

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 podeu deixar en blanc alguna de les caixetes si creieu que no cal cap instrucció en aquell punt.

Problema 1 (6 punts)

Diem que un element és un *màxim local* en una llista si és més gran que tots els elements que el precedeixen. Fixeu-vos que el primer element d'una llista no buida sempre és doncs màxim local.

Ens donen una llista d'ints, i volem reordenar-la de manera que primer contingui tots els màxims locals i després tots els elements que no són màxims locals. En tots dos trams cal mantenir l'ordre original dels elements. Per simplificar suposem que la llista donada no és buida.

Per exemple, si la llista és

[-5, -6, -2, 2, 0, -1, 4, 2, 2, 6, 4, 8, 9, 7, 10, 6],

es vol que quedi

[-5, -2, 2, 4, 6, 8, 9, 10, -6, 0, -1, 2, 2, 4, 7, 6].

La capçalera de la funció desitjada és:

```
/* Pre:  l = L, l.size() > 0 */
/* Post: l conté primer els elements que són màxims locals de L, seguits dels elements que no
són màxims locals de L, respectant l'ordre original en cadascun dels dos trams */
void arregla ( list <int>& l );
```

I ho volem fer amb un algorisme iteratiu on el bucle manté aquest invariant:

Inv: 1) si `it2 == l.end()` llavors `l` és com a la Post.
Altrament, 2) `it1` i `it2` apunten a elements de `l`,
3) els elements anteriors a `it1` són els màxims locals
de `L` que apareixien abans que `it2` en `L`, en el seu ordre relatiu original,
4) els elements entre `it1` (inclòs) i anteriors a `it2`
són els no màxims locals de `L` que apareixien abans que `it2` en `L`,
en el seu ordre relatiu original,
5) `max` és el màxim dels elements que eren abans que `it2` en `L`.

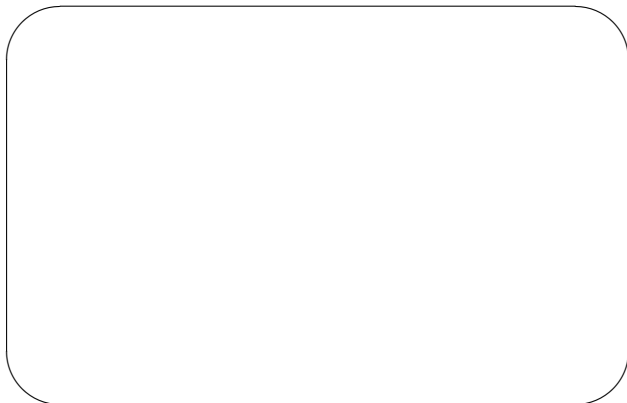
Per exemple, amb la llista de l'exemple, a l'inici d'alguna de les iteracions de l'algorisme la llista `l` seria

[-5, -2, 2, 4, 6, -6, 0, -1, 2, 2, 4, 8, 9, 7, 10, 6],

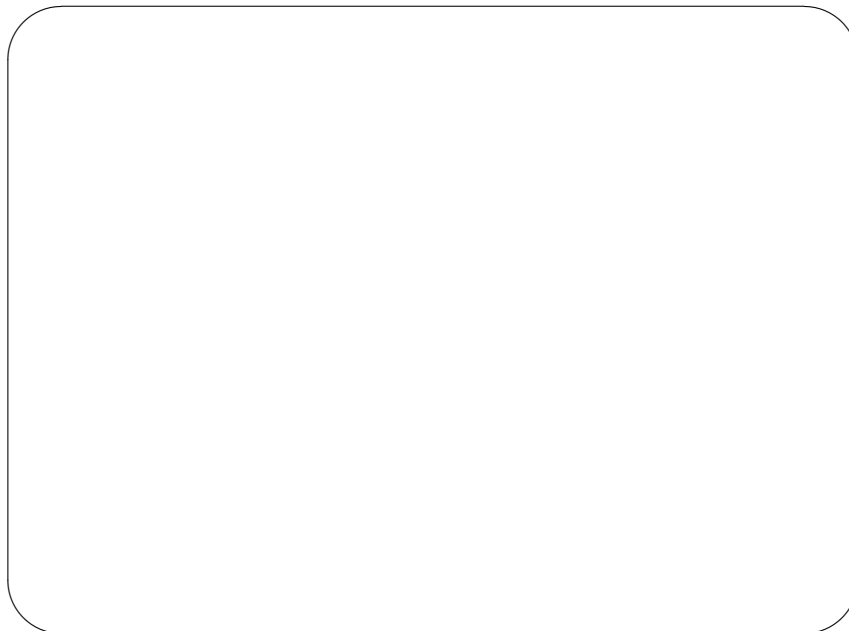
amb `it1` apuntant a l'element -6 i `it2` apuntant a l'element 8, que seria el següent element a tractar.

1.1 (4 punts) Completeu l'algorisme següent de manera que satisfaci l'invariant Inv. Noteu que les variables `it1`, `it2` i `max` no s'inicialitzen en el codi ja donat.

```
void arregla ( list <int>&l) {  
    list <int>::iterator it1 , it2 ;  
    int max;
```



```
    while (it2 != l.end()) {
```



```
    }  
}
```

Cognoms

Nom

DNI

1.2 (2 punts) Justifiqueu que el codi de la segona caixa manté les parts 3, 4, 5 d'Inv.

(pàgina en blanc intencionadament)

Cognoms

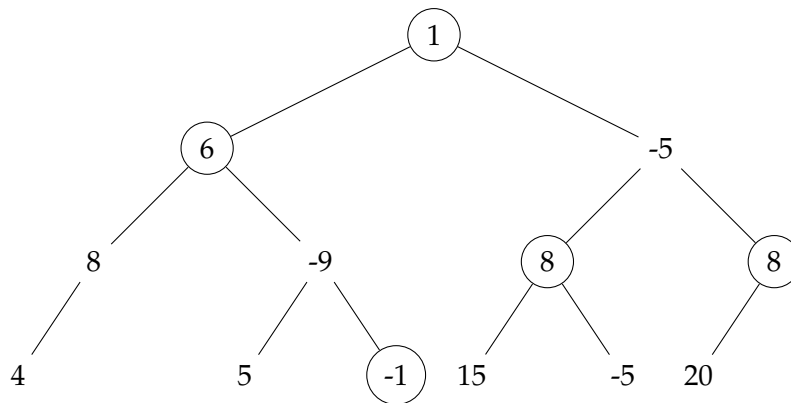
Nom

DNI

Problema 2 (4 punts)

En un arbre binari d'enters diem que un node és *ponderat* si és més gran que la suma dels seus ascendents i més petit que la suma dels seus descendents. Per definició, la suma dels ascendents de l'arrel i la suma de descendents d'una fulla són 0.

Per exemple, al següent arbre són nodes ponderats els nodes encerclats.



La fulla -1 és ponderada perquè és més gran que la suma dels seus ascendents (-2) i més petita que la suma dels seus descendents (0). El node -9 no és ponderat perquè, malgrat que és més petit que la suma dels seus descendents (4), és més petit que la suma dels seus ascendents (7). Tampoc no ho és el 8 de l'esquerra del tot perquè, malgrat ser més gran que la suma dels seus ascendents (7), no és més petit que la suma dels seus descendents (4).

Volem donar una implementació eficient de la funció:

```
int nombre_ponderats(Arbre<int>& a);  
/* Pre: a = A */  
/* Post: el resultat és el nombre de nodes ponderats d'A */
```

Es proposa una implementació incompleta d'una funció d'immersió d'aquesta. Tingueu en compte que:

- És poc probable que resolgueu el problema bé si no comenceu per entendre per què cadascun dels nodes de l'exemple és o no és ponderat.
- La immersió conté tant un reforçament de la Pre com un afebliment de la Post, i en concret s'afegeix un paràmetre d'entrada i un de sortida.
- És impossible resoldre el problema si no penseu bé primerament la Pre, la Post, i els paràmetres d'immersió.
- Si no us surt la solució sencera a la primera, penseu primer com ho faríeu si la condició de ponderat fos únicament ser més gran que la suma dels antecessors i després penseu què us falta per incorporar l'altra condició.
- Una solució bona tarda temps lineal en la mida de l'arbre, i en visita cada node exactament un cop.

/* Pre:

*/

/* Post:

*/

```
int i_nombre_ponderats(Arbre<int>& a, int sum_asc, int& sum_desc) {
```

```
    if (a.es_buit ()) {
```

```
        return 0;
```

```
    } else {
```

```
        int arr = a.arrel ();
```

```
        Arbre <int> fe, fd;
```

```
        if ( ) {
```

```
            return ;
```

```
        } else {
```

```
            return ;
```

```
        }
```

```
    }
```

```
}
```

Doneu el codi de *nombre_ponderats*, que crida a *i_nombre_ponderats*:

```
int nombre_ponderats(Arbre<int>& a) {
```

```
}
```