**Rust Programming Lab #12 1st November 2022**

**Traits**

Understanding Traits is essential to be able to follow much published Rust.code 😊, so in this exercise, we will create some of our own.

Assume that we have a collection of people in our organization: they all share some common **traits**, e.g. they have a name, an age and a location. However, they have many different functions and attributes associated with that function, so that we should define struct’s for each person’s function, e.g. managers will have a list of employees that they manage, shift workers will have a shift (e.g. morning, evening, night), software engineers will have a list of skills, e.g. Python, C, C++ (even Rust 😊), ….

**Steps**

1. Define a location – for simplicity, use a GPS Lat/Long - borrowing from a previous exercise is allowed 😉: formally, engineers call this ‘reuse’ – a desirable trait.
2. Add a function that prints readable GPS positions (see 1).
3. Add several (3 is enough) **struct**s that define occupations. Note that all must have the common attributes, name, age, location.
4. **One** of these occupations must have a list as one of its attributes.
   1. In the answer section, explain why you chose the data type of the list.
5. Define an ‘**Employee**’ trait, that has methods to extract the attributes, e.g. find the age or find the location.
6. Add the functions for this trait.
7. A slightly more challenging task uses the fact that if a struct has the ‘**Display**’ trait, it can use the ‘**{}**’ format to display the struct’s value, *e.g.*

**struct LL { … }; let p = LL{…}; println!(“Pos is {}”, p);**

* 1. Look up the documentation for the **Display** trait.
  2. Observe that it has a single function, **fmt**, which is invoked when the format directive ‘**{}**’ appears in a format string.
  3. Add this function as a trait to your Lat/Long structure (copy and paste recommended here – you don’t need all the details of the fmt (at this stage, anyway 😊). Copy the basic format, from your previous function to print a position, as needed.

1. Now make a simple function that reads Employee details from a file.
   1. Use the ‘named association’ idea[[1]](#footnote-1) and structure your file like this

**Role: Manager  
Name: John Smith  
Age: 35  
StaffList: Bob Brown, Alice Green  
Pos: 14.34567, 100.45675**

**Role: SoftEng**

**Name: Jim Jones  
Age: 24  
Skills: C++, Python, Rust  
Pos: 14.34567, 100.45675**

* 1. You can choose simple labels to name the attributes. Note that labels without spaces may simplify your code.
  2. Ignore spaces starting labels.
  3. Choose separators, e.g. ‘:’, ‘,’. Two will suffice (keep it simple!).
  4. Make sure the file format is easily changed – the managers in a different company may insist on ‘;’ as a separator instead of ‘,’ and want different position labels (or change the language to ภาษาไทย 😊.
  5. In your final report, explain (with details) how you will maintain this program, *e.g.* change the language used for keywords from English to ภาษาไทย or any other language of your choice!
  6. Note that it should require no changes to read names in any language.

1. For simplicity, you can read this file from standard input.
2. As you read the file, print out the employee details and calculate the average age of employees.   
   Notes:
   1. Rust does not appear to support class hierarchies as in C++, Java or Python, but you can achieve the same effect by adding an ‘**employee\_type**’ attribue using an enum. Further, the trait may define function implementation, e.g. **fn getEmployeeType() -> EmployType { .. }**  
      which is ‘inherited’ by all structs.
   2. There is a way to make an array (or vector) of an enum which includes instances of individual (but different) types. The brave among you might try this, but it is ***optional*** at this stage. Simply printing employee details as you read them is OK, just make a simple enum, added as an attribute.

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

**Part of 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 *again*.**

**Then submit a short report AND your code for final assessment for this lab.**

***Your report should be a miniature user manual, it should list the keywords used and the separators allowed. Miniature means .. keep it short and concise .. and you will not be tested for English grammar 😊***

**Use EduGo. Submit in two weeks, before Nov 15.**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Attendance** | **01286120** | **Elementary Systems Programming** | **1 Nov 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** | Sketch the structs used for emloyees | TA |
|  | Show the details of the struct including a list  Explain why you chose the data type for the list |  |
| **Ex 2** | Show the implementations of the trait functions to the TA for signoff | **TA** |
| **Ex 3** | Did you implement the Display trait correctly for your position structure?  Show that you could add **println!(“Pos {}”, pos)** to your code. | **TA**  Trait  Prints OK |
| **Ex 4** | ***Now*** .. Explain how you plan to make your program easy to maintain, *i.e.* change to use different keywords and separators? Just give brief details here.  *Complete the code and submit it before the deadline*. | **TA** |
|  | Complete a short report and submit your report and code by Nov 15. |  |

1. Note these labelled fields formats are (somewhat) similar to those used in XML (Extended Markup Language) files, which is also similar to the HTML markup used for web pages. Most web browsers will read correctly marked up XML files directly. If you are very brave, you will find several Rust crates which parse XML files: you may try them as an alternative, but you may need a large coffee supply to learn how to use them 😊. [↑](#footnote-ref-1)