CMSC B113 Computer Science 1 - Spring 2025
Programming Assignment #4

Overview

Computational Linguistics is an interdisciplinary field that applies concepts from the field of computer science, math, and logic to solving problems related to and getting an understanding of linguistics and natural (or "human") languages. This field includes such tasks as speech recognition, machine translation, text summarization, and social media mining, among many others.

In many cases, problems in computational linguistics involve taking a large input, e.g., a text document or audio clip, and breaking it into smaller, individual components. These individual components can then be visualized using a **histogram**, which is a graphical display of data using vertical bars of different heights, typically to show the distribution of values in some range. For instance, a histogram can be used to visualize the frequency or number of occurrences of certain words, individual letters, or combinations of letters within a certain document.

Task: In this assignment, you will write a Java program that displays a histogram of the number of occurrences of letters in Tolstoy's novel *War and Peace*. The histogram that your program produces will look something like this:

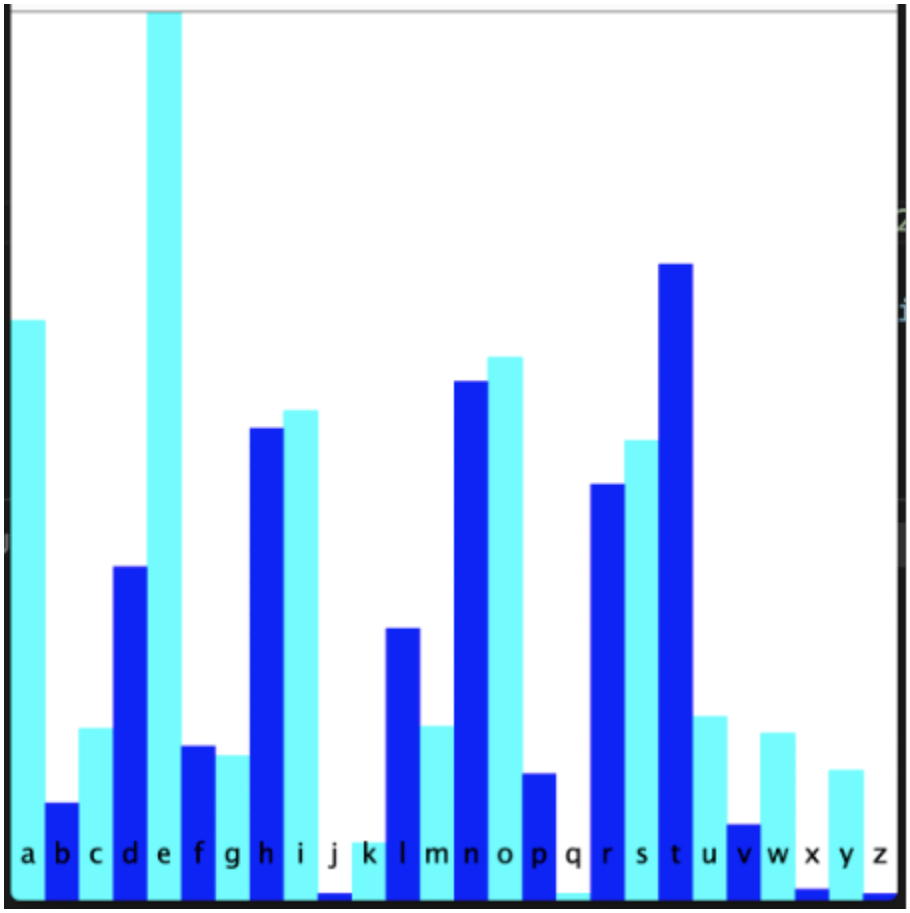

Figure 1: Histogram of letters from Tolstoy's *War and Peace*.

In completing this assignment, you will learn to:

- Work with the Java **`char`** datatype
- Store program data using arrays
- Read from a text file using the **`StdIn`** library
- Display formatted output using the **`StdOut`** library
- Do a data visualization using the **`StdDraw`** library

As you may imagine, there are several steps involved in generating this histogram, but don't worry! This assignment is broken up into separate parts that will help you build up toward a solution to this problem. Our only advice to you is:

Start Early!

We have split up the entire task in six parts. While each part will take no more than 20-30 mins, please allow yourself time to reflect and take a break between each part. Note that in this assignment, you will also be asked to write up a brief report regarding your findings and observations. Be sure to allocate time for doing that write-up, in addition to programming and testing your solution.

Code Understandability and Readability

In addition to writing code that correctly implements the specification, you are also asked to write code that is easy to read and understand.

Part of your score on this assignment will be determined by:

- **Variable naming:** Variables should have meaningful names that indicate what they represent, using full English words or common abbreviations, e.g. "`wins`" or "`votes`" instead of "`w`" or "`vot`".

- **Appearance:** The code should be formatted so that indentation and spacing make it easy to understand which parts of the code are within the bodies of if-statements and loops. Additionally, there should be spacing between variables and operators to make it easy to read each individual line of code.

Please speak with your instructor if you have questions about these requirements!

CMSC B113 Computer Science 1 - Spring 2025
Programming Assignment #4

Part 0: Counting Letters

To begin, write a program, **`LetterFrequency`**, that inputs an entire text, one letter at a time and prints out a count of each letter in the alphabet (i.e. 'a'..'z'). You can assume that all text will be in lowercase letters. Your program should ignore all letters other than those in the range 'a'..'z' (i.e. ignore digit, punctuation, etc.).

To be sure that your program correctly keeps track of the number of occurrences of lowercase letters, try it with the text of Tolstoy's "War and Peace", which is available in the file **`tolstoy.txt`** (posted on the class website). Download this file into the same directory/folder as your Java source code, and then run the program like this:

```
java-introcs LetterFrequency < tolstoy.txt
```

You should see the number of occurrences for each of the characters 'a' through 'z'. Your program should produce the following output:

```
a: 204416
b: 34371
c: 60658
d: 117751
e: 313007
f: 54504
g: 50907
h: 166515
i: 172640
j: 2485
k: 20282
l: 96032
m: 61282
n: 183126
o: 191487
p: 44717
q: 2319
r: 146889
s: 162125
t: 224506
u: 64917
v: 26787
w: 58928
x: 4032
y: 45936
z: 2386
```

Complete this part before proceeding to Part 1. If you are having trouble finishing this part, please visit a TA session or consult your instructor.

Part 1: Scale Values

At this point, we know the number of occurrences of each lowercase letter, but before we can draw the histogram, we need to scale the values so that the letter that occurs the most has a value of 1.0, and the other letters' values are scaled accordingly.

For instance, using the example of "War and Peace," the letter 'e' occurs 313,007 times, which is the most. So, we scale the value for 'e' to 1.0. Then we determine the value for each of the other letters by dividing by 313,007, such that 'a' gets a value of 204,416 / 313,007 = 0.653, since 'a' occurs 65.3% as much as 'e', and so on.

Modify your **`LetterFrequency`** program so that it determines which letter appears the most, and then scales the values for each letter based on the number of occurrences of the most common letter.

Your program should *not* assume that 'e' is always the most common letter: it should find the maximum value in the **`count[]`** array and then use that to determine which letter occurred the most.

Once the program has determined that, it should print out each letter 'a' through 'z' along with its new, scaled value. The value should be displayed to three digits of precision, i.e. three digits after the decimal point, using **`StdOut.printf()`**.

For instance, when running the program with "War and Peace" (tolstoy.txt) as the input, the output for this part should be as follows:

```
a: 0.653
b: 0.110
c: 0.194
d: 0.376
e: 1.000
f: 0.174
g: 0.163
h: 0.532
i: 0.552
j: 0.008
k: 0.065
l: 0.307
m: 0.196
n: 0.585
o: 0.612
```

```
p: 0.143
q: 0.007
r: 0.469
s: 0.518
t: 0.717
u: 0.207
v: 0.086
w: 0.188
x: 0.013
y: 0.147
z: 0.008
```

Be sure you complete this part before moving on to Part 2!

Part 2: Draw Histogram

Now that we know the relative/scaled values for the number of occurrences of each letter, we can use the **`StdDraw`** library to draw the histogram and represent these values visually.

Our histogram will consist of a rectangle for each letter, going left to right ‘a’ through ‘z’, with the height of each letter being equal to the relative/scaled value calculated in Part 1.

To draw a rectangle using this library, you can call **`StdDraw.rectangle(x, y, r1, r2)`** where:

- **`x`** is the x-axis coordinate of the center of the rectangle
- **`y`** is the y-axis coordinate of the center of the rectangle
- **`r1`** the x-axis distance from the center of the rectangle to a vertical side, i.e. half the width
- **`r2`** is y-axis distance from the center of the rectangle to a horizontal side, i.e. half the height

Recall that the default coordinate system in the **`StdDraw`** library is [0.0, 1.0] on both the x- and y-axes. For our purposes, this is fine for the y-axis because the relative/scaled values from Part 1 are all in [0.0, 1.0] anyway, and will represent the height of each rectangle.

However, since we will have 26 rectangles, we can rescale the x-coordinates to [0.0, 26.0] using **`StdDraw.setXscale(0, 26)`** so that the width of each rectangle is 1 and not 1/26. This will make the math a little easier.

Using the above, we can now describe how to draw the rectangle for the i^{th} letter (starting from 0, which corresponds to 'a'):

- **`x`** is *i* + 0.5; this is so that the x-coordinate of the center of the 'a' rectangle is 0.5, the x-coordinate of the center of the 'b' rectangle is 1.5, and so on.
- **`y`** is 0.5 times the letter's scaled value (from Part 2)
- **`r1`** is 0.5, since this is half the width
- **`r2`** is 0.5 times the letter's scaled value (from Part 2), since this is half the height

Modify your **`LetterFrequency`** program so that it uses **`StdDraw`** to draw a rectangle for each lowercase letter as described above. Your program output should look something like the picture shown below:

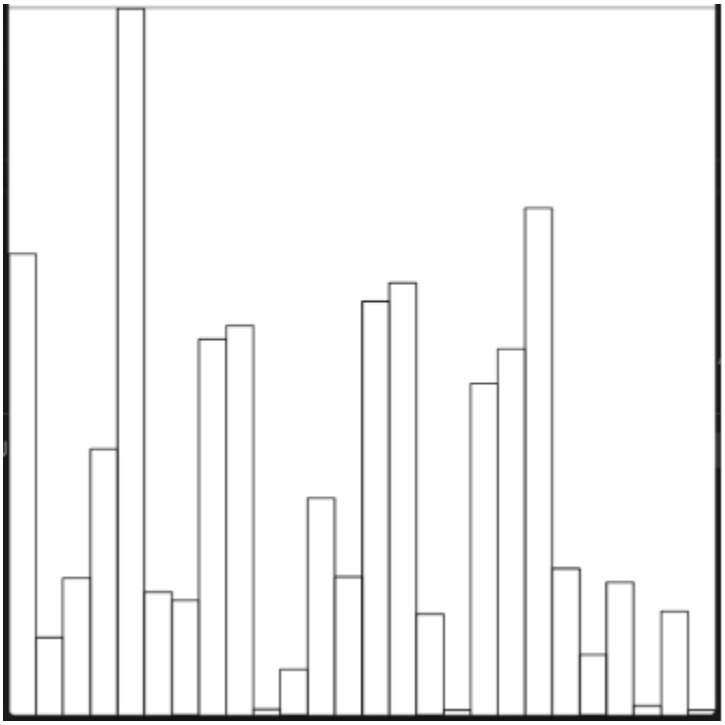

Figure 2: Rectangles representing normalized letter frequencies from *War and Peace*.

Part 3: Add Visual Elements

Now that we have the basic components of our histogram, we can make it a little easier to understand by adding color and text labels.

Modify your **`LetterFrequency`** program so that it uses **`StdDraw.filledRectangle()`** instead of **`StdDraw.rectangle()`** to draw each rectangle.

The inputs to the function are still the same, but this will draw a filled-in rectangle instead of just the outline. You can set the "pen color" that is used to draw and fill in the rectangle to different

colors as follows:

- `StdDraw.setPenColor(StdDraw.BLACK);`
- `StdDraw.setPenColor(StdDraw.BLUE);`
- `StdDraw.setPenColor(StdDraw.CYAN);`
- `StdDraw.setPenColor(StdDraw.GREEN);`
- `StdDraw.setPenColor(StdDraw.MAGENTA);`
- `StdDraw.setPenColor(StdDraw.ORANGE);`
- `StdDraw.setPenColor(StdDraw.RED);`

You may use any colors you like for drawing the rectangles, but no two adjacent rectangles may have the same color. You do not have to use all the colors above; you should simply alternate between two colors to make the chart easier to read.

Last, place each lowercase letter 'a' through 'z' at the bottom of its corresponding rectangle. You can do this using **`StdDraw.text(x, y, s)`** where:

- **`x`** is the x-axis coordinate of the center of the text that is being displayed
- **`y`** is the y-axis coordinate of the center of the text that is being displayed
- **`s`** is the String to display

Note that **`s`** is a String, not a **`char`**. If you have a **`char`** variable called **`c`**, you can convert it to a String using the command **`Character.toString(c)`**.

Your program should produce a histogram like the one shown on the first page of this document (see Figure 1). That histogram was generated by alternating between blue and cyan pen colors, and by placing the letter in the center of the rectangle on the x-axis, and at a height of 0.05 on the y-axis.

Part 4: Modifications

In the last part of this assignment, you will modify your program to analyze another text.

Download the text of another book from Project Gutenberg (http://gutenberg.org/); you can search for the book by its title or author, and once you find the link to the book, download it using the "Plain Text UTF-8" format version.

The text you download will contain uppercase letters and punctuation in addition to lowercase letters. However, your current program only counts the occurrences of lowercase letters and ignores uppercase letters. Modify your program so that it converts uppercase letters that it encounters to lowercase when keeping track of the number of occurrences of each letter. *Hint!* See https://www.tutorialspoint.com/java/lang/character_tolowercase.htm for how to do this!

Then, run the modified program to generate the histogram for the text you selected.

Part 5: Write Report

When you are finished implementing your **`LetterFrequency`** program, write a report in which you include the following:

- The program output for Part 1, i.e. scaled/relative number of occurrences for each letter, sorted alphabetically; be sure to display these to three digits of precision
- The final histogram that your program produced for Part 3 (you don't need the black-and-white one from Part 2): you should be able to include it in your document by taking a screenshot of the histogram or of your entire screen
- The name of the additional text that you used in Part 4, as well as the histogram that your program produced.
- Last, write a short (1-2 paragraph) analysis of the results: based on your findings, which letters are most common? Which are least common? What do you observe when comparing the histograms of the two texts?

Please be sure to include this report along with your complete Java code.

How to Submit

This assignment is due on **Tuesday, March 29 at the start of class**.

Only submit the final version of LetterFrequency.java; you do not need to submit intermediate solutions for each of the separate parts.