Introducción a la programación

Práctica 3: Introducción a Haskell
Segunda Parte

Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

g) **sumaDistintos:** que dados tres números enteros calcule la suma sin sumar repetidos (si los hubiera).

Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

g) **sumaDistintos:** que dados tres números enteros calcule la suma sin sumar repetidos (si los hubiera).

Esto tiene (al menos) dos interpretaciones posibles:

- Cuando hay algún número repetido no lo sumo
  - sumaDistintos(1,1,2) $= 2$
- Cuando hay algún número repetido lo sumo una sola vez
  - sumaDistintos(1,1,2) $= 3$

Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

g) **sumaDistintos:** que dados tres números enteros calcule la suma sin sumar repetidos (si los hubiera).

Esto tiene (al menos) dos interpretaciones posibles:

- Cuando hay algún número repetido no lo sumo
  - sumaDistintos(1,1,2) $= 2$
- Cuando hay algún número repetido lo sumo una sola vez
  - sumaDistintos(1,1,2) $= 3$

Una especificación **semi-formal** de la primera opción

```
problema sumaDistintos (x,y,z: ℤ) : ℤ {
  requiere: { - }
  asegura: {si los 3 paŕametros son distintos entonces res = x + y + z}
  asegura: {si 2 parámetros son iguales, res es igual al no repetido}
  asegura: {si los 3 parámetros son iguales, res = 0}
}
```

Ejercicio 2: Especificar e implementar las siguientes funciones, incluyendo su signatura.

g) **sumaDistintos:** que dados tres números enteros calcule la suma sin sumar repetidos (si los hubiera).

Esto tiene (al menos) dos interpretaciones posibles:

- Cuando hay algún número repetido no lo sumo
  - sumaDistintos(1,1,2) $= 2$
- Cuando hay algún número repetido lo sumo una sola vez
  - sumaDistintos(1,1,2) $= 3$

Una especificación **formal** de la primera opción

```
problema sumaDistintos (x,y,z: ℤ) : ℤ {
  requiere: {True}
  asegura: {( (x ≠ y) ∧ (x ≠ z) ∧ (y ≠ z) ) → res = x + y + z}
  asegura: {( (x = y) ∧ (x ≠ z) ∧ (y ≠ z) ) → res = z}
  asegura: {( (x ≠ y) ∧ (x = z) ∧ (y ≠ z) ) → res = y}
  asegura: {( (x ≠ y) ∧ (x ≠ z) ∧ (y = z) ) → res = x}
  asegura: {( (x = y) ∧ (x = z) ∧ (y = z) ) → res = 0}
}
```

Ejercicio 4. Especificar e implementar las siguientes funciones utilizando tuplas para representar pares, ternas de números

f) **posPrimerPar:** dada una terna de enteros, devuelve la posición del primer número par si es que hay alguno, y devuelve 4 si son todos impares).

Ejercicio 4. Especificar e implementar las siguientes funciones utilizando tuplas para representar pares, ternas de números

f) **posPrimerPar:** dada una terna de enteros, devuelve la posición del primer número par si es que hay alguno, y devuelve 4 si son todos impares).

Una especificación **semi-formal** de la primera opción

problema posPrimerPar (t: $\mathbb{Z}\times\mathbb{Z}\times\mathbb{Z}$) : $\mathbb{Z}$ {
  requiere: { - }
  asegura: {si algun elemento es par, entonces res es la posición del primer elemento par}
  asegura: {si ningún elemento es par, entonces $res = 4$}
}

Ejercicio 4. Especificar e implementar las siguientes funciones utilizando tuplas para representar pares, ternas de números

f) **posPrimerPar:** dada una terna de enteros, devuelve la posición del primer número par si es que hay alguno, y devuelve 4 si son todos impares).

Una especificación **formal** de la primera opción

$$
\begin{aligned}
&\texttt{problema posPrimerPar}\ (\mathrm{t}\colon \mathbb{Z}\times\mathbb{Z}\times\mathbb{Z}) : \mathbb{Z}\ \ \{\\
&\quad \texttt{requiere:}\ \{True\}\\
&\quad \texttt{asegura:}\ \{hayPar(t) \to (0 \leq res < 3 \land (\forall i : \mathbb{Z})((\ 0 \leq i <\\
&\qquad\qquad 3 \land esPar(t_i)\ ) \to i \geq res) \land esPar(t_{res})\ )\}\\
&\quad \texttt{asegura:}\ \{\neg hayPar(t) \to res = 4\}\\
&\}
\end{aligned}
$$

Ejercicio 9. A partir de las siguientes implementaciones en Haskell, describir en lenguaje natural qué hacen y especificarlas semiformalmente.

d)
```
f4 :: Float -> Float -> Float
f4 x y = (x+y)/2
```

e)
```
f5 :: (Float, Float) -> Float
f5 (x, y) = (x+y)/2
```

Ejercicio 9. A partir de las siguientes implementaciones en Haskell, describir en lenguaje natural qué hacen y especificarlas semiformalmente.

d)
```haskell
f4 :: Float -> Float -> Float
f4 x y = (x+y)/2
```

e)
```haskell
f5 :: (Float, Float) -> Float
f5 (x, y) = (x+y)/2
```

- ¿Qué hacen estas dos funciones?
- ¿Hacen lo mismo?
- ¿Son **iguales**?

Ejercicio 9. A partir de las siguientes implementaciones en Haskell, describir en lenguaje natural qué hacen y especificarlas semiformalmente.

d)
```
f4 :: Float -> Float -> Float
f4 x y = (x+y)/2
```

e)
```
f5 :: (Float, Float) -> Float
f5 (x, y) = (x+y)/2
```

- ¿Qué hacen estas dos funciones?
- ¿Hacen lo mismo?
- ¿Son **iguales**?

problema f4 (x,y: $\mathbb{R}$) : $\mathbb{R}$ {
  requiere: $\{True\}$
  asegura: $\{res = (x+y)/2\}$
}

problema f5 (t: $\mathbb{R} \times \mathbb{R}$) : $\mathbb{R}$ {
  requiere: $\{True\}$
  asegura: $\{res = (t_0 + t_1)/2\}$
}