

同济大学计算机系

OOP 矩阵大作业实验报告



学 号 2152118

姓 名 史君宝

专 业 计算机科学与技术（计科1班）

完成时间 2023.10.20

一、设计思路与功能描述

1. 得分点

- (1) 基本的矩阵运算界面，以及它们的实现
- (2) 实现 OTSU 算法，求最优二值化的灰度值阈值
- (3) 实现其他处理方法，腐蚀运算，膨胀运算，开运算，闭运算

2. 设计思路

- (1) 美化简约的处理界面。

这个题目给的就是一个矩阵运算的菜单页面，看到一些学长的报告，也是将其设计成一个类似于计算器的思路。所以在这一题中，比较重要的就是设计具体的处理界面，尽量将其美化简约。

- (2) 一定的错误处理。

由于在过程中，需要输入的数据比较多，所以也需要一定的错误处理，这样能够减少因为某一次错误输入，而导致的重输。

3. 功能描述

- (1) 函数声明：

```
//矩阵结构体声明
struct matrix {
    int row;
    int col;
    int a[256 * 256];
};

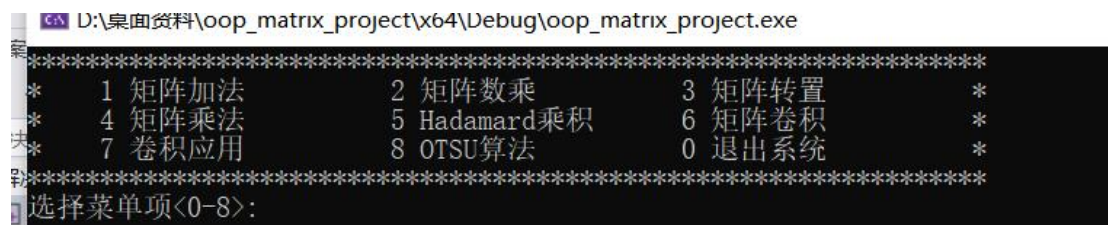
//函数的声明
void define_matrix(); //确定矩阵元素
void display_matrix(); //显示矩阵元素
void menu(); //主菜单
void matirplus(); //矩阵加法
void nummulti(); //矩阵数乘
void matritrans(); //矩阵转置
void matrimulti(); //矩阵乘法
void hadamulti(); //矩阵hadamard乘积
void conv(); //矩阵卷积
void conv_demo(matrix& A, int kernel[], matrix& sum); //封装矩阵卷积，帮助实现卷积应用
void demo(); //卷积应用
int OTSU_t(); //OTSU算法求最优阈值
void OTSU(); //OTSU算法
Mat Erode(Mat src, Mat nuclear); //腐蚀算法
Mat Dilate(Mat src, Mat nuclear); //膨胀算法
void OTSU_etr(); //OTSU其他算法应用
void OTSU_order(); //OTSU界面选择算法
void wait_for_enter(); //等待输入
```

(2) menu 函数:

功能:初始化屏幕, 让使用者选择对应的矩阵运算。

```
void menu() {  
    for (int i = 0; i < 66; ++i)  
        cout << "*";  
    cout << endl;  
    cout << "*   1 矩阵加法           2 矩阵数乘           3 矩阵转置           *" << endl;  
    cout << "*   4 矩阵乘法           5 Hadamard乘积       6 矩阵卷积           *" << endl;  
    cout << "*   7 卷积应用           8 OTSU算法           0 退出系统       *" << endl;  
    for (int i = 0; i < 66; ++i)  
        cout << "*";  
    cout << endl;  
    cout << "选择菜单项<0-8>:";
```

实际效果:

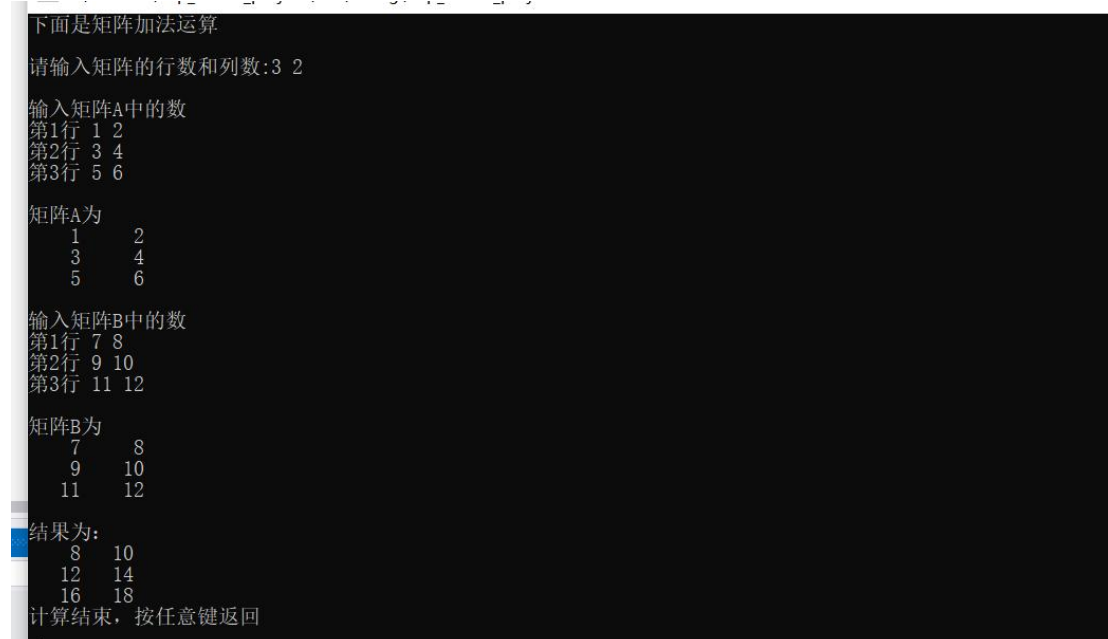


```
D:\桌面资料\oop_matrix_project\x64\Debug\oop_matrix_project.exe  
*****  
*   1 矩阵加法           2 矩阵数乘           3 矩阵转置           *  
*   4 矩阵乘法           5 Hadamard乘积       6 矩阵卷积           *  
*   7 卷积应用           8 OTSU算法           0 退出系统       *  
*****  
选择菜单项<0-8>:
```

(3) 矩阵加法

功能:在输入 1 之后, 会进入矩阵加法, 进行运算。

效果:



```
D:\桌面资料\oop_matrix_project\x64\Debug\oop_matrix_project.exe  
下面是矩阵加法运算  
请输入矩阵的行数和列数:3 2  
输入矩阵A中的数  
第1行 1 2  
第2行 3 4  
第3行 5 6  
矩阵A为  
1 2  
3 4  
5 6  
输入矩阵B中的数  
第1行 7 8  
第2行 9 10  
第3行 11 12  
矩阵B为  
7 8  
9 10  
11 12  
结果为:  
8 10  
12 14  
16 18  
计算结束, 按任意键返回
```

(4) 矩阵数乘:

功能:在输入 2 之后, 会进入矩阵数乘, 进行运算。

效果:

```
D:\桌面资料\oop_matrix_project\x64\Debug\oop_matrix_project.exe
下面是矩阵数乘运算
请输入矩阵的行数和列数:2 3
输入矩阵A中的数
第1行 1 5 8
第2行 2 8 9
矩阵A为
    1    5    8
    2    8    9
请输入矩阵要乘的数:5
结果为:
    5   25   40
   10   40   45
计算结束, 按任意键返回
```

(5) 矩阵转置

功能:在输入 3 之后, 会进入矩阵转置, 进行运算。

效果:

```
D:\桌面资料\oop_matrix_project\x64\Debug\oop_matrix_project.exe
下面是矩阵转置运算
请输入矩阵的行数和列数:2 3
输入矩阵A中的数
第1行 1 2 3
第2行 4 5 6
5
矩阵A为
    1    2    3
    4    5    6
结果为:
    1    4
    2    5
    3    6
计算结束, 按任意键返回
```

(6) 矩阵乘法

功能:在输入 4 之后, 会进入矩阵乘法, 进行运算。

效果:

```
C:\> D:\桌面资料\oop_matrix_project\x64\Debug\oop_matrix_proj

下面是矩阵乘法运算

请输入矩阵A的行数和列数:2 3
请输入矩阵B的行数和列数:2 3
两矩阵行列不匹配, 不能相乘
计算结束, 按任意键返回
```

```
C:\> D:\桌面资料\oop_matrix_project\x64\Debug\oop_matrix_project.exe

下面是矩阵乘法运算

请输入矩阵A的行数和列数:2 3
请输入矩阵B的行数和列数:3 2

输入矩阵A中的数
第1行 1 2 3
第2行 4 5 6

矩阵A为
    1    2    3
    4    5    6

输入矩阵B中的数
第1行 10 5
第2行 8 2
第3行 15 3

矩阵B为
    10    5
     8    2
    15    3

结果为:
    71    18
   170    48
计算结束, 按任意键返回
```

(7) Hadamard 乘积

功能:在输入 5 之后, 会进入矩阵 Hadamard 乘积, 进行运算。

效果:

```
D:\桌面资料\oop_matrix_project\x64\Debug\oop_matrix_project.exe
下面是Hadamard乘积运算

请输入矩阵的行数和列数:2 2

输入矩阵A中的数
第1行 1 2
第2行 3 4

矩阵A为
    1    2
    3    4

输入矩阵B中的数
第1行 1 3
第2行 2 4

矩阵B为
    1    3
    2    4

结果为:
    1    6
    6   16
计算结束, 按任意键返回
```

(8) 矩阵卷积

功能:在输入 6 之后, 会进入矩阵卷积, 进行运算。

效果:

```
下面是矩阵卷积运算

请输入矩阵的行数和列数:3 3

输入矩阵A中的数
第1行 0 25 75
第2行 0 75 80
第3行 0 75 80

矩阵A为
    0    25    75
    0    75    80
    0    75    80

输入矩阵B中的数
第1行 -1 0 1
第2行 -1 0 1
第3行 -1 0 1

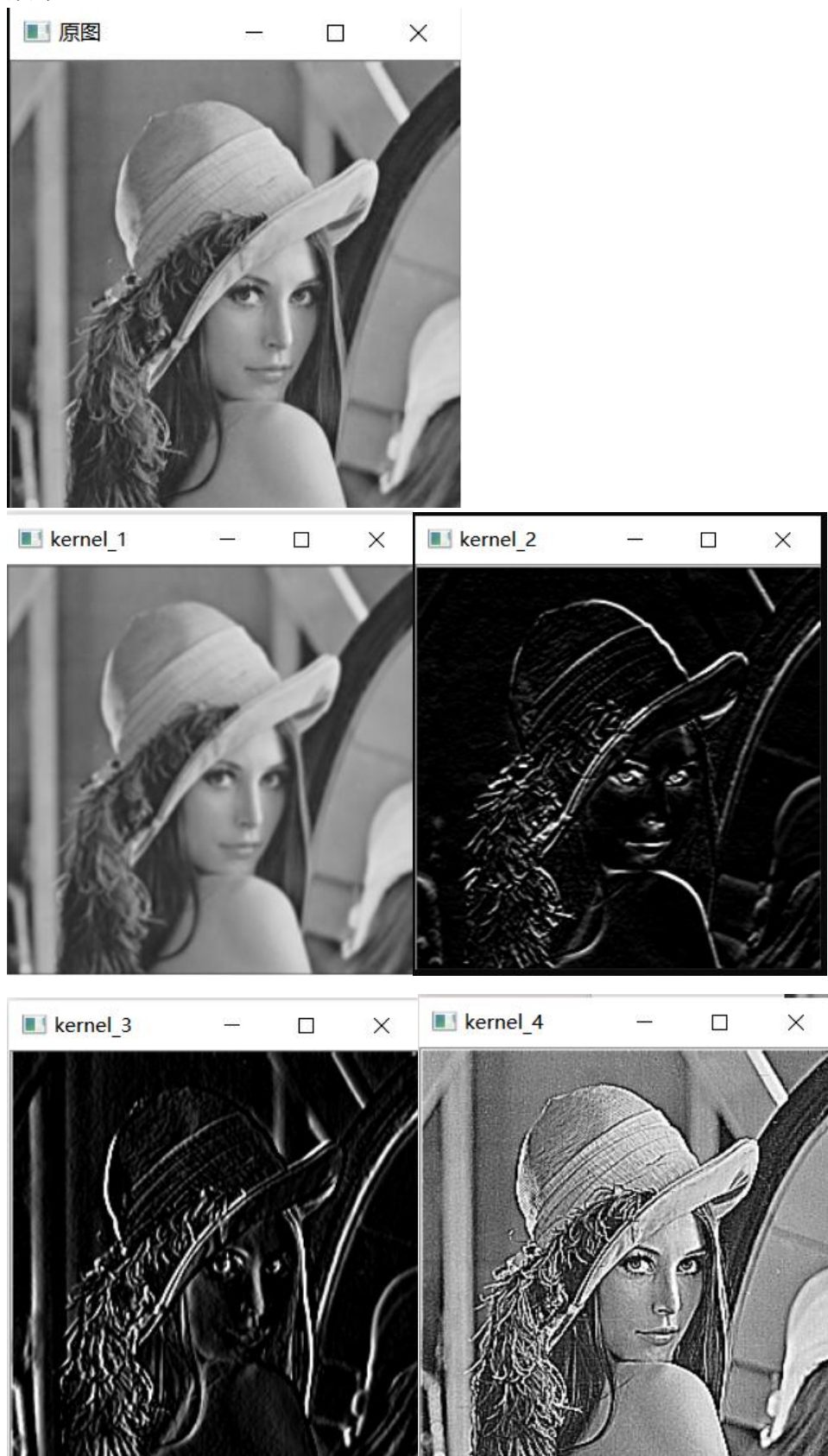
矩阵B为
   -1    0    1
   -1    0    1
   -1    0    1

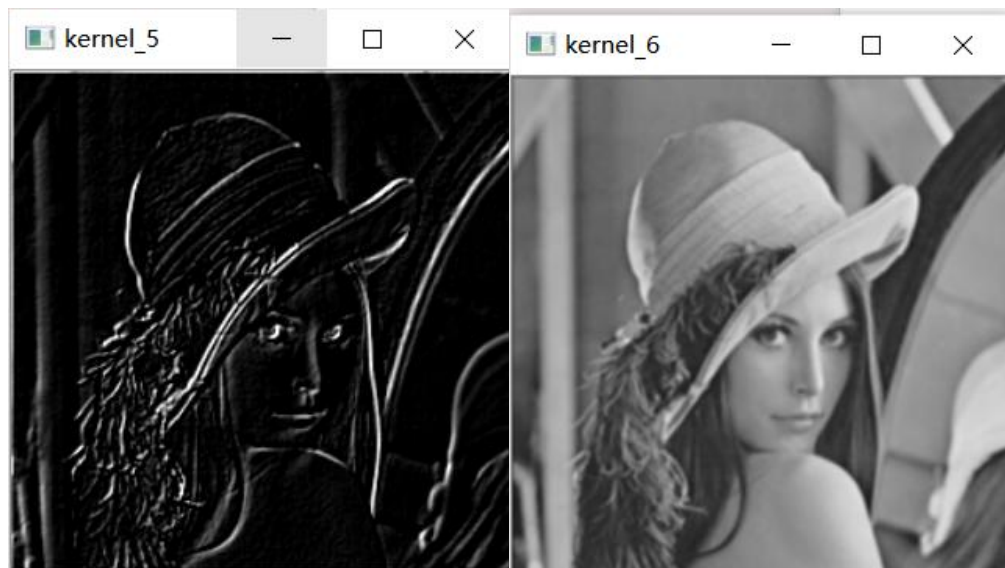
结果为:
   100   155  -100
   175   235  -175
   150   160  -150
计算结束, 按任意键返回
```


(9) 卷积应用

功能:在输入 7 之后, 会进入卷积应用, 进行运算。

效果:





(10) OTSU 运算

功能:在输入 8 之后, 会进入 OTSU 算法, 进行运算。

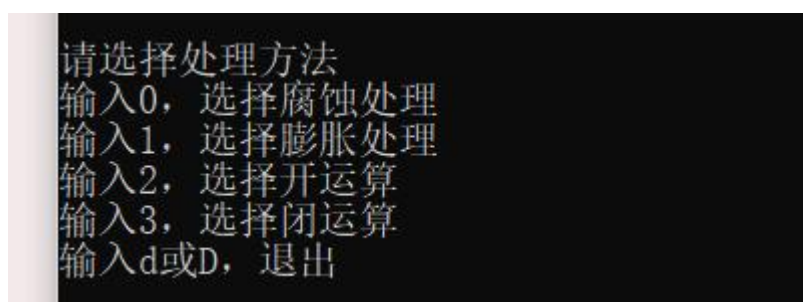
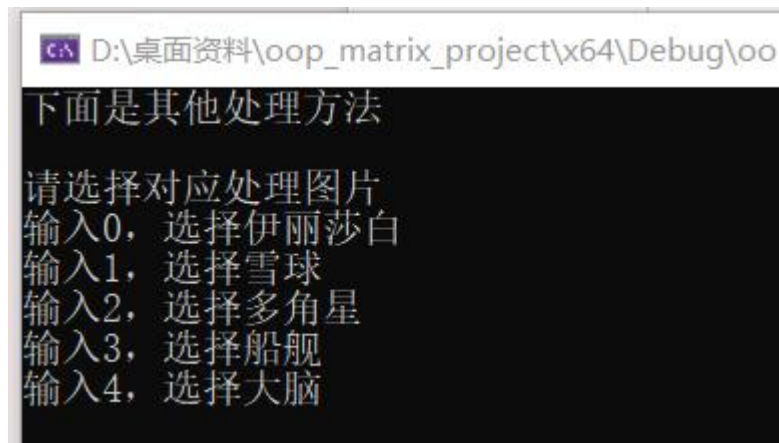
效果:



输入 0



输入 1



原图:



二值化原图:



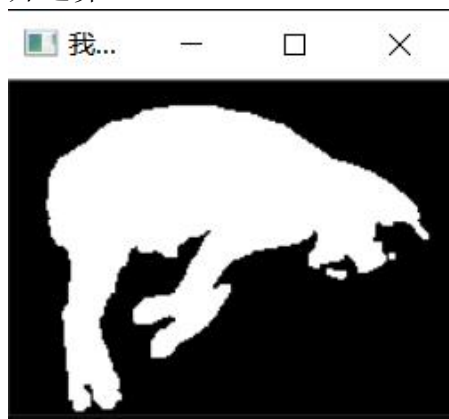
腐蚀算法:



膨胀算法:



开运算:



闭运算



二、问题与解决方法

1. 遇到的问题

- (1) 函数爆栈问题，stack overflow。
- (2) 图像处理的问题，imread 读取灰度值。
- (3) 其他的图像处理算法。

2. 解决的方法

(1) 函数爆栈问题，stack overflow。

由于在过程中，尤其是矩阵卷积，以及后面的图像处理方面，需要将图像上 256*256 个像素点都要储存起来。所以在过程中不能在调用函数中申请太多的内存空间，导致函数爆栈。

具体解决：

可以将栈内存转移到堆内存，比如利用动态内存申请，比如应用 new int 数组，最后需要记住 delete 删除堆内存，防止内存泄漏。

在本题中，自己仍然正常的申请，只不过，在过程中，将函数调用的传参过程改用引用来进行传参，这样能够实现栈内存的复用，这样能够避免函数爆栈的情况。

(2) 图像处理的问题，imread 读取灰度值。

在图像处理过程中，采用原来 imread 函数读取灰度值，然后将其转存在数组中，进一步的利用矩阵卷积进行处理。

处理过程：

```
system("cls"); // 清屏函数
cout << "下面是OTSU算法" << endl << endl;

//对vs+opencv正确配置后方可使用，此处只给出一段读取并显示图像的参考代码，其余功能流程自行设计和查阅文献
Mat image = imread(".\\res\\demolena.jpg", CV_LOAD_IMAGE_GRAYSCALE); // 图像的灰度值存放在格式为Mat的变量image中
imshow("原图", image);

struct matrix A;
A.row = image.rows;
A.col = image.cols;

for (int i = 0; i < A.row; i++)
    for (int j = 0; j < A.col; j++)
        A.a[A.col * i + j] = (int)image.at<uchar>(i, j);
```

但是在之后进行处理获得的图像只是将左上角大概 1/4 的区域显示出来，迟迟找不到问题。经过查找资料之后，是在 imread 中还是读取了 RGB，不仅仅使用了灰度值，就导致了没有覆盖完整个图片。查找资料之后，在 imread 中加入一个参数 CV_LOAD_IMAGE_GRAYSCALE

(3) 其他的图像处理算法。

在问题的最后需要给一些其他的图像处理算法。我们已经使用的是 OTSU 算法，这是一种二值化的方法，需要通过一个公式计算出最优的分割阈值。

之后自己在网上找了一些其他的算法，比如腐蚀算法，膨胀算法还有对应的开运算，闭运算等等。在这里出现了一些问题，在网上找了一段时间，比修改它们的算法。

三、心得体会

这次大作业总体来说并不难，使用的是 OpenCV 的库函数，在刚开始的配置过程花费了一点时间，但是总的来说还是比较顺利的。

同时老师的准备也很充分。这次实验本就实用性，类似于一个简单的矩阵计算器。老师也提供了很多的资料，比较喜欢的是一个学长写的“论线性代数”，看着很有乐子，赞叹于学长的有才。

希望各位助教手下留情，嘻嘻。

四、源代码

```
#include <conio.h>
#include <iostream>
#include <opencv2/opencv.hpp>
using namespace cv;
using namespace std;

// 此框架若有不完美可以在作业中任意修改

//矩阵结构体声明
struct matrix {
    int row;
    int col;
    int a[256 * 256];
};

//函数的声明
void define_matrix();    //确定矩阵元素
void display_matrix();   //显示矩阵元素
void menu();             //主菜单
void matriplus();        //矩阵加法
```

```

void nummulti();           //矩阵数乘
void matritrans();         //矩阵转置
void matrimulti();         //矩阵乘法
void hadamulti();          //矩阵 hadamard 乘积
void conv();               //矩阵卷积
void conv_demo(matrix& A, int kernel[], matrix& sum); //封装矩阵卷积,
帮助实现卷积应用
void demo();               //卷积应用
int OTSU_t();              //OTSU 算法求最优阈值
void OTSU();               //OTSU 算法
Mat Erode(Mat src, Mat nuclear); //腐蚀算法
Mat Dilate(Mat src, Mat nuclear); //膨胀算法
void OTSU_etr();           //OTSU 其他算法应用
void OTSU_order();         //OTSU 界面选择算法
void wait_for_enter();     //等待输入

```

```

void define_matrix(matrix& A, char str) {
    int num = 0, data;
    cout << "\n 输入矩阵" << str << "中的数" << endl;
    for (int i = 0; i < A.row; ++i)
    {
        cout << "第" << i + 1 << "行 ";
        for (int j = 0; j < A.col; ++j)
        {
            cin >> data;
            A.a[num] = data;
            num++;
        }
    }
}

```

```

void display_matrix(matrix& A, char str) {
    int num = 0;
    cout << "\n 矩阵" << str << "为" << endl;
    for (int i = 0; i < A.row; ++i)
    {
        for (int j = 0; j < A.col; ++j)
        {
            cout << setw(5) << setfill(' ') << A.a[num] << " ";
            num++;
        }
        cout << endl;
    }
}

```



```

    }
}

void menu() {
    for (int i = 0; i < 66; ++i)
        cout << "*";
    cout << endl;
    cout << "*    1 矩阵加法          2 矩阵数乘          3 矩阵转置"
    *" << endl;
    cout << "*    4 矩阵乘法          5 Hadamard 乘积      6 矩阵卷积"
    *" << endl;
    cout << "*    7 卷积应用          8 OTSU 算法          0 退出系统"
    *" << endl;
    for (int i = 0; i < 66; ++i)
        cout << "*";
    cout << endl;
    cout << "选择菜单项<0-8>:";
}

```

```

void matriplus() {

    system("cls"); // 清屏函数
    cout << "下面是矩阵加法运算" << endl << endl;

    int rowA = 0, colA = 0;
    int rowB = 0, colB = 0;

    cout << "请输入矩阵的行数和列数:";
    cin >> rowA >> colA;
    while (rowA <= 0 || colA <= 0)
    {
        cout << "输错了，请重新输入" << endl;
        cin.clear();
        cin.ignore(65535);
        cout << "请输入矩阵的行数和列数:";
        cin >> rowA >> colA;
    }

    struct matrix A, B;
    A.row = rowA;
    A.col = colA;
    B.row = rowA;
    B.col = colA;
}

```

```

define_matrix(A, 'A');
display_matrix(A, 'A');
define_matrix(B, 'B');
display_matrix(B, 'B');

int num = 0;
char ch;

cout << "\n 结果为: " << endl;
for (int i = 0; i < A.row; ++i)
{
    for (int j = 0; j < A.col; ++j)
    {
        cout << setw(5) << setfill(' ') << A.a[num] + B.a[num];
        num++;
    }
    cout << endl;
}

cout << "计算结束, 按任意键返回" << endl;
ch = _getch();
return;
}

void nummulti() {

    system("cls"); // 清屏函数
    cout << "下面是矩阵数乘运算" << endl << endl;

    int rowA = 0, colA = 0, data_num;

    cout << "请输入矩阵的行数和列数:";
    cin >> rowA >> colA;
    while (rowA <= 0 || colA <= 0)
    {
        cout << "输错了, 请重新输入" << endl;
        cin.clear();
        cin.ignore(65535);
        cout << "请输入矩阵的行数和列数:";
        cin >> rowA >> colA;
    }

    struct matrix A;
    A.row = rowA;

```

```

A.col = colA;

define_matrix(A, 'A');
display_matrix(A, 'A');

cout << "\n 请输入矩阵要乘的数:";
cin >> data_num;

int num = 0;
char ch;

cout << "\n 结果为: " << endl;
for (int i = 0; i < A.row; ++i)
{
    for (int j = 0; j < A.col; ++j)
    {
        cout << setw(5) << setfill(' ') << A.a[num] * data_num;
        num++;
    }
    cout << endl;
}

cout << "计算结束, 按任意键返回" << endl;
ch = _getch();
return;
}

void matritrans() {

    system("cls"); // 清屏函数
    cout << "下面是矩阵转置运算" << endl << endl;

    int rowA = 0, colA = 0;

    cout << "请输入矩阵的行数和列数:";
    cin >> rowA >> colA;
    while (rowA <= 0 || colA <= 0)
    {
        cout << "输错了, 请重新输入" << endl;
        cin.clear();
        cin.ignore(65535);
        cout << "请输入矩阵的行数和列数:";
        cin >> rowA >> colA;
    }
}

```

```

    struct matrix A;
    A.row = rowA;
    A.col = colA;

    define_matrix(A, 'A');
    display_matrix(A, 'A');

    char ch;

    cout << "\n 结果为: " << endl;
    for (int i = 0; i < A.col; ++i)
    {
        for (int j = 0; j < A.row; ++j)
        {
            cout << setw(5) << setfill(' ') << A.a[j * colA + i];
        }
        cout << endl;
    }

    cout << "计算结束, 按任意键返回" << endl;
    ch = _getch();
    return;
}

void matrimulti() {

    system("cls"); // 清屏函数
    cout << "下面是矩阵乘法运算" << endl << endl;

    int rowA = 0, colA = 0, rowB = 0, colB = 0;

    cout << "\n 请输入矩阵 A 的行数和列数:";
    cin >> rowA >> colA;
    while (rowA <= 0 || colA <= 0)
    {
        cout << "输错了, 请重新输入" << endl;
        cin.clear();
        cin.ignore(65535);
        cout << "请输入矩阵的行数和列数:";
        cin >> rowA >> colA;
    }

    cout << "\n 请输入矩阵 B 的行数和列数:";

```

```

cin >> rowB >> colB;
while (rowB <= 0 || colB <= 0)
{
    cout << "输错了，请重新输入" << endl;
    cin.clear();
    cin.ignore(65535);
    cout << "请输入矩阵的行数和列数:";
    cin >> rowB >> colB;
}

if (colA != rowB) {
    cout << "\n 两矩阵行列不匹配，不能相乘" << endl;
}
else {
    struct matrix A, B, sum;

    A.row = rowA;
    A.col = colA;
    B.row = rowB;
    B.col = colB;
    sum.row = rowA;
    sum.col = colB;

    define_matrix(A, 'A');
    display_matrix(A, 'A');
    define_matrix(B, 'B');
    display_matrix(B, 'B');

    int num = 0;

    cout << "\n 结果为: " << endl;
    for (int i = 0; i < sum.row; ++i)
    {
        for (int j = 0; j < sum.col; ++j)
        {
            sum.a[num] = 0;
            for (int k = 0; k < A.col; ++k)
            {
                sum.a[num] += A.a[i * A.col + k] * B.a[k * B.col +
j];
            }
            cout << setw(5) << setfill(' ') << sum.a[num];
            num++;
        }
    }
}

```



```

        cout << endl;
    }
}

char ch;
cout << "计算结束，按任意键返回" << endl;
ch = _getch();
return;
}

void hadamulti() {

    system("cls"); // 清屏函数
    cout << "下面是 Hadamard 乘积运算" << endl << endl;

    int rowA = 0, colA = 0;

    cout << "\n 请输入矩阵的行数和列数:";
    cin >> rowA >> colA;
    while (rowA <= 0 || colA <= 0)
    {
        cout << "输错了，请重新输入" << endl;
        cin.clear();
        cin.ignore(65535);
        cout << "请输入矩阵的行数和列数:";
        cin >> rowA >> colA;
    }

    struct matrix A, B;

    A.row = rowA;
    A.col = colA;
    B.row = rowA;
    B.col = colA;

    define_matrix(A, 'A');
    display_matrix(A, 'A');
    define_matrix(B, 'B');
    display_matrix(B, 'B');

    int num = 0;
    cout << "\n 结果为: " << endl;
    for (int i = 0; i < A.row; ++i)
    {

```

```

        for (int j = 0; j < A.col; ++j)
        {
            cout << setw(5) << setfill(' ') << A.a[num] * B.a[num] << "
";
            num++;
        }
        cout << endl;
    }

    char ch;
    cout << "计算结束，按任意键返回" << endl;
    ch = _getch();
    return;
}

```

```

void conv() {

    system("cls"); // 清屏函数
    cout << "下面是矩阵卷积运算" << endl << endl;

    int rowA = 0, colA = 0;

    cout << "\n 请输入矩阵的行数和列数:";
    cin >> rowA >> colA;
    while (rowA <= 0 || colA <= 0)
    {
        cout << "输错了，请重新输入" << endl;
        cin.clear();
        cin.ignore(65535);
        cout << "请输入矩阵的行数和列数:";
        cin >> rowA >> colA;
    }

    struct matrix A, B, sum;
    A.row = rowA;
    A.col = colA;
    B.row = 3;
    B.col = 3;

    define_matrix(A, 'A');
    display_matrix(A, 'A');
    define_matrix(B, 'B');
    display_matrix(B, 'B');
}

```

```

int num = 0;
cout << "\n 结果为: " << endl;
for (int i = 0; i < A.row; ++i)
{
    for (int j = 0; j < A.col; ++j)
    {
        sum.a[num] = 0;
        for (int k = -1; k <= 1; ++k)
        {
            for (int l = -1; l <= 1; ++l)
            {
                if (i + k < 0 || i + k >= A.row || j + l < 0 || j +
1 >= A.col) {
                    continue;
                }
                else {
                    sum.a[num] += A.a[(i + k) * A.col + (j + l)] *
B.a[(1 + k) * B.col + (1 + l)];
                }
            }
        }
        cout << setw(5) << setfill(' ') << sum.a[num];
    }
    cout << endl;
}

char ch;
cout << "计算结束, 按任意键返回" << endl;
ch = _getch();
return;
}

```

```

void conv_demo(matrix& A, int kernel[], matrix& sum) {

```

```

    int num = 0, t = 0;
    for (int i = 0; i < 9; ++i)
        t += kernel[i];
    if (t == 0)
        t = 1;

    for (int i = 0; i < A.row; ++i)
    {
        for (int j = 0; j < A.col; ++j)
        {

```

```

        sum.a[num] = 0;
        for (int k = -1; k <= 1; ++k)
        {
            for (int l = -1; l <= 1; ++l)
            {
                if (i + k < 0 || i + k >= A.row || j + l < 0 || j +
l >= A.col) {
                    continue;
                }
                else {
                    sum.a[num] += A.a[(i + k) * A.col + (j + l)] *
kernel[(1 + k) * 3 + (1 + l)];
                }
            }
        }
        sum.a[num] /= t;
        if (sum.a[num] < 0)
            sum.a[num] = 0;
        else if (sum.a[num] > 255)
            sum.a[num] = 255;

        num++;
    }
}
return;
}

```

```

void demo()
{
    system("cls"); // 清屏函数
    cout << "下面是卷积应用" << endl << endl;

```

//对 vs+opencv 正确配置后方可使用, 此处只给出一段读取并显示图像的参考代码, 其余功能流程自行设计和查阅文献

```

    Mat image = imread(".\\res\\demolena.jpg", CV_LOAD_IMAGE_GRAYSCALE);
// 图像的灰度值存放在格式为 Mat 的变量 image 中
    imshow("原图", image);

```

```

    struct matrix A, sum;
    A.row = image.rows;
    A.col = image.cols;

```

```

    for (int i = 0; i < A.row; i++)

```

```

        for (int j = 0; j < A.col; j++)
            A.a[A.col * i + j] = (int)image.at<uchar>(i, j);

int kernel[6][9] = { 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    -1, -2, -1, 0, 0, 0, 1, 2, 1,
                    -1, 0, 1, -2, 0, 2, -1, 0, 1,
                    -1, -1, -1, -1, 9, -1, -1, -1, -1,
                    -1, -1, 0, -1, 0, 1, 0, 1, 1,
                    1, 2, 1, 2, 4, 2, 1, 2, 1 };

Mat img;
img.create(A.row, A.col, CV_8U);
char str[] = "kernel_1";
for (int i = 0; i < 6; ++i)
{
    conv_demo(A, kernel[i], sum);
    for (int i = 0; i < A.row; i++)
        for (int j = 0; j < A.col; j++)
            img.at<uchar>(i, j) = (uchar)sum.a[A.col * i + j];

    str[7] = char('1' + i);
    imshow(str, img);
}

waitKey(0);
return;
}

```

```

int OTSU_t(matrix& A) {

    //num_color 数组存储每个灰度值的数
    int num_color[256] = { 0 }, num = 0, all = A.row * A.col;
    double w0, w1, u0, u1, u2;
    //G 为当前灰度值, great_G 为最优阈值
    double G, great_G = 0;

    for (int i = 0; i < A.row; ++i)
    {
        for (int j = 0; j < A.col; ++j)
        {
            num_color[A.a[num]]++;

```



```

        num++;
    }
}

//low_color 数组为低于某个灰度值的总数
//low_col_mul 数组为低于某个灰度值的数乘以对应灰度值的总和
int low_color[256] = { num_color[0] }, low_col_mul[256] = { 0 };
for (int i = 1; i < 256; ++i)
{
    low_color[i] += low_color[i - 1] + num_color[i];
    low_col_mul[i] += low_col_mul[i - 1] + i * num_color[i];
}

int now_color = 0, great_color = 0;
for (now_color = 0; now_color < 256; ++now_color)
{
    w0 = low_color[now_color];
    w1 = low_color[255] - w0;
    u0 = low_col_mul[now_color];
    u1 = low_col_mul[255] - u0;

    u0 = u0 / w0;
    u1 = u1 / w1;
    w0 = w0 / all;
    w1 = w1 / all;

    u2 = w0 * u0 + w1 * u1;
    G = w0 * (u0 - u2) * (u0 - u2) + w1 * (u1 - u2) * (u1 - u2);
    if (G > great_G)
    {
        great_G = G;
        great_color = now_color;
    }
}

return great_color;
}

void OTSU() {

    system("cls"); // 清屏函数
    cout << "下面是 OTSU 算法" << endl << endl;

    //对 vs+opencv 正确配置后方可使用, 此处只给出一段读取并显示图像的参

```

考代码，其余功能流程自行设计和查阅文献

```
Mat image = imread(".\\res\\demolena.jpg", CV_LOAD_IMAGE_GRAYSCALE);  
// 图像的灰度值存放在格式为 Mat 的变量 image 中  
imshow("原图", image);
```

```
struct matrix A;  
A.row = image.rows;  
A.col = image.cols;  
  
for (int i = 0; i < A.row; i++)  
    for (int j = 0; j < A.col; j++)  
        A.a[A.col * i + j] = (int)image.at<uchar>(i, j);
```

```
int t = OTSU_t(A);
```

```
Mat img;  
img.create(A.row, A.col, CV_8U);
```

```
for (int i = 0; i < A.row; i++)  
{  
    for (int j = 0; j < A.col; j++)  
    {  
        if (A.a[A.col * i + j] > t)  
            img.at<uchar>(i, j) = (uchar)(255);  
        else  
            img.at<uchar>(i, j) = (uchar)(0);  
    }  
}
```

```
imshow("OTSU 处理图", img);
```

```
waitKey(0);  
return;
```

```
}
```

```
Mat Erode(Mat src, Mat nuclear)
```

```
{  
    Mat dst(src.rows, src.cols, CV_8U, Scalar(0));  
    for (int i = 0; i < src.rows; ++i)  
    {  
        for (int j = 0; j < src.cols; ++j)  
        {  
            //确定是否需要腐蚀  
            int flag = 1;
```

```

        for (int x = 0; x < nuclear.rows; ++x)
        {
            for (int y = 0; y < nuclear.cols; ++y)
            {
                if (flag && nuclear.at<uchar>(x, y) && (i + x) <
src.rows && (j + y) < src.cols)
                    flag    &=    nuclear.at<uchar>(x,    y)    &
src.at<uchar>(i + x, j + y);
            }
            if (!flag)
                break;
        }

        if (flag && (i + 1) < src.rows && (j + 1) < src.cols) //
完全对上时
            dst.at<uchar>(i + 1, j + 1) = uchar(255);
    }
}
return dst;
}

```

```

Mat Dilate(Mat src, Mat nuclear)
{
    Mat dst(src.rows, src.cols, CV_8U, Scalar(0));
    for (int i = 0; i < src.rows; ++i)
    {
        for (int j = 0; j < src.cols; ++j)
        {
            //确定是否需要膨胀
            if (src.at<uchar>(i, j))
            {
                for (int x = 0; x < nuclear.rows; ++x)
                {
                    for (int y = 0; y < nuclear.cols; ++y)
                    {
                        if ((i + x - 1) < src.rows && (i + x - 1) >= 0
&& (j + y - 1) < src.cols && (j + y - 1) >= 0)
                            dst.at<uchar>(i + x - 1, j + y - 1) =
uchar(255);
                    }
                }
            }
        }
    }
}

```

```

        return dst;
    }

void OTSU_etr() {

    system("cls"); // 清屏函数
    cout << "下面是其他处理方法" << endl << endl;

    char ch;
    cout << "请选择对应处理图片" << endl;
    cout << "输入 0, 选择伊丽莎白" << endl;
    cout << "输入 1, 选择雪球" << endl;
    cout << "输入 2, 选择多角星" << endl;
    cout << "输入 3, 选择船舰" << endl;
    cout << "输入 4, 选择大脑" << endl;

    Mat image;
    ch = _getch();
    while (ch < '0' || ch > '4') {
        cout << "请重新选择" << endl;
        ch = _getch();
    }

    if (ch == '0')
        image = imread(".\\res\\demolena.jpg",
CV_LOAD_IMAGE_GRAYSCALE); //灰度原图
    if (ch == '1')
        image = imread(".\\res\\snowball.jpg",
CV_LOAD_IMAGE_GRAYSCALE); //灰度原图
    if (ch == '2')
        image = imread(".\\res\\polyhedrosis.jpg",
CV_LOAD_IMAGE_GRAYSCALE); //灰度原图
    if (ch == '3')
        image = imread(".\\res\\ship.jpg", CV_LOAD_IMAGE_GRAYSCALE); //
灰度原图
    if (ch == '4')
        image = imread(".\\res\\brain.jpg",
CV_LOAD_IMAGE_GRAYSCALE); //灰度原图

    cout << "\n\n 请选择处理方法" << endl;
    cout << "输入 0, 选择腐蚀处理" << endl;
    cout << "输入 1, 选择膨胀处理" << endl;
    cout << "输入 2, 选择开运算" << endl;
    cout << "输入 3, 选择闭运算" << endl;

```

```

cout << "输入 d 或 D, 退出" << endl;

Mat binaryImage;
binaryImage.create(image.rows, image.cols, CV_8U);
for (int i = 0; i < image.rows; i++)
{
    for (int j = 0; j < image.cols; j++)
    {
        if ((int)image.at<uchar>(i, j) > 127)
            binaryImage.at<uchar>(i, j) = (uchar)(255);
        else
            binaryImage.at<uchar>(i, j) = (uchar)(0);
    }
}

int nuclear_array[3][3] = { 1, 1, 1,
                           1, 1, 1,
                           1, 1, 1 };
Mat nuclear = Mat(3, 3, CV_8U, nuclear_array);

ch = _getch();
while (1)
{
    while ((ch < '0' || ch > '3') && ch != 'd' && ch != 'D')
    {
        cout << "请重新选择" << endl;
        ch = _getch();
    }

    if (ch == '0')
    {
        imshow("原图", image);
        imshow("二值化处理", binaryImage);
        Mat erodedImage;
        imshow("我的腐蚀运算", Erode(binaryImage, nuclear));
    }

    if (ch == '1')
    {
        imshow("原图", image);
        imshow("二值化处理", binaryImage);
        Mat dilatedImage;
        imshow("我的膨胀运算", Dilate(binaryImage, nuclear));
    }
}

```



```

    }

    if (ch == '2')
    {
        imshow("原图", image);
        imshow("二值化处理", binaryImage);
        Mat OpenImage;
        imshow("我的开运算", Dilate(Erode(binaryImage, nuclear),
nuclear)); //开运算即先腐蚀再膨胀

    }

    if (ch == '3')
    {
        imshow("原图", image);
        imshow("二值化处理", binaryImage);
        Mat CloseImage;
        imshow("我的闭运算", Erode(Dilate(binaryImage, nuclear),
nuclear)); //闭运算即先膨胀再腐蚀
    }

    if (ch == 'd' || ch == 'D')
        return;

    waitKey(0);

    cout << "\n 请再次选择，观察其他处理方法" << endl;
    ch = _getch();
}

void OTSU_order() {

    system("cls"); // 清屏函数
    cout << "下面是 OTSU 等其他算法" << endl << endl;

    char ch;
    cout << "请选择对应算法" << endl;
    cout << "输入 0，选择简单 OTSU" << endl;
    cout << "输入 1，选择其他处理方法" << endl;

    ch = _getch();
}

```

```

while (ch != '0' && ch != '1')
{
    cout << "请重新输入" << endl;
    ch = _getch();
}

if (ch == '0')
    OTSU();

if (ch == '1')
    OTSU_etr();

return;
}

void wait_for_enter()
{
    cout << endl << "按回车键继续";
    while (_getch() != '\r')
        ;
    cout << endl;
}

int main()
{
    // 定义相关变量
    char choice, ch;

    while (true) // 注意该循环退出的条件
    {
        system("cls"); // 清屏函数

        menu(); // 调用菜单显示函数，自行补充完成

        // 按要求输入菜单选择项 choice
        choice = _getch();

        if (choice == '0') // 选择退出
        {
            cout << "\n\n确定退出吗?" << endl;
            cout << "如果是请输入 Y 或 y, 否则请输入任意值后回车" << endl;
            cin >> ch;
            if (ch == 'y' || ch == 'Y')
                break;
        }
    }
}

```

```

        else
            continue;
    }

    switch (choice)
    {
        // 下述矩阵操作函数自行设计并完成（包括函数参数及返回类型
        // 等），若选择加分项，请自行补充
        case '1':
            matriplus();
            break;
        case '2':
            nummulti();
            break;
        case '3':
            matritrans();
            break;
        case '4':
            matrimulti();
            break;
        case '5':
            hadamulti();
            break;
        case '6':
            conv();
            break;
        case '7':
            demo();
            break;
        case '8':
            OTSU_order();
            break;
        default:
            cout << "\n 输入错误，请从新输入" << endl;
            wait_for_enter();
    }
}

return 0;
}

```