矩阵运算大作业报告

国豪 06 班 2251079 隋建政

2023.5.25

1.设计思路与功能描述

1.1 整体思路

由于总共要实现有关矩阵运算的多种功能，因此按照一个功能利用一个函数的形式实现，在根据不同的运算方法整合出相似的内容封装为相应的函数。

1.2 各个功能

1.2.1 错误处理

错误处理主要发生在用户输入的过程中，主要分为矩阵的输入、矩阵行列的输入以及参与运算的数字的输入，将这三类操作分别封装为函数 input_rac()、input_matrix 以及 input_k。函数的主体为一个 while 循环，每次输入时均判断 cin.good()的值，若出现非法字符则会出现警告并要求用户重新输入，重新输入前利用 cin.clear()清除 cin.good()的状态并利用 cin.ignore()跳过多余的输入内容，当正确输入使得循环结束后，仍然利用 cin.clear()和 cin.ignore()返回初始状态并略过多余字符，减少程序出错的可能性。

1.2.2 矩阵输入

在各项内容的输入中，矩阵的输入是最为特殊的，由于不确定矩阵的大小，因此先要求用户自行输入矩阵的行数以及列数，再根据行列的数值动态申请相应大小的空间，以存放一个 double 类型的矩阵。在矩阵的输入中，由于是按照二维数组处理，因此是以由左到右由上到下的顺序进行输入的。

1.2.3 矩阵输出

矩阵输出的方式很简单，只要利用行列的数值将二维数组中的内容逐一输出即可，在输出过程中我加入了格式化控制符 setw(7)，目的是让输出的矩阵看起来更加整齐美观，更容易辨别行与列的界限。

1.2.3 矩阵的基础运算

该程序的前六项为矩阵的基础运算，其各个功能的实现方法大同小异，首先要求用户输入相应的内容（如矩阵、行列、参与运算的数字等），其次根据相应不同的功能实现进行不同的计算操作并得到结果矩阵，最后一步是将结果矩阵输出，大体通过以上三步就实现了所有的功能。

1.2.4 矩阵卷积

在矩阵的各种基础运算中，卷积操作是较为特殊的一个。由于要求该程序的卷积 padding 设置为 1，那么在矩阵输入的阶段便进行附加 padding 的操作，即将原先的 input_matrix()升级为函数 input_matrix_with_padding()，首先初始矩阵就设置为用户输入行列均大二的矩阵，并将初始元素全部设置为 0，在读入用户键盘输入的数据并设置矩阵元素时仅设置 matrix[1][1]到 matrix[row][column]的的元素，就完成了原先矩阵外套一圈 0 的操作。

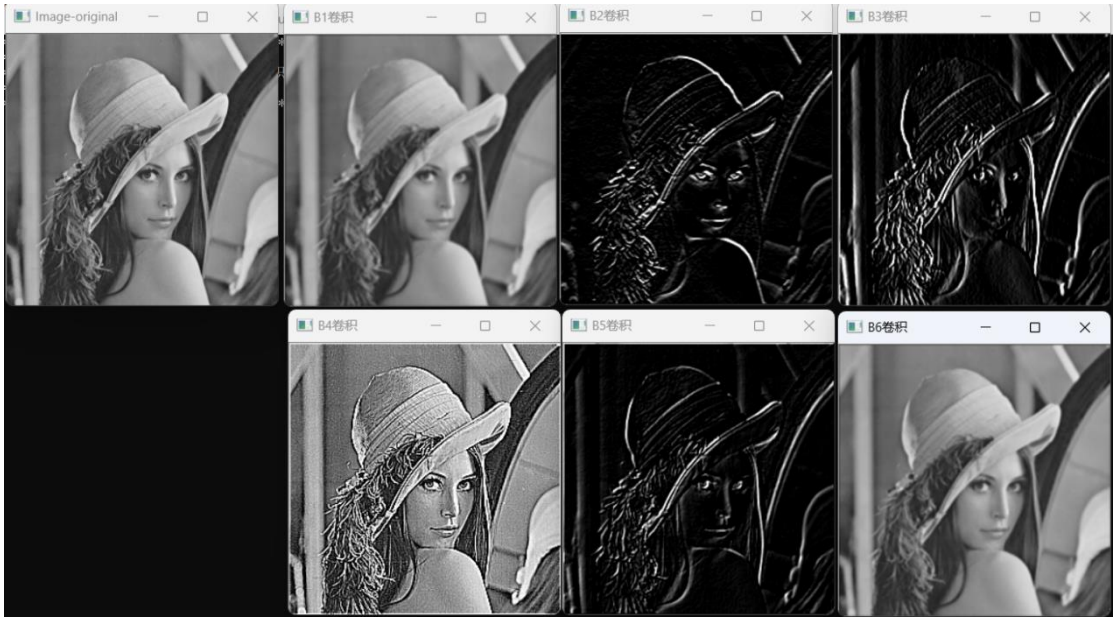
由于 Kernel size 设置为 3，padding 设置为 1 的卷积操作会得到原先大小的矩阵，因此结果矩阵设置为原先大小即可。计算卷积时以结果矩阵每个元素的位置进行计算，将相应的位置传入 calculate_convolution()函数就可进行相应的操作，得到相应位置的元素数值。

1.2.5 卷积应用

卷积应用的主体思路主要分为三步。首先将原图的 Mat 类转换为可用的二维数组（矩阵）形式，然后将得到的矩阵对相应的卷积核进行卷积运算，最后是将卷积运算得到的矩阵转换为 Mat 类，并通过 imshow 函数打开图片。

将 Mat 类转换为矩阵时，直接利用 input_matrix_with_padding()类似的算法即可，再将卷积操作和转换 Mat 共同放入一个 output_image()函数中，只需传入相应的卷积核函数，以及图片名称即可实现区分。

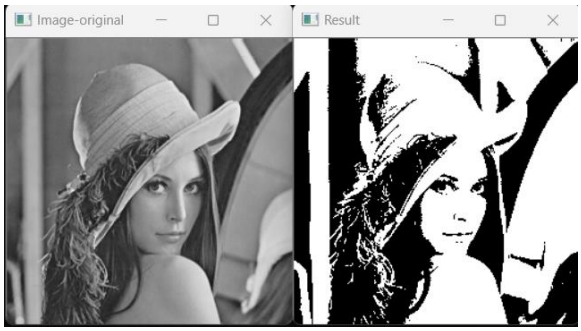
以下是我卷积应用的结果：

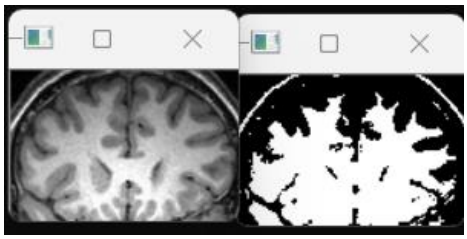


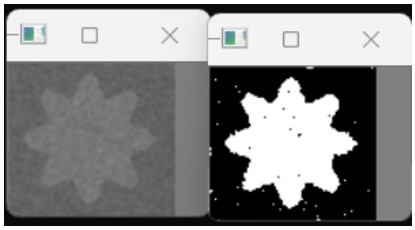
1.26 OSTU 算法

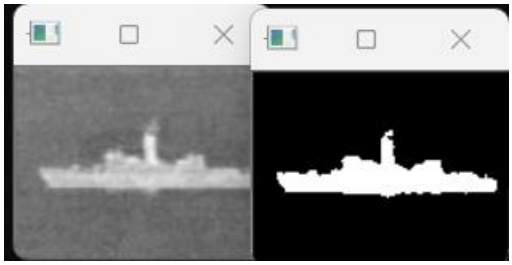
OSTU 算法的实现不算困难，大体思路为将所有灰度的像素点分为两类，根据计算方法从灰度 0 遍历到灰度 255，寻找一个最合适的阈值，将阈值以下的改为 0，将阈值以上的改为 255，就完成了二值化的操作。但经过反复修改，程序中的 OSTU 算法项只能一项一项地输出图片，每输出一组图片就需要手动点击关闭才能现实下一组，只能辛苦老师和助教哥哥姐姐们了。

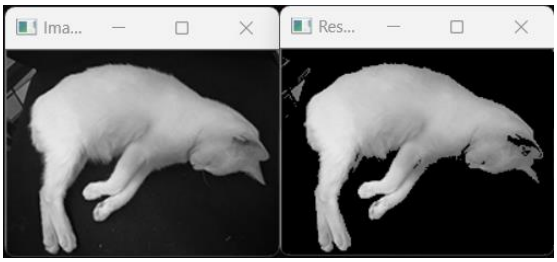
以下是我 OSTU 算法地处理结果：











2.遇到的问题及解决方案

2.1 卷积应用中的问题 1

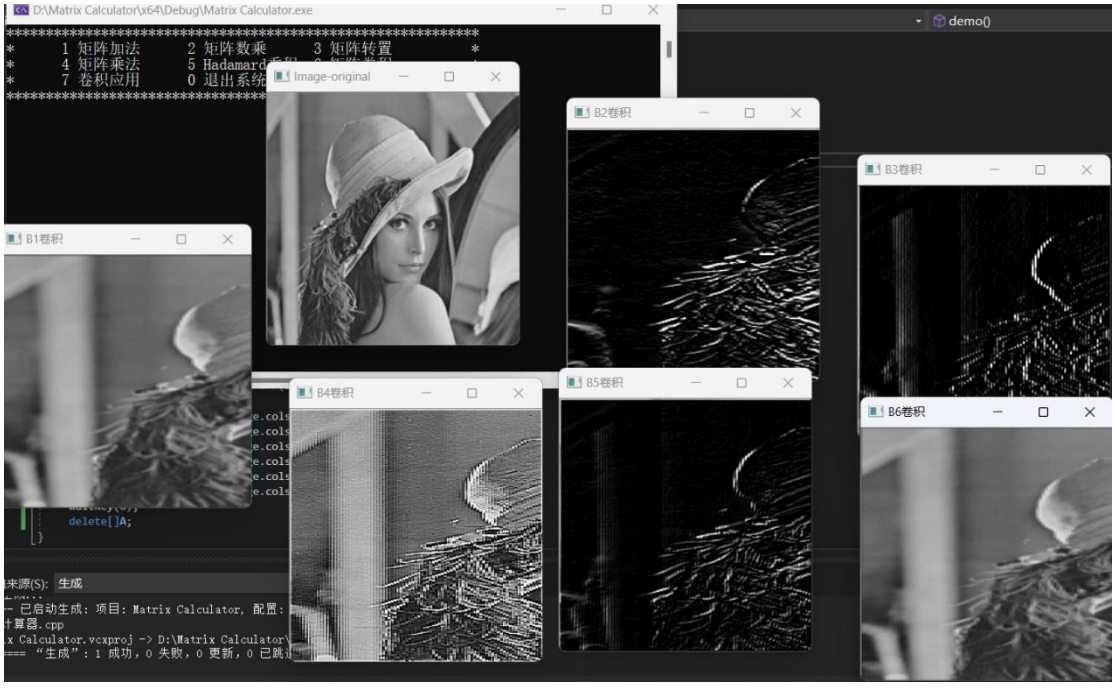
在刚开始尝试卷积应用时，我遇到了一个困扰我很久的问题，就是但所有操作就绪，在打开计算完卷积的图片时，会出现一张完全灰色的图片并且电脑出现死机。

起初我认为是 Mat 与二维数组相互转换过程中出的问题，为此我反复建材检查这几段程序但是却并没有发现问题，在卷积计算过程中也不存在计算的问题。于是我将原图也放在新图片的位置进行打开，发现也会出现相同的问题，后来经过查验资料，我了解到原来是 imshow()函数的后面必须添加 cv::waitkey()函数，代表图像打开的时长，若参数为 0 则无限期打开，在函数结尾添加了 waitkey(0)

后便成功解决了问题。

2.2 卷积应用中的问题 2

在终于成功打开图片后，我遇到了新的问题，卷积后的图片呈现放大拉伸



的效果，具体如下图。

在确认其他均无误的情况下，我认为可能是图片在初始化生成 Mat 类时出现了问题，于是我将初始生成 Mat 转化而成的数组再转化回 Mat 类并进行打开，发现果然也出现了相同的问题，经过资料的查找，我在 imread 函数中加入了参数 IMREAD_GRAYSCALE，结果成功解决了问题。

2.3opencv 配置时遇到的问题

当配置 opencv 时，我尝试配置了两次，始终显示头文件以及 cv 命名空间无法使用，一直到第三次配置时我才恍然大悟，在项目属性进行配置时，我只点击了确认而没有点击应用，因此实际上一直没有配置成功，当点击应用时问题一

下子就解决了。

3.心得体会

在本次大作业的完成过程中，我对于动态内存申请有了全新的认识，其自由灵活的操作方法让本次作业变得得心应手，有很好的规避掉了二维数组不可以用变量作为元素数量的问题，作为参量传参时也无需提前规定数组的大小，可谓一举多得。

同时也了解了矩阵卷积运算的方法，在图像卷积操作方面，以结果来看，B1 与 B6 似乎是对原图进行了不同程度的模糊化处理，而 B1 的模糊化程度明显更重，这通过矩阵运算似乎很好理解，B1 为完全平均化，而 B6 则是越靠近中间的像素点在最终画面的占比越大，因此 B6 的模糊程度较低也是可以理解的，B4 的结果同样容易理解，着重中间的像素点自然会让画面的对比度（不知道这个术语是否正确）更强，其余三个图像让我并没有什么头绪。不过似乎矩阵卷积可以利用在修图软件中的滤镜功能，以及调整画面对比度饱和度的功能当中（瞎猜的）。

最后是收获了软件配置方面的教训，要记得点应用而不是确认就完了。

4.源代码

```
#include <conio.h>
#include <iostream>
#include <iomanip>
#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
void wait_for_enter()
{cout << endl
        << "按回车键继续";
    while (_getch() != '\r');
    cout << endl
        << endl;
}
void menu()
{   for (int i = 0; i < 60; 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        0 退出系统                                *" << endl;
    for (int i = 0; i < 60; i++)
        cout << '*';
    cout << endl;
}
void input_rac(int *p_row,int *p_column)
{
    cout << "请输入矩阵的行数以及列数：" << endl;
    while (1) {
        cin >> *p_row >> *p_column;
        if (!cin.good()) { cout << "输入有误，请重新输入：" << endl;
            cin.clear();
            cin.ignore(1024, '\n');}
        else
```

```
            break;
        }
        cin.clear();
        cin.ignore(1024, '\n');
}
void input_k(double* p_k)

        while (1) {
            cin >> *p_k;
            if (!cin.good()) {
                cout << "输入有误，请重新输入： " << endl;
                cin.clear();
                cin.ignore(1024, '\n');
            }
            else
                break;
        }
        cin.clear();
        cin.ignore(1024, '\n');
}

void input_matrix(double** A, int row, int column)
{
        int i, j;
        while (1) {
            for (i = 0; i < row; i++)
                for (j = 0; j < column; j++)
                    cin >> A[i][j];
            if (!cin.good()) {
                cout << "输入有误，请重新输入： " << endl;
                cin.clear();
                cin.ignore(1024,'\n');
            }
            else
                break;
        }
        cin.clear();
        cin.ignore(1024, '\n');
}

void output_matrix(double** A, int row, int column)
{
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < column; j++)
```

```
            cout << setw(7) << A[i][j] << ' ';
        cout << endl;
    }
}

void input_matrix_with_padding(double** A, int row, int column)
{
    for (int i = 0; i < row + 2; i++)
        for (int j = 0; j < column + 2; j++)
            A[i][j] = 0;
    while (1) {
        for (int i = 1; i < row+1; i++)
            for (int j = 1; j < column+1; j++)
                cin >> A[i][j];
        if (!cin.good()) {
            cout << "输入有误，请重新输入： " << endl;
            cin.clear();
            cin.ignore(1024, '\n');
        }
        else
            break;
    }
    cin.clear();
    cin.ignore(1024, '\n');

}

double calculate_convolution(double** A, double** K, int i0, int j0)
{
    double sum = 0;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            sum += (K[i][j] * A[i0 + i][j0 + j]);
    return sum;
}

int calculate_convolution(int **A, int K[3][3], int i0, int j0)
{
    int sum = 0;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            sum += (K[i][j] * A[i0 + i][j0 + j]);
    return sum;
}
```

```
void output_image(int row, int column, int K[3][3], int **A,const char ch[])
{
    int sum = 0;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            sum += K[i][j];
    if (sum == 0)
        sum = 1;
    int** B = new int* [row];
    for (int i = 0; i < row; i++)
        B[i] = new int[column];
    for (int i = 0; i < row; i++)
        for (int j = 0; j < column; j++) {
            B[i][j] = calculate_convolution(A, K, i, j)/sum;
            if (B[i][j] < 0)
                B[i][j] = 0;
            if (B[i][j] > 255)
                B[i][j] = 255;
        }
    Mat image1(row, column, CV_8UC1);
    for (int i = 0; i < row; i++)
        for (int j = 0; j < column; j++)
            image1.at<uchar>(i, j) = B[i][j];
    imshow(ch, image1);
    delete[]B;
}

void demo()
{
    // /* 对 vs2019+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("demolena.jpg", IMREAD_GRAYSCALE); // 图像的灰度值存放在格式为 Mat 的变量 image 中
     imshow("Image-original", image);


    // 提示：Mat 格式可与数组相互转换
     int B1[3][3] = { 1,1,1,1,1,1,1,1,1 }, B2[3][3] = { -1,-2,-1,0,0,0,1,2,1 }, B3[3][3] = { -1,0,1,-2,0,2,-1,0,1 },
         B4[3][3] = { -1,-1,-1,-1,9,-1,-1,-1,-1 }, B5[3][3] = { -1,-1,0,-1,0,1,0,1,1 }, B6[3][3] = { 1,2,1,2,4,2,1,2,1 };
     int** A = new int* [image.rows + 2];
     for (int i = 0; i < image.rows+2; i++)
         A[i] = new int[image.cols+2];
     for (int i = 0; i < image.rows +2; i++)
```

```
            for (int j = 0; j < image.cols + 2; j++)
                A[i][j] = 0;
        for (int i = 1; i < image.rows+1; i++)
            for (int j = 1; j < image.cols+1; j++)
                A[i][j] = image.at<uchar>(i-1,j-1);

        output_image(image.rows, image.cols, B1, A, "B1 卷积");
        output_image(image.rows, image.cols, B2, A, "B2 卷积");
        output_image(image.rows, image.cols, B3, A, "B3 卷积");
        output_image(image.rows, image.cols, B4, A, "B4 卷积");
        output_image(image.rows, image.cols, B5, A, "B5 卷积");
        output_image(image.rows, image.cols, B6, A, "B6 卷积");
        waitKey(0);
        delete[]A;
}

void matriplus()
{
    system("cls");
    int row, column, * p_row = &row, * p_column = &column;

    input_rac(p_row,p_column);
    cout << "请输入矩阵" << row << "行" << column << "列的矩阵 A：" << endl;
    double** A = new double* [row];
    double** B = new double* [row];
    for (int i = 0; i < row; i++) {
        A[i] = new double[column];
        B[i] = new double[column];
    }
    input_matrix(A, row, column);
    cout << "请输入矩阵" << row << "行" << column << "列的矩阵 B：" << endl;
    input_matrix(B, row, column);

    for (int i = 0; i < row; i++)
        for (int j = 0; j < column; j++)
            A[i][j] += B[i][j];

    cout << "矩阵 C=A+B 为：" << endl;
    output_matrix(A, row, column);

    delete[]A;
    delete[]B;
}
```

```
void nummulti()
{
    system("cls");
    int row, column, * p_row = &row, * p_column = &column;
    double k, * p_k = &k;

    input_rac(p_row, p_column);
    double** A = new double* [row];
    for (int i = 0; i < row; i++) {
        A[i] = new double[column];
    }
    cout << "请输入矩阵" << row << "行" << column << "列的矩阵 A：" << endl;
    input_matrix(A, row, column);
    cout << "请输入与之相乘的实数 k：" << endl;
    input_k(p_k);

    for (int i = 0; i < row; i++)
        for (int j = 0; j < column; j++)
            A[i][j] *= k;

    cout << "矩阵 k*A 为：" << endl;
    output_matrix(A, row, column);
    delete[]A;
}

void matritrans()
{

    system("cls");
    int row, column, * p_row = &row, * p_column = &column;

    input_rac(p_row, p_column);
    double** A = new double* [row];
    double** B = new double* [column];
    for (int i = 0; i < row; i++)
        A[i] = new double[column];
    for (int i = 0; i < column; i++)
        B[i] = new double[row];

    cout << "请输入矩阵" << row << "行" << column << "列的矩阵 A：" << endl;
    input_matrix(A, row, column);

    for (int i = 0; i < row; i++)
        for (int j = 0; j < column; j++)
```

```
            B[j][i] = A[i][j];

    cout << "矩阵 A 的转置为： " << endl;
    output_matrix(B, column, row);
    delete[]A;
    delete[]B;
}

void matrimulti()
{
    system("cls");
    int row1, column1, row2, column2, * p_r1 = &row1, * p_r2 = &row2, * p_c1 = &column1, * p_c2 = &column2;

    while (1) {
        cout << "左侧的矩阵： " << endl;
        input_rac(p_r1, p_c1);
        double** A = new double* [row1];
        for (int i = 0; i < row1; i++) {
            A[i] = new double[column1];
        }
        cout << "请输入矩阵" << row1 << "行" << column1 << "列的矩阵 A： " << endl;
        input_matrix(A, row1, column1);

        cout << "右侧的矩阵： " << endl;
        input_rac(p_r2, p_c2);
        double** B = new double* [row2];
        for (int i = 0; i < row2; i++) {
            B[i] = new double[column2];
        }
        cout << "请输入矩阵" << row2 << "行" << column2 << "列的矩阵 B： " << endl;
        input_matrix(B, row2, column2);

        if (column1 == row2) {
            double** C = new double* [row1];
            for (int i = 0; i < row1; i++)
                C[i] = new double[column2];
            for (int i = 0; i < row1; i++) {
                for (int j = 0; j < column2; j++) {
                    C[i][j] = 0;
                    for (int k = 0; k < row2; k++)
                        C[i][j] += (A[i][k] * B[k][j]);
                }
            }
```

```
            cout << "矩阵 A 与矩阵 B 的乘积矩阵为：" << endl;
            output_matrix(C, row1, column2);
            delete[]A;
            delete[]B;
            delete[]C;
            break;
        }

        cout << "左侧矩阵的列数不等于右侧矩阵的行数，请重新输入" << endl;
        delete[]A;
        delete[]B;
    }
}

void hadamulti()
{
    system("cls");
    int row, column, * p_row = &row, * p_column = &column;

    input_rac(p_row, p_column);
    double** A = new double* [row];
    double** B = new double* [row];
    for (int i = 0; i < row; i++) {
        A[i] = new double[column];
        B[i] = new double[column];
    }
    cout << "请输入矩阵" << row << "行" << column << "列的矩阵 A：" << endl;
    input_matrix(A, row, column);
    cout << "请输入矩阵" << row << "行" << column << "列的矩阵 B：" << endl;
    input_matrix(B, row, column);

    for (int i = 0; i < row; i++)
        for (int j = 0; j < column; j++)
            A[i][j] *= B[i][j];

    cout << "矩阵 C=A+B 为：" << endl;
    output_matrix(A, row, column);
    delete[]A;
    delete[]B;
}

void conv()
{
    system("cls");
```

```
    int row, column, * p_row = &row, * p_column = &column;

    input_rac(p_row, p_column);
    double** A = new double* [row+2];
    for (int i = 0; i < row + 2; i++)
        A[i] = new double[column + 2];
    cout << "请输入矩阵" << row << "行" << column << "列的矩阵 A:  " << endl;
    input_matrix_with_padding(A, row, column);
    cout << "请输入 Kernel 矩阵(3x3):" << endl;
    double** K = new double*[3];
    for (int i = 0; i < 3; i++)
        K[i] = new double[3];
    input_matrix(K, 3, 3);

    double** B = new double* [row];
    for (int i = 0; i < row; i++)
        B[i] = new double[column];
    for (int i = 0; i < row; i++)
        for (int j = 0; j < column; j++)
            B[i][j] = calculate_convolution(A, K, i, j);

    cout << "矩阵 A 的卷积计算结果为:  " << endl;
    output_matrix(B, row, column);
    delete[]A;
    delete[]B;
    delete[]K;
}

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

    wait_for_enter();
    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

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

        if (choice == '0') // 选择退出
        {
            cout << "\n 确定退出吗?" << 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;
            default:
                cout << "\n  输入错误，请从新输入" << endl;
                wait_for_enter();
        }
        wait_for_enter();
    }
    return 0;
}
```