CS205 C/C++ Program Design-Assignment 4

姓名:刘旭坤 学号:11912823

要求与实现:

1. (20 points) The class Matrix should be designed for a matrix with float elements. It should contain some members to describe its number of rows and columns, data and others. Please use private keyword to protect the data inside

创建`Matrix`类并使用`private`

```
class Matrix
{
private:
    int x = 0, y = 0;
    float *matrix = nullptr;
    int *num = nullptr;
```

2. (35 points) Please design some constructors, destructor, operator =, operator << and some others as what Lecture Notes in Week 11 describe. Since the data of a matrix is normally large, please do not use data copy in constructors and operator =. You can follow the design of cv::Mat in OpenCV.

在重载所要求运算符的基础上重载`=`运算。引用传递并使用原子加减法保证多线程安全与内存管理。

```
void Matrix::operator=(const Matrix &b)
{

    if (num == b.num)
    {
        return;
    }

    x = b.x;
    y = b.y;
    //*num = *num - 1;
    //_MT_DECR(*num);
    __sync_fetch_and_sub(num,1);
    if (*num == 0)
    {

        delete[] matrix;
        delete num;
    }
    num = b.num;
    //_MT_INCR(*num);
    __sync_fetch_and_add(num,1);
    matrix = b.matrix;
    //*num += 1;
}
```

3. (25 points) Please implement operator * overloading such that the class can support: C = A * B, C = A * b, and C = a * B (Capital letters are for matrices. Small letters are for scalars).

重载了所要求的函数。

```
friend Matrix operator*(const float &x, const Matrix &b);

friend Matrix operator*(const Matrix &b, const float &x);

friend std::ostream& operator <<(std::ostream& os,const Matrix& b);
```

4. (10 points) Compile and run your program on an ARM development board

使用`scp`,`ssh`方式传输文件，连接树莓派。成功编译运行。

```
satan@satan-c:/var/www/html$ g++ main.cpp Matrix.cpp Matrix.h -o Matrix.exe
satan@satan-c:/var/www/html$ ./Matrix.exe
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000

0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000

1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000
1.000000 1.000000 1.000000 1.000000 1.000000

5.000000 5.000000 5.000000 5.000000 5.000000
5.000000 5.000000 5.000000 5.000000 5.000000
5.000000 5.000000 5.000000 5.000000 5.000000
5.000000 5.000000 5.000000 5.000000 5.000000
5.000000 5.000000 5.000000 5.000000 5.000000

del:
0xaaaac7a0f7c0
0xaaaac7a0f830
25.000000 25.000000 25.000000 25.000000 25.000000
25.000000 25.000000 25.000000 25.000000 25.000000
25.000000 25.000000 25.000000 25.000000 25.000000
25.000000 25.000000 25.000000 25.000000 25.000000
25.000000 25.000000 25.000000 25.000000 25.000000

del:
0xaaaac7a0efd0
0xaaaac7a0f040
del:
0xaaaac7a0f710
0xaaaac7a0f7a0
del:
0xaaaac7a0f680
0xaaaac7a0f6f0
```

5. (5 points) Please use cmake to manage your source code.

使用如下Cmake代码进行管理。

```
cmake_minimum_required(VERSION 3.10)
project(test)

set(CMAKE_CXX_STANDARD 17)

add_executable(test main.cpp Matrix.h Matrix.cpp)
```

代码与实现过程

- ## 代码结构

  代码由三个文件与Cmake配置组成：

文件名	功能
Matrix.h	头文件，定义函数
Matrix.cpp	函数实现
main.cpp	测试函数入口

- ## Class结构

```
class Matrix
{
private:
    int x = 0, y = 0;
    float *matrix = nullptr;
    int *num = nullptr;
```

Matrix类共有四个变量，根据题目要求，全部为`private`类型。其中：

`x`代表矩阵的行数，`y`代表矩阵的列数。

`float`数组保存矩阵内容。处于速度考量，此处用一维数组保存二维数据。

`num`记录数组当前被引用的次数。用于内存的自动回收机制。

- ## 主要函数

```
    Matrix();

    Matrix(int x, int y);

    Matrix(const Matrix &b);

    bool clone(const Matrix &b);

    ~Matrix();

    void operator=(const Matrix &b);

    void getAdd();

    Matrix operator+(const Matrix &b);

    Matrix operator-(const Matrix &b);

    Matrix operator*(const Matrix &b);

    Matrix operator^(const long long &b);

    friend Matrix operator*(const float &x, const Matrix &b);
```

```
    friend Matrix operator*(const Matrix &b, const float &x);

    friend std::ostream& operator <<(std::ostream& os,const Matrix& b);

    bool operator==(const Matrix &b);

    bool operator!=(const Matrix &b);

    void resize(int X, int Y);

    void set(int v);

    std::string out();

    std::string to_string(float Num);

    void operator*=(const Matrix &b);

    void operator+=(const Matrix &b);

    void operator-=(const Matrix &b);
```

其中构造函数有三种，分别为默认构造，指定矩阵大小构造，通过已有的矩阵构造（与`clone`类似，属于深拷贝）

析构函数实现了类似`cv::Mat`的功能，可以自动判断是否需要释放内存，无需手动释放。

`operator=`按照要求重载等于运算，浅拷贝加快速度。

`getAdd`用于跟踪指针。

重载`<<`，可以使用`cout`输出矩阵。

重载`==,!=`，可以进行矩阵比较。

`resize`可以改变矩阵形状，扩大补零，减小截取。

此外程序重载了`+,-,*,^,+=,*=,-=`等常用函数，其中*使用友元函数满足多种要求。

- **主要函数实现**
  - `void operator=(const Matrix &b);`

    将深拷贝更改为浅拷贝，增加程序执行效率。其中实现参考了`cv::Mat`。在类中定义`int *num`变量记录引用次数。赋值时改变数组与num指针的指向。 更新引用次数。考虑到实际应用场景，即：矩阵运算大多会使用多线程。对于引用次数的更新使用原子加减法。`__sync_fetch_and_sub(num,1);`大大增大了程序的适用范围。
  - `~Matrix();`

    本程序的析构函数通过引用计数实现内存的自动删除。与默认析构函数的不同在于其只会删除引用计数为0的数组空间。此函数同样是线程安全的。此外，为了更好的体现出删除的时机，程序会在删除时输出删除地址。
  - `Matrix operator^(const long long &b);`

    实现了矩阵快速幂，可以在`O(n^3logk)`的时间复杂度下求出方阵的k次方。具体实现过程利用了位运算的性质。
  - `void resize(int X, int Y);`

更改矩阵的大小。为了改善运算效率与空间消耗。程序使用`memcpy`进行内存区块的整体平移。并及时清除多余或原先的空间。

- ## 问题与解决方案
  - ### 默认构造函数污染指针

```
  Matrix Matrix::operator*(const Matrix &b)
      {
          Matrix ans(this->x, b.y);
          if (this->y != b.x)
              return Matrix();
          for (int i = 0; i < x; i++)
          {
              for (int k = 0; k < x; k++)
              {
                  for (int j = 0; j < b.y; j++)
                  {
                      ans.matrix[i * b.y + j] += (this->matrix[i * y + k] *
b.matrix[k * b.y + j]);
                  }
              }
          }
          return ans;
      }
```

上述函数在执行过程中即使没有命中if判断也会调用构造函数创建`Matrix`对象，并在函数结束后调用析构函数。在特定情况下会与引用计数冲突导致误删除。正确解决方案是`return ans;`即：

```
  Matrix Matrix::operator*(const Matrix &b)
      {
          Matrix ans(this->x, b.y);
          if (this->y != b.x)
              return ans;
          for (int i = 0; i < x; i++)
          {
              for (int k = 0; k < x; k++)
              {
                  for (int j = 0; j < b.y; j++)
                  {
                      ans.matrix[i * b.y + j] += (this->matrix[i * y + k] *
b.matrix[k * b.y + j]);
                  }
              }
          }
          return ans;
      }
```

  - `new[]`与`sizeof`在某些情况下会冲突。

```
this->matrix = new float[x * y];
cout<<sizeof(matrix);
memset(matrix, 0, sizeof(float)*x*y);
```

上述函数输出为8，因此使用：

`memset(matrix, 0, sizeof(matrix));`并不能完全清空数组

需要使用：

`memset(matrix, 0, sizeof(float)*x*y);`达到目的。