CS205 C/ C++ Programming – Project 5

Name: 凌硕（Ling Shuo）
SID：11912409

Contents

Part 1 – Analysis

In this project, we are required to create a class of matrix and achieve some simple functions in C++. To achieve the fifth requirement, we created a derived class of the matrix. The base class is called `class Matrix`, and the derived class is called `class ROI_of_Matrix`.

1 – Class for matrices

Consider the class of a matrix, I should put the numbers of rows, columns and channels in it. They should be nonnegative integers so I used `size_t` type. And I used a pointer `elements` to describe the address of the elements of a matrix. One important thing is that I used **class templates** to allow the elements of the matrix to be of different types. Another important member `size_t * count` of this class is to describe how many matrix variables **shared the same pointer** `elements` by simultaneously.
Above all, the class has members `size_t rows; size_t columns; size_t channels; size_t * count; T *elements`.
Remark 1 : It is worth mentioning that we set the members to be protected which make it impossible for users to modify them as they like.
Please see the code in Part 2. And this part of the code is written in file ***matrix.hpp***.

2 – Constructors

In this part, I created four constructors for the class.

1) The first one is with no parameters, I set the rows, columns and channels of the matrix to be zero. And the two pointers will be set to `NULL`.
2) The second one is with two parameters `size_t rows, size_t columns`. The rows and columns of the matrix will be set with these two arguments if they are positive and the channel will be set to one as default. The pointer `count` will point to one and the pointer `elements` will be initialized with `this->elements = new T[rows * columns]()`.
3) The third one is with four parameters `size_t rows, size_t columns, size_t channels`. Users can specify the number of channels of the matrix through the third parameter (if it is legal), the rest of the logic is the same as the second constructor.
4) The fourth one is with one parameter `const Matrix<T> & m`. We can create a new matrix according to a given matrix `m`. The assignment on the number of rows, columns and channels is trivial. But we need to keep the two pointers of the old and new matrices **the same** and **add one** to the number of the pointer `count` point to (if `count` is not a empty pointer). In this way, we implemented the matrix assignment under the requirement 3 which require us to **avoid memory hard copy**.

Please see the code in Part 2. And this part of the code is written in file ***matrix.hpp***.

3 – Destructor

To create a destructor consistent with our class, we subtract one from the number pointed by

pointer `count` if it used to be greater than 1 and do nothing to the pointer `elements`. This is because there are other matrices who share the same `count` and `elements`. If the number pointed by pointer `count` was one, we will delete the pointer `count` and `elements`.

Remark 2 : We will check to see if the pointers (`count` and `elements`) are empty before I do above operations.

Please see the code in Part 2. And this part of the code is written in file ***matrix.hpp***.

4 – Some basic functions

In this part, I create some basic functions for users to change some information and get some information about the matrix they have created, since the member variables are protected.

1) `size_t getCount()`: trivial.
2) `size_t getRows()`: trivial.
3) `size_t getColumns()`: trivial.
4) `size_t getChannels()`: trivial.
5) `void setSize(size_t a, size_t b, size_t c)`: if the matrix is created by the first constructor (i.e. the matrix has rows, columns and channels with all zero and the two pointers are `NULL`), users can set the size as they like and the two pointers will be initialized as the same way in constructors.
6) `void setData(T * array, size_t len_of_array)`: set the entries of the matrix by the given array. And it is worth noting that we should check that the length of the array (given by parameter `size_t len_of_array`) is consistent with the size of the matrix.
7) `void change_one_entry(size_t a, size_t b, size_t c, T value)`: We provide the user with a function that modifies matrix elements. It is important that the original pointer `count` should be subtracted by one (if it was larger than one; otherwise, delete it and pointer `elements` and apply new ones). And we apply new `count` and `elements` to store the new data.
8) `void random_generation(T min, T max)`: set the elements randomly from the interval given by (if the size of the matrix is not zero).

Remark 3 : The creation of this part of the functions is to make it more convenient and safe for users to use the matrix class, and provide convenience for users to change the matrix value while managing the memory safely.

Please see the code in Part 2. And this part of the code is written in file ***matrix.hpp***.

5 – Overloading of some operators

In this part, we Implement some frequently used operators. And the **approach in project 4** on matrix multiplication are repeated here.

1) `friend std::ostream & operator<<(std::ostream & os, const Matrix & m)`: Overloading the output function, based on whether the matrix is empty to output.
2) `friend std::istream & operator>>(std::istream & is, Matrix & m)`: Overloading the input function which is useful to matrix defined by constructor 1.

3) `Matrix & operator=(const Matrix & m)`: Overloading the assignment operator. The approach is the same as for the fourth constructor.
4) `Matrix operator+(const Matrix & m) const`: Overloading the addition operator. The method is simple, but the implementation needs to check whether the two matrices are empty (rows, columns and channels are zeros and two pointers are `NULL`) and whether the sizes of the two matrices are insistent.
5) `Matrix operator-(const Matrix & m) const`: similar to 4).
6) `Matrix operator*(const Matrix & m) const`: Overloading the multiplication operator. The method is simple if we note that we just need to do the normal matrix multiplication **in each channel**, but the implementation needs to check whether the two matrices are empty (rows, columns and channels are zeros and two pointers are `NULL`) and whether the sizes of the two matrices are insistent. It is worth mentioning that we rewrote the method for some **specialized** classes (i.e. T = float, etc.) to achieve the effect in Project 4 (using **SIMD and OpenMP** to speed up, and the method of speeding up is also used in other operators).
7) `bool operator==(const Matrix & m) const`: Overloading the '==' operator. The results are obtained by comparing the size of each dimension and the value of each element of the two matrices. Be careful to check that the matrix is empty beforehand.

Remark 4 : We have also overloaded the **matrix and constant** operators (as an example: `Matrix operator+(T t) const`) and **constant and matrix** operators (as an example: `friend Matrix operator+(T t, const Matrix & m)`) for user convenience. Note that constant and matrix operators need us to write **friend functions** which are not member functions.

Please see the code in Part 2. And this part of the code is written in file ***matrix.hpp***.

6 – Find the inverse (if it exists)

Finding the inverse of a matrix is one of the classical problems of matrix operations. We implement it by three functions.

1) `template<typename T>`
   `T det_for_one_channel(T * array, size_t rows)` : (not a member function)
   In this part, we are trying to find the determinant of a square matrix with channel just one. We achieve the function by recursion. For matrix $A = (a_{i,j})_{i,j=1,\dots n}$, $\det(A) = \sum_{i=1}^{n}(-1)^{1+i}\det(A_{1,i})$, where $A_{j,i}$ is the matrix given by removing the $j-th$ row and the $i-th$ column from A. Since the size of $A_{1,i}$ is small than A, the recursion makes sense.
2) `template<typename T>`
   `double * inverse_for_one_channel(T * array, size_t rows)`: (not a member function)
   In this part, we are trying to find the inverse of a square matrix with channel just one.
   For square matrix $A = (a_{i,j})_{i,j=1,\dots n}$, if $\det(A) = 0$, A has no inverse. Else, $A^{-1} = \frac{A^*}{\det(A)}$, where $A^* = (a^*_{i,j})_{i,j=1,\dots n}$ and $a^*_{i,j} = (-1)^{i+j}\det(A_{j,i})$ which $A_{j,i}$ is defined in Part1-6.1). If the matrix is not invertible, output `NULL`.

3) `Matrix<double> inverse() const`:
   We can easily get the channel-wise inverse of the matrix by the function in 2). Note that the output is in type `Matrix<double>`. The only thing need to notice is that if the result of function 2) is `NULL` for a matrix in one of the channels, we set the result matrix to be the zero matrix for that channel. When the user sees the zero matrix, he knows that the original matrix in the corresponding channel is not invertible (this does not cause confusion, because the zero matrix cannot be regarded as the inverse of any invertible matrix).

Please see the code in Part 2. And this part of the code is written in file ***matrix.hpp***.

7 – Class of ROI

Note that the users may be interested in certain region (we call it ROI, region of interest) of the matrix in addition to the matrix itself. So we create a subclass of class matrix, that is, `class ROI_of_Matrix : public Matrix<T>`. In this way, you can **avoid memory hard copy** if you consider the ROI of a matrix. The extra member variables are as following:

```
private:
  size_t BeginRowIndex;
  size_t BeginColumnIndex;
  size_t EndRowIndex;
  size_t EndColumnIndex;
```

If you have ROI of a matrix, you can use this subclass and we provide some functions for the derived class.

1) `ROI_of_Matrix()`: Constructor one, invoke the constructor one of class Matrix and set the four derived member variables to be zero.
2) `ROI_of_Matrix(size_t rows, size_t columns)`: Constructor two, invoke the constructor two of class Matrix and set the four derived member variables to be zero.
3) `ROI_of_Matrix(size_t rows, size_t columns, size_t channels)`: Constructor three, invoke the constructor two of class Matrix (since the class 'ROI_of_Matrix' is only open for matrices with channel one) and set the four derived member variables to be zero.
4) `ROI_of_Matrix(size_t rows, size_t columns, size_t b_r_i, size_t b_c_i, size_t e_r_i, size_t e_c_i)`: Constructor four, invoke the constructor two of class Matrix and set the four derived member variables according to the parameters (if they are legal).
5) `ROI_of_Matrix(size_t rows, size_t columns, size_t channels, size_t b_r_i, size_t b_c_i, size_t e_r_i, size_t e_c_i)`: Constructor five, invoke the constructor two of class Matrix (since the class 'ROI_of_Matrix' is only open for matrices with channel one) and set the four derived member variables according to the parameters (if they are legal).
6) `ROI_of_Matrix(const ROI_of_Matrix<T> & m)`: Constructor six, invoke the copy constructor of class Matrix and set the four derived member variables according to the parameter ROI_Matrix m.
7) `~ROI_of_Matrix()`: Destructor, invoke the destructor in the base class.

8) `void showROI()`: Output the matrix and the ROI of it. Note that we did not overload operator '<<' in this derived class since we want to have two tools for this derived class to output.
9) `Matrix<T> turn_to_matrix()`: If you want to do something to the ROI of a matrix rather than the whole matrix, please turn it to a new matrix by this function.
10) `void turned_from_matrix(Matrix<T> & m,size_t b_r_i, size_t b_c_i, size_t e_r_i, size_t e_c_i)`: This is an important function which transform a matrix with some arguments to a ROI_Matrix. Note that in this function, we assign the address of elements of a matrix to the ROI_Matrix and add one to the number pointed by pointer count. In this way, we can get a new ROI_Matrix that can record ROI from a matrix **without hard copy** (Whether the arguments are legal or not has been taken into account and will not be described here).

Please see the code in Part 2. And this part of the code is written in file ***matrix.hpp***.

8 – Testing for class Matrix

We tested the `class Matrix` in this part, see Part 3 for details. We can compile it by

```
g++ -fopenmp -mavx2 test1.cpp -o test1
```

This part of the code is written in file ***test1.cpp***.

9 – Testing for class ROI_of_Matrix

We tested the `class ROI_of_Matrix`, see Part 3 for details. We can compile it by

```
g++ -fopenmp -mavx2 test2.cpp -o test2
```

This part of the code is written in file ***test2.cpp***.

Part 2 – Code

In this part, I put some important code in ***matrix.hpp***.

```
#ifndef _MATRIX_HPP
#define _MATRIX_HPP

template<typename T>  //T could be unsigned char, short, int, float, double
class Matrix
{
  protected:
    size_t rows;
    size_t columns;
    size_t channels;
    size_t * count;  //denote the numbers of matrices whose elements are the same;
    T *elements;  //store the data; malloc with elements, free with elements at the same time

  public:
```

```
Matrix()  //constructor
{
    this->rows = 0;
    this->columns = 0;
    this->channels = 0;
    this->count = NULL;
    this->elements = NULL;
}

Matrix(size_t rows, size_t columns, size_t channels)  //constructor
{
    if(rows != 0 and columns != 0 and channels != 0)
    {
        this->rows = rows;
        this->columns = columns;
        this->channels = channels;
        this->count = new size_t(1);
        this->elements = new T[rows * columns * channels]();
    }
    else
    {
        this->rows = 0;
        this->columns = 0;
        this->channels = 0;
        this->count = NULL;
        this->elements = NULL;
    }
}

Matrix(const Matrix<T> & m)  //When using Matrix b = a, this constructor will be invoked
{
    if(count == NULL and elements == NULL)
    {
        this->rows = m.rows;
        this->columns = m.columns;
        this->channels = m.channels;
        if(m.count != NULL and m.elements != NULL)
        {
            this->count = m.count;
            (*count) ++;
            this->elements = m.elements;
        }
        else
        {
```

```
            this->count = NULL;
            this->elements = NULL;
        }
    }
    else if(count == m.count || elements == m.elements)
    {
        cout << "Error of assignment since you can't assign a matrix to itself!" << endl;
    }
    else if(count != NULL and elements != NULL)
    {
        if(*(this->count)==1)
        {
            delete []this->count;
            delete []this->elements;
            count = NULL;
            elements = NULL;
        }
        else //*(this->count)>1
        {
            (*count) --;
        }
        rows = m.rows;
        columns = m.columns;
        channels = m.channels;
        count = m.count;
        if(count != NULL)
        {
            (*count) ++;
        }
        elements = m.elements;
    }
    else
    {
        cout << "Error of assignment!" <<endl;
    }
}

~Matrix()  //destructor
{
    if(count == NULL or elements == NULL)
    {
        if(count != NULL)
        {
            delete []count;
```

```
                count = NULL;
            }
            if(elements != NULL)
            {
                delete []elements;
                elements = NULL;
            }
        }
        else if(*(this->count) == 1)
        {
            delete []count;
            delete []elements;
            count = NULL;
            elements = NULL;
        }
        else  //*(this->count) > 1
        {
            *(this->count) -= 1;
        }
    }

    void setSize(size_t a, size_t b, size_t c)
    {
        if(rows == 0 and columns ==0 and channels ==0 and count == NULL and elements == NULL)
        {
            rows = a;
            columns = b;
            channels = c;
            count = new size_t(1);
            elements = new T[rows * columns * channels]();
        }
        else
        {
            cout<<"The size of the matrix is defined (or the matrix is invalid), so you cannot set
its size!"<<endl;
        }
    }
    void change_one_entry(size_t a, size_t b, size_t c, T value)  //change one entry in place
(a,b,c) to T
    {   //note that a is from zero to rows-1, and b, c are similar
        if(count == NULL || elements == NULL)
        {
            cout<<"Changing failed since the matrix is not valid!"<<endl;
            return;
```

```
        }
        else if(rows < a + 1 || columns < b + 1 || channels < c + 1)
        {
            cout<<"Changing failed since the input position is wrong!"<<endl;
            return;
        }
        else
        {
            if(*count == 1)
            {
                elements[c * rows * columns + a * columns + b] = value;
            }
            else
            {
                size_t * count_temp = new size_t(1);
                T * elements_temp = new T[rows * columns * channels]();
                for(size_t i = 0 ; i < rows ; i++)
                {
                    for(size_t j = 0 ; j < columns ; j++)
                    {
                        for(size_t k = 0 ; k < channels ; k++)
                        {
                            elements_temp[k * rows * columns + i * columns + j] = elements[k * rows *
columns + i * columns + j];
                        }
                    }
                }
                elements_temp[c * rows * columns + a * columns + b] = value;
                (*count) -= 1;
                count = count_temp;
                elements = elements_temp;
            }
        }
    }

    friend std::ostream & operator<<(std::ostream & os, const Matrix & m)  //friend function is not
a member function
    {
        std::string str = "";
        if(m.rows == 0 || m.columns == 0 || m.channels == 0 || m.count == NULL || m.elements == NULL)
        {
            str = "This is an invalid matrix!\n";
        }
        else
```

```
        {
            str = "This is an matrix with rows "+ std::to_string(m.rows) + ", columns "\
            + std::to_string(m.columns) + " and channel " + std::to_string(m.channels)+" .\n";
            for(size_t k = 0 ; k < m.channels ; k++)
            {
                str = str + "The channel " + std::to_string(k) + " is:\n";
                for(size_t i = 0 ; i < m.rows ; i++)
                {
                    for(size_t j = 0 ; j < m.columns ; j++)
                    {
                        str = str + std::to_string(m.elements[k * m.rows * m.columns + i * m.columns
+ j]) + "  ";
                    }
                    str = str + ";\n";
                }
            }
        }
        os<<str;
        return os;
    }

    friend std::istream & operator>>(std::istream & is, Matrix & m)
    {
        if(m.rows != 0 and m.columns != 0 and m.channels != 0)
        {
            cout<<"For this matrix which has been defined, you cannot input the data! You can change
the elements by function change_one_entry."<<endl;
        }

        else if(m.rows == 0 and m.columns == 0 and m.channels == 0 and m.count == NULL and m.elements
== NULL)
        {
            cout<<"Please enter the numbers of rows, columns and channels of the matrix,
respectively"<<endl;
            size_t a,b,c;
            cin>>a>>b>>c;
            if((a <= 0) || (b <= 0) || (c <= 0))
            {
                cout<<"Illegal input!"<<endl;
                goto flag;
            }
            else
            {
                m.setSize(a, b, c);
```

```
                for(size_t k = 0 ; k < m.channels ; k++)
                {
                    cout<<"Pleasr enter the matrix in channel "<<std::to_string(k);
                    cout<<". To enter a "<< std::to_string(m.rows)<<" by "<<
std::to_string(m.columns);
                    cout<<" matrix, please follow the order of row major."<<endl;
                    for(size_t i = 0 ; i < m.rows ; i++)
                    {
                        for(size_t j = 0 ; j < m.columns ; j++)
                        {
                            is >> m.elements [k * m.rows * m.columns + i * m.columns + j];
                        }
                    }
                }
            }

        }

        else
        {
            cout<<"There is something wrong about the matrix which has been defined before!"<<endl;
        }

        flag:
        return is;
    }

    Matrix & operator=(const Matrix & m)  //When using b = a, this constructor will be invoked
(different from the copy constructor)
    {   //Similar with copy constructor, and I omit it here.
        return *this;
    }

    Matrix operator*(const Matrix & m) const
    {
        if(count == NULL || elements == NULL || m.count == NULL || m.elements == NULL)
        {
            cout<<"Failure of multiplication!"<<endl;
            Matrix<T> result;
            return result;
        }
        else if((columns == m.rows) and (channels == m.channels) and (rows * columns * channels !=
0))
        {
```

```
            Matrix<T> result(rows, m.columns, channels);
            #pragma omp parallel for
            for(size_t k = 0 ; k < channels ; k++)
            {

                for(size_t i = 0 ; i < result.rows ; i++)
                {
                    for(size_t j = 0 ; j < result.columns ; j++)
                    {
                        for(size_t l = 0 ; l < columns ; l++)
                        {
                            result.elements[k * result.rows * result.columns + i * result.columns +
j] += \
                            elements[k * rows * columns + i * columns + l] * \
                            m.elements[k * m.rows * m.columns + l * m.columns + j];
                        }
                    }
                }

            }
            return result;
        }
        else
        {
            cout<<"Failure of multiplication!"<<endl;
            Matrix<T> result;
            return result;
        }
    }

    Matrix operator*(T t) const
    {
        if(count == NULL || elements == NULL)
        {
            cout<<"Failure of multiplication!"<<endl;
            Matrix<T> result;
            return result;
        }
        else if(rows * columns * channels != 0)
        {
            Matrix<T> result(rows, columns, channels);
            #pragma omp parallel for
            for(size_t k = 0 ; k < channels ; k++)
            {
```

```
                for(size_t i = 0 ; i < result.rows ; i++)
                {
                    for(size_t j = 0 ; j < result.columns ; j++)
                    {
                        result.elements[k * result.rows * result.columns + i * result.columns + j] =
\
                        elements[k * rows * columns + i * columns + j] * t;

                    }
                }

            }
            return result;
        }
        else
        {
            cout<<"Failure of multiplication!"<<endl;
            Matrix<T> result;
            return result;
        }
    }

    friend Matrix operator*(T t, const Matrix & m)
    {
        return m * t;
    }

    Matrix<double> inverse()  const  //Note: the output is a matrix of double
    {
        if(rows * columns * channels != 0 and rows == columns and count != NULL and elements != NULL)
        {
            Matrix<double> result(rows, columns, channels);
            for(size_t k = 0; k < channels; k++)
            {
                T * temp = new T[rows * columns]();
                for(size_t i = 0; i < rows; i++)
                {
                    for(size_t j = 0; j < columns; j++)
                    {
                        temp[i * columns + j] = elements[k * rows * columns + i * columns + j];
                    }
                }
                double * result_k = new double[rows * columns]();
```

```
                result_k = inverse_for_one_channel<T>(temp, rows);
                if(result_k != NULL)
                {
                    for(size_t i = 0; i < rows; i++)
                    {
                        for(size_t j = 0; j < columns; j++)
                        {
                            result.change_one_entry(i, j, k, result_k[i * columns + j]);
                        }
                    }
                }
                else
                {
                    for(size_t i = 0; i < rows; i++)
                    {
                        for(size_t j = 0; j < columns; j++)
                        {
                            result.change_one_entry(i, j, k, 0);
                        }
                    }
                }
                delete []result_k;
                delete []temp;
            }
            return result;
        }
        else
        {
            Matrix<double> result;
            return result;
        }
    }


};



template<typename T>
class ROI_of_Matrix : public Matrix<T>
{
  private:
    size_t BeginRowIndex;
    size_t BeginColumnIndex;
    size_t EndRowIndex;
```

```
    size_t EndColumnIndex;

  public:
    ROI_of_Matrix(): Matrix<T>()
    {
        BeginRowIndex = 0;
        BeginColumnIndex = 0;
        EndRowIndex = 0;
        EndColumnIndex = 0;
    }

    ROI_of_Matrix(size_t rows, size_t columns, size_t channels): Matrix<T>(rows, columns)
    {
        cout<<"The class 'ROI_of_Matrix' is only open for matrices with channel 1!"<<endl;
        BeginRowIndex = 0;
        BeginColumnIndex = 0;
        EndRowIndex = 0;
        EndColumnIndex = 0;
    }

    ROI_of_Matrix(size_t rows, size_t columns, size_t channels, size_t b_r_i, size_t b_c_i, size_t
e_r_i, size_t e_c_i): Matrix<T>(rows, columns, 1)
    {
        cout<<"The class 'ROI_of_Matrix' is only open for matrices with channel 1!"<<endl;
        if(b_r_i <= rows - 1 and b_c_i <= columns - 1 and e_r_i <= rows - 1 and e_c_i <= columns - 1\
        and b_r_i <= e_r_i and b_c_i <= e_c_i\
        and ((b_r_i - e_r_i) != 0 or (b_c_i - e_c_i) != 0) )
        {
            BeginRowIndex = b_r_i;
            BeginColumnIndex = b_c_i;
            EndRowIndex = e_r_i;
            EndColumnIndex = e_c_i;
        }
        else
        {
            cout<<"Failed to define ROI as you like since the arguments are inconsistent or
incorrect!"<<endl;
            BeginRowIndex = 0;
            BeginColumnIndex = 0;
            EndRowIndex = 0;
            EndColumnIndex = 0;
        }
    }
```

```
    ROI_of_Matrix(const ROI_of_Matrix<T> & m): Matrix<T>(m)
    {
        BeginRowIndex = m.BeginRowIndex;
        BeginColumnIndex = m.BeginColumnIndex;
        EndRowIndex = m.EndRowIndex;
        EndColumnIndex = m.EndColumnIndex;
    }

    ~ROI_of_Matrix(){}

    void showROI()  //不对子类做<<的重载是考虑到对子类也想保留输出整个矩阵（而不仅仅是 ROI）的函数
    {
        if(this->rows == 0 || this->columns == 0 || this->channels == 0 || this->count == NULL ||
this->elements == NULL)
        {
            cout<<"This is an invalid matrix, and thus invalid ROI!"<<endl;
        }
        else if(BeginRowIndex == 0 and BeginColumnIndex == 0 and EndRowIndex == 0 and EndColumnIndex
== 0)
        {
            cout<<"The ROI of the matrix is equal to itself!"<<endl;
            cout<<(*this)<<endl;
        }
        else
        {
            cout<<"This is an matrix with rows "<<std::to_string(this->rows)\
            <<" and columns "<<std::to_string(this->columns)<<"."<<endl;
            //cout<<(*this)<<endl;
            for(size_t i = 0 ; i < this->rows ; i++)
            {
                for(size_t j = 0 ; j < this->columns ; j++)
                {
                    cout<<this->elements[i * this->columns + j]<<"  ";
                }
                cout<<endl;
            }
            cout<<"And the ROI is form row "<<std::to_string(BeginRowIndex)<<", column "\
            <<std::to_string(BeginColumnIndex)<<" to row "<<std::to_string(EndRowIndex)<<", column "\
            <<std::to_string(EndColumnIndex)<<endl;
            for(size_t i = 0 ; i < this->rows ; i++)
            {
                for(size_t j = 0 ; j < this->columns ; j++)
                {
```

```
                    if(i >= BeginRowIndex and i <= EndRowIndex and j >= BeginColumnIndex and j <=
EndColumnIndex)
                    {
                        cout<<std::to_string(this->elements[i * this->columns + j])<<"  ";
                    }
                }
                if(i >= BeginRowIndex and i <= EndRowIndex)
                {
                    cout<<";"<<endl;
                }
            }
        }
    }

    Matrix<T> turn_to_matrix()  //if you want to do something to the ROI, please turn it to a new
matrix
    {
        if(this->rows == 0 || this->columns == 0 || this->channels == 0 \
        || this->count == NULL || this->elements == NULL)  //case1
        {
            cout<<"The matrix in class 'ROI_of_Matirx' is an invalid matrix, so the output matirx is
also an invalid matrix!"<<endl;
            Matrix<T> result;
            return result;
        }
        else if(BeginRowIndex == 0 and BeginColumnIndex == 0 and EndRowIndex == 0 and EndColumnIndex
== 0)  //case2
        {
            Matrix<T> result(this->rows, this->columns);
            for(size_t i = 0 ; i < this->rows ; i++)
            {
                for(size_t j = 0 ; j < this->columns ; j++)
                {
                    result.change_one_entry(i, j, 0, this->elements[i * this->columns + j]);
                }
            }
            return result;
        }
        else  //case3
        {
            Matrix<T> result(EndRowIndex - BeginRowIndex + 1, EndColumnIndex - BeginColumnIndex + 1);
            T * value = new T[(EndRowIndex - BeginRowIndex + 1) * (EndColumnIndex - BeginColumnIndex
+ 1)];
            size_t flag = 0;
```

```
            for(size_t i = 0 ; i < this->rows ; i++)
            {
                for(size_t j = 0 ; j < this->columns ; j++)
                {
                    if(i >= BeginRowIndex and i <= EndRowIndex and j >= BeginColumnIndex and j <=
EndColumnIndex)
                    {
                        value[flag] = this->elements[i * this->columns + j];
                        flag ++;
                    }
                }
            }
            result.setData(value, (EndRowIndex - BeginRowIndex + 1) * (EndColumnIndex -
BeginColumnIndex + 1));
            delete []value;
            return result;
        }
    }

    void turned_from_matrix(Matrix<T> & m,size_t b_r_i, size_t b_c_i, size_t e_r_i, size_t e_c_i)
    {
        if(m.getRows() != 0 and m.getColumns() != 0 and m.getChannels() == 1 and
m.get_pointer_count() != NULL and m.get_pointer_elements() != NULL)
        {
            if(this->count == NULL and this->elements == NULL)
            {
                this->rows = m.getRows();
                this->columns = m.getColumns();
                this->channels = 1;
                this->count = m.get_pointer_count();
                (*(this->count)) ++;
                this->elements = m.get_pointer_elements();
                if(b_r_i <= this->rows - 1 and b_c_i <= this->columns - 1 and e_r_i <= this->rows - 1
\
                and e_c_i <= this->columns - 1 and b_r_i <= e_r_i and b_c_i <= e_c_i \
                and ((b_r_i - e_r_i) != 0 or (b_c_i - e_c_i) != 0) )
                {
                    BeginRowIndex = b_r_i;
                    BeginColumnIndex = b_c_i;
                    EndRowIndex = e_r_i;
                    EndColumnIndex = e_c_i;
                }
                this->BeginRowIndex = b_r_i;
                this->BeginColumnIndex = b_c_i;
```

```
                this->EndRowIndex = e_r_i;
                this->EndColumnIndex = e_c_i;
            }
            else
            {
                cout<<"This ROI_matrix is not empty! Turning failed!"<<endl;
            }
        }
        else
        {
            cout<<"This matrix does not qualify for conversion to ROI! Turning failed!"<<endl;
        }
    }

};

template<>
Matrix<float> Matrix<float>::operator*(const Matrix<float> & m) const
{    //similar with the function in project4, and we omit it here.
}

template<>
Matrix<double> Matrix<double>::operator*(const Matrix<double> & m) const
{    //similar with the function in project4, and we omit it here.
}


#endif
```

Part 3 – Result & Verification

1 – Testing for class Matrix

1) This code mainly tests the construction of the matrix and the setting of the values, and the following output indicates that the construction is correct.

```
Matrix<float> m1;  //testing for constructor 1
Matrix<float> m2(3,3);  //testing for constructor 2
Matrix<float> m3(3,3,2);  //testing for constructor 3

float * m2_data = new float[9]{-1.0f,2.0f,-3.0f,4.0f,0.0f,1.0f,2.0f,-3.0f,-4.0f};
m1.setData(m2_data,9);  //testing for function setData()
m2.setData(m2_data,9);  //testing for function setData()

float * m3_data = new float[18]{-1.0f,2.0f,-3.0f,4.0f,0.0f,1.0f,2.0f,-3.0f,-4.0f,1.0f,2.0f,-3.0f,-4.0f,1.1f,-1.0f,2.0f,-3.0f,4.0f};
m3.setData(m3_data,18);  //testing for function setData()

Matrix<float> m4 = m3;  //testing for copy constructor
Matrix<float> m5;
m5 = m3;  //testing for overloading operator =
Matrix<float> m6(1,11,1);
m6 = m3;  //testing for overloading operator =
```

```
Setting succeeded but there is so much data that some of it wasn't used!
m1 : This is an invalid matrix!

m2 : This is an matrix with rows 3, columns 3 and channel 1 .
The channel 0 is:
-1.000000  2.000000  -3.000000  ;
4.000000  0.000000  1.000000  ;
2.000000  -3.000000  -4.000000  ;

m3 : This is an matrix with rows 3, columns 3 and channel 2 .
The channel 0 is:
-1.000000  2.000000  -3.000000  ;
4.000000  0.000000  1.000000  ;
2.000000  -3.000000  -4.000000  ;
The channel 1 is:
1.000000  2.000000  -3.000000  ;
-4.000000  1.100000  -1.000000  ;
2.000000  -3.000000  4.000000  ;

m4 : This is an matrix with rows 3, columns 3 and channel 2 .
The channel 0 is:
-1.000000  2.000000  -3.000000  ;
4.000000  0.000000  1.000000  ;
2.000000  -3.000000  -4.000000  ;
The channel 1 is:
1.000000  2.000000  -3.000000  ;
-4.000000  1.100000  -1.000000  ;
2.000000  -3.000000  4.000000  ;

m5 : This is an matrix with rows 3, columns 3 and channel 2 .
The channel 0 is:
-1.000000  2.000000  -3.000000  ;
4.000000  0.000000  1.000000  ;
2.000000  -3.000000  -4.000000  ;
The channel 1 is:
1.000000  2.000000  -3.000000  ;
-4.000000  1.100000  -1.000000  ;
2.000000  -3.000000  4.000000  ;
```

```
m6 : This is an matrix with rows 3, columns 3 and channel 2 .
The channel 0 is:
-1.000000  2.000000  -3.000000  ;
4.000000  0.000000  1.000000  ;
2.000000  -3.000000  -4.000000  ;
The channel 1 is:
1.000000  2.000000  -3.000000  ;
-4.000000  1.100000  -1.000000  ;
2.000000  -3.000000  4.000000  ;

m1.getCount() = 0
m2.getCount() = 1
m3.getCount() = 4
m4.getCount() = 4
m5.getCount() = 4
m6.getCount() = 4
```

2) This part of code tests the '<<' and '>>' operator.

```
cin>>m2;  //testing for overloading operator >>
cin>>m1;  //testing for overloading operator >>
cout<<m1;  //testing for overloading operator <<
```

```
For this matrix which has been defined, you cannot input the data! You can change the elements by function change_one_entry.
Please enter the numbers of rows, columns and channels of the matrix, respectively
2
2
2
Pleasr enter the matrix in channel 0. To enter a 2 by 2 matrix, please follow the order of row major.
1
2
3
4
Pleasr enter the matrix in channel 1. To enter a 2 by 2 matrix, please follow the order of row major.
3
2
1
4
This is an matrix with rows 2, columns 2 and channel 2 .
The channel 0 is:
1.000000  2.000000  ;
3.000000  4.000000  ;
The channel 1 is:
3.000000  2.000000  ;
1.000000  4.000000  ;
```

3) This part of code tests for overloading operators '+', '-', '*', '==='. And the result are all correct since we verified this by Matlab (see Appendix for Matlab code).

```
Matrix<float> m10(4,4,2);
m10.random_generation(-1.0f,1.0f);
Matrix<float> m11;
m11 = m8 + m10;  //equivalently, "Matrix<float> k = h + j;" is also acceptable
cout<<"m8 : "<<m8<<endl;
cout<<"m10 : "<<m10<<endl;
cout<<"m11 = m8 + m10 : "<<m11<<endl;  //testing for overloading operator +
Matrix<float> m12 = m8 - m10;
cout<<"m12 = m8 - m10 : "<<m12<<endl;  //testing for overloading operator -
Matrix<float> m13 = m8 * m10;
cout<<"m13 = m8 * m10 : "<<m13<<endl;  //testing for overloading operator *
bool b1 = m8 == m10;
cout<<"b1 = (m8 == m10) : "<<b1<<endl;  //testing for overloading operator ==
bool b2 = m8 == m8;
cout<<"b2 = (m8 == m8) : "<<b2<<endl;  //testing for overloading operator ==
```

```
m8 : This is an matrix with rows 4, columns 4 and channel 2 .
The channel 0 is:
-2.147881  -5.016920  -6.840259  1.237558  ;
7.290521  -0.801593  2.226153  7.019300  ;
-6.217145  7.727583  -9.566592  -9.537289  ;
-7.704797  -0.031849  6.587893  -2.839044  ;
The channel 1 is:
-4.248395  6.264703  -5.752336  -7.280943  ;
4.128837  7.103493  -8.034158  -9.111702  ;
0.293632  -3.888115  0.252735  -8.909906  ;
-1.753165  2.177663  -7.990342  6.098954  ;

m10 : This is an matrix with rows 4, columns 4 and channel 2 .
The channel 0 is:
-0.214788  -0.501692  -0.684026  0.123756  ;
0.729052  -0.080159  0.222615  0.701930  ;
-0.621715  0.772758  -0.956659  -0.953729  ;
-0.770480  -0.003185  0.658789  -0.283904  ;
The channel 1 is:
-0.424839  0.626470  -0.575234  -0.728094  ;
0.412884  0.710349  -0.803416  -0.911170  ;
0.029363  -0.388811  0.025274  -0.890991  ;
-0.175316  0.217766  -0.799034  0.609895  ;

m11 = m8 + m10 : This is an matrix with rows 4, columns 4 and channel 2 .
The channel 0 is:
-2.362669  -5.518612  -7.524284  1.361313  ;
8.019573  -0.881752  2.448768  7.721230  ;
-6.838860  8.500341  -10.523252  -10.491017  ;
-8.475277  -0.035034  7.246683  -3.122948  ;
The channel 1 is:
-4.673234  6.891173  -6.327569  -8.009038  ;
4.541720  7.813843  -8.837574  -10.022872  ;
0.322995  -4.276927  0.278009  -9.800897  ;
-1.928481  2.395429  -8.789376  6.708849  ;

m12 = m8 - m10 : This is an matrix with rows 4, columns 4 and channel 2 .
The channel 0 is:
-1.933093  -4.515228  -6.156233  1.113802  ;
6.561469  -0.721434  2.003538  6.317370  ;
-5.595431  6.954824  -8.609933  -8.583560  ;
-6.934317  -0.028664  5.929104  -2.555140  ;
The channel 1 is:
-3.823555  5.638233  -5.177103  -6.552849  ;
3.715953  6.393144  -7.230742  -8.200532  ;
0.264269  -3.499303  0.227462  -8.018916  ;
-1.577848  1.959896  -7.191308  5.489059  ;

m13 = m8 * m10 : This is an matrix with rows 4, columns 4 and channel 2 .
The channel 0 is:
0.102919  -3.810080  7.711450  2.385065  ;
-8.942579  -1.895418  -2.670781  -3.776375  ;
20.265156  -4.862633  8.841871  16.486427  ;
-0.276684  8.967878  -2.909511  -6.452916  ;
The channel 1 is:
5.499042  2.439665  3.083000  -1.930314  ;
2.540347  8.772106  -1.004593  -7.877500  ;
-0.160611  -4.616511  10.080574  -2.330353  ;
0.340066  4.883479  -5.816307  10.131292  ;

b1 = (m8 == m10) : 0
b2 = (m8 == m8) : 1
```

4) This part of code tests the function for finding channel-wise inverse of a matrix. And the result are all correct since we verified this by Matlab (see Appendix for Matlab code).

```
Matrix<double> m14 = m10.inverse();
cout<<"the channel-wise inversr of matrix m10 : "<<m14<<endl;  //testing for function inverse
```

```
 the channel-wise inversr of matrix m10 : This is an matrix with rows 4, columns 4 and channel 2 .
 The channel 0 is:
 -1.188826  -1.933805  -0.980673  -2.004985  ;
 -0.087610  2.563298  1.508164  1.232931  ;
 -0.759228  -0.566212  -0.551145  0.120613  ;
 1.465544  3.905466  1.365589  2.185003  ;
 The channel 1 is:
 -1.259666  1.066023  -0.177604  -0.170638  ;
 0.594916  0.359136  -1.424614  -0.834453  ;
 0.213294  -0.233890  -0.752203  -1.193683  ;
 -0.295073  -0.128223  -0.527861  0.324656  ;
```

2 – Testing for class ROI_of_Matrix

1) This part of code tests for function `showROI()`

```
ROI_of_Matrix<int> r_m1(4,4,1,1,2,2);
r_m1.random_generation(-10,10);
cout<<"r_m1: "<<endl;
r_m1.showROI();
```

```
r_m1:
This is an matrix with rows 4 and columns 4.
1  -6  3  -4
8  3  3  -1
-8  0  -1  7
1  -6  1  -3
And the ROI is form row 1, column 1 to row 2, column 2
3  3  ;
0  -1  ;
```

2) This part of code tests for function `turn_to_matrix()`

```
Matrix<double> m5 = r_m4.turn_to_matrix();
cout<<"r_m4: "<<endl;
r_m4.showROI();
cout<<"r_m4.turn_to_matrix() = "<<m5<<endl;
```

```
r_m4:
This is an matrix with rows 4 and columns 4.
1.91174  -6.91578  3.17839  -4.75335
8.26539  3.88696  3.9553  -1.47978
-8.49152  0.864308  -1.78794  7.15851
1.93497  -6.89777  1.7069  -3.33988
And the ROI is form row 2, column 1 to row 3, column 1
0.864308  ;
-6.897767  ;
r_m4.turn_to_matrix() = This is an matrix with rows 2, columns 1 and channel 1 .
The channel 0 is:
0.864308  ;
-6.897767  ;
```

3) This part of code tests for function `turned_from_matrix()`: To show that m11 and r_m10 shared a same pointer elements, please see their results of getcount().

```
ROI_of_Matrix<float> r_m10;
Matrix<float> m11(4,4,1);
m11.random_generation(-10,10);
r_m10.turned_from_matrix(m11,1,1,2,2);
cout<<"Matrix m11 is: "<<m11<<endl;
cout<<"ROI_Matirx turned from m11 with ROI from (1,1) to (2,2) is: "<<endl;
r_m10.showROI();
cout<<"To show that m11 and r_m10 shared a same pointer elements, please see their getcount():"<<endl;
cout<<"r_m10.getcount() = "<<r_m10.getCount()<<endl;
cout<<"m11.getcount() = "<<m11.getCount()<<endl;
```

```
Matrix m11 is: This is an matrix with rows 4, columns 4 and channel 1 .
The channel 0 is:
1.911741  -6.915780  3.178391  -4.753350  ;
8.265387  3.886964  3.955300  -1.479778  ;
-8.491518  0.864308  -1.787942  7.158505  ;
1.934970  -6.897767  1.706897  -3.339884  ;

ROI_Matirx turned from m11 with ROI from (1,1) to (2,2) is:
This is an matrix with rows 4 and columns 4.
1.91174  -6.91578  3.17839  -4.75335
8.26539  3.88696  3.9553  -1.47978
-8.49152  0.864308  -1.78794  7.1585
1.93497  -6.89777  1.7069  -3.33988
And the ROI is form row 1, column 1 to row 2, column 2
3.886964  3.955300  ;
0.864308  -1.787942  ;
To show that m11 and r_m10 shared a same pointer elements, please see their getcount():
r_m10.getcount() = 2
m11.getcount() = 2
```

Part 4 - Difficulties & Solutions

1. Use an extra pointer `size_t * count` in class matrix to denote the numbers of matrices whose elements are the same. In this way, we are able to better **manage memory** and **avoid hard copies** when the assignment operation was called.
2. The **overloading** of functions and **friend functions** are used to support matrix and constant and constant and matrix operations.
3. For users, they cannot add matrices of different types. If they want to do so, for example, add a matrix of int type to a matrix of float type, they can create two matrix of float type and add them.
4. For some specific types, we rewrote the method in project 4 to speed up matrix operations by **SIMD** and **OpenMP**.
5. A **derived class** is used to describe ROI of a matrix, which let us avoid hard copy when we consider the ROI of a matrix.

Appendix – Matlab code and result for verifying the result given by operators in my code

Code:

```
clear;clc;

m8 = zeros(4,4,2);
m8(:,:,1) = [-2.147881  -5.016920  -6.840259  1.237558  ;
7.290521  -0.801593  2.226153  7.019300  ;
-6.217145  7.727583  -9.566592  -9.537289  ;
-7.704797  -0.031849  6.587893  -2.839044  ];
m8(:,:,2) = [-4.248395  6.264703  -5.752336  -7.280943  ;
4.128837  7.103493  -8.034158  -9.111702  ;
0.293632  -3.888115  0.252735  -8.909906  ;
-1.753165  2.177663  -7.990342  6.098954];

m10 = zeros(4,4,2);
m10(:,:,1) = [-0.214788  -0.501692  -0.684026  0.123756  ;
0.729052  -0.080159  0.222615  0.701930  ;
-0.621715  0.772758  -0.956659  -0.953729  ;
-0.770480  -0.003185  0.658789  -0.283904];
m10(:,:,2) = [-0.424839  0.626470  -0.575234  -0.728094  ;
0.412884  0.710349  -0.803416  -0.911170  ;
0.029363  -0.388811  0.025274  -0.890991  ;
-0.175316  0.217766  -0.799034  0.609895 ];

disp('m8 + m10')
disp(m8+m10)
disp('m8 - m10')
disp(m8-m10)
disp('m8 * m10')
disp(pagemtimes(m8,m10))
disp('channel-wise inverse of m10')
disp(pageinv(m10))
```

Output:

```
m8 + m10

(:,:,1) =

   -2.3627   -5.5186   -7.5243    1.3613
    8.0196   -0.8818    2.4488    7.7212
   -6.8389    8.5003  -10.5233  -10.4910
   -8.4753   -0.0350    7.2467   -3.1229


(:,:,2) =

   -4.6732    6.8912   -6.3276   -8.0090
    4.5417    7.8138   -8.8376  -10.0229
    0.3230   -4.2769    0.2780   -9.8009
   -1.9285    2.3954   -8.7894    6.7088
```

```
m8 - m10

(:,:,1) =

   -1.9331   -4.5152   -6.1562    1.1138
    6.5615   -0.7214    2.0035    6.3174
   -5.5954    6.9548   -8.6099   -8.5836
   -6.9343   -0.0287    5.9291   -2.5551


(:,:,2) =

   -3.8236    5.6382   -5.1771   -6.5528
    3.7160    6.3931   -7.2307   -8.2005
    0.2643   -3.4993    0.2275   -8.0189
   -1.5778    1.9599   -7.1913    5.4891
```

```
m8 * m10

(:,:,1) =

    0.1029   -3.8101    7.7114    2.3851
   -8.9426   -1.8954   -2.6708   -3.7764
   20.2652   -4.8626    8.8419   16.4864
   -0.2767    8.9679   -2.9095   -6.4529


(:,:,2) =

    5.4990    2.4397    3.0830   -1.9303
    2.5403    8.7721   -1.0046   -7.8775
   -0.1606   -4.6165   10.0806   -2.3303
    0.3401    4.8835   -5.8163   10.1313
```

```
channel-wise inverse of m10

(:,:,1) =

   -1.1888   -1.9338   -0.9807   -2.0050
   -0.0876    2.5633    1.5082    1.2329
   -0.7592   -0.5662   -0.5511    0.1206
    1.4655    3.9055    1.3656    2.1850


(:,:,2) =

   -1.2597    1.0660   -0.1776   -0.1706
    0.5949    0.3591   -1.4246   -0.8345
    0.2133   -0.2339   -0.7522   -1.1937
   -0.2951   -0.1282   -0.5279    0.3247
```