Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»
Факультет інформаційних технологій
Кафедра інформаційних технологій та комп'ютерної інженерії

Звіт
з лабораторної роботи №6
дисципліни "Програмування"

Тема роботи «Робота з файлами »
сьомий варіант

Виконав:
ст. гр. 123-24-1
О.Д Корж

Прийняв:
ас. каф. ІТКІ

Є.Д Радіонов

Дніпро
2024

Лабораторна робота №6
Робота з динамічними векторами та матрицями

Мета роботи: закріпити знання та набути навичок роботи з функціями файлового введення/виведення верхнього рівня, а також закріпити навички опрацювання динамічних масивів даних.

Хід лабораторної роботи

Розглянуто структуру FILE. Вивчено приклади функції введення/виведення векторів і матриц

Завдання

1. В контексті проєкту *TestMyLib*, реалізувати код нових функцій та модифікувати коди розроблених раніше в роботі №5 функцій для підтримки файлового введення/виведення:

```
double *load_vector_from_text_stream(FILE *stream, double *data, int count);
double **load_matrix_from_text_stream(FILE *stream, double **data, int rows, int cols);
double *load_vector_from_binary_stream(FILE *stream, double *data, int count);
double **load_matrix_from_binary_stream(FILE *stream, double **data, int rows, int cols);
void save_vector_to_text_stream(FILE *stream, const double *data, int count);
void save_matrix_to_text_stream(FILE *stream, const double **data, int rows, int cols);
void save_vector_to_binary_stream(FILE *stream, const double *data, int count);
void save_matrix_to_binary_stream(FILE *stream, const double **data, int rows, int cols);
void print_vector(FILE *stream, const double *data, int count);
void print_matrix(FILE *stream, const double **data, int rows, int cols);
```

При цьому використовувати такі формати зберігання вектора та матриці в бінарному файлі:

– у випадку вектору даних:

<кількість_елементів_вектора> <елемент_1> … <елемент_n>

– у випадку матриці даних:

<кількість_рядків> <кількість_стовпців> <елемент_1> … <елемент_n>

```
#ifndef MYLIB_H

#define MYLIB_H


#include <stdio.h>


// Функції для роботи з динамічними векторами та матрицями

double get_min_from_vector(const double *data, int count);

double get_min_from_matrix(const double **data, int rows, int cols);

double get_max_from_vector(const double *data, int count);

double get_max_from_matrix(const double **data, int rows, int cols);
```

```
double get_avg_from_vector(const double *data, int count);

double get_avg_from_matrix(const double **data, int rows, int cols);

void get_stat_from_vector(const double *data, int count, double *min, double
*max, double *avg);

void get_stat_from_matrix(const double **data, int rows, int cols, double *min,
double *max, double *avg);

double *input_vector_from_console(double *data, int count);

double **input_matrix_from_console(double **data, int rows, int cols);

void print_vector(FILE *stream, const double *data, int count);

void print_matrix(FILE *stream, const double **data, int rows, int cols);


// Нові функції для роботи з файлами

double *load_vector_from_text_stream(FILE *stream, double *data, int count);

double **load_matrix_from_text_stream(FILE *stream, double **data, int rows, int
cols);

double *load_vector_from_binary_stream(FILE *stream, double *data, int count);

double **load_matrix_from_binary_stream(FILE *stream, double **data, int rows,
int cols);

void save_vector_to_text_stream(FILE *stream, const double *data, int count);

void save_matrix_to_text_stream(FILE *stream, const double **data, int rows, int
cols);

void save_vector_to_binary_stream(FILE *stream, const double *data, int count);

void save_matrix_to_binary_stream(FILE *stream, const double **data, int rows,
int cols);


#endif // MYLIB_H


#include <stdio.h>

#include <stdlib.h>

#include "mylib.h"


// Мінімальне значення з вектору

double get_min_from_vector(const double *data, int count)

{

    if (count == 0) {

        return 0;
```

```
    }
    double min = data[0];
    for (int i = 1; i < count; i++)
    {
        if (data[i] < min)
        {
            min = data[i];
        }
    }
    return min;
}

// Мінімальне значення з матриці
double get_min_from_matrix(const double **data, int rows, int cols)
{
    if (rows == 0 || cols == 0) {
        return 0;
    }
    double min = data[0][0];
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            if (data[i][j] < min)
            {
                min = data[i][j];
            }
        }
    }
    return min;
}

// Максимальне значення з вектору
```

```
double get_max_from_vector(const double *data, int count)
{
    if (count == 0) {
        return 0;
    }
    double max = data[0];
    for (int i = 1; i < count; i++)
    {
        if (data[i] > max)
        {
            max = data[i];
        }
    }
    return max;
}

// Максимальне значення з матриці
double get_max_from_matrix(const double **data, int rows, int cols)
{
    if (rows == 0 || cols == 0) {
        return 0;
    }

    double max = data[0][0];
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            if (data[i][j] > max)
            {
                max = data[i][j];
            }
        }
```

```
    }
    return max;
}

// Середнє значення з вектору
double get_avg_from_vector(const double *data, int count)
{
    if (count == 0) {
        return 0;
    }
    double sum = 0;
    for (int i = 0; i < count; i++)
    {
        sum += data[i];
    }
    return sum / count;
}

// Середнє значення з матриці
double get_avg_from_matrix(const double **data, int rows, int cols)
{
    if (rows == 0 || cols == 0) {
        return 0;
    }
    double sum = 0;
    int total_elements = rows * cols;
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            sum += data[i][j];
        }
    }
```

```
    return sum / total_elements;
}

// Статистика для вектору
void get_stat_from_vector(const double *data, int count, double *min, double
*max, double *avg)
{
    *min = get_min_from_vector(data, count);
    *max = get_max_from_vector(data, count);
    *avg = get_avg_from_vector(data, count);
}

// Статистика для матриці
void get_stat_from_matrix(const double **data, int rows, int cols, double *min,
double *max, double *avg)
{
    *min = get_min_from_matrix(data, rows, cols);
    *max = get_max_from_matrix(data, rows, cols);
    *avg = get_avg_from_matrix(data, rows, cols);
}

// Введення вектору з консолі
double *input_vector_from_console(double *data, int count)
{
    data = (double *)malloc(count * sizeof(double));
    if (data == NULL)
    {
        printf("Помилка виділення пам'яті.\n");
        return NULL;
    }
    for (int i = 0; i < count; i++)
    {
        printf("Введіть елемент %d: ", i + 1);
        if (scanf("%lf", &data[i]) != 1) {
```

```
            return NULL;
        }
    }
    return data;
}

// Введення матриці з консолі
double **input_matrix_from_console(double **data, int rows, int cols)
{
    data = (double **)malloc(rows * sizeof(double *));
    for (int i = 0; i < rows; i++)
    {
        data[i] = (double *)malloc(cols * sizeof(double));
    }
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            printf("Введіть елемент [%d][%d]: ", i + 1, j + 1);
            if (scanf("%lf", &data[i][j]) != 1) {
                return NULL;
            }
        }
    }
    return data;
}

void print_vector(FILE *stream, const double *data, int count)
{
    if (count == 0) {
        fprintf(stderr, "Error, count is zero\n");
        fflush(stderr);
        exit(EXIT_FAILURE);
```

```
    }
    for (int i = 0; i < count; i++)
    {
        fprintf(stream, "%lf ", data[i]);
    }
    fprintf(stream, "\n");
}

void print_matrix(FILE *stream, const double **data, int rows, int cols)
{
    if (rows == 0 || cols == 0) {
        fprintf(stderr, "Error, rows or columns is zero\n");
        fflush(stderr);
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            fprintf(stream, "%lf ", data[i][j]);
        }
        fprintf(stream, "\n");
    }
}

// Нові функції для роботи з файлами
double *load_vector_from_text_stream(FILE *stream, double *data, int count)
{
    if (stream == NULL)
    {
        perror("Failed to open file");
        return NULL;
    }
```

```
    if (data == NULL)
    {
        data = (double *)malloc(count * sizeof(double));
        if (data == NULL)
        {
            perror("Failed to allocate memory");
            return NULL;
        }
    }

    int read_count;
    if (fscanf(stream, "%d", &read_count) != 1 || read_count != count)
    {
        fprintf(stderr, "Error: The number of elements in the file does not
match the expected count\n");
        free(data);
        return NULL;
    }

    for (int i = 0; i < count; i++)
    {
        if (fscanf(stream, "%lf", &data[i]) != 1)
        {
            fprintf(stderr, "Error: Failed to read vector element\n");
            free(data);
            return NULL;
        }
    }

    return data;
}
```

```
double **load_matrix_from_text_stream(FILE *stream, double **data, int rows, int
cols)

{


    if (stream == NULL)

    {

        perror("Failed to open file");

        return NULL;

    }


    if (data == NULL)

    {

        data = (double **)malloc(rows * sizeof(double *));

        if (data == NULL)

        {

            perror("Failed to allocate memory");

            return NULL;

        }

        for (int i = 0; i < rows; i++)

        {

            data[i] = (double *)malloc(cols * sizeof(double));

            if (data[i] == NULL)

            {

                perror("Failed to allocate memory");

                for (int j = 0; j < i; j++)

                {

                    free(data[j]);

                }

                free(data);

                return NULL;

            }

        }

    }
```

```
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            if (fscanf(stream, "%lf", &data[i][j]) != 1)
            {
                fprintf(stderr, "Error: Failed to read matrix element\n");
                for (int k = 0; k <= i; k++)
                {
                    free(data[k]);
                }
                free(data);
                return NULL;
            }
        }
    }

    return data;
}

void save_vector_to_text_stream(FILE *stream, const double *data, int count)
{
    if (stream == NULL)
    {
        perror("Failed to open file");
        return;
    }

    fprintf(stream, "%d\n", count);
    for (int i = 0; i < count; i++)
    {
        fprintf(stream, "%lf ", data[i]);
    }
```

```
    fprintf(stream, "\n");
}

void save_matrix_to_text_stream(FILE *stream, const double **data, int rows, int
cols)
{
    fprintf(stream, "%d %d\n", rows, cols);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            fprintf(stream, "%lf ", data[i][j]);
        }
        fprintf(stream, "\n");
    }
}

void save_vector_to_binary_stream(FILE *stream, const double *data, int count)
{
    fwrite(&count, sizeof(int), 1, stream);
    fwrite(data, sizeof(double), count, stream);
}

void save_matrix_to_binary_stream(FILE *stream, const double **data, int rows,
int cols)
{
    fwrite(&rows, sizeof(int), 1, stream);
    fwrite(&cols, sizeof(int), 1, stream);
    for (int i = 0; i < rows; i++)
    {
        fwrite(data[i], sizeof(double), cols, stream);
    }
}
```

```
double *load_vector_from_binary_stream(FILE *stream, double *data, int count)
{
    if (stream == NULL)
    {
        perror("Failed to open file");
        return NULL;
    }

    if (data == NULL)
    {
        data = (double *)malloc(count * sizeof(double));
        if (data == NULL)
        {
            perror("Failed to allocate memory");
            return NULL;
        }
    }

    int read_count;
    if (fread(&read_count, sizeof(int), 1, stream) != 1)
    {
        perror("Failed to read vector count");
        free(data);
        return NULL;
    }

    if (read_count != count)
    {
        fprintf(stderr, "Error: The number of elements in the file does not
match the expected count\n");
        free(data);
        return NULL;
    }
```

```
    if (fread(data, sizeof(double), count, stream) != count)
    {
        perror("Failed to read vector data");
        free(data);
        return NULL;
    }

    return data;
}

double **load_matrix_from_binary_stream(FILE *stream, double **data, int rows,
int cols)
{
    if (stream == NULL)
    {
        perror("Failed to open file");
        return NULL;
    }

    if (data == NULL)
    {
        data = (double **)malloc(rows * sizeof(double *));
        if (data == NULL)
        {
            perror("Failed to allocate memory");
            return NULL;
        }
        for (int i = 0; i < rows; i++)
        {
            data[i] = (double *)malloc(cols * sizeof(double));
            if (data[i] == NULL)
            {
                perror("Failed to allocate memory");
                for (int j = 0; j < i; j++)
```

```
                {
                    free(data[j]);
                }
                free(data);
                return NULL;
            }
        }
    }

    int read_rows, read_cols;
    if (fread(&read_rows, sizeof(int), 1, stream) != 1 || fread(&read_cols, sizeof(int), 1, stream) != 1)
    {
        perror("Failed to read matrix dimensions");
        for (int i = 0; i < rows; i++)
        {
            free(data[i]);
        }
```

```
#include <stdio.h>
#include <stdlib.h>
#include "mylib.h"

int main() {
    int vector_size, rows, cols;
    double *v = NULL;
    double **m = NULL;
    FILE *file;

    // Запит кількості елементів у векторі та матриці
    printf("Введіть кількість елементів у векторі: ");
    scanf("%d", &vector_size);
```

```
printf("Введіть кількість рядків у матриці: ");
scanf("%d", &rows);
printf("Введіть кількість стовпців у матриці: ");
scanf("%d", &cols);

// Тестування введення з консолі
v = input_vector_from_console(v, vector_size);
m = input_matrix_from_console(m, rows, cols);

// Збереження даних у файли
file = fopen("vector.txt", "w");
if (file == NULL) {
    perror("Не вдалося відкрити файл vector.txt для запису");
    return 1;
}
save_vector_to_text_stream(file, v, vector_size);
fclose(file);

file = fopen("matrix.txt", "w");
if (file == NULL) {
    perror("Не вдалося відкрити файл matrix.txt для запису");
    return 1;
}
save_matrix_to_text_stream(file, (const double **)m, rows, cols);
fclose(file);

// Завантаження з файлів
file = fopen("vector.txt", "r");
if (file == NULL) {
    perror("Не вдалося відкрити файл vector.txt для читання");
    return 1;
}
```

```
double *v_loaded = load_vector_from_text_stream(file, NULL, vector_size);
fclose(file);

file = fopen("matrix.txt", "r");
if (file == NULL) {
    perror("Не вдалося відкрити файл matrix.txt для читання");
    return 1;
}
double **m_loaded = load_matrix_from_text_stream(file, NULL, rows, cols);
fclose(file);

// Виведення завантажених даних
printf("\n--- Завантажений вектор ---\n");
print_vector(stdout, v_loaded, vector_size);

printf("\n--- Завантажена матриця ---\n");
print_matrix(stdout, (const double **)m_loaded, rows, cols);

// Вектор для запису у двійковий файл
vector_size = 5;
double vector[] = {1.1, 2.2, 3.3, 4.4, 5.5};

// Приклад матриці
rows = 3; cols = 3;
double *matrix[] = {
    (double[]){1.0, 2.0, 3.0},
    (double[]){4.0, 5.0, 6.0},
    (double[]){7.0, 8.0, 9.0}
};

// Відкриваємо файли для запису в двійковому форматі
FILE *vector_file = fopen("vector.bin", "wb");
FILE *matrix_file = fopen("matrix.bin", "wb");
```

```
    if (vector_file == NULL || matrix_file == NULL) {
        perror("Не вдалося відкрити файл для запису");
        return 1;
    }

    // Записуємо вектор у двійковий файл
    save_vector_to_binary_stream(vector_file, vector, vector_size);
    fclose(vector_file);

    // Записуємо матрицю у двійковий файл
    save_matrix_to_binary_stream(matrix_file, (const double **)matrix, rows,
cols);
    fclose(matrix_file);

    printf("Вектор і матриця успішно збережені у двійкові файли.\n");

    file = fopen("vector.bin", "r");
    if (file == NULL) {
        perror("Не вдалося відкрити файл vector.txt для читання");
        return 1;
    }
    v_loaded = load_vector_from_binary_stream(file, NULL, vector_size);
    fclose(file);

    file = fopen("matrix.bin", "r");
    if (file == NULL) {
        perror("Не вдалося відкрити файл matrix.txt для читання");
        return 1;
    }
    m_loaded = load_matrix_from_binary_stream(file, NULL, rows, cols);
    fclose(file);

    // Виведення завантажених даних
```

```
    printf("\n--- Завантажений вектор ---\n");
    print_vector(stdout, v_loaded, vector_size);

    printf("\n--- Завантажена матриця ---\n");
    print_matrix(stdout, (const double **)m_loaded, rows, cols);

    // Звільнення пам'яті
    free(v);
    free(v_loaded);
    free(m);
    free(m_loaded);

    return 0;
}
```

2. Протестувати в основній програмі виклики нових функцій

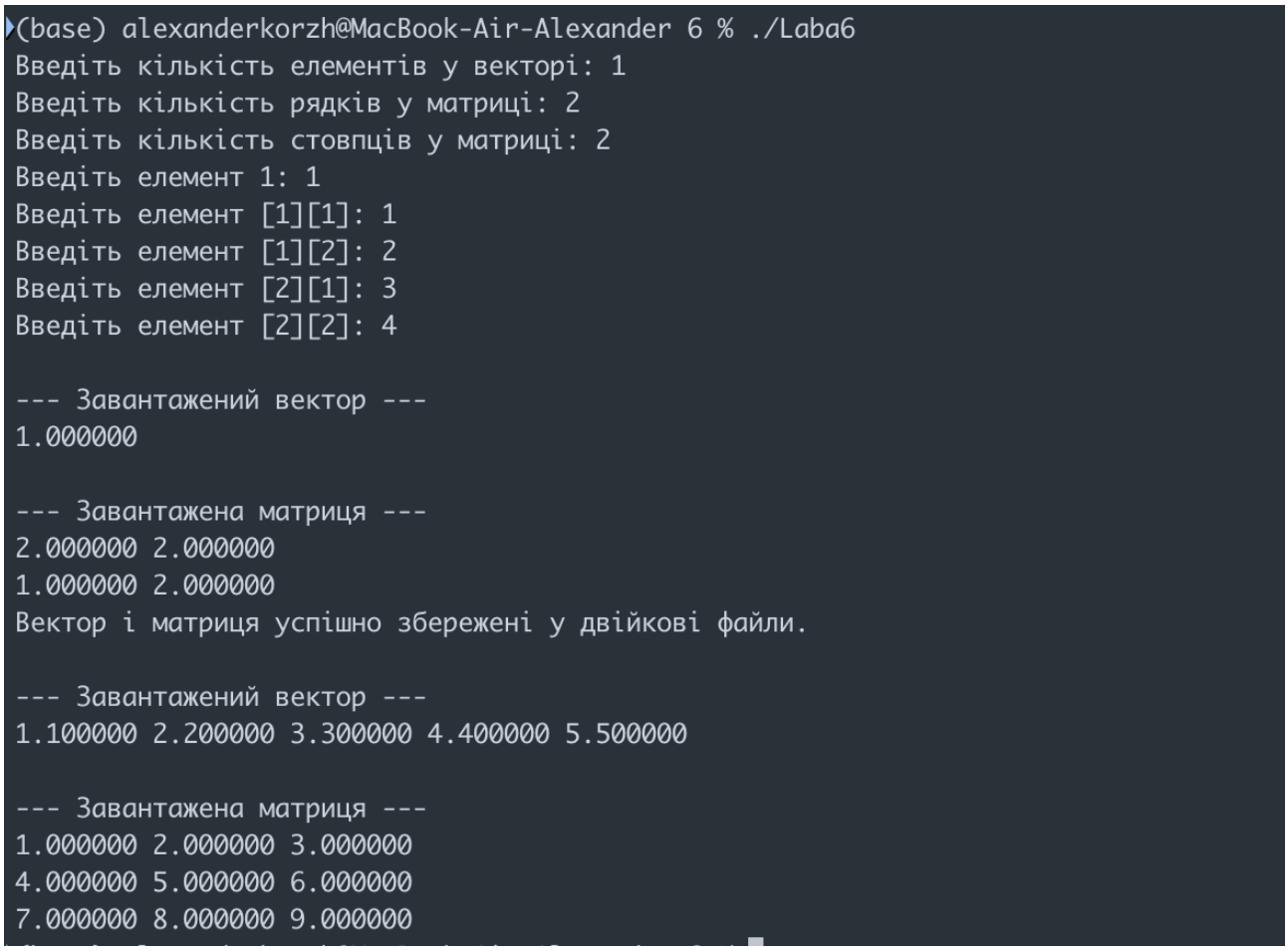

Рисунок 2 – знімок екрану перевірки роботи програми

Висновки

Вивчено на практиці як застосовувати функції введення/виведення. Вивчено структуру FILE. Засвоєно на практиці як працювати з текстовими та бінарними файлами. Під час роботи було використані функції fread(), fwrite()