

Cognoms

Nom

DNI

OBSERVACIÓ: Cal fer servir els espais indicats per entrar la resposta. Penseu bé la vostra solució abans de començar a escriure-hi. Podeu ser penalitzats fins a 1 punt si heu de demanar un nou full d'examen perquè us heu equivocat, o si la solució és bruta o si ocupa espai important fora de les caixetes.

Noteu que algunes de les caixetes per a codi poden deixar-se en blanc si creieu que no cal cap instrucció en aquell punt.

Exercici 1 - Concatenació per nivells de llistes

(6 punts)

Tenim un vector v de $Llista<T>$, on T és un tipus qualsevol. Volem construir una mètode de la classe que construeixi una sola llista que contingui primer tots els primers elements de les llistes, després tots els segons elements de les llistes, després tots els tercers elements de les llistes, etc.

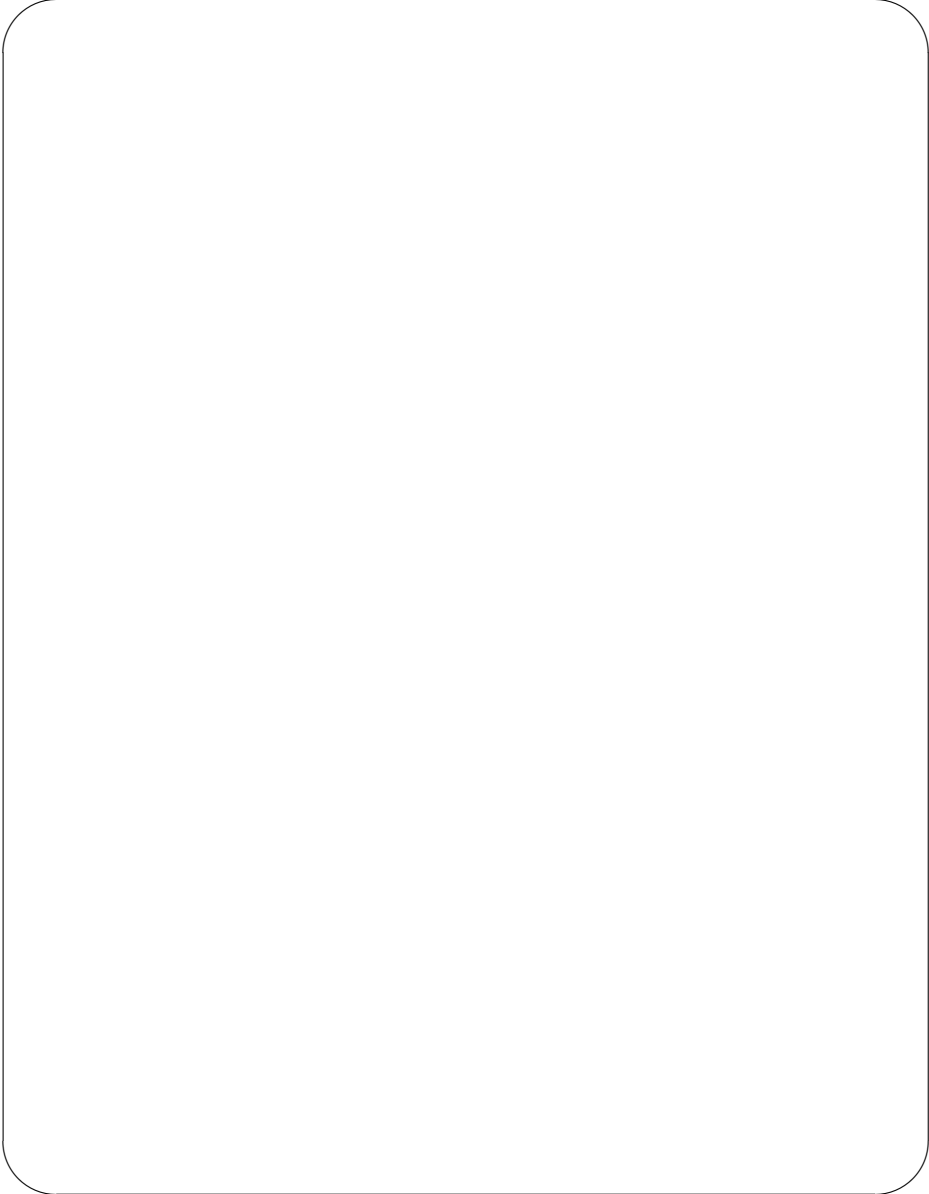
P.ex. si v té mida 5 i $v[0] = (a, b, c)$, $v[1] = (d, e, f, g)$, $v[2] = (h)$, $v[3]$ és buida i $v[4] = (x, y, z)$, al final la llista destí ha de contenir $(a, d, h, x, b, e, y, c, f, z, g)$ i v ha de contenir només llistes buides.

(a) (3 punts) Doneu una implementació d'un nou mètode de la classe $Llista$ anomenat $transferir$. Per exemple, si el paràmetre implícit és (a, b, c) i $dest$ és (x, y, z) , després de la crida el paràmetre implícit ha de ser (b, c) i $dest$ ha de ser (x, y, z, a) . Cal implementar l'operació accedint directament a la representació privada de la classe i no es pot usar cap operació pública de la classe.

Recordem que la representació privada del tipus $Llista<T>$ és

```
struct node_llista {
    T info;
    node_llista * seg;
    node_llista * ant;
};

int longitud;
node_llista * primer_node;
node_llista * ultim_node;
node_llista * act;
```

```
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 */
{

}

```

Cognoms

Nom

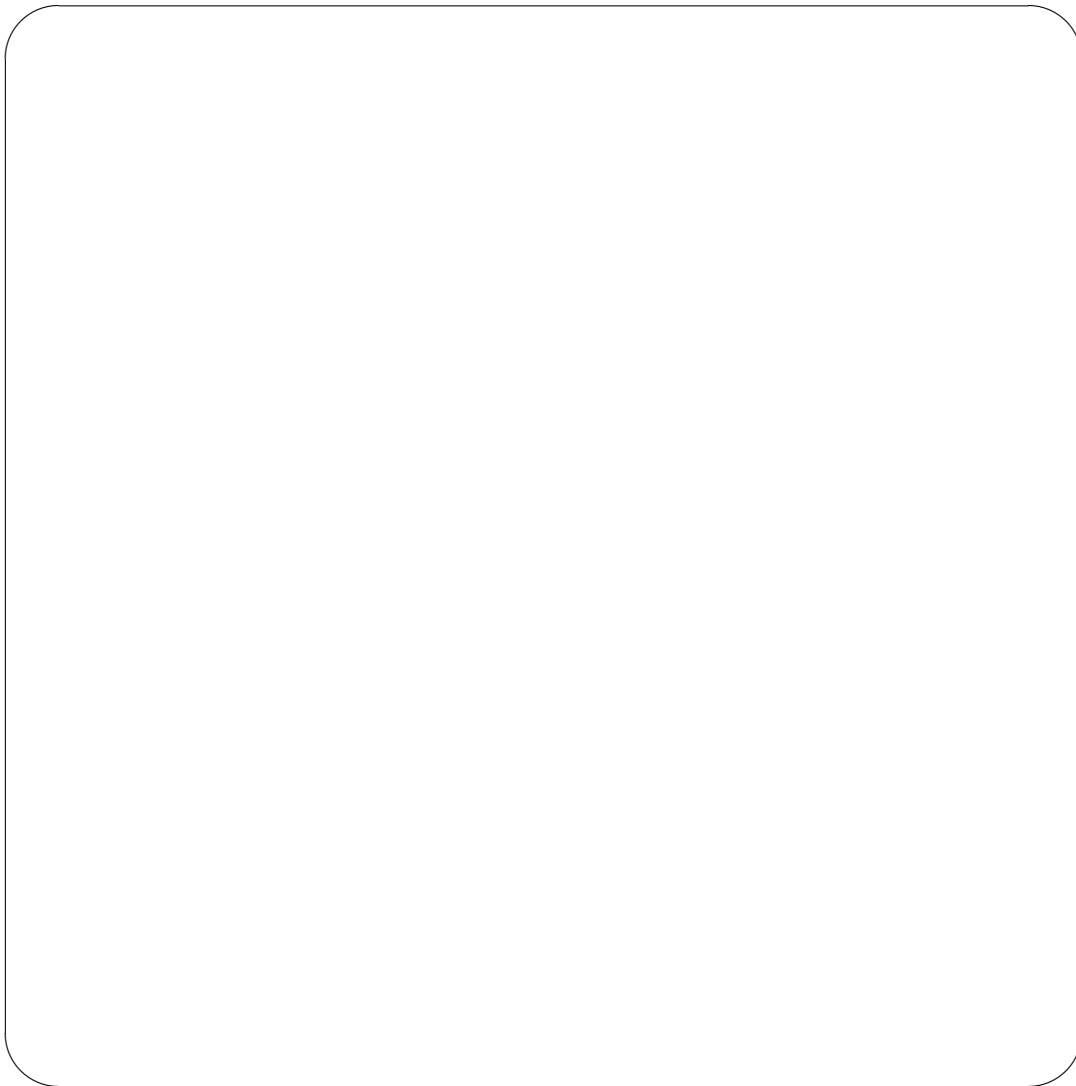
DNI

(b) (1 punt) Completeu el codi de la funció `concat_per_nivells` perquè satisfaci la seva especificació i respecti l'invariant donat. L'operació 1) és pròpia de la classe `Llista<T>`, 2) s'ha de programar accedint a la seva representació, 3) s'ha de basar en el mètode `transferir` de l'apartat (a) i 4) no pot fer servir cap altre mètode públic o privat de la classe.

```
void Llista<T>::concat_per_nivells (vector<Llista<T>>& v)
/* Pre: v = V, alguna llista de V no és buida, el p.i. és buit i és
       un objecte diferent de totes les llistes contingudes a v */
/* Post: v conté llistes buides i el p.i. conté la concatenació per nivells de
       les llistes de V */
{
    bool alguna_no_buida = true;
    while (alguna_no_buida) {
        /* Invariant: el p.i. conté, per a alguna k,  $k \geq 0$ , la concatenació
           dels k primers nivells de les llistes contingudes a V;
           aquests k primers nivells han estat esborrats de les llistes de v;
           alguna_no_buida és cert si i només si en aquest moment
           hi ha alguna llista no buida en v */
        
        for (int i = 0; i < v.size (); ++i) {
            
        }
        
    }
}
```

(c) (2 punts) La solució donada té l'inconvenient que farà un nombre d'operacions proporcional a $v.size() * (\text{longitud de la llista més llarga continguda a } v)$. Això és innecessàriament ineficient si, per exemple, v conté una sola llista molt llarga i moltes de curtes. Descriviu alguna solució més eficient que la proposada, *sense donar codi* però de forma prou precisa perquè quedi clara la implementació. Fem notar que n'hi ha una que té cost proporcional a $v.size() + (\text{suma de les longituds de les llistes contingudes a } v)$. És millor que doneu alguna millora, encara que no sigui aquesta solució tan òptima, que no pas que deixeu l'apartat en blanc.

Podeu fer servir estructures auxiliars com ara piles, cues o llistes d'int's, però **no** estructures addicionals que continguin elements de tipus T, ni en general crear, copiar, o modificar elements de tipus T.



Cognoms

Nom

DNI

Exercici 2 - Comptar nodes iguals al pare en arbre n-ari

(4 punts)

Llegiu-vos l'exercici complet abans de començar a resoldre'l.

Recordem primer que la representació privada del tipus `ArbreNari<T>` és

```
struct node_arbreNari {
    T info;
    vector<node_arbreNari*> seg;
};

int N;
node_arbreNari* primer_node;
```

Sigui `T` un tipus amb una operació d'igualtat `==` i considereu el problema següent: Donat un arbre n -ari, es vol saber quants nodes de l'arbre tenen un valor igual que el del seu pare. Per resoldre'l, volem implementar la funció següent dins de la classe `ArbreNari<T>`, accedint directament a la representació privada de la classe i sense usar cap operació pública de la classe.

```
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 */
{
    
}
}
```

Per tal que produeixi el resultat desitjat, cal dissenyar i usar una funció auxiliar recursiva que heu de completar també (a la pàgina següent). Ompliu amb codi els forats deixats en blanc en les dues funcions, així com els destinats a la Pre i la Post de la funció auxiliar.

```
static int rec_iguals (  )
/* Pre:  */
/* Post:

*/
{
  int r = 0;
  int s = 

  for (int i = 0; i < s; ++i) {

  }
  return r;
}
```