# CS205 C/ C++ Program Design

# Assignment 4

Name: 王康

SID: 11510815

## Part 1. Source Code

https://github.com/apostlewang/CS205_CPP/tree/master/Assignment4

## Part 2. Result & Verification

### 1.Class Matrix

头文件如下

```
class Matrix
{
private:
    /* data */
    int num_rows;
    int num_columns;
    float * data;
    int *refcount;

public:
    Matrix(int num_rows, int num_columns, float* data);
    Matrix(const Matrix &m);
    ~Matrix();
    Matrix operator* (const Matrix &m) const;
    Matrix operator* (const float a) const;
    Matrix& operator= (const Matrix &m);
    float operator() (int row, int column) const;
    friend Matrix operator* (const float a,const Matrix &m);
    friend ostream& operator<<(ostream &os,const Matrix &m);
    friend float* multiMatrix(const Matrix &a, const Matrix &b);
};
```

数据部分设置为private，其中refcout会统计一块data数据区域被使用多少次（共享一块内存的对象也共享这个数字），使用copy constructor和重载的赋值运算符均会使得其增加。对应到destructor的设计，就是只有当某块数据区域的refcount为0时才会删除这块数据，否则对refcount减去1。

下面是函数的声明，均在Matrix.cpp中具体实现。

# 2.Constructors, destructor, operators overloading

## Constructors

```
Matrix::Matrix(int num_rows, int num_columns, float* data){
    this->num_columns = num_columns;
    this->num_rows = num_rows;
    this->data = data;
    this->refcount = new int(0);
}

Matrix::Matrix(const Matrix &m){
    this->num_columns = m.num_columns;
    this->num_rows = m.num_rows;
    this->data = m.data;
    this->refcount = m.refcount;
    *refcount += 1;
}
```

两种构造方法，第一种通过传入行数，列数和数据的指针构造（参考opencv的实现，浅拷贝），第二种是copy方式构造，值得注意的是这里的data也是共享的，被多一个对象指向后对应的refcount就增加1。

## Destructor

```
Matrix::~Matrix(){
    if(*refcount == 0){
        delete []data;
    }
    else{
        *refcount -= 1;
    }
}
```

只有当某块数据区域的refcount为0时才会删除这块数据释放内存，否则只对refcount指向的数值减去1。

## operator =

```
Matrix& Matrix::operator= (const Matrix &m){
    if (this == &m){
        return *this;
    }
    this->num_columns = m.num_columns;
    this->num_rows = m.num_rows;
    if(*(this->refcount) == 0){
```

```
        delete [] this->data;
    }
    else{
        *(this->refcount) -= 1;
    }
    this->refcount = m.refcount;
    *refcount += 1;
    this->data = m.data;
    return *this;
}
```

前面的if判断可以避免某个对象赋值给自身，之后对于被赋值的对象，行数列数可以直接对应拷贝，接着因为该对象原来的数据区域将不再被它引用，判断其refcount是否为0，若为0则释放内存，否则refount减去1。

### operator <<

```
ostream& operator<<(ostream &os,const Matrix &m){
    for (size_t i = 0; i < m.num_rows; i++)
    {
        for (size_t j = 0; j < m.num_columns; j++)
        {
            os << setw(10)<< left << m.data[i*m.num_columns+j];
        }
        os << endl;
    }
    os << endl;
    return os;
}
```

重载输出，打印矩阵。

### operator ()

```
float Matrix::operator() (int row, int column) const{
    assert(row < this->num_rows && column < this->num_columns && row >= 0 &&
column >= 0);
    return data[row*num_columns+column];
}
```

输出对应位置的元素（序号从0开始）。

## 3.Operator * overloading

```
Matrix Matrix::operator* (const Matrix &m) const{
    float* newdata = multiMatrix(*this,m);
    Matrix result = Matrix(num_rows,m.num_columns,newdata);
    return result;
```

```
}

Matrix Matrix::operator* (const float a) const{
    float * newdata = new float[num_columns*num_rows];
    for (size_t i = 0; i < num_columns*num_rows; i++)
    {
        newdata[i] = data[i]*a;
    }
    Matrix result = Matrix(num_rows,num_columns,newdata);
    return result;
}
Matrix operator* (const float a,const Matrix &m){
    float * newdata = new float[m.num_columns*m.num_rows];
    for (size_t i = 0; i < m.num_columns*m.num_rows; i++)
    {
        newdata[i] = m.data[i]*a;
    }
    Matrix result = Matrix(m.num_rows,m.num_columns,newdata);
    return result;
}
```

第一个矩阵乘矩阵，主要乘法运算由multiMatrix函数提供，计算完成后构造结果对应的矩阵并返回。

第二个和第三个类似，标量乘矩阵，对应元素相乘即可，不同的是一个是类的成员函数，一个是友元函数。

在main中验证三种重载运算符“*”的代码：

```
    Matrix C = A*1.2;
    cout << "C:" << endl << C;
    Matrix D = 0.83*A;
    cout << "D:" << endl << D;
    cout << "C*D:" << endl << C*D;
```

运行结果均正确。

## 4.Compile and run the program on an ARM development board

在树莓派上的运行（ssh连接后操作）截图如下：

```
pi@raspberrypi:~/Documents $ make
Scanning dependencies of target matrix
[ 33%] Building CXX object CMakeFiles/matrix.dir/main.cpp.o
[ 66%] Linking CXX executable matrix
[100%] Built target matrix
pi@raspberrypi:~/Documents $ ./matrix
B:
1         2         3         4
5         6         7         8
9         10        11        12
13        14        15        16

C:
1.2       2.4       3.6       4.8
6         7.2       8.4       9.6
10.8      12        13.2      14.4
15.6      16.8      18        19.2

D:
0.83      1.66      2.49      3.32
4.15      4.98      5.81      6.64
7.47      8.3       9.13      9.96
10.79     11.62     12.45     13.28

C*D:
89.64     99.6      109.56    119.52
201.192   227.088   252.984   278.88
312.744   354.576   396.408   438.24
424.296   482.064   539.832   597.6

C=A:
1         2         3         4
5         6         7         8
9         10        11        12
13        14        15        16

C(1,3) = 8
```

## 5.Host in Github

https://github.com/apostlewang/CS205_CPP/tree/master/Assignment4

## 6.Use Cmake to manage code

CMakeLists文件内容如下：

```
project(mymatrix)
add_executable(matrix main.cpp Matrix.cpp)
```

使用cmake CMakeLists.txt即可生成makefile，执行make命令编译程序。