Лабораторная работа № 5

Многофайловый проект, условная компиляция, утилита Make.

Задание 1.1

Постановка задачи:

Напишите программу из нескольких файлов (модулей), включая файл основной программы. Файлы должны содержать вынесенные отдельно функции для выделения памяти под динамические двумерные и одномерные массивы и функции для перемножения матриц. Собрать проект используя утилиту Make.

Математическая модель:

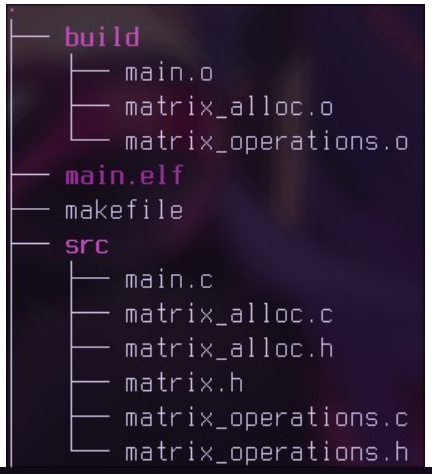
$$A*B=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}.\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}=\begin{bmatrix}30 & 36 & 42\\66 & 81 & 96\\102 & 126 & 150\end{bmatrix}$$

Список идентификаторов:

Имя	**Тип**	**Назначение**
	main.c	
A	Matrix	Первая матрица для умножения.
B	Matrix	Вторая матрица для умножения.
C	Matrix	Результат умножения AB.
D	Matrix	Результат сложения B+C.
i	int	Аргумент циклов.
j	int	Аргумент циклов.
	matrix_alloc.c	
i	int	Аргумент циклов.
rows	unsigned	Количество строк создаваемой матрицы
colums	unsigned	Количество столбцов создаваемой матрицы.
A	Matrix	Матрица под которую нужно выделить или освободить память.
len	unsigned	Длинна массива под который выделяется

		память.
array	double*	Динамический выделенный массив.
	matrix_operations.c	
i	int	Аргумент циклов.
j	int	Аргумент циклов.
k	int	Аргумент циклов.
A	Matrix	Первая матрица для умножения/сложения.
B	Matrix	Вторая матрица для умножения/сложения.
C	Matrix	Результат умножения/сложения.
num	double	Число на которое умножается матрица для умножения матрицы на число.

Код программы:




```
CC=gcc
CFLAGS=-g -Wall -I.
BUILD_DIR=build
SRC_DIR=src

all: matrix_operations.o matrix_alloc.o main.o
    $(CC) -o main.elf $(BUILD_DIR)/*.o $(CFLAGS)

main.o:
    $(CC) -o $(BUILD_DIR)/main.o $(SRC_DIR)/main.c $(CFLAGS) -c

matrix_operations.o:
    $(CC) -o $(BUILD_DIR)/matrix_operations.o $(SRC_DIR)/matrix_operations.c -c

matrix_alloc.o:
    $(CC) -o $(BUILD_DIR)/matrix_alloc.o $(SRC_DIR)/matrix_alloc.c -c

clean:
    rm $(BUILD_DIR)/*.o
```

main.c:

```c
#include <stdio.h>
#include "matrix.h"
#include "matrix_operations.h"
#include "matrix_alloc.h"


int
main()
{
    Matrix A = matrix_alloc(3, 3);
    Matrix B = matrix_alloc(3, 3);

    for(int i = 0, k = 1; i < 3; i++){
        for(int j = 0; j < 3; j++, k++){
            A.data[i][j] = k;
            B.data[i][j] = k;
        }
    }

    printf("Matrix 1\n");
    matrix_print(A);
    printf("Matrix 2\n");
    matrix_print(B);

    Matrix C = matrix_mult(A, B);
    printf("Multiplicatoin result:\n");
    matrix_print(C);

    Matrix D = matrix_add(B, C);
    printf("Addition result:\n");
    matrix_print(D);


    matrix_free(A);
    matrix_free(B);
    matrix_free(C);
    matrix_free(D);
    return 0;
}
```

matrix_operations.h:

```
 1 #ifndef MATRIX_OPERATAOINS
 2 #define MATRIX_OPERATAOINS
 3
 4 #include "matrix.h"
 5
 6 Matrix matrix_mult(Matrix A, Matrix B);
 7 Matrix matrix_add(Matrix A, Matrix B);
 8 void matrix_print(Matrix A);
 9 Matrix matrix_number_product(Matrix A, double num);
10
11 #endif
```

matrix.h

```
 1 #ifndef MATRIX_H
 2 #define MATRIX_H
 3
 4 typedef struct {
 5     int rows;
 6     int cols;
 7     double** data;
 8 } Matrix;
 9
10 #endif
```

matrix_operations.c:

```c
#include "matrix_operations.h"
#include "matrix.h"
#include "matrix_alloc.h"
#include "stdio.h"


Matrix matrix_mult(Matrix A, Matrix B)
{
    Matrix C = matrix_alloc(A.rows, B.cols);

    for(int i = 0; i < A.rows; i++){
        for(int j = 0; j < B.cols; j++){
            for(int k = 0; k < A.cols; k++){
                C.data[i][j] += A.data[i][k]*B.data[k][j];
            }
        }
    }
    return C;
}

Matrix matrix_add(Matrix A, Matrix B)
{
    Matrix C = matrix_alloc(A.rows, B.cols);

    for(int i = 0; i < A.rows; i++){
        for(int j = 0; j < B.cols; j++){
                C.data[i][j] += A.data[i][j]+B.data[i][j];
        }
    }
    return C;
}

void matrix_print(Matrix A)
{
    for(int i = 0; i < A.rows; i++){
        for(int j = 0; j < A.cols; j++){
            printf("%lf ", A.data[i][j]);
        }
        putchar('\n');
    }
}


Matrix matrix_number_product(Matrix A, double num)
{
    Matrix C = matrix_alloc(A.rows, A.cols);
    for(int i = 0; i < C.rows; i++){
        for(int j = 0; j < C.cols; j++)
            C.data[i][j] = C.data[i][j]*num;
    }
    return C;
}
```

matrix_alloc.h

```h
#ifndef MATRIX_ALLOC
#define MATRIX_ALLOC

#include "matrix.h"

Matrix matrix_alloc(unsigned rows, unsigned colums);
void matrix_free(Matrix);
double *array_alloc(unsigned len);

#endif
```

matrix_alloc.c

```c
#include "matrix_alloc.h"
#include <stdlib.h>
#include "matrix.h"

void matrix_free(Matrix A)
{
    for(int i = 0; i < A.rows; i++)
        free(A.data[i]);
    free(A.data);
}

Matrix matrix_alloc(unsigned rows, unsigned colums)
{
    Matrix A = {rows, colums, NULL};

    A.data = (double **)(malloc(sizeof(double*)*rows));
    for(int i = 0; i < rows; i++)
        A.data[i] = calloc(colums, sizeof(double));
    return A;
}

double *array_alloc(unsigned len){
    double *array = NULL;
    array = calloc(len, sizeof(double));
    return array;
}
```

Результаты выполненной работы:

```
>make
gcc -o build/matrix_operations.o src/matrix_operations.c -c
gcc -o build/matrix_alloc.o src/matrix_alloc.c -c
gcc -o build/main.o src/main.c -g -Wall -I. -c
gcc -o main.elf build/*.o -g -Wall -I.
>./main.elf
Matrix 1
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
Matrix 2
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
Multiplicatoin result:
30.000000 36.000000 42.000000
66.000000 81.000000 96.000000
102.000000 126.000000 150.000000
Addition result:
31.000000 38.000000 45.000000
70.000000 86.000000 102.000000
109.000000 134.000000 159.000000
```