**Rust Programming Lab #3 16th August 2022**

**Standards***and more*

**Rust by example** **- Learn by trying**

**Exercise 1 Standards**

In lectures, it was noted that software engineers distinguish themselves from hackers, not by remembering trivia, but by being able to find important data quickly.

Standards, by their very nature, are long and detailed – and dull reading. However they are essential for ensuring compatibility, interworking and quality in many aspects of our life and certainly in engineering.

One important standard is the **digital representations of floating point numbers**, the applicable standard is IEEE 754. No one should need to remember the details, *e.g.* how many bits in the mantissa of a double, but often engineering calculations will need to know the expected accuracy of a result, so you should know where to look this up when needed. Your can find the IEEE 754 standard in many websites, textbooks, manuals, etc. Find a website that lists the details for single, double and quadruple floating point numbers. Copy this website to the answer sheet.

**Exercise 2 Character codes**

Similarly, to ensure efficient interchange of documents in different languages, unicodes are defined for representing characters in most languages. Find a website that describes the unicode standard, *e.g.* how characters in English, French, Chinese, Thai, *etc*., are represented. You will probably need to find two websites – one that describes unicodes in general and another that has Thai character unicodes. *If Thai is* ***not*** *your native language, answer this question (and the next exercise!) for your native language 😊.*

**Exercise 3 Thai in your digital document**

In this exercise, you can learn the basics of strings of characters, by following some examples: more formal details will appear in lectures later.  
In Rust, you can declare and initialize a string using a list of characters in double quotes, *e.g.*

**let example = “Example string”;**

**let my\_name = String::from(“\u{0E01}\u{0E02}\u{0E31}\u{0E01}");**

and print the string on your terminal with

**println!(“{}”, my\_name);**

However, you will find that the default Windows DOS command line (**cmd**) cannot interpret unicodes sensibly! However, if you write a unicode string to a file, and reload the file with WordPad (or probably most other editors), the Thai characters will be displayed correctly 😊. The problem is that DOS (dating back to 1980 or so) is monolingual and only knows the 8-bit ASCII character set: more modern editors seem to understand unicode though 😊.

Download the example code for this assignment and edit it. Using the unicode tables that you found in Exercise 2, initialize the string with your name and print it out to check whether the combination of Windows and the Rust compiler will actually generate something reasonable – look in the **name1.txt** file though 😊.

Now you can build this into a small function, that prints your name and student number, at the top of *every assignment* that you submit for this course. You have already seen a Rust function – every program starts with a function called **main**. You can build a function that prints your name, number, date, *etc*., at the top of every output.

**fn write\_my\_name(my\_name:String, s\_id:i64) ->  
 std::io::Result<()> {**

**let mut f = File::create("name1.txt")?;**

**let my\_name = "\u{0E01}\u{0E02}\u{0E31}\u{0E01}";**

**let s\_id:i64 = 650000011;**

**let contents = ……..**

**let e = if let Err(e) =   
 write!(f,"{}", contents)**

**{ e } else { todo!() };**

**Ok(())**

**}**

The function starts with the reserved word **fn**, followed by a ***name,*** a ***parameter list*** in **(),** a return type following the **->** operator and a block, **{ … }.** This function returns a **Result** structure (used to capture returned input or output error data) and has no parameters, the argument list is empty (). At the end of the function, you will see the **Ok(())** – the **Result**. Note that you will need to build a string using the **format** function and then write it to your file.

In the Exercise 3 box, fill in the unicodes for **your** name! *Your first name will be sufficient if your name has more than 10 or so characters.*

**Exercise 4 Adding a date and time to your output.**

First, you will need to add the **chrono** crate to your manifest. Look it up in the crate registry, find the latest version number and add it to your manifest.

The **chrono** crate provides two types of time: local and Coordinated Universal Time (UTC), the international reference time. UTC generally replaces the old Greenwich Mean Time, based on the time at Greenwich, England at 0° longtitude – see <https://en.wikipedia.org/wiki/Coordinated_Universal_Time>.

Commands

**let utc\_here: DateTime<Utc> = Utc::now();**

**let local: DateTime<Local> = Local::now();**

will return a structure which contains the times (very precisely!). Look up the documentation for Rust’s DateTime and find the functions which will extract the year, month, day, etc, from **utc\_here** or **local**, e.g.

**let year = utc\_here.year();**

Now you can add a date and time to your report – by adding a few more statements to your **write\_my\_name** function**.** List the functions that you used (*you don’t need to add them all 😊*) to the Exercise 4 box. Expand the **format** command to include the extra date data.

# There is also a [std](https://doc.rust-lang.org/std/index.html)::[time](https://doc.rust-lang.org/std/time/index.html)::[SystemTime](https://doc.rust-lang.org/std/time/struct.SystemTime.html) function in the std library: you may look it up and use it instead. Indicate which functions you used in the answer box.

**Note** both modules allow you to also find the *difference* between two times, *i.e.* the current time and some reference time: maybe you can use this later to time some of your code 😊.

**Fill in a sample of your results in the attendance record and hand it in.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attendance** | **01286120** | **Elementary Systems Programming** | **16 Aug 2022** |

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

|  |  |
| --- | --- |
| **Exercise 1** | Standard for floating point number representations |
| Name of the standard:  url:  *Note there are certainly many places to find this. Try to find an* ***authoritative*** *site, but any one will do.* | |
| **Exercise 2** | Unicode character representations |
| Name of website:  url:  Website for Thai characters  Name:  url: | |
| **Exercise 3** | Unicode characters for your name |
| *(First ~10 if your name is really long 😊)*  Were you able to print it sensibly in a file and display it? *YES / NO* | |
| **Exercise 4** | Useful date and time functions *(Don’t list them all – show us where to find the rest!)* |
| URL used: | |