**Rust Programming Lab #11 25th October 2022**

**User friendly user input**

One of my frustrations with (supposedly friendly) user input is caused by programs that insist on entering long strings of digits, e.g. telephone numbers, credit card numbers, etc, as ‘numeric digit only’ strings, for example, strings which don’t allow you to enter a phone number as ‘086 224 9197’, i.e. insisting the you cannot include the spaces, which many of us use to help remembering (and checking it after you typed it) an otherwise meaningless string of digits.

**Exercise 1**

Write a user friendly function that lets you input a telephone number in a form that you use to remember it. So it should

1. accept (and ignore) spaces (any number of them)
2. a number with or without a leading ‘0’
3. an international number, preceded by ‘+’.

You should define a structure, that, after decoding the string that you typed, classifies it, using an enum, as local, international, likely\_scammer or special. Then, if it was preceded by ‘+’ take the next two digits as the international direct dial (IDD) code, e.g. 66 for Thailand, and store it as appropriate integer. If there was no ‘+’, it should assume the IDD code is 66. Naturally, since you want this code to work when you get that great job in Australia, the default IDD code should be easily changed. Then after checking that it has the correct number of digits (see rules below) stores the number itself.

**Special rule:** Thailand has defined an IDD code for calls originating from internet sources - +657: so you should recognise this as ‘likely\_scammer’.

**Personalization rule**: Since some numbers are more important than others, you should allow them to marked as ‘never to be forgotten’ or ‘special’, e.g. your mother’s number. To keep this code simple (for the moment, you can expand it later), accept **one** (optional) string, e.g. ‘mother’ at the end of your number and mark the number as a ‘special’ one, e.g. ‘legal’ examples are

+66 86 224 5678

+61 20 407 1915 mother

072 2456 123

**Number of digits rules:**

1. An international number, preceded by ‘+’ should have exactly 11 digits.
2. A local number preceded by ‘0’, should have exactly 10 digits.
3. A local number with no preceding ‘0’ should have exactly 9 digits.

**Readable output:** Add a function that prints a telephone number in a readable form, e.g.
+(66) 862249213
Minimal formatting of the output is OK, *don’t go overboard*, just make it readable -- not forgetting to flag special numbers 😊.

**Testing your code**

Make a test file that contains a collection of valid and invalid numbers and check that your program decodes and prints valid ones and prints a reasonably useful message that reports incorrect input. As a minimum, it might just repeat the input string and add ‘is invalid’.

**Notes and hints:**

1. You can start with a skeleton borrowed from the letter frequency lab (#6).
2. Your code must be designed for change – see Exercise 2.
3. Read each line as a string, using the fn read\_text\_line() -> String from lab 6. More elaborate strategies are OK, but not necessary and don’t gain extra credit.
4. There are two strategies for decoding the character string to an integer
	1. Start with a zero sum: loop, multiplying the current sum by 10, adding the value of the next digit and continue to the end or
	2. push each valid digit into a vector as you read it, left to right, then, to convert the string into a number, you simply pop each digit, in turn, multiplying by **10n**, for **n∈{0,11}**, each time and summing, to convert your digit string to a number.
5. If you study exercise 3 first, you may find a way to solve the credit card problem that you can use in the telephone number problem 😊.

**Prohibitions:** Rust provides several functions to make this exercise easier. **DO NOT USE** them to handle this problem: *code it by yourself*. At the end, those of you who have discovered some of these ‘useful’ functions in various crates may answer the (optional) question in **Exercise 4**.

**Exercise 2**

If you designed your program well, you can trivially convert it to read telephone numbers in official Thai numbers and other languages, including Chinese characters. Look up the relevant Unicodes and show how to change your program to read official Thai numbers.

**Exercise 3**

I find a similar problem with credit card numbers. Although they are printed with spaces to make them easier to remember (and understood – the first four digits are usually a code for the issuing company, e.g. Visa, Mastercard, …), but I have seen software that has the same frustrations.

Having solved the telephone number problem, you should be able to cut out the section that decodes number strings and build a ***separate function*** that decodes digit strings into numbers. CC numbers are somewhat simpler, most have exactly 16 digits. Here the credit card struct has only two components, the bank code (4 digits) and the number.(12 digits). So your ‘input\_CC’ function should return the decoded credit card number.

**Exercise 4**

Optional question for some who have bravely struggled through many Rust examples and the documentation that goes with them.

**Answer the questions on the attendance sheet, have a TA sign it off.**

**This lab will be marked from results on your attendance sheet. The TAs will confirm that your code was acceptable. Read the error codes reported for Lab 6 – see below.**

**Again, if Rust is disturbing your peace, you are STRONGLY RECOMMENDED to show your proposed solution to the main decoding function with one of the TAs BEFORE proceeding to write much more.**

**You should also note that several questions are related to good coding practices, the assignmement error codes – linked to Lab 6, Sept 13 on the Rust web page – may help**

**https://kris.kmitl.ac.th/clinic/Courses/Rust/Lab/AssignErrorCodes.docx**

**IF you are confident: You may skip the checkpoints and fill them in at the end.**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Attendance**  | **01286120**  | **Elementary Systems Programming** | **18 Oct 2022** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name (Thai script\*)** |  | **Student ID****65011**

|  |  |  |
| --- | --- | --- |
|  |  |  |

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

|  |  |  |
| --- | --- | --- |
| **Ex 1** | What structure(s) did you use for telephone numbers? | TA |
|  | How did you convert your program to work in Australia? |  |
| **Ex 2** | How did you make your program handle letters in another script, e.g. official Thai numbers? |  |
|  |  |  |
| **Ex 3** | What structure(s) did you use for credit code numbers? | **TA** |
| **Ex 4** | ***(Optional)***If you found Rust crate functions that helped to solve this problem:1. Were they really useful?
2. Were you able to code the program yourself faster than understanding the Rust documentation 😊?
3. If yes, for (b), which functions were useful and intelligibly documented?
 |  |