**Error! Unknown document property name.**

**Error! Unknown document property name.**

High Level Design & Low Level Design

The purpose of this document is to provide with a template for documenting both HLD & LLD.

**Document Control :**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Project Revision History** | | | | | | | | |
|  |  |  | |  |  |  |  |  |
| **Date** | **Version** | **Author** | **Brief Description of Changes** | | | | **Approver Signature** | |
|  |  |  |  | | | |  | |
|  |  |  |  | | | |  | |
|  |  |  |  | | | |  | |

[1. Introduction 3](#_Toc368912248)

[1.1. Intended Audience 3](#_Toc368912249)

[1.2. Acronyms/Abbreviations 3](#_Toc368912250)

[1.3. Project Purpose 3](#_Toc368912251)

[1.4. Key Project Objectives 3](#_Toc368912252)

[1.5. Project Scope and Limitation 3](#_Toc368912253)

[1.5.1. In Scope 3](#_Toc368912254)

[1.5.2. Out of scope 3](#_Toc368912255)

[1.6. Functional Overview 3](#_Toc368912256)

[1.7. Assumptions, Dependencies & Constraints 3](#_Toc368912257)

[1.8. Risks 3](#_Toc368912258)

[2. Design Overview 3](#_Toc368912259)

[2.1. Design Objectives 3](#_Toc368912260)

[2.1.1. Recommended Architecture 3](#_Toc368912261)

[2.2. Architectural Strategies 3](#_Toc368912262)

[2.2.1. Design Alternative 3](#_Toc368912263)

[2.2.2. Reuse of Existing Common Services/Utilities 3](#_Toc368912264)

[2.2.3. Creation of New Common Services/Utilities 3](#_Toc368912265)

[2.2.4. User Interface Paradigms 3](#_Toc368912266)

[2.2.5. System Interface Paradigms 3](#_Toc368912267)

[2.2.6. Error Detection / Exceptional Handling 3](#_Toc368912268)

[2.2.7. Memory Management 3](#_Toc368912269)

[2.2.8. Performance 3](#_Toc368912270)

[2.2.9. Security 3](#_Toc368912271)

[2.2.10. Concurrency and Synchronization 3](#_Toc368912272)

[2.2.11. Housekeeping and Maintenance 3](#_Toc368912273)

[3. System Architecture 3](#_Toc368912274)

[3.1. System Architecture Diagram. (Not Necessary) 3](#_Toc368912275)

[3.2. System Use-Cases 3](#_Toc368912276)

[3.3. Subsystem Architecture 3](#_Toc368912277)

[3.4. System Interfaces 3](#_Toc368912278)

[3.4.1. Internal Interfaces 3](#_Toc368912279)

[3.4.2. External Interfaces 3](#_Toc368912280)

[4. Detailed System Design 3](#_Toc368912281)

[4.1. Key Entities 3](#_Toc368912282)

[4.2. Detailed-Level Database Design 3](#_Toc368912283)

[4.2.1. Data Mapping Information 3](#_Toc368912284)

[4.2.2. Data Conversion 3](#_Toc368912285)

[4.3. Archival and retention requirements 3](#_Toc368912286)

[4.4. Disaster and Failure Recovery 3](#_Toc368912287)

[4.5. Business Process workflow 3](#_Toc368912288)

[4.6. Business Process Modeling and Management (as applicable) 3](#_Toc368912289)

[4.7. Business Logic 3](#_Toc368912290)

[4.8. Variables 3](#_Toc368912291)

[4.9. Activity / Class Diagrams (as applicable) 3](#_Toc368912292)

[4.10. Data Migration 3](#_Toc368912293)

[4.10.1. Architectural Representation 3](#_Toc368912294)

[4.10.2. Architectural Goals and Constraints 3](#_Toc368912295)

[4.10.3. Logical View 3](#_Toc368912296)

[4.10.4. Architecturally Significant Design Packages 3](#_Toc368912297)

[4.10.5. Data model 3](#_Toc368912298)

[4.10.6. Deployment View 3](#_Toc368912299)

[5. Environment Description 3](#_Toc368912300)

[5.1. Time Zone Support 3](#_Toc368912301)

[5.2. Language Support 3](#_Toc368912302)

[5.3. User Desktop Requirements 3](#_Toc368912303)

[5.4. Server-Side Requirements 3](#_Toc368912304)

[5.4.1. Deployment Considerations 3](#_Toc368912305)

[5.4.2. Application Server Disk Space 3](#_Toc368912306)

[5.4.3. Database Server Disk Space 3](#_Toc368912307)

[5.4.4. Integration Requirements 3](#_Toc368912308)

[5.4.5. Jobs 3](#_Toc368912309)

[5.4.6. Network 3](#_Toc368912310)

[5.4.7. Others 3](#_Toc368912311)

[5.5. Configuration 3](#_Toc368912312)

[5.5.1. Operating System 3](#_Toc368912313)

[5.5.2. Database 3](#_Toc368912314)

[5.5.3. Network 3](#_Toc368912315)

[5.5.4. Desktop 3](#_Toc368912316)

[6. References 3](#_Toc368912317)

[7. Appendix 3](#_Toc368912318)

# 

# Introduction

The Word Frequency Counter is a concept of counting the frequency of words. Word Frequency Counter can help you to count the frequency usage of each word in each given text file, helping you understand the rate of word repetition in a file.

The scope of the project is limited to implementation of a multithreaded word frequency counter to process 2 or more input files using threads. Counter Count the frequency of every word in a text file and can also exclude the simple words which you don’t want to count.

After counting frequency of words, we will store frequency in common hash table. Then the main thread will access the global variable and display the word-frequency data on the screen.

## Intended Audience

|  |  |
| --- | --- |
| User | Count the frequency of words |

## Acronyms/Abbreviations

|  |  |
| --- | --- |
| WFC | Multithreaded Word Frequency Counter |

## Project Purpose

The main purpose of this project is to count frequency of words in given file and store the words and their count in the hash table using thread.

## Key Project Objectives

* The main purpose of this project is to count frequency of words in given file.
* Store the words and their count in the hash table.
* Use thread to handle multiple files at same time to save execution time.

## Project Scope and Limitation

### In Scope

The scope of the project is limited to implementation of a multithreaded word frequency counter to process two or more input files using threads. Counter Count the frequency of every unique word but few words can be excluded of user’s choice.

### Out of scope

It doesn't give any indication of how important the use of a word was for each event.

## Functional Overview

* Using command line argument, we will pass file name.
* According to the number of arguments, main function will create threads.
* Thread will handle function that will count frequency of word.
* Data will be stored in the common hash table.
* Main thread will display the output.

## Assumptions, Dependencies & Constraints

* Operating System:  Windows 7 & above
* Software: Linux Terminal, GCC Compiler
* Hardware: Min. 2 GB RAM, Min. 250 HD
* Possible and/or probable changes in functionality – NA

## Risks

* While displaying the words it will not print according to file, it will randomly print the words.

# Design Overview

## Design Objectives

* In design first it will take file name through command line.
* It will calculate frequency of words from file.
* It will display words and their frequency count.

### 2.1.1 Recommended Architecture

## Architectural Strategies

We can follow the Bottom-up approach in our project.

In this approach, bottom level modules developed first (Lower level module developed, tested and debugged). Then the next module developed, tested and debugged. This process is continued until all modules have been completed.

This approach is exactly opposite to the top-down approach. This approach is good for reusability of code.

In our project, first it will take file name through command line. First file will have the word which user doesn’t want to count if those are in other files. It will calculate frequency of words from other given text files. It will display words and their frequency count.

### 2.2.1 Design Alternative

**NA**

### 2.2.2 Reuse of Existing Common Services/Utilities

In WFC is using existing c functions for checking data in file is character or not:

* seekg()
* tellg()

### 2.2.3 Creation of New Common Services/Utilities

In WFC we are creating an error handling block so that if any given file is not available the error message will be displayed.

### 2.2.4 User Interface Paradigms

User will enter file name then program will calculate frequency of words and it will print the count.

### 2.2.5 System Interface Paradigms

Good design creates good projects. If the system has a good interface and it satisfies user requirements, then the software can reach new heights. The WFC offers basic word frequency count functionality.

### 2.2.6 Error Detection / Exceptional Handling

These elements are the main factors for cleanliness or quality in a code:

Maintainability: Allows us to easily find and fix new bugs, without the fear of breaking current functionality.

Extensibility: Allows us to easily add to our code base, implementing new or changed requirements without breaking existing functionality. Extensibility provides flexibility and enables a high level of reusability for our code base.

Readability: Allows us to easily read the code and discover its purpose without spending too much time digging.

### 2.2.7 Memory Management

### For memory management we will use install valgrind. Valgrind will analyze the program as it runs and report any errors or issues it finds, such as memory leaks or accesses to uninitialized memory.

* Compile your program with debugging symbols: Make sure to compile your program with debugging symbols enabled. This will allow Valgrind to provide more detailed information about the errors it finds.
* Run your program through Valgrind: Use the following command to run your program through Valgrind.
* **valgrind --leak-check=full ./myprogram**

### 2.2.8 Performance

* When user passes file name using command line, it will check file is exist or not.
* When user passes file name using command line, it will check file is empty or not .

### 2.2.9 Security

A File Security System is a console application that can be used to prevent unauthorized access to a file.

### 2.2.10 Concurrency and Synchronization

**NA**

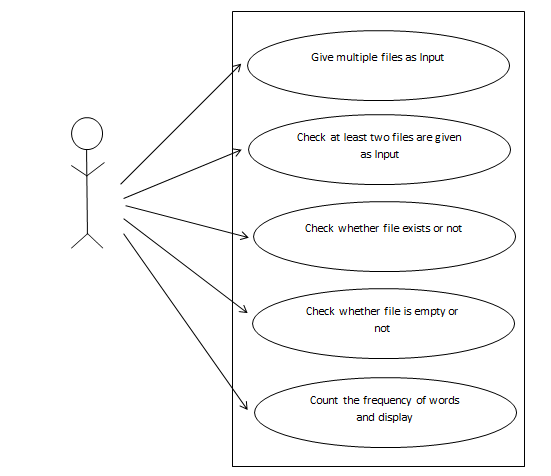
### 2.2.11 Housekeeping and Maintenance

**NA**

# System Architecture

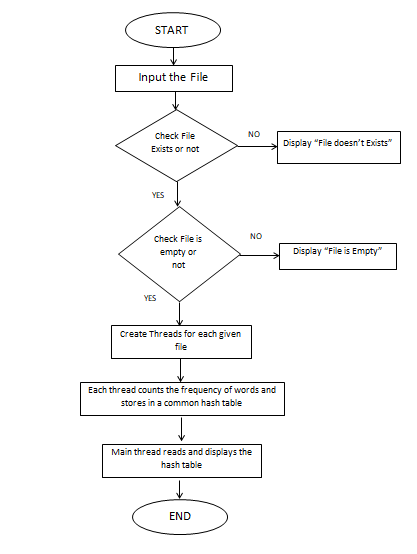
## System Architecture Diagram. (Not Necessary)

## System Use-Cases



## 

## 3.3. Subsystem Architecture



## System Interfaces

The user interface is responsible for all the interactions with the users. User

interface always effects the user mind because how easy and how functional

The project depends on the user’s point of view.

### Internal Interfaces

As an Internal Interface we are using Ubuntu Linux distribution. It is an operating system that is made up of a collection of software based on Linux kernel or you can say distribution contains the Linux kernel and supporting libraries and software.

### External Interfaces

# Detailed System Design

The WFC is a system will count the frequency of words in given files. Word Frequency Counter can help you to count the frequency usage of each and every word in a given text file, helping you understand the rate of word repetition in file.

The scope of the project is limited to implementation of a multithreaded word frequency counter to process 2 or more input files using threads. Counter Count the frequency of every unique word but simple words like or, and, for may be exclude.

After passing files through command line argument, it will check file exist or not. After opening file it will check file is empty or not, if the will be empty it will display error.

If files are available and non-empty, then it will exclude the desired words inside the first input file and then count the frequency of words inside other files.

After counting frequency of words, we will store frequency in common hash table. Then the main thread will access the global variable and display the word-frequency data on the screen.

## 4.1. Key Entities

NA

## Detailed-Level Database Design

NA

### 4.2.1 Data Mapping Information

NA

### 4.2.2 Data Conversion

NA

## Archival and retention requirements

NA

## Disaster and Failure Recovery

NA

## Business Process workflow

NA

## Business Process Modeling and Management (as applicable)

NA

## Business Logic

NA

## Variables

NA

## Activity / Class Diagrams (as applicable)

NA

## Data Migration

NA

### 4.10.1 Architectural Representation

NA

### 4.10.2 Architectural Goals and Constraints

NA

### 4.10.3 Logical View

NA

### 4.10.4 Architecturally Significant Design Packages

NA

### 4.10.5 Data model

NA

### 4.10.2 Deployment View

NA

# Environment Description

The complete details of the System Environment we can provide here

## 5.1 Time Zone Support

The number of seconds of time difference between the local time zone and Coordinated Universal Time [UTC].

## Language Support

C language is used in this project. It was developed by Bjarne Stroustrup.

## User Desktop Requirements

* Windows: 7 or above
* Processor : Minimum 1GHz and more
* Hard Drive : Min. 250GB
* Memory (RAM) : Min. 2GB

## Server-Side Requirements

NA

### 5.4.1 Deployment Considerations

NA

### 5.4.2 Application Server Disk Space

NA

### 5.4.3 Database Server Disk Space

NA

### 5.4.4 Integration Requirements

NA

### 5.4.5 Jobs

NA

### 5.4.6. Network

NA

### 5.4.7. Others

NA

## Configuration

For making MFWC we need windows version above 7 and Ubuntu Linux Distribution in our system.

### Operating System

* + Windows: 7 or above
  + Processor : Minimum 1GHz and more
  + Hard Drive : Min. 250GB
  + Memory (RAM) : Min. 2GB

### 5.5.2 Database

NA

### 5.5.3 Network

NA

### 5.5.4 Desktop

Ubuntu Linux

# References

* <https://www.w3resource.com/c-programming-exercises/file-handling/c-file-handling-exercise-7.php>
* https://codeforwin.org/2018/02/c-program-count-occurrences-of-all-words-a-file.html

# Appendix

C++ functions

**Change Log**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **QMS Template Version Control (Maintained by QA)** | | | | | |
|  |  |  |  |  |  |
| **Date** | **Version** | **Author** | | **Description** | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |