Project 3 矩阵类

Name:郭一潼 SID:11911702 CS205 Project3 Report 代码在如下链接
https://github.com/Godblessmycode1/CS205

1. 介绍

1.1 Project要求介绍

1. 只能使用c语言
2. 矩阵类包括行，列以及数据
3. 矩阵储存浮点数
4. 实现特定API
5. API易于使用

1.2 Project完成情况介绍

完成了矩阵类的设计，并实现了如下API：

1. 创建矩阵createMatrix
2. 矩阵赋值valueMatrix
3. 矩阵相加addMatrix
4. 矩阵复制copyMatrix
5. 矩阵相乘mulMatrix
6. 矩阵相减subtractMatrix
7. 矩阵加常数addScalar
8. 矩阵减常数subScalar

9. 矩阵乘常数mulScalar
10. 查找最大值最小值minValue/maxValue
11. 回收矩阵类的空间deleteMatrix
12. 求矩阵行列式detrMatrix
13. 求矩阵的逆reverseMatrix

1.3 开发环境

- x86_64
    - vscode (version 1.71)
    - WSL (version 2)
    - Ubuntu(22.04)
    - g++(11.2.0)

2. 基本数据结构及代码思路

本次Project采用的矩阵数据结构，用int储存矩阵的长和宽，用float*来储存二维数组（二维数组的本质也是一维数组从内存角度）。代码如下。

```
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
#include <float.h>
#include<math.h>
typedef struct
{
    int row;
    int column;
    float* matrix_data;
}Matrix;
```

本次project的代码思路在于可用性与节省内存，为了方便用户使用，**需要将结果保存的矩阵操作函数返回值为Matrix类型指针，在能通过已知操作数矩阵的信息判断储存操作结果的矩阵维度的情况下，不需要createMatrix。同时createMatrix和valueMatrix方法进行分开，方便了更改矩阵的赋值。**
同时为了节省内存，使用了方法freeMatrix。在函数调用过程中给，如果遇到后续不需要的指针变量，函数内部自动调用了free方法。本次project内存申请全部使用malloc函数

3.基本方法介绍

在创建矩阵时会判断矩阵的行与列参数是否为正，在其他函数调用过程中，若参数为指针，会首先判断其是否为NULL。

3.1 createMatrix

createMatrix的代码思路是，为了方便使用，返回变量是Matrix* 而非void，符合用户使用习惯。调用时参数为(Matrix* , int row,int column)。先申请Matrix*,同时再申请一个float*,内存大小为sizeof(float)* row* column,同时调用memset将float指针范围内的内容全部初始化为0。后续通过valueMatrix赋值。

```
Matrix* createMatrix(Matrix* matrix,int row,int column){ //矩阵初始化
      if(row>0&&column>0){
           matrix=(Matrix*)malloc(sizeof(matrix)); //方便free
           matrix->row=row;
           matrix->column=column;
           matrix->matrix_data=(float*)malloc(sizeof(float)*row*column); //方便
free
           memset(matrix->matrix_data,0,sizeof(float)*row*column);//初始化为0
           return matrix;
      }
      else{
        printf("The row or column is not positive\n");
        return NULL;
      }
}
```

3.2 valueMatrix

valueMatrix参数为(Matrix* input,float* array)，然后通过传入float* array,通过memcpy函数将array有效范围内的内存值copy到input->matrix_data里,达到了不影响array内部值又赋值的目的。

```
void valueMatrix(Matrix* input,float* array){
 if (input!=NULL&&array!=NULL)
 {
     memcpy(input->matrix_data,array,sizeof(float)*input->row*input->column);
 }
 else{
    printf("Input an invalid pointer in valueMatrix\n");
 }
}
```

3.3 addMatrix/subMatrix

addMatrix参数为(Matrix* res,Matrix* matrix1,Matrix* matrix2),再执行矩阵相加的代码逻辑前，需要先判断他们的大小是否匹配（长与宽相同）。再将matrix1与matrix2的每个元素相加结果储存到res中,相减同理。

```
Matrix* addMatrix(Matrix* res,Matrix* matrix1,Matrix* matrix2){
    if(matrix1==NULL||matrix2==NULL){
        printf("The input is not valid in addMatrix\n");
        return NULL;
    }
    if(matrix1->row==matrix2->row&&matrix1->column==matrix2->column){
        res=createMatrix(res,matrix1->row,matrix1->column);
        float* res_data=res->matrix_data;
        float* matrix1_data=matrix1->matrix_data;
        float* matrix2_data=matrix2->matrix_data;
            for(int i=0;i<matrix1->row;i++){
                for(int j=0;j<matrix2->column;j++){
```

```
                    (*res_data++)=(*matrix1_data++)+(*matrix2_data++);//相加并移动
一个float单位
                }
            }
        return res;
    }
    else{
        printf("The matrix size does not match add\n");
        printf("The row and column of matrix1 is %d,%d\n",matrix1->row,matrix1-
>column);
        printf("The row and column of matrix2 is %d,%d\n",matrix2->row,matrix2-
>column);
        return NULL;
    }
}
Matrix* subtractMatrix(Matrix* res,Matrix* matrix1,Matrix* matrix2){ //第一个矩阵减
第二个矩阵
     if(matrix1==NULL||matrix2==NULL){
        printf("The input is not valid in subtractMatrix\n");
        return NULL;
     }
     if(matrix1->row==matrix2->row&&matrix1->column==matrix2->column){
        res=createMatrix(res,matrix1->row,matrix1->column);
        float* res_data=res->matrix_data;
        float* matrix1_data=matrix1->matrix_data;
        float* matrix2_data=matrix2->matrix_data;
            for(int i=0;i<matrix1->row;i++){
                for(int j=0;j<matrix2->column;j++){
                    (*res_data++)=(*matrix1_data++)-(*matrix2_data++);//相减并移动
一个float单位
                }
            }
        return res;
    }
    else{
        printf("The matrix size does not match substract\n");
        printf("The row and column of matrix1 is %d,%d\n",matrix1->row,matrix1-
>column);
        printf("The row and column of matrix2 is %d,%d\n",matrix2->row,matrix2-
>column);
    }
}
```

3.4 mulMatrix

mulMatrix参数为(Matrix* res,Matrix* matrix1,Matrix* matrix2),代码逻辑是矩阵乘法的规则，乘法规则链接如下：https://baike.baidu.com/item/矩阵乘法/5446029

```
Matrix* mulMatrix(Matrix* res,Matrix* matrix1,Matrix* matrix2){ //here is the code
    if(matrix1==NULL||matrix2==NULL){
        printf("The input is not valid in mulMatrix\n");
```

```
        return NULL;
    }
    if(matrix1->column==matrix2->row&&matrix1->row==matrix2->column){
        res=createMatrix(res,matrix1->row,matrix2->column);
        float* res_data=res->matrix_data;
        float* matrix1_data=matrix1->matrix_data;
        float* matrix2_data=matrix2->matrix_data;
        for(int i=0;i<matrix1->row;i++){
            for(int j=0;j<matrix2->column;j++){
                for(int k=0;k<matrix1->column;k++){
                    res_data[i*matrix2->column+j]+=matrix1_data[i*matrix1-
>column+k]*matrix2_data[k*matrix2->column+j];
                }
            }
        }
        return res;
    }
    else{
        printf("The matrix size does not match multiply\n");
        printf("The row and column of matrix1 is %d,%d\n",matrix1->row,matrix1-
>column);
        printf("The row and column of matrix2 is %d,%d\n",matrix2->row,matrix2-
>column);
        return NULL;
    }
}
```

3.5 addScalar/subScalar/MulScalar

参数为(Matrix* input,float scalar),代码逻辑为将input中每个元素与scalar都进行加减和乘。

```
void addScalar(Matrix* input,float scalar){
    if(input==NULL){
        printf("The input matrix is not valid in addScalar\n");
        return;
    }
    else{
        float* data=input->matrix_data;
        for(int i=0;i<input->row;i++){
            for(int j=0;j<input->column;j++){
                *(data)=*(data)+scalar;
                data++;
            }
        }
    }
}
void subScalar(Matrix* input,float scalar){
    if(input==NULL){
        printf("The input matrix is not valid in subScalar\n");
        return;
    }
```

```
    else{
        float* data=input->matrix_data;
        for(int i=0;i<input->row;i++){
            for(int j=0;j<input->column;j++){
                *(data)=*(data)-scalar;
                data++;
            }
        }
    }
}
void mulScalar(Matrix* input,float scalar){
     if(input==NULL){
        printf("The input matrix is not valid\n");
        return;
    }
    else{
        float* data=input->matrix_data;
        for(int i=0;i<input->row;i++){
            for(int j=0;j<input->column;j++){
                *(data)=*(data)*scalar;
                data++;
            }
        }
    }
}
```

3.6 minValue/maxValue

参数为(Matrix* input,float* res),返回值为int,因为可能输入Matrix*为NULL,不存在最大值/最小值。所以返回值设置为int类型，只有返回1时，将结果保存在res里，同时res的值也才有效。

```
int minValue(Matrix* input,float* res){
    if(input==NULL){
        printf("The input matrix is not valid in minValue\n");
        return 0;
    }
    else{
       float temp=FLT_MAX;
       float* data=input->matrix_data;
       for(int i=0;i<input->row;i++){
        for(int j=0;j<input->column;j++){
            if(*(data)<temp){
                temp=*(data);
                data++;
            }
        }
       }
       *res=temp;
       return 1;
    }
}
```

```
int maxValue(Matrix* input,float* res){
    if(input==NULL){
        printf("The input matrix is not valid in maxValue\n");
        return 0;
    }
    else{
       float temp=FLT_MIN;
       float* data=input->matrix_data;
       for(int i=0;i<input->row;i++){
        for(int j=0;j<input->column;j++){
            if(*(data)>temp){
                temp=*(data);
                data++;
            }
        }
       }
       *res=temp;
       return 1;
    }
}
```

3.7 deleteMatrix

经过输入检验后，参数为(Matrix* matrix),首先要free(matrix->matrix_data),如果直接free(matrix)的话，matrix->matrix_data的有效内存相当于永远找不到了，这样会造成内存泄露。因此首先free(matrix->matrix_data);

```
void deleteMatrix(Matrix* matrix){
    free(matrix->matrix_data);
    free(matrix);
}
```

3.8 transposedMatrix

转置矩阵规则如下https://baike.baidu.com/item/转置矩阵/3380917

```
Matrix* transposedMatrix(Matrix* src,Matrix* dst){//src是原矩阵，dst是目标矩阵。
    if (src==NULL)
    {
        printf("The input is not valid in transposedMatrix\n");
        return NULL;
    }
    dst=createMatrix(dst,src->column,src->row);//转置行列互换
    for(int i=0;i<dst->row;i++){
        for(int j=0;j<dst->column;j++){
            dst->matrix_data[i*dst->column+j]=src->matrix_data[j*src->column+i];
        }
    }
    return dst;
}
```

3.9 copyMatrix

矩阵拷贝，这里的值一定要通过memcpy,如果让两矩阵指针指向一处，那样会出现数据共享，产生问题。

```
Matrix* copyMatrix(Matrix* src,Matrix* dst){//src是源，dst是目的地
      if(src!=NULL){
         createMatrix(dst,src->row,src->column);
         valueMatrix(dst,src->matrix_data);
         return dst;
      }
      else{
        printf("At least one pointer is not valid in copyMatrix\n");
        return NULL;
      }
}
```

3.10 detrMatrix

此函数用于求行与列相等矩阵的行列式。行列式计算规则如下https://baike.baidu.com/item/行列式依行展开/22772377,通过资料我们可以发现，通过递归调用解决问题。

```
int detrMatrix(Matrix* matrix,float* res){ //求行列式
     if(matrix==NULL||res==NULL){
        printf("Input invalid pointer in detrMatrix\n");
         return 0;
     }
     else if (matrix->row!=matrix->column)
     {  printf("The matrix is not a square matrix , no determinet\n");
        return 0;
     }
     else{
       *res=det(matrix->matrix_data,matrix->row,matrix->row);
       return 1;
     }
}
float det(float* array,int n,int max){//array 是矩阵,n代表矩阵的长和宽,max代表输入
matrix的长度和宽度即递归调用中的最大维度（对于后续递归调用过程来说要释放内存的）。
     int row=0;//第一行本行列式计算一直用第一行做展开
     float M=0;//用于计算余子式的值;
     float res=0;//用于计算行列式结果
     if(n==1){
        return array[0];
     }
     else {
        for(int index=0;index<n;index++){
            M=minor(array,index,n,max); //M为除去这一行这一列的余子式值。
            float component=0;
            if(index%2==0){
```

```
                component=1.0f;
            }
            else{
                component=-1.0f;
            }
            res+=component*array[index]*M;
        }
    }
    if(n!=max){
        free(array); //这里一定要free! !!!!!!要不然会造成内存泄漏
    }
    return res;
}
float minor(float* array,int index,int n,int max){ //n代表为n阶余子式.index代表选取的的行数为展开的行。
    float* sub_array=(float*)malloc(sizeof(float)*(n-1)*(n-1));
    memset(sub_array,0,sizeof(float)*(n-1)*(n-1));
    int row_length=n-1;
    int column_length=n-1;
    for(int i=0;i<row_length;i++){
        for(int j=0;j<column_length;j++){
            if (j<index)
            {
                sub_array[i*column_length+j]=array[(i+1)*n+j];
            }
            else{
                sub_array[i*column_length+j]=array[(i+1)*n+j+1];
            }
        }
    }
    return det(sub_array,n-1,max);
}
```

3.11 reverseMatrix

可你矩阵求法如下https://baike.baidu.com/item/可逆矩阵/11035614,首先需要知道矩阵的行列式determinant，然后再transposedMatrix，再mulScalar(1/determinant),则得到可逆矩阵。

```
Matrix* reverseMatrix(Matrix* src,Matrix* dst){ //将src的逆矩阵放入dst中
        float determinant=0;
        if (src==NULL)
        {   printf("Src is not valid in reverseMatrix\n");
            return NULL;
        }
        else if(detrMatrix(src,&determinant)!=1){
            return NULL;
        }
        else{
            if(determinant==0){
                printf("The determinant is 0, no reverse matrix");
                return NULL;
```

```
                }
                else{
                    dst=transposedMatrix(src,dst);
                    mulScalar(dst,1.0f/determinant);
                    return dst;
                }
            }
    }
```

4. 结果

测试样例位于test.cpp中，运行即可。

```
The matrix1 is
1.000000 2.000000 3.000000 4.000000
5.000000 6.000000 7.000000 8.000000
9.000000 10.000000 11.000000 12.000000
The matrix2 is
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
10.000000 11.000000 12.000000
The matrix3 is
2.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
The matrix4 is
1.000000 2.000000 3.000000 4.000000
5.000000 6.000000 7.000000 8.000000
9.000000 10.000000 11.000000 12.000000
The copy matrix5 is
1.000000 2.000000 3.000000 4.000000
5.000000 6.000000 7.000000 8.000000
9.000000 10.000000 11.000000 12.000000
The matrix size does not match addMatrix
The row and column of matrix1 is 3,4
The row and column of matrix2 is 4,3
The res1 matrix after matrix1 multiply matrix2 is
70.000000 80.000000 90.000000
158.000000 184.000000 210.000000
246.000000 288.000000 330.000000
The res1 after addScalar is
71.000000 81.000000 91.000000
159.000000 185.000000 211.000000
247.000000 289.000000 331.000000
The res1 after subScalar is
70.000000 80.000000 90.000000
158.000000 184.000000 210.000000
246.000000 288.000000 330.000000
The res1 after mulScalar is
420.000000 480.000000 540.000000
948.000000 1104.000000 1260.000000
1476.000000 1728.000000 1980.000000
The tran matrix is
420.000000 948.000000 1476.000000
480.000000 1104.000000 1728.000000
540.000000 1260.000000 1980.000000
```

```
The max value is 1980.000000
The min value is 420.000000
The determinant of matrix3 is -3.000000
The reverse matrix of matrix3 is
-0.666667 -1.333333 -2.333333
-0.666667 -1.666667 -2.666667
-1.000000 -2.000000 -3.000000
```

5. 结论

通过本次lab我了解了malloc和free的具体用法和用处，以及更好的了解了如何提高可用性，以及内存泄露的危害，对于指针的理解也加强了，能够更灵活的利用指针。
同时发现访问一个指针/数组 Data* array里index为i的数据时,可以使用 array[i]的形式，也可以使用*(array+i),了解后，个人可能喜欢前者。