Travaux Dirigés d'algorithmique n°5

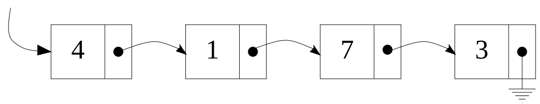
Cours d'Informatique de Deuxième Année

—DEUG Sciences S3-MI—

Listes chaînées

Dans cette séance de travaux dirigés, nous nous intéressons à l'implémentation des listes par des "listes chaînées" (plus précisément, il s'agit de "listes simplement chaînées par pointeurs").

Définition de la structure: Un élément d'une liste chaînée, appelé une *cellule*, contient les informations que l'on souhaite manipuler et également l'adresse de la cellule suivante. Par exemple, la liste d'entiers $(4, 1, 7, 3)$ est représentée par la liste chaînée d'entiers suivante :



Se donner une telle liste chaînée c'est se donner un pointeur sur le premier élément de la liste chaînée.
Les définitions de types suivantes mettent en oeuvre la notion de "liste chaînée d'entiers" :

```
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.
```

▸ **Exercice 1. (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 cellule soit initialisé à la valeur* `NULL`*.*

▸ **Exercice 2. (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 3. (création d'une liste)** *Ecrire une fonction de nom* `LitListe` *de création d'une liste chaînée d'entiers non nuls saisis au clavier. La fonction retournera un pointeur sur le premier élément de la liste ainsi créée.*

▸ **Exercice 4. (affichage d'une liste)**

- *Ecrire une fonction de nom* `AffListe` *qui affiche les éléments de la liste chaînée passée en paramètre.*
- *Ecrire une fonction de nom* `AffListeInv` *qui affiche, dans l'ordre inverse du chaînage, les éléments de la liste chaînée passée en paramètre.*

  Pour la liste chaînée donnée auparavant, `AffListe` *devra produire la sortie :* `4 1 7 3` *et* `AffListeInv` *la sortie :* `3 7 1 4`.

▸ **Exercice 5. (suppression dans une liste)**

- *Ecrire une fonction* `EffaceUn` *qui supprime la première occurence d'un entier* `n` *dans une liste chaînée.*
- *Ecrire une fonction* `EffaceTout` *qui supprime toutes les occurences d'un entier* `n` *dans une liste chaînée.*

▸ **Exercice 6. (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 7. (inversion d'une liste)** *Ecrire une fonction* `Miroir` *qui renvoie l'image miroir de la liste chaînée passée en paramètre. Cette fonction ne doit pas détruire la liste chaînée d'origine mais créer une nouvelle liste totalement indépendante.*

▸ **Exercice 8. (inversion d'une liste sur place)** *Ecrire une fonction* `MiroirSurPlace` *qui renvoie l'image miroir de la liste chaînée passée en paramètre. Cette fonction ne doit pas créer de nouvelles cellules. Elle modifiera simplement le chaînage de la liste chaînée d'origine.*