Ejercicio 1. 2 puntos

1. [1 punto]

   Completar en las siguientes especificaciones nombres adecuados para el problema a, los parámetros b y c, y las etiquetas x, y, z, u y w.

   `problema a` (in b: $seq\langle Char \times Char\rangle$, in c: $seq\langle Char\rangle$) : $seq\langle Char\rangle$ {

   `requiere x:` { $(\forall i, j : \mathbb{Z})((0 \leq i < |b| \wedge 0 \leq j < |b| \wedge i \neq j) \rightarrow b[i]_0 \neq b[j]_0)$ }

   `requiere y:` { $(\forall i, j : \mathbb{Z})((0 \leq i < |b| \wedge 0 \leq j < |b| \wedge i \neq j) \rightarrow b[i]_1 \neq b[j]_1)$ }

   `requiere z:` { $(\forall i : \mathbb{Z})(0 \leq i < |c| \rightarrow (\exists j : \mathbb{Z})(0 \leq j < |b| \wedge b[j]_0 = c[i]))$ }

   `asegura u:` { $|resultado| = |c|$ }

   `asegura w:` { $(\forall i : \mathbb{Z})(0 \leq i < |c| \rightarrow (\exists j : \mathbb{Z})(0 \leq j < |b| \wedge b[j]_0 = c[i] \wedge b[j]_1 = resultado[i]))$ }

   }

   Algunos ejemplos:

   $b =< (`a`, `A`), (`b`, `B`), ..., (`z`, `Z`) >$
   $c =< `h`, `o`, `l`, `a` >$
   Si aplicamos $a(b, c) =< `H`, `O`, `L`, `A` >$

   En este caso la especificación sería de un programa que reemplaza cada aparición de los caracteres que están como primeros elementos de las tuplas por los caracteres que están como segundos elementos de esas tuplas.

   Una propuesta de etiquetas (se podrian haber usado otras):

   $a = codificar$
   $b = codigo$
   $c = palabra$
   $x = losCodigosSonTodosDiferentes$
   $y = losValoresSonTodosDiferentes$
   $z = soloTieneCodigos$
   $u = mismaLongitud$
   $w = estaCodificada$

2. [1 punto]

   Especificar el siguiente problema (se puede especificar de manera formal o semi-formal):

   Dados los inputs *b:* $seq\langle Char \times Char\rangle$, *m:* $seq\langle seq\langle Char\rangle\rangle$ y *n:* $seq\langle seq\langle Char\rangle\rangle$, retornar verdadero si n es igual al resultado de aplicar el `problema a` (del punto 1.1) a cada elemento de la secuencia m.

Tenemos que tener en cuenta un par de consideraciones:

1. para poder usar el problema anterior (codificar) tenemos que satisfacer los requiere sobre b y c
2. quizas tengamos que agregar nuevos requiere a nuestro nuevo problema
3. seguro que tenemos que escribir al menos 1 asegura del nuevo problema

`problema sonTodasCodificaciones` (in *b:* $seq\langle Char \times Char\rangle$, in *m:* $seq\langle seq\langle Char\rangle\rangle$, in *n:* $seq\langle seq\langle Char\rangle\rangle$): Bool) :
{
}

Lo primero que vamos a hacer es escribir el asegura del nuevo problema. Lo que queremos es devolver true/false dependiendo de los valores de n y m que le pasamos como parámetro. Entonces

asegura {$res = true \iff$ n es el resultado de aplicar $codificar()$ a cada uno de los elementos de m con b como primer parámetro}

Como n es una $seq\langle seq\langle Char\rangle\rangle$ eso quiere decir que cada elemento de la secuencia podría ser o no el resultado de codificar el elemento correspondiente de m:

asegura $\{res = true \iff (\forall i : \mathbb{Z})(0 \leq i < |n| \rightarrow (n[i] = codificar(b, m[i])))\}$

Pero si escribimos esto tenemos que estar seguro que cada vez que accedemos a $n[i]$ podemos acceder a $m[i]$. Notar que el rango de i esta definido sobre n y no sobre m. Entonces lo deberíamos agregar como una restricción del nuevo problema:

requiere $NyMTienenLaMismaLongitud$ $\{|n| = |m|\}$

Y como habíamos mencionado al principio, para poder usar $codificar()$ tenemos que estar seguros que los parámetros que le estamos pasando cumple sus requiere. Entonces tenemos que agregar nuevos requiere:

requiere $losCodigosSonTodosDiferentes$ {los códigos que estan en b son todos diferentes}

requiere $losValoresSonTodosDiferentes$ {los valores que están en b son todos diferentes}

Y lo ultimo que tenemos que requerir es que todos los elementos de m cumplan el tercer requiere:

requiere $todosSoloTienenCodigos$ {todos los elementos de m cumplen soloTieneCodigos}

NOTA: podemos ver que NO podemos usar el predicado $soloTieneCodigos$ porque habla sólo de una secuencia, y nosotros necesitamos que el requiere aplique sobre todos las secuencia de secuencias de m.

Si lo especificamos formalmente:

`problema sonTodasCodificaciones` (in b: , in m: $seq\langle seq\langle Char\rangle\rangle$, in n: $seq\langle seq\langle Char\rangle\rangle$) : Bool {
`requiere losCodigosSonTodosDiferentes:` $\{ (\forall i, j : \mathbb{Z})((0 \leq i < |b| \land 0 \leq j < |b| \land i \neq j) \rightarrow b[i]_0 \neq b[j]_0) \}$
`requiere losValoresSonTodosDiferentes:` $\{ (\forall i, j : \mathbb{Z})((0 \leq i < |b| \land 0 \leq j < |b| \land i \neq j) \rightarrow b[i]_1 \neq b[j]_1) \}$
`requiere todosSoloTienenCodigos:` $\{ (\forall k : \mathbb{Z})(0 \leq k < |m| \rightarrow (\forall i : \mathbb{Z})(0 \leq i < |m[k]| \rightarrow (\exists j : \mathbb{Z})(0 \leq j < |b| \land b[j]_0 = m[k][i])))$ }
`requiere NyMTienenLaMismaLongitud:` $\{ |m| = |n| \}$
`asegura esElResultadoDeCodificar:` $\{res = true \iff (\forall i : \mathbb{Z})(0 \leq i < |n| \rightarrow (n[i] = codificar(b, m[i])))\}$
}

Ejercicio 2. 4 puntos

1. [2 puntos]

   Programar en Haskell una función que satisfaga la especificación del `problema a` del Ejercicio 1. Recordá escribir los tipos de los parámetros.

   Tenemos varias formas de implementar esta función. Lo vamos a hacer usando una función principal con recursión sobre los elementos de la palabra y una función auxiliar que reemplace la letra a codificar:

```haskell
codificar :: [(Char, Char)] -> [Char] -> [Char]
codificar _ [] = []
codificar codigo (letra:ys) = (codificarLetra codigo letra):codificar codigo ys

codificarLetra  :: [(Char, Char)] -> Char -> Char
-- no hay caso vacio por especificacion
codificarLetra ((f,s):xs) letra | letra == f = s
                                | otherwise = codificarLetra xs letra
```

2. [2 puntos]

   Programar en Python una función que satisfaga la especificación del `problema a` del Ejercicio 1. Recordá escribir los tipos de los parámetros y variables que uses en tu implementación.

```python
def codificar (codigo: list[(str, str)], input: str) -> str:
    # Obs: en python no hay tipo char, y el tipo str es analogo a una
    # secuencia de char
    # Tambien era valido tiparlo como list[str]
  res: str = ""  # inicializo el str vacio, en caso de ser lista res = []

  for letra in input:
    for tupla in codigo:
      if letra == tupla[0]:
        res += tupla[1] # En caso de ser lista res.append(tupla[1])
```

```
  return res
```

Nota: hay diferentes formas de implementar la función. Este es sólo un ejemplo posible.

Otra posible solución usando un diccionario:

```
def codificar (codigo: list[(str, str)], input: list[str]) -> list[str]:
  res: list[str] = []
  d:dict[(str, str)] = {} # codigo no puede ser un dict de entrada,
                          # hay que respetar los tipos de la especificacion

  for (clave, valor) in codigo:
    d[clave] = valor

  for letra in input:
    res.append(d[letra]]

  return res
```

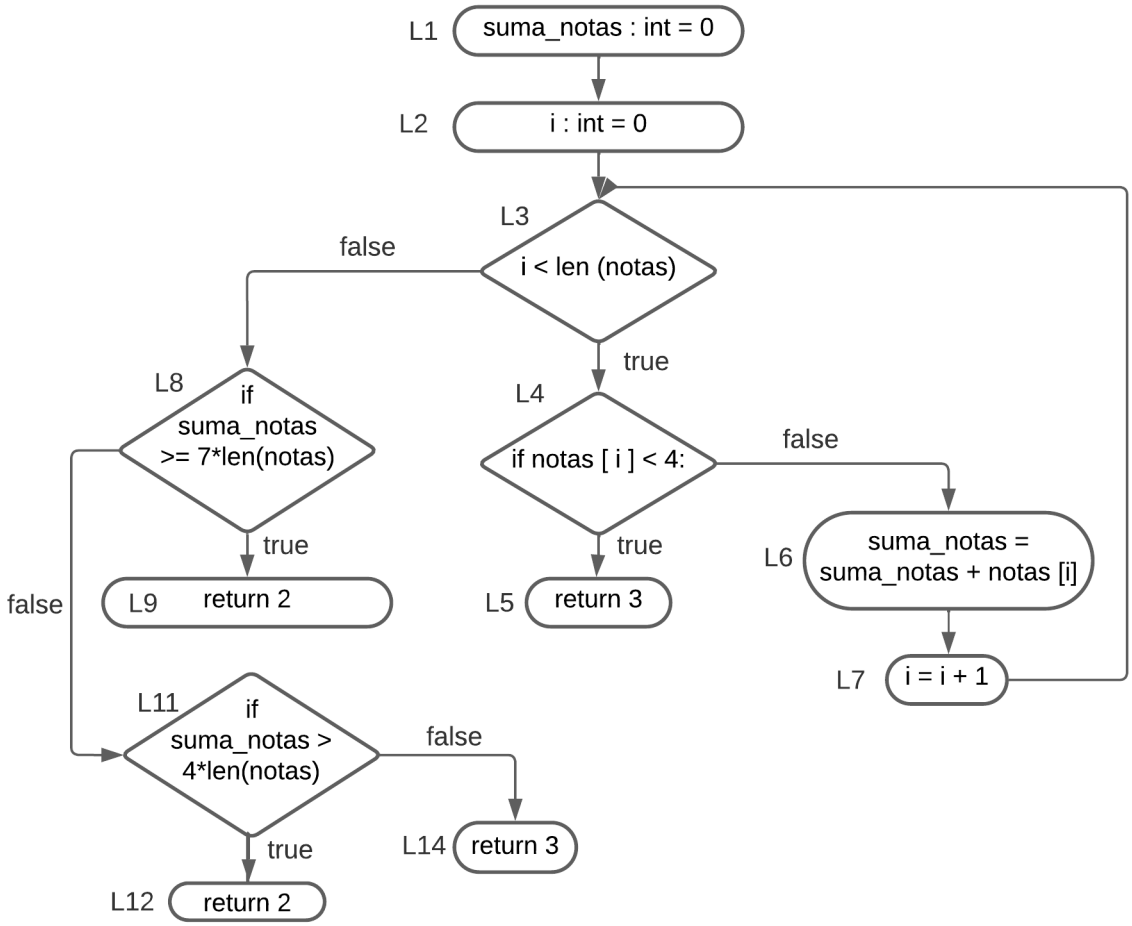
Ejercicio 3. 2 puntos

Sea la siguiente especificación del problema `aprobado` y una posible implementación en lenguaje imperativo:

`problema aprobado` (in notas: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
`requiere:` $\{|notas| > 0\}$
`requiere:` $\{(\forall i : \mathbb{Z})(0 \leq i < |notas| \rightarrow 0 \leq notas[i] \leq 10)\}$
`asegura:` $\{result = 1 \leftrightarrow$ todos los elementos de *notas* son mayores o iguales a 4 y el promedio es mayor o igual a 7$\}$
`asegura:` $\{result = 2 \leftrightarrow$ todos los elementos de *notas* son mayores o iguales a 4 y el promedio está entre 4 (inclusive) y 7$\}$
`asegura:` $\{result = 3 \leftrightarrow$ alguno de los elementos de *notas* es menor a 4 o el promedio es menor a 4$\}$
}

```
def aprobado(notas: list[int]) -> int:
L1:   suma_notas: int = 0
L2:   i: int = 0
L3:   while i < len(notas):
L4:      if notas[i] < 4:
L5:         return 3
L6:      suma_notas = suma_notas + notas[i]
L7:      i = i + 1
L8:   if suma_notas >= 7 * len(notas):
L9:      return 2
L10:  else:
L11:     if suma_notas > 4 * len(notas):
L12:        return 2
L13:     else:
L14:        return 3
```

1. Dar el diagrama de control de flujo (control-flow graph) del programa `aprobado`.



2. Escribir un test suite que ejecute todas las líneas del programa `aprobado`.

   Necesitamos cubrir todos los nodos. Como en particular hay 4 nodos return, necesitamos por lo menos 4 casos:

   - notas=[1], resultado esperado: 3, resultado obtenido: 3
   - notas=[5], resultado esperado: 2, resultado obtenido: 2
   - notas=[10], resultado esperado: 1, resultado obtenido: 2 ($ERROR$)
   - notas=[4], resultado esperado: 2, resultado obtenido: 3 ($ERROR$)

3. Escribir un test suite que tenga un cubrimiento de **al menos** el 50 por ciento de decisiones ("branches") del programa.

   La suite del punto anterior cubre todos los arcos, asi que con esa alcanza.

4. Explicar cuál/es es/son el/los error/es en la implementación. ¿Los test suites de los puntos anteriores detectan algún defecto en la implementación? De no ser así, modificarlos para que lo hagan.

   - La especificación dice que en el caso en que todos los elementos de notas son mayores o iguales a 4 y el promedio es mayor o igual a 7 el valor de retorno debe ser 1, y no es lo que sucede. Para solucionarlo habría que modificar la linea L9
   - El caso promedio=4 (sin valores menores a 4) no esta bien implementado. Para solucionarlo habría que modificar la L11

   Ambos errores son detectados por la suite propuesta.

Ejercicio 4. 2 puntos

1. [1 punto] Suponga las siguientes dos especificaciones de los problemas p1 y p2:

```
problema p1(x:Int):Int {
  requiere A;
  asegura C;
  }
```

```
problema p2(x:Int):Int {
  requiere B;
  asegura C;
  }
```

Si A es más fuerte que B, ¿Es cierto que todo algoritmo que satisface la especificación p1 también satisface la especificación p2? ¿Y al revés?, es decir, ¿Es cierto que todo algoritmo que satisface la especificación p2 también satisface la especificación p1? Justifique.

Supongamos que tenemos un algoritmo que satisface la especificación de p2. Esto quiere decir, que si se cumplen los requiere de p2 (B), el algoritmo satisface el asegura de p2 (C).

Como el enunciado nos dice que A es mas fuerte que B, eso significa que A es mas restrictivo que B, o lo que es equivalente, que los casos que satisfacen A son menos que los que satisfacen B. Supongamos que x es uno de esos valores que satisface B y no A.

Entonces voy a poder encontrar un algoritmo que cumpla la especificación de p1 para x (ya que su requiere no lo pide), pero que no satisfaga la especificación para p2.

Ejemplo:

problema p1(x:Int):Int {
  requiere A: $\{x = 2\}$;
  asegura C;
  }

problema p2(x:Int):Int {
  requiere B: $\{x \bmod 2 == 0\}$;
  asegura C;
  }

El caso contrario vale (si A es mas fuerte que B, todo algoritmo que satisface p2 también satisface p1).

2. [1 punto] ¿Es posible que haya un test suite con 100 % de cubrimiento de nodos que todos los test pasen pero que igual el programa tenga un bug? Justifique.

   Si, es posible. Veamos la siguiente implementación en python la función *abs*():

```
def abs(n: int) -> int:
  if n == 0:
    return 0
  else:
    return x
```

   Y la siguiente suite de casos de test:

   - x = 0, resultado esperado: 0, resultado observado: 0
   - x = 1, resultado esperado: 1, resultado observado: 1

   Entonces tenemos un conjunto de casos de test que cubren todos los nodos y que producen el resultado esperado, pero claramente la función está implementada de manera incorrecta para los valores de $x < 0$.