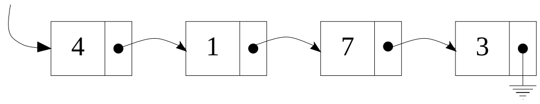
Travaux Dirigés d'Algorithmique n°5

Cours d'Informatique de Deuxième Année

—Deug SSM M3-MI—

Cette feuille propose un certain nombre d'exercices nous permettant de découvrir la notion de "liste chaînée par pointeurs". Nous verrons en particulier l'utilisation de structures et la définition de nouveaux types en langage C.

‣ **Exercice 1.** (**Définition de la structure**) Une *liste* est un ensemble de *cellules* "simplement chaînées" : chaque cellule contient principalement 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.

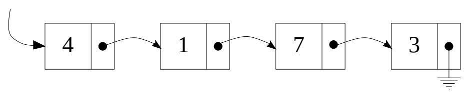
‣ **Exercice 2.** *Ecrire une fonction de nom* `AlloueCellule` *créant une nouvelle cellule* `nc` *et retournant un pointeur sur* `nc`. *En déduire la fonction de nom* `CreerCellule` *prenant en argument un entier* `val` *et retournant un pointeur sur une nouvelle cellule contenant la valeur* `val`. *Nous veillerons à ce que le champ* `PtrSuiv` *de cette structure soit initialisé à la valeur* `NULL`.

▸ **Exercice 3.** *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.** *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.** *Donner 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.*

▸ **Exercice 6.** *Donner 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 7.** *Donner une fonction de nom* `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 réplique exacte de la liste chaînée de départ.*

▸ **Exercice 8.** *Ecrire une fonction de nom* `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.** *Ecrire une fonction de nom* `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.*