**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 |  | Барабанов И. В. |
| 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 <algorithm>  
#include <iostream>  
#include <string>  
#include <fstream>  
#include <limits>  
#include <vector>  
  
using namespace std;  
  
string words[50], lower\_words[50], punctuation[50];  
  
bool is\_letter\_or\_number(const char ch) {  
 return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0' && ch <= '9');  
}  
  
int linear\_search\_substring(const string &text, const string &pattern) {  
 const size\_t n = text.length();  
 const size\_t m = pattern.length();  
  
 for (int i = 0; i <= n - m; i++) {  
 int j;  
 for (j = 0; j < m; j++) {  
 if (tolower(text[i + j]) != tolower(pattern[j]))  
 break;  
 }  
  
 if (j == m)  
 return i;  
 }  
  
 return -1;  
}  
  
int search\_substrings\_count(string text, const string &pattern) {  
 int count = 0, index;  
 do {  
 index = linear\_search\_substring(text, pattern);  
 if (index != -1) {  
 count++;  
 text = text.substr(index + 1, text.length() - index - 1);  
 }  
 } while (index != -1);  
 return count;  
}  
  
vector<int> compute\_lps(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 = compute\_lps(pattern);  
 vector<int> occurrences;  
 int n = text.length();  
 int m = pattern.length();  
 int i = 0;  
 int j = 0;  
  
 while (i < n) {  
 if (tolower(text[i]) == tolower(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;  
}  
  
  
int main() {  
 cout << "Выберите способ ввода текста:" << endl;  
 cout << "1. С клавиатуры" << endl;  
 cout << "2. Из файла" << endl;  
 int ans = -1;  
 do {  
 cout << "Ваш выбор: ";  
 cin >> ans;  
 } while (ans != 1 && ans != 2);  
  
 cin.ignore(numeric\_limits<streamsize>::*max*(), '\n');  
  
  
 string text;  
 if (ans == 1) {  
 cout << "Введите текст" << endl;  
 getline(cin, text);  
 } else {  
 cout << "Введите путь до файла с текстом: ";  
 string path;  
 getline(cin, path);  
 fstream file(path, ios\_base::*in*);  
 if (!file.is\_open()) {  
 cout << "Не удалось открыть файл, завершаю выполнение.";  
 return 1;  
 }  
 getline(file, text);  
 file.close();  
 }  
  
 int start = 0, ind1 = 0, ind2 = 0;  
 for (int i = 0; i < text.length() - 1; i++) {  
 if (is\_letter\_or\_number(text[i]) && !is\_letter\_or\_number(text[i + 1])) {  
 words[ind1++] = text.substr(start, i - start + 1);  
 start = i + 1;  
 } else if (!is\_letter\_or\_number(text[i]) && is\_letter\_or\_number(text[i + 1])) {  
 punctuation[ind2++] = text.substr(start, i - start + 1);  
 start = i + 1;  
 }  
 }  
 punctuation[ind2++] = ".";  
  
 for (int i = 0; i < ind1; i++) {  
 for (int j = 1; j < words[i].length(); j++) {  
 words[i][j] = tolower(words[i][j]);  
 }  
 }  
  
  
 for (int i = 0; i < ind2; i++) {  
 string buffer;  
 buffer.push\_back(punctuation[i][0]);  
 for (int j = 1; j < punctuation[i].length(); j++) {  
 if (buffer.back() == '.' &&  
 j + 1 < punctuation[i].length() &&  
 punctuation[i][j] == '.' &&  
 punctuation[i][j + 1] == '.') {  
 buffer.push\_back('.');  
 buffer.push\_back('.');  
 j++;  
 continue;  
 }  
  
 if (buffer.back() != punctuation[i][j]) {  
 buffer.push\_back(punctuation[i][j]);  
 }  
 }  
 punctuation[i] = buffer;  
 }  
 for (int i = 0; i < ind1; i++) {  
 cout << words[i] << punctuation[i];  
 lower\_words[i] = words[i];  
 lower\_words[i][0] = tolower(lower\_words[i][0]);  
 }  
  
 cout << endl;  
 sort(lower\_words, lower\_words + ind1);  
 cout << "Задание 3. Вариант 2. Слова последовательности в алфавитном порядке." << endl;  
 for (int i = 0; i < ind1; i++) {  
 cout << lower\_words[i] << " ";  
 }  
 cout << endl;  
 cout << "Задание 4. Вариант 2. Вывести на экран количество символов в каждом слове исходной последовательности." <<  
 endl;  
 for (int i = 0; i < ind1; i++) {  
 cout << words[i] << " - " << words[i].length() << " || ";  
 }  
 cout << endl;  
 cout << "Задание 5. Поиск подстроки в строке методом линейного поиска и КМП." << endl;  
 string long\_line;  
 for (int i = 0; i < ind1; i++) {  
 long\_line += words[i] + punctuation[i];  
 }  
 string pattern;  
 cout << "Введите подстроку для поиска: ";  
 getline(cin, pattern);  
 int result = search\_substrings\_count(long\_line, pattern);  
 auto kmp\_result = kmp(long\_line, pattern);  
 cout << "Найдено " << result << " подстрок." << endl;  
 cout << "Найдено " << kmp\_result.size() << " подстрок." << endl;  
 return 0;  
}