Travaux Pratiques d'algorithmique n°7

Cours d'Informatique de Deuxième Année

—L2.1—

Listes chaînées

Le but de ce TP est de manipuler les listes chaînées. Nous considérons ici des listes chaînées d'entiers (appelées parfois tout simplement des listes) et nous enrichissons l'ensemble des fonctions vues en TD.

Les définitions de type suivantes permettent de représenter une liste chaînée d'entiers :

```
typedef struct _cellule
{
   int valeur;                      /* donnee stockee : un entier.       */
   struct _cellule * suivant;       /* pointeur sur la cellule suivante. */
} Cellule;                          /* definition d'un nouveau type.     */
typedef Cellule * Liste;            /* definition d'un nouveau type.     */
```

▸ **Exercice 1.** *Implémenter les fonctions suivantes vues en td et écrire une fonction* `int main (void)` *permettant de les tester. C'est dans la fonction* `main` *que seront traité les echecs eventuels des fonctions :*

- `Liste AlloueCellule(int val)` *qui alloue l'espace mémoire nécessaire pour une nouvelle cellule et retourne l'adresse de cette nouvelle cellule apr{es avoir affecté la valeur* `val` *au champ* `valeur` *et* `NULL` *au champ* `suivant`*.S'il n'y a plus de place disponible la fonction renvoie la valeur* `NULL`

- `Liste InsereEnTete(Liste *p, int val)` *qui insère la valeur* `val` *en tête de la liste chaînée passée en premier argument ; la fonction renvoie l'adresse de la nouvelle tête de liste ou* `NULL` *en cas de problème*

- `Liste LireListe(Liste *p)` *qui crée une liste chaînée d'entiers non nuls en insérant successivement les entiers lus en tête de liste à partir de la liste chaînée vide ;*

- `void AfficheListe(Liste p)` *qui affiche dans l'ordre du chaînage, les éléments de la liste passée en paramètre.*

- `void AfficheInvListe(Liste p)` *qui affiche dans l'ordre inverse du chaînage, les éléments de la liste passée en paramètre.*

▸ **Exercice 2.** *Écrire les fonctions suivantes :*

- `Recherche(Liste p, int val)` *qui renvoye l'adresse de la cellule contenant l'entier* `val`*. La fonction renvoye* `NULL` *si l'entier* `val` *n'est pas présent dans la liste* `p` *;*
- `Minimum` *qui renvoye l'adresse de la cellule contenant la valeur minimale de la liste passée en paramètre.*
- `Dernier` *qui renvoye l'adresse de la dernière cellule de la liste passée en paramètre.*

▸ **Exercice 3.** *Écrire les fonctions :*

- `Liste InsereEnFin(Liste *p, int val)` *qui insère l'entier* `val` *en fin de liste ;*
- `Liste InsereApres(Liste p, int val, int apres)` *qui insère l'entier* `val` *après l'entier* `apres` *dans la liste* `p`*.*

▸ **Exercice 4.** *La liste chaînée obtenue par la fonction* `CreeListe` *de l'exercice 1 contient les entiers dans l'ordre inverse de l' ordre d'entrée. Écrire une autre fonction qui crée une liste chaînée d'entiers non nuls lus sur l'entrée standard telle que l'ordre de chaînage soit le même que l'ordre d'entrée des entiers.*

▸ **Exercice 5.** *Écrire une fonction qui compte le nombre d'occurrences d'un entier dans une liste chaînée.*

▸ **Exercice 6.** *Écrire une fonction qui supprime la première occurrence d'un entier dans une liste chaînée.*

▸ **Exercice 7.** *Écrire une fonction* `ConcateneListes` *qui concatène deux listes. La fonction reçoit deux listes et transforme la première en la concaténation des deux listes. Cette fonction ne doit pas créer de nouvelles cellules : elle modifie directement les listes passées en paramètres.*

▸ **Exercice 8.** *Écrire une fonction de prototype* `Liste ExtraitInf(Liste *p, int n);` *qui extrait de la liste passée en premier paramètre toutes les cellules contenant des valeurs inférieures à l'entier* `n`*. Les cellules enlevées ne seront pas supprimées physiquement mais elles formeront une autre liste que la fonction renvoye. Par exemple, appliquée à la liste* $l = (5, 7, 2, 1, 9, 8, 10, 15, 4, 20)$ *et à l'entier* 6*, la fonction doit construire et renvoyer la nouvelle liste* $(5, 2, 1, 4)$ *et elle doit transformer* l *en* $l = (7, 9, 8, 10, 15, 20)$.