C/C++ Programming

Project4: A Class to Describe a Matrix

张立远 12012724 电子与电气工程系

Introduction

在本次project中，我们着重利用C++语言以及其面向对象的特性，实现一个Matrix的类以及其相关的操作。相比于C的struct，C++的class更加灵活，提供了更多可用的API，提高了数据访问的安全性，可以实现类继承和泛型等面向对象编程的特性。在我的代码中，实现了以下几个主要功能：

- `class Matrix`的定义、成员变量、用`raw pointer`(*之所以没有用`shared_ptr`，是因为`shared_ptr`对数组的适配性不太好，例如`[]`操作符访问数组元素就不适用于`shared_ptr`，需要额外重载，有点麻烦，`delete []`也不适用于`shared_ptr`*)表示矩阵的`data`，同时有一个成员变量`count::short`来表示当前指向`data`的指针数量，以实现动态内存管理和安全释放
- 模板类：`Matrix<T>` 以实现泛型，`T` 可以是`int,short,unsigned char,float,double`
- 基于软拷贝（soft copy）的拷贝构造器
- 可安全、动态地释放`data`指针指向的内存空间的析构函数： `~Matrix()`
- 操作符重载: = , == , () , + , - , *
- 矩阵加法模板函数，可适用于不同类型的`Matrix`
- 矩阵乘法模板函数，可适用于不同类型的`Matrix`

  (*由于本次project的重点更多倾向于C++的面向对象特性，而不是快速矩阵乘法，因此这里没有包括project3中的快速矩阵乘法，这里的实现方法是最朴素的三重循环矩阵乘法，且本次的测试用矩阵的维数也较小，不进行运行时间的测算分析。)
- 矩阵初始化赋值模板函数：将每一个元素赋予相同的值，可适用于不同类型的`Matrix`
- 矩阵初始化赋值模板函数：将每一个元素赋予一个指定范围的随机值，可适用于不同类型的`Matrix`
- 矩阵输出函数：可将矩阵打印到`console`中
- 矩阵输出函数：可将矩阵写入一个`CSV`文件中
- 矩阵ROI(Region of Interest)的访问，利用操作符重载实现，i.e. `ROI(x1,x2,y1,y2,pmat)`

Implementation

文件结构

```
-projec4
    --main.cpp
    --matirx.hpp
```

之所以没有把部分函数写在一个额外的matrix.cpp中，是因为几乎所有的函数都是模板函数，经过我的测试以及网上资料的建议，模板函数不建议卸载.cpp文件中，会导致链接时编译器不进行特化，因而导致main.cpp无法识别到模板函数。必要的代价，就是.hpp文件过长。

类声明和类定义

自定义的MyChar类，为了方便实现对unsigned char类型变量的加法操作符重载，因为操作符重载函数一般来说必须是某个类的成员函数，而不能孤立地存在。我们把两个unsigned char的加法定义为它们的ASCII码之和，众所周知，unsigned char的范围时0~255，而两个unsigned char相加可能会超出这个范围，因此如果发生溢出时，结果会对256取模，保证结果约束在0~255的范围内。

```
class MyChar
{
    unsigned char val;

public:
    MyChar(char val) : val(val) {}
    unsigned char get()
    {
        return this->val;
    }
    unsigned char operator+(MyChar &other)
    {
        int result;
        result = static_cast<int>(val) + static_cast<int>(other.val);
        if (result > 255)
        {
            result = result - 256;
        }
        return static_cast<unsigned char>(result);
    }
};
```

模板类Matrix，首先包含一个断言static_assert，规定泛型T只能为int,short,unsigned char,float,double中的一个，否则程序会结束并报错。使用T *来表示矩阵数据存放位置，count::short是用来对指向data的值的指针数量进行计数，以实现动态内存管理和安全释放。

```
#define TEMPLATE template <typename T>
TEMPLATE
class Matrix
{

    //ChatGPT usage starts here
    static_assert(std::is_same<T, int>::value || std::is_same<T, short>::value ||
std::is_same<T, unsigned char>::value ||
                      std::is_same<T, float>::value || std::is_same<T, double>::value,
                  "T must be int, short, unsigned char, float or double");
    //ChatGPT usage ends here

    size_t rows;
    size_t cols;
    T *data;
    short count;

public:
```

```
    ... ...
```

公共成员函数

首先包含一个count的setter，这里有一个非常tricky的过程，这么做是为了在下面的copy constructor:

Matrix(const Matrix &other) : rows(other.rows), cols(other.cols), data(new T[other.rows * other.cols]())中能够更改other.count的值（因为参数列表中other带有关键字const，其成员变量不能直接被更改），而至于为什么other一定要是const的，是因为如果不是这样，在调用copy constructor的时候编译器就会报错：

```
cannot bind non-const lvalue reference of type ‘Matrix<unsigned char>&’ to an rvalue of type ‘Matrix<unsigned char>
```

这个错误通常发生在试图将一个临时对象（rvalue）绑定到非常量左值引用（non-const lvalue reference）时，这来源于C++的一个语法规范，它要求非常量左值引用不能绑定到临时对象，因为这样的绑定通常是不安全的。这是为了防止在引用生命周期内，临时对象销毁导致引用无效，从而产生未定义的行为。

接下来是constructor，有两个版本，分别是带参和不带参的，在初始化时，count的值将被设置为1，malloc为数组分配合适的内存空间。

再然后时copy constructor，这里实现的是软拷贝，因为对于数据量大的矩阵，硬拷贝会重新开辟一块新的内存空间，导致空间资源占用过多。当一个矩阵被拷贝时，成员变量count的值会+1，表明有一个新的指针指向数组的内存空间，同时拷贝的target矩阵的count会被赋值成和被拷贝的矩阵一样，表明它们现在共享这块内存空间。

destructor通过一个if语句，判断count的值，只有当count的值大于0时，data才会被释放，同时count的值-1，表明少了一个指向数据内存空间的指针存在，这样避免了指针重复释放的问题，提高了程序的稳健性。

```
    ... ...
void setCount(short count)
    {
        this->count = count;
    }

    // constructor without parameters
    Matrix() : rows(0),
               cols(0),
               count(1)
    {
        data = (T *)malloc(rows * cols * sizeof(T));
    }

    // constructor with parameters
    Matrix(size_t rows, size_t cols) : rows(rows),
                                       cols(cols),
                                       count(1)
    {
        data = (T *)malloc(rows * cols * sizeof(T));
    }
    // copy constructor
    Matrix(const Matrix &other) : rows(other.rows), cols(other.cols), data(new T[other.rows *
other.cols]())
```

```
    {
        this->rows = other.rows;
        this->cols = other.cols;
        this->data = other.data;
        setCount(this->count + 1);
        this->count = other.count;
    }

    // destructor
    ~Matrix()
    {
        if (count > 0)
        {
            delete[] data;
            count--;
        }
    }
    ... ...
```

操作符重载：

- 矩阵元素访问，i.e. `mat(i,j)`: 这里做了一个输入参数判断，若超界则无法访问，抛出异常
- 矩阵乘法: 这里做了输入参数判断，判断输入的两个矩阵的维数是否符合矩阵乘法规则，若不符合就直接`exit()`，在函数体中调用成员函数`mat_multiply(Matirx*,Matrix*,Matrix*)`
- 矩阵加法：这里做了输入参数判断，判断输入的两个矩阵是否维数相等，若不相等就直接`exit()`，在函数体内调用成员函数`mat_add(Matirx*,Matrix*,Matrix*)`
- 矩阵减法：这里做了输入参数判断，判断输入的两个矩阵是否维数相等，若不相等就直接`exit()`，在函数体内调用成员函数`mat_substract(Matirx*,Matrix*,Matrix*)`
- 判断两个矩阵是否相等, i.e. `mat1 == mat2`：值得注意的是，我在这里定义的“相等”指的是两个矩阵维数相同，且`data`数组中的每一个元素具有相同的值，并不要求`data`指针指向同一个地址且`count`的值相同。这样定义更加符合我们实际运算的需求。

```
    ... ...
// operator overloading for element access
    const T &operator()(size_t i, size_t j) const
    {
        if (i >= rows || j >= cols)
        {
            throw out_of_range("Index out of range");
        }
        return data[i * cols + j];
    }

    // operator overloading for matrix multiplication

    Matrix operator*(Matrix &other)
    {
        if (cols != other.rows)
        {
            printf("the two matrices cannot be multiplied due to dimension, (rows,cols) of a:
(%zu,%zu), (rows,cols) of b: (%zu,%zu),",
                   rows, cols, other.rows, other.cols);
            exit(EXIT_FAILURE);
        }
        Matrix out = Matrix(rows, other.cols);
```

```
    bool result = mat_multiply(this, &other, &out);
    if (result)
    {
        return out;
    }
    else
    {
        cout << "exiting..." << endl;
    }
}

// operator overloading for matrix addition
Matrix operator+(Matrix &other)
{
    Matrix out = Matrix(rows, other.cols);
    bool result = mat_add(this, &other, &out);
    if (result)
    {
        return out;
    }
    else
    {
        cout << "exiting..." << endl;
        exit(EXIT_FAILURE);
    }
}

// operator overloading for matrix substraction
Matrix operator-(Matrix &other)
{
    Matrix out = Matrix(rows, other.cols);
    bool result = mat_substract(this, &other, &out);
    if (result)
    {
        return out;
    }
    else
    {
        cout << "exiting..." << endl;
        exit(EXIT_FAILURE);
    }
}

// operator overloading to decide whether two matrices are equal
bool operator==(Matrix &other)
{
    if (this->rows == other.rows && this->cols == other.cols)
    {

        for (size_t i = 0; i < this->rows * this->cols; i++)
        {
            if (this->data[i] != other.data[i])
            {
                return false;
            }
        }
        return true;
    }
```

```
        else
            return false;
    }
    ... ...
```

以下是更多的成员函数：

- 矩阵乘法函数，运用简单的三重循环实现
- 矩阵加法函数，当泛型T为unsigned char时，有一个相应的if条件分支来调用在MyChar类中重载过的unsigned char加法操作符，来实现两个unsigned char的加法
- 矩阵加法函数，当泛型T为unsigned char时，有一个相应的if条件分支来调用在MyChar类中重载过的unsigned char减法操作符，来实现两个unsigned char的减法
- 提取矩阵的ROI(Region of Interest)，输入参数x1,x2,y1,y2指定了我们想要的矩阵区域为从第y1+1行第x1+1个元素到第y2+1行第x2+2个元素之间所构成的矩形区域
- 给矩阵中每一个元素赋予一个随机值，这里我用到了C++11中的产生随机数的方法：default_random_engine(time(0))，这个函数会根据当前时间返回一个独一无二的随机数种子（随机数发生器），当泛型T为整形（包括int,short,unsigned char）时，我们会调用uniform_int_distribution(begin,end)，来返回一个整数类型的、均匀分布的、在指定范围内的随机数；当泛型T为浮点型（包括float,double）时，我们会调用uniform_real_distribution(begin,end)，来返回一个浮点型的（实数）、均匀分布的、在指定范围内的随机数。当然，我们也可以选择其它我们想要的概率分布，如正态分布、指数分布等等，都有相应的API
- 给矩阵中的每一个元素赋予一个指定的值
- 打印矩阵中的每一个元素到console中
- 将矩阵按照CSV的格式，即同一行中相邻元素用逗号隔开，行与行之间用换行符隔开，写入CSV文件中。CSV格式是被广泛运用于数据科学中的数据交换标准格式，科学计算工具如MATLAB，pandas，numpy等都支持CSV作为数据导入，因此，添加这样一个方法，可以方便我们把矩阵输出成CSV，再导入如MATLAB之类的科学计算平台进行计算，通过比对两者结果来检验我们程序运算的正确性

```
bool mat_multiply(Matrix *pa, Matrix *pb, Matrix *pout)
    {
        if (pa->cols != pb->rows)
        {
            printf("the two matrices cannot be multiplied due to incompatible dimensions,
(rows,cols) of a: (%zu,%zu), (rows,cols) of b: (%zu,%zu),",
                   pa->rows, pa->cols, pb->rows, pb->cols);
            return false;
        }

        size_t m = pa->rows;
        size_t n = pa->cols;
        size_t p = pb->cols;

        for (size_t i = 0; i < m; i++)
        {
            for (size_t j = 0; j < p; j++)
            {

                T *prowA = &(pa->data[i * n]);
                T *prowB = &(pb->data[j]);
                pout->data[i * n + j] = 0; // 初始化
                for (size_t k = 0; k < n; k++)
                {
                    pout->data[i * n + j] += prowA[k] * prowB[k];
```

```
                }
            }
        }
        return true;
    }

    bool mat_add(Matrix *pa, Matrix *pb, Matrix *pout)
    {
        if (pa->cols != pb->rows || pa->rows != pb->rows)
        {
            printf("the two matrices have different sizes, cannot be added, (rows,cols) of a:
(%zu,%zu), (rows,cols) of b: (%zu,%zu),",
                   pa->rows, pa->cols, pb->rows, pb->rows);
            return false;
        }
        for (size_t i = 0; i < pa->cols * pb->rows; i++)
        {
            if (is_same<T, unsigned char>::value)
            {
                MyChar *char_a = new MyChar(pa->data[i]);
                MyChar *char_b = new MyChar(pb->data[i]);
                pout->data[i] = char_a->get() + char_b->get(); // Operator already overloaded
for char add
            }
            pout->data[i] = pa->data[i] + pb->data[i];
        }
        return true;
    }

    bool mat_substract(Matrix *pa, Matrix *pb, Matrix *pout)
    {
        if (pa->cols != pb->rows || pa->rows != pb->rows)
        {
            printf("the two matrices have different sizes, cannot be conducting substraction,
(rows,cols) of a: (%zu,%zu), (rows,cols) of b: (%zu,%zu),",
                   pa->rows, pa->cols, pb->rows, pb->rows);
            return false;
        }
        for (size_t i = 0; i < pa->cols * pb->rows; i++)
        {
            if (is_same<T, unsigned char>::value)
            {
                MyChar *char_a = new MyChar(pa->data[i]);
                MyChar *char_b = new MyChar(pb->data[i]);
                pout->data[i] = char_a->get() - char_b->get(); // Operator already overloaded
for char substraction
            }
            pout->data[i] = pa->data[i] - pb->data[i];
        }
        return true;
    }


    Matrix ROI(size_t x1, size_t x2, size_t y1, size_t y2, Matrix * pmat)
    {
        if(x2-x1+1>pmat->cols || y2-y1+1>pmat->rows)
        {
            cout << "the ROI you desire must be a subset of the matrix" << endl;
```

```
            exit(EXIT_FAILURE);
        }
        Matrix result = Matrix(y2-y1+1,x2-x1+1);
        for(size_t i=y1;i<y2+1;i++)
        {
            for(size_t j=x1;j<x2+1;j++)
            {
                result.data[(i-y1)*pmat->cols + j - y1] = pmat->data[i*pmat->cols + j];
            }

            result.count = pmat->count;
        }
        return result;
    }

    //ChatGPT reference begins here
    bool AssignRandomValue(float begin, float end, Matrix *pmat)
    {
        static default_random_engine generator(time(0));

        if (is_same<T, int>::value)
        {
            uniform_int_distribution<int> distribution(begin, end);
            for (size_t i = 0; i < (pmat->rows) * (pmat->cols); i++)
            {
                pmat->data[i] = distribution(generator);
            }
        }
        else if (is_same<T, short>::value)
        {
            uniform_int_distribution<int> distribution(begin, end);
            for (size_t i = 0; i < (pmat->rows) * (pmat->cols); i++)
            {
                pmat->data[i] = distribution(generator) % 32768;
            }
        }

        else if (is_same<T, float>::value || is_same<T, double>::value)
        {
            uniform_real_distribution<float> distribution(begin, end);
            for (size_t i = 0; i < (pmat->rows) * (pmat->cols); i++)
            {
                pmat->data[i] = distribution(generator);
            }
        }
        else if (is_same<T, unsigned char>::value)
        {
            uniform_int_distribution<int> distribution(begin, end);
            for (size_t i = 0; i < (pmat->rows) * (pmat->cols); i++)
            {
                pmat->data[i] = distribution(generator) % 256;
            }
        }
        return true;
    }
    //ChatGPT reference ends here

    bool AssignSameValue(T value, Matrix *pmat)
```

```
    {
        for (size_t i = 0; i < pmat->rows * pmat->cols; i++)
        {
            pmat->data[i] = value;
        }
        return true;
    }

    bool printMatrix(Matrix *pmat)
    {
        if (is_same<T, int>::value || is_same<T, short>::value)
        {
            for (size_t i = 0; i < pmat->rows; i++)
            {
                for (size_t j = 0; j < pmat->cols; j++)
                {
                    printf("%d\t", pmat->data[i * pmat->cols + j]);
                }
                printf("\n");
            }
            return true;
        }
        else if (is_same<T, double>::value || is_same<T, float>::value)
        {
            for (size_t i = 0; i < pmat->rows; i++)
            {
                for (size_t j = 0; j < pmat->cols; j++)
                {
                    printf("%f\t", pmat->data[i * pmat->cols + j]);
                }
                printf("\n");
            }
            return true;
        }
        else if (is_same<T, unsigned char>::value)
        {
            for (size_t i = 0; i < pmat->rows; i++)
            {
                for (size_t j = 0; j < pmat->cols; j++)
                {
                    printf("%c\t", pmat->data[i * pmat->cols + j]);
                }
                printf("\n");
            }
            return true;
        }
        else
        {
            printf("Error: type of T must be one of the five: int, short, unsigned char, float,
double\n");
            return false;
        }
    }

    bool writeMatrixToCSV(Matrix *pmat, string filename)
    {
        std::ofstream outFile(filename);
```

```
        if (!outFile)
        {
            std::cerr << "Error: Could not open the file!" << std::endl;
            return false;
        }

        for (size_t i = 0; i < pmat->rows; i++)
        {
            for (size_t j = 0; j < pmat->cols; j++)
            {
                outFile << pmat->data[i * pmat->cols + j];
                if (j < pmat->cols - 1)
                {
                    outFile << ",";
                }
            }
            outFile << std::endl;
        }

        outFile.close();

        if (outFile.fail())
        {
            std::cerr << "Error: Could not close the file!" << std::endl;
            return false;
        }

        return true;
    }
};
```

Test

矩阵运算

- 加法（减法结果同理，这里只展示加法的结果）：

  整数类型矩阵加法

```
int main()
{
    Matrix<int> mat_int_1 = Matrix<int>(5,5);
    Matrix<int> mat_int_2 = Matrix<int>(5,5);
    mat_int_1.AssignRandomValue(1,10,&mat_int_1);
    mat_int_2.AssignRandomValue(1,10,&mat_int_2);
    Matrix<int> mat_int_added = mat_int_1 + mat_int_2;
    mat_int_1.printMatrix(&mat_int_1);
    cout << endl;
    mat_int_2.printMatrix(&mat_int_2);
    cout << endl;
    mat_int_added.printMatrix(&mat_int_added);
    mat_int_added.writeMatrixToCSV(&mat_int_added,"added.csv");
```

```
}
```

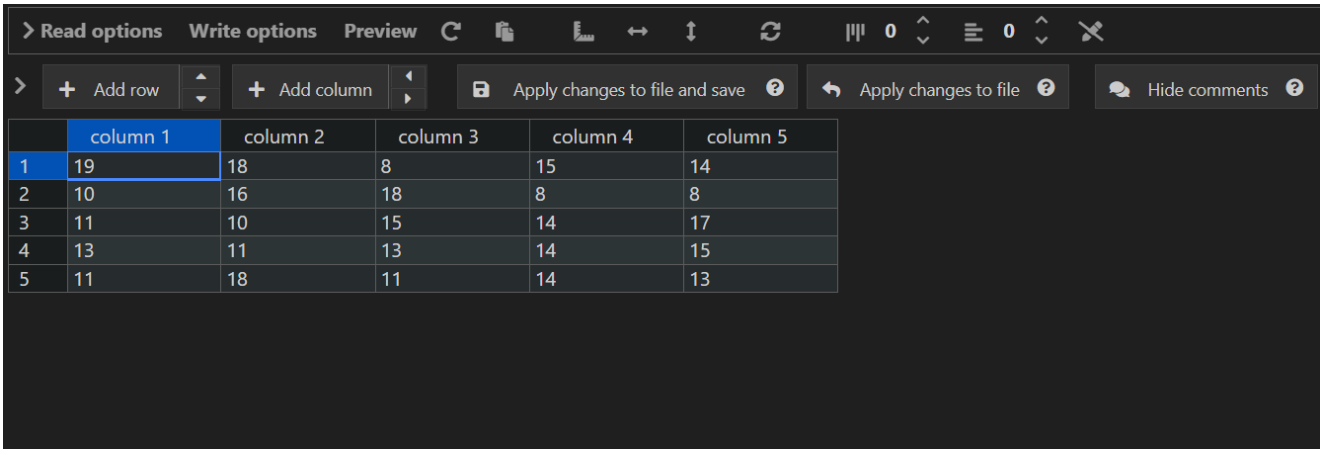
结果：

```
(base) zhang@Liyuan:~/C++/C-project4$ ./main.out
9       9       3       8       7
5       8       10      4       3
4       7       10      8       9
5       8       9       7       5
3       8       2       4       9

10      9       5       7       7
5       8       8       4       5
7       3       5       6       8
8       3       4       7       10
8       10      9       10      4

19      18      8       15      14
10      16      18      8       8
11      10      15      14      17
13      11      13      14      15
11      18      11      14      13
```

结果写入到CSV文件：



	column 1	column 2	column 3	column 4	column 5
1	19	18	8	15	14
2	10	16	18	8	8
3	11	10	15	14	17
4	13	11	13	14	15
5	11	18	11	14	13

浮点类型矩阵加法：

```
int main()
{
    Matrix<double> mat_double_1 = Matrix<double>(5,5);
    Matrix<double> mat_double_2 = Matrix<double>(5,5);
    mat_double_1.AssignRandomValue(1,10,&mat_double_1);
    mat_double_2.AssignRandomValue(1,10,&mat_double_2);
    Matrix<double> mat_double_added = mat_double_1 + mat_double_2;
    mat_double_1.printMatrix(&mat_double_1);
    cout << endl;
    mat_double_2.printMatrix(&mat_double_2);
    cout << endl;
```

```
    mat_double_added.printMatrix(&mat_double_added);
    mat_double_added.writeMatrixToCSV(&mat_double_added,"added.csv");
}
```

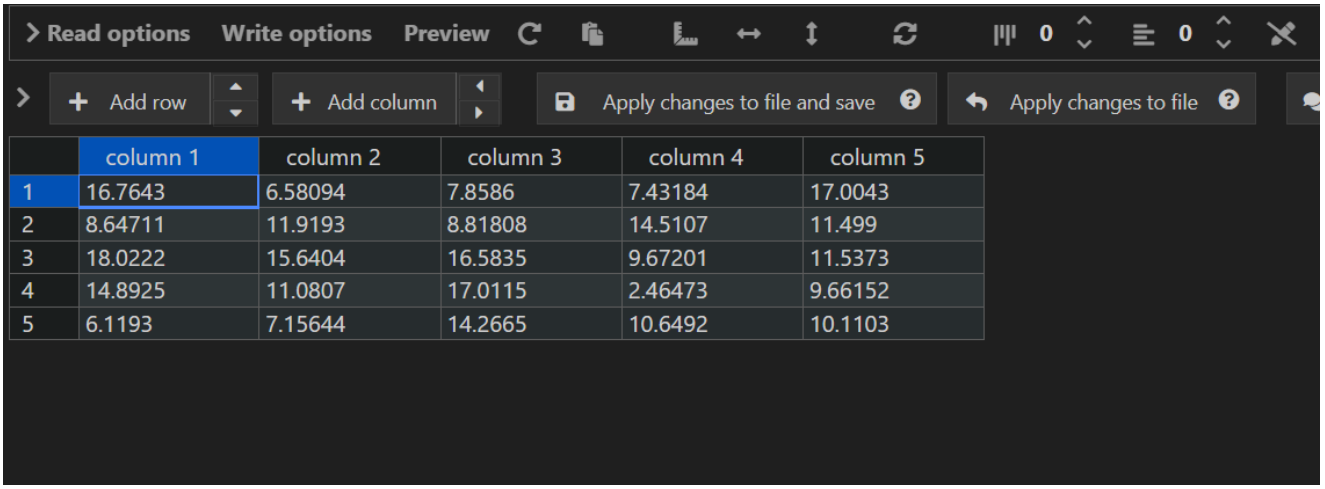
结果：

```
(base) zhang@Liyuan:~/C++/C-project4$ ./main.out
8.324049        3.294647        2.140729        3.234031        9.365502
1.992107        7.335590        4.266999        9.452527        6.628972
8.131454        6.342593        9.968690        5.769872        5.230891
9.586166        2.692569        8.009062        1.304386        4.813910
3.390519        2.448385        8.005677        7.415048        7.708005

8.440250        3.286293        5.717867        4.197813        7.638820
6.655000        4.583718        4.551084        5.058182        4.869988
9.890718        9.297770        6.614798        3.902142        6.306438
5.306324        8.388112        9.002389        1.160341        4.847607
2.728786        4.708054        6.260798        3.234153        2.402267

16.764299       6.580940        7.858597        7.431844        17.004323
8.647106        11.919309       8.818083        14.510709       11.498960
18.022172       15.640363       16.583488       9.672014        11.537330
14.892490       11.080681       17.011451       2.464727        9.661517
6.119305        7.156439        14.266475       10.649200       10.110272
```

结果写入到CSV文件中：



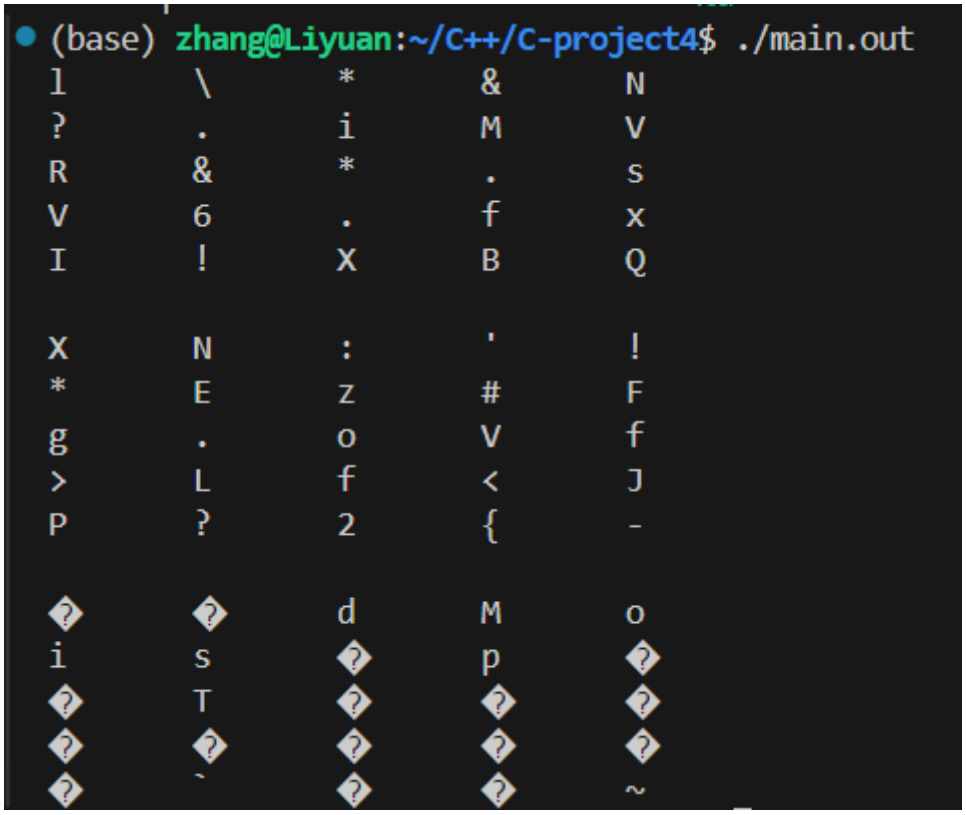
	column 1	column 2	column 3	column 4	column 5
1	16.7643	6.58094	7.8586	7.43184	17.0043
2	8.64711	11.9193	8.81808	14.5107	11.499
3	18.0222	15.6404	16.5835	9.67201	11.5373
4	14.8925	11.0807	17.0115	2.46473	9.66152
5	6.1193	7.15644	14.2665	10.6492	10.1103

unsigned char类型矩阵加法：

```
int main()
{
    Matrix<unsigned char> mat_unsigned_char_1 = Matrix<unsigned char>(5,5);
    Matrix<unsigned char> mat_unsigned_char_2 = Matrix<unsigned char>(5,5);
    mat_unsigned_char_1.AssignRandomValue(33,126,&mat_unsigned_char_1);
    mat_unsigned_char_2.AssignRandomValue(33,126,&mat_unsigned_char_2);
    Matrix<unsigned char> mat_unsigned_char_added = mat_unsigned_char_1 + mat_unsigned_char_2;
    mat_unsigned_char_1.printMatrix(&mat_unsigned_char_1);
    cout << endl;
    mat_unsigned_char_2.printMatrix(&mat_unsigned_char_2);
    cout << endl;
```

```
    mat_unsigned_char_added.printMatrix(&mat_unsigned_char_added);
    mat_unsigned_char_added.writeMatrixToCSV(&mat_unsigned_char_added,"added.csv");
}
```

结果：



由于在ASCII码表中，只有编号为33~126的字符可以被打印出来，但是它们的和很有可能超出这个范围，因而无法被打印出来，也无法写入CSV文件（除非写入的是它们的ASCII编码）。

- 乘法（只展示整型和浮点型）：

  整数类型乘法：

```
int main()
{
    Matrix<int> mat_int_1 = Matrix<int>(5,5);
    Matrix<int> mat_int_2 = Matrix<int>(5,5);
    mat_int_1.AssignRandomValue(1,10,&mat_int_1);
    mat_int_2.AssignRandomValue(1,10,&mat_int_2);
    Matrix<int> mat_int_multiplied = mat_int_1 * mat_int_2;
    mat_int_1.printMatrix(&mat_int_1);
    cout << endl;
    mat_int_2.printMatrix(&mat_int_2);
    cout << endl;
    mat_int_multiplied.printMatrix(&mat_int_multiplied);
    mat_int_multiplied.writeMatrixToCSV(&mat_int_multiplied,"multiplied.csv");
}
```

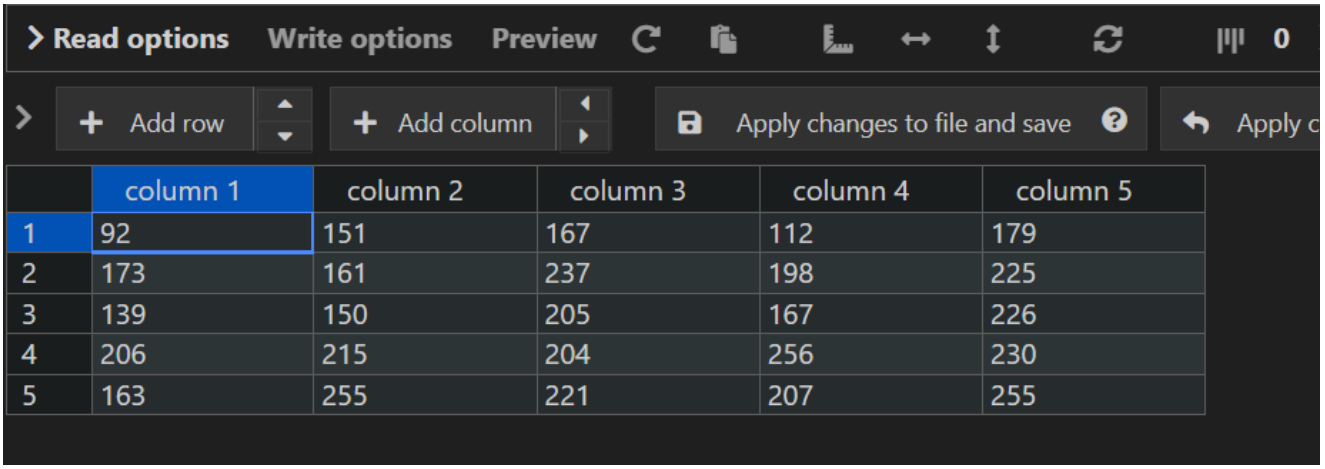
结果：

```
(base) zhang@Liyuan:~/C++/C-project4$ ./main.out
9       3       3       6       1
6       1       10      8       5
6       2       9       9       1
2       8       8       8       7
10      9       3       8       4

3       7       10      1       8
7       8       10      2       10
1       7       3       6       6
1       6       8       7       4
2       5       2       5       1

92      151     167     112     179
173     161     237     198     225
139     150     205     167     226
206     215     204     256     230
163     255     221     207     255
```

将结果写入CSV文件：



	column 1	column 2	column 3	column 4	column 5
1	92	151	167	112	179
2	173	161	237	198	225
3	139	150	205	167	226
4	206	215	204	256	230
5	163	255	221	207	255

浮点类型乘法：

```cpp
int main()
 {
   Matrix<double> mat_double_1 = Matrix<double>(5, 5);
   Matrix<double> mat_double_2 = Matrix<double>(5, 5);
   mat_double_1.AssignRandomValue(1, 10, &mat_double_1);
   mat_double_2.AssignRandomValue(1, 10, &mat_double_2);
   Matrix<double> mat_double_multiplied = mat_double_1 * mat_double_2;
   mat_double_1.printMatrix(&mat_double_1);
   cout << endl;
   mat_double_2.printMatrix(&mat_double_2);
   cout << endl;
   mat_double_multiplied.printMatrix(&mat_double_multiplied);
   mat_double_multiplied.writeMatrixToCSV(&mat_double_multiplied, "multiplied.csv");
```

```
}
```

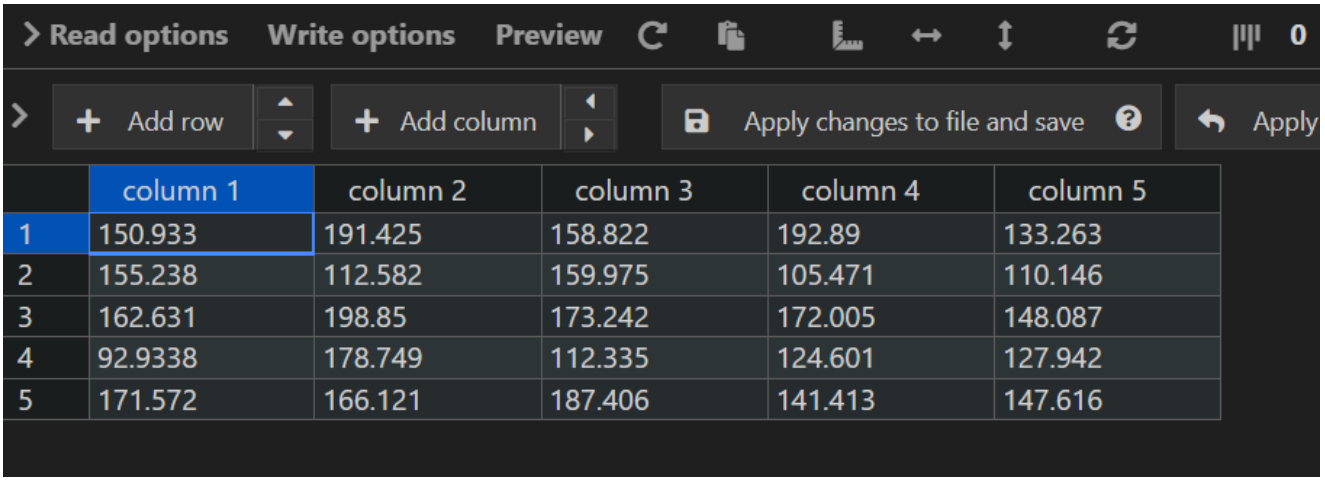
结果：

```
(base) zhang@Liyuan:~/C++/C-project4$ ./main.out
8.383569        4.639526        6.519205        8.268285        2.060343
2.176719        5.109897        1.033171        9.501694        4.968446
8.664753        5.504103        2.460147        7.690772        6.806626
5.956137        3.783635        3.553768        1.180410        9.149127
1.371354        5.354862        5.173046        9.384590        7.807207

1.733604        9.679287        1.771576        8.872922        3.193291
8.644916        5.099810        2.506100        6.025809        4.772720
4.112008        5.514078        8.116543        7.741459        3.707029
3.037049        1.683039        5.834738        6.431466        9.646368
6.504926        2.290085        2.458511        7.193449        9.311007

150.933478      191.425162      158.821979      192.890187      133.262573
155.237545      112.582367      159.974619      105.471170      110.145597
162.630538      198.849838      173.242280      172.005203      148.087139
92.933753       178.749283      112.335213      124.601105      127.941542
171.572498      166.120687      187.406035      141.413369      147.616477
```

将结果写入CSV文件中：



	column 1	column 2	column 3	column 4	column 5
1	150.933	191.425	158.822	192.89	133.263
2	155.238	112.582	159.975	105.471	110.146
3	162.631	198.85	173.242	172.005	148.087
4	92.9338	178.749	112.335	124.601	127.942
5	171.572	166.121	187.406	141.413	147.616