Exercise 1

Problem 1.a

Safe: any READ that does not overlap a WRITE returns the most recently written value.
Problem 1.a

Regular: any READ that overlaps a WRITE returns the value written by the last preceding WRITE, or any of the values written by overlapping WRITEs.
Problem 1.a

Atomic: READs and WRITEs have a linearization order.

=> NOT atomic!
Problem 1.b

None of the above (not safe).
Problem 1.c

Atomic

Diagram: 
- p1: WRITE(1) → WRITE(2) → WRITE(1)
- p2: READ() = 1
- p3: READ() = 2
Problem 2.a

The transformation works for multi-valued registers and regular registers.

- We use an array of SRSW *registers* \( \text{Reg}[1, \ldots, N] \)
  
  **Read()**
  
  ```
  return (\text{Reg}[i].\text{read}());
  ```

  **Write(v)**
  
  ```
  for j = 1 to N
     \text{Reg}[j].\text{write}(v);
  ```
Problem 2.a

For regular registers:

- We use an array of SRSW **registers** Reg[1,..,N]
- **Read()**
  - return (Reg[i].read());
- **Write(v)**
  - for j = 1 to N
    - Reg[j].write(v);

If Pi.Read is concurrent with some Write
- Either Reg[i].read is concurrent with Reg[i].write
- Or Reg[i].read is not concurrent with Reg[i].write
If concurrent
- Reg[i].read returns as a regular register
If not
- Either Reg[i] has been written by WRITE
- Or not
- ...
The transformation does **NOT** work for atomic registers.

We use an array of SRSW *registers* \( \text{Reg}[1,..,N] \)

**Read()**

```plaintext```
return (Reg[i].read());
```

**Write(v)**

```plaintext```
for j = 1 to N
   Reg[j].write(v);
```

- Use an array of SRSW atomic registers
Problem 2.b

For atomic registers:

We use an array of SRSW **registers** $\text{Reg}[1,..,N]$

- **Read()**
  - return ($\text{Reg}[i].\text{read()}$);

- **Write(v)**
  - for $j = 1$ to $N$
    - $\text{Reg}[j].\text{write}(v)$;

If $\text{Pi.Read}$ is concurrent with some $\text{Write}$
- Either $\text{Reg}[i].\text{read}$ is concurrent with $\text{Reg}[i].\text{write}$
- Or $\text{Reg}[i].\text{read}$ is not concurrent with $\text{Reg}[i].\text{write}$

If concurrent
- $\text{Reg}[i].\text{read}$ returns as a regular register

If not
- Either $\text{Reg}[i]$ has been written by $\text{WRITE}$
- Or not
- ...
Problem 2.b

The transformation does **NOT** work for atomic registers.

- We use an array of SRSW **registers** Reg[1,..,N]
- **Read()**
  - return (Reg[i].read());
- **Write(v)**
  - for j = 1 to N
    - Reg[j].write(v);
Problem 3.a

The transformation does **NOT** work for multi-valued registers.

- We use one MRSW safe register
- **Read()**
  - return(Reg.read());

- **Write(v)**
  - if old ≠ v then
    - Reg.write(v);
    - old := v;

- Use one MRSW safe multi-valued register
- Reg.read may return arbitrarily when being concurrent with Reg.write
Problem 3.b

The resulting register is **NOT** (binary MRSW) atomic.

- **We use one MRSW safe register**
- **Read()**
  - `return(Reg.read());`
- **Write(v)**
  - if old ≠ v then
    - Reg.write(v);
  - old := v;