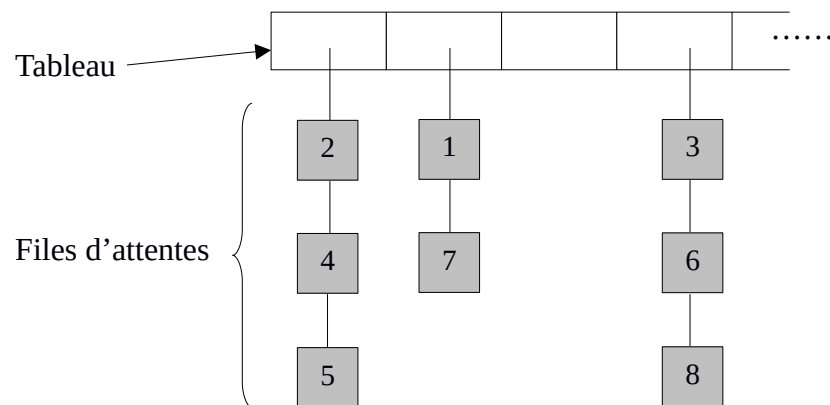


Algorithmique		
ISC	Epreuve en blanc	
Nom :		
Durée : 135 min., aucun document autorisé.		

Exercice 1. Tableau de listes (3 pts)

La gestion des tâches destinées à un ensemble d'imprimantes peut être réalisée à l'aide de la structure de données :



Chaque case du tableau (cases blanches) correspond à une imprimante, elle contient une structure pointant sur une **file d'attente** (pointeurs **tete** et **queue**).

Une file d'attente (cases grises) contient l'ensemble des impressions destinées à une imprimante. Les jobs d'impression sont identifiés par un nombre entier.

Ecrire en C, la structure de données ci-dessus.

```
typedef struct Element_ {
    int job;
    struct Element_* next;
}Element;
```

```
typedef struct FA_ {
    Element* tete;
    Element* queue;
}FA;
```

```
typedef struct Printers_ {
    FA* printers;
    int nprinters;
}Printers;
```

Exercice 2. Hachage (4 pts)

On stocke dans un tableau à 10 positions des numéros de téléphone interne à 3 chiffres. Ces numéros sont tous compris entre 100 et 299. Un numéro est noté N . On utilise la fonction de hachage : $F(N) = N \bmod 10$. Parmi les deux fonctions de gestion des collisions suivantes (X est incrémenté à chaque collision) :

1. $C_1(X) = (F(N) + 5 * X) \bmod 10$

2. $C_2(X) = (F(N) + 3 * X) \bmod 10$

la moins bonne est la première fonction, car elle ne permet pas d'atteindre toutes les cases non-occupées du tableau en cas de collision. En effet, pour une valeur N telle que $F(N) = 0$, les valeurs successives de $C_1(X)$ sont toujours 0 ou 5.

On place 145, 167, 110, 175, 210 puis 215 dans le tableau et on obtient :

Index	Numéro	Etat
0	110	OCC
1	215	OCC
2		FREE
3	210	OCC
4		FREE
5	145	OCC
6		FREE
7	167	OCC
8	175	OCC
9		FREE

210 collision

175 collision, 215 collision

215 collision

Puis on supprime 145 du tableau rempli en mettant simplement l'état de la case de occupé (OCC) à détruit (DEL).

Index	Numéro	Etat
0	110	OCC
1	215	OCC
2		FREE
3	210	OCC
4		FREE
5	145	DEL
6		FREE
7	167	OCC
8	175	OCC
9		FREE

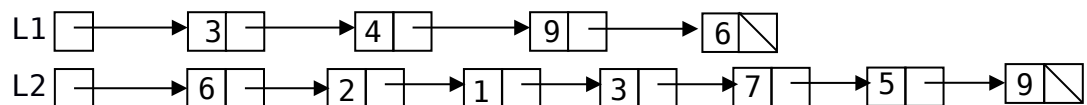
Pour rechercher 175, on calcule $F(175) = 5$, comme l'état de la case est à détruit (DEL), on va à la case suivante d'index 8 où on trouve 175. Si on recherche une valeur qui n'est pas dans le tableau comme 180, on arrête la recherche quand on tombe sur une case libre (FREE) ou quand on a visité toutes les cases.

Exercice 3. Intersection de listes (5 pts)

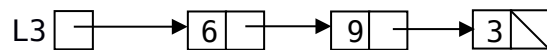
Soient L1 et L2 deux listes simplement chaînées contenant des entiers.

```
struct Element {  
    int nb;  
    Element* next;  
}  
bool is_empty(Element* lst) { return NULL == lst; }  
bool contains(Element* lst,int x) {  
    while (!is_empty(lst)) {  
        if (lst->nb == x) return true;  
        lst = lst->next;  
    }  
    return false;  
}  
void insert(Element** lst,int x) {  
    Element* tmp = (Element*)malloc(sizeof(Element));  
    tmp->nb = x;  
    tmp->next = *lst;  
    *lst = tmp;  
}  
Element* intersect(Element* L1, Element* L2) {  
    Element* L3 = NULL;  
    while (!is_empty(L2)) {  
        if (contains(L1,L2->nb)) insert(&L3,L2->nb);  
        L2 = L2->next;  
    }  
    return L3;  
}
```

Exemple



L'intersection des listes L1 et L2 est la liste :



Exercice 4. (2 pts) Tas

Soit un arbre binaire stocké dans un tableau unidimensionnel avec la racine en 1^{ère} position, puis pour un indice k :

- Les enfants se trouvent aux indices $2*k$ et $2*k+1$
- Le père se trouve à l'indice $k/2$.

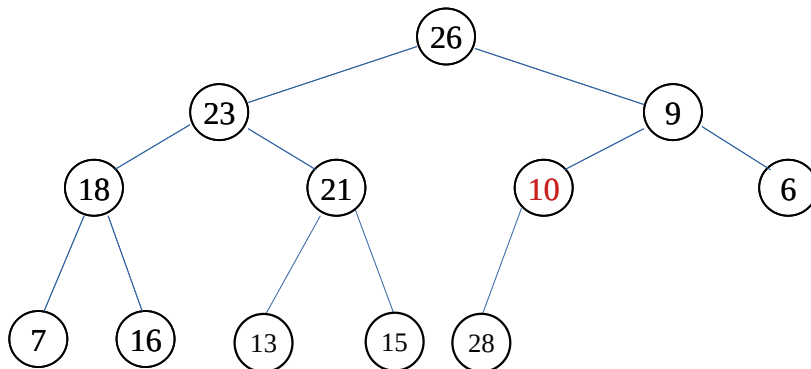
26	23	9	18	21	10	6	7	16	13	15	28
----	----	---	----	----	----	---	---	----	----	----	----

Effectuer l'entassement de cet arbre en vous aidant d'une représentation classique des arbres.

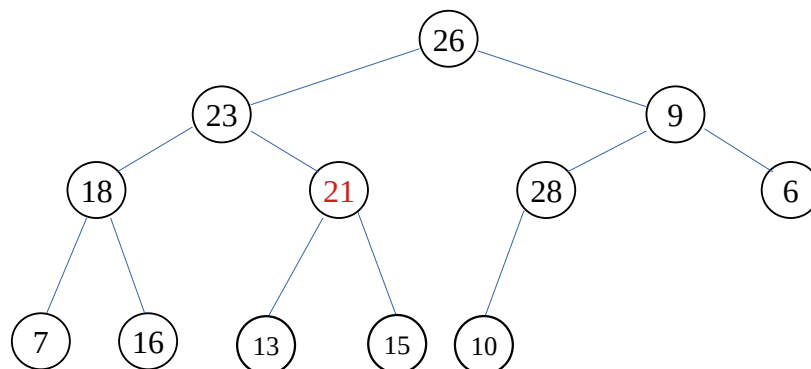
Il ne faut effectuer que la première phase du tri par tas.

Il n'est donc pas nécessaire d'effectuer le tri proprement dit.

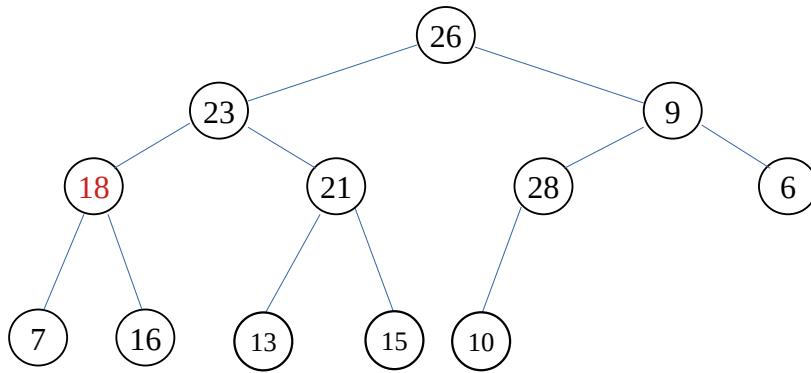
Le tableau a une taille de 12, on commence alors par considérer l'élément se trouvant à la sixième position (indice 5), soit 10. On effectue alors un tamisage sur cet élément. Puisqu'il est mal placé, il est échangé avec 28



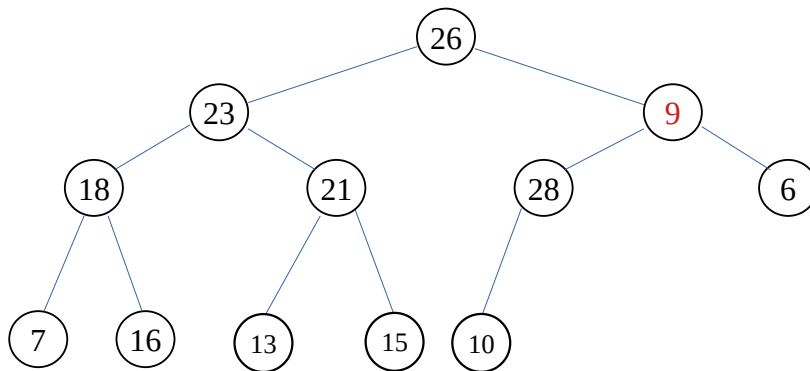
On considère ensuite l'élément précédent dans le tableau, le 21, qui est déjà bien placé.



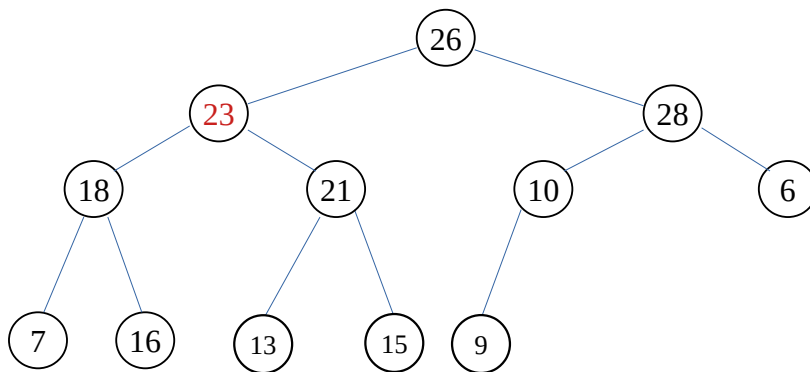
On considère ensuite le 18, qui est bien placé.



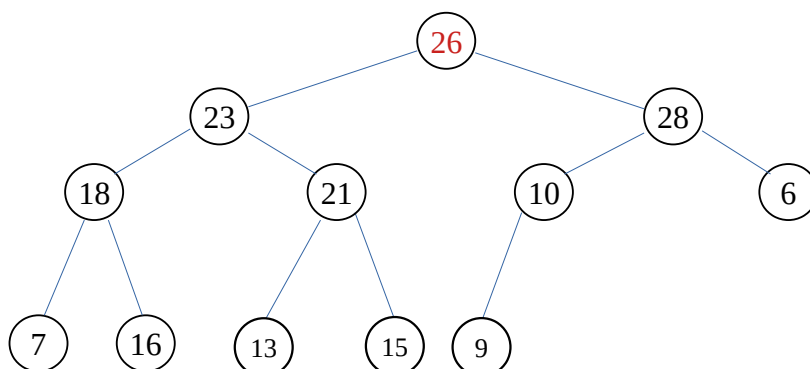
Puis le 9, qui est mal placé. L'application récursive du tamisage va le faire descendre à sa place.



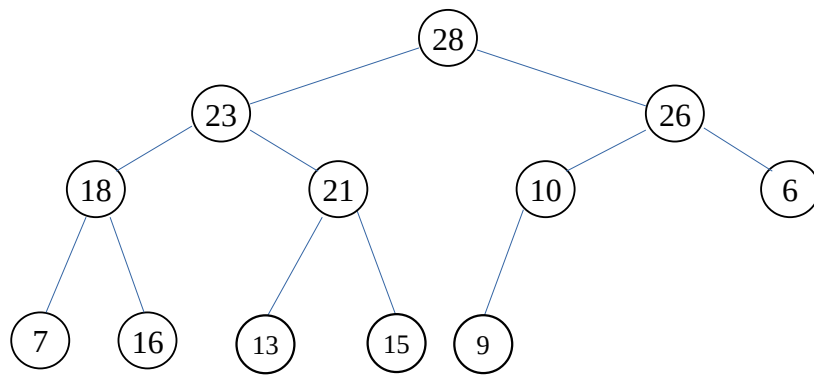
Le 23 est déjà bien placé.



Le 26 enfin est tamisé.



Voici le tas dans son état final, sous forme d'arbre :



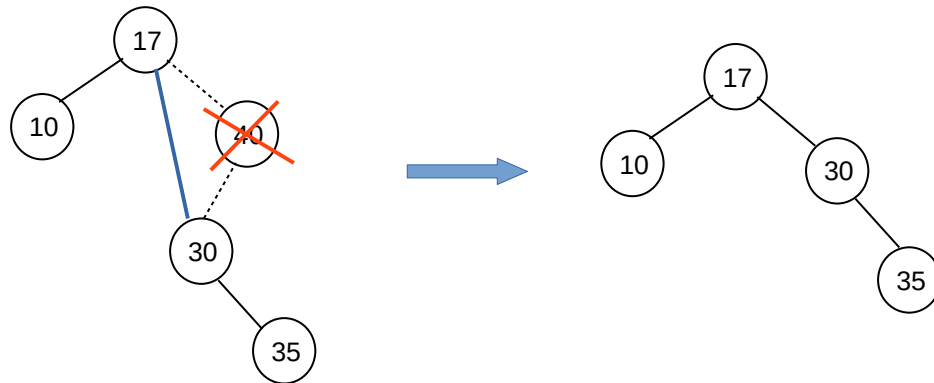
Et sous forme de tableau :

28	23	26	18	21	10	6	7	16	13	15	9
----	----	----	----	----	----	---	---	----	----	----	---

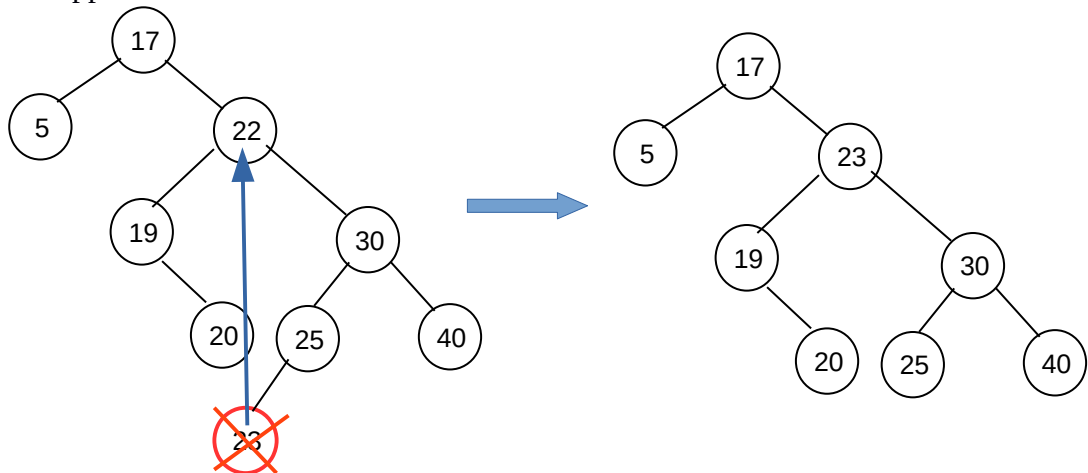
Exercice 5. Suppression dans un arbre binaire de recherche (4 pts)

Voici trois arbres binaires de recherche dans lesquels on supprime un nombre entier.

1. Suppression du nombre 40 dans l'arbre suivant :

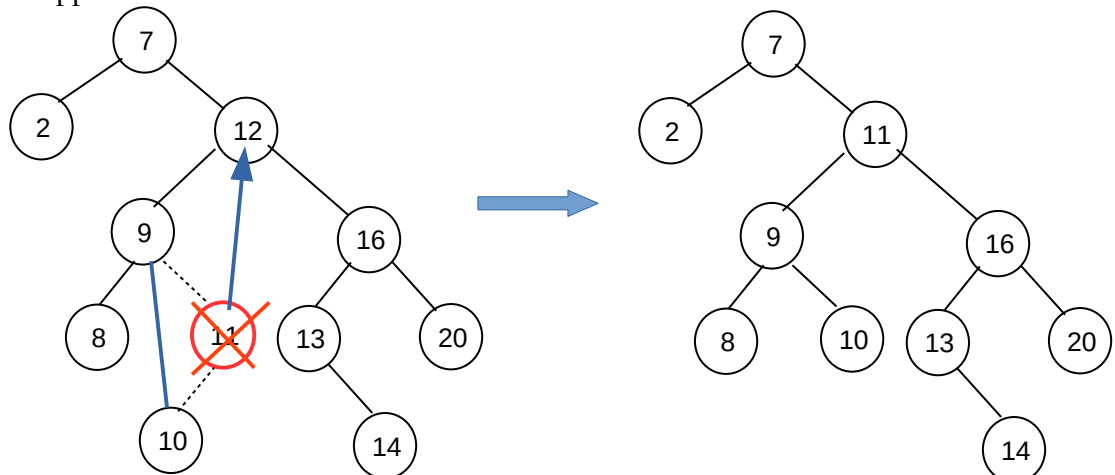


2. Suppression du nombre 22 dans l'arbre suivant :



On cherche le nœud avec la valeur qui suit 22, ici 23. On remplace 22 par 23. On supprime le nœud avec le 23 (de même si on avait pris la valeur qui précède 22, ici 20).

3. Supprimer le nombre 12 dans l'arbre suivant :



On cherche le nœud avec la valeur qui précède 12, ici 11. On remplace 12 par cette valeur. On supprime le nœud avec le 11 en procédant comme au cas 1. On aurait pu procéder similairement avec la valeur qui suit 12, ici 13.

Soit un arbre binaire défini selon la structure :