CMSC B113 - Computer Science 1
Fall 2020
Homework Assignment #4

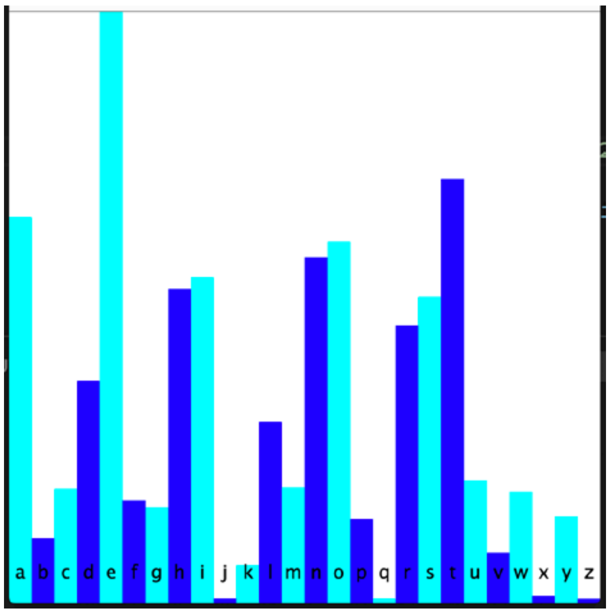
Overview

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.

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:



As you may imagine, there are a number of 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.

For each part, first read the explanation, then read it again! There are a number of details (and hints!) in each part. Then devise a solution to that part by clearly identifying all the inputs and outputs and sketching out the steps and calculations that will need to be performed. Code the steps into a Java program, then compile and run it. Double-check the program's answers with the ones we have provided to confirm that the results are correct.

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.

Learning Outcomes

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
- Draw basic shapes using the **`StdDraw`** library

Code Understandability and Readability

In addition to producing the correct output, in this assignment you are also asked to write code that is easy to read and understand. In particular, 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.

For example, a piece of code that determines the maximum value in an array called **values** should look something like this:

```
int max = values[0];
for (int i = 1; i < values.length; i++) {
    if (values[i] > max) {
        max = values[i];
    }
}
System.out.println("Max is " + max);
```

Not like this:

```
int a=values[0];
for (int i=1;i<values.length;i++) {
if (values[i]>a) {
a=values[i];
}
}
System.out.println("Max is "+a);
```

Please speak with your Instructor if you have any questions about this requirement!

Before You Begin

Because this assignment focuses on keeping track of the number of occurrences of individual letters in the program input, we will be making heavy use of the Java **`char`** datatype, which was presented in Section 1.2 of your course textbook. Review this part of your textbook if you are not familiar with this datatype.

A Java **`char`** variable represents a single character, which can be a letter like ‘a’ or ‘M’; a digit like ‘2’ or ‘8’; or punctuation like ‘+’ or ‘?’.

When working with letters, as we will do in this assignment, it is even possible to do math with **`char`** variables, for instance:

```
    char first = 'k';
    char second = (char)(first + 3);
    StdOut.printf("%c", second);
```

This code will print the letter ‘n’, since ‘n’ is three letters after ‘k’ in the alphabet, and the “%c” format string means to treat the value as a **`char`**.

Similarly, you could have something like this:

```
    char letter = 'g';
    int value = letter - 'a';
    StdOut.printf("%d", value);
```

This code will print 6, since ‘g’ is six letters after ‘a’.

Be sure you understand this before starting, and ask your Instructor or post a question on Piazza if you are not sure how this works!

Part 1: Count Characters

Start by creating a Java program called **LetterFrequency**.

The first step in building the histogram is to read the input text one character at a time, and to keep track of and then display (to stdout) the number of occurrences of each letter between ‘a’ and ‘z’.

For instance, if the input text were “java yeah!”, the program should indicate that the letter ‘a’ occurs three times and that the letters ‘e’, ‘h’, ‘j’, ‘v’, and ‘y’ each occur once. It should ignore the blank space between “java” and “yeah!”, as well as the ‘!’ character, since those are not letters.

Your program should display the number of occurrences for all 26 letters, even if a given letter does not appear. For instance, in the above example, the output should look like this:

```
a: 3
b: 0
c: 0
d: 0
e: 1
. . . output omitted but you get the idea ...
x: 0
y: 1
z: 0
```

For simplicity, you may assume that your program only needs to keep track of lowercase letters, i.e. that any uppercase characters in the input have already been converted to lowercase, and your program only needs to track the 26 lowercase letters used in English.

The following hints should help you with this part of the assignment:

- You can use **`StdIn.isEmpty()`** to determine whether or not there is more input text to read.
- You can use **`StdIn.readChar()`** to read a single character from the input and store the character in a **`char`** variable.
- Use a 26-element array of counts to keep track of number of occurrences of each letter. *Do not define 26 different variables to store the number of occurrences of each letter!*
- You should store the number of occurrences of the letter ‘a’ in index #0, the number of occurrences of ‘b’ in index #1, and so on. You can use the “math with **`char`** variables” advice from above to figure out the corresponding array index for each letter.
- To determine whether a **`char`** variable holds a lowercase letter, you can determine whether it is in the range [‘a’, ‘z’] in the same way in which you would determine whether an **`int`** variable is in a certain numerical range by using comparison operators.

To test your program with the default source for stdin, i.e. the Terminal, then follow these steps:

1. Run the program with the command `java-introcs LetterFrequency`
2. Type characters into the Terminal. Don't worry about putting spaces between them; your program should be able to read each character one at a time. Your program should only be counting the occurrences of lowercase letters, i.e. 'a' through 'z'.
3. When you are done entering characters, you can indicate that there are no more characters to read (so that `StdIn.isEmpty()` will return true) by typing Ctrl-D (that is, pressing the Control and D keys at the same time) if you are using a Mac, or Enter then Ctrl-Z then Enter if you are using Windows.
4. Your program should then display the number of times you entered each character from 'a' to 'z'.

Once you have tested your program by entering characters into the Terminal, try it with the text of "War and Peace", which is available in the file `tolstoy.txt`. Assuming that `tolstoy.txt` is in the same directory as your Java source code, all you need to do is run the program like this:

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

Then 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
```

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

Part 2: 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 scales the values for each letter based on the number of occurrences of the letter that appears the most, and then prints out each letter 'a' through 'z' and its new, scaled value to three digits of precision, i.e. three digits after the decimal point.

For instance, when running the program with "War and Peace" 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 3!

Part 3: 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 2.

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

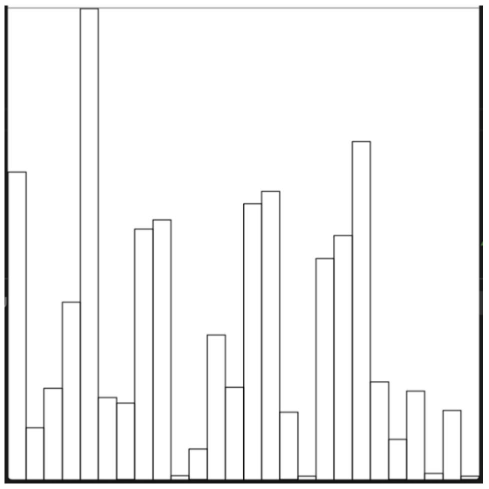
Recall that the default coordinate system in the `StdDraw` library is [0.0, 1.0] on both the x- and y-axes. For our purposes, this is fine for the y-axis because the relative/scaled values from Part 2 are all in [0.0, 1.0] anyway, and will represent the height of each rectangle.

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 is '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 here →



Part 4: 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 may simply alternate between two colors to make the chart easier to read. More than two colors will not be good.

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)`**. Note also that if you are doing the "math with **`char`** variables" approach as described above, the result will be an **`int`**, which you will need to cast to a **`char`** before using **`Character.toString`**.

Your program should produce a histogram similar to the one shown on the first page of this document. 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 0.5 on the y-axis.

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. the number of occurrences for each letter, sorted alphabetically
- The program output for Part 2, i.e. scaled/relative number of occurrences for each letter, sorted alphabetically; be sure to display these to three digits of precision
- This final histogram that your program produced for Parts 3 and 4; you only need to include the final one from Part 4, and you should be able to include it in your document by taking a screen shot of the histogram or of your entire screen
- 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 surprises you?

Extra Credit

Run your program on one other text. You can download texts from Project Gutenberg (http://gutenberg.org/). You will need to either pre-convert the text into all lowercase, or modify your program to account for the conversion. Once done, compare the two histograms for any similarities or discrepancies. Include your results in your report.

Submitting Your Solution

To submit your work, copy/paste the source code of your Java program, along with the report described in Part 5, into a *single* PDF, and submit the PDF in the "Assignment #4" subfolder of the Dropbox folder that your instructor created for you.

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

Please be sure to put all of your code into a *single* document and please only submit a PDF, not a Microsoft Word .docx file or the .java source files. Your work will not be graded if you submit anything other than a PDF file.