Exam A Thought Protocol

1. 1. Write a procedure that sorts a sequence of natural numbers that are -1 terminated. The program is supposed to do the following steps.
      (a) receive a value and store it.
      (b) reveive another value and store it.
      (c) if the second receivedvalue is -1, then send out the first value, followed by -1. Afterwards, return back to step (a).
      (d) else send out the smaller value and keep the other one.
      (e) go back to step (b)

```
1  void sortr(intchan in, intchan out) {
2
3
4
5
6
7
8
9
10     }
```

   2. Write a mainAgent pipeline. Use 4 agents, then send out the sequence 6,1,8,3. Then the pipeline should return the result to the user and terminate. The other agents are allowerd to live forever.

2. Below is an implementation of a semaphore.

```
struct semaphore{
    int tickets = 0; //may never be negative!
    void up(){

    tickets++;

    }

    void down(){
        while (tickets<1) {
            //do nothing, just wait
        }

        tickets--;

    }
}

```

1. (a) The above implementation is faulty regarding concurrency in two ways. State where the problems are and why they are problematic. You can use the line numbers to refer to the code.

   (b) Correct the problems of the code. You may modify the above code by adding statements, removing statements or modifying statements (strikeout old statements if necessary).

2. We want to implement a barrier: Users can enter the barrier, which blocks until the limit of the barrier is reached. For now we set the amount of spaces to 2. Make sure that nobody can enter the barrier until the previous "generation" have left.
   You are not allowed to use any message passing here.

```
monitor Barrier{
int spaces = 2, waitingAgents = 0;


void enter() {
    //assure all Agents from the previous generation have left.


    waitingAgents++;



    waitingAgents--;
    //clean up



            }
        }

```

3. (a) The ADTV (Allgemeiner deutscher Tandem Verein) wants to increase their member numbers. For that they host an event where users can enter a boarding area which holds 42 people. If there are more guests, they need to wait. Only 2 people can use a tandem, if there is only one, this person needs to wait. Use the Semaphore and the Barrier from the previous exercises. *don't use any message passing or additional locks*

```
struct BoardingArea {
    Semaphore people;
```

```
3       Barrier Tandemlock;
4
5       void userEnter() {
6
7       }
8
9       void goOnTandem(){
10
11      }
12
13      void userExit(){
14
15      }
16
17      void start() { //only called once for initialization
18
19
20
21      }
22  }
23
```

(b) Why is turning the *struct BoardingArea* into a *monitor BoardingArea* for extra safety not a good idea?

3. Transform the following Pseuco monitor into a Java monitor. Use as few `signal()`, `signalAll()`, `notify()` and `notifyAll()` as possible, and only use `signalAll()` or `notifyAll()` if necessary. The spaces don't indicate how much lines of code you wil actually need. *Only use implict locks*

```
1  monitor SampleMonitor {
2      //...
3      condition left with (toes>shoes);
4      condition right with (raspberry == "nom");
5
6      void walk ()
7      {
8          waitForCondition(right);
9          doSomething();
10         signal(left);
11         doSomethingElse();
12         signalAll(right);
13     }
14 }
15
```

```
1  public        class SampleMonitor       {
2  //...
3
4
5      public           walk()       {
6
7
8
9
10         doSomething();
11
12
13
```

```
14              doSomethingElse();
15
16
17
18          }
19      }
20
```

---

4. 1. What is the importance of fences in memory models?
   2. Consider the following program:

x=13	
2: x=17;	2: r1=x;
	3: r2=x;

   Give a weakly consistent execution where in line 2 we read 17 and in line 3 we read 13.
   3. Can this also happen under sequential consistency?
   4. Can we use fences to prevent this behaviour? If so, use as few fences as possible and explain where you placed them. You can indicate fences with arrows in above code with arrows. If this is not possible, explain why.
   5. Now we want to assume happens-before-consistency. Insert as few Instructions as possible to prevent the behaviour of (2).

5. Answer the following questions. Correct Answers award you 1 point, false answers give 0 points. If you tick "no answer", you will be rewarded with 0.5 points.
   1. A Rust program that compiles does not contain any deadlocks.
      ◯ yes ◯ no ◯ no answer
   2. In Rust it is allowed that multiple threads have exclusive references to the same data as long as they don't access it at the same time.
      ◯ yes ◯ no ◯ no answer
   3. An Arc can be used to share ownership between multiple threads.
      ◯ yes ◯ no ◯ no answer
   4. If a monitor merely manages multiple `int`s, it can be replaced by a class with respectively many `AtomicInteger`s.
      ◯ yes ◯ no ◯ no answer
   5. If a thread calls `wait()` while no other thread is running, the program is guaranteed to deadlock.
      ◯ yes ◯ no ◯ no answer
   6. If a select-case statement has multiple default cases, one is chosen randomly.
      ◯ yes ◯ no ◯ no answer