

Universitat Politècnica de Catalunya  
Facultat d'Informàtica de Barcelona

Cognoms, Nom

D.N.I.

[illegible]

**Titulació:** Grau en Enginyeria Informàtica

**Curs: Q1 2021–2022 (2n Parcial)**

**Assignatura:** Programació 2 (PRO2)

**Data:** 13 de gener de 2022

**Duració:** 2h 30m

1. **(5 punts)** Donada la següent representació d'una classe de llistes amb punt d'interès, doblement encadenades i sense sentinella

```
class Llista {
private:
    struct node_llista {
        int info;
        node_llista* seg;
        node_llista* ant;
    };
    int longitud;
    node_llista* primer_node;
    node_llista* ultim_node;
    node_llista* act;
    ... // especificació i implementació d'operacions privades

public:
    ... // especificació i implementació d'operacions públiques
};
```

es demana implementar el mètode **suavitza** amb la següent especificació:

```
void suavitzar();
/* Pre: La llista implícita conté almenys dos elements */
/* Post: S'ha afegit a la llista implícita un nou element al mig dels dos
elements consecutius que estan a una distància de valor més gran (en cas d'empat,
els que siguin més a prop de l'inici de la llista); el valor del nou element
és la (part entera de la) mitjana del valor d'aquests dos elements veïns;
el punt d'interès de la llista resultant passa a apuntar al nou element afegit */
```

Per exemple, si  $L = [12, 5, -3, -7, 9, 4, 10]$  llavors després d'executar `L.suavitza()` tindrem  $L = [12, 5, -3, -7, 1, 9, 4, 10]$ , ja que  $\{-7, 9\}$  són els dos elements consecutius que estan a una distància més gran (16) i la seva mitjana és 1.

El codi d'aquesta operació no ha de cridar a cap altre mètode públic o privat de la classe `Llista`, ni de cap altra classe, només consultar i modificar els atributs i els nodes de la llista implícita. Per a la vostra solució ompliu el codi que falta a les

capses indicades. Cada capsa ha de contenir exactament una instrucció simple o una expressió. S'ha de respectar el següent invariant del bucle:

```
/* Inv: Sigui L la llista implícita, distmax es la distància màxima de dos elements consecutius de
la subllista de L entre primer_node i ant, i primer_distmax apunta al primer d'ells;
antseg = ant->seg */
```

---

## SOLUCIÓ:

```
void Llista::suavitza(){
    node_llista* ant = primer_node ;
    node_llista* antseg = ant->seg ;
    int distmax = abs(antseg->info - ant->info) ;
    node_llista* primer_distmax = ant ;
    ant = antseg ;
    antseg = ant->seg ; // Aquí s'ha de complir l'Inv
    while (antseg != nullptr) { // també val ant != ultim_node
        int dist = abs(antseg->info - ant->info) ;
        if (dist > distmax) {
            distmax = dist ;
            primer_distmax = ant ;
        }
        ant = antseg ;
        antseg = ant->seg ;
    }
    node_llista* segon_distmax = primer_distmax->seg ;
    act = new node_llista ;
    act->info = (primer_distmax->info + segon_distmax->info)/2 ;
    primer_distmax->seg = act ;
    act->ant = primer_distmax ;
    act->seg = segon_distmax ;
    segon_distmax->ant = act ;
    ++longitud;
}
```

Alternativament, es poden deixar les dues capsas abans del `while` buides si les quatre primeres línies són les següents:

```
node_llista* ant = primer_node->seg ;
node_llista* antseg = ant->seg ;
int distmax = abs(ant->info - primer_node->info) ;
node_llista* primer_distmax = primer_node ;
```

2. (2,5 punts) Donada la següent definició d'una classe `Arbre` en C++

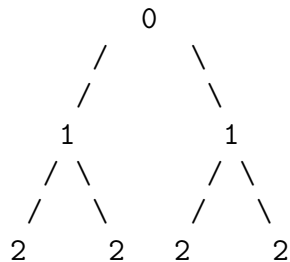
```
class Arbre{
private:
    struct node_arbre {
        int info;
        node_arbre* segE;
        node_arbre* segD;
    };
    node_arbre* primer_node; // apuntador a l'arrel de l'arbre
    ...

public:
    ...
};
```

es demana implementar el mètode `construeix_ple` amb la següent especificació:

```
void Arbre::construeix_ple(int k);
/* Pre: L'arbre implícit és buit i  $k \geq 0$  */
/* Post: L'arbre implícit és un arbre binari amb k nivells plens de nodes,
on cada node té com a valor la seva profunditat a l'arbre */
```

Per exemple, per  $k = 3$  l'arbre implícit resultant de l'operació és:



Per implementar `construeix_ple` haureu de fer servir aquesta immersió:

```
static node_arbre* Arbre::i_construeix_ple(int k, int prof);
/* Pre: k >= 0,  0 <= prof <= k */
/* Post: retorna un apuntador al node arrel (si existeix) d'un
arbre binari amb (k-prof) nivells plens de nodes,
on cada node té com a valor la seva profunditat a l'arbre + prof */
```

Ompliu les capses amb el vostre codi, respectant la resta del codi ja escrit. Cada capsa ha de contenir exactament una instrucció simple o una expressió. El codi no ha de cridar a cap altre mètode públic o privat de la classe **Arbre**.

---

## SOLUCIÓ:

```
// public:
//
void Arbre::construeix_ple(int k) {
    // Escriu aquí la teva implementació
    primer_node = i_construeix_ple(k,0);
}

// private:
//
static node_arbre* Arbre::i_construeix_ple(int k, int prof) {
    if (prof == k) { // també val prof >= k
        // Cas base
        return nullptr;
    }
    else {
        // Cas recursiu
        node_arbre* p = new node_arbre;
        p->info = prof;
        p->segE = i_construeix_ple(k,prof+1);
        p->segD = i_construeix_ple(k,prof+1);
        return p;
    }
}
```

3. (2,5 punts) Donada la següent definició d'una classe `ArbreGen` en C++

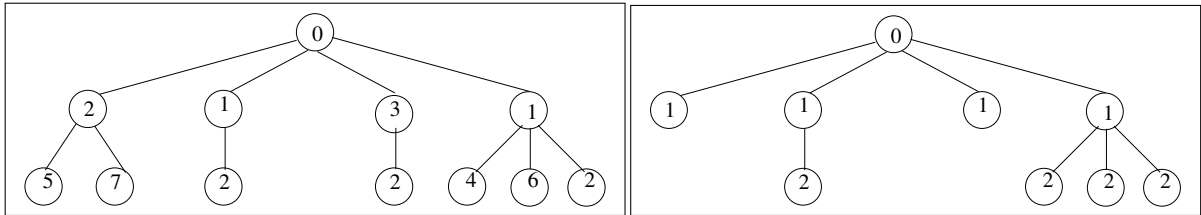
```
class ArbreGen{
private:
    struct node_arbreGen {
        int info;
        vector<node_arbreGen*> seg;
    };
    node_arbreGen* primer_node; // apuntador a l'arrel de l'arbre
    ...
    /* Pre: cert */
    /* Post no fa res si m és nullptr; en cas contrari, allibera
       espai de tots els nodes de la jerarquia que té el node
       apuntat per m com a arrel */
    static void esborra_node_arbreGen(node_arbreGen* m);

public:
    ...
};
```

es demana implementar un mètode públic **corregeix** amb la següent especificació:

```
void ArbreGen::corregeix();
/* Pre: L'arbre implícit = A i no és buit */
/* Post: L'arbre implícit és com A, però pels nodes x visitats en preordre
tals que el seu valor és diferent de la seva profunditat, s'ha substituït
el subarbre amb arrel x per una fulla amb valor la seva profunditat */
```

Per exemple, l'arbre de l'esquerra quedaria modificat com es veu a la dreta.



Per implementar **correg** heureu de fer servir aquesta immersió:

```
static node_arbreGen* ArbreGen::i_corregeix(node_arbreGen* p, int prof);
/* Pre: p apunta al node arrel d'una jerarquia de nodes d'un ArbreGen (p != nullptr)
   i prof és la profunditat associada al node apuntat per p */
/* Post: retorna un apuntador al node arrel d'una jerarquia de nodes d'un
   ArbreGen que és com l'apuntada per p, però pels nodes x visitats en preordre
   tals que el seu valor és diferent de la seva profunditat, s'ha substituït
   el subarbre amb arrel x per una fulla amb valor la seva profunditat */
```

Ompliu les capses amb el vostre codi, respectant la resta del codi ja escrit. Cada capsa pot contenir ara més d'una instrucció. El codi no ha de cridar a cap mètode públic de la classe `ArbreGen` i només pot cridar als mètodes privats `i_corregeix` i `esborra_node_arbreGen`.

---

## SOLUCIÓ:

```
// public:
//
void ArbreGen::corregeix() {
    // Escriu aquí la teva implementació
    primer_node = i_corregeix(primer_node, 0);
}

// private:
//
static node_arbreGen* ArbreGen::i_corregeix(node_arbreGen* p, int prof) {
    if (p->info != prof) {
        // Cas base (calia omplir la capsa amb les 3 línies següents)
        esborra_node_arbreGen(p);
        p = new node_arbreGen;
        p->info = prof;
    }
    else {
        // Cas recursiu (calia omplir la capsa amb les 2 línies següents)
        for (int i = 0; i < p->seg.size(); ++i)
            p->seg[i] = i_corregeix(p->seg[i], prof+1);
    }
    return p ;
}
```