Segon examen parcial de PRO2 **Durada: 2h 50m** **22/06/2018**

Cognoms **Nom** **DNI**

Problema 1 (5 punts)

Volem construir una estructura de dades lineal i simplement encadenada, anomenada `Memoria`, que doni suport a un sistema operatiu en la gestió de la memòria dinàmica.

En el nostre sistema, els programes fan peticions de memòria al sistema amb un paràmetre b que és la quantitat de bytes que necessiten. El sistema busca una adreça de memòria a partir de la qual hi hagi b bytes consecutius disponibles, els reserva i retorna al programa demandant un identificador per al bloc de memòria reservat. Els identificadors es donen de forma successiva, 0, 1, 2, . . . , a mesura que hi ha peticions.

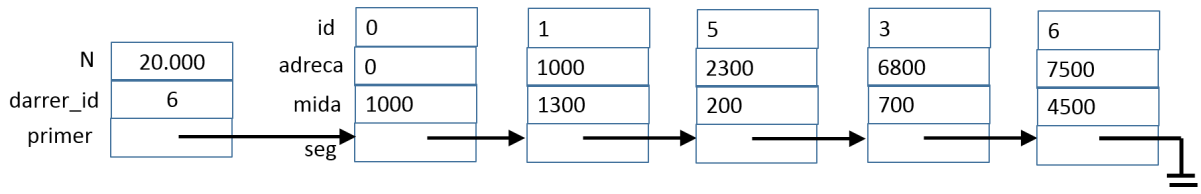
La informació es guarda en una seqüència de nodes cadascun dels quals conté la informació d'un bloc de memòria concedit. Un `node` conté l'identificador que se li ha donat al bloc de memòria que representa aquest node, l'adreça de memòria on comença el bloc, el nombre de bytes que el formen i un apuntador al següent node. Important és que **en la nostra implementació els nodes es mantenen ordenats de manera creixent per adreça de memòria** (no per identificador).

Un objecte de la classe `Memoria` conté: 1) un atribut N que és la mida de la memòria que gestiona i que s'assigna en crear l'objecte; 2) el darrer identificador assignat fins ara, i 3) un apuntador al primer bloc. Les adreces de memòria van des de 0 a $N - 1$. La representació del tipus `Memoria` en C++ és aquesta:

```
class Memoria {

  private:
    struct Node {
        int id;
        int adreca;
        int mida;
        Node* seg;
    };

    Node* primer;   // apuntador a primer node
    int N;          // la màxima adreça de memòria possible es N-1
    int darrer_id ; // darrer identificador lliurat
      ... mètodes privats ....

  public:
      ... mètodes públics ....
};
```

Hi ha un bloc de memòria especial, que és el bloc que el sistema operatiu es guarda per a ell mateix al principi de l'execució. Comença a l'adreça 0 (i, per tant, sempre esta representat pel primer node), té identificador 0 i no s'esborra mai.

Per exemple, la figura següent mostra el contingut (o estat) d'un objecte t de tipus `Memoria` en un moment determinat. En aquest moment hi ha 5 blocs reservats.



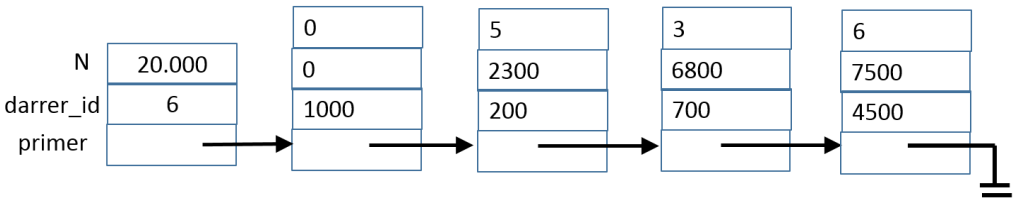
El primer bloc comença a l'adreça 0, ocupa 1000 bytes i té identificador 0. El segon bloc comença a l'adreça 1000, ocupa 1300 bytes i té identificador 1. El tercer bloc comença a l'adreça 2300, ocupa 200 bytes i té identificador 5. Hi ha un forat de 4300 bytes lliures entre aquest bloc i el següent, que té identificador 3, comença a l'adreça 6800 i ocupa 700 bytes. A continuació existeix un darrer bloc, que té identificador 6, comença a l'adreça 7500 i ocupa 4500 bytes. Després hi ha un forat amb memòria lliure entre la posició 12000 i la darrera posició, que té l'adreça 19999.

L'identificador del darrer bloc concedit és el 6. Això vol dir que hi ha dos identificadors, el 2 i el 4, que s'han d'haver esborrat de t (és a dir, alliberats) abans del moment que mostra la figura.

Concretament, en aquest exercici, es demana implementar eficientment els mètodes públics **demanar** i **alliberar** de la classe `Memoria`, especificats a continuació:

```
void alliberar (int x);
/* Pre: x > 0. */
/* Post: Si el paràmetre implícit (p.i.) contenia un bloc de memòria amb identificador x,
la memòria que ocupava el bloc amb identificador x ha estat alliberada y l'identificador x
ha deixat de ser vàlid, perquè el p.i. ja no conté un bloc de memòria amb identificador x. */
```

Per exemple, si `t` és un objecte de tipus `Memoria` amb el contingut mostrat en la figura anterior, després de la crida `t.alliberar(1)` el contingut de t hauria de ser el següent:




```
int demanar(int b);
/* Pre: b > 0
/* Post: Si no hi ha b bytes consecutius lliures al p.i., el resultat és −1; en un altre cas, es
reserva un bloc format pels primers b bytes consecutius lliures (és a dir, no reservats) del p.i.,
se li assigna com a identificador el enter següent a l'últim identificador assignat i es retorna
aquest identificador com a resultat. */
```

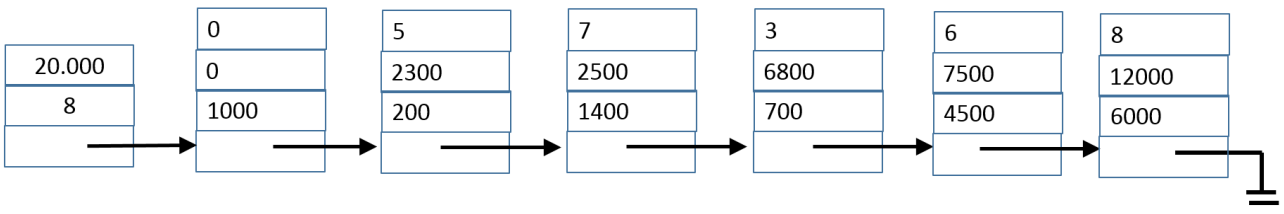
Per exemple, si `t` és un objecte de tipus `Memoria` amb el contingut mostrat en la figura anterior, després de la seqüencia de crides següent

```
t.demanar(1400);
t.demanar(6000);
```

el resultat hauria de ser

```
7
8
```

i el contingut del objecte de tipus `Memoria` t el següent:



Restriccions del problema:

- Cal respondre dins de l'espai donat, respectant l'estructura que es proposa, però no és imprescindible omplir totes les caixes.
- No es poden fer servir estructures de dades auxiliars (com ara `piles`, `cues` o `llistes`). Heu de treballar directament amb dades de tipus `Memoria`, `Node` i punter a `Node`.
- Els mètodes públics **demanar** i **alliberar** haurien d'examinar una vegada com a màxim cada node del paràmetre implícit.

Cognoms **Nom** **DNI**

1.1 Implementació de **alliberar**. [**2 punts**]

void *alliberar* (**int** *x*) {
Node∗ *p* = *primer*;

while (   ) *p* = *p*→ *seg*;

}

1.2 Implementació de **demanar**. [**3 punts**]

int *demanar*(**int** *b*) {
Node∗ *p* = *primer*;

while (   ) {
p = *pseg*;
pseg = *pseg*→*seg*;
}

if (   ) **return** −1;
Node∗ *aux* = **new** *node*;

return *darrer_id* ; }

Cognoms	Nom	DNI

Problema 2 (5 punts)

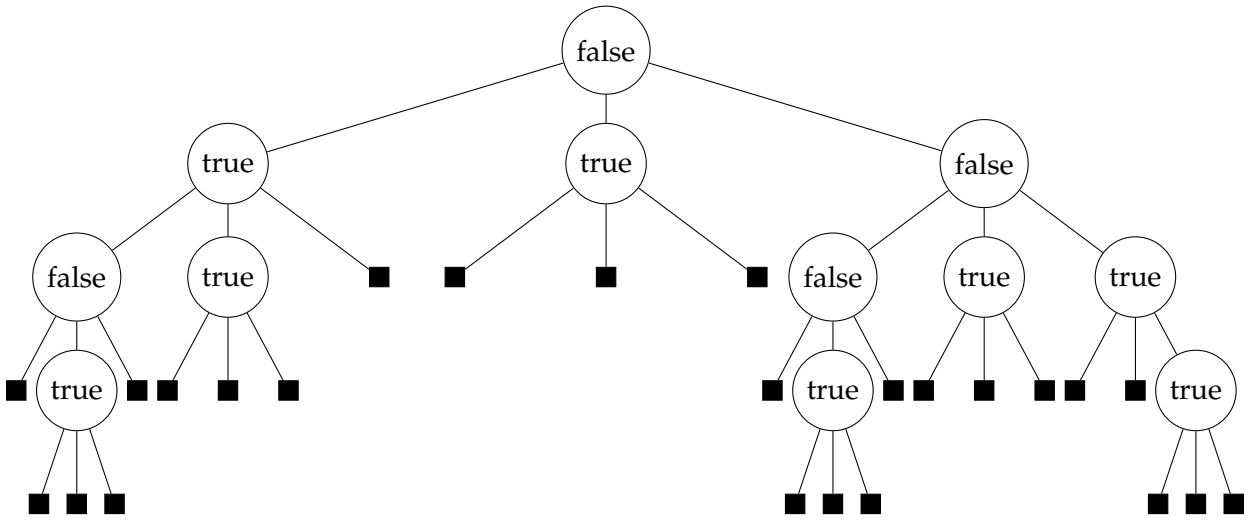
En aquest exercici utilitzarem la classe *ArbreNari*, vista a la classe de teoria, per representar conjunts de paraules formades únicament per lletres minúscules de l'alfabet anglès. Concretament, per representar un conjunt de paraules $\mathbf{s} = \{p_1, \ldots, p_n\}$ utilitzarem un *arbre N-ari de booleans* **t** amb aritat $N = 26$. Aquest arbre 26-ari de booleans es defineix de la manera següent:

1. El conjunt de paraules buit (és a dir, $\{\}$) es representa mitjançant l'arbre buit.
2. Si **s** és un conjunt de paraules no buit i **s** conté la paraula buida (és a dir, si hi ha un $k \in \{1, \ldots, n\}$ tal que $p_k = ""$), el camp *info* del node arrel de **t** és igual a *true*.
3. Si **s** és un conjunt de paraules no buit i **s** no conté la paraula buida (és a dir, si no hi ha un $k \in \{1, \ldots, n\}$ tal que $p_k = ""$), el camp *info* del node arrel de **t** és igual a *false*.
4. Cada node de l'arbre **t** amb **info** igual a **true** representa una paraula del conjunt **s**. En particular, el nombre d'elements del conjunt **s** és igual al nombre de nodes de l'arbre **t** amb **info** igual a **true**.
5. Per a tot $i = 0, \ldots, 25$, el fill i-èsim de **t** conté el subconjunt de paraules de **s** que comencen per la lletra i-èsima de l'alfabet anglès (és a dir, pel caràcter resultant d'avaluar l'expressió `char('a' + i)`).

Pel que fa al punt 5 de la definició anterior, cal aclarir que:

- En aquest exercici enumerem els fills d'un arbre 26-ari de booleans **t** de 0 a 25, en lloc d'1 a 26, per similitud amb la notació de vectors que s'utilitza per representar els punters que apunten a aquests fills en la representació del tipus *Node* de la classe *ArbreNari*. De la mateixa manera, enumerem les lletres de l'alfabet anglès de 0 a 25, de manera que la lletra 0-èsima correspon al caràcter `'a'`, la lletra 1-èsima correspon al caràcter `'b'`, ... i la lletra 25-èsima al caràcter `'z'`.
- Si el conjunt **s** no conté cap paraula que comenci per la lletra i-èsima de l'alfabet anglès, el fill i-èsim de **t** és buit.
- Si denotem amb $\mathbf{s_i}$ el subconjunt de paraules de **s** que comencen per la lletra i-èsima de l'alfabet anglès i $\mathbf{s_i}$ és no buit, llavors el fill i-èsim de **t** conté el conjunt de paraules resultant d'eliminar el primer caràcter de totes les paraules de $\mathbf{s_i}$.

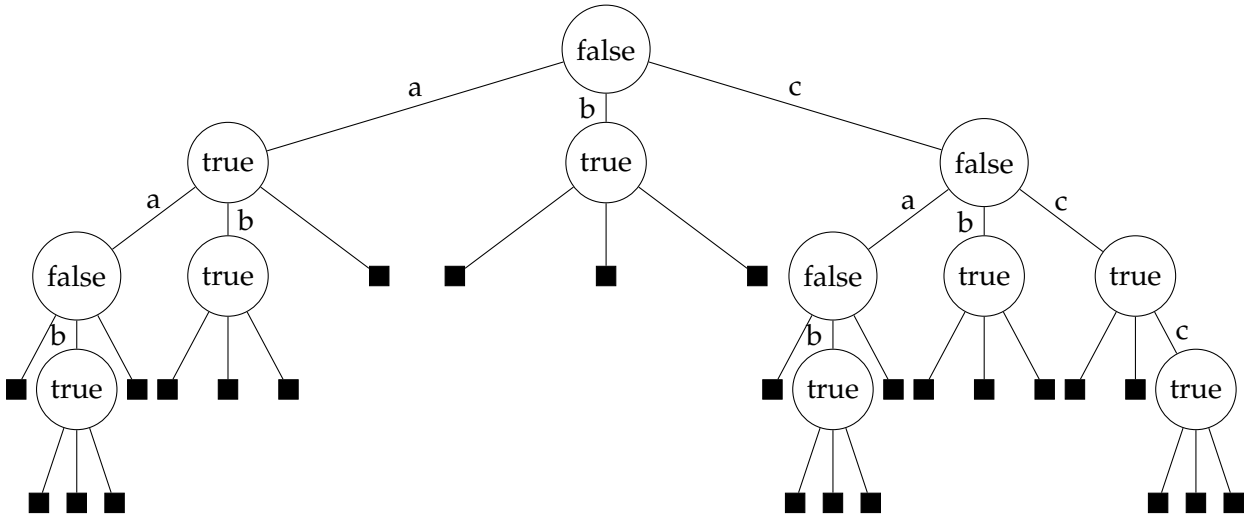
En l'exemple següent, considerem únicament conjunts de paraules formades per les tres primeres lletres de l'alfabet (és a dir, $\{a, b, c\}$) i fem servir, per tant, un arbre ternari de booleans per representar aquests conjunts en lloc d'un arbre 26-ari de booleans. L'arbre ternari de booleans següent representa el conjunt de paraules $x = \{a,\ aab,\ ab,\ b,\ cab,\ cb,\ cc,\ ccc\}$.



Observeu que dibuixem els nodes de l'arbre mitjançant cercles que contenen el valor del seu camp **info**. Els enllaços de cada node es representen mitjançant eixos que apunten a altres nodes o al valor **NULL** (concretament els enllaços **seg[0], seg[1], seg[2]** es representen d'esquerra a dreta, respectivament). El

valor **NULL** es representa mitjançant un quadrat negre. En el context d'aquest enunciat, la constant **NULL** equival a la constant **nullptr**.

Indicació: Concretament, si a l'arbre anterior etiquetem els enllaços **seg[0], seg[1], seg[2]** amb les lletres de l'alfabet associades a cadascun (és a dir, amb **a, b, c**, respectivament), les paraules del conjunt representat per aquest arbre $x = \{a, aab, ab, b, cab, cb, cc, ccc\}$ s'obtenen **concatenant els caràcters dels enllaços que condueixen des de l'arrel fins a cadascun dels nodes amb info igual a true**. Observeu que només etiquetem els enllaços que condueixen a nodes, no els que apunten a **NULL**.



Concretament, es demana implementar els mètodes públics **pertany** i **totes** i el mètode privat **comencen_prefix**.

bool *pertany*(**const** *string*& *p*) **const**;
/* Pre: p és una cadena de caràcters formada per lletres minúscules de l'alfabet anglès. */
/* Post: El resultat indica si p pertany al conjunt de paraules representat pel p.i. */

void *totes* (*list* <*string*>& *v*) **const**;
/* Pre: v = V. V és una llista buida. */
/* Post: v conté les paraules del conjunt representat pel p.i. ordenades en ordre alfabètic. */

En la implementació del mètode `totes` heu d'utilitzar el mètode privat **comencen_prefix** especificat a continuació.

static void *comencen_prefix*(*Node** *m*, **const** *string*& *prefix* , *list* <*string*>& *v*);
/* Pre: V = $(p_1, ..., p_k)$. V està ordenada en ordre alfabètic. Les paraules de V
precedeixen a la paraula 'prefix' en l'ordre lexicogràfic (és a dir, alfabètic). */
/* Post: v =$(p_1, ..., p_k, q_1, ..., q_h)$, on $(q_1, ..., q_h)$ és la llista de paraules de la
forma $(prefix + e_1, ..., prefix + e_h)$ i $(e_1, ..., e_h)$ és la llista de paraules del conjunt que
representa la jerarquia de nodes apuntada per m ordenades en ordre alfabètic. */

Observació: Recordeu que l'operador + permet concatenar dues strings, o un string i un caràcter. Per exemple, el resultat d'avaluar l'expressió $"abc" + "wyz"$ és $"abcwyz"$. De la mateixa manera, el resultat d'avaluar l'expressió $"awy" +' z'$ és $"awyz"$. En la vostra solució podeu utilitzar també els mètodes públics de les classes `string` i `list` de la STL de C++.

Per exemple, si inicialment la variable `d` és igual a l'string *c*, la variable `w` és igual a la llista de paraules (a, aab, ab, b) i la variable `p` apunta al node arrel del fill 2-èsim (és a dir, el tercer fill) de l'arbre de la figura, després de la crida `comencen_prefix(p, d, w)` la variable `w` ha de ser igual a la llista de paraules $(a, aab, ab, b, cab, cb, cc, ccc)$.

També podeu utilitzar altres mètodes privats auxiliars que treballen directament amb dades de tipus `Node` i de tipus punter a `Node`. En aquest cas, heu de: 1) escriure la capçalera, la precondició i la postcondició de cada mètode auxiliar; 2) implementar-lo, i 3) implementar els mètodes **pertany, totes** i **comencen_prefix** fent servir aquests mètodes auxiliars. Donem a continuació la definició del tipus *ArbreNari*, que heu de fer servir per resoldre aquest problema.

Cognoms	Nom	DNI

```
template <class T> class ArbreNari {
  private:
    struct Node {
        T info;
        vector<Node*> seg;
    };
    int N;
    Node* primer;
     ...                    // especificació i implementació de mètodes privats
  public:
     ...                    // especificació i implementació de mètodes públics ;
```

2.1 Implementació de **pertany**. Donada una paraula p de longitud m, el mètode **pertany** hauria de examinar com a màxim m nodes del p.i (l'arbre que representa el conjunto de paraules). **[2.5 punts]**.

```
bool pertany(const string & p) const;
/* Pre: p és una cadena de caràcters formada per lletres minúscules de l'alfabet anglès. */
/* Post: El resultat indica si p pertany al conjunt de paraules representat pel p.i. */
```

Cognoms	**Nom**	**DNI**

2.2 Implementació de **totes** i de **comencen_prefix**. El mètode **comencen_prefix** hauria d'examinar una i solament una vegada cada node del p.i. (l'arbre que representa el conjunt de paraules). **[2.5 punts]**.

void *totes* (*list* <*string*>& *v*) **const**;
/* Pre: v = V. V és una llista buida. */
/* Post: v conté les paraules del conjunt de paraules representat pel p. i. ordenades en ordre alfabètic. */