Package

线性代数工具包

欧阳浩岚 18340133 *数据科学与计算机学院 计算机类5班*

运行界面

欢迎界面

```
Welcome to my Linear Algebra Package!
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
There are several functions that you can use to help your calculation of linear algebra

        1.Find the Soltion of the Matrix
        2.Find The Inverse of the Matrix
        3.LU Factorization of the Matrix
        4.Calculate the Determinant of the Matrix
        5.Matrix Multiplication
        6.Scalr operation on Matrix
        7.QR Factorization of the Matrix
```

测试样例1

```
Find the solution of Ax=b

Enter the matrix A:
[1,-2,1;0,2,-8;-4,5,9]

Enter vector b:
[0;8;-9]

The solution set is :[29;16;3]

-----------------------------------------------------------------------------------------
```

```
Find the solution of Ax=b

Enter the matrix A:
[0,1,-4;2,-3,2;5,-8,7]

Enter vector b:
[8;1;1]

No solution!

-----------------------------------------------------------------------------------------
```

```
Find the solution of Ax=b

Enter the matrix A:
[3,5,-4;-3,-2,4;6,1,-8]

Enter vector b:
[7;-1;-4]

The solution set is : X3 * [4/3;0;1] + [-1;2;0] ( X3 is free variable )

--------------------------------------------------------------------------------
```

测试样例2

```
Enter the matrix that you want to find the inverse:
[2,5;-3,-7]


[

      -7      -5

       3       2
]


--------------------------------------------------------------------------------
```

```
Enter the matrix that you want to find the inverse:
[1,0;1,0]

Inverse Not Found!

--------------------------------------------------------------------------------
```

测试样例3

```
Enter the matrix that you want to do the LU factorization:
[2,4,-1,5,-2;-4,-5,3,-8,1;2,-5,-4,1,8;-6,0,7,-3,1]

[

       1       0       0       0

      -2       1       0       0

       1      -3       1       0

      -3       4       2       1
]

[

       2       4      -1       5      -2

       0       3       1       2      -3

       0       0       0       2       1

       0       0       0       0       5
]

--------------------------------------------------------------------------------
```

测试样例4

```
Enter the matrix that you want to calculate the determinant:
[1,5,0;2,4,-2;-1,-2,1]

The determinant of the matrix is 0

--------------------------------------------------------------------------------
```

```
Enter the matrix that you want to calculate the determinant:
[3,-7,8,9,-6;0,2,-5,7,3;0,0,1,5,0;0,0,2,4,-1;0,0,0,-2,0]

The determinant of the matrix is -12

--------------------------------------------------------------------------------
```

测试样例5

```
Select the multiplication you want to do(only a character):
a)Scalar Product
b)Matrix Multiplication:a

Enter the first matrix:
[1,0,1;0,1,0;1,0,1]

Enter the second matrix:
[1,0,1;0,1,0;1,0,1]

[

       1       0       1

       0       1       0

       1       0       1
]


--------------------------------------------------------------------------------
```

```
Select the multiplication you want to do(only a character):
a)Scalar Product
b)Matrix Multiplication:b

Enter the first matrix:
[1,0,1;0,1,0;1,0,1]

Enter the second matrix:
[1,0,1;0,1,0;1,0,1]

[

       2       0       2

       0       1       0

       2       0       2
]


--------------------------------------------------------------------------------
```

测试样例6

```
Please select the functions that you want to use(q/Q to quit the program):6

Enter your calculation formular:
2+[1,0,0;0,1,0;0,0,1]
[

       3       2       2

       2       3       2

       2       2       3
]

--------------------------------------------------------------------------------
```

```
Enter your calculation formular:
2-[1,0,0;0,1,0;0,0,1]
[

       1       2       2

       2       1       2

       2       2       1
]
```

```
Enter your calculation formular:
2*[1,0,0;0,1,0;0,0,1]
[

       2       0       0

       0       2       0

       0       0       2
]

--------------------------------------------------------------------------------
```

```
Enter your calculation formular:
[1,0,0;0,1,0;0,0,1]/2
[

      1/2      0       0

       0      1/2      0

       0       0      1/2
]

--------------------------------------------------------------------------------
```

测试样例7

```
Please enter the matrix that you want to do the QR factorization:
[1,0,0;1,1,0;1,1,1;1,1,1]

[

       1     -3/4      0

       1      1/4    -2/3

       1      1/4     1/3

       1      1/4     1/3
]

[

       1      3/4     1/2

       0       1      2/3

       0       0       1
]

--------------------------------------------------------------------------------
```

使用说明

输入输出格式

考虑到分数带来的精度问题，我使用了一个结构体去表示每个具体的数，而每个数包含有分子与分母两个int类型的变量。 故在输入输出时会支持分数形式的输入与输出，完全和手算结果一样(若要用精度表示只需要用一个double类型存储分子分母的商)
由于想要避免在输入矩阵的时候要求用户首先输入每一个矩阵的行和列数，我参考了 MATLAB 的输入格式

比如说：一个3*3矩阵
1 2 3
4 5 6
7 8 9
直接输入 `[1,2,3;4,5,6;7,8,9]` 即可(以逗号分隔每一行中不同列的数，以分号区分每一行的的输入)。

具体功能的输出界面说明

1.线性方程求解

按照程序提示，分别输入矩阵A和向量B，即可得到方程的解
相较于题目的要求，我还在我的工具包中额外添加了解集功能，即如果是多解情况，就把方程的解集求出来

2.矩阵的逆

按照上方输入矩阵格式输入一个矩阵，即可得到矩阵的逆

3.矩阵LU分解

同样是按照题目要求，输入一个矩阵即可得到两个矩阵。按顺序，第一个矩阵是L，第二个是U

4.行列式

输入一个矩阵，如果不是方阵则提示矩阵不存在行列式，如果是方阵则输出它的行列式

5.矩阵乘法

进入这个功能会有两个备选项，a)是点乘，b)是非点乘。按要求输入两个矩阵之后就可以得到相应的结果了

6.标量操作

考虑到这是一个四则运算的计算，我采取了允许用户自己输入计算表达式的方式。你可以让一个数字直接加矩阵
例：`[1,2,3;4,5,6;7,8,9]+1` 输出结果会是 `[2,3,4;5,6,7;8,9,10]` 你也可以是 `1+[1,2,3;4,5,6;7,8,9]`
这样也会输出 `[2,3,4;5,6,7;8,9,10]`
两种格式都是被允许的，同时，功能还支持加减乘除这四种运算

7.矩阵QR分解

虽然本程序支持分数表示，但是根号的表示我还没有想到怎么更简洁的表示，故此QR分解姑且算是半QR分解， 即我的Q的每个列向量没有化简到单位向量，这也是一个遗憾。
本功能的具体输入输出格式参照LU分解。

还未开发出的功能

虽然本工具包只有7个功能，但是矩阵的基本操作的函数我都已经写好了，包含有：三种基本行列变换、化简矩阵 到行接梯形矩阵与行最简形矩阵、求方阵的行列式、求矩阵的逆、QR分解
后续我将会改进QR分解，以至实现矩阵特征值的计算、对角化、SVD分解等更多更有用的功能！

代码

header.c

```
/**
 *    author:  ouyanghaolan
 *    created: 07.01.2019
**/
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
```

```
#include<math.h>
#include<string.h>
#define MAXStr 100000
/*Newly designed data type which can display fraction*/
typedef struct DATABASE{
    int numer;//分子
    int deno;//分母
}database;
/*-----------------------------------------------*/
/*struct matrix*/
typedef struct MATRIX{
    database **data;
    int row,col;
}matrix;
/*-----------------------------------------------*/
/* database operation */
database Add(database a1,database a2);// a1+a2
database Subtract(database a1,database a2);// a1-a2
database Multiply(database a1,database a2);// a1*a2
database Divide(database a1,database a2);// a1/a2
void CorrectData(database *data);// Applied only if deno<0
void gcd(database*data1);
void gcd_Row(database *row,int col);
/*-----------------------------------------------*/
/* Matrix Construction */
matrix *InitializeMatrix(int Row,int Col);
void ReleaseMatrix(matrix *mat);
matrix *ReadMatrix();
database read(char *str,int *index);
void PrintMatrix(const matrix mat);
/*-----------------------------------------------*/
/*Elementary row operation*/
void Replacement(const database row1[],database row2[],const int col,const database multip
void Interchange(database *row1,database *row2,const int col);
void Scaling(database *row,const database scales,const int col);
/*-------------------------------------------------------------------------------------------
/* basic matrix operation */
bool RowReduceToEchelonForm(matrix *mat);
void EchelonFormToReduceEchelonForm(matrix*mat);
void RowReduceToRowReduceEchelonForm(matrix *mat);
/*-----------------------------------------------*/
/* Operation on IO */
void print(database data);
/*-----------------------------------------------*/
/*funcions*/
database Determinant(const matrix mat1);
void FindSolution(matrix mata,matrix matb);
void SolutionSet(matrix mat);
matrix *FindInverse(matrix mat);
matrix *LUFactorization(matrix *mat);
matrix *MatrixMultiplication(matrix *mat1,matrix *mat2);
matrix *MatrixMultiplicationScalarProduct(matrix *mat1,matrix *mat2);
void ScalarOperation(char q,matrix *mat,database scalar,bool operand);
database InnerProduct(const database vector1[],const database vector2[],const int num);
matrix *QRFactorization(matrix *mat);
```

```
/*-----------------------------------------------------------------*/
/*functions UI display*/
void ui();
void FindSolutionDisplay();
void FindInverseDisplay();
void LUFactorizationDisplay();
void DeterminantDisplay();
void MatrixMultiplicationDisplay();
void ScalarOperationDisplay();
void QRFactorizationDisplay();
void selection(char op);
/*-----------------------------------------------------------------*/
```

main.c

```
/**
 *    author:  ouyanghaolan
 *    created: 07.01.2019
**/
#include"header.h"

int main(){
        ui();
        return 0;
}
```

basicfunction.c

```
/**
 *    author:  ouyanghaolan
 *    created: 07.01.2019
**/
#include"header.h"

/*Functions for newly design database*/
database Add(database a1,database a2){
        database result;
        result.numer=a1.numer * a2.deno +a1.deno *a2.numer;
        result.deno=a1.deno * a2.deno;
        if(result.deno<0){
                CorrectData(&result);
        }
        if(result.numer!=0&&result.deno!=1)
                gcd(&result);
        return result;
}
database Subtract(database a1,database a2){
        /* result=a1-a2 */
        database result;
```

```
        result.numer=a1.numer * a2.deno - a2.numer * a1.deno;
        result.deno=a1.deno * a2.deno;
        if(result.deno<0){
                CorrectData(&result);
        }
        if(result.numer!=0&&result.deno!=1)
                gcd(&result);
        return result;
}
database Multiply(database a1,database a2){
        database result={0,1};
        if(a1.numer==0||a2.numer==0)
                result.numer=0;
        else{
                result.numer=a1.numer * a2.numer;
                result.deno=a1.deno * a2.deno;
                if(result.deno<0){
                        CorrectData(&result);
                }
                if(result.numer!=0&&result.deno!=1)
                        gcd(&result);
        }
        return result;
}

database Divide(database a1,database a2){
        database result={0,1};//error needed to be added if a2=0
        if(a1.numer==0)
                result.numer=0;
        else{
                result.numer=a1.numer * a2.deno;
                result.deno=a1.deno * a2.numer;
                if(result.deno<0){
                        CorrectData(&result);
                }
                if(result.numer!=0&&result.deno!=1)
                        gcd(&result);
                }
        return result;
}
void CorrectData(database *data){
        data->deno*=(-1);
        data->numer*=(-1);
}
void gcd_Row(database *row,int col){
    int n,n1,n2,i,temp;
    for(i=1;i<=col;i++){
        if(row[i].numer==0)
            continue;
        n1=abs(row[i].numer),n2=row[i].deno;
        if(n1<n2){
            temp=n1;
            n1=n2;
            n2=temp;
        }
```

```
            n=n1;
            while(n!=0){
                n=n1%n2;
                n1=n2;
                n2=n;
            }
            row[i].numer/=n1;
            row[i].deno/=n1;
        }
    }
    /* To simplify my database*/
    void gcd(database*data1){
        int n,n1=abs(data1->numer),n2=data1->deno,temp;
        if(n1<n2){
            temp=n1;
            n1=n2;
            n2=temp;
        }
        n=n1;
        while(n!=0){
            n=n1%n2;
            n1=n2;
            n2=n;
        }
        data1->numer/=n1;
        data1->deno/=n1;
    }
    /*-----------------------------------------------------------------*/
    /*IO for my newly designed database*/
    database read(char *str,int *index){
            bool negative=false;
            database data={0,1};
            for(;!(str[*index]>='0'&&str[*index]<='9');(*index)++)
                    if(str[*index]=='-')
                            negative=true;
            for(;str[*index]>='0'&&str[*index]<='9';(*index)++)
                    data.numer=data.numer*10+str[*index]-'0';
        if(negative)
                    data.numer*=-1;
            if(str[*index]=='/'){
            if(str[*index]>='0'&&str[*index]<='9'){
                            data.deno=0;
                            for((*index)++;str[*index]>='0'&&str[*index]<='9';(*index)++)
                                    data.deno=data.deno*10+str[*index]-'0';
                    }
        }
            return data;
    }
    void print(database data){
            if(data.numer==0||data.deno==1)
                    printf("%d",data.numer);
            else
                    printf("%d/%d",data.numer,data.deno);
    }
    /*-----------------------------------------------------------------*/
```

```
/*row reduction of matrix*/
bool RowReduceToEchelonForm(matrix *mat){
    int i,j,counter=1,p;
    bool negative=false;
        bool status=false;
    database multiple;
    for(j=1;j<=mat->col&&counter<=mat->row;j++){
        status=false;
                if(mat->data[counter][j].numer==0){
            for(p=counter+1;p<=mat->row;p++){
                if(mat->data[p][j].numer!=0){
                    Interchange(mat->data[p],mat->data[counter],mat->col);
                    negative=!negative;
                                        status=true;
                    break;
                }
            }
        }
        else{
                status=true;
        }
        if(status==false)
            continue;
        for(i=counter+1;i<=mat->row;i++){
            if(mat->data[i][j].numer==0)
                continue;
            multiple=Multiply((database){-1,1},Divide(mat->data[i][j],mat->data[counter][j
            if(multiple.deno<0){
                CorrectData(&multiple);
            }
                        Replacement(mat->data[counter],mat->data[i],mat->col,multiple);
        }
        counter++;
    }
        return negative;
}
void EchelonFormToReduceEchelonForm(matrix*mat){
    int i,j,k,pivot[mat->row+1];
        database multiple,scales;
        bool status=false;
        for(i=1;i<=mat->row;i++)
                pivot[i]=0;
        pivot[1]=1;
        for(i=mat->row;i>=2;i--){
                status=false;
                for(j=1;j<=mat->col;j++){
                        if(mat->data[i][j].numer!=0){
                                status=true;
                                break;
                        }
                }
                if(status==false){
                        continue;
                }
                else{
```

```
                        pivot[i]=j;
                        for(k=i-1;k>=1;k--){
                                if(mat->data[k][j].numer==0)
                    continue;
                multiple=Multiply((database){-1,1},Divide(mat->data[k][j],mat->data[i][j]));
                if(multiple.deno<0){
                        CorrectData(&multiple);
                }
                                Replacement(mat->data[i],mat->data[k],mat->col,multiple);
                        }
                }
        }
        for(i=1;i<=mat->row;i++){
                if(pivot[i]!=0){
                        scales=Divide((database){1,1},mat->data[i][pivot[i]]);
                        Scaling(mat->data[i],scales,mat->col);
                }
        }
}
void RowReduceToRowReduceEchelonForm(matrix *mat){
        RowReduceToEchelonForm(mat);
    EchelonFormToReduceEchelonForm(mat);

}
/*------------------------------------------------------------------------------------*/
/*Three elementary row operation*/
void Replacement(const database row1[],database row2[],const int col,const database multip
    //row2 is the row which will be replaced
    int i;
    database temp;
        for(i=1;i<=col;i++){
        temp=Multiply(row1[i],multiple);
                row2[i]=Add(temp,row2[i]);
    }
}
void Interchange(database *row1,database *row2,const int col){
    int i;
    database temp;
    for(i=1;i<=col;i++){
        temp=row1[i];
        row1[i]=row2[i];
        row2[i]=temp;
    }
}
void Scaling(database *row,const database scales,const int col){
    int i;
    for(i=1;i<=col;i++){
        row[i]=Multiply(row[i],scales);
    }
}
/*-------------------------------------------------------------------*/
/*Matrix initialization release read print*/
matrix* ReadMatrix(){
    bool colStatus=true;
        int i,j,len,row=1,col=1,curr=-1,index=0;
```

```
    char str[MAXStr];
        gets(str);
        len=strlen(str);
        while(str[++curr]!=']'){
                if(str[curr]==';'){
                        colStatus=false;
                        row++;
                }
                else if(str[curr]==','&&colStatus)
                        col++;
        }
        matrix *mat=InitializeMatrix(row,col);
        for(i=1;i<=row;i++){
        for(j=1;j<=col;j++)
            mat->data[i][j]=read(str,&index);
    }
    return mat;
}
void PrintMatrix(const matrix mat){
    int i,j;
    printf("[");
        for(i=1;i<=mat.row;i++){
                printf("\n\n%4c",' ');
                for(j=1;j<=mat.col;j++){
            if(mat.data[i][j].deno==1||mat.data[i][j].numer==0)
                printf("%4d%4c",mat.data[i][j].numer,' ');
            else
                printf("%3d/%-3d ",mat.data[i][j].numer,mat.data[i][j].deno);
        }
    }
    puts("\n]");
}
matrix *InitializeMatrix(int Row,int Col){
    database data={0,1};
        matrix *mat=(matrix*)malloc(sizeof(matrix));
    int i,j;
    mat->row=Row,mat->col=Col;
    mat->data=(database**)malloc((Row+1)*sizeof(database*));
    mat->data[0]=(database*)malloc(sizeof(database)*(Col+1));
    for(i=1;i<=Row;i++){
        mat->data[i]=(database*)malloc(sizeof(database)*(Col+1));
        for(j=1;j<=Col;j++)
            mat->data[i][j]=data;
    }
    return mat;
}
void ReleaseMatrix(matrix *mat){
    int i,j;
    for(i=0;i<=mat->row;i++){
        free(mat->data[i]);
    }
    free(mat->data);
    free(mat);
}
/*--------------------------------------------------------------------------*/
```

function.c

```
/**
 *    author:  ouyanghaolan
 *    created: 07.01.2019
**/
#include"header.h"

database Determinant(const matrix mat1){
    if(mat1.row!=mat1.col){
        printf("Wrong!It is not a square Matrix");
        return (database){1,0};//if there don't exit the determinant return 1/0 to indicat
    }
    else{
                int i,j;
        database mul={1,1};
        matrix *mat2=InitializeMatrix(mat1.row,mat1.col);
                for(i=1;i<=mat1.row;i++)
                        for(j=1;j<=mat1.col;j++)
                                mat2->data[i][j]=mat1.data[i][j];
                bool negative=RowReduceToEchelonForm(mat2);
        for(i=1;i<=mat1.row;i++)
            mul=Multiply(mul,mat2->data[i][i]);
        if(negative)
                        mul.numer*=(-1);
        ReleaseMatrix(mat2);
                return mul;
    }
}

void SolutionSet(matrix mat){//Input a row reduced matrix and print out the solution set
        int i,j=1;
        bool col[mat.col+1],frevar=false;
        database free[mat.row+1][mat.col+1];
        for(i=1;i<=mat.col;i++){
                col[i]=false;
        }
        for(i=1;i<=mat.row;i++){
                for(j;j<=mat.col;j++){
                        if(mat.data[i][j].numer!=0){
                                col[j]=true;
                                j++;
                                break;
                        }
                }
        }
        for(i=1;i<=mat.col-1;i++){
                if(col[i]==false){
                        for(j=1;j<=mat.row;j++)
                                free[j][i]=(i==j?(database){1,1}:Multiply((database){-1,1}
                }
```

```
        }
        printf("The solution set is :");
        for(j=1;j<=mat.col-1;j++){
                if(col[j]==false){
                        frevar=true;
                        printf(" X%d * [",j);
                        for(i=1;i<=mat.row;i++){
                                print(free[i][j]);
                                printf("%c",i==mat.row?']':';');
                        }
                        printf(" + ");
                }
        }
        printf("[");
        for(i=1;i<=mat.row;i++){
                print(mat.data[i][mat.col]);
                printf("%c",i==mat.row?']':';');
        }
        if(frevar){
                printf(" ( ");
                for(j=1;j<=mat.col-1;j++){
                        if(col[j]==false)
                                printf("X%d ",j);
                }
                printf("is free variable )");
        }
        puts("");

}//still needs to improve the format.

void FindSolution(matrix mata,matrix matb){
        bool status=true;
        int i,j;
        matrix *mat=InitializeMatrix(mata.row,mata.col+1);
        for(i=1;i<=mata.row;i++){
                for(j=1;j<=mata.col;j++)
                        mat->data[i][j]=mata.data[i][j];
                mat->data[i][j]=matb.data[i][1];
        }
        RowReduceToRowReduceEchelonForm(mat);
        for(i=mata.row;i>=1;i--){
                if(mat->data[i][mat->col].numer!=0){
                        status=false;
                        for(j=1;j<=mat->col-1;j++)
                                if(mat->data[i][j].numer!=0){
                                        status=true;
                                        break;
                                }
                        break;
                }
        }
        if(status==false)
                puts("No solution!");
        else{
                SolutionSet(*mat);
```

```
        }
        return ;
}


matrix *FindInverse(matrix mat){
        if(mat.col!=mat.row)
                return NULL;
        database deter=Determinant(mat);
        if(deter.numer==0){
                return NULL;
        }
        matrix *inver=InitializeMatrix(mat.row,mat.col*2);
        int i,j;
        for(i=1;i<=mat.row;i++){
                for(j=1;j<=mat.col;j++){
                        inver->data[i][j]=mat.data[i][j];
                }
                for(j;j<=2*mat.col;j++){
                        if(i==j-mat.col)
                                inver->data[i][j].numer=1;
                        else
                                inver->data[i][j].numer=0;
                }
        }
        RowReduceToRowReduceEchelonForm(inver);
        matrix *inverse=InitializeMatrix(mat.row,mat.col);
        for(i=1;i<=mat.row;i++){
                for(j=1;j<=mat.col;j++){
                        inverse->data[i][j]=inver->data[i][mat.col+j];
                }
        }
        ReleaseMatrix(inver);
        return inverse;
}//need to be Release

matrix *LUFactorization(matrix *mat){
        matrix *L=InitializeMatrix(mat->row,mat->row);
        database data={1,1},multiple;
        int i,j,k,pivot=0,q;
        for(i=1;i<=mat->row;i++){
                L->data[i][i].numer=1;
                for(k=pivot+1;k<=mat->col;k++)
                        if(mat->data[i][k].numer!=0){
                                pivot=k;
                                data=mat->data[i][k];
                                break;
                        }
                for(q=i+1;q<=mat->row&&k<=mat->col;q++){
                        L->data[q][i]=Divide(mat->data[q][k],data);
                        multiple=Multiply((database){-1,1},Divide(mat->data[q][k],data));
                        Replacement(mat->data[i],mat->data[q],mat->col,multiple);
                }
        }
        return L;
```

```
}//return L back where mat become U

matrix *MatrixMultiplicationScalarProduct(matrix *mat1,matrix *mat2){
        int i,j;
        for(i=1;i<=mat1->row;i++){
                for(j=1;j<=mat1->col;j++){
                        mat1->data[i][j]=Multiply(mat1->data[i][j],mat2->data[i][j]);
                }
        }
        ReleaseMatrix(mat2);
        return mat1;
}

matrix *MatrixMultiplication(matrix *mat1,matrix *mat2){
        int i,j,k,q;
        database sum={0,1};
        matrix *NewMat=InitializeMatrix(mat1->row,mat2->col);
        for(i=1;i<=mat1->row;i++){
                for(j=1;j<=mat2->col;j++){
                        sum=(database){0,1};
                        for(k=1;k<=mat1->col;k++){
                                sum=Add(sum,Multiply(mat1->data[i][k],mat2->data[k][j]));
                        }
                        NewMat->data[i][j]=sum;
                }
        }
        ReleaseMatrix(mat2);
        ReleaseMatrix(mat1);
        return NewMat;
}

void ScalarOperation(char q,matrix *mat,database scalar,bool operand){
        int i,j;
        char operation[4]={'+','-','*','/'};
        database (*op)(database,database);
        database (*opera[4])(database,database)= { Add , Subtract , Multiply , Divide };
        for(i=0;i<=3;i++){
                if(q==operation[i]){
                        op=opera[i];
                        break;
                }
        }
        for(i=1;i<=mat->row;i++){
                for(j=1;j<=mat->col;j++){
                        if(operand)
                                mat->data[i][j]=op(mat->data[i][j],scalar);
                        else
                                mat->data[i][j]=op(scalar,mat->data[i][j]);
                }
        }
}

database InnerProduct(const database vector1[],const database vector2[],const int num){
        database sum={0,1};
        int i;
```

```
        for(i=1;i<=num;i++){
                sum=Add(sum,Multiply(vector1[i],vector2[i]));
        }
        return sum;
}

matrix *QRFactorization(matrix *mat){
        int i,j,k;
        database row1[mat->row+1],row2[mat->row+1];
        database scalar;
        matrix *R=InitializeMatrix(mat->col,mat->col);
        for(i=1;i<=mat->col;i++)
                for(j=1;j<=mat->col;j++)
                        R->data[i][j].numer=(i==j?1:0);
        for(i=2;i<=mat->col;i++){
                for(j=1;j<=i-1;j++){
                        for(k=1;k<=mat->row;k++){
                                row1[k]=mat->data[k][j];
                                row2[k]=mat->data[k][i];
                        }
                        scalar=Divide(InnerProduct(row1,row2,mat->row),InnerProduct(row1,r
                        R->data[j][i]=scalar;
                        for(k=1;k<=mat->row;k++){
                                mat->data[k][i]=Subtract(mat->data[k][i],Multiply(scalar,m
                        }
                }
        }
        return R;
}
```

UI.c

```
/**
 *    author:  ouyanghaolan
 *    created: 07.01.2019
**/
#include"header.h"
void ui(){
    char ch;
    puts("Welcome to my Linear Algebra Package!");
    puts("-------------------------------------------------------"
    "--------------------------------------");
    puts("-------------------------------------------------------"
    "--------------------------------------");
        puts("There are several functions that you can use to help"
    " your calculation of linear algebra\n");
        printf("\t1.Find the Soltion of the Matrix\n\t2.Find The Inverse of"
    "the Matrix\n\t3.LU Factorization of the Matrix\n"
        "\t4.Calculate the Determinant of the Matrix\n\t5.Matrix Multiplication"
    "\n\t6.Scalr operation on Matrix\n\t7.QR Factorization of the Matrix\n\n");
    printf("Please select the functions "
    "that you want to use(q/Q to quit the program):");
```

```
    while(scanf("%c",&ch)!=EOF){
        getchar();
                if(ch>='1'&&ch<='7')
            selection(ch-'0');
        else if(ch=='q'||ch=='Q'){
            break;
        }
        else
            printf("Error!Please enter a valid character!\n");
                puts("\n---------------------------------------------------------------"
        "----------------------------");
                printf("Functions:\n\t"
        "1.Find the Soltion of the Matrix\n\t2.Find The Inverse of the"
        " Matrix\n\t3.LU Factorization of the Matrix\n"
                "\t4.Calculate the Determinant of the Matrix\n\t5.Matrix Multiplication"
        "\n\t6.Scalr operation on Matrix\n\t7.QR Factorization of the Matrix\n\n");
        printf("Please select the functions that "
        "you want to use(q/Q to quit the program):");
    }
    puts("Thank you for using the application!\nHave a nice day!");
}

void selection(char op){
    switch(op){
        case 1:FindSolutionDisplay();break;
        case 2:FindInverseDisplay();break;
        case 3:LUFactorizationDisplay();break;
        case 4:DeterminantDisplay();break;
        case 5:MatrixMultiplicationDisplay();break;
        case 6:ScalarOperationDisplay();break;
        case 7:QRFactorizationDisplay();break;
    }
}
/*Function Display*/
void FindSolutionDisplay(){
        puts("Find the solution of Ax=b");
        puts("\nEnter the matrix A:");
        matrix *matA=ReadMatrix();
        puts("\nEnter vector b:");
        matrix *matb=ReadMatrix();
        puts("");
        FindSolution(*matA,*matb);
    ReleaseMatrix(matA);
    ReleaseMatrix(matb);
}

void FindInverseDisplay(){
        printf("\nEnter the matrix that you want to find the inverse:\n");
    matrix *mat=ReadMatrix();
    puts("");
        matrix *inver=FindInverse(*mat);
    if(inver!=NULL){
                puts("");
                PrintMatrix(*inver);
        puts("");
```

```
            ReleaseMatrix(inver);
            ReleaseMatrix(mat);
            }
            else
                    puts("Inverse Not Found!");
    }

    void LUFactorizationDisplay(){
        puts("\nEnter the matrix that you want to do the LU factorization:");
        matrix *mat=ReadMatrix();
        matrix *L=LUFactorization(mat);
        puts("");
        PrintMatrix(*L);
        puts("");
        PrintMatrix(*mat);
        ReleaseMatrix(L);
        ReleaseMatrix(mat);
    }

    void DeterminantDisplay(){
        puts("\nEnter the matrix that you want to calculate the determinant:");
        matrix *mat=ReadMatrix();
        database data=Determinant(*mat);
        if(data.deno==0)
            puts("\nError!The matrix is not a square matrix!");
        else{
            printf("\nThe determinant of the matrix is ");
            print(data);
            puts("");
        }
        ReleaseMatrix(mat);
    }

    void MatrixMultiplicationDisplay(){
        char ch;
        puts("\nSelect the multiplication you want to do(only a character):");
        printf("a)Scalar Product\nb)Matrix Multiplication:");
        ch=getchar();
        puts("\nEnter the first matrix:");
        matrix *mat1=ReadMatrix();
        puts("\nEnter the second matrix:");
        matrix *mat2=ReadMatrix();
        matrix *result;
        switch(ch){
            case 'a':result=MatrixMultiplicationScalarProduct(mat1,mat2);break;
            case 'b':result=MatrixMultiplication(mat1,mat2);break;
        }
        puts("");
        PrintMatrix(*result);
        puts("");
        ReleaseMatrix(result);
    }
    void ScalarOperationDisplay(){
            printf("\nEnter your calculation formular:\n");
            int index=0,curr,row=1,col=1,i,j;
```

```
        bool colStatus=true,operand=true;//true indicate left false indicate right
        database scalar;
        matrix *mat;
        char str[MAXStr],op;
        gets(str);
        for(;str[index]==' ';index++);
        if(str[index]!='['){
                operand=false;
                scalar=read(str,&index);
                for(;str[index]==' ';index++);
                op=str[index];
                curr=index++;
                while(str[++curr]!=']'){
                        if(str[curr]==';'){
                                colStatus=false;
                                row++;
                        }
                        else if(str[curr]==','&&colStatus)
                                col++;
                }
                mat=InitializeMatrix(row,col);
                for(i=1;i<=row;i++){
                for(j=1;j<=col;j++)
                mat->data[i][j]=read(str,&index);
        }
        }
        else{
                curr=index;
                while(str[++curr]!=']'){
                        if(str[curr]==';'){
                                colStatus=false;
                                row++;
                        }
                        else if(str[curr]==','&&colStatus)
                                col++;
                }
                mat=InitializeMatrix(row,col);
                for(i=1;i<=row;i++){
                for(j=1;j<=col;j++)
                mat->data[i][j]=read(str,&index);
        }
        for(;str[index]==' '||str[index]==']';index++);
        op=str[index];
        scalar=read(str,&index);
        }
        ScalarOperation(op,mat,scalar,operand);
        PrintMatrix(*mat);
    ReleaseMatrix(mat);
}

void QRFactorizationDisplay(){
    printf("\nEnter the matrix that you want to do the QR factorization:\n");
    matrix *mat=ReadMatrix();
    matrix *R=QRFactorization(mat);
    puts("");
```

```
    PrintMatrix(*mat);
    puts("");
    PrintMatrix(*R);
    ReleaseMatrix(R);
    ReleaseMatrix(mat);
}

/*--------------------------------------------------------------------------*/
```