# Rapport : Liste Doublement Chaînée Circulaire

## 1. Problème

Implémenter une liste doublement chaînée circulaire en langage C permettant :
- L'ajout d'un élément en tête de liste.
- L'ajout d'un élément en fin de liste.
- L'affichage de la liste complète.

La structure doit garantir que la liste soit circulaire :
- Le dernier élément pointe vers le premier.
- Le premier élément pointe vers le dernier.

## 2. Principe

Une liste doublement chaînée circulaire est une structure de données dynamique où :
- Chaque élément contient une valeur (data) et deux pointeurs (*next* et *prev*).
- Les éléments sont reliés de manière circulaire : le dernier pointe vers le premier et le premier vers le dernier.

**Insertion en tête :**
On crée un nouvel élément. Celui-ci est relié au premier élément actuel (*tete*) et au dernier élément. Ensuite, on met à jour le pointeur du dernier pour qu'il pointe vers ce nouvel élément, et on met à jour la tête pour que ce soit le nouvel élément.

**Insertion en fin :**
On crée un nouvel élément. Celui-ci est relié au dernier élément actuel et à la tête. Ensuite, on met à jour le pointeur du dernier et celui de la tête pour maintenir la circularité.

**Affichage :**
On parcourt la liste à partir de la tête et on affiche les valeurs jusqu'à revenir à la tête, ce qui permet de parcourir tous les éléments exactement une fois.

## 3. Dictionnaire de Données

| Nom | Type | Rôle |
|---|---|---|
| element | Structure | Définit un nœud (valeur + pointeurs) |
| data | entier (int) | Valeur stockée dans un nœud |
| next | pointeur sur element | Pointe vers le nœud suivant |
| prev | pointeur sur element | Pointe vers le nœud précédent |
| tete | pointeur sur element | Pointe sur le premier élément de la liste |
| p | pointeur sur element | Pointeur temporaire vers le nouvel élément créé |
| temp | pointeur sur element | Utilisé pour parcourir la liste |

## 4. Pseudo-code (Algorithmes)

**Procédure insertionTete(valeur)**
p ← creerElement(valeur)
si tete = NULL alors
  p^.next ← p
  p^.prev ← p
  tete ← p
sinon
  dernier ← tete^.prev
  p^.next ← tete
  p^.prev ← dernier
  dernier^.next ← p
  tete^.prev ← p
  tete ← p
fin si

**Procédure insertionFin(valeur)**
p ← creerElement(valeur)
si tete = NULL alors
  p^.next ← p
  p^.prev ← p
  tete ← p
sinon
  dernier ← tete^.prev
  p^.next ← tete
  p^.prev ← dernier
  dernier^.next ← p
  tete^.prev ← p
fin si

**Procédure afficherListe()**
si tete = NULL alors
  afficher "Liste vide"
sinon
  temp ← tete
  répéter
    afficher temp^.data
    temp ← temp^.next
  jusqu'à temp = tete
fin si

## 5. Complexité

- Insertion en tête : O(1)
- Insertion en fin : O(1)
- Parcours (affichage de la liste complète) : O(n)