Software Requirements Specification

for

Word Frequency Counter

Version 0.1

Prepared by

Group – 02

Table of Contents ii Revision History ii 1. Introduction 1

WORD FREQUENCY COUNTER

1. Introduction

The Word Frequency Counter is a command-line based application developed to analyze multiple text files and determine how many times each unique word appears. The system is designed to process large amounts of textual data efficiently by using multithreading and hashing techniques. The application ignores predefined common words, treats words in a case-insensitive manner, and separates invalid words from valid inputs during processing.

This application demonstrates important operating system and data structure concepts such as file handling, POSIX threads, synchronization using mutex locks, and hash table implementation. Each input file is processed by a separate child thread, and all threads update a shared hash table to store the frequency of words. At the end of execution, the application displays the list of unique words along with their frequency of occurrence.

The system also provides flexibility to users by allowing them to modify the list of common words through an external file (commonwords.txt) without changing the program code. Any invalid words (such as those containing digits or special characters) are recorded separately in invalidwords.txt and are not included in the frequency calculation.

1.1 Purpose

The purpose of this document is to clearly define the requirements and expected behavior of the Word Frequency Counter application. This application is designed to analyze multiple text files and determine how many times each unique word appears, while excluding predefined common words and handling invalid inputs appropriately. This document serves as a reference for
developers, reviewers, and evaluators to understand the system's objectives and

functionality. 1.2 Document Conventions

This document follows a structured requirement format where each functional requirement is labeled as WFC01 to WFC09 (Word Frequency requirements). The term application refers to the Word Frequency Counter program. Mandatory requirements are those that must be implemented, while optional requirements indicate additional enhancements.

1.3 Intended Audience and Reading Suggestions

This document is intended for: Developers who will design and implement the application Evaluators and instructors reviewing the project Students and learners studying concepts such as multithreading, hashing, and file processing Readers are encouraged to first go through the
Introduction section for context, followed by Functional Requirements for detailed system behavior.

1.4 Project Scope

The scope of the project is limited to building a command-line application that processes multiple text files provided as input, counts the frequency of valid words using a hash table, ignores common words listed in a separate file, handles invalid words by storing them separately, and displays the final word frequency results. The system focuses on demonstrating file handling, data structures, and multithreading concepts in C.

1.5 References

- https://pubs.opengroup.org/onlinepubs/9699919799/
- https://www.gnu.org/software/libc/manual/
- https://man7.org/linux/man-pages/

2. Overall Description

2.1 Product Perspective

The Word Frequency Counter is a standalone, command-line based application designed to analyze multiple text files and compute the frequency of words appearing across them. The system operates as a single-user application and does not depend on any external systems or databases.

The application follows a multi-threaded client-side processing model, where each input text file is handled by a separate child thread. All threads share common resources such as a centralized hash table for word frequency storage and an output file for invalid words. Synchronization mechanisms are used to ensure data consistency and thread safety.

The system is implemented using object-oriented programming principles in C++ and is intended to run on a Linux-based environment.

2.2 Product Features

The main features of the Word Frequency Counter application are as follows:

- The application accepts multiple text files as input through command-line arguments
- Each input file is processed concurrently using POSIX threads
- Word comparison is performed in a case-insensitive manner
- Common words specified by the user are excluded from frequency calculation
- Invalid words containing digits or special characters are detected and discarded
- Invalid words are logged into a separate output file for reference
- A shared hash table is used to store and update word frequencies
- The final output displays unique words along with their total frequencies
- Optionally, different case forms of the same word can be displayed

2.3 Operating Environment

The operating environment required for the Word Frequency Counter application is as follows:

- Operating System: Linux (Ubuntu or equivalent)
- Kernel Version: Linux Kernel 5.x or above
- Programming Language: C++
- Compiler: GNU g++
- Threading Library: POSIX Threads (pthread)
- Interface Type: Command Line Interface (CLI)

2.4 Design and Implementation Constraints

The design and implementation of the system are subject to the following constraints:

- The application must be developed using the C++ programming language
- POSIX thread library must be used for multi-threaded execution
- Shared resources must be synchronized using mutex locks
- Standard Template Library (STL) must be used for data storage
- The application must be executed in a Linux environment
- A Makefile must be used to build the application using a two-step compilation process

2.5 User Documentation

User documentation for the Word Frequency Counter application includes:

- A user manual describing command-line usage and input format
- A README file explaining build steps, execution procedure, and directory structure
- Inline comments within the source code to assist developers and maintainers
- The project documentation will be made available along with the source code repository.

2.6 Assumptions and Dependencies

The assumptions and dependencies relevant to the system are as follows: Assumptions

- Input files are plain text files
- Users provide valid file names as command-line arguments
- The common words file is available and editable by the user
- The system is executed in a Linux terminal environment

Dependencies

- Availability of POSIX thread support
- GNU compiler tools must be installed
- File system permissions must allow file read and write operations

3.1 System Features

The following section defines the system features of the Word Frequency Counter (WFC) application, aligned to the formatting, rigor, and decomposition used in the WFC SRS.

3.1.1 Description and Priority:

- The system shall accept one or more input text files and process them to analyze word occurrences.
- Priority: Mandatory

3.1.2 Stimulus / Response Sequences:

- User provides one or more text files as input.
- System validates file existence and readability.
- Each file is read line by line for processing.

3.1.3 Functional Requirements:

WFC_01: The application shall count the frequency of words from multiple text files.

The system will accept multiple text files as input and analyze the content of each file to determine how many times each word appears. The frequency calculation will be

performed across all input files combined. Words identified as common words will not be included in the counting process. The goal of this requirement is to accurately measure the occurrence of meaningful words in large text data.

WFC_02: The application shall accept multiple filenames through command line arguments.

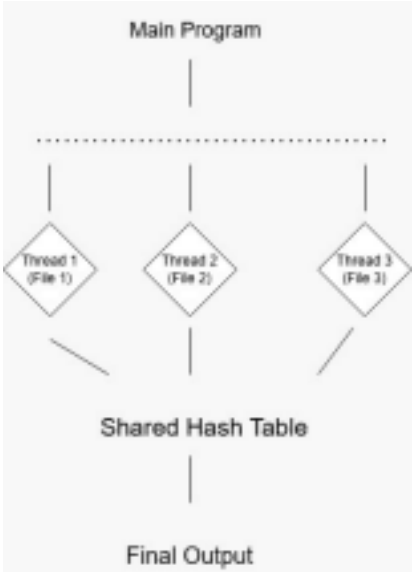
The user will execute the program by passing the names of text files as command line parameters. The application must be capable of reading and processing all provided files without requiring any modification to the program code. This allows flexibility in processing varying numbers of input files.

Example:

./wordfreq file1.txt file2.txt file3.txt

WFC_03: The system shall use a hash table to store word frequencies and perform case insensitive comparison.

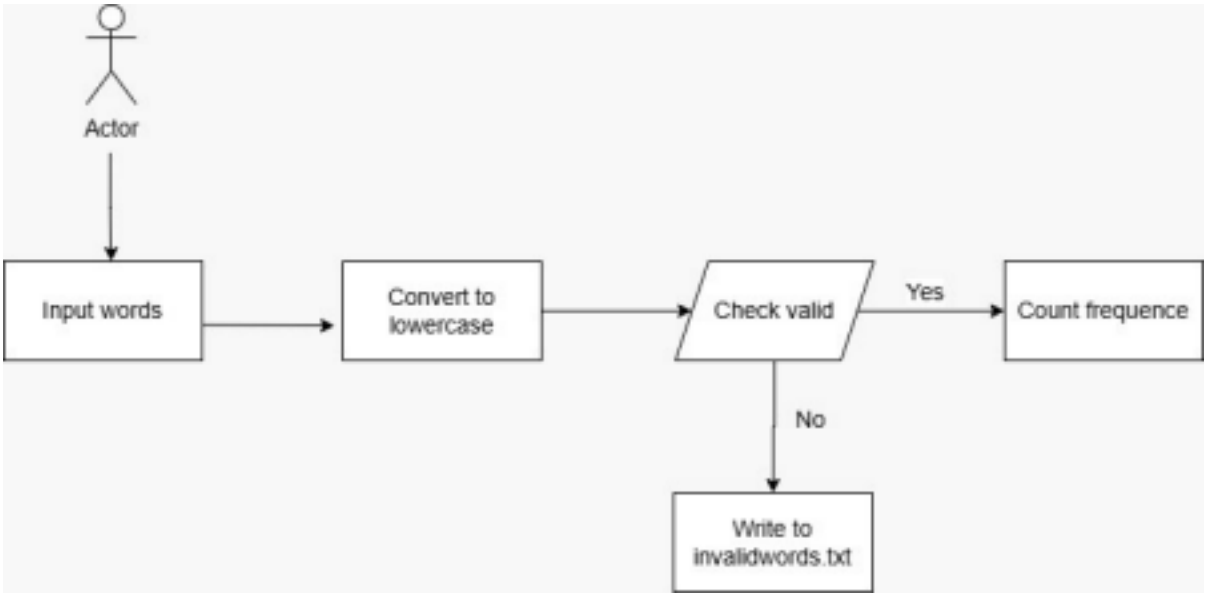
To ensure efficient storage and retrieval of words, a hash table data structure will be used. Before storing or comparing words, the application will convert them to a uniform case (lowercase). This ensures that words such as “Apple”, “APPLE”, and “apple” are treated as the same word and their counts are aggregated correctly.



WFC_04: The application shall exclude predefined common words from frequency calculation.

A separate file named commonwords.txt will contain a list of words that should be ignored during processing (e.g., is, the, and, or). The application will read this file at

the start of execution and store the common words in memory. Users can modify this file without changing the source code, making the system configurable.



WFC_05: Each input file shall be processed by a separate child thread.

For efficient processing, the application will create one child thread for each input file. Each thread will independently read its assigned file, tokenize words, validate them, and update the shared hash table. This demonstrates the use of multithreading to improve performance when handling multiple files.

WFC_06: All threads shall update a common shared hash table using synchronization mechanisms.

Since multiple threads will attempt to update the same hash table, synchronization techniques such as mutex locks will be used to prevent race conditions. This ensures data consistency and correctness of the word frequency count.

WFC_07: The application shall display all unique words along with their frequencies after processing.
Once all threads complete execution, the application will traverse the hash table and display the list of unique words along with the number of times they appeared across all files. The output will be shown on the terminal.

WFC_08: The application shall identify and discard invalid words.

Words containing digits, special characters, or non-alphabetic symbols will be treated as invalid. These words will not be included in the frequency calculation. Instead, they will be written into a separate file named invalidwords.txt for reference. The system may

also display an error message indicating the detection of invalid words.

WFC_09 (Optional): The system may display different forms of the same word.

As an enhancement, the application may keep track of different case forms of a word (e.g., Apple, APPLE) and display them along with the total frequency. This feature is optional and not mandatory for basic functionality.

4. External Interface Requirements

4.1 User Interfaces

The system uses a Command Line Interface (CLI) as the user interface.
The user executes the application by providing multiple input text file names as command-line arguments.

The command line interface is used to:

- display error and status messages,
- display the list of unique words along with their frequencies,
- display information about discarded invalid words.
- No graphical user interface is provided.

4.2 Hardware Interfaces

This system runs on general purpose computer hardware and does not directly interact with or modify any hardware components.

Below are the minimum hardware requirements for smooth execution of the application: System hardware requirements:

- 1 GB RAM
- 500 MHz processor
- Desktop or Laptop computer
- Linux based machine
- Terminal access

4.3 Software Interfaces

The system depends on the following software components to operate

correctly. Operating System:

- Linux

Linux is a Unix-like, open-source operating system used for executing the application, providing file system access, POSIX thread support and system services.

Compiler:

- g++

GNU C++ Compiler (g++) is used to compile the C++ source code of the

application. Libraries and Tools:

- POSIX thread library (pthread) for multi-threading support
- C++ Standard Template Library (STL) for containers, file handling and string processing
- Valgrind tool for memory leak detection
- CPPUnit framework for unit testing

4.4 Communications Interfaces

No network or inter-process communication interfaces are used in this system.

All interactions are performed locally through the operating system using files and shared memory within a single process using multiple threads.

5. Other Non-Functional Requirements

5.1 Performance Requirements

The system shall process multiple input text files concurrently using POSIX threads.

The application should be able to handle multiple medium-sized text files efficiently and generate the final word frequency report within a reasonable execution time.

Synchronization mechanisms shall ensure safe concurrent access to shared resources without causing significant performance degradation.

5.2 Safety Requirements

The system shall prevent data corruption during concurrent access to shared resources such as:

- the common hash table used for storing word frequencies, and
- the file used to store invalid words.
- Proper mutual exclusion mechanisms shall be used to avoid race conditions and inconsistent data states.
- The application shall terminate safely in case of unexpected input, file access failures, or runtime exceptions.

5.3 Software Quality Attributes

The system is designed to be:

- reliable, by handling invalid inputs and file access errors gracefully,
- maintainable, by following modular object-oriented design and coding standards,
- portable, by using standard C++ and POSIX libraries,
- testable, by supporting automated unit testing using CPPUnit
- scalable, by allowing multiple input files to be processed concurrently.

6. Other Requirements

The system shall support user-defined exclusion of common words through an external text file named commonwords.txt, which can be modified without recompiling the application.

The application shall generate an output file named invalidWords.txt to store all invalid words encountered during processing.

Appendix A: Glossary

CLI – Command Line Interface
POSIX – Portable Operating System Interface
STL – Standard Template Library
SRS – Software Requirement Specification
OOP – Object Oriented Programming
RAM – Random Access Memory
CPU – Central Processing Unit
OS – Operating System
G++ – GNU C++ Compiler
Pthread – POSIX thread library
HLD – High Level Design
LLD – Low Level Design
RTM – Requirement Traceability Matrix