矩阵的排列输出转置后的矩阵结果；

然后在最后一行输出判定结果“YES”或“NO”

样例输入：

2 3

1 2 0

3 -1 4

样例输出：

1 3

2 -1

0 4

NO

int m,n;

cin>>m>>n;

int a[10][10]={0};

int i,j;

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

cin>>a[i][j];

}

}

for(j=0;j<n;j++)

{

for(i=0;i<m;i++)

{

cout<<a[i][j]<<" ";

}

cout<<endl;

}

if(m!=n)

{

cout<<"NO"<<endl;

}

else

{

int issym=1;

for(i=0;i<m;i++)

{

if(issym==0)

{

break;

}

for(j=0;j<n;j++)

{

if(i<j)

{

if(a[i][j]!=a[j][i])

{

cout<<"NO"<<endl;

issym=0;

break;

}

}

}

}

if(issym==1)

{

cout<<"YES"<<endl;

}

}

代码示例2：

题目描述

一个n×n 的方格图（标号由 11 开始）上有 m 个探测器，每个探测器的探测半径 为r ，问这n×n 个点中有多少个点能被探测到。

输入格式

第一行 输入3个整数 n,m,r ;

接下来 m 行，每行输入两个整数 x,y，表示第 i 个探测器的坐标。

输出格式

第一行按照网格图的“逐行”输出所有能被探测到的点的坐标

第二行输出能被探测到的点的个数

样例输入：

5 2 1

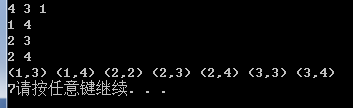
3 3

4 2

样例输出：

(2,3) (3,2) (3,3) (3,4) (4,1) (4,2) (4,3) (5,2)

8



int main()

{

int a[105][105]={0};

int n,m,r,sum=0;

int x,y;

cin>>n>>m>>r;

int i,j,k;

for(i=1;i<=m;i++)

{

cin>>x>>y; //边输入边计算，这样效率更高,不然还得再来个循环

a[x][y]=1; //标记探测器所在的点

for(k=1;k<=n;k++)

{

for(j=1;j<=n;j++)

{

double r1=sqrt(float((x-k)\*(x-k)+(y-j)\*(y-j)));//求出探测器与目标点的距离

if(r1<=r) //判断距离

{

a[k][j]=1;//标记目标点

}

}

}

}

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

if(a[i][j]==1)//能被探测到的目标点

{

sum++;

cout<<"("<<i<<","<<j<<") "; //输出坐标

}

}

}

cout<<endl;

cout<<sum; //输出被探测到的点的总数

return 0;

}