CS205 C/C++ Programming - Lab Assignment 2

Name: 敖子添 (Ao Zitian)
SID : 11911120

Part 1 - Analysis

To make a structure which could represent a matrix and containing all the information about a matrix, we could use two integer to record the number of row and number of column about our matrix. The details of our matrix could be stored in an one-dimensional array (pointer).

```
struct Matrix
{
    int row;
    int col;
    float *pMat;
};
```

Many methods about matrix

- create a matrix: we could use a pointer which point to a matrix to represent a matrix.

```
struct Matrix *createMatrix(int row, int col)
{
    struct Matrix *p = (struct Matrix *)malloc(sizeof(struct Matrix));
    p->row = row;
    p->col = col;
    p->pMat = (float *)malloc(sizeof(float) * row * col);
    return p;
}
```

- Besides, we also need a funciton which could give our new matrix value. Its name is `valueInMatrix` .

```
void valueInMatrix(struct Matrix *p)
{
    if (p != NULL)
    {
        float *mat = p->pMat;
        int row = p->row, col = p->col;
        float arr[row * col];
        for (int i = 0; i < row; i++)
```

```
            for (int j = 0; j < col; j++)
            {
                scanf("%f", &arr[i * col + j]);
            }
        for (int i = 0; i < row * col; i++)
            *(mat + i) = arr[i];
    }
    else
        printf("Matrix is NULL, please check!");
}
```

- delete a matrix: because we store a matrix by storing its pointer, we need only free all the pointer and let them point to `NULL` when we want to delete a matrix. Note: because a matrix contains three variables: two `int` and one `float*` , we need to free the `float*` first and free the pointer which point to out matrix next.

```
void deleteMatrix(struct Matrix *mat)
{
    if (mat->pMat != NULL)
    {
        free(mat->pMat);
        mat->pMat = NULL;
    }
    if (mat != NULL)
    {
        free(mat);
        mat = NULL;
    }
}
```

- copy a matrix: copy all the content from a matrix. It is easy to understand so we won't use much words to explain how to do it.

```
struct Matrix *copyMatrix(const struct Matrix *mat)
{
    if (mat != NULL)
    {
        struct Matrix *mat_copy = createMatrix(mat->row, mat->col);
        int count = mat->row * mat->col;
        float *p = mat->pMat;
        float *p_copy = mat_copy->pMat;
        for (int i = 0; i < count; i++)
        {
            *(p_copy + i) = *(p + i);
        }
        return mat_copy;
    }
    else
    {
        printf("Matrix is NULL, please check!");
        return NULL;
    }
}
```

We will only show the statement of each function instead of the defination of them because all of our code will be show in Part2.

- add two matrices: we will create a new matrix first and each element of it is the addtion of the input matrices.

```
struct Matrix *addMatrix(const struct Matrix *mat1, const struct Matrix *mat2);
```

- subtraction of two matrices: it is as same as `addMatrix` only to change `+` to `-`.

```
struct Matrix *subtractMatrix(const struct Matrix *mat1, const struct Matrix *mat2);
```

- add a scalar to a matrix: we should let all elements of matrix add a constant.

```
struct Matrix *scalarAddition(const struct Matrix *mat1, const struct Matrix *mat2);
```

- subtract a scalar from a matrix: we should let all elements of matrix minus a constant.

```
struct Matrix *scalarSubtraciton(const struct Matrix *mat1, const struct Matrix *mat2);
```

- multiply a matrix with a scalar: we should let all elements of matrix multiiple a constant.

```
struct Matrix *scalarMultiplication(const struct Matrix *mat, const float a);
```

- multiply two matrices: Using the matrix multiplicaiton formula: $c_{ij} = \Sigma_k a_{ik} b_{kj}$, the detailed will be shown in our code.

```
struct Matrix *scalarMultiplication(const struct Matrix *mat, const float a)
{
    if (mat != NULL)
    {
        struct Matrix *matp = createMatrix(mat->row, mat->col);
        int count = mat->row * mat->col;
        for (int i = 0; i < count; i++)
        {
            *(matp->pMat + i) = *(mat->pMat + i) * a;
        }
        return matp;
    }
    else
    {
        printf("Matrix is NULL!");
        return NULL;
    }
}
```

- find the minimal and maximal values of a matrix: traverse the matrix input and find the minimun/maximun value of it.

```
float minMatrix(const struct Matrix *mat);
float maxMatrix(const struct Matrix *mat);
```

- We need a function which could show the matrix we have input, we name it `outPutMatrix` .

```
void outPutMatrix(struct Matrix *mat)
{
    if (mat != NULL)
        for (int i = 0; i < mat->row; i++)
        {
            for (int j = 0; j < mat->col; j++)
                printf("%f ", *(mat->pMat + i * mat->col + j));
            printf("\n");
        }
    else
        printf("The matrix is NULL!");
}
```

Part 2 - Code

The demo of this project will be shown in Github:
[CPP Projects]

Part 3 - Result & Verification

- Case 1: Copy matrix

```
3 3
1 2 3
2 3 5
1 223 4
1.000000 2.000000 3.000000
```

```
2.000000 3.000000 5.000000
1.000000 223.000000 4.000000
```

- Case 2: add two matrices

```
3 3
1 2 3
2 3 4
3 4 5
3 3
1 2 3
4 5 2
2 3 1
2.000000 4.000000 6.000000
6.000000 8.000000 6.000000
5.000000 7.000000 6.000000
```

- Case 3: subtractMatrix

```
3 3
1 2 3
3 4 5
3 5 2
3 3
2 3 4
2 1 5
2 5 32
-1.000000 -1.000000 -1.000000
1.000000 3.000000 0.000000
1.000000 0.000000 -30.000000
```

- Case 4: add a scalar to a matrix (all element add 3)

```
2 3
12 32 2
241 1 2
15.000000 35.000000 5.000000
244.000000 4.000000 5.000000
```

- Case 5: multiply a matrix with a scalar (all elements multiple 3)

```
2 4
1 2 3 1
3 4 3 21
3.000000 6.000000 9.000000 3.000000
9.000000 12.000000 9.000000 63.000000
```

- Case 6: multiply two matrices

```
3 4
1 2 3 2
2 3 4 1
4 -2 23 2
4 2
1 2
3 2
1 1
1 1
12.000000 11.000000
16.000000 15.000000
23.000000 29.000000
```