CS205 C/ C++ Programming - Project4

> 11811901
> 樊顺

Part 1 Matrices Class

In this part, I using the template class to implement this class.
The advantage of using `template class` is that it can easily match multiple data storage formats. I used and tested the four types of data stored as `integer`, `double`, `float`, `unsigned char` and `short` in my project. The Matrix class can be well adapted to these data types.
The details of this class is :

```
template <class T>
class Matrix
{
public:
    int row;
    int col;
    int len;
    int high;
    int channel;
    T *p_mat;
    T *start;

public:
    Matrix();
    Matrix<T> create_mat(char *path);
```

```
    void delete_mat();
    void allocateMem();
    Matrix<T> roi(Matrix<T> &mat, Rect &rec);
    Matrix<T> operator+(const Matrix<T> &second);
    Matrix<T> operator-(const Matrix<T> &second);
    Matrix<T> operator*(const Matrix<T> &second);
    Matrix<T> operator*(const T &number);
    Matrix<T> operator*(const char &c);
    bool operator==(const Matrix<T> &second);
    void operator=(const Matrix<T> &second);
    void write_mat(char *path);
};
```

Description

The description of the `variables` and `methods` in the type is as follows:

Varibales

- `row` : Store the number of rows of the matrix.
- `col` : Store the number of columns of the matrix.
- `len` : This variable represents the length of the window in the ROI, and is set to 0 during initialization.
- `high` : This variable represents the height of the window in the ROI, and is set to 0 during initialization.
- `channel` : This variable is used to mark how many channels the matrix has.
- `*p_mat` : This is a template type 1D array pointer, this pointer points to the address of the array data stored in the Matrix class.
    - There are some implementation problems here. When I store, I separate the different channels and store them.
      In other words, if it is a 3-channel matrix, I have to store the structure like this:
channel1	channel2	channel3
- `*start` :This pointer is used to mark the starting point of the ROI, and it is only used for the ROI.

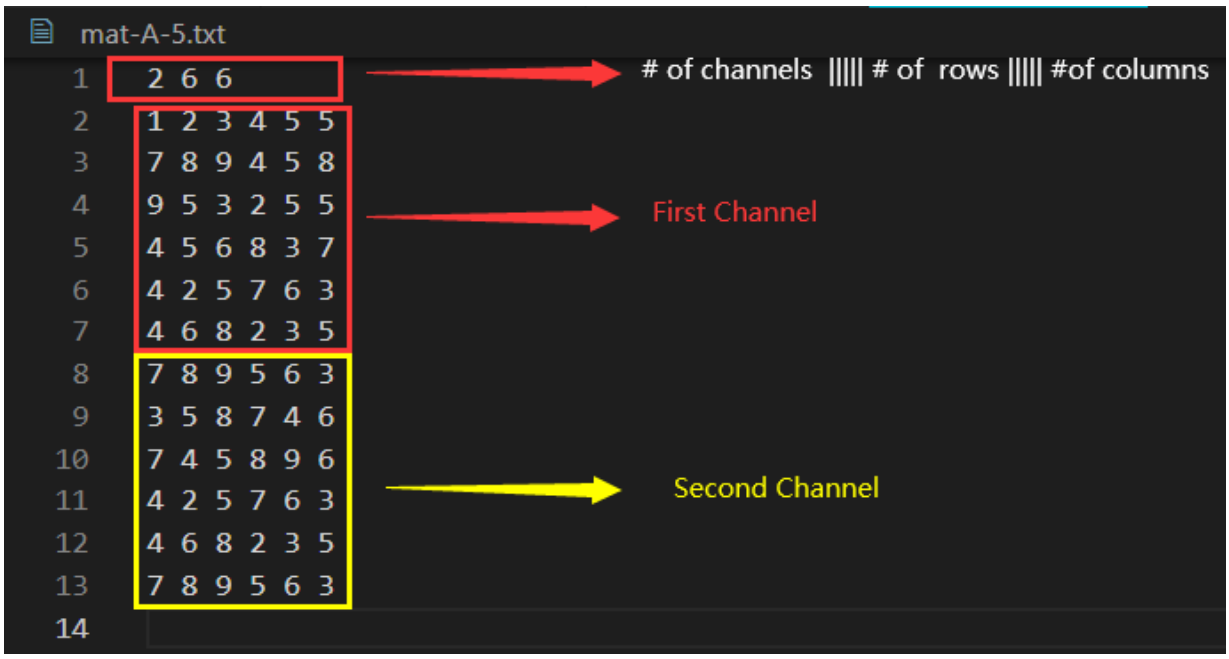
Methods

- `Matrix();` : No args constructor, in this constructor initialize the len and high.

```
template <class T>
Matrix<T>::Matrix()
{
   this->len = 0;
```

```
    this->high = 0;
};
```

- `Matrix<T> create_mat(char *path);`
  - **argument** : `char *path`
    - This argument means the file which will be read in the matrix
    - The file format looks like:



- `void delete_mat();`
  - This method is used to delet the memory of the matrix data in `Matrix<T>` class.
- `void allocateMem();`
  - This method is used to apply for storage space for storing matrix data.
- `Matrix<T> roi(Matrix<T> &mat, Rect &rec);`
  - **Arguments:**
    - `Matrix<T> &mat`
      - This parameter refers to the matrix used to make the ROI.
    - `Rect &rec`
      - The size of the window in the ROI operation is stored in Rect, which is stored in the matrix according to the coordinates of the two points on the upper left and lower right of the window.

```
class Rect
{
public:
    int x1;
    int y1;
```

```
    int x2;
    int y2;
public:
    Rect(int x1, int y1, int x2, int y2);
};
Rect::Rect(int x1, int y1, int x2, int y2)
{
    this->x1 = x1;
    this->y1 = y1;
    this->x2 = x2;
    this->y2 = y2;
}
```

> **The ROI code will be shown later.**

- `Matrix<T> operator+(const Matrix<T> &second);`
  - Overloaded addition operator.

```
template <class T>
Matrix<T> Matrix<T>::operator+(const Matrix<T> &second)
{
    Matrix<T> *Result = (Matrix<T> *)malloc(sizeof(Matrix<T>));
    Result->channel = second.channel;
    Result->row = second.row;
    Result->col = second.col;

    Result->allocateMem();
    for (int i = 0; i < this->channel; i++)
    {
        for (int j = 0; j < this->row; j++)
        {
            for (int k = 0; k < this->col; k++)
            {
                Result->p_mat[i * this->row * this->col + j * this->col + k] = this->p_mat[i * this->row * this->col + j * this->col + k] + second.p_mat[i * this->row * this->col + j * this->col + k];
            }
        }
    }
    for (int i = 0; i < this->channel * this->row * this->col; i++)
    {
        Result->p_mat[i] = this->p_mat[i] + second.p_mat[i];
    }
```

```
    return *Result;
}
```

- `Matrix<T> operator-(const Matrix<T> &second);`
    - Overloaded subtraction operator.

```
template <class T>
Matrix<T> Matrix<T>::operator-(const Matrix<T> &second)
{
    Matrix<T> *Result = (Matrix<T> *)malloc(sizeof(Matrix<T>));
    Result->channel = second.channel;
    Result->row = second.row;
    Result->col = second.col;

    Result->allocateMem();
    for (int i = 0; i < this->channel * this->row * this->col; i++)
    {
        Result->p_mat[i] = this->p_mat[i] - second.p_mat[i];
    }

    return *Result;
}
```

- `Matrix<T> operator*(const Matrix<T> &second);`
    - Overloaded the multiplication operator of two matrices.

```
template <class T>
Matrix<T> Matrix<T>::operator*(const Matrix<T> &second)
{
    Matrix<T> *Result = (Matrix<T> *)malloc(sizeof(Matrix<T>));
    Result->channel = second.channel;
    Result->row = second.row;
    Result->col = second.col;

    Result->allocateMem();
#pragma omp parallel for
    for (int c = 0; c < this->channel; c++)
    {
        for (int i = 0; i < Result->row; i++)
        {
            for (int k = 0; k < this->col; k++)
            {
                float tmp = this->p_mat[c * this->row * this->col + i * this->col +
k];
```

```
                for (int j = 0; j < Result->col; j++)
                {
                    Result->p_mat[c * this->row * this->col + i * this->col + j] +=
tmp * second.p_mat[c * this->row * this->col + k * this->col + j];
                }
            }
        }
    }
    return *Result;
}
```

- `Matrix<T> operator*(const T &number);`
  - Overloaded the multiplication operators for matrices and numbers.

```
template <class T>
Matrix<T> Matrix<T>::operator*(const T &number)
{
    Matrix<T> *Result = (Matrix<T> *)malloc(sizeof(Matrix<T>));
    Result->channel = this.channel;
    Result->row = this.row;
    Result->col = this.col;

    Result->allocateMem();

    for (int i = 0; i < this->channel * this->row * this->col; i++)
    {
        Result->p_mat[i] *= number;
    }
    return *Result;
}
```

- `Matrix<T> operator*(const char &c);`
  - The char input can be `any character`, and the purpose of this function is to **achieve matrix conversion to rank**.

```
template <class T>
Matrix<T> Matrix<T>::operator*(const char &c)
{

    Matrix<T> *Result = (Matrix<T> *)malloc(sizeof(Matrix<T>));
    Result->channel = this->channel;
    Result->row = this->row;
    Result->col = this->col;
```

```
    Result->allocateMem();
    int count = 0;
    for (int i = 0; i < this->channel; i++)
    {
        for (int j = 0; j < this->row; j++)
        {
            for (int k = 0; k < this->col; k++)
            {
                T x = this->p_mat[i * (this->row * this->col) + k * this->row + j];
                Result->p_mat[count] = x;
                count += 1;
            }
        }
    }
    return *Result;
}
```

- `bool operator==(const Matrix<T> &second);`
  - Overloaded the operator for judging whether two matrices are equal.

```
template <class T>
bool Matrix<T>::operator==(const Matrix<T> &second)
{
    bool result = true;
    if (this->channel = second.channel && this->row == second.row && this->col == second.row)
    {
        for (int i = 0; i < this->channel * this->row * this->col; i++)
        {
            if (this->p_mat[i] != second.p_mat[i])
            {
                result = false;
            }
        }
    }
    else
    {
        result = false;
    }
    return result;
}
```

- `void operator=(const Matrix<T> &second);`

- Overloaded the matrix assignment operator.

```
template <class T>
void Matrix<T>::operator=(const Matrix<T> &second)
{
    this->channel = second.channel;
    this->row = second.row;
    this->col = second.col;
    this->p_mat = second.p_mat;
}
```

- `void write_mat(char *path);`
  - Output the matrix in the parameter to a file.

Check operator accuracy

This part of the inspection is first by creating some multi-channel matrices with a size of `3*3~5*5`, and then manually calculating the results by themselves and comparing them with the program output results.
After there is no problem with this part, the `50*50~70*70` size of matrix of the resume calculates the results separately on the online cloud computing webpage and my own python check program, and then compares the results of the three to determine whether your results are accurate.

Part 2 ROI Method

The code to get the ROI is as follows

```
template <class T>
Matrix<T> Matrix<T>::roi(Matrix<T> &mat, Rect &rec)
{
    int len = rec.y2 - rec.y1 + 1;
    int high = rec.x2 - rec.x1 + 1;
    // This is the start point of the window
    // Then I make a new Matrix and set this as it begin point of matrix
    // I also store the len and hight, means the window size and this will be used in
read the data in window.
    T *ptr = mat.p_mat + (mat.row * rec.x1 + rec.y1);
    this->col = mat.col;
    this->row = mat.row;
    this->channel = mat.channel;
    this->high = high;
    this->len = len;
    this->p_mat = mat.p_mat;
    this->start = ptr;
```

```
    return *this;
}
```

My understanding of ROI is to pool the data in this area after obtaining a window, that is to say, ROI is for data pooling. So the main thing I pay attention to when implementing is the window size and the display of window data.

Check of ROI

The display of ROI is in method `void display_mat(Matrix<T> *ma)`
The main code is as follows:

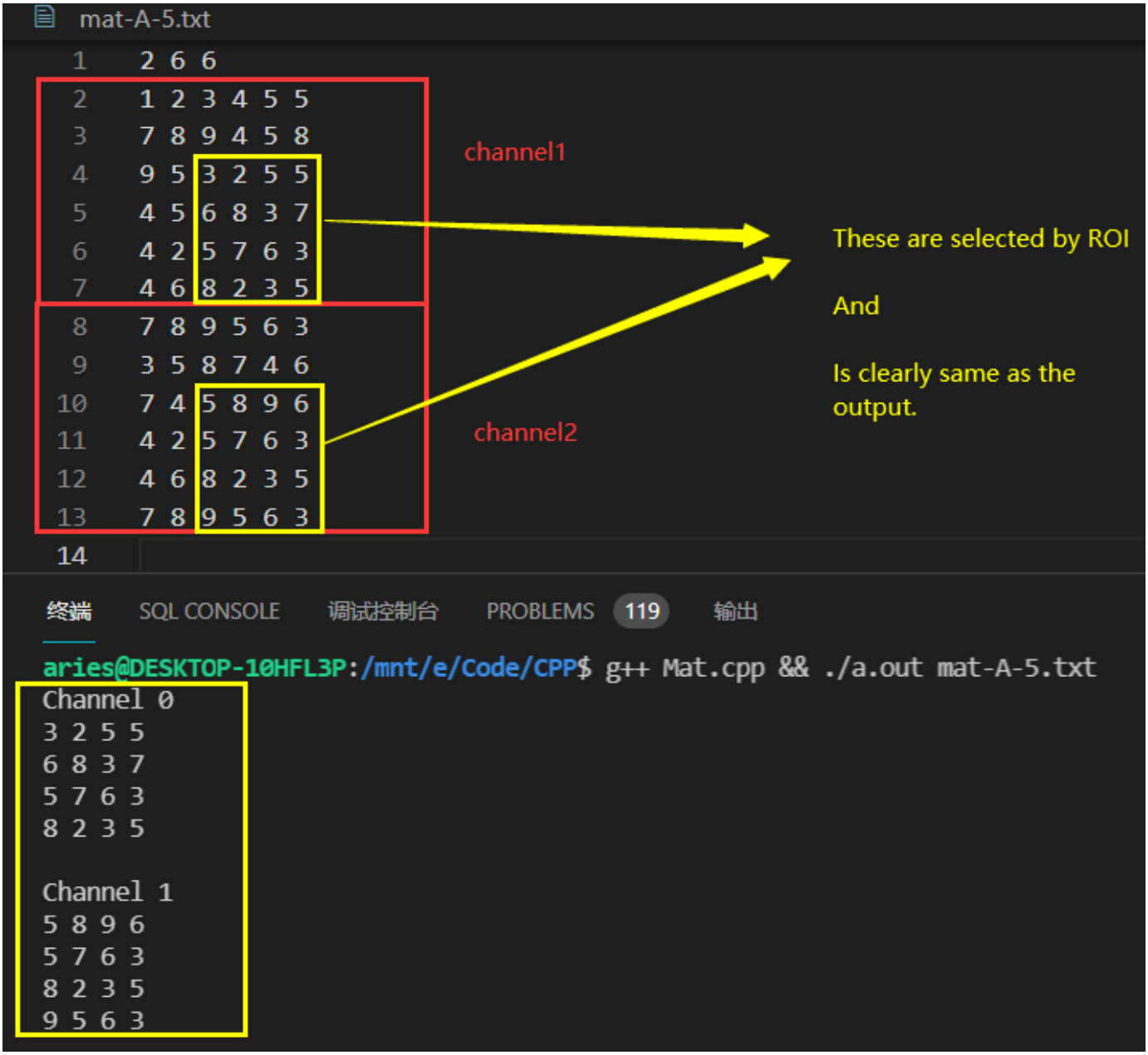
```
T *ptr = ma->start;
for (int c = 0; c < ma->channel; c++)
{
    cout << "Channel " << c << endl;
    for (int i = 0; i < ma->high; i++)
    {
        for (int j = 0; j < ma->len; j++)
        {
            cout.precision(3);
            cout << (double)ptr[j] << " ";
        }
        ptr += ma->col;
        cout << endl;
    }
    if (c + 1 < ma->channel)
    {
        ptr = ma->start + (ma->row * ma->col);
    }
    cout << endl;
}
```

When outputting, I convert all the output into a `double` type with **three significant digits** for output.
In order to avoid the `unsigned char` may cause the output characters and the result can not be seen.

In order to check whether it is correct or not, I created a `6 * 6` single-channel matrix and got the following results after running the following code.

```
Matrix<int> ma;
Matrix<int> *pt_ma = &ma;
ma = ma.create_mat(argv[1]);
Matrix<int> mb;
Matrix<int> *pt_mb = &mb;
mb = mb.create_mat(argv[1]);
```

```
Rect rec(2, 2, 5, 5);
Matrix<int> ro;
ro = ro.roi(ma, rec);
Matrix<int> *pt_ro = &ro;
display_mat(pt_ro);
```



Part3 Avoid Memory Hard Copy

In most cases, a hard copy of the memory can solve many problems, but multiple copies will cause a lot of time to be consumed. The best way to avoid hard copies suggested by the professor is to use soft copies and mark the number of citations.

But after two weeks of hard work, I found it difficult to fully implement such a function. So I chose a very inelegant way-pointers and references.
In this project, most of the matrix parameters are passed by pointers and references, all of which point

to the same storage area. Only when new content needs to be stored after calculation will reapply for memory space storage.

For example, ROI, etc., after creating a new `Matrix<T>`, initialize other int type variables, and finally adjust the memory space pointed to by the pointer, instead of applying for new memory and deep copy.

Part 4 Compare the difference between X86 and ARM platforms

When measuring the gap, I choose the most time-consuming matrix multiplication.
In the case of using multi-threading, the running time on the X86 platform and the ARM platform are as follows:

- This is the result on X86 plantform

```
aries@DESKTOP-10HFL3P:/mnt/e/Code/CPP$ g++ Mat.cpp && ./a.out mat-A-2048.txt
all time : 50s
Finish Free Mem of Matrix
```

- This is the result on ARM plantform

```
[root@ecs001-0021-0042 cpp_project4]# g++ Mat.cpp && ./a.out mat-A-2048.txt
all time : 133s
Finish Free Mem of Matrix
```

  It can be clearly seen that there is a big difference between ARM and X86, but due to the use of multi-threading, after careful inspection, it is found that the number of threads between the two is relatively large.

- This is the CPU information of X86

```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          16
```

- This is ARM's CPU information

```
Architecture:                    aarch64
CPU op-mode(s):                  64-bit
Byte Order:                      Little Endian
CPU(s):                          2
```

> **All obtained by command:** `lscpu`

In order to distinguish the difference between the two, I canceled the multi-threaded acceleration.
The strange thing is that when I deleted `#pragma omp parallel for`, the speed did not change, but when I added -O3 optimization, something amazing happened:

- This is the result on X86 plantform

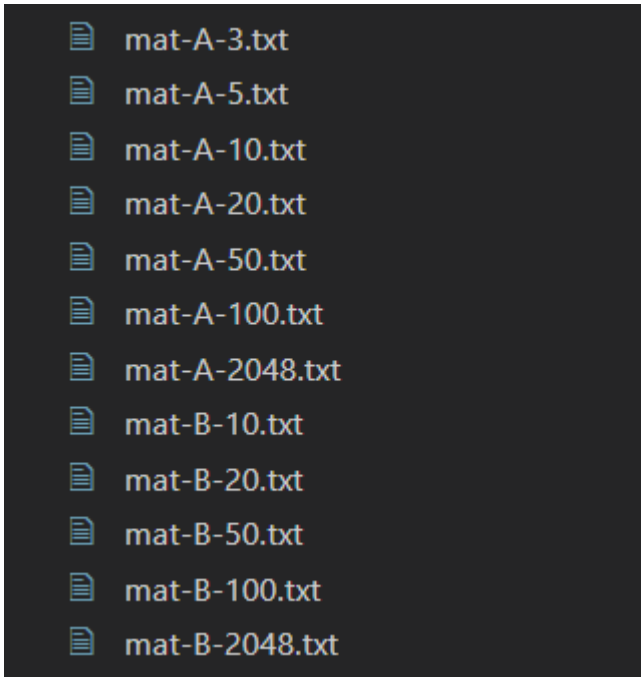
```
aries@DESKTOP-10HFL3P:/mnt/e/Code/CPP$ g++ -O3 Mat.cpp && ./a.out mat-A-2048.txt mat-B-2048.txt
all time : 2s
Finish Free Mem of Matrix
aries@DESKTOP-10HFL3P:/mnt/e/Code/CPP$ g++ Mat.cpp && ./a.out mat-A-2048.txt mat-B-2048.txt
all time : 49s
Finish Free Mem of Matrix
```

- This is the result on ARM plantform

```
[root@ecs001-0021-0042 cpp_project4]# g++ Mat.cpp && ./a.out mat-A-2048.txt mat-B-2048.txt
all time : 133s
Finish Free Mem of Matrix
[root@ecs001-0021-0042 cpp_project4]# g++ -O3 Mat.cpp && ./a.out mat-A-2048.txt mat-B-2048.txt
all time : 6s
Finish Free Mem of Matrix
```

When I saw this result, I was shocked. I suspected that my code was running incorrectly.
So I chose to randomly generate a matrix of 10 20 50 100 size, and after the calculation, the output result and the python code (the code I used to determine the accuracy in project3, I will not show the code here) and some recurring matrix clouds.
After comparing the results of the computing platform. I found that there is no problem with the result of my code.

- This is randomly generated data:



Part 5 Some implementation problems found but not resolved in the end

- In the process of implementation, it was found that it was difficult to implement soft copy. After many attempts and choosing to use pointers and references to avoid hard copy, I plan to read the OpenCv's source code again and try to implement soft copy.
- In the implementation, too many incorrect inputs were ignored, such as the matrix that did not meet the calculation requirements, the ROI window that exceeded the limit, and so on.