**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: Двумерные статические массивы. Указатели.

|  |  |  |
| --- | --- | --- |
| Студентка гр. 4372 |  | Хальметова Ю.Р. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

 Цель работы заключается в разработке программы, которая обрабатывает последовательность слов, введённых пользователем. Программа должна реализовывать следующие функции: удаление лишних пробелов между словами, устранение повторяющихся знаков препинания, корректировка регистра букв, а также добавление точки в конце текста, если это необходимо. Ввод данных может осуществляться как с клавиатуры, так и из файла, что предоставляет пользователю гибкость в выборе способа ввода.

**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’).

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.

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.

**Постановка задачи.**

Постановка задачи заключается в разработке программы, которая будет обрабатывать последовательность слов, вводимых пользователем. Программа должна выполнять следующие функции: удалять избыточные пробелы между словами, устранять повторяющиеся знаки препинания, корректировать регистр букв и добавлять точку в конце текста, если это необходимо. Ввод данных должен осуществляться как с клавиатуры, так и из файла, что обеспечит пользователю удобство и гибкость в выборе способа ввода.

**Выполнение работы.**

| Ввод пользователем и обработка данных | | Работа алгоритма и вывод на экран |
| --- | --- | --- |
| Меню | | |
| При запуске программы перед пользователем появляется, выбор задания. | | Меню: |
| 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. | | |
| При вводе пользователем корректного значения пункта меню пользователь выбирает способ ввода текста | | Запрашивает у пользователя способ ввода текста и выводит слова на экран |
| Редактирует входной текст | | |
| При запуске программы перед пользователем появляется, выбор задания. | | Удаляет лишние пробелы, лишние знаки препинания и исправляет регистр букв во входном тексте и выводит отредактированный текст на экран |
| Удаление из массива заданного набора букв или цифр | | |
| При запуске программы пользователь вводит набор символов, который будет удален из текста. | Удаляет из массива заданный набор символов и выводит на экран измененный текст | |
| Поиск всех подстрок, которую введет пользователь | | |
| Пользователь задает подстроку для поиска ее в тексте | Производит поиск при помощи алгоритма Бойера-Мура и выводит все подстроки, которую введет пользователь | |

**Выводы.**

Выводом работы является разработанная программа, которая обрабатывает последовательность слов, введённых пользователем. Программа успешно реализует следующие функции: удаление лишних пробелов между словами, устранение повторяющихся знаков препинания, корректировка регистра букв и добавление точки в конце текста, если это необходимо. Пользователь имеет возможность вводить данные как с клавиатуры, так и из файла, что обеспечивает гибкость в выборе способа ввода.

Приложение А

рабочий код

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

#include <string>

#include <cctype>

#include <regex>

#include <unistd.h>

using namespace std;

unordered\_map<char, int> createBadCharTable(const string& pattern) {

unordered\_map<char, int> badCharTable;

int length = pattern.length();

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

badCharTable[pattern[i]] = i;

}

return badCharTable;

}

vector<int> boyerMooreSearch(const string& text, const string& pattern) {

vector<int> positions;

int textLength = text.length();

int patternLength = pattern.length();

if (patternLength == 0 || textLength < patternLength) {

return positions;

}

auto badCharTable = createBadCharTable(pattern);

int shift = 0;

while (shift <= textLength - patternLength) {

int j = patternLength - 1;

while (j >= 0 && pattern[j] == text[shift + j]) {

j--;

}

if (j < 0) {

positions.push\_back(shift);

shift += (shift + patternLength < textLength) ?

patternLength - badCharTable[text[shift + patternLength]] : 1;

} else {

shift += max(1, j - badCharTable[text[shift + j]]);

}

}

return positions;

}

string removeCharacters(const string& word, const string& charsToRemove) {

string result;

for (char ch : word) {

if (charsToRemove.find(ch) == string::npos) {

result += ch;

}

}

return result;

}

bool containsDigits(const string& word) {

for (char ch : word) {

if (isdigit(ch)) {

return true;

}

}

return false;

}

string toLowerCase(const string& input) {

stringstream ss(input);

string word;

string result;

while (ss >> word) {

for (char& ch : word) {

ch = tolower(ch);

}

if (!result.empty()) {

result += " ";

}

result += word;

}

return result;

}

string removeExtraPunctuation(const string& input) {

stringstream ss(input);

string result;

char lastChar = '0';

int dotCount = 0;

string punctuation = "!#$%&'()\*+,-./:;<=>?@[^\_{|}~";

for (char currentChar : input) {

if (punctuation.find(currentChar) != string::npos) {

if (currentChar == '.') {

dotCount++;

if (dotCount <= 3) {

result += currentChar;

}

continue;

} else {

dotCount = 0;

if (currentChar == lastChar) {

continue;

}

}

} else {

dotCount = 0;

}

result += currentChar;

lastChar = currentChar;

}

return result;

}

string removeExtraSpaces(const string& input) {

stringstream ss(input);

string word;

string result;

while (ss >> word) {

if (!result.empty()) {

result += " ";

}

result += word;

}

return result;

}

bool isValidInput(const string& input, vector<string>& words) {

if (input.empty() || input.back() != '.') {

cout << "Ввод должен заканчиваться точкой." << endl;

return false;

}

string trimmedInput = input.substr(0, input.size() - 1);

istringstream iss(trimmedInput);

string word;

while (iss >> word) {

if (word.length() < 1 || word.length() > 10) {

cout << "Каждое слово должно содержать от 1 до 10 символов." << endl;

return false;

}

words.push\_back(word);

}

if (words.size() < 1 || words.size() > 50) {

cout << "Количество слов должно быть от 1 до 50." << endl;

return false;

}

return true;

}

string inputFromKeyboard() {

string userInput;

cout << "Введите последовательность слов (завершите точкой): ";

getline(cin, userInput);

return userInput;

}

string inputFromFile() {

string filename;

cout << "Введите имя файла: ";

cin >> filename;

ifstream file(filename);

if (!file) {

cout << "Файл не найден. Попробуйте снова." << endl;

return inputFromFile();

}

string content;

string line;

while (getline(file, line)) {

content += line;

}

return content;

}

int main() {

int choice;

vector<string> words;

string userInput;

string editedInput;

bool flag = true;

string cleanedInput;

system("clear");

while (flag) {

cout << endl;

cout << "Введите номер задания(1 - 4): ";

cin >> choice;

cout << endl;

cin.ignore();

switch (choice) {

case 1: {

int user\_choice;

cout << "Выберите способ ввода текста(1 - с клавиатуры, 2 - из файла): ";

cin >> user\_choice;

cin.ignore();

if (user\_choice == 1) {

userInput = inputFromKeyboard();

} else if (user\_choice == 2) {

userInput = inputFromFile();

} else {

cout << "Неверный выбор. Пожалуйста, попробуйте снова." << endl;

return 0;

}

if (isValidInput(userInput, words)) {

cout << "Введенные слова: ";

for (const auto& word : words) {

cout << word << " ";

}

cout << endl;

} else {

cout << "Нет корректных слов для вывода." << endl;

}

break;

}

case 2: {

editedInput = removeExtraSpaces(userInput);

editedInput = removeExtraPunctuation(editedInput);

editedInput = toLowerCase(editedInput);

cout << "Отредактированный текст: " << editedInput << endl;

break;

}

case 3: {

string charsToRemove;

cout << "Введите набор символов для удаления: ";

getline(cin, charsToRemove);

cout << "Введенные слова (без цифр и с удаленными символами): ";

vector<string> editedWords;

istringstream iss(editedInput);

string word;

while (iss >> word) {

editedWords.push\_back(word);

}

for (const auto& word : editedWords) {

if (!containsDigits(word)) {

string cleanedWord = removeCharacters(word, charsToRemove);

cleanedInput += cleanedWord + " ";

cout << cleanedWord << " ";

}

}

cout << endl;

break;

}

case 4:{

string substring;

cout << "Введите подстроку для поиска: ";

getline(cin, substring);

vector<int> bmPositions = boyerMooreSearch(cleanedInput, substring);

cout << "Результаты поиска Бойера-Мура: ";

for (int pos : bmPositions) {

cout << pos << " ";

}

cout << endl;

break;

}

default:

cout << "Неверный номер задания." << endl;

break;

}

}

return 0;

}