# TP 9 : Allocation dynamique de mémoire

Objectif :

- Maîtriser l'allocation dynamique de mémoire et son utilisation



# Les fonctions d'allocation dynamique de mémoire

## 1. Notions de base

Le détail des fonctions utilisées dans cette partie (*malloc*, *free*) est donné dans la section 6.11 du document « Introduction au langage C ».

## 2. Exercices

### Liste chaînée avec allocation dynamique de mémoire

Dans cet exercice nous réalisons les fonctions de gestion d'une liste chaînée de manière similaire au TP6, mais en utilisant l'allocation dynamique de mémoire (au lieu d'un tableau). Nous considérons les déclarations de type suivantes :

```
typedef struct element element ;
struct element {
      int valeur ;                  /* valeur de l'élément */
      struct element* suivant ;     /* adresse du successeur */
};

typedef element* liste ;
```

La création d'une liste vide, revient à faire la déclaration :

```
liste L = NULL ;  /* L contient l'adresse du premier élément de la liste */
```

La création d'un nouvel élément se réalise de la manière suivante :

```
element *e = (element* ) malloc(sizeof(element)) ;
```

Ensuite, les champs de *e* peuvent être consultés ou modifiés à l'aide de l'opérateur -> :

```
e->valeur = 12 ;
e-> suivant = NULL ;
```

Ou, de manière équivalente (à l'aide de l'opérateur de déréférencement *) :

```
(*e).valeur = 12 ;
(*e).suivant = NULL ;
```

Réaliser les fonctions C permettant de gérer une telle liste et tester les (au moins) dans les cas suivants : insertion et suppression et tête, en fin et en milieu de liste.

- `void afficherListe(liste *l)` : affiche tous les éléments de la liste dans l'ordre.
- `void insererElement(int x, liste *l)` : en supposant l triée, insère x dans l en maintenant L triée
- `void supprimerElement(int i, liste *l)` : supprime le i-ème élément de l.

NB : le type du paramètre *l* des fonctions ci-dessus est de type *liste**. Ceci est nécessaire dans le cas d'insertion/suppression en tête de liste. *Pourquoi ?*

### Liste chaînée avec allocation dynamique de mémoire

Nous souhaitons maintenant réaliser une liste « doublement chaînée », c'est-à-dire permettant un parcours dans les deux sens. Pour cela, on déclare les types suivants :

```
typedef struct element element
struct element{
      int valeur ;            /* valeur de l'élément */
      elementD* suivant ;     /* adresse du successeur */
      elementD* precedent ;   /* adresse du prédécesseur */
};

typedef element* listeD ;
```

Réaliser et tester les fonctions de gestion de cette liste.