COMP1000 C1 W2 Report 10859497

YOU ARE REQUIRED TO COMPLETE AND SUBMIT THIS ALONGSIDE YOUR CODE

The following document has been started for you. The intention is to list tests that need to be performed to evidence which requirements have been met. It is important to be honest and only claim something works if you can evidence it. Note that the assessor will be running a whole batch of tests on your code.

|  |  |  |
| --- | --- | --- |
| **Name(s)** | Dillen Stadden | **IDE**  **(VS)** |
| **Email(s)** | [dillen.stadden@gmail.com](mailto:dillen.stadden@gmail.com)  dillen.stadden@students.plymouth.ac.uk |
| **GitHub URL** | <https://github.com/dsplymouth/comp1000-referral-2023-24-twennekers-main>  The name is:  comp1000-referral-2023-24-twennekers-main |
| Task A Complete? (Yes, No, Partially) | Task A complete as requirements are met. | ? |
| Task B Complete? (Yes, No, Partially) | Task B complete as gui can be used by the user to complete requirements. | ? |
| For the command line tool, I deleted the build folder and hidden folder .vs | No as its within the repo of my github. |  |
| I have zipped my code and uploaded it to the DLE | Used github Repository as chosen method |  |
| Student ID | 10859497 |  |
| I have completed and submitted COMP1000 C1 W2 Report Template.docx | True |

# Task A:

|  |  |
| --- | --- |
| Requirements | Description |
| A1 - Open, read and display a text file | The user should be able to specify a text file to open via as the first parameter on the command line. The application will then read the contents of the file into a data structure (such as a string) and display it in the terminal window. |
| A2 - Specify and perform a search | The user should be able to specify a text file to open via as the first parameter on the command line. The application will then read the contents of the file into a data structure (such as a string) and display it in the terminal window. |
| A3 - Display search results | Upon performing a search, all matched results should be displayed in the console in a structured form. This should include the line number and word number of each match. |
| A4 - Display search statistics | The number of search hits should be calculated and displayed in the terminal as a percentage of the total number of words |
| A5 - Save Results | For each search, you should append the file “results.csv” with a summary of the results as comma separated values (csv). Each record should be written on a separate line and include: the name of the file being searched, the search term, the frequency of hits (%). |
| A6 - Class library | The solution should be written using C++ classes and the appropriate Object Orientated techniques. Justify your solution in your report. |

# Task A steps to solve each requirement:

1. A computer screen shot of code

   Description automatically generatedTo be able to accept a file name and then display the file contents to the console I made a function called “display\_file\_content” which takes in the file name – sifted for usable parameters and feeds the file name into the function and opens the file using an “ifstream” and reads it line by line using “getline”, each line is then printed to the console using “cout” and also includes error handling.  
   My code uses namespace std. This completes the A1 requirement.

Result for lorum.txt.

A black and white text

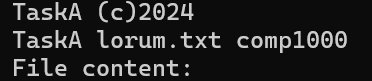
Description automatically generatedExpected.



1. If a file has been found and displayed the “search\_file” function is used to search for the search term in the file. The file name, search term and the bool “is\_regex” is put in as parameters. The “findArg” function checks to see if -regex has been used which determines if the “search\_file” should use regex.   
     
   The search function is the called and opens the file using “ifstream” reading line by line, each line is then put into separate words using “istringstream”. If regex is true then regex\_search will be used if not true then it will check if the search term is a substring of the word. When a match is found it is stored as pairs of lines and word numbers in the “matches” vector. This completes A2.

Result:

A black background with white text

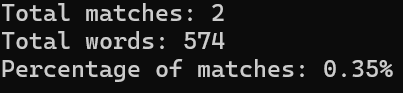
Description automatically generatedExpected Result.

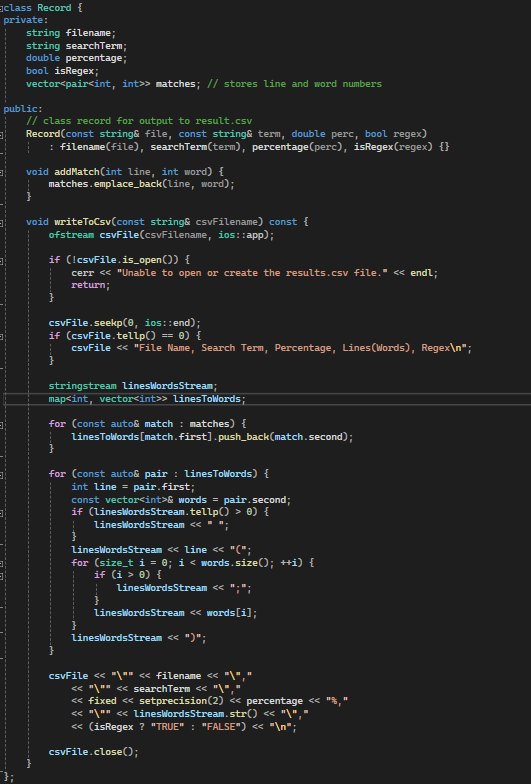
1. When the search is done all results should be displayed to the console including the line number and word number of each match, the “calculate\_statistics” is used to display the number of matches, total words and the percentage of matches out of the total words, printed in the console.

A screen shot of a computer code

Description automatically generatedThis completes A3. In addition the function calculates the percentage of matches out of the total words, and is printed using cout to the console, completing A5 requirement.

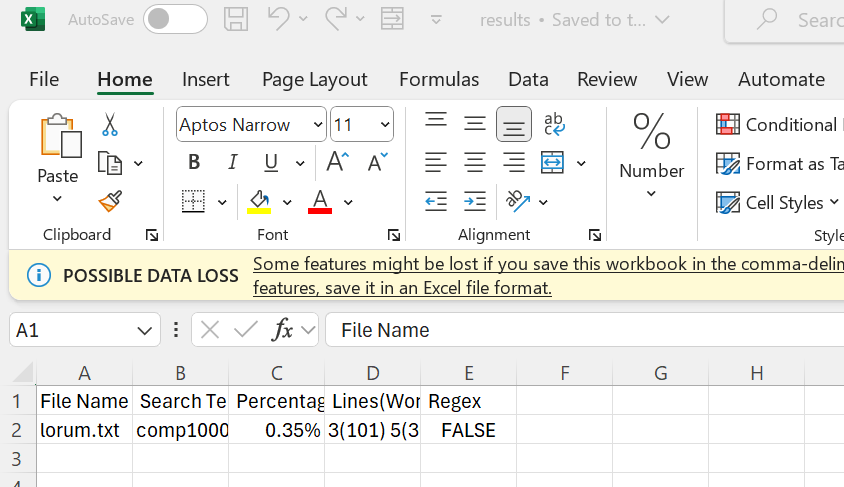
Result for comp1000 search term:





1. The Record class is used to save the results to “results.csv”, details like file name, search term, percentage, and if regex was used are appended. These are all initialised in the start of the class.   
   The addMatch function adds each match to the matches vector. The file is opened in append mode, so that old data is overwritten and if its empty, the case when results.csv isn’t already made, the headers are added first.   
   The data is formatted using linesWordsStream for each line containing matches (;) had to be used because (,)/commas would cause breaks between data belonging to one cell in excel. The data is then appended as a new row in the csv file. This meets the A5 requirement.  
     
   In order to succeed A6 the code uses the Record class and all data and methods related to the output of the program (the storing and writing search results). This means that the data is protected and can only be modified through certain methods. In addition, it makes the code more modular and easier to read.  
   The methods I’ve used are using modules to separate code and encapsulation which protects the data, making it easier to reuse and maintain. This succeeds the A6 requirement.

Result:



Expected Result.

# Task B - GUI

|  |  |
| --- | --- |
| Requirements | Description |
| B1 - Open, read and store a text file | Using the GUI, the user should be able to open a file using a dialog box that is standard for the platform. The application will then read the contents of the file into a data structure (such as a string) and display it in the GUI. For full marks, the view should scroll if it does not fit. |
| B2 - Specify a search term | Once a file has been fully read, the user shall be able to enter a search term via the GUI and search for matches. |
| B3 - Display search results | Upon performing a search, the results should be displayed in the GUI. This should include the location of each match. If it cannot all fit on the screen, the user shall be able to scroll through the content. |
| B4 - Display search statistics | The number of search hits should be presented in the GUI once the search has completed. |
| B5 - Save Results | For each search, you should append a results file with a summary of the results as comma separated values (csv). Each record should be written on a separate line and include: the name of the file being searched, the search term, the frequency of hits (%). For full marks, the user shall be able to use the GUI to specify the name and location of the file. |

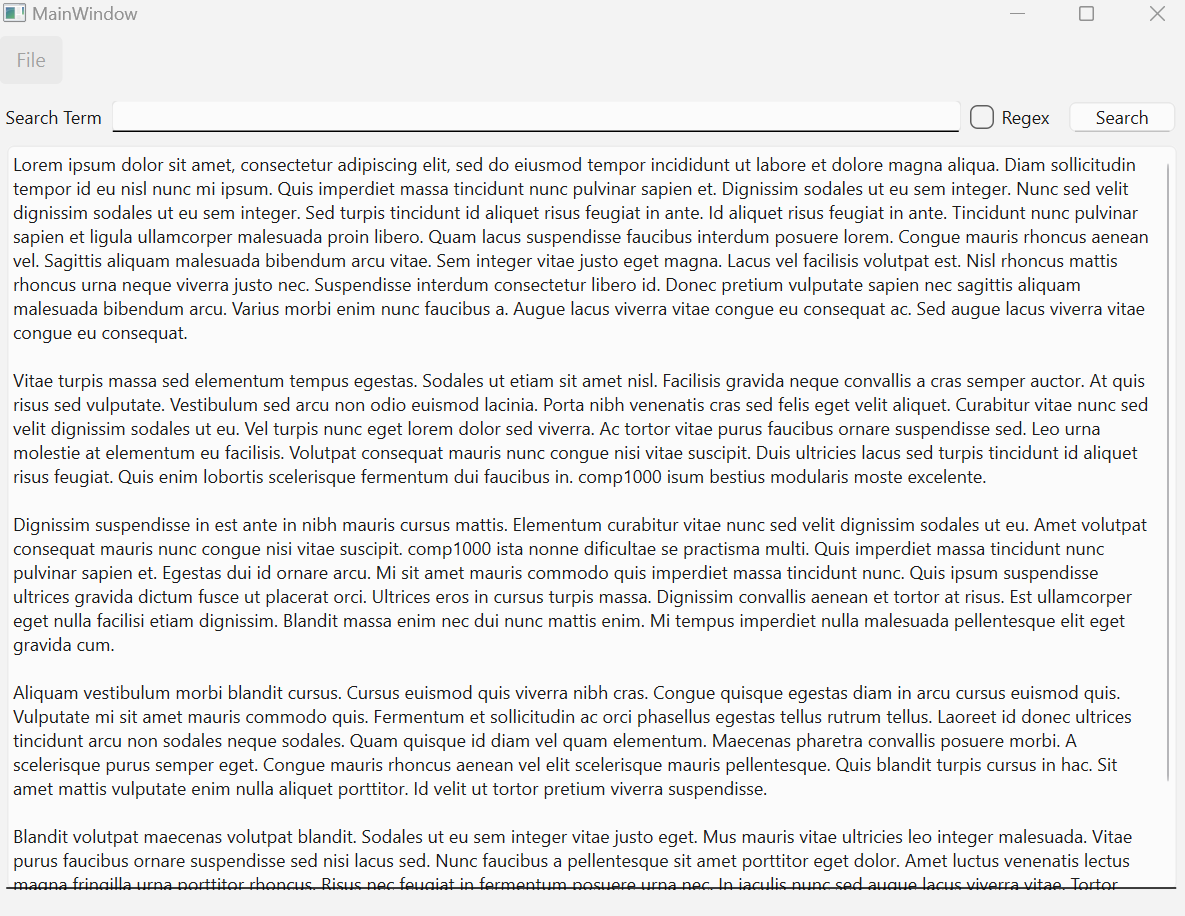
# Task B steps to solve each requirement:

A screen shot of a computer code

Description automatically generated

1. QFileDialog is used to open a file dialog which allows a user to select a text file. It then is opened in a read only mode and reads the content into currentFileContent using QTextStream.

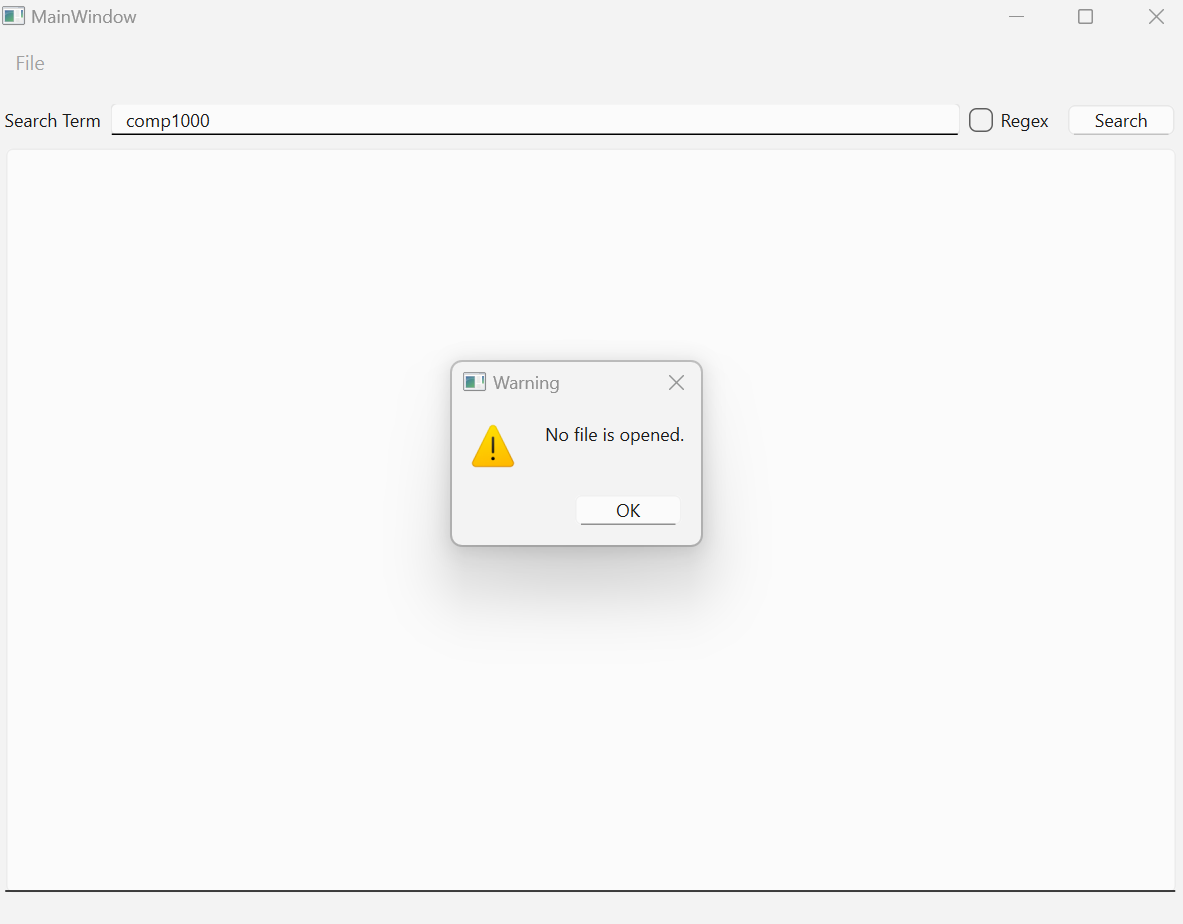
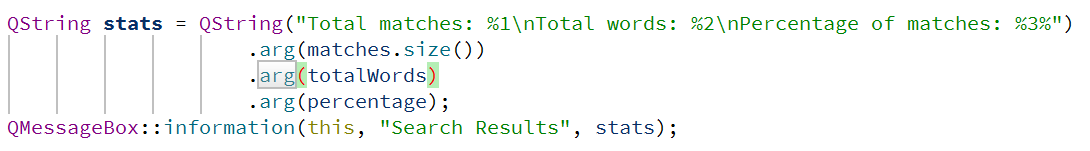
It is then displayed within QTextEditWidget and supports scrolling if the file doesn’t fit. The open file is under the file button on the application and is an QAction. Also the Exit button clears the file not app.  
  
  
Results:

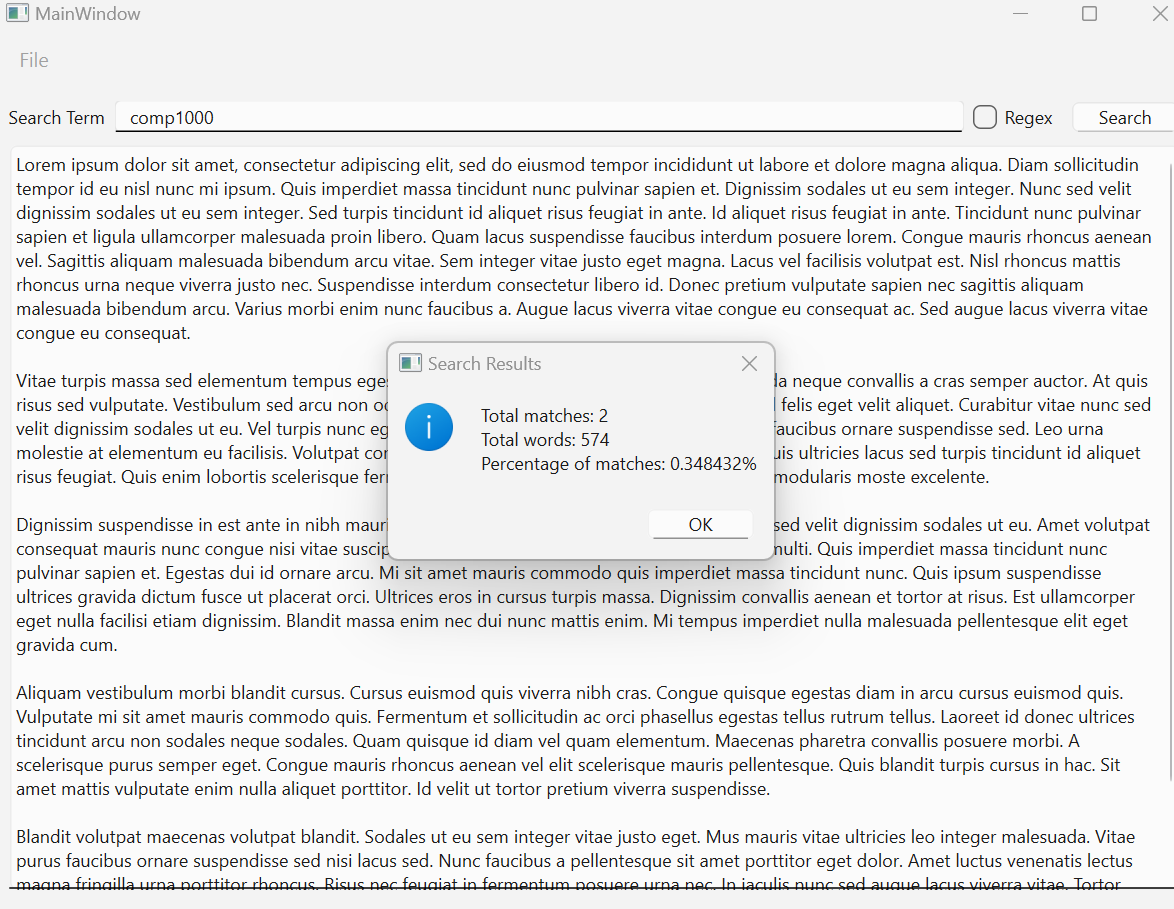
A screenshot of a computer

Description automatically generated



1. The user is able to type in the QLineEdit (searchLineEdit) and checks if regex is enable using the regex checkbox. Displays an error if no file is found.

  
  
  
  
  
  
  
  
  
  
  
  
Results:   
  
  
  
  
  


1. This code shows a dialog box with the total number of words, matches and percentages. The scrolling does not include the lines and words but the results.csv file has it included so it doesn’t have scrolling. This also presents the statistics that are calculated in the Calculate Statistics function. Which completes B3 and B4.
2. The code opens a csv file called results in append mode if the file is empty/not there it creates it and writes the headers first.  
   The code formats the data with search word, file location, total matches which include line numbers and word numbers, percentages and if regex was used. The code also displays if the data was written to the csv file. Completing B5.  
   \*It usually creates in the build folder\*  
   Results:

