**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: **"Поиск подстроки в строке. Работа с текстовыми строками"**

.

|  |  |  |
| --- | --- | --- |
| Студент гр. 3372 |  | Поконечный А.О |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

* Основные теоретические положения.
* При работе со строками часто будет возникать потребность в поиске набора символа или слов (поиска подстроки в строке). При условии, что текст может быть крайне большим, хочется, чтобы алгоритм поиска подстроки работал быстро.
* Самый простой способ подстроки в строке – Линейный поиск – циклическое сравнение всех символов строки с подстрокой. Действительно, этот способ первый приходит в голову, но очевидно, что он будет самым долгим.
* На первых двух итерациях цикла сравниваемые буквы не будут совпадать. На третьей же итерации, совпал символ ‘L’, это означает, что теперь нужно сравнивать следующий символ подстроки со следующим символом строки. Видно, что символы отличаются, поэтому алгоритм продолжает свою работу. На четвертой же итерации подстрока была найдена.
* Если представить, что исходная строка непорядок больше и подстрока находится в конце строки (или вовсе отсутствует), то сразу видны минусы данного алгоритма.
* Одним из самых популярных алгоритмов, который работает быстрее, чем приведенный выше алгоритм, является алгоритм Кнута-Морриса-Пратта (КМП). Идея заключается в том, что не нужно проходить и сравнивать абсолютно все символы строки, если известны символы, которые есть и в строке, и в подстроке.
* Суть алгоритма: дана подстрока S и строка T. Требуется определить индекс, начиная с которого образец S содержится в строке T. Если S не содержится в T, необходимо вернуть индекс, который не может быть интерпретирован как позиция в строке.
* Хоть алгоритм и работает быстрее, по-прежнему необходимо сначала пройти всю строку, чтобы определить префиксы или суффиксы (вхождение (индексы) символов).
* Алгоритм Бойера-Мура в отличие от КМП полностью не зависим и не требует заранее проходить по строке. Этот алгоритм считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке.
* Преимущество этого алгоритма в том, что ценной некоторого количества предварительных вычислений над подстрокой (но не над исходной строкой, в которой ведётся поиск), 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 строки 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 подстроки в строку.
* Правила построения таблицы сдвигов:
* 1) Значение элемента таблицы равно удаленности соответствующего символа от конца шаблона (подстроки).
* 2) Если символ встречается более одного раза, то применятся значение, соответствующее символу, наиболее близкому к концу шаблона.
* 3) Если символ в конце шаблона встречается 1 раз, ему соответствует значение, равное длине образа; если более одного раза – значение, соответствующее символу, наиболее близкому к концу образа.
* 4) Для символов, отсутствующих в образе, применяется значение, равное длине шаблона.
* Сначала была построена таблица отступов и подсчитана длина подстроки. Затем начинается алгоритм поиска подстроки в строке. Сравнивает символ ‘L’ строки и ‘O’ подстроки. Элементы не совпадают, поэтому необходимо определить длину отступа. Символ ‘L’ присутствует в таблице отступа, длина отступа равняется 1. Подстрока смещается на 1 символ вперед. На следующей итерации подстрока найдена.
* Постановка задачи.
* Необходимо написать программу, которая:
* 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:
* 4) Выполнить задание по варианту:
* 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**.

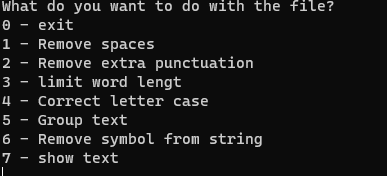
Часть кода

**Блок описания кода и использованных алгоритмов**

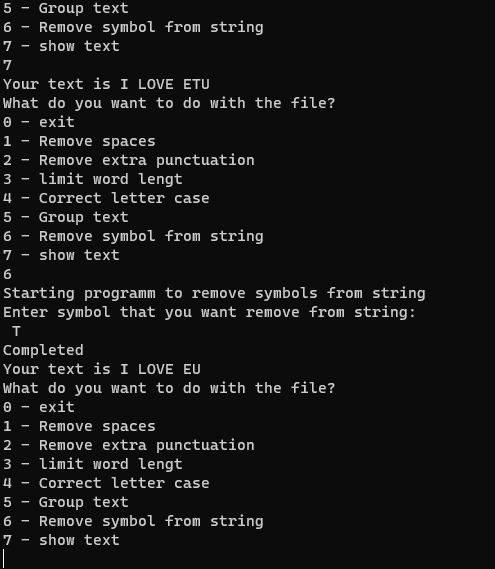
Использованы ссылки, switch, do, cout, cin, if. Это очень нужные конструкции, без которых бы данный код не заработал!!!

**Блок скриншотов работы программы**

Выбор задания



**Блок таблицы с тестовыми данными**



**Выводы.**

Выполнив данную работу мы поняли, как работать со строками!

Приложение А

рабочий код

#include <iostream>

#include <string>

#include <fstream>

#include <windows.h>

#include <unordered\_map>

#include <sstream>

#include <algorithm>

#include <cstring>

using namespace std;

const int maxWords = 50;

const int maxWordLength = 10;

void processString(const string& str, string& text, int &filein);

void removeExtraSpaces(string& text);

void limitWordsLength(string& text);

void removeExtraPunct(string& text);

void correctLetterCase(string& text);

void separateWords(const string& text);

void removeChars(string& text);

void removeChar(string& text);

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "");

string text;

string choice;

int filein = 0, whatyouwant = 0;

cout << "Choose what we will do (from file - enter file)/(from console - enter console)\n";

getline(cin, choice);

do {

if (!(filein)) {

processString(choice, text, filein);

}

cout<<"What do you want to do with the file?\n0 - exit\n1 - Remove spaces\n2 - Remove extra punctuation\n3 - limit word lengt\n4 - Correct letter case\n5 - Group text\n6 - Remove symbol from string\n7 - show text\n";

cin >> whatyouwant;

switch (whatyouwant) {

case 1: {

cout << "Removing spaces\n";

removeExtraSpaces(text);

cout << "Completed\n";

break;

}

case 2: {

cout << "Removing extra punctuation\n";

removeExtraPunct(text);

cout << "Completed\n";

break;

}

case 3: {

cout << "Limiting word lenght\n";

limitWordsLength(text);

cout << "Completed\n";

break;

}

case 4: {

cout << "Correcting letter case\n";

correctLetterCase(text);

cout << "Completed\n";

break;

}

case 5: {

cout << "Grouping starting\n";

limitWordsLength(text);

separateWords(text);

cout << "Completed\n";

break;

}

case 6: {

cout << "Starting programm to remove symbols from string\n";

removeChar(text);

cout << "Completed\n";

}

case 7: {

cout <<"Your text is "<< text << "\n";

break;

}

}

} while (whatyouwant);

return 0;

}

void processString(const string& str,string &text,int &filein ) {

unordered\_map<string, int> stringMap{

{"file", 1},

{"console", 2},

};

auto it = stringMap.find(str);

if (it != stringMap.end()) {

switch (it->second) {

case 1: {

cout << "You choosed file\n";

ifstream fin;

string path;

cout << "Enter the file name and extension, the file must be in txt format, saved in ANSI encoding:\n";

getline(cin, path);

fin.open(path);

if (!fin.is\_open()) {

cout << "Opening error\n";

}

else {

text = "";

string line;

while (getline(fin, line)) {

text += line + " ";

}

}

filein = 1;

fin.close();

break;

}

case 2: {

cout << "Enter the string: \n";

getline(cin, text);

filein = 1;

break;

}

default:

cout << "Unknown type\n";

}

}

}

void removeExtraSpaces(string& text) {

string output;

bool previousIsSpace = false;

for (char c : text) {

if (isspace(c)) {

if (!previousIsSpace) {

output += ' ';

}

previousIsSpace = true;

}

else {

output += c;

previousIsSpace = false;

}

}

output.erase(output.find\_last\_not\_of(" \n\r\t") + 1);

output.erase(0, output.find\_first\_not\_of(" \n\r\t"));

text = output;

}

void limitWordsLength(string& text) {

istringstream stream(text);

string word;

string\* words = new string[maxWords];

int wordCount = 0;

while (stream >> word) {

if (word.length() > maxWordLength) {

word = word.substr(0, maxWordLength);

}

if (wordCount < maxWords) {

words[wordCount] = word;

wordCount++;

}

else {

break;

}

}

text = "";

for (int i = 0; i < wordCount; i++) {

text += words[i];

if (i < wordCount - 1) {

text += " ";

}

if (i == (wordCount - 1)) {

text += ".";

}

}

delete[] words; // ПРОГРАММИРУЕМ КАК УМЕЕМ

}

void removeExtraPunct(string& text) {

string output;

bool previousIsPunctuation = false;

for (char c : text) {

if (ispunct(c) && !(c=='.')) {

if (previousIsPunctuation) {

}

else {

output += c;

previousIsPunctuation = true;

}

}

else {

output += c;

previousIsPunctuation = false;

}

}

text = output;

}

void correctLetterCase(string& text) {

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

if (isalpha(text[i]) && (i == 0 || text[i - 1] == ' ')) {

text[i] = toupper(text[i]);

}

else {

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

}

}

}

void separateWords(const string& text) {

string litters\_words, digit\_words, mix\_words;

string word;

istringstream stream(text);

while (stream >> word) {

if (all\_of(word.begin(), word.end(), isalpha)) {

litters\_words += word + " ";

}

else if (all\_of(word.begin(), word.end(), isdigit)) {

digit\_words += word + " ";

}

else {

mix\_words += word + " ";

}

}

cout << "Words only with litters " << litters\_words << endl;

cout << "Words only with digits " << digit\_words << endl;

cout << "Worlds mixed " << mix\_words << endl;

}

void removeChar(string& text) {

char input;

cout << "Enter symbol that you want remove from string:\n ";

cin >> input;

text.erase(remove(text.begin(), text.end(), input), text.end());

}