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 3

Виконав студент ІП-15, Борисик Владислав Тарасович
(шифр, прізвище, ім'я, по батькові)

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

Київ 2021

Лабораторна робота №9
Дослідження алгоритмів обходу масивів

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

Варіант 3

Задача

Розробити алгоритм та написати програму, яка складається з наступних дій:

1. Опису змінної індексованого типу (двовимірний масив) згідно з варіантом.
2. Ініціювання змінної, що описана в п.1 даного завдання.
3. Обчислення змінної, що описана в п.1, згідно з варіантом.

3	Задано матрицю дійсних чисел A[m,n]. При обході матриці по рядках знайти в ній перший додатний елемент X і його місцезнаходження. Порівняти X із середньоарифметичним значенням елементів під побічною діагоналлю.

Постановка задачі

За умовою задачі потрібно створити двовимірний масив (матрицю) та ініціалізувати його, обійти цей масив "змійкою" по рядках і знайти в ньому перший додатний елемент і його місцезнаходження у масиві, порівняти знайдений елемент із середньоарифметичним значенням елементів під побічною діагоналлю.

Результатом розв'язку вивід результату порівняння елемента X та середньоарифметичного значення елементів під побічною діагоналлю.

Побудова математичної моделі

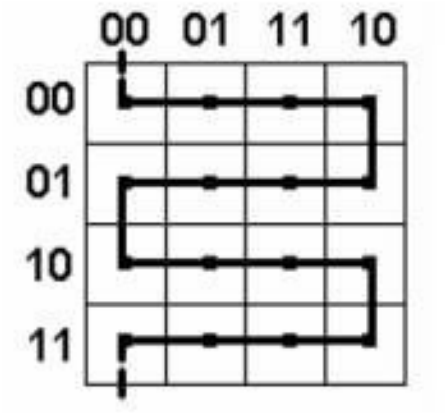
Складемо таблицю змінних

Змінна	Тип	Ім’я	Призначення
Розмір масиву	Цілий	size	Початкове дане
Двовимірний масив (матриця)	Дійсний	matrix	Проміжне дане
Перший додатний елемент масиву	Дійсний	X	Проміжне дане
Позиція елемента X по x	Цілий	x_pos	Проміжне дане

Позиція елемента X по y	Цілий	y_pos	Проміжне дане
Середньоарифметичне значенням елементів під побічною діагоналлю	Дійсний	average	Проміжне дане
Мінімальне значення випадкового числа	Дійсний	min	Проміжне дане
Максимальне значення випадкового числа	Дійсний	max	Проміжне дане
Проміжне значення для функції find_element()	Дійсний	value	Проміжне дане
Кількість чисел під побічною діагоналюю	Цілий	num_count	Проміжне дане

Для позначення операції *остачі від ділення* будемо використовувати оператор %.
Для позначення операції *менше дорівнює* будемо використовувати оператор <=.
Для позначення операції *більше* будемо використовувати оператор >.
Для позначення операції *рівне* будемо використовувати оператор ==.
Для позначення операції *ділення* будемо використовувати оператор /.
Для позначення логічної операції *і* будемо використовувати оператор &&.
Для того, щоб показати що аргумент функції передається за посиланням будемо використовувати оператор &.

1) Для обходу матриці по рядках будемо використовувати метод "змійки":



2) У прямокутній матриці розміром [m,n] неможливо визначити побічну діагональ, і, як наслідок, елементи під побічною діагоналлю. Тож, в задачі будемо генерувати квадратну матрицю [n,n].

Для генерації випадкових дійсних чисел будемо використовувати функцію

RandDouble(min, max). Ця функція приймає 2 аргументи:

min - мінімальне значення випадкового числа. Число дійсного типу.

max - максимальне значення випадкового числа. Число дійсного типу.
Функція повертає дійсне число в діапазоні між аргументами min і max включно.

Для ініціювання двовимірного масиву будемо використовувати власну функцію generate_matrix(&matrix, size).

Ця функція приймає 2 аргументи:

matrix - матриця (двовимірний масив), яка буде використана для ініціювання.
size - кількість рядків/стовпців двовимірного масиву. Число цілого типу.
Функція за допомогою іншої функції RandDouble(min, max) генерує дійсні випадкові числа і записує їх в двовимірний масив matrix. Нічого не повертає.

Для знаходження першого додатного елемента матриці і його позиції по x та y будемо використовувати власну функцію find_element(matrix, size, &x_pos, &y_pos).
Ця функція приймає 4 аргументи:

matrix - матриця (двовимірний масив).

size - кількість рядків/стовпців двовимірного масиву. Число цілого типу.

x_ pos - позиція першого додатного елемента по x. Число цілого типу.

y_ pos - позиція першого додатного елемента по y. Число цілого типу.

Функція повертає значення першого додатного елемента матриці.

Для знаходження середньоарифметичного значення елементів під побічною діагоналлю будемо використовувати власну функцію calculate_average(matrix, size).

Ця функція приймає 2 аргументи:

matrix - матриця (двовимірний масив).

size - кількість елементів масиву array. Число цілого типу.

Функція повертає середньоарифметичне значення елементів під побічною діагоналлю.

Для виведення результату порівняння першого додатного елементу і середньоарифметичного значення елементів під побічною діагоналлю будемо використовувати власну функцію compare(X, average).

Ця функція приймає 2 аргументи:

X - перший додатний елемент матриці. Число дійсного типу.

average - середньоарифметичне значення елементів під побічною діагоналлю.

Число дійсного типу.

Функція нічого не повертає.

1) Створюємо змінну size і присвоюємо їй значення, яке введе користувач.
2) Створюємо двовимірний масив matrix з розміром [size, size].
3) Ініціалізуємо двовимірний масив matrix за допомогою функції generate_matrix(&matrix, size).
4) Створюємо змінні x_pos і y_pos.
5) Створюємо змінну X і ініціалізуємо її за допомогою функції find_element(matrix, size, &x_pos, &y_pos).
6) Створюємо змінну average і ініціалізуємо її за допомогою функції

calculate_average(matrix, size).

7) Виводимо в консоль результат порівняння X і average за допомогою функції compare(X, average).

Розв'язання

Програмні специфікації запишемо у псевдокоді та графічній формі у вигляді

блок-схеми.

Крок 1. Визначимо основні дії.

Крок 2. Створення змінної size і присвоєння їй значення, яке введе користувач.

Крок 3. Створення двовимірного масиву matrix і його ініціалізація за допомогою функції generate_matrix(&matrix, size).
Крок 4. Створення змінних x_pos і y_pos.
Крок 5. Створення змінної X і ініціалізація її за допомогою функції

find_element(matrix, size, &x_pos, &y_pos)

Крок 6. Створення змінної average і її ініціалізація за допомогою функції

calculate_average(matrix, size).

Крок 7. Виведення в консоль результату порівняння X і average за допомогою функції compare(X, average).

Псевдокод

Підпрограма generate_matrix(&matrix, size):

Початок

min := -5.0

max := 5.0

повторити

для i **від** 0 **до** size

  повторити

  для j **від** 0 до size

    matrix[i][j] := RandDouble(min, max)

  все повторити

все повторити

Кінець

Підпрограма find_element(matrix, size, &x_pos, &y_pos):

value := 0

повторити

для i **від** 0 **до** size **з кроком** 1 **і** value <= 0

  якщо i % 2 == 0

    то

      повторити

      для j **від** 0 **до** size **з кроком** 1

        якщо matrix[i][j] > 0

          то

            value := matrix[i][j]

            x_pos := i+1

            y_pos := j+1

        все якщо

      все повторити

    інакше

      повторити

```
        для j від size-1 до 0 з кроком -1
            якщо matrix[i][j] > 0
                то
                    value := matrix[i][j]
                    x_pos := i+1
                    y_pos := j+1
            все якщо
        все повторити
    все якщо
все повторити
Кінець

Підпрограма calculate_average(matrix, size):
Початок
num := 0
num_count := 0
повторити
для i від 1 до size з кроком 1
    повторити
    для j від size-i до size з кроком 1
        sum := sum + matrix[i][j]
        num_count := num_count + 1
    все повторити
все повторити
average := sum/num_count
повернути average
Кінець

Підпрограма compare(X, average):
Початок
якщо X > average
```

то

виведення "X більше за average"

інакше

виведення " X менше за average "

все якщо

Кінець

Основна програма:

size

введення size

matrix[size][size]

generate_matrix(matrix, size)

x_pos := 0

y_pos := 0

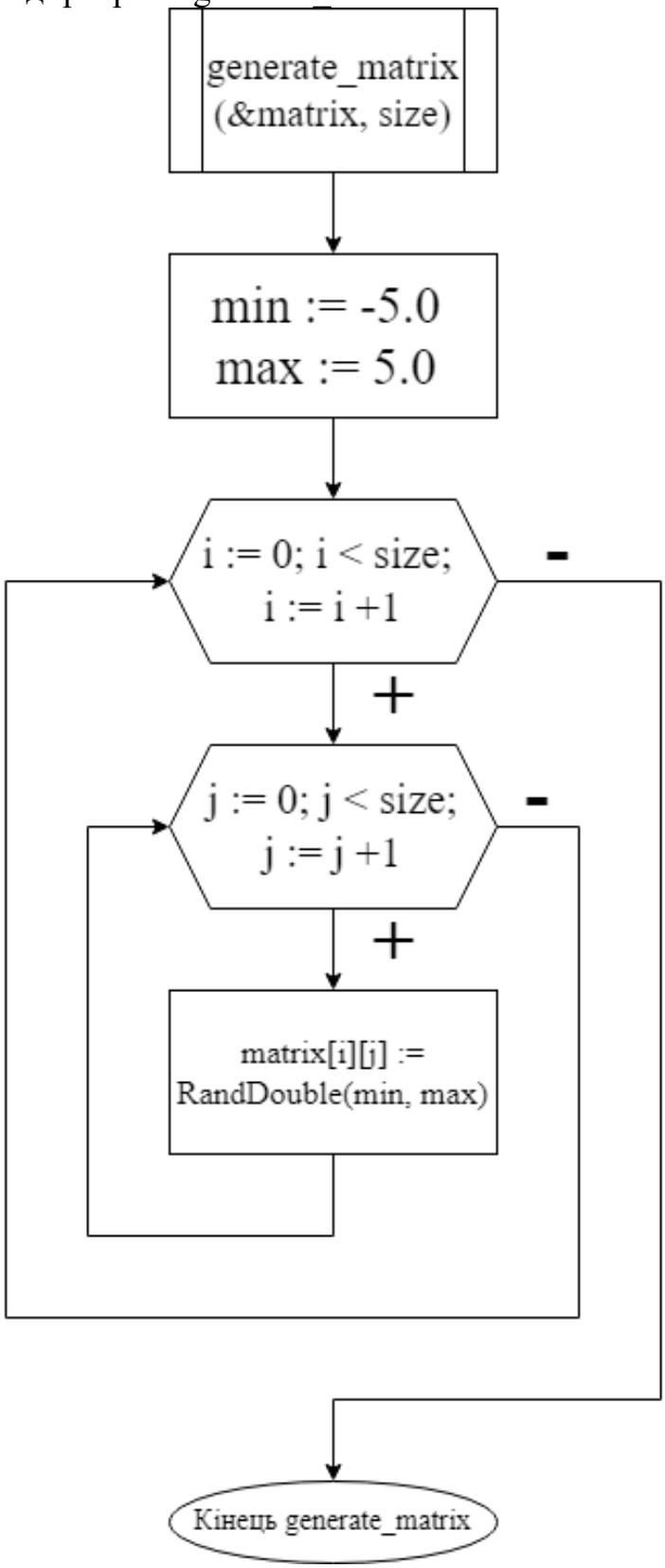
X := find_element(matrix, size, x_pos, y_pos)

average := calculate_average(matrix, size)

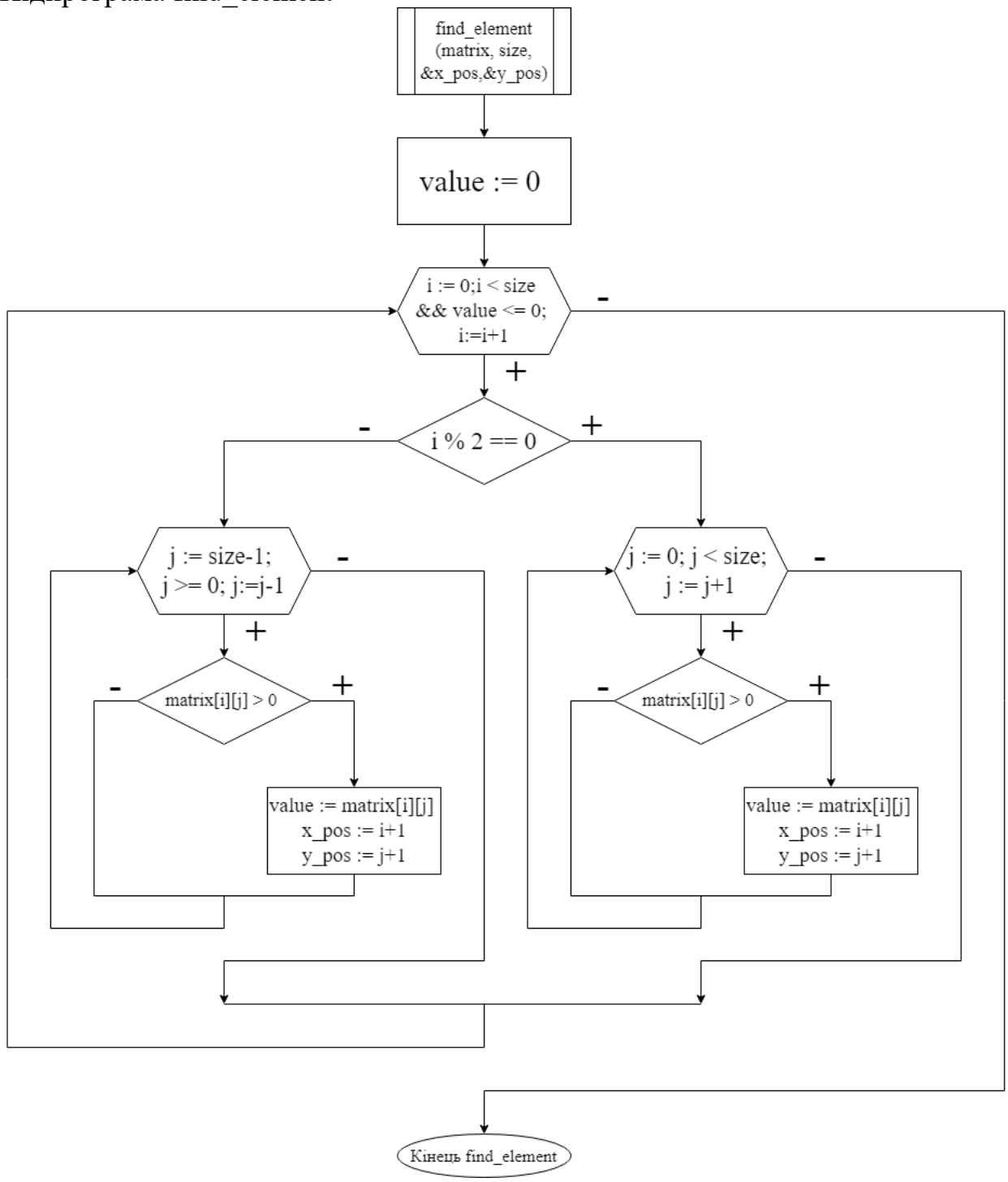
compare(X, average)

Блок-схема алгоритму

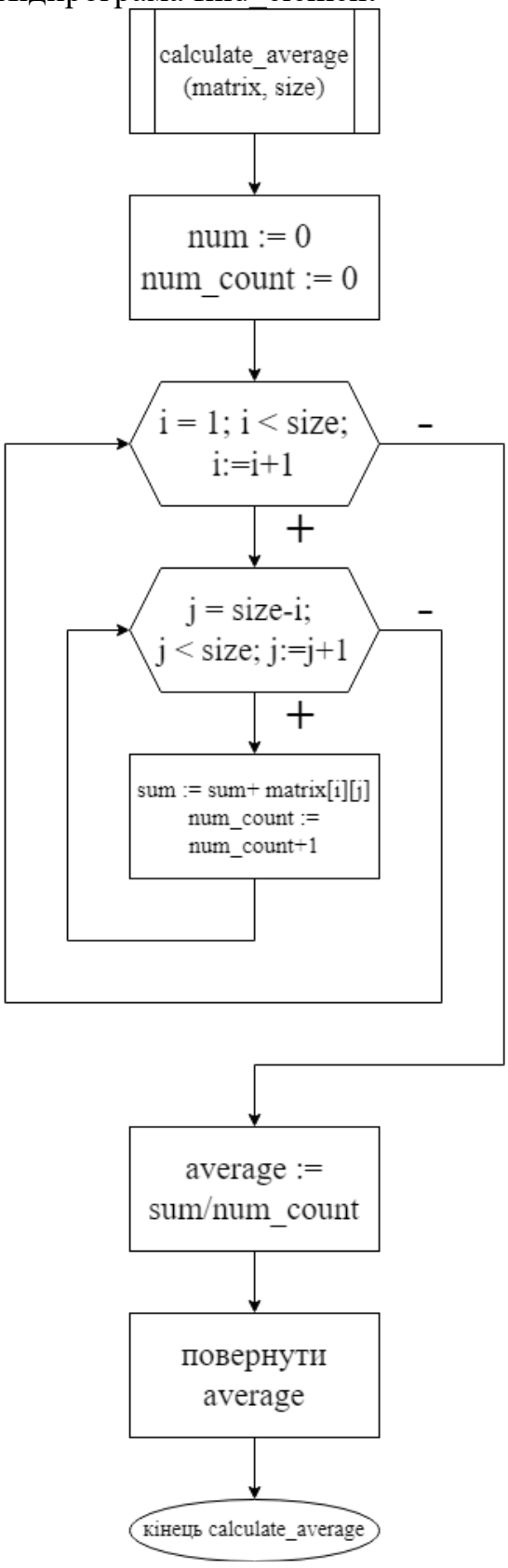
Підпрограма generate_matrix



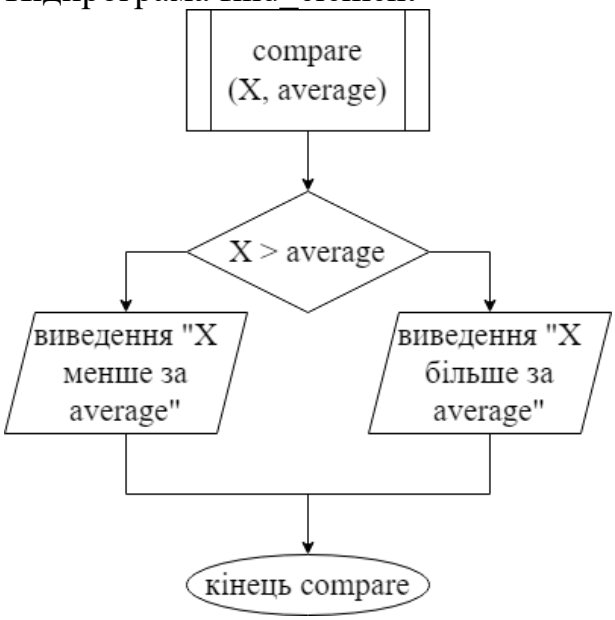
Підпрограма find_element



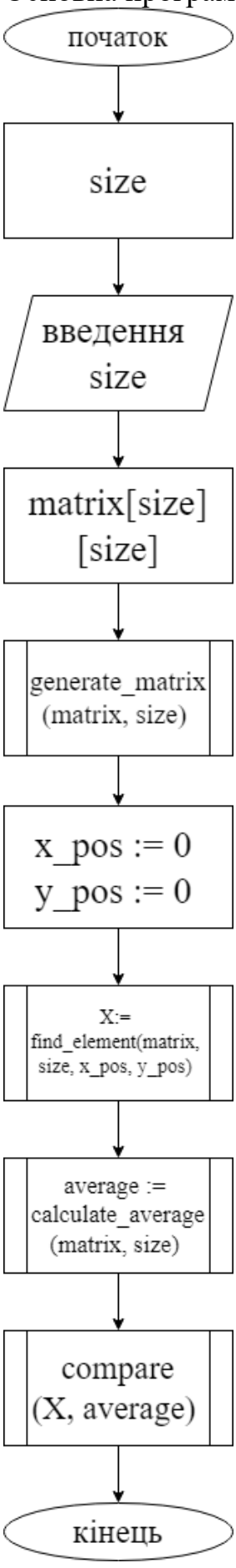
Підпрограма find_element



Підпрограма find_element



Основна програма



Код програми

```cpp
#include <iostream>
#include <ctime>
using namespace std;

void generate_matrix(float** matrix, int size);
void print_matrix(float** matrix, int size);
float find_element(float** matrix, int size, int &x_pos, int &y_pos);
float calculate_average(float** matrix, int size);
void compare(float X, float average);

int main() {
    srand(time(nullptr));

    int size = 0;
    cout << "Enter matrix size:";
    cin >> size;

    // створюємо матрицю
    auto **matrix = new float *[size];
    for(int i = 0; i < size; i++)
        matrix[i] = new float[size];

    // генеруємо матрицю випадковими дійсними числами
    generate_matrix(matrix, size);
    // виводимо в консоль елементи матриці
    print_matrix(matrix, size);

    // позиція елемента по x
```

```
    int x_pos = 0;
    // позиція елемента по y
    int y_pos = 0;
    // перший додатний елемент матриці (обхід робимо "змійкою" по рядках)
    float X = find_element(matrix, size,  &x_pos,  &y_pos);
    printf("First positive element: %.2f. His position x: %d, y: %d\n", X, x_pos, y_pos);

    // середньоарифметичне значення елементів під побічною діагоналлю
    float average = calculate_average(matrix, size);
    printf("Average: %.2f\n", average);

    // порівнюємо X та average
    compare(X, average);
}


void generate_matrix(float** matrix, int size){
    //мінімальне значеннння
    float min = -5.0;
    //максимальне значеннння
    float max =  5.0;

    for(int i = 0; i < size; i++){
        for(int j = 0; j < size; j++){
            //генеруємо випадкові дійсні числа від -5 до 5:
            //RAND_MAX - константа, значення якої = 32767
            //static_cast<float>(rand()) / RAND_MAX - генерує випадкові дійсні числа від 0 до 1
            //(max - min) * static_cast<float>(rand()) / RAND_MAX - генерує випадкові числа від 0 до 10
            //(max - min) * static_cast<float>(rand()) / RAND_MAX + min - генерує випадкові числа від -5 до 5
            matrix[i][j] = (max - min) * (static_cast<float>(rand()) / RAND_MAX) + min;
        }
    }
}

void print_matrix(float** matrix, int size){
    // виводимо в консоль елементи матриці з точністю до двох цифр після коми включно
    for(int i = 0; i < size; i++){
        for(int j = 0; j < size; j++)
            printf(" %5.2f ", matrix[i][j]);

        cout << "\n";
    }
    cout << "\n";
}

float find_element(float** matrix, int size, int &x_pos, int &y_pos){
    float value = 0;

    // прохід по рядках
    for(int i = 0; i < size && value <= 0; i++){
        // якщо номер рядку непарний (1,3,5,...) (але індекс парний)
        if(i % 2 == 0){
            // то проходимо його зліва на право
            for(int j = 0; j < size; j++)
                // якщо цей елемент більше за 0, то value = цьому елементу
```

```
84                  // також змінюємо значення позиції цього елемента по x і y
85                  if(matrix[i][j] > 0){
86                      value = matrix[i][j];
87                      x_pos = ++i;
88                      y_pos = ++j;
89                  }
90          }
91          // якщо номер рядку парний (2,4,6,...) (але індекс непарний)
92          else{
93              // то проходимо справа на ліво
94              for(int j = size - 1; j >= 0; j--)
95                  // якщо цей елемент більше за 0, то value = цьому елементу
96                  // також змінюємо значення позиції цього елемента по x і y
97                  if(matrix[i][j] > 0){
98                      value = matrix[i][j];
99                      x_pos = ++i;
100                     y_pos = ++j;
101                 }
102         }
103     }
104
105     return value;
106 }
107
108 float calculate_average(float** matrix, int size){
109     float sum = 0;
110     float num_count = 0;
111     cout << "Elements under the side diagonal:";
```

```
112
113     for(int i = 1; i < size; i++){
114         for(int j = size-i; j < size; j++){
115             printf(" %.2f", matrix[i][j]);
116             sum += matrix[i][j];
117             num_count++;
118         }
119     }
120     cout << "\n";
121
122     float average = sum/num_count;
123     return average;
124 }
125
126 void compare(float X, float average){
127     if(X > average)
128         printf("X(%.2f) is greater than average(%.2f)", X, average);
129     else
130         printf("X(%.2f) is less than average(%.2f)", X, average);
131 }
```

Результат виконання програми

```
"F:\Programs\CLion 2021.1.2\Projects\Labwork_9_ASD\cmake-build-debug\Labwork_9_ASD.exe"
Enter matrix size:5
 -3.02  -4.58  -0.59   3.39  -2.87
  3.20   4.94  -0.38   1.83   3.62
 -3.46  -2.84  -0.75  -0.26  -2.86
 -3.98  -1.86   4.36  -3.60  -2.07
 -3.91   1.21  -1.49  -0.74   2.21

First positive element: 3.39. His position x: 1, y: 4
Elements under the side diagonal: 3.62 -0.26 -2.86 4.36 -3.60 -2.07 1.21 -1.49 -0.74 2.21
Average: 0.04
X(3.39) is greater than average(0.04)
Process finished with exit code 0
```

Висновок

Протягом дев'ятої лабораторної роботи я дослідив алгоритми обходу масивів, набув практичних навичок використання цих алгоритмів під час складання програмних специфікацій. В результаті виконання роботи я отримав алгоритм, який створює двовимірний масив (матрицю) та ініціалізує його, обходить цей масив "змійкою" по рядках і знаходить в ньому перший додатний елемент і

його місцезнаходження у масиві, порівнює знайдений елемент із середньоарифметичним значенням елементів під побічною діагоналлю.