



Formato de especificaciones:

```
problema nombreDelProblema (nombreParametroEntrada1: tipoDeDato1, nombreParametro2: tipoDeDato2) : tipoDatoDeSalida {  
  requiere: {Una proposición que utiliza parámetros de entrada}  
  requiere: {Otra proposición que utiliza parámetros de entrada}  
  asegura: {Proposición que puede utilizar parámetros de entrada y salida para hablar del resultado}  
  asegura: {Otra proposición que puede utilizar parámetros de entrada y salida para hablar del resultado}  
}
```

**Ejercicio 1.** Dadas las siguientes especificaciones, dar valores de entrada y salida que cumplan con los requiere y asegura respectivamente:

- a) problema duplicar ( $x: \mathbb{Z}$ ) :  $\mathbb{Z}$  {  
 requiere: {True}  
 asegura: {*resultado* es el doble de  $x$ }  
}
- b) problema raizCuadrada ( $x: \mathbb{Z}$ ) :  $\mathbb{R}$  {  
 requiere: { $x$  es positivo}  
 asegura: {*resultado* es la raíz cuadrada de  $x$ }  
}
- c) problema enteroMasCercanoPositivo ( $x: \mathbb{R}$ ) :  $\mathbb{Z}$  {  
 requiere: {True}  
 asegura: {*resultado* es el entero más cercano de  $x$ }  
 asegura: {*resultado* es positivo}  
}
- d) problema raicesCuadradasUno ( $s: seq(\mathbb{Z})$ ) :  $seq(\mathbb{R})$  {  
 requiere: {Todos los elementos de  $s$  son positivos}  
 requiere: {No hay elementos repetidos en  $s$ }  
 asegura: {*resultado* tiene la misma cantidad de elementos que  $s$ }  
 asegura: {Los elementos de *resultado* son la salida de aplicar el problema **raizCuadrada** a todos los elementos de la secuencia  $s$ }  
 asegura: {El orden de la secuencia *resultado* es el mismo que en la secuencia  $s$ , luego de aplicar el problema **raizCuadrada**}  
}
- e) problema raicesCuadradasDos ( $s: seq(\mathbb{Z})$ ) :  $seq(\mathbb{R})$  {  
 requiere: {Todos los elementos de  $s$  son positivos}  
 requiere: {No hay elementos repetidos en  $s$ }  
 asegura: {*resultado* tiene la misma cantidad de elementos que  $s$ }  
 asegura: {Los elementos de *resultado* son la salida de aplicar el problema **raizCuadrada** a todos los elementos de la secuencia  $s$ }  
}

- f) problema `raicesCuadradasTres` ( $s: seq\langle\mathbb{Z}\rangle$ ) :  $seq\langle\mathbb{R}\rangle$  {  
     `requiere`: {Todos los elementos de  $s$  son positivos}  
     `requiere`: {No hay elementos repetidos en  $s$ }  
     `asegura`: {*resultado* tiene la misma cantidad de elementos que  $s$ }  
     `asegura`: {Los elementos de *resultado* son la salida de aplicar el problema `raizCuadrada` a uno o varios elementos de la secuencia  $s$ }  
 }
- g) problema `raicesCuadradasCuatro` ( $s: seq\langle\mathbb{Z}\rangle$ ) :  $seq\langle\mathbb{R}\rangle$  {  
     `requiere`: {Existen elementos de  $s$  que son positivos}  
     `requiere`: {No hay elementos repetidos en  $s$ }  
     `asegura`: {Los elementos de *resultado* son la salida de aplicar el problema `raizCuadrada` a todos los elementos de  $s$  que son positivos}  
 }
- h) problema `raicesCuadradasCinco` ( $s: seq\langle\mathbb{Z}\rangle$ ) :  $seq\langle\mathbb{R}\rangle$  {  
     `requiere`: {Todos los elementos de  $s$  son positivos}  
     `asegura`: {Cada *posición* de *resultado*, donde la *posición* es menor o igual a las de  $s$ , es igual a la salida de aplicar `raizCuadrada` al elemento que se encuentra en esa *posición* en  $s$ }  
 }
- i) problema `raicesCuadradasSeis` ( $s: seq\langle\mathbb{Z}\rangle$ ) :  $seq\langle\mathbb{R}\rangle$  {  
     `requiere`: {Todos los elementos de  $s$  son positivos}  
     `asegura`: {La longitud de *resultado* es como máximo la misma que  $s$ }  
     `asegura`: {Cada *posición* de *resultado*, donde la *posición* es menor o igual a las de  $s$ , es igual a la salida de aplicar `raizCuadrada` al elemento que se encuentra en esa *posición* en  $s$ }  
 }

**Ejercicio 2.** A partir de las especificaciones del *Ejercicio 1*, responder las siguientes preguntas:

1. En los problemas `raicesCuadradas` que utilizan el problema `raizCuadrada`, ¿Se puede eliminar el `requiere` “*Todos los elementos de  $s$  son positivos*”? Justificar.
2. ¿Qué consecuencia tiene en el resultado la diferencia de `asegura` entre los problemas `raicesCuadradasUno` y `raicesCuadradasDos`? Buscar un ejemplo de valor de entrada donde cada problema tenga distinto valor de salida.
3. De acuerdo con la respuesta del ítem anterior, ¿un algoritmo que satisface la especificación de `raicesCuadradasUno`, también satisface la especificación de `raicesCuadradasDos`? ¿y al revés?
4. Explicar en palabras las diferencias entre los problemas `raicesCuadradasCinco` y `raicesCuadradasSeis`. ¿Cómo influye el `asegura` de longitud máxima? Dada la entrada  $s = \langle 3, 9, 11, 15, 18 \rangle$ , ¿es  $\langle \sqrt{3}, \sqrt{9} \rangle$  una salida válida para ambos problemas? Y sea la entrada  $s = \langle 3, 9, 11 \rangle$ , ¿es  $\langle \sqrt{3}, \sqrt{9}, \sqrt{11}, \sqrt{13} \rangle$  una salida válida para el problema `raicesCuadradasCinco`?
5. ¿Qué cambia en el problema `raicesCuadradasCuatro` agregar un `asegura` que diga que *resultado* tiene la misma longitud que  $s$ ? Pensar ejemplos de valores de salida que cambien con este nuevo `asegura`.
6. Si los problemas `raicesCuadradasDos` y `raicesCuadradasTres` tienen el mismo *resultado* para la misma entrada (una secuencia específica de números), ¿quiere decir que son el mismo problema?
7. ¿Qué ocurre si eliminamos los `requiere` “*no hay repetidos*”? Dada la entrada  $s = \langle 4, 1, 1 \rangle$ , ¿es  $\langle 2, 2, 1 \rangle$  una salida válida para el problema `raicesCuadradasDos`?

**Ejercicio 3.** Dada la siguiente especificación del problema de ordenar una secuencia de enteros, en la que se debe tomar una secuencia de números enteros y devolver los mismos elementos ordenados de menor a mayor:

```
problema ordenar (s: seq(Z)) : seq(Z) {
    requiere: {True}
    asegura: {resultado es una secuencia en la cual cada elemento es estrictamente mayor que el anterior}
}
```

responder la siguientes preguntas:

- Dada  $s = \langle 4, 3, 5 \rangle$  como secuencia de entrada, ¿es  $resultado = \langle 3, 4, 5 \rangle$  una solución válida según la especificación?
- Dado  $s = \langle 4, 3, 3, 5 \rangle$  como secuencia de entrada, ¿es  $resultado = \langle 3, 3, 4, 5 \rangle$  una solución válida según la especificación? Corregir la especificación modificando el *requiere*.
- Si tomamos  $s = \langle 4, 3, 5 \rangle$  como secuencia de entrada, ¿es  $resultado = \langle 3, 4 \rangle$  una solución válida según la especificación? Corregir la especificación modificando el *asegura*.
- Si tomamos  $s = \langle 4, 3, 5 \rangle$  como secuencia de entrada, ¿es  $resultado = \langle 3, 4, 5, 6 \rangle$  una solución válida según la especificación? Corregir la especificación modificando el *asegura*.
- Dada  $s = \langle 8, 5, 7 \rangle$  como secuencia de entrada, ¿es  $resultado = \langle 1, 2, 3 \rangle$  una solución válida según la especificación?
- Escribir una especificación que permita recibir cualquier secuencia de enteros  $s$  como parámetro y garantice que *resultado* contiene el resultado de ordenar correctamente los elementos  $s$  de menor a mayor.

**Ejercicio 4.** Se desea especificar el problema de reemplazar cada elemento de una secuencia de enteros por su doble y se cuenta con la siguiente especificación:

```
problema duplicarTodos (s: seq(Z)) : seq(Z) {
    requiere: {True}
    asegura: {resultado tiene la misma cantidad de elementos que s}
}
```

- ¿Qué problemas tiene la especificación dada? Dar ejemplos de valores para *resultado* que satisfagan la especificación pero no sean respuestas correctas.
- Indicar cuál/es de los siguientes *asegura* debería/n ser agregado/s a la especificación. Justificar en cada caso por qué deberían o no ser agregados.
  - *asegura*: {Para cada valor  $x$  que pertenece a  $s$ , hay algún valor en *resultado* que es la salida de `duplicar( $x$ )`}
  - *asegura*: {En cada *posición* de *resultado*, el valor es mayor al valor en esa misma *posición* de  $s$ }
  - *asegura*: {En cada *posición* de *resultado*, el valor es igual a la salida de aplicar `duplicar` al valor en esa misma *posición* de  $s$ }
  - *asegura*: {Todos los elementos de *resultado* son números pares}

**Nota:** el problema `duplicar( $x$ )` está especificado en el Ejercicio 1.

**Ejercicio 5. (Frecuencia de bondis)** A Ciudad Universitaria (CU) llegan 8 líneas de colectivos: 28, 33, 34, 37, 45, 107, 160 y 166. Con el fin de controlar la frecuencia diaria de cada línea, un grupo de investigación del Departamento de Computación instaló cámaras y un sistema de reconocimiento de imágenes en el ingreso al predio. Durante el día, el sistema identifica el número de línea de cada colectivo que ingresa y lo registra en una secuencia ordenada.

- Especificar el problema `cantidadColectivosDeLinea` que a partir de una *secuencia de colectivos* registrada por el sistema de reconocimiento y el *número de una línea* que llega a CU, devuelva cuántos colectivos de esa línea ingresaron durante el día.
- Especificar el problema `lineaConMejorFrecuencia` que, a partir de dos números de líneas y una secuencia registrada por el sistema, devuelva cuál de las dos líneas tiene mejor frecuencia diaria. **Sugerencia:** utilizar `cantidadColectivosDeLinea`.

### Ejercicio 6. (Control de Calificaciones en el Departamento de Ciencias)

En el prestigioso Departamento de Ciencias de una reconocida universidad, un grupo de estudiantes se ha embarcado en su jornada académica, cursando diversas materias bajo el cuidadoso seguimiento del cuerpo docente. En este departamento, cada estudiante ha sido registrado con su respectivo nombre y apellido, asegurándose de que no existan duplicados.

La información relevante de las cursadas de los estudiantes se encuentra almacenada en un sistema que contiene una secuencia de tuplas en formato *Materia*  $\times$  *Calificación obtenida*. Las calificaciones se encuentran en un rango numérico entre 0 y 10.

El Departamento ha establecido una política de aprobación y recursado que dicta que si un estudiante aprueba una materia con una calificación igual o superior a 4, no deberá volver a cursarla, quedando esta materia registrada como aprobada en su expediente académico. Sin embargo, si no logra alcanzar la calificación mínima de aprobación, tendrá la posibilidad de recursar la materia en un futuro intento.

Además, existe en el sistema una estructura de datos llamada *CalificacionesDelDC* que contiene la información de los estudiantes y las calificaciones en sus cursadas. Esta estructura es una secuencia de tuplas en el formato *Alumno*  $\times$  *Cursada*, donde *Alumno* es el nombre y apellido del estudiante y *Cursada* es la secuencia de tuplas mencionada previamente.

Considerando esta información y los siguientes renombres de tipos:

*Renombre Alumno* = *String*

*Renombre Materia* = *String*

*Renombre Cursada* = *seq*(*Materia*  $\times$   $\mathbb{R}$ )

*Renombre CalificacionesDelDC* = *seq*(*Alumno*  $\times$  *Cursada*)

- a) Especificar **problema promedioDeAlumno** (*alumno*: *Alumno*, *calificaciones*: *CalificacionesDelDC*) :  $\mathbb{R}$
- b) Especificar el problema que, dado el listado de materias cursadas por un estudiante, indique en qué materia tuvo mayor calificación. ¿Cómo se debe modificar la especificación para devolver *el listado de materias* en las cuales tuvo mejor calificación?
- c) Especificar el problema que, dada una materia y las calificaciones del DC, devuelve todos los estudiantes que cursaron y aprobaron esa materia. ¿Cómo debe modificarse la especificación para que los nombres se devuelvan en orden alfabético?, ¿este cambio reduce o amplía la cantidad de programas que resolverían el problema?
- d) Especificar el problema de devolver una secuencia con los promedios de todos los estudiantes.

## Principio de sustitución

**Ejercicio 7.** Contamos con las siguientes especificaciones del problema **pares**:

```
problema pares1 (s: seq( $\mathbb{Z}$ )) : seq( $\mathbb{Z}$ ) {  
  requiere: {s no tiene elementos repetidos}  
  asegura: {Los elementos de resultado son pares y pertenecen a s}  
  asegura: {Los elementos de s que son pares, pertenecen a resultado}  
  asegura: {resultado no tiene elementos repetidos}  
}
```

```
problema pares2 (s: seq( $\mathbb{Z}$ )) : seq( $\mathbb{Z}$ ) {  
  requiere: {s no tiene elementos repetidos}  
  asegura: {Los elementos de resultado son pares y pertenecen a s}  
  asegura: {Los elementos de s que son pares, pertenecen a resultado}  
  asegura: {resultado no tiene elementos repetidos}  
  asegura: {resultado está ordenada de manera creciente}  
}
```

- a) Si contamos con un algoritmo *P* que satisface **pares1**, ¿satisface *P* la especificación **pares2**? Justificar.
- b) Si contamos con un algoritmo *P* que satisface **pares2**, ¿satisface *P* la especificación **pares1**? Justificar.
- c) ¿Cuál es la relación de fuerza entre la postcondición de **pares1** y la de **pares2**?

**Ejercicio 8.** Contamos con las siguientes especificaciones del problema `sumarAbsMayorA5`:

```
problema sumarAbsMayorA5-version1 (s: seq(Z)) : Z {  
  requiere: {True}  
  asegura: {resultado es la sumatoria de todos los elementos de s cuyo valor absoluto es mayor a 5}  
}
```

```
problema sumarAbsMayorA5-version2 (s: seq(Z)) : Z {  
  requiere: {Todos los elementos de s son positivos}  
  asegura: {resultado es la sumatoria de todos los elementos de s cuyo valor absoluto es mayor a 5}  
}
```

```
problema sumarAbsMayorA5-version3 (s: seq(Z)) : Z {  
  requiere: {Todos los elementos de s son mayores a 10}  
  asegura: {resultado es la sumatoria de todos los elementos de s cuyo valor absoluto es mayor a 5}  
}
```

- a) ¿Cuál es la relación de fuerza entre los *requiere* de cada especificación?
- b) ¿Cuál de las especificaciones tiene el dominio más restringido y cuál menos?
- c) Desde el punto de vista de un programador, ¿qué especificación es *más fácil* de implementar? Justificar.
- d) Desde el punto de vista de un usuario, ¿qué contrato es *más conveniente*? Justificar.