CS205 Proj#3 Report

Name: 刘啸涵 (Xiaohan Liu)

SID: 11911925

Github Link: here

Part 1 - Analysis

In this project we need to implement a simple matrix library, which provides a `Matrix` struct, and some basic operations like matrix multiplication and addition. The library should be easy and safe to use, so it should be capable of handling various input errors.

Part 2 - Code

Design of the `matrix` struct

Here, we impelemented a rather simple struct for matrix.

```
typedef struct
{
    int row;
    int column;
    float data[MAX_SIZE];
} Matrix;
```

The data of the matrix is stored in row-major style. For example, the element a_{ij} is stored in `data[i*column+j]`.

Error handling

The functions might encounter with a bunch of illegal inputs. However, c itself does not provide support for error handling.

We refered to the implementation of STM32 HAL Library. All operations, excepts `create_matrix()`, will return a `calc_status` value. The calc_status indicates what kind of problem the program has encountered with.

```
typedef enum
{
    MAT_SUCCESS = 0,
    MAT_ILLEGAL_SIZE_ERROR = 1,
    MAT_NULLPTR_ERROR = 2,
    MAT_SIZE_MISMATCH_ERROR = 3,
} calc_status;
```

We also provide a simple error handling function which prints the error infomation. This function is set as `weak` and can be overridden.

```
void __attribute__((weak)) error_handler(calc_status status)
{
    switch(status){
        case MAT_ILLEGAL_SIZE_ERROR:
            printf("creating a matrix with illegal size\n");
            break;
        case MAT_NULLPTR_ERROR:
            printf("operand is null\n");
            break;
        case MAT_SIZE_MISMATCH_ERROR:
            printf("Operand size mismatch\n");
            break;
    }
}
```

Example: Matrix Multiplication

```
calc_status multiply_matrix(Matrix* left_op, Matrix* right_op, Matrix* result)
{
    if(left_op == NULL || right_op == NULL || result == NULL){
        return MAT_NULLPTR_ERROR;
    }

    if(left_op -> column != right_op->row){
        return MAT_SIZE_MISMATCH_ERROR;
    }

    if(left_op -> row * right_op -> column > MAX_SIZE){
        return MAT_ILLEGAL_SIZE_ERROR;
    }

    int result_row = left_op->row;
    int result_col = right_op->column;

    result->row = result_row;
    result->column = result_col;

    for(int i = 0; i < result_row; i++){
        for(int j = 0; j < result_col; j++){
            result->data[i*result_col+j] = 0.0;
            for(int k=0;k<left_op->column;k++){
                result->data[i*result_col+j] +=
                    left_op->data[i*left_op->column+k] *
right_op->data[k*right_op->column+j];
            }
        }
    }

    return MAT_SUCCESS;
}
```

Construction

The implementation of matrix creation is very naive: here we simply allocate space for a matrix and set the attributes. Before allocation the function will go through a size check. If the size exceeds `MAX_SIZE` it will return a NULL pointer.

```
Matrix* create_empty_matrix(int row, int column)
{
    /* Check if the size is legal. */
    if( row < 0 || column < 0 || row * column > MAX_SIZE){
        error_handler(MAT_ILLEGAL_SIZE_ERROR);
        return NULL;
    }

    /* Create Matrix */
    Matrix* p_Mat = (Matrix*) malloc (sizeof(Matrix));

    if(p_Mat != NULL){ //if malloc is successful
        p_Mat->row = row;
        p_Mat->column = column;
        for(int i = 0; i < MAX_SIZE; i++){
            p_Mat->data[i] = 0.0;
        }
    }

    return p_Mat;
}
```

We believe that the users can take care of the possible returned value `NULL` by themselves. Returning `NULL` is quite common in memory allocation of C, so there is no need for babysitting here.

Destruction

In destruction, we will free the allocated memory and set the pointer to `NULL`. Since the pointer itself will be changed in this function, the parameter should be a second order pointer.

```
calc_status delete_matrix(Matrix** pp_mat)
{

    if(*pp_mat == NULL){
        return MAT_NULLPTR_ERROR;
    }
    free(*pp_mat);
    *pp_mat = NULL;
    return MAT_SUCCESS;
}
```

Part 3 - Result & Verification

Test case #1: Matrix Multiplication

```
float data[3] = {1.0,2.0,1.0};
Matrix* example_vector = create_matrix(1,3,data);

float identity[9] = {1.0,0.0,0.0,
                        0.0,1.0,0.0,
                        0.0,0.0,1.0};
Matrix* example_mat = create_matrix(3,3,identity);

Matrix* example_output = create_empty_matrix(1,1);
calc_status status =
multiply_matrix(example_vector,example_mat,example_output);
if(status == MAT_SUCCESS){
    print_matrix(example_output);
}else{
    error_handler(status);
}
```

Output:

```
1.000000 2.000000 1.000000
```

Test case #2: Error Handling

```
status =
add_matrix(example_vector,example_mat,example_output);
if(status == MAT_SUCCESS){
    print_matrix(example_output);
}else{
    error_handler(status);
}
```

Output:

```
Operand size mismatch
```

Test case #4: Deleting Matrix

```
status = delete_matrix(&example_output);
if(status == MAT_SUCCESS){
    if(example_output==NULL){
        printf("Mat is successfully freed.\n");
    }
}else{
    error_handler(status);
}
```

Here we checked if the matrix is successfully set to Null.

Output:

```
Mat is successfully freed.
```

Other test cases

The outher test cases can be found in `demo.c`.

Cpp Wrapper Test

Our library is compatible with c++. We developed a simple wrapper class of `Matrix` and tested it.

```
SimpleMat operator * (const SimpleMat& leftop, const
SimpleMat& rightop){
    Matrix* result = new Matrix;
    calc_status status =
multiply_matrix(leftop.pImpl,rightop.pImpl,result);
    if(status != MAT_SUCCESS){
        throw std::exception();
    }
    SimpleMat mat(result->row,result->column,result->data);
    delete result;
    return mat;
};
```

The program will throw an exception when encoutered with a calculation failure. (I haven 't implemented the error class for the wrapper, sorry about that.)

Part 4 - Difficulties & Solutions

Destruction

In destruction, we will free the allocated memory and set the pointer to `NULL`. Since the pointer itself will be changed in this function, the parameter should be a second order pointer.