Lab Book

Daniel Bates

Contents

Rust Labs 3

Week 1 – Lab A 3

Q1. Installs 3

Q2. Hello World 4

Week 2 – Lab B 5

Q1. First threads 5

Q2. Joining threads 6

Q3. Experimentation 7

Week 3 – Lab C........ 13

Q1. Multiple Rust Files 13

Q2. Ownership 14

Q3. Classes 18

Week 4 – Lab D 20

Q1. Ownership Limitations........ 20

Q2. Reference Counters 21

Week 5 – Lab E........ 24

Q1. Thread safe printing 24

Q2. Triangles and OpenGL 26

Week 6 – Lab F........ 27

Q1. Particles 27

Q2. Threaded Particles 31

Week 7 – Lab G 33

Q1. Colliding particles 33

Q2. Recording collisions using an Atomic........ 35

Week 8 – Lab H 36

Q1. Condition variables 36

Q2. Striped arrays, sequential access 38

Q3. Striped arrays, random access 39

CUDA Labs 40

Week 1 – Lab 1........ 40

Q1. Setting up CUDA 40

Q2. CUDA error checking 41

Q3. Check time range 42
Week 2 – Lab 2 43
Q1. Understand the block and thread indices 43
Q2. Find vector addition using multiple 1D thread blocks 43
Q3. Understand the thread indices for 2D blocks 44
Q4. Find vector addition using one 2D thread block 45
Q5. Find vector addition using multiple 2D thread blocks 45
Q6. Matrix addition 46
Week 3 – Lab 3 48
Q1. Vector dot-product 48
Q2. Vector dot-product using unified memory 49
Q3. Vector dot-product using shared Memory 51
Week 4 – Lab 4 53
Q1. Create an OpenGL-CUDA program based on a CUDA SDK sample 53
Q2. Understand pixel colour 54
Q3. Image transformation 55
Q4. Image smoothing 58
Week 5 – Lab 5 60
Q1. A simple matrix multiplication program in CUDA using one thread block 60
Q2. Compare the performance of the CUDA solution against the CPU solution 61
Week 6 – Lab 6 63
Q1. Set up a virtual canvas and draw on it an image in CUDA 63
Q2. Drawing a checkboard in CUDA 64
Q3. Drawing the Mandelbrot and Julia Sets 66
Week 7 – Lab 7 68
Q1. Drawing based on a canvas of size [-1, 1]x[-1, 1] 68
Q2. Write a simple ray caster 69
Week 8 – Lab 8 70
Q1. Draw a box without front wall 70
Q2. Free motion animation 70
Q3. Ball-box walls collision animation 72
Final Lab 73
GPU Design 73
CPU Design 73
Performance comparison 73
Reflection 73

Rust Labs

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:

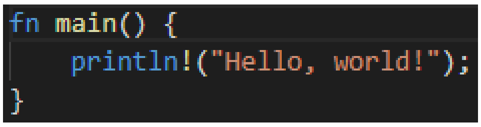
N/A

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:

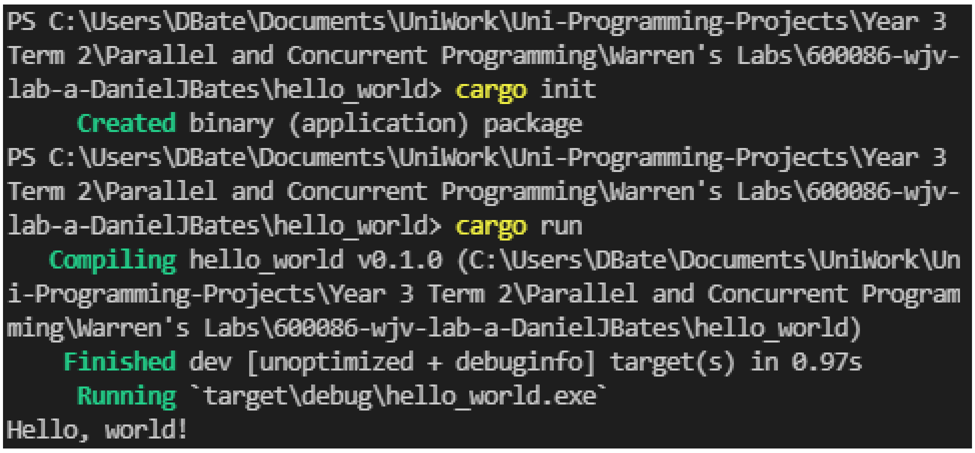


```
fn main() {
    println!("Hello, world!");
}
```

Test Data:

N/A

Sample Output:



```
PS C:\Users\DBate\Documents\UniWork\Uni-Programming-Projects\Year 3
Term 2\Parallel and Concurrent Programming\Warren's Labs\600086-wjv-
lab-a-DanielJBates\hello_world> cargo init
     Created binary (application) package
PS C:\Users\DBate\Documents\UniWork\Uni-Programming-Projects\Year 3
Term 2\Parallel and Concurrent Programming\Warren's Labs\600086-wjv-
lab-a-DanielJBates\hello_world> cargo run
   Compiling hello_world v0.1.0 (C:\Users\DBate\Documents\UniWork\Un
i-Programming-Projects\Year 3 Term 2\Parallel and Concurrent Program
ming\Warren's Labs\600086-wjv-lab-a-DanielJBates\hello_world)
    Finished dev [unoptimized + debuginfo] target(s) in 0.97s
     Running `target\debug\hello_world.exe`
Hello, world!
```

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

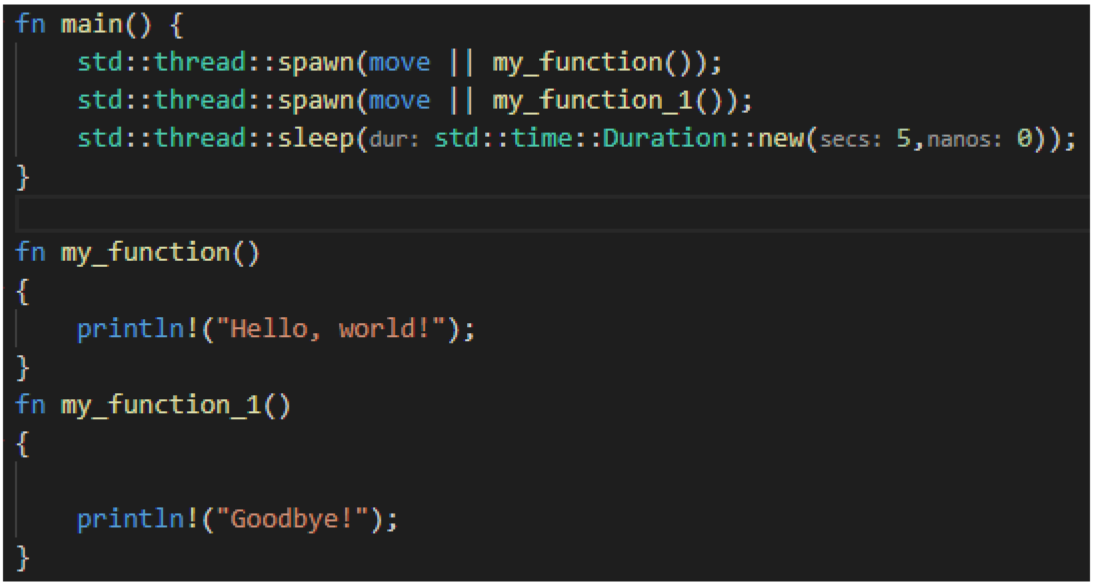
Date: 9th Feb 2022

Q1. First threads

Question:

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

Solution:

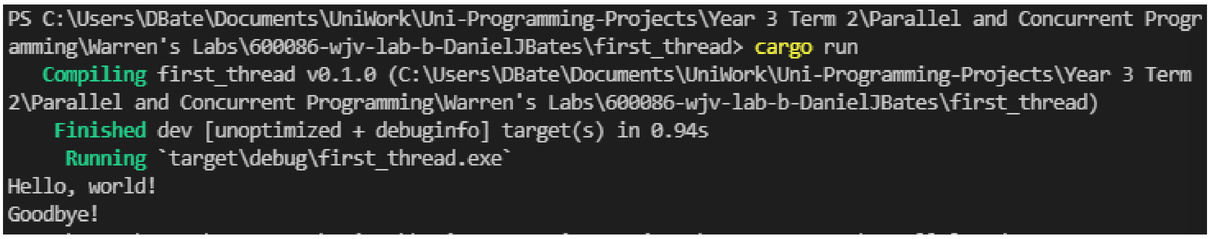


```
fn main() {
    std::thread::spawn(move || my_function());
    std::thread::spawn(move || my_function_1());
    std::thread::sleep(dur: std::time::Duration::new(secs: 5,nanos: 0));
}

fn my_function()
{
    println!("Hello, world!");
}
fn my_function_1()
{

    println!("Goodbye!");
}
```

Test data:

n/a

Sample output:



```
PS C:\Users\DBate\Documents\UniWork\Uni-Programming-Projects\Year 3 Term 2\Parallel and Concurrent Progr
amming\Warren's Labs\600086-wjv-lab-b-DanielJBates\first_thread> cargo run
   Compiling first_thread v0.1.0 (C:\Users\DBate\Documents\UniWork\Uni-Programming-Projects\Year 3 Term
2\Parallel and Concurrent Programming\Warren's Labs\600086-wjv-lab-b-DanielJBates\first_thread)
    Finished dev [unoptimized + debuginfo] target(s) in 0.94s
     Running `target\debug\first_thread.exe`
Hello, world!
Goodbye!
```

Reflection:

This is threading 101

Metadata:

Threads

Further information:

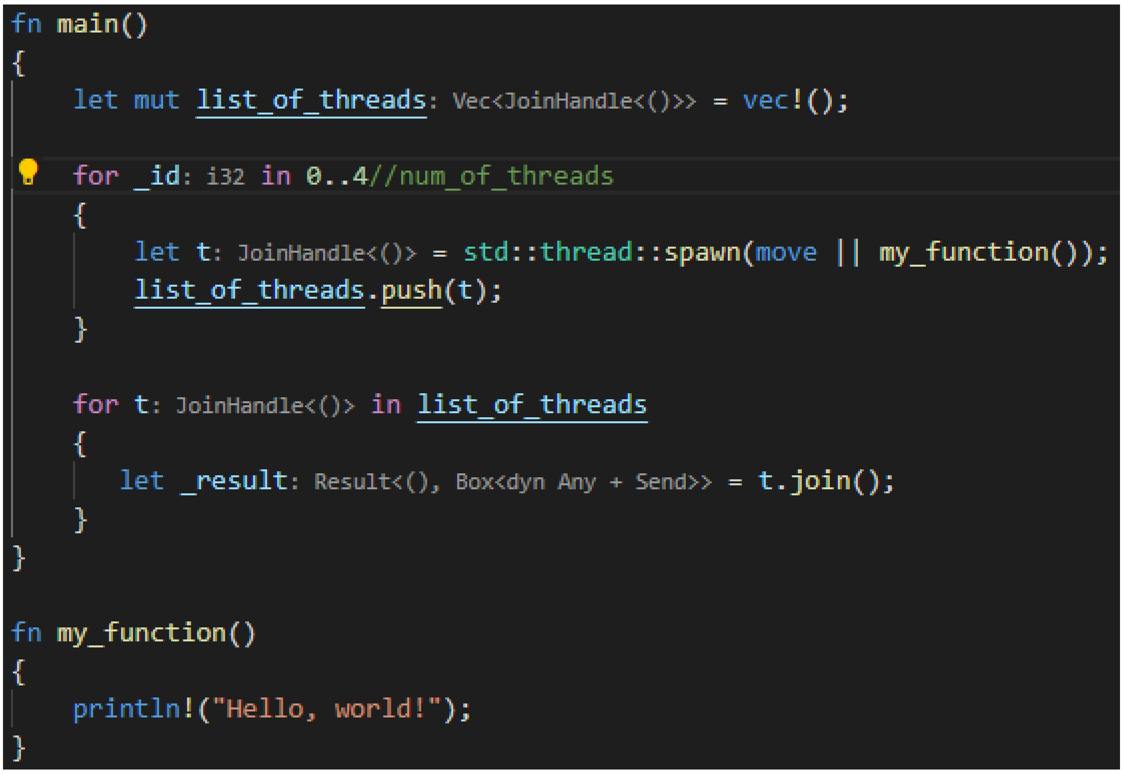
N/A

Q2. Joining threads

Question:

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

Solution:

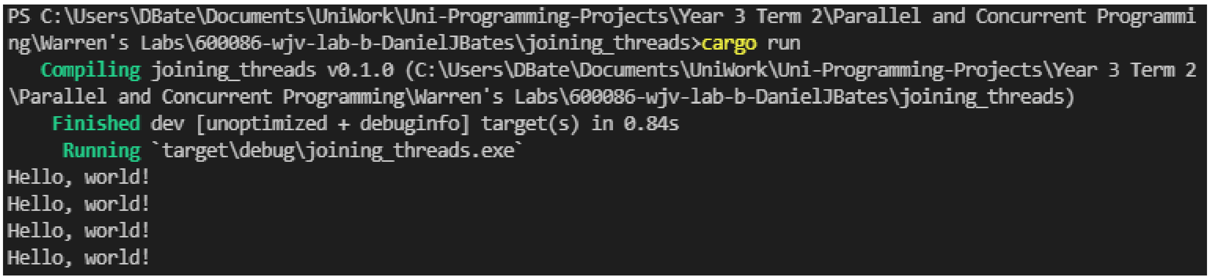


```rust
fn main()
{
    let mut list_of_threads: Vec<JoinHandle<()>> = vec!();

    for _id: i32 in 0..4//num_of_threads
    {
        let t: JoinHandle<()> = std::thread::spawn(move || my_function());
        list_of_threads.push(t);
    }

    for t: JoinHandle<()> in list_of_threads
    {
       let _result: Result<(), Box<dyn Any + Send>> = t.join();
    }
}

fn my_function()
{
    println!("Hello, world!");
}
```

Test Data:

N/A

Sample Output:



```
PS C:\Users\DBate\Documents\UniWork\Uni-Programming-Projects\Year 3 Term 2\Parallel and Concurrent Programmi
ng\Warren's Labs\600086-wjv-lab-b-DanielJBates\joining_threads>cargo run
   Compiling joining_threads v0.1.0 (C:\Users\DBate\Documents\UniWork\Uni-Programming-Projects\Year 3 Term 2
\Parallel and Concurrent Programming\Warren's Labs\600086-wjv-lab-b-DanielJBates\joining_threads)
    Finished dev [unoptimized + debuginfo] target(s) in 0.84s
     Running `target\debug\joining_threads.exe`
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

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:

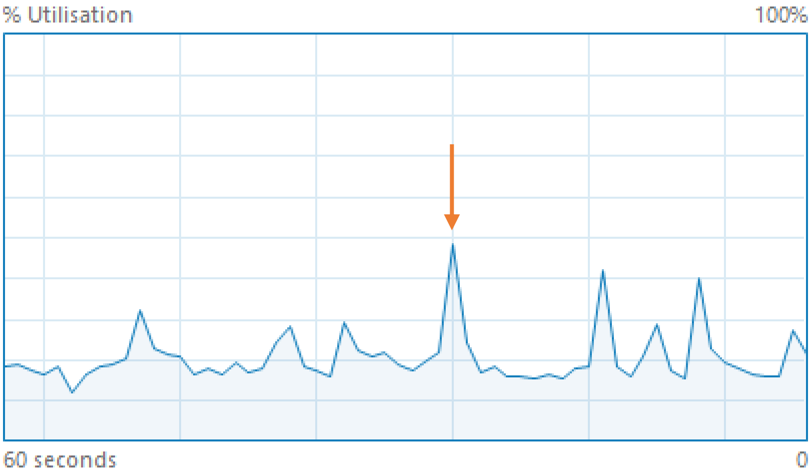
N/A

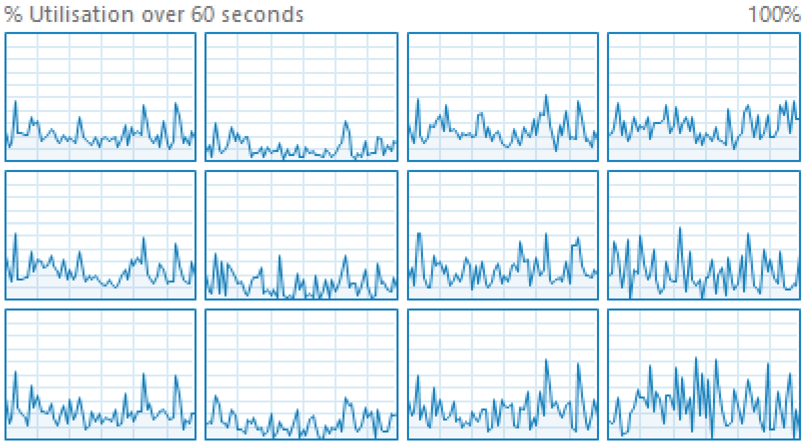
Test data:

N/A

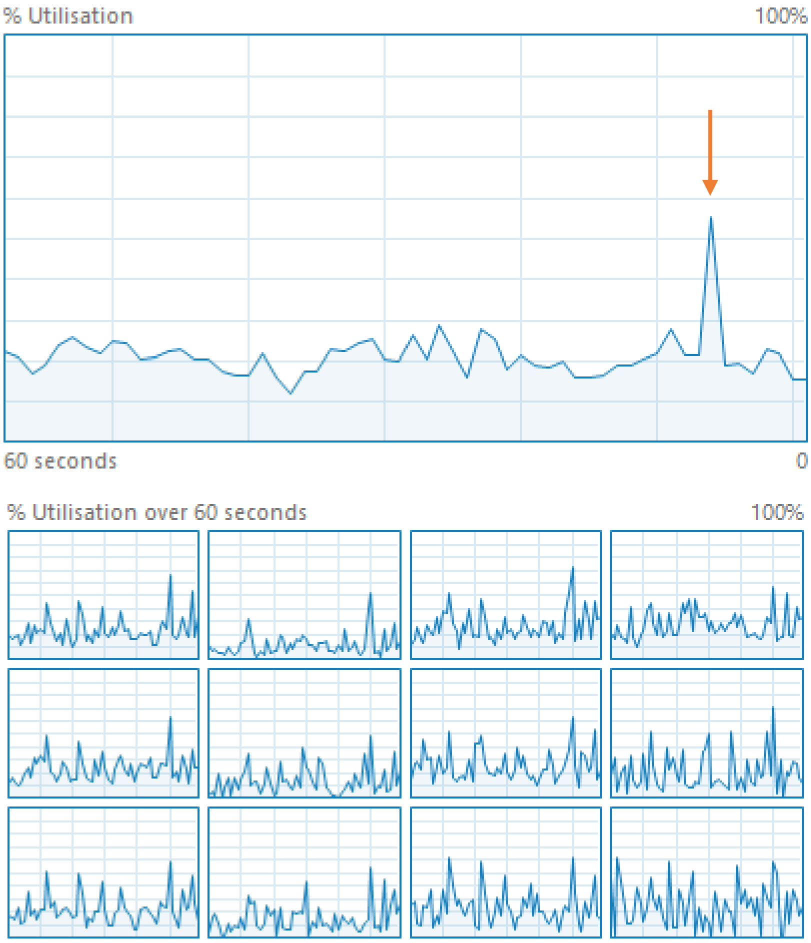
Sample output:

512 threads (Hello World) –

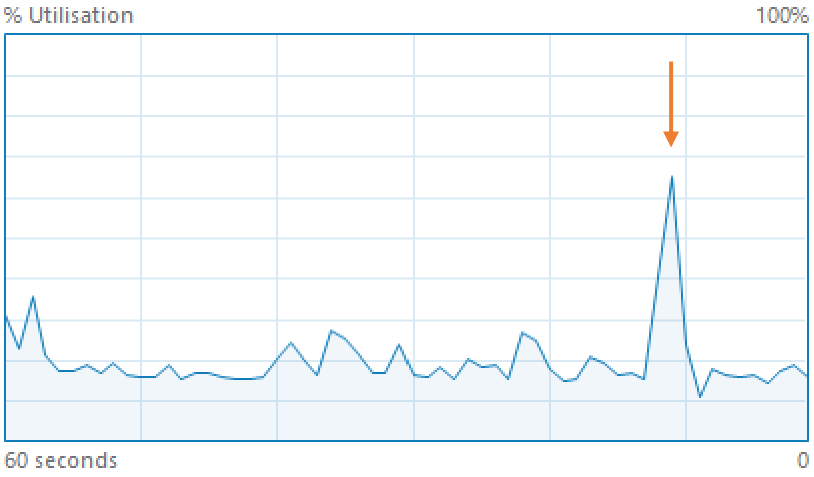


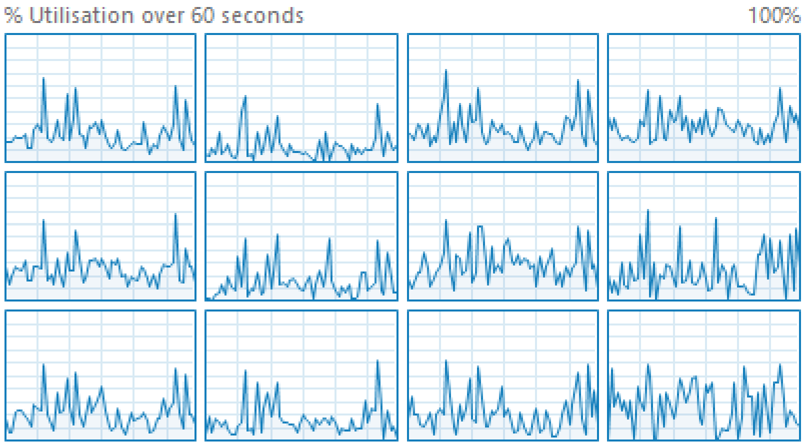


1024 threads (Hello World) –

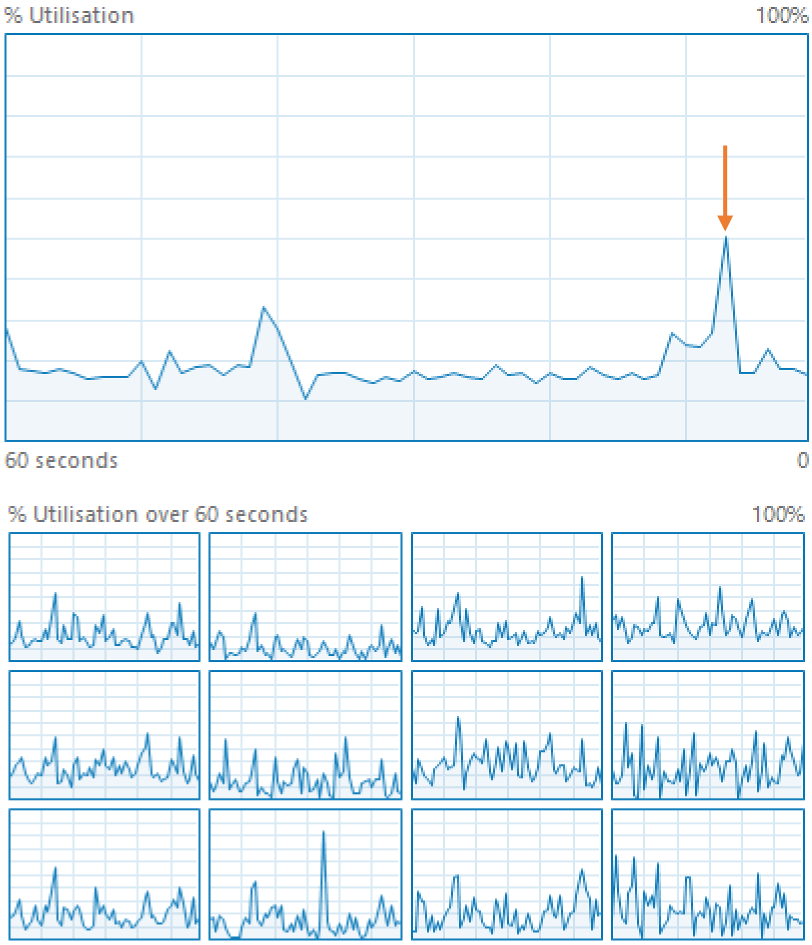


2048 threads (Hello World) –

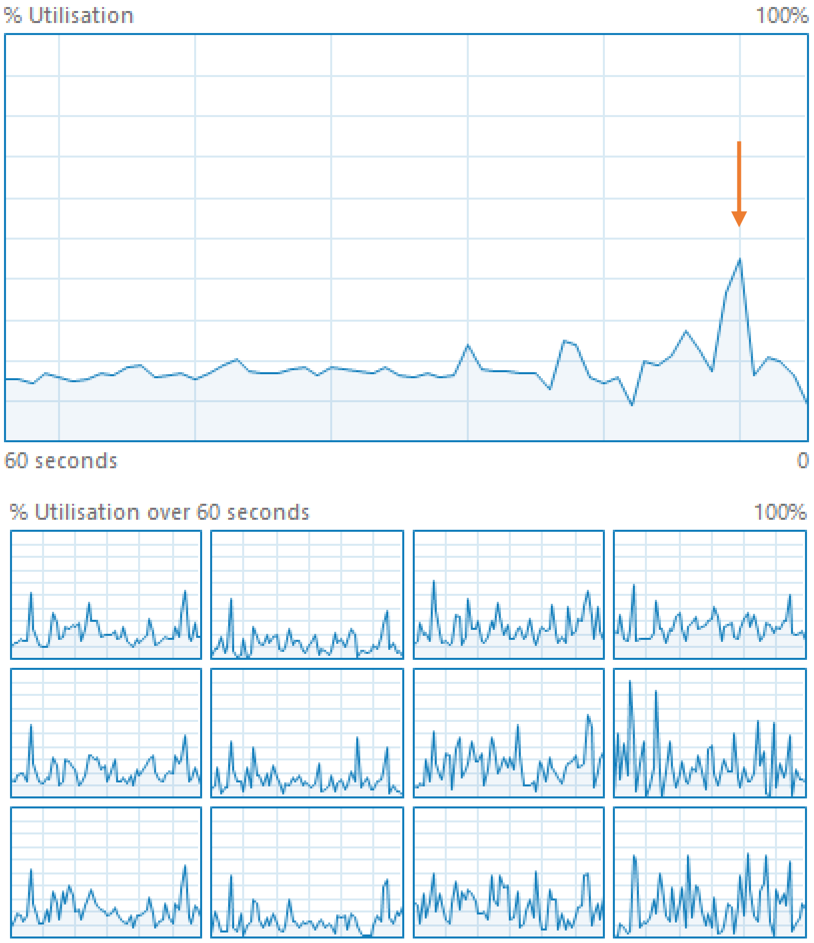




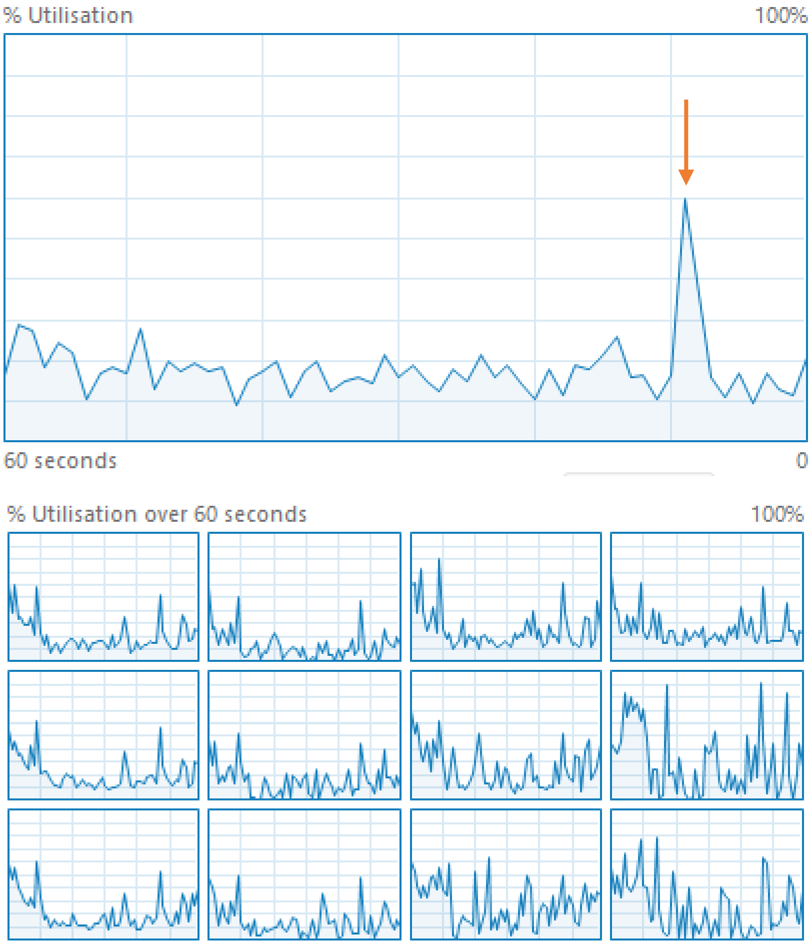
512 threads (1 + 2 + 5 * 25) –



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

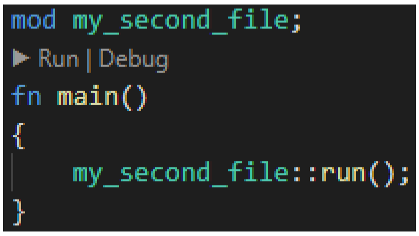
Date: 16/02/2022

Q1. Multiple Rust Files

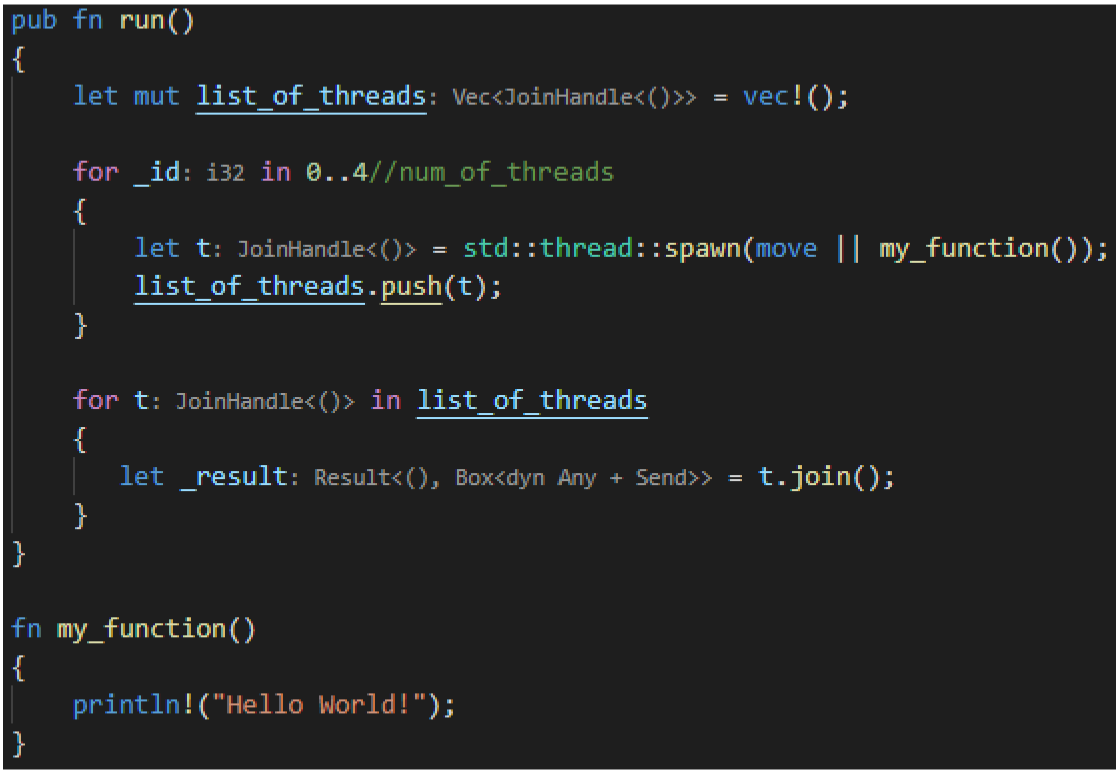
Question:

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

Solution:



```
mod my_second_file;
▶ Run | Debug
fn main()
{
    my_second_file::run();
}
```



```
pub fn run()
{
    let mut list_of_threads: Vec<JoinHandle<()>> = vec!();

    for _id: i32 in 0..4//num_of_threads
    {
        let t: JoinHandle<()> = std::thread::spawn(move || my_function());
        list_of_threads.push(t);
    }

    for t: JoinHandle<()> in list_of_threads
    {
       let _result: Result<(), Box<dyn Any + Send>> = t.join();
    }
}

fn my_function()
{
    println!("Hello World!");
}
```

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

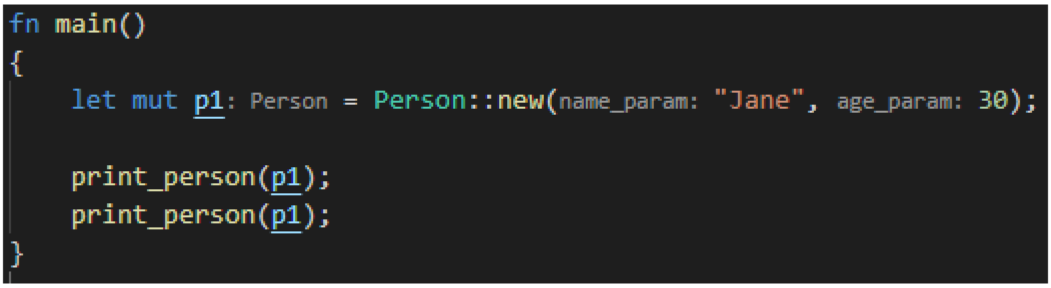
Part 3 – Modify print_person to use a reference

Part 4 – Add the function increment_age which takes a mutable reference as a parameter

Part 5 – Cause borrowing to fail

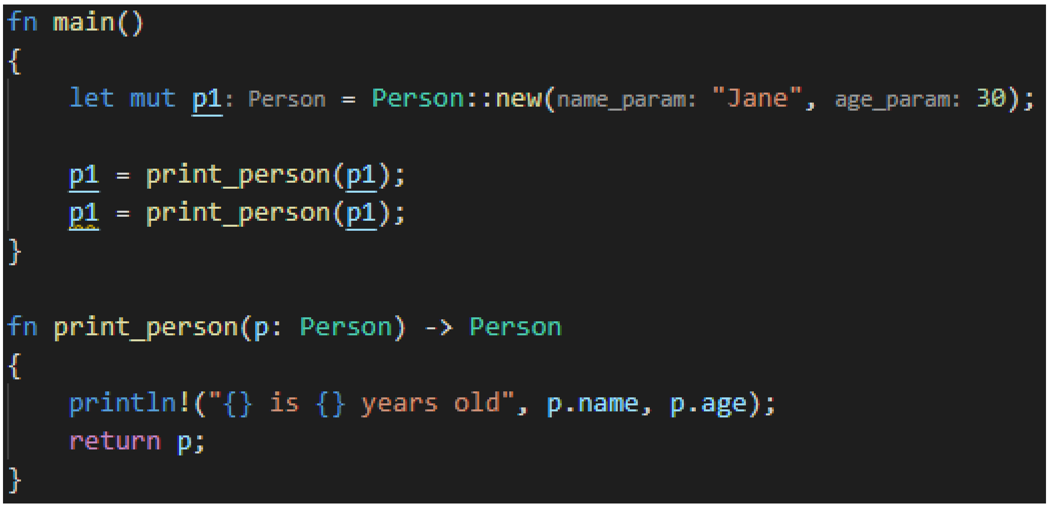
Solution:

Part 1 –



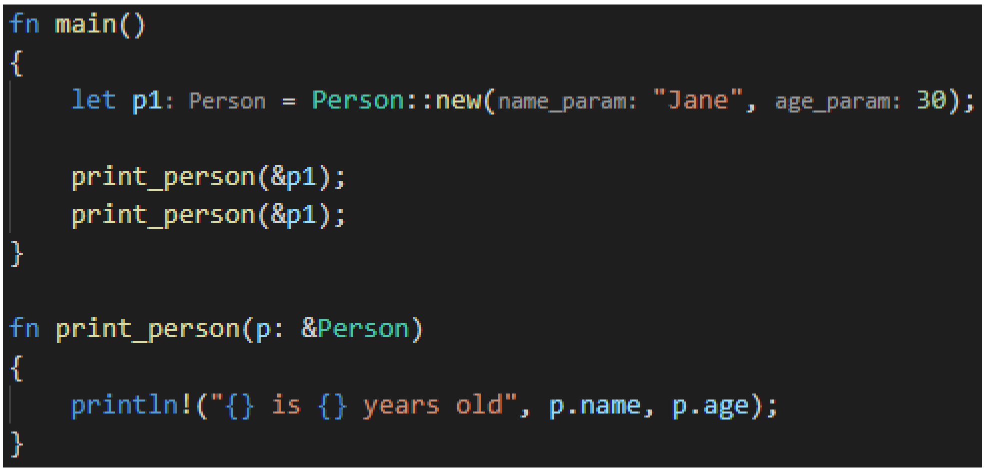
```
fn main()
{
    let mut p1: Person = Person::new(name_param: "Jane", age_param: 30);

    print_person(p1);
    print_person(p1);
}
```

Part 2 –



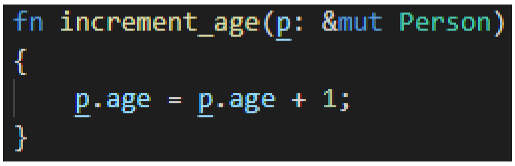
```
fn main()
{
    let mut p1: Person = Person::new(name_param: "Jane", age_param: 30);

    p1 = print_person(p1);
    p1 = print_person(p1);
}

fn print_person(p: Person) -> Person
{
    println!("{} is {} years old", p.name, p.age);
    return p;
}
```

Part 3 –



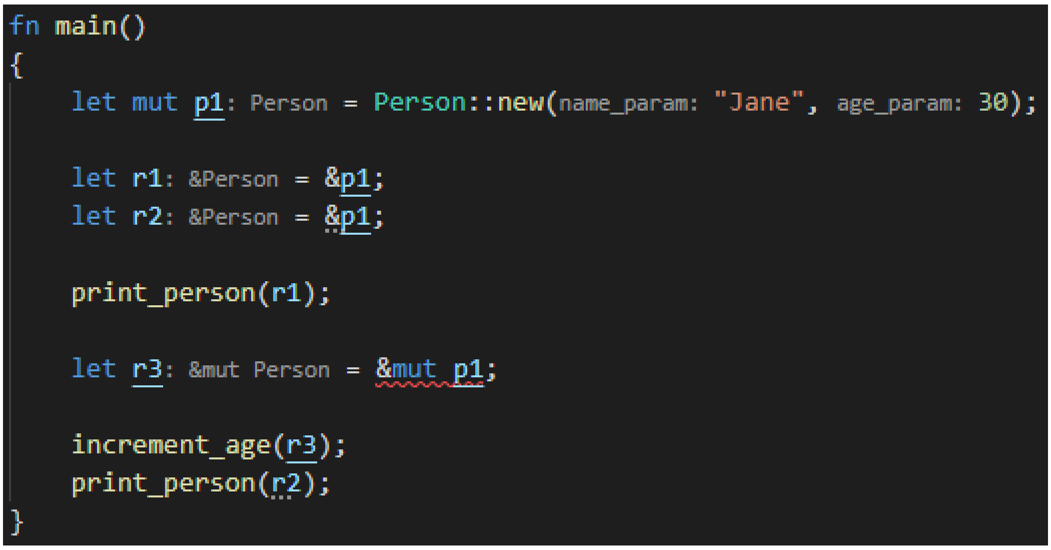
```
fn main()
{
    let p1: Person = Person::new(name_param: "Jane", age_param: 30);

    print_person(&p1);
    print_person(&p1);
}

fn print_person(p: &Person)
{
    println!("{} is {} years old", p.name, p.age);
}
```

Part 4 –



```
fn increment_age(p: &mut Person)
{
    p.age = p.age + 1;
}
```

Part 5 –



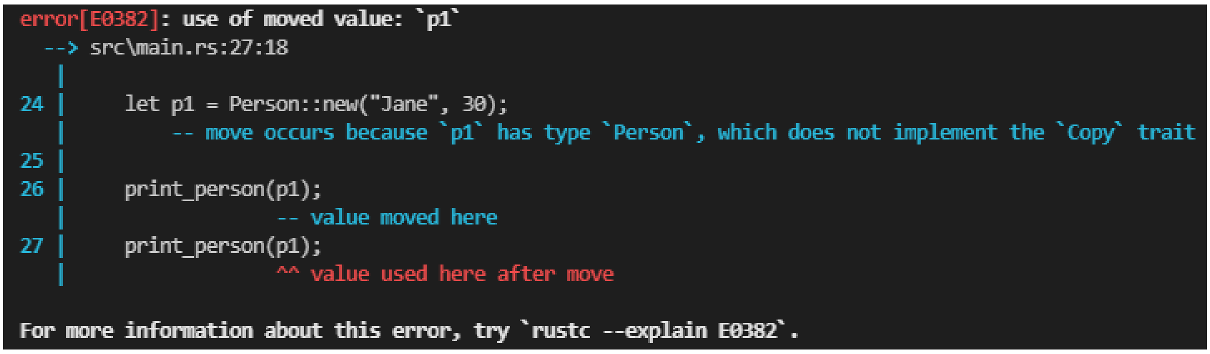
```
fn main()
{
    let mut p1: Person = Person::new(name_param: "Jane", age_param: 30);

    let r1: &Person = &p1;
    let r2: &Person = &p1;

    print_person(r1);

    let r3: &mut Person = &mut p1;

    increment_age(r3);
    print_person(r2);
}
```

Test data:

All parts are N/A

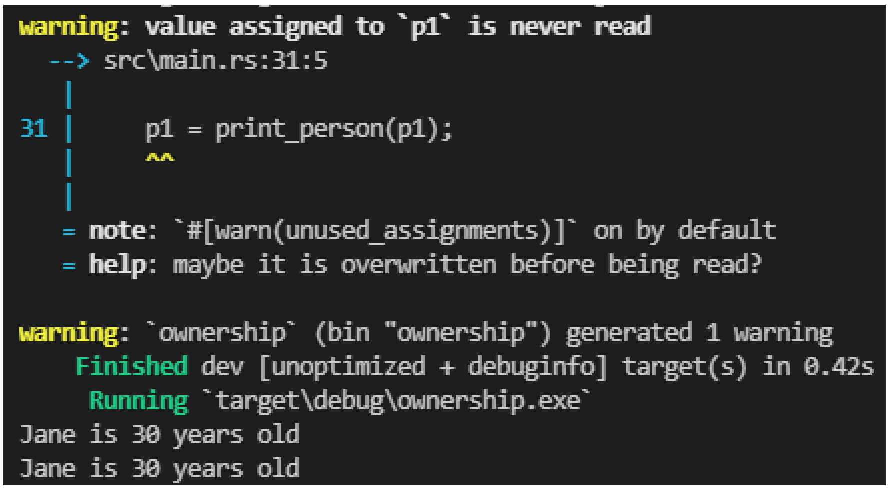
Sample output:

Part 1 –



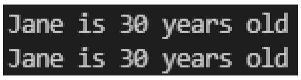
```
error[E0382]: use of moved value: `p1`
  --> src\main.rs:27:18
   |
24 |     let p1 = Person::new("Jane", 30);
   |         -- move occurs because `p1` has type `Person`, which does not implement the `Copy` trait
25 |
26 |     print_person(p1);
   |                  -- value moved here
27 |     print_person(p1);
   |                  ^^ value used here after move

For more information about this error, try `rustc --explain E0382`.
```

Part 2 –



```
warning: value assigned to `p1` is never read
  --> src\main.rs:31:5
   |
31 |     p1 = print_person(p1);
   |     ^^
   |
   = note: `#[warn(unused_assignments)]` on by default
   = help: maybe it is overwritten before being read?

warning: `ownership` (bin "ownership") generated 1 warning
    Finished dev [unoptimized + debuginfo] target(s) in 0.42s
     Running `target\debug\ownership.exe`
Jane is 30 years old
Jane is 30 years old
```

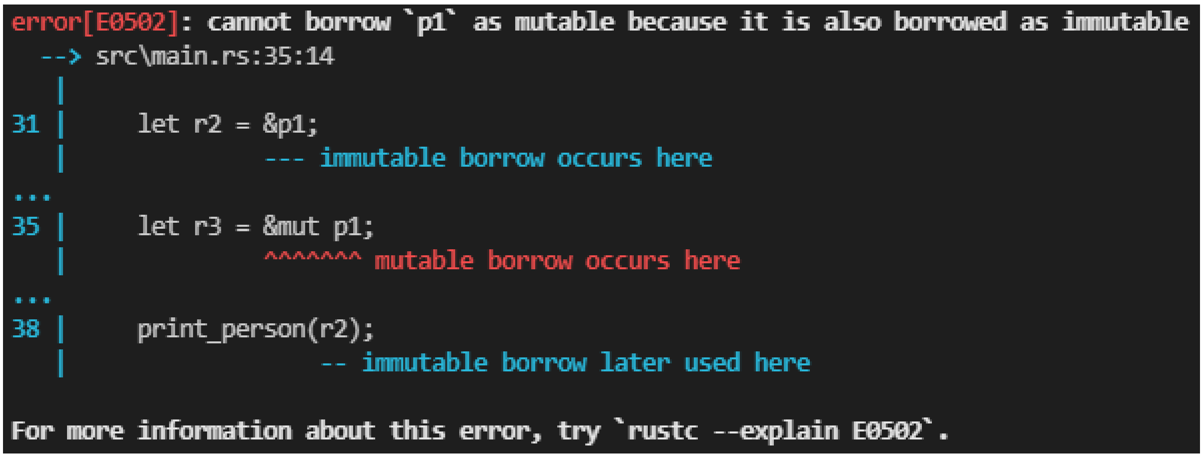
Part 3 –



```
Jane is 30 years old
Jane is 30 years old
```

Part 4 – N/A

Part 5 –



```
error[E0502]: cannot borrow `p1` as mutable because it is also borrowed as immutable
  --> src\main.rs:35:14
   |
31 |     let r2 = &p1;
   |              --- immutable borrow occurs here
...
35 |     let r3 = &mut p1;
   |              ^^^^^^^ mutable borrow occurs here
...
38 |     print_person(r2);
   |                  -- immutable borrow later used here

For more information about this error, try `rustc --explain E0502`.
```

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

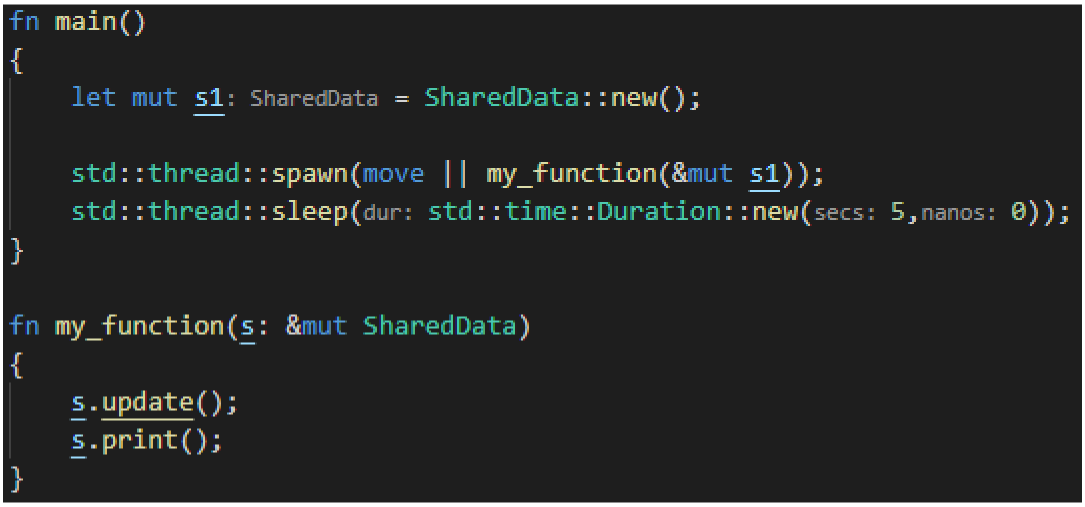
Question:

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

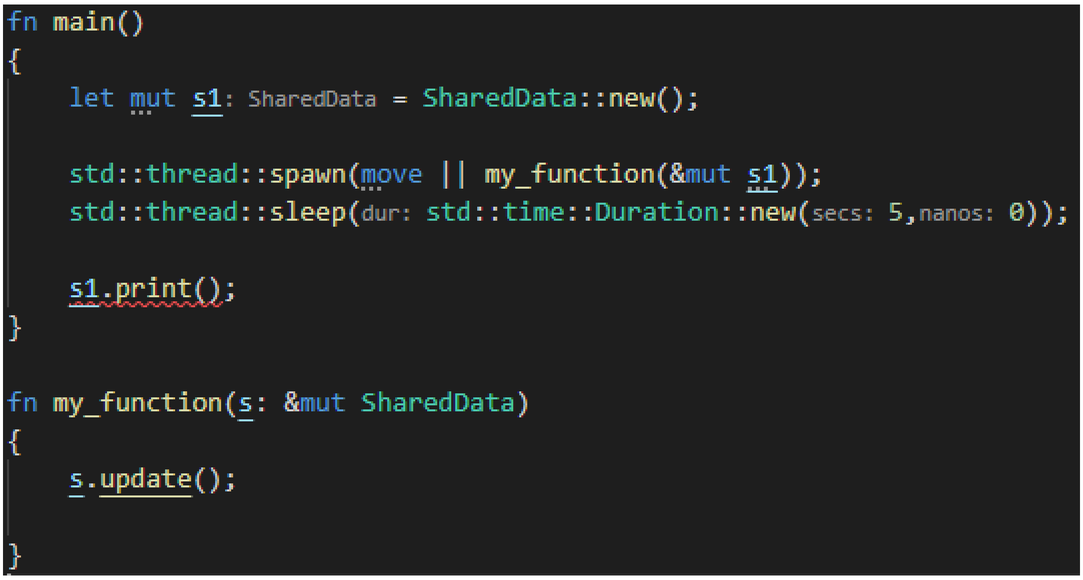
Solution:

Part 1 –



```
fn main()
{
    let mut s1: SharedData = SharedData::new();

    std::thread::spawn(move || my_function(&mut s1));
    std::thread::sleep(dur: std::time::Duration::new(secs: 5,nanos: 0));
}

fn my_function(s: &mut SharedData)
{
    s.update();
    s.print();
}
```

Part 2 –



```
fn main()
{
    let mut s1: SharedData = SharedData::new();

    std::thread::spawn(move || my_function(&mut s1));
    std::thread::sleep(dur: std::time::Duration::new(secs: 5,nanos: 0));

    s1.print();
}

fn my_function(s: &mut SharedData)
{
    s.update();

}
```

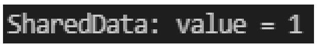
Test data:

Part 1 – N/A

Part 2 – N/A

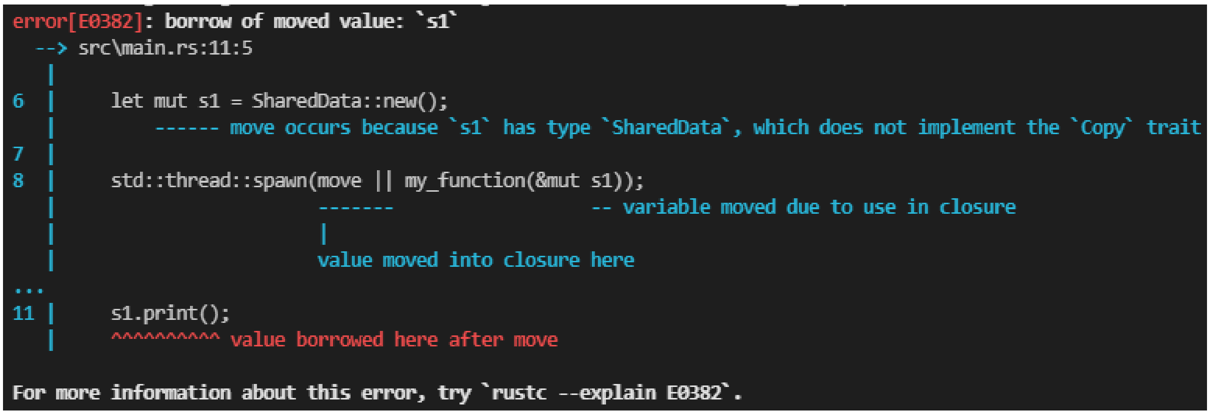
Sample output:

Part 1 –



```
SharedData: value = 1
```

Part 2 –



```
error[E0382]: borrow of moved value: `s1`
  --> src\main.rs:11:5
   |
6  |     let mut s1 = SharedData::new();
   |         ------ move occurs because `s1` has type `SharedData`, which does not implement the `Copy` trait
7  |
8  |     std::thread::spawn(move || my_function(&mut s1));
   |                        -------                  -- variable moved due to use in closure
   |                        |
   |                        value moved into closure here
...
11 |     s1.print();
   |     ^^^^^^^^^^ value borrowed here after move

For more information about this error, try `rustc --explain E0382`.
```

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

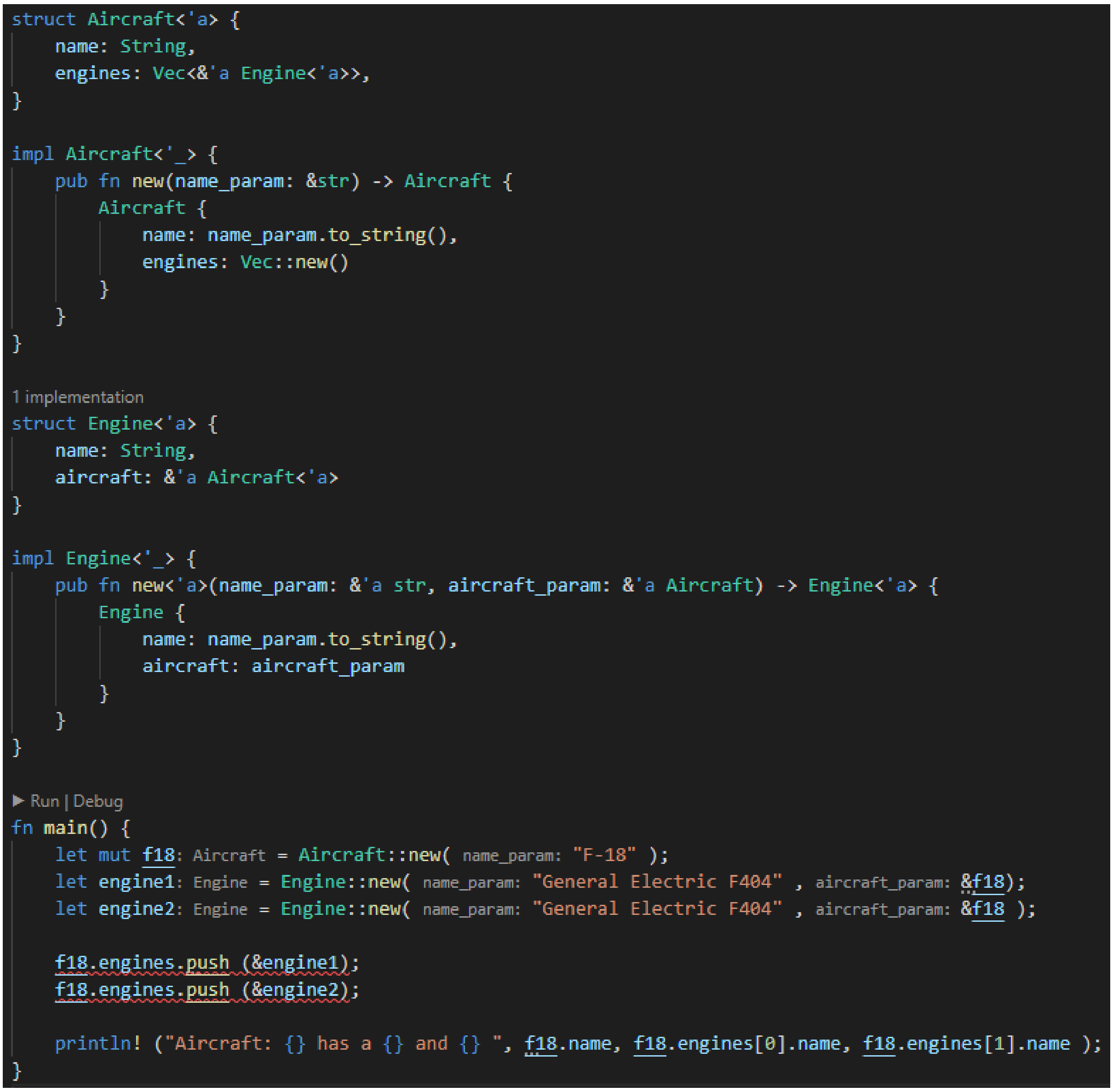
Date: 23/02/2022

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:

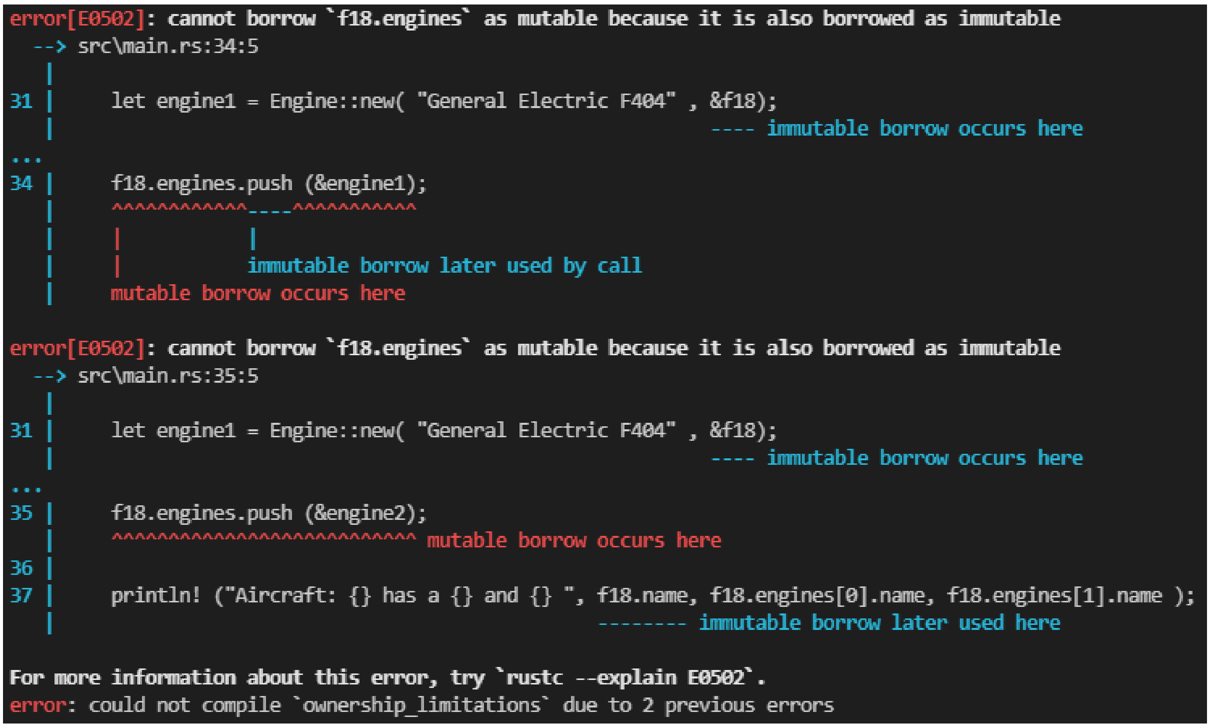


```
struct Aircraft<'a> {
    name: String,
    engines: Vec<&'a Engine<'a>>,
}

impl Aircraft<'_> {
    pub fn new(name_param: &str) -> Aircraft {
        Aircraft {
            name: name_param.to_string(),
            engines: Vec::new()
        }
    }
}

1 implementation
struct Engine<'a> {
    name: String,
    aircraft: &'a Aircraft<'a>
}

impl Engine<'_> {
    pub fn new<'a>(name_param: &'a str, aircraft_param: &'a Aircraft) -> Engine<'a> {
        Engine {
            name: name_param.to_string(),
            aircraft: aircraft_param
        }
    }
}

▶ Run | Debug
fn main() {
    let mut f18: Aircraft = Aircraft::new( name_param: "F-18" );
    let engine1: Engine = Engine::new( name_param: "General Electric F404" , aircraft_param: &f18);
    let engine2: Engine = Engine::new( name_param: "General Electric F404" , aircraft_param: &f18 );

    f18.engines.push (&engine1);
    f18.engines.push (&engine2);

    println! ("Aircraft: {} has a {} and {} ", f18.name, f18.engines[0].name, f18.engines[1].name );
}
```

Test data:

N/A

Sample output:



```
error[E0502]: cannot borrow `f18.engines` as mutable because it is also borrowed as immutable
  --> src\main.rs:34:5
   |
31 |     let engine1 = Engine::new( "General Electric F404" , &f18);
   |                                                          ---- immutable borrow occurs here
...
34 |     f18.engines.push (&engine1);
   |     ^^^^^^^^^^^^----^^^^^^^^^^^
   |     |           |
   |     |           immutable borrow later used by call
   |     mutable borrow occurs here

error[E0502]: cannot borrow `f18.engines` as mutable because it is also borrowed as immutable
  --> src\main.rs:35:5
   |
31 |     let engine1 = Engine::new( "General Electric F404" , &f18);
   |                                                          ---- immutable borrow occurs here
...
35 |     f18.engines.push (&engine2);
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^ mutable borrow occurs here
36 |
37 |     println! ("Aircraft: {} has a {} and {} ", f18.name, f18.engines[0].name, f18.engines[1].name );
   |                                                --------- immutable borrow later used here

For more information about this error, try `rustc --explain E0502`.
error: could not compile `ownership_limitations` due to 2 previous errors
```

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:

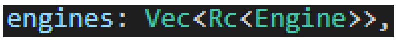
Part 1 – Examine the provided code. Explain what is happening with the reference counters and why we do not need to pass them as references

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

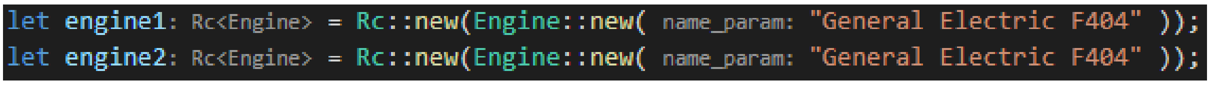
Solution:

Part 1 –



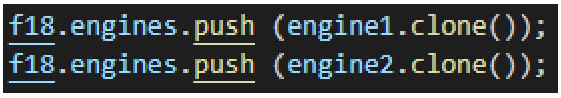
```
engines: Vec<Rc<Engine>>,
```

A vector of reference counters for Engine objects is created



```
let engine1: Rc<Engine> = Rc::new(Engine::new( name_param: "General Electric F404" ));
let engine2: Rc<Engine> = Rc::new(Engine::new( name_param: "General Electric F404" ));
```

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



```
f18.engines.push (engine1.clone());
f18.engines.push (engine2.clone());
```

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

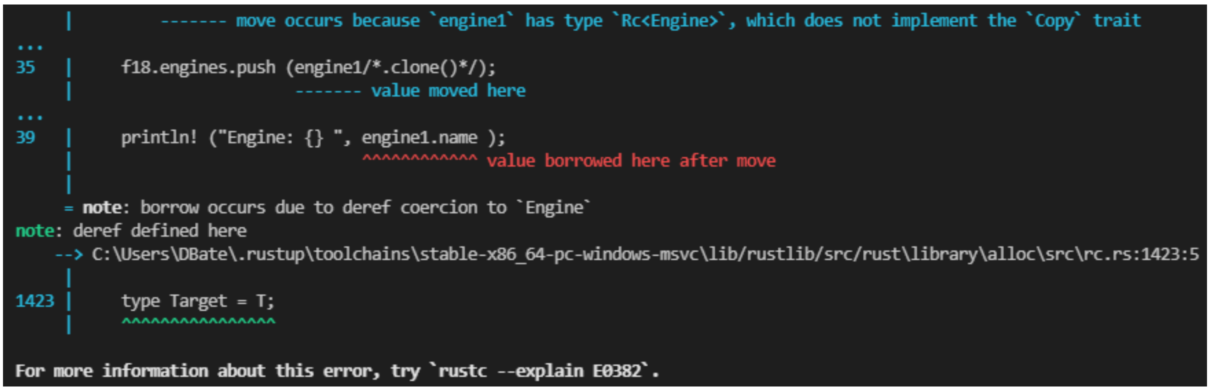
Test data:

Part 1 – N/A

Sample output:

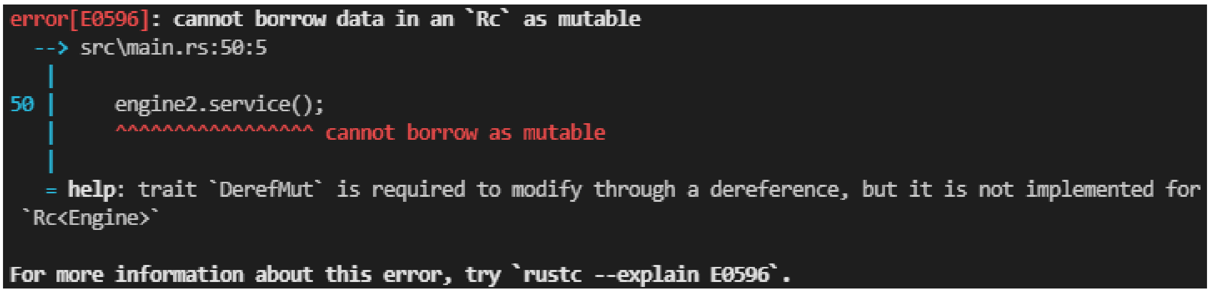
Part 1 – N/A

Part 2 –



```
     |         ------- move occurs because `engine1` has type `Rc<Engine>`, which does not implement the `Copy` trait
...
35   |     f18.engines.push (engine1/*.clone()*/);
     |                       ------- value moved here
...
39   |     println! ("Engine: {} ", engine1.name );
     |                              ^^^^^^^^^^^^ value borrowed here after move
     |
     = note: borrow occurs due to deref coercion to `Engine`
note: deref defined here
    --> C:\Users\DBate\.rustup\toolchains\stable-x86_64-pc-windows-msvc\lib/rustlib/src/rust\library\alloc\src\rc.rs:1423:5
     |
1423 |     type Target = T;
     |     ^^^^^^^^^^^^^^^^

For more information about this error, try `rustc --explain E0382`.
```

Part 3 –



```
error[E0596]: cannot borrow data in an `Rc` as mutable
  --> src\main.rs:50:5
   |
50 |     engine2.service();
   |     ^^^^^^^^^^^^^^^^^ cannot borrow as mutable
   |
   = help: trait `DerefMut` is required to modify through a dereference, but it is not implemented for
 `Rc<Engine>`

For more information about this error, try `rustc --explain E0596`.
```

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

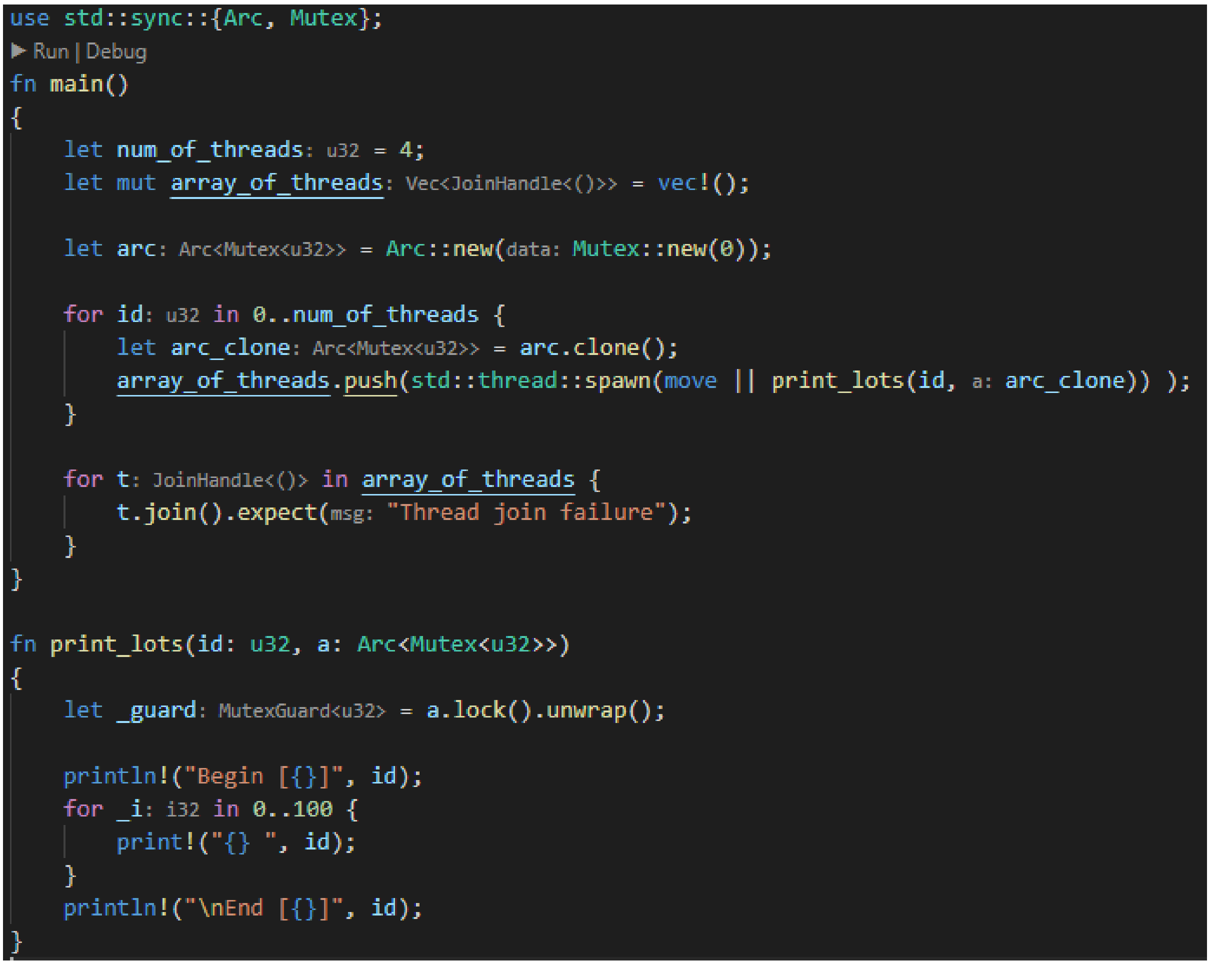
Question:

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 –

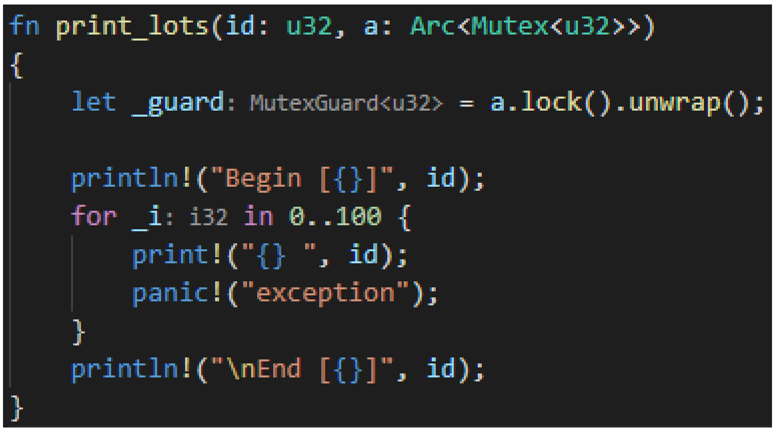


```
use std::sync::{Arc, Mutex};
▶ Run | Debug
fn main()
{
    let num_of_threads: u32 = 4;
    let mut array_of_threads: Vec<JoinHandle<()>> = vec!();

    let arc: Arc<Mutex<u32>> = Arc::new(data: Mutex::new(0));

    for id: u32 in 0..num_of_threads {
        let arc_clone: Arc<Mutex<u32>> = arc.clone();
        array_of_threads.push(std::thread::spawn(move || print_lots(id, a: arc_clone)) );
    }

    for t: JoinHandle<()> in array_of_threads {
        t.join().expect(msg: "Thread join failure");
    }
}

fn print_lots(id: u32, a: Arc<Mutex<u32>>)
{
    let _guard: MutexGuard<u32> = a.lock().unwrap();

    println!("Begin [{}]", id);
    for _i: i32 in 0..100 {
        print!("{} ", id);
    }
    println!("\nEnd [{}]", id);
}
```

Part 4 –

Q2.

Q4.



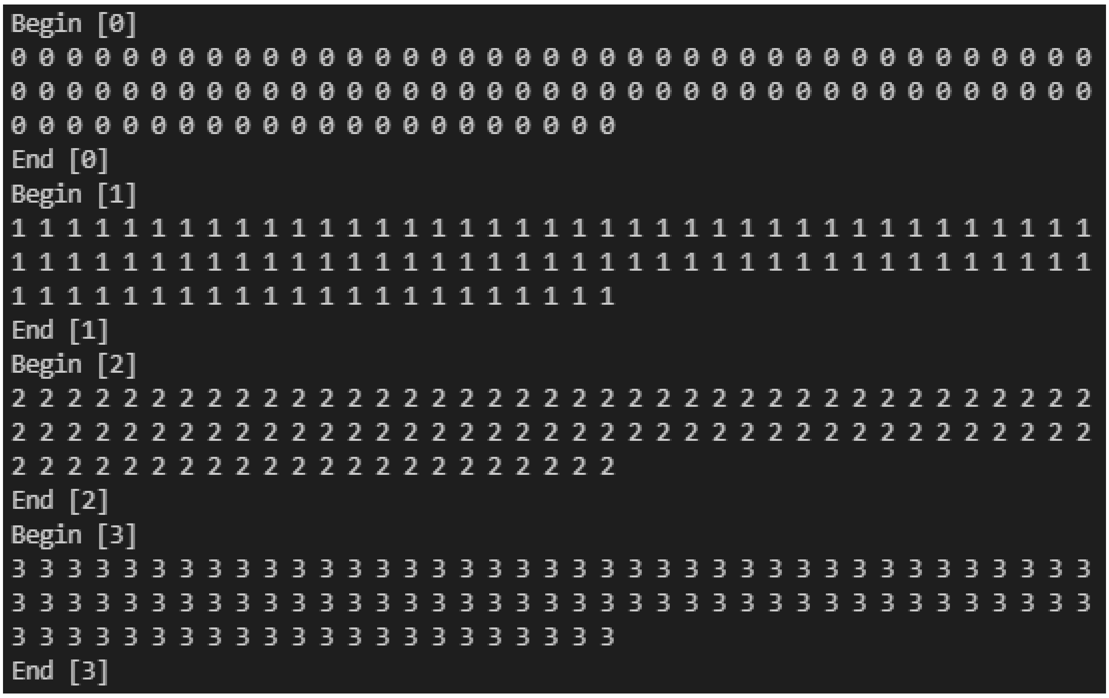
```
fn print_lots(id: u32, a: Arc<Mutex<u32>>)
{
    let _guard: MutexGuard<u32> = a.lock().unwrap();

    println!("Begin [{}]", id);
    for _i: i32 in 0..100 {
        print!("{} ", id);
        panic!("exception");
    }
    println!("\nEnd [{}]", id);
}
```

Test data:

All parts – N/A

Sample output:

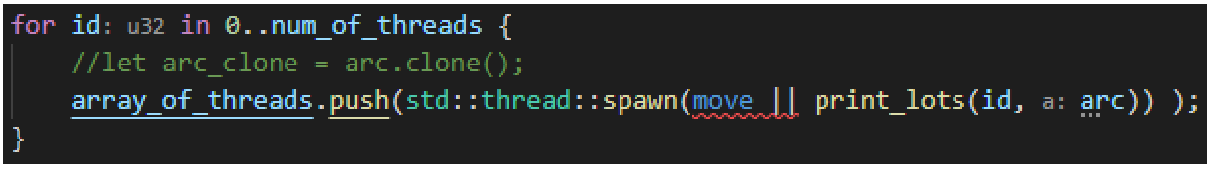
Part 1 –



```
Begin [0]
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
End [0]
Begin [1]
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
End [1]
Begin [2]
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
End [2]
Begin [3]
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
End [3]
```

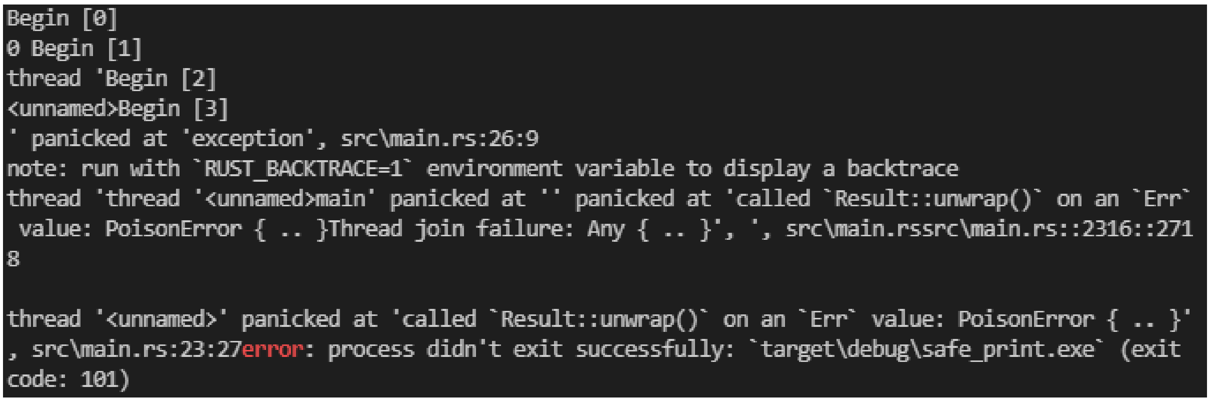
Part 2 –

Q2.



```
for id: u32 in 0..num_of_threads {
    //let arc_clone = arc.clone();
    array_of_threads.push(std::thread::spawn(move || print_lots(id, a: arc)) );
}
```

Q4.



```
Begin [0]
0 Begin [1]
thread 'Begin [2]
<unnamed>Begin [3]
' panicked at 'exception', src\main.rs:26:9
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
thread 'thread '<unnamed>main' panicked at '' panicked at 'called `Result::unwrap()` on an `Err`
 value: PoisonError { .. }Thread join failure: Any { .. }', ', src\main.rssrc\main.rs::2316::271
8

thread '<unnamed>' panicked at 'called `Result::unwrap()` on an `Err` value: PoisonError { .. }'
, src\main.rs:23:27error: process didn't exit successfully: `target\debug\safe_print.exe` (exit
code: 101)
```

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:

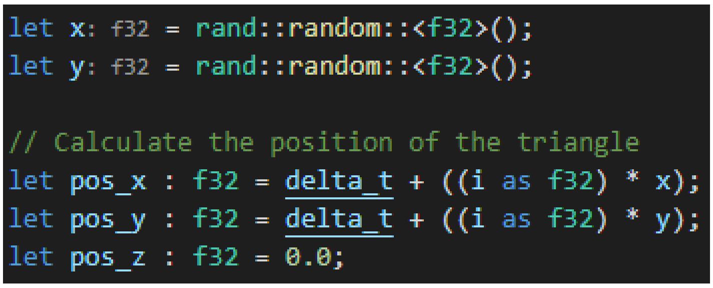
N/A

Q2. Triangles and OpenGL

Question:

Update the code to make the triangles move more chaotically

Solution:

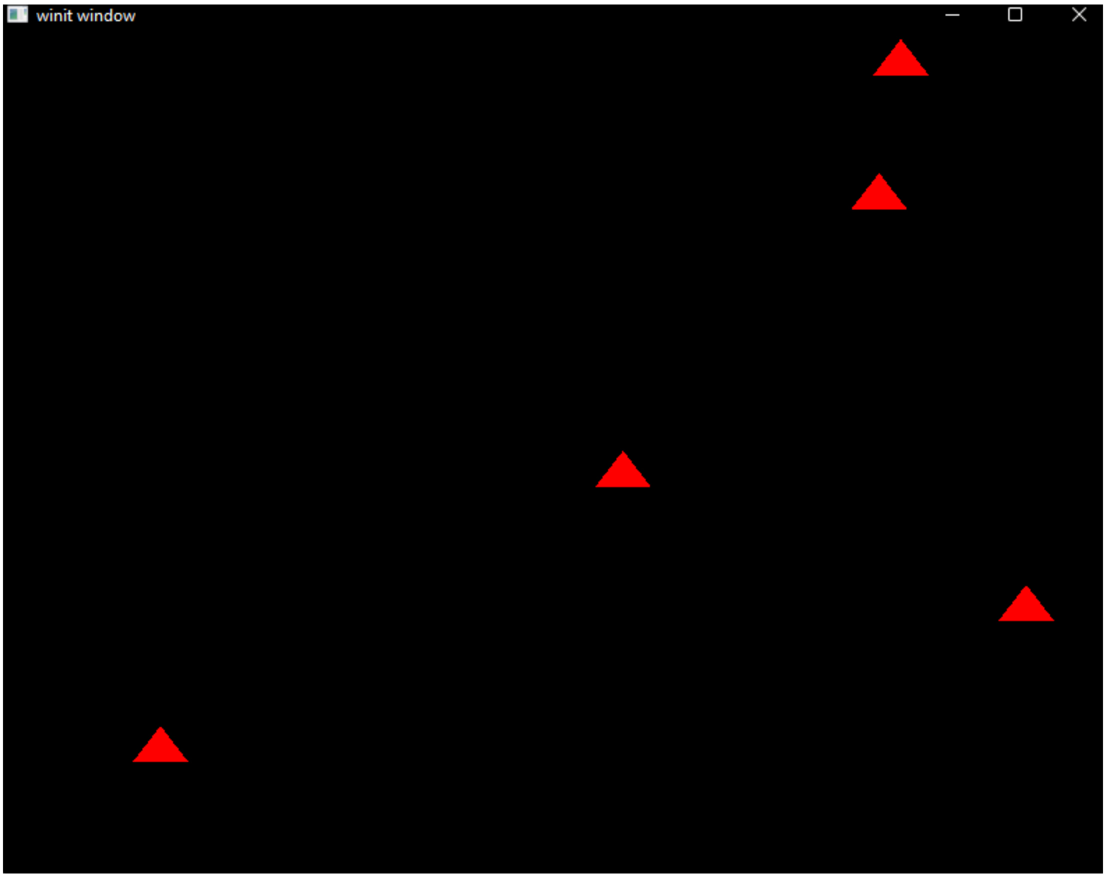


```
let x: f32 = rand::random::<f32>();
let y: f32 = rand::random::<f32>();

// Calculate the position of the triangle
let pos_x : f32 = delta_t + ((i as f32) * x);
let pos_y : f32 = delta_t + ((i as f32) * y);
let pos_z : f32 = 0.0;
```

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:

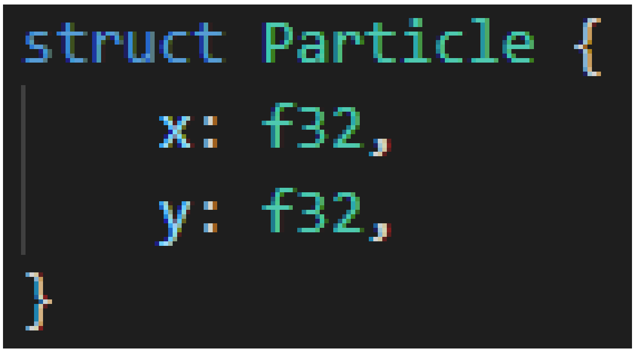
Part 1 – Implement structs described in the lab sheet

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

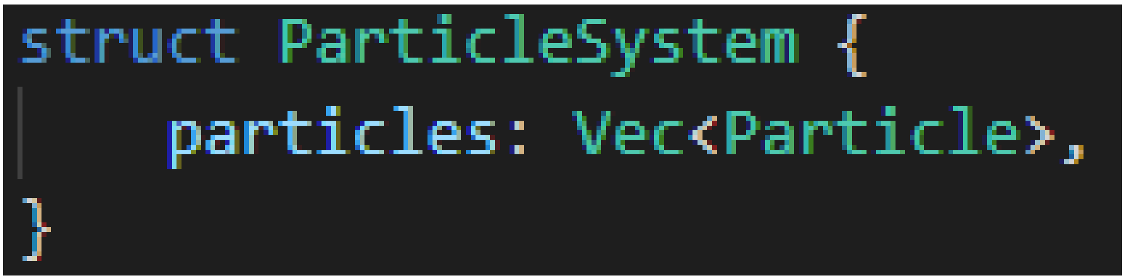
Part 3 – Test release mode and add the macro described in the lab sheet

Solution:

Part 1 –

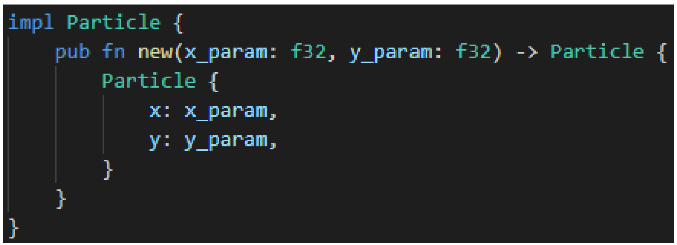


```rust
struct Particle {
    x: f32,
    y: f32,
}
```

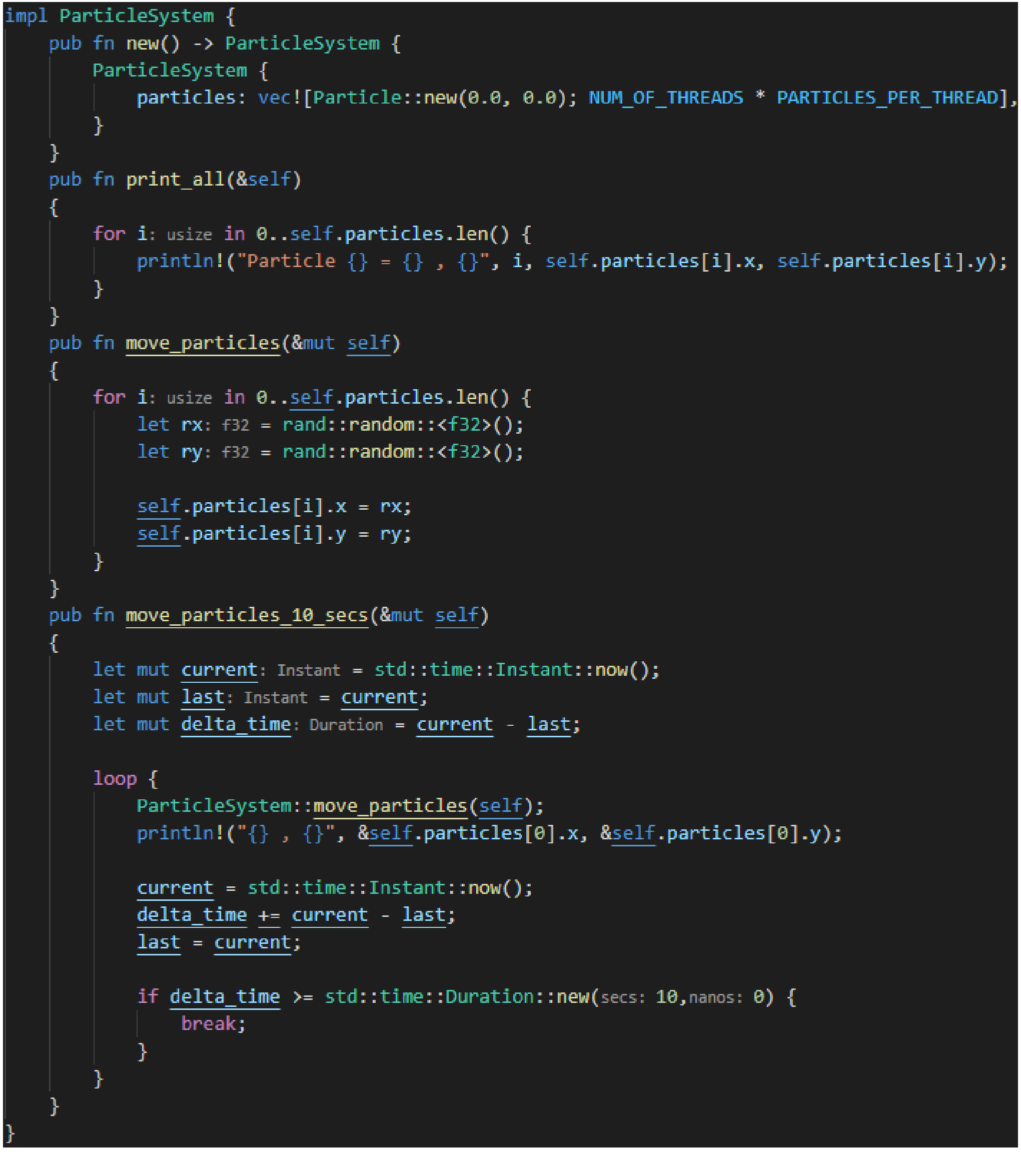


```rust
struct ParticleSystem {
    particles: Vec<Particle>,
}
```

Part 2 –

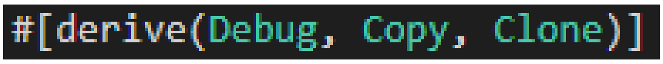


```
impl Particle {
    pub fn new(x_param: f32, y_param: f32) -> Particle {
        Particle {
            x: x_param,
            y: y_param,
        }
    }
}
```

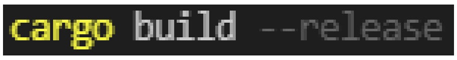


```
impl ParticleSystem {
    pub fn new() -> ParticleSystem {
        ParticleSystem {
            particles: vec![Particle::new(0.0, 0.0); NUM_OF_THREADS * PARTICLES_PER_THREAD],
        }
    }
    pub fn print_all(&self)
    {
        for i: usize in 0..self.particles.len() {
            println!("Particle {} = {} , {}", i, self.particles[i].x, self.particles[i].y);
        }
    }
    pub fn move_particles(&mut self)
    {
        for i: usize in 0..self.particles.len() {
            let rx: f32 = rand::random::<f32>();
            let ry: f32 = rand::random::<f32>();

            self.particles[i].x = rx;
            self.particles[i].y = ry;
        }
    }
    pub fn move_particles_10_secs(&mut self)
    {
        let mut current: Instant = std::time::Instant::now();
        let mut last: Instant = current;
        let mut delta_time: Duration = current - last;

        loop {
            ParticleSystem::move_particles(self);
            println!("{} , {}", &self.particles[0].x, &self.particles[0].y);

            current = std::time::Instant::now();
            delta_time += current - last;
            last = current;

            if delta_time >= std::time::Duration::new(secs: 10,nanos: 0) {
                break;
            }
        }
    }
}
```

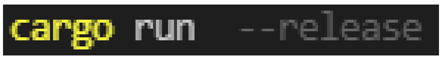
Part 3 –



```
#[derive(Debug, Copy, Clone)]
```



```
cargo build --release
```

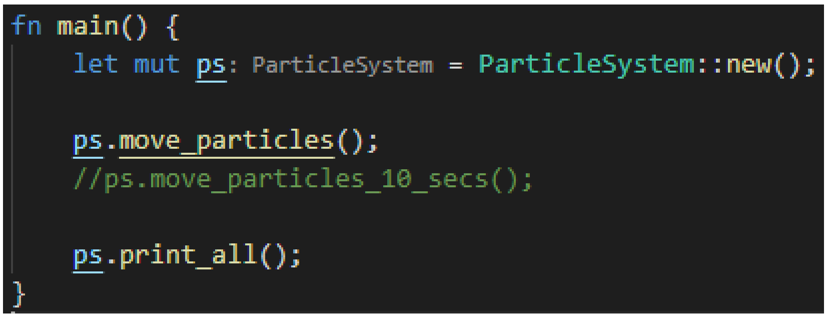


```
cargo run  --release
```

Test data:

Part 1 – N/A

Part 2 –



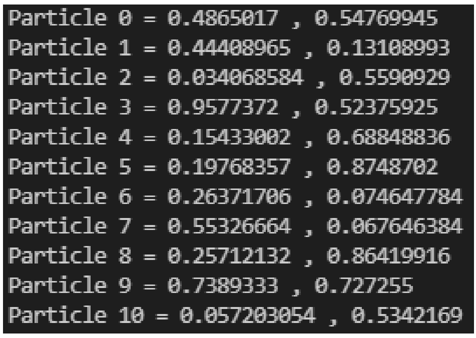
```
fn main() {
    let mut ps: ParticleSystem = ParticleSystem::new();

    ps.move_particles();
    //ps.move_particles_10_secs();

    ps.print_all();
}
```

Part 3 – N/A

Sample output:

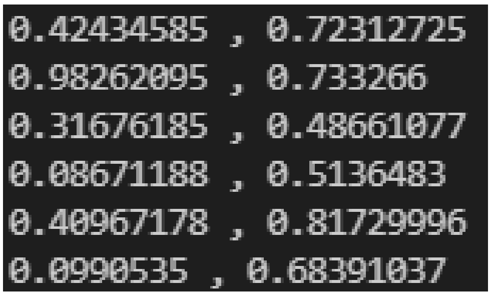
Part 1 – N/A

Part 2 –



```
Particle 0 = 0.4865017 , 0.54769945
Particle 1 = 0.44408965 , 0.13108993
Particle 2 = 0.034068584 , 0.5590929
Particle 3 = 0.9577372 , 0.52375925
Particle 4 = 0.15433002 , 0.68848836
Particle 5 = 0.19768357 , 0.8748702
Particle 6 = 0.26371706 , 0.074647784
Particle 7 = 0.55326664 , 0.067646384
Particle 8 = 0.25712132 , 0.86419916
Particle 9 = 0.7389333 , 0.727255
Particle 10 = 0.057203054 , 0.5342169
```

ETC.



```
0.42434585 , 0.72312725
0.98262095 , 0.733266
0.31676185 , 0.48661077
0.08671188 , 0.5136483
0.40967178 , 0.81729996
0.0990535 , 0.68391037
```

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:

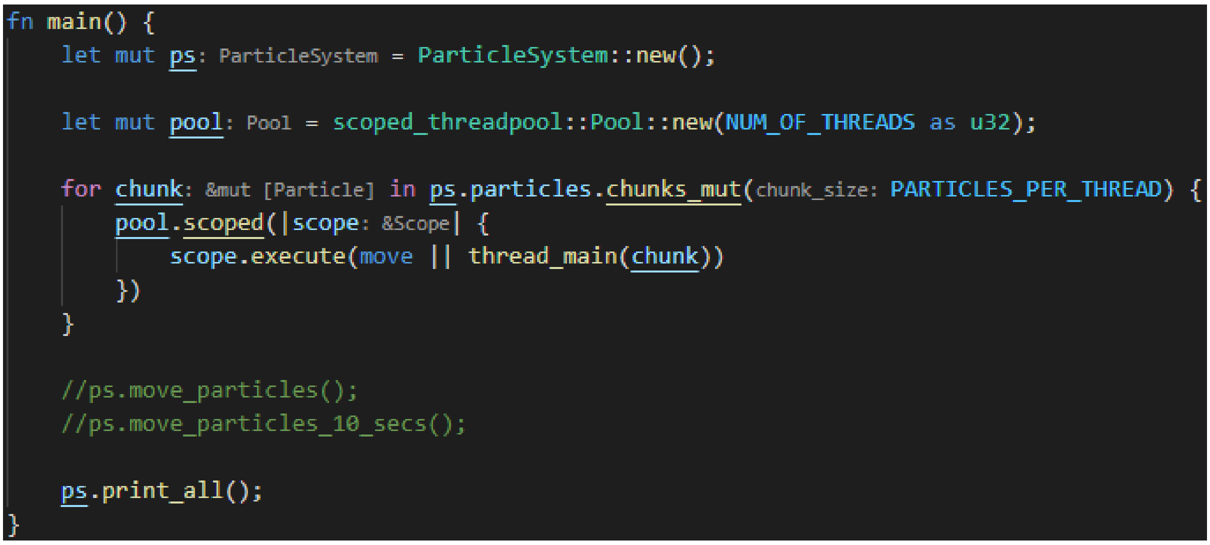
N/A

Q2. Threaded Particles

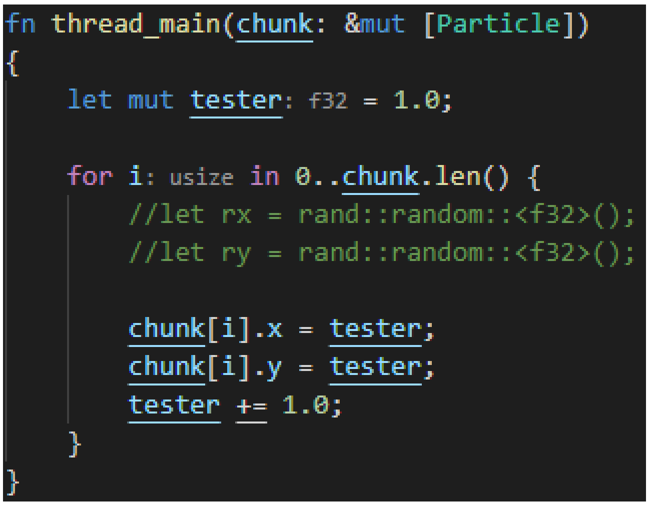
Question:

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

Solution:



```
fn main() {
    let mut ps: ParticleSystem = ParticleSystem::new();

    let mut pool: Pool = scoped_threadpool::Pool::new(NUM_OF_THREADS as u32);

    for chunk: &mut [Particle] in ps.particles.chunks_mut(chunk_size: PARTICLES_PER_THREAD) {
        pool.scoped(|scope: &Scope| {
            scope.execute(move || thread_main(chunk))
        })
    }

    //ps.move_particles();
    //ps.move_particles_10_secs();

    ps.print_all();
}
```

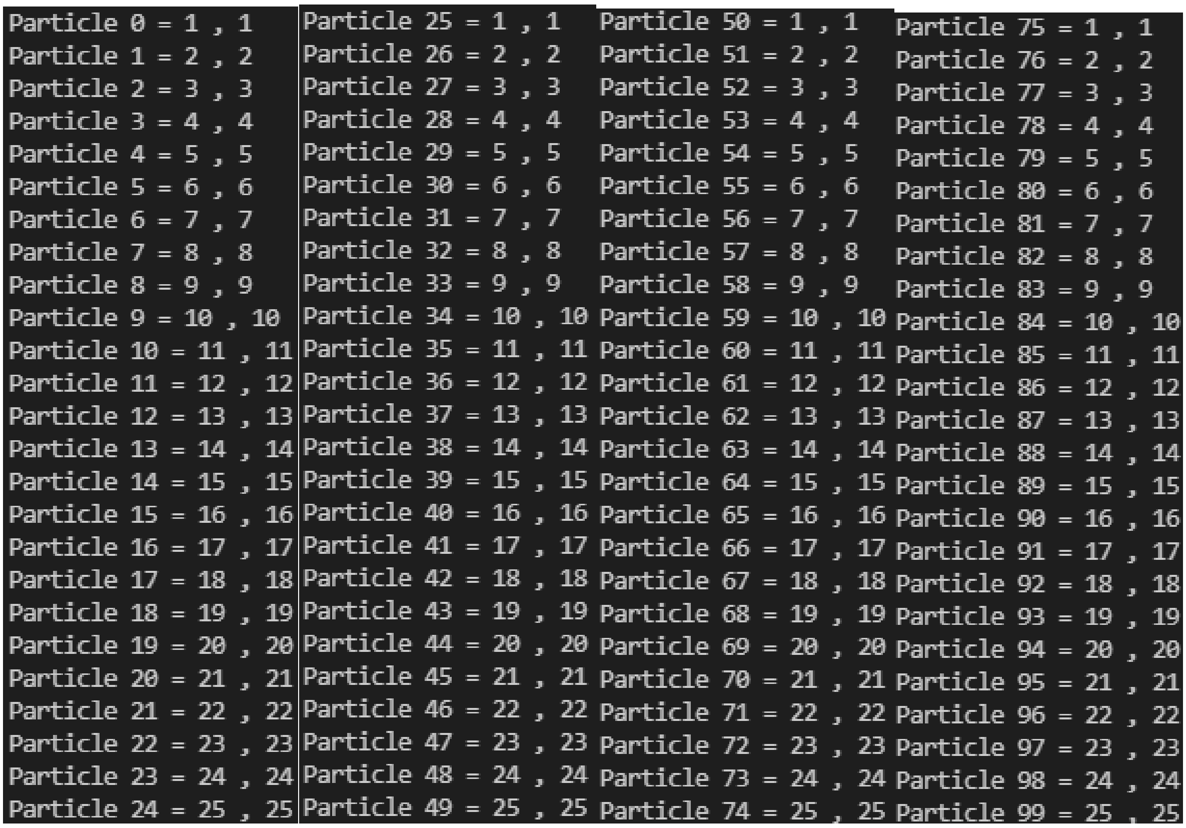


```
fn thread_main(chunk: &mut [Particle])
{
    let mut tester: f32 = 1.0;

    for i: usize in 0..chunk.len() {
        //let rx = rand::random::<f32>();
        //let ry = rand::random::<f32>();

        chunk[i].x = tester;
        chunk[i].y = tester;
        tester += 1.0;
    }
}
```

Test data:

N/A

Sample output:



```
Particle 0 = 1 , 1     Particle 25 = 1 , 1   Particle 50 = 1 , 1   Particle 75 = 1 , 1
Particle 1 = 2 , 2     Particle 26 = 2 , 2   Particle 51 = 2 , 2   Particle 76 = 2 , 2
Particle 2 = 3 , 3     Particle 27 = 3 , 3   Particle 52 = 3 , 3   Particle 77 = 3 , 3
Particle 3 = 4 , 4     Particle 28 = 4 , 4   Particle 53 = 4 , 4   Particle 78 = 4 , 4
Particle 4 = 5 , 5     Particle 29 = 5 , 5   Particle 54 = 5 , 5   Particle 79 = 5 , 5
Particle 5 = 6 , 6     Particle 30 = 6 , 6   Particle 55 = 6 , 6   Particle 80 = 6 , 6
Particle 6 = 7 , 7     Particle 31 = 7 , 7   Particle 56 = 7 , 7   Particle 81 = 7 , 7
Particle 7 = 8 , 8     Particle 32 = 8 , 8   Particle 57 = 8 , 8   Particle 82 = 8 , 8
Particle 8 = 9 , 9     Particle 33 = 9 , 9   Particle 58 = 9 , 9   Particle 83 = 9 , 9
Particle 9 = 10 , 10   Particle 34 = 10 , 10 Particle 59 = 10 , 10 Particle 84 = 10 , 10
Particle 10 = 11 , 11  Particle 35 = 11 , 11 Particle 60 = 11 , 11 Particle 85 = 11 , 11
Particle 11 = 12 , 12  Particle 36 = 12 , 12 Particle 61 = 12 , 12 Particle 86 = 12 , 12
Particle 12 = 13 , 13  Particle 37 = 13 , 13 Particle 62 = 13 , 13 Particle 87 = 13 , 13
Particle 13 = 14 , 14  Particle 38 = 14 , 14 Particle 63 = 14 , 14 Particle 88 = 14 , 14
Particle 14 = 15 , 15  Particle 39 = 15 , 15 Particle 64 = 15 , 15 Particle 89 = 15 , 15
Particle 15 = 16 , 16  Particle 40 = 16 , 16 Particle 65 = 16 , 16 Particle 90 = 16 , 16
Particle 16 = 17 , 17  Particle 41 = 17 , 17 Particle 66 = 17 , 17 Particle 91 = 17 , 17
Particle 17 = 18 , 18  Particle 42 = 18 , 18 Particle 67 = 18 , 18 Particle 92 = 18 , 18
Particle 18 = 19 , 19  Particle 43 = 19 , 19 Particle 68 = 19 , 19 Particle 93 = 19 , 19
Particle 19 = 20 , 20  Particle 44 = 20 , 20 Particle 69 = 20 , 20 Particle 94 = 20 , 20
Particle 20 = 21 , 21  Particle 45 = 21 , 21 Particle 70 = 21 , 21 Particle 95 = 21 , 21
Particle 21 = 22 , 22  Particle 46 = 22 , 22 Particle 71 = 22 , 22 Particle 96 = 22 , 22
Particle 22 = 23 , 23  Particle 47 = 23 , 23 Particle 72 = 23 , 23 Particle 97 = 23 , 23
Particle 23 = 24 , 24  Particle 48 = 24 , 24 Particle 73 = 24 , 24 Particle 98 = 24 , 24
Particle 24 = 25 , 25  Particle 49 = 25 , 25 Particle 74 = 25 , 25 Particle 99 = 25 , 25
```

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

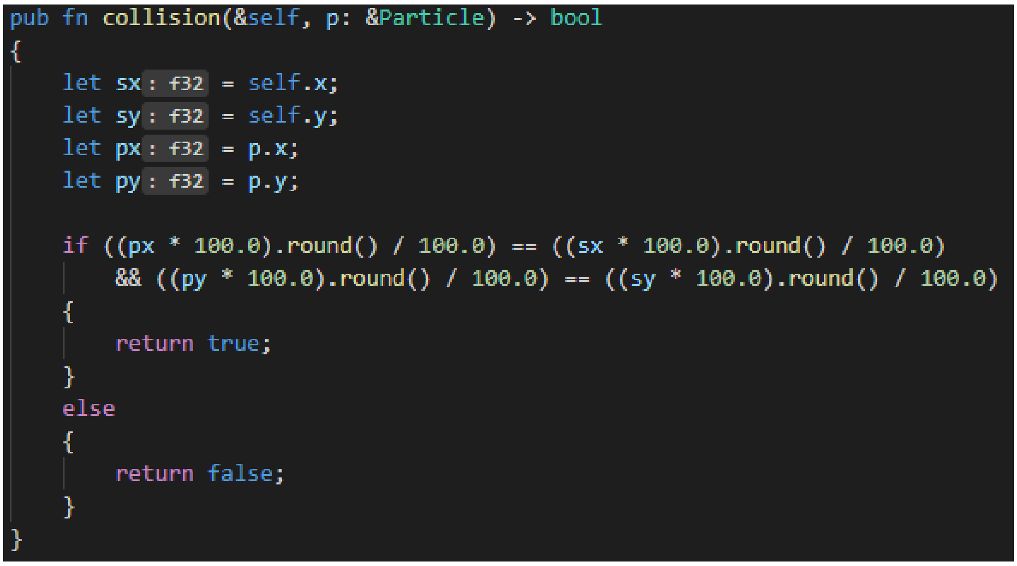
Question:

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

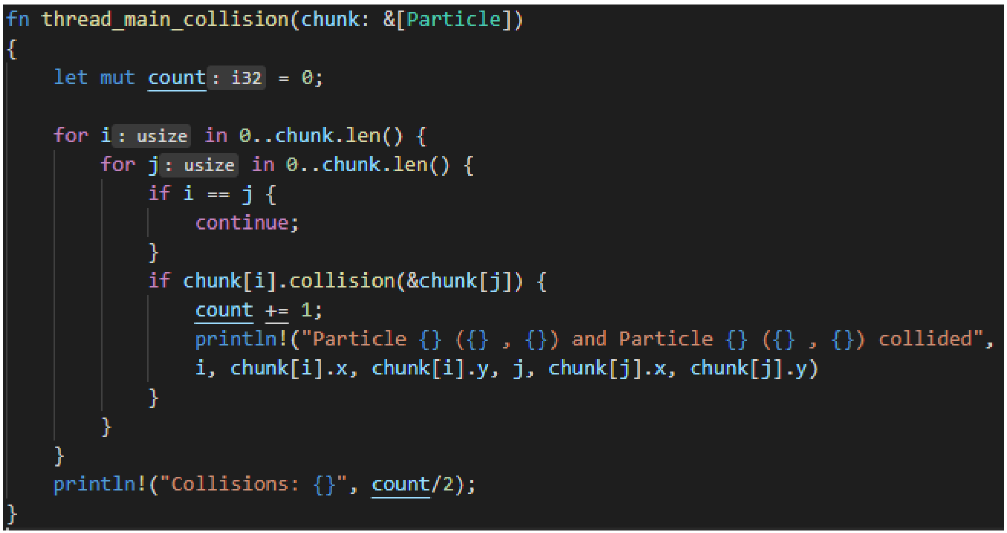
Solution:

Part 1 –

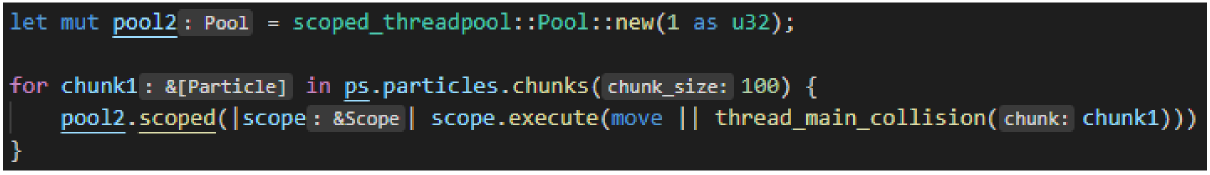


```
pub fn collision(&self, p: &Particle) -> bool
{
    let sx: f32 = self.x;
    let sy: f32 = self.y;
    let px: f32 = p.x;
    let py: f32 = p.y;

    if ((px * 100.0).round() / 100.0) == ((sx * 100.0).round() / 100.0)
        && ((py * 100.0).round() / 100.0) == ((sy * 100.0).round() / 100.0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Part 2 –



```
fn thread_main_collision(chunk: &[Particle])
{
    let mut count: i32 = 0;

    for i: usize in 0..chunk.len() {
        for j: usize in 0..chunk.len() {
            if i == j {
                continue;
            }
            if chunk[i].collision(&chunk[j]) {
                count += 1;
                println!("Particle {} ({} , {}) and Particle {} ({} , {}) collided",
                i, chunk[i].x, chunk[i].y, j, chunk[j].x, chunk[j].y)
            }
        }
    }
    println!("Collisions: {}", count/2);
}
```



```
let mut pool2: Pool = scoped_threadpool::Pool::new(1 as u32);

for chunk1: &[Particle] in ps.particles.chunks(chunk_size: 100) {
    pool2.scoped(|scope: &Scope| scope.execute(move || thread_main_collision(chunk: chunk1)))
}
```

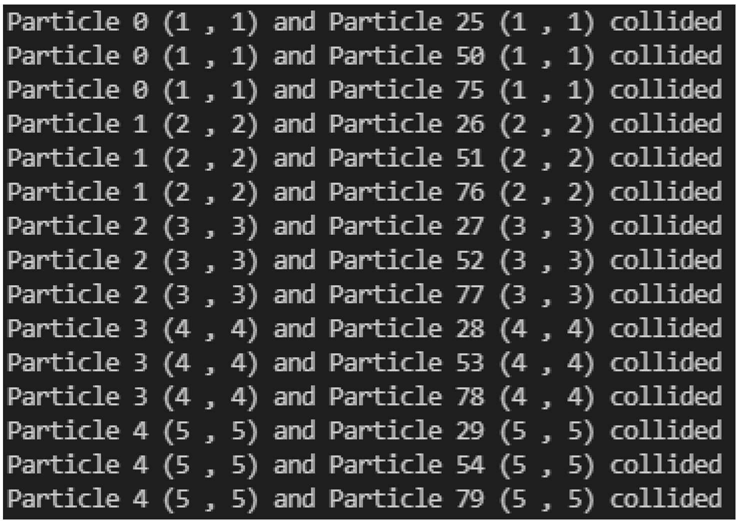
Test data:

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

Sample output:

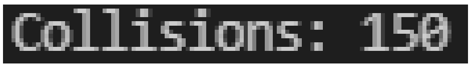
Part 1 – N/A

Part 2 –



```
Particle 0 (1 , 1) and Particle 25 (1 , 1) collided
Particle 0 (1 , 1) and Particle 50 (1 , 1) collided
Particle 0 (1 , 1) and Particle 75 (1 , 1) collided
Particle 1 (2 , 2) and Particle 26 (2 , 2) collided
Particle 1 (2 , 2) and Particle 51 (2 , 2) collided
Particle 1 (2 , 2) and Particle 76 (2 , 2) collided
Particle 2 (3 , 3) and Particle 27 (3 , 3) collided
Particle 2 (3 , 3) and Particle 52 (3 , 3) collided
Particle 2 (3 , 3) and Particle 77 (3 , 3) collided
Particle 3 (4 , 4) and Particle 28 (4 , 4) collided
Particle 3 (4 , 4) and Particle 53 (4 , 4) collided
Particle 3 (4 , 4) and Particle 78 (4 , 4) collided
Particle 4 (5 , 5) and Particle 29 (5 , 5) collided
Particle 4 (5 , 5) and Particle 54 (5 , 5) collided
Particle 4 (5 , 5) and Particle 79 (5 , 5) collided
```

ETC.



```
Collisions: 150
```

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:

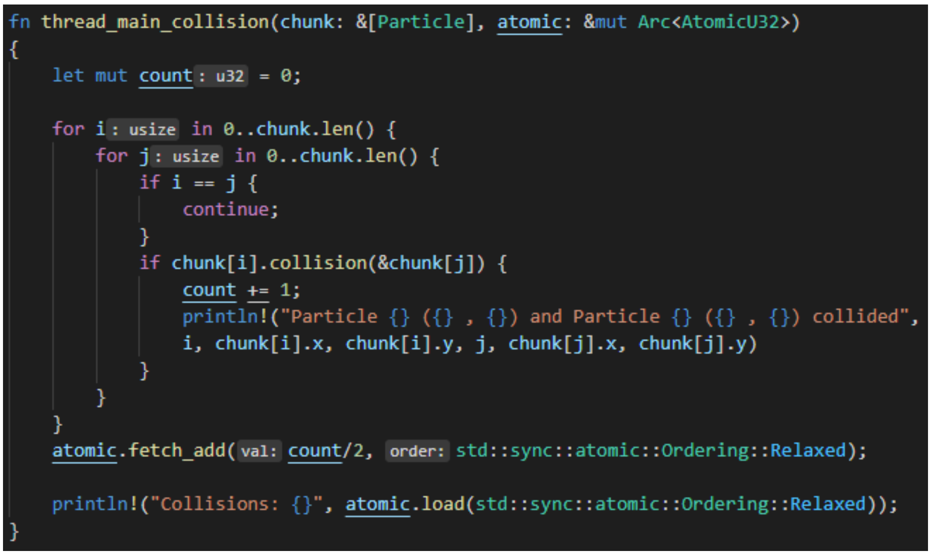
I'm not sure how using chunks for the collisions would work. How would you be able to check for collisions between chunks

Q2. Recording collisions using an Atomic

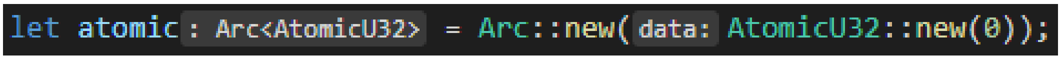
Question:

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

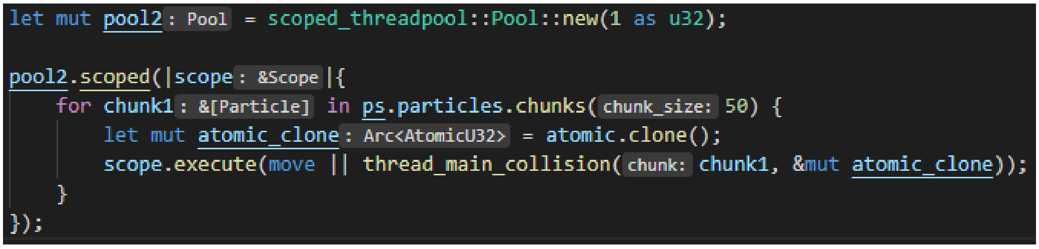
Solution:



```
fn thread_main_collision(chunk: &[Particle], atomic: &mut Arc<AtomicU32>)
{
    let mut count: u32 = 0;

    for i: usize in 0..chunk.len() {
        for j: usize in 0..chunk.len() {
            if i == j {
                continue;
            }
            if chunk[i].collision(&chunk[j]) {
                count += 1;
                println!("Particle {} ({} , {}) and Particle {} ({} , {}) collided",
                i, chunk[i].x, chunk[i].y, j, chunk[j].x, chunk[j].y)
            }
        }
    }
    atomic.fetch_add(val: count/2, order: std::sync::atomic::Ordering::Relaxed);

    println!("Collisions: {}", atomic.load(std::sync::atomic::Ordering::Relaxed));
}
```



```
let atomic: Arc<AtomicU32> = Arc::new(data: AtomicU32::new(0));
```

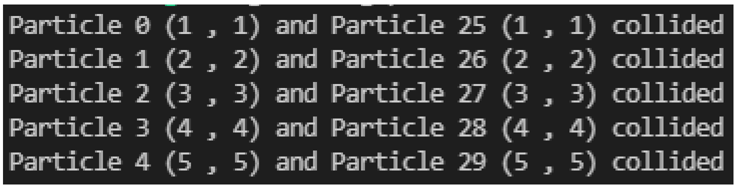


```
let mut pool2: Pool = scoped_threadpool::Pool::new(1 as u32);

pool2.scoped(|scope: &Scope|{
    for chunk1: &[Particle] in ps.particles.chunks(chunk_size: 50) {
        let mut atomic_clone: Arc<AtomicU32> = atomic.clone();
        scope.execute(move || thread_main_collision(chunk: chunk1, &mut atomic_clone));
    }
});
```

Test data:

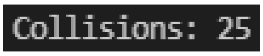
N/A

Sample output:



```
Particle 0 (1 , 1) and Particle 25 (1 , 1) collided
Particle 1 (2 , 2) and Particle 26 (2 , 2) collided
Particle 2 (3 , 3) and Particle 27 (3 , 3) collided
Particle 3 (4 , 4) and Particle 28 (4 , 4) collided
Particle 4 (5 , 5) and Particle 29 (5 , 5) collided
```

ETC.



```
Collisions: 25
```

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

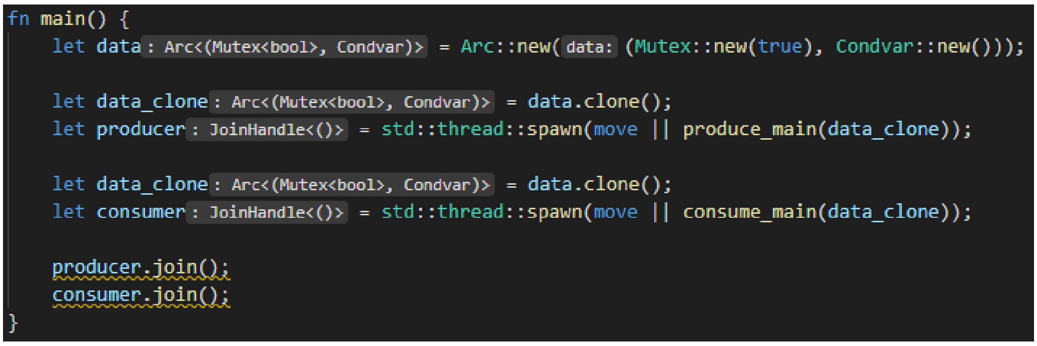
Date: 23/03/2022

Q1. Condition variables

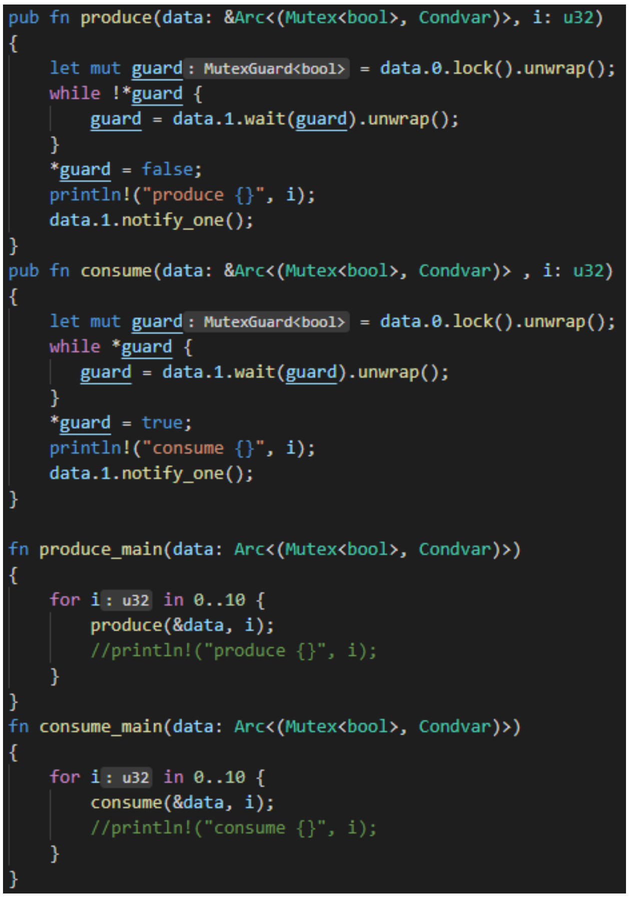
Question:

Create a producer / consumer model using condition variables

Solution:



```
fn main() {
    let data: Arc<(Mutex<bool>, Condvar)> = Arc::new(data: (Mutex::new(true), Condvar::new()));

    let data_clone: Arc<(Mutex<bool>, Condvar)> = data.clone();
    let producer: JoinHandle<()> = std::thread::spawn(move || produce_main(data_clone));

    let data_clone: Arc<(Mutex<bool>, Condvar)> = data.clone();
    let consumer: JoinHandle<()> = std::thread::spawn(move || consume_main(data_clone));

    producer.join();
    consumer.join();
}
```

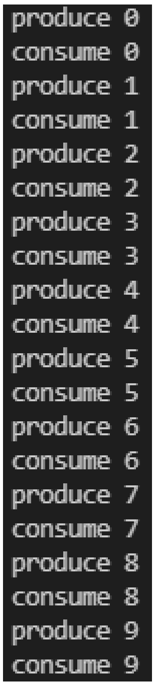


```
pub fn produce(data: &Arc<(Mutex<bool>, Condvar)>, i: u32)
{
    let mut guard: MutexGuard<bool> = data.0.lock().unwrap();
    while !*guard {
        guard = data.1.wait(guard).unwrap();
    }
    *guard = false;
    println!("produce {}", i);
    data.1.notify_one();
}
pub fn consume(data: &Arc<(Mutex<bool>, Condvar)> , i: u32)
{
    let mut guard: MutexGuard<bool> = data.0.lock().unwrap();
    while *guard {
        guard = data.1.wait(guard).unwrap();
    }
    *guard = true;
    println!("consume {}", i);
    data.1.notify_one();
}

fn produce_main(data: Arc<(Mutex<bool>, Condvar)>)
{
    for i: u32 in 0..10 {
        produce(&data, i);
        //println!("produce {}", i);
    }
}
fn consume_main(data: Arc<(Mutex<bool>, Condvar)>)
{
    for i: u32 in 0..10 {
        consume(&data, i);
        //println!("consume {}", i);
    }
}
```

Test data:

N/A

Sample output:



```
produce 0
consume 0
produce 1
consume 1
produce 2
consume 2
produce 3
consume 3
produce 4
consume 4
produce 5
consume 5
produce 6
consume 6
produce 7
consume 7
produce 8
consume 8
produce 9
consume 9
```

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:

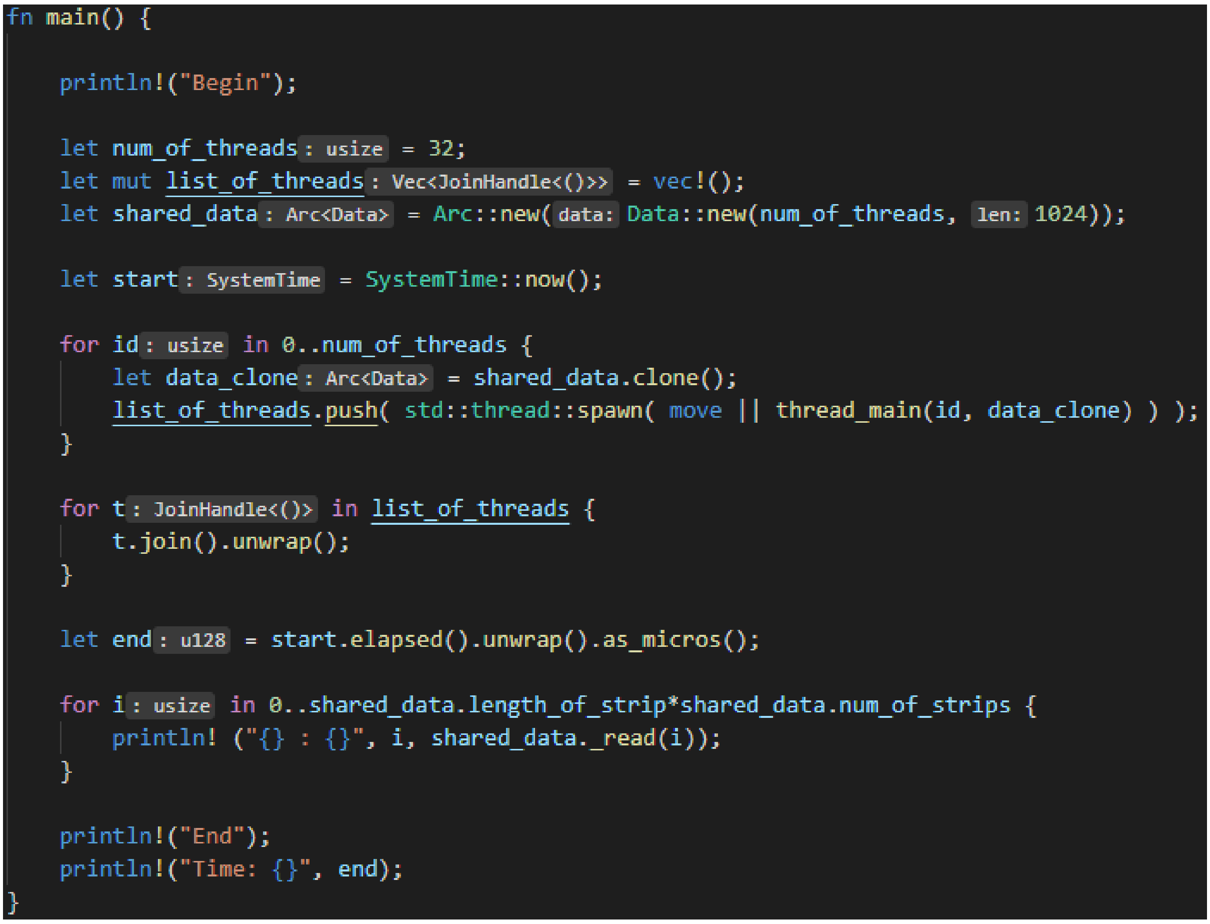
N/A

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:

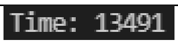
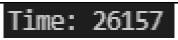
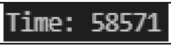
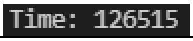
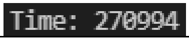


```
fn main() {

    println!("Begin");

    let num_of_threads: usize = 32;
    let mut list_of_threads: Vec<JoinHandle<()>> = vec!();
    let shared_data: Arc<Data> = Arc::new(data: Data::new(num_of_threads, len: 1024));

    let start: SystemTime = SystemTime::now();

    for id: usize in 0..num_of_threads {
        let data_clone: Arc<Data> = shared_data.clone();
        list_of_threads.push( std::thread::spawn( move || thread_main(id, data_clone) ) );
    }

    for t: JoinHandle<()> in list_of_threads {
        t.join().unwrap();
    }

    let end: u128 = start.elapsed().unwrap().as_micros();

    for i: usize in 0..shared_data.length_of_strip*shared_data.num_of_strips {
        println! ("{} : {}", i, shared_data._read(i));
    }

    println!("End");
    println!("Time: {}", end);
}
```

Test data:

2, 4, 8, 16, 32 threads

Sample output:

Threads	Strips	Time
2	16384	 Time: 13491
4	8192	 Time: 26157
8	4096	 Time: 58571
16	2048	 Time: 126515
32	1024	 Time: 270994

Reflection:

The timing seems to double each time

Metadata:

Timings

Further information:

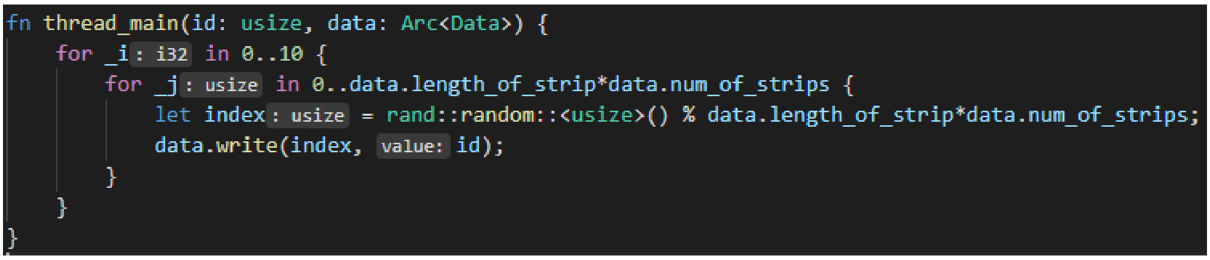
N/A

Q3. Striped arrays, random access

Question:

Modify the code to implement random access

Solution:

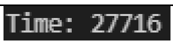
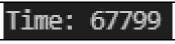
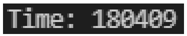
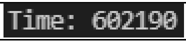
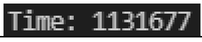


```
fn thread_main(id: usize, data: Arc<Data>) {
    for _i: i32 in 0..10 {
        for _j: usize in 0..data.length_of_strip*data.num_of_strips {
            let index: usize = rand::random::<usize>() % data.length_of_strip*data.num_of_strips;
            data.write(index, value: id);
        }
    }
}
```

Test data:

2, 4, 8, 16, 32 threads

Sample output:

Threads	Strips	Time
2	16384	 Time: 27716
4	8192	 Time: 67799
8	4096	 Time: 180409
16	2048	 Time: 602190
32	1024	 Time: 1131677

Reflection:

Random access seems to be significantly worse for scaling

Metadata:

Timings 2

Further information:

N/A

CUDA Labs

Week 1 – Lab 1

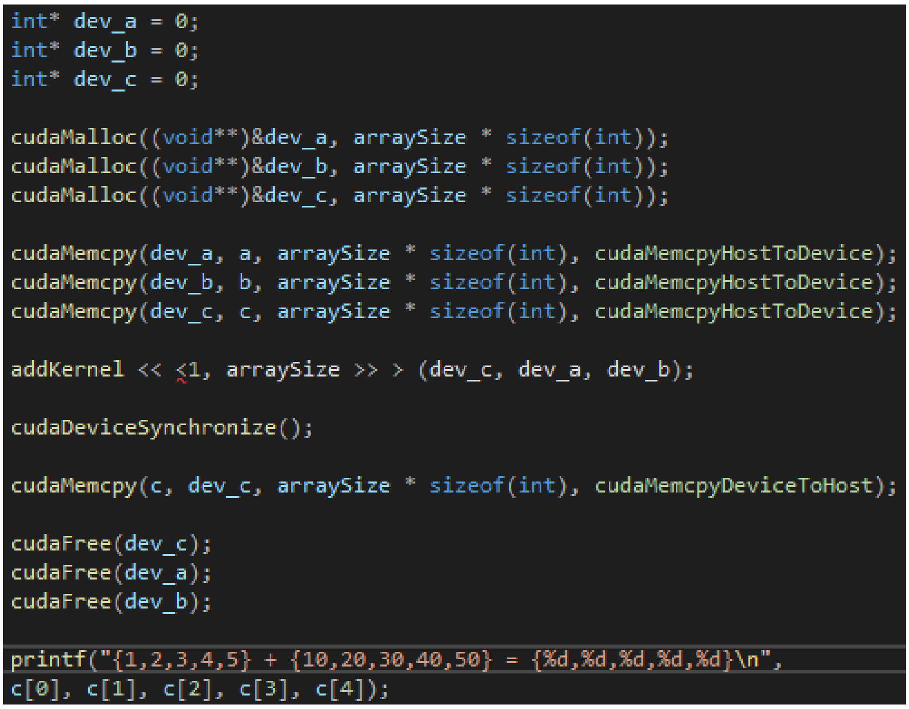
Date: 2nd Feb 2022

Q1. Setting up CUDA

Question:

Set up the default CUDA program. Create a solution that adds 2 arrays together on the GPU

Solution:



```
int* dev_a = 0;
int* dev_b = 0;
int* dev_c = 0;

cudaMalloc((void**)&dev_a, arraySize * sizeof(int));
cudaMalloc((void**)&dev_b, arraySize * sizeof(int));
cudaMalloc((void**)&dev_c, arraySize * sizeof(int));

cudaMemcpy(dev_a, a, arraySize * sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(dev_b, b, arraySize * sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(dev_c, c, arraySize * sizeof(int), cudaMemcpyHostToDevice);

addKernel << <1, arraySize >> > (dev_c, dev_a, dev_b);

cudaDeviceSynchronize();

cudaMemcpy(c, dev_c, arraySize * sizeof(int), cudaMemcpyDeviceToHost);

cudaFree(dev_c);
cudaFree(dev_a);
cudaFree(dev_b);

printf("{1,2,3,4,5} + {10,20,30,40,50} = {%d,%d,%d,%d,%d}\n",
c[0], c[1], c[2], c[3], c[4]);
```

Test data:

a = {1,2,3,4,5}   b = {10,20,30,40,50}

Sample output:

{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}

Reflection:

The process can be broken down into 5 steps

1. Allocate memory
2. Copy data to GPU
3. Perform addition
4. Copy results to CPU
5. Release GPU buffers

Metadata:

GPU vector addition

Further information:

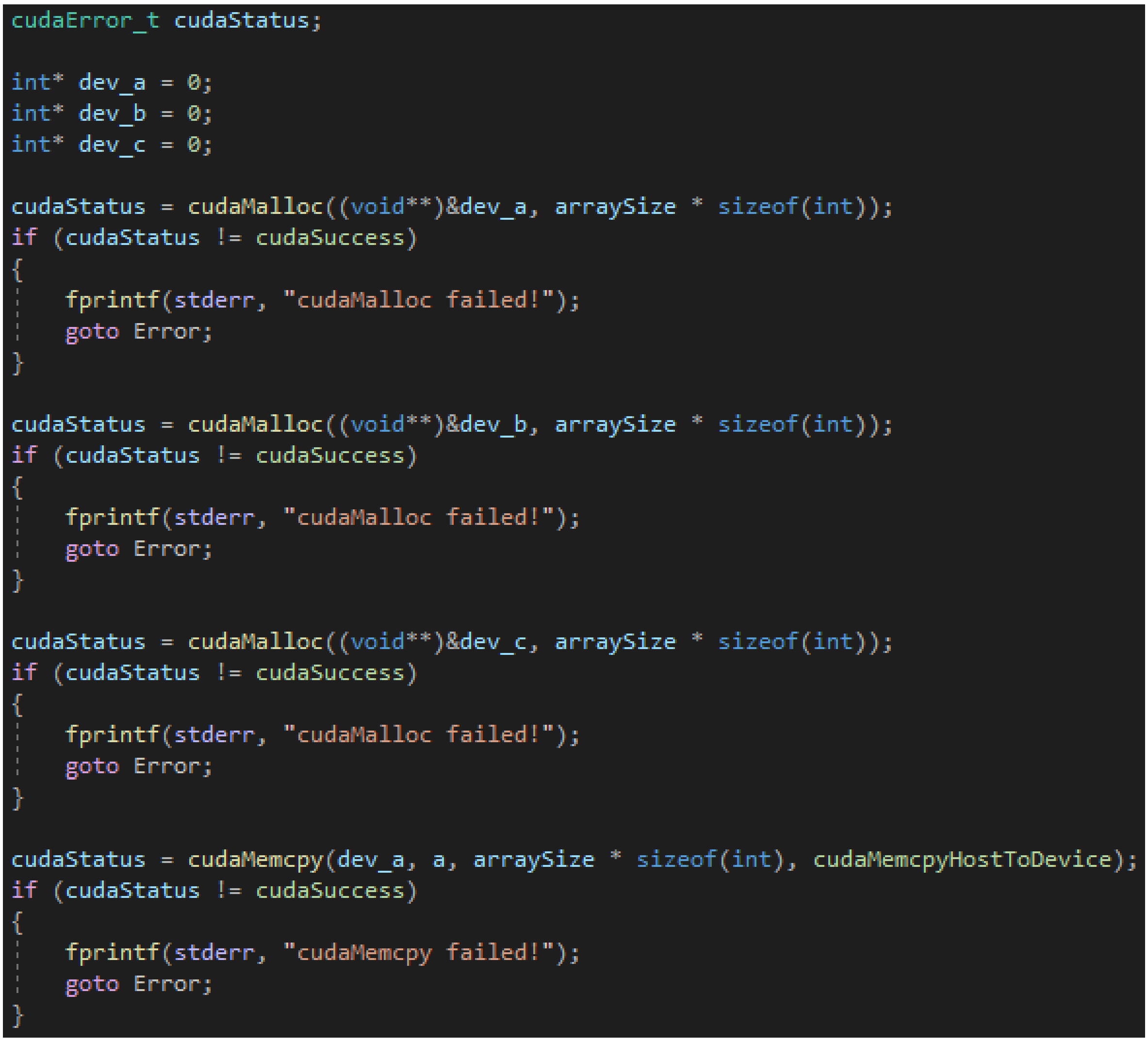
N/A

Q2. CUDA error checking

Question:

Add CUDA error checking

Solution:



```
cudaError_t cudaStatus;

int* dev_a = 0;
int* dev_b = 0;
int* dev_c = 0;

cudaStatus = cudaMalloc((void**)&dev_a, arraySize * sizeof(int));
if (cudaStatus != cudaSuccess)
{
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}

cudaStatus = cudaMalloc((void**)&dev_b, arraySize * sizeof(int));
if (cudaStatus != cudaSuccess)
{
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}

cudaStatus = cudaMalloc((void**)&dev_c, arraySize * sizeof(int));
if (cudaStatus != cudaSuccess)
{
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}

cudaStatus = cudaMemcpy(dev_a, a, arraySize * sizeof(int), cudaMemcpyHostToDevice);
if (cudaStatus != cudaSuccess)
{
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}
```

*Snippet own code (the whole thing takes up too much space)

Test data:

N/A

Sample output:

N/A

Reflection:

Error checking seems really simple to implement this way but I think using exceptions would be a better way to do it

Metadata:

Errors

Further information:

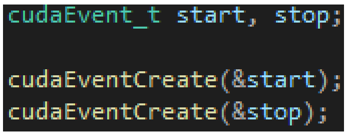
N/A

Q3. Check time range

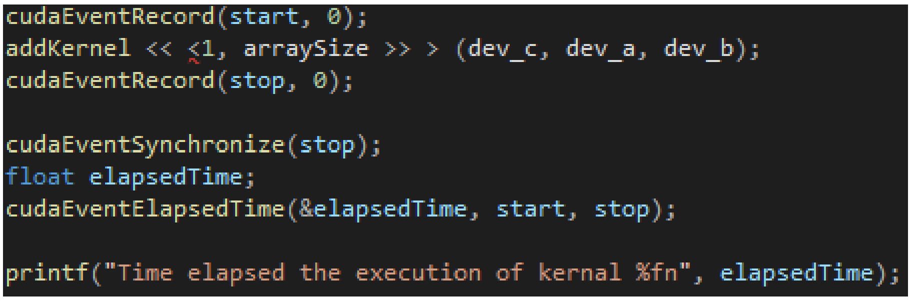
Question:

Create a timer for the kernel execution

Solution:



```
cudaEvent_t start, stop;

cudaEventCreate(&start);
cudaEventCreate(&stop);
```



```
cudaEventRecord(start, 0);
addKernel << <1, arraySize >> > (dev_c, dev_a, dev_b);
cudaEventRecord(stop, 0);

cudaEventSynchronize(stop);
float elapsedTime;
cudaEventElapsedTime(&elapsedTime, start, stop);

printf("Time elapsed the execution of kernal %fn", elapsedTime);
```

Test data:

N/A

Sample output:

N/A

Reflection:

Pretty simple to create a timer

Metadata:

Timer

Further information:

N/A

Week 2 – Lab 2

Date: 9th Feb 2022

Q1. Understand the block and thread indices

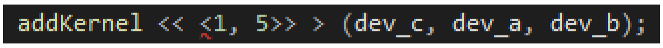
Question:

List the values for the built-in variables threadIdx.x and blockIdx.x corresponding to the given thread configurations used for executing the kernel addKernel( ) function on GPU

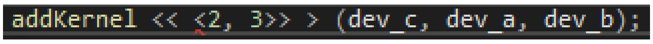
Solution:

N/A

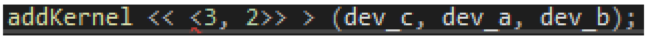
Test data:



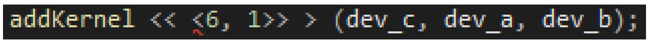
```
addKernel << <1, 5>> > (dev_c, dev_a, dev_b);
```



```
addKernel << <2, 3>> > (dev_c, dev_a, dev_b);
```

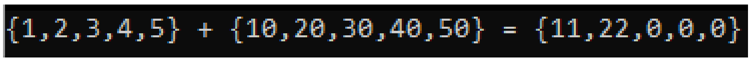


```
addKernel << <3, 2>> > (dev_c, dev_a, dev_b);
```

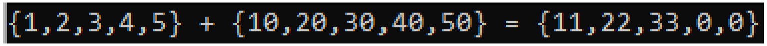


```
addKernel << <6, 1>> > (dev_c, dev_a, dev_b);
```

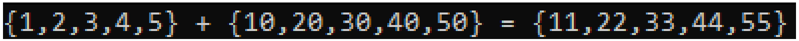
Sample output:



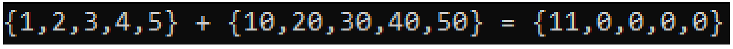
```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,0,0,0}
```



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,0,0}
```



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
```



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,0,0,0,0}
```

Reflection:

I think the first number in “addKernel << <1, 5 >> > (dev_c, dev_a, dev_b);” represents the number of blocks and the second represents the size of the blocks. So, in this case there is 1 block that has 5 threads.

Metadata:

Understanding threads and blocks

Further information:

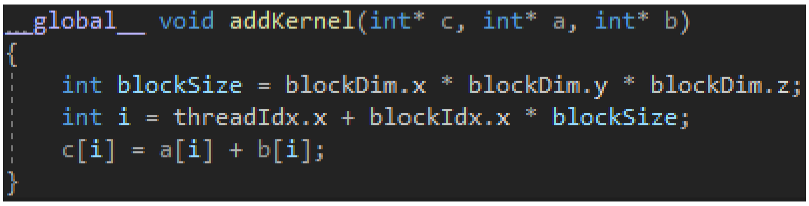
The lab sheet is a bit confusing to read to me, wasn’t really sure what was being asked of me

Q2. Find vector addition using multiple 1D thread blocks

Question:

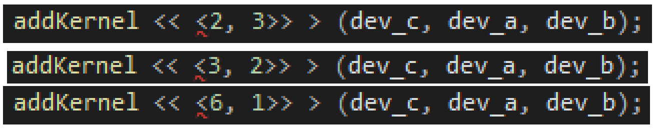
For the vector addition problem considered in the CUDA template, find the solution based on the given thread configurations.

Solution:



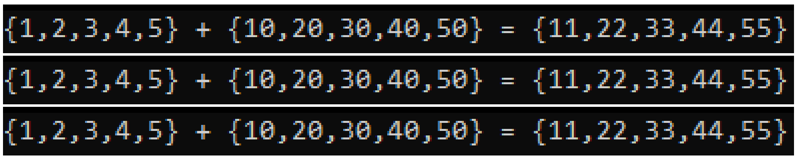
```
__global__ void addKernel(int* c, int* a, int* b)
{
    int blockSize = blockDim.x * blockDim.y * blockDim.z;
    int i = threadIdx.x + blockIdx.x * blockSize;
    c[i] = a[i] + b[i];
}
```

Test data:



```
addKernel << <2, 3>> > (dev_c, dev_a, dev_b);
addKernel << <3, 2>> > (dev_c, dev_a, dev_b);
addKernel << <6, 1>> > (dev_c, dev_a, dev_b);
```

Sample output:



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
```

Reflection:

A 1D vector with multiple blocks has an index equal to threadIdx.x + blockIdx.x * blockSize

Metadata:

Multiple 1D threads

Further information:

N/A

Q3. Understand the thread indices for 2D blocks

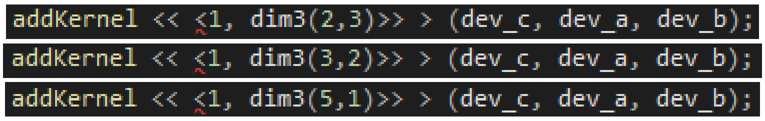
Question:

List the values for the built-in variables threadIdx.x and threadIdx.y corresponding to given thread configurations used for executing the kernel addKernel( ) function on GPU

Solution:

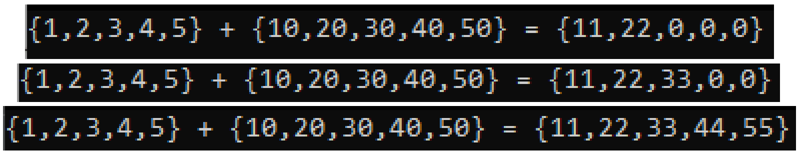
N/A

Test data:



```
addKernel << <1, dim3(2,3)>> > (dev_c, dev_a, dev_b);
addKernel << <1, dim3(3,2)>> > (dev_c, dev_a, dev_b);
addKernel << <1, dim3(5,1)>> > (dev_c, dev_a, dev_b);
```

Sample output:



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,0,0,0}
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,0,0}
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
```

Reflection:

Only the first index of the dim3 is currently used.

Metadata:

Indices for 2D blocks

Further information:

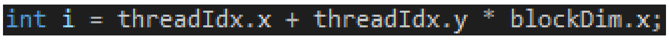
N/A

Q4. Find vector addition using one 2D thread block

Question:

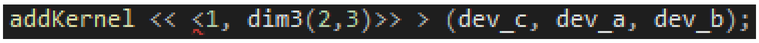
For the vector addition problem considered in the CUDA template, find the solution based on the given thread configurations

Solution:

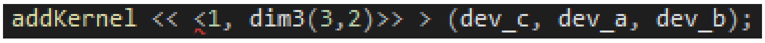


```
int i = threadIdx.x + threadIdx.y * blockDim.x;
```

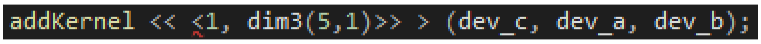
Test data:



```
addKernel << <1, dim3(2,3)>> > (dev_c, dev_a, dev_b);
```

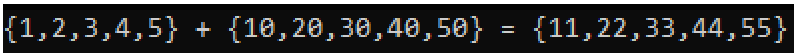


```
addKernel << <1, dim3(3,2)>> > (dev_c, dev_a, dev_b);
```

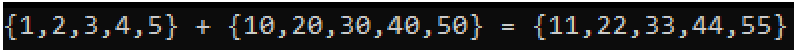


```
addKernel << <1, dim3(5,1)>> > (dev_c, dev_a, dev_b);
```

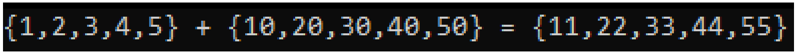
Sample output:



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
```



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
```



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
```

Reflection:

The threads index in a 2D thread block is equal to threadIdx.x + threadIdx.y * blockDim.x

Metadata:

One 2D block

Further information:

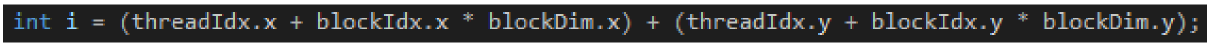
N/A

Q5. Find vector addition using multiple 2D thread blocks

Question:

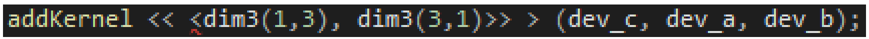
For the vector addition problem considered in the CUDA template, find the solution based on the given thread configurations

Solution:

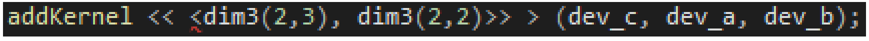


```
int i = (threadIdx.x + blockIdx.x * blockDim.x) + (threadIdx.y + blockIdx.y * blockDim.y);
```

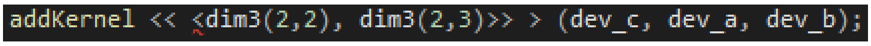
Test data:



```
addKernel << <dim3(1,3), dim3(3,1)>> > (dev_c, dev_a, dev_b);
```

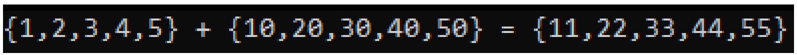


```
addKernel << <dim3(2,3), dim3(2,2)>> > (dev_c, dev_a, dev_b);
```

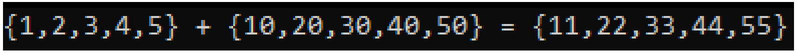


```
addKernel << <dim3(2,2), dim3(2,3)>> > (dev_c, dev_a, dev_b);
```

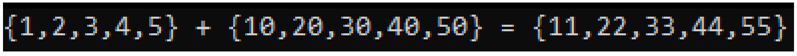
Sample output:



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
```



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
```



```
{1,2,3,4,5} + {10,20,30,40,50} = {11,22,33,44,55}
```

Reflection:

The thread index for multiple 2D blocks is equal to:

(threadIdx.x + blockIdx.x * blockDim.x) + (threadIdx.y + blockIdx.y * blockDim.y)

Metadata:

Multiple 2D blocks

Further information:

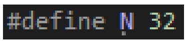
While I have successfully done the 2D block section of the lab I don't feel as though I thoroughly understand it yet. Will ask if it can be explained to me again next lab

Q6. Matrix addition

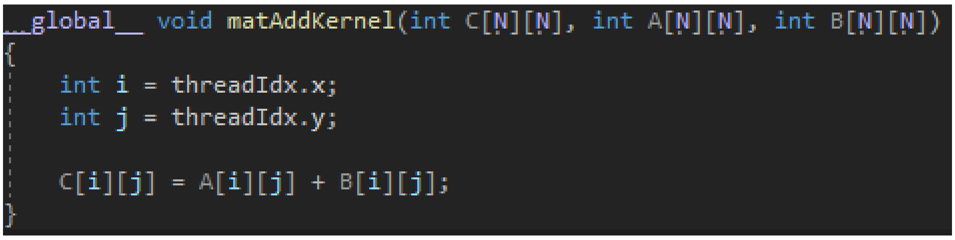
Question:

Write a CUDA program to find the addition of two matrices

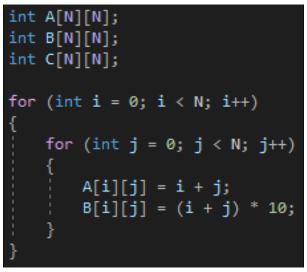
Solution:



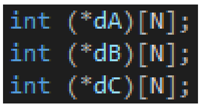
```
#define N 32
```



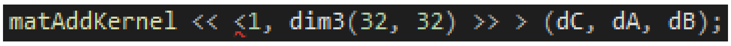
```
__global__ void matAddKernel(int C[N][N], int A[N][N], int B[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;

    C[i][j] = A[i][j] + B[i][j];
}
```



```
int A[N][N];
int B[N][N];
int C[N][N];

for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j++)
    {
        A[i][j] = i + j;
        B[i][j] = (i + j) * 10;
    }
}
```



```
int (*dA)[N];
int (*dB)[N];
int (*dC)[N];
```



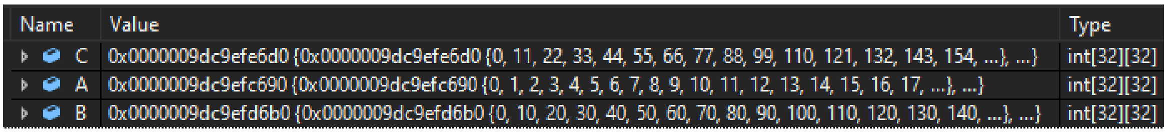
```
matAddKernel << <1, dim3(32, 32) >> > (dC, dA, dB);
```

*I've just included key parts that I think are important

Test data:

N/A

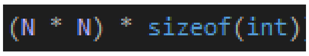
Sample output:



Name	Value	Type
C	0x0000009dc9efe6d0 {0x0000009dc9efe6d0 {0, 11, 22, 33, 44, 55, 66, 77, 88, 99, 110, 121, 132, 143, 154, ...}, ...}	int[32][32]
A	0x0000009dc9efc690 {0x0000009dc9efc690 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ...}, ...}	int[32][32]
B	0x0000009dc9efd6b0 {0x0000009dc9efd6b0 {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, ...}, ...}	int[32][32]

Reflection:

For the cudaMalloc, cudaMemcpy etc the size is as follows:



```
(N * N) * sizeof(int)
```

This task was very confusing I don't really understand the dA,dB and dC variables and stuff like that

Metadata:

Matrix addition

Further information:

I think this task didn’t have enough guidance on the actual lab sheet, I had to google how to do it eventually as I ran out of time in the lab and even after rewatching the lectures this week multiple times I still couldn’t understand what I was doing wrong

Week 3 – Lab 3

Date: 16/02/2022

Q1. Vector dot-product

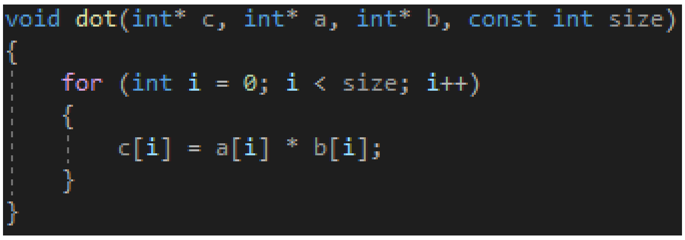
Question:

Part 1 – Write a C++ program to calculate the dot-product of two vectors

Part 2 – Write a C++ program to calculate the dot-product of two vectors on the GPU & calculate the sum of the dot-product of the two vectors on the CPU

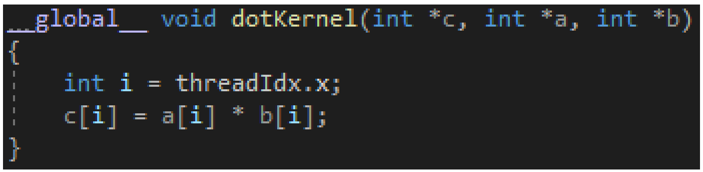
Solution:

Part 1 –

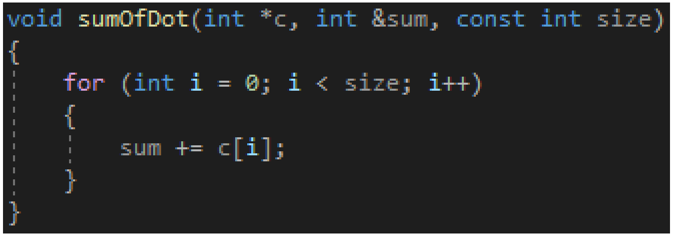


```
void dot(int* c, int* a, int* b, const int size)
{
    for (int i = 0; i < size; i++)
    {
        c[i] = a[i] * b[i];
    }
}
```

Part 2 –



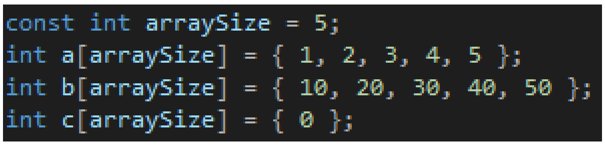
```
__global__ void dotKernel(int *c, int *a, int *b)
{
    int i = threadIdx.x;
    c[i] = a[i] * b[i];
}
```



```
void sumOfDot(int *c, int &sum, const int size)
{
    for (int i = 0; i < size; i++)
    {
        sum += c[i];
    }
}
```

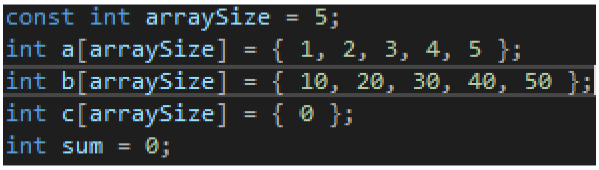
Test data:

Part 1 –



```
const int arraySize = 5;
int a[arraySize] = { 1, 2, 3, 4, 5 };
int b[arraySize] = { 10, 20, 30, 40, 50 };
int c[arraySize] = { 0 };
```

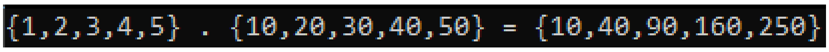
Part 2 –



```
const int arraySize = 5;
int a[arraySize] = { 1, 2, 3, 4, 5 };
int b[arraySize] = { 10, 20, 30, 40, 50 };
int c[arraySize] = { 0 };
int sum = 0;
```

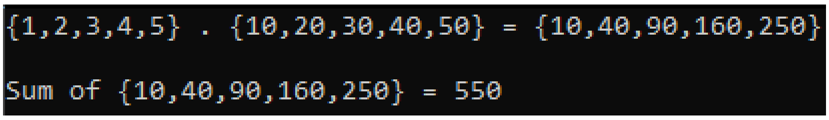
Sample output:

Part 1 –



```
{1,2,3,4,5} . {10,20,30,40,50} = {10,40,90,160,250}
```

Part 2 –



```
{1,2,3,4,5} . {10,20,30,40,50} = {10,40,90,160,250}

Sum of {10,40,90,160,250} = 550
```

Reflection:

This is pretty much the same as lab 1

Metadata:

Dot-product

Further information:

N/A

Q2. Vector dot-product using unified memory

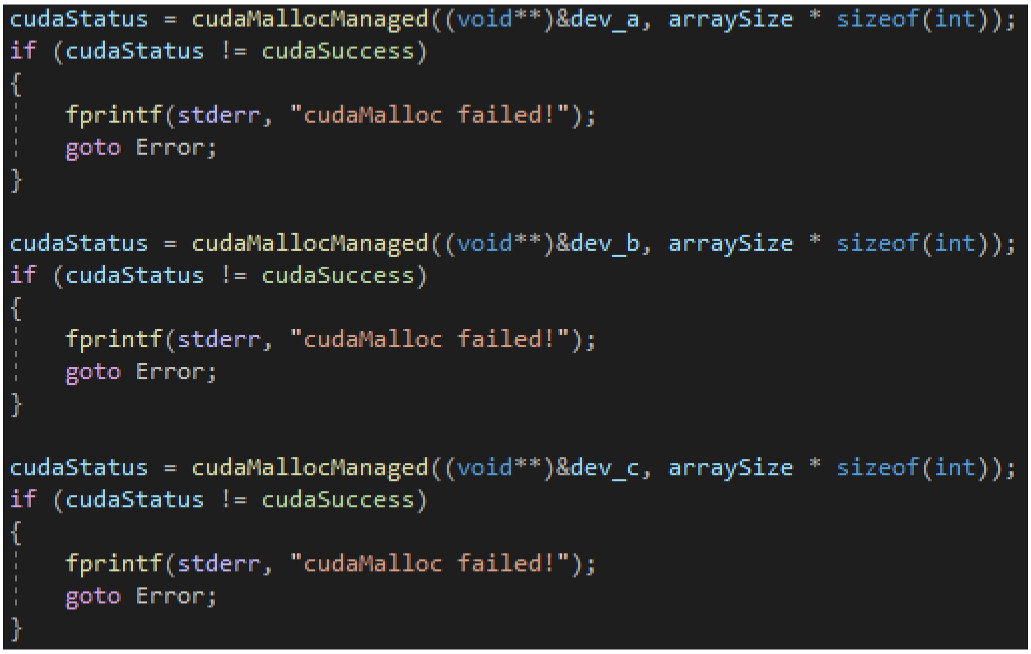
Question:

Part 1 – Create a CUDA program that calculates the vector dot-product using managed memory

Part 2 – Create a CUDA program that calculates the vector dot-product using GPU-declared __managed__ memory

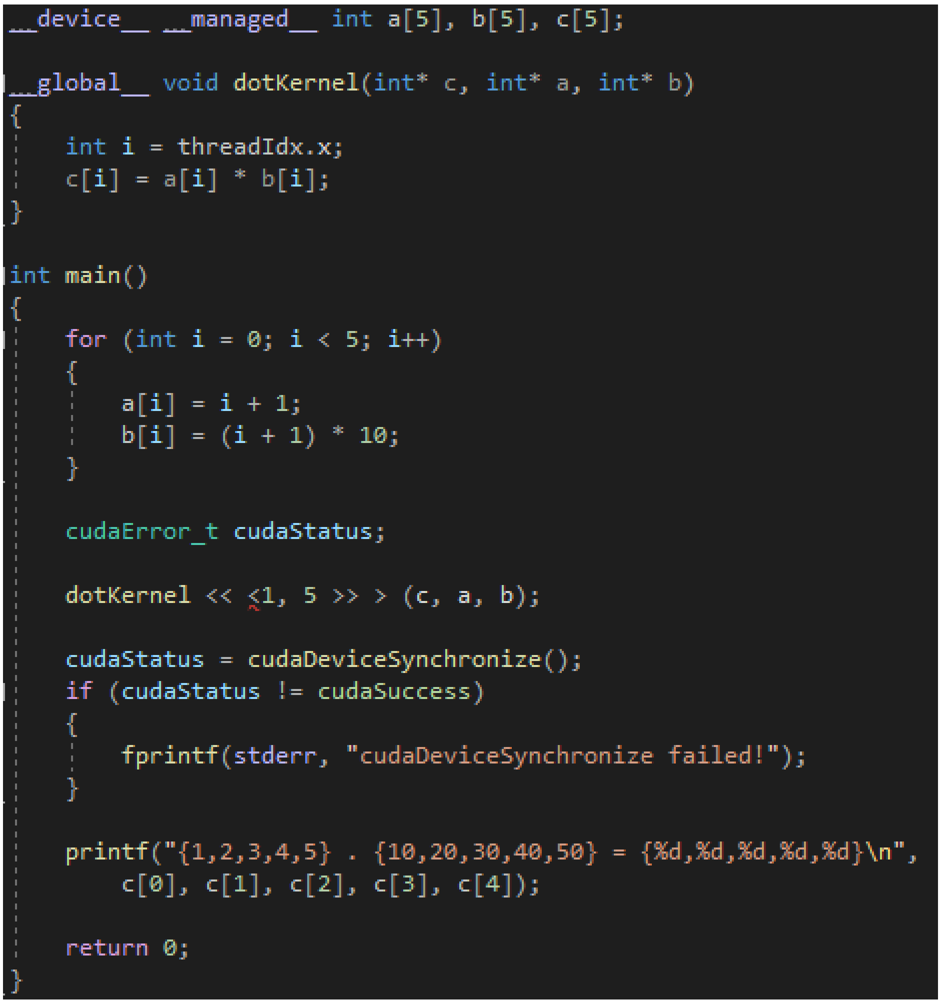
Solution:

Part 1 –



```
cudaStatus = cudaMallocManaged((void**)&dev_a, arraySize * sizeof(int));
if (cudaStatus != cudaSuccess)
{
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}

cudaStatus = cudaMallocManaged((void**)&dev_b, arraySize * sizeof(int));
if (cudaStatus != cudaSuccess)
{
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}

cudaStatus = cudaMallocManaged((void**)&dev_c, arraySize * sizeof(int));
if (cudaStatus != cudaSuccess)
{
    fprintf(stderr, "cudaMalloc failed!");
    goto Error;
}
```

Part 2 –



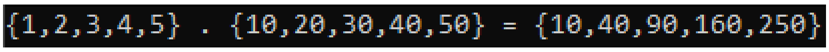
```
__device__ __managed__ int a[5], b[5], c[5];

__global__ void dotKernel(int* c, int* a, int* b)
{
    int i = threadIdx.x;
    c[i] = a[i] * b[i];
}

int main()
{
    for (int i = 0; i < 5; i++)
    {
        a[i] = i + 1;
        b[i] = (i + 1) * 10;
    }

    cudaError_t cudaStatus;

    dotKernel << <1, 5 >> > (c, a, b);

    cudaStatus = cudaDeviceSynchronize();
    if (cudaStatus != cudaSuccess)
    {
        fprintf(stderr, "cudaDeviceSynchronize failed!");
    }

    printf("{1,2,3,4,5} . {10,20,30,40,50} = {%d,%d,%d,%d,%d}\n",
        c[0], c[1], c[2], c[3], c[4]);

    return 0;
}
```

Test data:

N/A

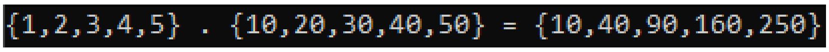
Sample output:

Part 1 –



```
{1,2,3,4,5} . {10,20,30,40,50} = {10,40,90,160,250}
```

Part 2 –



```
{1,2,3,4,5} . {10,20,30,40,50} = {10,40,90,160,250}
```

Reflection:

This way of doing things seems to cut out a lot of code compared to the methods that have been previously used in the labs

Metadata:

Unified memory

Further information:

N/A

Q3. Vector dot-product using shared Memory

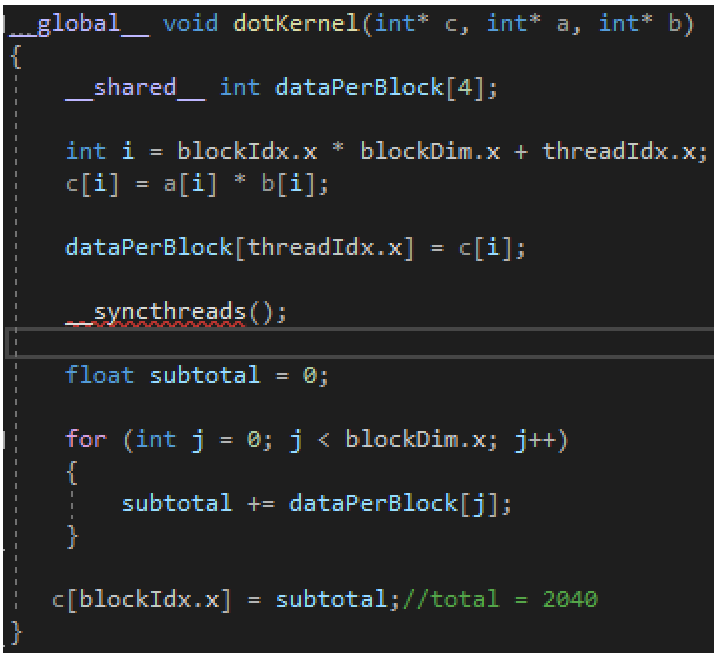
Question:

Part 1 – Analyse the process given in the lab sheet and identify areas where thread execution needs to be synchronized by calling CUDA function: __syncthreads();

Part 2 – Consider different thread configurations, for example, <<>>, <<>>, <<>> and observe if the given program can calculate the vector dot-product correctly. If not, analyse the issues and consider how to fix them.

Solution:

Part 1 –



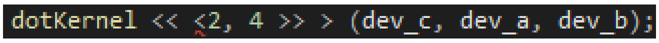
```
__global__ void dotKernel(int* c, int* a, int* b)
{
    __shared__ int dataPerBlock[4];

    int i = blockIdx.x * blockDim.x + threadIdx.x;
    c[i] = a[i] * b[i];

    dataPerBlock[threadIdx.x] = c[i];

    __syncthreads();

    float subtotal = 0;

    for (int j = 0; j < blockDim.x; j++)
    {
        subtotal += dataPerBlock[j];
    }

   c[blockIdx.x] = subtotal;//total = 2040
}
```

Part 2 – N/A

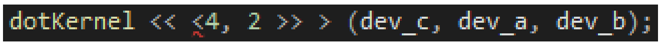
Test data:

Part 1 – N/A

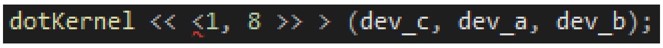
Part 2 –



```
dotKernel << <2, 4 >> > (dev_c, dev_a, dev_b);
```



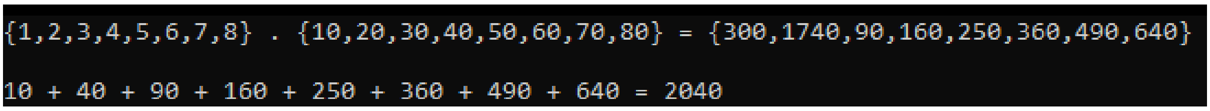
```
dotKernel << <4, 2 >> > (dev_c, dev_a, dev_b);
```



```
dotKernel << <1, 8 >> > (dev_c, dev_a, dev_b);
```

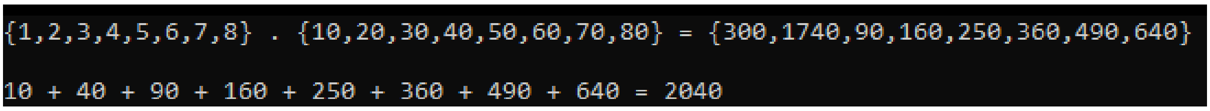
Sample output:

Part 1 –

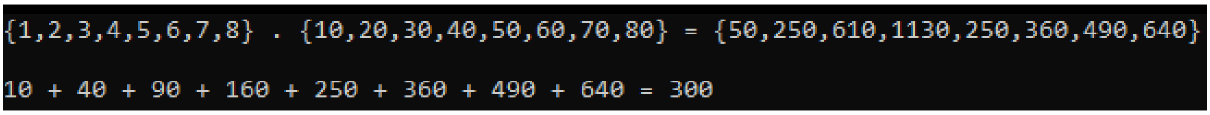


```
{1,2,3,4,5,6,7,8} . {10,20,30,40,50,60,70,80} = {300,1740,90,160,250,360,490,640}

10 + 40 + 90 + 160 + 250 + 360 + 490 + 640 = 2040
```

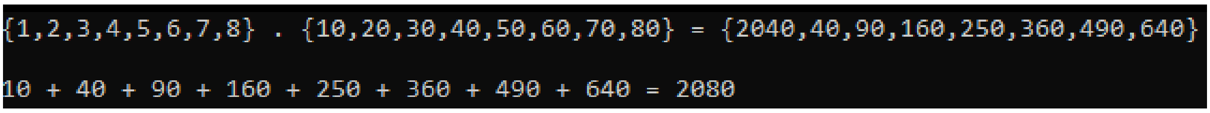
Part 2 –



```
{1,2,3,4,5,6,7,8} . {10,20,30,40,50,60,70,80} = {300,1740,90,160,250,360,490,640}

10 + 40 + 90 + 160 + 250 + 360 + 490 + 640 = 2040
```



```
{1,2,3,4,5,6,7,8} . {10,20,30,40,50,60,70,80} = {50,250,610,1130,250,360,490,640}

10 + 40 + 90 + 160 + 250 + 360 + 490 + 640 = 300
```



```
{1,2,3,4,5,6,7,8} . {10,20,30,40,50,60,70,80} = {2040,40,90,160,250,360,490,640}

10 + 40 + 90 + 160 + 250 + 360 + 490 + 640 = 2080
```

Reflection:

Part 1 – I don't really understand what it is doing I get the same results with and without __syncthreads()

Part 2 – The program only gets the right sum with the 2,4 layout. This could fixed by making the number of blocks and how many values of the c vector added together as the same.

Metadata:

Shared memory

Further information:

N/A

Week 4 – Lab 4

Date: 23/02/2022

Q1. Create an OpenGL-CUDA program based on a CUDA SDK sample

Question:

Set up the project and make sure it compiles

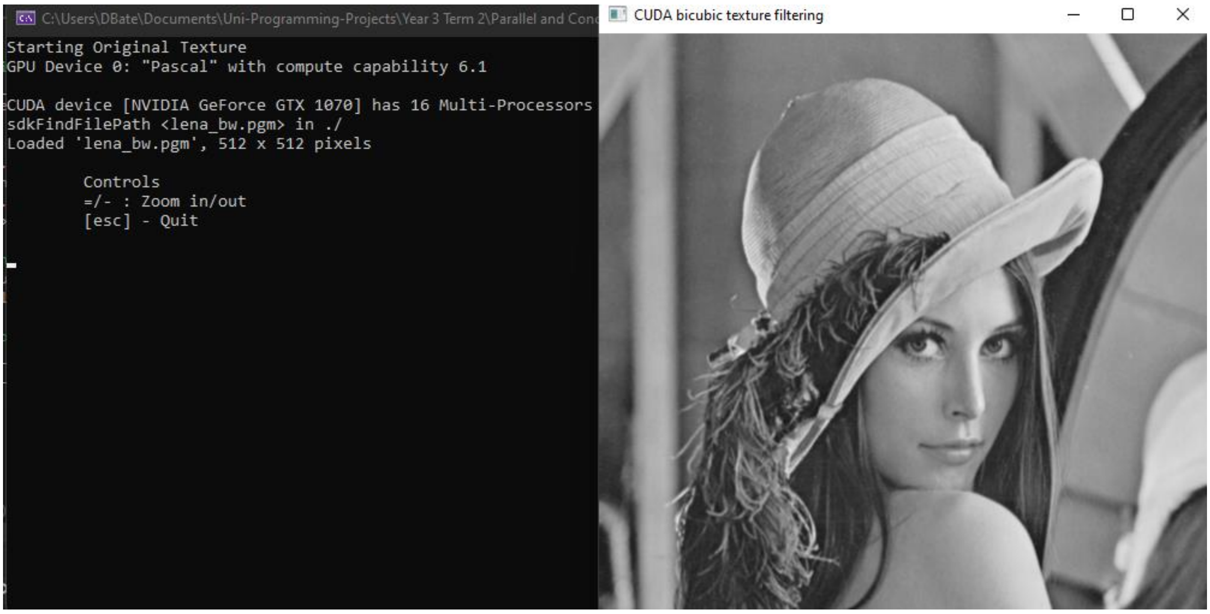
Solution:

N/A

Test data:

N/A

Sample output:



C:\Users\DBate\Documents\Uni-Programming-Projects\Year 3 Term 2\Parallel and Con
CUDA bicubic texture filtering

```
Starting Original Texture
GPU Device 0: "Pascal" with compute capability 6.1

CUDA device [NVIDIA GeForce GTX 1070] has 16 Multi-Processors
sdkFindFilePath <lena_bw.pgm> in ./
Loaded 'lena_bw.pgm', 512 x 512 pixels

        Controls
        =/- : Zoom in/out
        [esc] - Quit
```

Reflection:

The hidden files need to made visible in the file explorer

Metadata:

CUDA SDK

Further information:

N/A

Q2. Understand pixel colour

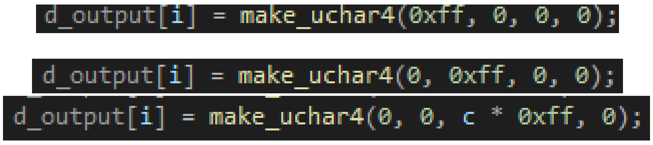
Question:

Edit the line “d_output[i] = make_uchar4(c * 0xff, c * 0xff, c * 0xff, 0);” in various way to understand pixel colouring works

Solution:

N/A

Test data:

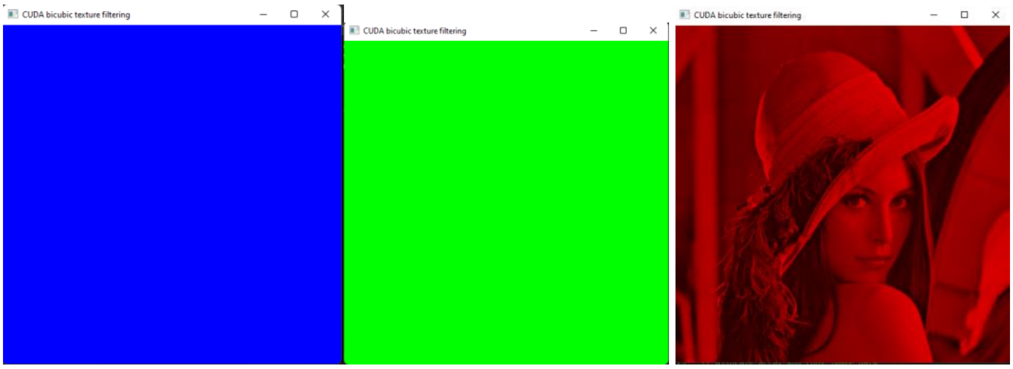


```
d_output[i] = make_uchar4(0xff, 0, 0, 0);
```

```
d_output[i] = make_uchar4(0, 0xff, 0, 0);
```

```
d_output[i] = make_uchar4(0, 0, c * 0xff, 0);
```

Sample output:


CUDA bicubic texture filtering
CUDA bicubic texture filtering
CUDA bicubic texture filtering

Reflection:

For some reason colours are ordered BGR instead of RGB which is what I’m used to

Metadata:

Pixel colours

Further information:

N/A

Q3. Image transformation

Question:

Part 1 – Translate the image

Part 2 – Scale the image

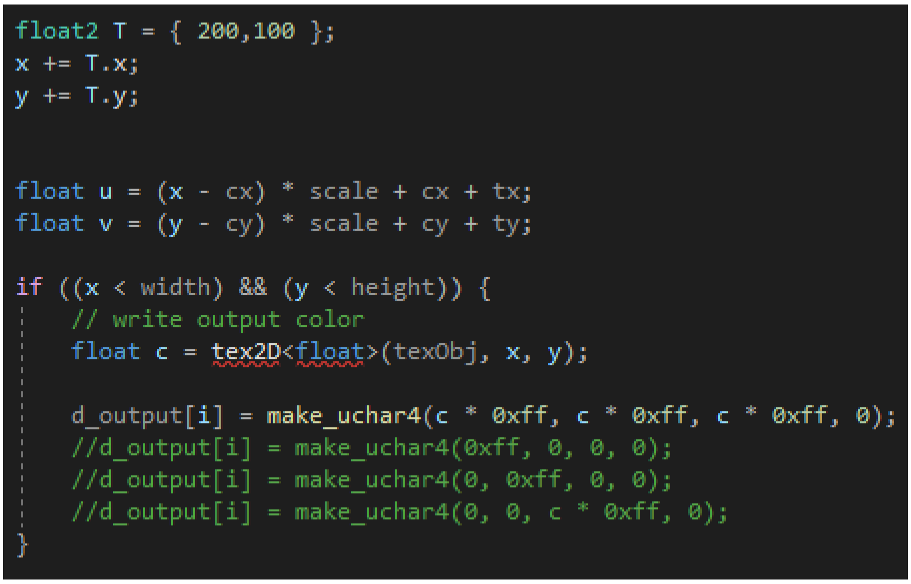
Part 3 – Rotate the image

Part 4 – Scale by position

Part 5 – Rotate by image centre

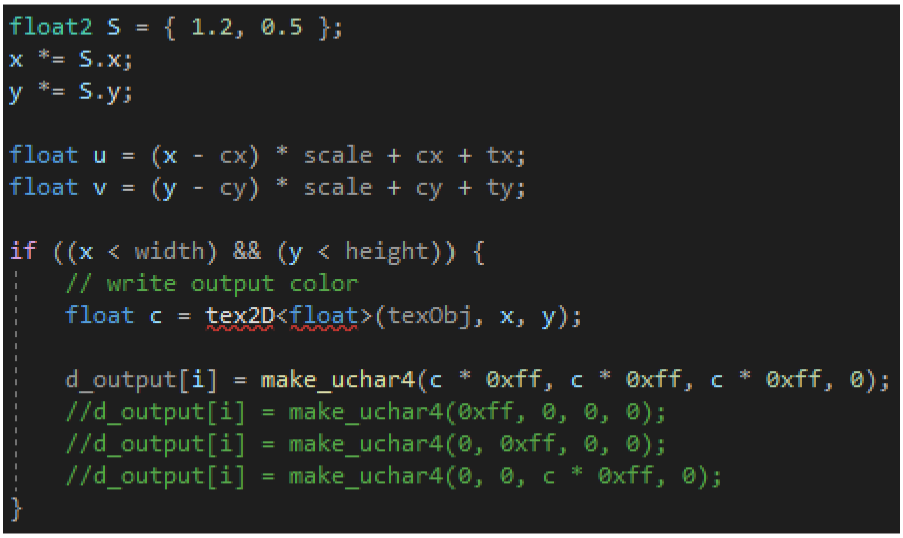
Solution:

Part 1 –



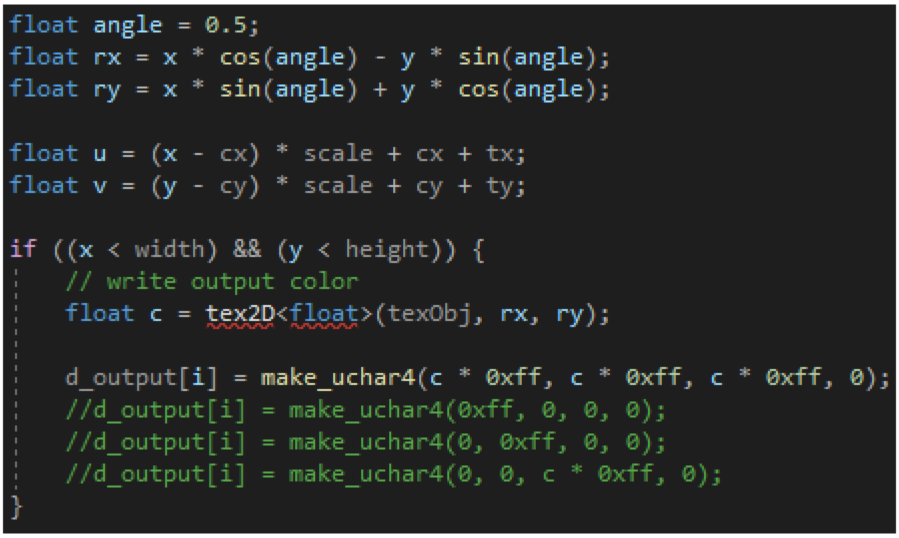
```
float2 T = { 200,100 };
x += T.x;
y += T.y;


float u = (x - cx) * scale + cx + tx;
float v = (y - cy) * scale + cy + ty;

if ((x < width) && (y < height)) {
    // write output color
    float c = tex2D<float>(texObj, x, y);

    d_output[i] = make_uchar4(c * 0xff, c * 0xff, c * 0xff, 0);
    //d_output[i] = make_uchar4(0xff, 0, 0, 0);
    //d_output[i] = make_uchar4(0, 0xff, 0, 0);
    //d_output[i] = make_uchar4(0, 0, c * 0xff, 0);
}
```

Part 2 –



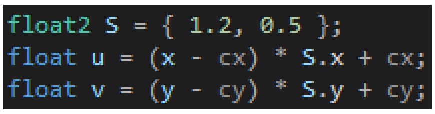
```
float2 S = { 1.2, 0.5 };
x *= S.x;
y *= S.y;

float u = (x - cx) * scale + cx + tx;
float v = (y - cy) * scale + cy + ty;

if ((x < width) && (y < height)) {
    // write output color
    float c = tex2D<float>(texObj, x, y);

    d_output[i] = make_uchar4(c * 0xff, c * 0xff, c * 0xff, 0);
    //d_output[i] = make_uchar4(0xff, 0, 0, 0);
    //d_output[i] = make_uchar4(0, 0xff, 0, 0);
    //d_output[i] = make_uchar4(0, 0, c * 0xff, 0);
}
```

Part 3 –



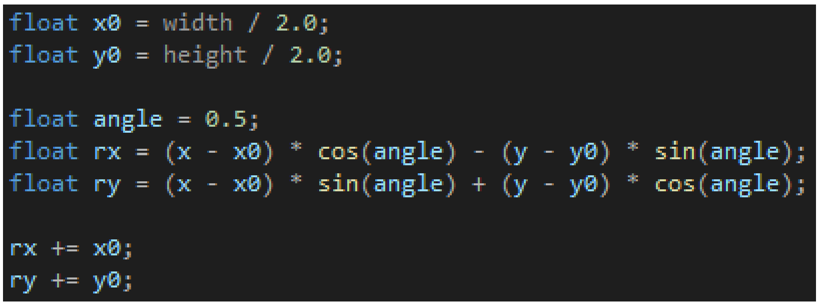
```
float angle = 0.5;
float rx = x * cos(angle) - y * sin(angle);
float ry = x * sin(angle) + y * cos(angle);

float u = (x - cx) * scale + cx + tx;
float v = (y - cy) * scale + cy + ty;

if ((x < width) && (y < height)) {
    // write output color
    float c = tex2D<float>(texObj, rx, ry);

    d_output[i] = make_uchar4(c * 0xff, c * 0xff, c * 0xff, 0);
    //d_output[i] = make_uchar4(0xff, 0, 0, 0);
    //d_output[i] = make_uchar4(0, 0xff, 0, 0);
    //d_output[i] = make_uchar4(0, 0, c * 0xff, 0);
}
```

Part 4 –



```
float2 S = { 1.2, 0.5 };
float u = (x - cx) * S.x + cx;
float v = (y - cy) * S.y + cy;
```

Part 5 –



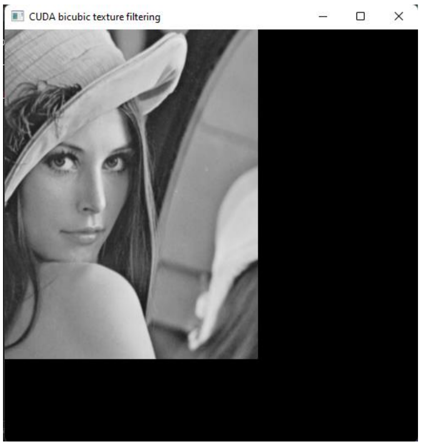
```
float x0 = width / 2.0;
float y0 = height / 2.0;

float angle = 0.5;
float rx = (x - x0) * cos(angle) - (y - y0) * sin(angle);
float ry = (x - x0) * sin(angle) + (y - y0) * cos(angle);

rx += x0;
ry += y0;
```

Test data:

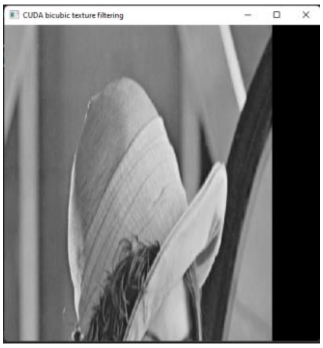
All parts – N/A

Sample output:

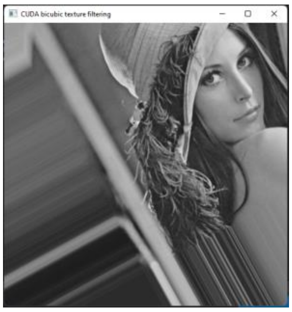
Part 1 –



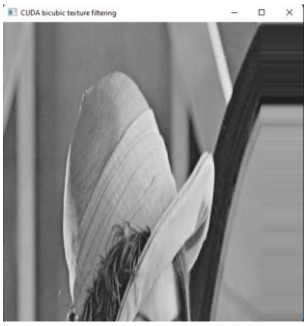
Part 2 –



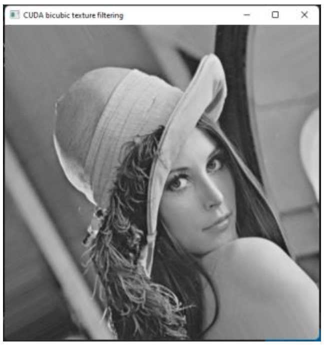
Part 3 –



Part 4 –



Part 5 –



Reflection:

These were all pretty straight forward apart for the rotating by image centre

Metadata:

Transformations

Further information:

N/A

Q4. Image smoothing

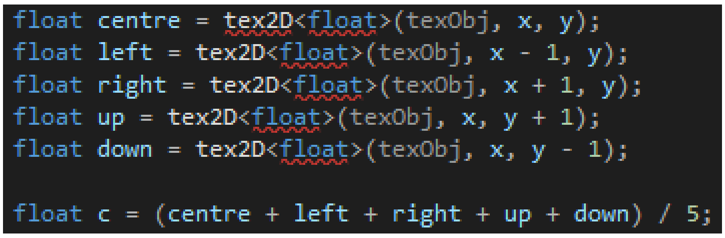
Question:

Part 1 – Implement image smoothing

Part 2 – Modify thread configurations and observe performance

Solution:

Part 1 –



```
float centre = tex2D<float>(texObj, x, y);
float left = tex2D<float>(texObj, x - 1, y);
float right = tex2D<float>(texObj, x + 1, y);
float up = tex2D<float>(texObj, x, y + 1);
float down = tex2D<float>(texObj, x, y - 1);

float c = (centre + left + right + up + down) / 5;
```

Test data:

Part 1 – N/A

Sample output:

Part 1 –

Part 2 –

blocksize	Time
16x16	0.365568n
32x32	0.437248n
24x24	0.390144n

Reflection:

Part 1 – This was pretty straight forward

Part 2 – The threads failed to execute when I tried 64x64 configuration

Metadata:

Smooth

Further information:

N/A

Week 5 – Lab 5

Date: 02/03/2022

Q1. A simple matrix multiplication program in CUDA using one thread block

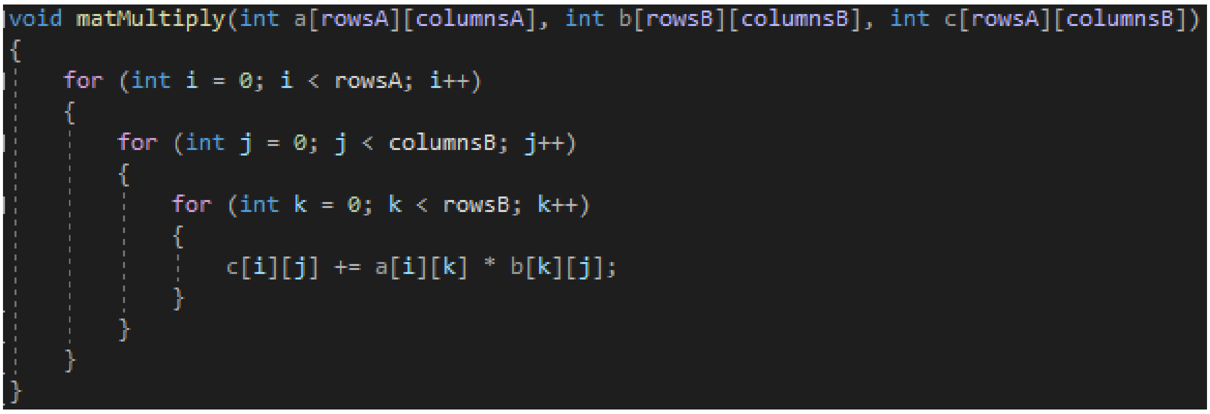
Question:

Part 1 – Write a matrix multiplication function in c++

Part 2 – Write a matrix multiplication function in CUDA using 1 thread block

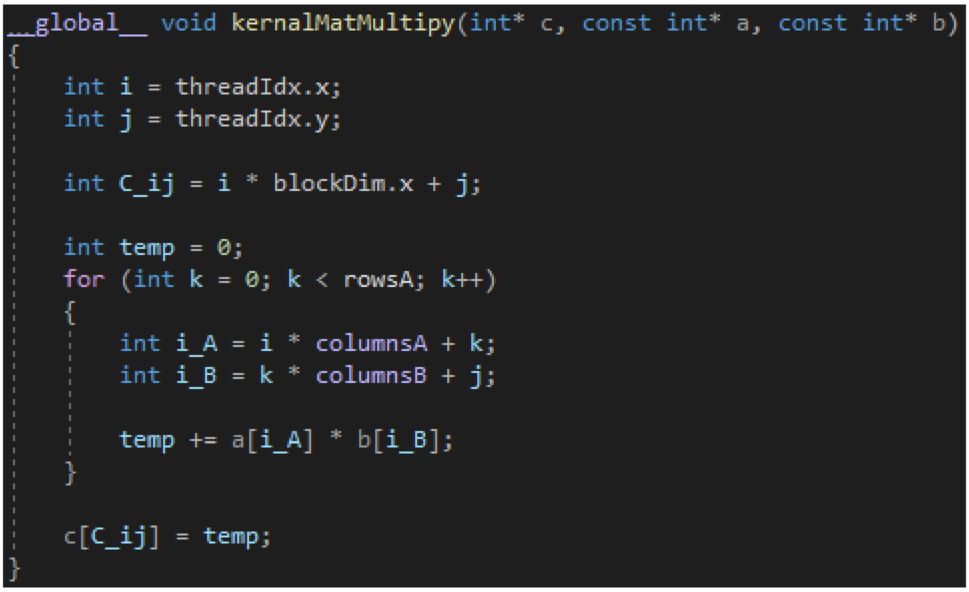
Solution:

Part 1 –



```
void matMultiply(int a[rowsA][columnsA], int b[rowsB][columnsB], int c[rowsA][columnsB])
{
    for (int i = 0; i < rowsA; i++)
    {
        for (int j = 0; j < columnsB; j++)
        {
            for (int k = 0; k < rowsB; k++)
            {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```

Part 2 –



```
__global__ void kernalMatMultipy(int* c, const int* a, const int* b)
{
    int i = threadIdx.x;
    int j = threadIdx.y;

    int C_ij = i * blockDim.x + j;

    int temp = 0;
    for (int k = 0; k < rowsA; k++)
    {
        int i_A = i * columnsA + k;
        int i_B = k * columnsB + j;

        temp += a[i_A] * b[i_B];
    }

    c[C_ij] = temp;
}
```

Test data:

A = (1,2,3,4,5,6,7,8,9) B = (10,11,12,13,14,15,16,17,18)

Sample output:

C = (84,90,96,201,216,231,318,342,366)

Reflection:

I don't really understand why you wouldn't use 2D arrays

Metadata:

Matrix Multiply

Further information:

N/A

Q2. Compare the performance of the CUDA solution against the CPU solution

Question:

Part 1 – 8x8

Part 2 – 32x32

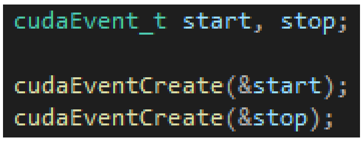
Part 3 – 256x256

Part 4 – 512x512

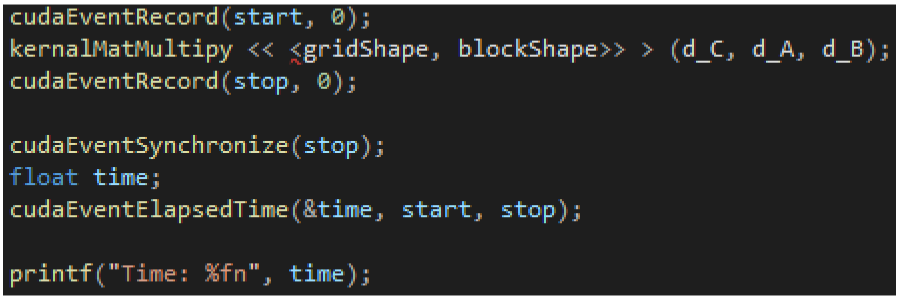
Part 5 – 1024x1024

Solution:

All Parts –



```
cudaEvent_t start, stop;

cudaEventCreate(&start);
cudaEventCreate(&stop);
```



```
cudaEventRecord(start, 0);
kernalMatMultipy << <gridShape, blockShape>> > (d_C, d_A, d_B);
cudaEventRecord(stop, 0);

cudaEventSynchronize(stop);
float time;
cudaEventElapsedTime(&time, start, stop);

printf("Time: %fn", time);
```

Test data:

N/A

Sample output:

Part	Time
1 – CPU	0.007168n
1 – GPU	0.036864n
2 – CPU	0.006144n
2 – GPU	0.059392n
3 – CPU	0.004192n
3 – GPU	0.008224n
4 – CPU	Stack overflow
4 – GPU	Stack overflow
5 – CPU	Stack overflow
5 – GPU	Stack overflow

Reflection:

My timings are really weird and inconsistent but I have no clue why. I had similar issues with a rust lab session.

Metadata:

Performance

Further information:

N/A

Week 6 – Lab 6

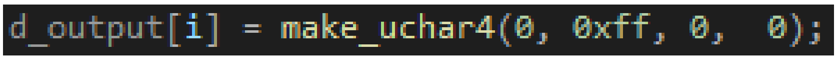
Date: 09/03/2022

Q1. Set up a virtual canvas and draw on it an image in CUDA

Question:

Set up the project then make the image green

Solution:

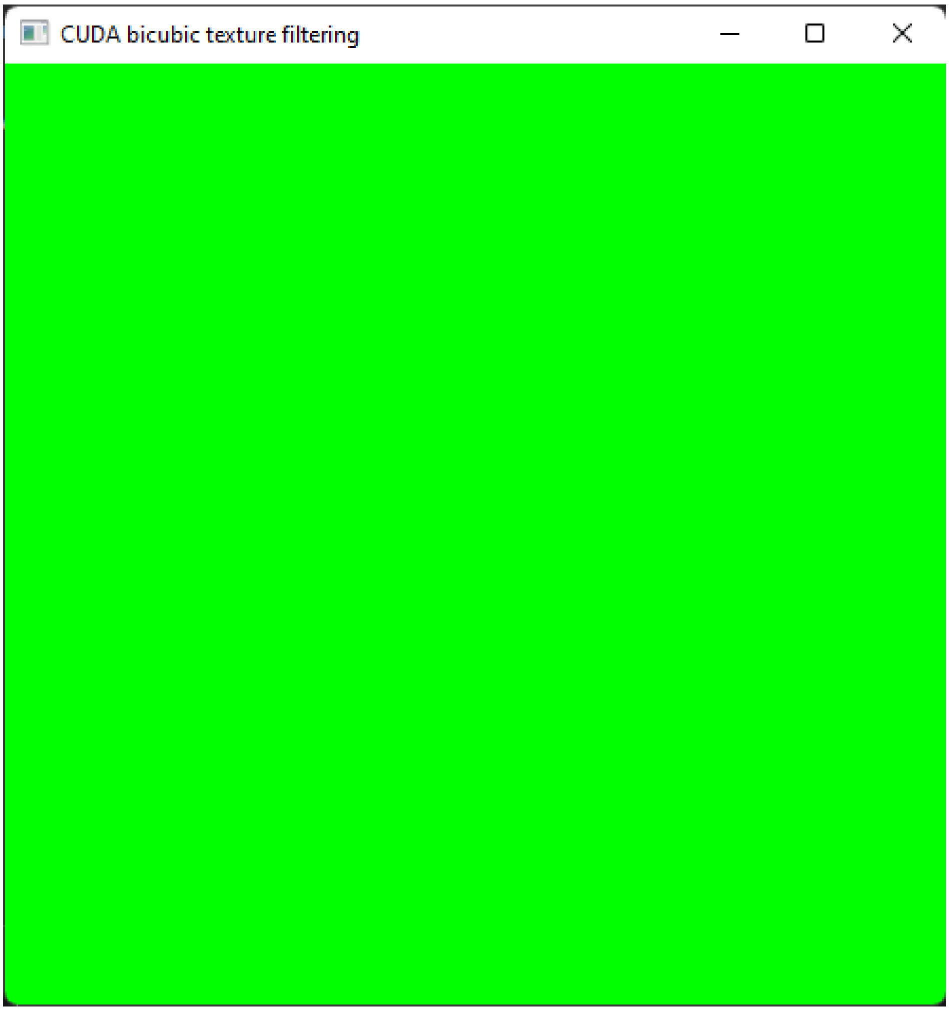


```
d_output[i] = make_uchar4(0, 0xff, 0,  0);
```

Test data:

N/A

Sample output:



Reflection:

This was really straight forward

Metadata:

Set up

Further information:

N/A

Q2. Drawing a checkboard in CUDA

Question:

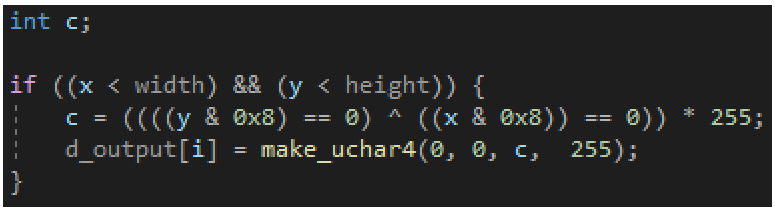
Part 1 – Modify the code to produce a checkerboard image

Part 2 – Modify the code to produce larger red-blocks in the checkerboard

Part 3 – Modify the code to produce a red disc

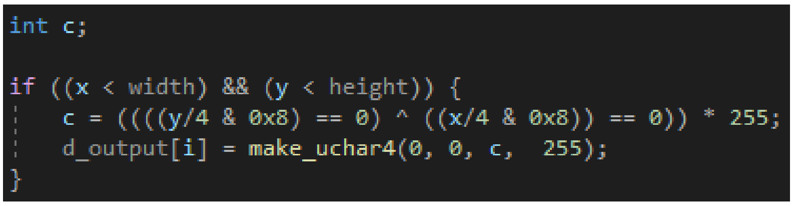
Solution:

Part 1 –



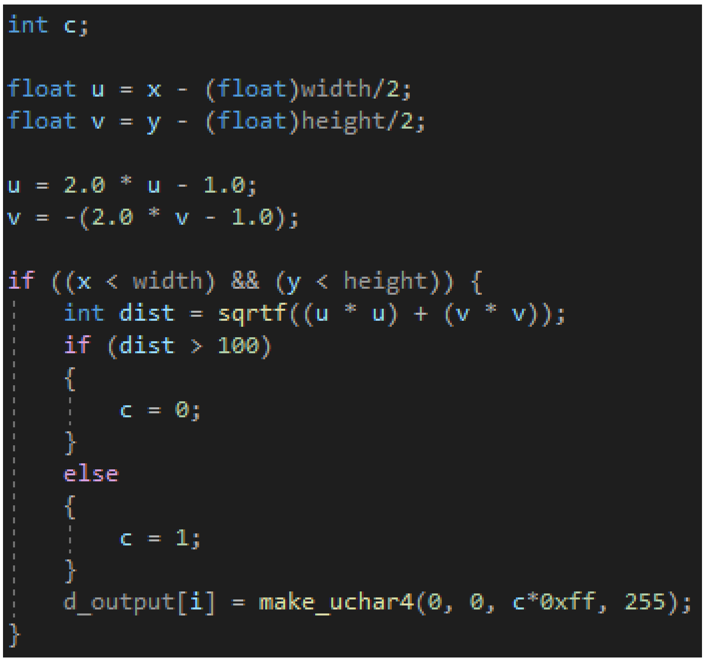
```
int c;

if ((x < width) && (y < height)) {
    c = ((((y & 0x8) == 0) ^ ((x & 0x8)) == 0)) * 255;
    d_output[i] = make_uchar4(0, 0, c,  255);
}
```

Part 2 –



```
int c;

if ((x < width) && (y < height)) {
    c = ((((y/4 & 0x8) == 0) ^ ((x/4 & 0x8)) == 0)) * 255;
    d_output[i] = make_uchar4(0, 0, c,  255);
}
```

Part 3 –



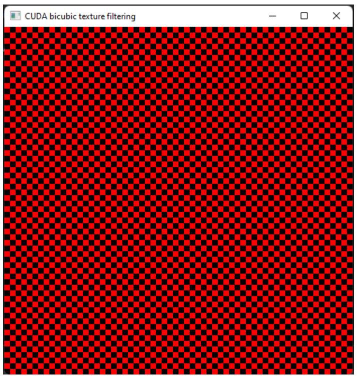
```
int c;

float u = x - (float)width/2;
float v = y - (float)height/2;

u = 2.0 * u - 1.0;
v = -(2.0 * v - 1.0);

if ((x < width) && (y < height)) {
    int dist = sqrtf((u * u) + (v * v));
    if (dist > 100)
    {
        c = 0;
    }
    else
    {
        c = 1;
    }
    d_output[i] = make_uchar4(0, 0, c*0xff, 255);
}
```

Test data:

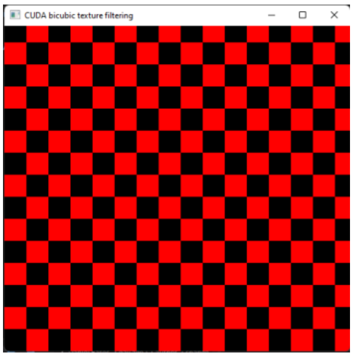
All Parts – N/A

Sample output:

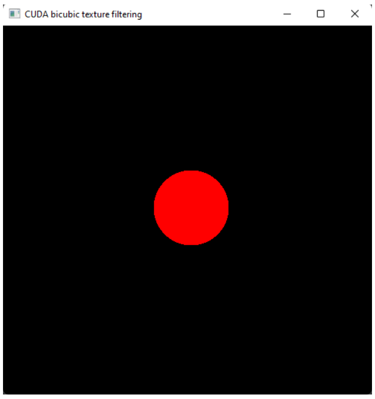
Part 1 –



Part 2 –



Part 3 –



Reflection:

The circle drawing took me a while to do but I think I was just over complicating it. Other than that, everything else was very straight forward

Metadata:

Checkers & Discs

Further information:

N/A

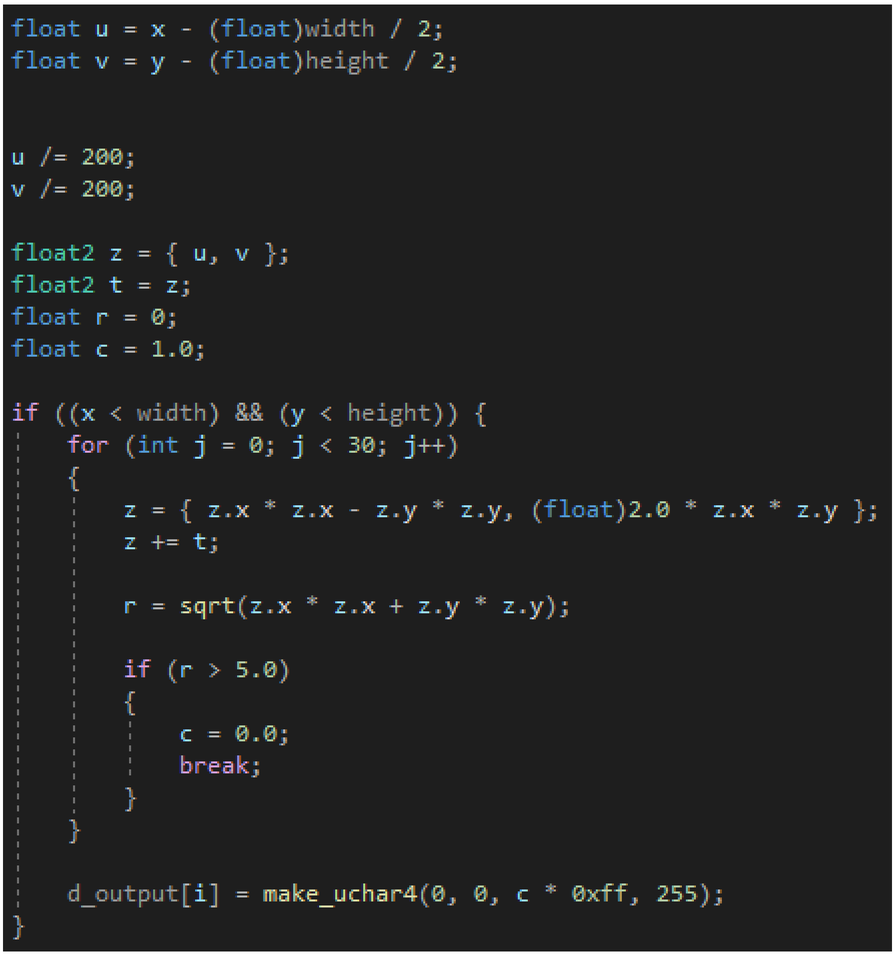
Q3. Drawing the Mandelbrot and Julia Sets

Question:

Implement Mandelbrot and Julia sets

Solution:

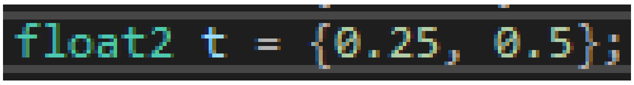
Mandelbrot Set –



```
float u = x - (float)width / 2;
float v = y - (float)height / 2;


u /= 200;
v /= 200;

float2 z = { u, v };
float2 t = z;
float r = 0;
float c = 1.0;

if ((x < width) && (y < height)) {
    for (int j = 0; j < 30; j++)
    {
        z = { z.x * z.x - z.y * z.y, (float)2.0 * z.x * z.y };
        z += t;

        r = sqrt(z.x * z.x + z.y * z.y);

        if (r > 5.0)
        {
            c = 0.0;
            break;
        }
    }

    d_output[i] = make_uchar4(0, 0, c * 0xff, 255);
}
```

Julia Set –

Same as Mandelbrot set except the following:



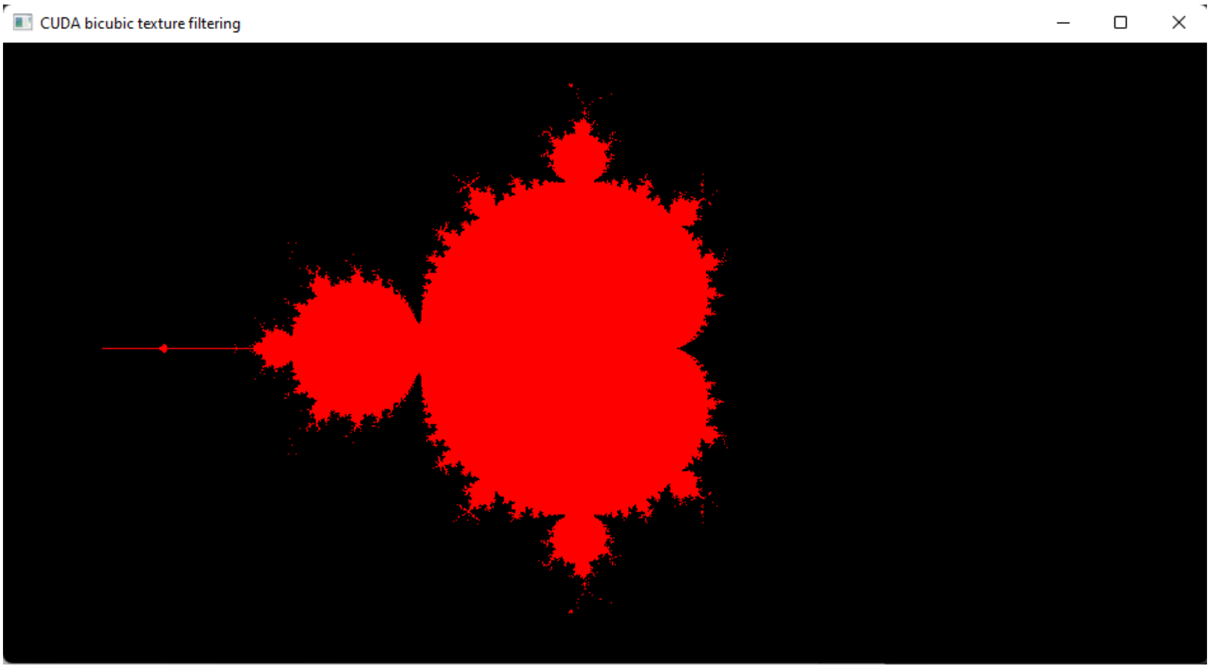
```
float2 t = {0.25, 0.5};
```

Test data:

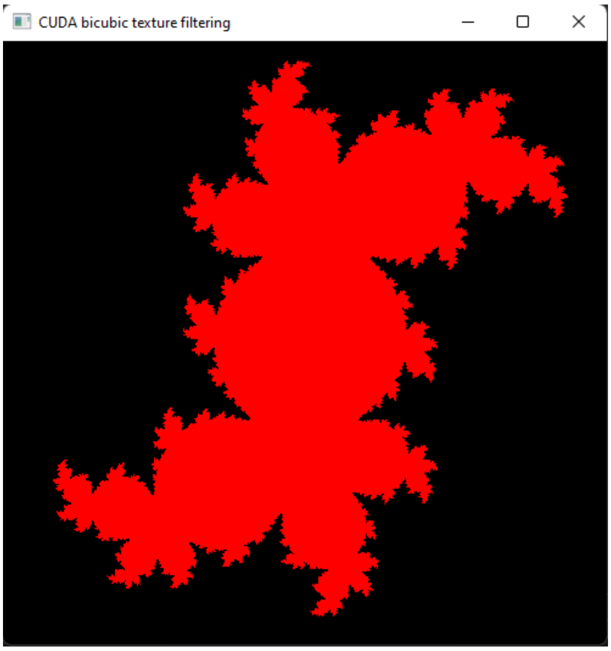
N/A

Sample output:

Mandelbrot Set –



Julia Set –



Reflection:

Fairly straightforward but to scale the image size I have to divide the u and v by 200 instead of multiply by 4.0 and I'm not sure why this was the case.

Metadata:

Mendelbrot & Julia

Further information:

N/A

Week 7 – Lab 7

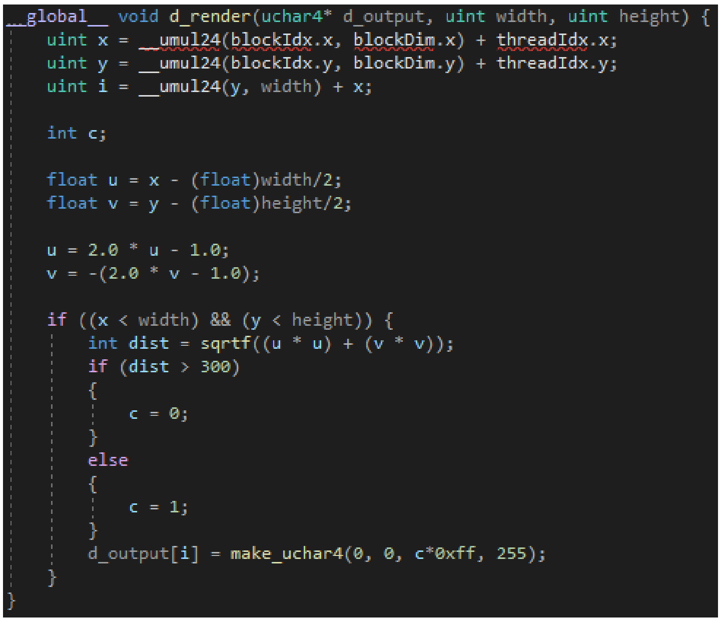
Date: 16/03/2022

Q1. Drawing based on a canvas of size [-1, 1]x[-1, 1]

Question:

Draw a disc based on pixel coordinates defined in float type variables in [-1, 1]x[-1, 1]

Solution:

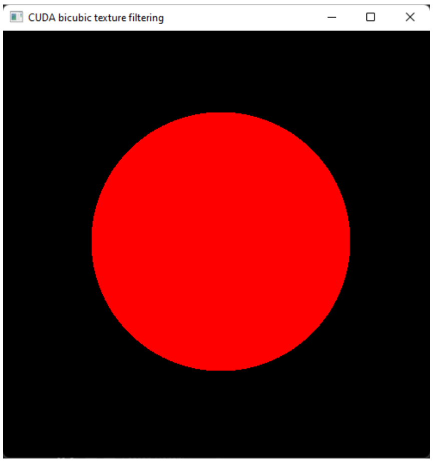


```
__global__ void d_render(uchar4* d_output, uint width, uint height) {
    uint x = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
    uint y = __umul24(blockIdx.y, blockDim.y) + threadIdx.y;
    uint i = __umul24(y, width) + x;

    int c;

    float u = x - (float)width/2;
    float v = y - (float)height/2;

    u = 2.0 * u - 1.0;
    v = -(2.0 * v - 1.0);

    if ((x < width) && (y < height)) {
        int dist = sqrtf((u * u) + (v * v));
        if (dist > 300)
        {
            c = 0;
        }
        else
        {
            c = 1;
        }
        d_output[i] = make_uchar4(0, 0, c*0xff, 255);
    }
}
```

Test data:

N/A

Sample output:



Reflection:

This was just repeating a task of lab 6 so I just copied the code I used for task 4 of lab 6

Metadata:

Drawing Disc

Further information:

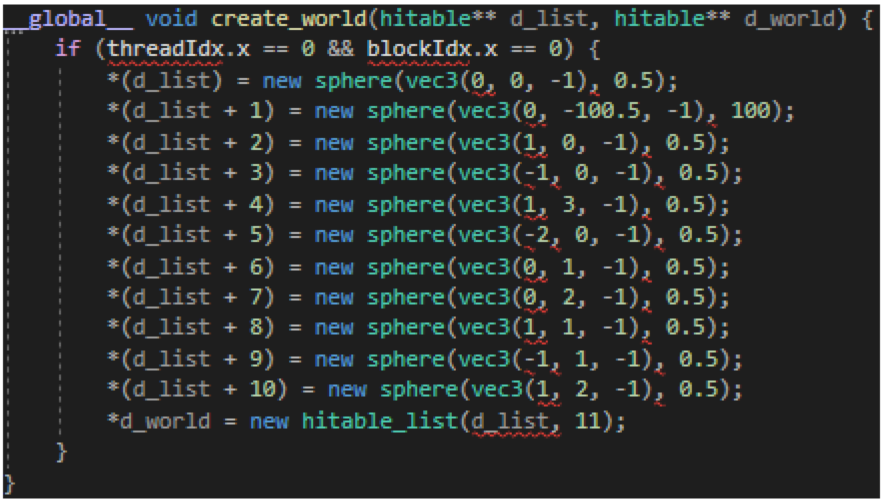
N/A

Q2. Write a simple ray caster

Question:

Write a simple ray caster by implementing the code given in the lab sheet. Then add ten new spheres to the world

Solution:

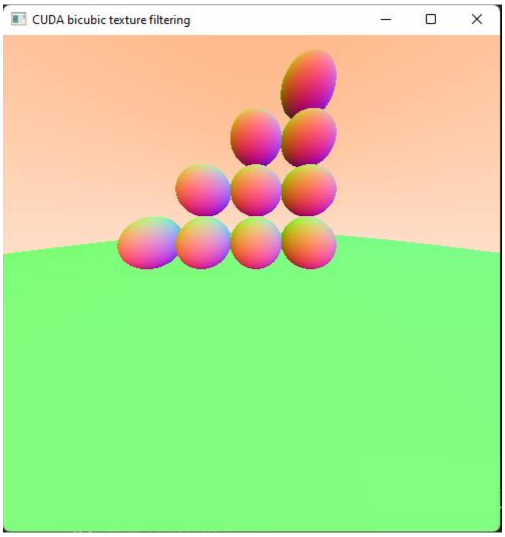


```
__global__ void create_world(hitable** d_list, hitable** d_world) {
    if (threadIdx.x == 0 && blockIdx.x == 0) {
        *(d_list) = new sphere(vec3(0, 0, -1), 0.5);
        *(d_list + 1) = new sphere(vec3(0, -100.5, -1), 100);
        *(d_list + 2) = new sphere(vec3(1, 0, -1), 0.5);
        *(d_list + 3) = new sphere(vec3(-1, 0, -1), 0.5);
        *(d_list + 4) = new sphere(vec3(1, 3, -1), 0.5);
        *(d_list + 5) = new sphere(vec3(-2, 0, -1), 0.5);
        *(d_list + 6) = new sphere(vec3(0, 1, -1), 0.5);
        *(d_list + 7) = new sphere(vec3(0, 2, -1), 0.5);
        *(d_list + 8) = new sphere(vec3(1, 1, -1), 0.5);
        *(d_list + 9) = new sphere(vec3(-1, 1, -1), 0.5);
        *(d_list + 10) = new sphere(vec3(1, 2, -1), 0.5);
        *d_world = new hitable_list(d_list, 11);
    }
}
```

Test data:

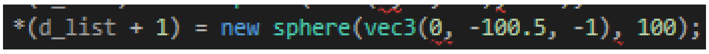
N/A

Sample output:



Reflection:

I'm not sure what the following line is doing:



```
*(d_list + 1) = new sphere(vec3(0, -100.5, -1), 100);
```

I think its serving as a skybox but I'm not sure.

Metadata:

Spheres

Further information:

N/A

Week 8 – Lab 8

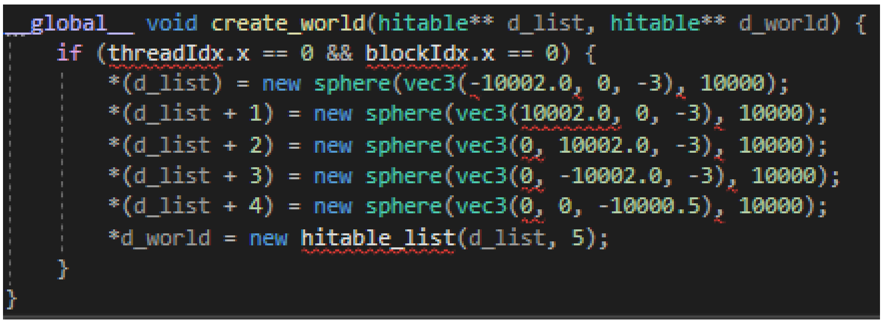
Date: 23/03/2022

Q1. Draw a box without front wall

Question:

Using the spheres create a box without a front wall in the create world method

Solution:

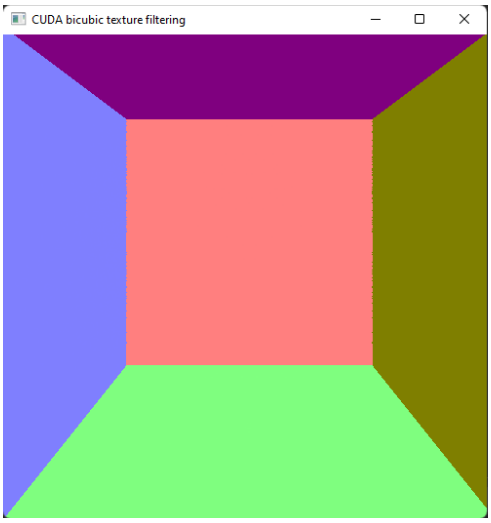


```
__global__ void create_world(hitable** d_list, hitable** d_world) {
    if (threadIdx.x == 0 && blockIdx.x == 0) {
        *(d_list) = new sphere(vec3(-10002.0, 0, -3), 10000);
        *(d_list + 1) = new sphere(vec3(10002.0, 0, -3), 10000);
        *(d_list + 2) = new sphere(vec3(0, 10002.0, -3), 10000);
        *(d_list + 3) = new sphere(vec3(0, -10002.0, -3), 10000);
        *(d_list + 4) = new sphere(vec3(0, 0, -10000.5), 10000);
        *d_world = new hitable_list(d_list, 5);
    }
}
```

Test data:

N/A

Sample output:



Reflection:

This was very simple to implement. I’m not sure how to implement the other way the lab sheet mentions though.

Metadata:

Box

Further information:

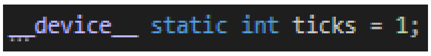
N/A

Q2. Free motion animation

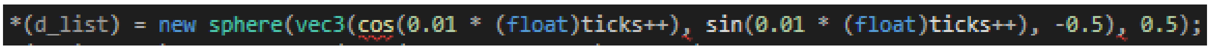
Question:

Draw a ball rotating by the centre of the box

Solution:



```
__device__ static int ticks = 1;
```

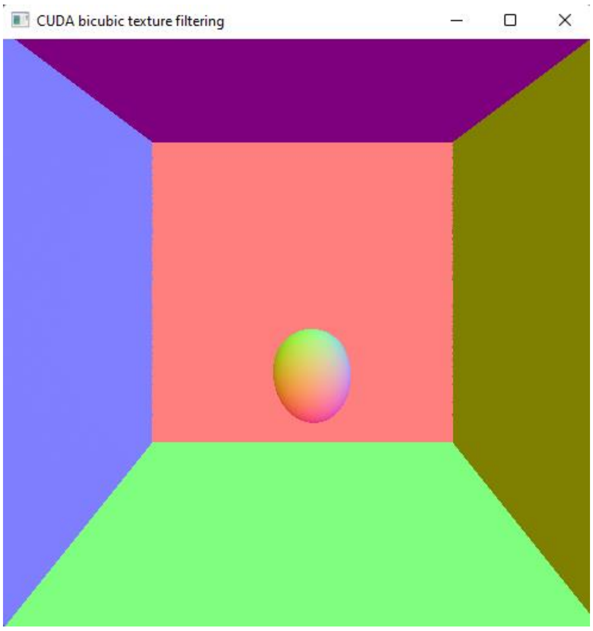


```
*(d_list) = new sphere(vec3(cos(0.01 * (float)ticks++), sin(0.01 * (float)ticks++), -0.5), 0.5);
```

Test data:

N/A

Sample output:



Reflection:

The rotating sphere becomes distorted into and egg shape but I'm not sure why this is

Metadata:

Rotation

Further information:

N/A

Q3. Ball-box walls collision animation

Question:

Animate a ball in motion with an initial velocity that is reflected when the ball collides with a wall. When a collision takes place the ball should also change colour

Solution:

Test data:

Sample output:

Reflection:

I can’t get it to work properly at all and the way it was explained to me to do in the lab sessions causes multiple errors which means the code won’t compile. I’m not sure if this is because I’m on version 11.6 instead of 11.5

Metadata:

Further information:

Final Lab

GPU Design

The plan for the GPU implementations was use 1 block with 1 thread for each particle in the particle system for both the dynamics and the temperature. For the collisions I was planning on splitting each collision type, wall-particle collisions, floor-particle collisions and particle-particle collision, into separate blocks with each block having as many threads as there are particles.

CPU Design

The particle dynamics and particles temperature were handled over 2 threads each in a 2 separate scoped thread pools, I chose to use the scoped thread pools so I could utilise chunking to split the work between threads and so the ownership of the particle system would be less of an issue. In the particle dynamics thread pool, before the particle system is chunked and the threads run, 5 new particles are created and added to the particle system. This keeps a continues stream of particles being produced.

The collisions were handled slightly differently. Since I didn't want to chunk the particle system, so I could check to see if any 1 particle is colliding with any other, and I wanted each collision being checked to be done in its own thread. I used 3 threads, 1 for wall-particle collisions, 1 for floor-particle collisions and 1 for particle-particle collision. This method requires me to use an Arc and Mutex for the particle system to avoid any ownership issues.

However, by using these 2 different methods there are issues that happen with glium since the particle system gets moved into closure by putting it in an Arc so the dynamics and temperature threads can't access the particle system to chunk and use it. So, the dynamics and temperature threads had to be changed to use the Arc and Mutex for the particle system.

Performance comparison

I left my GPU implementation too late so unfortunately I don't have enough to do a comparison between implementations.

Reflection

In hindsight I think I spent too much time focusing on the visualisation aspects, since I didn't really understand it, when I should have focused on getting the functions and threading done first then added visualisation later on. I also spent too much time on just the CPU implementation as a whole and should have left more time to work on the GPU implementation, especially as that is the side that my knowledge and understanding of is weaker. I also think that the way I approached the tasks was wrong, instead of doing a rough design for everything and then trying to get it all to work together, I should have done one aspect at a time, so focus on dynamics then temperature etc.