Problema 1

public:

```
void pop_back() {
/* Pre: El paràmetre implícit és igual a la llista {e_1, ..., e_n} amb n ≥ 1.*/
/* Post: El paràmetre implícit és igual a la llista {e_1, ..., e_{n-1}}. Si el punt d'interès apuntava a e_n,
després d'aplicar aquesta operació el punt d'interès apuntarà al final del parametre implícit (i.e. end()).
Si el punt d'interès no apuntava a e_n abans d'aplicar aquesta operació, seguirà apuntant al mateix
element al qual apuntava abans d'aplicar aquesta operació. */
  if (act == ultim_node) act = nullptr;
  Node* aux = ultim_node;
  ultim_node = ultim_node->ant;
  if (ultim_node == nullptr) primer_node = nullptr;
  else ultim_node->seg = nullptr;
  delete aux;
  longitud--;
}
```

```
void interseccio_ordenada (const Llista & c1, const Llista & c2) {
/* Pre: El paràmetre implícit és buit. c1 i c2 estan ordenades en ordre creixent
i no contenen elements repetits. */
/* Post: El paràmetre implícit conté els elements que pertanyen a la intersecció de
c1 i c2 en el mateix ordre en què estan en c1 i c2. El punt d'interès del paràmetre
implicit apunta al seu inici. c1 i c2 no canvien.*/
  Node* i1 = c1.primer_node;
  Node* i2 = c2.primer_node;
  while (i1 ≠ nullptr and i2 ≠ nullptr) {
    if (i1->info == i2->info) {
      Node* aux = new Node;
      aux->info = i1->info;
      if (primer_node == nullptr) {
        primer_node = aux;
        ultim_node = aux;
        act = aux;
      }
      else {
        ultim_node->seg = aux;
        aux->ant = ultim_node;
        ultim_node = aux;
      }
      longitud++;
      i1 = i1->seg;
      i2 = i2->seg;
    }
    else if (i1->info < i2->info) i1 = i1->seg;
    else i2 = i2->seg;
  }
  if (primer_node ≠ nullptr) {
    primer_node->ant = nullptr;
    ultim_node->seg = nullptr;
  }
}
```

Problema 2

```
#include <string>

public:

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 los nodos cuyo valor es x y todos sus descendientes;
en otro caso, el parámetro implícito no varía (es decir, es A). */
  if (primer_node ≠ nullptr)  treu_subjerarquias (primer_node, x);
}


private:

static void treu_subjerarquias (Node_arbre*& m, const string& x) {
/* Pre: m = M. M apunta a una jerarquía de nodos A no vacía. */
/* Post: Si x es el valor de algún nodo de A, m apunta a una jerarquía de
nodos que es el resultado de 'eliminar' de A los nodos cuyo valor es x y
todos sus descendientes; en otro caso, la jerarquía de nodos a la que apunta
m no varía (es decir, es A). */
    if (m→info == x) {
        esborra_node_arbre (m);
        m = nullptr;
    }
    else {
        if (m→segE ≠ nullptr) treu_subjerarquias (m→segE, x);
        if (m→segD ≠ nullptr) treu_subjerarquias (m→segD, x);
    }
}

// Implementar esborra_node_arbre o una operación con otro nombre que haga
// lo mismo que esborra_node_arbre.
```