**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Информационных Систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

Тема: Работа с текстовыми строками.

|  |  |  |
| --- | --- | --- |
| Студент гр. 3372 |  | Кипень В. А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Необходимо написать программу, которая реализует поставленную задачу:

1) С клавиатуры или с файла (\*) (пользователь сам может выбрать способ ввода) вводится последовательность, содержащая от 1 до 50 слов, в каждом из которых от 1 до 10 строчных латинских букв и цифр. Между соседними словами произвольное количество пробелов. За последним символом стоит точка.

2) Необходимо отредактировать входной текст:

· удалить лишние пробелы;

· удалить лишние знаки препинания

· исправить регистр букв, если это требуется

3) Вывести на экран только те слова последовательности, в которых встречаются одинаковые буквы.

4) Вывести на экран ту же последовательность, заменив во всех словах цифры на буквы латинского алфавита, номера которых в алфавите равны заменяемой цифре.

5) Необходимо найти все подстроки, которую введёт пользователь в имеющейся строке. (Линейный поиск)

**Основные теоретические положения.**

Массив представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение.

Все массивы можно разделить на две группы: одномерные и многомерные. Описание массива в программе отличается от объявления обычной переменной наличием размерности массива, которая задается в квадратных скобках после имени.

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных.

Внутреннее представление величин целого типа – целое число в двоичном коде. При использовании спецификатора signed старший бит числа интерпретируется как знаковый (0 – положительное число, 1 – отрицательное). Для кодирования целых чисел со знаком применяется прямой, обратный и дополнительный коды.

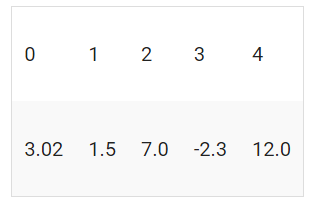


Рисунок 1 – одномерный массив из 5 float-элементов

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Когда элементы отсортированы, их проще найти, производить с ними различные операции. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива.

Все данные (переменные, константы и др.) хранятся в памяти. Память представляет собой непрерывную последовательность ячеек (байтов), каждая из которых имеет свой номер – адрес.

При определении, например, некоторой переменной, она располагается в памяти по определенному адресу и занимает столько ячеек, сколько требует тип этой переменной.

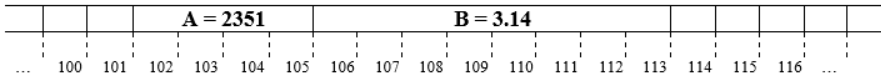


Рисунок 2 – наглядное представление ячеек памяти

Указатель – переменная, служащая для хранения адреса памяти.

**Постановка задачи.**

Необходимо разработать алгоритм и написать программу, которая позволяет:

1) Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка *N* (6,8,10) случайными числами от 1 до  N\*N “спиралью” и “змейкой”. Пользователь должен видеть процесс заполнения квадратичной матрицы.

2)    Получает новую матрицу, из матрицы п. 1, используя арифметику указателей, переставляя ее блоки в соответствии со схемами.

3)    Используя арифметику указателей, сортирует элементы любой сортировкой из списка:

* Shaker sort;
* Comb sort;
* Insert sort;
* Quick sort;

4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

**Выполнение работы.**

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| Меню | |
| При запуске программы перед пользователем появляется окно с меню, где он может выбрать пункт, который требуется выполнить | Меню:    Проверка на ввод символов, которые не входят в диапазон выбора: |

|  |  |
| --- | --- |
| Инициализация массива | |
| При вводе пользователем корректного значения пункта меню (1) изначально происходит инициализация матрицы, длиной 6\*6 элементов и диапазоном значений от 1 до 100 | Как только все данные заданы, на экран выводится массив, заполняющийся в реальном времени (с анимацией) А)спиралью и Б) змейкой:  А) Б) |
| Перестановка блоков | |
| При вводе пользователем корректного значения пункта меню (2) происходит перестановка блоков матрицы (четыре вариации) и вывод получившегося массива | A)  B)  C)  D) |
| Сортировка матрицы | |
| При вводе пользователем корректного значения пункта меню (3) происходит сортировка матрицы методом “insert sort”, и вывод получившегося массива |  |
| Арифметические операции с элементами матрицы | |
| При вводе пользователем корректного значения пункта меню (4) применяется операция (ко всем элементам матрицы) сложения/вычитания/умножения/деления (на выбор) на введённое пользователем число и вывод получившегося массива |  |

**Выводы.**

Была создана программа, в которой реализован следующий функционал:

1) Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка *N* (6,8,10) случайными числами от 1 до  N\*N “спиралью” и “змейкой”. Пользователь должен видеть процесс заполнения квадратичной матрицы.

2)    Получает новую матрицу, из матрицы п. 1, используя арифметику указателей, переставляя ее блоки в соответствии со схемами.

3)    Используя арифметику указателей, сортирует элементы любой сортировкой из списка:

* Shaker sort;
* Comb sort;
* Insert sort;
* Quick sort;

4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

Приложение А

рабочий код

#include <iostream>

#include "windows.h"

using namespace std;

int getRandomValueFromRange(int leftLimit, int rightLimit) {

    return leftLimit + rand() % (rightLimit - leftLimit + 1);

}

void matrixBlockReplacement(int\* blocks[], int\* blocksNew[], const int N) {

    const int distance = (N - 4) / 2;

    for (int i = 0; i < 4; i++) {

        for (int \*next = blocks[i], \*end = blocks[i] + N / 2 - 1, cnt = 1,

        \*next2 = blocksNew[i]; cnt < (N / 2) \* (N / 2) - distance; next++, next2++ ) {

            if (next == end) {

                blocks[i] += N;

                blocksNew[i] += N;

                \*next2 = \*next;

                next = blocks[i];

                next2 = blocksNew[i];

                end = blocks[i] + N / 2 - 1;

            }

            cnt++;

            if (cnt >= (N / 2) \* (N / 2) - distance) break;

            \*next2 = \*next;

        }

    }

}

void matrixPrint(int \*arr, const int N) {

    for (int \*next = arr, \*prev = arr, \*end = arr + N \* N - 1; next <= end; next++) {

        if (next - N == prev) {

            cout << "\n";

            prev = next;

        }

        printf("%03d ", \*next);

    }

    cout << "\n\n";

}

void matrixFill(int \*arr, const int N, const bool isRandom = false, int number = 0) {

    if (isRandom) {

        for (int \*next = arr, \*prev = arr, \*end = arr + N \* N - 1; next <= end; next++) {

            \*next = getRandomValueFromRange(1, 100);

        }

    }

    else {

        for (int \*next = arr, \*prev = arr, \*end = arr + N \* N - 1; next <= end; next++) {

            \*next = number;

        }

    }

}

void matrixOperation(int \*arr, const int N, const int operationType, int number = 0) {

    switch (operationType)

    {

        case 0: {

            for (int \*next = arr, \*prev = arr, \*end = arr + N \* N - 1; next <= end; next++) {

                \*next += number;

            }

            break;

        }

        case 1: {

            for (int \*next = arr, \*prev = arr, \*end = arr + N \* N - 1; next <= end; next++) {

                \*next -= number;

            }

            break;

        }

        case 2: {

            for (int \*next = arr, \*prev = arr, \*end = arr + N \* N - 1; next <= end; next++) {

                \*next \*= number;

            }

            break;

        }

        case 3: {

            for (int \*next = arr, \*prev = arr, \*end = arr + N \* N - 1; next <= end; next++) {

                \*next /= number;

            }

            break;

        }

    }

}

void arrayPrint(int \*arr, const int N) {

    for (int \*next = arr, \*prev = arr, \*end = arr + N - 1; next <= end; next++) {

        printf("%03d ", \*next);

    }

    cout << "\n\n";

}

void bubbleSort(int \*arr, const int N) {

    for (int i = 0; i < N - 1; i++) {

        for (int j = 0; j < N - i - 1; j++) {

            int swap = \*(arr + j);

            if (\*(arr + j) > \*(arr + j + 1)) {

                \*(arr + j) = \*(arr + j + 1);

               \*(arr + j + 1) = swap;

            }

        }

    }

}

void shakerSort(int \*arr, int n) {

    for (int \*left = arr, \*right = arr + n - 1; left < right;) {

        for (int \*i = left; i < right; i++) {

            int swap = \*i;

            if (\*i > \*(i + 1)) {

                \*i = \*(i + 1);

                \*(i + 1) = swap;

            }

        }

        right--;

        for (int \*i = right; i > left; i--) {

            int swap = \*(i - 1);

            if (\*(i - 1) > \*i) {

                \*(i - 1) = \*i;

                \*i = swap;

            }

        }

        left++;

    }

}

void combSort(int \*arr, int N) {

    float k = 1.247;

    float S = N - 1;

    while (S >= 1) {

        for (int i = 0; i + S < N; i++) {

            int swap = \*(arr + i);

            if (\*(arr + i) > \*(arr + i + (int) S)) {

                \*(arr + i) = \*(arr + i + (int) S);

                \*(arr + i + (int) S) = swap;

            }

        }

        S /= k;

    }

}

void insertSort(int \*arr, const int N) {

    int key;

    for (int \*i = &arr[1], \*j = 0, \*end = arr + N \* N - 1; i <= end; i++) {

        key = \*i;

        j = i - 1;

        while (j >= arr && \*j > key) {

            \*(j + 1) = \*j;

            j = j - 1;

        }

        \*(j + 1) = key;

    }

}

int main() {

    setlocale(LC\_ALL, "Russian");

    cout << "Navigation:" << "\n"

    << "1) Fill a quadratic integer matrix NxN with random numbers" << "\n"

    << "2) Obtains a new matrix from the matrix of (1), rearranging its blocks" << "\n"

    << "3) Using pointer arithmetic, sorts elements (insert sort algorithm)" << "\n"

    << "4) Decrements, increases, multiplies, or divides all elements of a matrix by <number>" << "\n"

    << "5) IDZ #9" << "\n";

    const int N = 6;

    int arr[N \* N] = {};

    while(true) {

        cin.clear(); *// Clearing the input stream from possible errors*

        cin.sync();

        short int workPoint;

        cout << "Select point of work (number 1 to 5): ";

        cin >> workPoint;

        switch (workPoint)

        {

            case 1: {

                const int elementSize = 4;

                int cnt = 1;

                int k = 1;

                HANDLE hStdout;

                COORD destCoord;

                hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

                destCoord.X = -elementSize;

                matrixFill(arr, N); *// clear the matrix (for better visibility of the algorithm)*

                while (cnt <= N \* N) {

                    int \*upLeftCorner = &arr[(k - 1) \* N + (k - 1)];

                    int \*upRightCorner = &arr[k \* N - 1 - (k - 1)];

                    int \*downLeftCorner = &arr[N \* N - (k \* N) + (k - 1)];

                    int \*downRightCorner = &arr[N \* N - ((k - 1) \* N) - 1 - (k - 1)];

                    for (int \*next = upLeftCorner, \*end = upRightCorner; next <= end; next++) {

                        \*next = getRandomValueFromRange(1, N \* N);

                        int posY = (next - arr) / N;

                        destCoord.X += elementSize;

                        destCoord.Y = posY;

                        SetConsoleCursorPosition(hStdout, destCoord);

                        printf("%03d\r", \*next);

                        cout.flush();

                        Sleep(200);

                        cnt++;

                    }

                    for (int \*next = upRightCorner + N, \*end = downRightCorner; next <= end; next += N) {

                        \*next = getRandomValueFromRange(1, N \* N);

                        int posY = (next - arr) / N;

                        destCoord.Y = posY;

                        SetConsoleCursorPosition(hStdout, destCoord);

                        printf("%03d\r", \*next);

                        cout.flush();

                        Sleep(200);

                        cnt++;

                    }

                    for (int \*next = downRightCorner - 1, \*end = downLeftCorner; next >= end; next--) {

                        \*next = getRandomValueFromRange(1, N \* N);

                        int posY = (next - arr) / N;

                        destCoord.X -= elementSize;

                        destCoord.Y = posY;

                        SetConsoleCursorPosition(hStdout, destCoord);

                        printf("%03d\r", \*next);

                        cout.flush();

                        Sleep(200);

                        cnt++;

                    }

                    for (int \*next = downLeftCorner - N, \*end = upLeftCorner + N; next >= end; next -= N) {

                        \*next = getRandomValueFromRange(1, N \* N);

                        int posY = (next - arr) / N;

                        destCoord.Y = posY;

                        SetConsoleCursorPosition(hStdout, destCoord);

                        printf("%03d\r", \*next);

                        cout.flush();

                        Sleep(200);

                        cnt++;

                    }

                    k++;

                }

                destCoord.X = -elementSize;

                destCoord.Y += N \* N;

                cout << "\n";

                matrixFill(arr, N); *// clear the matrix (for better visibility of the algorithm)*

                cnt = 1;

                k = 0;

                int \*upBorder = &arr[k];

                int \*downBorder = &arr[N \* N - N + k];

                while (cnt <= N \* N) {

                    destCoord.X += elementSize;

                    for (int \*next = upBorder + k, \*end = downBorder + k; next <= end; next += N) {

                        \*next = getRandomValueFromRange(1, N \* N);

                        int posY = (next - arr) / N;

                        destCoord.Y = posY;

                        SetConsoleCursorPosition(hStdout, destCoord);

                        printf("%03d\r", \*next);

                        cout.flush();

                        Sleep(200);

                        cnt++;

                    }

                    k++;

                    destCoord.X += 4;

                    for (int \*next = downBorder + k, \*end = upBorder + k; next >= end; next -= N) {

                        \*next = getRandomValueFromRange(1, N \* N);

                        int posY = (next - arr) / N;

                        destCoord.Y = posY;

                        SetConsoleCursorPosition(hStdout, destCoord);

                        printf("%03d\r", \*next);

                        cout.flush();

                        Sleep(200);

                        cnt++;

                    }

                    k++;

                }

                for (int i = 0; i < N; i++) cout << "\n";

                break;

            }

            case 2: {

                int arrNew[N \* N] = {};

                cout << "\n" << "2.A)" << "\n";

                int\* blocksA[4] = {&arr[0], &arr[N / 2], &arr[N / 2 + N \* (N / 2)], &arr[(N \* N) / 2]};

                int\* blocksNewA[4] = {&arrNew[N / 2], &arrNew[N / 2 + N \* (N / 2)], &arrNew[(N \* N) / 2], &arrNew[0]};

                matrixBlockReplacement(blocksA, blocksNewA, N);

                matrixPrint(arrNew, N);

                cout << "\n" << "2.B)" << "\n";

                blocksA[0] = &arr[0];

                blocksA[1] = &arr[N / 2 + N \* (N / 2)];

                blocksA[2] = &arr[(N \* N) / 2];

                blocksA[3] = &arr[N / 2];

                blocksNewA[0] = &arrNew[N / 2 + N \* (N / 2)];

                blocksNewA[1] = &arrNew[0];

                blocksNewA[2] = &arrNew[N / 2];

                blocksNewA[3] = &arrNew[(N \* N) / 2];

                matrixBlockReplacement(blocksA, blocksNewA, N);

                matrixPrint(arrNew, N);

                cout << "\n" << "2.C)" << "\n";

                blocksA[0] = &arr[0];

                blocksA[1] = &arr[(N \* N) / 2];

                blocksA[2] = &arr[N / 2];

                blocksA[3] = &arr[N / 2 + N \* (N / 2)];

                blocksNewA[0] = &arrNew[(N \* N) / 2];

                blocksNewA[1] = &arrNew[0];

                blocksNewA[2] = &arrNew[N / 2 + N \* (N / 2)];

                blocksNewA[3] = &arrNew[N / 2];

                matrixBlockReplacement(blocksA, blocksNewA, N);

                matrixPrint(arrNew, N);

                cout << "\n" << "2.D)" << "\n";

                blocksA[0] = &arr[0];

                blocksA[1] = &arr[N / 2];

                blocksA[2] = &arr[(N \* N) / 2];

                blocksA[3] = &arr[N / 2 + N \* (N / 2)];

                blocksNewA[0] = &arrNew[N / 2];

                blocksNewA[1] = &arrNew[0];

                blocksNewA[2] = &arrNew[N / 2 + N \* (N / 2)];

                blocksNewA[3] = &arrNew[(N \* N) / 2];

                matrixBlockReplacement(blocksA, blocksNewA, N);

                matrixPrint(arrNew, N);

                break;

            }

            case 3: {

                insertSort(arr, N);

                matrixPrint(arr, N);

                break;

            }

            case 4: {

                int operationType;

                cout << "\n" << "Enter an operation type (0:+, 1:-, 2:\*. 3:/): ";

                cin >> operationType;

                if (!cin.good() || operationType < 0 || operationType > 3) {

                    cout << "\n" << "You entered an incorrect value";

                    break;

                }

                int a;

                cout << "\n" << "Enter an integer: ";

                cin >> a;

                if (!cin.good()) {

                    cout << "\n" << "You entered an incorrect value";

                    break;

                }

                matrixOperation(arr, N, operationType, a);

                matrixPrint(arr, N);

                break;

            }

            case 5: {

                matrixFill(arr, N, true);

                matrixPrint(arr, N);

                int arrNew[N \* N] = {};

                int \*starty = &arr[0];

                int \*endy = &arr[N - 1];

                int \*startNew = &arrNew[0];

                for (int \*next = starty, \*end = endy,

                \*next2 = startNew; endy < arr + N \* N; next++, next2 += N) {

                    if (next == end ) {

                        starty += N;

                        startNew += 1;

                        endy = starty + N - 1;

                        \*next2 = \*next;

                        next = starty;

                        next2 = startNew;

                        end = endy;

                    }

                    if (end > arr + N \* N) break;

                    \*next2 = \*next;

                }

                matrixPrint(arrNew, N);

                break;

            }

            default: {

                cout << "\n" << "You did not enter a number in the range from 1 to 5";

                break;

            }

        }

        cin.clear(); *// Clearing the input stream from possible errors*

        cin.sync();

        char clearMatrix;

        cout << "\n" << "Clear the result matrix? (Y/N) ";

        cin >> clearMatrix;

        if (clearMatrix == 'Y' || clearMatrix == 'y') {

            matrixFill(arr, N);

        }

        cin.clear();

        cin.sync();

        char stopFlag;

        cout << "\n" << "Continue the program? (Y/N) ";

        cin >> stopFlag;

        if (stopFlag != 'Y' && stopFlag != 'y') {

            break;

        }

    }

    return 0;

}