**Université de Strasbourg**
**UFR de mathématique et d'informatique**

*L2 informatique*
*automne 2016*

# Structures de données et algorithmes

TP5 : Double chaînage et itérateur

## Introduction

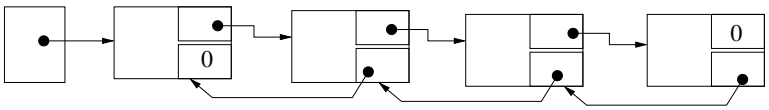
Pour parcourir un tableau C, on utilise habituellement une variable comme un indice `i` dans un tableau (en architecture, on parle de déplacement) et on passe d'un élément au suivant en faisant `i++` pour le parcourir dans le sens croissant et `i--` dans le sens décroissant ou rétrograde. L'objectif de ce TP est de faire quelque chose de semblable avec des listes doublement chaînées mais on s'arrangera pour être autant que possible indépendant de l'implantation avec la notion d'itérateur.

## Double chaînage

1. Expliquez pourquoi il n'est pas efficace de parcourir en sens rétrograde une liste implantée avec un simple chaînage.
2. On utilise le un double chaînage pour pallier ce problème en définissant la structure de données

```
typedef struct sldc {
                    S v;              // valeur
                    struct sldc *s;   // suivant
                    struct sldc *p;   // précédent
                   } *Liste ;
```

   pour implanter des objets de ce genre :




3. Expliquez pourquoi la description de ce type abstrait avec cette structure de données est insuffisante
4. Implantez les opérations standard définissant le type Liste : liste vide, ajout en tête, insertion en i-ème position, suppression, accès à la ième position
5. écrivez un programme de test avec une liste vide, adjonction en tête, insertion, suppression .... (faites un makefile)

## Itérateur

L'autre problème abordé dans ce sujet concerne la visibilité des structures de données lorsqu'on veut faire un parcours efficace : il faut garder un pointeur pt sur une cellule et passer du précédent en faisant pt = pt->s ou pt = pt->p. On va cacher un peu ceci (expliquez pourquoi c'est bien de "cacher" une structure de données) à travers un nouveau type de données :

```
typedef struct {
                 Liste first; // premier élément de la liste
                 Liste cur;   // élément courant
               } *Iter;
    Iter init(Liste L);
    Bool end(Iter i)
    Iter next(Iter i); // avec un effet de bord et une indic. de fin
    Iter prev(iter i);
    S value(Iter i);
```

On pourra ainsi écrire des choses du genre :

```
Iter i;
   for(i = init(L1); !fin(i); next(i)).......
```

1. Programmez les fonctions indiquées.
2. En utilisant ceci, programmez une fonction qui cherche un élément dans une liste et testez la.
3. Reprenez ce code en prenant cette fois-ci comme implantation des listes des simples tableaux.
4. Pour les serpentophiles :

   ```
   #define foreach(V,L) Iter _I = init(L); for( V = val(_I) ; ! end(_I); next(_I), V = val(_I))
   ```

   permet d'écrire des choses comme :

   ```
   foreach(v, t) printf("%d",v);
   ```