**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.

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. |  | Адигюзалова А.А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Изучение текстовых строк как массивов символов и способов работы с ними.

**Основные теоретические положения.**

**Массив** представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Все массивы можно разделить на две группы: одномерные и многомерные. Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных. Объявление в программах одномерных массивов выполняется в соответствии со следующим правилом:

**<Базовый тип элементов> <Идентификатор массива>[<Количество элементов>]**

Так, например:

int ArrInt [10], A1 [20];

Обращение к определенному элементу массива осуществляется с помощью указания значения индекса этого элемента:

A1 [4] = -1200;

cout << A1 [4];

При обращении к конкретному элементу массива этот элемент можно рассматривать как обычную переменную, тип которой соответствует базовому типу элементов массива, и осуществлять со значением этого элемента любые операции, которые характерны для базового типа.

При объявлении массива его можно инициализировать определенными значениями:

short S [5] = {1, 4, 9, 16, 25};

Или:

short S [] = {1, 4, 9, 16, 25};

**Текстовые строки** представляются с помощью одномерных массивов символов. В языке C++ текстовая строка представляет собой набор символов, обязательно заканчивающийся нулевым символом (‘\0’). 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 текстовых литералов:

char S1[15] = “This is text”;

char S2[] = “Пример текста”;

При выводе строк можно использовать форматирование (манипуляторы или функции потока вывода). Ввод текста с клавиатуры можно осуществлять разными способами. Непосредственное чтение текстовых строк из потока вывода осуществляется до первого знака пробела. Для того, чтобы прочесть всю строку полностью, можно воспользоваться одной из функций gets или get\_s.

Существует несколько способов поиска подстроки в строке:

1. Прямой поиск (линейный поиск) – это циклическое сравнение всех символов строки с подстрокой. Он является самым долгим. Одной из вариаций данного алгоритма является реализация таблицы включений.
2. Алгоритм Кнута-Мориса-Пратта – это эффективный алгоритм поиска подстроки в строке. Он основан на принципе, что при несовпадении символов между искомой подстрокой и текстом, можно использовать уже полученные результаты для пропуска частей текста, которые точно не совпадут.
3. Алгоритм Бойера – Мура считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке. Его преимущество заключается в том, что ценной некоторого количества предварительных вычислений над подстрокой, подстрока сравнивается с исходным текстом не во всех позициях (пропускаются позиции, которые точно не дадут положительный результат). Поиск подстроки ускоряется благодаря созданию таблиц сдвигов. Сравнение подстроки со строкой начинается с последнего символа подстроки, а затем происходит прыжок, длина которого определяется по таблице сдвигов. Таблица сдвигов строится по подстроке так, чтобы перепрыгнуть максимальное количество символов строки и не пропустить вхождение строки в подстроку.

**Постановка задачи.**

Необходимо разработать алгоритм и написать программу, которая реализует следующие задачи:

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 соседними словами произвольное количество пробелов. За последним символом стоит точка.
2. Отредактировать входной текст: удалить лишние пробелы; удалить лишние знаки препинания (под «лишними» подразумевается несколько подряд идущих знаков); исправить регистр букв, если это требуется.
3. Вывести на экран слова последовательности в обратном порядке.
4. Вывести на экран ту же последовательность, заменив во всех словах первую букву соответствующей прописной.
5. Необходимо найти все подстроки, которые введет пользователь в имеющейся строке. Реализовать алгоритм линейного поиска и алгоритма Бойера-Мура.

**Выполнение работы.**

Код программы представлен в приложении А.

Описание кода и использованных алгоритмов:

При запуске программы перед пользователем появляется окно, в котором на экран выводится

1. Дается возможность ввести строку текста.
2. Исправленная строка.
3. Последовательность в обратном порядке.
4. Та же строка, но во всех словах первая буква заменена на соответствующую прописную.
5. Интерактивное поле, где пользователь может ввести подстроку, которую он хочет найти в строке.
6. Результат поиска с использованием алгоритма линейного поиска.
7. Результат поиска с использованием алгоритма Бойера – Мура.

Работа алгоритма и вывод на экран:



Рис. 1. Поле для ввода строки.



Рис.2 Исправленная строка.



Рис. 3. Вывод строки в обратном порядке.



Рис. 4. Вывод той же строки, на с заменой первой буквы слов на соответствующую прописную



Рис. 5. Интерактивное поле для ввода подстроки



Рис. 6. Результат поиска подстроки с использованием линейного поиска и алгоритма Бойера – Мура.

Тестовые данные:

|  |  |
| --- | --- |
| String | Substring |
| Hello, world | world |

**Выводы.**

В ходе практической работы было изучение представление текстовых строк в виде массивов символом, а также способы работы с такими строками.

Приложение А

рабочий код

#include <iostream>

#include <stdio.h>

using namespace std;

unsigned factArrayLength(char\* stringArr) {

int length = 0;

while (stringArr[length]) {

length++;

}

return length;

}

int deletePunctuation(char\* arr, char character, char \*modifiedArr) {

int count = 0;

int indexModifiedArr = 0;

for (int i = 0; arr[i] != '\0'; i++) {

if (arr[i] == character) {

count++;

if (count <= 1) {

modifiedArr[indexModifiedArr] = arr[i];

indexModifiedArr++;

}

}

else {

modifiedArr[indexModifiedArr] = arr[i];

indexModifiedArr++;

count = 0;

}

modifiedArr[indexModifiedArr] = '\0';

}

return 1;

}

int deleteElipsisMarks(char\* arr, char character, char\* modifiedArr, int factLength) {

int count = 0;

int indexModifiedArr = 0;

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

if (arr[i] == character) {

count++;

if (count <= 3) {

modifiedArr[indexModifiedArr] = arr[i];

indexModifiedArr++;

}

}

else {

modifiedArr[indexModifiedArr] = arr[i];

indexModifiedArr++;

count = 0;

}

modifiedArr[indexModifiedArr + 1] = '\0';

}

return 1;

}

int changeLetterCase(char\* arr, int factLength) {

const int differenceBetweenLettersLatin = 32;

if (arr[0] >= 97 and arr[0] <= 122)

arr[0] -= differenceBetweenLettersLatin;

for (int index = 1; index < factLength - 2; index++) {

if (arr[index] == ' ' and arr[index - 1] == '.' and arr[index + 1] >= 97 and arr[index + 1] <= 122) {

arr[index + 1] -= differenceBetweenLettersLatin;

}

if (arr[index] == ' ' and arr[index - 1] != '.' and arr[index + 1] >= 65 and arr[index + 1] <= 90) {

arr[index + 1] += differenceBetweenLettersLatin;

}

if (arr[index] != ' ' and (arr[index - 2] != '.' and arr[index - 1] != '.') and arr[index] >= 65 and arr[index] <= 90)

arr[index] += differenceBetweenLettersLatin;

}

return 1;

}

void changeToCapitalLetter(char\* arr, int factLength) {

const int differenceBetweenLettersLatin = 32;

const int differenceBetweenLettersRussianSecondHalf = 80;

for (int index = 1; index < factLength; index++) {

if (arr[index] == ' ') {

if (arr[index - 1] >= 97 && arr[index - 1] <= 122 or arr[index - 1] >= 160 && arr[index - 1] <= 175)

arr[index - 1] = arr[index - 1] - differenceBetweenLettersLatin;

if (arr[index - 1] >= 224 && arr[index - 1] <= 239)

arr[index - 1] = arr[index - 1] - differenceBetweenLettersRussianSecondHalf;

}

}

}

void backwardArrayModification(char\* arr, int factLength, char \*backwardArray) {

int index = 0;

for (int i = factLength - 1; i > -1; i--) {

backwardArray[index] = arr[i];

index++;

backwardArray[index + 1] = '\0';

}

}

int linearSearch(char\* stringArr, char\* substringArr) {

int factLengthStringArr = factArrayLength(stringArr);

int factLengthSubstringArr = factArrayLength(substringArr);

int count = 0;

for (int i = 0; i <= factLengthStringArr - factLengthSubstringArr; i++) {

int j;

for (j = 0; j < factLengthSubstringArr; j++) {

if (stringArr[i + j] != substringArr[j])

break;

}

if (j == factLengthSubstringArr)

count++;

}

return count;

}

int checkWordAmount(char\* stringArr, int factLength) {

bool check = true;

int index = 0;

int count = 0;

for (index = 0; index < factLength; index++) {

if (stringArr[index] == ' ') {

if (count >= 10) {

check = false;

break;

}

count = 0;

}

else

count++;

}

if (check != true)

return -1;

}

int max(int numberOne, int numberTwo) {

return (numberOne > numberTwo) ? numberOne : numberTwo;

}

void badCharacterHeuristic(char\* stringArr, int size, int badCharacter[256]) {

int i, index;

for (i = 0; i < 256; i++)

badCharacter[i] = -1;

for (i = 0; i < size; i++) {

index = stringArr[i];

badCharacter[index] = i;

}

}

int searchBoyerMoor(char\* stringArr, char\* substringArr) {

int factLengthStringArr = factArrayLength(stringArr);

int factLengthSubstringArr = factArrayLength(substringArr);

int badCharacter[256];

int count = 0;

badCharacterHeuristic(substringArr, factLengthSubstringArr, badCharacter);

int shift = 0;

while (shift <= (factLengthStringArr - factLengthSubstringArr)) {

int j = factLengthSubstringArr - 1;

while (j >= 0 && substringArr[j] == stringArr[shift + j])

j--;

if (j < 0) {

count++;

shift += (shift + factLengthSubstringArr < factLengthStringArr) ? factLengthSubstringArr - badCharacter[stringArr[shift + factLengthSubstringArr]] : 1;

}

else {

shift += max(1, j - badCharacter[stringArr[shift + j]]);

}

}

return count;

}

int main()

{

int const approximateLength = 100;

char stringArr[approximateLength];

char finalModification[approximateLength];

int factLengthFinal, checkCorrectness = 1;

cout << "Enter your string (!NOTE Your string mast contain 1 to 50 words, and each of them should consist of 1 to 10 letters and/or numbers; the string should end in \".\")" << endl;

cin.getline(stringArr, approximateLength);

cin.sync();

int factLengthString = factArrayLength(stringArr);

char modifiedArrSpaces[approximateLength];

deletePunctuation(stringArr, ' ', modifiedArrSpaces);

char modifiedArrCommas[approximateLength];

deletePunctuation(modifiedArrSpaces, ',', modifiedArrCommas);

char modifiedArrExclamationMark[approximateLength];

deletePunctuation(modifiedArrCommas, '!', modifiedArrExclamationMark);

char modifiedArrQuestionMark[approximateLength];

deletePunctuation(modifiedArrExclamationMark, '?', modifiedArrQuestionMark);

char modifiedArrSlash[approximateLength];

deletePunctuation(modifiedArrQuestionMark, '/', modifiedArrSlash);

char modifiedArrColon[approximateLength];

deletePunctuation(modifiedArrSlash, ':', modifiedArrColon);

char modifiedArrSemicolon[approximateLength];

deletePunctuation(modifiedArrColon, ';', modifiedArrSemicolon);

char modifiedArrRoundBracketLeft[approximateLength];

deletePunctuation(modifiedArrSemicolon, '(', modifiedArrRoundBracketLeft);

char modifiedArrRoundBracketRight[approximateLength];

deletePunctuation(modifiedArrRoundBracketLeft, ')', modifiedArrRoundBracketRight);

char modifiedArrSquareBracketLeft[approximateLength];

deletePunctuation(modifiedArrRoundBracketRight, '[', modifiedArrSquareBracketLeft);

char modifiedArrSquareBracketRight[approximateLength];

deletePunctuation(modifiedArrSquareBracketLeft, ']', modifiedArrSquareBracketRight);

char modifiedArrDash[approximateLength];

deletePunctuation(modifiedArrSquareBracketRight, '-', modifiedArrDash);

char modifiedArrQuotationMarks[approximateLength];

deletePunctuation(modifiedArrDash, '\"', modifiedArrQuotationMarks);

int factLength = factArrayLength(modifiedArrSlash);

deleteElipsisMarks(modifiedArrSlash, '.', finalModification, factLength);

factLengthFinal = factArrayLength(finalModification);

int checkWords = checkWordAmount(finalModification, factLengthFinal);

if (stringArr[factLengthString - 1] != '.' or factLengthFinal > 50 or checkWords == -1) {

cout << "Please, try again";

checkCorrectness = 0;

}

changeLetterCase(finalModification, factLengthFinal);

cout << "Your fixed array looks like: " << endl << finalModification << endl;

char backwardArray[approximateLength];

backwardArrayModification(finalModification, factLengthFinal, backwardArray);

cout << "Your string backwards looks like: " << endl << backwardArray << endl;

changeToCapitalLetter(backwardArray, factLengthFinal);

cout << "The backward string with the first letter of each word being turned into a capital one looks like: " << endl << backwardArray << endl;

cout << "What substring do you want to do find in your string: " << endl;

char substringArr[approximateLength];

cin.getline(substringArr, approximateLength);

cin.sync();

cout << "Linear Search: " << endl;

int countLinearSearch = linearSearch(finalModification, substringArr);

cout << "The amount of times your substring can be found in the string is " << countLinearSearch << endl;

cout << "Boyer-Moor Search: " << endl;

int countBoyerMoor = searchBoyerMoor(finalModification, substringArr);

cout << "The amount of times your substring can be found in the string is " << countBoyerMoor << endl;

}