PG2 – LAB 1: HISTOGRAM

CONTENTS

OVERVIEW

There will be 3 parts to Lab 1:

A. Read Methods, Menu loop (Lecture 1)
B. Lists and Dictionaries, string splitting (Lecture 2)
C. File Input / Output (Lecture 3)

PART A – READ METHODS, MENU LOOP

For Part A, you will create the basic application for Lab 1.

- You will create a menu loop in Main.
- You will create 3 methods to make it easier to get user input.

Part A-1: Console Application

SETUP

Create a **C# console application (.NET Framework)**.

GRADING: 2 POINTS

Part A-2: ReadInteger

Create a method called **ReadInteger** that will ask the user to input a number. The method should show a prompt, read the user's input (Console.ReadLine maybe?), and return the integer**.** Console.ReadLine will give you a string so you will need to convert the string to an integer. **DO NOT THROW AN UNHANDLED EXCEPTION.** If the user does NOT enter an integer OR the integer is not within the min-max range, show an error message to them, show the prompt again and ask for the user's input. You'll need a **loop** for this. **Do not return until the user enters a valid integer**.

NAME	RETURNS	PARAMETERS	COMMENTS
ReadInteger	int	string prompt int min int max	Show the prompt, read input, return integer

EXAMPLE USAGE

```
int year = ReadInteger("Year: ", 1908, 2021);
int passengers = ReadInteger("Number of passengers: ", 1, 10);
```

EXAMPLE OUTPUT

Year: **steve**
That is not an integer. Please try again.
Year: **2019**

GRADING: 10 POINTS

COMMON MISTAKES:

- -2 pts: Calling int.Parse after already calling int.TryParse. If you call int.TryParse and it returns true, then the string is converted and the number is stored in the out parameter.
- -2 pts: Calling the ReadInteger method recursively. A simple loop is better in this scenario so do not use recursion.
- -2 pts: Not checking the number against the min and max parameters. ReadInteger should not return until the user enters a number AND the number falls within the min-max range.
- -5 pts: using int.Parse without a try-catch. ReadInteger should not throw an unhandled exception. Catch the exception using a try-catch and show a message to the user. Continue looping until the input is valid.

Part A-3: ReadString

Create a method called **ReadString** that will ask the user for a string. Instead of returning the value like in ReadInteger, you should use **pass by reference** to get the string back to the caller. The method should show a prompt, read the user's input, store the input in the ref parameter. If the user does NOT enter a string, show an error message to them, show the prompt again and ask for the user's input. You'll need a loop for this. **Do not return until the user enters something.** You should use the IsNullOrEmpty or the IsNullOrWhiteSpace methods of the string class to check if the string is empty.

NAME	RETURNS	PARAMETERS	COMMENTS
ReadString	void	string prompt ref string value	Show the prompt, read input, store in ref parameter

EXAMPLE USAGE

```
string make = string.Empty;
ReadString("Make: ", ref make);

string model = string.Empty;
ReadString("Model: ", ref model);
```

EXAMPLE OUTPUT

Make: **Ford**

GRADING: 5 POINTS

COMMON MISTAKES:

- -1 pt: converting the string input to a number as part of validation. The only validation you need to check is whether the input is empty or not.
- -2 pts: returning without checking if the input is empty. ReadString should not return if the user's input is empty. You should use the IsNullOrEmpty or the IsNullOrWhiteSpace methods of the string class to check.

Part A-4: ReadChoice

Create a method called **ReadChoice** that will ask the user to select from a list of options, like a menu. Instead of returning like ReadInteger or passing back the value like ReadString, you should return the selection through an **out parameter**. The method should show a list of options to the user, show a prompt, get the user's selection, and return the selection through an **out parameter**.

You'll need to pass the list of options as a **string array**. Something like string[] {"1. Add Car", "2. Show Cars", "3. Exit" }. The method should loop over the array and show each option on a new line. Then it should show the prompt and read the user's input**. You should use your ReadInteger method you created earlier.**

NAME	RETURNS	PARAMETERS	COMMENTS
ReadChoice	void	string prompt string[] options out int selection	Reuse the ReadInteger method to get the user's selection

EXAMPLE USAGE

```
int menuChoice = 0;
string[] mainMenu = new string[] { "1. Add Car", "2. Show Cars", "3. Exit" };
ReadChoice("Choice? ", mainMenu, out menuChoice);
```

EXAMPLE OUTPUT

1. Add Car

2. Show Cars
3. Exit

Choice? **Steve**
That is not a number. Please try again.
Choice? **2**

GRADING: 5 POINTS

COMMON MISTAKES:

- -1 pt: duplicating the ReadInteger logic. ReadChoice should call ReadInteger instead of duplicating the code.
- -1 pt: hardcoding the range passed to ReadInteger. You should use the options.Length for the max value passed to ReadInteger.

Part A-5: Menu loop

You will need to create a loop in **Main** that handles the menu options for lab 1.

- This should be a simple **while** loop that loops while the menu selection is NOT exit.
- Inside the while loop, you should…
    - Call **ReadChoice** to show the menu and get the user's menu selection
    - use a **switch** statement that has logic for each menu option.
        - Option 1: Show Histogram
        - Option 2: Search for Word
        - Option 3: Save Histogram
        - Option 4: Load Histogram
        - Option 5: Remove Word
        - Option 6: Exit

NOTE: for Part A, you will only need code to handle the exit option.

You should have a menu loop that shows the menu and let's the user select a menu option.

GRADING: 3 POINTS

COMMON MISTAKES:

- -1 pt: Exit does not exit.

PART B – CREATE THE WORD COUNTS

For Part B, you will need to get the speech text, split the speech to get the words, create a dictionary to hold the word counts and show the histogram chart.

Part B-1: The Speech

NOTE: find the data to use for this project in the **speechString.txt** file for the lab.

Create a method called **GetSpeech**. It should return the string from the speechString.txt file. **Copy the text from the file to the method.**

NAME	RETURNS	PARAMETERS	COMMENTS
GetSpeech	String	(none)	Returns the string that is supplied in the speechString.txt file.

Call the GetSpeech method from Main. Do this BEFORE the menu loop starts.

Split the string into an **array of words** that appear in the string. You need to figure out what the possible delimiter(s) are to give you all the words (look at all the characters in the string that separate the words).

Now convert array of words to a list of strings.

GRADING: 10 POINTS

COMMON MISTAKES:

- -2 pts: no GetSpeech method.
- -2 pts: not splitting on the correct delimiters. To get the words, you need all punctuation and the escape sequences (\n, \t, \r) in your list of delimiters.
- -3 pts: not converting the string array to a List. The Split method returns an array of strings. Convert that to a List<string>.

Part B-2: Word counts

THE DICTIONARY

Now that you have the **list of words**, you need to calculate how many times each word appears in the list of words. Create a **Dictionary** to store those counts. The key of the dictionary will be the words and the value will be the counts. Loop over the **List of words** and ***put*** or ***update*** the word in the dictionary.

NOTES:

- Make it **case-insensitive** meaning that if the word is upper-case and lower-case in the data, only 1 will appear in the dictionary. For example, 'The' and 'the' are the same word so only one should be in the dictionary. **HINT: look at the different constructors for Dictionary to make this easier**.
- You need to check if the word is in the dictionary to decide if it should be **added** or if it should be **updated**. Use **ContainsKey** or **TryGetValue**.

GRADING: 10 POINTS

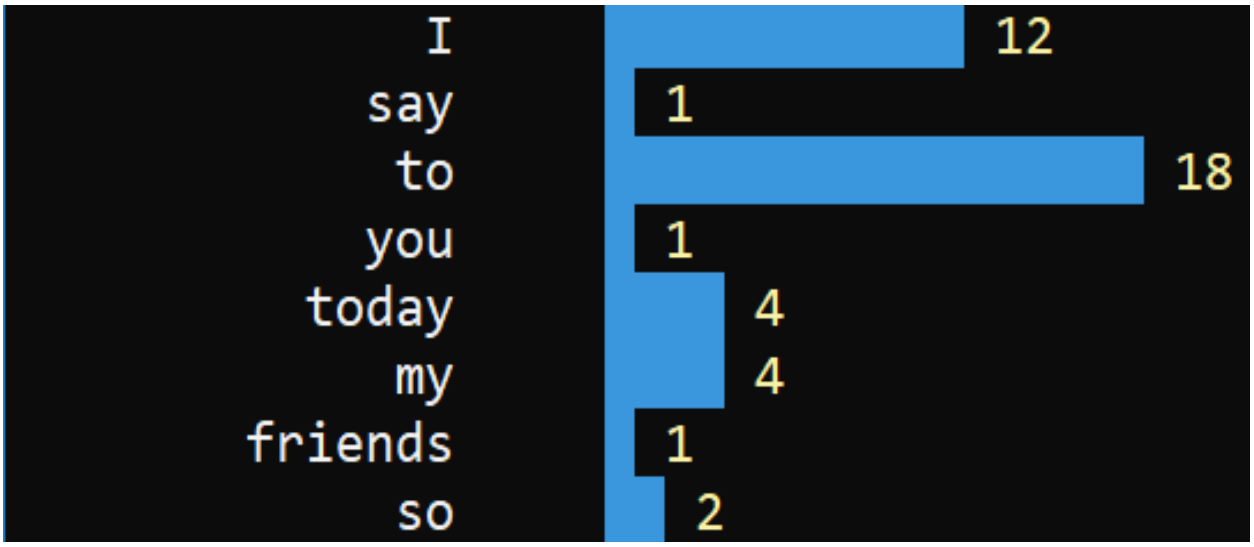
COMMON MISTAKES:

- -5 pts: not calculating the word counts correctly. Loop over the list of words from the speech. Check if the word is in the dictionary and update the value if it is or add it if it is not.
- -3 pts: using a list to store unique words. The keys in the dictionary are unique so no other list is needed outside of the list of words for the speech.

Part B-3: Show Histogram

Now you have the information you need to add logic to the menu for the "Show Histogram" option. For each word in the dictionary, print the **word**, the **count** and a **bar** representing the count as a horizontal bar chart (see screenshot). **Format your chart** to make it look nice! Use Console.CursorLeft to align the bars.

EXAMPLE OUTPUT:



GRADING: 10 POINTS

COMMON MISTAKES:

- -5 pts: not showing the bar for the chart.
- -2 pts: chart is not formatted well.

Part B-4: Search for Word

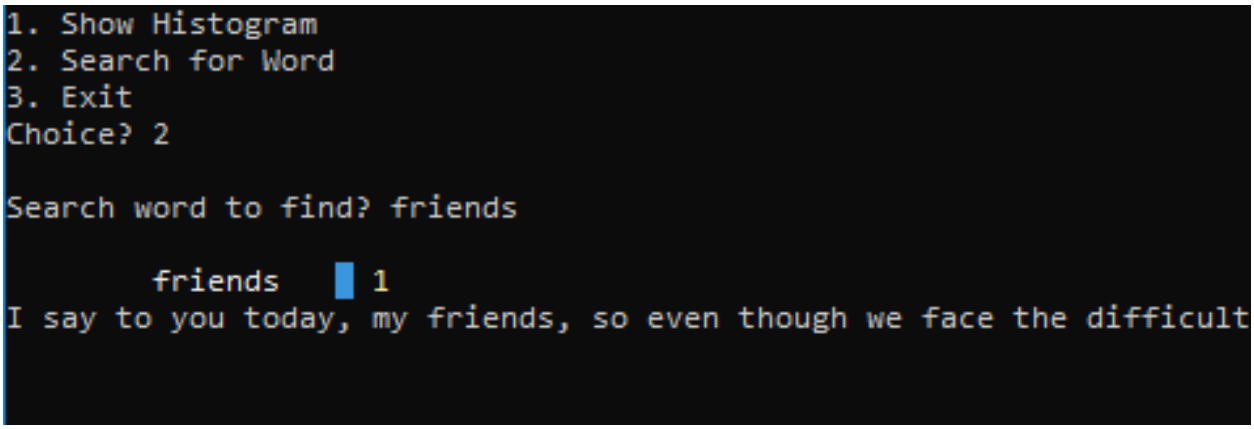
Now you have the information you need to add logic to the menu for the "Search for Word" option. **Ask the user for a word to search for** ("What word do you want to find? "). **Use ReadString to get the word from the user!**

If the word is in the Dictionary, print the **word**, **bar,** and **count**.

Show the sentences that the word appears in. HINT: you can split the original speech text on different delimiters to get the sentences.

If the word is NOT in the dictionary, print "<word> is not found.". (replace <word> with what the user entered)

EXAMPLE OUTPUT:



GRADING: 15 POINTS

COMMON MISTAKES:

- -10 pts: not showing the sentences for the search word.
- -2 pts: using a loop to find the search word in the dictionary. Using a loop to find a key in a dictionary defeats the purpose of using a dictionary. Use one of the build-in methods (ContainsKey or TryGetValue) instead.
- -1 pt: using .Contains to find the search word in a sentence string. Contains will give you false positives (Example: trying to find "any" using Contains will match with "anywhere").
- -2 pts: trying to access a key's value without first checking if the key exists.
- -2 pts: the bar is missing from the output.
- -1 pt: not using ReadString to get the search word from the user

PART C – FILE INPUT / OUTPUT

Part C-1: NuGet & Json.NET

NuGet is the package manager for .NET – it's a place to grab helpful code from 3rd parties. For this lab, you'll need to use NuGet to grab Json.NET.

To add a reference to Json.NET, right-click the References node under your class library project and select "Manage NuGet Packages…". Select the "Browse" link in the top-left of the page that is loaded in the IDE. Enter "Newtonsoft.Json" in the search box. Select the item in the list of search results and in the right panel of the page, select Install.

GRADING: 2 POINTS

Part C-2: Loading the Speech Text

You need to **read the data** in from the supplied **speech.csv** file.

- To add the file to your project, drag the file from File Explorer onto the name of your Console Application in Solution Explorer in Visual Studio.
- Right-click the file in Solution Explorer and select Properties
  - In the Properties, set **Copy to Output Direction** to **Copy Always** OR **Copy if newer**

Create a method called **GetSpeechFromFile**. It should read all the text from the speech.csv file and return it as 1 string.

NAME	RETURNS	PARAMETERS	COMMENTS
GetSpeechFromFile	string	(none)	read all the text from the speech.csv file and return it as 1 string.

Call the **GetSpeechFromFile** method from Main ***instead of*** the GetSpeech.

GRADING: 3 POINTS

COMMON MISTAKES:

- -2 pts: not using the speech returned from GetSpeechFromFile for the dictionary
- -1 pt: not putting StreamReader in a using statement

Part C-3: Save the Histogram

Now you have the information you need to add logic to the menu for the "**Save the Histogram**" option. You want to serialize your histogram data (dictionary) to a save file.

- Ask the user for the name of the save file. **Use ReadString to get the name of the file**.
- **If the name does not have the json extension**, add it to the file name. Look at the Path methods GetExtension, HasExtension, and ChangeExtension to make sure you get the extension set correctly.
- You will need to **serialize** the dictionary in JSON format. Use the JSON.net library.

GRADING: 10 POINTS

COMMON MISTAKES:

- -1 pt: not using ReadString to get the search word from the user
- -1 pt: not ensuring the filename has a .json extension.
- -1 pt: not changing the extension correctly
- -4 pts: not serializing the dictionary
- -4 pts: not serializing in JSON format

Part C-4: Load the Histogram

Now you have the information you need to add logic to the menu for the "**Load the Histogram**" option.

- Ask the user to enter the name of the save file to load. **Use ReadString to get the name of the file.**
- Load the text from the file and **deserialize** the histogram data back into the dictionary. HINT: the type that you serialize is the same type that you deserialize.
- **DO NOT THROW an unhandled exception** if the file does not exist or if the data doesn't match.

GRADING: 10 POINTS

COMMON MISTAKES:

- -1 pt: not using ReadString to get the search word from the user
- -3 pts: not checking if the file exists before loading from it.
- -3 pts: not handling possible exceptions thrown from deserializing the json text

Part C-5: Remove Word

Ask the user for a word to remove. **Use ReadString to get the word to remove**.

Remove the word from the dictionary. If the word is not in the dictionary, show "<word> is not found". (replace <word> with what the user entered). **Do not use ContainsKey or TryGetValue.**

GRADING: 5 POINTS

COMMON MISTAKES:

- -1 pt: not using ReadString to get the search word from the user
- -2 pts: using ContainsKey or TryGetValue before calling Remove
- -2 pts: looping over the dictionary to find the key to remove

LAB 1: RUBRIC

Part A

FEATURE	VALUE
Part A-1: Console Application	2
Part A-2: ReadInteger	10
Part A-3: ReadString	5
Part A-4: ReadChoice	5
Part A-5: Menu Loop	3
TOTAL	25

Part B

FEATURE	VALUE
Part B-1: The Speech	10
Part B-2: Word Counts	10
Part B-3: Show Histogram	10
Part B-4: Search for word	15
TOTAL	45

Part C

FEATURE	VALUE
Part C-1: File Input / Output	2
Part C-2: Loading the Speech Text	3
Part C-3: Save the Histogram	10
Part C-4: Load the Histogram	10
Part C-5: Remove Word	5

TOTAL	30

PROGRAMMER'S CHALLENGE

As with every programmer's challenge, remember the following…

1. Do the rubric first. Make sure you have something to turn in for the assignment.
2. When attempting the challenge, don't break your other code.
3. You have other assignments so don't sacrifice them to work on the challenges.

List Challenge

It would be nice to see the histogram data displayed in a sorted way. Two ways that would be interesting:

- Sort the word data alphabetically by word
- Sort the word data by the count

When selecting "Show Histogram", ask the user which to sort on then show the sorted word data chart.

The challenge is to sort the list of words by either the word or the count. Sorting by the words in a list should be easy. How would you sort by count? Those are stored in the dictionary.