|  |  |
| --- | --- |
|  | **Rochester Institute of Technology**  **Golisano College of Computing and Information Sciences**  **School of Interactive Games and Media** |

**Before you start:**

* **You have two hours to complete this part and submit it back to MyCourses. The clock starts when you downloaded Exam1.zip file, so plan your time carefully.**
* **Late work will NOT be graded.**

**Restrictions:**

* This is an individual exam, you are not allowed to collaborate with others or seek outside help.
* No access to any other computer or information on another computer is permitted. For example, you may not use the Internet or any other network services. The only exception to this rule is the use of MyCourses and the use of Visual Studio’s help functionality.
* Only Visual Studio 2015 may be used as an editor for your code.
* Failure to comply with these restrictions will be considered cheating and you will be subject to the ramifications described in the Department’s Academic Dishonesty Policy.

**Notes:**

* REMEMBER TO SAVE OFTEN! Any computer can crash. You are responsible for saving your code.
* Do NOT delete your files or restart your computer until AFTER you’ve verified that you have uploaded the correct zip file. **It is your responsibility to make certain the work you submitted is your final version of the code.**
* Your program must compile and run to get full credit. Look at the Grading Sheet for specifics on how this Practicum will be graded.

**Requirements:**

* You are expected to name your variables, constants, objects, and classes in accordance with our Naming Conventions in the Programming Standards handout you received in class.
* You are expected to align and indent your code, and to use white space to improve readability.
* Comments, indentation, and self-descriptive variable names are required.

#### THE PROBLEM

## Introduction:

### Background:

This assignment will test your knowledge of collections, exceptions, and file processing.

**The Assignment:**

Your program will analyze a file of text and count the number of times each word appears in the file. In addition, it will have a search functionality, allowing you to see how many times a specific word appears in the document.

There is a partial project named “Exam1BaseCode.zip”. You need to unzip this archive and add your additional code into it. A test file is used with this assignment. This file is found in the directory with the base source code and is named “TestText.txt”.

### Requirements:

The program will consist of two classes. “TestFileData” will have the main method and will drive the testing of the functional class. “FileData” will handle all of the file processing.

## FileData Requirements:

1. This class will analyze the data from a text file, counting the number of occurrences of each word in the file. **Case is important: ‘Test’ is different from ‘test’**.
2. You can only use collections to store your data. **You will lose points if you use arrays in your solution.** (Exception: you can use the Split method from the String class to split a line of text into words.) The type and number of collections you use when processing the file are entirely up to you. **However, the final set of data, both words and counts will end up stored in a Dictionary.**
3. Your class will have a parameterized constructor that will accept a string as an argument. This string will be the path to the file you are using. The constructor will attempt to open the file for use. If an exception occurs, DO NOT handle it in the constructor. The Main method will have to take care of this one.
4. The Analyze method will populate your collection(s) with counts of the number of occurrences of each word in the file. The method will take no parameters and will return no values. The word counts are case sensitive. Read in each line of the file and process it, tracking the number of occurrences. You need to handle any exceptions that occur inside of the method by listing the message on the console, closing the file and returning from the method. If there is no exception, close the file and return.

**Note that this method can use any type of collections when processing the data, but the final results must be stored in a Dictionary where the word is the key and the occurrence count is the value.**

1. The Search method will take no parameters and will return no values. The method will contain a loop that prompts the user for a word. It will then try to locate that word in the Dictionary created by the Analyze method. The word and its count will displayed if it exists. A message will appear if the word is not present. (See the sample runs for examples of the output.) The loop will end if the user enters the word “EXIT” ( all in caps).
2. The ListWords method will take no parameters and will not return any data. This method will list the words and word counts found in the Dictionary in ascending alphabetical order (i.e. A to Z). It returns after listing all of the data. Hint: some collections have a method for sorting the collection contents.

## TestFileData Requirements:

1. The class contains the Main method and will drive all of the testing. There is some code already in the Main, that code cannot be changed or deleted. Make certain all of your new code appears before the comment telling you not to change or add code after the comment lines.
2. Get a file path to the text file being analyzed from the user.
3. Create a FileData object, passing the file path to the constructor.
4. If there is a problem opening the file, an exception will be thrown by the FileData constructor. This exception must be handled in the Main by listing the exception and then ending the program.
5. Call the Analyze method of FileData to process the file.
6. Call the Search method of FileData to list the results.
7. Call the ListWords method to list out the words and their counts.

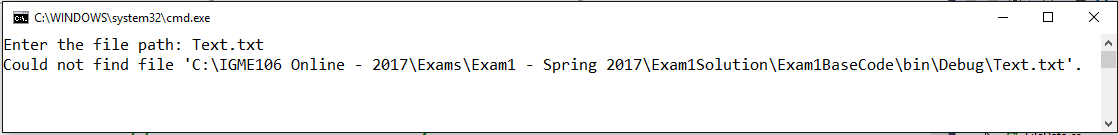
**Data File:**

The data file for this part is named TestText.txt and is found in with the source code. You may move the file to any location you wish within the project. The file contains the Constitution of the United States. Punctuation and special characters like parentheses have been removed from the file.

Hint: Read over the grade sheet to make certain you have met the graded requirements.

### Sample Runs:

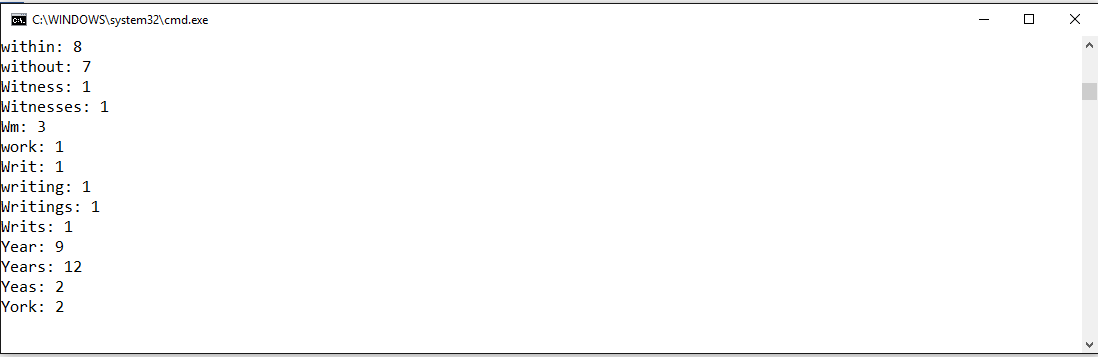
Bad File Path:



Correct Run:



Output of words continues on and ends with these lines:



**Submitting your work:**

You will be submitting your exam through MyCourses. Follow the instructions below to package up and submit the code:

1. Create a zip file containing the **entire** project. (If you submit only your C# files, you will lose 25 points immediately. The entire project must be submitted.)
2. Once you have the zip file ready, check it to be certain all of your work is present.
3. Bring up a browser and go to <https://mycourses.rit.edu/>
4. Login using your DCE account (the one that looks like “abc1234”)
5. Go to the conference for this course.
6. Locate the Dropbox entry at the top of the site
7. Click on the Exam1 dropbox.
8. Use the “browse” button to locate your .zip file. After you’ve done that, press upload to send the file.
9. **Press the submit button to actually upload the zip file.**
10. **Verify that it submitted correctly.**

**Note that I do not give makeup exams if you do not submit your work correctly.**

**Gradesheet**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Max** | **Earned** |
| **Part I Points** | **50** |  |
| **Test Class:**   * Gets a file path from the user * Creates a FileData object * Catches all exceptions from the FileData constructor * Lists exception message and exits * Calls the Analyze method * Calls the Search method * Calls the ListWords method   **FileData Class:**   * Constructor gets a file path and opens the text file * Uses one or more collections to handle the data * Reads each line of the file * Gets each word from the line * Creates a Dictionary of words and occurrence counts * Displays the Search output in as seen in the sample run * Displays the ListWords output as seen in samples * Closes the file when done reading it. | **2**  **2**  **2**  **2**  **1**  **1**  **1**  **5**  **5**  **5**  **2**  **5**  **5**  **5**  **1** |  |
| General Requirements:   * Program works correctly | **6** |  |
| Deductions: (Applied if necessary)   * Program does not compile * Did not submit complete project * Used arrays in the project | **-25**  **-25**  **-15** |  |
| **Part 3 Points** | **50** |  |
| **Total Grade** |  |  |

Coding Standards: (Points deducted after Program Design/Requirements Points are assigned.)

* Naming Conventions: Deduction (0-5 pts) \_\_\_\_
* Code comments: Deduction (0-5 pts) \_\_\_\_
* Alignment & Indentation Deduction (0-5 pts) \_\_\_\_
* Multiple classes in a single file Deduction (-5 pts) \_\_\_\_

Coding Standards Deductions: \_\_\_\_\_\_\_\_\_\_\_

**Grade (Total Grade – Coding Standards Deductions): \_\_\_\_\_\_\_\_\_\_\_**