



Los ejercicios de *Debugging* constituyen el Taller 2. Los ejercicios de *Testing* no se entregan. Para aprobar el taller, todos los tests que les proveemos deben pasar. Recuerden subir únicamente el archivo `Debugging.java` al campus.

Última fecha de entrega: domingo 07/04.

1. Debugging

Los siguientes ejercicios tienen ya un código que los implementa, aunque con algunos errores. Su objetivo es encontrar estos errores y arreglarlos para que pasen los tests.

Ejercicio 1. Debuggear el código que implementa el operador `xor` (que se simboliza con \vee). Este operador representa un “o exclusivo”: $A \vee B$ es verdadero cuando A ó B es verdadero, pero no ambos. Cumple la siguiente tabla de verdad:

A	B	$A \vee B$
false	false	false
false	true	true
true	false	true
true	true	false

Ejercicio 2. Debuggear el código que implementa la función `iguales`, que determina si dos arreglos de número enteros son iguales.

```
proc iguales (in xs: seq<Z>, in ys: seq<Z>) : Bool {  
  requiere {true}  
  asegura {res = true  $\leftrightarrow$  |xs| = |ys|  $\wedge_L$  ( $\forall i : \mathbb{N}$ ) ( $i < |xs| \rightarrow_L xs[i] = ys[i]$ )}
```

Ejercicio 3. Debuggear el código que implementa la función `todosPositivos`, que determina si todos los valores de un arreglo de número enteros son positivos.

```
proc todosPositivos (in xs: seq<Z>) : Bool {  
  requiere {true}  
  asegura {res = true  $\leftrightarrow$  ( $\forall x : \mathbb{N}$ ) ( $x \in xs \rightarrow 0 < x$ )}
```

Ejercicio 4. Debuggear el código que implementa la función `maximo`, que devuelve el máximo valor de un arreglo de números enteros.

```
proc maximo (in xs: seq<Z>) : Z {  
  requiere {|xs| > 0}  
  asegura {res  $\in xs \wedge$  ( $\forall x : \mathbb{N}$ ) ( $x \in xs \rightarrow x \leq res$ )}
```

Ejercicio 5. Debuggear el código que implementa la función `ordenado`, que determina si un arreglo de número enteros está ordenado de manera creciente.

```
proc ordenado (in xs: seq<Z>) : Bool {  
  requiere {true}  
  asegura {res = true  $\leftrightarrow$  ( $\forall i, j : \mathbb{N}$ ) ( $i < j \wedge j < |xs| \rightarrow_L xs[i] \leq xs[j]$ )}
```

2. Testing

Los siguientes ejercicios tienen ya un código que los implementa correctamente. Su objetivo es generar un conjunto de tests que cubra todos los casos de uso.

Ejercicio 6. Crear tests para la función `fizzBuzz`, que dado un número natural devuelve un `String` según corresponda:

- Si es múltiplo de 3 y 5: "FizzBuzz"
- Si es múltiplo de 3 pero no de 5: "Fizz"
- Si es múltiplo de 5 pero no de 3: "Buzz"
- Si no es múltiplo ni de 3 y ni de 5: devuelve el número como `String`

Ejercicio 7. Crear tests para la función `numeroCombinatorio`.

```
proc numeroCombinatorio (in n: ℕ, in k ℕ) : ℕ {  
  requiere {true}  
  asegura {res =  $\binom{n}{k}$ }  
}
```

Ejercicio 8. Crear tests para la función `repeticionesConsecutivas`, que devuelve el tamaño de la subsecuencia más larga de un arreglo tal que todos los elementos de la subsecuencia son iguales.

```
proc repeticionesConsecutivas (in xs: seq(ℤ)) : ℕ {  
  requiere {true}  
  asegura {existeSecuencia(res, xs) ∧ ¬existeSecuencia(res + 1, xs)}  
  pred existeSecuencia (largo : ℕ, xs: seq(ℤ)) {  
    (∃ posInicial: ℕ) (  
      posInicial + largo ≤ |xs| ∧L (∀i : ℕ) (  
        i < largo →L xs[posInicial] = xs[posInicial + i]  
      )  
    )  
  }  
}
```