Algoritmos y Estructuras de Datos

Guía de laboratorio 2
Debugging, testing y manejo de archivos

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Los ejercicios de *Debugging* y de *Manejo de Archivos* 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 los archivos `Debugging.java` y `Archivos.java` al repositorio de entrega dentro de la carpeta taller2. **Última fecha de entrega:** domingo 03/09.

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 $\veebar$). Este operador representa un "o exclusivo": $A \veebar B$ es verdadero cuando A ó B es verdadero, pero no ambos. Cumple la siguiente tabla de verdad:

A	B	$A \veebar B$
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⟩) = res : Bool  {
      requiere {true}
      asegura {res = true ↔ |xs| = |ys| ∧_L (∀i : ℕ) (i < |xs| ⟶_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⟩) = res : Bool  {
      requiere {true}
      asegura {res = true ↔ (∀x : ℕ) (x ∈ xs ⟶ 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⟩) = res : ℤ  {
      requiere {|xs| > 0}
      asegura {res ∈ xs ∧ (∀x : ℕ) (x ∈ xs ⟶ x ≤ 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⟩) = res : Bool  {
      requiere {true}
      asegura {res = true ↔ (∀i, j : ℕ) (i < j ∧ j < |xs| ⟶_L xs[i] ≤ ys[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) = res : ℕ  {
      requiere {true}
      asegura {res = (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⟩) = res : 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]
              )
          )
      }
}
```

3. Manejo de archivos

Ejercicio 9. Crear una función `leerVector`, que dado un *Scanner* y un *largo* : ℕ, lee un arreglo de tamaño *largo* del scanner.

Ejercicio 10. Crear una función `leerMatriz`, que dado un *Scanner* y *filas*, *columnas* : ℕ, lee una matriz de tamaño *filas* × *columnas* del scanner. Pensar como reutilizar el código del ejercicio anterior.

Ejercicio 11. Crear una función `imprimirPiramide`, que dado un *PrintStream* y *alto* : ℕ, imprime una pirámide con el alto pedido en el stream. Una pirámide es un texto con espacios y asteriscos, con *alto* cantidad de filas, donde cada fila tiene una cantidad impar creciente de asteriscos centrados por espacios. Por ejemplo, la pirámide de *alto* = 4 es:

```
   *
  ***
 *****
*******
```

Notar que la primer línea tiene 3 espacios a la izquierda, 1 asterisco y 3 espacios más a la derecha.