**Problema 1.**

```
// Apartat 1
void nullify() {
  primer_node = ultim_node = act = nullptr;
  longitud = 0;
}



// Apartat 2
static node_llista* search(node_llista* p, const T& x) {
  while (p != nullptr and p->info != x)
     p = p->seg;
  return p;
}

// L'apartat 3 hi és a la pàgina següent.
```

```
// Apartat 3
void splice(const T& x, Llista& l2) {
  if (l2.primer_node != nullptr) { // si l2 es buida no cal fer res

    if (primer_node == nullptr) { // la llista implícita es buida
      primer_node = l2.primer_node;
      ultim_node = l2.ultim_node;
      longitud = l2.longitud;
      l2.nullify();
    }
    else {  // la llista implícita no es buida i l2 tampoc

      node_llista* p = search(primer_node, x);
      // p == nullptr i x no està a la llista o
      // p apunta a la primera aparició d'x en la llista implícita

      if (p == nullptr) {
        // l2 s'ha de transferir al final de la llista implicita
        // i per tant canvia l'ultim
        ultim_node->seg = l2.primer_node;
        l2.primer_node->ant = ultim_node;
        ultim_node = l2.ultim_node;
      }
      else if (p != primer_node) {
        // p != nullptr i p != primer_node
        // l2 s'ha de transferir a un punt intermig de la llista implicita
        // no canvia ni el primer ni l'últim
        node_llista* q = p->ant;
        q->seg = l2.primer_node;
        l2.primer_node->ant = q;
        p->ant = l2.ultim_node;
        l2.ultim_node->seg = p;
      }
      else {
        // p == primer_node != nullptr
        // l2 s'ha de transferir al principi de la llista implicita
        // i per tant canvia el primer
        primer_node->ant = l2.ultim_node;
        l2.ultim_node->seg = primer_node;
        primer_node = l2.primer_node;
      }
      longitud += l2.longitud;
      l2.nullify();
    }
  }
}
```

**Problema 2.**

Definim `node_arb_max`, mètode static privat de la classe Arbre, com a immersió d'`arbre_maxims`. Adaptant bé l'especificació d'`arbre_maxims` podrem simplificar molt l'especificació (i el codi) de `node_arb_max`.

Donada una jerarquia de nodes apuntada per un punter a `node_arbre`, definim la seva "jerarquia de màxims" de manera equivalent a l'arbre de màxims.

També farem servir la funció `max3(x,y,z)`, que retorna el màxim de 3 elements de tipus T.

```
// Pre: p != nullptr, i tot node de la jerarquia apuntada
//      per p té 0 o dos seguents diferents de nullptr
// Post: el resultat apunta a la jerarquia de maxims de la jerarquia
//       apuntada per p
static node_arbre* node_arb_max(node_arbre* p) {
  // per la Pre, com que p != nullptr, el resultat apuntarà com a minim
  // a un node
  node_arbre* n = new node_arbre;
  if (p->segE == nullptr) {
    // per la pre, p->segD tambe es igual a nullptr
    n->segE = n->segD = nullptr;
    n->info = p->info;
  } else {
    // p->segE != nullptr, p->segD != nullptr, podem fer crides recursives
    n->segE = node_arb_max(p->segE);
    n->segD = node_arb_max(p->segD);
    // HI: n->segE apunta a la jerarquia de maxims de la jerarquia
    //     apuntada per p->segE;  n->segD apunta a la jerarquia de maxims
    //     de la jerarquia apuntada per p->segD;
    n->info = max3(n->segE->info, n->segD->info, p->info);
  }
  return n;
}
```

Ara programem la función original fent servir la immersió. Noteu que la precondició d'`arbre_maxims` implica la de `node_arb_max`.

```
Arbre arbre_maxims(){
  Arbre a;
  a.primer_node = node_arb_max(primer_node);
  return a;
}
```