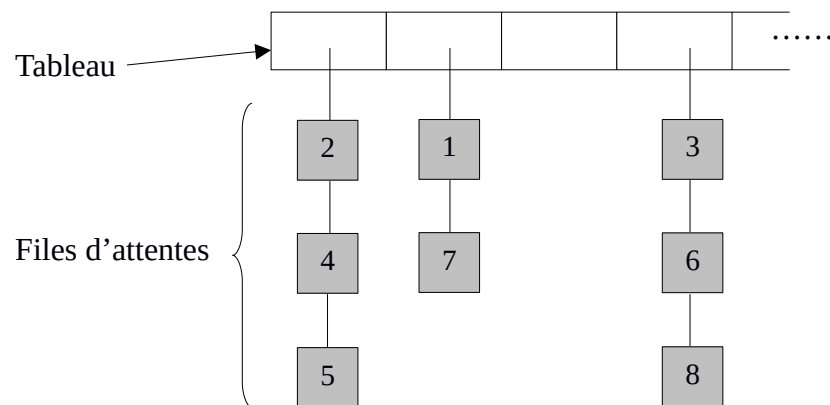


Algorithmique		
ISC	Epreuve en blanc	
Nom :		
Durée : 135 min., aucun document autorisé.		

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.

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 pour cela la fonction de hachage : $F(N) = N \bmod 10$.

Choisir parmi les deux fonctions de gestion des collisions suivantes, la meilleure (X est incrémenté à chaque collision) :

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

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

Justifier très clairement votre choix pour la fonction de gestion des collisions.

Placer 145, 167, 110, 175, 210 puis 215 dans le tableau :

Index	Numéro	Etat
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

Puis supprimer 145 du tableau rempli, expliquer comment retrouver 175.

Indiquer l'état de la case du tableau à chaque insertion ou suppression.

Exercice 3. Intersection de listes (5 pts)

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

- Définir la structure **Element** permettant de créer de telles listes.
- Ecrire en langage C une fonction **intersect** qui prend en paramètre deux listes $L1$ et $L2$; elle retourne une nouvelle liste qui est l'intersection de $L1$ et $L2$, c.-à-d. une liste contenant les éléments communs à $L1$ et $L2$. On considère que chacune des listes $L1$ et $L2$ ne contient pas d'éléments répétés.

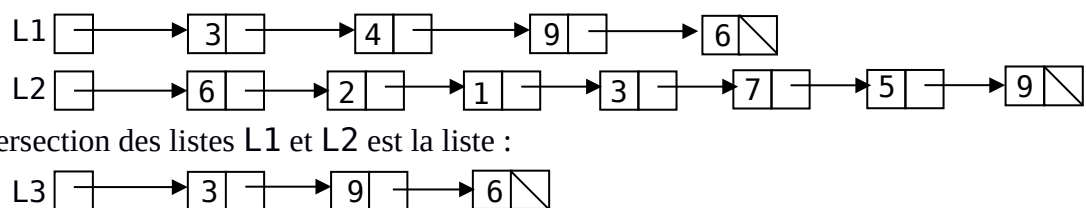
Voici l'entête de la fonction :

```
Element* intersect(Element* L1, Element* L2);
```

Votre code doit être structuré avec des fonctions.

Vous devez absolument faire des dessins !

Exemple



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.

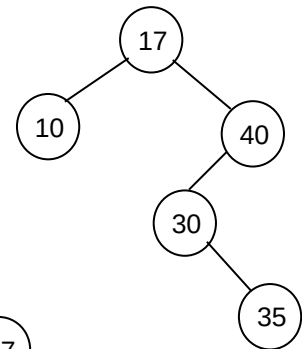
Il ne faut effectuer que la première phase du tri par tas.

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

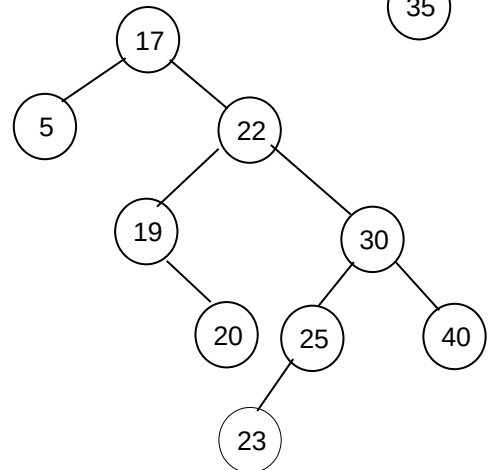
Exercice 5. Suppression dans un arbre binaire de recherche (4 pts)

Voici trois arbres binaires de recherche dans lesquels on supprime un nombre entier. Réécrire l'arbre après la suppression. Dessiner aussi les étapes intermédiaires.

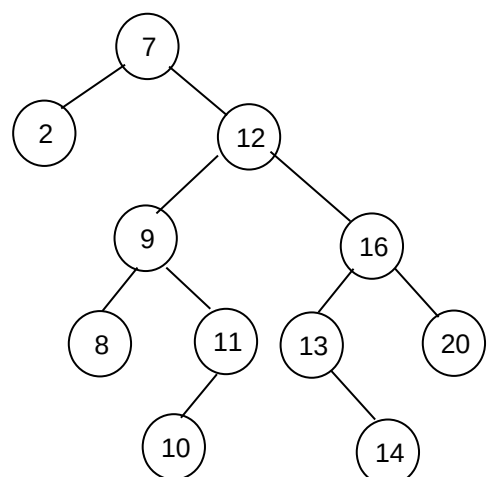
1. Supprimer le nombre 40 dans l'arbre suivant :



2. Supprimer le nombre 22 dans l'arbre suivant :



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



Exercice 6. Création d'un arbre binaire (4 pts)

Soit un arbre binaire défini selon la structure :

```
typedef struct _node {
    char ch;
    _node* gauche;
    _node* droite;
} node;
typedef node* arbre;
```

Ecrire une fonction récursive `build_tree` en langage C qui prend en paramètres un pointeur `ptree` sur un arbre (supposé vide) et un entier `prof`, et qui transforme `*ptree` en un arbre complet de profondeur `prof` dont tous les nœuds contiennent le caractère '\$'.

Voici l'entête de la procédure :

```
void build_tree(arbre* ptree, int niv);
```

Pour simplifier, on considère que le paramètre `ptree`, passé à la fonction `build_tree`, pointe sur un arbre vide, c.-à-d. que `NULL == *ptree`.

Par exemple, l'appel `build_tree(&tree,3)` produit l'arbre : `tree`

