# Guía de Ejercicios 2: Especificación formal de problemas

[Version: 19 de agosto de 2024]

**Objetivos:**

- Aplicar lógica de primer orden para predicar sobre problemas computacionales.
- Entrenar la lectura y escritura de pre y post condiciones de problemas computacionales en lógica formal.
- Entrenar la especificación de problemas que involucran secuencias.

---

## Predicados lógicos

**Ejercicio 1.** Dadas las variables $m, n$ de tipo `int`, $x$ de tipo `float`, $cond$ de tipo `bool`, $str$ de tipo `string` y $vec$ de tipo `vector<int>`, escribir cada uno de los siguientes predicados en lógica formal.

(a) Si $cond$ es verdadero entonces la suma de $n$ y $m$ es mayor a 10.

(b) O bien $m$ es positivo o bien $x$ es igual a $3.14$, pero no ambas.

(c) Si el primer elemento de $vec$ es 5, entonces $str$ dice "Hola".

(d) La diferencia entre el primer elemento de $vec$ y $m$ es igual a $n$.

**Ejercicio 2.** Considere las variables del ejercicio anterior con los siguientes valores: $m = 1, n = 42, x = 0.66, cond =$ `true`, $str =$ `"Rebeca"`, $vec =$ `{6,8,13,0,22}`. Para cada uno de los siguientes predicados indique si toma el valor *verdadero, falso* o *indefinido*.

(a) $n/(m-1) = 0 \ \vee \ (n = 1)$

(b) $vec[3] = m$

(c) $str =$ `"Rebeca"` $\wedge \ vec[5] = m$

(d) $x = 0.66 \Rightarrow str =$ `"Guber"`

**Ejercicio 3.** Dadas las variables $m, n$ de tipo `int`, $str$ de tipo `string` y $vec$ de tipo `vector<int>`, escribir cada uno de los siguientes predicados en lógica formal.

(a) $str$ no contiene la letra `'x'`.

(b) $m$ es una posición válida de $vec$ y todos los elementos de $vec$ hasta la posición $m$ son iguales a cero.

(c) La suma de todos los elementos de $vec$ está entre $m$ y $n$.

(d) El elemento en la posición $m$ de $vec$ es el máximo del vector.

**Ejercicio 4.** Considere las variables del ejercicio anterior con los siguientes valores: $m = 1, n = 42, str =$ `"Rebeca"`, $vec =$ `{-1, 6,8,13,0,22}`. Para cada uno de los siguientes predicados indique si toma el valor *verdadero, falso* o *indefinido*.

(a) $(\forall i : \text{int})(vec[i] = 0)$

(b) $(\exists i : \text{int})(0 \leq i < |vec| \land vec[i] = 0)$

(c) $(\exists i : \text{int})(i > m \land str[i] = \texttt{'e'})$

(d) $\sum_{i=m}^{i<|vec|} vec[i] + n = 487$

(e) $(\forall i : \text{int})(\ 0 \leq i < |vec| \Rightarrow (\exists j : \text{int})$
$(0 \leq j < i \land vec[j] < vec[i])\ )$

**Ejercicio 5.** Escribir las siguientes funciones auxiliares en el lenguaje de especificación. (No confundir con especificar problemas)

(a) contarPares($v$ : `vector<int>`) : `int`
Dado un vector de enteros $v$, cuenta la cantidad de elementos pares.

(b) sumarPrimosHastaI($v$ : `vector<int>`, $i$ : `int`) : `int`
Dado un vector de enteros $v$, suma los elementos primos desde la posición $0$ hasta $i$.

(c) promedio($v$ : `vector<float>`) : `float`
Dado un vector de floats $v$, calcula el promedio de los elementos.

(d) contarVocales($txt$ : `vector<char>`) : `int`
Dado un vector de chars $txt$, cuenta la cantidad de vocales en el texto.

## Especificación de problemas

**Ejercicio 6.** Especificar formalmente cada uno de los siguientes problemas escribiendo *encabezado*, *precondición* y *poscondición*.

(a) Dado un entero $n \geq 0$, devolver un `string` con una línea de $n$ asteriscos.

(b) Dado un `string`, calcular la cantidad de veces que contiene la letra `'a'`.

(c) Dado un entero no negativo, devolver un `vector<int>` con su representación en base 2. Por ejemplo, para 26 debe devolver `{1,1,0,1,0}`.

(d) Dado un `vector<int>` con una representación en base 2, calcular el valor del número representado. Por ejemplo, para `{1,1,0,1,0}` debe devolver 26.

(e) Dado un `string`, determinar si se trata de un palíndromo (que se lee igual al derecho y al revés). Por ejemplo, `"reconocer"` es un palíndromo, pero `"desconocer"` no lo es.

(f) Dados dos `string`, devolver cuántas veces el primero está contenido en el segundo.

**Ejercicio 7.** Sea un programa $P$ que cumple la especificación (a), £cumplirá también la especificación (b)?. Y si un programa $P'$ cumple la especificación (b), £cumplirá también la especificación (a)? Justificar.

(a) `int problema_uno(vector<int> vec)`
Pre: $|vec| \leq 10$
Post: $res > \sum_{0 \leq i < |vec|} vec[i]$

(b) `int problema_dos(vector<int> vec)`
Pre: $|vec| \leq 100$
Post: $res > 5 + \sum_{0 \leq i < |vec|} vec[i]$

**Ejercicio 8.** Cada una de las siguientes especificaciones contiene errores (con respecto a la descripción del problema en español). Identificar cuál es el error y cómo subsanarlo.

(a) Dado un `vector<float>` $vec$ y un índice del vector, indicar si el elemento en esa posición es mayor a $37.5$.

`bool es_mayor(vector<float> vec, int i)`
Pre: $\top$
Post: $res = \text{true} \iff vec[i] > 37.5$

(b) Dado un número entero $n$, devolver otro número entero que sea múltiplo de $n$.

`int multiplo(int n)`
Pre: $\top$
Post: $res = n * 2$

(c) Dado un `vector<int>` $vec$ y un índice del vector, indicar si tiene ceros en sus posiciones pares.

`bool ceros_en_pares(vector<int> vec)`
Pre: $\top$
Post: $res = \text{true} \iff \big((\forall i : \text{int})(0 \leq i < |vec|) \implies (i \mod 2 = 0 \land vec[i] = 0)\big)$

(d) Dado un `vector<int>` $v$, devuelve la suma de sus elementos y modifica el vector poniendo ceros en todas sus posiciones.

`int suma_y_reset(vector<int> v)`
Pre: $v = v_0$
Post: $|v| = |v_0| \land res = \sum_{0 \leq i < |v|} v[i] \land (\forall i : \text{int})(0 \leq i < |v| \implies v[i] = 0)$

**Ejercicio 9.** Considere el siguiente problema: dado un número entero $n$ entre 1 y 10 (incluidos) se debe devolver un `vector<int>` con esa cantidad de elementos y que sólo contenga números pares. Para cada una de las siguientes propuestas indique si tiene un error y cómo subsanarlo.

(a) `vector<int> vector_con_pares(int n)`
Pre: $1 \leq n \leq 10$
Post: $|res| = n$

(b) Pre: $1 \leq n \leq 10$
Post: $|res| = n \land (\forall i : \text{int})(0 \leq i < |res| \implies res[i] = 2)$

(c) Pre: $n \geq 0$
Post: $|res| = n \land (\forall i : \text{int})(0 \leq i < |res| \implies res[i] \mod 2 = 0)$

(d) Pre: $n = 5$
Post: $|res| = n \land (\forall i : \text{int})(0 \leq i < |res| \implies res[i] \mod 2 = 0)$

**Ejercicio 10.** Especificar formalmente cada uno de los siguientes problemas escribiendo *encabezado*, *precondición* y *poscondición*.

(a) Dada una secuencia no vacía de `float`, encontrar su máximo elemento. Por ejemplo, para la lista `{1.0, 12.7, 1.0, 8.8, 12.7, 3.0}`, debería devolverse `12.7`.

(b) Dada una secuencia de `string`, construir un `string` que sea la concatenación de todos sus elementos. Por ejemplo, para la secuencia `["ab","c","","def"]`, se debe devolver `"abcdef"`.

(c) Dada una lista de enteros, determinar si está ordenada en forma estrictamente creciente. Por ejemplo, `{1,4,6}` y `{}` están ordenadas, pero `{6,4,1}` y `{4,6,6}` no lo están. Notar que una lista vacía se considera ordenada.

(d) Dada una lista de enteros, ordenar la lista de manera creciente. Por ejemplo, si la lista de entrada es `{4,1,6}` se debe modificar por `{1,4,6}`.

(e) Dado un número natural `n`, construir una lista con los primeros `n` números naturales impares, ordenados de menor a mayor. Por ejemplo, para `n=3`, la lista esperada es `{1,3,5}`.

(f) Dada una lista de enteros `ls` y un entero `n`, modificar `ls` sumando `n` a cada uno de sus elementos. Por ejemplo, para `ls={1,9,7,7}` y `n=10`, el estado final de `ls` debe ser `{11,19,17,17}`.

(g) Dada una lista de enteros `ls` y un entero `n`, modificar `ls` de manera que contenga a todos sus elementos originales pero sumando `n` a cada uno. En este caso, no se requiere que se preserve el órden de la lista original. Por ejemplo, para `ls`={1,9,7,7} y `n`=10, un estado final válido para `ls` es {19,17,11,17}.

(h) Dado un `string txt` de longitud arbitraria y un `string sep` de un caracter de longitud, separar `txt` en cada aparición de `sep`, construyendo así una nueva lista. Por ejemplo, para `txt`="a;bb;c;;ddd;" y `sep`=";", la lista resultante es {"a","bb","c","","ddd",""}.

(i) Dado un `vector<float>` $v$ y un `vector<vector<int>>` $w$ de la misma longitud, devuelve un **bool** que indica si los números de $v$ son los promedios de las secuencias de $w$. Es decir, se devuelve `true` si cada valor de $v$ se corresponde con una de las secuencias de $w$ indicando el promedio de sus valores, abarcando todas las secuencias de $w$.

**Ejercicio 11.** Por cada especificación del ejercicio anterior:

(a) Escriba dos ejemplos de valores de entrada que no cumplan la precondición.

(b) Escriba dos ejemplos de valores de entrada que cumplan la precondición, junto con los valores de salida correspondientes, que deben cumplir la poscondición.

**Ejercicio 12.** Se desea modelar el problema de actualizar un **stock de ingredientes** luego de preparar tantas porciones como sea posible de una **receta** dada.

- Un **stock** de ingredientes es un vector de pares (`string`, **int**) donde el `string` indica el nombre del ingrediente y el **int** indica cuántas unidades de dicho ingrediente se poseen en stock. Sólo se listan ingredientes con alguna unidad en stock y no hay nombres de ingredientes repetidos.
- Una **receta** es un vector de pares (`string`, **int**) donde el **int** indica cuántas unidades del ingrediente son necesarias para preparar **una porción** de la receta. Sólo se listan ingredientes de los que se requiere al menos una unidad y no hay nombres de ingredientes repetidos.

Nos interesa especificar el problema **CocinarMuchos** que recibe un stock de ingredientes $S$ y una receta $R$ y devuelve un entero que indica cuántas porciones de $R$ es posible preparar. Además, *actualiza* el stock $S$ con lo que haya sobrado de ingredientes.

**Ejemplo** Para un stock inicial

$$S = \{(\text{"papa"}, 15), (\text{"huevo"}, 4), (\text{"zanahoria"}, 3), (\text{"chocolate"}, 2)\}$$

y para una receta de ensalada $R = \{(\text{"papa"}, 2), (\text{"huevo"}, 1)\}$, se devolverá como resultado el número **4** de porciones preparadas, y quederá el stock final

$$S = \{(\text{"papa"}, 7), (\text{"zanahoria"}, 3), (\text{"chocolate"}, 2)\}$$

Se sugiere definir y utilizar los siguientes predicados auxiliares

- esStockValido(S: `vector<pair<string, int>>`), que indica si el vector de pares S cumple las condiciones de ser un stock. Notar que son las mismas condiciones para que el vector S sea una receta válida.
- sePuedeCocinarN(n: **int**, R: `vector<pair<string, int>>`, S: `vector<pair<string, int>>`), que indica si es posible cocinar al menos `n` porciones de la receta `R` usando los ingredientes del stock `S`.

- esStockActualizado(
  $S_1$: `vector<pair<string, int>>`,
  $S_2$: `vector<pair<string, int>>`, $R$: `vector<pair<string, int>>`,
  $n$: `int`)

  que indica si $S_2$ es el estado final del stock $S_1$ luego de preparar $n$ porciones de la receta $R$.