

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.

## Problema 1 (5 punts)

Considerem la representació habitual amb nodes de la classe *Pila* per manegar piles genèriques de tipus T:

```
template <class T> class Pila {
private:
    struct node_pila {
        T info;
        node_pila * seg;
    };
    int longitud;
    node_pila * primer;
    ... // especificació i implementació d'operacions privades
public:
    ... // especificació i implementació d'operacions públiques
};
```

### Apartat 1.1 (2.5 punts)

Volem afegir una operació sobre piles d'enters denominada *subintervals* amb la següent especificació pre/post:

```
void subintervals (int dif);
/* Pre: El p.i. conté una pila d'enters ordenada creixentment, el paràmetre
dif és un enter positiu.
Post: El p.i. és una pila on tota subseqüència maximal de nombres on la
diferència entre valors consecutius és menor o igual que dif ha estat
substituïda pel primer i l'últim de la subseqüència
*/
```

Per exemple, amb la crida *subintervals(1)* i donada la pila :

1 2 3 5 6 9 10 11 16 17 18 20 21 27 30 34 37

on l'element 1 és el top de la pila, obtindríem:

1 3 5 6 9 11 16 18 20 21 27 30 34 37

i amb la crida *subintervals(4)* obtindríem:

1 11 16 21 27 37

Ens donen la següent implementació incompleta en la qual heu d'acabar d'omplir els forats:

```
void subintervals (int dif) {  
    node_pila * api;  
    node_pila * apf;  
    if (primer != NULL) {  
  
          
  
        while (apf != NULL) {  
            if (api != apf and api->seg != apf) {  
                node_pila * aptmp;  
  
                aptmp =   
  
  
  
  
                --longitud;  
            }  
            if (apf -> seg == NULL or  
                  
                ) {  
                  
  
            }  
            apf = apf -> seg;  
        }  
    }  
}
```

Cognoms

Nom

DNI

--	--	--

### Apartat 1.2 (2.5 punts)

Ens donen una implementació d'una nova operació pública de la classe Pila anomenada `primer_de_cada_n`, amb la següent especificació:

```
void primer_de_cada_n(Pila &p, int n);  
/* Pre: el p.i. conté una pila, el paràmetre n és un enter  $\geq 2$  i p es la pila buida.  
   Post: el p.i. és una pila on els elements que són el primer de cada n han  
   estat eliminats i la pila p conté el primer element de cada n del p.i.  
*/
```

La implementació que ens donen és la següent:

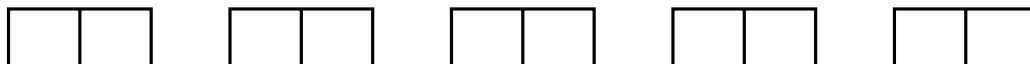
```
void primer_de_cada_n(Pila& p, int n) {  
    if (primer  $\neq$  NULL) {  
        node_pila* ap1;  
        node_pila* ap2;  
  
        p.primer = primer;  
        ap2 = p.primer;  
        primer = primer  $\rightarrow$  seg;  
        ap1 = primer;  
        int cmpt = 1;  
  
        while (ap1  $\neq$  NULL) {  
            while ((cmpt < (n - 1))) {  
                ap1 = ap1  $\rightarrow$  seg;  
                ++cmpt;  
            }  
            if (ap1  $\rightarrow$  seg == NULL) {  
                ap2  $\rightarrow$  seg = NULL;  
                ap1 = ap1  $\rightarrow$  seg;  
            } else {  
                ap2  $\rightarrow$  seg = ap1  $\rightarrow$  seg;  
                ap2 = ap2  $\rightarrow$  seg;  
                ap1  $\rightarrow$  seg = ap2  $\rightarrow$  seg;  
                ap1 = ap1  $\rightarrow$  seg;  
            }  
            cmpt = 1;  
        }  
    }  
}
```

Emplena el gràfic següent indicant els valors i encadenaments dels nodes i els atributs primer i longitud del paràmetre implícit i la pila p (pels encadenaments i primer dibuixeu la fletxa de l'encadenament o poseu NULL segons calgui) que donaria aquesta implementació amb paràmetre  $n = 2$  i la pila d'enters:

1 2 3 4 5

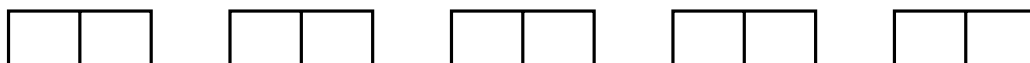
Paràmetre Implícit

Primer:	Longitud:
---------	-----------



Paràmetre p

Primer:	Longitud:
---------	-----------



Si no dóna cap resultat o produeix error d'execució, explica on i per què:

Emplena ara el gràfic següent suposant que tenim la mateixa pila però  $n = 3$ :

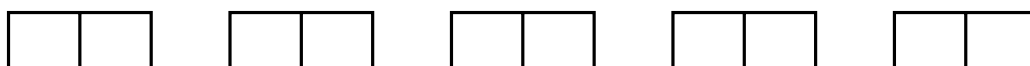
Paràmetre Implícit

Primer:	Longitud:
---------	-----------



Paràmetre p

Primer:	Longitud:
---------	-----------



Si no dóna cap resultat o produeix error d'execució, explica on i per què:

Cognoms

Nom

DNI

## Problema 2 (5 punts)

Sigui  $T$  un tipus en què hi ha definit un ordre amb les operacions de comparació habituals ( $==$ ,  $<$ ,  $<=$ , etc.). L'arbre de cerca associat a un conjunt de  $T$  es defineix així:

- L'arbre associat a un conjunt buit és l'arbre buit.
- L'arbre associat a un conjunt  $C$  no buit té a l'arrel un element  $x$  del conjunt tal que la meitat dels elements de  $C - \{x\}$  són més petits que  $x$ , i l'altra meitat són més grans que  $x$ ; si  $C$  té cardinalitat parella, això no és exactament possible, i llavors el segon conjunt té exactament un element més que el primer. El fill esquerre de l'arbre és l'arbre de cerca dels elements de  $C$  més petits que  $x$ , i el fill dret és l'arbre de cerca dels elements de  $C$  més grans que  $x$ .

Recordem la implementació dels arbres binaris:

```
template <class T> class Arbre {  
    private:  
        struct node {  
            T info;  
            node* segE; // fill esquerre  
            node* segD; // fill dret  
        };  
        node* primer;  
        ... // especificació i implementació d'operacions privades  
    public:  
        ... // especificació i implementació d'operacions públiques  
};
```

Per als apartats següents tingueu en compte que:

- Cal resoldre'ls accedint a la implementació; no es poden usar operacions públiques de la classe.
- Podeu fer servir el procediment `sort()` de vectors.
- Es permet fer servir més d'un `return` en una funció si el codi queda clar.
- Es valorarà molt l'eficiència.

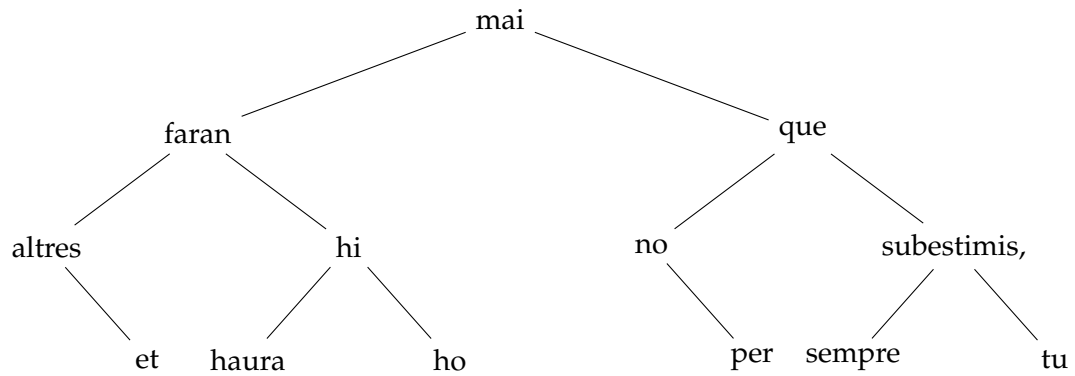
### Apartat 2.1 (2 punts)

Volem fer un mètode de la classe `Arbre<T>` que retorni l'arbre de cerca associat a un conjunt de  $T$  donat com a vector. Del vector es pot suposar únicament que no té elements repetits.

Per exemple, si  $T$  és `string` i el vector donat és

```
[ "mai" "no" "et" "subestimis," "sempre" "hi" "haura" "altres" "que" "ho" "faran" "per" "tu" ]
```

el mètode hauria de deixar en el paràmetre implícit l'arbre



Resoleu el problema omplint les capses donades de la funció `cons_arbre_cerca` i la seva funció d'immersió `i_cons_arbre_cerca`:

*/\* Pre: El paràmetre implícit és l'arbre buit; v = V i no te elements repetits \*/*  
*/\* Post: El paràmetre implícit conté l'arbre de cerca de V \*/*

**void** `cons_arbre_cerca` (*vector*<T>& v) {

}

*/\* Pre:* *\*/*

*/\* Post:* *\*/*

`i_cons_arbre_cerca`(**const** *vector*<T>& v, **int** i, **int** j) {

**if** (  ) {  
     **return** *NULL*;  
**}** **else** {

    }  
 }

**Cognoms**

**Nom**

**DNI**

### Apartat 2.2 (2 punts)

Volem fer un mètode de la classe `Arbre<T>` que digui si un element donat  $x$  és o no és a un conjunt que ens donen com a arbre de cerca. Cal aprofitar fortament que l'arbre té l'estructura descrita abans per assegurar l'eficiència.

Resoleu el problema omplint les capses donades de la funció `cerca` i la seva funció d'immersió `i_cerca`:

`/* Pre: El paràmetre implícit és un arbre de cerca d'algun conjunt C */`

`/* Post: El resultat diu si x pertany a C */`

`bool cerca(const T& x) {`

`}`

`/* Pre:`  `*/`

`/* Post:`  `*/`

`static bool i_cerca(` `, const T& x) {`

`if (`  `)`

`return` `;`

`else if (`  `)`

`return` `;`

`else if (`  `)`

`return` `;`

`else`

`return` `;`

`}`

### Apartat 2.3 (1 punt)

Expliqueu per què el nombre de comparacions entre elements de tipus T que farà cerca amb un arbre de mida  $N$  serà aproximadament  $2 \log_2 N$ , si l'element buscat no és a l'arbre.

(Nota: això és en la solució que s'espera. Si no és el cas per a la vostra solució, digueu quantes en faria i per què).