# naaaaaaaaæaaæaaæaa a aæaaana

## Le langage C

## TD3

Une liste chainée se constitue de cellules. Dans un programme C, une liste est manipulée via un pointeur dont la valeur est l'adresse de la première cellule de la liste ou NULL si la liste est vide. Chaque cellule est liée à la cellule suivante via un pointeur suivant. Pour accéder la i ème cellule, il faut parcourir la liste de la première cellule jusqu'à la i ème.

Cette structure de donnée est particulièrement intéressante pour programmer une pile.

Dans ce TD, nous voyons comment programmer une liste simplement chainée, comment utiliser une liste pour programmer une pile et un sac.

## naæaæaaaæaa

- **Proposer le type *t_cell* pour mettre en oeuvre une cellule de la liste**
- **Programmer les fonctions suivantes :**
    1. *l_vide* qui retourne une liste vide

       *t_cell * l_vide()*
    2. *l_add_tete* qui ajoute un entier en tete de liste et retourne la nouvelle tête de liste.

       *t_cell * l_add_tete(int val, t_cell * queue)*
    3. *l_add_queue* qui ajoute un entier en queue de liste et retourne la tête de liste.

       *t_cell* l_add_queue(int val, t_cell * queue)*
    4. *l_copie* qui copie une liste et retourne la tête de la nouvelle liste

       *t_cell * l_copie (t_cell * tete)*
    5. *l_affiche* qui affiche la liste sur la sortie standard tel que montré dans l'exemple

       *l_affiche (t_cell * tete)*
    6. *l_libere* libère l'espace mémoire alloue pour la liste

       *void l_libere (t_cell * tete)*
    7. l_taille qui retourne le nombre de cellules de la liste.

       *Int l_taille(t_cell * tete)*
    8. l_sont-egales qui retourne 1 si les liste contiennent les même entiers et dans le même ordre, et qui retourne 0 sinon

       *int l_sont_egales (t_cell * l1, t_cell * l2)*
    9. l_inverse retourne une copie de la liste chainée inversée

       *t_cell * l_inverse (t_cell * tete)*

**Exemple d'utilisation de la liste :**

```
t_cell * l1 = l_vide() , * cpy, * inverse ;
l1 = l_add_tete(23, l1);
l1= l_add_queue(0, l1);
l1= l_add_tete(5, l1);
l_affiche(l1); /* Affiche : [5]->[23]->[0]->nil */
cpy = l_copie(l1->suiv);
l_affiche(cpy); /* Affiche :[23]->[0]->nil */
inverse = l_inverse(l1);
l_affiche(inverse); /* [0]->[23]->[5]->nil */
t_cell *cpy2=l_copy(l1);
printf("egale ?%d ",l_sont_egales(l1,cpy2));
```

```
printf("egale ?%d ",l_sont_egales(l1,inverse));
l_libere (l1);
l_libere (cpy);
l_libere (cpy2);
l_libere (inverse);
```

## aææ

On désire mettre en œuvre une pile en utilisant la liste chainée.

**Proposer la mise en œuvre du type *t_pile* et celle des fonctions suivantes :**

- *p_new* qui construit une pile vide

  *void p_new (t_pile * p)*
- *p_push* qui ajoute un entier en tete de pile

  *void p_push(t_pile * p, int v)*
- *p_top* qui retourne l'entier en tete de pile

  *int p_top(t_pile * p)*
- *p_pop* qui dépile et retourne l'entier dépilé

  *int p_pop(t_pile * p)*
- *p_size* qui retourne la taille de la pile

  *int p_size(t_pile * p)*
- *p_is_empty* qui retourne 1 si la pile est vide, 0 sinon

  *int p_is_empty(t_pile *p)*

**Proposer un programme de test le plus complet possible pour la pile.**

## aaa

Un Sac est un ensemble qui autorise les doublons. On désire mettre en œuvre un sac en utilisant une liste chainée.
**Proposer le type *t_sac* et la mise en œuvre des fonctions suivantes :**

- *s_new* qui construit un sac vide

  *void s_new (t_sac * s)*
- *s_add* qui ajoute un entier dans le sac et retourne son nombre d'occurrences après ajout

  *int s_add (t_sac * s, int v)*
- *s_add_n* qui ajoute n occurrences d'un entier dans le sac

  *int s_add_n (t_sac * s, int v, int n)*
- *s_remove* qui supprime une occurrence d'un entier dans le sac et retourne le nombre d'occurrences après suppression

  *int s_remove (t_sac * s, int v)*
- *s_remove_n* qui supprime n occurrence d'un entier dans le sac et retourne le nombre d'occurrences après suppression

  *int s_remove_n (t_sac * s, int v, int n)*
- *s_occurences_of* qui retourne le nombre d'occurrences d'un entier dans le sac

  *int s_occurences_of (t_sac * s, int v)*
- *s_contains* qui retourne 1 si l'entier passé en argument est dans le sac, 0 sinon

  *int s_contains (t_sac * s, int v)*

**Proposer un programme de test le plus complet possible**