Project 5: A Class for Matrices

- Author: Jiacheng Luo
- Student ID: 12112910

1 Analysis

1.1 Requirement Statement

In this project, I design a **class template** for matrices, which satisfy the following requirements:

- The class template has a variable *channel* to implement **multi-channel** matrix. In other words, the element of the matrix can be a **vector**. As we know, an image can be viewed as a multi-channel matrix (i.e. We can use a three channel matrix to describe an RGB image), so this feature plays a great role in **image processing**.
- The class template can be instantiated to support different types of multi-channel matrix data including **unsigned char**, **short**, **int**, **float**, **double**, etc. With the use of class templates, in addition to the storage type requirements mentioned in the project requirements, it can be easily extended to support more kinds of data.
- All functions and procedures (i.e. constructor and destructor) are carefully designed to **avoid problems of memory management.** We have done a lot of testing, so you can use this class template with confidence.
- Some **operators** are **overloaded** (i.e. Plus `+`, minus `-`, multiplication `*`, transpose `~`, assign `=`, judge equality `==` etc.) to support common **matrix operations**. This will make it more convenient to use the matrix.
- In particular, the **assignment operator** `=` is overloaded to **avoid hard memory copy** when data of one matrix is assigned to another ones.
- Apply the idea of **ROI** (as known as **Region of Interest**) to matrices : try to revise the desired sub-matrix **without hard copy**.
- This project uses the **optimization** scheme in **Project 4** (i.e. SIMD for x86, NEON for arm) to optimize matrix operation. For the convenience of presentation, the report only shows the relevant optimization of **matrix multiplication**. Also, we have implemented **cross platform**, and matrix templates can be optimized on **x86** platform or **arm** platform.
- We **tested cross platform** programs on **x86** and **arm** platforms, and **compared the differences**.

I think this project has **successfully** completed **ALL** the requirements listed in the **project requirements**.

In my opinion, **Project 5** is highly related to **Project 3** and **Project 4**, so please allow me to introduce my understandings.

- In **Project 3**, we define a structure for matrices in C. Some operations, such as multiplication, matrix copy, memory management, are realized by corresponding functions. However, since the programming language is C language, we cannot use more efficient tools.
- In **Project 4**, We focus on using some techniques to accelerate matrix multiplication, and try to realize the cross platform function of matrix multiplication.

Therefore, in the **project 5**, we should pay more attention to how to use the **features of C++** to create a fast and easy to use matrix class template. Specifically, we need to use the ideas related to matrix management in **project 3** and matrix multiplication optimization in **project 4** to implement matrix templates in **project 5**.

By the way, we can see that all the 5 projects are **closely related**.

1.2 Idea Description

1.2.1 Some Preliminary Plan

In `cv::Mat` in **openCV**, it uses **union** containing 5 pointers of different data type pointing to matrix data, which **share the same memory**. However, it does not use **C++ features** and cannot be extended to more general **data types**. Therefore, I did not use this method in my project. Another idea is to use a **structure** and a predefined data **type** to realize different types of matrices. Although this method can solve the problem, it is not **elegant** and does not use C++ features, so I gave it up.

Next, I will introduce to you how I implemented it.

1.2.2 Data Class Template Structure

In order to manage the memory safely (No memory leakage or memory redundancy), we try to use the idea of **smart pointers**. In **C++ 11** standard, a smart pointer called `shared_ptr` is implemented, and its functions as a pointer, but records how many `shared_ptr` jointly point to one object.

Following the idea of "**reference counting**", first of all, I defined a class temple `template <class type> class Data` to store the **matrix data**, and users can use `Data<type>(length)` to request the data memory space of the matrix:

```cpp
template <class type>
class Data {
 private:
  type *data;
  size_t refcount;
 public:
  Data(size_t length);
  ~Data();
  type *getDataValue();
  void addRefCount();
  void minusRefCount();
  size_t getRefCount();
};
```

In this class template, two private **member variables** are defined:

- `type *data` is the requested space to store data of a given type.
- `size_t refcount` is how many matrices point to this data memory.

Also, a **constructor** and a **destructor** are defined:

- `Data(size_t length);` request a data memory space.
- `~Data();` is used to free data memory space.

Finally, **four member** functions are defined:

- `type *getDataValue();` returns the first address pointing to the actual data storage space.
- `size_t getRefCount();` returns how matrices point to this data memory.

- `void addRefCount();` adds the `refcount` by one.
- `void minusRefCount();` minus the `refcount` by one.

1.2.3 Matrix Class Template Structure

After defining a class temple `template <class type> class Data`, then we can define the class temple `template <class type> class Matrix` which can expand to different types of matrices.

```cpp
template <class type>
class Matrix {
 private:
  size_t rows, cols, channels, size;
  Data<type> *dataptr;
  static size_t count;
#ifdef _DEBUG_
  size_t mat_id;
#endif
 public:
  Matrix();
  Matrix(size_t rows, size_t cols, size_t channels);
  Matrix(size_t rows, size_t cols, size_t channels, const type *data);
  Matrix(const Matrix &matrix);
  ~Matrix();
  Data<type> *getDataPtr() { return dataptr; };
  size_t getRows() { return rows; };
  size_t getCols() { return cols; };
  size_t getChannels() { return channels; };
  size_t getSize() { return size; };
#ifdef _DEBUG_
  size_t getmat_id() { return mat_id; };
#endif
  type getElement(const size_t row_id, const size_t col_id,
                  const size_t channel_id);
  void setElement(const size_t row_id, const size_t col_id,
                  const size_t channel_id, type element);
  bool operator==(Matrix matrix_cmp);
  Matrix<type> &operator=(const Matrix &matrix_copy);
  Matrix<type> operator+(Matrix matrix);
  Matrix<type> operator-(Matrix matrix);
  Matrix<type> operator*(Matrix matrix);
  Matrix<type> operator~();
  Matrix<type> &submat(size_t a, size_t b, std::string fileName);
  static size_t getCount();
};
template <class type>
size_t Matrix<type>::count = 0;
```

Use the `_DEBUG_` macro to define, close and open the debugging mode, and output debugging information when necessary to facilitate the verification of the program.

In this class template, 6 or 7 (depend on whether the `_DEBUG_` macro is enabled or not) private member variables are defined:

- `size_t rows` is the number of rows of the matrix.
- `size_t cols` is the number of columns of the matrix.
- `size_t channels` is the number of channels of the matrix.
- `size_t size` is the size of each channel of the matrix.
- `Data<type> *dataptr` is based on the class template `template <class type> class Data` and is a pointer of the class template, storing the matrix data. It is the key to implement the soft copy and ROI.
- `static size_t count` is a static variable, used to store the number of generated matrices. And we use `template <class type> size_t Matrix<type>::count = 0;` to set the value of the variable `count` to **0**.
- `size_t mat_id` If the `_DEBUG_` macro is enabled, we will use `mat_id` records the number of each matrix.

Also, we define **3 constructors** and **1 copy constructor** to create matrix object.

- `Matrix()` is the default constructor to create an empty matrix object.
- `Matrix(size_t rows, size_t cols, size_t channels);` create a matrix object whose number of rows, columns and channels are `rows`, `cols`, `channels` respectively, and all the element is set to 0 by default.
- `Matrix(size_t rows, size_t cols, size_t channels, const type *data);` create a matrix object whose number of rows, columns and channels are `rows`, `cols`, `channels` respectively, the elements is set by dynamic pointer array `data` which is already known.
- `Matrix(const Matrix &matrix);` create a matrix according to a matrix already created, which is designed to achieve soft copy and avoid problems of memory management.

What's more, `~Matrix()` is the destructor of the matrix which is also designed to achieve soft copy and avoid problems of memory management.

In addition, we have some "**getters**" and "**setters**" to get or set the information of a matrix object:

```
  Data<type> *getDataPtr() { return dataptr; };
  size_t getRows() { return rows; };
  size_t getCols() { return cols; };
  size_t getChannels() { return channels; };
  size_t getSize() { return size; };
#ifdef _DEBUG_
  size_t getmat_id() { return mat_id; };
#endif
  type getElement(const size_t row_id, const size_t col_id,
                  const size_t channel_id);
  void setElement(const size_t row_id, const size_t col_id,
                  const size_t channel_id, type element);
  static size_t getCount();
```

And, we overload the operators:

- `bool operator==(Matrix matrix_cmp)` compares two matrix and returns whether these matrix are equal or not within the accuracy range (float or double type matrices in particular).
- `Matrix<type> &operator=(const Matrix &matrix_copy)` assign a matrix to another one.
- `Matrix<type> operator+(Matrix matrix)` matrix addition.
- `Matrix<type> operator-(Matrix matrix)` matrix subtraction.
- `Matrix<type> operator*(Matrix matrix)` matrix multiplication.
- `Matrix<type> operator~()` matrix transpose.
- `Matrix<type> &submat(size_t a, size_t b, std::string fileName)` Using the idea of ROI, replace a part of the matrix with the matrix in the file.

Finally, besides the matrix template, we also define some convenient symbol overloads.

- `template <class type> std::istream &operator>>(std::istream &is, Matrix<type> &matrix)` is to read the matrix from the input stream.
- `template <class type> std::ostream &operator<<(std::ostream &os, Matrix<type> &matrix)` is to print the matrix to the output stream.
- `template <class type> Matrix<type> operator*(type num, Matrix<type> matrix)` is to realize the left number multiplication of matrix.
- `template <class type> Matrix<type> operator*(Matrix<type> matrix, type num)` is to realize the right number multiplication of matrix.

1.3 Some Important Realization

1.3.1 Assignment Operator

In particular, we overload the assignment operator `=`, following these steps:

1. If the object is **assigned to itself**, then do nothing but **return the reference of itself**.
2. Copy all the four corresponding member variables: *rows*, *columns*, *size*, and *channel*.
3. Judge weather is the data is empty(as know as NULL), if so, do nothing.
4. If not, with the idea of smart pointer, using `template <class type> class Data` to properly manage the memory: just use `=` to assign the address of data of two matrices. In this sway, two pointers point to the same space, sharing the same data, which is actually called **"soft copy"**.

Of course, it is very simple to allocate memory directly. The difficulty is how to manage memory, including deleting the matrix, without causing memory redundancy and leakage. The way to manage memory is to use `template <class type> void Data<type>::addRefCount()` and `template <class type> void Data<type>::minusRefCount()` in order to add or minus the reference count of different data structures.

```cpp
template <class type>
Matrix<type> &Matrix<type>::operator=(const Matrix &mat_copy) {
#ifdef _DEBUG_
  std::cout << "Matrix<type> & Matrix<type>::operator=(const Matrix & mat_copy)"
            << std::endl;
#endif
  if (this == &mat_copy) return *this;
  if (dataptr != NULL) {
    dataptr->minusRefCount();
    delete dataptr;
  }
  rows = mat_copy.rows;
  cols = mat_copy.cols;
  size = mat_copy.size;
  channels = mat_copy.channels;
  if (mat_copy.dataptr) {
    dataptr = mat_copy.dataptr;
    dataptr->addRefCount();
  }
  return *this;
}
```

1.3.2 Region of Interest

To avoid making a memory hard copy when implementing a region of interest (ROI) in imaging or video analysis, we can use pointer arithmetic to create a sub-image or sub-frame that references the original image or frame data in memory, rather than creating a new copy of the data. This can save memory and computational resources, and make it possible to process the region of interest in real time.

Similarly, in our case of matrix class, we need to find the sub-matrix with the given indices (i.e. start index and end index). What's more, in order to avoid hard copy, we can follow the above idea if soft copy, we just assign the address of two pointers the same, sharing the same space. Then we can revise the data of the sub-matrix by pointers.

```
Matrix<type> &Matrix<type>::submat(size_t a, size_t b, std::string fileName) {
  if (a > size || b > size) return *this;
  size_t a_row = a / cols + 1;
  size_t a_col = a % cols + 1;
  size_t b_row = b / cols + 1;
  size_t b_col = b % cols + 1;
  size_t max_row = std::max(a_row, b_row), min_row = std::min(a_row, b_row);
  size_t max_col = std::max(a_col, b_col), min_col = std::min(a_col, b_col);
  size_t size_row = max_row - min_row + 1, size_col = max_col - min_col + 1;
  std::ifstream fin(fileName);
  Matrix ROI_matrix;
  fin >> ROI_matrix;
  fin.close();
  if (size_row * size_col != ROI_matrix.getRows() * ROI_matrix.getCols()) {
    return *this;
  }
  size_t copydata_index = 0;
  type *ROI_ptr = dataptr->getDataValue();
  for (size_t i = (min_row - 1) * cols + min_col - 1;
       i <= (max_row - 1) * cols + max_col - 1; i++) {
    size_t ptr_row = i / cols + 1, ptr_col = i % cols + 1;
    if (ptr_row >= min_row && ptr_row <= max_row && ptr_col >= min_col &&
        ptr_col <= max_col) {
```

```
      *(ROI_ptr + i) = ROI_matrix.getDataPtr()->getDataValue()[copydata_index];
      copydata_index++;
    }
  }
  return *this;
}
```

1.3.3 Matrix Multiplication

In order to facilitate users' use, we overloaded the multiplication operator, and specifically implemented the scalar-multiply-matrix function. For the convenience of users, we overload the operators of several matrix multiplication algorithms **instead of** directly setting them as **member functions**.

In this project, we use the optimization scheme of **Project 4** to provide cross platform implementation.

- Use `i-k-j` loops instead `i-j-k` loop. We assume it in Project 4 that the former is faster than the later.
- Use SIMD (including mavx2 and mavx512f) to optimize especially for x86 platform.
- Use NEON to optimize especially for x86 platform.
- Use `-O3` optimization parameter to speed my code up.

Our method is used for **row column vector dot multiplication** in matrix multiplication.

```
template <class type>
void vectorMul(type *dest, type *v1, type *v2, size_t len) {
#ifdef Use_x86_SIMD_mavx512f
  // use avx512: -mavx512f
  __m512 sum512 = _mm512_setzero_ps();
  size_t p, add_each = 512 / sizeof(type);
  for (p = 0; p + add_each - 1 < len;
       p += add_each, v1 += add_each, v2 += add_each) {
    sum512 = _mm512_add_ps(
        sum512, _mm512_mul_ps(_mm512_loadu_ps(v1), _mm512_loadu_ps(v2)));
  }
  *dest += _mm512_reduce_add_ps(sum512);
  for (; p < len; p++, v1++, v2++) {
    *dest += (*v1) * (*v2);
  }
#else
#ifdef Use_x86_SIMD_mavx2
  // use avx2: -mavx2
  __m256 sum256 = _mm256_setzero_ps();
  size_t p, add_each = 256 / sizeof(type);
  for (p = 0; p + add_each - 1 < len;
       p += add_each, v1 += add_each, v2 += add_each) {
    sum256 = _mm256_add_ps(
        sum256, _mm256_mul_ps(_mm256_loadu_ps(v1), _mm256_loadu_ps(v2)));
  }
  const __m128 sumQuad = _mm_add_ps(_mm256_castps256_ps128(sum256),
                                    _mm256_extractf128_ps(sum256, 1));
  const __m128 loDual = sumQuad;
  const __m128 hiDual = _mm_movehl_ps(sumQuad, sumQuad);
  const __m128 sumDual = _mm_add_ps(loDual, hiDual);
  const __m128 lo = sumDual;
  const __m128 hi = _mm_shuffle_ps(sumDual, sumDual, 0x1);
  const __m128 sum = _mm_add_ss(lo, hi);
  *dest += _mm_cvtss_f32(sum);
  for (; p < len; p++, v1++, v2++) {
    *dest += (*v1) * (*v2);
  }
#else
#ifdef Use_arm_NEON
  float32x4_t a, b;
  float32x4_t c = vdupq_n_f32(0);
  float sum[4] = {0};
  size_t p, add_each = 128 / sizeof(type);
  for (p = 0; p + add_each < len; p += add_each) {
    a = vld1q_f32(v1 + p);
    b = vld1q_f32(v2 + p);
    c = vaddq_f32(c, vmulq_f32(a, b));
  }
  vst1q_f32(sum, c);
  *dest += (sum[0] + sum[1] + sum[2] + sum[3]);
  for (; p < len; p++) {
    *dest += v1[p] * v2[p];
  }
#else
  for (size_t p = 0; p < len; p++) {
    *dest += v1[p] * v2[p];
  }
#endif
#endif
#endif
}
```

2 Code

2.1 Notice

Since the whole code is relatively long, we do not show the whole code in this part (You can see the whole code in my GitHub). Instead, we just show some key parts and give necessary explanations to describe the idea analyzed in **Part 1**.

2.2 How to View and Execute Code

This project has been open source on **GitHub**. You can access my code in `src`.

Project URL: https://github.com/Maystern/SUSTech_cpp_Project05_a-class-for-matrices.git。

The project is recommended to run in the `Ubuntu` environment.

1. Use command `git clone https://github.com/Maystern/SUSTech_cpp_Project05_a-class-for-matrices.git` to download the item into current directory.
2. In current directory, use command `cd SUSTech_cpp_Project05_a-class-for-matrices` to access thee root directory of item.
3. Execute the `sh Run_MatrixClassTest.sh` command in the project root directory. After the command is executed, the original code file will be automatically compiled with `cmake` in `./build`, and the generated binary executable `matrixClassTest` located in the `./build/bin` directory will be moved to the project root directory.
4. You can use `./matrixClassTest` in the item directory to run each example code.
5. Your can modify the `CMakeLists.txt` in `src` to choose whether turn on instruction set optimization or compilation optimization.

2.3 Class Temple: Data

The definition of `Data` class temple is shown in **Part 1**. Here, we show its implements.

```cpp
template <class type>
Data<type>::Data(size_t length) {
  this->data = new type[length]{};
  this->refcount = 1;
}
template <class type>
Data<type>::~Data() {
  delete[] this->data;
  this->data = NULL;
}
template <class type>
type *Data<type>::getDataValue() {
  return this->data;
}
template <class type>
void Data<type>::addRefCount() {
  this->refcount++;
}
template <class type>
void Data<type>::minusRefCount() {
  this->refcount--;
}
template <class type>
size_t Data<type>::getRefCount() {
  return this->refcount;
}
```

- In order to store matrix data, the **constructor** attempts to use `new[length]{}` for creating a space in the computer memory. You need to pass in the amount of memory space (input variable `length`) that needs to be created.
- The destructor here is to delete the space of stored matrix data. Here the pointer `data` points to the data. We use `delete[]` here to meet `new[]`.

- We use the identifier variable `refcount` uniquely identified in the address space to indicate how many matrices refer to this space. When the matrix is added or decreased, the value of this variable will change. This is the key to memory management.
- Since we need to change or get the value of `refcount` outside the class `Data`, we should implement some member functions: `addRefCount()`、 `minusRefCount`、 `getRefCount()`.
- Since we need to use the data in overloading the operator `+`, `-`, `*`, `~` in matrix calculation, the member function `getDataValue()` is included.

2.4 Class Temple: Matrix

The definition of `Matrix` class temple is already shown in **Part 1**. Please move to **1.2.3 Matrix Class Template Structure** part for details.

Here we will show the reasons why we design another class temple `Data`:

- Since we overload the assignment operator `=` by making the two matrices share the same space of data (as known as **soft copy**), we must consider how to delete this space just once, avoiding the problems of **double free**. At first, we want to declare the pointer pointing to matrix data as smart pointer `shared_ptr`. However, although `shared_ptr` allows many pointers to point the same space and delete the space just once automatically, it must be defined to point to the new memory rather than the memory of data which has already existed. This is problematic.
- After that, we try to use the idea of `shared_ptr` rather than directly use it. We create an extra variable of type `int` in the memory of matrix storage data to store how many matrices this memory is shared by. For the encapsulation of the program, we use an additional class `data` to describe the data space. If there are other pointers pointing to this space with **assignment operator** `=` or **copy constructor** called, the counter will add **1**, and if some pointers arrives at the end of their life circle, the count will minus 1. The memory of matrix data will be released if the counter is 1.

2.4.1 Matrix Constructor Code

```cpp
template <class type>
Matrix<type>::Matrix(size_t rows, size_t cols, size_t channels) {
  count++;
#ifdef _DEBUG_
  mat_id = count;
  std::cout << "Matrix<type>::Matrix(size_t rows, size_t cols, size_t "
               "channels), Matrix #"
            << mat_id << std::endl;
#endif
  this->rows = rows;
  this->cols = cols;
  this->channels = channels;
  this->size = rows * cols;
  dataptr = new Data<type>(rows * cols * channels);
  memset(dataptr->getDataValue(), 0, sizeof(type) * rows * cols * channels);
}
```

2.4.2 Matrix Destructor Code

```cpp
template <class type>
Matrix<type>::~Matrix() {
  count--;
#ifdef _DEBUG_
  std::cout << "~Matrix(), Matrix #" << mat_id << std::endl;
#endif
  if (dataptr == NULL) {
#ifdef _DEBUG_
    std::cout << "Matrix #" << mat_id << " data removed" << std::endl;
#endif
  }
  if (dataptr != NULL && dataptr->getRefCount() == 1) {
#ifdef _DEBUG_
    std::cout << "Matrix #" << mat_id << " data removed" << std::endl;
#endif
    delete dataptr;
    dataptr = NULL;
  }
  if (dataptr != NULL && dataptr->getRefCount() > 1) {
#ifdef _DEBUG_
    std::cout << "Matrix #" << mat_id << " ptr removed" << std::endl;
#endif
    dataptr->minusRefCount();
  }
}
```

2.4.3 Matrix Assignment Copy Code

```cpp
template <class type>
Matrix<type> &Matrix<type>::operator=(const Matrix &mat_copy) {
#ifdef _DEBUG_
  std::cout << "Matrix<type> & Matrix<type>::operator=(const Matrix & mat_copy)"
            << std::endl;
#endif
  if (this == &mat_copy) return *this;
  if (dataptr != NULL) {
    dataptr->minusRefCount();
    delete dataptr;
  }
  rows = mat_copy.rows;
  cols = mat_copy.cols;
  size = mat_copy.size;
  channels = mat_copy.channels;
  if (mat_copy.dataptr) {
    dataptr = mat_copy.dataptr;
    dataptr->addRefCount();
  }
  return *this;
}
```

2.4.4 Read Matrix from input Stream

The input and output are highly **linked**, which means that the matrix printed by the output stream can be read by the input stream, which is very beneficial to file operation.

```
template <class type>
std::istream &operator>>(std::istream &is, Matrix<type> &matrix) {
  std::string str;
  getline(is, str);
  str.erase(std::remove_if(str.begin(), str.end(), ::isspace), str.end());
  if (str == "Empty Matrix") {
    matrix = Matrix<type>();
    return is;
  }
  size_t channels = 0, rows = 0, cols = 1;
  for (size_t i = 0; i < str.length() - 1; i++)
    if (str[i] == '[' && str[i + 1] == '[') channels++;
  for (size_t i = 2; i < str.length(); i++) {
    if (str[i] == ']') break;
    if (str[i] == ',') cols++;
  }
  for (size_t i = 1; i < str.length() - 1; i++) {
    if (str[i] == ']' && str[i + 1] == ']') break;
    if (str[i] == '[') rows++;
  }
  matrix = Matrix<type>(rows, cols, channels);
  type *data = matrix.getDataPtr()->getDataValue();
  for (size_t i = 0; i < str.length(); i++)
    if (str[i] != '[' && str[i] != ']' && str[i] != ',') {
      size_t j = i;
      while (j < str.length() && str[j] != '[' && str[j] != ']' &&
             str[j] != ',') {
        j++;
      }
      j--;
      *data = string_to_type<type>(str.substr(i, j - i + 1).c_str());
      data++;
      i = j;
    }
  return is;
}
```

The code below base on these functions which we defined (more details please see the whole code) :

- `template <class type> type string_to_type(const char *str)`: Enter a string and output a value of type.

2.4.5 Write Matrix to Output Stream

The input and output are highly **linked**, which means that the matrix printed by the output stream can be read by the input stream, which is very beneficial to file operation.

```cpp
template <class type>
std::ostream &operator<<(std::ostream &os, Matrix<type> &matrix) {
  if (matrix.getSize() == 0) {
    os << "Empty Matrix";
    return os;
  }
  for (size_t t = 0; t < matrix.getChannels(); t++) {
    os << "[";
    for (size_t i = 0; i < matrix.getRows(); i++) {
      os << "[";
      for (size_t j = 0; j < matrix.getCols(); j++) {
        os << matrix.getDataPtr()->getDataValue()[t * matrix.getSize() +
                                                  i * matrix.getCols() + j];
        if (j != matrix.getCols() - 1) os << ", ";
      }
      if (i != matrix.getRows() - 1)
        os << "],";
      else
        os << "]";
    }
    if (t != matrix.getChannels() - 1)
      os << "],";
    else
      os << "]";
  }
  return os;
}
```

3 Result and Verification

In this part, we will make test cases. For convenience, we just test the data type `float`.

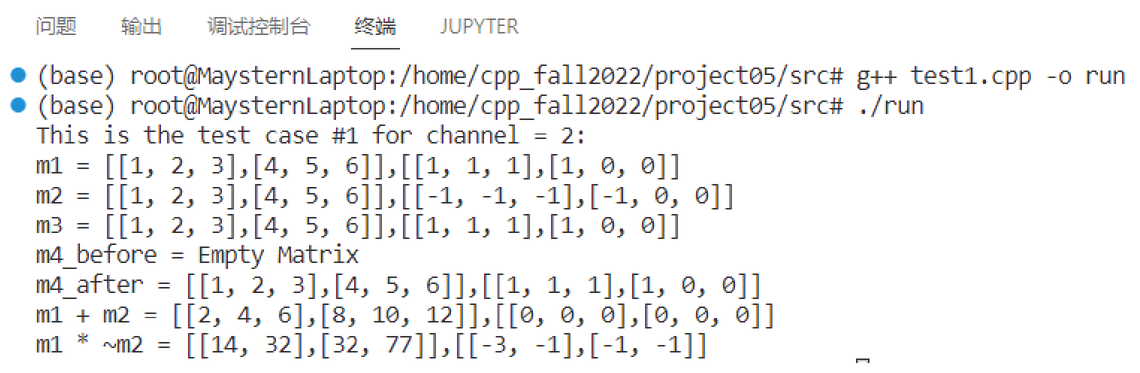
3.1 Test Case #1: Channel = 2

Test the member variable *channel* = 2, and for convenience, the remaining test cases are tested with *channel* = 1. You can see the whole code in `test1.cpp`.

```cpp
// #define _DEBUG_
std::cout << "This is the test case #1 for channel = 2:" << std::endl;
float m1_values[12] = {1.f,2.f,3.f,4.f,5.f,6.f,1.f,1.f,1.f,1.f};
Matrix<float> m1(2, 3, 2, m1_values);
```

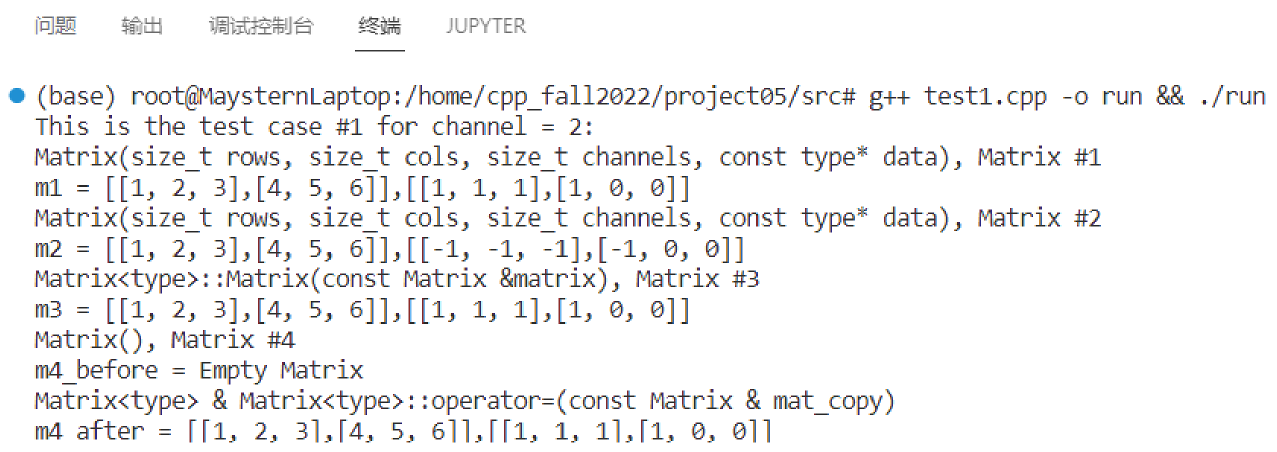
```cpp
std::cout << "m1 = " << m1 << std::endl;

float m2_values[12] = {1.f,2.f,3.f,4.f,5.f,6.f,-1.f,-1.f,-1.f,-1.f};
Matrix<float> m2(2, 3, 2, m2_values);
std::cout << "m2 = " << m2 << std::endl;

Matrix<float>m3 = m1;
std::cout << "m3 = " << m3 << std::endl;

Matrix<float>m4;
std::cout << "m4_before = " << m4 << std::endl;
m4 = m1;
std::cout << "m4_after = " << m4 << std::endl;

Matrix<float>m_add = m1 + m2;
std::cout << "m1 + m2 = " << m_add << std::endl;

Matrix<float>m_mul = m1 * ~m2;
std::cout << "m1 * ~m2 = " << m_mul << std::endl;
```

If we not use `_DEBUG_`, the output should be:



```
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test1.cpp -o run
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run
This is the test case #1 for channel = 2:
m1 = [[1, 2, 3],[4, 5, 6]],[[1, 1, 1],[1, 0, 0]]
m2 = [[1, 2, 3],[4, 5, 6]],[[-1, -1, -1],[-1, 0, 0]]
m3 = [[1, 2, 3],[4, 5, 6]],[[1, 1, 1],[1, 0, 0]]
m4_before = Empty Matrix
m4_after = [[1, 2, 3],[4, 5, 6]],[[1, 1, 1],[1, 0, 0]]
m1 + m2 = [[2, 4, 6],[8, 10, 12]],[[0, 0, 0],[0, 0, 0]]
m1 * ~m2 = [[14, 32],[32, 77]],[[-3, -1],[-1, -1]]
```

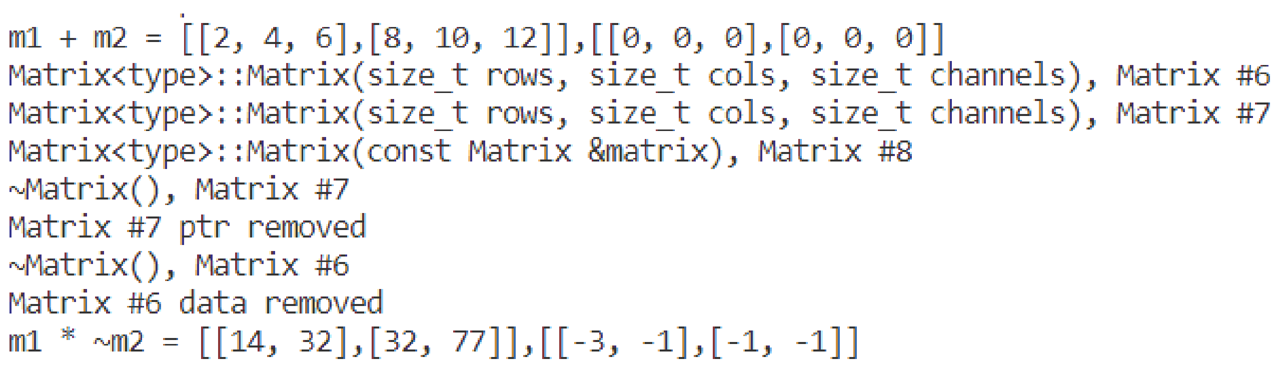
If we use `_DEBUG_`, more details should be seen:



```
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test1.cpp -o run && ./run
This is the test case #1 for channel = 2:
Matrix(size_t rows, size_t cols, size_t channels, const type* data), Matrix #1
m1 = [[1, 2, 3],[4, 5, 6]],[[1, 1, 1],[1, 0, 0]]
Matrix(size_t rows, size_t cols, size_t channels, const type* data), Matrix #2
m2 = [[1, 2, 3],[4, 5, 6]],[[-1, -1, -1],[-1, 0, 0]]
Matrix<type>::Matrix(const Matrix &matrix), Matrix #3
m3 = [[1, 2, 3],[4, 5, 6]],[[1, 1, 1],[1, 0, 0]]
Matrix(), Matrix #4
m4_before = Empty Matrix
Matrix<type> & Matrix<type>::operator=(const Matrix & mat_copy)
m4 after = [[1, 2, 3],[4, 5, 6]],[[1, 1, 1],[1, 0, 0]]
```

- As you can see, there are 2 size 2×3 matrices in one class object $m1$. It is equivalent that one element of matrix $m1$ stores two float numbers.

- The constructors, copy constructor and the overloading function of assignment operator `=` work well for $channel = 2$.

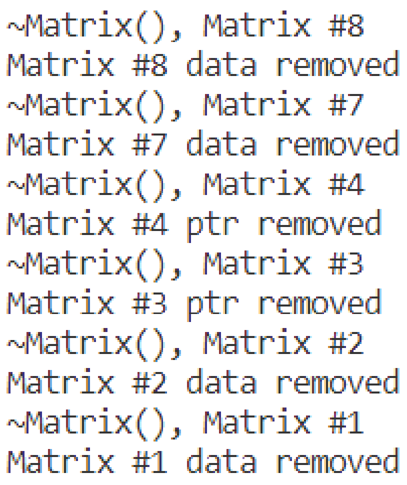


```
m1 + m2 = [[2, 4, 6],[8, 10, 12]],[[0, 0, 0],[0, 0, 0]]
Matrix<type>::Matrix(size_t rows, size_t cols, size_t channels), Matrix #6
Matrix<type>::Matrix(size_t rows, size_t cols, size_t channels), Matrix #7
Matrix<type>::Matrix(const Matrix &matrix), Matrix #8
~Matrix(), Matrix #7
Matrix #7 ptr removed
~Matrix(), Matrix #6
Matrix #6 data removed
m1 * ~m2 = [[14, 32],[32, 77]],[[-3, -1],[-1, -1]]
```

- Here we define the operators (Plus `+`, minus `-`, multiplication `*`) for matrices **channel by channel**. As you can see, the addition and the multiplication of matrices work well in this way,
- i.e. $m_1[1] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, m_1[2] = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}, m_1[1] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, m_1[2] = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 0 & 0 \end{bmatrix}$

  $m_1 + m_2 = (m_1[1] + m_2[1], m1[2] + m2[2]) \quad m_1 \times m_2^T = (m_1[1] \times m_2[1]^T, m_1[2] \times m_2[2]^T)$

  You can easily check that the result is consistent with that in the above screenshot.



```
~Matrix(), Matrix #8
Matrix #8 data removed
~Matrix(), Matrix #7
Matrix #7 data removed
~Matrix(), Matrix #4
Matrix #4 ptr removed
~Matrix(), Matrix #3
Matrix #3 ptr removed
~Matrix(), Matrix #2
Matrix #2 data removed
~Matrix(), Matrix #1
Matrix #1 data removed
```

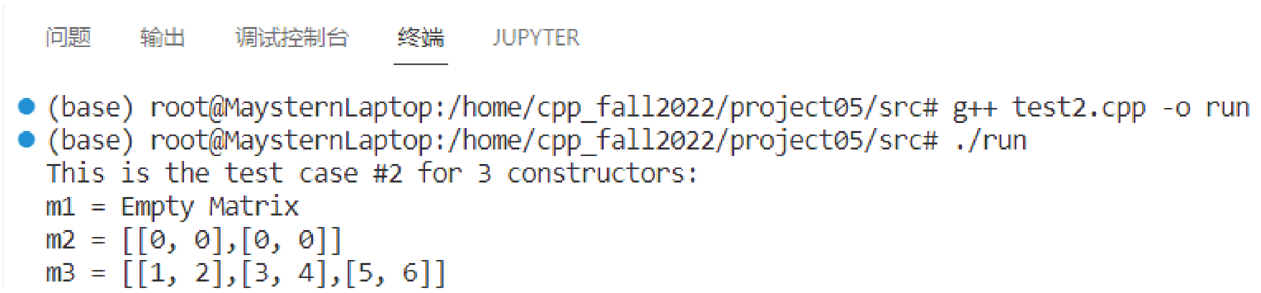
- The memory management is successful. We call copy constructor to copy m_1 (**Matrix #1**) to m_3 (**Matrix #3**) **softly**, and call overloading function of assignment operator `=` to assign m_1 (**Matrix #1**) to m_4 (**Matrix #4**) **softly**. Hence, there are **3** pointers pointing to the matrix data of m_1. As you can see, we just delete the pointer of m_3 and m_4, and finally release the space of the data of m_1.

3.2 Test Case #2: 3 Constructor

Test the **3** constructor for matrix creation. You can see the whole code in `test2.cpp`.

```cpp
// #define _DEBUG_
std::cout << "This is the test case #2 for 3 constructors: " << std::endl;
Matrix<float>m1;
std::cout << "m1 = " << m1 << std::endl;
Matrix<float>m2(2, 2, 1);
std::cout << "m2 = " << m2 << std::endl;
float values[6] = {1, 2, 3, 4, 5, 6};
Matrix<float>m3(3, 2, 1, values);
std::cout << "m3 = " << m3 << std::endl;
```

If we not use `_DEBUG_`, the output should be:

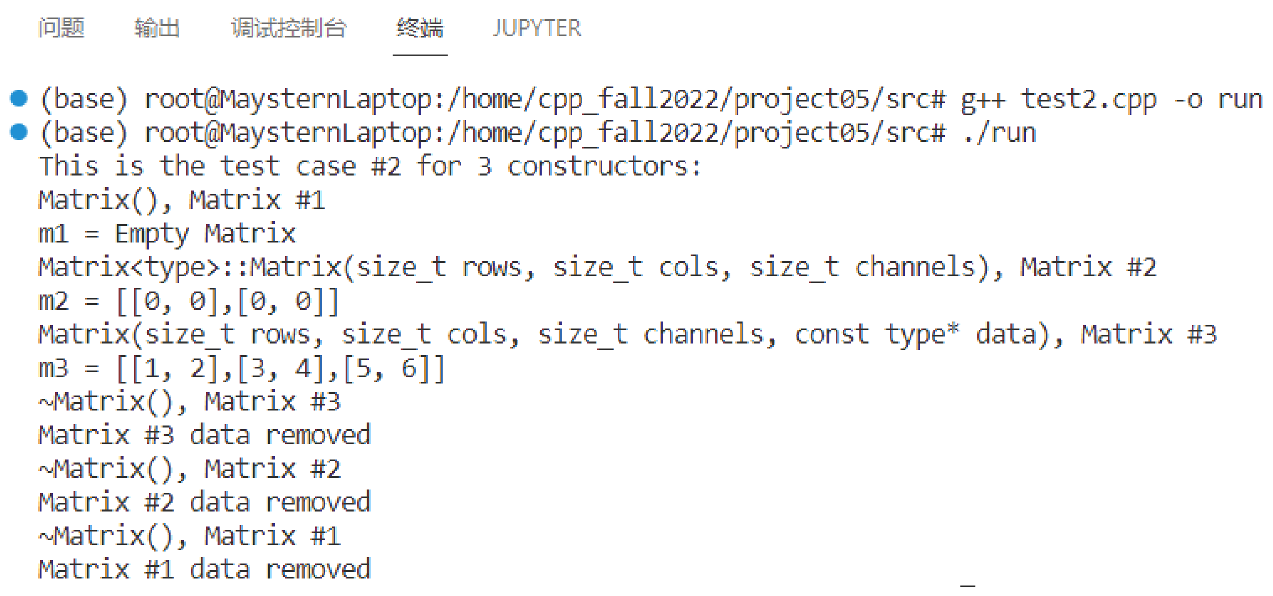


```
● (base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test2.cpp -o run
● (base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run
  This is the test case #2 for 3 constructors:
  m1 = Empty Matrix
  m2 = [[0, 0],[0, 0]]
  m3 = [[1, 2],[3, 4],[5, 6]]
```

- We overloaded the default constructor of the matrix, so we can specifically identify an empty matrix.

```cpp
Matrix<type>::Matrix() {
  dataptr = NULL;
  count++;
  size = 0;
#ifdef _DEBUG_
  mat_id = count;
  std::cout << "Matrix(), Matrix #" << mat_id << std::endl;
#endif
}
// ...
template <class type>
std::ostream &operator<<(std::ostream &os, Matrix<type> &matrix) {
  if (matrix.getSize() == 0) {
    os << "Empty Matrix";
    return os;
  }
    //...
}
```

If we use `_DEBUG_`, more details should be seen:



```
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test2.cpp -o run
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run
This is the test case #2 for 3 constructors:
Matrix(), Matrix #1
m1 = Empty Matrix
Matrix<type>::Matrix(size_t rows, size_t cols, size_t channels), Matrix #2
m2 = [[0, 0],[0, 0]]
Matrix(size_t rows, size_t cols, size_t channels, const type* data), Matrix #3
m3 = [[1, 2],[3, 4],[5, 6]]
~Matrix(), Matrix #3
Matrix #3 data removed
~Matrix(), Matrix #2
Matrix #2 data removed
~Matrix(), Matrix #1
Matrix #1 data removed
```

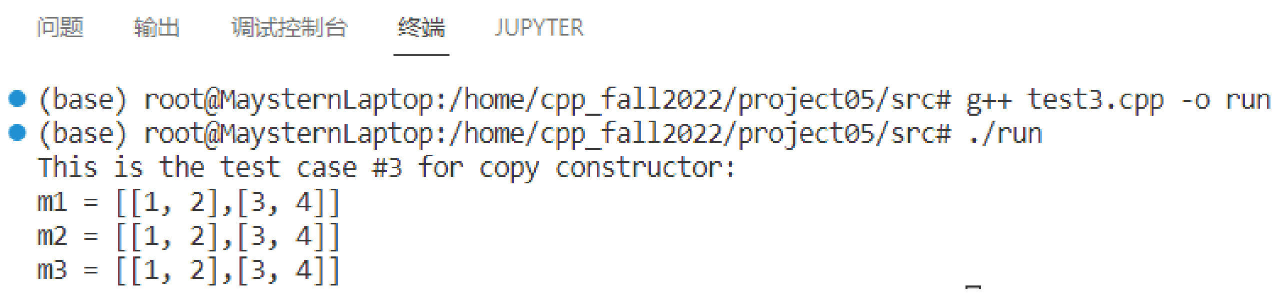
- As you can see, 3 constructors are called correspondingly to create the objects.

3.3 Test Case #3: Copy Constructor

Test the copy constructor for memory management. You can see the whole code in `test3.cpp`.

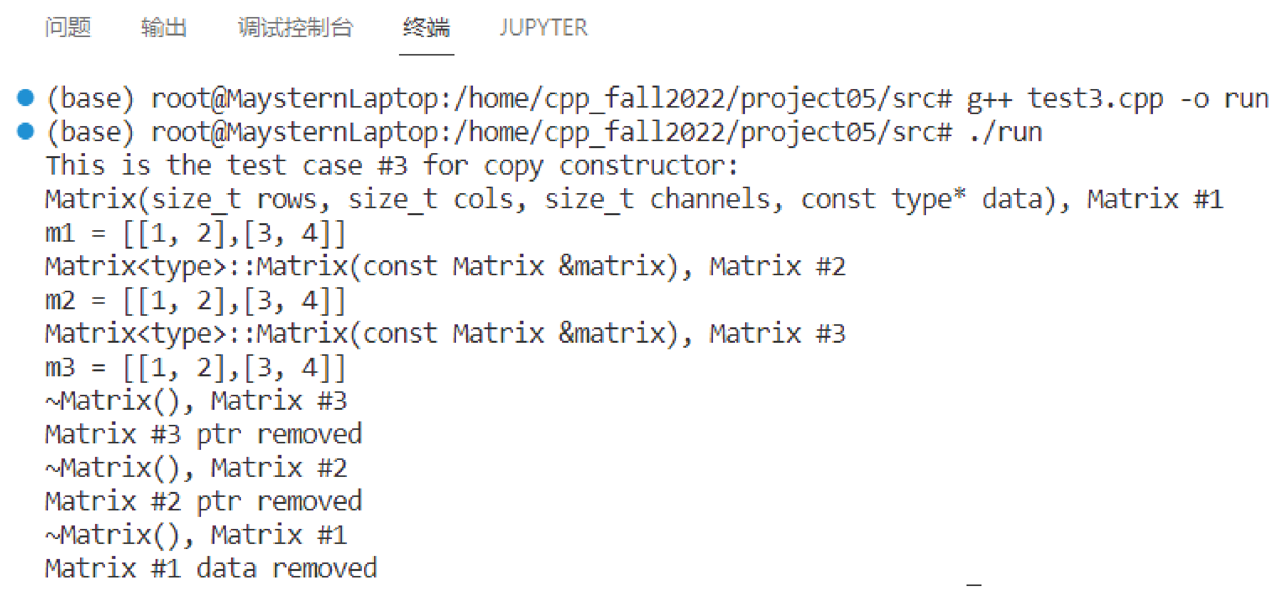
```cpp
// #define _DEBUG_
std::cout << "This is the test case #3 for copy constructor: " << std::endl;
float values[4] = {1, 2, 3, 4};
Matrix<float> m1(2, 2, 1, values);
std::cout << "m1 = " << m1 << std::endl;
Matrix<float>m2(m1);
std::cout << "m2 = " << m2 << std::endl;
Matrix<float>m3 = m1;
std::cout << "m3 = " << m3 << std::endl;
```

If we not use `_DEBUG_`, the output should be:



```
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test3.cpp -o run
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run
This is the test case #3 for copy constructor:
m1 = [[1, 2],[3, 4]]
m2 = [[1, 2],[3, 4]]
m3 = [[1, 2],[3, 4]]
```

If we use `_DEBUG_`, more details should be seen:



```
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test3.cpp -o run
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run
This is the test case #3 for copy constructor:
Matrix(size_t rows, size_t cols, size_t channels, const type* data), Matrix #1
m1 = [[1, 2],[3, 4]]
Matrix<type>::Matrix(const Matrix &matrix), Matrix #2
m2 = [[1, 2],[3, 4]]
Matrix<type>::Matrix(const Matrix &matrix), Matrix #3
m3 = [[1, 2],[3, 4]]
~Matrix(), Matrix #3
Matrix #3 ptr removed
~Matrix(), Matrix #2
Matrix #2 ptr removed
~Matrix(), Matrix #1
Matrix #1 data removed
```

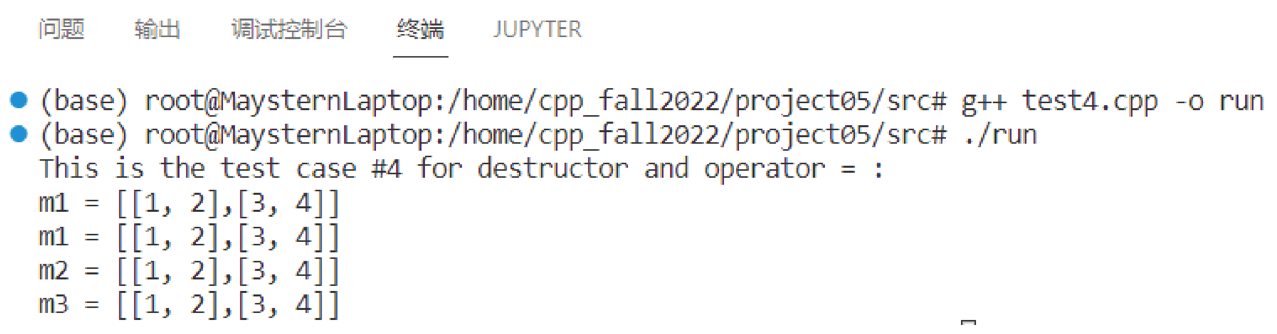
- As you can see, the creation of matrices m_2, m_3 call the copy constructor in class `Matrix`.
- Also, the memory management is successful.

3.4 Test Case #4: Destructor, Assignment operator, and soft copy

Test the destructor and the assignment operator `=` overloading function for memory management (**soft copy**). You can see the whole code in `test4.cpp`.

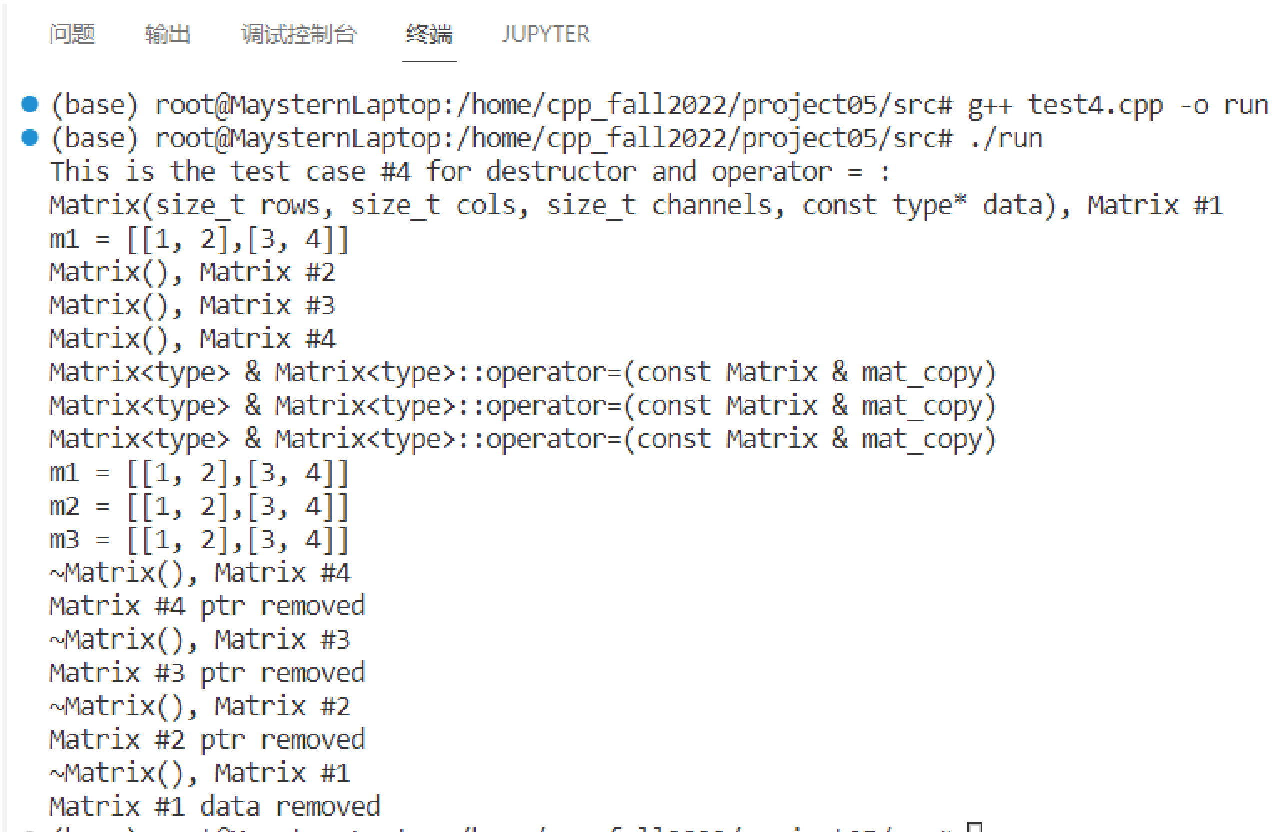
```cpp
// #define _DEBUG_
std::cout << "This is the test case #4 for destructor and operator = : " << std::endl;
float values[6] = {1, 2, 3, 4, 5, 6};
Matrix<float> m0(2, 2, 1, values);
std::cout << "m1 = " << m0 << std::endl;
Matrix<float>m1, m2, m3;
m1 = m2 = m3 = m0;
std::cout << "m1 = " << m1 << std::endl;
std::cout << "m2 = " << m2 << std::endl;
std::cout << "m3 = " << m3 << std::endl;
```

If we not use `_DEBUG_`, the output should be:



```
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test4.cpp -o run
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run
This is the test case #4 for destructor and operator = :
m1 = [[1, 2],[3, 4]]
m1 = [[1, 2],[3, 4]]
m2 = [[1, 2],[3, 4]]
m3 = [[1, 2],[3, 4]]
```

If we use `_DEBUG_`, more details should be seen:



```
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test4.cpp -o run
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run
This is the test case #4 for destructor and operator = :
Matrix(size_t rows, size_t cols, size_t channels, const type* data), Matrix #1
m1 = [[1, 2],[3, 4]]
Matrix(), Matrix #2
Matrix(), Matrix #3
Matrix(), Matrix #4
Matrix<type> & Matrix<type>::operator=(const Matrix & mat_copy)
Matrix<type> & Matrix<type>::operator=(const Matrix & mat_copy)
Matrix<type> & Matrix<type>::operator=(const Matrix & mat_copy)
m1 = [[1, 2],[3, 4]]
m2 = [[1, 2],[3, 4]]
m3 = [[1, 2],[3, 4]]
~Matrix(), Matrix #4
Matrix #4 ptr removed
~Matrix(), Matrix #3
Matrix #3 ptr removed
~Matrix(), Matrix #2
Matrix #2 ptr removed
~Matrix(), Matrix #1
Matrix #1 data removed
```

- As you can see, the code `m1 = m2 = m3 = m0` calls the assignment operator `=` overloading functions 3 times.
- This actually achieves **soft copy** since 4 pointers point to the same space of data of matrix m_0.
- The memory management is successful since we remove the first 3 pointers and just delete the space of data **once**.

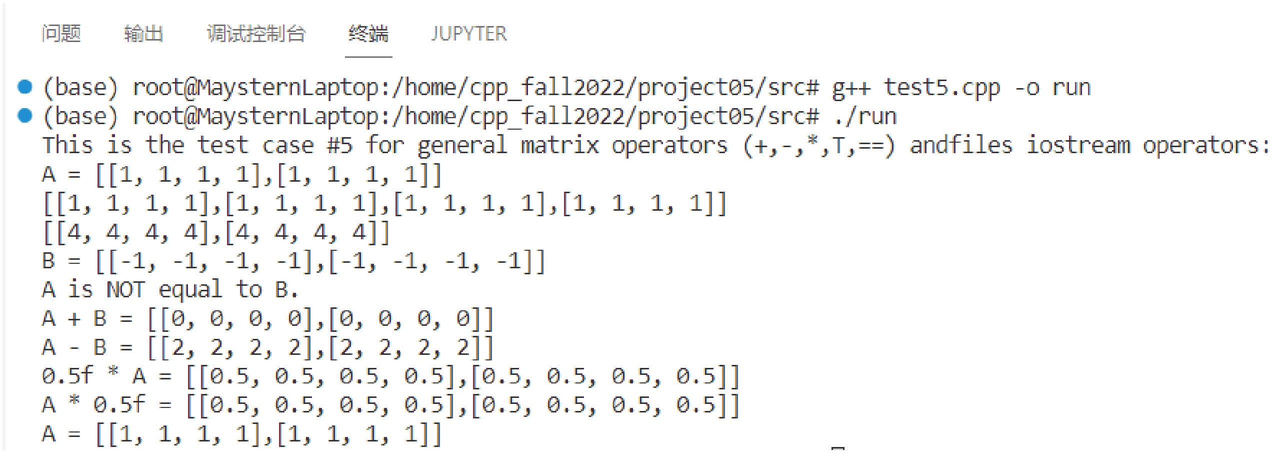
3.5 Test Case#5: General Matrix Operators and Files IO stream Operators

Test the overloading functions for general matrix operators () `+`, `-`, `*`, `~`) and files IO stream operators (`<<`, `>>`). You can see the whole code in `test5.cpp`.

```cpp
// #define _DEBUG_
std::cout << "This is the test case #5 for general matrix operators (+,-,*,T,==)
andfiles iostream operators:" << std::endl;
float A_Data[8] = {1, 1, 1, 1, 1, 1, 1, 1};
Matrix<float> A = Matrix(2, 4, 1, A_Data);
std::cout << "A = " << A << std::endl;
std::ifstream  fin("ROI_original.txt");
Matrix<float> A_test;
fin >> A_test;
fin.close();
std::cout << A_test << std::endl;

std::ofstream ofile;
ofile.open("out_test_5.txt");
Matrix<float>m_mul = A * ~A_test;
ofile << m_mul;
ofile.close();
```

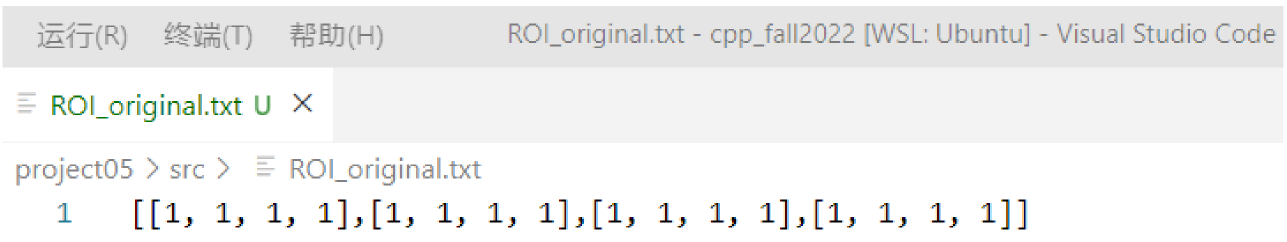
```cpp
std::cout << m_mul << std::endl;

float B_Data[8] = {-1, -1, -1, -1, -1, -1, -1, -1};
Matrix<float>B = Matrix(2, 4, 1, B_Data);
std::cout << "B = " << B << std::endl;

bool AB = (A == B);
if (AB == 0) {
    std::cout << "A is NOT equal to B." << std::endl;
}

Matrix<float> add = A + B;
std::cout << "A + B = " << add << std::endl;
Matrix<float> minus = A - B;
std::cout << "A - B = " <<  minus << std::endl;
Matrix<float> val_mul_matrix = 0.5f * A;
std::cout << "0.5f * A = " << val_mul_matrix << std::endl;
Matrix<float> matrix_mul_val = A * 0.5f;
std::cout << "A * 0.5f = " << matrix_mul_val << std::endl;
std::cout << "A = " << A << std::endl;
```

If we not use `_DEBUG_`, the output should be:



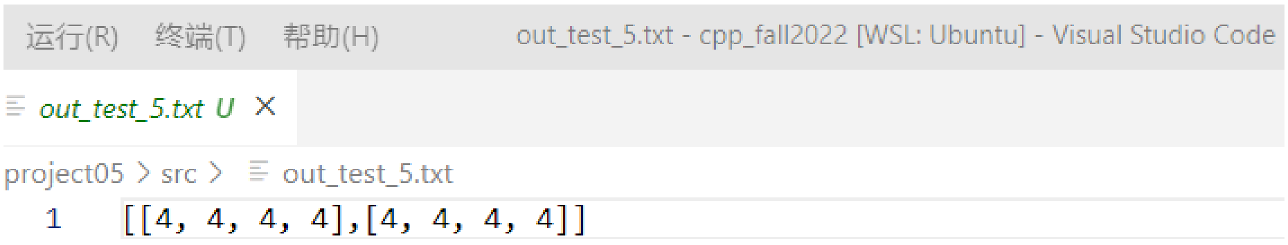
```
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test5.cpp -o run
(base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run
This is the test case #5 for general matrix operators (+,-,*,T,==) andfiles iostream operators:
A = [[1, 1, 1, 1],[1, 1, 1, 1]]
[[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1]]
[[4, 4, 4, 4],[4, 4, 4, 4]]
B = [[-1, -1, -1, -1],[-1, -1, -1, -1]]
A is NOT equal to B.
A + B = [[0, 0, 0, 0],[0, 0, 0, 0]]
A - B = [[2, 2, 2, 2],[2, 2, 2, 2]]
0.5f * A = [[0.5, 0.5, 0.5, 0.5],[0.5, 0.5, 0.5, 0.5]]
A * 0.5f = [[0.5, 0.5, 0.5, 0.5],[0.5, 0.5, 0.5, 0.5]]
A = [[1, 1, 1, 1],[1, 1, 1, 1]]
```

- A is a 2×4 matrix with all entries to be 1. B is a 2×4 matrix with all entries to be -1. A_{test} is loaded from `ROI_original.txt`, which is a 4×4 matrix with all entries to be 1.


运行(R)
终端(T)
帮助(H)
ROI_original.txt - cpp_fall2022 [WSL: Ubuntu] - Visual Studio Code
ROI_original.txt U
project05 > src > ROI_original.txt

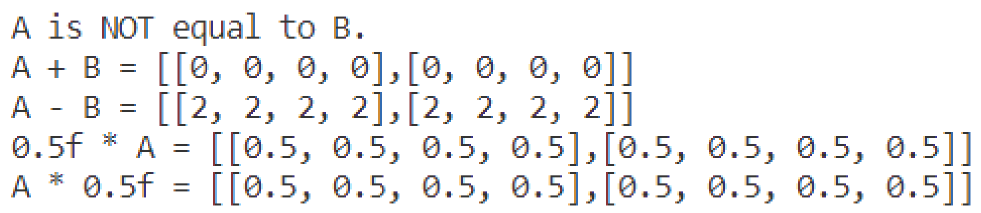
```
[[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1]]
```

- As you can see, A_{test} is read from file and output onto screen successfully.


运行(R)
终端(T)
帮助(H)
out_test_5.txt - cpp_fall2022 [WSL: Ubuntu] - Visual Studio Code
out_test_5.txt U
project05 > src > out_test_5.txt

```
[[4, 4, 4, 4],[4, 4, 4, 4]]
```

- We test the matrix multiplication with general `i-k-j` loop for $A \times A_{test}^T$, which also test the transpose of matrix.



```
A is NOT equal to B.
A + B = [[0, 0, 0, 0],[0, 0, 0, 0]]
A - B = [[2, 2, 2, 2],[2, 2, 2, 2]]
0.5f * A = [[0.5, 0.5, 0.5, 0.5],[0.5, 0.5, 0.5, 0.5]]
A * 0.5f = [[0.5, 0.5, 0.5, 0.5],[0.5, 0.5, 0.5, 0.5]]
```

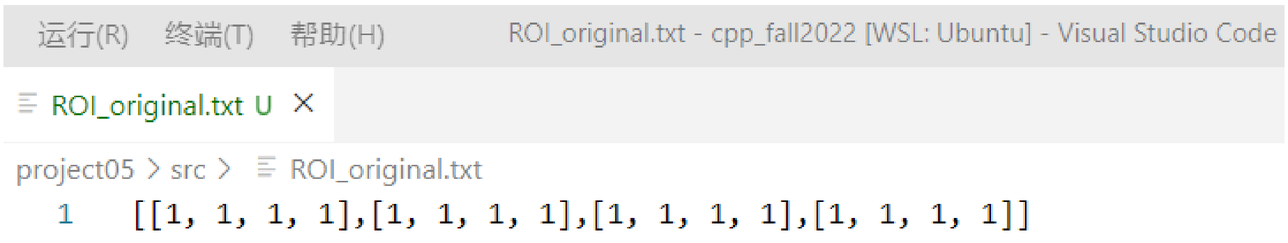
- We overload the operator `+`, `-`, `*` with the object returned rather than reference. In the operator overloading function. we create a temporary matrix to store the result of operations of matrices and then return this temporary object. This is confirmed by the calling of constructors and destructor during the program.
- We test the overloading function if operator `==` by testing `A == B`. Since all entries of A are 1, while all entries of B are -1, they are *NOT* equal even if they have the same $rows, cols, size, channel$.

3.6 Test Case#6: ROI with Soft Copy

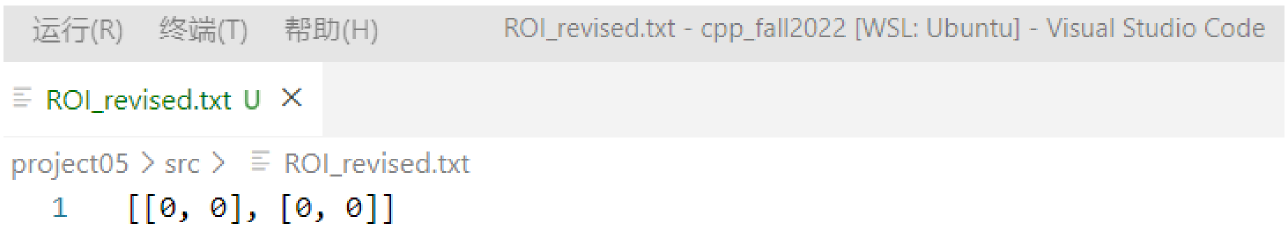
Test the member functions for **ROI** with **soft copy**. You can see the whole code in `test6.cpp`.

```
// #define _DEBUG_
std::cout << "This is the test case #5 for ROI:" << std::endl;
std::ifstream fin("ROI_original.txt");
Matrix<float> ROI_original;
fin >> ROI_original;
fin.close();
std::cout << "ROI_original = " << ROI_original << std::endl;
ROI_original.submat(6, 9, "ROI_revised.txt");
std::ofstream ofile("ROI_final.txt");
ofile << ROI_original;
ofile.close();
std::cout << "ROI_final = " << ROI_original << std::endl;
```

- In this case, the ROI is the center 2×2 sub-matrix, 2 given indices are 6 and 9, restricting the location of *ROI*.

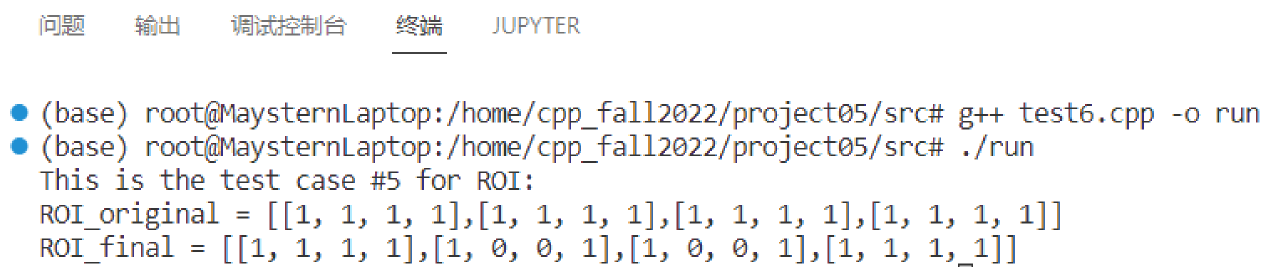

运行(R)
终端(T)
帮助(H)
ROI_original.txt - cpp_fall2022 [WSL: Ubuntu] - Visual Studio Code
ROI_original.txt U
project05 > src > ROI_original.txt

```
[[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1]]
```

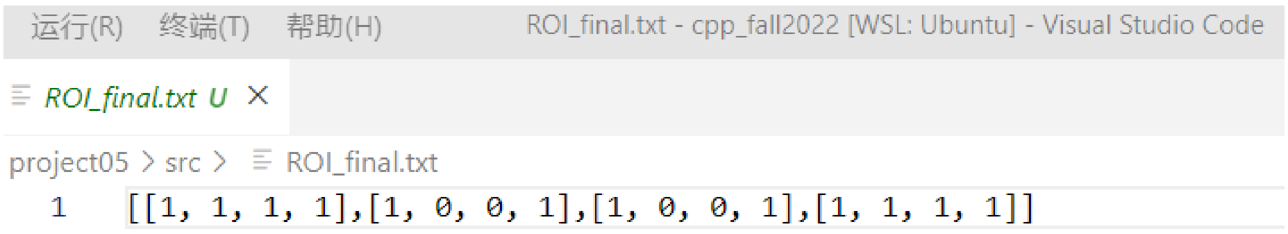

运行(R) 终端(T) 帮助(H) ROI_revised.txt - cpp_fall2022 [WSL: Ubuntu] - Visual Studio Code
ROI_revised.txt U
project05 > src > ROI_revised.txt

```
1  [[0, 0], [0, 0]]
```

- As you can see, the data in ROI is change with revised data in `ROI_revised.txt`.



```
● (base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test6.cpp -o run
● (base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run
  This is the test case #5 for ROI:
  ROI_original = [[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1],[1, 1, 1, 1]]
  ROI_final = [[1, 1, 1, 1],[1, 0, 0, 1],[1, 0, 0, 1],[1, 1, 1, 1]]
```


运行(R) 终端(T) 帮助(H) ROI_final.txt - cpp_fall2022 [WSL: Ubuntu] - Visual Studio Code
ROI_final.txt U
project05 > src > ROI_final.txt

```
1  [[1, 1, 1, 1],[1, 0, 0, 1],[1, 0, 0, 1],[1, 1, 1, 1]]
```

3.7 Test Case#7: General Loop Matrix Multiplication in Different Platforms

Test the overloading functions of operator `*` for general `i-k-j` loop matrix multiplication in different platforms (x86, ARM). You can see the whole code in `test7.cpp`.

The general `i-k-j` loop matrix multiplication code is as follows:

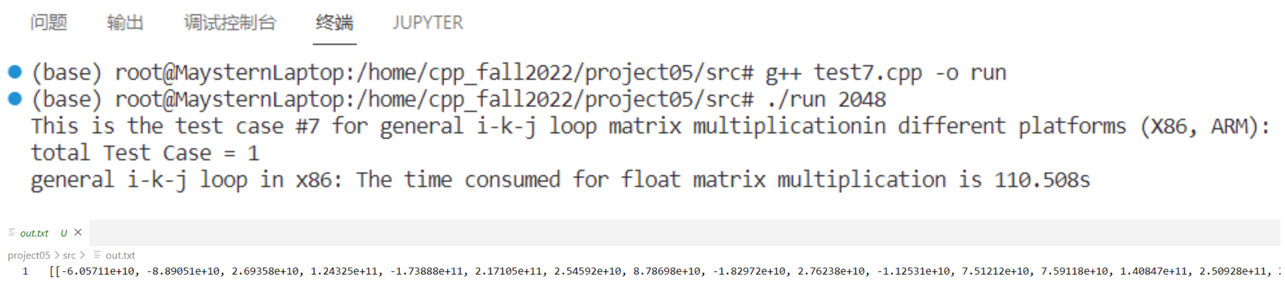
```cpp
template <class type>
Matrix<type> Matrix<type>::operator*(Matrix<type> matrix) {
  if (cols != matrix.getRows() || channels != matrix.getChannels()) {
    return *this;
  }
  clock_t start, end;
  start = clock();
  Matrix result = Matrix(rows, matrix.getCols(), channels);
  for (size_t t = 0; t < result.getChannels(); t++) {
      for (size_t i = 0; i < rows; i++) {
          for (size_t k = 0; k < matrix.getRows(); k++) {
              for (size_t j = 0; j < matrix.getCols(); j++) {
                  result.getDataPtr()->getDataValue()[t * result.getSize() +
                  i * matrix.getCols() + j] += dataptr->getDataValue()[t *
                  size + i * cols + k] *
                  matrix.getDataPtr()->getDataValue()[t * matrix.getSize() +
                  k * matrix.getCols() + j];
              }
          }
      }
  }
  end = clock();
  std::cout << "general i-k-j loop: The time consumed for matrix multiplication is " << double(end - start) / CLOCKS_PER_SEC << "s" << std::endl;
  return result;
}
```

The test code is as follows:

```cpp
float random(float l, float r) {
    float tmp = 1.0 * rand() / RAND_MAX;
    return (r - l) * tmp + l;
}
int main(int argc, char **argv) {
    std::ofstream fout("out.txt");
    std::cout << "This is the test case #7 for general i-k-j loop matrix multiplicationin different platforms (X86, ARM):" << std::endl;
    srand(time(0));
    printf("total Test Case = %d\n", argc - 1);
    for (int i = 1; i < argc; i++) {
        size_t matrixSize = atoi(*(argv + i));
        float *data1 = (float *) malloc(sizeof(float) * matrixSize * matrixSize);
        float *data2 = (float *) malloc(sizeof(float) * matrixSize * matrixSize);
        for (int j = 0; j < matrixSize * matrixSize; j++) {
            data1[j] = random(-1e5, 1e5);
            data2[j] = random(-1e5, 1e5);
        }
        Matrix<float> A = Matrix<float>(matrixSize, matrixSize, 1, data1);
        Matrix<float> B = Matrix<float>(matrixSize, matrixSize, 1, data2);
        Matrix<float> C = A * B;
        fout << C;
        free(data1);
        free(data2);
    }
    return 0;
}
```

This is the test result on x86 platform (**My Personal Computer**):

Item	**Infomation**
Framework	x86_64
CPU	1 8-core-CPU， AMD Ryzen 7 5800H with Radeon Graphics @ 3.20 GHz
Memory	16 GB
Cache	512 KB (L1 Cache) 4.0MB (L2 Cache) 16.0 KB (L3 Cache)

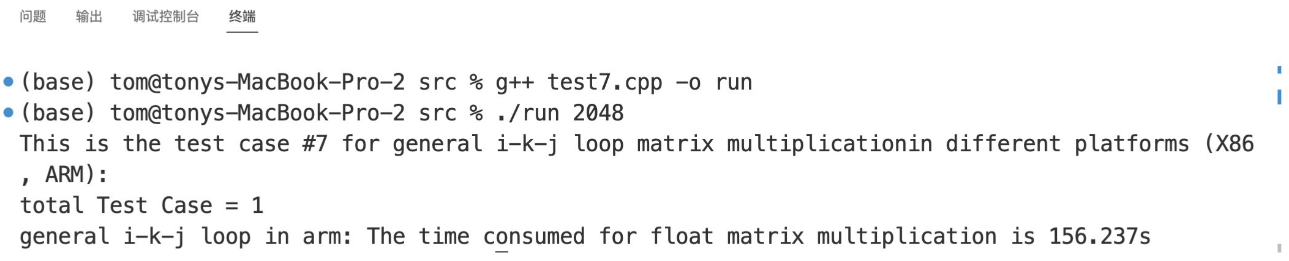


```
● (base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test7.cpp -o run
● (base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run 2048
  This is the test case #7 for general i-k-j loop matrix multiplicationin different platforms (X86, ARM):
  total Test Case = 1
  general i-k-j loop in x86: The time consumed for float matrix multiplication is 110.508s
```

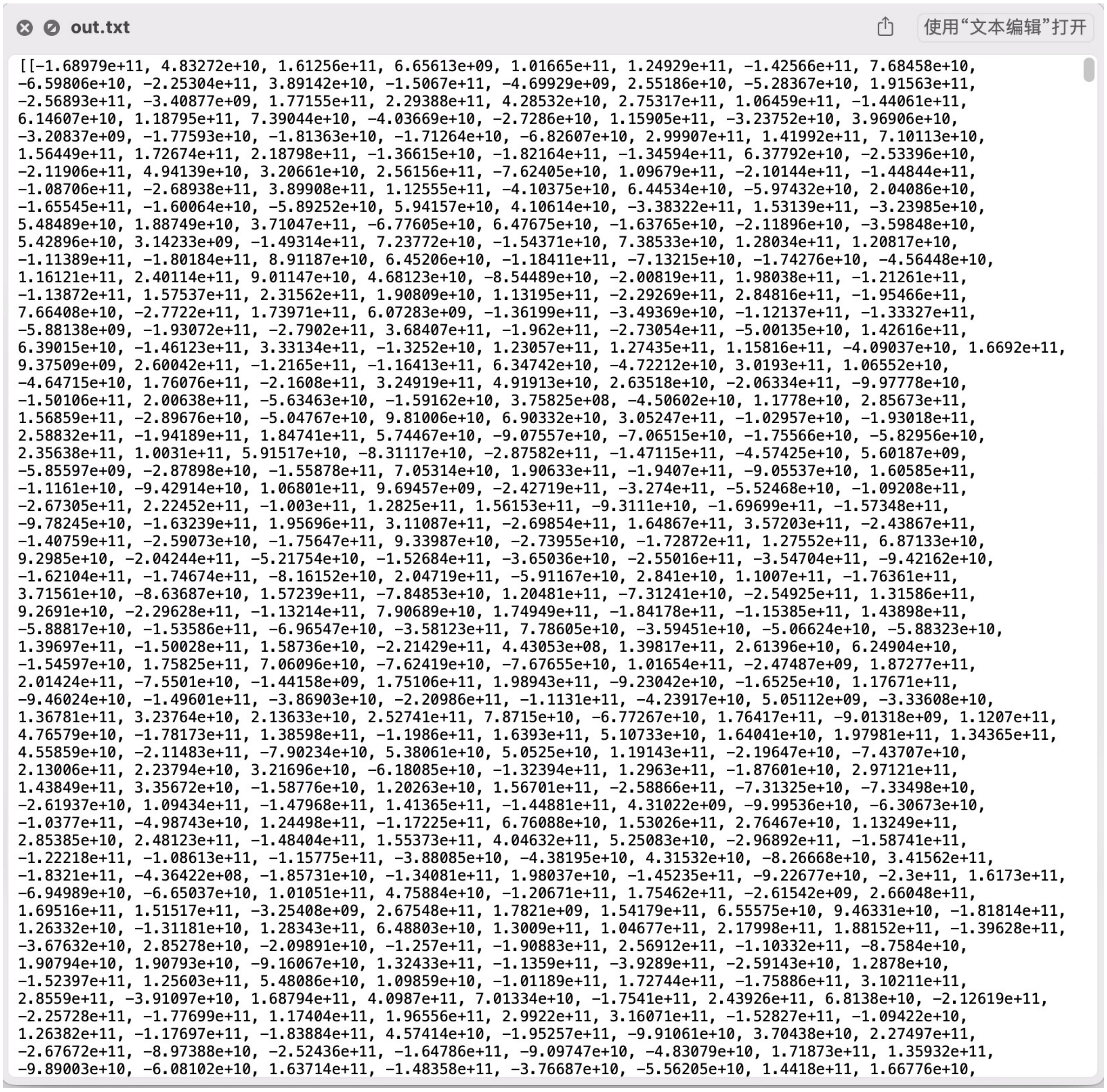
```
[[-6.05711e+10, -8.89051e+10, 2.69358e+10, 1.24325e+11, -1.73888e+11, 2.17105e+11, 2.54592e+10, 8.78698e+10, -1.82972e+10, 2.76238e+10, -1.12531e+10, 7.51212e+10, 7.59118e+10, 1.40847e+11, 2.50928e+11,
```

This is the test result on arm platform (**My Classmate Yucheng Zhu's Personal Computer**):

Item	**Infomation**
Framework	arm_64
CPU	1 8-core-CPU， Apple M1 Pro @ 3.35 GHz
Memory	16 GB
Cache	192 KB (L1 Cache) 4.0MB (L2 Cache)



```
● (base) tom@tonys-MacBook-Pro-2 src % g++ test7.cpp -o run
● (base) tom@tonys-MacBook-Pro-2 src % ./run 2048
  This is the test case #7 for general i-k-j loop matrix multiplicationin different platforms (X86
  , ARM):
  total Test Case = 1
  general i-k-j loop in arm: The time consumed for float matrix multiplication is 156.237s
```



```
[[-1.68979e+11, 4.83272e+10, 1.61256e+11, 6.65613e+09, 1.01665e+11, 1.24929e+11, -1.42566e+11, 7.68458e+10,
-6.59806e+10, -2.25304e+11, 3.89142e+10, -1.5067e+11, -4.69929e+09, 2.55186e+10, -5.28367e+10, 1.91563e+11,
-2.56893e+11, -3.40877e+09, 1.77155e+11, 2.29388e+11, 4.28532e+10, 2.75317e+11, 1.06459e+11, -1.44061e+11,
6.14607e+10, 1.18795e+11, 7.39044e+10, -4.03669e+10, -2.7286e+10, 1.15905e+11, -3.23752e+10, 3.96906e+10,
-3.20837e+09, -1.77593e+10, -1.81363e+10, -1.71264e+10, -6.82607e+10, 2.99907e+11, 1.41992e+11, 7.10113e+10,
1.56449e+11, 1.72674e+11, 2.18798e+11, -1.36615e+10, -1.82164e+11, -1.34594e+11, 6.37792e+10, -2.53396e+10,
-2.11906e+11, 4.94139e+10, 3.20661e+10, 2.56156e+11, -7.62405e+10, 1.09679e+11, -2.10144e+11, -1.44844e+11,
-1.08706e+11, -2.68938e+11, 3.89908e+11, 1.12555e+11, -4.10375e+10, 6.44534e+10, -5.97432e+10, 2.04086e+10,
-1.65545e+11, -1.60064e+10, -5.89252e+10, 5.94157e+10, 4.10614e+10, -3.38322e+11, 1.53139e+11, -3.23985e+10,
5.48489e+10, 1.88749e+10, 3.71047e+11, -6.77605e+10, 6.47675e+10, -1.63765e+10, -2.11896e+10, -3.59848e+10,
5.42896e+10, 3.14233e+09, -1.49314e+11, 7.23772e+10, -1.54371e+10, 7.38533e+10, 1.28034e+11, 1.20817e+10,
-1.11389e+11, -1.80184e+11, 8.91187e+10, 6.45206e+10, -1.18411e+11, -7.13215e+10, -1.74276e+10, -4.56448e+10,
1.16121e+11, 2.40114e+11, 9.01147e+10, 4.68123e+10, -8.54489e+10, -2.00819e+11, 1.98038e+11, -1.21261e+11,
-1.13872e+11, 1.57537e+11, 2.31562e+11, 1.90809e+10, 1.13195e+11, -2.29269e+11, 2.84816e+11, -1.95466e+11,
7.66408e+10, -2.7722e+11, 1.73971e+11, 6.07283e+09, -1.36199e+11, -3.49369e+10, -1.12137e+11, -1.33327e+11,
-5.88138e+09, -1.93072e+11, -2.7902e+11, 3.68407e+11, -1.962e+11, -2.73054e+11, -5.00135e+10, 1.42616e+11,
6.39015e+10, -1.46123e+11, 3.33134e+11, -1.3252e+10, 1.23057e+11, 1.27435e+11, 1.15816e+11, -4.09037e+10, 1.6692e+11,
9.37509e+09, 2.60042e+11, -1.2165e+11, -1.16413e+11, 6.34742e+10, -4.72212e+10, 3.0193e+11, 1.06552e+10,
-4.64715e+10, 1.76076e+11, -2.1608e+11, 3.24919e+11, 4.91913e+10, 2.63518e+10, -2.06334e+11, -9.97778e+10,
-1.50106e+11, 2.00638e+11, -5.63463e+10, -1.59162e+10, 3.75825e+08, -4.50602e+10, 1.1778e+10, 2.85673e+11,
1.56859e+11, -2.89676e+10, -5.04767e+10, 9.81006e+10, 6.90332e+10, 3.05247e+11, -1.02957e+10, -1.93018e+11,
2.58832e+11, -1.94189e+11, 1.84741e+11, 5.74467e+10, -9.07557e+10, -7.06515e+10, -1.75566e+10, -5.82956e+10,
2.35638e+11, 1.0031e+11, 5.91517e+10, -8.31117e+10, -2.87582e+11, -1.47115e+11, -4.57425e+10, 5.60187e+09,
-5.85597e+09, -2.87898e+10, -1.55878e+11, 7.05314e+10, 1.90633e+11, -1.9407e+11, -9.05537e+10, 1.60585e+11,
-1.1161e+10, -9.42914e+10, 1.06801e+11, 9.69457e+09, -2.42719e+11, -3.274e+11, -5.52468e+10, -1.09208e+11,
-2.67305e+11, 2.22452e+11, -1.003e+11, 1.2825e+11, 1.56153e+11, -9.3111e+10, -1.69699e+11, -1.57348e+11,
-9.78245e+10, -1.63239e+11, 1.95696e+11, 3.11087e+11, -2.69854e+11, 1.64867e+11, 3.57203e+11, -2.43867e+11,
-1.40759e+11, -2.59073e+10, -1.75647e+11, 9.33987e+10, -2.73955e+10, -1.72872e+11, 1.27552e+11, 6.87133e+10,
9.2985e+10, -2.04244e+11, -5.21754e+10, -1.52684e+11, -3.65036e+10, -2.55016e+11, -3.54704e+11, -9.42162e+10,
-1.62104e+11, -1.74674e+11, -8.16152e+10, 2.04719e+11, -5.91167e+10, 2.841e+10, 1.1007e+11, -1.76361e+11,
3.71561e+10, -8.63687e+10, 1.57239e+11, -7.84853e+10, 1.20481e+11, -7.31241e+10, -2.54925e+11, 1.31586e+11,
9.2691e+10, -2.29628e+11, -1.13214e+11, 7.90689e+10, 1.74949e+11, -1.84178e+11, -1.15385e+11, 1.43898e+11,
-5.88817e+10, -1.53586e+11, -6.96547e+10, -3.58123e+11, 7.78605e+10, -3.59451e+10, -5.06624e+10, -5.88323e+10,
1.39697e+11, -1.50028e+11, 1.58736e+10, -2.21429e+11, 4.43053e+08, 1.39817e+11, 2.61396e+10, 6.24904e+10,
-1.54597e+10, 1.75825e+11, 7.06096e+10, -7.62419e+10, -7.67655e+10, 1.01654e+11, -2.47487e+09, 1.87277e+11,
2.01424e+11, -7.5501e+10, -1.44158e+09, 1.75106e+11, 1.98943e+11, -9.23042e+10, -1.6525e+10, 1.17671e+11,
-9.46024e+10, -1.49601e+11, -3.86903e+10, -2.20986e+11, -1.1131e+11, -4.23917e+10, 5.05112e+09, -3.33608e+10,
1.36781e+11, 3.23764e+10, 2.13633e+10, 2.52741e+11, 7.8715e+10, -6.77267e+10, 1.76417e+11, -9.01318e+09, 1.1207e+11,
4.76579e+10, -1.78173e+11, 1.38598e+11, -1.1986e+11, 1.6393e+11, 5.10733e+10, 1.64041e+10, 1.97981e+11, 1.34365e+11,
4.55859e+10, -2.11483e+11, -7.90234e+10, 5.38061e+10, 5.0525e+10, 1.19143e+11, -2.19647e+10, -7.43707e+10,
2.13006e+11, 2.23794e+10, 3.21696e+10, -6.18085e+10, -1.32394e+11, 1.2963e+11, -1.87601e+10, 2.97121e+11,
1.43849e+11, 3.35672e+10, -1.58776e+10, 1.20263e+10, 1.56701e+11, -2.58866e+11, -7.31325e+10, -7.33498e+10,
-2.61937e+10, 1.09434e+11, -1.47968e+11, 1.41365e+11, -1.44881e+11, 4.31022e+09, -9.99536e+10, -6.30673e+10,
-1.0377e+11, -4.98743e+10, 1.24498e+11, -1.17225e+11, 6.76088e+10, 1.53026e+11, 2.76467e+10, 1.13249e+11,
2.85385e+10, 2.48123e+11, -1.48404e+11, 1.55373e+11, 4.04632e+11, 5.25083e+10, -2.96892e+11, -1.58741e+11,
-1.22218e+11, -1.08613e+11, -1.15775e+11, -3.88085e+10, -4.38195e+10, 4.31532e+10, -8.26668e+10, 3.41562e+11,
-1.8321e+11, -4.36422e+08, -1.85731e+10, -1.34081e+11, 1.98037e+10, -1.45235e+11, -9.22677e+10, -2.3e+11, 1.6173e+11,
-6.94989e+10, -6.65037e+10, 1.01051e+11, 4.75884e+10, -1.20671e+11, 1.75462e+11, -2.61542e+09, 2.66048e+11,
1.69516e+11, 1.51517e+11, -3.25408e+09, 2.67548e+11, 1.7821e+09, 1.54179e+11, 6.55575e+10, 9.46331e+10, -1.81814e+11,
1.26332e+10, -1.31181e+10, 1.28343e+11, 6.48803e+10, 1.3009e+11, 1.04677e+11, 2.17998e+11, 1.88152e+11, -1.39628e+11,
-3.67632e+10, 2.85278e+10, -2.09891e+10, -1.257e+11, -1.90883e+11, 2.56912e+11, -1.10332e+11, -8.7584e+10,
1.90794e+10, 1.90793e+10, -9.16067e+10, 1.32433e+11, -1.1359e+11, -3.9289e+11, -2.59143e+10, 1.2878e+10,
-1.52397e+11, 1.25603e+11, 5.48086e+10, 1.09859e+10, -1.01189e+11, 1.72744e+11, -1.75886e+11, 3.10211e+11,
2.8559e+11, -3.91097e+10, 1.68794e+11, 4.0987e+11, 7.01334e+10, -1.7541e+11, 2.43926e+11, 6.8138e+10, -2.12619e+11,
-2.25728e+11, -1.77699e+11, 1.17404e+11, 1.96556e+11, 2.9922e+11, 3.16071e+11, -1.52827e+11, -1.09422e+10,
1.26382e+11, -1.17697e+11, -1.83884e+11, 4.57414e+10, -1.95257e+11, -9.91061e+10, 3.70438e+10, 2.27497e+11,
-2.67672e+11, -8.97388e+10, -2.52436e+11, -1.64786e+11, -9.09747e+10, -4.83079e+10, 1.71873e+11, 1.35932e+11,
-9.89003e+10, -6.08102e+10, 1.63714e+11, -1.48358e+11, -3.76687e+10, -5.56205e+10, 1.4418e+11, 1.66776e+10,
```

If we use tables to present data:

Platform	Result
x86_64 (My Personal Computer)	110.508s
arm_64 (Yucheng Zhu's Personal Computer)	156.237s

- As we can see: The efficiency of general `i-k-j` loop matrix multiplication using arm platform and x86 platform is roughly the **same**.
- And x86 platform is **slightly faster** than x86 platform.
- Also, we can conclude that this matrix class with the general `i-k-j` loop matrix multiplication overloaded can work in two different platforms (x86, ARM).
- In other words, our program implements **cross platform**.

3.8 Test Case#8: (Matrix Multiplication with Optimization Applied

Test the overloading functions of operator for matrix multiplication with optimization applied (x86 SSE, ARM Neon) You can see the whole code in `test8.cpp`.

The matrix multiplication with optimization applied code is as follows:

```cpp
template <class type>
Matrix<type> Matrix<type>::operator*(Matrix<type> matrix) {
  if (cols != matrix.getRows() || channels != matrix.getChannels()) {
    return *this;
  }
  clock_t start, end;
  start = clock();
  Matrix result = Matrix(rows, matrix.getCols(), channels);
  type *v1 = new type[matrix.getRows()]{};
  type *v2 = new type[matrix.getRows()]{};
  for (size_t t = 0; t < result.getChannels(); t++) {
    for (size_t i = 0; i < rows; i++) {
      for (size_t k = 0; k <= matrix.getRows(); k++) {
        v1[k] = dataptr->getDataValue()[t * size + i * cols + k];
      }
      for (size_t j = 0; j < matrix.getCols(); j++) {
        for (size_t k = 0; k <= matrix.getRows(); k++) {
          v2[k] = matrix.getDataPtr()->getDataValue()[t * matrix.getSize() +
                                                      k * matrix.getCols() + j];
        }
        vectorMul(result.getDataPtr()->getDataValue() + t * result.getSize() +
                      i * matrix.getCols() + j,
                  v1, v2, matrix.getRows());
      }
    }
  }
  delete[] v1;
  delete[] v2;
  end = clock();
  std::cout << "general i-k-j loop: The time consumed for matrix multiplication is " << double(end - start) / CLOCKS_PER_SEC << "s" << std::endl;
  return result;
}
```

The code below is based on `void vectorMul(type *dest, type *v1, type *v2, size_t len)` which is used for **row column vector dot multiplication** in matrix multiplication (The Code is shown below, please move to part **1.3.3 Matrix Multiplication**)

The test code is as follows:

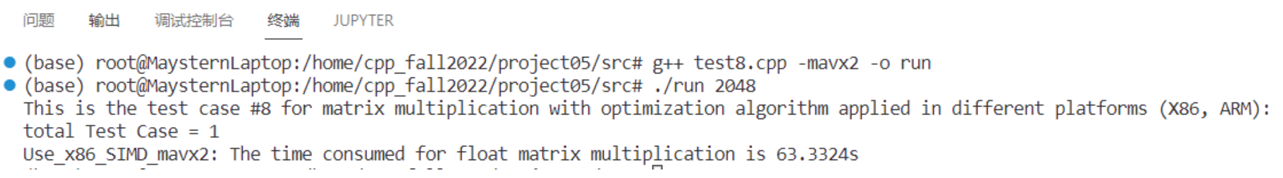
```cpp
// #define _DEBUG_
```

```cpp

#define Use_x86_SIMD_mavx2
// #define Use_x86_SIMD_mavx512f
// #define Use_arm_NEON

float random(float l, float r) {
    float tmp = 1.0 * rand() / RAND_MAX;
    return (r - l) * tmp + l;
}
int main(int argc, char **argv) {
    std::ofstream fout("out.txt");
    std::cout << "This is the test case #8 for matrix multiplication with optimization algorithm applied in different platforms (X86, ARM):" << std::endl;
    srand(time(0));
    printf("total Test Case = %d\n", argc - 1);
    for (int i = 1; i < argc; i++) {
        size_t matrixSize = atoi(*(argv + i));
        float *data1 = (float *) malloc(sizeof(float) * matrixSize * matrixSize);
        float *data2 = (float *) malloc(sizeof(float) * matrixSize * matrixSize);
        for (int j = 0; j < matrixSize * matrixSize; j++) {
            data1[j] = random(-1e5, 1e5);
            data2[j] = random(-1e5, 1e5);
        }
        Matrix<float> A = Matrix<float>(matrixSize, matrixSize, 1, data1);
        Matrix<float> B = Matrix<float>(matrixSize, matrixSize, 1, data2);
        Matrix<float> C = A * B;
        fout << C;
        free(data1);
        free(data2);
    }
    return 0;
}
```

This is the test result on x86 platform (**My Personal Computer**):

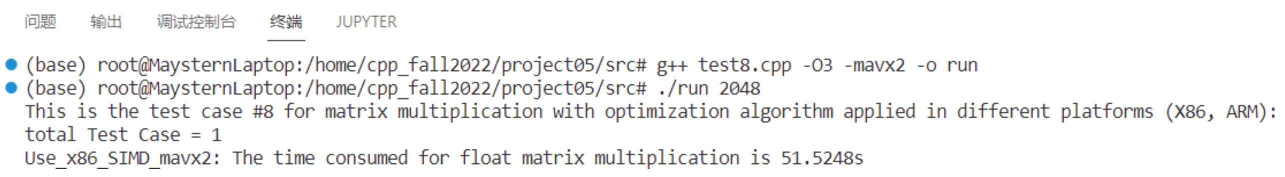
Item	**Infomation**
Framework	x86_64
CPU	1 8-core-CPU， AMD Ryzen 7 5800H with Radeon Graphics @ 3.20 GHz
Memory	16 GB
Cache	512 KB (L1 Cache) 4.0MB (L2 Cache) 16.0 KB (L3 Cache)

- Disable `-O3` Optimization



```
● (base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test8.cpp -mavx2 -o run
● (base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run 2048
  This is the test case #8 for matrix multiplication with optimization algorithm applied in different platforms (X86, ARM):
  total Test Case = 1
  Use_x86_SIMD_mavx2: The time consumed for float matrix multiplication is 63.3324s
```

- Enable `-O3` Optimization

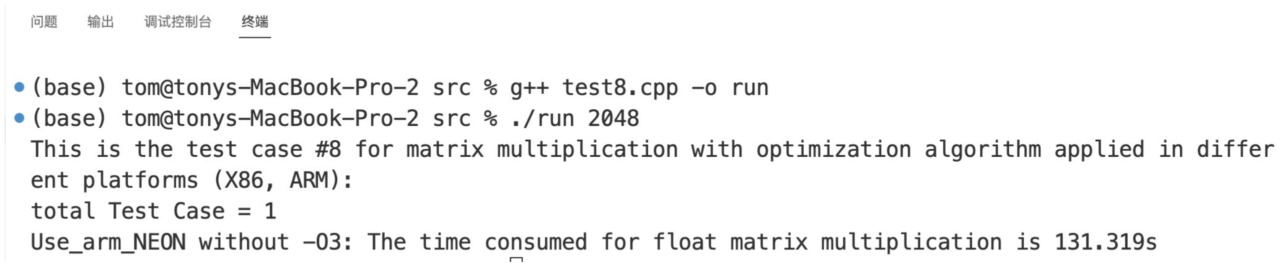


```
● (base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# g++ test8.cpp -O3 -mavx2 -o run
● (base) root@MaysternLaptop:/home/cpp_fall2022/project05/src# ./run 2048
  This is the test case #8 for matrix multiplication with optimization algorithm applied in different platforms (X86, ARM):
  total Test Case = 1
  Use_x86_SIMD_mavx2: The time consumed for float matrix multiplication is 51.5248s
```

This is the test result on arm platform (**My Classmate Yucheng Zhu's Personal Computer**):

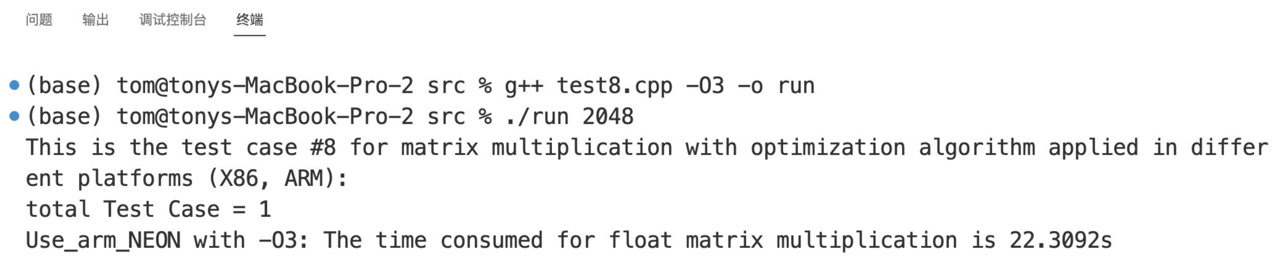
Item	Infomation
Framework	arm_64
CPU	1 8-core-CPU， Apple M1 Pro @ 3.35 GHz
Memory	16 GB
Cache	192 KB (L1 Cache) 4.0MB (L2 Cache)

- Disable `-O3` Optimization



```
● (base) tom@tonys-MacBook-Pro-2 src % g++ test8.cpp -o run
● (base) tom@tonys-MacBook-Pro-2 src % ./run 2048
  This is the test case #8 for matrix multiplication with optimization algorithm applied in differ
  ent platforms (X86, ARM):
  total Test Case = 1
  Use_arm_NEON without -O3: The time consumed for float matrix multiplication is 131.319s
```

- Enable `-O3` Optimization



```
● (base) tom@tonys-MacBook-Pro-2 src % g++ test8.cpp -O3 -o run
● (base) tom@tonys-MacBook-Pro-2 src % ./run 2048
  This is the test case #8 for matrix multiplication with optimization algorithm applied in differ
  ent platforms (X86, ARM):
  total Test Case = 1
  Use_arm_NEON with -O3: The time consumed for float matrix multiplication is 22.3092s
```

If we use tables to present data:

Platform	Optimization	Result
x86_64 (My Personal Computer)	None	110.508s
arm_64 (Yucheng Zhu's Personal Computer)	None	156.237s
x86_64 (My Personal Computer)	SIMD (without `-O3`)	63.332s
arm_64 (Yucheng Zhu's Personal Computer)	NEON (without `-O3`)	131.319s
x86_64 (My Personal Computer)	SIMD (with `-O3`)	51.525s
arm_64 (Yucheng Zhu's Personal Computer)	NEON (with `-O3`)	22..309s

In the comparison **before and after optimization**, you can see that

- In both platforms(x86, ARM), using instruction set optimization accelerates matrix multiplication.
- Using `-O3` optimization will greatly speed up the instruction set optimization matrix multiplication.

In the comparison of **different platforms,** you can see that

- When the `-O3` optimization is not used, the matrix multiplication speed of the x86 platform and the optimization degree using the instruction set are **slightly greater** than those of the arm platform
- When using the `-O3` optimization, the matrix multiplication speed of the x86 platform and the optimization degree using the instruction set are **far inferior** to the arm platform

If we compare this project (**Project 5**) with the previous one (**Project 4**)

- This project (**Project 5**) is a little bit slower than that in previous one (**Project 4**). It is because we add the member variable **channel** in this project, so there are actually 4 loops to compute the matrix multiplication, which is much more time-consuming. Moreover, to realize **soft copy**, we design the class `Data`. Then we need use pointers twice to get the matrix data, which is really complex and has high cost.

4 Difficulties and Solutions

4.1 Soft Copy

The idea is borrowed from **smart pointers:** `shared_ptr`, which is adding one variable to count the pointers pointing to the space of matrix data. Then rewrite the **destructor**, **constructor** and **copy constructor** of `Matrix` to release the memory just **once** to avoid **double free**.

4.2 Check Memory Management

Using `_ debug_` Macro definition turns debug mode on or off. When the debug mode is enabled, debugging information will be output when running the code in the library. We add the output of `ptr removed` or `data removed` to represent that the destructor removes the pointer or the matrix data, respectively. To identify whose pointer or data is removed, we add one member variable `mat_id`. Moreover, we add one `static` variable `count` and one member variable `mat_id` to manage the existing matrices in class `Matrix`. In this case, matrices that have been created will have an unique number. so we cam identify the matrices easily.

4.3 ROI with Soft Copy

Using the idea of **soft copy**, we should not storage the copied data in a new memory. Instead, we should revise the data in **ROI** through pointers. When we use `for-loop` to move the **pointer**, its access to memory is continues. However, the **ROI** may not be continues space. Here, we still let the pointer move in the `for-loop`. When the pointer moves into the **ROI**, we change the data.

4.4 Channel

We still store data in a way similar to a **one-dimensional array**, and add two member variables `channels` and `size` when storing the matrix. The one-dimensional array stores the data in **each channel** in **order of row priority**. We can access the data at a certain position `(channel_id, row_id, col_id)` in the array through simple calculation.

4.5 Advanced Ideas

- In **Project 1** and **Project 2**, We have defined **large integer classes** and **large real number classes**, and overloaded some **high-performance calculations**. We can use them in the matrix class to **avoid** the **loss of precision** when using the **default type**.
- In **Project 3** and **Project 4**, We use **C** language to define the **structure** of a matrix, and carry out different **optimizations** to explore ways to greatly accelerate matrix multiplication.

We applied these past projects to this project and got **wonderful** results.

As I said at the beginning of this project report, "By the way, we can see that all the 5 projects are **closely related**."

5 Summary

Because the report contains some **code** and running **screenshots**, this report looks longer than usual.

I hope you will not get annoyed.

So far, all the projects in this semester have been completed. Your courses are very **exciting** and **fascinating**, and the projects are also very **interesting** and **challenging** (I carefully complete all 5 projects, and I really gain a lot from them). I am very willing to learn your courses. Thank you for your guidance in **Fall semester 2022**.

Best wishes to you!

Jiacheng Luo

18th December, 2022