EXERCICI 1

APARTAT (a)

```
void Llista<T>::transferir(Llista<T>& dest)
/* Pre: el p.i. té un primer element x seguit d'una llista L; dest = D;
   el p.i. i dest són objectes diferents */
/* Post: el p.i. conté L; dest conté D seguida de x;
   el punt d'interès del p.i. no canvia si era damunt de L
   i si era damunt de x ara és damunt el primer element de L
   (que pot ser l'element fictici);
   el punt d'interès de dest no es modifica */
{

    node_llista* aux = primer;

    // modificar enllaços del p.i:
    if (act == primer) act = primer->seg;
    primer = primer -> seg;
    if (primer != NULL) primer->ant = NULL;
    else ultim == NULL;

    // modificar enllaços en el node propiament
    aux->ant = dest.ultim;
    aux->seg = NULL;

    // modificar enllaços en dest
    if (dest.primer == NULL) dest.primer = aux;
    else dest.ultim->seg = aux;
    dest->ultim = aux;

    --longitud;
    ++dest.longitud;

}
```

APARTAT (b)

Caixeta 1:

```
        alguna_no_buida = false;
```

Caixeta 2:

```
           if (v[i].longitud > 0) {
               v[i].transfer(*this);
               if (v[i].longitud > 0) alguna_no_buida = true;
           }
```

Es igualment correcte utilitzar .primer o .ultim != NULL
en comptes de .longitud > 0

Caixeta 3:

```
           en blanc
```

APARTAT (c)

Una manera:

- en comptes del booleà alguna_no_buida,
  guardar l'índex de la primera llista no buida que es troba
  per començar la següent iteració des d'allà des d'allà, i
  idem amb la última llista no buida per no seguir fins al final.
  Es fa fàcilment amb un parell de variables int addicionals.

  Aquesta solució estalvia força feina si hi ha llistes curtes al principi o al final,
  però en el cas pitjor no s'estalvia gran cosa (per exemple, si la primera i la última
  llista són molt llargues i totes les altres són buides)

La manera bona (i aquest és el nivell d'explicació que s'espera, més o menys).

- guardar una llista de les posicions de v amb llistes no buides
  i recorrer només aquesta llista en el for. Si un element de la llista conté el valor i,
  és que v[i] no és buida. Hem de transferir el primer element de v[i] a dest, i
  si després de fer això v[i] queda buida, cal fer l.erase

Cal fer un primer bucle extern posant tothom a dins de la llista
o bé que la primera iteració ja transfereixi elements però en comptes de fer
erases dels elements que queden buits faci inserts dels elements que no quedin buits.
Un booleà primera_iteracio va bé.

Això dona una solució d'eficiència proporcional a
V.size() + (suma de longituds de les llistes que hi ha a V)

EXERCICI 2

Solució a:

```
int iguals() const
 /* Pre: cert */
 /* Post: el resultat és el nombre de nodes de l'arbre $n$-ari del paràmetre
 implícit que tenen el mateix valor que el seu pare */
{
   if (primer_node == NULL) return 0;
   else return rec_iguals(primer_node);
}

int rec_iguals(node_arbreNari* m)
/* Pre: m != NULL */
/* Post: el resultat es el nombre de nodes de la jerarquia de nodes
         que comenca en el node apuntat per m que tenen el mateix
         valor que el seu pare */
{
   int r = 0;
   int s = m->seg.size();
   for (int i = 0; i < s; ++i) {
      if (m->seg[i] != NULL) {
         r += rec_iguals(m->seg[i]);
         if (m->info == m->seg[i]->info) ++r;
      }
   }
   return r;
}
```

Solució b), pitjor (més paràmetres, Post de la funció recursiva difícil d'expressar)

```
int iguals() const
 /* Pre: cert */
 /* Post: el resultat és el nombre de nodes de l'arbre $n$-ari del paràmetre
 implícit que tenen el mateix valor que el seu pare */
{
   if (primer_node == NULL) return 0;
   else return rec_iguals(primer_node->info, primer_node->seg);
}

static int rec_iguals (const T& w, const vector<node_arbreNari*>& v)
/* Pre: cert */
/* Post: el resultat es la suma del nombre de nodes de les jerarquies de nodes
         que comenca en els nodes apuntats des de v que tenen el mateix
         valor que el seu pare, mes els nodes apuntats des de v que tenen el valor w */
{
   int r = 0;
   int s = v.size();
   for (int i = 0; i < s; ++i) {
      if (v[i] != NULL) {
         r += rec_iguals(v[i]->info, v[i]->seg);
         if (w == v[i]->info) ++r;
      }
   }
   return r;
}
```

Solució c) Com la b), però passant només un punter cada vegada. Té els
mateixos inconvenients i requereix codi addicional, per exemple un bucle a
l'operació pública.