法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

第三课　　多维数组

林沐

内容概述

1.二维数组引入
2.二维数组与内存
3.多维数组初始化
4.例1-矩阵逆置
5.例2-杨辉三角
6.多维数组作为函数参数
7.例3-矩阵相乘
8.例4-旋转矩阵
9.例5-N皇后问题

二维数组引入

输入**班级数量**，每个班的**学生数量**，按照班级的学生数量读取各个班的**学生成绩**。计算每个班学生**平均成绩**，最后输出各个班**学生成绩**与班级**平均成绩**。

```
Please input class number: 3
Please input student number of all classes:
5 6 4
Please input all grades:
80 73 62 98 92
81 85 73 59 90 87
79 99 61 68

class 0 students' grade:
80 73 62 98 92
average = 81.00

class 1 students' grade:
81 85 73 59 90 87
average = 79.17

class 2 students' grade:
79 99 61 68
average = 76.75
```

```c
#include <stdio.h>
#define MAX_CLASS_NUM 20    //最多有20个班
#define MAX_STUDENT_NUM 50 //每个班最多50个人
int main(){
    int class_num;                                          //班级数量
    int students_num[MAX_CLASS_NUM];                        //每个班学生数量
    int grade[MAX_CLASS_NUM][MAX_STUDENT_NUM];              //各个班的各个同学的成绩
    double class_average[MAX_CLASS_NUM];                    //各个班平均分
    int i, j;

    printf("Please input class number: ");
    scanf("%d", &class_num);
    printf("Please input student number of all classes:\n");
    for (i = 0; i < class_num; i++){
        scanf("%d", &students_num[i]);
    }                                                       //读取数据
    printf("Please input all grades:\n");
    for (i = 0; i < class_num; i++){
        for (j = 0; j < students_num[i]; j++){
            scanf("%d", &grade[i][j]);
        }
    }

    for (i = 0; i < class_num; i++){
        double sum = 0;
        for (j = 0; j < students_num[i]; j++){
            sum += grade[i][j];
        }
        class_average[i] = sum / students_num[i];
    }                                                       //计算各个班级平均分

    printf("\n");
    for (i = 0; i < class_num; i++){
        printf("class %d students' grade:\n", i);
        for (j = 0; j < students_num[i]; j++){
            printf("%d ", grade[i][j]);
        }
        printf("\n");
        printf("average = %.2lf\n", class_average[i]);
        printf("\n");
    }                                                       //输出各个班的分数与平均分

    return 0;
}
```

二维数组与内存

```
number:
0028FF00 1      0028FF04 2      0028FF08 3      0028FF0C 4      0028FF10 5
0028FF14 6      0028FF18 7      0028FF1C 8      0028FF20 9      0028FF24 10
0028FF28 11     0028FF2C 12     0028FF30 13     0028FF34 14     0028FF38 15
number[0]:0028FF00
number[1]:0028FF14
number[2]:0028FF28
```

声明一个 3*5 的 二维数组:

int number [3] [5] ;

数组类型 数组名 第一维 第二维

这些元素可以组成一个3*5的矩阵:

[0][0] [0][1] [0][2] [0][3] [0][4]

[1][0] [1][1] [1][2] [1][3] [1][4]

[2][0] [2][1] [2][2] [2][3] [2][4]

第一个索引选择行，第二个索引选择列，各个元素按照顺序存储在内存中

也可将二维数组看作是数组的数组:

number[0] [0][1][2][3][4]

number[1] [0][1][2][3][4]

number[2] [0][1][2][3][4]

```c
#include <stdio.h>
int main(){
    int number[3][5];
    int count = 0;
    int i, j;
    for (i = 0; i < 3; i++){
        for (j = 0; j < 5; j++){
            number[i][j] = ++count;
        }
    }
    printf("number:\n");
    for (i = 0; i < 3; i++){
        for (j = 0; j < 5; j++){
            printf("%p %d\t",
                &number[i][j], number[i][j]);
        }
    }
    for (i = 0; i < 3; i++){
        printf("number[%d]:%p\n", i, number[i]);
    }
    return 0;
}
```

//打印每个元素的地址与元素值

//打印number[0]、number[1]、number[2]的值

多维数组初始化

多维数组和一维数组的初始化类似，将对应的**初始值**放到大括号中，例如**二维数组**需要有**两层**嵌套括号初始化，使用**两层循环**访问对应的元素；**三维数组**有3层嵌套括号初始化，使用**三层循环**访问对应的元素。

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15

10 20 30 40 50
60 70 80 90 100
110 120 130 140 150

请按任意键继续. . .
```

```c
#include <stdio.h>
int main(){
    int number[3][5] = {                    //二维数组初始化
        {1, 2, 3, 4, 5},
        {6, 7, 8, 9, 10},
        {11, 12, 13, 14, 15}
    };
    int number2[3][5] = {0};  //初始化数组各元素的值为0
    int number3[2][3][5] = {
        {
            {1, 2, 3, 4, 5},
            {6, 7, 8, 9, 10},          //三维数组初始化，三层括号
            {11, 12, 13, 14, 15}
        },
        {
            {10, 20, 30, 40, 50},
            {60, 70, 80, 90, 100},
            {110, 120, 130, 140, 150}
        }
    };
    int i, j, k;                                    //使用三层循环遍历三维数组
    for (i = 0; i < 2; i++){
        for (j = 0; j < 3; j++){
            for (k = 0; k < 5; k++){
                printf("%d ", number3[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }
    return 0;
}
```

例1-矩阵转置

问题描述：已知一个n*m的矩阵，矩阵的行数为n,列数为m。实现**矩阵的转置**，即行列互换。

输入与输出要求：输入两个整数n、m，代表矩阵的行数与列数，n、m的范围均是1—100。然后是n*m个整数，即此矩阵的元素。输出经过矩阵转置得到的新矩阵，新矩阵占m行，n列。

Input Sample:

3 5

5 5 5 5 5

3 3 3 3 3

1 1 1 1 1

Output Sample:

5 3 1

5 3 1

5 3 1

5 3 1

5 3 1

Input Sample:

4 4

2 3 1 99

23 231 367 198

15 29 37 1

5 0 -12 4

Output Sample:

2 23 15 5

3 231 29 0

1 367 37 -12

99 198 1 4

思考：

1.需要**几个**二维数组来解决这个问题?

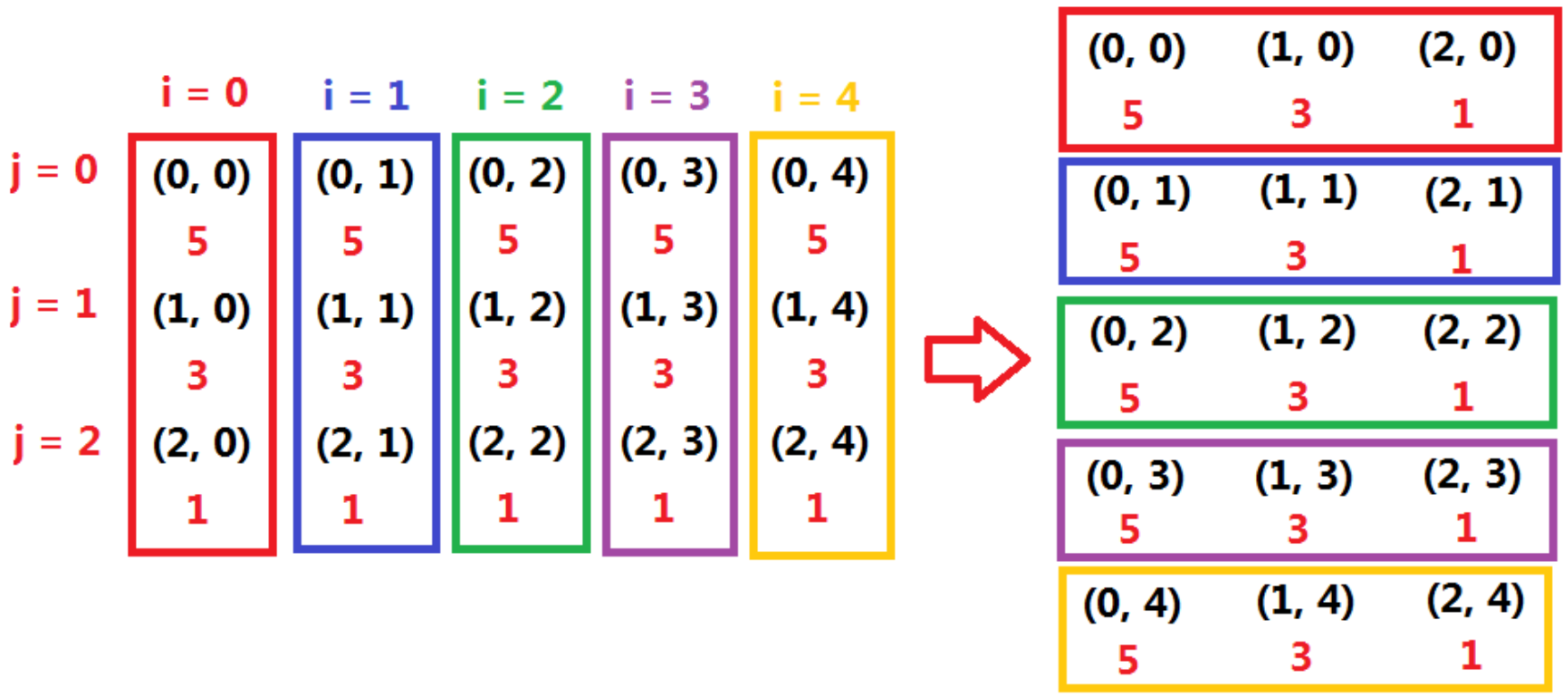
2.若只用**一个二维数组**，当读入原数据后，是否需要对读入数据后的二维数组进行**修改**?

3.若**不修改**输入数据后的二维数组，如何通过**遍历并打印**二维数组的元素来解决这个问题?

思考**1分钟**。

例1-算法设计

若不改变输入时矩阵的各个元素的值，将矩阵逆置后的效果打印出来，需要两层循环，外层循环矩阵的列(0至m-1)，内层循环矩阵的行(0至n-1)，将矩阵的元素进行打印。每次结束内层循环时，打印一个空行。



例1-课堂练习

```
#include <stdio.h>
#define MAXN 100
int main(){
    int n, m;
    int matrix[MAXN][MAXN];
    int i, j;
    scanf("%d %d", &n, &m);
    for (i = 0; i < n; i++){
        for (j = 0; j < m; j++){
            scanf("%d", 1 );
        }
    }
    for (i = 0; i < m; i++){
        for (j = 0; j < n; j++){
            printf("%d ", 2 );
        }
        3
    }
    return 0;
}
```

3分钟，填写代码，有问题提出！

例1-实现与测试

```
#include <stdio.h>
#define MAXN 100
int main(){
    int n, m;
    int matrix[MAXN][MAXN];
    int i, j;
    scanf("%d %d", &n, &m);
    for (i = 0; i < n; i++){
        for (j = 0; j < m; j++){
            scanf("%d", &matrix[i][j]);
        }
    }
    for (i = 0; i < m; i++){
        for (j = 0; j < n; j++){
            printf("%d ", matrix[j][i]);
        }
        printf("\n");
    }
    return 0;
}
```

```
3 5
5 5 5 5 5
3 3 3 3 3
1 1 1 1 1
5 3 1
5 3 1
5 3 1
5 3 1
5 3 1
请按任意键继续. . .
```

例2-杨辉三角

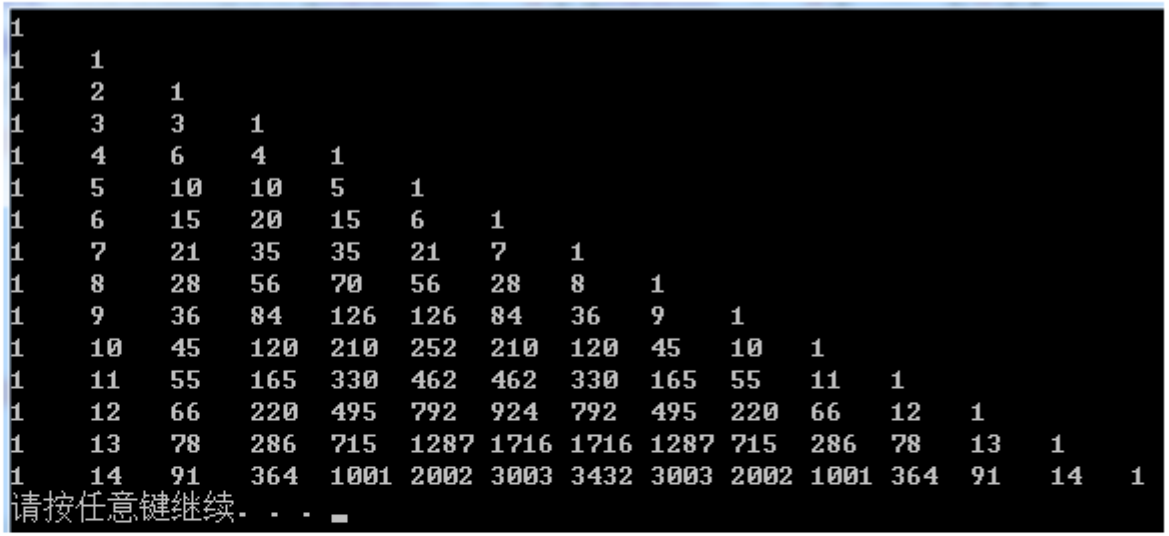
著名的杨辉三角如下：

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
1   6   15  20  15  6   1
1   7   21  35  35  21  7   1
1   8   28  56  70  56  28  8   1
1   9   36  84  126 126 84  36  9   1
...
```

将杨辉三角的前15行打印出来。

(a+b)^n，多项式的系数

程序运行效果：



思考：

打印杨辉三角的n行需要声明多大的二维数组?

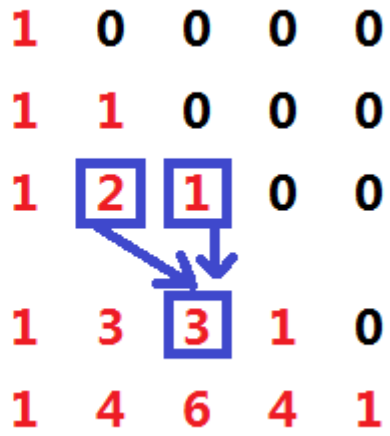
杨辉三角的计算方法是什么?

思考1分钟。

例2-思考

存储5阶的杨辉三角，需要一个5*5的二维数组，存储n阶杨辉三角，需要一个n*n的二维数组；杨辉三角的某一行某一列元素是，该元素的上一行的前一列元素与上一行的该列的元素的和。

1	0	0	0	0
1	1	0	0	0
1	2	1	0	0
1	3	3	1	0
1	4	6	4	1



思考:

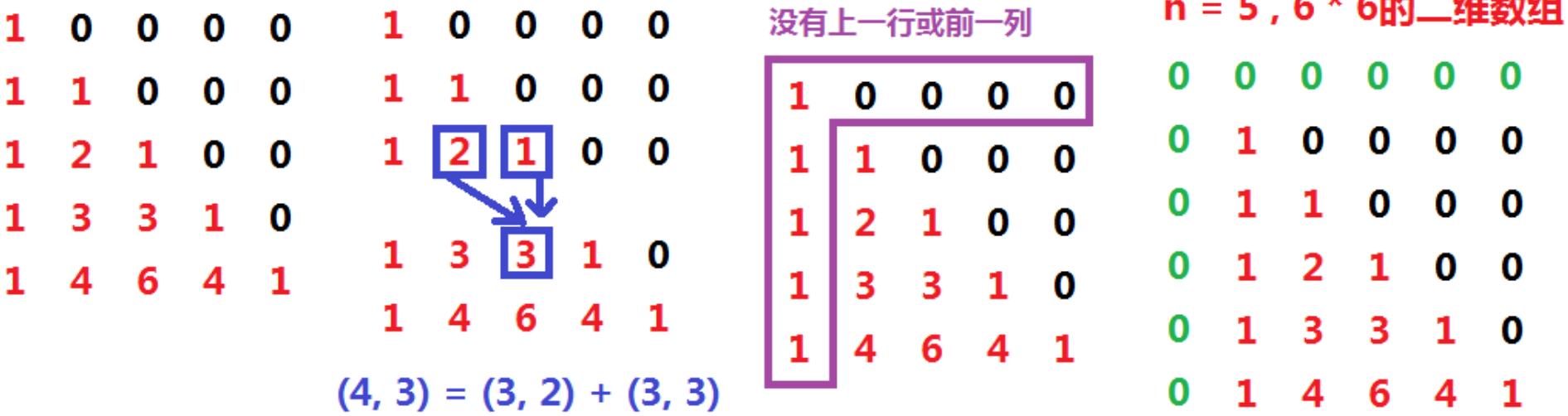
1.如何遍历该二维数组，计算杨辉三角的各项元素。

2.杨辉三角的各项元素计算方法是否一致，是否有边界问题，如何解决。

例2-分析

对于杨辉三角的第i行与第j列，即为该元素的**上一行与上一列**加上**上一行与该列**的元素值，即triangle(i, j) = triangle(i-1, j-1) + triangle(i-1, j)

对于**第一行或第一列**的元素，由于这些元素**没有**上一行或上一列，会遇到**边界问题**，处理边界问题时，在**最外层**设置一圈0，即使用**triangle[n+1][n+1]**保存n阶的杨辉三角，从第1行与第1列开始使用。



例2-算法设计

初始化triangle[1][1] = 1，

第1层循环，循环i，第2行计算至第15行，

第2层循环，循环j，第1列至第i列，

计算元素triangle[i][j]，即**上一行的前一列**triangle[i-1][j-1]元素值加**上一行的该列**元素triangle[i-1][j] 。

	j=0	1	2	3	4	5
i = 0	0	0	0	0	0	0
i = 1	0	1	0	0	0	0
i = 2	0	1	1	0	0	0
i = 3	0	1	2	1	0	0
i = 4	0	1	3	3	1	0
i = 5	0	1	4	6	4	1

	j=0	1	2	3	4	5
i = 0	0	0	0	0	0	0
i = 1	0	**1**	0	0	0	0
i = 2	0	0	0	0	0	0
i = 3	0	0	0	0	0	0
i = 4	0	0	0	0	0	0
i = 5	0	0	0	0	0	0

	j=0	1	2	3	4	5
i = 0	0	0	0	0	0	0
i = 1	0	1	0	0	0	0
i = 2	0	**1**	**1**	0	0	0
i = 3	0	0	0	0	0	0
i = 4	0	0	0	0	0	0
i = 5	0	0	0	0	0	0

	j=0	1	2	3	4	5
i = 0	0	0	0	0	0	0
i = 1	0	1	0	0	0	0
i = 2	0	1	1	0	0	0
i = 3	0	**1**	**2**	**1**	0	0
i = 4	0	0	0	0	0	0
i = 5	0	0	0	0	0	0

	j=0	1	2	3	4	5
i = 0	0	0	0	0	0	0
i = 1	0	1	0	0	0	0
i = 2	0	1	1	0	0	0
i = 3	0	1	2	1	0	0
i = 4	0	**1**	**3**	**3**	**1**	0
i = 5	0	0	0	0	0	0

	j=0	1	2	3	4	5
i = 0	0	0	0	0	0	0
i = 1	0	1	0	0	0	0
i = 2	0	1	1	0	0	0
i = 3	0	1	2	1	0	0
i = 4	0	1	3	3	1	0
i = 5	0	**1**	**4**	**6**	**4**	**1**

例2-课堂练习

```
#include <stdio.h>

#define MAXN 15
int main(){
    int triangle[MAXN+1][MAXN+1] = {0};  //设置一个更大一圈的二维数组
    int i, j;

    triangle[1][1] = 1;  //初始化杨辉三角第一个元素

    for (i = 2; 1 ; i++){

        for (j = 1; 2 ; j++){

            triangle[i][j] = 3
        }
    }
    for (i = 1; i <= MAXN; i++){
        for (j = 1; j <= i; j++){
            printf("%-5d", triangle[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

3分钟，填写代码，有问题提出！

例2-实现与测试

```
#include <stdio.h>
#define MAXN 15
int main(){
    int triangle[MAXN+1][MAXN+1] = {0};  //设置一个更大一圈的二维数组
    int i, j;
    triangle[1][1] = 1;   //初始化杨辉三角第一个元素
    for (i = 2; i <= MAXN; i++){

        for (j = 1; j <= i ; j++){

            triangle[i][j] = triangle[i-1][j-1] + triangle[i-1][j];
        }
    }
    for (i = 1; i <= MAXN; i++){
        for (j = 1; j <= i; j++){
            printf("%-5d", triangle[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
1    6    15   20   15   6    1
1    7    21   35   35   21   7    1
1    8    28   56   70   56   28   8    1
1    9    36   84   126  126  84   36   9    1
1    10   45   120  210  252  210  120  45   10   1
1    11   55   165  330  462  462  330  165  55   11   1
1    12   66   220  495  792  924  792  495  220  66   12   1
1    13   78   286  715  1287 1716 1716 1287 715  286  78   13   1
1    14   91   364  1001 2002 3003 3432 3003 2002 1001 364  91   14   1
请按任意键继续. . .
```

多维数组作为函数参数

```
3 5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
2d array sum = 120
2 3 5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
3d array sum = 240
```

在函数设计时，将多维数组当作**函数参数**时，需要指明数组**除第一维度的其他维度最大长度**(若指针形式无需指定)，并给出每一维的**有效长度**。

```c
#include <stdio.h>

//需要明确第二维最大长度
int compute_2D_array_sum(int array[][10],
                         int n, int m){
    int i, j;
    int sum = 0;
    //指定有效长度
    for (i = 0; i < n; i++){
        for (j = 0; j < m; j++){
            sum += array[i][j];
        }
    }
    return sum;
}

//需要明确第二、三维最大长度
int compute_3D_array_sum(int array[][10][10],
                         int n, int m, int p){
    int i, j, k;
    int sum = 0;
    //指定有效长度
    for (i = 0; i < n; i++){
        for (j = 0; j < m; j++){
            for (k = 0; k < p; k++){
                sum += array[i][j][k];
            }
        }
    }
    return sum;
}
```

```c
int main(){
    int a_2d[10][10] = {0};
    int a_3d[10][10][10] = {0};
    int n, m;
    int x, y, z;
    int i, j, k;
    scanf("%d %d", &n, &m);
    for (i = 0; i < n; i++){
        for (j = 0; j < m; j++){
            scanf("%d", &a_2d[i][j]);
        }
    }
    printf("2d array sum = %d\n",
           compute_2D_array_sum(a_2d, n, m));
    scanf("%d %d %d", &x, &y, &z);
    for (i = 0; i < x; i++){
        for (j = 0; j < y; j++){
            for (k = 0; k < z; k++)
                scanf("%d", &a_3d[i][j][k]);
        }
    }
    printf("3d array sum = %d\n",
           compute_3D_array_sum(a_3d, x, y, z));
    return 0;
}
```

例3-矩阵相乘

设计函数matrix_multipy，**函数原型：**

void matrix_multipy(int A[][MAX],int B[][MAX],int C[][MAX],int n,int m,int p);

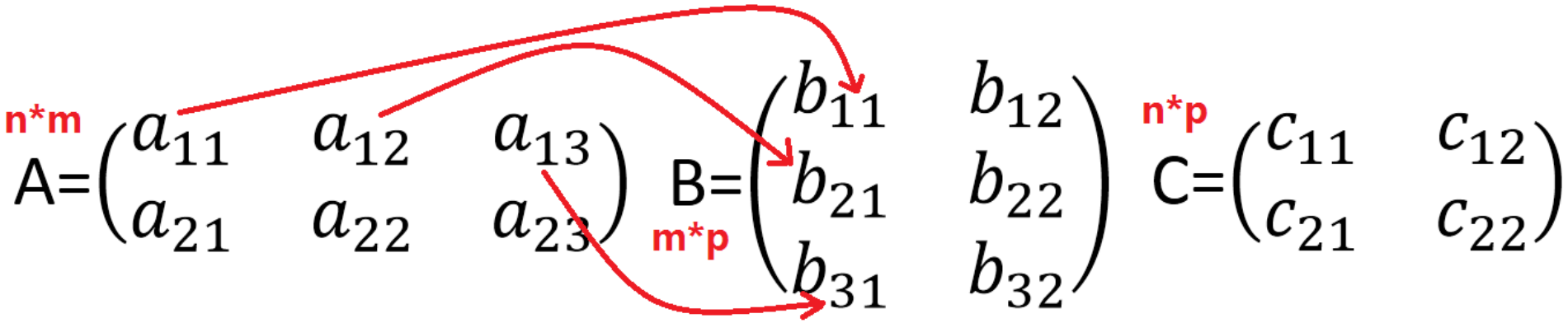
函数功能：将二维数组A中存储的是n*m的矩阵与二维数组B中存储的m*p的**矩阵相乘**，得到n*p的矩阵，存储在二维数组C中。

输入与输出要求:

输入三个整数n、m、p，n、m代表二维数组A的行数与列数，m、p代表二维数组B的行数与列数。n、m、p的范围均是1—100。然后是n*m个整数，即二维数组A的元素。最后是m*p个整数，即二维数组B的元素。输出二维数组C中的元素，n行p列。

Input Sample:	Output Sample:	Input Sample:	Output Sample:
3 4 3	587 640 1295	2 3 5	14 32 50 68 86
10 20 25 3	728 1580 850	1 2 3	32 77 122 167 212
20 10 4 50	470 420 1637	4 5 6	
4 30 10 5		1 4 7 10 13	
20 25 3		2 5 8 11 14	
10 4 50		3 6 9 12 15	
7 10 10			
4 20 5			

例3-思考与分析



$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \quad B=\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} \quad C=\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

如果

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21} + a_{13} * b_{31}$$
$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22} + a_{13} * b_{32}$$
$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31}$$
$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22} + a_{23} * b_{32}$$

则，C = A * B，

即C是矩阵A与矩阵B的成绩

思考：

n*m的二维数组A与**m*p**的二维数组B相乘，得到**n*p**的二维数组C，需要**几层**循环计算矩阵相乘，每层循环都在访问矩阵的哪些元素。设计**矩阵相乘**的算法。

思考**1分钟**。

例3-算法设计

n*m

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

$$B=\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

m*p

n*p

$$C=\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

如果

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21} + a_{13} * b_{31}$$
$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22} + a_{13} * b_{32}$$
$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31}$$
$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22} + a_{23} * b_{32}$$

则，C = A * B，

即C是矩阵A与矩阵B的成绩

n*m的二维数组A与m*p的二维数组B相乘，得到n*p的二维数组C:

需要**三层循环**计算:

第一层使用i，循环矩阵A的行，即矩阵C的行，

第二层使用j，循环矩阵B的列，即矩阵C的列，

第三层使用k，循环矩阵A的列，矩阵B的行，将**A[i][k]**与**B[k][j]**的乘积累加到**C[i][j]**。

例3-课堂练习

```
#define MAXN 100
void matrix_multipy(int A[][MAXN],
                    int B[][MAXN], int C[][MAXN],
                    int n, int m, int p){
    int i, j, k;
    for (i = 0; i < n; i++){
        for (j = 0; j < p; j++){
            C[i][j] = 0;
        }
    }
    for (i = 0; i < [1]; i++){

        for (j = 0; j < [2]; j++){

            for (k = 0; k < [3]; k++){

                [4] += [5];
            }
        }
    }
}
```

//矩阵A是n*m的，矩阵B是m*p的，矩阵C是n*p的

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

$$B=\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

$$C=\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21} + a_{13} * b_{31}$$
$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22} + a_{13} * b_{32}$$
$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31}$$
$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22} + a_{23} * b_{32}$$

3分钟，填写代码，有问题提出！

例3-实现

```
#define MAXN 100
void matrix_multipy(int A[][MAXN],
                    int B[][MAXN], int C[][MAXN],
                    int n, int m, int p){
    int i, j, k;
    for (i = 0; i < n; i++){
        for (j = 0; j < p; j++){
            C[i][j] = 0;
        }
    }
    for (i = 0; i < n ; i++){

        for (j = 0; j < p ; j++){

            for (k = 0; k < m  ; k++){

                C[i][j] += A[i][k] * B[k][j] ;

            }
        }
    }
}
```

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

$$B=\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

$$C=\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21} + a_{13} * b_{31}$$
$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22} + a_{13} * b_{32}$$
$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21} + a_{23} * b_{31}$$
$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22} + a_{23} * b_{32}$$

例3-测试

```
int main(){
    int A[MAXN][MAXN];
    int B[MAXN][MAXN];
    int C[MAXN][MAXN];
    int n, m, p;
    int i, j;
    scanf("%d %d %d", &n, &m, &p);
    for (i = 0; i < n; i++){
        for (j = 0; j < m; j++){
            scanf("%d", &A[i][j]);
        }
    }
    for (i = 0; i < m; i++){
        for (j = 0; j < p; j++){
            scanf("%d", &B[i][j]);
        }
    }
    matrix_multipy(A, B, C, n, m, p);
    for (i = 0; i < n; i++){
        for (j = 0; j < p; j++){
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```
3 4 3
10 20 25 3
20 10 4 50
4 30 10 5
20 25 3
10 4 50
7 10 10
4 20 5
587 640 1295
728 1580 850
470 420 1637
```

课间休息10分钟！

有问题提出！

例4-旋转矩阵

问题描述：旋转矩阵是一个n*n的矩阵，将整数1到n*n按照**旋转的方式**顺序装入一个n*n的旋转矩阵中。

输入与输出要求：输入一个整数n，代表旋转矩阵的**阶数**，n的范围是1—100。输出n*n的旋转矩阵。

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

Input Sample:

5

Output Sample:

```
1  2  3  4  5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
```

Input Sample:

8

Output Sample:

```
1  2  3  4  5  6  7  8
28 29 30 31 32 33 34 9
27 48 49 50 51 52 35 10
26 47 60 61 62 53 36 11
25 46 59 64 63 54 37 12
24 45 58 57 56 55 38 13
23 44 43 42 41 40 39 14
22 21 20 19 18 17 16 15
```

例4-思考与分析

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

Input Sample:

5

Output Sample:

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Input Sample:

8

Output Sample:

1	2	3	4	5	6	7	8
28	29	30	31	32	33	34	9
27	48	49	50	51	52	35	10
26	47	60	61	62	53	36	11
25	46	59	64	63	54	37	12
24	45	58	57	56	55	38	13
23	44	43	42	41	40	39	14
22	21	20	19	18	17	16	15

思考：

1.如何**遍历**数组，可以将1至n*n按照**旋转的方式**填进n*n的二维数组array中？

2.解决该问题，是否还可以使用二维数组的一般遍历方式按照**先行后列**的方式进行遍历？

3.需要设计**几层循环**解决该问题，思考整体算法。

例4-算法设计

设置四个方向，分别为右(0)、下(1)、左(2)、上(3)，方向变量direction，初始值为0(右)； 当前需要填充的二维数组的下标(位置)，行下标x，列下标y，初始值x = 0， y = 0(左上角)。

将i从1循环至n*n ，

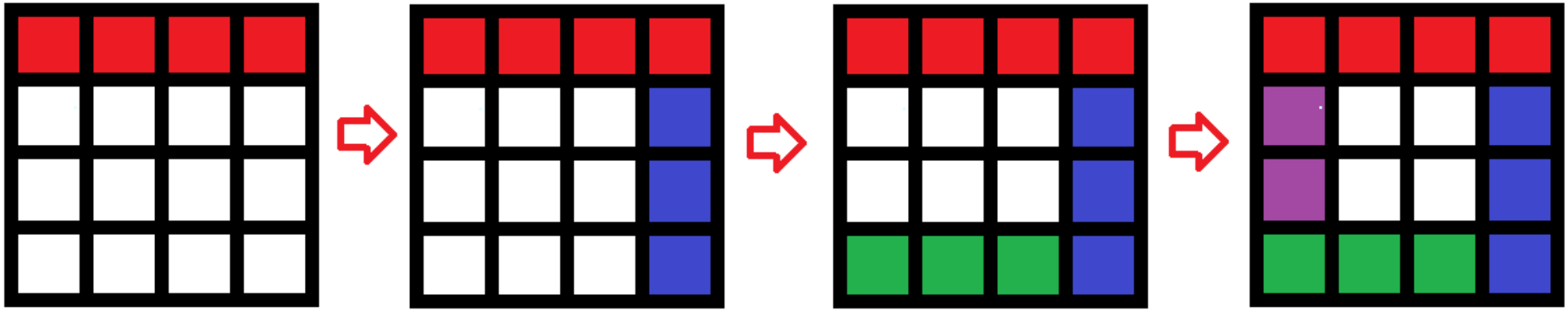
array[x][y] = i;

当方向向右:检查是否可以向右走一步，是则向右一步，否则方向改为向下，并向下走一步。

当方向向下:检查是否可以向下走一步，是则向下一步，否则方向改为向左，并向左走一步。

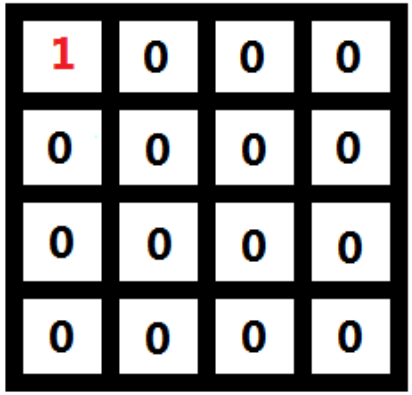
当方向向左:检查是否可以向左走一步，是则向左一步，否则方向改为向上，并向上走一步。

当方向向上:检查是否可以向上走一步，是则向上一步，否则方向改为向右，并向右走一步。

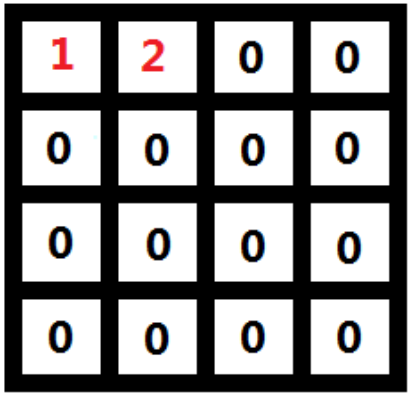


例4-算法设计

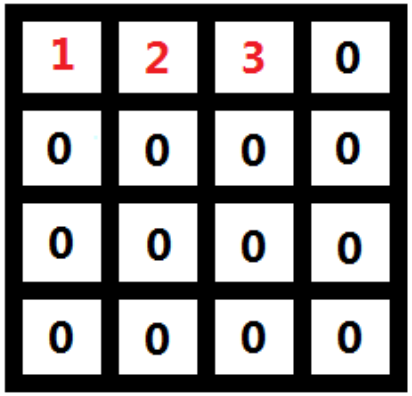
x = 0, y = 0, i = 1



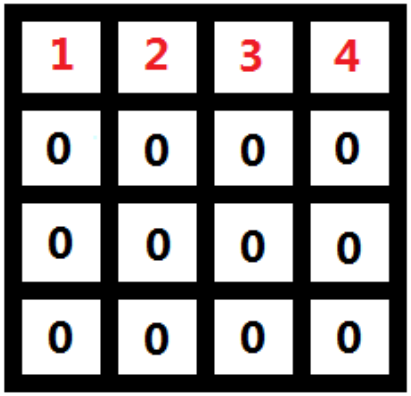
x = 0, y = 1, i = 2



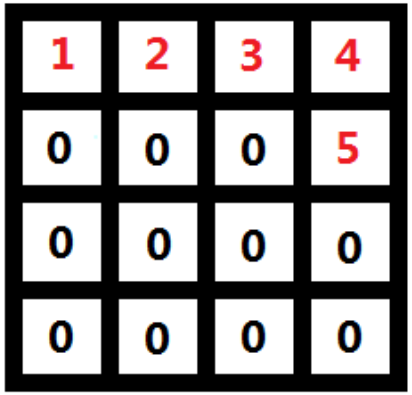
x = 0, y = 2, i = 3



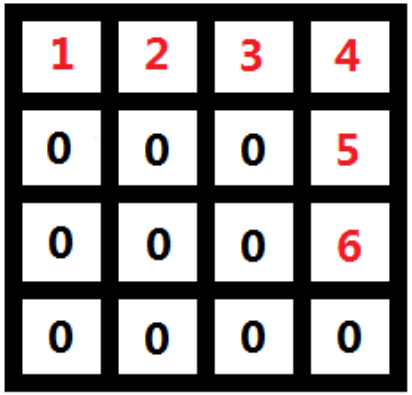
x = 0, y = 3, i = 4



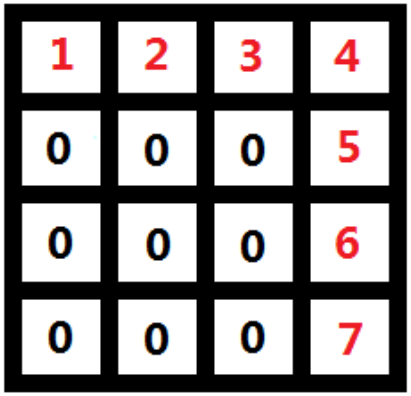
x = 1, y = 3, i = 5



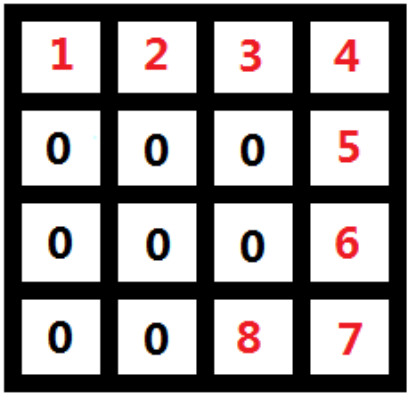
x = 2, y = 3, i = 6



x = 3, y = 3, i = 7



x = 3, y = 2, i = 8



例4-课堂练习

```
#include <stdio.h>

#define RIGHT 0
#define DOWN 1
#define LEFT 2
#define UP 3

int main(){
    int array[100][100] = {0};
    int n;
    scanf("%d", &n);
    int direction = RIGHT;
    int m = n * n;
    int x = 0;
    int y = 0;
    int i, j;
    for (i = 1; i <= m; i++){
        1
        if (direction == RIGHT){
            if (                2                ){
                x++;
                direction = DOWN;
            }
            else{
                y++;
            }
        }
        else if (       3       ){
            if (x + 1 == n || array[x+1][y] != 0){
                y--;
                direction = LEFT;
            }
            else{
                4
            }
        }
        else if (direction == LEFT){
            if (y == 0 || array[x][y-1] != 0){
                x--;
                direction = UP;
            }
            else{
                y--;
            }
        }
        else if (direction == UP){
            if (x == 0 || array[x-1][y] != 0){
                5
                direction = RIGHT;
            }
            else{
                x--;
            }
        }
        array[x][y] = i;
    }
    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            printf("%-4d", array[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

例4-实现与测试

```
5
1    2    3    4    5
16   17   18   19   6
15   24   25   20   7
14   23   22   21   8
13   12   11   10   9
```

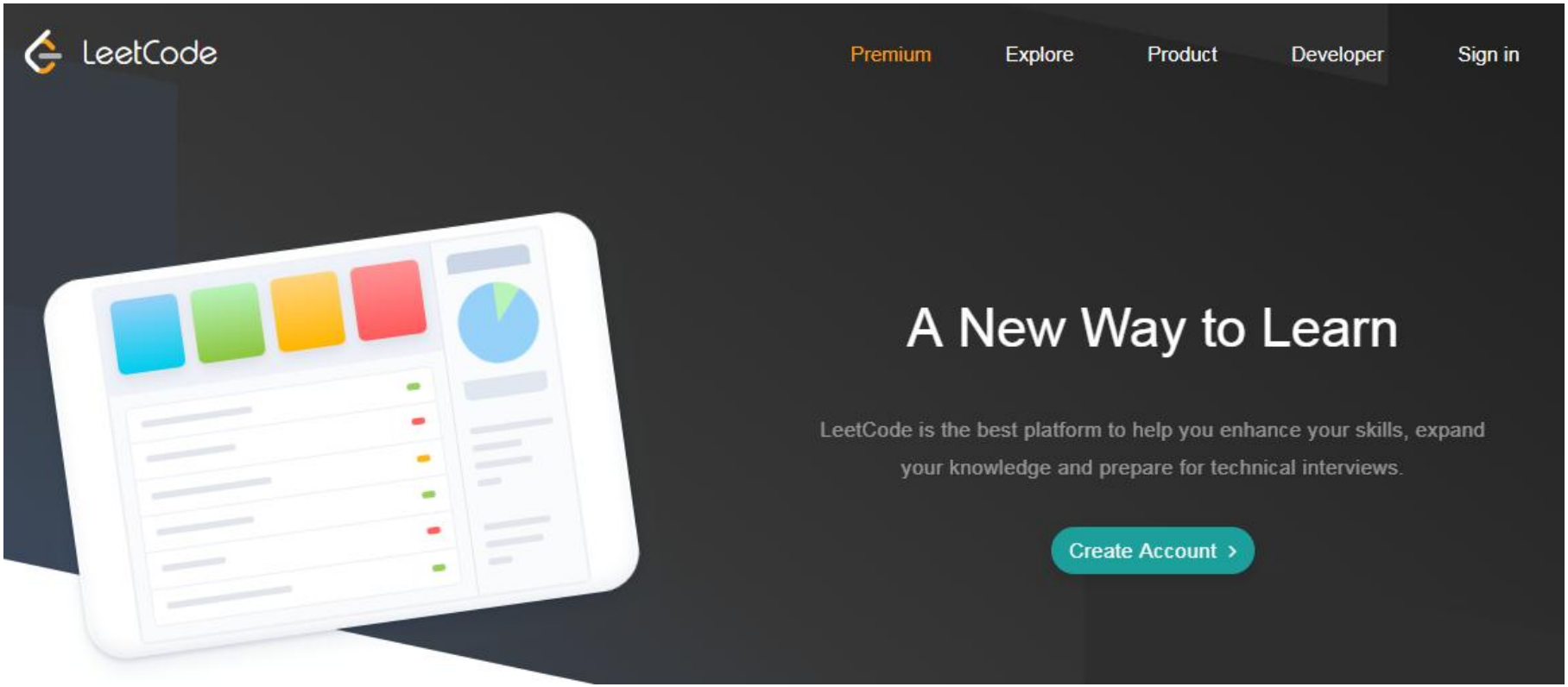
```c
#include <stdio.h>

#define RIGHT 0
#define DOWN 1
#define LEFT 2
#define UP 3

int main(){
    int array[100][100] = {0};
    int n;
    scanf("%d", &n);
    int direction = RIGHT;
    int m = n * n;
    int x = 0;
    int y = 0;
    int i, j;
    for (i = 1; i <= m; i++){
        array[x][y] = i;
        if (direction == RIGHT){
            if ( y + 1 == n || array[x][y+1] != 0 ){
                x++;
                direction = DOWN;
            }
            else{
                y++;
            }
        }
        else if ( direction == DOWN ){
            if (x + 1 == n || array[x+1][y] != 0){
                y--;
                direction = LEFT;
            }
            else{
                x++;
            }
        }
        else if (direction == LEFT){
            if (y == 0 || array[x][y-1] != 0){
                x--;
                direction = UP;
            }
            else{
                y--;
            }
        }
        else if (direction == UP){
            if (x == 0 || array[x-1][y] != 0){
                y++;
                direction = RIGHT;
            }
            else{
                x--;
            }
        }
    }
    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            printf("%-4d", array[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Leetcode简介

题目平均**质量很高**的面试题集合网站。站内题目提供**算法类型分类**、**公司分类**，并提供**测试数据**帮助我们查找问题的错误。



例4-旋转矩阵-leetcode版本

59. Spiral Matrix II

题目

描述

Given a positive integer *n*, generate a square matrix filled with elements from 1 to n^2 in spiral order.

Example:

```
Input: 3
Output:
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
```

参考样例，输入一个3，输出一个二维数组

C

语言选择，包括C/C++/python/Java/.../Go

```c
/**
 * Return an array of arrays.
 * Note: The returned array must be malloced, assume caller calls free().
 */
int** generateMatrix(int n) {

}
```

//代码提交，例如C语言提交函数，C++提交类等

例4-旋转矩阵-leetcode实现

```
int** generateMatrix(int n) {        //动态内存分配
    int **array = (int **)malloc(sizeof(int *) * n);
    int i, j;
    for (i = 0; i < n; i++){
        array[i] = (int *)malloc(sizeof(int) * n);
    }
    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            array[i][j] = 0;
        }
    }
    int direction = RIGHT;
    int m = n * n;
    int x = 0;
    int y = 0;
    ●●●
    return array;
}
```

例4-旋转矩阵-leetcode测试

```
int main(){
    int **array = NULL;
    int n = 5;
    int i, j;
    array = generateMatrix(n);
    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            printf("%-4d", array[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Spiral Matrix II

Submission Detail

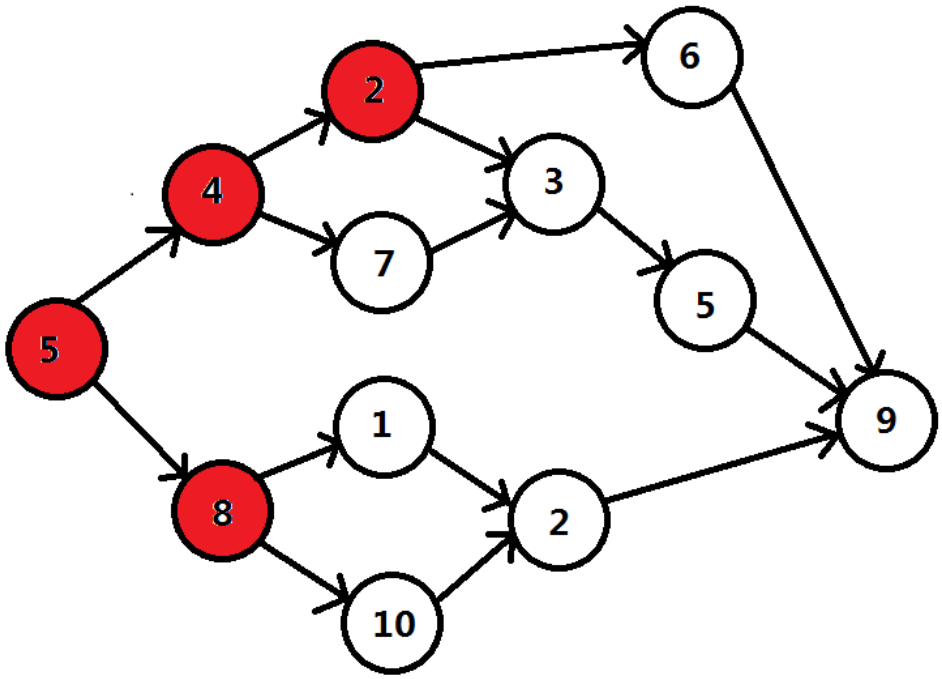
20 / 20 test cases passed. Status: Accepted

Runtime: **0 ms** Submitted: **0 minutes ago**

```
1   2   3   4   5
16  17  18  19  6
15  24  25  20  7
14  23  22  21  8
13  12  11  10  9
请按任意键继续. . .
```

算法设计:回溯搜索法

回溯法又称为**试探法**，但当**探索**到某一步时，发现原先**选择达不到目标**，就**退回一步重新选择**，这种**走不通就退回再走**的技术为回溯法。



找出路径上值的和大于30的所有路径

5 4 2 6 9	sum = 26
5 4 2 3 5 9	sum = 28
5 4 7 3 5 9	sum = 33
5 8 1 2 9	sum = 25
5 8 10 2 9	sum = 34

例5-N皇后问题

N皇后问题是计算机科学中**最为经典**的问题之一，该问题可**追溯**到1848年，由国际西洋棋棋手马克斯·贝瑟尔于提出了**8皇后**问题。将N个皇后放摆放在N*N的棋盘中，**互相不可攻击**，有多少种**摆放方式**?

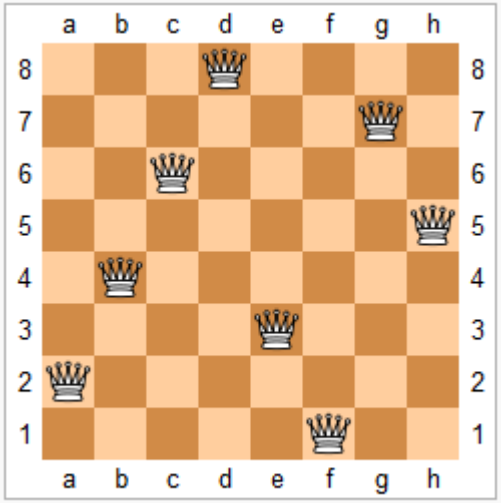
//传入皇后的个数(即棋盘的大小，传出摆放方法的个数

```
int totalNQueens(int n) {

}
```

```
[
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // Solution 2
  "Q...",
  "...Q",
  ".Q.."]
]
```



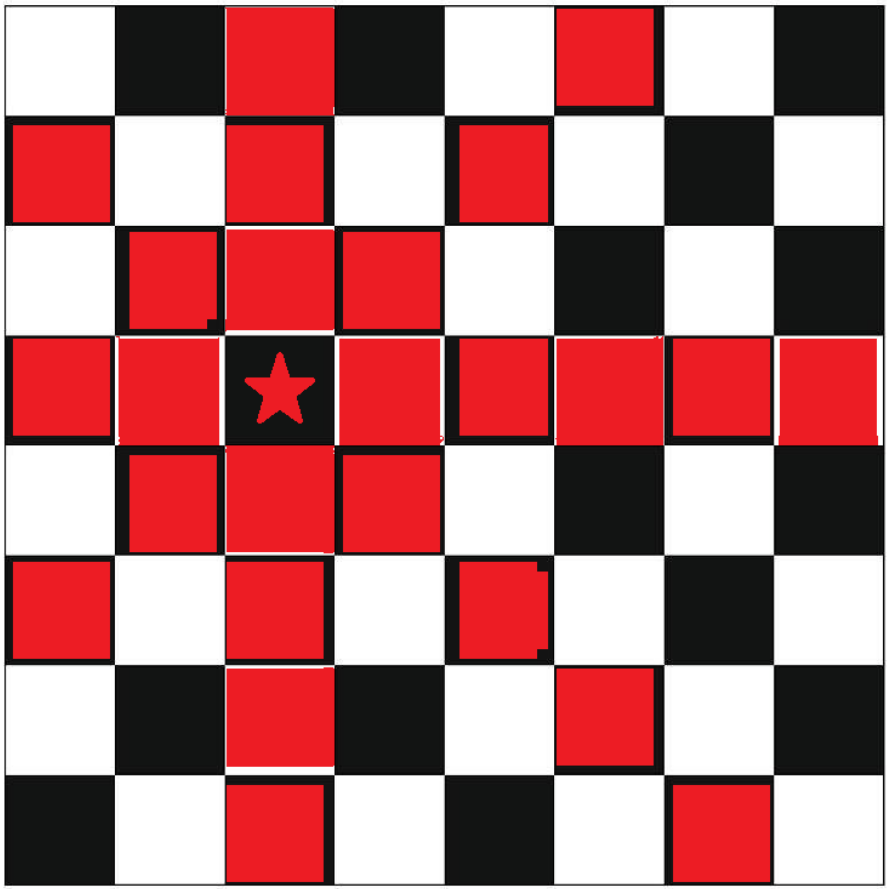
One solution to the eight queens puzzle

选自 **LeetCode 52. N-Queens II**
https://leetcode.com/problems/n-queens-ii/description/

例5-皇后的攻击范围

若在棋盘上已放置一个皇后，它实际上**占据**了哪些位置？

以这个皇后为中心，上、下、左、右、左上、左下、右上、右下，**8个方向**的位置全部**被占据**。



思考:

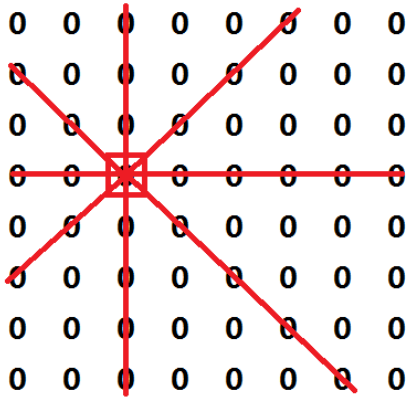
若在棋盘上放置一个皇后，如右图，标记为**红色位置**即不可再放其他皇后了，如何**设计算法**与**数据存储**，实现这一过程？

例5-棋盘与皇后表示

使用二维数组mark[][]表示一张空棋盘:

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

假设在(x, y)位置放置一个皇后，
即数组的第x行，第y列放置皇后:
如x=4, y=3;第4行，第3列



设置方向数组:

(x-1, y-1)	(x-1, y)	(x-1, y+1)
(x, y-1)	(x, y)	(x, y+1)
(x+1, y-1)	(x+1, y)	(x+1, y+1)

```
static const int dx[] = {-1, 1, 0, 0, -1, -1, 1, 1};
static const int dy[] = {0, 0, -1, 1, -1, 1, -1, 1};
```

按照方向数组的8个方向分别延伸N个距离，只要不超过边界，
mark[][] = 1

0	0	1	0	0	1	0	0
1	0	1	0	1	0	0	0
0	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1
0	1	1	1	0	0	0	0
1	0	1	0	1	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	0

例5-放置皇后，课堂练习

```
//第x行，y列放置皇后，mark[行][列]表示一张棋盘
void put_down_the_queen(int x, int y,
                        int mark[][MAXN], int n){
    static const int dx[] = {-1, 1, 0, 0, -1, -1, 1, 1};  //方向数组
    static const int dy[] = {0, 0, -1, 1, -1, 1, -1, 1};
    mark[x][y] = 1;  // (x, y)放置皇后 进行标记
    int i, j;
    for (i = 1; i < n; i++){          //8个方向，每个方向向外延伸1至N-1
        for (j = 0; j < 8; j++){
            int new_x = [ 1 ]
            int new_y = [ 2 ]
            if (new_x >= 0 && new_x < n &&
                new_y >= 0 && new_y < n){
                [ 3 ]
            }
        }
    }
}
```

3分钟时间填写代码，
有问题随时提出！

例5-放置皇后，实现

```
//第x行，y列放置皇后，mark[行][列]表示一张棋盘
void put_down_the_queen(int x, int y,
                        int mark[][MAXN], int n){
    static const int dx[] = {-1, 1, 0, 0, -1, -1, 1, 1};  //方向数组
    static const int dy[] = {0, 0, -1, 1, -1, 1, -1, 1};
    mark[x][y] = 1;  // (x, y)放置皇后 进行标记
    int i, j;
    for (i = 1; i < n; i++){          //8个方向，每个方向向外延伸1至N-1
        for (j = 0; j < 8; j++){
            int new_x =  x + i * dx[j];

            int new_y =  y + i * dy[j];

            if (new_x >= 0 && new_x < n &&
                new_y >= 0 && new_y < n){

                mark[new_x][new_y] = 1;

            }
        }
    }
}
```

例5-回溯算法

N皇后问题，对于N*N的棋盘，每行都要放置1个且只能放置1个皇后。

利用递归对棋盘的每一行放置皇后，放置时，按列顺序寻找可以放置皇后的列，若可以放置皇后，将皇后放置该位置，并更新mark标记数组，递归进行下一行的皇后放置；当该次递归结束后，恢复mark数组，并尝试下一个可能放皇后的列。

当递归可以完成N行的N个皇后放置，则将该结果保存并返回。

例5-回溯算法-4皇后举例

初始化:

	0列	1列	2列	3列
0行:	0	0	0	0
1行:	0	0	0	0
2行:	0	0	0	0
3行:	0	0	0	0

递归第0行
尝试第0列

Q	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1

递归第1行
尝试第2列

Q	1	1	1
1	1	Q	1
1	1	1	1
1	0	1	1

递归第2行
哪列都没法放!

Q	1	1	1
1	1	Q	1
1	1	1	1
1	0	1	1

回溯第1行
尝试第3列

Q	1	1	1
1	1	1	Q
1	0	1	1
1	1	0	1

递归第2行
尝试第1列

Q	1	1	1
1	1	1	Q
1	Q	1	1
1	1	1	1

递归第3行
哪列都没法放!

Q	1	1	1
1	1	1	Q
1	Q	1	1
1	1	1	1

一直回溯到第0行
尝试第1列

1	Q	1	1
1	1	1	0
0	1	0	1
0	1	0	0

递归第1行
尝试第3列

1	Q	1	1
1	1	1	Q
0	1	1	1
0	1	0	1

递归第2行
尝试第0列

1	Q	1	1
1	1	1	Q
Q	1	1	1
1	1	0	1

递归第3行
尝试第2列

1	Q	1	1
1	1	1	Q
Q	1	1	1
1	1	Q	1

递归第4行
当行数为N时
即找到结果!

1	Q	1	1
1	1	1	Q
Q	1	1	1
1	1	Q	1

例5-回溯算法，课堂练习

```
//将数组mark2的内容拷贝至数组mark1
void copy_array(int mark1[][MAXN],int mark2[][MAXN], int n){
    int i, j;
    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            mark1[i][j] = mark2[i][j];
        }
    }
}
//k 代表完成了几个皇后的放置(正在放置第k行皇后)
void backtracking(int k, int n, int mark[][MAXN], int *result){
    if ( 1 ){
        (*result)++;
        2
    }
    int i;
    for (i = 0; i < n; i++){          //按顺序尝试第0至第n-1列
        if ( 3 ){
            int tmp_mark[MAXN][MAXN];
            copy_array(tmp_mark, mark, n);
            put_down_the_queen(k, i, mark, n);  //放置皇后
            backtracking( 4 , n, mark, result); //递归下一行皇后放置
            5
        }
    }
}
int totalNQueens(int n) {
    int mark[MAXN][MAXN] = {0};
    int result = 0;
    backtracking(0, n, mark, &result);
    return result;
}
```

3分钟时间填写代码，
有问题随时提出！

例5-回溯算法，实现

```
//将数组mark2的内容拷贝至数组mark1
void copy_array(int mark1[][MAXN],int mark2[][MAXN], int n){
    int i, j;
    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            mark1[i][j] = mark2[i][j];
        }
    }
}
//k 代表完成了几个皇后的放置(正在放置第k行皇后)
void backtracking(int k, int n, int mark[][MAXN], int *result){
    if ( k == n ){          //当k == n时，代表完成了第0至第n-1行
        (*result)++;
        return;
    }
    int i;
    for (i = 0; i < n; i++){          //按顺序尝试第0至第n-1列
        if ( mark[k][i] == 0 ){
            int tmp_mark[MAXN][MAXN];          //记录回溯前的mark镜像
            copy_array(tmp_mark, mark, n);
            put_down_the_queen(k, i, mark, n);          //放置皇后
            backtracking( k + 1 , n, mark, result);          //递归下一行皇后放置
            copy_array(mark, tmp_mark, n);
        }
    }                                          //将mark重新赋值为回溯前状态
}
int totalNQueens(int n) {
    int mark[MAXN][MAXN] = {0};
    int result = 0;
    backtracking(0, n, mark, &result);
    return result;
}
```

例5-测试与leetcode提交结果

```
int main(){
    printf("%d\n", totalNQueens(8));
    return 0;
}
```

```
92
请按任意键继续. . .
```

N-Queens II

Submission Detail

9 / 9 test cases passed. Status: Accepted

Runtime: **12 ms** Submitted: **0 minutes ago**

关于程序的**调试**，见”例7-LeetCode 52.N-Queens II-调试”

结束

非常感谢大家！

林沐

联系我们

小象学院：互联网新技术在线教育领航者

– 微信公众号：**小象学院**