**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 | Некрасова А.В. |  |
| 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 <cmath>**

**#include <vector>**

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

**#include <iomanip> // Для std::setw**

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

**#include <random>**

**#include <stdio.h>**

**#include <cstring>**

**using namespace std;**

**void aaa(int& integer) // Двоичное представление в памяти согласно условию**

**{**

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

**if (integer >= 0)**

**{**

**for (int i = 1; i <= sizeof(int) \* 8; i = i + 2)**

**{**

**integer = integer & (~(1 << i));**

**};**

**}**

**else**

**{**

**for (int i = 0; i <= sizeof(int) \* 8 - 1; i = i + 2)**

**{**

**integer = integer | (1 << i);**

**};**

**};**

**}**

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

**int start = 0, end = n - 1;**

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

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

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

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

**}**

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

**bool swapped = true;**

**int start = 0, end = n - 1;**

**while (swapped) {**

**swapped = false;**

**for (int i = start; i < end; ++i) { // Проход слева направо**

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

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

**swapped = true;**

**}**

**}**

**if (!swapped) {**

**break;**

**}**

**swapped = false;**

**--end;**

**for (int i = end - 1; i >= start; --i) { // Проход справа налево**

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

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

**swapped = true;**

**}**

**}**

**++start;**

**}**

**}**

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

**int start = 0, end = n - 1, count = 0;**

**float k = 1.247;**

**while (end >= 1)**

**{**

**for (int i = 0; i + end < 100; i++)**

**{**

**if (arr[i] > arr[int(i + end)])**

**swap(arr[int(i + end)], arr[i]);**

**}**

**end /= k;**

**}**

**while (true) {**

**for (int i = 0; i < 99; i++) {**

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

**swap(arr[int(i + end)], arr[i]);**

**}**

**else count++;**

**}**

**if (count == 99)**

**break;**

**else**

**count = 0;**

**}**

**}**

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

**int i, key, j;**

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

**key = arr[i];**

**j = i - 1;**

**while (j >= 0 && arr[j] > key) {**

**arr[j + 1] = arr[j];**

**j = j - 1;**

**}**

**arr[j + 1] = key;**

**}**

**}**

**void QuickSort(int\* arr, int end, int begin) {**

**int mid;**

**int f = begin;**

**int l = end;**

**mid = arr[(f + l) / 2];**

**while (f < l) {**

**while (arr[f] < mid) f++;**

**while (arr[l] > mid) l--;**

**if (f <= l) {**

**swap(arr[f], arr[l]);**

**f++;**

**l--;**

**}**

**}**

**if (begin < l) QuickSort(arr, l, begin);**

**if (f < end) QuickSort(arr, end, f);**

**}**

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

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

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

**}**

**std::cout << std::endl;**

**}**

**void ShellSort(int arr[]) {**

**int n = 100;**

**for (int gap = n / 2; gap > 0; gap /= 2) {**

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

**int temp = arr[i];**

**int j;**

**for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {**

**arr[j] = arr[j - gap];**

**}**

**arr[j] = temp;**

**}**

**}**

**}**

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

**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 fillMatrixRandom3x3(int\*\* matrix, int N) {**

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

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

**matrix[i][j] = rand() % 61 - 30;**

**}**

**}**

**}**

**int determinant3x3(int\*\* matrix) {**

**int\* row0 = matrix[0];**

**int\* row1 = matrix[1];**

**int\* row2 = matrix[2];**

**//применяем правило треугольника**

**int det = (\*row0) \* (\*(row1 + 1) \* \*(row2 + 2) - \*(row1 + 2) \* \*(row2 + 1)) -**

**(\*(row0 + 1)) \* (\*(row1) \* \*(row2 + 2) - \*(row1 + 2) \* \*(row2)) +**

**(\*(row0 + 2)) \* (\*(row1) \* \*(row2 + 1) - \*(row1 + 1) \* \*(row2));**

**return det;**

**}**

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

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

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

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

**}**

**cout << endl;**

**}**

**}**

**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 VerticalSnake(HANDLE hStdout, int N) {**

**random\_device rd; //генератор случайных чисел**

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

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

**int value = 1;**

**int startRow = 1;**

**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 вывода**

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

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**}**

**else { //нечетные столбцы**

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

**setCursorPosition(hStdout, j \* 5, startRow + i + 2); //позиция вывода**

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

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**}**

**}**

**}**

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

**random\_device rd; //генератор случайных чисел**

**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 and top <= bottom) {**

**//заполняем верхнюю строку**

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

**setCursorPosition(hStdout, i \* 5, top + N + 4); //позиция для вывода**

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

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**top++;**

**//заполняем правый столбец**

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

**setCursorPosition(hStdout, right \* 5, i + N + 4); //позиция для вывода**

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

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**right--;**

**//заполняем нижнюю строку**

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

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

**setCursorPosition(hStdout, i \* 5, bottom + N + 4); //позиция для вывода**

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

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**bottom--;**

**}**

**//заполняем левый столбец**

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

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

**setCursorPosition(hStdout, left \* 5, i + N + 4); //позиция для вывода**

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

**cout.flush();**

**this\_thread::sleep\_for(chrono::milliseconds(100));**

**}**

**left++;**

**}**

**}**

**}**

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

**int halfN = N / 2;**

**int\* quadrants[4]; //массив указателей**

**//выделяем память для каждого блока**

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

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

**}**

**for (int i = 0; i < halfN; ++i) { //перебор эл-ов блоков и заполнение их зн-ями из исходной матрицы**

**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 printB(int\* matrix, int N) {**

**int halfN = N / 2;**

**//создаем массив для хранения блоков**

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

**//выделяем память**

**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 printC(int\* matrix, int N) {**

**int halfN = N / 2;**

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

**//выделяем память для каждого блока**

**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 printD(int\* matrix, int N) {**

**int halfN = N / 2;**

**//создаем массив для хранения блоков**

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

**//выделяем память**

**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 << "На ноль делить нельзя" << 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;**

**}**

**}**

**void DeleteProbels(char\* str) {**

**int l = strlen(str);**

**if (l == 0) return;**

**char\* result = new char[l + 1];**

**int j = 0;**

**bool probel = false;**

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

**if (str[i] == ' ') {**

**if (!probel) {**

**result[j++] = ' '; // Добавляем один пробел**

**probel = true; // Устанавливаем флаг, что мы в пробеле**

**}**

**}**

**else {**

**result[j++] = str[i]; // Добавляем текущий символ**

**probel = false; // Сбрасываем флаг**

**}**

**}**

**if (j > 0 && result[0] == ' ') {**

**std::memmove(result, result + 1, j - 1); // Удаляем пробел в начале**

**--j; // Уменьшаем длину результирующей строки**

**}**

**if (j > 0 && result[j - 1] == ' ') {**

**--j; // Удаляем пробел в конце**

**}**

**result[j] = '\0';**

**strcpy\_s(str, l + 1, result);**

**delete[] result;**

**}**

**bool Sign(char c) {**

**return (c == '!' || c == '?' || c == '.' || c == ',' || c == ';' || c == ':');**

**}**

**void RemoveSigns(const char\* input, char\* output) {**

**bool last\_was\_punct = false;**

**bool last\_was\_ellipsis = false;**

**int output\_index = 0;**

**int length = strlen(input);**

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

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

**// Проверяем, является ли текущий символ знаком препинания**

**if (Sign(current)) {**

**// Проверяем, не является ли это многоточием**

**if (current == '.' && i + 1 < length && input[i + 1] == '.' && i + 2 < length && input[i + 2] == '.') {**

**// Если предыдущий символ не был многоточием, добавляем его**

**if (!last\_was\_ellipsis) {**

**output[output\_index++] = '.';**

**output[output\_index++] = '.';**

**output[output\_index++] = '.';**

**last\_was\_ellipsis = true;**

**i += 2; // Пропускаем следующие два символа**

**}**

**}**

**else {**

**// Если это не многоточие и предыдущий символ также был знаком препинания**

**if (!last\_was\_punct) {**

**output[output\_index++] = current; // Добавляем текущий знак**

**last\_was\_punct = true; // Устанавливаем флаг**

**}**

**}**

**}**

**else {**

**output[output\_index++] = current; // Добавляем обычный символ**

**last\_was\_punct = false; // Сбрасываем флаг**

**last\_was\_ellipsis = false; // Сбрасываем флаг для многоточия**

**}**

**}**

**output[output\_index] = '\0'; // Завершаем строку нулем**

**}**

**void Register(char\* input) {**

**if (input[0] == '\0')**

**return;**

**input[0] = toupper(input[0]);**

**for (size\_t i = 1; input[i] != '\0'; ++i) {**

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

**}**

**}**

**void NoNumbers(const char\* str) {**

**const char\* start = str;**

**const char\* end;**

**while (\*start) {**

**// Пропускаем пробелы**

**while (\*start == ' ') {**

**start++;**

**}**

**// Если достигнут конец строки, выходим из цикла**

**if (\*start == '\0') {**

**break;**

**}**

**// Находим конец слова**

**end = start;**

**while (\*end != ' ' && \*end != '\0') {**

**end++;**

**}**

**// Проверяем слово на наличие цифр**

**bool yesNumbers = false;**

**for (const char\* символ = start; символ < end; символ++) {**

**if (isdigit(\*символ)) {**

**yesNumbers = true;**

**break;**

**}**

**}**

**// Если слово не содержит цифр, выводим его**

**if (!yesNumbers) {**

**for (const char\* символ = start; символ < end; символ++) {**

**cout << \*символ;**

**}**

**cout << ' ';**

**}**

**// Переходим к следующему слову**

**start = end;**

**}**

**}**

**void NumberOfLetters(const char\* str) {**

**const char\* start = str;**

**const char\* end;**

**while (\*start) {**

**// Пропускаем пробелы**

**while (\*start == ' ') {**

**start++;**

**}**

**// Если достигнут конец строки, выходим из цикла**

**if (\*start == '\0') {**

**break;**

**}**

**// Находим конец слова**

**end = start;**

**while (\*end != ' ' && \*end != '\0') {**

**end++;**

**}**

**// Подсчитываем количество букв в слове**

**int number = 0;**

**for (const char\* symbol = start; symbol < end; symbol++) {**

**if (isalpha(\*symbol)) { // Проверяем, является ли символ буквой**

**number++;**

**}**

**}**

**// Выводим количество букв в слове**

**cout << "Количество букв в слове \"";**

**for (const char\* symbol = start; symbol < end; symbol++) {**

**cout << \*symbol;**

**}**

**cout << "\": " << number << endl;**

**// Переходим к следующему слову**

**start = end;**

**}**

**}**

**void LinearSearch(const char\* str, const char\* substr) { // Ф У Н К Ц И Я Д Л Я Л И Н Е Й Н О Г О П О И С К А**

**int lengthStr = strlen(str);**

**int lengthSubstr = strlen(substr);**

**bool bober = false;**

**// Линейный поиск подстроки**

**for (int i = 0; i <= lengthStr - lengthSubstr; i++) {**

**int j;**

**for (j = 0; j < lengthSubstr; j++) {**

**if (str[i + j] != substr[j]) {**

**break;**

**}**

**}**

**if (j == lengthSubstr) { // Если мы нашли подстроку**

**cout << i << ' ';**

**bober = true;**

**}**

**}**

**if (!bober) {**

**cout << "Подстрока не найдена." << endl;**

**}**

**}**

**void PrefixFunction(const char\* substr, vector<int>& prefix) {**

**int lengthSubstr = strlen(substr);**

**prefix.resize(lengthSubstr);**

**prefix[0] = 0; // Начальное значение префикса**

**int j = 0; // Длина предыдущего префикса**

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

**while (j > 0 && substr[i] != substr[j]) {**

**j = prefix[j - 1]; // Переход к предыдущему префиксу**

**}**

**if (substr[i] == substr[j]) {**

**++j; // Увеличиваем длину префикса**

**}**

**prefix[i] = j; // Устанавливаем значение префикса**

**}**

**}**

**/\*void АлгоритмКМП(const char\* str, const char\* substr) {**

**int lengthStr = strlen(str);**

**int lengthSubstr = strlen(substr);**

**vector<int> prefix;**

**PrefixFunction(substr, prefix);**

**int j = 0; // Индекс для подстроки**

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

**while (j > 0 && str[i] != substr[j]) {**

**j = prefix[j - 1]; // Переход к предыдущему префиксу**

**}**

**if (str[i] == substr[j]) {**

**++j; // Увеличиваем длину совпадения**

**}**

**if (j == lengthSubstr) {**

**cout << i - j + 1 << ' ';**

**j = prefix[j - 1]; // Продолжаем поиск**

**}**

**}**

**}\*/**

**int main()**

**{**

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

**int doop;**

**do{**

**cout << "Перечень работ: \n" << "1) Л/р 1\n" << "2) Л/р 2\n" << "3) Л/р 3\n" << "4) Л/р 4\n" << "Ваш выбор: ";**

**int numberofwork;**

**cin >> numberofwork;**

**if (numberofwork == 1){**

**cout << "Условие: \n";**

**cout << "Если введенное число отрицательное, то значения четных битов равно 1,\nесли введенное число положительное, то значения нечетных битов равно 0.\n";**

**cout << '\n';**

**//ЗАДАНИЕ 1**

**cout << "ЗАДАНИЕ 1:\n";**

**cout << sizeof(int) << " байт памяти под тип данных int\n";**

**cout << sizeof(short int) << " байт памяти под тип данных short int\n";**

**cout << sizeof(unsigned short int) << " байт памяти под тип данных unsigned short int\n";**

**cout << sizeof(long int) << " байт памяти под тип данных long int\n";**

**cout << sizeof(float) << " байт памяти под тип данных float\n";**

**cout << sizeof(double) << " байт памяти под тип данных double\n";**

**cout << sizeof(long double) << " байт памяти под тип данных long double\n";**

**cout << sizeof(char) << " байт памяти под тип данных char\n";**

**cout << sizeof(bool) << " байт памяти под тип данных bool\n";**

**cout << sizeof(long long) << " байт памяти под тип данных long long\n";**

**cout << '\n';**

**//ЗАДАНИЕ 2**

**cout << "ЗАДАНИЕ 2:\n";**

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

**int integer;**

**cin >> integer;**

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

**cout << "Двоичное представление в памяти целого числа " << integer << " : ";**

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

**{**

**if (integer & mask)**

**cout << '1';**

**else**

**cout << '0';**

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

**cout << ' ';**

**}**

**cout << '\n';**

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

**cout << "Двоичное представление в памяти целого числа " << integer << " с условием: ";**

**aaa(integer);**

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

**{**

**if (integer & mask)**

**cout << '1';**

**else**

**cout << '0';**

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

**cout << ' ';**

**};**

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

**//ЗАДАНИЕ 3**

**cout << '\n' << "ЗАДАНИЕ 3:\n";**

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

**union**

**{**

**float typefloat;**

**int ib;**

**};**

**cin >> typefloat;**

**//cout << &typefloat << ' ' << &ib << '\n';**

**//cout << typefloat << ' ' << ib << '\n';**

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

**cout << "Двоичное представление в памяти вещественного числа " << typefloat << ": ";**

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

**{**

**if (ib & mask)**

**cout << '1';**

**else**

**cout << '0';**

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

**cout << ' ';**

**}**

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

**cout << '\n';**

**//ЗАДАНИЕ 4**

**cout << '\n' << "ЗАДАНИЕ 4:\n";**

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

**union**

**{**

**double typedouble;**

**unsigned int part[2];**

**};**

**cin >> typedouble;**

**cout << "Двоичное представление в памяти числа " << typedouble << " типа double: \n";**

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

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

**{**

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

**{**

**if (part[nomber] & mask)**

**cout << '1';**

**else**

**cout << '0';**

**}**

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

**}**

**cout << '\n';**

**}**

**if (numberofwork == 2) {**

**srand(time(0)); //чтобы случайные числа были каждый раз разные**

**int massiv[100], massiv0[100], sort, loop = 1;**

**for (int i = 0; i < 100; ++i)**

**massiv[i] = rand() % (199) - 99; //присваиваем массиву рандомные значения**

**cout << "Массив: ";**

**PrintArray(massiv, 100);**

**copy(massiv, massiv + 100, massiv0);**

**do {**

**cout << "\nВыберете сортировку массива: 1) BubbleSort 2) ShakerSort 3) CombSort 4) InsertionSort 5) QuickSort 6) ShellSort(идз)\nВаш выбор: ";**

**cin >> sort;**

**if (sort == 1) {**

**auto start = chrono::high\_resolution\_clock::now(); // Начало измерения времени**

**BubbleSort(massiv, 100);**

**auto end = chrono::high\_resolution\_clock::now(); // Конец измерения времени**

**cout << "Отсортированный массив (BubbleSort): ";**

**PrintArray(massiv, 100);**

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

**cout << "Время сортировки: " << duration.count() << " миллисекунд" << std::endl;**

**copy(massiv0, massiv0 + 100, massiv);**

**}**

**if (sort == 2) {**

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

**ShakerSort(massiv, 100);**

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

**cout << '\n' << "Отсортированный массив (ShakerSort): ";**

**PrintArray(massiv, 100);**

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

**cout << "Время сортировки: " << duration.count() << " миллисекунд" << std::endl;**

**copy(massiv0, massiv0 + 100, massiv);**

**}**

**if (sort == 3) {**

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

**CombSort(massiv, 100);**

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

**cout << '\n' << "Отсортированный массив (CombSort): ";**

**PrintArray(massiv, 100);**

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

**cout << "Время сортировки: " << duration.count() << " миллисекунд" << std::endl;**

**copy(massiv0, massiv0 + 100, massiv);**

**}**

**if (sort == 4) {**

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

**InsertionSort(massiv, 100);**

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

**cout << '\n' << "Отсортированный массив (InsertionSort): ";**

**PrintArray(massiv, 100);**

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

**cout << "Время сортировки: " << duration.count() << " миллисекунд" << std::endl;**

**copy(massiv0, massiv0 + 100, massiv);**

**}**

**if (sort == 5) {**

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

**QuickSort(massiv, 99, 0);**

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

**cout << '\n' << "Отсортированный массив (QuickSort): ";**

**PrintArray(massiv, 100);**

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

**cout << "Время сортировки: " << duration.count() << " миллисекунд" << std::endl;**

**copy(massiv0, massiv0 + 100, massiv);**

**}**

**if (sort == 6) {**

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

**ShellSort(massiv);**

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

**cout << '\n' << "Отсортированный массив (ShellSort): ";**

**PrintArray(massiv, 100);**

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

**cout << "Время сортировки: " << duration.count() << " миллисекунд" << std::endl;**

**copy(massiv0, massiv0 + 100, massiv);**

**}**

**cout << "Будем применять к циклу другие сортировки? Да - 1, нет - 0. Ваш вариант: ";**

**cin >> loop;**

**} while (loop == 1);**

**int maxmassiv = -101, minmassiv = 101, sredmassiv, kmassiv = 0, kmassiv0 = 0;**

**QuickSort(massiv, 99, 0); //РАБОТА С ОТСОРТИРОВАННЫМ МАССИВОМ (massiv)**

**auto start1 = chrono::high\_resolution\_clock::now();**

**maxmassiv = massiv[99];**

**auto end1 = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::milli> duration1 = end1 - start1;**

**auto start2 = chrono::high\_resolution\_clock::now();**

**minmassiv = massiv[0];**

**auto end2 = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::milli> duration2 = end2 - start2;**

**sredmassiv = (maxmassiv + minmassiv) / 2;**

**cout << "\nСреднее значение максимального и минимального чисел в массиве: " << std::round(sredmassiv) << "\n";**

**cout << "\nРАБОТА С ОТСОРТИРОВАННЫМ МАССИВОМ\nВремя нахождения максимального эл - та: " << duration1.count() << " миллисекунд" << std::endl;**

**cout << "Время нахождения минимального эл - та: " << duration2.count() << " миллисекунд" << std::endl;**

**cout << "Индексы эл - ов, которые равны среднему значению: ";**

**auto start3 = chrono::high\_resolution\_clock::now();**

**for (int i = 0; i < 100 && massiv[i] <= sredmassiv; i++) {**

**if (massiv[i] == sredmassiv) {**

**cout << i << ' ';**

**++kmassiv;**

**}**

**}**

**auto end3 = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::milli> duration3 = end3 - start3;**

**cout << "\nВремя поиска индексов: " << duration3.count() << " миллисекунд\n";**

**cout << "Кол - во эл - ов, равных среднему значению: " << kmassiv << "\n";**

**int maxmassiv0 = massiv0[0], minmassiv0 = massiv0[0];**

**auto start4 = chrono::high\_resolution\_clock::now(); //РАБОТА С НЕОТСОРТИРОВАННЫМ МАССИВОМ (massiv0)**

**for (int i = 1; i < 100; ++i) {**

**if (massiv0[i] > massiv0[i + 1])**

**maxmassiv0 = massiv0[i];**

**}**

**auto end4 = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::milli> duration4 = end4 - start4;**

**cout << "\nРАБОТА С НЕОТСОРТИРОВАННЫМ МАССИВОМ\nВремя нахождения максимального эл - та: " << duration4.count() << " миллисекунд" << std::endl;**

**auto start5 = chrono::high\_resolution\_clock::now();**

**for (int i = 1; i < 100; ++i) {**

**if (massiv0[i] < massiv0[i + 1])**

**minmassiv0 = massiv0[i];**

**}**

**auto end5 = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::milli> duration5 = end5 - start5;**

**cout << "Время нахождения минимального эл - та: " << duration5.count() << " миллисекунд" << std::endl;**

**cout << "Индексы эл - ов, которые равны среднему значению: ";**

**auto start6 = chrono::high\_resolution\_clock::now();**

**for (int i = 0; i < 100; i++) {**

**if (massiv0[i] == sredmassiv) {**

**cout << i << ' ';**

**++kmassiv0;**

**}**

**}**

**auto end6 = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::milli> duration6 = end6 - start6;**

**cout << "\nВремя поиска индексов: " << duration6.count() << " миллисекунд" << std::endl;**

**cout << "Кол - во эл - ов, равных среднему значению: " << kmassiv0 << "\n";**

**//ЗАДАНИЕ 5**

**int a, b, target, kolmina = 0, kolmaxb = 0, loop1 = 1, loop2 = 1, loop3 = 1;**

**do {**

**cout << "\nЗадание 5. Введите число a (выведем кол - во элементов, которые меньше числа a): ";**

**cin >> a;**

**for (int i = 0; i < 100; i++) {**

**if (massiv[i] < a)**

**kolmina++;**

**}**

**cout << "Кол - во эл - ов, меньших заданного числа a: " << kolmina;**

**kolmina = 0;**

**cout << "\nБудем вводить число a еще? Да - 1, нет - 0. Ваш вариант: ";**

**cin >> loop1;**

**} while (loop1 == 1);**

**//ЗАДАНИЕ 6**

**do {**

**cout << "\nЗадание 6. Введите число b (выведем кол - во элементов, которые больше числа b): ";**

**cin >> b;**

**for (int i = 0; i < 100; i++) {**

**if (massiv[i] > b)**

**kolmaxb++;**

**}**

**cout << "Кол - во эл - ов, больших заданного числа b: " << kolmaxb;**

**kolmaxb = 0;**

**cout << "\nБудем вводить число b еще? Да - 1, нет - 0. Ваш вариант: ";**

**cin >> loop2;**

**} while (loop2 == 1);**

**//ЗАДАНИЕ 7**

**do {**

**cout << "\nЗадание 7. Введите число для поиска: ";**

**cin >> target;**

**int left = 0, right = 99, flag = 0;**

**auto start7 = chrono::high\_resolution\_clock::now();**

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

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

**if (massiv[mid] == target) {**

**cout << "Элемент найден";**

**flag = 1;**

**break;**

**}**

**if (massiv[mid] < target) {**

**left = mid + 1; // Ищем в правой половине**

**}**

**else**

**right = mid - 1; // Ищем в левой половине**

**}**

**if (flag == 0)**

**cout << "Элемент не найден";**

**auto end7 = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::milli> duration7 = end7 - start7;**

**auto start8 = chrono::high\_resolution\_clock::now();**

**for (int i = 0; i < 100; i++) {**

**if (massiv[i] == target)**

**break;**

**}**

**auto end8 = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::milli> duration8 = end8 - start8;**

**cout << "\nВремя методом бинарного поиска: " << duration7.count() << " миллисекунд";**

**cout << "\nВремя методом перебора: " << duration8.count() << " миллисекунд";**

**cout << "\nБудем вводить число для поиска еще? Да - 1, нет - 0. Ваш вариант: ";**

**cin >> loop3;**

**} while (loop3 == 1);**

**//ЗАДАНИЕ 8**

**int indeks1, indeks2, bober = 1, loop4;**

**do {**

**cout << "\nЗадание 8. Чтобы поменять элементы местами, введите первый индекс: ";**

**cin >> indeks1;**

**cout << "Введите второй индекс: ";**

**cin >> indeks2;**

**cout << "Массив после опреации обмена эл - ов:\n";**

**auto start9 = chrono::high\_resolution\_clock::now();**

**bober = massiv[indeks1];**

**massiv[indeks1] = massiv[indeks2];**

**massiv[indeks2] = bober;**

**auto end9 = chrono::high\_resolution\_clock::now();**

**chrono::duration<double, std::milli> duration9 = end9 - start9;**

**PrintArray(massiv, 100);**

**cout << "Время обмена эл - ов местами: " << duration9.count() << " миллисекунд";**

**cout << "\nБудем еще менять элементы местами? Да - 1, нет - 0. Ваш вариант: ";**

**cin >> loop4;**

**} while (loop4 == 1);**

**}**

**if (numberofwork == 3) {**

**system("cls");**

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

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

**int N;**

**cout << "Выберите порядок 6 или 8 или 10: ";**

**cin >> N;**

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

**cout << "Введите другое значение" << endl;**

**return 1;**

**}**

**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 блоки**

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

**if (j % 2 == 0) { //четные**

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

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

**}**

**}**

**else { //нечетные**

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

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

**}**

**}**

**}**

**VerticalSnake(hStdout, N);**

**cout << endl;**

**Spiral(hStdout, N);**

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

**int choice;**

**do {**

**cout << "\nВыберите один из вариантов:\n";**

**cout << "1) Случайная матрица\n";**

**cout << "2) Перестоновка 4-1-2-3 (a)\n";**

**cout << "3) Перестоновка 3-4-1-2 (b)\n";**

**cout << "4) Перестоновка 4-3-2-1 (c)\n";**

**cout << "5) Перестоновка 2-1-4-3 (d)\n";**

**cout << "6) Сортировка shaker sort\n";**

**cout << "7) Операции с матрицей\n";**

**cout << "Ваш выбор: ";**

**cin >> choice;**

**switch (choice) {**

**case 1:**

**cout << "Случайная матрица:\n";**

**fillAndPrintMatrix(matrix, N);**

**break;**

**case 2:**

**printA(matrix, N);**

**break;**

**case 3:**

**printB(matrix, N);**

**break;**

**case 4:**

**printC(matrix, N);**

**break;**

**case 5:**

**printD(matrix, N);**

**break;**

**case 6:**

**shakerSortMatrix(matrix, N);**

**break;**

**case 7:**

**cout << endl;**

**break;**

**default:**

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

**}**

**} while (choice != 7);**

**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)));**

**int\*\* matrix1 = new int\* [3];**

**N = 3;**

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

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

**}**

**fillMatrixRandom3x3(matrix1, N);**

**cout << "\nИДЗ ЗАДАНИЕ 10\nМатрица 3x3:" << endl;**

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

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

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

**}**

**cout << endl;**

**}**

**int det = determinant3x3(matrix1);**

**cout << "\nОпределитель матрицы 3x3: " << det;**

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

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

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

**}**

**delete[] matrix1;**

**delete[] matrix;**

**}**

**if (numberofwork == 4) {**

**const int q = 501;**

**char strr0[q], str[q], substr[q];**

**cout << "Введите строку: ";**

**cin.ignore();**

**cin.getline(strr0, 501);**

**cout << "Отформатированная строка: ";**

**DeleteProbels(strr0);**

**RemoveSigns(strr0, str);**

**Register(str);**

**cout << str << endl << "\nСлова последовательности, не содержащие цифр: ";**

**NoNumbers(str);**

**cout << endl << "Кол-во символов в каждом слове исходной последовательности" << endl;**

**NumberOfLetters(str);**

**cout << endl << "ПОИСК ПОДСТРОКИ В СТРОКЕ" << endl << "Введите подстроку: ";**

**cin.getline(substr, 501);**

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

**LinearSearch(str, substr);**

**}**

**cout << "\n\n\nБУДЕМ СМОТРЕТЬ ДРУГИЕ ЛАБЫ? Да - 1, нет - 0. Ваш вариант: ";**

**cin >> doop;**

**if (doop == 1)**

**system("cls");**

**} while (doop == 1);**

**return 0;**

**}**

**Выводы.**

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