Segundo examen parcial de PRO2 Modelo B Duración: 2h 50m 19/01/2018

Apellidos **Nombre** **DNI**

Problema 1 (6 puntos)

En este problema debéis implementar algunos métodos públicos de la clase *Llista*, cuya implementación habéis visto en clase de teoría. Mostramos a continuación la representación del tipo *Llista*, que utiliza nodos *doblemente encadenaos* con punteros al elemento siguiente (`seg`) y al elemento anterior (`ant`). Esta implementación de la clase *Llista* contiene los siguientes atributos: (1) `longitud`, de tipo entero; (2) `primer_node`, un puntero a `Node` que apunta al nodo que representa el primer elemento de la lista; (3) `ultim_node`, un puntero a `Node` que apunta al nodo que representa el último elemento de la lista; y (4) `act`, un puntero a `Node` que apunta al nodo que representa el elemento actual de la lista, denominado *el punto de interés* de la lista.

```
template <class T> class Llista {
  private:
    struct Node {
        T info;
        Node* seg;
        Node* ant;
    };
    int longitud;
    Node* primer_node;
    Node* ultim_node;
    Node* act;        ...   // especificación e implementación de operaciones privadas
  public:             ...   // especificación e implementación de operaciones públicas
};
```

En vuestras respuestas a este problema no podéis utilizar ningún método privado o público de la clase `Llista` que habéis visto en teoría (por ejemplo, `copia_node_llista`, `esborra_node_llista`, `afegir`, `eliminar`, `concat`, `inici`, `fi`, `avanca`, `retrocedeix`, `actual` o `modifica_actual`). Si utilizáis algún método privado o público auxiliar en vuestra respuesta a algún apartado, debéis especificarlo –escribiendo claramente la cabecera, la precondición y la postcondición– e implementarlo en dicha respuesta.

1.1 Definid el método público `pop_back()`, que elimina el último elemento de la lista parámetro implícito. Tened en cuenta que la lista parámetro implícito inicialmente no es vacía, pero puede quedar vacía después de aplicar esta operación. Por ejemplo, si a es $\{1,5,3,4,2\}$ y el punto de interés apuntaba a 1, después de la llamada `a.pop_back()`, a debe ser $\{1,5,3,4\}$ y el punt de interés debe apuntar a 1. [2 punts]

```
void pop_back();
/* Pre: El parámetro implícito es igual a la lista {e_1, ..., e_n} con n >= 1. */
/* Post: El parámetro implícito es igual a la lista {e_1, ..., e_{n-1}}. Si el punto de interés apuntaba
   a e_n, después de aplicar esta operación el punto de interés apuntará al final del parámetro implícito.
   Si el punto de interés no apuntaba a e_n antes de aplicar esta operación, seguirá apuntando al mismo
   lugar al que apuntaba antes de aplicar esta operación. */
```

1.2 Definid el método público `interseccio_ordenada`, que modifica el parámetro implícito de manera que contenga la intersección de las listas `c1` y `c2`. Por ejemplo, si *c* es la lista vacía, *a* es la lista $\{1,3,5,7,8,9,10\}$ y *b* es la lista $\{-1,1,2,3,7,8,9\}$, después de la llamada `c.interseccio_ordenada(a,b)`, *c* debe ser la lista $\{1,3,7,8,9\}$ y su punto de interés debe apuntar a 1. [4 puntos]

void *interseccio_ordenada* (**const** *Llista* & *c1*, **const** *Llista* & *c2*) {
/* Pre: El parámetro implícito es vacío. c1 y c2 están ordenadas crecientemente y no contienen elementos repetidos. */
/* Post: El parámetro implícito contiene los elementos que pertenecen a la intersección de c1 y c2 en el mismo orden en que están en c1 y c2. El punto de interés del parámetro implicito apunta a su inicio. c1 y c2 no cambian.*/

Segundo examen parcial de PRO2 Modelo B Duración: 2h 50m 19/01/2018

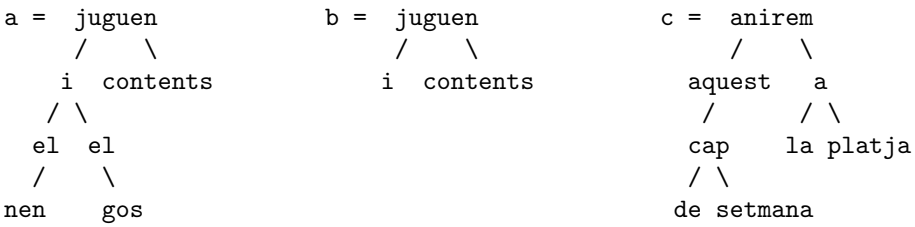
Apellidos **Nombre** **DNI**

Problema 2 (4 puntos)

Implementad eficientemente el método público `treu_subarbres`, especificado a continuación.

```
void treu_subarbres (const string& x);
/* Pre: El parámetro implícito es un árbol binario de string A. */
/* Post: Si x es el valor de algún nodo de A, el parámetro implícito es el resultado
   de eliminar de A todos los nodos con valor x y todos sus descendientes; en otro caso,
   el parámetro implícito no varía (es decir, es A). */
```

Por ejemplo, si t es igual al árbol a de la figura y x es "el", después de la llamada `t.treu_subarbres(x)`, t debe ser el árbol b de la figura. Del mismo modo, si s es el árbol c de la figura y z es "lo", después de la llamada `s.treu_subarbres(z)`, s no varía, es decir, s debe ser el árbol c de la figura.



Damos a continuación la definición del tipo `Arbre`, que debéis utilizar para resolver este problema.

```
template <class T> class Arbre {
  private:
    struct Node_arbre {
      T info;
      Node_arbre* segE;
      Node_arbre* segD;
    };
    Node_arbre* primer_node;
    ...                 // especificación e implementación de operaciones privadas
  public:
    ...                 // especificación e implementación de operaciones públicas
};
```

Si utilizáis algún método privado o público de la clase `Arbre` que habéis visto en clase de teoría (por ejemplo, `copia_node_arbre`, `esborra_node_arbre`, `a_buit`, `es_buit`, `arrel`, `plantar` o `fills`) en vuestra respuesta, debéis especificarlo –escribiendo claramente su cabecera, precondición y postcondición– e implementarlo en dicha respuesta.

Concretamente, se pide implementar eficientemente el método público `treu_subarbres` utilizando varios métodos privados auxiliares que trabajen directamente con datos de tipo `Node_arbre` y de tipo puntero a `Node_arbre`. Debéis

- Escribir la cabecera, la precondición y la postcondición de los métodos auxiliares.
- Implementar los métodos auxiliares.
- Implementar el método público `treu_subarbres` utilizando uno de los métodos auxiliares.

Observad que el método `treu_subarbres` debe liberar la memoria de todos los nodos que se eliminen del parámetro implícito.

void *treu_subarbres* (**const** *string*& *x*);
/* Pre: El parámetro implícito es un árbol binario de string A. */
/* Post: Si x es el valor de algún nodo de A, el parámetro implícito es el resultado
de eliminar de A todos los nodos con valor x y todos sus descendientes; en otro caso,
el parámetro implícito no varía (es decir, es A). */

Métodos auxiliares: especificación e implementación.