# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра информационных систем**

# ОТЧЕТ

**по практической работе №4**

# по дисциплине «Программирование» Тема: Текстовые строки как массивы символов

|  |  |
| --- | --- |
| Студентка гр. 3372 | Шорсткая А.А. |
| Преподаватель | Глущенко А. Г. |

Санкт-Петербург 2023

# Цель работы.

Разработать программу, позволяющая выполнять различные операции с массивами, включая сортировку и поиск.

# Основные теоретические положения.

Массив – структура данных, хранящая набор значений, идентифицируемых по индексу или набору индексов, принимающих целые значения из некоторого заданного непрерывного диапазона. Одномерный массив можно рассматривать как реализацию абстрактного типа данных. Подробная информация о массиве представлена на рисунке 1.



Рисунок 1 – Одномерный массив

Сортировка массива – это расположение элементов массива в некотором заданном порядке (по возрастанию или убыванию). Например, на рисунке 2 представлена сортировка пузырьком.

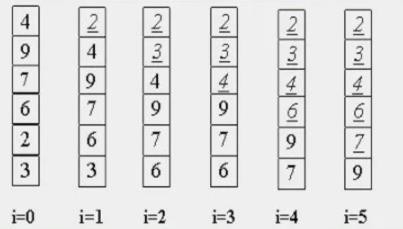


Рисунок 2 – Сортировка пузырьком В данной работе представлены следующие сортировки:

* сортировка полным перебором (от меньше к большему) – это сортировка выполняющая полный перебор всех элементов массива сравнивая и переставляя их при удовлетворенности условий;
* «быстрая сортировка» – один из самых популярных алгоритмов, в котором используется принцип «разделяй и властвуй»;
* сортировка «слиянием» – это сортировка, где мы делится массив пополам, каждый из них сортируется слиянием и потом соединяются оба массива.

Поиск элемента в массиве дает возможность удостоверится о наличии данного элемента в части массива.

В неотсортированном массиве поиск элемента необходимо просматривать весь массив (линейный поиск).

В отсортированном массиве поиск элемента можно осуществлять, не просматривая весь массив. Например, бинарный поиск представлен на рисунке 3.

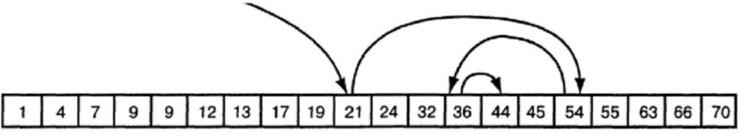


Рисунок 3 – Бинарный поиск

# Постановка задачи.

Разработать алгоритм и написать программу, которая позволяет:

1. Создать целочисленный массив размерности N = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.
2. Отсортировать заданный в пункте 1 массив […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.
3. Найти максимальный и минимальный элемент массива. Подсчитать время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.
4. Выводить среднее значение (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном. Выводить индексы всех элементов, которые равны этому значению, и их количество. Подсчитать время поиска.
5. Выводить количество элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем.
6. Выводить количество элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем.
7. Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором.
8. Менять местами элементы массива, индексы которых вводит пользователь. Выводить скорость обмена, используя библиотеку chrono.

# Выполнение работы.

Код программы представлен в приложении А.

# Выводы.

Данная практическая работа позволяет сформировать навыки и умения работы с массивами, усвоить принцип сортировки и поиска, сравнить время работы различных функций и методов.

# Ход работы

1. При запуске программы необходимо выбрать номер действия (рисунок 1).

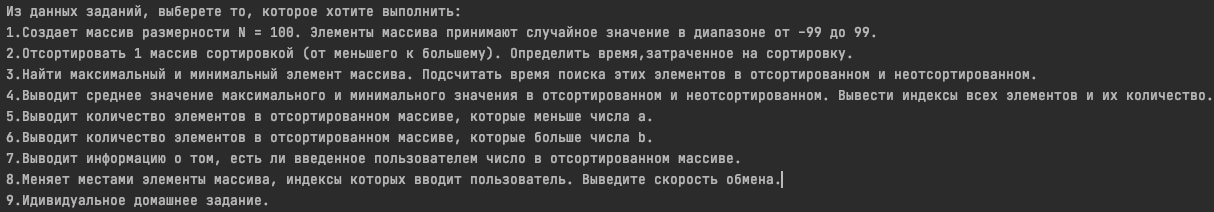


Рисунок 1 – Выбор номер поведения программы

1. При выборе пункта «1» выводится массив размерности N = 100. Элементы массива принимают случайное значение в диапазоне от -99 до 99 (рисунок 2).

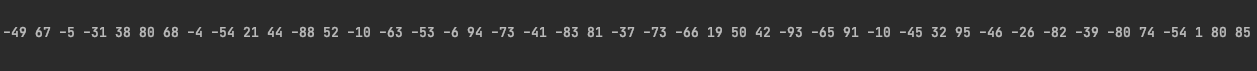


Рисунок 2 – Вывод массива

1. При выборе пункта «2» осуществляется сортировка «пузырьком» и отображение время выполнения сортировки (рисунок 3).

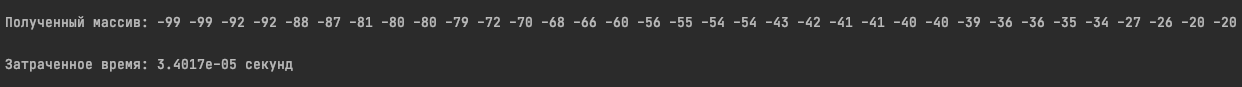


Рисунок 3 – Сортировка «пузырьком»

1. При выборе пункта «3» осуществляется поиск минимального и максимального элемента в неотсортированном и отсортированном массиве и расчет времени этого поиска (рисунок 4).

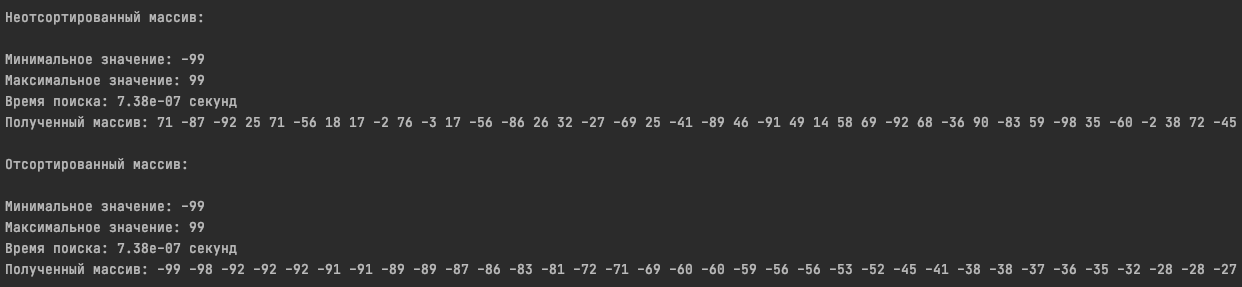


Рисунок 4 – Сортировка «слиянием»

1. При выборе пункта «4» отображается среднее минимального и максимального элемента в неотсортированном и отсортированном массиве, также выводятся индексы всех элементов и их количество и время поиска (рисунок 5)

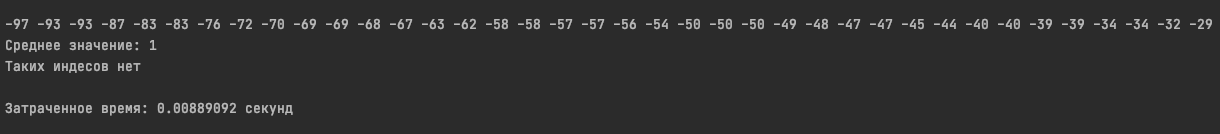


Рисунок 5 – Подсчет и поиск среднего в массиве в неотсортированном и отсортированном массиве

6. При выборе пункта «5» и «6» отображается количество элементов массива, меньше и больше введенного значения пользователя соответственно (рисунок 6,7).

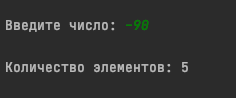


Рисунок 6 – Отображение количества элементов, меньше введённого значения

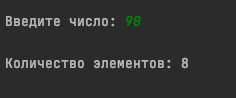


Рисунок 7 – Отображение количества элементов, больше введённого значения

1. При выборе пункта «7» выводится информацию о том, есть ли введенное пользователем число в отсортированном массиве, также сравнивает скорость работы бинарного поиска с обычным перебором (рисунок 8)

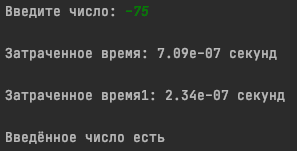


Рисунок 8 – Отображение информации о том, что есть введённое число в массиве и сравнение скорости работы бинарного поиска с обычным перебором

1. При выборе пункта «8» осуществляется смена элементов массива при помощи индексов, введенных пользователем и выводится скорость обмена (рисунок 9)

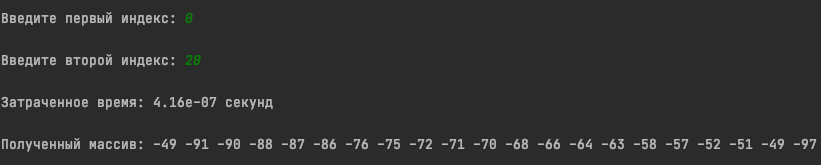


Рисунок 9 – Смена элементов

# ПРИЛОЖЕНИЕ А РАБОЧИЙ КОД

Листинг А.1 – Рабочий код программы

#include <iostream>  
#include <string>  
#include <vector>  
#include <cctype>  
using namespace std;  
  
int searchSubstring(string text, string pattern) {  
 int n = text.length(); // длина текста  
 int m = pattern.length(); // длина подстроки  
  
 for (int i = 0; i <= n - m; i++) {  
 int j;  
 for (j = 0; j < m; j++) {  
 if (text[i + j] != pattern[j])  
 break;  
 }  
 if (j == m) // если все символы совпали  
 return i; // вернуть индекс начала подстроки  
 }  
 return -1; // подстрока не найдена  
}  
  
std::vector<int> suffixes(const std::string& pattern) {  
 int length = pattern.length();  
 std::vector<int> suffix(length, 0);  
  
 int g, f;  
 g = f = length - 1;  
 for (int i = length - 2; i >= 0; --i) {  
 if (i > g && suffix[i + length - 1 - f] < i - g) {  
 suffix[i] = suffix[i + length - 1 - f];  
 } else {  
 if (i < g) {  
 g = i;  
 }  
 f = i;  
 while (g >= 0 && pattern[g] == pattern[g + length - 1 - f]) {  
 --g;  
 }  
 suffix[i] = f - g;  
 }  
 }  
  
 return suffix;  
}  
  
std::vector<int> boyerMoore(const std::string& text, const std::string& pattern) {  
 int n = text.length();  
 int m = pattern.length();  
 std::vector<int> occurrences;  
  
 std::vector<int> shift(256, m);  
 for (int i = 0; i < m - 1; ++i) {  
 shift[pattern[i]] = m - i - 1;  
 }  
  
 std::vector<int> suffix = suffixes(pattern);  
  
 int i = 0;  
 while (i <= n - m) {  
 int j = m - 1;  
 while (j >= 0 && pattern[j] == text[i + j]) {  
 --j;  
 }  
 if (j < 0) {  
 occurrences.push\_back(i);  
 i += shift[text[i + m]];  
 } else {  
 int x = suffix[j];  
 int y = j - x;  
 int z = m - 1 - j;  
 i += std::max(shift[text[i + j]] - y, z);  
 }  
 }  
  
 return occurrences;  
}  
  
int main() {  
 setlocale(LC\_ALL, "ru");  
 srand(time(NULL));  
 bool task = true;  
 int choose = 0, idz = 0, idzChoose = -1;  
  
 while (choose != -1) {  
 cout << "\n\n\nИз данных заданий, выберете то, которое хотите выполнить: \n"  
 "1.С клавиатуры вводится последовательность, содержащая от 1 до 50 слов, в каждом из которых от 1 до 10 строчных латинских букв и цифр. Между словами произвольное количество пробелов. За последним символом стоит точка\n"  
 "2.Редактирование входного текста.\n"  
 "3.Вывести на экран только те слова последовательности, в которых первая буква слова встречается в этом слове еще раз.\n"  
 "4.Вывести на экран ту же последовательность, заменив во всех словах первую букву соответствующей прописной буквой.\n"  
 "5.Необходимо найти все подстроки, которую введёт пользователь в имеющейся строке. Реализуйте алгоритм Линейный поиск.\n\n\n";  
  
 cout << "Задание: ";  
 cin >> choose;  
 cout << "\n";  
  
 if (choose != 1 and choose != 2 and choose != 3 and choose != 4 and choose != 5 and choose != 0)  
 cout << "Такого варианта ответа нет \n";  
  
 else  
 switch (choose) {  
 case 0: {  
 cout << "Пока \n\n";  
 choose = -1;  
 break;  
 }  
  
 case 1: {  
 std::string sequence;  
 std::cout << "Введите последовательность слов:" << std::endl;  
 std::getline(std::cin, sequence, '.'); // вводим последовательность до точки  
  
 int wordCount = 0;  
 std::string currentWord;  
 // перебираем символы в последовательности  
 for (char c : sequence) {  
 // если символ - пробел, то заканчиваем текущее слово и начинаем следующее  
 if (c == ' ') {  
 if (!currentWord.empty()) {  
 std::cout << currentWord << " " ;  
 currentWord.clear();  
 }  
 } else if (isalnum(c)) { // добавляем буквы и цифры в слово  
 currentWord += c;  
 } else { // игнорируем другие символы  
 continue;  
 }  
 }  
 // печатаем последнее слово, если оно есть  
 if (!currentWord.empty()) {  
 std::cout << currentWord << "." << std::endl;  
 }  
 break;  
 }  
  
 case 2: {  
 std::string inputText;  
 std::cout << "Введите текст: ";  
 std::getline(std::cin, inputText);  
 std::getline(std::cin, inputText);  
 std::cout << "Вывели:" << inputText<< '\n';  
  
 // Удаление лишних пробелов  
 bool previousIsSpace = false;  
 for (auto it = inputText.begin(); it != inputText.end();) {  
 if (std::isspace(\*it)) {  
 if (previousIsSpace) {  
 it = inputText.erase(it);  
 } else {  
 previousIsSpace = true;  
 ++it;  
 }  
 } else {  
 previousIsSpace = false;  
 ++it;  
 }  
 }  
 // Удаление лишних знаков препинания  
 for (auto it = inputText.begin(); it != inputText.end();) {  
 if (std::ispunct(\*it)) {  
 auto next = std::next(it);  
 while (next != inputText.end() && std::ispunct(\*next)) {  
 next = inputText.erase(next);  
 }  
 ++it;  
 } else {  
 ++it;  
 }  
 }  
 // Исправление регистра букв  
 for (auto &ch : inputText) {  
 if (std::isalpha(ch)) {  
 if (std::islower(ch)) {  
 ch = std::tolower(ch);  
 } else {  
 ch = std::tolower(ch);  
 }  
 }  
 }  
 std::cout << "Отредактированный текст: " << inputText;  
 break;  
 }  
  
 case 3:{  
 std::string inputText;  
 std::cout << "Введите текст: ";  
 std::getline(std::cin, inputText);  
 std::getline(std::cin, inputText);  
  
 bool is\_new\_word = true;  
 bool need\_to\_print\_word = false;  
 int from = 0;  
 char first\_char = inputText[0];  
 for (int i = 0; i < inputText.length(); i++) {  
 if (is\_new\_word && !ispunct(inputText[i]) && inputText[i] != ' ') {  
 from = i;  
 first\_char = inputText[i];  
 is\_new\_word = false;  
 } else if (ispunct(inputText[i]) || inputText[i] == ' ') {  
 if (need\_to\_print\_word) {  
 cout << inputText.substr(from, i - from) << '\n';  
 }  
 need\_to\_print\_word = false;  
 is\_new\_word = true;  
 } else if (first\_char == inputText[i]) {  
 need\_to\_print\_word = true;  
 }  
 }  
 if (need\_to\_print\_word) {  
 cout << inputText.substr(from, inputText.length()) << '\n';  
 }  
 break;  
 }  
  
 case 4:{  
 std::string sequence;  
 std::cout << "Введите последовательность: " << std::endl;  
 std::getline(std::cin, sequence);  
 std::getline(std::cin, sequence);  
  
 bool needToLower = true;  
  
 for (auto &ch : sequence) {  
 if (ch == ' '){  
 needToLower = true;  
 }  
 if (std::isalpha(ch) && needToLower == true) {  
 if (std::isupper(ch)) {  
 ch = std::tolower(ch);  
 needToLower = false;  
 }  
 }  
 }  
 std::cout << "Отредактированный текст: " << sequence;  
 break;  
 }  
  
 case 5:{  
 string text, pattern;  
 std::cout << "Введите строку: ";  
 std::cin >> text;  
 std::cout << "Введите подстроку: ";  
 std::getline(std::cin, pattern);  
 std::getline(std::cin, pattern);  
  
 int result = searchSubstring(text, pattern);  
 if (result == -1)  
 cout << "Подстрока не найдена" << endl;  
 else  
 cout << "Подстрока найдена в позиции: " << result << endl;  
  
  
 std::string text1, pattern1;  
 std::cout << "Введите строку: ";  
 std::cin >> text1;  
 std::cout << "Введите подстроку: ";  
 std::getline(std::cin, pattern1);  
 std::getline(std::cin, pattern1);  
  
 std::vector<int> occurrences = boyerMoore(text1, pattern1);  
  
 if (!occurrences.empty()) {  
 std::cout << "Pattern found at positions: ";  
 for (int i = 0; i < occurrences.size(); ++i) {  
 std::cout << occurrences[i] << " ";  
 }  
 } else {  
 std::cout << "Pattern not found.";  
 }  
  
 break;  
 }  
 }  
 }  
 return 0;  
}