CS205 C/C++ Programming - Projetct05

姓名：秦世华

SID：11812309

git hub link: https://github.com/Kishore4399/C-C-Project.git

1). Design a class for matrices, and the class should contain the data of a matrix and related information such the number of rows, the number of columns, the number of channels, etc.

Part 1-Code:

struct MatrixValue:

```
template <typename T>
struct MatrixValue{
        int *refCount;
        size_t _row, _column, _size;
        T * _elem;
        MatrixValue();
        MatrixValue(size_t r, size_t c);
        MatrixValue(size_t r, size_t c, T * data);
        bool create(size_t size,  const T * data);
        bool release();
        ~MatrixValue();
};
```

class Matrix:

```
template <typename T>
class Matrix
{
private:

    MatrixValue<T> *value = new MatrixValue<T>();

public:
    // this constructor is to construct a full zero matrix with given size
    Matrix(size_t r, size_t c): value(new MatrixValue<T>(r,c)){}
    Matrix(size_t r, size_t c, T * data): value(new MatrixValue<T>(r,c, data)){}
    Matrix(const Matrix<T> &rhs);
    Matrix(){};
    ~Matrix();
    void printMatrix() const;

    T getElement(size_t r, size_t c) const;
    bool setElement(size_t r, size_t c, T inValue);
    size_t shape(int flag) const;

    Matrix<T> ROI(size_t startX, size_t startY, size_t endX, size_t endY) const;
    Matrix<T>& operator=(const Matrix<T> &B);
    Matrix<T> operator+(const Matrix<T> &B) const;
    Matrix<T> operator-(const Matrix<T> &B) const;
    Matrix<T> operator*(const T scalar) const;
    Matrix<T> operator*(const Matrix<T> &B) const;
    friend Matrix<T> matmul(const Matrix<T> &A, const Matrix<T> &B)
    {
        if (A.value->_elem == NULL || B.value->_elem == NULL)
        {
            std::cerr << "matrix data invalid" << std::endl;
            std::cerr <<"this error happened in matmul"<< std::endl;
            exit(-1);
        }

        if (A.value->_column != B.value->_row)
        {
            ERROR_SIZE_MATCH;
            std::cerr <<"this error happened in matmul"<< std::endl;
            exit(-1);
        }

        size_t i, j, k;
        Matrix<T> MAT_PRO(A.value->_row, B.value->_column);

        for(i = 0; i < A.value->_row; i++)
        {
            for(j = 0; j < B.value->_column; j++)
            {
                for(k = 0; k < A.value->_row; k++)
                {
                    MAT_PRO.value->_elem[j + i * B.value->_column] += A.value-
>_elem[(i * A.value->_column) + k] * B.value->_elem[(k * B.value->_column) + j];
                }
            }
        }

        return  MAT_PRO;
    }
    friend Matrix<T> operator*(const T scalar, const Matrix<T> &Mat){
        return Mat * scalar;
    }
}
```

以上为本次project基于C++所构建的 Matrix 类，其中包括了四个构造函数（包含copy constructor）

2). The class should support different data types. It means that the matrix elements can be unsigned char, short, int, float, double, etc.

Part 1-Code:

```
Matrix<float> fmat0(4,4);
fmat0.setElement(0,1,1.6666f);
fmat0.printMatrix();
Matrix<unsigned char> fmat1(4,4);
fmat1.setElement(0,1,'2');
fmat1.printMatrix();
Matrix<short> fmat2(4,4);
fmat2.setElement(0,1,3);
fmat2.printMatrix();
Matrix<int> fmat3(4,4);
fmat3.setElement(0,1,3);
fmat3.printMatrix();
Matrix<double> fmat4(4,4);
fmat4.setElement(0,1,1.6666);
fmat4.printMatrix();
```

Part 2- Result & Verification

方便构造了 不同元素类型的矩阵 unsigned char, short, int, float, double，其中在初始化 unsigned char 类型的的时候，由于memset 方法无法降级使用 因此添加了判断条件，并使用unsigned char a = '0'; memset(elem, a, _size * sizeof(T));来代替原有的 memset(elem, 0, _size * sizeof(T));

```
constructor used
---------------Matrix Print Start----------------
0 1.6666 0 0
0 0 0 0
0 0 0 0
0 0 0 0
---------------Matrix Print End----------------
constructor used
---------------Matrix Print Start----------------
0 2 0 0
0 0 0 0
0 0 0 0
0 0 0 0
---------------Matrix Print End----------------
constructor used
---------------Matrix Print Start----------------
0 3 0 0
0 0 0 0
0 0 0 0
0 0 0 0
---------------Matrix Print End----------------
constructor used
---------------Matrix Print Start----------------
0 3 0 0
```

```
0 0 0 0
0 0 0 0
0 0 0 0
---------------Matrix Print End----------------
constructor used
---------------Matrix Print Start----------------
0 1.6666 0 0
0 0 0 0
0 0 0 0
0 0 0 0
---------------Matrix Print End----------------
deconstructor used
deconstructor used
deconstructor used
deconstructor used
deconstructor used
```

3). Do not use memory hard copy if a matrix object is assigned to another. Please carefully handle the memory management to avoid memory leaks and to release memory multiple times.

Part 1-Code:

为了实现指针计数，将数据包装成一个结构体，里面包含矩阵的基本数据，以及用于记录引用的变量refCount

```
template <typename T>
struct MatrixValue{
        int *refCount;
        size_t _row, _column, _size;
        T * _elem;
        MatrixValue();
        MatrixValue(size_t r, size_t c);
        MatrixValue(size_t r, size_t c, T * data);
        bool create(size_t size,  const T * data);
        bool release();
        ~MatrixValue();
};
```

在初始化结构结构体 MatrixValue 时，使用三个constructor 都会讲refCount 初始化 为 1，代码如下:

```
template<typename T>
MatrixValue<T>::MatrixValue():refCount(new int(1));
MatrixValue<T>::MatrixValue(size_t r, size_t c, T * data):refCount(new int(1));
MatrixValue<T>::MatrixValue(size_t r, size_t c):refCount(new int(1))
```

在使用 copy constructor 和 operator "=" 时，Matrix对象中 变量 MatrixValue *value 所指向的结构体中的变量refCount 增加1， 即++refCount。同时在作用域结束时调用Matirx的 destructor时refCount减1 即 --refCount， 当refcount==0时，删除 value所指向内存内容。代码如下:

```
// copy constructor
```

```
template<typename T>
Matrix<T>::Matrix(const Matrix<T> &rhs){

    if(rhs.value->_elem == NULL)
        {
         std::cerr << "Matrix(const Matrix<T> & rhs) : rhs data invalid" <<
std::endl;
         exit(-1);
        }
        value -> _size = 0;
        value -> _elem = NULL;
        value->create(rhs.value->_size, rhs.value->_elem);
        this->value -> _column = rhs.value->_column;
        this->value -> _row = rhs.value->_row;
        this->value->refCount = rhs.value->refCount;

        std::cout<<"copy constructor refCount:"<< *value->refCount << std::endl;
        ++ *value->refCount;
        std::cout<<"copy constructor refCount:"<< *value->refCount << std::endl;

        std::cout << "copy constructor used" << std::endl;
}

// overloading operator =
template<typename T>
Matrix<T>& Matrix<T>::operator=(const Matrix<T> &B){
    if(this == &B)
    {
        return * this;
    }

    if (B.value->_elem == NULL)
    {
        std::cerr << "matrix data invalid" << std::endl;
        std::cerr << "This error happened operator=" << std::endl;
        exit(-1);
    }

    value->_row = B.value->_row, value->_column = B.value->_column;
    this->value->refCount = B.value->refCount;
    bool flag = value->create(B.value->_size, B.value->_elem);
    std::cout<<flag<<std::endl;
    if(flag == false)
    {
        std::cerr << "create() fail" << std::endl;
        std::cerr << "This error happened operator=" << std::endl;
        exit(-1);
    }
    ++ *value->refCount;
    return * this;
}
// Matrix destructor
template<typename T>
Matrix<T>::~Matrix(){if(--*value->refCount==0) delete value;}
```

Part 2- Result & Verification

在创建 fmat1 后， 分别使用 copy constructor 和 operator = 对 fmat2 和 fmat3 赋值，在一个作用域结束之后，调用Matrix fmat2的destructor, 输出表明 refCount成功减一，同时依旧可以打印出 fmat1 中的数据，说明fmat1中指向的结构体地址内容并没有随之销毁

```
float *data = new float[8];
memset(data, 0, 8*sizeof(float));
Matrix<float> fmat1(2, 4, data);
fmat1.setElement(0,1,1.6f);
fmat1.printMatrix();
// use copy constructor
{Matrix<float> fmat2 = fmat1;
fmat2.printMatrix();}
fmat1.printMatrix();
// use operator =
Matrix<float> fmat3;
fmat3 = fmat1;
fmat3.printMatrix();
```

```
constructor with data used
---------------Matrix Print Start----------------
0 1.6 0 0
0 0 0 0
refcount = 1
---------------Matrix Print End----------------
copy constructor used
---------------Matrix Print Start----------------
0 1.6 0 0
0 0 0 0
refcount = 2
---------------Matrix Print End----------------
---------------Matrix Print Start----------------
0 1.6 0 0
0 0 0 0
refcount = 1
---------------Matrix Print End----------------
1
---------------Matrix Print Start----------------
0 1.6 0 0
0 0 0 0
refcount = 2
---------------Matrix Print End----------------
```

4).Implement some frequently used operators including but not limited to =, ==, +, -, *, etc. Surely the matrix multiplication in Project 4 can be included.

Part 1-Code:

```
float *data = new float[8];
memset(data, 0, 8*sizeof(float));
Matrix<float> fmat1(2, 4, data);
fmat1.setElement(0,1,1.6f);
fmat1.printMatrix();

// use copy constructor
{Matrix<float> fmat2 = fmat1;
fmat2.printMatrix();}
fmat1.printMatrix();
// use operator =
Matrix<float> fmat3;
fmat3 = fmat1;
fmat3.printMatrix();

Matrix<float> fmat4(2, 4, data);
fmat4.setElement(1,1,1.6f);
fmat4.printMatrix();

Matrix<float> fmat5 = fmat1 + fmat4;
fmat5.printMatrix();

Matrix<float> fmat6 = fmat1 - fmat4;
fmat6.printMatrix();

Matrix<float> fmat7 = fmat1 * 3;
fmat7.printMatrix();

Matrix<float> fmat8(4, 2, data);
fmat8.setElement(1,1,1.6f);
fmat8.printMatrix();
Matrix<float> fmat9 = fmat1 * fmat8;
fmat9.printMatrix();

Matrix<float> fmat10 = 3 * fmat1;
fmat10.printMatrix();

cout<<"The length of row of the fmat10: "<<fmat10.shape(0)<<endl;
cout<<"The length of column of the fmat10: "<<fmat10.shape(1)<<endl;
```

Part 2- Result & Verification

```
constructor with data used
---------------Matrix Print Start----------------
0 1.6 0 0
0 0 0 0
---------------Matrix Print End----------------
copy constructor used
---------------Matrix Print Start----------------
0 1.6 0 0
0 0 0 0
---------------Matrix Print End----------------
```

```
---------------Matrix Print Start----------------
0 1.6 0 0
0 0 0 0
---------------Matrix Print End----------------
---------------Matrix Print Start----------------
0 1.6 0 0
0 0 0 0
---------------Matrix Print End----------------
constructor with data used
---------------Matrix Print Start----------------
0 0 0 0
0 1.6 0 0
---------------Matrix Print End----------------
constructor used
---------------Matrix Print Start----------------
0 1.6 0 0
0 1.6 0 0
---------------Matrix Print End----------------
constructor used
---------------Matrix Print Start----------------
0 1.6 0 0
0 -1.6 0 0
---------------Matrix Print End----------------
constructor used
---------------Matrix Print Start----------------
0 4.8 0 0
0 0 0 0
---------------Matrix Print End----------------
constructor with data used
---------------Matrix Print Start----------------
0 0
0 1.6
0 0
0 0
---------------Matrix Print End----------------
constructor used
---------------Matrix Print Start----------------
0 2.56
0 0
---------------Matrix Print End----------------
constructor used
---------------Matrix Print Start----------------
0 4.8 0 0
0 0 0 0
---------------Matrix Print End----------------
The length of row of the fmat10: 2
The length of column of the fmat10: 4
```

5). Implement region of interest (ROI) to avoid memory hard copy.

为了实现region of interest, 将原有的 int refCount 改成了 int *refCount, 这样可以方便每一个应用的对象可以用同一个refCount 的地址记录。

Part 1-Code:

```
template<typename T>
Matrix<T> Matrix<T>::ROI(size_t startX, size_t startY, size_t endX, size_t endY)
const
{
    if(value->_elem == NULL)
    {
        std::cerr << "Matrix<T>& ROI(const Matrix<T> BigMatrix, size_t startX,
size_t startY, size_t endX, size_t end) : rhs data invalid" << std::endl;
        exit(-1);
    }
    if(startX < 0 || startY < 0 || endX < 0 || endY < 0 || startX > value-
>_column || endX > value->_column || startY > value->_row || endY > value->_row)
    {
        std::cerr<< "ROI: input coord out of range" << std::endl;
        exit(-1);
    }
    size_t row = endX - startX + 1;
    size_t column = endY - startY + 1;
    size_t size = row * column;
    Matrix<T> ROIMat(row, column);

    size_t elem_count = 0;
    for(size_t i=0; i < row; i++){
        for(size_t j=0; j < column; j++){
            ROIMat.value->_elem[elem_count] = value->_elem[(startX + i) * value-
>_column + startY + j];
            ++elem_count;
        }
    }
    ROIMat.value->refCount = this->value->refCount;
    ++ *value->refCount;
    return ROIMat;

}
```

Part 2- Result & Verification

```
float *data = new float[20];
    memset(data, 0, 20*sizeof(float));
    Matrix<float> fmat1(5, 4, data);
    fmat1.setElement(0,1,1.6f);
    fmat1.setElement(1,0,1.5f);
    fmat1.printMatrix();

    {Matrix<float> fmat5 = fmat1.ROI(1,0,4,2);
    fmat5.printMatrix();}

// use copy constructor
    {Matrix<float> fmat2 = fmat1;
    fmat2.printMatrix();
    fmat1.printMatrix();}
```

```
// use operator =
    Matrix<float> fmat3;
    fmat3 = fmat1;
    fmat3.printMatrix();
```

分别检验 copy constructor , operator "=" 和 ROI在调用是指针应用情况，我们将 ROI 和 copy constructor 的部分代码使用{}分别装入到作用域中，发现调用 copy constructor , operator "=" 和 ROI 是指针的refCount 都会加一， 并且作用域解释之后，三个对象共享的数据_elem 仍然留存了下来，直到 --refCount为 0 时，delete _elem 才会成功。

```
constructor with data used
---------------Matrix Print Start----------------
0 1.6 0 0
1.5 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
refcount = 1
---------------Matrix Print End----------------
constructor used
---------------Matrix Print Start----------------
1.5 0 0
0 0 0
0 0 0
0 0 0
refcount = 2
---------------Matrix Print End----------------
copy constructor refCount:1
copy constructor refCount:2
copy constructor used
---------------Matrix Print Start----------------
0 1.6 0 0
1.5 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
refcount = 2
---------------Matrix Print End----------------
---------------Matrix Print Start----------------
0 1.6 0 0
1.5 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
refcount = 2
---------------Matrix Print End----------------
1
---------------Matrix Print Start----------------
0 1.6 0 0
1.5 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
refcount = 2
---------------Matrix Print End----------------
```