# TD #2 – APL2

*18 novembre 2019*
*Erwan Kerrien*

Rappel : si vous avez des questions sur ce TD ou sur le cours, n'hésitez pas à m'envoyer un mail à Erwan.Kerrien@inria.fr (je consulte plus rarement mon mail Erwan.Kerrien@univ-lorraine.fr).

## 1 Listes chaînées

L'objectif de cet exercice est de proposer une représentation via une liste chaînée des opérations vues en cours sur une liste récursive. Par souci de concision, nous appellerons `Liste` la sorte pour la liste récursive. On rappelle la signature de cette sorte (la sorte `Élément` définit la valeur `élément_invalide`) :

```
Sorte : Liste
Utilise : Elément, Booléen
Opérations :
  liste_vide  :  -> Liste
  Contenu     : Liste -> Elément
  Succ        : Liste -> Liste
  EstVide     : Liste -> Booléen
  Créér       : Elément x Liste -> Liste
  Détruire    : Liste -> Liste
```

Les trois premières opérations sont génériques à un TAD de liste récursive. Les trois dernières sont ajoutées avec l'idée d'une représentation en liste chaînée :

- `Créer` va en effet créer une cellule, ce qui implique, ainsi que nous l'avons vu en cours, d'allouer la place mémoire nécessaire pour une cellule, pour ensuite l'initialiser (avec le bon élément et une Liste, éventuellement liste_vide, en successeur). La cellule (ou un pointeur vers cette cellule si on utilise le langage C) sera renvoyée. Je vous rappelle le cours : une cellule est une liste à un seul élément. La fonction `Créer` renvoie liste_vide si l'élément passé est élément_invalide.
- `Détruire` va faire l'opération inverse sur la mémoire, c'est-à-dire qu'elle va libérer la mémoire allouée pour la première cellule de la liste et renvoyer le successeur de cette cellule, c'est-à-dire le reste de la liste. `Détruire` la liste_vide renvoie liste_vide.
- liste_vide doit renvoyer une liste vide. Pour fixer les idées, en C, ce sera un pointeur NULL (adresse mémoire = 0x00000000)
- De même, `Succ`(liste_vide) renvoie liste_vide et `Contenu`(liste_vide) renvoie élément_invalide.

Il vous est demandé d'écrire les algorithmes des fonctions suivantes en vous basant uniquement sur les informations de la sorte ci-dessus.

1. Écrire la fonction `Afficher` qui affiche les éléments d'une Liste $L$.
2. Écrire la fonction `Longueur` qui prend en entrée une Liste $L$ et en renvoie la longueur (nombre d'éléments, identique à la fonction `Taille` du cours)
3. Écrire la fonction `Rechercher` qui teste si un Elément $E$ est dans une Liste $L$ est renvoie la Liste dont il est le premier élément si c'est le cas, ou liste_vide sinon.
   *Note : cela donne une implantation directe de la fonction* `EstDans` *vue en cours puisque cette fonction sera équivalente à* `Renvoyer non(EstVide(Rechercher(E,L)))`
4. Écrire la fonction `Supprimer` qui prend une Liste $L$, ainsi qu'un entier $r$ et supprime l'élément de rang $r$ dans $L$. La fonction renvoie la liste mise à jour. *Attention à bien* `Détruire` *la cellule correspondant à l'élément supprimé.* Si le rang $r$ est supérieur à la longueur de la Liste, rien n'est fait.
   Que faudrait-il faire si la fonction `Détruire` ne renvoyait pas le reste de la liste, et était simplement une procédure ?
5. Écrire la fonction `Ajouter` qui prend entrée une Liste $L$, un `Elément E`, et un `Entier r`, et qui ajoute `E` à `L` de telle manière qu'il a le rang `r` dans la liste mise à jour. La fonction renvoie cette liste mise à jour. Si le rang $r$ est supérieur à la longueur de la Liste, $E$ est ajouté à la fin.

6. Écrire la fonction `Inverser` qui recopie une Liste $L$ dans une nouvelle Liste $Linv$, mais avec un ordre inverse pour les éléments. La fonction renvoie $Linv$.

   Que faudrait-il faire pour faire une fonction `Copie` qui recopie $L$ dans le bon ordre ?

7. Écrire la fonction `Vider` qui vide une Liste $L$ (à la fin $L$ vaut liste_vide) en libérant proprement la mémoire allouée pour toutes les cellules de $L$.