Project 5: A Class for Matrices

12010508华羽霄

1. Design a class for matrices, and the class should contain the data of a matrix and related information such the number of rows, the number of columns, the number of channels, etc.

```
template<class T>
class Matrix
{
    private:
    shared_ptr<T> data;
    size_t row;
    size_t col;
    //这里偷懒了，我把矩阵的通道统一定义为1
    size_t channel;

    public:
    Matrix(){
        this->row = 1;
        this->col = 1;
        this->channel=1;
        this->data=nullptr;
    }

    Matrix(size_t& row, size_t& col, std::shared_ptr<T>& data) {
        this->row = row;
        this->col = col;
        this->channel=1;
        this->data = data;
    }
}
```

2. The class should support different data types. It means that the matrix elements can be unsigned char, short, int, float, double, etc.

```
//宏定义数据类型，可以按需修改
// #define dataType unsigned char
// #define dataType short
// #define dataType int
#define dataType float
// #define dataType double
```

3. Do not use memory hard copy if a matrix object is assigned to another. Please carefully handle the memory management to avoid memory leaks and to release memory multiple times.

```
//一开始我是用宏定义规定矩阵大小，但我发现这样创建的数据类型是int，后来改成在main方法里直接申明
    size_t nSize=1000;
    size_t total=nSize*nSize;

    //这是我最开始采用的方法，静态指针，但显然不符合要求，容易产生内存冲突
    // static dataType * p1;
    // p1=(dataType*)malloc(total*sizeof(dataType));
    //之后我又试了智能指针，但是初始化的语法不对
    // shared_ptr<dataType> sp1=make_shared<dataType>(total);
    // sp1=(dataType*)malloc(total*sizeof(dataType));

    shared_ptr<dataType> ptr1(new dataType[total],[](dataType *p){delete [] p;});
    shared_ptr<dataType> ptr2(new dataType[total],[](dataType *p){delete [] p;});
    shared_ptr<dataType> ptr3(new dataType[total],[](dataType *p){delete [] p;});

    for(size_t each=0;each<total;each++){
        ptr1.get()[each]=(dataType)rand()/RAND_MAX;
        ptr2.get()[each]=(dataType)rand()/RAND_MAX;
        ptr3.get()[each]=0;
  }

    //这是最开始的用静态指针的方法创建矩阵
    // auto matC = Matrix<dataType>(nSize,nSize,p1);

    Matrix<dataType> matA = Matrix<dataType>(nSize,nSize,ptr1);
    Matrix<dataType> matB = Matrix<dataType>(nSize,nSize,ptr2);
    Matrix<dataType> matC = Matrix<dataType>(nSize,nSize,ptr3);
```

4. Implement some frequently used operators including but not limited to =, == , +, -, *, etc. Surely the matrix multiplication in Project 4 can be included.

```
//为了判断矩阵大小是否相等，另外写的一个方法
    void if_legal(Matrix<T> matA, Matrix<T> matB){
        if(matA.row!=matB.col||matA.col!=matB.row){
            cerr<< "size not match" << endl;
            abort();
        }
    }
```

(1) =

```
void operator=(Matrix mat){
        this->row = mat.row;
        this->col = mat.col;
        //释放智能指针所申请的空间，并重新分配内容
        this->data.reset();
        this->data=mat.data;

        //想简化代码，但不知道是什么原因，下面这句话一直报错
        // this->data.reset(mat.data);
    }
```

```
start = clock();
    matC = matA;
    end = clock();
    duration = (end-start)/1000;
    printf("assign finished, assign time: %d ms\n",duration);
```

```
(base) hyx20020222@huayuxiaodeMacBook-Pro pro5 % g++ main.cpp --std=c++17
(base) hyx20020222@huayuxiaodeMacBook-Pro pro5 % ./a.out
assign finished, assign time: 0 ms
```

(2) ==

```
bool operator==(Matrix mat){
        //首先判断矩阵大小是否相等
        if(this->row==mat.row&&this->col==mat.col){
            for(size_t i = 0; i < mat.col*mat.row; i++){
                //对矩阵中每一个元素进行判断
                if(mat.data.get()[i]!=this->data.get()[i]){
                    return false;
                }
                return true;
            }
        }
        return false;
    }
```

```
start = clock();
    bool result=(matC == matA);
    end = clock();
    duration = (end-start)/1000;
    cout<<"compare result: "<<result<<", ";
    printf("compare time: %d ms\n",duration);
```

```
(base) hyx20020222@huayuxiaodeMacBook-Pro pro5 % g++ main.cpp --std=c++17
(base) hyx20020222@huayuxiaodeMacBook-Pro pro5 % ./a.out
compare result: 0, compare time: 0 ms
```

(3) +, -

```
//加减法道理都一样，这里我就以加法为例
Matrix operator+(float num){
        shared_ptr<T> result = make_shared<T>(this->row*this->col);

        //这是我一开始写的，但后来优化时发现，只需要一层循环就可以完成
        // for(size_t c = 0; c < col; c++){
        //     for (size_t r = 0; r < row; r++){
        //         result.get()[c*this->row+r]=this->data.get()[c*this->row+r]+num;
        //     }
        // }

        for(size_t each=0;each<row*col;each++){
            //矩阵数加，对矩阵中每一个元素做加法
            result.get()[each]=this->data.get()[each]+num;
        }
        return Matrix(this->row,this->col,result);
    }

    Matrix operator+(Matrix mat){
        if_legal(*this,mat);

        shared_ptr<T> result = make_shared<T>(this->row*this->col);
        for(size_t c = 0; c < col; c ++){
            for (size_t r = 0; r < row; r++){
                result.get()[c*this->row+r]=this->data.get()[c*this->row+r]+mat.data.get ()[c*this->row+r];
            }
        }
        return Matrix(this->row,this->col,result);
    }
```

```
start = clock();
    matC = matA+1;
    end = clock();
    duration = (end-start)/1000;
    printf("numerical add, add time: %d ms\n",duration);

    start = clock();
    matC = matA + matB;
    end = clock();
    duration = (end-start)/1000;
    printf("matrix add, add time: %d ms\n",duration);
```

```
(base) hyx20020222@huayuxiaodeMacBook-Pro pro5 % g++ main.cpp --std=c++17
(base) hyx20020222@huayuxiaodeMacBook-Pro pro5 % ./a.out
numerical add, add time: 5 ms
matrix add, add time: 6 ms
```

(4) *

```
Matrix operator*(float num){
        shared_ptr<T> result = make_shared<T>(this->row*this->col);

        for(size_t each=0;each<row*col;each++){
            result.get()[each]=this->data.get()[each]*num;
        }

        return Matrix(this->row,this->col,result);
    }

    Matrix operator*(Matrix<T> mat){
        if_legal(*this,mat);

        shared_ptr<T> result = make_shared<T>(mat.row*mat.col);

        //使用 openblas 计算，不清楚什么原因，一直说无法调用该函数
        // cblas_sgemm(CblasRowMajor,CblasNoTrans,CblasNoTrans,
        // mat.col,mat.col,mat.col,1.0,this->data,this->row,mat.data,mat.row,0.0,result,mat.row);

        for(size_t each=0;each<row*col;each++){
            result.get()[each]=this->data.get()[each]*mat.data.get()[each];
        }

        return Matrix(this->row,this->col,result);
    }
```

```
start = clock();
    matC = matA*2;
    end = clock();
    duration = (end-start)/1000;
    printf("numerical product time: %d ms\n",duration);

    start = clock();
    matC = matA*matB;
    end = clock();
    duration = (end-start)/1000;
    printf("matrix product time: %d ms\n",duration);
```

```
(base) hyx20020222@huayuxiaodeMacBook-Pro pro5 % g++ main.cpp --std=c++17
(base) hyx20020222@huayuxiaodeMacBook-Pro pro5 % ./a.out
numerical product time: 4 ms
matrix product time: 5 ms
```

(5) /

```
Matrix operator/(float num){
        //检查被除数是否为0，尤其注意不能直接用"num==0"，因为float并不精确
        if(fabs(num) < FLT_EPSILON){
```

```
            cerr<<"divided by zero"<<endl;
            abort();
        }

        shared_ptr<T> result = make_shared<T>(this->row*this->col);

        for(size_t each=0;each<row*col;each++){
            result.get()[each]=this->data.get()[each]/num;
        }

        return Matrix(this->row,this->col,result);
    }

    Matrix operator/(Matrix<T> mat){
        if_legal(*this,mat);

        //提前检查被除矩阵中每个元素是否为0，防止程序出错
        for(size_t each=0;each<row*col;each++){
            if(fabs(mat.data.get()[each]) < FLT_EPSILON){
                cerr<<"divided by zero"<<endl;
                abort();
            }
        }

        shared_ptr<T> result = make_shared<T>(mat.row*mat.col);

        for(size_t each=0;each<row*col;each++){
            result.get()[each]=this->data.get()[each]*mat.data.get()[each];
        }

        return Matrix(this->row,this->col,result);
    }
```

```
start = clock();
    matC = matA/2;
    end = clock();
    duration = (end-start)/1000;
    printf("numerical divide time: %d ms\n",duration);

    start = clock();
    matC = matA/matB;
    end = clock();
    duration = (end-start)/1000;
    printf("matrix divide time: %d ms\n",duration);
```

```
(base) hyx20020222@huayuxiaodeMacBook-Pro pro5 % g++ main.cpp --std=c++17
(base) hyx20020222@huayuxiaodeMacBook-Pro pro5 % ./a.out
numerical divide time: 4 ms
matrix divide time: 9 ms
```

5. Implement region of interest (ROI) to avoid memory hard copy.

网上搜了一圈，了解到opencv里面，ROI是用于规定矩阵聚焦的范围，但是怎么应用到内存管理，我还不太明白。

6. Test your program on X86 and ARM platforms, and describe the differences.

以上结果均运行在macbook pro m1 2020，这是一台arm笔记本，在我的另一台magicbook 14上运行时，只有前两个方法（=，==）有输出，其他方法没有任何报错，也没有任何输出，程序直接结束。

参考文献

- https://frama-c.com/2013/05/09/Definition-of-FLT_EPSILON.html
- https://qastack.cn/programming/29383/converting-bool-to-text-in-c
- https://blog.csdn.net/weixin_34247155/article/details/94732034
- https://blog.csdn.net/lzn211/article/details/109147985
- https://learn.microsoft.com/zh-cn/cpp/cpp/how-to-create-and-use-shared-ptr-instances?view=msvc-170
- https://www.cnblogs.com/wxquare/p/4759020.html
- https://blog.csdn.net/weixin_43705457/article/details/97617676
- https://www.cnblogs.com/xiaoshiwang/p/9726511.html
- https://www.cnblogs.com/fnlingnzb-learner/p/6903966.html
- https://zhuanlan.zhihu.com/p/36995444
- https://www.cnblogs.com/jikexianfeng/articles/5651661.html
- https://blog.csdn.net/starboybenben/article/details/49802555