**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. 2372 |  | Котлова П. В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение текстовых строк, класса string; применение ввода, вывода и обработки текстовых строк, используя арифметику указателей и функции; научиться осуществлять поиск подстроки в строке при помощи алгоритмов: Линейного поиска, алгоритмов КМП и Бойера-Мура.

**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’).

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

char S1[15] = “This is text”;

char S2[] = “Пример текста”;

При выводе строк можно использовать форматирование (манипуляторы или функции потока вывода).

char Str[21] = “Это пример текста”;

cout  <<  Str << endl;

cout  <<  “Это текстовый литерал.” << endl;

Предпочтительно использование функции потока ввода cin.getline:

const int N = 21;

char Str [N];

cin.getline (Str, N);      // Пусть введена строка “Это пример текста”

cout  <<  Str << endl;  // На экран будет выведено “ Это пример текста”

**Обработка текстовых строк**

* Функция **strlen(char \*s)** – возвращает фактическую длину текстовой строки, хранящейся в символьном массиве **s** (см. аналог **my\_StrLen(char \*S)**).
* Функция **strcpy(char \*dest, char \*source)** – копирует содержимое строки **source** в строку **dest**.
* Функция **strcat(char \*s1, char \*s2)** – добавляет содержимое строки **s2** в конец строки **s1**.
* Функция **strcmp(char \*s1, char \*s2)** – осуществляет лексикографическое сравнение строк **s1**и **s2**. Возвращает значение 0, если строки одинаковы (равны), значение большее 0 при **s1 > s2**и отрицательное значение при **s1 < s2**.

**Класс string**

Класс string предназначен для работы со строками типа char, которые представляют собой строчку с завершающим нулем (символ ‘\0’). Класс string был введен как альтернативный вариант для работы со строками типа char.

Чтобы использовать возможности класса string, нужно подключить библиотеку <string> и пространство имен std. Объявление же переменной типа string осуществляется схоже с обычной переменной:

string S1; // Переменная с именем s1 типа string

string S2 = “Пример”; // объявление с инициализацией

Создание нового типа string было обусловлено недостатками работы с строками символов, который показывал тип char. В сравнении с этим типом string имеет ряд основных преимуществ:

· возможность использования для обработки строк стандартные операторы С++(=,+,<,==,>,+=,!=,<=,>=,[])(=,+,<,==,>,+=,!=,<=,>=,[]). Использование типа char приводило требовало написание чрезмерного программного кода;

· обеспечение лучшей надежности программного кода;

· обеспечение строки, как самостоятельного типа данных.

Класс string обладает широким функционалом:

·        функция compare() сравнивает одну часть строки с другой;

·        функция length() определяет длину строки;

·        функции find() и rfind() служат для поиска подстроки в строке (отличаются функции лишь направлением поиска);

·        функция erase() служит для удаления символов;

·        функция replace() выполняет замену символов;

·        функция insert() необходима, чтобы вставить одну строку в заданную позицию другой строки;

**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) Выполнить задание по варианту:

|  |  |
| --- | --- |
| 1 | Вывести на экран слова последовательности в обратном порядке. |
| 2 | Вывести на экран слова последовательности в алфавитном порядке. |
| 3 | Вывести на экран слова последовательности, не содержащие цифр. |
| 4 | Вывести на экран только те слова последовательности, в которых встречаются одинаковые буквы. |
| 5 | Вывести на экран только те слова последовательности, в которых первая буква слова встречается в этом слове еще раз. |
| 6 | После окончания ввода последовательности вывести на экран сначала все слова, содержащие только буквы, затем слова, содержащие только цифры, а потом слова, содержащие и буквы, и цифры. |

4) Выполнить задание по варианту:

|  |  |
| --- | --- |
| 1 | Вывести на экран ту же последовательность, заменив во всех словах первую букву соответствующей прописной буквой. |
| 2 | Вывести на экран количество символов в каждом слове исходной последовательности. |
| 3 | Вывести на экран ту же последовательность, удалив из всех слов заданный набор букв и (или) цифр. |
| 4 | Вывести на экран ту же последовательность, заменив во всех словах цифры на буквы латинского алфавита, номера которых в алфавите равны заменяемой цифре. |
| 5 | Вывести на экран ту же последовательность, переместив все цифры, содержащиеся в словах, в конец соответствующих слов. |
| 6 | Вывести все слова исходной последовательности на экран вертикально. |
| 7 | Вывести на экран  все слова последовательности в две или три колонки (в зависимости от количества слов) с выравниванием слов по правой границе колонки. |

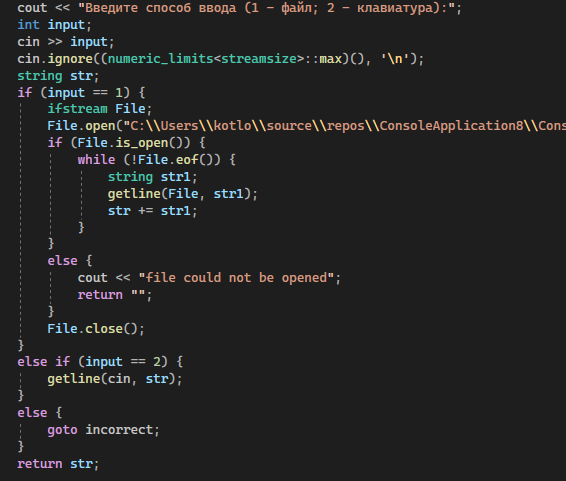
5)  Необходимо найти все подстроки, которую введёт пользователь в имеющейся строке. Реализуйте два алгоритма: первый алгоритма – Линейный поиск, а второй алгоритм согласно вашему номеру в списке. Четные номера должны реализовать алгоритм КМП, а нечетные – Бойера-Мура. (\*)

**Выполнение работы.**

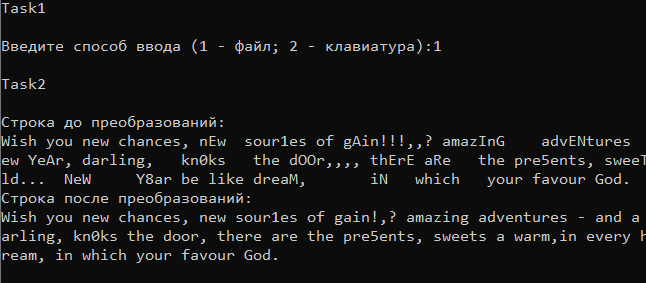
Код программы представлен в приложении А.

Описание кода и использованных алгоритмов.

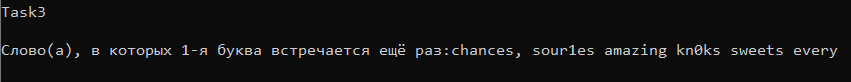
1. Ввод пользователем. Создание строки

****

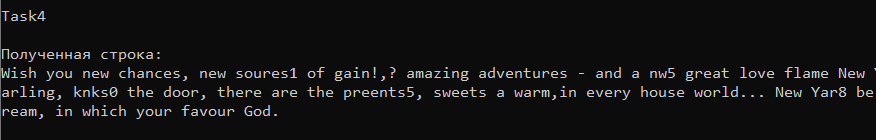
1. Редактирование строки

****

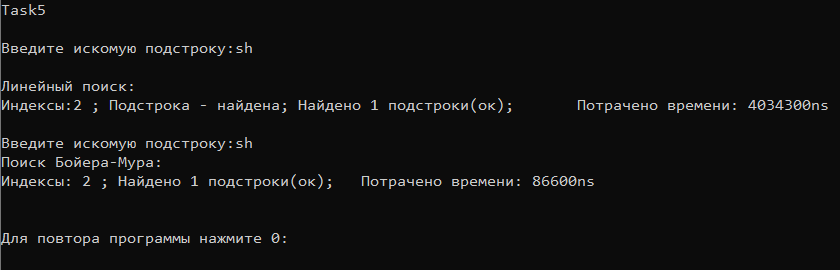
1. Вывести на экран только те слова последовательности, в которых первая буква слова встречается в этом слове еще раз.

****

1. Вывести на экран ту же последовательность, переместив все цифры, содержащиеся в словах, в конец соответствующих слов.

****

1. Поиск подстроки (Линейный алгоритм, алгоритм Бойера\_Мура).

****

**Выводы.**

В ходе работы были изучены текстовые строки, ввод, вывод и обработка текстовых строк; реализованы алгоритмы поиска подстроки в строке. В ходе работы была повторена работа с функциями и указателями.

Приложение А

рабочий код

#include <iostream>

#include <fstream>

#include <string>

#include <chrono>

#include <limits>

using namespace std;

//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.

string task1() {

incorrect:

cout << "Введите способ ввода (1 - файл; 2 - клавиатура):";

int input;

cin >> input;

cin.ignore((numeric\_limits<streamsize>::max)(), '\n');

string str;

if (input == 1) {

ifstream File;

File.open("C:\\Users\\kotlo\\source\\repos\\ConsoleApplication8\\ConsoleApplication8\\T.txt");

if (File.is\_open()) {

while (!File.eof()) {

string str1;

getline(File, str1);

str += str1;

}

}

else {

cout << "file could not be opened";

return "";

}

File.close();

}

else if (input == 2) {

getline(cin, str);

}

else {

goto incorrect;

}

return str;

}

//2. Необходимо отредактировать входной текст:

//·удалить лишние пробелы;

//·удалить лишние знаки препинания (под «лишними» подразумевается несколько подряд идущих знаков;

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

string task2(string str) {

string strRes = "";

cout << "Строка до преобразований:" << "\n" << str << "\n";

bool flag = true;

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

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

if (flag) {

continue;

}

else {

flag = true;

strRes += str[i];

}

}

else if (str[i] >= 'a' && str[i] <= 'z' || str[i] >= 'A' && str[i] <= 'Z' || str[i] >= '0' && str[i] <= '9') {

if (str[i] >= 'A' && str[i] <= 'Z' && !flag) {

strRes += str[i] + 32;

}

else {

strRes += str[i];

}

flag = false;

}

else {

flag = false;

if (str[i] == '.' && str[i + 1] == '.' && str[i + 2] == '.') {

i += 2;

strRes += "...";

continue;

}

else if (str[i] == '.') {

strRes += '.';

break;

}

if (str[i] != str[i - 1]) {

strRes += str[i];

}

}

}

cout << "Строка после преобразований:" << "\n" << strRes;

return strRes;

}

//3.5. Вывести на экран только те слова последовательности, в которых первая буква слова встречается в этом слове еще раз.

void task3(string str) {

cout << "Слово(а), в которых 1-я буква встречается ещё раз:";

bool flagSpace = false, flag = false;

char first = 0;

string str1 = "";

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

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

if (flag) {

cout << str1 << " ";

}

flagSpace = true;

flag = false;

str1 = "";

}

else {

if (flagSpace) {

flagSpace = false;

str1 += str[i];

first = str[i];

continue;

}

if (first == str[i]) {

flag = true;

}

str1 += str[i];

}

}

}

//4.5. Вывести на экран ту же последовательность, переместив все цифры, содержащиеся в словах, в конец соответствующих слов.

void task4(string str) {

cout << "Полученная строка: " << "\n";

string str1 = "";

bool flag = true;

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

if (str[i] >= '0' && str[i] <= '9') {

str1 += str[i];

str.erase(i, 1);

flag = true;

if (str[i] >= '0' && str[i] <= '9') {

i--;

}

}

else if ((str[i] == ' ' || !(str[i] >= 'a' && str[i] <= 'z' || str[i] >= 'A' && str[i] <= 'Z')) && flag) {

str.insert(i, str1);

i += str1.length() - 1;

str1 = "";

flag = false;

}

}

cout << str;

}

//5. Необходимо найти все подстроки, которую введёт пользователь в имеющейся строке. Реализуйте два алгоритма:

// первый алгоритма – Линейный поиск, а второй – Бойера-Мура.

void linesearch(string str)

{

string substr, strRes = "не найдена";

int count = 0;

cout << "Введите искомую подстроку:";

getline (cin, substr);

cout << "\nЛинейный поиск:" << "\n";

cout << "Индексы:";

string str1 = "";

auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();

for (int i = 0; i <= str.length() - substr.length(); i++) {

str1 = "";

for (int j = i; j < substr.length() + i; j++) {

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

str1 += str[j];

}

else {

continue;

}

}

if (substr == str1) {

cout << i << " ";

strRes = "найдена";

count += 1;

}

}

auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();

cout << "; Подстрока - " << strRes << "; Найдено " << count << " подстроки(ок);\t" << "Потрачено времени: " << end - start << "ns\n";

}

# define NO\_OF\_CHARS 256

void badCharHeuristic(string str, int size, int badchar[NO\_OF\_CHARS])

{

int i;

// Initialize all occurrences as -1

for (i = 0; i < NO\_OF\_CHARS; i++)

badchar[i] = -1;

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

badchar[(int)str[i]] = i;

}

void BoyerMoores(string str)

{

string substr;

cout << "Введите искомую подстроку:";

getline(cin, substr);

cout << "Поиск Бойера-Мура:\n";

int m = substr.size();

int n = str.size();

int count = 0;

int badchar[NO\_OF\_CHARS];

badCharHeuristic(substr, m, badchar);

int s = 0;

cout << "Индексы: ";

auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();

while (s <= (n - m))

{

int j = m-1;

while (j >= 0 && substr[j] == str[s + j])

j--;

if (j < 0)

{

cout << s << " ";

count += 1;

s += (s + m < n) ? m - badchar[str[s + m]] : 1;

}

else

s += max(1, j - badchar[str[s + j]]);

}

auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();

cout << "; Найдено " << count << " подстроки(ок);\t" << "Потрачено времени: " << end - start << "ns\n";

}

void newFile(string str) {

fstream File("C:\\Users\\kotlo\\source\\repos\\ConsoleApplication8\\ConsoleApplication8\\T\_new.txt", ios::out | ios::trunc);

if (File.is\_open()) {

File << str;

}

else {

cout << "file could not be opened";

}

}

int main4() {

setlocale(LC\_ALL, "RUSSIAN");

string str;

string substr;

//string sub;

//cout << "Введите подстроку, которую вы хотите найти в строке:";

//getline(cin, sub);

int input = 0;

while (!input)

{

cout << "\nTask1\n\n";

str = task1();

cout << "\nTask2\n\n";

str = task2(str);

cout << "\n";

cout << "\nTask3\n\n";

task3(str);

cout << "\n";

cout << "\nTask4\n\n";

task4(str);

cout << "\n";

cout << "\nTask5\n\n";

linesearch(str);

cout << "\n";

BoyerMoores(str);

cout << "\n";

cout << "\n";

cout << "Для повтора программы нажмите 0:";

cin >> input;

}

newFile(str);

return 0;

}