Travaux Dirigés d'algorithmique n°5

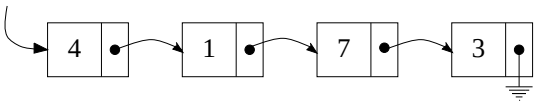
Cours d'Informatique de Deuxième Année

—DEUG Sciences M3-MI—

Listes chaînées

Dans cette séance de travaux dirigés, nous introduisons la notion de "liste chaînée par pointeurs". Cette feuille propose un certains nombre d'exercices nous permettant de manipuler ce nouveau concept.

▸ **Exercice 1. (définition de la structure)** *Une liste est un ensemble de cellules "simplement chaînées" : chaque cellule contient les informations que l'on souhaite manipuler mais il faut en plus lier toutes ces cellules pour en faire une liste chaînée.*



Nous souhaitons donc réaliser un chaînage de cellules. Analyser la définition de type suivante, et en déduire la signification de l'expression "liste chaînée par **pointeurs***".*

```
typedef struct Cellule
{ int Val;                        # donnee stockee : un entier.
  struct Cellule * PtrSuiv;       # pointeur sur la cellule suivante.
} StrCellule;                     # definition d'un nouveau type.

typedef StrCellule * PtrCellule;  # definition d'un nouveau type.
```

Quand nous utilisons des tableaux, nous devons faire de la place ou combler du vide pour respectivement insérer ou supprimer des éléments (voir le cas de la gestion d'un ensemble par tableau trié). Dans le cas d'un stockage par liste chaînée, ce problème ne se pose plus mais le prix à payer c'est qu'il faut stocker, en plus des informations manipulées, l'adresse de la cellule suivante dans l'ordre du chaînage.

Nous considérons donc des listes chaînées d'entiers : se donner une telle liste, c'est se donner un pointeur sur le premier élément de la liste ; ceci motive donc la définition du nouveau type `PtrCellule` (pointeur sur cellule). Afin de simplifier la programmation, nous nous proposons d'implémenter quelques fonctions basiques de manipulation. Dans la suite, nous parlerons de "liste chaînée" pour dénoter un pointeur de type `PtrCellule` sur la première cellule (de type `StrCellule`) d'une liste chaînée par pointeurs.

▸ **Exercice 2. (création d'une nouvelle cellule)**

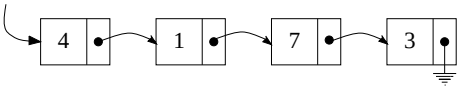
- *Ecrire une fonction* `AlloueCellule` *créant une nouvelle cellule* `nc` *et retournant un pointeur sur* `nc`.
- *En déduire une fonction de nom* `CreerCellule` *prenant en argument un entier* `val` *et retournant un pointeur sur une nouvelle cellule contenant la valeur* `val`. *Nous veillerons à ce que le champ* `PtrSuiv` *de cette structure soit initialisé à la valeur* `NULL`.

▸ **Exercice 3. (insertion d'un élément)** *Ecrire une fonction de nom* `InsListe` *prenant en argument une liste chaînée référencée par le pointeur* `PtrDebut` *et un entier* `val`. *La fonction retournera un pointeur sur la liste chaînée obtenue par insertion en tête de liste, de l'élément* `val`.

▸ **Exercice 4. (création d'une liste)** *Ecrire une fonction de nom* `LitListe` *de création d'une liste chaînée d'entiers saisis au clavier, utilisant la fonction* `InsListe`. *Nous arrêterons la saisie lorsque l'utilisateur entrera l'entier* 0 *et nous retournerons alors un pointeur sur le premier élément de la liste ainsi créée.*

▸ **Exercice 5. (affichage d'une liste)**

- *Ecrire une fonction de nom* `AffListe` *prenant en argument une liste chaînée référencée par le pointeur* `PtrDebut`. *La fonction affichera les éléments de la liste dans l'ordre du chaînage.*
- *Ecrire une fonction de nom* `AffListeInv` *prenant en argument une liste chaînée référencée par le pointeur* `PtrDebut`. *La fonction affichera les éléments de la liste dans l'ordre inverse du chaînage. L'exécution sur la liste*

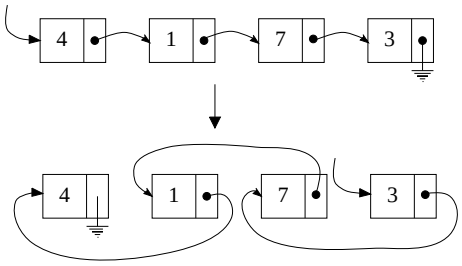


devra produire la sortie : 3 7 1 4.

▸ **Exercice 6. (suppression dans une liste)**

- *Ecrire une fonction* `Efface` *prenant en arguments une liste chaînée référencée par le pointeur* `PtrDebut`, *un entier* `n`, *et qui supprime la première occurence de* `n` *dans la liste chaînée référencée par* `PtrDebut`.
- *En déduire une fonction* `EffaceTout` *qui supprime toute les occurences d'un entier* `n`.

▸ **Exercice 7. (duplication d'une liste)** *Ecrire une fonction* `DupListe` *prenant en argument une liste chaînée référencée par le pointeur* `PtrDebut`*. La fonction retournera une liste chaînée (totalement indépendante de la liste originale) qui sera une recopie exacte de la liste chaînée de départ.*

▸ **Exercice 8. (inversion d'une liste)** *Ecrire une fonction* `InvListe` *prenant en argument une liste chaînée référencée par le pointeur* `PtrDebut`*. La fonction inversera l'ordre des éléments de la liste chaînée en créant une nouvelle liste chaînée totalement indépendante de la liste d'origine. Elle ne devra surtout pas détruire le chaînage de la liste originale.*

▸ **Exercice 9. (Inversion d'une liste sur place)** *Ecrire une fonction* `InvSurPlace` *prenant en argument une liste chaînée référencée par le pointeur* `PtrDebut`*. La fonction inversera l'ordre des éléments de la liste chaînée "sur place", i.e. en modifiant simplement le chaînage initial, lequel sera donc détruit.*