

Introducción a la programación

Repaso

Especificación

Completar las siguientes especificaciones:

problema a (in b: $seq\langle\mathbb{Z}\rangle$, in c: \mathbb{Z}) : \mathbb{Z} {
 requiere x: $\{c \in b\}$
 asegura y: $\{(\exists i : \mathbb{Z})(0 \leq i < |b| \wedge b[i] = c \wedge resultado = i)\}$
}

Especificación

Completar las siguientes especificaciones:

problema a (in b: $seq\langle\mathbb{Z}\rangle$, in c: \mathbb{Z}) : \mathbb{Z} {
 requiere x: $\{c \in b\}$
 asegura y: $\{(\exists i : \mathbb{Z})(0 \leq i < |b| \wedge b[i] = c \wedge resultado = i)\}$
}

Primero miremos las expresiones y tratemos de escribirlas en lenguaje natural

Especificación

problema a (in b: $seq\langle\mathbb{Z}\rangle$, in c: \mathbb{Z}) : \mathbb{Z} {
 requiere x: $\{c \in b \leadsto c \text{ pertenece a } b\}$
 asegura y: $\{(\exists i : \mathbb{Z})(0 \leq i < |b| \wedge b[i] = c \wedge resultado = i) \leadsto$
 resultado es una posición de *b* donde está el elemento *c*
}

Especificación

problema a (in b: $\text{seq}\langle\mathbb{Z}\rangle$, in c: \mathbb{Z}) : \mathbb{Z} {
 requiere x: $\{c \in b \leadsto c \text{ pertenece a } b\}$
 asegura y: $\{(\exists i : \mathbb{Z})(0 \leq i < |b| \wedge b[i] = c \wedge \text{resultado} = i) \leadsto$
 resultado es una posición de *b* donde está el elemento *c*
}

Ahora podemos ponerle nombres a los requiere y asegura

Especificación

```
problema a (in b: seq<Z>, in c: Z) : Z {  
  requiere pertenece: {c ∈ b}  
  asegura estaElementoEnPosicion:  
    {(∃i : Z)(0 ≤ i < |b| ∧ b[i] = c ∧ resultado = i)}  
}
```

Especificación

problema a (in b: $seq\langle\mathbb{Z}\rangle$, in c: \mathbb{Z}) : \mathbb{Z} {
 requiere pertenece: $\{c \in b\}$
 asegura estaElementoEnPosicion:
 $\{(\exists i : \mathbb{Z})(0 \leq i < |b| \wedge b[i] = c \wedge resultado = i)\}$
}

Y ahora podemos ponerle nombre al problema

Especificación

```
problema buscarPosicion (in b:  $seq\langle\mathbb{Z}\rangle$ , in c:  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere pertenece:  $\{c \in b\}$   
  asegura estaElementoEnPosicion:  
     $\{(\exists i : \mathbb{Z})(0 \leq i < |b| \wedge b[i] = c \wedge resultado = i)\}$   
}
```


Especificación

```
problema buscarPosicion (in b: seq<ℤ>, in c: ℤ) : ℤ {  
  requiere pertenece: {c ∈ b}  
  asegura estaElementoEnPosicion:  
    {(∃i : ℤ)(0 ≤ i < |b| ∧ b[i] = c ∧ resultado = i)}  
}
```

Y finalmente reemplazamos las variables en los predicados

Especificación

Finalmente nos queda:

```
problema buscarPosicion (in l: seq<ℤ>, in elemento: ℤ) : ℤ {  
  requiere pertenece: {elemento ∈ l}  
  asegura estaElementoEnPosicion:  
    { (∃ i : ℤ) (0 ≤ i < |l| ∧ l[i] = elemento ∧ resultado = i) }  
}
```

Especificación (más difícil)

```
problema a (in b: seq<Z>) : seq<Z> {  
  requiere: {True}  
  asegura w: {|resultado| ≤ |b|}  
  asegura x: {(∀i : Z)(0 ≤ i < |resultado| →  
    (∃j : Z)(0 ≤ j < |b| ∧ resultado[i] = b[j]))}  
  asegura y: {(∀i : Z)(0 ≤ i < |b| →  
    (∃j : Z)(0 ≤ j < |resultado| ∧ b[i] = resultado[j]))}  
  asegura z: {(∀i, j : Z)((0 ≤ i, j < |resultado| ∧  
    resultado[i] = resultado[j]) → i = j)}  
}
```

Especificación (más difícil)

problema a (in b: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$ {
 requiere: {True}
 asegura w: $\{|resultado| \leq |b| \leadsto$
 la long de resultado es a lo sumo la de la lista original}
 asegura x: $\{(\forall i : \mathbb{Z})(0 \leq i < |resultado| \rightarrow$
 $(\exists j : \mathbb{Z})(0 \leq j < |b| \wedge resultado[i] = b[j])) \leadsto$
 todos los elementos de resultado están en b}
 asegura y: $\{(\forall i : \mathbb{Z})(0 \leq i < |b| \rightarrow$
 $(\exists j : \mathbb{Z})(0 \leq j < |resultado| \wedge b[i] = resultado[j])) \leadsto$
 todos los elementos de b están en resultado}
 asegura z: $\{(\forall i, j : \mathbb{Z})((0 \leq i, j < |resultado| \wedge$
 $resultado[i] = resultado[j]) \rightarrow i = j) \leadsto$
 resultado no tiene elementos repetidos}
}

Especificación (más difícil)

```
problema a (in b: seq<Z>) : seq<Z> {  
  requiere: {True}  
  
  asegura tieneLongMenorOIgual:  $\{|resultado| \leq |b|\}$   
  asegura estanTodosEnB:  $\{(\forall i : \mathbb{Z})(0 \leq i < |resultado| \rightarrow$   
     $(\exists j : \mathbb{Z})(0 \leq j < |b| \wedge resultado[i] = b[j]))\}$   
  
  asegura estanTodosEnResultado:  $\{(\forall i : \mathbb{Z})(0 \leq i < |b| \rightarrow$   
     $(\exists j : \mathbb{Z})(0 \leq j < |resultado| \wedge b[i] = resultado[j]))\}$   
  
  asegura sinRepetidos:  $\{(\forall i, j : \mathbb{Z})((0 \leq i, j < |resultado| \wedge$   
     $resultado[i] = resultado[j]) \rightarrow i = j)\}$   
}
```

Especificación (más difícil)

```
problema eliminarRepetidos (in l: seq<ℤ>) : seq<ℤ> {  
  requiere: {True}  
  
  asegura tieneLongMenorOIgual: {|resultado| ≤ |l|}  
  asegura estanTodosEnL: {(∀i : ℤ)(0 ≤ i < |resultado| →  
    (∃j : ℤ)(0 ≤ j < |l| ∧ resultado[i] = l[j]))}  
  
  asegura estanTodosEnResultado: {(∀i : ℤ)(0 ≤ i < |l| →  
    (∃j : ℤ)(0 ≤ j < |resultado| ∧ l[i] = resultado[j]))}  
  
  asegura sinRepetidos: {(∀i, j : ℤ)((0 ≤ i, j < |resultado| ∧  
    resultado[i] = resultado[j]) → i = j)}  
}
```

Como podemos extender esa especificación para que devuelva además una lista de pares de enteros que indique los elementos y cantidades eliminadas?

Como podemos extender esa especificación para que devuelva además una lista de pares de enteros que indique los elementos y cantidades eliminadas?

```
problema eliminarYcontarRepetidos (in  $l : seq\langle \mathbb{Z} \rangle$ ) :  
   $seq\langle \mathbb{Z} \rangle \times seq\langle \mathbb{Z} \times \mathbb{Z} \rangle$  {  
    requiere: {...}  
    asegura: {...}  
  }
```


Pensemos primero qué queremos decir...

Pensemos primero qué queremos decir...

```
problema eliminarYcontarRepetidos (in l: seq(Z)) :  
seq(Z) × seq(Z × Z) {  
  requiere: {True}  
  asegura noQuedanRepetidos: {...}  
  asegura contarRepeticionesEliminadas: {...}  
}
```

Pensemos primero qué queremos decir...

```
problema eliminarYcontarRepetidos (in l: seq<Z>) :  
seq<Z> × seq<Z × Z> {  
  requiere: {True}  
  asegura noQuedanRepetidos: {...}  
  asegura contarRepeticionesEliminadas: {...}  
}
```

```
problema contarRepeticionesEliminadas (in l: seq<Z>) :  
seq<Z × Z> {  
  requiere: {True}  
  asegura estanLosRepetidosEnRes: {...}  
  asegura cantRepeticionesOk: {...}  
  asegura noFaltaNinguno: {...}  
}
```

```

problema eliminarYcontarRepetidos (in l: seq<Z>) :
seq<Z> × seq<Z × Z> {
  requiere: {True}
  asegura noQuedanRepetidos: {resultado0=eliminarRepetidos(l)}
  asegura contarRepeticionesEliminadas:
    {resultado1=contarRepetidos(l)}
}

```

```

problema contarRepetidos (in l: seq<Z>) : seq<Z × Z> {
  requiere: {True}
  asegura estanLosRepetidosEnRes: {(∀i : Z)(0 ≤ i <
    |resultado| → cantApariciones(resultado[i]0, l) > 1)}
  asegura cantRepeticionesOk: {(∀i : Z)(0 ≤ i < |resultado| →
    (cantApariciones(resultado[i]0, l) - 1 = resultado[i]1) ∧
    cantApariciones(resultado[i], resultado) = 1)}
  asegura noFaltaNinguno:
    {(∀i : Z)(0 ≤ i < |l| ∧ cantApariciones(l[i], l) > 1) →
    estaEnComp0(l[i], resultado)}
}

```

```

pred estaEnComp0 (in elem: Z, in l: seq<Z × Z>) {
  (∃i : Z)(0 ≤ i < |l| ∧ l[i]0 = elem)
}

```

Ahora vamos a programar!

- ▶ Implementar en Haskell la función
`eliminarYcontarRepetidos :: [Int] -> ([Int], [(Int, Int)])`
- ▶ Implementar en Python la función `def sacarRepetidos(l:list) -> list`

Sea el siguiente programa y su especificación:

```
def f(s1: list[int], s2: list[int]):
    i:int = 0
    a:int = 0
    b:int = 0
    while i < len(s1):
        a = s1[i]
        if i >= len(s2):
            b = 0
        else:
            b = s2[i]
        s1[i] = a + b
        if i < len(s2):
            if a - b > 0:
                s2[i] = b - a
            else:
                s2[i] = a - b
        i += 1
```

```
problema f (inout s1: seq(Z), inout s2: seq(Z)) {
    requiere: {True}
    modifica: {s1}
    modifica: {s2}
    asegura: {
        ( $\forall i \in \mathbb{Z})((0 \leq i < |s1| \wedge 0 \leq i < |s2|) \rightarrow$ 
        ( $s1[i] = s1@pre[i] + s2@pre[i] \wedge$ 
        ( $s2[i] = abs(s1@pre[i] - s2@pre[i]))$ 
        ) }
    asegura: {( $\forall i \in \mathbb{Z})((i \geq |s1| \wedge 0 \leq i < |s2|) \rightarrow s2[i] = s2@pre[i])$ }
    asegura: {( $\forall i \in \mathbb{Z})((i \geq |s2| \wedge 0 \leq i < |s1|) \rightarrow s1[i] = s1@pre[i])$ }
}
```

Zoom a la especificación

```
problema f (inout s1: seq⟨ℤ⟩, inout s2: seq⟨ℤ⟩) {  
  requiere: {True}  
  modifica: {s1}  
  modifica: {s2}  
  asegura: {  
    ( $\forall i \in \mathbb{Z}((0 \leq i < |s1| \wedge 0 \leq i < |s2|) \rightarrow$   
      ( $s1[i] = s1@pre[i] + s2@pre[i] \wedge$   
        ( $s2[i] = abs(s1@pre[i] - s2@pre[i]))$   
    ) }  
  asegura: {( $\forall i \in \mathbb{Z}((i \geq |s1| \wedge 0 \leq i < |s2|) \rightarrow s2[i] = s2@pre[i])$ }  
  asegura: {( $\forall i \in \mathbb{Z}((i \geq |s2| \wedge 0 \leq i < |s1|) \rightarrow s1[i] = s1@pre[i])$ }  
}
```

Cada caso de test propuesto debe contener la entrada y el resultado esperado.

1. Describir el diagrama de control de flujo (*control-flow graph*) del programa.
2. Escribir un conjunto de casos de test (o *test suite*) que cubra todas las sentencias. Mostrar qué líneas cubre cada test. Este conjunto de tests ¿cubre todas las decisiones? (Justificar).
3. Escribir un *test* que encuentre el defecto presente en el código (una entrada que cumple la precondition pero tal que el resultado de ejecutar el código no cumple la postcondición).
4. ¿Es posible escribir para este programa un *test suite* que cubra todas las decisiones pero que no encuentre el defecto en el código? En caso afirmativo, escribir el test suite; en caso negativo, justificarlo.

Preguntas teóricas

1. ¿Qué diferencia hay entre testing de caja blanca y de caja negra?
2. ¿Que es una variable InOut?