Lebanese University
Faculty of Sciences 1

I2204F – INFO216 F

Programmation impérative

Final 2018-2019
Mardi 22 Janvier pm
Durée: 2 heures

Exercice I: Liste et fichiers (15 points)

Considérez les déclarations suivantes :

```
typedef struct {
        char nom[20];
        int age;
}element;
```

```
typedef struct noeud {
        element donnee;
        struct noeud *suivant;
} noeud;
```

1. Ecrire une fonction qui crée une liste simplement chaînée d'éléments où les noms et les âges sont lus à partir du clavier.
   L'utilisateur fournira un nom vide pour indiquer qu'il n'y a plus d'éléments à enregistrer.
2. Ecrire une fonction qui crée un fichier (texte ou binaire) et enregistre les éléments de la liste chaînée créée en ordre inverse dans le fichier.

Exercice II: Tri (10 points)

Ecrire une fonction qui trie les éléments distincts d'une liste simplement chaînée d'entiers de la manière suivante:

- Les éléments impairs sont dans le même ordre au début de la liste,
- Les éléments pairs sont en ordre inverse à la fin de la liste.

Attention: vous ne pouvez itérer la liste qu'une seule fois. L'utilisation de tableaux est interdite. Aucune allocation ou désallocation de mémoire n'est autorisée.

Exemple: Après le tri, la liste suivante

0→ 5 →4 →9 →40 →1 →26 →7 →6 →8 →61→ 100→ 101

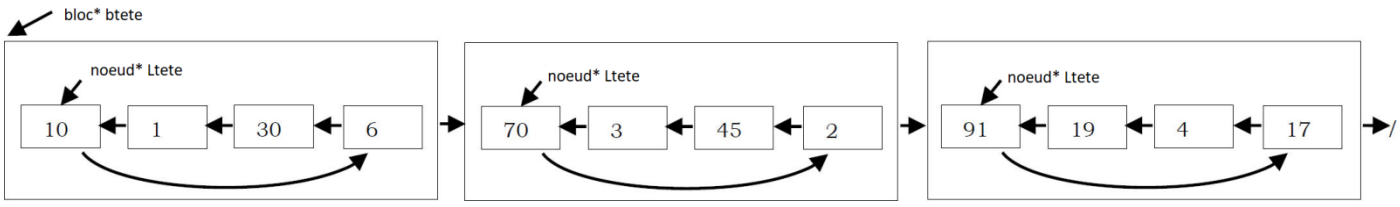
devient

5 →9→ 1→ 7→61→ 101 →100 →8 →6 →26 →40 →4 →0

Exercice III: Liste à liens déroulés (28 points)

On considère dans cet exercice une simple variante de la liste simplement chaînée appelée listes à liens déroulés.

Une liste chaînée à liens déroulés stocke plusieurs éléments dans chaque nœud (appelons-le un **bloc** pour notre commodité). Dans chaque bloc, une liste chaînée circulaire est utilisée pour connecter tous les nœuds.



Nous considérons qu'il n'y aura pas plus de *n* éléments dans la liste à liens déroulés à tout moment. Pour simplifier ce problème, tous les blocs, sauf le dernier, doivent contenir exactement $\lceil\sqrt{n}\rceil$ éléments. Ainsi, il n'y aura pas plus de $\lceil\sqrt{n}\rceil$ blocs à tout moment.

Considérez les déclarations suivantes:

```
typedef int element;
typedef struct noeud {
       element donnee;
       struct noeud* suivant;
} noeud;
```

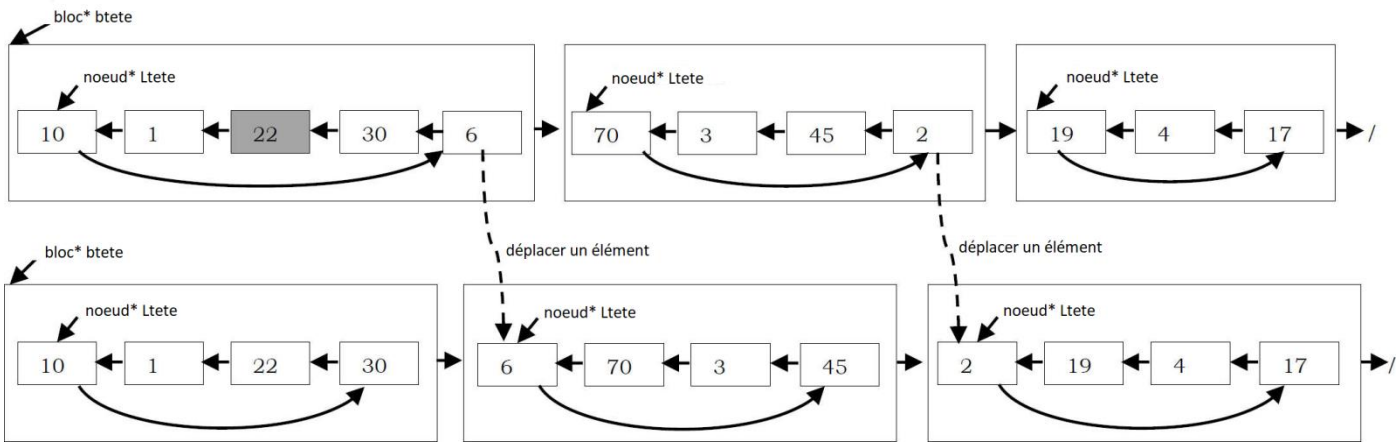
```
typedef struct bloc {
       int NbdeNoeud;
       struct bloc* suivant;
       noeud* Ltete;
} bloc;
```

Pour rechercher le $k^{ème}$ élément dans une liste à liens déroulés:

a. parcourir la liste des blocs jusqu'à celui qui contient le *k*ième nœud, c'est-à-dire le bloc $\lceil\frac{k}{\sqrt{n}}\rceil$;
b. trouver le (k mod$\lceil\sqrt{n}\rceil$)^{ème} nœud dans la liste circulaire de ce bloc.

1. Ecrire une fonction qui recherche le $k^{ème}$ élément dans une liste chaînée à liens déroulés et renvoie un pointeur sur le bloc et un pointeur sur le nœud contenant le $k^{ème}$ élément.
   `void chercherKElement(bloc* btete, int k, bloc** pbloc, noeud** pnoeud);`

Lors de l'insertion d'un nœud, nous devons réorganiser les nœuds dans la liste à liens déroulés pour conserver les propriétés mentionnées précédemment, chaque bloc contenant $\lceil\sqrt{n}\rceil$ nœuds. Supposons que nous insérons un nœud *x* après le $i^{ème}$ nœud, et que x devrait être placé dans le $j^{ème}$ bloc. Les nœuds du $j^{ème}$ bloc et des blocs situés après le $j^{ème}$ bloc doivent être décalés vers la fin de la liste, de sorte que chacun d'entre eux possède toujours $\lceil\sqrt{n}\rceil$ nœuds. De plus, un nouveau bloc doit être ajouté à la fin si le dernier bloc de la liste manque d'espace, c'est-à-dire qu'il a plus de $\lceil\sqrt{n}\rceil$ nœuds.



Étapes montrant l'ajout de l'élément 22 à la position 3 et l'opération de décalage des noeuds

2. Ecrire la fonction **décaler**, qui étant donné un pointeur de bloc, supprime un noeud de la fin de la liste circulaire dans le bloc et insère le noeud en tête de la liste circulaire dans le bloc suivant, et ainsi de suite.
   Prototype: `void decaler(bloc* btete);`
   Attention: A part le cas général, considérez le cas où un nouveau bloc doit être ajouté.

3. En utilisant les fonctions écrites ci-dessus, écrire une fonction qui ajoute un élément à la $k^{ème}$ position dans une liste à liens déroulés.
   Prototype: `void ajouterElement(bloc** btete, int k, element x);`
   Attention: A part le cas général, considérez le cas où la liste à liens déroulés est vide et lorsque *k* est égal à 0.

Bonne chance!