# Travaux Dirigés
# Séance nr. 1 : liste chainée « générique » et dictionnaire

## Exercice 1 : indexation de chaînes de caractères dans une « table » (45min)

Il arrive fréquemment de devoir ranger des chaînes de caractères dans une table, par exemple des noms, prénoms ou adresses postales. Il s'agit d'un simple dictionnaire de mots.

Pour modéliser ce dictionnaire, nous construisons une **table d'adressage indirecte *TString*** permettant de ranger des adresses de chaînes de caractères dans **un tableau contiguë**. La table associe à une chaîne donnée, un entier non signé qui représente un pointeur (l'adresse mémoire du premier caractère de la chaîne). L'adressage indirect consiste à passer par un indice intermédiaire qui permettra de faire un **rangement ordonné** des chaînes (ordre lexicographique) et donc une recherche dichotomique, plus rapide qu'une recherche séquentielle. En pratique, on utilisera donc un tableau d'adresses mémoire ***char **chaines***. Les chaînes, quant à elles, sont rangées de façon dispersée en mémoire. Chaque chaine est donc allouée individuellement avec *malloc*.

Faire une représentation graphique de la structure TString définie par :

```
typedef struct s_TString {
      char  **chaines ;
      int   nb_chaines ;
} *TString ;
```

En termes d'opérations, on définira :

- la création et destruction d'une table TString ;
- la recherche : cette opération renvoie l'adresse mémoire du premier caractère de la chaîne ; Si la chaîne n'existe pas dans la table alors on renvoie NULL. La recherche doit être dichotomique et exploiter le rangement ordonné.
  ```
  char *rechmot(TString dico, char *mot) ;
  ```
- l'adjonction : l'indice / adresse est créé au moment de l'adjonction d'une chaîne dans la table; La chaîne passée en paramètre est copiée en mémoire avec *strcpy* dans une zone nouvellement allouée avec *malloc*. Attention : faire en sorte qu'une chaîne n'existe qu'une fois dans la table. :
  ```
  char *adjmot(TString dico, char *mot) ;
  ```
- la suppression : retire la chaîne de la table si celle-ci existe.
  ```
  char *supmot(TString dico, char *mot) ;
  ```
- min et max : renvoient resp. la première et dernière chaîne de la table par ordre lexicographique.

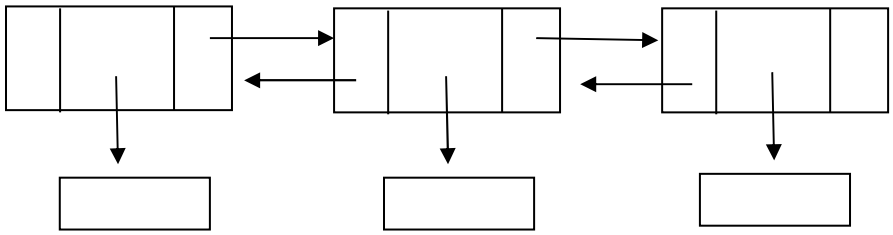
Attention à la gestion des débordements mémoire !
Quel est (sont) l' (les) inconvénient(s) d'un rangement « dispersé » des chaînes en mémoire ?

## Exercice 2 : vers des listes génériques d' « objets » (1h15)

Dans cet exercice on se propose de définir une structure permettant de gérer des listes d'éléments, ces éléments étant quelconque, c'est-à-dire de type « void * ». Pour ces éléments, on suppose qu'il existe des opérations de base :

```
void    destroy(void *e) ;
int     compare(void *x1, void *x2) ; // se comporte comme strcmp
void    *clone(void *x) ; // permet de faire une copie / un clone en mémoire
```

Proposer une structure basée sur **une liste doublement chaînée** permettant de ranger des éléments :



La structure de donnée associée est la suivante :

```
typedef struct s_node {
        void *val;
        struct s_node *suiv, *prec;
} *listeg;
```

On suppose que pour toutes les opérations suivantes *ll* pointe nécessairement sur le premier élément de la liste. On écrira les opérations usuelles suivantes :

*listeg adjt(listeg ll, void *x) ;*
*listeg adjq(listeg ll, void *x) ;*
*listeg supt(listeg ll) ;*
*listeg rech(listeg ll, void *x, int (*cmp)(void *, void *), void *(*clone)(void *)) ;*
*void *tête(listeg ll) ;*
*int longueur(listeg ll) ;*
*bool estvide(listeg ll)* ;
*void detruire(listeg ll, void (*det)(void *x)) ;*

La recherche nécessite une opération de comparaison de valeurs. Elle se fera en utilisant un pointeur de fonction : *int cmp(void *, void *)* passé en paramètre. Cette fonction renvoie 0 si les éléments sont égaux. L'opération de recherche renvoie l'ensemble des éléments (donc une liste) pour lesquels compare vaut 0. Les éléments sont dupliqués dans cette nouvelle liste à l'aide de la fonction de clonage également passée en paramètre.

Cette représentation permet de ranger n'importe quel type d'objets dans une liste, mais elle a l'inconvénient de ne permettre qu'un rangement d'éléments ayant tous le même « type » en raison du pointeur de fonction passé en paramètre. Ce sont donc des listes homogènes.

De quelle façon pourrait-on éviter d'avoir à passer en paramètre des pointeurs de fonction ?
Indication : lié directement à chaque objet les pointeurs de fonction correspondant à cet objet.

Faire un schéma.

Quel problème se pose pour la comparaison d'éléments ?