# Liste Simplement Chaînée Circulaire

## I. Liste simplement chaînée circulaire

### 1. Problème :

Il s’agit ici pour nous de manipuler une « liste simplement chaînée circulaire » en appliquant des opérations telles que :  
• l’insertion en tête  
• l’insertion en fin  
  
Contraintes : Cependant nous faisons face à des difficultés telles que la création de la structure, la gestion dynamique de la mémoire et la détermination d’une complexité optimale pour les insertions.

### 2. Principe :

Afin de résoudre ce problème, plusieurs solutions s’offrent à nous et nous comptons procéder comme suit :  
i. Utiliser une liste doublement chaînée circulaire, car cette structure présente des avantages tels que : le dernier nœud pointe vers le premier et vice-versa. Le plus important des avantages est que nous restons en complexité O(1) lors de l’insertion ou de la suppression en tête et en fin.

### 3. Algorithme :

Algorithme Gestion\_Liste\_Simplement\_Chainee\_Circulaire  
Type  
 Cellule = Enregistrement  
 val : entier;  
 suiv : ^Cellule;  
 FinEnreg;  
  
 Liste = ^Cellule;  
  
Var  
 n, z, y, i : entier;  
 l, p, m, der, courant, nod : Liste;  
  
Debut  
 ecrire("Entrez la longueur de la liste : ");  
 lire(n);  
  
 Si (n <= 0) Alors  
 nod <- NULL;  
 Sinon  
 l <- NULL;  
 der <- NULL;  
  
 Pour i de 0 à n-1 Faire  
 Allouer(p);  
 Si (p = NULL) Alors  
 ecrire("erreur : allocation mémoire");  
 Retourner -1;  
 FinSi;  
  
 ecrire("Entrez l'élément : ");  
 lire(p^.val);  
  
 Si (l = NULL) Alors  
 p^.suiv <- p;  
 l <- p;  
 der<- p;  
 Sinon  
 p^.suiv <- l;  
 der^.suiv <- p;  
 der<- p;  
 FinSi;  
 FinPour;  
  
 nod <- l;  
 FinSi;  
  
 ecrire("Entrez la valeur à ajouter en tête : ");  
 lire(z);  
  
 Allouer(m);  
 Si (m = NULL) Alors  
 ecrire("erreur : allocation mémoire");  
 Retourner -1;  
 FinSi;  
  
 m^.val <- z;  
  
 Si (nod = NULL) Alors  
 m^.suiv <- m;  
 nod <- m;  
 Sinon  
 m^.suiv <- nod;   
 nod <- m;  
 FinSi;  
  
 ecrire("Éléments de la liste : ");  
 Si (nod = NULL) Alors  
 ecrire("Liste vide");  
 Sinon  
 courant <- nod;  
 Répéter  
 ecrire(courant^.val);  
 courant <- courant^.suiv;  
 Jusquà (courant = nod);  
 ecrire("");  
 FinSi;  
  
 ecrire("Entrez la valeur à ajouter en fin : ");  
 lire(y);  
  
 Allouer(m);  
 Si (m = NULL) Alors  
 ecrire("erreur : allocation mémoire");  
 Retourner -1;  
 FinSi;  
  
 m^.val <- y;  
  
 Si (nod = NULL) Alors  
 m^.suiv <- m;  
 nod <- m;  
 Sinon  
 m^.suiv <- nod;  
 der^.suiv <- m;  
 FinSi;  
  
 ecrire("Éléments de la liste : ");  
 Si (nod = NULL) Alors  
 ecrire("Liste vide");  
 Sinon  
 courant <- nod;  
 Répéter  
 ecrire(courant^.val);  
 courant <- courant^.suiv;  
 Jusquà (courant = nod);  
 ecrire("");  
 FinSi;  
  
 Si (nod <> NULL) Alors  
 courant <- nod^.suiv;  
 TantQue (courant <> nod) Faire  
 p <- courant;  
 courant <- courant^.suiv;  
 Désallouer(p);  
 FinTantQue;  
 Désallouer(nod);  
 nod <- NULL;  
 FinSi;  
  
 ecrire("Programme terminé avec succès");  
 Retourner 0;  
Fin.

### 4. Dictionnaire de données :

#### 4.1. Type de données

Type Description  
liste Structure représentant un nœud de la liste  
list Pointeur vers un nœud de la liste

#### 4.2. Structures

typedef struct cellule {  
 int val; // Donnée stockée dans le nœud  
 struct cellule\* suiv; // Pointeur vers le nœud suivant  
 struct cellule\* prec; // Pointeur vers le nœud précédent  
} liste;

#### 4.3. Variables principales

Variable Type Description  
l list Pointeur vers la tête de la liste  
P, m list Pointeurs temporaires pour nouveaux nœuds  
courant list Pointeur pour le parcours  
der list Pointeur vers le dernier nœud  
n int Taille de la liste à créer  
X int Valeur à insérer

### 5. Complexité :

#### 5.1. Analyse asymptotique

Opération Complexité temporelle Complexité spatiale  
creernoeud(n) O(n) O(n)  
InsererTete(l,x) O(1) O(1)  
InsererFin(l,x) O(1) O(1)  
Display(l) O(n) O(1)

#### 5.2. Justification des complexités

insererTete() et insererFin() en O(1) :  
• Grâce à la structure circulaire, on accède directement au dernier élément via l->prec  
• Pas besoin de parcourir toute la liste  
• Nombre constant d'opérations (4 affectations de pointeurs)  
  
creernoeud() en O(n) :  
• Chaque insertion est en O(1)  
• On répète n fois → O(n)  
  
display() en O(n) :  
• Parcours de tous les n éléments  
• S'arrête quand on revient au point de départ