Основи програмування – 1. Алгоритми та структури даних

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 9 з дисципліни
«Алгоритми та структури даних-1.
Основи алгоритмізації»

«ДОСЛІДЖЕННЯ АЛГОРИТМІВ ОБХОДУ МАСИВІВ»

Варіант 13

Виконав студент ІП-13, Жмайло Дмитро Олександрович
(шифр, прізвище, ім'я, по батькові)

Перевірила Вєчерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2021

Лабораторна робота 9

Дослідження рекурсивних алгоритмів

Мета – дослідити алгоритми обходу масивів, набути практичних навичок використання цих алгоритмів під час складання програмних специфікацій.

Індивідуальне завдання

Варіант 13

№	Опис варіанту
13	Задано матрицю дійсних чисел A[m,n]. В кожному рядку матриці знайти останній максимальний елемент і його місцезнаходження. Обміняти знайдене значення X з елементом останнього стовбця.

Постановка задачі

Необхідно ініціалізувати двовимірний масив розмірності **m** x **n** (беручи до уваги, що індексація масивів починається з нульового елементу) випадковими дійсними значеннями. Потім необхідно пройтися по рядкам цього масиву “змійкою” та знайти індекси максимальних елементів кожного рядка й перенести їх до відповідних елементів одновимірного масиву, який доведеться створити для виконання алгоритму. Потім необхідно в кожному рядку замінити місцями останній елемент рядка та знайдений максимальний елемент рядка.

Побудова математичної моделі

Відповідно до умови складемо таблицю змінних:

Змінна	***Тип***	***Назва***	***Призначення***
Заданий двовимірний масив	Дійсний	matrix	Проміжні дані
Одновимірний масив з індексами мінімальних елементів рядків	Цілий	arr_max	Проміжні дані
Задана розмірність матриці (рядки)	Цілий	m	Вхідні дані
Задана розмірність матриці (стовпчики)	Цілий	n	Вхідні дані
Лічильник j	Цілий	j	Проміжні дані
Лічильник k	Цілий	k	Проміжні дані
Розмірність матриці (рядки) в підпрограмі	Цілий	rows	Проміжні дані (змінна в підпрограмі)
Розмірність матриці (стовпці) в підпрограмі	Цілий	columns	Проміжні дані (змінна в підпрограмі)
Допоміжна змінна для пошуку максимального значення	Дійсний	temp_max	Проміжні дані (змінна в підпрограмі)
Поточний індекс максимального елементу рядка	Дісний	max_index	Проміжні дані (змінна в підпрограмі)
Довільний одновимірний масив у підпрограмі	Цілий	arr	Проміжні дані (змінна в підпрограмі)
Двовимірний масив початкової матриці в підпрограмі	Дійсний	array	Проміжні дані (змінна в підпрограмі)
Мінімальне значення випадкової генерації	Цілий	min_rand	Початкові дані (змінна в підпрограмі)

Максимальне значення випадкової генерації	Цілий	max_rand	Початкові дані (змінна в підпрограмі)
Довжина одновимірного масиву	Цілий	length	Проміжні дані (змінна в підпрограмі)

Для створення цілочисельних випадкових значень будемо використовувати функцію **rand() % a**, де **a** - ціле число; яка буде створювати цілі випадкові числа в діапазоні значень від 0 (включно) до **a** (невключно). Для зручності будемо генерувати значення елементів матриці в діапазоні від **-25 до 25**.

Використаємо функцію **a** % **b**, яка буде знаходити остачу від натурального числа **a** при діленні на натуральне число **b**

Отже, ми будемо заповняти матрицю за допомогою підпрограми **fill_matrix,** яка буде за допомогою двох арифметичних циклів (один вкладений в інший) заповнювати масив розмірності **m** x **n** (враховуючи індексацію з 0-го елементу) випадковими значеннями, згенерованими за допомогою підпрограми функції **rand() % a**, яка буде формувати випадкові дійсні числа за допомогою виразу:

(rand() % (max_rand * 10 - min_rand * 10 + 1) + min_rand * 10) / 10.0;

де **min_rand** - це мінімальна границя генерації, а

max_rand - максимальна границя генерації

(Множенння на 10 і ділення на 10 використано для генерації дійсних чисел)

Використаємо підпрограму **display_matrix,** яка за допомогою двох арифметичних циклів (один вкладений в інший) буде виводити згенеровані значення елементів матриці.

Далі використаємо підпрограму **find_max**, яка за допомогою двох арифметичних циклів (один вкладений в інший) буде ініціалізувати одновимірний масив індексами останніх максимальних значень, взятих з рядків матриці.

За допомогою підпрограми **replace_elements** будемо змінювати початкову матрицю відповідно до умови задачі (максимальний елемент рядка з останнім елементом стовпчика матриці в кожному рядку).

Використаємо підпрограму **display_arr,** яка за допомогою одного арифметичного циклу буде виводити індекси знайдених максимальних елементів у матриці (індекси починаються від 0, як і у масиві)

Розв’язання:

Програмні специфікації запишемо у псевдокоді та графічній формі у вигляді блок-схеми.

Крок 1. Визначимо основні дії;

Крок 2. Деталізуємо підпрограму створення та заповнення матриці випадковими значеннями (**fill_matrix**) та її виклик;

Крок 3. Деталізуємо підпрограму виведення матриці **(display_matrix)** та її виклик;

Крок 4. Деталізуємо підпрограму пошуку індексів максимальних елементів рядків (**find_max**) та її виклик;

Крок 5. Деталізуємо підпрограму виведення одновимірного масиву **(display_arr)** та її виклик;

Крок 6. Деталізуємо підпрограму обміну значень максимального елементу кожного рядка та останнім елементом того ж рядка **(replace_elements)** та її виклик;

Псевдокод:

Основна програма:

початок

Введення **m**

Введення **n**

arr_max[m]

matrix[m][n] := fill_matrix(**m**, **n**)

display_matrix (**matrix, m, n**)

arr_max := find_max(**matrix, m, n**)

display_arr**(arr_max, m**)

matrix := replace_elements**(matrix, arr_max, m, n**)

display_matrix (**matrix, m, n**)

кінець

Підпрограми:

підпрограма fill_matrix(matrix, rows, columns)

min_rand := -25

max_rand := 25

повторити для i **від** 0 **до** rows

повторити для j **від** 0 **до** columns

matrix[i][j] := (rand() % (max_rand * 10 - min_rand * 10 + 1) + min_rand * 10) / 10.0

все повторити

все повторити

все підпрограма

підпрограма display_matrix(array, size)

повторити для i **від** 0 **до** size

повторити для j **від** 0 **до** size

Виведення array[i][j]

все повторити

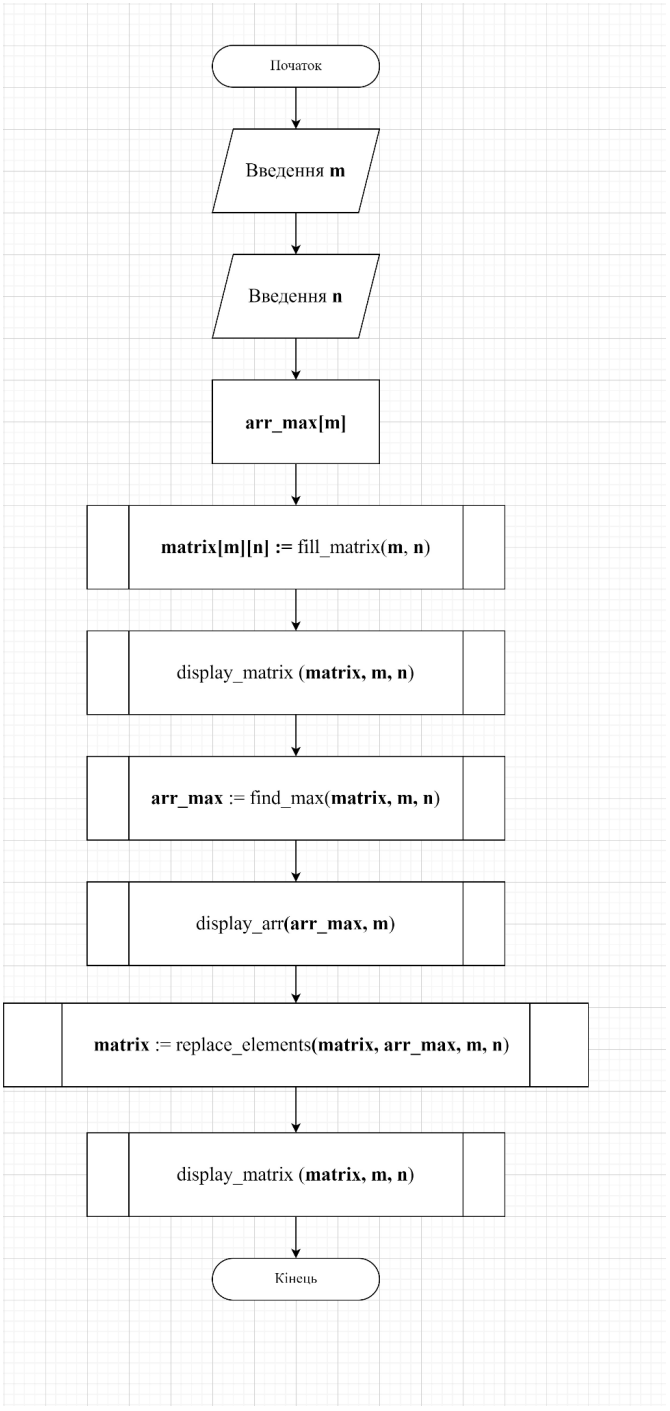
все повторити

все підпрограма

```
підпрограма find_max(array, rows, columns)
    arr[rows]
    повторити для i від 0 до rows
        temp_max := array[i][0];
        max_index := 0;
        якщо (i % 2 == 1)
          то
            повторити для j від 0 до columns
                якщо (array[i][j] >= temp_max)
                  то
                    temp_max := array[i][j];
                    max_index := j;
                все якщо
            все повторити
          інакше
            повторити для j від columns до 0 з кроком -1
                якщо (array[i][j] >= temp_max)
                  то
                    temp_max := array[i][j];
                    max_index := j;
                все якщо
            все повторити
        все якщо
        arr[i] := max_index;
    все повторити
```

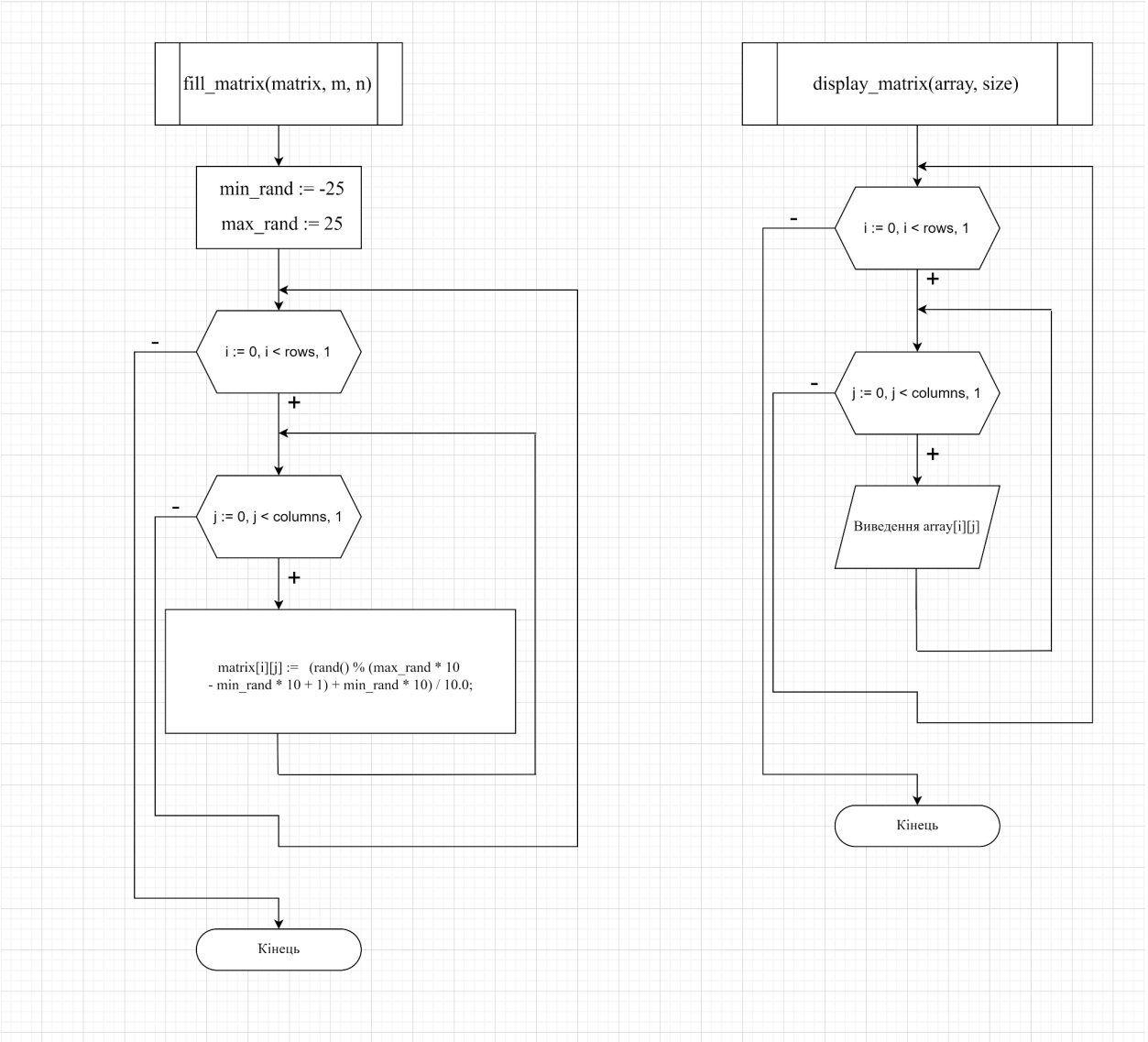
```
        return arr;
все підпрограма



підпрограма display_arr(arr, length)
        повторити для i від 0 до length
            Виведення arr[i]
        все повторити
все підпрограма

підпрограма replace_elements(matrix, arr, rows, columns)
        повторити для i від 0 до rows
            max_index := arr[i]
            temp_max := matrix[i][max_index]
            matrix[i][max_index] := matrix[i][columns - 1]
            matrix[i][columns - 1] := temp_max
        все повторити
все підпрограма
```

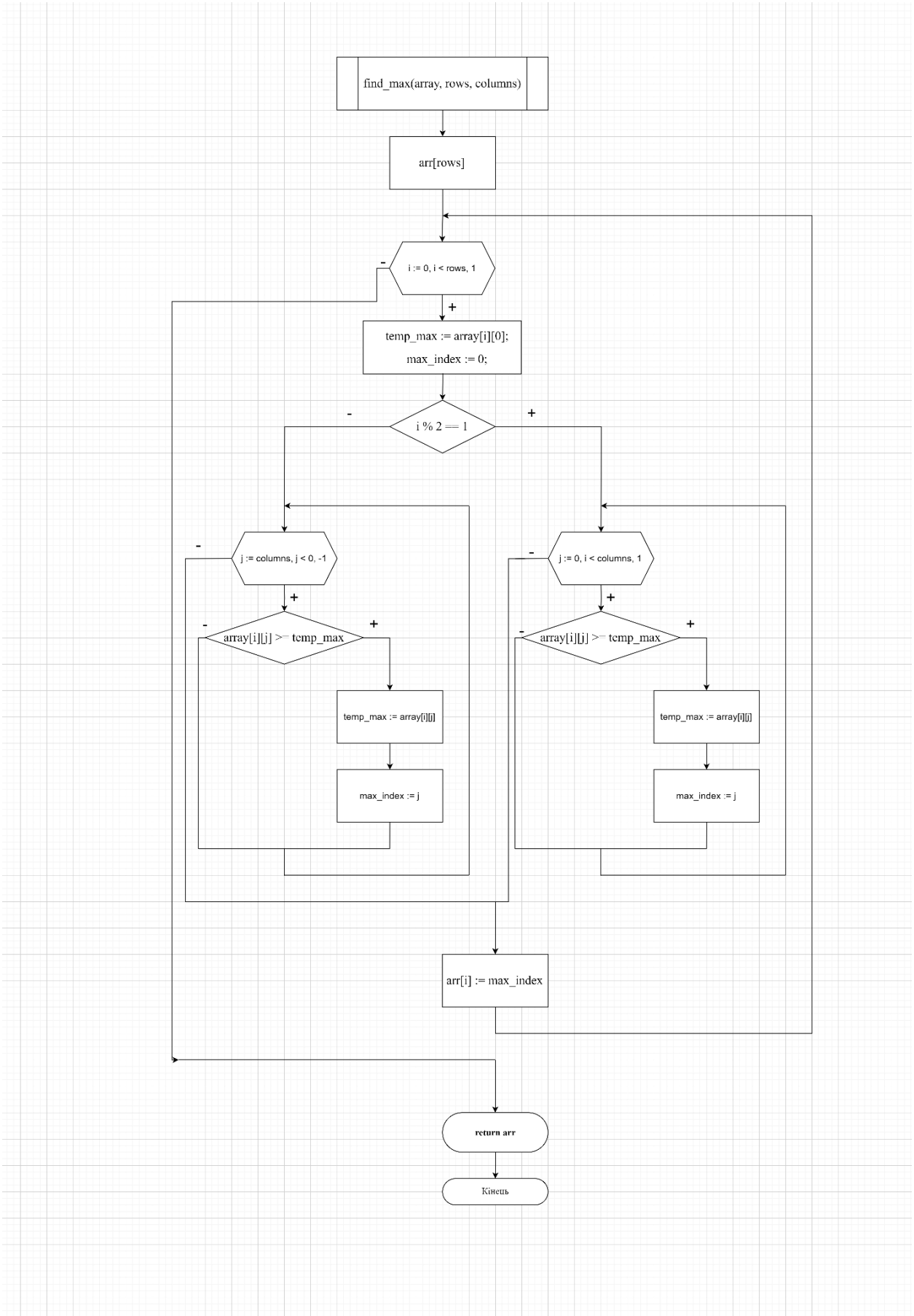
Блок-схема:

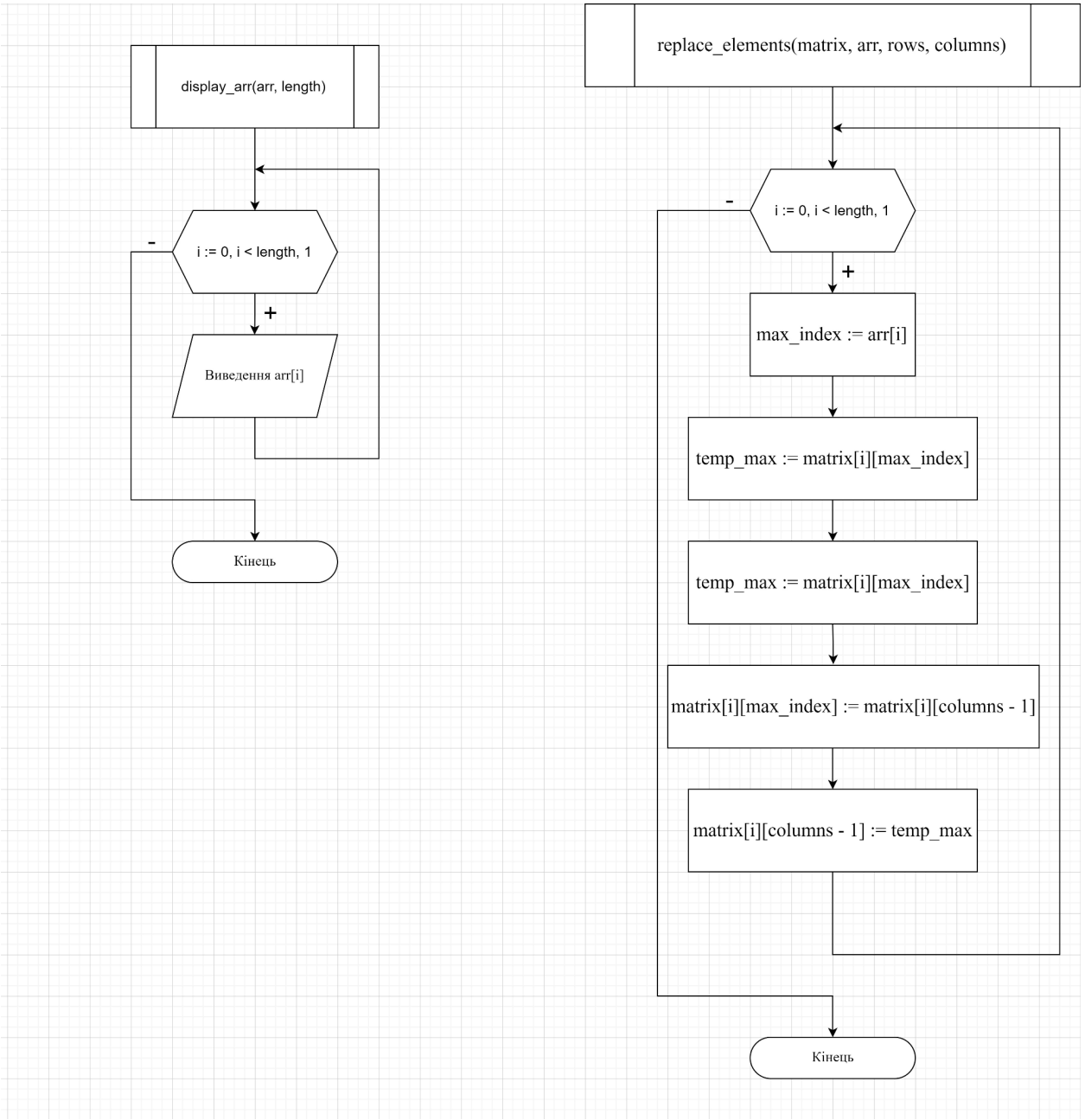


Основна програма:

Підпрограми:




find_max(array, rows, columns)
arr[rows]
i := 0, i < rows, 1
-
+
temp_max := array[i][0];
max_index := 0;
i % 2 == 1
-
+
j := columns, j < 0, -1
-
+
array[i][j] >= temp_max
-
+
temp_max := array[i][j]
max_index := j
j := 0, i < columns, 1
-
+
array[i][j] >= temp_max
-
+
temp_max := array[i][j]
max_index := j
arr[i] := max_index
return arr
Кінець


display_arr(arr, length)
i := 0, i < length, 1
-
+
Виведення arr[i]
Кінець
replace_elements(matrix, arr, rows, columns)
i := 0, i < length, 1
-
+
max_index := arr[i]
temp_max := matrix[i][max_index]
temp_max := matrix[i][max_index]
matrix[i][max_index] := matrix[i][columns - 1]
matrix[i][columns - 1] := temp_max
Кінець

Код програми: (C++)

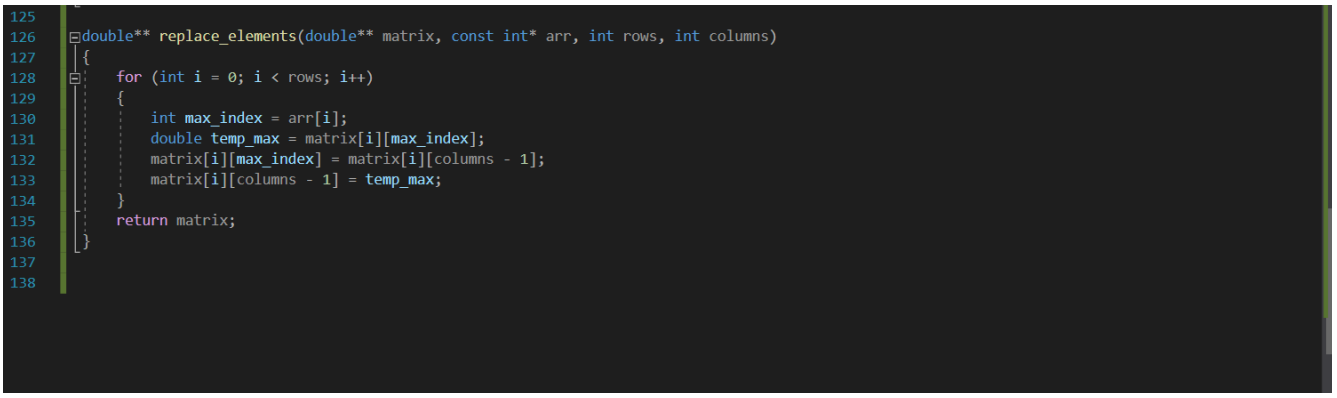
```cpp
#include <iostream>
#include <ctime>
#include <iomanip>
#include <math.h>

using namespace std;

double** fill_matrix(int, int);
void display_matrix(double**, int, int);
void display_arr(int*, int);
void delete_matrix(double**, int);
int* find_max_index(double**, int, int);
double** replace_elements(double**, const int*, int, int);

int main()
{
    srand((unsigned int)time(NULL));
    int m, n;
    cout << "Input matrix size(rows): ";
    cin >> m;
    cout << "Input matrix size(columns): ";
    cin >> n;

    int* array_max;
    double** matrix;

    matrix = fill_matrix(m, n);
    cout << "Generated matrix: " << endl;
    display_matrix(matrix, m, n);

    array_max = find_max_index(matrix, m, n);

    cout << "Indexes of max elements:" << endl;
```

```cpp
    array_max = find_max_index(matrix, m, n);

    cout << "Indexes of max elements:" << endl;
    display_arr(array_max, m);

    matrix = replace_elements(matrix, array_max, m, n);
    cout << "Result: " << endl;
    display_matrix(matrix, m, n);
    delete_matrix(matrix, m);

}
```

```cpp
double** fill_matrix(int rows, int columns)
{
    int min_rand = -25;
    int max_rand = 25;
    srand(time(NULL));
    double** matrix = new double* [rows];
    for (int i = 0; i < rows; i++)
    {
        matrix[i] = new double[columns];
    }
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            matrix[i][j] = (double)(rand() % (max_rand * 10 - min_rand * 10 + 1) + min_rand * 10) / 10.0;
        }
    }
    return matrix;
```

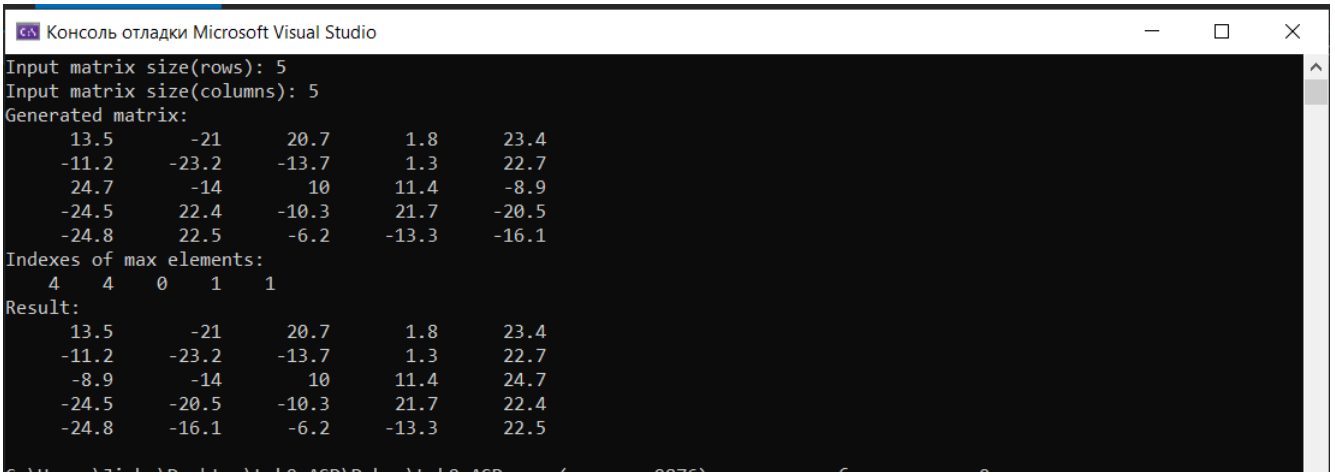
```cpp
void display_matrix(double** matrix, int rows, int columns)
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            cout << setw(10) << matrix[i][j];
        }
        cout << endl;
    }
}

void display_arr(int* arr, int length)
{
    for (int i = 0; i < length; i++)
    {
        cout << setw(5) << arr[i];
    }
    cout << endl;
}

void delete_matrix(double** matrix, int m)
{
    for (int i = 0; i < m; i++) {
        delete[] matrix[i];
    }
    delete[] matrix;
}
```

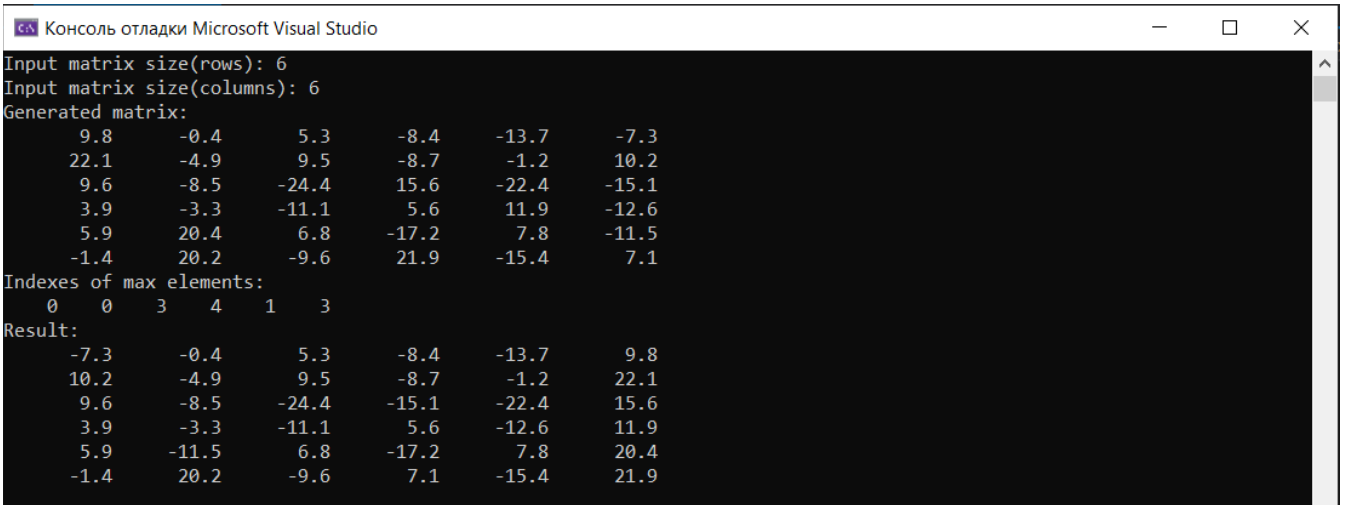
```cpp
int* find_max_index(double** array, int m, int n) {
    int* arr = new int[m];
    for (int i = 0; i < m; i++)
    {
        double temp_max = array[i][0];
        int max_index = 0;
        if (i % 2 == 1)
        {
            for (int j = 0; j < n; j++)
            {
                if (array[i][j] >= temp_max)
                {
                    temp_max = array[i][j];
                    max_index = j;
                }
            }
        }
        else
        {
            for (int j = n; j > 0; j--)
            {
                if (array[i][j] >= temp_max)
                {
                    temp_max = array[i][j];
                    max_index = j;
                }
            }
        }

        arr[i] = max_index;
    }
    return arr;
}
```



```
125
126 double** replace_elements(double** matrix, const int* arr, int rows, int columns)
127 {
128     for (int i = 0; i < rows; i++)
129     {
130         int max_index = arr[i];
131         double temp_max = matrix[i][max_index];
132         matrix[i][max_index] = matrix[i][columns - 1];
133         matrix[i][columns - 1] = temp_max;
134     }
135     return matrix;
136 }
137
138
```

Випробування програми:





Висновок: На цій лабораторній роботі ми дослідили дослідили алгоритми обходу масивів, навчилися застосовувати їх у програмах та описувати їх у вигляді псевдокоду та блок-схем. Випробували алгоритм вручну та за допомогою мови С++ та звірили результати. Закріпили знання щодо алгоритмів обходу масивів на практиці та побачили їх у дії.

Код програми(текстовий вигляд):

```
#include <iostream>
#include <ctime>
#include <iomanip>
#include <math.h>

using namespace std;

double** fill_matrix(int, int);
void display_matrix(double**, int, int);
void display_arr(int*, int);
void delete_matrix(double**, int);
int* find_max_index(double**, int, int);
double** replace_elements(double**, const int*, int, int);

int main()
{
   srand((unsigned int)time(NULL));
   int m, n;
   cout << "Input matrix size(rows): ";
```

```
    cin >> m;
    cout << "Input matrix size(columns): ";
    cin >> n;

    int* array_max;
    double** matrix;

    matrix = fill_matrix(m, n);
    cout << "Generated matrix: " << endl;
    display_matrix(matrix, m, n);

    array_max = find_max_index(matrix, m, n);

    cout << "Indexes of max elements:" << endl;
    display_arr(array_max, m);

    matrix = replace_elements(matrix, array_max, m, n);
    cout << "Result: " << endl;
    display_matrix(matrix, m, n);
    delete_matrix(matrix, m);

}

double** fill_matrix(int rows, int columns)
{
```

```
    int min_rand = -25;
    int max_rand = 25;
    srand(time(NULL));
    double** matrix = new double* [rows];
    for (int i = 0; i < rows; i++)
    {
        matrix[i] = new double[columns];
    }
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            matrix[i][j] = (double)(rand() % (max_rand * 10 - min_rand * 10 + 1)
+ min_rand * 10) / 10.0;
        }
    }
    return matrix;
}

void display_matrix(double** matrix, int rows, int columns)
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
```

```
            cout << setw(10) << matrix[i][j];
        }
        cout << endl;
    }
}

void display_arr(int* arr, int length)
{
    for (int i = 0; i < length; i++)
    {
        cout << setw(5) << arr[i];
    }
    cout << endl;
}

void delete_matrix(double** matrix, int m)
{
    for (int i = 0; i < m; i++) {
        delete[] matrix[i];
    }
    delete[] matrix;
}

int* find_max_index(double** array, int rows, int columns)
{
```

```
int* arr = new int[rows];
for (int i = 0; i < rows; i++)
{
   double temp_max = array[i][0];
   int max_index = 0;
   if (i % 2 == 1)
   {
      for (int j = 0; j < columns; j++)
      {
         if (array[i][j] >= temp_max)
         {
            temp_max = array[i][j];
            max_index = j;
         }
      }
   }
   else
   {
      for (int j = columns; j > 0; j--)
      {
         if (array[i][j] >= temp_max)
         {
            temp_max = array[i][j];
            max_index = j;
         }
```

```
            }
        }

        arr[i] = max_index;
    }
    return arr;
}

double** replace_elements(double** matrix, const int* arr, int rows, int columns)
{
    for (int i = 0; i < rows; i++)
    {
        int max_index = arr[i];
        double temp_max = matrix[i][max_index];
        matrix[i][max_index] = matrix[i][columns - 1];
        matrix[i][columns - 1] = temp_max;
    }
    return matrix;
}
```