项目经理带你-零基础学习 C/C++
【从入门到精通】

项目七 人工智能之地形导航系统

第 1 节 项目需求

项目需求

2018 年 11 月 26 日，“洞察”号火星探测器在火星上成功着陆，执行人类首次探究火星“内心深处”的任务。火星上地形比较复杂，高低起伏，有山峰和低谷，稍有不慎，就会引起“翻车”。因此我们必须要识别地形上的各个最高点（峰点）和最低点（谷点），以便为探测器提供导航数据。

峰点就是一个其周围所有点的海拔都低于它的点。

203	109	181
289	**300**	264
190	250	188

现在我们要做的就是分析来自地图上的海拔数据，以确定地形中峰点的数目和位置。

第 2 节 项目精讲-由线到面：二维数组

数组回顾

数组，就是多个同类型的元素的有序“组合”。如下的一组女兵，注：同类型是指都是女兵，不能混入男兵 -:)

二维数组，就是指含有多个数组的数组！

如果把一维数组理解为一行数据，那么，二维数组可形象地表示为行列结构。

二维数组的定义

和数组一样，需要先定义，再使用。
int a[25] ;　　//一行女兵
实例：
int　　a[5][25];　　//五行女兵
//定义了一个二维数组，
//数组名是“a”，
//包含 5 行 25 列，共 125 元素，
//每个元素是 int 类型的变量

二维数组的初始化

int a[3][4];　//二维数组元素的值可能是随机的（全局变量会初始化为 0，局部变量值随机）

方式一　初始化时指定每行的值

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

注：最外围括号内部的每个括号相当于初始化一行，括号中可以省略某些元素的初始化

方式二 初始化时从头开始，依次序进行

int a[3][4]={ 1，2，3，4，5，6，7，8，9，10，11，12};
int a[3][4]={ 0}; //所有元素都初始化为 0
注：后面的多个元素可以不指定，不指定全部初始化为 0

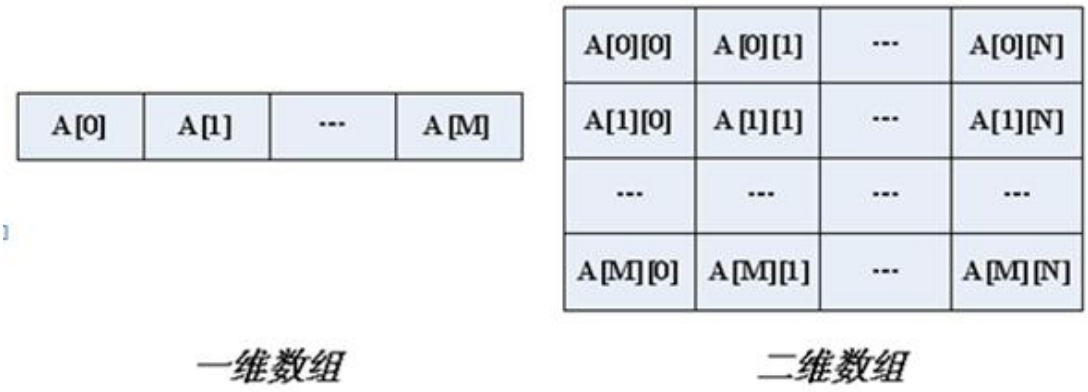
```
#include <stdlib.h>
#include <stdio.h>

int main(void){
    //int ages[5][25];//定义一个二维数组
    int i=0, j=0;

    //初始化
    //第一种方式  初始化时指定每行的值
    int a[3][4]={
        {1},//省略掉得列会默认置零
        {5,6,7},
        {9,10,11,12}
    };

    //第二种方式 初始化时从头开始，依次序进行
    int a1[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
    int a2[3][4]={1};//只初始化第一个，其他得默认置零

    for(i=0; i<3; i++){
        for(j=0; j<4; j++){
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
    system("pause");

    return 0;
}
```

二维数组的访问

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

a[0][0] 1	a[0][1] 2	a[0][2] 3	a[0][3] 4
a[1][0] 5	a[1][1] 6	a[1][2] 7	a[1][3] 8
a[2][0] 9	a[2][1] 10	a[2][2] 11	a[2][3] 12

如下图表示，左侧表示的是一个大小为 M+1 的一维数组，右侧表示的是一个大小为(M+1)*(N+1)的二维数组。

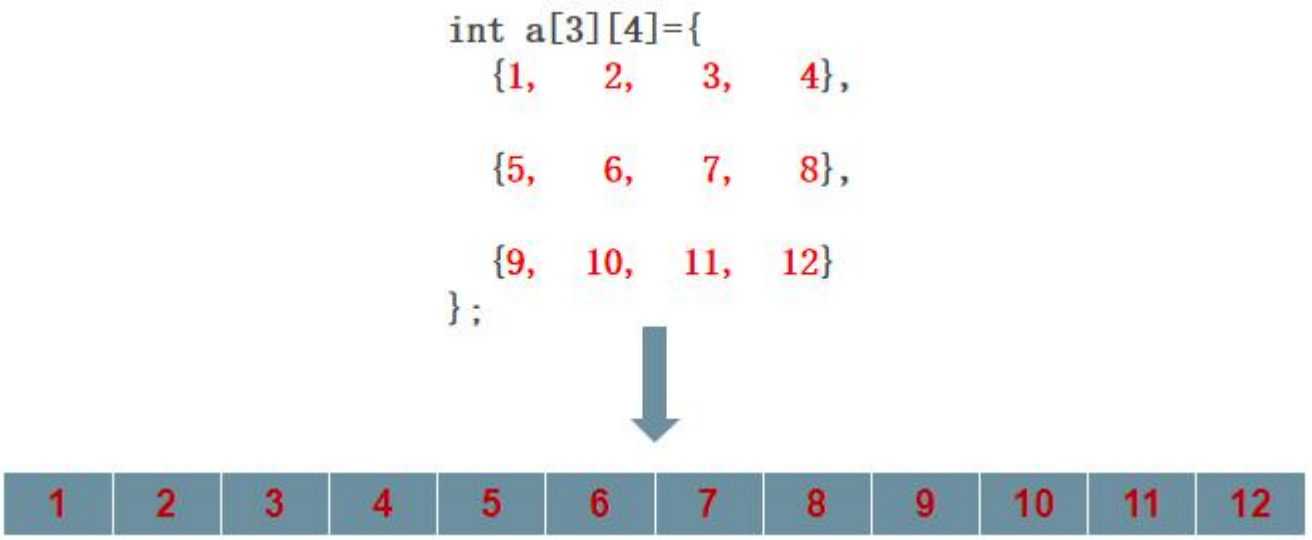



```
int i=0, j=0;
int a[3][4];

//给数组成员赋值

for(int i=0;i<12; i++){
      a[i/4][i%4] = i+1;
}
//或
for(int i=0;i<3; i++){
    for(int j=0;j<4; j++){
        a[i][j] = 4*i+j+1;
     }
}
```

```
//输出
for(int i=0;i<3; i++){
   for(int j=0;j<4; j++){
      printf("%d  ",a[i][j]);
    }
printf("\n");
}
```

第 3 节 项目精讲-二维数组的存储方式

一维数组是按**顺序存储**的，二维数组呢？ 同样也是！



设置断点调试，可以看到二位数组中的所有元素在内存中的存储方式

```
0x012FF770   00 00 00 00   ....
0x012FF774   01 00 00 00   ....
0x012FF778   02 00 00 00   ....
0x012FF77C   03 00 00 00   ....
0x012FF780   04 00 00 00   ....
0x012FF784   05 00 00 00   ....
0x012FF788   06 00 00 00   ....
0x012FF78C   07 00 00 00   ....
0x012FF790   08 00 00 00   ....
0x012FF794   09 00 00 00   ....
0x012FF798   0a 00 00 00   ....
0x012FF79C   0b 00 00 00   ....
```

第 4 节 项目精讲-更高维度：多维数组

上面讨论的二维数组的相关内容都适用于三维数组或更多维的数组。声明一个三维数组：
int girl[3][8][5];
可以把一维数组想象成一排女兵，把二维数组想象成一个女兵方阵，把三维数组想象成多个女兵方阵。这样，当你要找其中的一个女兵时，你只要知道她在哪个方阵（从 0、1、2 中选择），在哪一行（从 0-7）中选择，在哪一列（从 0-4 中选择）

第 5 节 二维数组作为函数的参数

切记！ 数组作为函数的参数传递，不是单纯的值传递，传递的是数组本身。
二维数组作为函数的参数:

```
#include <stdio.h>
#include <stdlib.h>

//版本 1  指明参数
void print1(int a[3][4]){
    for(int i=0;i<3; i++){
        for(int j=0;j<4; j++){
            printf("%d  ",a[i][j]);
        }
        printf("\n");
    }
}

//版本 1  省略一个高维参数
void print2(int a[][4], int lines){
    for(int i=0;i<lines; i++){
        for(int j=0;j<4; j++){
            printf("%d  ",a[i][j]);
        }
        printf("\n");
    }
}

int main(void){
    //int arr[3][4]={{},{3,4}};
    int a[3][4]={0};
    int i=0;
    int j=0;

    //给数组成员赋值
    for(int i=0;i<3; i++){
        for(int j=0;j<4; j++){
            a[i][j] = 4*i+j;
        }
    }

    print1(a);
   print2(a);

}
```

第 6 节 项目精讲-常见错误总结

数组定义时编译器不能确定数组的大小

int a[3][] ;
int a[][4] ;
int a[3][]={{1，2}，{3，4}，{5，6}}
一条原则：仅定义时（无初始化）不能省略，有初始化可以省略高维

严禁数组越界

在使用数组时，要防止数组下标超出边界。也就是说，必须确保下标是有效的值。
int a[10] ;　　a[10]=100;　//错误，访问越界，a[10] 的成员是 a[0] - a[9]
int a[3][4];　　a[3][0]=666; //错误, a[3][4]　包含了 3 行 4 列的数组，行下标也是从 0 开始，有效范围 0 - 2

数组（无论几维）传参并不是整个数组的复制

```
#include <stdio.h>
#include <stdlib.h>

//int arr[3][4];

void fun(int a[3][4]){
    for(int i=0;i<3;i++){
        for(int j=0;j<4;j++){
            a[i][j]=0;
        }
    }
}

int main(void){
    //int arr[3][4]={{},{3,4}};
    int a[3][4]={0};
    int i=0;
    int j=0;

    //给数组成员赋值
    for(int i=0;i<3; i++){
        for(int j=0;j<4; j++){
            a[i][j] = 4*i+j;
        }
    }
```

```
    fun(a);

    //输出
    for(int i=0;i<3; i++){
        for(int j=0;j<4; j++){
            printf("%d  ",a[i][j]);
        }
        printf("\n");
    }
```

第 7 节 项目实现

假设下面的数据代表一个 6 x 7 的网格，加了下划线的网格即为峰点。

5039	5127	5238	5259	5248	5310	5299
5150	5392	5410	5401	5320	5820	5321
5290	5560	5490	5421	5530	5831	5210
5110	5429	5430	5411	5459	5630	5319
4920	5129	4921	5821	4722	4921	5129
5023	5129	4822	4872	4794	4862	4245

为了描述峰点的位置，我们需要使用一个位置方案：使用二维数组描述
假定左上角是[0][0],那么向下移动，则行号加 1；向右移动，则列号加 1,
那么这些峰点的位置就可以描述为:[2][1] [2][5] [4][3]。

位置确定后，与周围 4 个邻节点比较即可确定峰点！（**注：**网格边界点缺乏 4 个相邻点不计算峰点）

地形数据保存于文件中。

算法设计

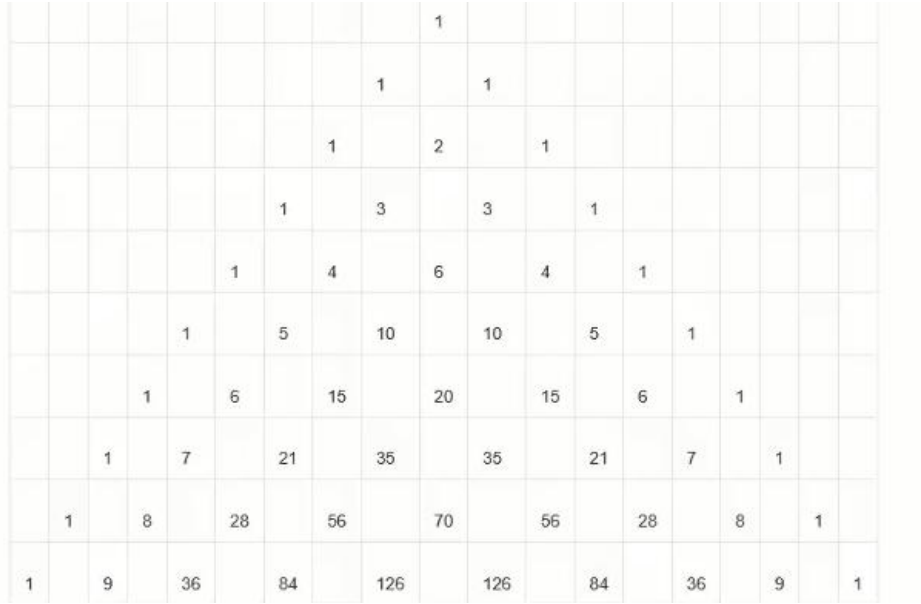
1) 将地形数据从文件读入二维数组;
2) 逐行遍历二维数组的每个元素，确定是否峰值并打印结果。

```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

#define N  64

bool isPeak(const double grid[][N], int r, int c);

int main(){

    int nrows, ncols;
    double map[N][N];
    string filename;
    ifstream file;

    cout <<"请输入文件名.\n";
    cin  >> filename;
    file.open(filename.c_str());
    if(file.fail()){
        cerr<<"打开输入文件出错.\n";
        exit(1);
    }

    file>>nrows>>ncols;

    if(nrows > N || ncols > N){
        cerr<<"网格太大，调整程序.\n";
        exit(1);
    }

    //从数据文件读数据到数组
    for(int i=0; i<nrows; ++i){
        for(int j=0; j<ncols; ++j){
            file>>map[i][j];
        }
    }

    //判断并打印峰值位置
    for(int i=1;i<nrows-1; ++i){
        for(int j=1;j<ncols-1; ++j){
            if(isPeak(map, i, j)){
                cout<<"峰值出现在行: "<<i<<" 列:"<<j<<endl;
            }
        }
    }
```

```
    //关闭文件
    file.close();

    //结束程序
    return 0;
}

bool isPeak(const double grid[][N], int i, int j){
    if((grid[i-1][j]<grid[i][j]) &&
       (grid[i+1][j]<grid[i][j]) &&
       (grid[i][j-1]<grid[i][j]) &&
       (grid[i][j+1]<grid[i][j]))
       return true;
    else
       return false;
}
```

第 8 节 编程思维修炼

编写程序打印出杨辉三角形 。



杨辉三角形规律：

1

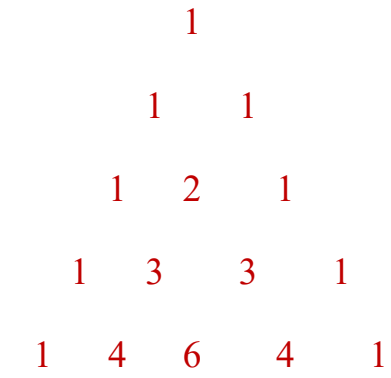
1 1

1 2 1

1 3 3 1

1 4 6 4 1

三角形图案输出



实现思路：
我们定义一个二维数组，所有元素先初始化为 0；
给数组的第 1 列和对角线元素赋值为 1；
其余元素 a[i][j]=a[i-1][j-1]+a[i-1][j]；

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

编程实现：

```
#include <iostream>
#include <stdlib.h>
#include <iomanip>

using namespace std;

#define N 10

int main(void){

    int a[N][N] = {0}; //二维数组所有元素清零

    for(int i=0; i<N; i++){
        for(int j=0; j<=i; j++){//j<=i,第 i 行，仅有 i 个数字
            if(j==0 || i==j){//第一列和对角线置为 1
                a[i][j] = 1;
            }else {
                a[i][j]=a[i-1][j-1]+a[i-1][j];
            }
        }
    }

    for(int i=0; i<N; i++){
        cout<<setw((N-i)*4)<<a[i][0];

        for(int j=1; j<=i; j++){
            cout<<setw(8)<<a[i][j];
        }
        cout<<endl;
    }

    system("pause");
    return 0;
}
```

第 9 节 职场修炼：公司派系斗争中怎样站队？

“三个女人一台戏”

有人的地方就有江湖！有江湖的地方就有斗争！

方圆之道 - “做人要方，做事要圆”

1. 不要跟直接领导起冲突

2. 你站的一方，有利益并且对你好

3. 你站的一边，背后有更大的靠山

4. 你站的一边，没有人和你有利益冲突

5. 如果缺乏以上的判断力，请保持佛系-中立

第 10 节 逼格提升：防御式编程

防御式编程（Defensive Programming）是提高软件质量的辅助手段

怎么理解呢？防御式编程思想的理解可以参考防御式驾驶：

广州司机闯红灯致13伤 目击者：她脚踩松糕鞋 有人被车带出好几米

新京报 05-21 11:51 关注

新京报讯（记者 张彤 周世玲）今日（5月21日），广州市公安局交警部门通报称，当日8时50分许，在天河区林和中路林乐路口，一车辆闯红灯并撞上行人及两辆汽车，造成13人受伤，目前受伤人员已送往医院救治，肇事司机丁某已经被警方控制。

在防御式驾驶中要建立这样一种思维，那就是你水远也不能确定另一位司机将要做什么。这样才能确保在其他人做出危险动作时你也不会受到伤害。你要承担起保护自己的责任，哪怕是其他司机犯的错误。

核心思想：子程序应该不因传入错误数据而被破坏，哪怕是由其他子程序产生的错误数据。这种思想是将可能出现的错误造成的影响控制在有限的范围内。

具体措施：

1. 对输入进行体检

（1）检查输入源（如：文件，网络、控制台等）数据的合法性

（2）检查每一个函数输入参数的合法性

处理方式：一旦非法输入被发现，那么应该根据情况进行处理。防御式编程的最佳的形式是在一开始就不引入错误。

2. 对非预期错误使用断言

（1）空指针。

（2）输入或者输出参数的值不在预期范围内。

（3）数组的越界。

处理方式：如果断言的条件返回错误，则终止程序执行。

原型定义：

```
#include <assert.h>
void assert( int expression );
```

assert 的作用是先计算表达式 expression ，如果其值为假（即为 0），那么它先向 stderr 打印一条出错信息，

然后通过调用 abort 来终止程序运行。

```
bool isPeak(int grid[N][N], int r, int c){
    assert( 0<r && r<N-1);
    assert( 0<c && c<N-1);

    if((grid[r][c] > grid[r-1][c])&&
        (grid[r][c] > grid[r+1][c])&&
        (grid[r][c] > grid[r][c-1])&&
        (grid[r][c] > grid[r][c+1])){
            return true;
    }else{
        return false;
    }
}
```

第 11 节 项目练习

按照下面的要求修改寻找峰点的程序：

1. 打印出网格中峰点的数目。
2. 打印出谷点的位置。假定谷点是一个比邻接点海拔都要低的点。编写一个名为 isValley 的函数供你的程序调用 。
3. 找出并打印海拔数据中最高点和最低点的位置及其海拔。编写一个名为 extremes 的函数供你的程序调用。
4. 修改函数 isPeak(), 使用 8 个邻接点来判断峰点，而不再是只使用 4 个邻近点判断。