**Rust Programming Lab #6 13th September 2022**

**Letter Frequency**

**Exercise 1**

Determining the frequency of occurrence of alphabetic letters in ‘usual’ text has many uses. For example, this table was used by to design ‘Morse Code’ (named after Samual Morse, accredited with designing a practical telegraph, 1837).It is a scheme of dots and dashes used in telegraphy before we had optical fibres:

|  |  |
| --- | --- |
| Table  Description automatically generated | A picture containing shape  Description automatically generated |
| Letter frequency | Morse Code |

As you can see, frequent letters, E and T, are assigned to single ‘dots’ or ‘dashes’, with the next frequent ones (A, I, N, M) assigned to double patterns and least frequent ones (J,Q,X,Z) assigned to four element patterns. Such tables can also be used to decipher simple transposition cryptographic codes (the most frequently occurring code must be an ‘E’, *etc.*) and in games: Scabble players will recognize the points assigned to letters – frequent letters, E,A,T,R,N, etc score 1 point, whereas Q and Z score 10.

For this assignment, determine the frequency of letters in Thai (read Thai = your native language, ‘Thai’ is shorter 😉). Select a text, pass it into your program, count the occurrences of each ‘letter’ and build a frequency table, listing the fraction of occurrence of each letter as % or frequnecy ∈ (0,1). Count consonants, vowels, tone marks (อ่ อ๋ อ๊ อั) and special marks (ดัด, etc) as ‘letters’ for this exercise.

Steps:

1. Check the Unicode table for Thai characters and determine the range – from lowest to highest. Use this to determine how many entries your table will need.
2. Define a struct for entries in the table: it should have a **char**, count (**u64**) and relative frequency (**f64**) entries.
3. Define an array or a vector (which is best for this application?) to hold the table.
4. Read a piece of Thai text (the sample code will read stdin one line at a time), and increment entries in your table for every letter found in that text.
5. Since your text will certainly contain non-Thai letters (number, Latin characters, etc) and even some spaces, add a table entry to count them too. This will also allow you to check that your table has reasonable counts!
6. Once you have read the input and built the table, calculate the relative frequencies (as % or fraction in (0,1).
7. Generate a readable table and send it to stdout or a file if you follow the example in a previous lab.
8. Once your program is working satisfactorily on a small test text, find a much larger piece of mostly Thai text and run your program on it. A much larger piece should have at least 5000 characters. The larger the piece of text, the more likely your frequencies are reliable and useful.
9. Working with at least one partner (group of 3 maximum – a larger group will require too much coordination for this short report), find some different texts – one for each partner – and compare the frequencies you obtained.
10. Write a short report, including your table and a report on differences between your table and your partners’ tables. For this purpose, the frequencies of the most common letters only are likely to be interesting: low frequency letters will show wide variations and not say anything significant. Perhaps comment of the 10 most frequent letters in your text and those of your partners. However, the complete absence of some letters *vs* their appearance in other texts may be interesting.
11. Your report should note the source of your text, e.g. URL, name of author and book, .. Maybe you have a suitable essay for any other purpose – describe it.

### Notes

1. You may find texts from any suitable source: newspaper articles, text books, novels, scientific papers, etc. Word documents will contain many formatting marks, so should be saved as *plain text* before using them. Similarly images will give your program a headache (unless you add a lot of code to skip them!), so edit them out of the text before using it.
2. You can use a string iterator to read the characters on each line. Remember to use

**for c in buf.chars() { … }**

to read characters not bytes.

1. You and your partner(s) should choose texts from ***different*** sources, e.g. if you choose a novel, your partner should choose a newspaper article or paper or … This may show interesting differences in word usage (and therefore letter frequencies in different sources). However, only some simple comments on the differences are needed. If you find this interesting – maybe it’s a longer project that may lead to an interesting journal article - later 😊.
2. Separate your code into several small functions (see more about this in the next lecture). A program that contains only one long main function will earn very low marks. Divide the tasks into small functions.
3. You can use **stdin** to read your text and **stdout** to generate your report:

**cargo run <input.txt >report.txt**

1. Use **eprint!(..)** or **eprintln!(..)** to generate diagnostic output that you do not want to include in your final report.
2. This function will read one line of text from your file. At the end of the file, the returned **String** will have 0 length, so add a **break** from your reading loop and jump to your function to print your report.

**fn read\_text\_line() -> String {**

**eprint!("Text to be processed ");**

**let mut buffer = String::new();**

**let result = io::stdin().read\_line(&mut buffer);**

**eprintln!("Buffer ({}) [{}]", buffer.len(), buffer );**

**buffer**

**}**

**Prepare a short report and submit on goEdu**

[ELEMENTARY SYSTEMS PROGRAMMING [Section 2][65-1-01286120-2]](https://goedu.kmitl.ac.th/course/view.php?id=10151)

Topic 1

**before the next lab (Sept 20. Your deadline is 3pm, Sept 20.**

**Website: kris.kmitl.ac.th/clinic/Courses/Rust/**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attendance** | **01286120** | **Elementary Systems Programming** | **6 Sep 2022** |

|  |  |  |
| --- | --- | --- |
| **Name (Thai script\*)** |  | **Student ID** |
| **(Latin characters -  as you enrolled)** |  |
| **\****Please write clearly: practice for one farang who is trying to improve* **😉** | | |

|  |  |  |
| --- | --- | --- |
| **Check 1** | Sketch the **struct** of the structure you chose | TA |
|  | |
| **Check 2** | Array times | TA |
| Array size used:  Average time to add/element  Average time to access/element | |
| **Check 3** | Vector times | TA |
| Vector size used:  Average time to build/element  Initial capacity (at least 3 values, including max) Average time  Average time to access/element  Same for all initial capacities? | |
| **Check 4** | Vector building | TA |
| List capacities as they were changed | |
| **Check 5** Access times using iterators  Array  Vector | | TA |
| **Check 6** Short summary of your results. | | TA ( \_ / 5) |