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 № 8 з дисципліни
«Алгоритми та структури даних-1.
Основи алгоритмізації»

«Дослідження алгоритмів пошуку та сортування»

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 6

Дослідження рекурсивних алгоритмів

Мета – дослідити алгоритми пошуку та сортування, набути практичних навичок використання цих алгоритмів під час складання програмних специфікацій.

Індивідуальне завдання

Варіант 13

№ варіанта	Розмірність	Тип даних	Обчислення значень елементів одновимірного масиву
13	6 x 6	Цілий	Із додатних значень елементів головної діагоналі двовимірного масиву. Відсортувати методом бульбашки за зростанням.

Постановка задачі

Необхідно ініціалізувати двовимірний масив розмірності 5x5 (враховуючи, що індексація елементів масива починається з 0) випадковими цілими значеннями (додатними та від’ємними). Потім необхідно створити одновимірний масив, використовуючи додатні елементи головної діагоналі початкового двовимірного масиву й відсортувати його елементи методом бульбашки за зростанням.

Побудова математичної моделі

Відповідно до умови складемо таблицю змінних:

Змінна	***Тип***	***Назва***	***Призначення***
Заданий двовимірний масив	Цілий	matrix	Початкові дані
Одновимірний масив з додатних значень головної діагоналі	Цілий	arr_diagonal	Проміжні дані
Задана розмірність матриці	Цілий	n	Початкові дані
Лічильник i	Цілий	i	Проміжні дані
Лічильник j	Цілий	j	Проміжні дані
Лічильник k	Цілий	k	Проміжні дані
Змінна для виконання бульбашкового сортування	Цілий	temp_bubble	Проміжні дані
Розмірність матриці в підпрограмі	Цілий	size	Проміжні дані (змінна в підпрограмі)
Розмірність одновимірного масиву в підпрограмі	Цілий	length	Проміжні дані (змінна в підпрограмі)
Довільний двовимірний масив у підпрограмі	Цілий	array	Проміжні дані (змінна в підпрограмі)
Довільний одновимірний масив у підпрограмі	Цілий	arr	Проміжні дані (змінна в підпрограмі)

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

Отже, ми будемо заповняти матрицю за допомогою підпрограми **create_matrix,** яка буде за допомогою двох арифметичних циклів (один вкладений в інший) заповнювати масив розмірності **5x5** (враховуючи індексацію з 0-го елементу) випадковими значеннями, згенерованими за допомогою функції **rand(a, b)**.

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

Далі використаємо підпрограму **create_arr**, яка за допомогою арифметичного циклу буде ініціалізувати одновимірний масив додатними значеннями, взятими з головної діагоналі матриці.

За допомогою підпрограми **display_arr** виведемо значення елементів одновимірного масиву, використовуючи арифметичний цикл з додатковлю умовою, що значення поточного елемента не дорівнює нулю.

За допомогою підпрограми **bubble_sort,** використовуючи бульбашкове сортування відсортуємо значення одновимірного масиву за зростанням та потім виведемо результат знову використавши підпрограму **display_arr**.

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 (**create_matrix**) та її виклик;

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

Крок 4. Деталізуємо підпрограму створення та заповнення одновимірного масиву додатними значеннями з головної діагоналі матриці **(create_arr)** та її виклик;

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

Крок 6. Деталізуємо підпрограму елементів одновимірного масиву за зростанням за допомогою бульбашкового сортування **(bubble_sort)** та її виклик;

Крок 7. Деталізуємо виклик підпрограми **display_arr** для перегляду результату виконання програми;

Псевдокод:

Основна програма:

початок

n := 6

matrix[n][n]

arr_diagonal[n]

create_matrix(matrix, n)

display_matrix(matrix, n)

create_arr(arr_diagonal, matrix, n)

display_arr(arr_diagonal, n)

bubble_sort(arr_diagonal, n)

display_arr(arr_diagonal, n)

кінець

Підпрограми:

підпрограма create_matrix(matrix, size)

min_rand := -25

max_rand := 25

повторити для i **від** 0 **до** size

повторити для j **від** 0 **до** size

matrix[i][j] := rand(min_rand, max_rand)

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

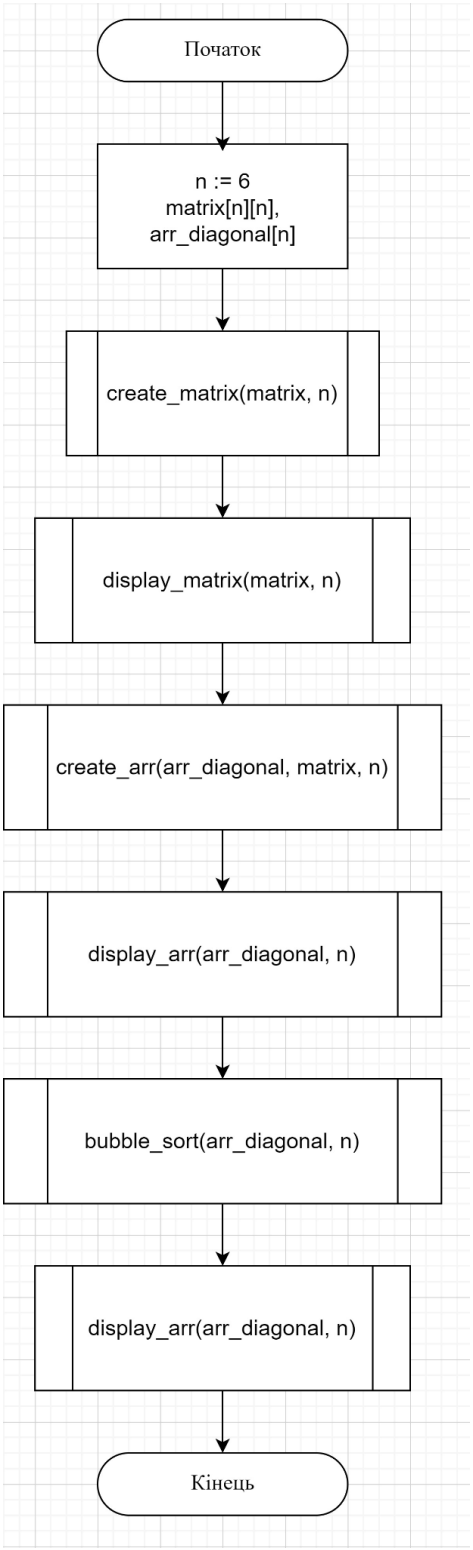
```
підпрограма create_arr(arr_diagonal, array, size)
        k := 0
        повторити для i від 0 до size
                якщо array[i][i] > 0
                  то
                        arr_diagonal[k] := array[i][i]
                        k := k + 1
                все якщо
        все повторити
все підпрограма

підпрограма display_arr(arr, length)
        i := 0
        поки ( (i < length) && (arr[i] != 0) ) повторити
              Виведення arr[i]
        все повторити
все підпрограма
```

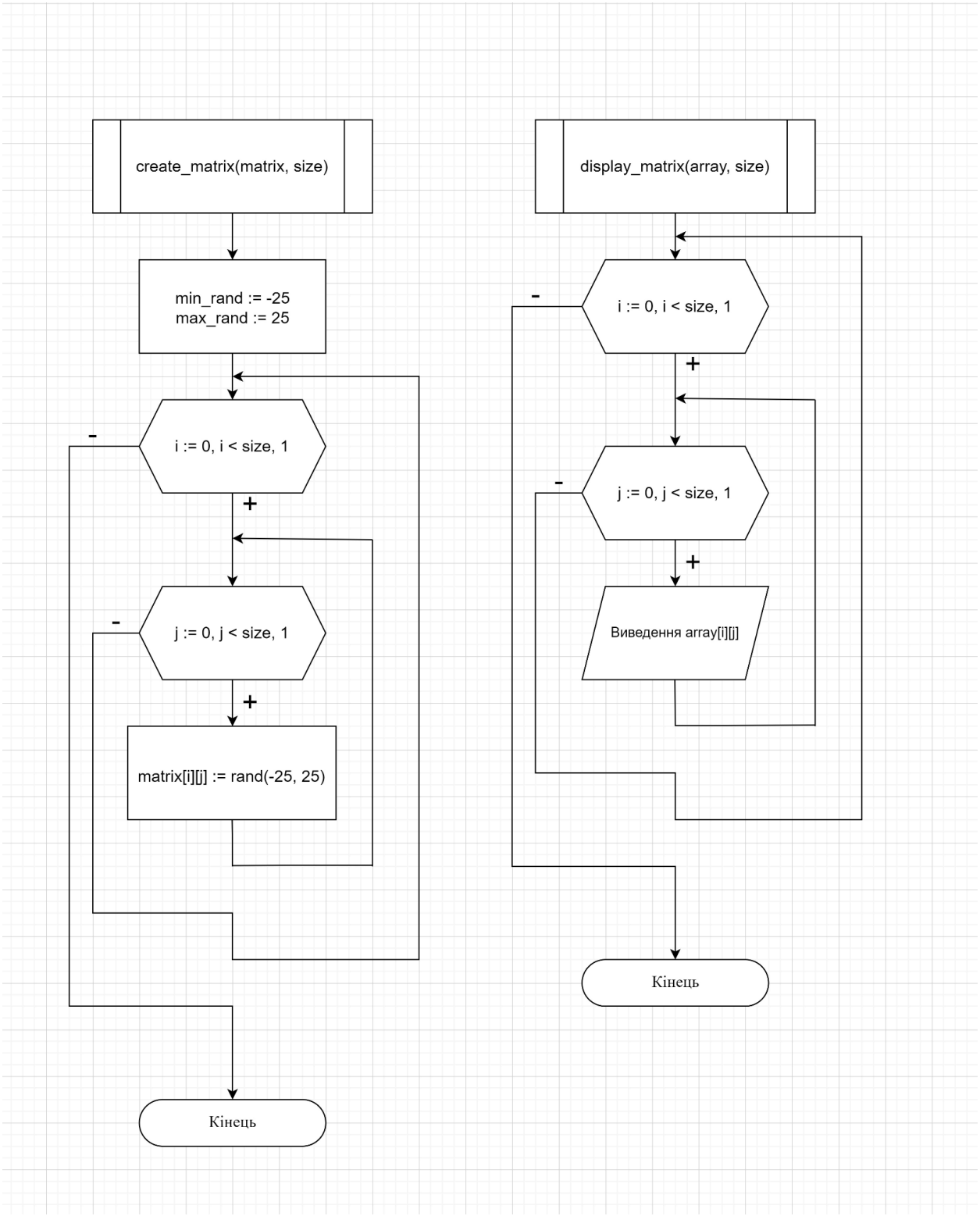
```
підпрограма bubble_sort(arr_diagonal, length)
        i := 0
        поки ( (i < length - 1) && (arr_diagonal[i] != 0) ) повторити
            j := 0
            поки ( (j < length - 1 - i) && (arr_diagonal[i] != 0) ) повторити
                якщо arr_diagonal[j] > arr_diagonal[j+1]
                   то
                        temp_bubble := arr_diagonal[j]
                        arr_diagonal[j] := arr_diagonal[j+1]
                        arr_diagonal[j+1] := temp_bubble
                все якщо
                j := j + 1
            все повторити
            i := i + 1
        все повторити
все підпрограма
```

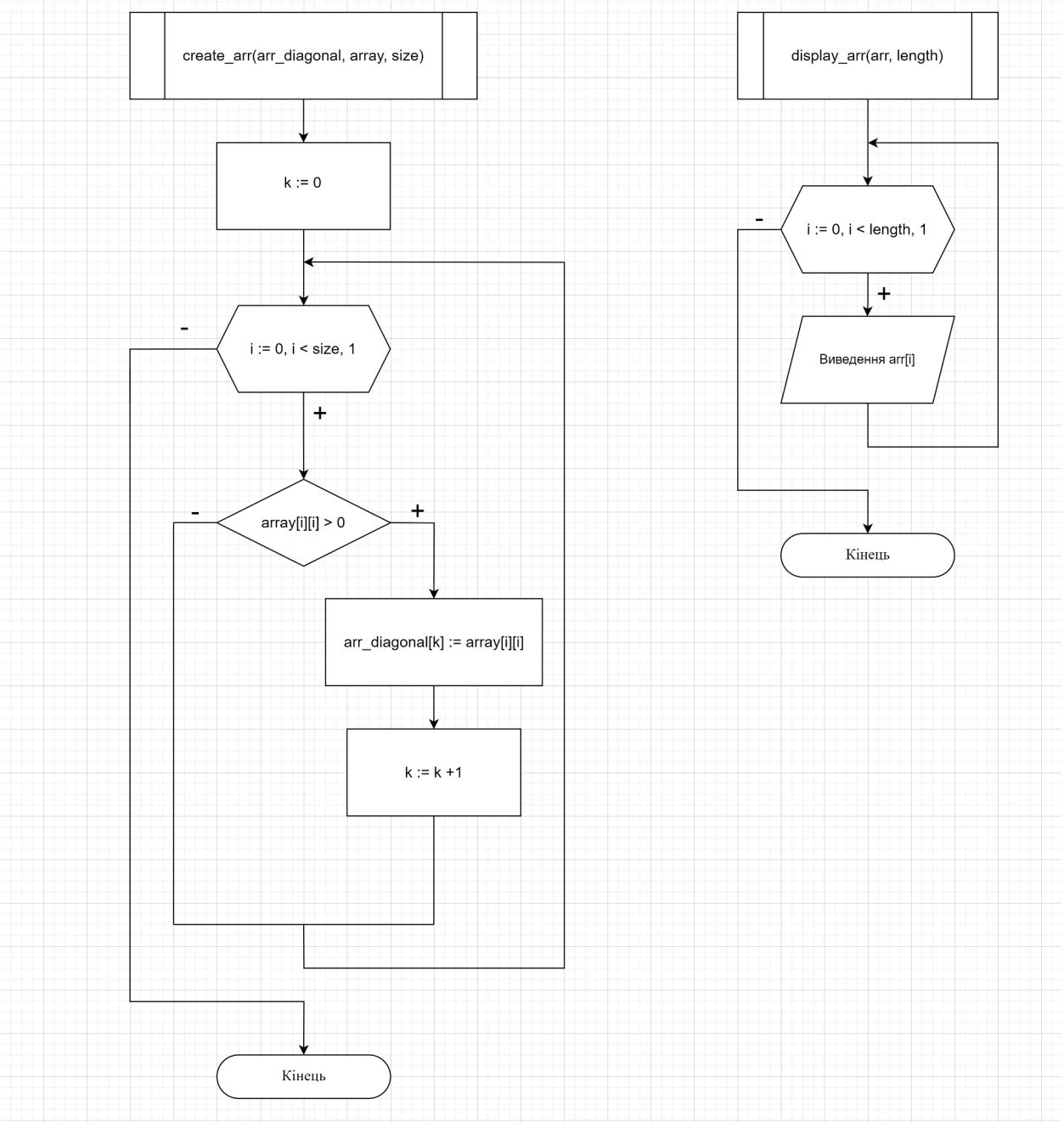
Блок-схема:

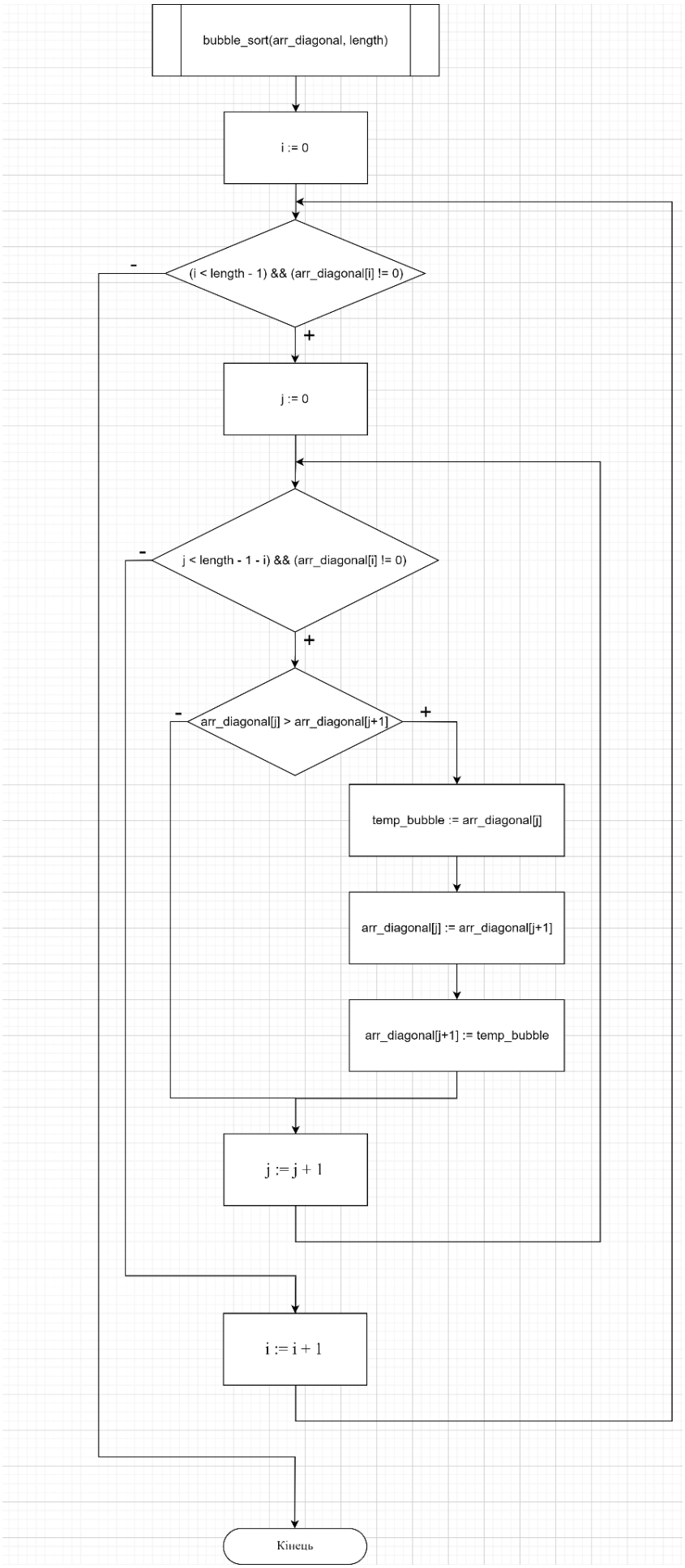
Основна програма:



Підпрограми:




create_arr(arr_diagonal, array, size)
k := 0
i := 0, i < size, 1
-
+
array[i][i] > 0
-
+
arr_diagonal[k] := array[i][i]
k := k +1
Кінець
display_arr(arr, length)
i := 0, i < length, 1
-
+
Виведення arr[i]
Кінець


bubble_sort(arr_diagonal, length)
i := 0
(i < length - 1) && (arr_diagonal[i] != 0)
-
+
j := 0
j < length - 1 - i) && (arr_diagonal[i] != 0)
-
+
arr_diagonal[j] > arr_diagonal[j+1]
-
+
temp_bubble := arr_diagonal[j]
arr_diagonal[j] := arr_diagonal[j+1]
arr_diagonal[j+1] := temp_bubble
j := j + 1
i := i + 1
Кінець

Код програми: (C++)

```
#include <iostream>
#include <iomanip>
#include <ctime>
#include <cmath>
using namespace std;
int** create_matrix(int);
int* create_arr(int**, int);
void display_matrix(int**, int);
void display_arr(int*, int);
int* bubble_search(int*, int);
void delete_matrix(int**, int);
int main()
{
   int n = 6;
   int** matrix;
   int* arr_diagonal = new int[n];
   matrix = create_matrix(n);
   arr_diagonal = create_arr(matrix, n);

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

   cout << "Generated array of diagonals: " << endl;
```

```
    display_arr(arr_diagonal, n);

    arr_diagonal = bubble_search(arr_diagonal, n);
    cout << "\n" << "Bubble sorted array of diagonals: " << endl;
    display_arr(arr_diagonal, n);
    delete_matrix(matrix, n);
    delete[] arr_diagonal;
}
```

```
int** create_matrix(int size)
{
   int min_rand = -25;
   int max_rand = 25;
   srand(time(0));
   int** matrix = new int* [size];
   for (int i = 0; i < size; i++)
   {
      matrix[i] = new int[6];
   }
   for (int i = 0; i < size; i++)
   {
      for (int j = 0; j < size; j++)
      {
         matrix[i][j] = rand() % (max_rand + abs(min_rand) + 1) -
abs(min_rand);
      }
   }
   return matrix;
}
```

```
int* create_arr(int** array, int size)
{
   int* arr_diagonal = new int[size];
   for (int i = 0; i < size; i++)
   {
      for (int j = 0; j < size; j++)
      {
         if (j == i) {
            if (array[i][j] > 0)
            {
               arr_diagonal[i] = array[i][j];
            }
            else
            {
               arr_diagonal[i] = 0;
            }
         }
      }
   }
   return arr_diagonal;
}
```

```
void display_matrix(int** matrix, int size)
{
   for (int i = 0; i < size; i++)
   {
      for (int j = 0; j < size; j++)
      {
         cout << setw(6) << matrix[i][j];
      }
      cout << endl;
   }
}

void display_arr(int* arr, int size)
{
   for (int i = 0; i < size; i++)
   {
      if (arr[i] != 0) {
         cout << setw(6) << arr[i];
      }
   }
   cout << endl;
}
```

```
int* bubble_search(int* arr, int size)
{
   int temp;


   for (int i = 0; i < size - 1; i++)
   {
      for (int j = 0; j < size - 1; j++)
      {
         if (arr[j] > arr[j + 1])
         {
            temp = arr[j + 1];
            arr[j + 1] = arr[j];
            arr[j] = temp;
         }
      }
   }
   return arr;
}
void delete_matrix(int** matrix, int size) {
   for (int i = 0; i < size; i++)
   {
      delete[] matrix[i];
   }
   delete[] matrix;
}
```


Generated matrix:
-1 16 2 -7 22 4
7 1 -23 24 -15 19
-5 4 -17 10 -12 18
-24 -21 -15 -6 10 8
-1 6 -21 10 -13 -8
23 -19 13 -5 13 -6
Generated array of diagonals:
1
Bubble sorted array of diagonals:
1


Generated matrix:
-3 3 -6 -8 3 -1
10 6 -2 -3 -25 -15
9 -8 -19 12 21 -11
-9 18 20 20 -12 -10
-24 17 13 6 22 2
-16 -25 -25 12 4 -9
Generated array of diagonals:
6 20 22
Bubble sorted array of diagonals:
6 20 22


Generated matrix:
-1 -23 -20 -17 -7 -12
-21 -4 1 20 -15 17
1 -16 21 -6 7 -24
0 11 23 14 10 -23
2 -22 -19 -18 22 24
5 4 -4 23 9 17
Generated array of diagonals:
21 14 22 17
Bubble sorted array of diagonals:
14 17 21 22


Generated matrix:
-14 14 -21 0 -10 25
11 -23 12 23 -6 -22
9 -22 24 -22 -4 13
9 1 19 12 -22 -8
-24 0 -25 -7 5 -17
-20 -7 20 -7 19 -22
Generated array of diagonals:
24 12 5
Bubble sorted array of diagonals:
5 12 24

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