Project 3

刘一郴 12112609

1. Code Component

1.1. head.h

Some include and define to easy following programing.

```
// some parameters and includes
#ifndef _HEAD_H
#define _HEAD_H

#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>
typedef long long ll;
#define ri register int
#define For(i, a, b) for(ri i= a;i<= b;i++)
#define Ford(i, a, b) for(ri i= a;i>= b;i--)

#endif
```

1.2. Matrix.h

1.2.1. struction Matrix

Specially it has a pointer to next Matrix which can be used to form the **Linked List** to record current existed Matrixs.

```
typedef struct Matrix{
    int row;            //row number
    int col;            //column number
    float * entry;      //store all entries in wich entry[(i - 1)* col + (j - 1)] is entry in (i, j)
    struct Matrix * pNext;//pointer of next Matrix in Matrix linked list containing all Matrix
} Matrix;
```

1.2.2. pFirstMatrix

It is used to show the head of **Linked List** proposed above, **initially NULL**.

```
//the head of Matrix linked list
Matrix * pFirstMatrix;
```

1.2.3. functions

Claims of Matrix Functions including all requested functions as well as some **extra functions** like **transposeMatrix(), standardMatrix(), gussianEliminationMatrix(), rankOfMatrix(), attachMatrix(), inverseMatrix()**.

```
12  //create new matrix and return the pointer
13  Matrix * createMatrix(const int row, const int col, const float * const entry);
14  //check if Matrix exist return true IFF Matrix exists
15  bool existMatrix(const Matrix * const pMatrix);
16  //delete Matrix pointed by pointer return true IFF success
17  bool deleteMatrix(Matrix * const pMatrix);
18  //copy the data from source Matrix to destination Matrix return true IFF success
19  bool copyMatrix(Matrix * const dest, const Matrix * const src);
20  //add tow Matrix return new Matrix pointer ; return NULL IFF sizes of two matrix do not match or Matrix do not exist
21  Matrix * addMatrix(const Matrix * const pAugend, const Matrix * const pAddend);
22  //subtract tow Matrix and return new Matrix pointer ; return NULL IFF sizes of two matrix do not match or Matrix do not exist
23  Matrix * subtractMatrix(const Matrix * const  pMinuend, const Matrix * const pSubtrahend);
24  //add scalar to a Matrix return new Matrix; return NULL IFF Matrix do not exist
25  Matrix * addScalarToMatrix(const Matrix * const pMatrix, const float scalar);
26  //subtract scalar from a Matrix return new Matrix; return NULL IFF Matrix do not exist
27  Matrix * subtractScalarFromMatrix(const Matrix * const pMatrix, const float scalar);
28  //multiply Matrix with scalar and return new Matrix; return NULL IFF Matrix do not exist
29  Matrix * multiplyMatrixWithScalar(const Matrix * const pMatrix, const float scalar);
30  //multiply two Matrix and return new Matrix; return NULL IFF sizes of two matrix do not match or Matrix do not exist
31  Matrix * multiplyMatrix(const Matrix * const pA, const Matrix * const pB);
32  //Find the position of max value of Matrix as (rowIndex - 1) * MatrixColumnNumber + (columnIndex - 1) form; return 0 IFF Matrix do not exist
33  int MaxValuePositionOfMatrix(const Matrix * const pMatrix);
34  //Find the position of min value of Matrix as (rowIndex - 1) * MatrixColumnNumber + (columnIndex - 1) form; return 0 IFF Matrix do not exist
35  int MinValuePositionOfMatrix(const Matrix * const pMatrix);
36  //print Matrix in bash IFF Matrix exist
37  void printMatrix(const Matrix * const pMatrix);
38  //return the transpose of Matrix ; return NULL IFF Matrix do NOT exist
39  Matrix * transposeMatrix(const Matrix * const pMatrix);
40  //return standard Matrix
41  Matrix * standardMatrix(const int size, const float val);
42  //return Gussian Elimination of Matrix ; return NULL IFF Matrix do NOT exist or size do NOT match
43  Matrix * gussianEliminationMatrix(const Matrix * const pMatrix);
```

```
44  //get the rank of Matrix
45  int rankOfMatrix(const Matrix * const pMatrix);
46  //return attached Matrix from two Matrix like A B then A|B ; return -1 IFF Matrix do NOT exist or size do NOT match
47  Matrix * attachMatrix(const Matrix * const pA, const Matrix * const pB);
48  //return the inverse of Matrix; return NULL IFF Matrix do NOT exist or Matrix is NOT a square Matrix
49  Matrix * inverseMatrix(const Matrix * const pMatrix);
```

1.3. Matrix.c

Because the code length is to long, we pick some import function as examples. Moreover, some function is **quite easy and needless to explain**.

1.3.1. gussianEliminationMatrix

return Gussian Elimination of Matrix ; return NULL IFF Matrix do NOT exist or size do NOT match.

Reduce the matrix into triangular form.

```
237  //return Gussian Elimination of Matrix ; return NULL IFF Matrix do NOT exist or size do NOT match
238  Matrix * gussianEliminationMatrix(const Matrix * const pMatrix){
239      if(!existMatrix(pMatrix) ){
240          return NULL;
241      }
242      Matrix * ans = createMatrix(1, 1, NULL);
243      copyMatrix(ans, pMatrix);
244
245      int lines = 0;
246      For(i, 1, ans->col){
247          lines++;
248          if(lines >= ans->row){
249              break;
250          }
251          int tmp = 0;
252          For(j, lines, ans->row){
253              if(abs(ans->entry[(j - 1) * ans->col + (i - 1)]) > P){
254                  tmp = j;
255                  break;
256              }
257          }
258          if(tmp == 0){
259              lines--;
260              continue;
261          }
262          For(j, i, ans->col){
263              float tmpVal = ans->entry[(lines - 1) * ans->col + (j - 1)];
264              ans->entry[(lines - 1) * ans->col + (j - 1)] = ans->entry[(tmp - 1) * ans->col + (j - 1)];
265              ans->entry[(tmp - 1) * ans->col + (j - 1)] = tmpVal;
266          }
267          For(j, lines + 1, ans->row){
268              if(abs(ans->entry[(j - 1)* ans->col + (i - 1)]) > P){
269                  float b = ans->entry[(j - 1)* ans->col + (i - 1)] / ans->entry[(lines - 1)* ans->col + (i - 1)];
270                  For(k, i, ans->col){
271                      ans->entry[(j - 1)* ans->col + (k - 1)] -= b * ans->entry[(lines - 1)* ans->col + (k - 1)];
272                  }
273              }
274          }
275      }
276      return ans;
277  }
```

1.3.2. inverseMatrix

return the inverse of Matrix; return NULL IFF Matrix do NOT exist or Matrix is NOT a square Matrix.

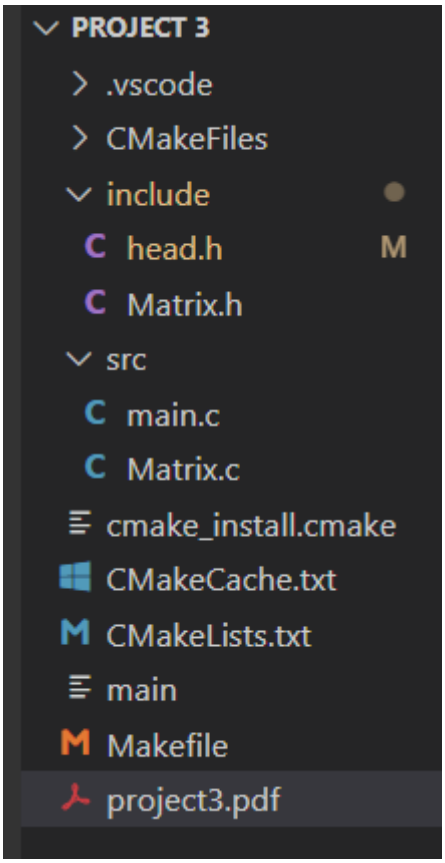
Using the theorem [A | I] = A * [I | A^-1]

```
317  //return the inverse of Matrix; return NULL IFF Matrix do NOT exist or Matrix is NOT a square Matrix
318  Matrix * inverseMatrix(const Matrix * const pMatrix){
319      if(!existMatrix(pMatrix) ){
320          return NULL;
321      }
322      if(pMatrix->row != pMatrix->col){
323          return NULL;
324      }
325      if(rankOfMatrix(pMatrix) != pMatrix->row ){
326          return NULL;
327      }
328      Matrix * I = standardMatrix(pMatrix->row, 1);
329      Matrix * AB = attachMatrix(pMatrix, I);
330      deleteMatrix(I);
331      Matrix * TMP = gussianEliminationMatrix(AB);
332      deleteMatrix(AB);
333      AB = TMP;
334      Ford(i, AB->row, 1){
335          Ford(j, AB->col, i){
336              AB->entry[(i - 1)* AB->col + (j - 1)]/= AB->entry[(i - 1)* AB->col + (i - 1)];
337          }
```

```
        For(j, 1, i - 1){
            if(abs(AB->entry[(j - 1) * AB->col + (i - 1)]) > P ){
                float b = AB->entry[(j - 1)* AB->col + (i - 1)];
                For(k, i, AB->col){
                    AB->entry[(j - 1)* AB->col + (k - 1)] -= b * AB->entry[(i - 1)* AB->col + (k - 1)];
                }
            }
        }
    }
    Matrix * ans = createMatrix(pMatrix->row, pMatrix->col, NULL);
    For(i, 1, ans->row){
        For(j, 1, ans->col){
            ans->entry[(i - 1) * ans->col + (j - 1)] = AB->entry[(i - 1) * AB->col + (ans->col + j - 1)];
        }
    }
    deleteMatrix(AB);
    return ans;
}
```

2. requirements

2.1. The programming language can only be C, not C++. Please save your source code into *.c files, and compile them using a C compiler such as gcc (not g++). Try to use Makefile or CMake to manage your source code.



2.2. Design a struct for matrices, and the struct should contain the

data of a matrix, the number of columns, the number of rows, etc.

```
typedef struct Matrix{
    int row;              //row number
    int col;              //column number
    float * entry;        //store all entries in wich entry[(i - 1)* col + (j - 1)] is entry in (i, j)
    struct Matrix * pNext;//pointer of next Matrix in Matrix linked list containing all Matrix
} Matrix;
```

2.3. Only float elements in a matrix are supported. You do not need to implement some other types.

As above figure

2.4. Implement some functions

All the required functions are implemented as well as some **extra functions** like **transposeMatrix(), standardMatrix(), gussianEliminationMatrix(), rankOfMatrix(), attachMatrix(), inverseMatrix()**.

2.5. The designed functions should be safe and easy to use. Suppose you are designing a library for others to use. You do not need to focus on the optimization in this project, ease of use is more important.

In this project all the existing Matrixs are stored in one **Linked List**, so we can check if the Matrixs given by user is valid by following function **existMatrix()**.

```
26  //check if Matrix exist return true IFF Matrix exists
27  bool existMatrix(const Matrix * const pMatrix){
28      if(pMatrix == NULL){
29          return false;
30      }
31      Matrix * pTmp = pFirstMatrix;
32      while(pTmp!= NULL){
33          if(pTmp == pMatrix){
34              return true;
35          }
36          pTmp = pTmp->pNext;
37      }
38      return false;
39  }
```

3. highlights

3.1. well written description for functions

All the **parameter type** are well claimed.

All the **description for functions** are well written.

```
12  //create new matrix and return the pointer
13  Matrix * createMatrix(const int row, const int col, const float * const entry);
14  //check if Matrix exist return true IFF Matrix exists
15  bool existMatrix(const Matrix * const pMatrix);
16  //delete Matrix pointed by pointer return true IFF success
17  bool deleteMatrix(Matrix * const pMatrix);
18  //copy the data from source Matrix to destination Matrix return true IFF success
19  bool copyMatrix(Matrix * const dest, const Matrix * const src);
20  //add tow Matrix return new Matrix pointer ; return NULL IFF sizes of two matrix do not match or Matrix do not exist
21  Matrix * addMatrix(const Matrix * const pAugend, const Matrix * const pAddend);
22  //subtract tow Matrix and return new Matrix pointer ; return NULL IFF sizes of two matrix do not match or Matrix do not exist
23  Matrix * subtractMatrix(const Matrix * const  pMinuend, const Matrix * const pSubtrahend);
24  //add scalar to a Matrix return new Matrix; return NULL IFF Matrix do not exist
25  Matrix * addScalarToMatrix(const Matrix * const pMatrix, const float scalar);
26  //subtract scalar from a Matrix return new Matrix; return NULL IFF Matrix do not exist
27  Matrix * subtractScalarFromMatrix(const Matrix * const pMatrix, const float scalar);
28  //multiply Matrix with scalar and return new Matrix; return NULL IFF Matrix do not exist
29  Matrix * multiplyMatrixWithScalar(const Matrix * const pMatrix, const float scalar);
30  //multiply two Matrix and return new Matrix; return NULL IFF sizes of two matrix do not match or Matrix do not exist
31  Matrix * multiplyMatrix(const Matrix * const pA, const Matrix * const pB);
32  //Find the position of max value of Matrix as (rowIndex - 1) * MatrixColumnNumber + (columnIndex - 1) form; return 0 IFF Matrix do not exist
33  int MaxValuePositionOfMatrix(const Matrix * const pMatrix);
34  //Find the position of min value of Matrix as (rowIndex - 1) * MatrixColumnNumber + (columnIndex - 1) form; return 0 IFF Matrix do not exist
35  int MinValuePositionOfMatrix(const Matrix * const pMatrix);
36  //print Matrix in bash IFF Matrix exist
37  void printMatrix(const Matrix * const pMatrix);
38  //return the transpose of Matrix ; return NULL IFF Matrix do NOT exist
39  Matrix * transposeMatrix(const Matrix * const pMatrix);
40  //return standard Matrix
41  Matrix * standardMatrix(const int size, const float val);
42  //return Gussian Elimination of Matrix ; return NULL IFF Matrix do NOT exist or size do NOT match
43  Matrix * gussianEliminationMatrix(const Matrix * const pMatrix);
```

```
44  //get the rank of Matrix
45  int rankOfMatrix(const Matrix * const pMatrix);
46  //return attached Matrix from two Matrix like A B then A|B ; return -1 IFF Matrix do NOT exist or size do NOT match
47  Matrix * attachMatrix(const Matrix * const pA, const Matrix * const pB);
48  //return the inverse of Matrix; return NULL IFF Matrix do NOT exist or Matrix is NOT a square Matrix
49  Matrix * inverseMatrix(const Matrix * const pMatrix);
```

3.2. pointer check

For every given pointer of Matrix, function will check if it

exists which make functions **rubust**.

However, it indeed consume more time each time the function are invoked which most of time are **even duplicated**. So **it would be better for user to avoid the unexisted pointer by themselves**.

3.3. more useful functions

transposeMatrix(), standardMatrix(), gussianEliminationMatrix(), rankOfMatrix(), attachMatrix(), inverseMatrix() are useful and necessary tools for practical using to solve Matrix problem. So they are added specially.