**Rust Programming Lab #8 4th October 2022**

**Speeding Up Engineering Calculations with Look Up Tables**

Please note the instruction to submit Word files (I can easily edit PDF to add comments) AND your code files.

Many engineering calculations need mathematical functions (trigonometric, logs, ..). These are usually calculated by summing repeated series, *e.g.*

A picture containing text, clock

Description automatically generated

These functioins converge to the true values if they use sufficient terms in the sum, but this is necessarily slow, especially if your ***x*** is an **f64** and you expect a result accurate to 1 in 1015! This accuracy may be needed if you are trying to hit that asteroid that is 90,000 km away (<https://www.nasa.gov/press-release/nasa-s-dart-mission-hits-asteroid-in-first-ever-planetary-defense-test>) but possibly overkill if your robot needs to hit a ping-pong ball that is 1 m away. However, if it is controlled by a microprocessor (e.g. an Arduino at 16 MHz), it may not have enough cycles to calculate that final bit in an f64 value, but a much lower accuracy will suffice.

**Interpolated look up tables** will generally provide the *needed* accuracy using only a few multiplications and thus few cycles. The trade-off is in time to compute the look up tables, but you only need to create them once.

If you create a 91 entry table, t:[f64;91], containing the value of sin(x) for every degree (in the first quadrant), then if you need to calculate sin(63.235°), then you look up (*not forgetting to convert to radians first!*)

sin(63) = x0 = t[63] = 0.891006524188368343  
sin(64) = x1 = t[64] = 0.898794046299167482

then

sin(63.235°) = x0 + (x1-x0) × (63.235° - 63)/(64 – 63)

which you will find is not too far from the value computed by the library **sin( )** function. This is known as ***linear interpolation*** and works well if the function is smooth. Most trig, exponential and log functions are sufficiently smooth.

**Tasks**

1. Build your look up table, **lut**: start with a small one, e.g. 10 entries for 0, 10, … , 90. You can expand it to a much larger one later.
2. Fill it with values for **sin( )** computed by the library function. Don’t forget to convert to radians 😊.
3. Create a function **lut\_sin**(lut, x) that computes sin(x) by interpolation in your function.
4. Create another array of test values. If the test values are the same as the values used to create the table, then you will get a perfect answer, *so make sure that your test values range from 0 to 90*, and include some non-integer values, e.g.

[0.0, 0.234, 5.856, 10.100, 19.34, …..]

1. A good test set includes some known values, e.g. 0, 30, 45, 60, so that you can quickly verify that your function is correct. ***Always*** *include zeros or nulls in your tests!!*
2. In a loop, compare the ***sin*** values computed from your function with the library one.
3. Print these differences as a table.
4. Note that for **f64** values, the precision is ~1 in 1016, so more than 16 decimal digits in your report are just noise! Rust will allow you suppress them using

**println!(“x = {:20.16}”, x );**

which prints only 16 digits after the decimal point in a 20 character width field.

1. When your function is working for a small table, e.g. 10 entries, generate a larger table, e.g. 101 entries, and report the differences for the same set of test values.
2. Run the program for a much larger table, e.g. 1001 entries, report the differences now.
3. Make a *short* summary of your conclusions *on the lab sheet* and ask the TA to sign it off.
4. Before next week:
   1. The directions here guide you to make a function that works for 0-90°. Brush up your trigonometry and augment your function so that it handles all angles, +ve or -ve.
   2. Now add a function that calculates cos. Can you use the same table?
   3. Test one other function, e.g. log, exp or tan, and report its results.
   4. Did you notice any difference with the behaviour of log or exp or tan?
   5. Make a full report (using the report template) and submit it via **goEdu** before next week’s lab.
   6. Your submission should include
      1. your report in Word (**not PDF!**) ***and***
      2. your program code (this may be several files – don’t forget to save your first experiment code, before you modify it for the next experiment!).
   7. Marks will be deducted for poor programming practices, e.g. magic numbers, 1000 line functions, … Ignore bad advice from **rustc** to remove unneeded parentheses 😊!

**Tick off the exercise on the lab sheet (✓ or 🗶 =** *if didn’t get it***) or answer the question asked, ask a TA to sign it off and hand it in.**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Attendance** | **01286120** | **Elementary Systems Programming** | **4 Oct 2022** |

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

|  |  |  |  |
| --- | --- | --- | --- |
| Exercise | Comment | Completed  (Your check) | **TA** |
| 1 | **Brief** summary of your observations for ***sin*** with various table sizes.  *(Your report will have a more complete summary.)* |  |  |
| 2 | How will you handle ***cos*** ?  *(Implement it for your report.)* |  |  |