**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: Обработка текста и чтение данных из файла

|  |  |  |
| --- | --- | --- |
| Студент гр. 4372 |  | Демьяненко И.А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Разработать программу для обработки текстовых данных, которая позволяет пользователю вводить текст (с клавиатуры или из файла), анализировать его и исправлять опечатки; выполнять поиск подстроки в строке.

**Основные теоретические положения.**

Библиотека <fstream> предоставляет классы для работы с файловыми потоками.

1. Классы для работы с файлами

* std::ifstream: Класс для чтения данных из файлов. Он позволяет открывать файлы для чтения.
* std::ofstream: Класс для записи данных в файлы. Он используется для создания и записи в файлы.
* std::fstream: Класс, который объединяет функциональность ifstream и ofstream, позволяя как читать, так и записывать данные в файл.

1. Открытие файла

Файл открывается с помощью метода open(), который принимает имя файла и режим открытия (например, std::ios::in для чтения, std::ios::out для записи). Также можно указать режимы при создании объекта, например: std::ifstream file("example.txt");

1. Проверка успешности открытия файла

После открытия файла важно проверить, был ли файл успешно открыт, используя метод is\_open() или проверяя состояние потока:

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

std::cerr << "Ошибка открытия файла!" << std::endl;

}

1. Чтение данных

Данные из файла можно читать с помощью метода getline() для чтения строк целиком, включая пробелы:

std::string line;

std::getline(file, line);

1. Режимы открытия файла

При открытии файла можно использовать различные режимы, такие как:

* std::ios::in: Открытие для чтения.
* std::ios::out: Открытие для записи (файл будет создан, если он не существует).
* std::ios::app: Открытие для добавления данных в конец файла.
* std::ios::binary: Открытие в бинарном режиме (для работы с бинарными файлами).

Класс std::string в C++ представляет собой удобный и мощный инструмент для работы с текстовыми строками. Он является частью стандартной библиотеки и предоставляет множество функций для манипуляции строками. Класс std::string позволяет динамически управлять памятью, что делает его более гибким по сравнению с обычными массивами символов. Строки могут изменять свою длину в процессе выполнения программы, что упрощает работу с текстовыми данными.

Основные операции, которые можно выполнять с объектами класса std::string, включают создание строк, их копирование, конкатенацию (объединение), сравнение и поиск подстрок. Для создания строки можно использовать конструкторы, которые принимают различные параметры, такие как литералы строк, другие строки или количество повторений символа.

Класс std::string поддерживает оператор + для конкатенации строк, а также оператор +=, который позволяет добавлять к существующей строке. Сравнение строк можно выполнять с помощью операторов ==, !=, <, >, и других, что позволяет легко проверять равенство и порядок строк.

Методы класса std::string предоставляют возможность получения длины строки с помощью метода length() или size(), доступа к отдельным символам через оператор индексирования [], а также извлечения подстрок с помощью метода substr(). Для поиска подстрок можно использовать метод find(), который возвращает индекс первого вхождения подстроки или std::string::npos, если подстрока не найдена.

**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 алфавитном порядке;
4. Вывести на экран количество символов в каждом слове исходной последовательности;
5. Реализовать два алгоритма поиска подстроки в строке: линейный алгоритм и КМП.

**Выполнение работы.**

Код программы представлен в приложении А.

| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| --- | --- |
| Загрузка текста в программу | |
| При запуске программы пользователь выбирает вариант загрузки текста в программу. Доступные опции: с клавиатуры или из файла. | Выбор ввода текста из файла:    Ввод текста с клавиатуры для последующей обработки: |
| Обработка текста | |
| Пользователь видит исправленный текст, отсортированные слова, подсчёт количества букв в каждом слове. Ему предлагается возможность начать поиск подстроки в строке. | В тексте убираются лишние пробелы, знаки препинания, исправляется регистр букв в словах. Выводится список слов в алфавитном порядке и подсчитывается количество букв в каждом слове: |
| Поиск подстроки в строке | |
| Пользователь вводит подстроку для поиска | Программа обрабатывает ввод пользователя и выводит количество найденных в строке подстрок при помощи алгоритмов линейного поиска и КМП: |

**Выводы.**

Разработанная программа предназначена для обработки текстовых данных, позволяя вводить текст с клавиатуры или из файла, выделять слова и знаки препинания, сортировать слова в алфавитном порядке и выводить количество символов в каждом слове. Были изучены два метода поиска подстрок: линейный поиск и алгоритм Кнута-Морриса-Пратта (КМП).

Приложение А

рабочий код

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <cstring>

using namespace std;

const int N = 512;

char line[N];

char removeChars[N];

char toSearch[N];

void task1() {

cout << "Выберите: Ручной ввод (0) или Ввод с файла (1)" << "\n";

int choice;

cin >> choice;

cin.ignore();

ifstream file("text.txt");

switch (choice)

{

case 0:

cout << "Введите строку" << "\n";

cin.getline(line , N);

break;

case 1:

cout << "Чтение файла text.txt" << endl;

file.getline(line, N);

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

cout << "Не удалось открыть файл, завершаю выполнение.";

}

file.close();

break;

default:

cout << "Некорректный ввод" << "\n";

break;

}

}

void task2() {

int countDot = 0, i = 0, j = 0;

i = 0, j = 0;

char prevSymb;

while (line[i] != '\0') {

if (ispunct(line[i]) != 0) {

if (line[i] == '.' && prevSymb == '.') {

countDot++;

if (countDot <= 2) {

line[j++] = line[i];

}

}

else if (ispunct(prevSymb) != 0) {

line[j++] = ' ';

i++;

}

else

{

line[j++] = line[i];

}

}

else {

line[j++] = line[i];

countDot = 0;

}

prevSymb = line[i];

i++;

}

line[j] = '\0';

bool isSpace = false;

i = 0, j = 0;

while (line[i] != '\0') {

if (line[i] != ' ') {

line[j++] = line[i];

isSpace = false;

}

else if (isSpace == false) {

line[j++] = ' ';

isSpace = true;

}

i++;

}

line[j] = '\0';

i = 0;

prevSymb = ' ';

while (line[i] != '\0') {

if (isalpha(line[i]) && prevSymb != ' ') {

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

}

prevSymb = line[i];

i++;

}

}

bool findDup(char\* word) {

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

for (int j = i + 1; word[j] != '\0'; ++j) {

if (word[i] == word[j]) {

return true;

}

}

}

return false;

}

void task3() {

int i = 0, j = 0;

char word[20] = {};

while (line[i] != '\0') {

if (line[i] == ' ' || ispunct(line[i]) != 0) {

if (findDup(word) == true) {

cout << word << " ";

for (int k = 0; word[k] != '\0'; k++) {

word[k] = 0;

}

j = 0;

i++;

}

else {

for (int k = 0; word[k] != '\0'; k++) {

word[k] = 0;

}

j = 0;

i++;

}

}

word[j] = line[i];

j++;

i++;

}

}

void task4() {

cin.ignore();

cout << "Введите символы для удаления: ";

cin.getline(removeChars, N);

int i = 0, j = 0;

while (line[i] != '\0') {

bool toRemove = false;

for (int k = 0; removeChars[k] != '\0'; k++) {

if (line[i] == removeChars[k]) {

toRemove = true;

break;

}

}

if (toRemove == false) {

line[j++] = line[i];

}

i++;

}

line[j] = '\0';

}

void linearSearch(char\* line, char\* pattern) {

int lenLine = strlen(line);

int lenPat = strlen(pattern);

int count = 0;

for (int i = 0; i <= lenLine - lenPat; ++i) {

int j;

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

if (line[i + j] != pattern[j]) {

break;

}

}

if (j == lenPat) {

cout << "Индекс начала подстроки: " << i << endl;

count++;

}

}

if (count == 0) {

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

}

}

vector<int> computeLps(string pattern) {

int n = pattern.length();

vector<int> lps(n, 0);

int len = 0;

int i = 1;

while (i < n) {

if (pattern[i] == pattern[len]) {

len++;

lps[i] = len;

i++;

}

else {

if (len != 0) {

len = lps[len - 1];

}

else {

lps[i] = 0;

i++;

}

}

}

return lps;

}

vector<int> kmp(string text, string pattern) {

vector<int> lps = computeLps(pattern);

vector<int> occurrences;

int n = text.length();

int m = pattern.length();

int i = 0;

int j = 0;

while (i < n) {

if (text[i] == pattern[j]) {

i++;

j++;

}

if (j == m) {

occurrences.push\_back(i - j);

j = lps[j - 1];

}

else if (i < n && text[i] != pattern[j]) {

if (j != 0) {

j = lps[j - 1];

}

else {

i++;

}

}

}

return occurrences;

}

void task5() {

cin.ignore();

cout << "Введите строку, которую хотите найти: ";

cin.getline(toSearch, N);

cout << "Выберите способ поиска: Линейный поиск(0) или Алгоритм КМП(1)";

int choice;

cin >> choice;

vector<int> occurrences = kmp(line, toSearch);

switch (choice)

{

case 0:

linearSearch(line, toSearch);

break;

case 1:

if (occurrences.empty()) {

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

}

else {

cout << "Индекс начала подстроки: ";

for (int i : occurrences) {

cout << i << " ";

}

cout << endl;

}

break;

default:

cout << "Некорректный ввод" << "\n";

break;

}

}

int main()

{

setlocale(0, "");

int choosetask;

do {

cout << "Выберите задание: ";

cin >> choosetask;

switch (choosetask)

{

case 1:

task1();

cout << line << "\n" << endl;

break;

case 2:

task2();

cout << line << "\n" << endl;

break;

case 3:

task3();

cout << "\n" << endl;

break;

case 4:

task4();

cout << line << "\n" << endl;

break;

case 5:

task5();

break;

default:

cout << "Некорректный ввод" << "\n" << endl;

break;

}

} while (choosetask != 0);

}