**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 массив символов

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. |  | Саевич И. И. 4372 |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Изучение текстовых строк; получение практических навыков работы с ними (обработка строк, алгоритмы поиска подпоследовательностей);

**Основные теоретические положения.**

Текстовые строки представляются с помощью одномерных массивов символов. В языке C++ текстовая строка представляет собой набор символов, обязательно заканчивающийся нулевым символом (‘\0’). Поэтому, если вы хотите создать текстовый массив для хранения 10 (N) символов, нужно выделить память под 11(N+1) символов.

Объявленный таким образом массив может использоваться для хранения текстовых строк, содержащих не более 10 символов. Нулевой символ позволяет определить границу между содержащимся в строке текстом и неиспользованной частью строки.

При определении строковых переменных их можно инициализировать конкретными значениями с помощью строковых литералов:

char S1[15] = “This is text”;

char S2[] = “Пример текста”;

Последние два элемента переменной просто не используются, а строка автоматически подстраивается под длину инициализирующего текста.

При работе со строками можно обращаться к отдельным символам строки как в обычном одномерном массиве с помощью индексов:

cout << S1[0]; // На экране будет выведен символ ‘T’

Если строка формируется при помощи цикла (или иного способа), то необходимо в ее конец обязательно записать нулевой символ '\0'.

**Постановка задачи.**

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 (под «лишними» подразумевается несколько подряд идущих знаков (обратите внимание, что «…» - корректное использование знака) в тексте);

· исправить регистр букв, если это требуется (пример некорректного использования регистра букв: пРиМЕр);

3) Вывести на экран слова последовательности в обратном порядке.

4) Вывести на экран ту же последовательность, заменив во всех словах первую букву соответствующей прописной буквой.

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

**Выполнение работы.**

Код программы представлен в приложении A.

Пользователь вводит некоторую последовательность символов.



Программа редактирует введённый пользователем текст, а именно: убирает лишние знаки препинания, лишние пробелы и исправляет некорректное использование регистра букв.



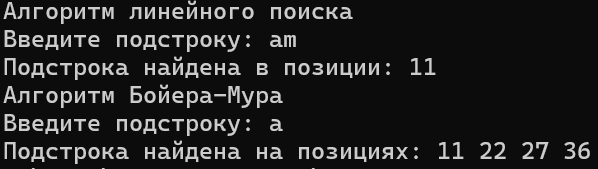
Выводит на экран ту же последовательность слов в обратном порядке.



Выводит на экран ту же последовательность, заменив во всех словах первую букву соответствующей заглавной буквой.



Находит индекс первого вхождения введённой пользователем подпоследовательности при помощи алгоритма линейного поиска, а также при помощи алгоритма Бойера-Мура находит индексы всех вхождений подстроки.



**Выводы.**

В ходе работы мы изучили текстовые строки, научились обрабатывать их, а также освоили алгоритмы поиска подстрок.

Приложение А

рабочий код

﻿ #define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#include <fstream>

#include <cstring>

#include <cctype>

#include <vector>

using namespace std;

const int maxWords = 50;

const int maxLength = 11;

const int maxInputSize = 1000;

void RemoveExtraSpaces(char\* str)

{

int n = strlen(str);

int j = 0;

int count = 0;

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

{

if (str[i] != ' ' || (i > 0 && str[i - 1] != ' '))

{

str[j] = str[i];

j++;

}

}

str[j] = '\0';

}

void RemoveExtraPunctuation(char\* str)

{

int n = strlen(str);

int j = 0;

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

{

if (!ispunct(str[i]) || (i > 0 && !ispunct(str[i - 1])))

{

str[j++] = str[i];

}

}

str[j] = '\0';

}

void FixLetterCase(char\* str)

{

int n = strlen(str);

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

{

if (str[i - 1] != ' ' && isupper(str[i]))

str[i] = tolower(str[i]);

}

}

void EditText(char\* text)

{

RemoveExtraSpaces(text);

RemoveExtraPunctuation(text);

FixLetterCase(text);

}

int TextToWords(const char\* text, char(&words)[maxWords][maxLength])

{

int wordCount = 0;

char text1[maxInputSize];

strcpy(text1, text);

char\* token = strtok(text1, " .,:;-?!()[]{}<>|/+=\*&^%$#@~`№");

while (token != nullptr && wordCount < maxWords)

{

strcpy(words[wordCount], token);

wordCount++;

token = strtok(nullptr, " .,:;-?!()[]{}<>|/+=\*&^%$#@~`№");

}

return wordCount;

}

void ChangeRegister(char (&words)[maxWords][maxLength], int count)

{

for (int i = 0; i < count; i++)

words[i][0] = toupper(words[i][0]);

}

int LinearSearch(char\* text, char\* pattern)

{

int n = strlen(text);

int m = strlen(pattern);

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)

return i;

}

return -1;

}

vector<int> Suffixes(const char\* pattern) {

int length = strlen(pattern);

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;

}

vector<int> boyerMoore(const char\* text, const char\* pattern) {

int n = strlen(text);

int m = strlen(pattern);

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 += max(shift[text[i + j]] - y, z);

}

}

return occurrences;

}

int main()

{

setlocale(0, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char text[maxInputSize];

char words[maxWords][maxLength];

cout << "Введите последовательность слов, заканчивающуюся точкой:\n";

cin.getline(text, maxInputSize);

EditText(text);

cout << "Отредактированный текст:\n";

cout << text;

int count = TextToWords(text, words);

cout << endl << "Количество слов: " << count << endl;

cout << "Слова в обратном порядке: " << endl;

for (int i = count - 1; i >= 0; --i)

cout << words[i] << " ";

cout << endl;

ChangeRegister(words, count);

for (int i = 0; i < count; i++)

cout << words[i] << " ";

cout << endl;

char pattern1[maxLength];

char pattern2[maxLength];

cout << "Алгоритм линейного поиска\nВведите подстроку: ";

cin.get(pattern1, maxLength);

int result = LinearSearch(text, pattern1);

if (result == -1)

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

else

cout << "Подстрока найдена в позиции: " << result << endl;

cout << "Алгоритм Бойера-Мура\nВведите подстроку: ";

cin >> pattern2;

vector<int> occurrences = boyerMoore(text, pattern2);

if (!occurrences.empty())

{

cout << "Подстрока найдена на позициях: ";

for (int i = 0; i < occurrences.size(); ++i)

cout << occurrences[i] << " ";

}

else

cout << "Подстрока не найдена";

return 0;

}