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

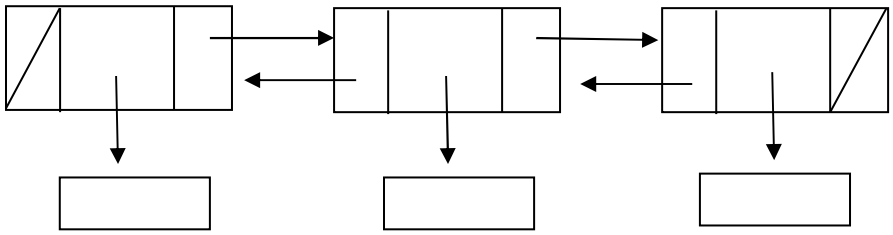
Dans cet exercice on se propose de définir une structure permettant de gérer des listes d'éléments, **ces éléments étant des objets quelconques**.

En pratique, on utilise le type ***void ** pour représenter un objet arbitraire.

Pour ces objets ***void ***, on suppose que les opérations suivantes sont définies :

```
void    destroy(void *e) ; // libère l'espace mémoire de l'objet
void    *clone(void *x) ; // permet de faire une copie / un clone en mémoire
void    affiche(void *e) ; // sans commentaire...
int     compare(void *x1, void *x2) ; // se comporte comme strcmp
```

*1. Proposer un module basé sur **une structure de données doublement chaînée** permettant de ranger des éléments quelconques dans une liste:*



Le type de liste associé est le suivant :

```
typedef struct s_node {
       void *val;  // pointeur vers objet quelconque
       struct s_node *suiv, *prec;
} *listeg;
```

Ecrire les opérations usuelles :

*listeg adjtete(listeg lst, void *x, void * (*clone)(void *x)) ;*
*listeg adjqueue(listeg lst, void *x, void * (*clone)(void *x)) ;*
*listeg suptete(listeg lst, void (*det)(void *x)) ;*

*void *tête(listeg lst) ;*
int longueur(listeg lst) ;
bool estvide(listeg lst) ;

*void detruire(listeg lst, void (*det)(void *x)) ;*
*void affichage(listeg lst, void (*affiche)(void *x)) ;*

On suppose que pour toutes ces opérations, *lst* pointe nécessairement sur le premier élément de la liste.

Notez que les opérations d'adjonction en tête et en queue utilisent un pointeur de fonction permettant de dupliquer (cloner) l'objet x en paramètre. En effet, ce sont des clones qui sont ajoutés à la liste : cela permet d'éviter des conflits de gestion de plusieurs références à un même objet.

La suppression nécessite également une fonction de destruction permettant de libérer l'espace mémoire de l'objet retiré de la liste.

2. Recherche d'objets

La recherche d'objets dans une liste nécessite une opération de comparaison. Elle se fera en utilisant un pointeur de fonction : *int cmp(void *, void *)* passé en paramètre. La fonction *cmp* est supposée se comporter comme *strcmp* pour les chaines de caractères: elle renvoie 0 si les éléments sont égaux, sinon elle renvoie une valeur entière positive ou négative selon que le premier objet est plus grand ou plus petit que le second. Ecrire l'opération :

*listeg rech(listeg lst, void *x, int (*cmp)(void *, void *), void *(*clone)(void *)) ;*

Cette opération ne renvoie pas un booléen mais **la liste de tous les objets** pour lesquels la comparaison vaut 0. Les objets sont dupliqués dans cette nouvelle liste à l'aide de la fonction de clonage également passée en paramètre.

3. Application

Nous allons utiliser la liste précédente pour ranger des prénoms (chaînes de caractères, char *). On écrira d'abord les opérations :

```
void    destroy(void *e) ;
void    *clone(void *x) ;
void    affiche(void *e) ;
```

sachant que, dans ces fonctions, les void * doivent être « castés » (convertis) en des char *.

Ecrire ensuite une opération de comparaison de prénoms :

```
int     compare(void  *x1, void *x2) ;
```

qui **n'est pas un strcmp**, mais qui ne compare que la première lettre de chacun des deux prénoms. Par exemple : *compare(« pierre », « p »)* ou *compare(« pierre », »paul »)* renvoient VRAI, alors que *compare(« pierre », « m »)* renvoie FAUX.

Ecrire ensuite un **main** qui créée une liste contenant les prénoms suivants : « claire », « jean », « amélie », « pierre », « amien », « paul », « michel », « valentine », « arnaud ».

Ecrire dans ce main un code qui récupère et affiche tous les prénoms commençant par la lettre « a ».

Exercices complémentaires de réflexion

4. Listes de listes

La représentation précédente permet de ranger n'importe quel type d'objets dans une liste, alors pourquoi ne pas l'utiliser pour créer des listes dont les éléments sont des listes, c'est-à-dire des ***listes de listes d'objets quelconques*** ?

Mais si l'objet d'une liste est elle-même une liste d'objets quelconques, il faut réfléchir à l'écriture des opérations : destroy, affiche, clone et compare. Quel problème se pose alors ?

5. Listes hétérogènes

Pourquoi la représentation précédente a-t-elle l'inconvénient de ne permettre qu'un rangement d'objets ayant tous le même « type » ?

Ce sont donc des listes dites ***homogènes***. Certaines opérations nécessitent des fonctions sur les objets, comme la recherche, la destruction, l'affichage, etc.

De quelle façon pourrait-on éviter d'avoir à passer en paramètre des pointeurs de fonction, ce qui permettrait de gérer des listes hétérogènes ?

Indication : lier directement à chaque objet les pointeurs de fonction correspondant à cet objet. Faire un schéma.

Quel problème se pose néanmoins pour la recherche d'éléments dans une liste hétérogène ?