Aquino, Kelvin Yap **COMP1000**
10736109

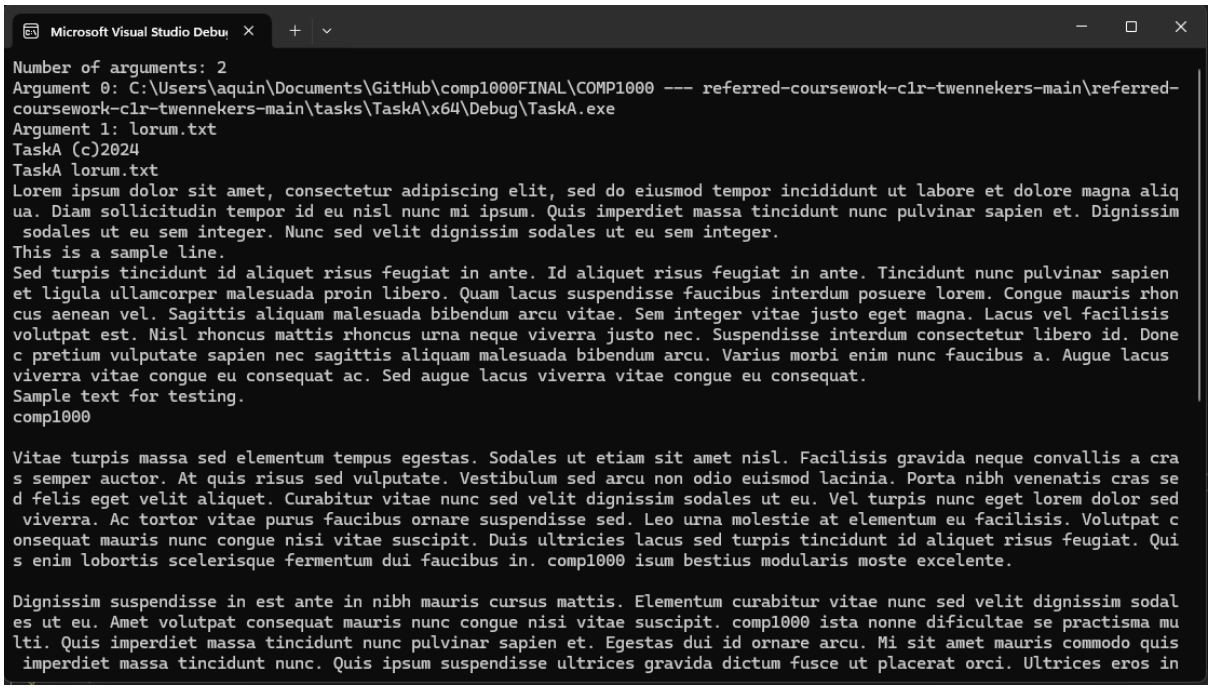
Part A – Command Line Tool

A1: Open, Read, and Display a Text File

Methodology:

1. Compile and run the command line tool with the argument “lorum.txt”.
2. Check the console for the displayed text.

Expected Result: The content of the file should be displayed in the terminal.
Actual Result: The content of the file is displayed as expected.



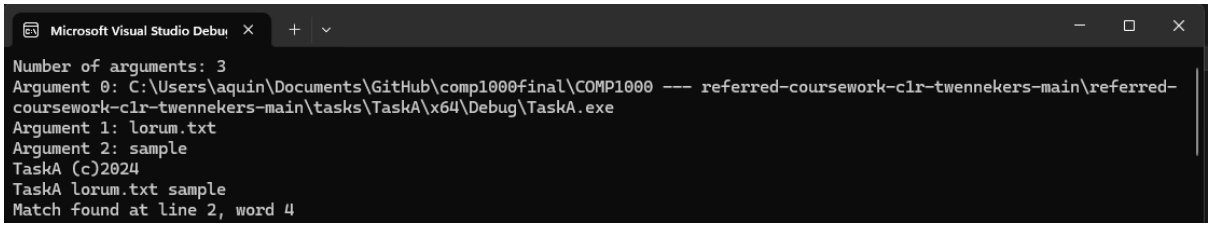
```
Number of arguments: 2
Argument 0: C:\Users\aquin\Documents\GitHub\comp1000FINAL\COMP1000 --- referred-coursework-c1r-twennekers-main\referred-
coursework-c1r-twennekers-main\tasks\TaskA\x64\Debug\TaskA.exe
Argument 1: lorum.txt
TaskA (c)2024
TaskA lorum.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliq
ua. Diam sollicitudin tempor id eu nisl nunc mi ipsum. Quis imperdiet massa tincidunt nunc pulvinar sapien et. Dignissim
 sodales ut eu sem integer. Nunc sed velit dignissim sodales ut eu sem integer.
This is a sample line.
Sed turpis tincidunt id aliquet risus feugiat in ante. Id aliquet risus feugiat in ante. Tincidunt nunc pulvinar sapien
et ligula ullamcorper malesuada proin libero. Quam lacus suspendisse faucibus interdum posuere lorem. Congue mauris rhon
cus aenean vel. Sagittis aliquam malesuada bibendum arcu vitae. Sem integer vitae justo eget magna. Lacus vel facilisis
volutpat est. Nisl rhoncus mattis rhoncus urna neque viverra justo nec. Suspendisse interdum consectetur libero id. Done
c pretium vulputate sapien nec sagittis aliquam malesuada bibendum arcu. Varius morbi enim nunc faucibus a. Augue lacus
viverra vitae congue eu consequat ac. Sed augue lacus viverra vitae congue eu consequat.
Sample text for testing.
comp1000

Vitae turpis massa sed elementum tempus egestas. Sodales ut etiam sit amet nisl. Facilisis gravida neque convallis a cra
s semper auctor. At quis risus sed vulputate. Vestibulum sed arcu non odio euismod lacinia. Porta nibh venenatis cras se
d felis eget velit aliquet. Curabitur vitae nunc sed velit dignissim sodales ut eu. Vel turpis nunc eget lorem dolor sed
 viverra. Ac tortor vitae purus faucibus ornare suspendisse sed. Leo urna molestie at elementum eu facilisis. Volutpat c
onsequat mauris nunc congue nisi vitae suscipit. Duis ultricies lacus sed turpis tincidunt id aliquet risus feugiat. Qui
s enim lobortis scelerisque fermentum dui faucibus in. comp1000 isum bestius modularis moste excelente.

Dignissim suspendisse in est ante in nibh mauris cursus mattis. Elementum curabitur vitae nunc sed velit dignissim sodal
es ut eu. Amet volutpat consequat mauris nunc congue nisi vitae suscipit. comp1000 ista nonne dificultae se practisma mu
lti. Quis imperdiet massa tincidunt nunc pulvinar sapien et. Egestas dui id ornare arcu. Mi sit amet mauris commodo quis
 imperdiet massa tincidunt nunc. Quis ipsum suspendisse ultrices gravida dictum fusce ut placerat orci. Ultrices eros in
```

A2: Specify and Perform a Search

Methodology:

1. I added the word “sample” in the lorum.txt file.
2. Compile and run the command line tool with arguments “lorum.txt sample”.
3. The tool should search for the term "sample" in the file.

Expected Result: The tool should search for "sample" and process the occurrences.
Actual Result: The word sample has been found at line 2 as expected.



```
Number of arguments: 3
Argument 0: C:\Users\aquin\Documents\GitHub\comp1000final\COMP1000 --- referred-coursework-c1r-twennekers-main\referred-
coursework-c1r-twennekers-main\tasks\TaskA\x64\Debug\TaskA.exe
Argument 1: lorum.txt
Argument 2: sample
TaskA (c)2024
TaskA lorum.txt sample
Match found at line 2, word 4
```

A3: Display Search Results

Methodology:

1. I added the words “comp1000” in the lorum.txt file.
2. Compile and run the command line tool with arguments “lorum.txt comp1000”.
3. Check the console for the average word length.

Expected Result: The results should include the line number and word number for each match.
Actual Result: The results include line number and word number as expected.

```
Microsoft Visual Studio Debu
Number of arguments: 3
Argument 0: C:\Users\aquin\Documents\GitHub\comp1000final\COMP1000 --- referred-coursework-c1r-twennekers-main\referred-
coursework-c1r-twennekers-main\tasks\TaskA\x64\Debug\TaskA.exe
Argument 1: lorum.txt
Argument 2: comp1000
TaskA (c)2024
TaskA lorum.txt comp1000
Match found at line 5, word 1
Match found at line 7, word 101
Match found at line 9, word 30
```

A4: Display Search Statistics

Methodology:

1. Compile and run the command line tool with arguments “lorum.txt comp1000”.
2. Observe if the tool calculates and displays the number of search hits as a percentage of the total number of words.

Expected Result: The number of search hits as a percentage of the total words should be displayed.
Actual Result: The results display total words, total matches, and percentage of total number as expected.

```
Microsoft Visual Studio Debu
Number of arguments: 3
Argument 0: C:\Users\aquin\Documents\GitHub\comp1000final\COMP1000 --- referred-coursework-c1r-twennekers-main\referred-
coursework-c1r-twennekers-main\tasks\TaskA\x64\Debug\TaskA.exe
Argument 1: lorum.txt
Argument 2: comp1000
TaskA (c)2024
TaskA lorum.txt comp1000
Match found at line 5, word 1
Match found at line 7, word 101
Match found at line 9, word 30
Number of search hits: 3
Average Word Length: 5.60274
Total Words: 584
Search hit frequency: 0.513699%
```

A5: Save Results

Methodology:

1. Compile and run the command line tool with arguments lorum.txt sample.
2. Check the results.csv file for an entry with the filename, search term, and frequency.

Expected Result: The results.csv file should have an entry with the filename, search term, and frequency of hits (%).
Actual Result: Results.csv contained entries with filename, search term, and frequency of hits as expected.

```
> comp1000final > COMP1000 --- referred-cours
lorum.txt,sample,0
lorum.txt,sample,0
lorum.txt,sample,0.171233
lorum.txt,comp1000,0.513699
```

A6: Class Library

Methodology:

1. Check if the code is organized into classes.
2. Verify that each class is encapsulated and handles specific functionality.
3. Ensure the use of object-oriented principles like encapsulation, abstraction, inheritance, and polymorphism where applicable.

Expected Result:

1. The solution should include well-defined classes such as “TextFile”, “Search”, “Statistics”, and “Readability”.
2. Each class should encapsulate relevant data and methods.
3. The main program should create instances of these classes and use them to perform the required operations.

Actual Result: Verify the presence of these classes in the code and their proper usage.

1. TextFile Class: Handles file operations.

```
class TextFile {
public:
    bool load(const std::string& fileName);
    std::vector<std::string> getLines() const;
    void displayContent() const;

private:
    std::vector<std::string> lines;
};
```

- Encapsulation: The “lines” member variable is private, and access to it is provided through public member functions.
- Responsibility: This class is responsible for loading and providing access to the lines of text from a file.

2. Search Class: Handles search operations.

```
class Search {
public:
    Search(const std::string& searchTerm, bool isRegex);
    void execute(const std::vector<std::string>& lines);
    void displayResults() const;
    int getMatchCount() const { return matchCount; }

private:
    std::string searchTerm;
    bool isRegex;
    std::vector<std::pair<int, int>> results; // line number, word number
    int matchCount = 0;
};
```

- Encapsulation: The “searchTerm”, “isRegex”, “results”, and “matchCount” are encapsulated within the class.
- Responsibility: This class is responsible for performing the search and storing the results.

3. Statistics Class: Handles statistical calculations.

```
class Statistics {
public:
    Statistics(const std::vector<std::string>& lines);
    void displayWordCounts() const;
    double calculateAverageWordLength() const;
    std::vector<std::pair<std::string, int>> getTopNWords(int N) const;
    std::map<std::string, int> getWordCounts() const;

private:
    std::map<std::string, int> wordCounts;
    void countWords(const std::vector<std::string>& lines);
};
```

- Encapsulation: The “wordCounts” member variable is private, and access to it is provided through public member functions.
- Responsibility: This class is responsible for calculating word counts, average word length, and providing the top N words.

4. Readability Class: Handles readability calculations.

```
class Readability {
public:
    static int countSyllables(const std::string& word);
    static int countSentences(const std::vector<std::string>& lines);
    static double calculateFleschReadingEase(int totalWords, int totalSentences, int totalSyllables);
    static double calculateFleschKincaidGradeLevel(int totalWords, int totalSentences, int totalSyllables);
};
```

- Encapsulation: This class provides static methods for calculating readability scores, encapsulating the logic within the class.
- Responsibility: This class is responsible for calculating readability metrics.

Justification for Object-Oriented Techniques:

1. Encapsulation:
   - Each class encapsulates data and methods relevant to its functionality, making the code modular and easy to manage.
2. Abstraction:
   - By abstracting file operations, search functionality, and statistical calculations into separate classes, the code becomes more readable and maintainable.
3. Modularity:
   - The modular approach allows for easy testing and debugging of individual components.
4. Reusability:
   - The classes can be reused in other projects or extended with additional features without affecting other parts of the code.

Conclusion

By structuring the solution with well-defined classes and adhering to Object-Oriented principles, the code achieves modularity, encapsulation, and reusability. Each class handles a specific responsibility, making the solution easier to understand, maintain, and extend. The main function coordinates these classes to perform the overall task, demonstrating a clear separation of concerns and a robust design.

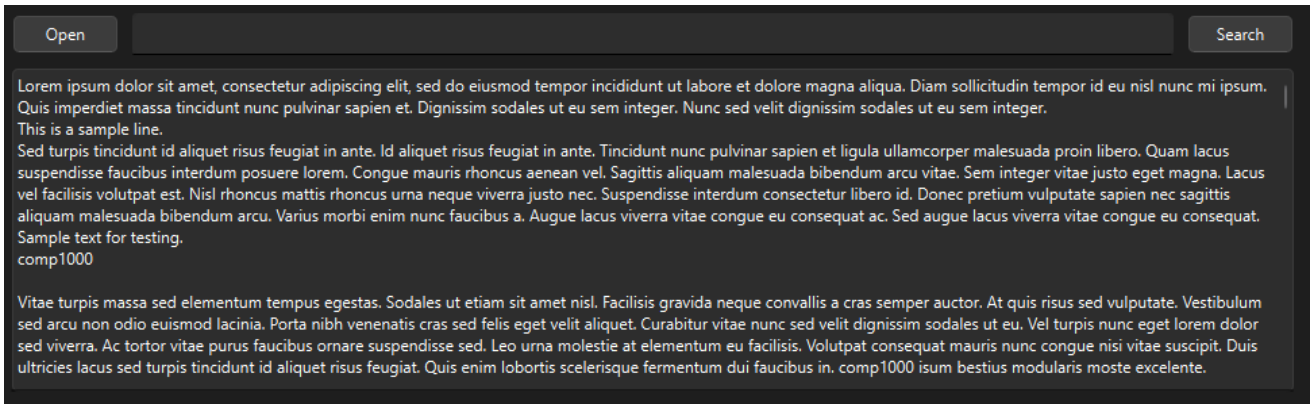
Part B – Graphical Interface

B1: Open, read, and store a text file

Methodology:

1. Run the GUI application.
2. Use the "Open" button to select and open lorum.txt.
3. Check if the file content is displayed in the GUI.

Expected Result: The contents of “lorum.txt” should be displayed in the GUI.
Actual Result: The contents of "lorum.txt” is displayed in the GUI as expected.

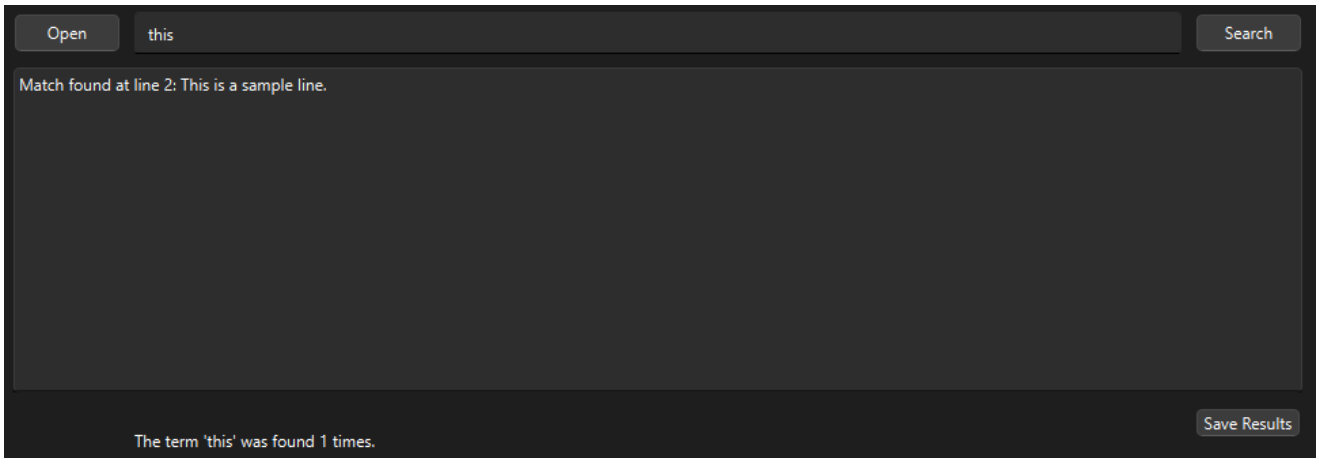


B2: Specify a Search Term

Methodology:

1. Run the GUI application.
2. Enter a search term in the search input field.
3. Initiate the search using the "Search" button.

Expected Result: The application should search for the specified term.
Actual Result: The application displays the specified term in the GUI as expected.

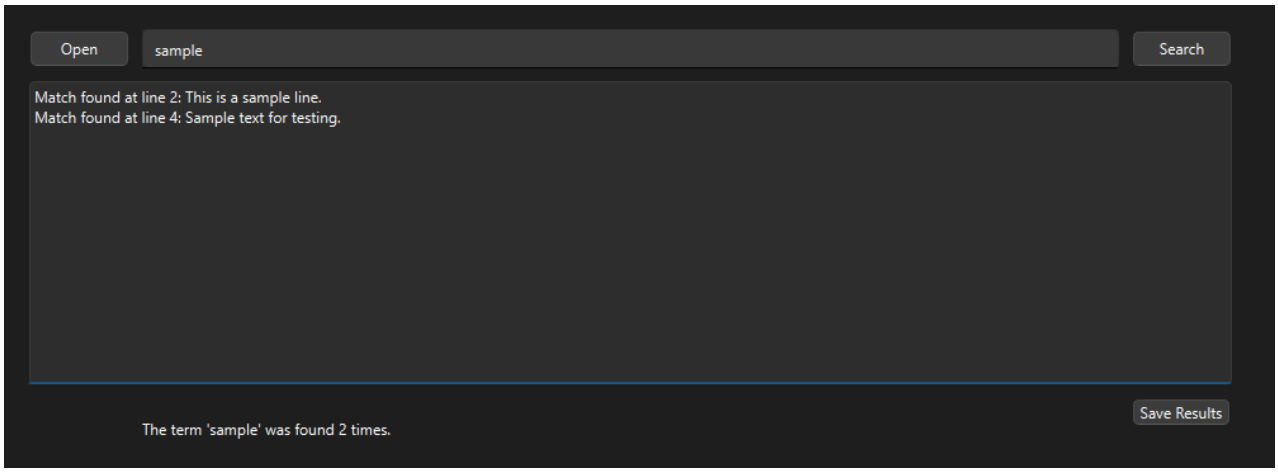


B3: Display Search Results

Methodology:

1. Run the GUI application.
2. Perform a search for the term "sample".
3. Check if the search results are displayed in the GUI.

Expected Result: The results should include the location of each match.
Actual Result: The application displays the word “sample” and includes the location of the match as expected.

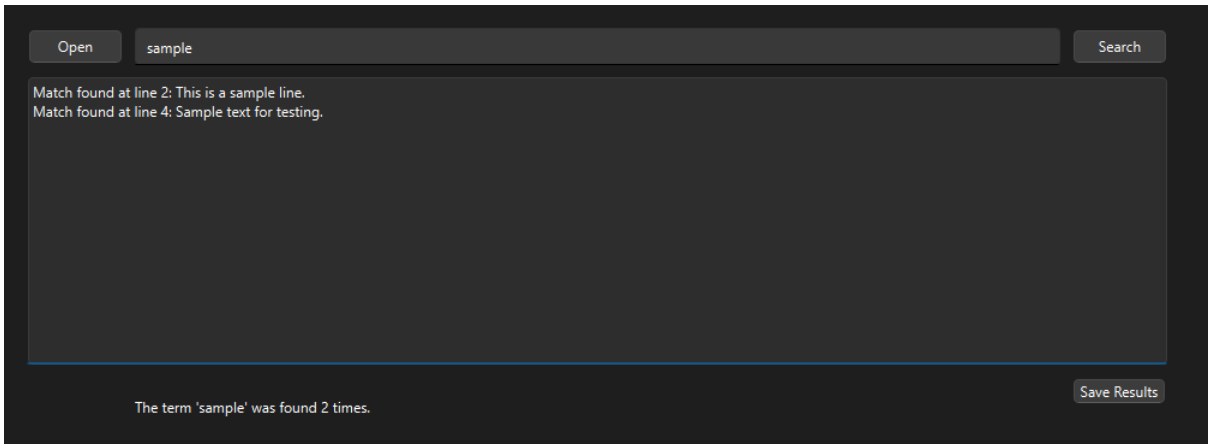


B4: Display Search Statistics

Methodology:

1. Run the GUI application.
2. Perform a search for the term "sample".
3. Observe if the number of search hits is displayed in the GUI.

Expected Result: The number of search hits should be displayed in the GUI.
Actual Result: The number of times the word “sample” was found is displayed in the GUI at the bottom left as expected.

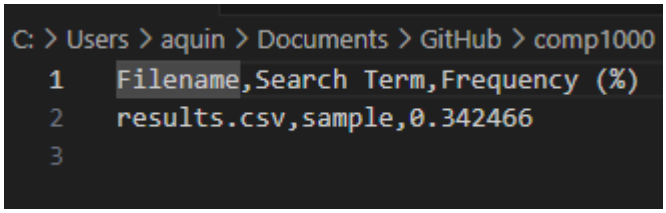


B5: Task B5: Save Results

Methodology:

1. Run the GUI application.
2. Perform a search and use the "Save Results" button.
3. Save the results to a specified location.
4. Verify the contents of the saved CSV file.

Expected Result: The results should be saved in the specified CSV file with the filename, search term, and frequency of hits (%).
Actual Result: The results have been saved in a csv file with the filename, search term, and frequency of hits as expected.



GitHub Repository:

URL: https://github.com/kjaquino1/comp1000final.git