Объяснение проделанной работы

После поставленной задачи

Написать программу, которая считывает текстовый файл и выводит на

экран топ-10 самых часто встречающихся слов в этом файле.

Для решения задачи выбрано использовать коллекцию Map, где ключом будет слово, а значением - количество его повторений в файле.

use std::collections::HashMap;  
use std::fs::File;  
use std::io::{self, BufRead};  
use std::path::Path;  
  
fn main() -> io::Result<()> {  
  
 let input\_path = "/Users/admin/Pr1/text/BednayaLiza.txt";  
  
 let input\_file = File::*open*(input\_path)?;  
 let reader = io::BufReader::*new*(input\_file);  
  
 let mut word\_count: HashMap<String, usize> = HashMap::*new*();  
  
 // Читаем файл построчно  
 for line in reader.lines() {  
 if let *Ok*(line) = line {  
   
 for word in line.split\_whitespace() {  
 let word = word.to\_lowercase();  
 \*word\_count.entry(word).or\_insert(0) += 1;  
 }  
 }  
 }  
  
   
 let mut word\_count\_vec: Vec<\_> = word\_count.iter().collect();  
 word\_count\_vec.sort\_by(|a, b| b.1.cmp(a.1));   
  
  
 println!("Топ-10 самых часто встречающихся слов:");  
 for (word, count) in word\_count\_vec.iter().take(10) {  
 println!("{}: {}", word, count);  
 }  
  
 *Ok*(())  
}

Листинг кода задания 1.

use std::collections::HashMap;

use std::fs::File;

use std::io::{self, BufRead};

use std::path::Path;

• std::collections::HashMap: Импортируем HashMap, который представляет собой ассоциативный массив (или словарь), где ключи и значения могут быть произвольных типов. В нашем случае ключом будет слово (строка), а значением — количество его вхождений.

• std::fs::File: Импортируем структуру File, которая позволяет работать с файлами (открывать, читать, записывать).

• std::io::{self, BufRead}: Импортируем модуль ввода-вывода. BufRead позволяет читать данные из буферизованных источников, таких как файлы.

• std::path::Path: Импортируем Path, который используется для работы с путями к файлам (хотя в данном коде он не используется).

**Открытие файла**

let input\_path = "/Users/admin/Pr1/text/BednayaLiza.txt";

let input\_file = File::open(input\_path)?;

let reader = io::BufReader::new(input\_file);

1. **Указание пути к файлу**: Переменная input\_path содержит путь к текстовому файлу, который мы хотим прочитать.

2. **Открытие файла**: File::open(input\_path)? открывает файл для чтения. Если файл не удается открыть (например, если он не существует), программа завершится с ошибкой.

3. **Создание буферизованного читателя**: io::BufReader::new(input\_file) создает буферизованный читатель, который позволяет более эффективно читать данные из файла построчно.

**Подсчет слов**

let mut word\_count: HashMap<String, usize> = HashMap::new();

• Создаем новый пустой HashMap, который будет хранить слова и их частоту. Ключи — это строки (String), а значения — целые числа (usize), представляющие количество вхождений каждого слова.

**Чтение файла построчно**

for line in reader.lines() {

if let Ok(line) = line {

for word in line.split\_whitespace() {

let word = word.to\_lowercase();

\*word\_count.entry(word).or\_insert(0) += 1;

}

}

}

1. **Цикл по строкам**: for line in reader.lines() проходит по всем строкам в файле.

2. **Обработка результата чтения**: if let Ok(line) = line проверяет, успешно ли считана строка. Если произошла ошибка, строка игнорируется.

3. **Разделение строки на слова**: line.split\_whitespace() разбивает строку на слова, игнорируя пробелы и другие разделители.

4. **Приведение к нижнему регистру**: word.to\_lowercase() приводит слово к нижнему регистру для учета регистра.

5. **Обновление счетчика**:

word\_count.entry(word).or\_insert(0) проверяет, есть ли уже это слово в HashMap. Если нет, оно добавляется со значением 0.

**Сортировка слов по частоте**

let mut word\_count\_vec: Vec<\_> = word\_count.iter().collect();

word\_count\_vec.sort\_by(|a, b| b.1.cmp(a.1));

1. **Преобразование вектора**: word\_count.iter().collect() создает вектор пар (слово, частота) из элементов HashMap.

2. **Сортировка**: sort\_by(|a, b| b.1.cmp(a.1)) сортирует вектор по убыванию частоты (по второму элементу пары).

**Вывод результата**

println!("Топ-10 самых часто встречающихся слов:");

for (word, count) in word\_count\_vec.iter().take(10) {

println!("{}: {}", word, count);

}

**Вывод заголовка**: Печатаем заголовок для списка слов.

**Цикл по топ-10 словам**: Используем .iter().take(10) для перебора первых 10 элементов отсортированного вектора и выводим каждое слово и его частоту

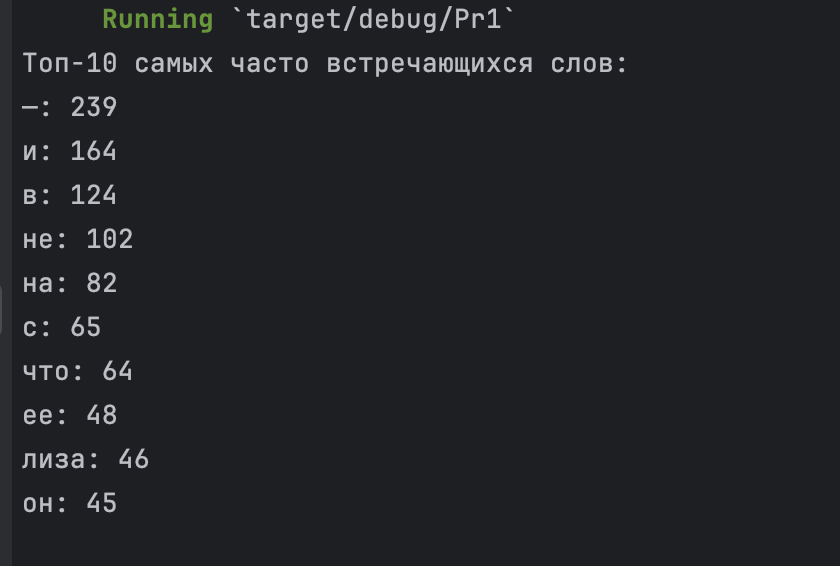


Рис 2 вывод результата кода

Мы видим ошибку. Программа выводит не только слова, но и разные символы такие как знаки препинания и тд.

Нам необходимо исправить это, проведем рефакторинг нашего кода

Вместо части с простым чтением файла

for line in reader.lines() {  
 if let *Ok*(line) = line {  
   
 for word in line.split\_whitespace() {  
 let word = word.to\_lowercase();  
 \*word\_count.entry(word).or\_insert(0) += 1;  
 }  
 }  
 }

Добавим проверку на символы не являющиеся словами

for word in line.split\_whitespace() {  
 // Удаляем знаки препинания, включая дефисы  
 let cleaned\_word = word  
 .replace(&['-','—', '.', ',', ';', '!', '?', ':'][..], "")  
 .to\_lowercase();  
  
 if !cleaned\_word.is\_empty() {  
 \*word\_count.entry(cleaned\_word).or\_insert(0) += 1;  
 }  
}

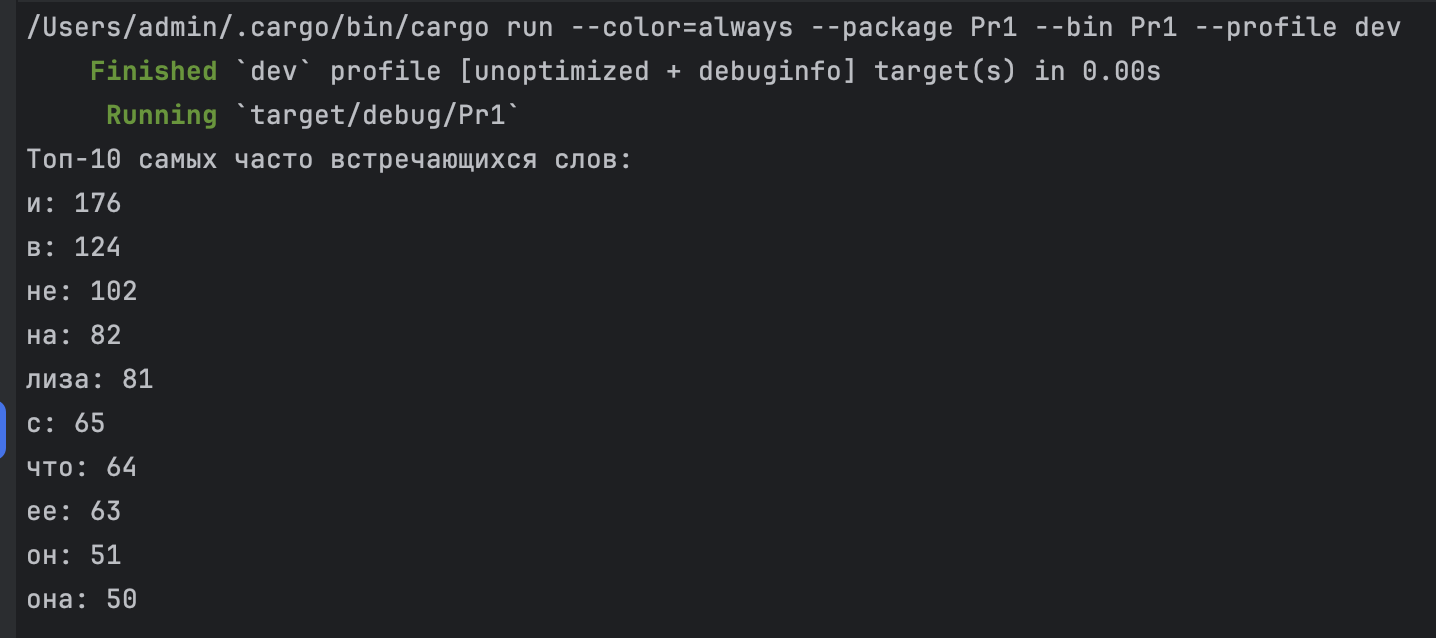


Рис 3 вывод программы после исправления ошибки

Выводы о проделанной работе

В ходе выполнения задачи по разработке программы для подсчета частоты слов в текстовом файле были достигнуты следующие результаты:

1. **Выбор структуры данных**:

2. **Чтение файла**:

3. **Обработка текста**:

4. **Подсчет частоты**:

5. **Сортировка и вывод результатов**

Разработанная программа эффективно решает поставленную задачу — подсчет частоты слов в текстовом файле. Использование структуры данных HashMap обеспечило быструю и удобную работу с данными, а реализация обработки текста позволила получить корректные результаты. Программа может быть доработана для поддержки дополнительных функций, таких как игнорирование стоп-слов или учет различных форм слов (например, единственного и множественного числа).