Додаток 1

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

«Дослідження лінійних алгоритмів»

Варіант 12

Виконав студент Дойчев Костянтин Миколайович
(шифр, прізвище, ім'я, по батькові)

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

Київ 2021

Лабораторна робота №9

Тема: дослідження алгоритмів обходу масивів

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

Варіант 12

1) Постановка задачі:

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

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

Опис - Задано матрицю дійсних чисел A[m,n]. При обході матриці по стовпчиках знайти в ній останній мінімальний елемент і його місцезнаходження. Обміняти знайдене значення X з елементом першого рядка

Розв'язання

Крок 1. Визначимо основні дії.

Крок 2. Заповнимо матрицю випадковими дійсними числами

Крок 3. Знайдемо мінімальний елемент

Крок 4. Обміняємо знайдений елемент з першим елементом рядка

2) Побудова математичної моделі:

Таблиця імен змінних

Змінна	Тип	Ім’я	Призначення

Таким чином, математичне формулювання задачі зводиться до

3) Псевдокод алгоритму

Крок 1:

Початок
Кінець

Крок 2:

Початок
Кінець

Крок 3:

Початок
Кінець

Крок 4:

Початок
Кінець

4) Блок схема алгоритму

5) Код алгоритму:

main.cpp

```cpp
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <string>
#include "utils.cpp"
#include "matrix-output.cpp"

typedef struct {
    float value;
    int row;
    int column;
} Min;

template<typename T, size_t rows, size_t columns>
void fillMatrix(T (&matrix)[rows][columns], T (*callback)(T));

float fillMatrixCallback(float element);

template<typename T, size_t rows, size_t columns>
void findMinElement(T (&matrix)[rows][columns], Min &minElement);

void defineMinElement(float currentValue, int row, int column, Min &minElement);

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

    const int ROWS = 5;
    const int COLUMNS = 4;
    float matrix[ROWS][COLUMNS];
```

main

main.cpp

```cpp
    fillMatrix( &: matrix, fillMatrixCallback);
    std::cout << "Matrix" << std::endl;
    printMatrix( &: matrix);

    Min minElement;
    minElement.value = matrix[0][0];
    minElement.row = 0;
    minElement.column = 0;

    findMinElement( &: matrix,  &: minElement);
    std::cout << "Min element: " << minElement.value << std::endl;

    swapElements( &: matrix[0][0],  &: matrix[minElement.row][minElement.column]);

    std::cout << "Matrix" << std::endl;
    printMatrix( &: matrix);

    return 0;
}


/**
 *
 * @tparam T
 * @tparam rows - matrix row length
 * @tparam columns - matrix column length
 * @param matrix
 * @param callback - function that will be called at iteration
 */
```

main

main.cpp

```cpp
template<typename T, size_t rows, size_t columns>
void fillMatrix(T (&matrix)[rows][columns], T (*callback)(T)) {
    for (int row = 0; row < rows; row++) {
        for (int column = 0; column < columns; column++) {
            matrix[row][column] = callback(matrix[row][column]);
        }
    }
}

float fillMatrixCallback(float element) {
    const int MIN_VALUE = -1000;
    const int MAX_VALUE = 1000;
    return randomizeValue(MIN_VALUE, MAX_VALUE) / 10.0;
}

/*
 *
 * @tparam T
 * @tparam rows - matrix row length
 * @tparam columns - matrix column length
 * @param matrix
 * @param minElement - min element structure
 */
template<typename T, size_t rows, size_t columns>
void findMinElement(T (&matrix)[rows][columns], Min &minElement) {
    /**
     * @description if direction is > 0, the algorithm will go from top to bottom
     * else if direction is < 0, the algorithm will go from bottom to top
     */
```

findMinElement

main.cpp

```
82  */
83  template<typename T, size_t rows, size_t columns>
84  void findMinElement(T (&matrix)[rows][columns], Min &minElement) {
85      /**
86       * @description if direction is > 0, the algorithm will go from top to bottom
87       * else if direction is < 0, the algorithm will go from bottom to top
88       */
89      int direction = 1;
90      for (int column = 0; column < columns; column++) {
91          if (direction > 0) {
92              for (int row = 0; row < rows; row++) {
93                  float currentElement = matrix[row][column];
94                  defineMinElement(currentElement, row, column,  &: minElement);
95              }
96          } else if (direction < 0) {
97              for (int row = rows - 1; row >= 0; row--) {
98                  float currentElement = matrix[row][column];
99                  defineMinElement(currentElement, row, column,  &: minElement);
100             }
101         }
102         direction *= -1;
103     }
104 }
105
106 void defineMinElement(float currentValue, int row, int column, Min &minElement) {
107     if (currentValue <= minElement.value) {
108         minElement.value = currentValue;
109         minElement.row = row;
110         minElement.column = column;
111     }
```

f findMinElement

main.cpp | utils.cpp

```cpp
// Created by Kostia  on 21.12.2021.
//
#include <cstdlib>
#include "utils.h"

/**
 * This function generates a random number between min and max value
 * @param min - min value
 * @param max - max value
 * @return
 */
int randomizeValue(int min, int max){
    return rand() % (max - min + 1) + min;
}

/**
 * This function swap two values
 * @tparam T
 * @param a - first value
 * @param b - second value
 */
template<typename T>
void swapElements(T &a, T &b) {
    T tmp = a;
    a = b;
    b = tmp;
}
```

randomizeValue

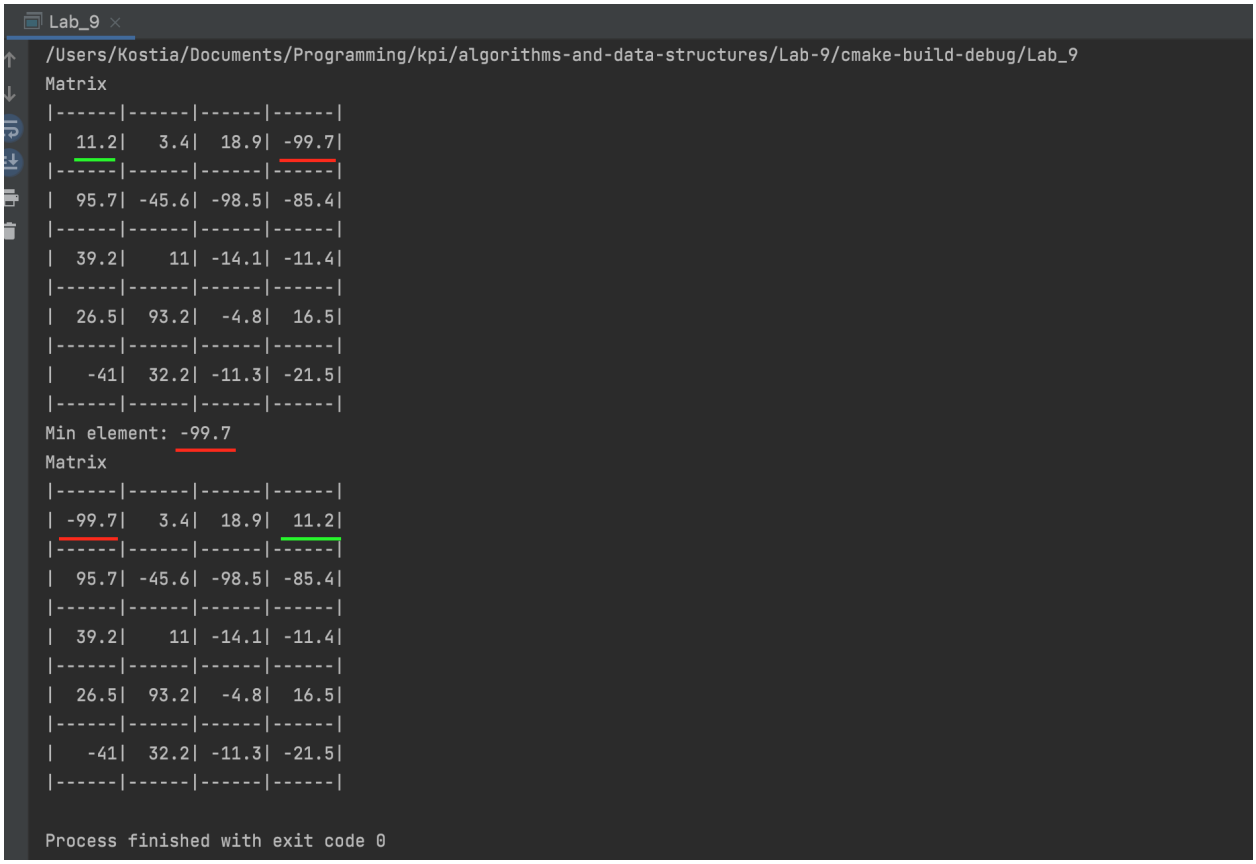
main.cpp | matrix-output.cpp | utils.cpp

```cpp
/**
 *
 * @tparam T
 * @tparam rows - matrix row length
 * @tparam columns - matrix column length
 * @param matrix
 */
template<typename T, size_t rows, size_t columns>
void printMatrix(T (&matrix)[rows][columns]) {
    const int CELL_WIDTH = 6;
    std::cout << generateMatrixRow(CELL_WIDTH, columns, content: "-") << std::endl;
    for (int i = 0; i < rows; i++) {
        std::cout << "|";
        for (int j = 0; j < columns; j++) {
            std::cout << std::setw( n: 6) << matrix[i][j] << "|";
        }
        std::cout << std::endl;
        std::cout << generateMatrixRow(CELL_WIDTH, columns, content: "-") << std::endl;
    }
}
```

main.cpp | matrix-output.cpp | utils.cpp

```cpp
        for (int j = 0; j < columns; j++) {
            std::cout << std::setw( n: 6) << matrix[i][j] << "|";
        }
        std::cout << std::endl;
        std::cout << generateMatrixRow(CELL_WIDTH, columns, content: "-") << std::endl;
    }
}

/**
 *
 * @param width - cell width
 * @param columns - number of columns
 * @param content - character that should be displayed
 * @return divider
 */
std::string generateMatrixRow(int width, int columns, std::string content) {
    std::string res = "|";
    for (int i = 0; i < columns; i++) {
        for (int j = 0; j < width; j++) {
            res += content;
        }
        res += "|";
    }
    return res;
}
```



6) Виновки:

Дослідив алгоритм обходу масивів, набув практичних навичок використання цих алгоритмів під час складання програмних специфікацій. Написав код алгоритму.