600086 Lab Book – Daniel Bates

# Week 1 – Lab A

Date: 2nd Feb 2022

# Q1. Installs

### Question:

Install Rust and the stated extensions for visual studio code

### Solution:

N/A

### Test Data:

N/A

### Sample Output:

N/A

### Reflection:

Just getting things ready for future labs

### Metadata:

Set up

### Further Information:

N/A

# Q2. Hello World

### Question:

Create a new project folder. Use the visual studio code terminal to first change the location to the new folder. Use “cargo init” to setup a new project then use “cargo run” to run the project.

### Solution:



### Test Data:

N/A

### Sample Output:

### 

### Reflection:

Just the basic stuff

### Metadata:

Hello World

### Further Information:

Useful resources for learning Rust:

<https://www.youtube.com/watch?v=zF34dRivLOw>

<https://doc.rust-lang.org/stable/book/>

# Week 2 – Lab B

Date: 9th Feb 2022

## Q1. First threads

### Question:

Replace the synchronous call to your function with an asynchronous call.

### Solution:

### Test data:

n/a

### Sample output:

### Reflection:

This is threading 101

### Metadata:

Threads

### Further information:

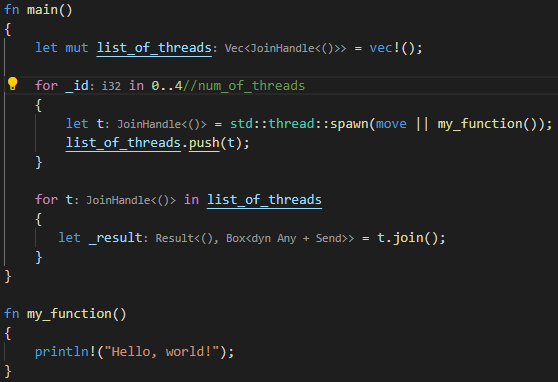
N/A

# Q2. Joining threads

### Question:

Add code to create an arbitrary number of threads and then join them.

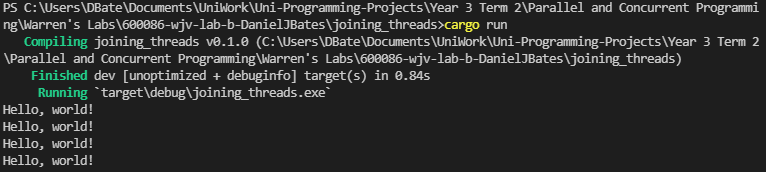
### Solution:



### Test Data:

N/A

### Sample Output:



### Reflection:

Rust is a bit complicated to wrap my head around. Need to do some reading on the language a bit for next lab

### Metadata:

Joining threads

### Further Information:

N/A

## Q3. Experimentation

### Question:

experiment with giving the threads items of work, as well as altering the number of threads used

### Solution:

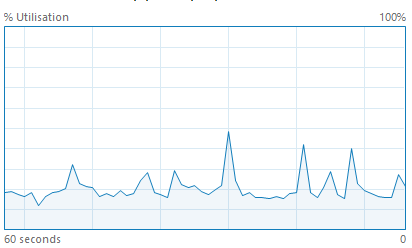
N/A

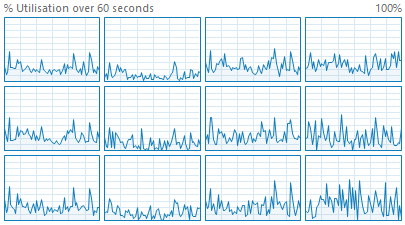
### Test data:

N/A

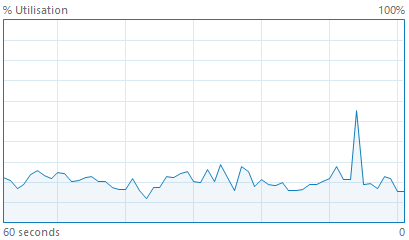
### Sample output:

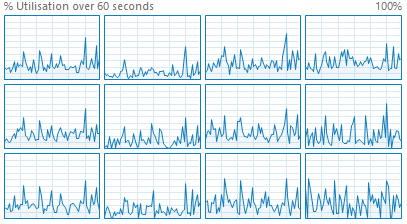
**512 threads (Hello World) –**



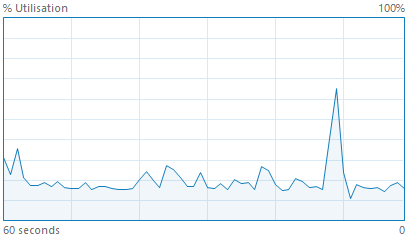


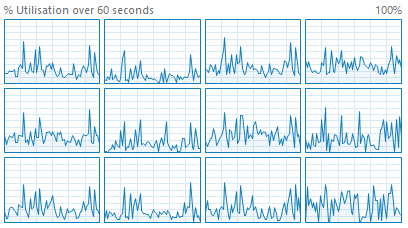
**1024 threads (Hello World) –**



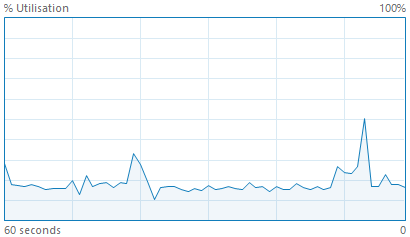


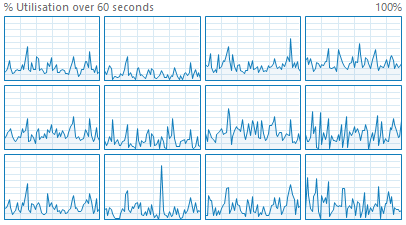
**2048 threads (Hello World) –**

****

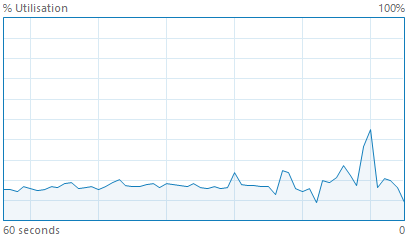
****

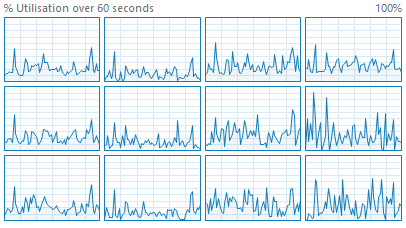
**512 threads (1 + 2 + 5 \* 25) –**



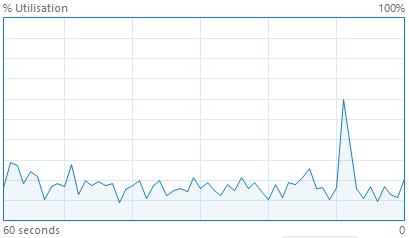


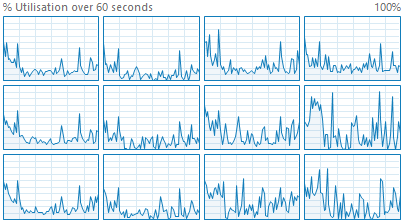
**1024 threads (1 + 2 + 5 \* 25) –**

****

****

**2048 threads (1 + 2 + 5 \* 25) –**

****

****

### Reflection:

Graphs are a bit messy need redoing.

There is a spike whenever the application is run

### Metadata:

N/A

### Further information:

REDO Graphs

# Week 3 – Lab C

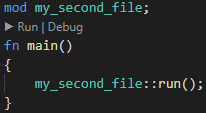
Date: 16/02/2022

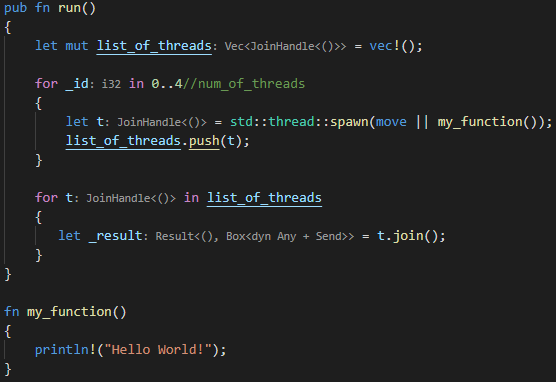
## Q1. Multiple Rust Files

### Question:

### Move your thread main function, from the previous lab to its own Rust file

### Solution:





### Test data:

N/A

### Sample output:

Hello World! X4

### Reflection:

Functions need to be public to be used by a file outside of the original file

### Metadata:

N/A

### Further information:

N/A

## Q2. Ownership

### Question:

**Part 1 –** Try to print p1 twice

**Part 2 –** Alter the code so that the print\_person returns the Person object back to the main

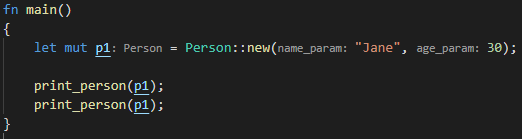
**Part 3 –** Modify print\_person to use a reference

**Part 4 –** Add the function increment\_age which takes a mutable reference as a parameter

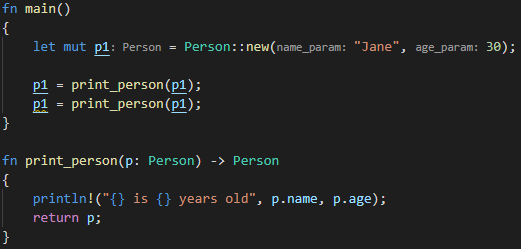
**Part 5 –** Cause borrowing to fail

### Solution:

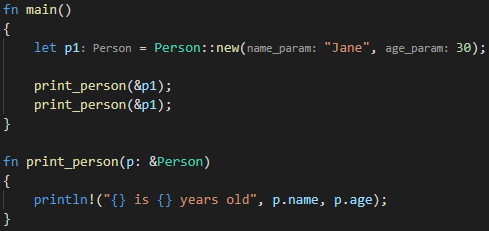
**Part 1 –**



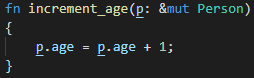
**Part 2 –**

****

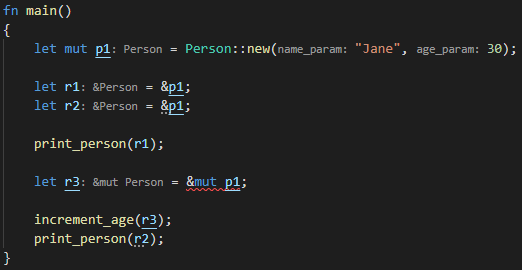
**Part 3 –**

****

**Part 4 –**

****

**Part 5 –**

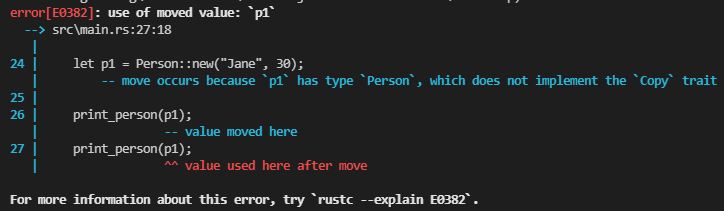
****

### Test data:

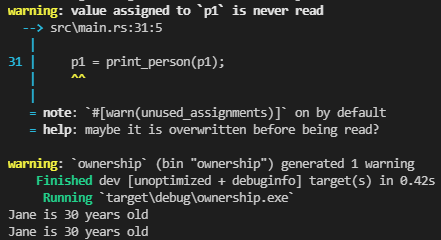
**All parts are N/A**

### Sample output:

**Part 1 –**

****

**Part 2 –**

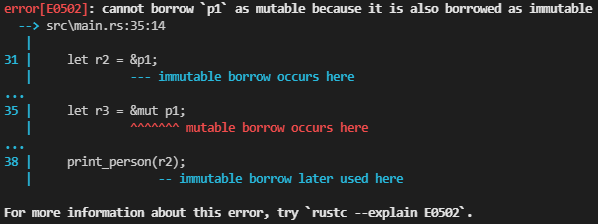


**Part 3 –**



**Part 4 –** N/A

**Part 5 –**

****

### Reflection:

Not sure if you would ever use explicit mutable and immutable references

### Metadata:

Ownership

### Further information:

Rust does not allow overriding of function names

<https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html>

## Q3. Classes

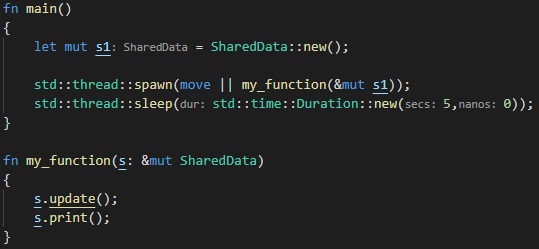
### Question:

**Part 1 –** Create a new thread function which takes SharedData as a parameter and then calls the update and print functions

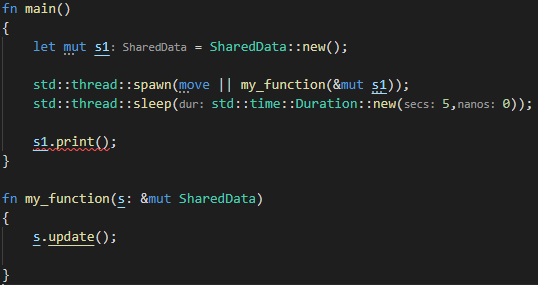
**Part 2 –** move the print function from your thread function to the main program

### Solution:

**Part 1 –**

****

**Part 2 –**

****

### Test data:

**Part 1 –** N/A

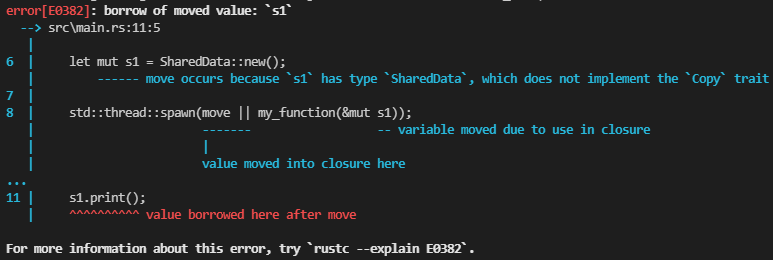
**Part 2 –** N/A

### Sample output:

**Part 1 –**



**Part 2 –**

****

### Reflection:

Couldn’t get part 1 to work with the template I used from the joining threads lab

### Metadata:

Classes

### Further information:

The fix to the problem addressed in part 2 will be covered later

# Week 4 – Lab D

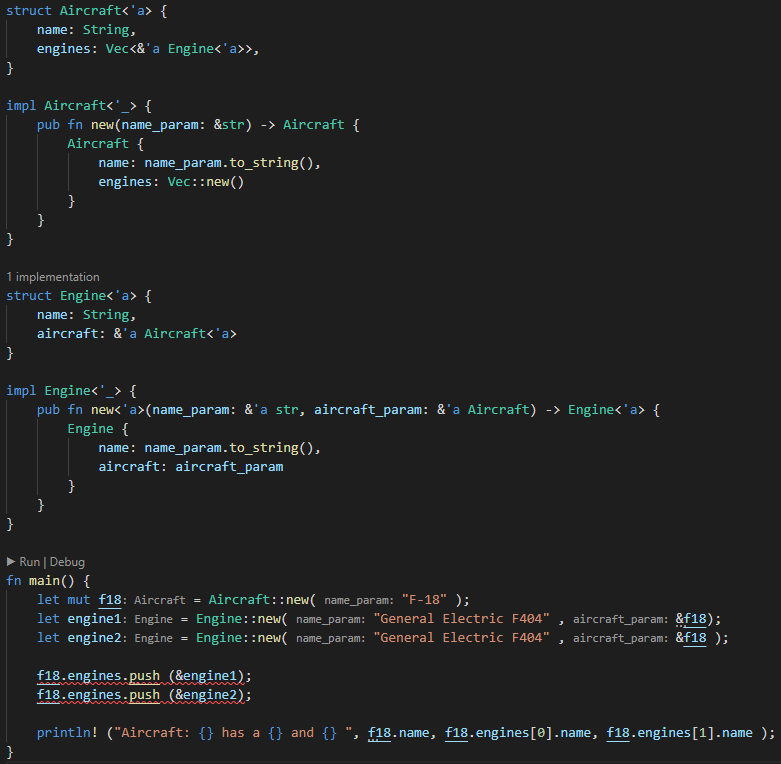
Date: 23/02/2022

## Q1. Ownership Limitations

### Question:

Read and understand the syntax of the provided code. Try to expand the code to include a data member in Engine that links to the Aircraft

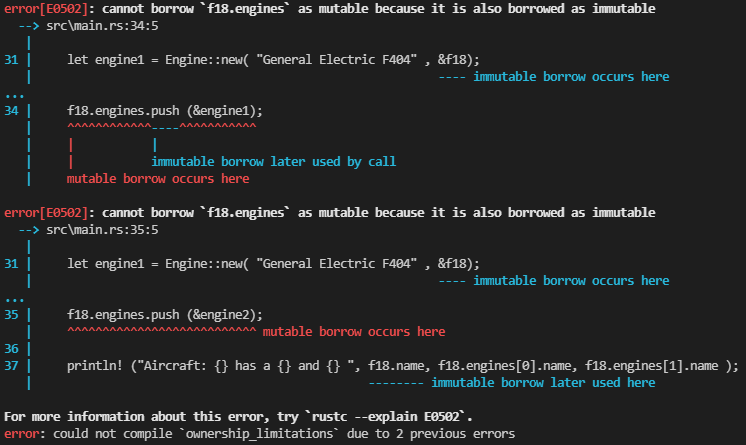
### Solution:



### Test data:

N/A

### Sample output:



### Reflection:

The limitation with the current code is that due to ownership restrictions it is not possible to link the Aircraft to an Engine

### Metadata:

Limitations

### Further information:

'a notation attached to the reference is called a lifetime parameter. It allows the compiler to determine whether all references are going to stay "alive" at least as long as the "parent"

## Q2. Reference Counters

### Question:

**Part 1 –** Examine the provided code. Explain what is happening with the reference counters and why we do not need to pass them as references

**Part 2 –** Remove the clone() method. Explain why this program now fails to build

**Part 3 –** Add a new boolean data member requires\_service to Engine. Add a new method service(&mut self) to Engine. This method will just set the requires\_service data member to false. Try to call the service method on engine1. Explain why the error is occurring

### Solution:

**Part 1 –**

****

A vector of reference counters for Engine objects is created



For each engine a new reference counter is made for a new engine object



Each engine is cloned creating another pointer to the same allocation associated with reference counter. I think we don’t need to pass as a reference here because a new reference is being created by clone and the reference counter for that object then increases the count

**Part 2 –** The program does not build now because the value for engine1 gets moved when it gets put into the list of engines for the aircraft. So, when the program tries to print that engines name later it can’t because the engine object has been borrowed and not returned

**Part 3 –** I think the error is occurring because the references created by the reference counter are immutable

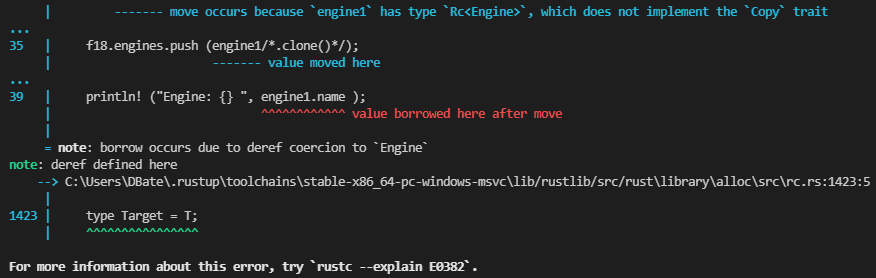
### Test data:

**Part 1 –** N/A

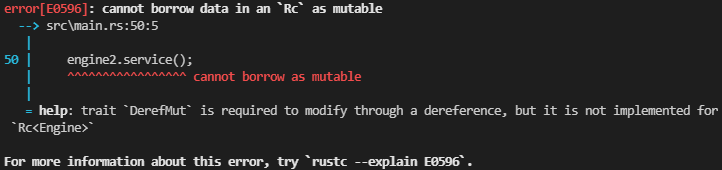
### Sample output:

**Part 1 –** N/A

**Part 2 –**



**Part 3 –**



### Reflection:

**Part 1 –** I think I understand how RC works but not well enough to properly explain it

**Part 2 –** This part was just the same as the ownership issues previously covered

**Part 3 –** The solution to this will be covered in a later lab

### Metadata:

Understanding RC a bit more

### Further information:

Ask about my library enallage to see if I’m understanding it correctly

# Week 5 – Lab E

Date: 02/02/2022

## Q1. Thread safe printing

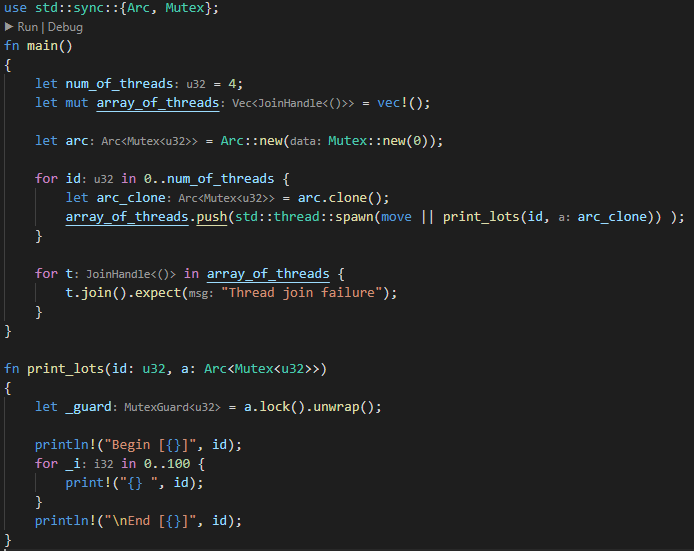
### Question:

**Part 1 –** Implement the thread safe printing

**Part 2 – Q1.**What happens to your code if you fail to release the mutex? **Q2.**Are you able to verify this in your code? **Q3.**What happens if you raise an exception within the critical section? **Q4.**Extend your code to verify your answer

### Solution:

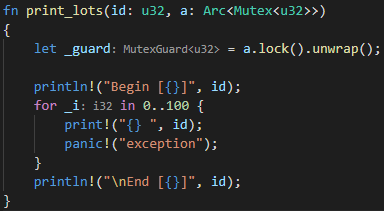
**Part 1 –**



**Part 4 –**

**Q2.**

**Q4.**

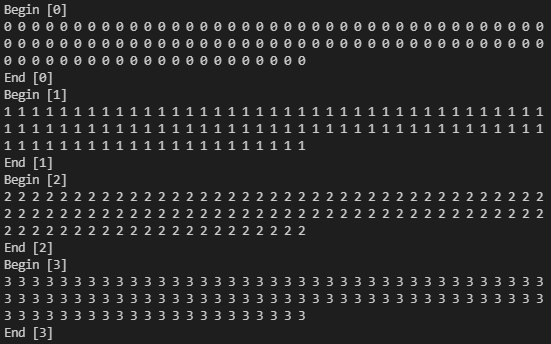
****

### Test data:

**All parts –** N/A

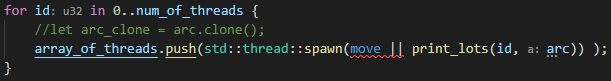
### Sample output:

**Part 1 –**

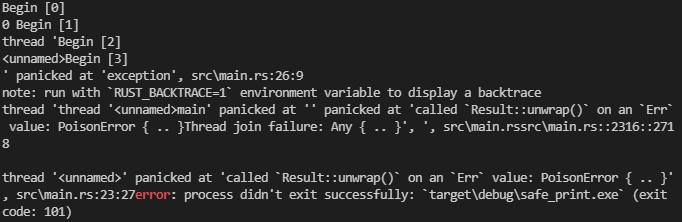


**Part 2 –**

**Q2.**

****

**Q4.**



### Reflection:

**Part 1 –** I’m not really sure what an ARC does differently to an RC

**Part 2 –**

I wasn’t 100% sure what question 1 meant but I’ve answer what I think it means

**Q1.** If the mutex isn’t released then in the first iteration of the thread loop the mutex is moved so in all iterations after the mutex and be used. In short it creates an ownership error.

After Tuesday live lecture I understand what is meant. The program reaches a deadlock

**Q3.** When I raised an exception in the critical section I get a “PoisonError”

### Metadata:

Errors

### Further information:

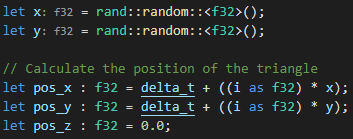
N/A

## Q2. Triangles and OpenGL

### Question:

Update the code to make the triangles move more chaotically

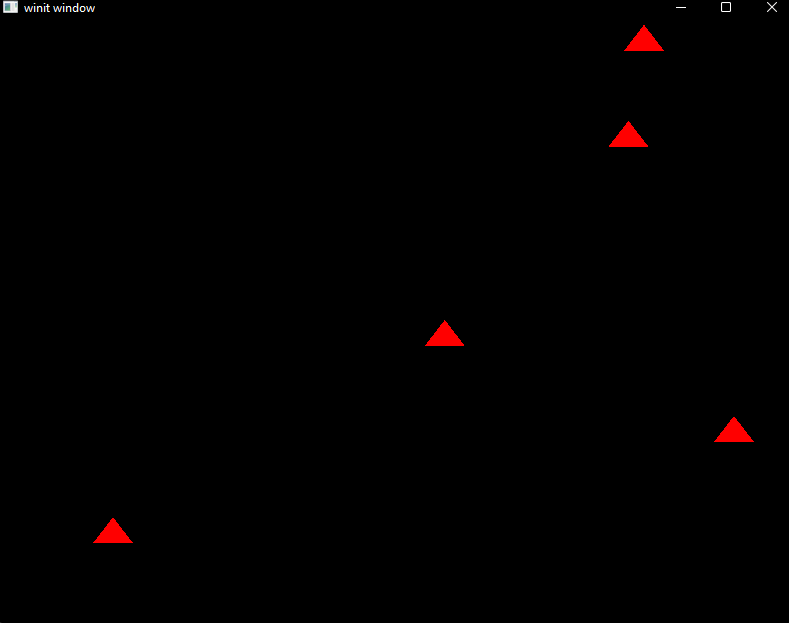
### Solution:



### Test data:

N/A

### Sample output:



### Reflection:

Rnad was acting weird to import the crate

### Metadata:

Chaos

### Further information:

N/A

# Week 6 – Lab F

Date: 09/03/2022

## Q1. Particles

### Question:

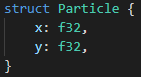
**Part 1 –** Implement structs described in the lab sheet

**Part 2 –** Implement functions for these structs as described in the lab sheet

**Part 3 –** Test release mode and add the macro described in the lab sheet

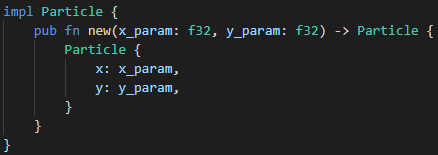
### Solution:

**Part 1 –**





**Part 2 –**

**Text

Description automatically generated**

**Part 3 –**

****

****

****

### Test data:

**Part 1 –** N/A

**Part 2 –**

**Text

Description automatically generated**

**Part 3 –** N/A

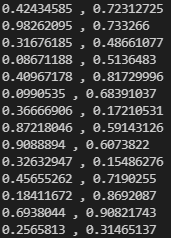
### Sample output:

**Part 1 –** N/A

**Part 2 –**

**Text

Description automatically generatedETC.**

**ETC.**

**Part 3 –** N/A

### Reflection:

The timer took me a bit to figure out. I was using a timer crate at first, but I think it only works with a threaded solution, so I just went with using delta time. Not sure what #[derive(Debug, Copy, Clone)] does

### Metadata:

Particles

### Further information:

N/A

## Q2. Threaded Particles

### Question:

Implement a threaded version of the pervious question using chunks and scoped thread pools

### Solution:

Text

Description automatically generated

Text

Description automatically generated

### Test data:

N/A

### Sample output:

Text

Description automatically generatedA picture containing text

Description automatically generatedText

Description automatically generated with medium confidenceA picture containing text, keyboard

Description automatically generated

### Reflection:

This was a bit confusing at first although I had similar code the first time I did it. I redid the code after I understood it more thanks to the Tuesday lecture

### Metadata:

Chunks

### Further information:

I changed the randomness of the movement for more predictable outcomes for testing

# Week 7 – Lab G

Date: 16/03/2022

## Q1. Colliding particles

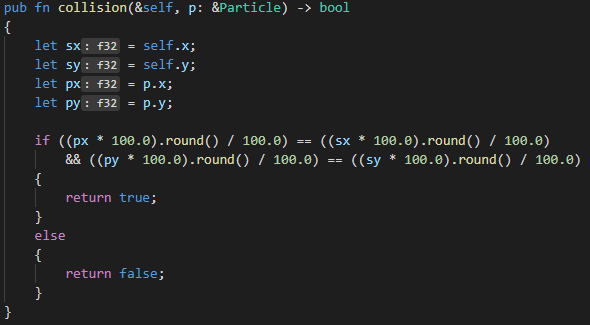
### Question:

**Part 1 –** Create a new function that checks if a particle collides with or is very close to (within 2 d.p.) another particle

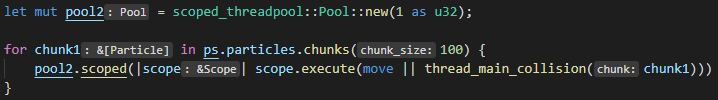
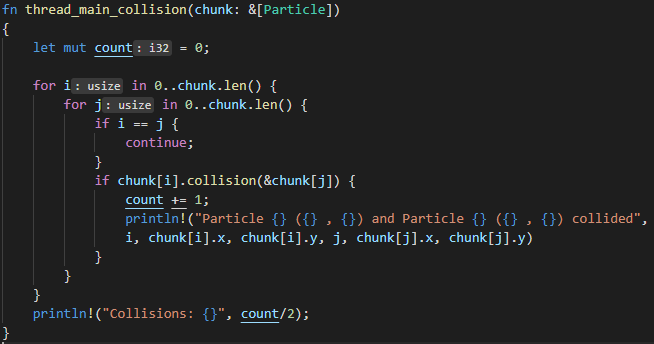
**Part 2 –** Add a counter to count the number of collisions

### Solution:

**Part 1 –**



**Part 2 –**

****

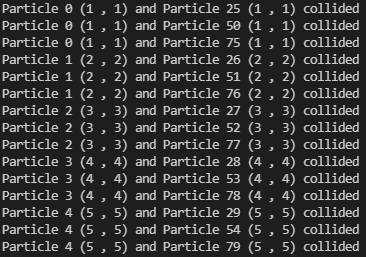
### Test data:

**All Parts –** The same tester code was used as last lab

### Sample output:

**Part 1 –** N/A

**Part 2 –**

**ETC.**



### Reflection:

**Q1.** Is locking required in your solution to prevent race conditions?

**A1.** I didn’t use any locking in my solution

**Q2.** Are there any other race conditions that can occur in your code?

**A2.** I don’t think there are because I used 2 different thread pools. If I had just used 1 I think I would have and then I would also need to use locking

**Q3.** Are there any optimisations you can make to your code?

**A3.** I could have the collisions stored in a list so the program is counting the same collision twice

### Metadata:

Collisions

### Further information:

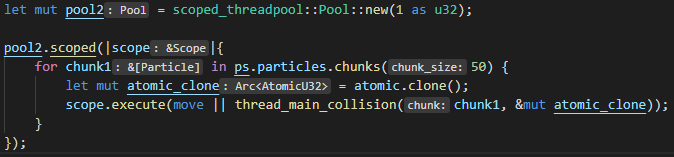
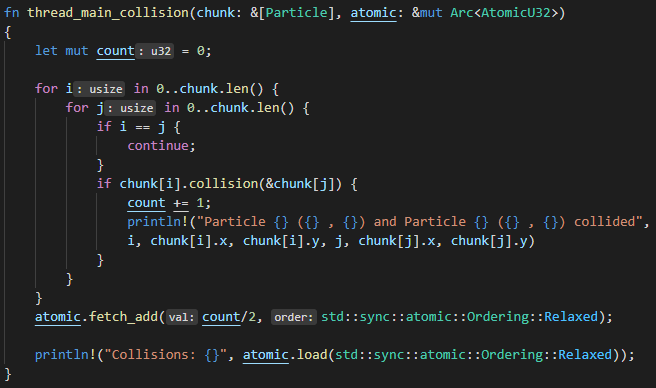
I’m not sure how using chunks for the collisions would work. How would you be able to check for collisions between chunks

## Q2. Recording collisions using an Atomic

### Question:

Replace the local counter with an atomic counter to measure the number of collisions across all threads

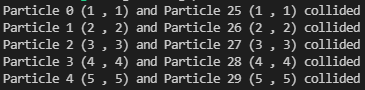
### Solution:



### Test data:

N/A

### Sample output:

**ETC.**



### Reflection:

I don’t understand the Ordering variable for atomic functions. And for some reason even though for collisions I only gave it 1 thread it executes twice and does the same thing again.

### Metadata:

Atomics

### Further information:

N/A

# Week 8 – Lab H

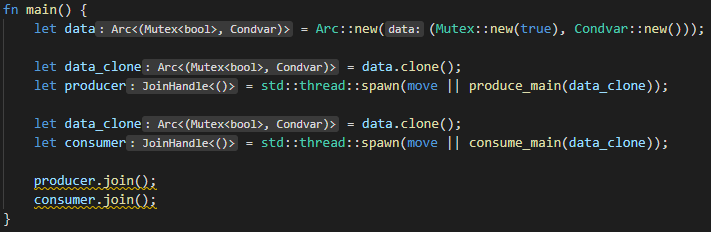
Date: 23/03/2022

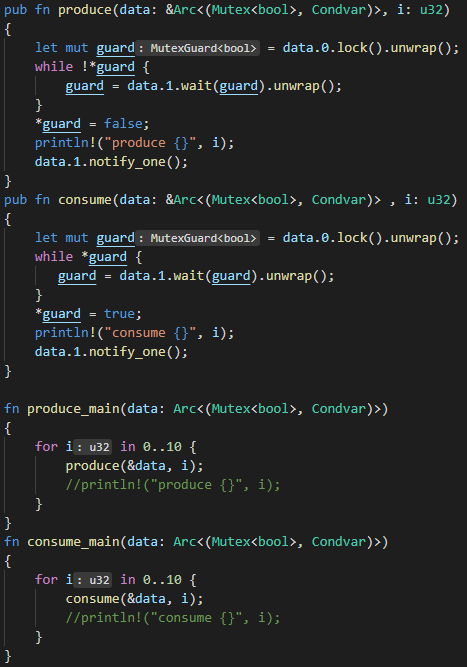
## Q1. Condition variables

### Question:

Create a producer / consumer model using condition variables

### Solution:

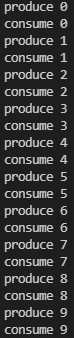




### Test data:

N/A

### Sample output:



### Reflection:

I got myself very confused on this, with a lot of help from Warren I got the task done. But I think I defiantly need to practice this more to better understand it in practice

### Metadata:

Condition Variables

### Further information:

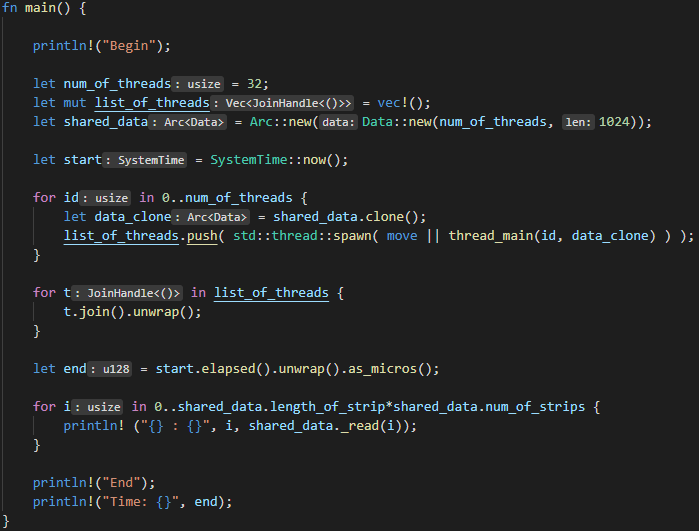
N/A

## Q2. Striped arrays, sequential access

### Question:

Add timing code to the example to measure the duration of array access. Alter the program to use 2, 4, 8, 16, 32 threads

### Solution:



### Test data:

2, 4, 8, 16, 32 threads

### Sample output:

|  |  |  |
| --- | --- | --- |
| **Threads** | **Strips** | **Time** |
| 2 | 16384 |  |
| 4 | 8192 |  |
| 8 | 4096 |  |
| 16 | 2048 |  |
| 32 | 1024 |  |

### Reflection:

The timing seems to double each time

### Metadata:

Timings

### Further information:

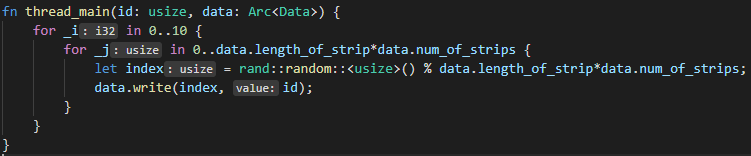
N/A

## Q3. Striped arrays, random access

### Question:

Modify the code to implement random access

### Solution:



### Test data:

2, 4, 8, 16, 32 threads

### Sample output:

|  |  |  |
| --- | --- | --- |
| **Threads** | **Strips** | **Time** |
| 2 | 16384 |  |
| 4 | 8192 |  |
| 8 | 4096 |  |
| 16 | 2048 |  |
| 32 | 1024 |  |

### Reflection:

Random access seems to be significantly worse for scaling

### Metadata:

Timings 2

### Further information:

N/A