**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по курсовой работе**

**по дисциплине «Программирование»**

Тема: КУРСОВАЯ РАБОТА

|  |  |  |
| --- | --- | --- |
| Студентка гр. 4373 | Репков А.А |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2024

**Цель работы.**

Объединение всех предыдущих практических работ по программированию.

**Постановка задачи.**

Необходимо объединить все 4 лабораторные работы в единый проект. Нужно добавить инфраструктуру переключения между заданиями (интерактивное меню).

**Выполнение работы.**

**#include <iostream>**

**#include <chrono>**

**#include <cstdlib> // Для функции rand() и srand()**

**#include <ctime>**

**#include <vector>**

**#include <iomanip> // Для setw**

**#include <Windows.h> // Для работы с Windows API**

**#include <thread> // Для использования this\_thread::sleep\_for**

**#include <random>**

**#include <sstream>**

**#include <string>**

**#include <cctype>**

**#include <algorithm>**

**#include <conio.h>**

**using namespace std;**

**using namespace chrono;**

**void clearConsole() {**

**system("cls");**

**}**

**//Функции 2 лабы**

**void swapElements(int arr[], int index1, int index2) {**

**swap(arr[index1], arr[index2]);**

**}**

**bool linearSearch(int arr[], int n, int value) {**

**for (int i = 0; i < n; ++i) {**

**if (arr[i] == value)**

**return true;**

**}**

**return false;**

**}**

**bool binarySearch(int arr[], int n, int value) {**

**int left = 0;**

**int right = n - 1;**

**while (left <= right) {**

**int mid = left + (right - left) / 2;**

**if (arr[mid] == value)**

**return true;**

**else if (arr[mid] < value)**

**left = mid + 1;**

**else**

**right = mid - 1;**

**}**

**return false;**

**}**

**int countBiggestThan(int arr[], int n, int b) {**

**int count = 0;**

**for (int i = 0; i < n; ++i) {**

**if (arr[i] > b) {**

**count++;**

**}**

**}**

**return count;**

**}**

**void bubbleSort(int arr[], int n) {**

**setlocale(LC\_ALL, "RU");**

**for (int i = 0; i < n - 1; ++i) {**

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

**if (arr[j] > arr[j + 1]) {**

**swap(arr[j], arr[j + 1]);**

**}**

**}**

**}**

**cout << "\n";**

**}**

**void findMinMax(int arr[], int n, int& min, int& max) {**

**min = arr[0];**

**max = arr[0];**

**for (int i = 1; i < n; ++i) {**

**if (arr[i] < min) {**

**min = arr[i];**

**}**

**if (arr[i] > max) {**

**max = arr[i];**

**}**

**}**

**cout << "\n";**

**}**

**void findIndices(int arr[], int n, int value, vector<int>& indices) {**

**indices.clear(); // Очистка вектора перед использованием**

**for (int i = 0; i < n; ++i) {**

**if (arr[i] == value) {**

**indices.push\_back(i); // Добавление индекса в вектор**

**}**

**}**

**cout << "\n";**

**}**

**int countLessThan(int arr[], int n, int a) {**

**int count = 0;**

**for (int i = 0; i < n; ++i) {**

**if (arr[i] < a) {**

**count++;**

**}**

**}**

**return count;**

**}**

**// Функции 3 лабы**

**void subtractAndDisplay(int\*\* matrix1, int\*\* matrix2, int N) {**

**cout << "Результат вычитания второй матрицы из первой:" << endl;**

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

**for (int j = 0; j < N; ++j) {**

**int result = matrix2[i][j] - matrix1[i][j];**

**cout << setw(4) << result;**

**cout.flush();**

**std::this\_thread::sleep\_for(std::chrono::milliseconds(200));**

**}**

**cout << endl;**

**}**

**}**

**void fillMatrixRandom(int\*\* matrix, int N) {**

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

**for (int j = 0; j < N; ++j) {**

**matrix[i][j] = rand() % (N \* N) + 1;**

**}**

**}**

**}**

**void fillMatrixSequential(int\*\* matrix, int N) {**

**int value = 1;**

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

**for (int j = 0; j < N; ++j) {**

**matrix[i][j] = value++;**

**}**

**}**

**}**

**void setCursorPosition(HANDLE hStdout, int x, int y) {**

**COORD coord;**

**coord.X = x;**

**coord.Y = y;**

**SetConsoleCursorPosition(hStdout, coord);**

**}**

**void fillAndPrintMatrix(int\* matrix, int N) {**

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

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**}**

**void fillAndPrintVerticalSnake(HANDLE hStdout, int N) {**

**random\_device rd;**

**mt19937 gen(rd());**

**uniform\_int\_distribution<> dis(1, N \* N);**

**int value = 1;**

**int startRow = 7;**

**setCursorPosition(hStdout, 0, startRow);**

**cout << "Задание 1" << endl;**

**for (int j = 0; j < N; ++j) {**

**if (j % 2 == 0) { // Четные столбцы**

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

**setCursorPosition(hStdout, j \* 5, startRow + i + 2); // Позиция для вывода с учётом отступа**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(200));**

**}**

**}**

**else { // Нечетные столбцы**

**for (int i = N - 1; i >= 0; --i) {**

**setCursorPosition(hStdout, j \* 5, startRow + i + 2); // Позиция для вывода с учётом отступа**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(200));**

**}**

**}**

**}**

**}**

**void fillAndPrintSpiral(HANDLE hStdout, int N) {**

**random\_device rd;**

**mt19937 gen(rd());**

**uniform\_int\_distribution<> dis(1, N \* N);**

**int value = 1;**

**int left = 0, right = N - 1, top = 0, bottom = N - 1;**

**while (left <= right && top <= bottom) {**

**// Заполняем верхнюю строку**

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

**setCursorPosition(hStdout, i \* 5, top + N + 11); // Позиция для вывода с отступом вниз**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(200));**

**}**

**top++;**

**// Заполняем правый столбец**

**for (int i = top; i <= bottom; ++i) {**

**setCursorPosition(hStdout, right \* 5, i + N + 11); // Позиция для вывода с отступом вниз**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(200));**

**}**

**right--;**

**// Заполняем нижнюю строку**

**if (top <= bottom) {**

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

**setCursorPosition(hStdout, i \* 5, bottom + N + 11); // Позиция для вывода с отступом вниз**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(200));**

**}**

**bottom--;**

**}**

**// Заполняем левый столбец**

**if (left <= right) {**

**for (int i = bottom; i >= top; --i) {**

**setCursorPosition(hStdout, left \* 5, i + N + 11); // Позиция для вывода с отступом вниз**

**cout << setw(4) << dis(gen);**

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(200));**

**}**

**left++;**

**}**

**}**

**}**

**void printQuadrantsA(int\* matrix, int N) {**

**int halfN = N / 2;**

**// Создаем массив для хранения квадрантов**

**int\* quadrants[4];**

**// Выделяем память для каждого квадранта**

**for (int i = 0; i < 4; ++i) {**

**quadrants[i] = new int[halfN \* halfN];**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 4-1-2-3 (A):" << endl;**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < 4; ++i) {**

**delete[] quadrants[i];**

**}**

**}**

**void printQuadrantsB(int\* matrix, int N) {**

**int halfN = N / 2;**

**// Создаем массив для хранения квадрантов**

**int\* quadrants[4];**

**// Выделяем память для каждого квадранта**

**for (int i = 0; i < 4; ++i) {**

**quadrants[i] = new int[halfN \* halfN];**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 3-4-1-2 (B):" << endl;**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < 4; ++i) {**

**delete[] quadrants[i];**

**}**

**}**

**void printQuadrantsC(int\* matrix, int N) {**

**int halfN = N / 2;**

**int\* quadrants[4];**

**// Выделяем память для каждого квадранта**

**for (int i = 0; i < 4; ++i) {**

**quadrants[i] = new int[halfN \* halfN];**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 4-3-2-1 (C):" << endl;**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < 4; ++i) {**

**delete[] quadrants[i];**

**}**

**}**

**void printQuadrantsD(int\* matrix, int N) {**

**int halfN = N / 2;**

**// Создаем массив для хранения квадрантов**

**int\* quadrants[4];**

**// Выделяем память для каждого квадранта**

**for (int i = 0; i < 4; ++i) {**

**quadrants[i] = new int[halfN \* halfN];**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**quadrants[0][i \* halfN + j] = \*(matrix + i \* N + j);**

**quadrants[1][i \* halfN + j] = \*(matrix + i \* N + (j + halfN));**

**quadrants[2][i \* halfN + j] = \*(matrix + (i + halfN) \* N + j);**

**quadrants[3][i \* halfN + j] = \*(matrix + (i + halfN) \* N + (j + halfN));**

**}**

**}**

**cout << "Матрица в порядке 2-1-4-3 (D):" << endl;**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[1][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[0][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < halfN; ++i) {**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[3][i \* halfN + j];**

**}**

**for (int j = 0; j < halfN; ++j) {**

**cout << setw(4) << quadrants[2][i \* halfN + j];**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < 4; ++i) {**

**delete[] quadrants[i];**

**}**

**}**

**void shakerSort(int\* array, int size) {**

**bool swapped = true;**

**int start = 0;**

**int end = size - 1;**

**while (swapped) {**

**swapped = false;**

**for (int i = start; i < end; ++i) {**

**if (\*(array + i) > \*(array + i + 1)) {**

**swap(\*(array + i), \*(array + i + 1));**

**swapped = true;**

**}**

**}**

**if (!swapped) break;**

**--end;**

**swapped = false;**

**for (int i = end - 1; i >= start; --i) {**

**if (\*(array + i) > \*(array + i + 1)) {**

**swap(\*(array + i), \*(array + i + 1));**

**swapped = true;**

**}**

**}**

**++start;**

**}**

**}**

**void shakerSortMatrix(int\* matrix, int N) {**

**shakerSort(matrix, N \* N);**

**cout << "Отсортированная матрица:" << endl;**

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

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**}**

**void manipulateMatrix(int\* matrix, int N, char operation, int number) {**

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

**switch (operation) {**

**case '+':**

**\*(matrix + i) += number;**

**break;**

**case '-':**

**\*(matrix + i) -= number;**

**break;**

**case '\*':**

**\*(matrix + i) \*= number;**

**break;**

**case '/':**

**if (number != 0)**

**\*(matrix + i) /= number;**

**else**

**cout << "Деление на ноль невозможно!" << endl;**

**return;**

**default:**

**cout << "Неверная операция!" << endl;**

**return;**

**}**

**}**

**cout << "Измененная матрица:" << endl;**

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

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**}**

**//Функции 4 лабы**

**string cleanString(const string& input) {**

**string result;**

**bool lastWasSpace = false;**

**bool lastWasPunctuation = false;**

**for (size\_t i = 0; i < input.size(); ++i) {**

**char ch = input[i];**

**if (ch == ' ') {**

**if (!lastWasSpace) {**

**result += ch;**

**lastWasSpace = true;**

**}**

**}**

**else if (ch == '.' || ch == ',' || ch == ';' || ch == ':') {**

**if (!lastWasPunctuation) {**

**result += ch;**

**lastWasPunctuation = true;**

**}**

**}**

**else {**

**result += ch;**

**lastWasSpace = false;**

**lastWasPunctuation = false;**

**}**

**}**

**if (!result.empty() && result.back() == '.') {**

**result.pop\_back();**

**}**

**return result;**

**}**

**string fixCapitalization(const string& input) {**

**string result = input;**

**bool newWord = true;**

**for (size\_t i = 0; i < result.size(); ++i) {**

**if (isalpha(result[i])) {**

**if (newWord) {**

**result[i] = toupper(result[i]);**

**newWord = false;**

**}**

**else {**

**result[i] = tolower(result[i]);**

**}**

**}**

**else {**

**newWord = true;**

**}**

**}**

**return result;**

**}**

**bool containsDigit(const string& word) {**

**for (char c : word) {**

**if (isdigit(c)) {**

**return true;**

**}**

**}**

**return false;**

**}**

**vector<size\_t> findSubstring(const string& text, const string& substring) {**

**vector<size\_t> positions;**

**size\_t pos = text.find(substring, 0);**

**while (pos != string::npos) {**

**positions.push\_back(pos);**

**pos = text.find(substring, pos + 1);**

**}**

**return positions;**

**}**

**//лабы**

**void lab1() {**

**setlocale(0, "");**

**cout << "Задание 1\n";**

**cout << "\nРазмер int: " << sizeof(int) << " байт\n";**

**cout << "Размер short int: " << sizeof(short int) << " байт\n";**

**cout << "Размер long int: " << sizeof(long int) << " байт\n";**

**cout << "Размер float: " << sizeof(float) << " байт\n";**

**cout << "Размер double: " << sizeof(double) << " байт\n";**

**cout << "Размер long double: " << sizeof(long double) << " байт\n";**

**cout << "Размер char: " << sizeof(char) << " байт\n";**

**cout << "Размер bool: " << sizeof(bool) << " байт\n";**

**int num;**

**unsigned int mask = 1 << (sizeof(int) \* 8) - 1;**

**cout << "\nЗадание 2\n";**

**cout << "Введите int num: ";**

**cin >> num;**

**cout << "Представление в памяти int: ";**

**for (int i = 1; i <= sizeof(int) \* 8; i++, mask >>= 1) {**

**if (mask & num) {**

**cout << "1";**

**}**

**else {**

**cout << "0";**

**}**

**if (i == 1 || i % 8 == 0) {**

**cout << " ";**

**}**

**}**

**cout << "\nИДЗ:";**

**int number;**

**mask = 1 << ((sizeof(int) \* 8) - 1);**

**cin >> number;**

**for (int i = 1; i <= sizeof(int) \* 8; i++, mask >>= 1) {**

**if (mask & number) {**

**cout << "1";**

**}**

**else {**

**cout << "0";**

**}**

**if (i == 1 || i % 8 == 0) {**

**cout << " ";**

**}**

**}**

**if (number >= 0) {**

**for (int i = 1; i <= sizeof(int) \* 8; ++i) {**

**if (i % 2 != 0) {**

**number |= (1 << i); // меняем на 1**

**}**

**mask >>= 1;**

**}**

**}**

**else {**

**for (int i = 1; i <= sizeof(int) \* 8; ++i) {**

**if (i % 2 == 0) {**

**number &= (~(1 << i)); // меняем на 0**

**}**

**mask >>= 1;**

**}**

**}**

**cout << endl;**

**mask = 1 << ((sizeof(int) \* 8) - 1);**

**for (int i = 1; i <= sizeof(int) \* 8; i++, mask >>= 1) {**

**if (mask & number) {**

**cout << "1";**

**}**

**else {**

**cout << "0";**

**}**

**if (i == 1 || i % 8 == 0) {**

**cout << " ";**

**}**

**}**

**cout << endl;**

**cout << number << endl;**

**union**

**{**

**int aI;**

**float bF;**

**};**

**cout << "\n" << "\n";**

**cout << "Задание 3";**

**cout << "\n";**

**cout << "Введите число типа float: ";**

**cin >> bF;**

**mask = 1 << (sizeof(float) \* 8) - 1;**

**cout << "\n";**

**cout << "Представление в памяти float: ";**

**for (int i = 1; i <= sizeof(float) \* 8; ++i, mask >>= 1)**

**{**

**if (aI & mask)**

**{**

**cout << '1';**

**}**

**else**

**{**

**cout << '0';**

**}**

**if (i == 1 || i == 9)**

**{**

**cout << " ";**

**}**

**}**

**union**

**{**

**double cD;**

**unsigned int mas[2];**

**};**

**cout << "\n" << "\n";**

**cout << "Задание 4";**

**cout << "\n";**

**cout << "Введите число типа double: ";**

**cin >> cD;**

**mask = 1 << (sizeof(int) \* 8) - 1;**

**cout << "\n";**

**cout << "Представление в памяти double: ";**

**for (int n = 1; n >= 0; --n)**

**{**

**for (int i = 1; i <= sizeof(int) \* 8; i++, mask >>= 1)**

**{**

**if (mas[n] & mask)**

**cout << '1';**

**else**

**cout << '0';**

**}**

**mask = 1 << (sizeof(int) \* 8 - 1);**

**}**

**}**

**void lab2() {**

**setlocale(LC\_ALL, "RU");**

**const int N = 100;**

**int arr[N];**

**srand(static\_cast<unsigned int>(time(0)));**

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

**arr[i] = rand() % 199 - 99;**

**}**

**cout << "Элементы массива до сортировки:" << endl;**

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

**cout << arr[i] << " ";**

**}**

**cout << endl;**

**cout << "\n";**

**int value\_to\_decrease;**

**cout << "Введите значение для уменьшения четных элементов: ";**

**cin >> value\_to\_decrease;**

**int end\_val = 9;**

**int start\_val = 1;**

**int random\_value = (rand() % (end\_val - start\_val + 1) + start\_val);**

**int count\_even = 0;**

**int count\_odd = 0;**

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

**if (i % 2 == 0) {**

**arr[i] -= value\_to\_decrease;**

**if (arr[i] % 2 == 0) {**

**count\_even += 1;**

**}**

**}**

**else if (i % 2 != 0 && arr[i] % 2 != 0) {**

**count\_odd += 1;**

**}**

**}**

**cout << "\n";**

**cout << "Массив после вычитания значения " << value\_to\_decrease << ":" << endl;**

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

**cout << arr[i] << " ";**

**}**

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

**if (i % 2 == 0) {**

**arr[i] \*= random\_value;**

**}**

**}**

**cout << "\n" << "\n";**

**cout << "Массив после умножения четных элементов на значение " << random\_value << ":" << endl;**

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

**cout << arr[i] << " ";**

**}**

**cout << "\n" << "\n";**

**cout << "Количество четных элементов: " << count\_even << endl;**

**cout << "Количество нечетных элементов: " << count\_odd << endl;**

**for (int i = 1; i < 10; i++) {**

**int counter\_del = 0;**

**for (int j = 0; j < N; j++) {**

**if (arr[j] % i == 0) {**

**counter\_del += 1;**

**}**

**};**

**cout << " Кол-во чисел делящихся на " << i << " равно " << counter\_del << endl;**

**};**

**//3 zad**

**// Время поиска минимального и максимального элемента в неотсортированном массиве**

**auto start = chrono::high\_resolution\_clock::now();**

**int minUnsorted, maxUnsorted;**

**findMinMax(arr, N, minUnsorted, maxUnsorted);**

**auto end = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::nano> elapsedUnsorted = end - start;**

**cout << "Минимальный элемент (неотсортированный): " << minUnsorted << endl;**

**cout << "Максимальный элемент (неотсортированный): " << maxUnsorted << endl;**

**cout << "Время поиска минимального и максимального элемента (неотсортированный): "**

**<< elapsedUnsorted.count() << " наносекунд." << endl;**

**bubbleSort(arr, N);// Сортировка баблсортом**

**cout << "Элементы массива после сортировки:" << endl;**

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

**cout << arr[i] << " ";**

**}**

**cout << endl;**

**cout << "\n";**

**start = chrono::high\_resolution\_clock::now();**

**int minSorted, maxSorted;**

**findMinMax(arr, N, minSorted, maxSorted);**

**end = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::nano> elapsedSorted = end - start;**

**cout << "Минимальный элемент (отсортированный): " << minSorted << endl;**

**cout << "Максимальный элемент (отсортированный): " << maxSorted << endl;**

**cout << "Время поиска (отсортированный): " << elapsedSorted.count() << " наносекунд" << endl;**

**//4 zad**

**//ср.знач макс и мин элементов**

**double averageUnsorted = round((minUnsorted + maxUnsorted) / 2.0);**

**double averageSorted = round((minSorted + maxSorted) / 2.0);**

**cout << "Среднее значение (неотсортированный): " << averageUnsorted << endl;**

**cout << "Среднее значение (отсортированный): " << averageSorted << endl;**

**// Поиск индексов для неотсортированного массива**

**vector<int> indicesUnsorted;**

**start = chrono::high\_resolution\_clock::now();**

**findIndices(arr, N, averageUnsorted, indicesUnsorted);**

**end = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::nano> elapsedIndicesUnsorted = end - start;**

**cout << "Индексы элементов, равных среднему значению (неотсортированный): ";**

**for (int index : indicesUnsorted) {**

**cout << index << " ";**

**}**

**cout << "\n";**

**cout << "Количество: " << indicesUnsorted.size() << endl;**

**cout << "Время поиска индексов (неотсортированный): " << elapsedIndicesUnsorted.count() << " наносекунд" << endl;**

**vector<int> indicesSorted;**

**start = chrono::high\_resolution\_clock::now();**

**findIndices(arr, N, averageSorted, indicesSorted);**

**end = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::nano> elapsedIndicesSorted = end - start; // Время в наносекундах**

**cout << "Индексы элементов, равных среднему значению (отсортированный): ";**

**for (int index : indicesSorted) {**

**cout << index << " ";**

**}**

**cout << "\n";**

**cout << "Количество: " << indicesSorted.size() << endl;**

**cout << "Время поиска индексов (отсортированный): " << elapsedIndicesSorted.count() << " наносекунд" << endl;**

**cout << "\n";**

**//5 zad**

**int a;**

**cout << "Введите число a: ";**

**cin >> a;**

**int countLess = countLessThan(arr, N, a);**

**cout << "Количество элементов в отсортированном массиве, которые меньше числа " << a << ": " << countLess << endl;**

**cout << "\n";**

**//6 zad**

**int b;**

**cout << "Введите число b: ";**

**cin >> b;**

**int countGreater = countBiggestThan(arr, N, b);**

**cout << "Количество элементов в отсортированном массиве, которые больше числа " << b << ": " << countGreater << endl;**

**cout << "\n";**

**//7 zad**

**int searchValue;**

**cout << "Введите число для поиска: ";**

**cin >> searchValue;**

**// Сравнение скорости бинарного поиска и линейного поиска**

**auto startLinear = chrono::high\_resolution\_clock::now();**

**bool foundLinear = linearSearch(arr, N, searchValue);**

**auto endLinear = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::nano> elapsedLinear = endLinear - startLinear; // Время в наносекундах**

**auto startBinary = chrono::high\_resolution\_clock::now();**

**bool foundBinary = binarySearch(arr, N, searchValue);**

**auto endBinary = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::nano> elapsedBinary = endBinary - startBinary; // Время в наносекундах**

**cout << "Результат линейного поиска: " << (foundLinear ? "Найдено" : "Не найдено") << endl;**

**cout << "Время линейного поиска: " << elapsedLinear.count() << " наносекунд" << endl;**

**cout << "Результат бинарного поиска: " << (foundBinary ? "Найдено" : "Не найдено") << endl;**

**cout << "Время бинарного поиска: " << elapsedBinary.count() << " наносекунд" << endl;**

**cout << "\n";**

**//8 zad**

**int index1, index2;**

**cout << "Введите индексы для обмена местами (от 0 до " << N - 1 << "): ";**

**cin >> index1 >> index2;**

**if (index1 >= 0 && index1 < N && index2 >= 0 && index2 < N) {**

**auto startSwap = chrono::high\_resolution\_clock::now();**

**swapElements(arr, index1, index2);**

**auto endSwap = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::nano> elapsedSwap = endSwap - startSwap; // Время в наносекундах**

**cout << "Элементы массива после обмена местами:" << endl;**

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

**cout << arr[i] << " ";**

**}**

**cout << endl;**

**cout << "\n";**

**cout << "Время обмена: " << elapsedSwap.count() << " наносекунд" << endl;**

**}**

**else {**

**cout << "Ошибка: индексы должны быть в диапазоне от 0 до " << N - 1 << "." << endl;**

**}**

**}**

**const int MAX\_SIZE = 10;**

**void lab3() {**

**HANDLE hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);**

**setlocale(LC\_ALL, "ru-RU");**

**int N;**

**cout << "Введите порядок матрицы (6, 8 или 10): ";**

**cin >> N;**

**if (N != 6 && N != 8 && N != 10) {**

**cout << "Неверный порядок матрицы!" << endl;**

**return;**

**}**

**random\_device rd;**

**mt19937 gen(rd());**

**uniform\_int\_distribution<> dis(1, N \* N);**

**int value = 1;**

**int\* matrix = new int[N \* N];**

**// Заполнение матрицы по схеме змейки перед разделением на квадранты**

**for (int j = 0; j < N; ++j) {**

**if (j % 2 == 0) { // Четные столбцы**

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

**\*(matrix + i \* N + j) = dis(gen);**

**}**

**}**

**else { // Нечетные столбцы**

**for (int i = N - 1; i >= 0; --i) {**

**\*(matrix + i \* N + j) = dis(gen);**

**}**

**}**

**}**

**fillAndPrintVerticalSnake(hStdout, N);**

**cout << endl;**

**fillAndPrintSpiral(hStdout, N);**

**cout << endl << endl << endl << endl;**

**int choice;**

**do {**

**cout << "\nЗадания 2 и 3:\n";**

**cout << "1. Показать случайную матрицу\n";**

**cout << "2. Квадранты 4-1-2-3 (A)\n";**

**cout << "3. Квадранты 3-4-1-2 (B)\n";**

**cout << "4. Квадранты 4-3-2-1 (C)\n";**

**cout << "5. Квадранты 2-1-4-3 (D)\n";**

**cout << "6. Сортировка всей матрицы (Shaker Sort)\n";**

**cout << "0. 4 Задание\n";**

**cout << "Введите ваш выбор: ";**

**cin >> choice;**

**switch (choice) {**

**case 1:**

**cout << "Случайная матрица:\n";**

**fillAndPrintMatrix(matrix, N);**

**break;**

**case 2:**

**printQuadrantsA(matrix, N);**

**break;**

**case 3:**

**printQuadrantsB(matrix, N);**

**break;**

**case 4:**

**printQuadrantsC(matrix, N);**

**break;**

**case 5:**

**printQuadrantsD(matrix, N);**

**break;**

**case 6:**

**shakerSortMatrix(matrix, N);**

**break;**

**case 0:**

**cout << endl;**

**break;**

**default:**

**cout << "Неверный выбор! Попробуйте снова." << endl;**

**}**

**} while (choice != 0);**

**cout << "Исходная матрица:" << endl;**

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

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << \*(matrix + i \* N + j);**

**}**

**cout << endl;**

**}**

**char operation;**

**int number;**

**cout << "Выберите операцию (+, -, \*, /): ";**

**cin >> operation;**

**cout << "Введите число для операции: ";**

**cin >> number;**

**manipulateMatrix(matrix, N, operation, number);**

**srand(static\_cast<unsigned>(time(0)));**

**cout << "Введите размер матрицы N: ";**

**cin >> N;**

**int\*\* matrix1 = new int\* [N];**

**int\*\* matrix2 = new int\* [N];**

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

**matrix1[i] = new int[N];**

**matrix2[i] = new int[N];**

**}**

**fillMatrixSequential(matrix1, N);**

**fillMatrixRandom(matrix2, N);**

**cout << "Первая матрица (от 1 до N^2):" << endl;**

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

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << matrix1[i][j];**

**}**

**cout << endl;**

**}**

**cout << "Вторая матрица (рандом от 1 до N^2):" << endl;**

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

**for (int j = 0; j < N; ++j) {**

**cout << setw(4) << matrix2[i][j];**

**}**

**cout << endl;**

**}**

**subtractAndDisplay(matrix1, matrix2, N);**

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

**delete[] matrix1[i];**

**delete[] matrix2[i];**

**}**

**delete[] matrix1;**

**delete[] matrix2;**

**delete[] matrix;**

**}**

**void lab4() {**

**cout << "Введите слова: ";**

**string input;**

**getline(cin, input);**

**// Очищаем строку от лишних пробелов и знаков препинания**

**input = cleanString(input);**

**// 3. Выводим слова без цифр**

**istringstream ss(input);**

**string word;**

**cout << "\nСлова без цифр: ";**

**while (ss >> word) {**

**if (!containsDigit(word)) {**

**cout << word << " ";**

**}**

**}**

**cout << endl;**

**// 4. Преобразуем каждое слово в строке, делая первую букву заглавной**

**ss.clear();**

**ss.str(input);**

**cout << "Слова с 1 заглавной: ";**

**while (ss >> word) {**

**cout << fixCapitalization(word) << " ";**

**}**

**cout << endl;**

**// 5. Линейный поиск подстроки**

**string substring;**

**cout << "Подстрока для поиска: ";**

**cin >> substring;**

**vector<size\_t> positions = findSubstring(input, substring);**

**if (positions.empty()) {**

**cout << "Не найдено." << endl;**

**}**

**else {**

**cout << "Подстрока найдена на позиции: ";**

**for (size\_t pos : positions) {**

**cout << pos << " ";**

**}**

**cout << endl;**

**}**

**}**

**int main() {**

**setlocale(LC\_ALL, "ru-RU");**

**int choice;**

**// Главное меню**

**do {**

**clearConsole();**

**cout << "Меню:\n";**

**cout << "1. Лабораторная 1\n";**

**cout << "2. Лабораторная 2\n";**

**cout << "3. Лабораторная 3\n";**

**cout << "4. Лабораторная 4\n";**

**cout << "Введите номер лабораторной работы: ";**

**cin >> choice;**

**switch (choice) {**

**case 1:**

**lab1();**

**break;**

**case 2:**

**lab2();**

**break;**

**case 3:**

**lab3();**

**break;**

**case 4:**

**lab4();**

**break;**

**}**

**cout << "\n" << "\n";**

**cout << "Нажмите 'C' для очистки консоли и возврата в меню.\n";**

**while (true) {**

**if (\_kbhit()) {**

**char ch = \_getch();**

**if (ch == 'c' || ch == 'C') {**

**break;**

**}**

**}**

**}**

**} while (choice != 0);**

**return 0;**

**}**

**Выводы.**

Я изучил строки, массивы, типы данных и поняла, как они представлены в памяти компьютера, научился работать с ними.