

Titulació: Grau en Enginyeria Informàtica

Curs: Q2 2021–2022 (1r Parcial)

Assignatura: Programació 2 (PRO2)

Data: 4 de abril de 2022

Duració: 2h

1. **Reevaluación** [5 puntos]

Tenemos los resultados de la reevaluación de una asignatura de la FIB en una lista que contiene los DNIs de los estudiantes inscritos en la reevaluación y, para cada uno de ellos, un booleano que indica si ha aprobado o no la reevaluación de la asignatura.

Más específicamente, hemos definido el tipo reeval así

```
struct reeval {
    int dni;
    bool aprueba;
};
```

y tenemos una lista list<reeval> 1 con los resultados de la reevaluación que está ordenada crecientemente por el campo entero dni.

Con estos resultados queremos actualizar las notas de los estudiantes de la asignatura, que tenemos representadas en un vector de objetos de la clase Estudiant ordenado también por DNI, poniendo un 5.0 a los estudiantes que han aprobado la reevaluación. Aplicaremos así la normativa de reevaluación de la FIB, en la que para poder inscribirse en la reevaluación la nota original ha de estar entre 3.5 y 4.9, los estudiantes que aprueban la reevaluación pasan a tener un 5.0, y los que no la aprueban, mantienen su nota original.

Para ello, deseamos implementar la siguiente acción:

```
void aplica_reevaluacion(const list<reeval> &1, vector<Estudiant> &v)
// Pre: l ordenada crecientemente por el campo dni de sus elementos,
// v ordenado crecientemente por el DNI de sus estudiantes,
// v = V, todos los dni's incluidos en l representan estudiantes
// incluidos en V que tienen una nota entre 3.5 y 4.9
// Post: v es como V excepto que,
// para cada elemento de la lista l de la forma <d,true>,
la nota del estudiante de v con DNI d es ahora 5.0
```

Por ejemplo, si el contenido inicial de v es (con DNI's no realistas)

```
[<111,9.4>, <222,4.3>, <333,3.5>, <444,7.5>, <555,NP>, <666,4.0>, <777,3.7>]
```

y el contenido de 1 es

```
[<222,true>, <333,false>, <777,true>]
```

el contenido final de v tras ejecutar aplica_reevaluación(1,v) ha de ser

```
[<111,9.4>, <222,5.0>, <333,3.5>, <444,7.5>, <555,NP>, <666,4.0>, <777,5.0>]
```

Os recordamos la especificación Pre/Post de dos métodos de la clase Estudiant que es necesario usar en la implementación de aplica_reevaluacion:

```
int consultar_DNI() const;
// Pre: cierto
// Post: el resultado es el DNI del parámetro implícito

void modificar_nota(double nota);
// Pre: el parámetro implícito tiene nota, 0 <= nota <= nota_maxima()
// Post: la nota del parámetro implícito pasa a ser nota</pre>
Se pide:
```

- (a) (2.5 puntos) Implementar la acción aplica_reevaluacion iterativamente.
- (b) (2.5 puntos) Escribir el invariante y la funcion de cota de todos los bucles del apartado anterior y justificar su corrección.

SOLUCIÓ:

2. Diferencias con el preorden de un árbol binario [5 puntos]

Para este problema, diremos que un árbol binario a es completo (full, en inglés) si todos sus niveles están llenos de nodos. Sea tam(a) el número de elementos de un árbol a, donde tam es una función abstracta que no se puede usar dentro del código. Otras propiedades que cumple un árbol completo a son:

- todos los caminos desde la raíz a las hojas tienen la misma longitud
- si a es vacío entonces es completo
- si a no es vacío, entonces:
 - $(a) \ \mathsf{tam}(a) = 1 + \mathsf{tam}(a.\mathrm{left}()) + \mathsf{tam}(a.\mathrm{right}())$
 - (b) tam(a.left()) = tam(a.right()) = (tam(a)-1)/2
 - (c) todos los subárboles de a son completos

Se desea implementar la siguiente función usando una función de inmersión recursiva.

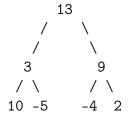
```
// Pre: a es un árbol completo, v.\operatorname{size}() = \operatorname{tam}(a)

// Post: el resultado es el número de diferencias (o errores) entre el recorrido en

// preorden de a y el recorrido de v en orden creciente de índice entre 0 y v.\operatorname{size}()-1

int num_errores(const BinTree<int> &a, const vector<int> &v)
```

Esto es, queremos saber cuántas veces el valor de un nodo de a no coincide con el valor de la posición de v correspondiente a ese nodo en preorden. Por ejemplo, si a es el siguiente árbol binario completo de enteros



y el vector v es [13 4 10 -5 9 -4 1], entonces v.size() = tam(a) = 7 y el resultado ha de ser $\text{num_errores}(a,v) = 2$, debido a los errores en las posiciones 1 y 6 del vector $(v[1]=4\neq 3 \text{ y } v[6]=1\neq 2)$.

Es obligatorio elegir una de las dos siguientes cabeceras para la función de inmersión:

• Opción 1:

• Opción 2:

Para la opción 1 es esencial tener en cuenta que el árbol dado a es completo, mientras que, para la opción 2, la solución desarrollada probablemente será válida (si es correcta) con cualquier árbol binario, no solo con aquellos que son completos.

Ninguna solución que use una inmersión distinta o altere la cabecera elegida se considerará válida. Asimismo, se valorarán muy negativamente las soluciones innecesariamente ineficientes en tiempo (si éste es mayor que proporcional al tamaño del árbol y del vector) o en espacio (por ejemplo, si se usara también algún vector auxiliar). Se pide:

- (a) (1 punto) Escribir la especificación Pre/Post de la función de inmersión elegida.
- (b) (2 puntos) Implementar la función de inmersión elegida.
- (c) (1.5 puntos) Justificar la corrección de la función de inmersión elegida incluyendo la demostración de que termina siempre.
- (d) (0.5 puntos) Implementar la función original (no recursiva) num_errores.

SOLUCIÓ: