INGI2131 Practical Exercises
Lab 11: Explicit State and Objects

This set of exercises will help you to understand explicit state manipulation by using cells and class-based objects. We will also see that using ports you can get the same stateful behaviour as using cells. Remember that for the project you are not allowed to use cells or class-based objects. The only non-deterministic language abstraction that you can use is port (The GUI is another story, because it is non-deterministic).

1. Consider the following code fragment. Answer questions (a-d) in the comments **before running the code**.

   ```
   declare
   A={NewCell 0}
   B={NewCell 0}
   T1=@A
   T2=@B
   {Show A==B} % a: What will be printed here
               % true, false, A, B or 0?

   {Show T1==T2} % b: What will be printed here
                 % true, false, A, B or 0?

   {Show T1=T2} % b: What will be printed here
                % true, false, A, B or 0?

   A:=@B
   {Show A==B} % b: What will be printed here
               % true, false, A, B or 0?
   ```

   Now confirm your answers by running this code in the OPI.

2. Implement the cell related procedures `NewCell`, `Access` and `Assign` using ports. `{Access C R}` binds `R` to the contents of cell `C`, like the `@` operator. `Assign C E` updates cell `C` to contain `E`, like the `:=` operator.

   Hint: `NewCell` *could return a port object (with internal state) that receives access and assign messages.* `Access` *and* `Assign` *would be procedures sending the respective message to the port object.*

3. Implement `NewPort` and `Send` with cells. Remember that each port has an associated stream. The messages sent to the port should be placed in the stream in FIFO order. That means that every new message will be added *at the end* of the stream.

4. Implement `NewPortClose`, `Send` and `Close`. This abstraction is exactly like port, but it has an extra procedure. By calling `{Close P}`, the stream becomes a list by

adding a `nil` at the end. After procedure `Close` is used, it is not possible to send more messages to the port.

5. The function `fun {Q A B} ... end` iteratively calculates the sum of the elements between `A` and `B` (`A` and `B` included). You can assume that $A > 0$ and $B > A$. Implement `Q` in such a way that it uses a cell to calculate the sum.

6. Handling state with class-based objects:

   a Implement the function `Q` from the previous question with class-based objects. For this, you can define a class `Counter` with two methods: `add(N)` that adds `N` to the current state of the object, and `read(N)` that binds `N` to the current state of the object.

   b Implement Ports with class-based objects. Define a class `Port` with two methods: `init` and `send`. Then, define `NewPort` and `Send` in terms of this class.

   c Implement PortClose with class-based objects. Define a class `PortClose` which inherits from class `Port`, and add method `close`. Use this class to implement methods `NewPortClose`, `Send` and `Close`.

   Observation: Now you know how to add ports to your favourite class-based object-oriented programming language.

7. **Extra exercise for curious minds** Redo exercise 2, but this time using *IsDet* instead of port.