

CS205 C/ C++ Program Design - Assignment 4

Name 吕昊泽 SID: 11912814

Part 1 Result & Verification

本程序使用的环境和 openblas 说明：本程序下面在 x86(windows)平台进行地测试测试均在 visual studio 2017 运行下得出，使用的 openblas 版本为 <https://github.com/xianyi/OpenBLAS/releases/tag/v0.3.10> 下载的 X86 版本。

使用 cmake 进行项目的操作将会在 cmake 部分进行讲解

在 ARM 开发板上的运行同样基于 cmake 文件 openblas 库是进行克隆的库，具体详见 cmake 部分

本文中提到的 opencv 资料均是自己通过 cmake 编译出的 visual studio 工程后进入项目看的源代码。

第一部分：Matrix 类

代码如下

```
class Matrix {
private:
    mutex *mu;
    int row_num, column_num;
    long size;
    float *matrixdata;
    int *refcount;
public:
    Matrix(int row_num=1,int column_num=1);
    Matrix(float value,int row_num=1, int column_num = 1);
    Matrix(int num);
    Matrix(string str);
    Matrix(int num, float value);
    Matrix(const Matrix &rhs);
    ~Matrix();
    Matrix &operator=(const Matrix &rhs);
```

```

    friend ostream &operator<< (ostream &os, Matrix &rhs);
    friend Matrix operator+(const Matrix& matrix1, const Matrix &matrix
2);
    friend Matrix operator-
(const Matrix &matrix1, const Matrix &matrix2);
    friend Matrix operator*(const Matrix &matrix1, const Matrix &matrix
2);
    friend Matrix operator*(float mul, const Matrix &matrix2);
    friend Matrix operator*(const Matrix &matrix1, float mul);
    friend Matrix operator/(float mul, const Matrix &matrix2);
    friend Matrix operator/(const Matrix &matrix1, float mul);
    int getRefCount();
    float operator()(int rowNum, int columnNum);
    friend bool operator==(const Matrix &matrix1, const Matrix &matrix2
);
    Matrix getSubMatrix(int rowStart, int rowEnd, int columnStart, int
columnEnd);
};
Matrix creatMatWithString();

```

Matrix 类中有成员变量 row_num column_num size 三个变量，用于维护矩阵的行数，列数和大小（row*column） 有一个指针 matrixdata 用于记录矩阵的数据，长度为 size 一个指针 refcount 记录矩阵的数据被重用了多少次 还有一个 mutex 指针 Mu 用于实现任何对 refcount 的变更为原子操作，防止多线程同时访问 refcount 导致 refcount 错误，导致矩阵异常析构

```

private:
    mutex *mu;
    int row_num, column_num;
    long size;
    float *matrixdata;
    int *refcount;

```

矩阵的基本思路就是用一维数组以行优先的方式储存矩阵，并且通过引用 copy 实现大矩阵数据的重用 用 refcount 记录使用的次数，在为 1 时析构。 可以看到矩阵类中重定义了各种运算符用来实现矩阵的加减乘除和相等判断

第二部分，构造方法和析构方法

```
Matrix(int row_num=1,int column_num=1);
Matrix(float value,int row_num=1, int column_num = 1);
Matrix(int num);
Matrix(string str);
Matrix(int num, float value);
Matrix(const Matrix &rhs);
~Matrix();
```

可以看到矩阵定义了多种构造方法，并且一些方法带有默认参数。除了简单的指定行列构建矩阵外，比较特别的是通过字符串的方式构建矩阵

```
Matrix::Matrix(string str) {
    int rowNum = getrowNum(str);
    int columnNum = testSameLength(str, rowNum);
    this->size = rowNum * columnNum;
    this->row_num = rowNum;
    this->column_num = columnNum;
    this->refcount = new int(1);
    this->mu = new mutex();
    this->matrixdata = new float[this->size];
    int i = 1;
    int j = 0;
    int length = str.length();
    while (i < length) {
        string str0 = "";
        while (i < length && str.at(i) != ',' && str.at(i) != ';' && str.at
(i) != ']') {
            str0 += str.at(i);
            i++;
        }
        this->matrixdata[j] = stof(str0);
        if (str.at(i) == ',') {
            j++;
            i++;
        }
        else if (str.at(i) == ';') {
            j++;
            i++;
        }
        else {
            break;
        }
    }
}
```

```

    }
}

```

在本程序中，可以通过两种方法用字符串初始化矩阵，分别是
 从命令行输入和内部 string 参数

```

Matrix creatMatWithString() {
    cout << "Enter your Matrix in form([1,2;3,4]): ";
    string str1;
    cin >> str1;
    if (testMatrix(str1)) {
        int num1 = getrowNum(str1);
        if (num1 > 1) {
            num1 = testSameLength(str1, num1);
        }
        if (num1 > 0) {
            Matrix matrix = Matrix(str1);
            return matrix;
        }
        else {
            cout << "矩阵每行元素个数不相同（返回一个 1*1 矩阵）" << endl;
        }
    }
    else {
        cout << "输入的矩阵格式错误（返回一个 1*1 矩阵）" << endl;
    }
    return Matrix(0, 1, 1);
}

Matrix creatMatWithString(string str) {
    if (testMatrix(str)) {
        int num1 = getrowNum(str);
        if (num1 > 1) {
            num1 = testSameLength(str, num1);
        }
        if (num1 > 0) {
            Matrix matrix = Matrix(str);
            return matrix;
        }
        else {
            cout << "矩阵每行元素个数不相同（返回一个 1*1 矩阵）" << endl;
        }
    }
}

```

```

    }
    else {
        cout << "输入的矩阵格式错误（返回一个 1*1 矩阵）" << endl;
    }
    return Matrix(0, 1, 1);
}

```

可以看到，Matrix (str) 构造方法通过以下两个方法调用 先判断字符串的合法性，然后获取行数，然后判断每行对应的列数是否相同，如果都通过了就进入正常的构造。值得一提的是，testMatrix 使用了类似状态机的思想，大大简化了代码长度。下面是命令行输入的一些简单样例。

Test case 1:

Input [1,2,3;4,5,6]

[1,1;1,1;1,1]

Output

```

Enter your first Matrix([1,2;3,4]): [1,2,3;4,5,6]
Enter your second Matrix: [1,2;3,4;5,6]
Matirx 1
1 2 3
4 5 6
Matrix 2
1 2
3 4
5 6
Matrix 3
22 28
49 64

```

非法输入

程序通过 testMatrix()和 getRwo()两个函数，对输入向量的格式和数字的合法性进行检测

以下是错误输入展示

```

input 0 to test the normal matrix Dot, input 1 to test the time cost of two big matrix Dot: 0
Enter your first Matrix([1,2;3,4]): [1.1.1,1,1]
Enter your second Matrix: [1;1;1]
The type of the float is wrong
输入的矩阵格式错误

```

数字输入错误，包含两个小数点

```

input 0 to test the normal matrix Dot, input 1 to test the time cost of two big matrix Dot: 0
Enter your first Matrix([1,2;3,4]): 【1,1,1】
Enter your second Matrix: [1;1;1]
The expression should begin with [
输入的矩阵格式错误

```

函数中括号使用了中文输入符

```
Enter your first Matrix([1,2;3,4]): [1,1,1,1]
Enter your second Matrix: [1;1;1]
The column of matrix1 is not equal to the row of matrix 2, the Matrix 3 will be a 0
Matrix 1
1 1 1 1
Matrix 2
1
1
1
1
Matrix 3
0
```

点乘的两个向量拥有的元素个数不相同，矩阵 3 将会为 0

```
input 0 to test the normal matrix Dot, input 1 to test the time cost of two big matrix Dot: 0
Enter your first Matrix([1,2;3,4]): [--1,1,1]
Enter your second Matrix: [1;1;1]
The type of the float is wrong
输入的矩阵格式错误
```

错误的输入了两个负号

```
input 0 to test the normal matrix Dot, input 1 to test the time cost of two big matrix Dot: 0
Enter your first Matrix([1,2;3,4]): [1,1,1
Enter your second Matrix: [1;1;1]
The expression should end with ]
输入的矩阵格式错误
```

没有右中括号

```
Enter your first Matrix([1,2;3,4]): [1,1,1]1]
Enter your second Matrix: [1;1;1]
中括号不应该在中间出现
输入的矩阵格式错误
```

```
input 0 to test the normal matrix Dot, input 1 to test the
Enter your first Matrix([1,2;3,4]): [1,1,]2]
Enter your second Matrix: [1;1;1]
The character , should be followed with a float
输入的矩阵格式错误
```

右中括号提前结束

```
Enter your first Matrix([1,2;3,4]): [1,,1]
Enter your second Matrix:
[2;2;2]
The character , should be followed with a float
输入的矩阵格式错误
```

,, 直接相连，没有中间的数字

```
input 0 to test the normal matrix Dot, input 1 to te
Enter your first Matrix([1,2;3,4]): [1,1,1;1,1]
Enter your second Matrix: [2;2]
矩阵每行元素个数不相同
```

矩阵每行元素不同

```
Enter your first Matrix([1,2;3,4]): [1+2]
Enter your second Matrix: [3;4]
The expression contain illegal character
输入的矩阵格式错误
```

数字之后的非法输入

析构方法

本程序由于矩阵类采用了共享的矩阵数据，以避免过大的矩阵数据造成内存的反

复浪费，所以采用了 refcount 变量进行内存使用次数的记录。因此相同数据的矩阵每对应生成一个，矩阵对应的 refcount 就+1，同样的，矩阵被析构一个，refcount 就-1 当 refcount 为 1 时，矩阵便被析构，空间被释放，具体实现如下

```
Matrix::~Matrix() {
    (*mu).lock();
    if (*(this->refcount) < 2) {
        delete[] this->matrixdata;
        delete this->refcount;
    }
    else {
        (*this->refcount)=(*this->refcount)-1;
    }
    (*mu).unlock();
}
```

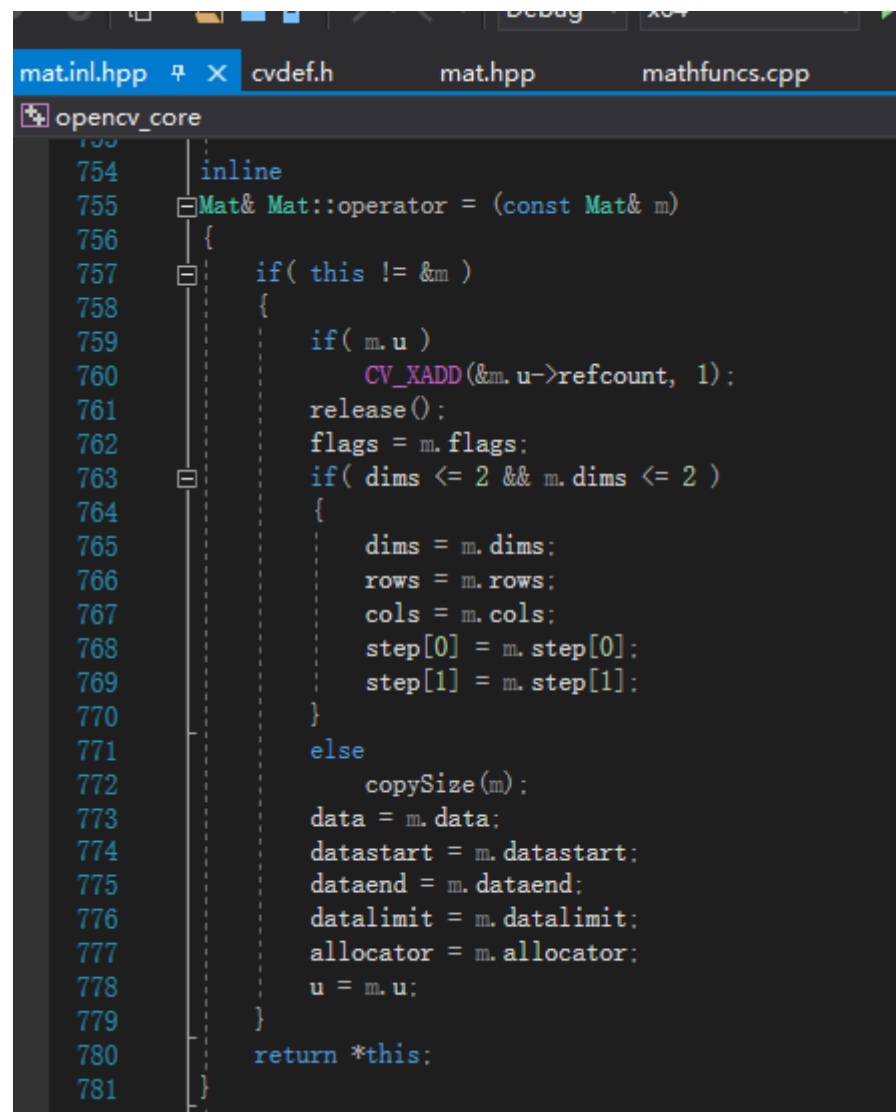
可以看到这里采用了互斥锁的方式来保证同一时间不会有多个线程访问 refcount 造成内存的损坏。但是与参考的 opencv 使用的宏定义不同，本程序采用了简单的互斥锁进行线程的保护。

```
void Mat::release()
{
    if( u && CV_XADD(&u->refcount, -1) == 1 )
        deallocate();
    u = NULL;
    datastart = dataend = datalimit = data = 0;
    for(int i = 0; i < dims; i++)
        size.p[i] = 0;
#ifdef _DEBUG
    flags = MAGIC_VAL;
    dims = rows = cols = 0;
    if(step.p != step.buf)
    {
        fastFree(step.p);
        step.p = step.buf;
        size.p = &rows;
    }
#endif
}
```

通过与 opencv 的对比可知同样是减一后进行判断，进行析构。

等于号的重载

本程序参考了 opencv 中等号重载的使用方法实现了类似的功能



```
754 inline
755 Mat& Mat::operator = (const Mat& m)
756 {
757     if( this != &m )
758     {
759         if( m.u )
760             CV_XADD(&m.u->refcount, 1);
761         release();
762         flags = m.flags;
763         if( dims <= 2 && m.dims <= 2 )
764         {
765             dims = m.dims;
766             rows = m.rows;
767             cols = m.cols;
768             step[0] = m.step[0];
769             step[1] = m.step[1];
770         }
771         else
772             copySize(m);
773         data = m.data;
774         datastart = m.datastart;
775         dataend = m.dataend;
776         datalimit = m.datalimit;
777         allocator = m.allocator;
778         u = m.u;
779     }
780     return *this;
781 }
```

以上是 opencv 中关于等号重载的实现，可以看到先利用宏定义对 refcount 进行 +1 操作，然后再通过判断矩阵数据的类型通过不同的方法将自身结构体中对应的各种数据拷贝到 this 中，最后返回 this

本程序实现的思路与之类似，但减少了复杂的判断，原因是 opencv 中矩阵数据是单独定义在一个结构体中的，拷贝起来要考虑其他参数的高效传递。


```

7     }
8     Matrix &Matrix::operator=(const Matrix &rhs) {
9         if (this != &rhs) {
10             this->column_num = rhs.column_num;
11             this->row_num = rhs.row_num;
12             this->matrixdata = rhs.matrixdata;
13             this->mu = rhs.mu;
14             (*mu).lock();
15             this->refcount = rhs.refcount;
16             (*this->refcount)++;
17             (*mu).unlock();
18         }
19         return *this;
20     }

```

同样利用了互斥锁实现了对 refcount 的操作为原子操作，同时将各个数据通过指针拷贝的方式传递到新的 this 中，最后返回 this。这样做高效的实现了矩阵的拷贝，同时实现了空间的复用，是程序运行时更加节约内存。

其他简单运算符的重载

除去前文提到的=和多种构造方法和析构方法外，本程序还重写了<<运算符，得以将矩阵直接输入到 cout 流中，实现矩阵的显示，代码实现和实例如下

```

}
ostream &operator<< (ostream &os, Matrix &rhs) {
    int columnNum = rhs.column_num;
    for (int i = 0; i < rhs.row_num; i++) {
        os << "| ";
        for (int j = 0; j < rhs.column_num; j++) {
            os << rhs.matrixdata[i*columnNum + j] << ' ';
        }
        os << "|" << endl;
    }
    return os;
}

```

```

matrix1
1 1
1 1
1 1
1 1
1 1
1 1
1 1
1 1
1 1
1 1
1 1
matrix2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
matrix3=matrix1*matrix2
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
matrix4=matrix1*2

```

这些都是将矩阵初始化后通过<<的形式输入

```

cout << matrix1;
Matrix matrix2 = Matrix(2, 2, 10);
cout << "matrix2 " << endl;
cout << matrix2;

```

给 cout 流

此外还重载了==号判断两个矩阵是否相等，基本思路就是判断横纵坐标和指针。

```

bool operator==(const Matrix &matrix1, const Matrix &matrix2) {
    if (matrix1.column_num != matrix2.column_num || matrix1.row_num != matrix2.row_num) {
        return false;
    }
    if (matrix1.matrixdata != matrix2.matrixdata) {
        return false;
    }
    return true;
}

```

```

-1
t j Test matrix6==matrix1
c (True

```

```

bool T = matrix1 == matrix6;
if (T) {
    cout << "True" << endl;
}
else {
    cout << "False" << endl;
}

```

还参考 opencv 重载了根据行列值取得子矩阵的方法，但 opencv 是自己定义了

pair 类型用于表示范围，如图

```
Mat rowRange(const Range& r) const;
```

也实现了起始位置和终止位置的方法

```
Mat colRange(int startcol, int endcol) const;
```

本程序参考的时候这，具体实现如下

```
Matrix Matrix::getSubMatrix(int rowStart, int rowEnd, int columnStart, int columnEnd) {
    if (rowStart <= 0 || columnStart <= 0) {
        cout << "parameter is wrong, return a matrix(1*1)with 0" << endl;
        return Matrix(0, 1, 1);
    }
    if (rowEnd > this->row_num || columnEnd > this->column_num) {
        cout << "parameter is wrong, return a matrix(1*1)with 0" << endl;
        return Matrix(0, 1, 1);
    }
    if (rowEnd <= rowStart || columnEnd <= columnStart) {
        cout << "parameter is wrong, return a matrix(1*1)with 0" << endl;
        return Matrix(0, 1, 1);
    }
    Matrix matrix = Matrix(0, (rowEnd - rowStart), (columnEnd - columnStart));
    int k = columnStart - 1;
    int j = 0;
    for (int i = rowStart - 1; i < rowEnd-1; i++) {
        while (k < columnEnd-1)
        {
            matrix.matrixdata[j] = this->matrixdata[i*this->column_num + k];
            k++;
            j++;
        }
        k = columnStart-1;
    }
    return matrix;
}
```

矩阵实现了错误判断以免程序异常终止

为了获取矩阵数据，还重载了（）为了防止获取不存在的值加入了判断机制

```
float Matrix::operator()(int rowNum, int columnNum) {
    if (rowNum > this->row_num || rowNum <= 0) {
        cout << "rowNum is wrong, will return a -1" << endl;
        return -1;
    }
    if (columnNum > this->column_num || columnNum <= 0) {
        cout << "rowNum is wrong, will return a -1" << endl;
        return -1;
    }
    return this->matrixdata[(rowNum-1)*this->column_num + columnNum - 1];
}
```

```

true
matrix8=matrix3[3-5][3-5]
{
| 4 4 |
| 4 4 |
}

matrix9=matrix1[3-5][3-5] (展示超限了会怎么样)
parameter is wrong, return a matrix(1*1) with 0
| 0 |

```

第三部分 运算符的重载

根据老师上课讲的运算符重载，本程序对乘法，除法加减运算符进行了重载

```

friend Matrix operator+(const Matrix& matrix1, const Matrix &matrix
2);
friend Matrix operator-
(const Matrix &matrix1, const Matrix &matrix2);
friend Matrix operator*(const Matrix &matrix1, const Matrix &matrix
2);
friend Matrix operator*(float mul, const Matrix &matrix2);
friend Matrix operator*(const Matrix &matrix1, float mul);
friend Matrix operator/(float mul, const Matrix &matrix2);
friend Matrix operator/(const Matrix &matrix1, float mul);

```

首先是矩阵乘法运算符，除了常见的矩阵乘矩阵，本程序还重载了常数与矩阵相乘的运算。常数与矩阵的运算比较简单，即所有位都乘对应数值即可，因为本质上只要矩阵没有错误，程序就不可能出错，所以没有设置纠错功能。在矩阵乘法的实现上，由于本程序使用了 cmake 进行管理，且要在 arm 开发板上进行测试，所以使用期中 pro 的加速方法显得不那么适用。因为期中加速方法中大量使用了 avx2 指令集，而该指令集并不能在 arm 开发板上正常运行。因此本程序采用了额外附加库的形式来实现矩阵乘法运算，即使用一直作为对比测试的 openblas 库实现了矩阵乘法的运算，这样借助 openblas 库的特性就可以实现不同平台的通用。以下是三个乘法方法的具体实现。

```

Matrix operator*(const Matrix &matrix1, const Matrix &matrix2) {
    if (matrix1.column_num == matrix2.row_num) {
        int rowNum = matrix1.row_num;
        int columnNum = matrix1.column_num;
        Matrix matrix3 = Matrix(matrix1.row_num, matrix2.column_num);
        const int M = matrix1.row_num;
        const int N = matrix2.column_num;
        const int K = matrix1.column_num;
        const int ida = K;
        const int idb = N;
        const int idc = N;
        cblas_sgemv(CblasRowMajor, CblasNoTrans, CblasNoTrans, M, N, K, 1, matrix1.matrixdata, ida, matrix2.matrixdata, idb, 0.0, matrix3.matrixdata, idc);
        return matrix3;
    }
    cout << "两个矩阵大小不同，方法会返回一个大小是1的方阵" << endl;
    Matrix matrix3 = Matrix(1, 1);
    return matrix3;
}

Matrix operator*(float mul, const Matrix &matrix2) {
    Matrix matrix3 = Matrix(matrix2.row_num, matrix2.column_num);
    for (int i = 0; i < matrix2.size; i++) {
        matrix3.matrixdata[i] = mul * matrix2.matrixdata[i];
    }
    return matrix3;
}

Matrix operator*(const Matrix &matrix1, float mul) {
    Matrix matrix3 = Matrix(matrix1.row_num, matrix1.column_num);
    for (int i = 0; i < matrix1.size; i++) {
        matrix3.matrixdata[i] = mul * matrix1.matrixdata[i];
    }
    return matrix3;
}
}

```

可以看到矩阵之间的乘法重载时进行了错误的判断，避免程序出现问题。

之后便是矩阵加减除的重载，由于这部分实现比较简单因此简单带过

```

Matrix operator/(float mul, const Matrix &matrix2) {
    if(mul==0){
        cout<<"除数不能是0"<<endl;
        return matrix2;
    }
    Matrix matrix3 = Matrix(matrix2.row_num, matrix2.column_num);
    for (int i = 0; i < matrix2.size; i++) {
        matrix3.matrixdata[i] = mul / matrix2.matrixdata[i];
    }
    return matrix3;
}

Matrix operator/(const Matrix &matrix1, float mul) {
    if(mul==0){
        cout<<"除数不能是0"<<endl;
        return matrix1;
    }
    Matrix matrix3 = Matrix(matrix1.row_num, matrix1.column_num);
    for (int i = 0; i < matrix1.size; i++) {
        matrix3.matrixdata[i] = matrix1.matrixdata[i] / mul;
    }
    return matrix3;
}
}

```

除法因为只能与常数进行除法运算，所以实现很简单，只需要判断除数是不是 0

至于加法减法，因为只能互相做加减，所以只需要判断矩阵的行列是不是相同的

即可

```

Matrix operator+(const Matrix& matrix1, const Matrix &matrix2) {
    if (matrix1.column_num == matrix2.column_num&&matrix1.row_num == matrix2.column_num) {
        int rowNum = matrix1.row_num;
        int columnNum = matrix1.column_num;
        Matrix matrix3 = Matrix(matrix1.row_num, matrix1.column_num);
        for (int i = 0; i < rowNum*columnNum; i++) {
            matrix3.matrixdata[i] = matrix1.matrixdata[i] + matrix2.matrixdata[i];
        }
        return matrix3;
    }
    cout << "两个矩阵大小不同，方法会返回一个大小是1的方阵" << endl;
    return Matrix(1,1);
}

Matrix operator-(const Matrix &matrix1, const Matrix &matrix2) {
    if (matrix1.column_num == matrix2.column_num&&matrix1.row_num == matrix2.column_num) {
        int rowNum = matrix1.row_num;
        int columnNum = matrix1.column_num;
        Matrix matrix3 = Matrix(matrix1.row_num, matrix1.column_num);
        for (int i = 0; i < rowNum*columnNum; i++) {
            matrix3.matrixdata[i] = matrix1.matrixdata[i] - matrix2.matrixdata[i];
        }
        return matrix3;
    }
    cout << "两个矩阵大小不同，方法会返回一个大小是1的方阵" << endl;
    return Matrix(1, 1);
}
}

```

简单的测试

本程序在 assignment4.cpp 中进行了简单的测试，测试代码和结果如下

```

int main()
{
    Matrix matrix1 = Matrix(1, 10, 2);
    cout << "matrix1" << endl;
    cout << matrix1;
    Matrix matrix2 = Matrix(2, 2, 10);
    cout << "matrix2 " << endl;
    cout << matrix2;
    Matrix matrix3 = matrix1 * matrix2;
    Matrix matrix4 = matrix1 * 2;
    Matrix matrix5 = matrix2 / 2;
    Matrix matrix6 = matrix1;
    cout << "matrix3=matrix1*matrix2" << endl;
    cout << matrix3;
    cout << "matrix4=natrix1*2" << endl;
    cout << matrix4;
    cout << "matrix5=matrix2/2" << endl;
    cout << matrix5;
    cout << "matrix6=matrix1" << endl;
    cout << "the refcount of matrix6 is " << matrix6.getRefcount() << endl;
    cout << "using string to create matrix7" << endl;
}

```

```

Matrix matrix7 = creatMatWithString();
cout << matrix7;

Matrix matrixList[10000];
cout << "下面进行 refcount 原子操作的演示" << endl;
#pragma omp parallel for
for (int i = 0; i < 10000; i++) {
    matrixList[i] = matrix1;
}
cout << matrix1.getRefCount() << endl;;

cout << "using matrix1(1,1) to get matrix1[1][1]" << endl;
cout << matrix1(1, 1) << endl;
cout << "using matrix(3,3) to get matrix1[3][2]" << endl;
cout << matrix1(3, 3) << endl;
cout << "Test matrix6==matrix1" << endl;
bool T = matrix1 == matrix6;
if (T) {
    cout << "True" << endl;
}
else {
    cout<<"False"<<endl;
}
Matrix matrix8 = matrix3.getSubMatrix(3, 5, 3, 5);
cout << "matrix8=matrix3[3-5][3-5]" << endl;
cout << matrix8 << endl;

cout << "matrix9=matrix1[3-5][3-5](展示超限了会怎么样)" << endl;
Matrix matrix9 = matrix1.getSubMatrix(3, 5, 3, 5);
cout << matrix9 << endl;

cout << "用内置字符串初始矩阵（[1,2;3,4]）" << endl;
string str3 = "[1,2;3,4]";
Matrix matrix10 = creatMatWithString(str3);
cout << matrix10 << endl;
}

```

```

1 1
1 1
matrix2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
matrix3=matrix1*matrix2
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
matrix4=matrix1*2
2 2
2 2
2 2
2 2
2 2
2 2
2 2
2 2
2 2
2 2
matrix5=matrix2/2
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
matrix6=matrix1
the refcount of matrix6 is 2
using string to create matrix7
Enter your Matrix in form([1, 2; 3, 4]): [1, 2; 3, 4]
1 2
3 4
下面进行refcount原子操作的演示
10002
using matrix1(1,1) to get matrix1[1][1]
1
using matrix(3,3) to get matrix1[3][2]
rowNum is wrong, will return a -1
-1
Test matrix6==matrix1
True
matrix8=matrix3[3-5][3-5]
4 4
4 4

matrix9=matrix1[3-5][3-5](展示超限了会怎么样)
parameter is wrong, return a matrix(1*1)with 0
0

用内置字符串初始矩阵 ([1, 2; 3, 4])
1 2
3 4

```



```

matrix3=matrix1*matrix2
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
matrix4=natrix1*2
2 2
2 2
2 2
2 2
2 2
2 2
2 2
2 2
2 2
2 2
2 2
matrix5=matrix2/2
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
matrix6=matrix1

```

第一项测试是矩阵乘法除法运算，测试了矩阵乘矩阵，矩阵乘常数,矩阵除常数

第二项测试是矩阵 refcount 变化的测试，并进行了原子操作的演示。

```
matrix6=matrix1
the refcout of matrix6 is 2
```

```
下面进行refcount原子操作的演示
10002
```

```
cout << "下面进行refcount原子操作的演示" << endl;
#pragma omp parallel for
for (int i = 0; i < 10000; i++) {
    matrixList[i] = matrix1;
}
cout << matrix1.getRefcount() << endl;;
```

程序在这里用 openmp 启用多线程对 matrix1 进行复制操作，最后 refcount 为正确的 10002 证明对 refcount 的操作是原子操作。

之后是对重载的 () 运算符的测试

```

using matrix1(1,1) to get matrix1[1][1]
1
using matrix(3,3) to get matrix1[3][2]
rowNum is wrong, will return a -1
-1

```

分为值存在和不存在两种情况

之后是子矩阵的获取，同样分为成功和不成功两种情况



```
matrix8=matrix3[3-5][3-5]
| 4 4 |
| 4 4 |

matrix9=matrix1[3-5][3-5](展示超限了会怎么样)
parameter is wrong, return a matrix(1*1)with 0
| 0 |
```







Part4 cmake 和 arm 开发板的运行

由于程序使用了 openblas 标准库，并希望在更多的平台进行运行，所以使用了 cmake 进行了代码的管理，本程序在 github 的结构和 cmake 原理介绍如下

1 结构：本文的 cpp 文件都放在根目录的 src 文件夹中





 assignment4.cpp	2020/12/6 0:17	C++ 源文件	2 KB
 Matrix.cpp	2020/12/6 10:47	C++ 源文件	12 KB

头文件都放在 include 文件夹中

 cblas.h	2020/6/16 4:16	C Header 源文件	47 KB
 f77blas.h	2020/6/16 4:16	C Header 源文件	47 KB
 lapack.h	2020/6/15 4:03	C Header 源文件	451 KB
 lapacke.h	2020/6/15 4:03	C Header 源文件	815 KB
 lapacke_config.h	2020/6/15 4:03	C Header 源文件	5 KB
 lapacke_mangling.h	2020/6/15 4:03	C Header 源文件	1 KB
 lapacke_utils.h	2020/6/15 4:03	C Header 源文件	33 KB
 Matrix.h	2020/12/6 0:18	C Header 源文件	2 KB
 openblas_config.h	2020/6/16 4:16	C Header 源文件	5 KB

其中包括了 openblas 库的头文件

需要链接的库存放在 lib 文件夹中

 libopenblas.a	2020/6/16 4:07	A 文件	30,174 KB
 libopenblas.dll	2020/6/16 4:16	应用程序扩展	23,707 KB
 libopenblas.dll.a	2020/6/16 4:16	A 文件	5,486 KB
 libopenblas.lib	2020/6/16 4:58	Object File Library	1,658 KB

Cmake 使用上，书写了 cmakeLists.txt 文件，用于在不同机器上生成 makefile

文件，具体实现如下

```
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
cmake_minimum_required(VERSION 3.4.1)

project(demo)

aux_source_directory(${PROJECT_SOURCE_DIR}/src DIR_SRC)

include_directories(${PROJECT_SOURCE_DIR}/include)

link_directories(${PROJECT_SOURCE_DIR}/lib)

add_executable(demo ${DIR_SRC})

target_link_libraries(demo libopenblas.lib)
```

第一行指定了 cmake 最小要求的版本号，第二行指定了工程名称。

第三行制定了源文件的目录，并将它们统一记录成 DIR_SRC.

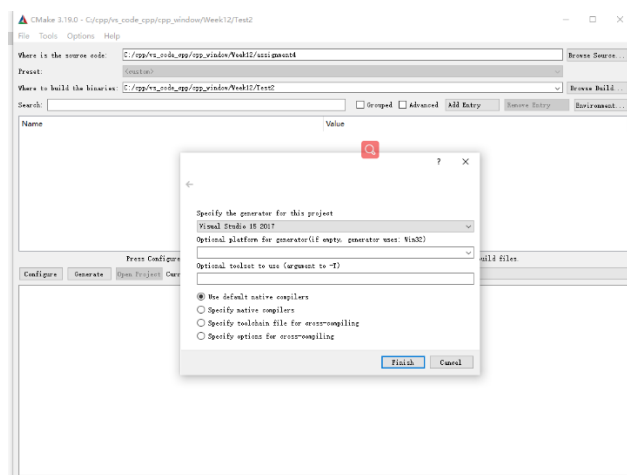
第四行制定了引用文件夹的目录。

第五行制定了链接目录

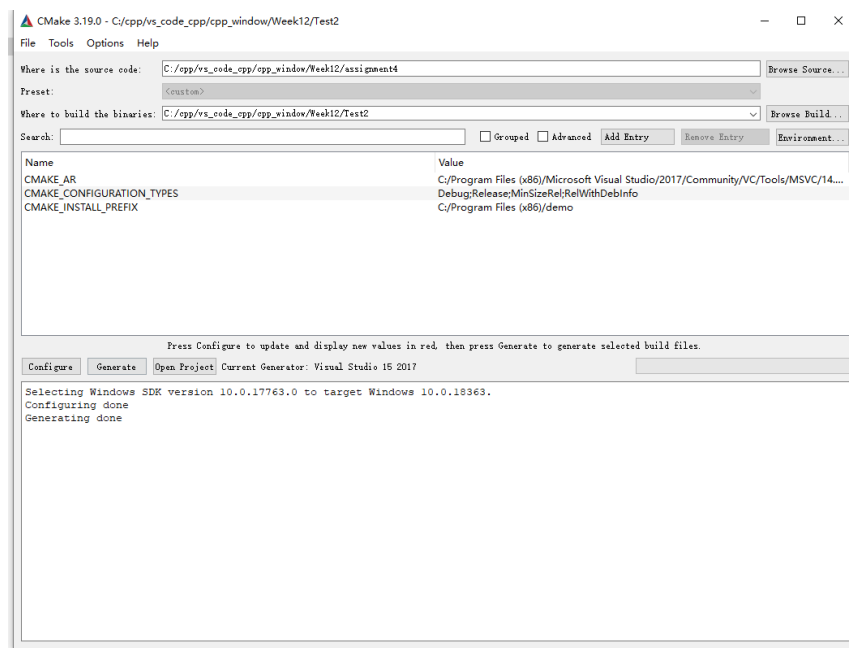
第六，七行生成 demo 执行程序并与 openblas.lib 链接

Windows 具体操作如下

首先用 cmake 选择源文件目录和工程生成目录。并选择生成 vs2017 工程



其 次 进 行 工 程 的 生 成



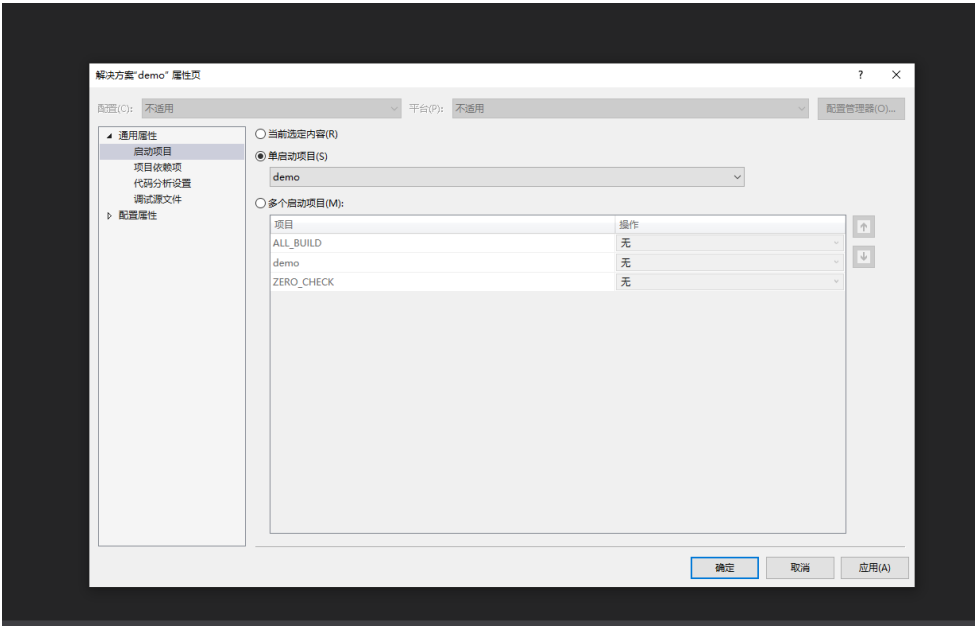
随后打开工程生成的目录，发现 vs2017 工程已经成功生成

名称	修改日期	类型	大小
CMakeFiles	2020/12/5 23:53	文件夹	
ALL_BUILD.vcxproj	2020/12/5 23:53	VC++ Project	17 KB
ALL_BUILD.vcxproj.filters	2020/12/5 23:53	VC++ Project Fil...	1 KB
cmake_install.cmake	2020/12/5 23:53	CMAKE 文件	2 KB
CMakeCache.txt	2020/12/5 23:53	文本文档	14 KB
demo.sln	2020/12/5 23:53	Visual Studio Sol...	4 KB
demo.vcxproj	2020/12/5 23:53	VC++ Project	28 KB
demo.vcxproj.filters	2020/12/5 23:53	VC++ Project Fil...	1 KB
ZERO_CHECK.vcxproj	2020/12/5 23:53	VC++ Project	16 KB
ZERO_CHECK.vcxproj.filters	2020/12/5 23:53	VC++ Project Fil...	1 KB

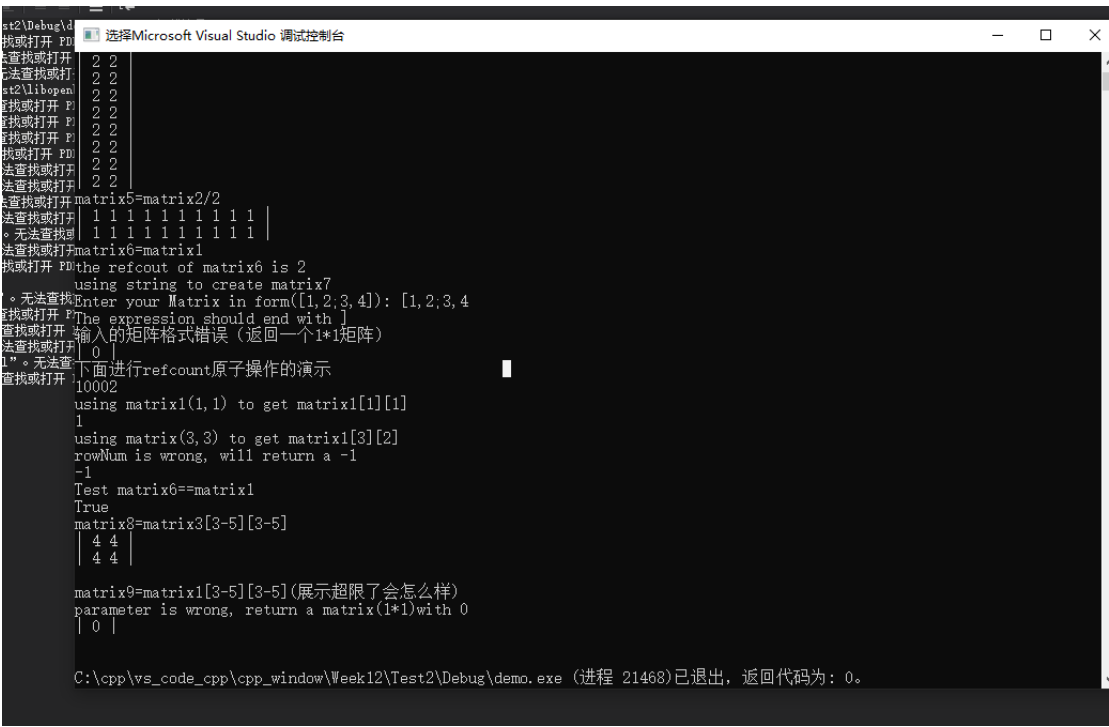
此时需要把源文件中的 libopenblas.dll 文件复制到工程目录下

名称	修改日期	类型	大小
.vs	2020/12/5 23:54	文件夹	
CMakeFiles	2020/12/5 23:53	文件夹	
ALL_BUILD.vcxproj	2020/12/5 23:53	VC++ Project	17 KB
ALL_BUILD.vcxproj.filters	2020/12/5 23:53	VC++ Project Fil...	1 KB
cmake_install.cmake	2020/12/5 23:53	CMAKE 文件	2 KB
CMakeCache.txt	2020/12/5 23:53	文本文档	14 KB
demo.sln	2020/12/5 23:53	Visual Studio Sol...	4 KB
demo.vcxproj	2020/12/5 23:53	VC++ Project	28 KB
demo.vcxproj.filters	2020/12/5 23:53	VC++ Project Fil...	1 KB
libopenblas.dll	2020/6/16 4:16	应用程序扩展	23,707 KB
ZERO_CHECK.vcxproj	2020/12/5 23:53	VC++ Project	16 KB
ZERO_CHECK.vcxproj.filters	2020/12/5 23:53	VC++ Project Fil...	1 KB

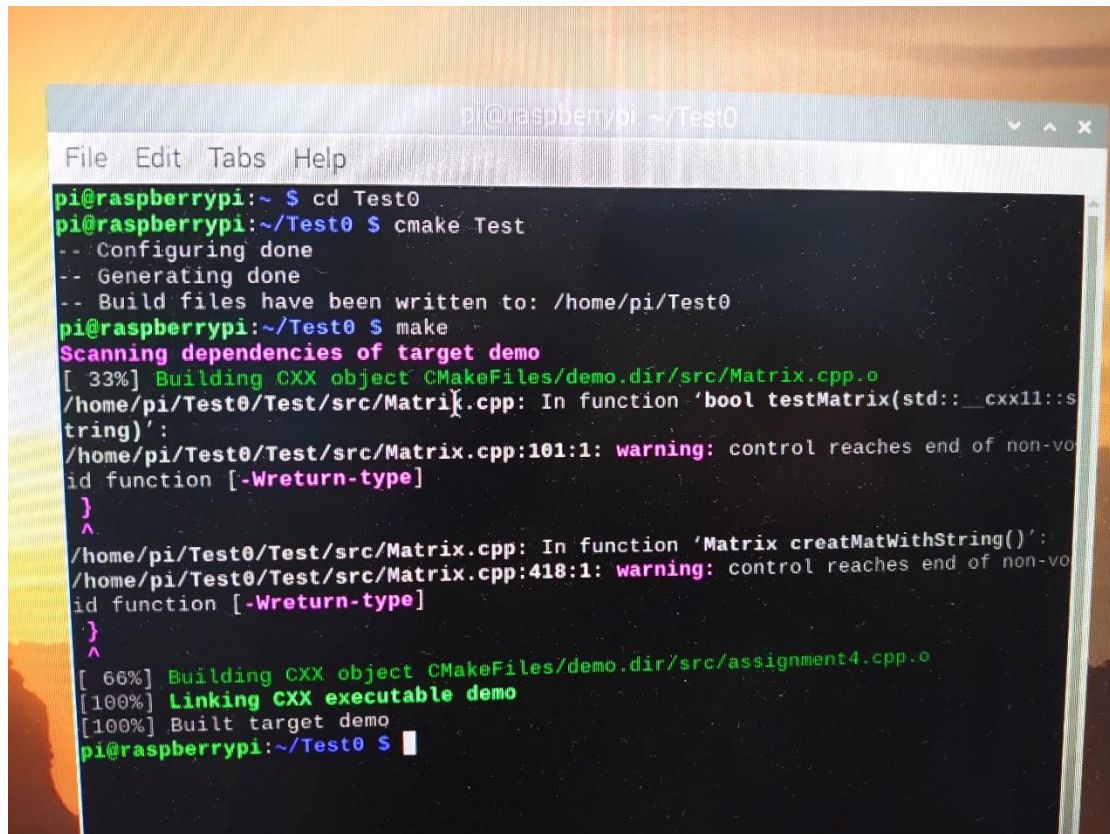
随后点击 demo.sln 打开工程



由于 camke 自动还会生成 all_build 和 zero_check 两个工程，所以要把单启动项改成 demo，随后程序就可以正常运行。

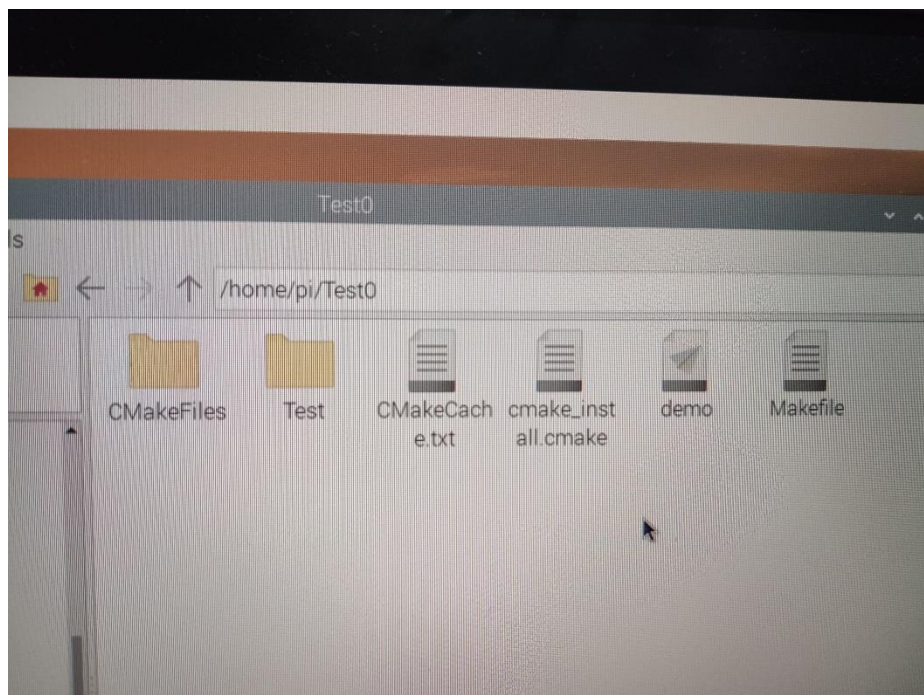


Arm 开发板使用方法



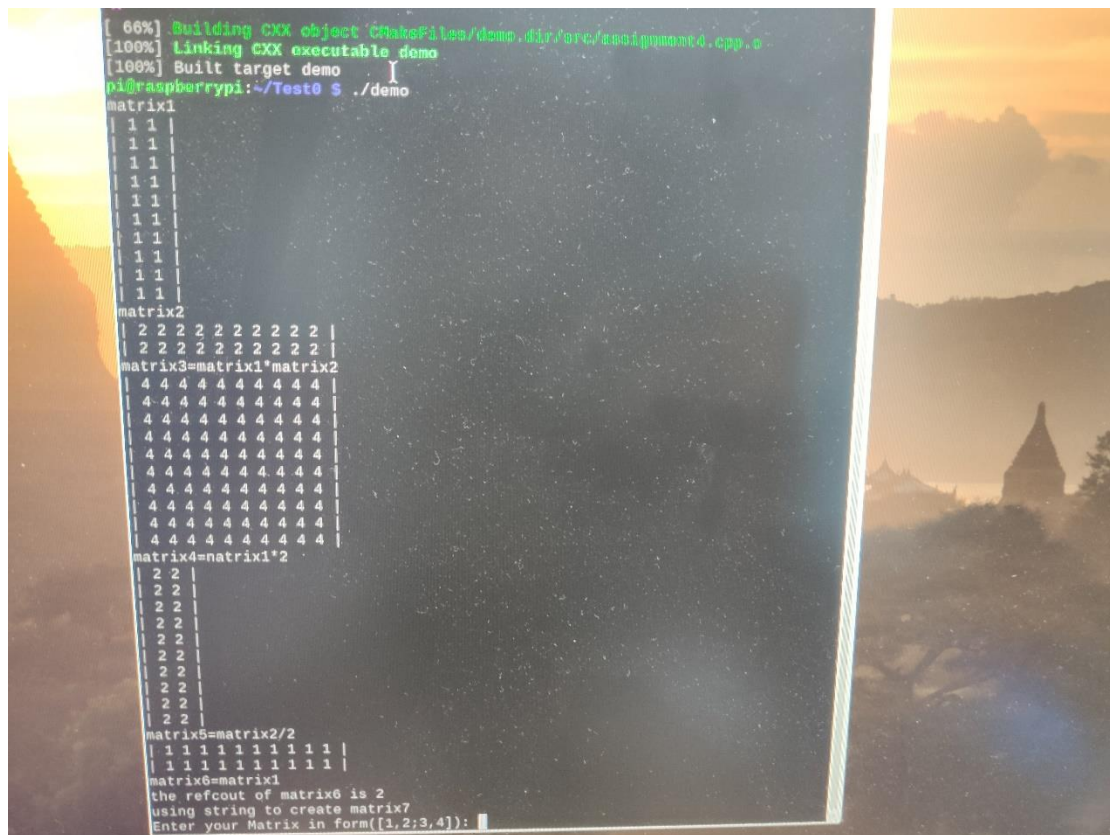
```
pi@raspberrypi ~ $ cd Test0
pi@raspberrypi:~/Test0 $ cmake Test
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/Test0
pi@raspberrypi:~/Test0 $ make
Scanning dependencies of target demo
[ 33%] Building CXX object CMakeFiles/demo.dir/src/Matrix.cpp.o
/home/pi/Test0/Test/src/Matrix.cpp: In function 'bool testMatrix(std::__cxx11::string)':
/home/pi/Test0/Test/src/Matrix.cpp:101:1: warning: control reaches end of non-void function [-Wreturn-type]
}
^
/home/pi/Test0/Test/src/Matrix.cpp: In function 'Matrix creatMatWithString()':
/home/pi/Test0/Test/src/Matrix.cpp:418:1: warning: control reaches end of non-void function [-Wreturn-type]
}
^
[ 66%] Building CXX object CMakeFiles/demo.dir/src/assignment4.cpp.o
[100%] Linking CXX executable demo
[100%] Built target demo
pi@raspberrypi:~/Test0 $
```

我用 U 盘将文件考到 Test0 文件夹中，文件夹名为 Test。然后执行 cmake 指令
生 成 Makefile 文 件



随后在用 make 指令，通过 makefile 文件生成可执行文件 demo

最 后 运 行 demo 文 件 即 可

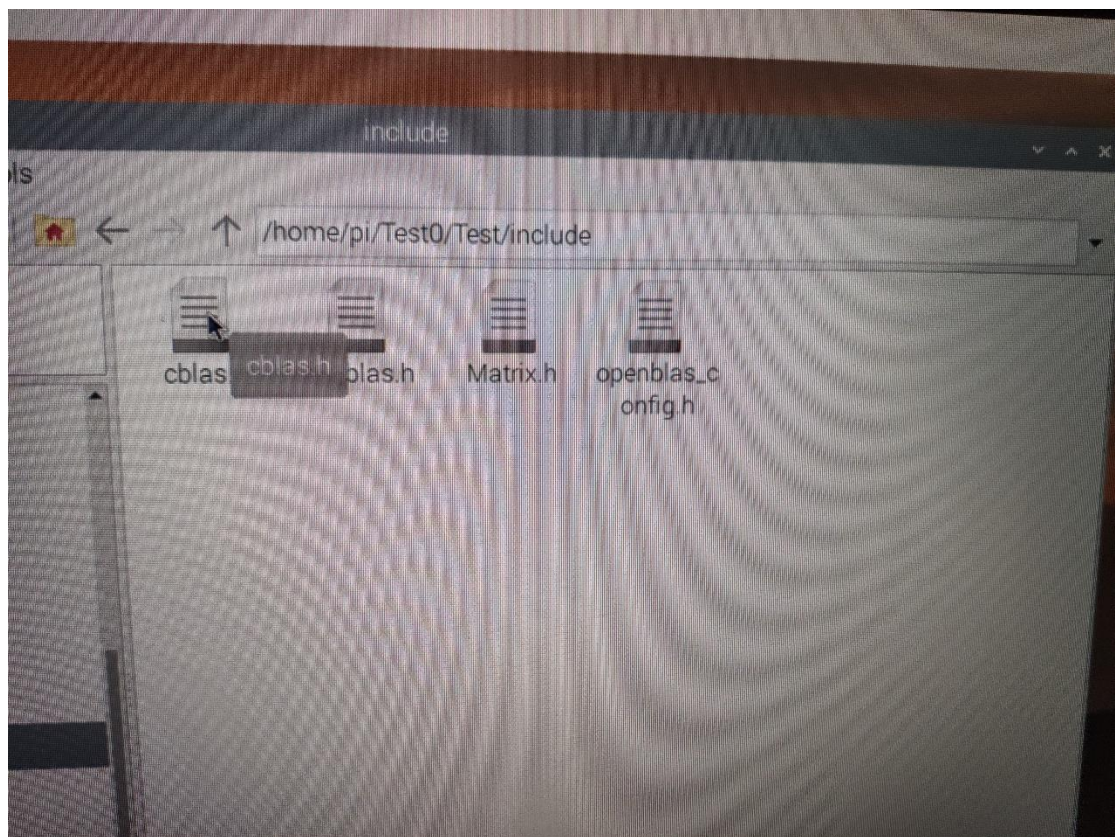
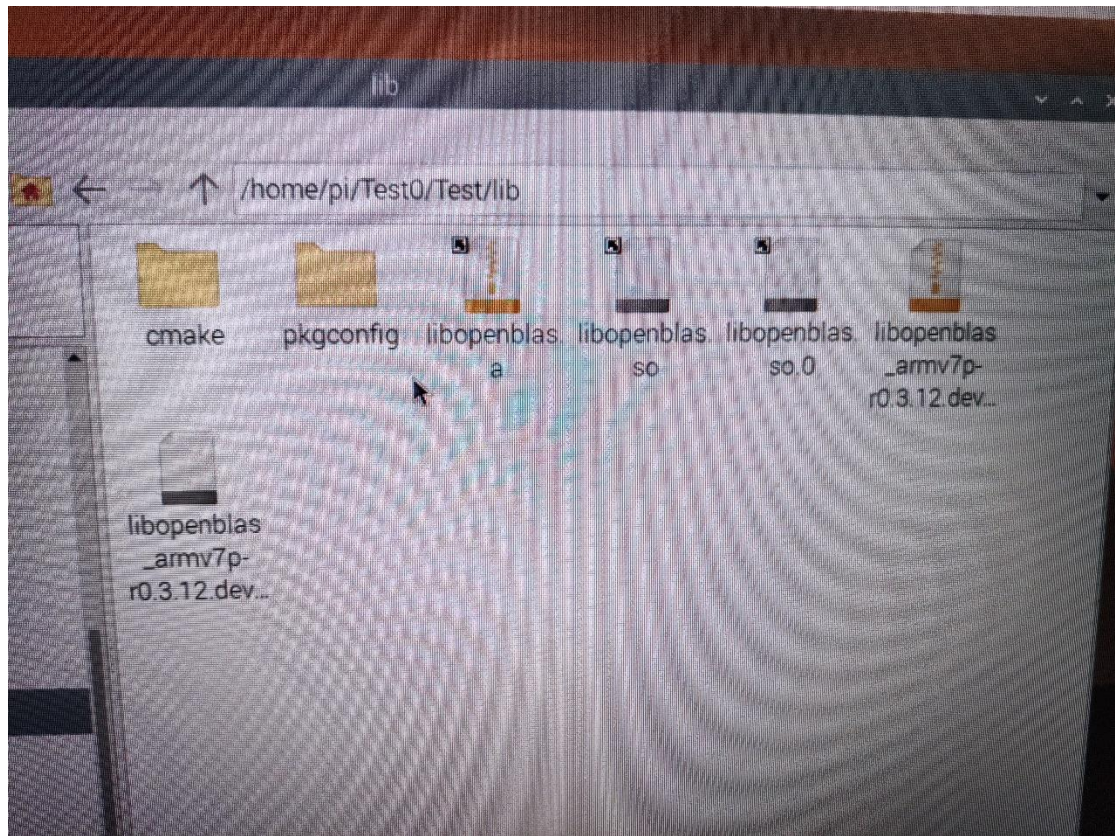


```
[ 66%] Building CXX object CMakeFiles/demo.dir/src/assignment4.cpp.o
[100%] Linking CXX executable demo
[100%] Built target demo
pi@raspberrypi:~/Test0 $ ./demo
matrix1
| 1 1 |
| 1 1 |
| 1 1 |
| 1 1 |
| 1 1 |
| 1 1 |
| 1 1 |
| 1 1 |
| 1 1 |
| 1 1 |
| 1 1 |
matrix2
| 2 2 2 2 2 2 2 2 2 2 |
| 2 2 2 2 2 2 2 2 2 2 |
matrix3=matrix1*matrix2
| 4 4 4 4 4 4 4 4 4 4 |
| 4 4 4 4 4 4 4 4 4 4 |
| 4 4 4 4 4 4 4 4 4 4 |
| 4 4 4 4 4 4 4 4 4 4 |
| 4 4 4 4 4 4 4 4 4 4 |
| 4 4 4 4 4 4 4 4 4 4 |
| 4 4 4 4 4 4 4 4 4 4 |
| 4 4 4 4 4 4 4 4 4 4 |
| 4 4 4 4 4 4 4 4 4 4 |
| 4 4 4 4 4 4 4 4 4 4 |
matrix4=matrix1*2
| 2 2 |
| 2 2 |
| 2 2 |
| 2 2 |
| 2 2 |
| 2 2 |
| 2 2 |
| 2 2 |
| 2 2 |
| 2 2 |
matrix5=matrix2/2
| 1 1 1 1 1 1 1 1 1 1 |
| 1 1 1 1 1 1 1 1 1 1 |
matrix6=matrix1
the refcount of matrix6 is 2
using string to create matrix7
Enter your Matrix in form[[1,2,3,4]]:
```

可以看到此时程序已经可以正常运行，但事情有这么简单吗。

由于 arm 开发板的环境与 x86 不同，所以不能直接使用自带的 x86 包，因此需要从 github 克隆一份 openblas，并用 cmake 生成可在 arm 使用的库，具体操作参考 <https://www.cnblogs.com/qujingtongxiao/p/10197784.html>

在用 cmake 和 make 指令将 openblas 库安装到指定位置后，要将 include 和 lib 的对应文件拷贝到项目所在的 include 和 lib 对应位置。如图：



这个时候再像之前如此操作便可以再 arm 开发板上成功运行使用 openblas 库

的程序

Part 4 总体概述和心路历程

可以说本次 as 的编程，并不像前几次 as 和期中 pro 那样，难点在各种优化上。在经历了前几次 as 的洗礼后，我们的代码水平有了长足长进，因此应对运算符重载已经不像前几次那么吃力。本次编程的重点更多的是程序在 arm 开发板上使用需要做到的通用性，如果仅仅像之前几次作业那样使用仅仅能在 x86 架构 https://github.com/Antoniano1963/vscode_cpp/tree/master/cpp_window/Week12/assignment4 下使用的 avx2 指令集进行加速的话，那么这个程序的通用性就很差，不能在 arm 环境下成功运行。解决这个问题方法便是使用 cmake 和 openblas 额外库，来对需要额外指令集的地方实现跨平台的处理。通过本次 as，我简单了解了 cmake 的使用，明白了如何编写 CMakeLists.txt 文件，并学会了用 cmake 编译 opencv 的源代码，得以更好的了解 opencv 工程。

Github:

https://github.com/Antoniano1963/vscode_cpp/tree/master/cpp_window/Week12/assignment4