# Liste Doublement Chaînée Circulaire

## 📋 Table des matières

1. [Description](#description)

2. [Structure de données](#structure-de-données)

3. [Algorithmes implémentés](#algorithmes-implémentés)

4. [Complexité](#complexité)

5. [Compilation et exécution](#compilation-et-exécution)

6. [Exemples d'utilisation](#exemples-dutilisation)

---

## Description

Ce programme implémente une \*\*liste doublement chaînée circulaire\*\* en langage C. Une liste doublement chaînée circulaire est une structure de données dynamique où :

- Chaque nœud contient un pointeur vers le nœud suivant ET précédent

- Le dernier nœud pointe vers le premier (circularité avant)

- Le premier nœud pointe vers le dernier (circularité arrière)

### Avantages

- Parcours bidirectionnel (avant et arrière)

- Accès direct au dernier élément via `tete->precedent` (O(1))

- Pas de pointeurs NULL à gérer

- Insertions/suppressions optimisées en tête et fin

---

## Structure de données

### Définition du nœud

```c

typedef struct liste {

int val; // Valeur stockée

struct liste \*suivant; // Pointeur vers le nœud suivant

struct liste \*precedent; // Pointeur vers le nœud précédent

} liste;

```

### Représentation visuelle

```

Liste: 6 <-> 3 <-> 1 <-> 5 <-> 7

↑ ↓

└─────────────────────────┘

```

---

## Algorithmes implémentés

### 1. Création d'un nœud (`create`)

\*\*Algorithme :\*\*

```

FONCTION create(val)

Allouer mémoire pour nouveau nœud

SI allocation échoue ALORS

Afficher erreur et quitter

FIN SI

nouveau.val ← val

nouveau.suivant ← nouveau // Auto-référence

nouveau.precedent ← nouveau // Auto-référence

RETOURNER nouveau

FIN FONCTION

```

\*\*Propriétés :\*\*

- Complexité : O(1)

- Gère l'erreur d'allocation mémoire

---

### 2. Insertion en tête (`inserttete`)

\*\*Algorithme :\*\*

```

FONCTION inserttete(tete, val)

nouveau ← create(val)

SI tete = NULL ALORS

// Liste vide

tete ← nouveau

SINON

dernier ← tete.precedent

// Mise à jour des 4 liens

nouveau.suivant ← tete

nouveau.precedent ← dernier

dernier.suivant ← nouveau

tete.precedent ← nouveau

// Nouvelle tête

tete ← nouveau

FIN SI

FIN FONCTION

```

\*\*Schéma :\*\*

```

Avant: 3 <-> 1 <-> 5

↑ ↓

└───────────┘

Insérer 6 en tête:

Après: 6 <-> 3 <-> 1 <-> 5

↑ ↓

└─────────────────┘

```

\*\*Propriétés :\*\*

- Complexité : O(1)

- Accès direct au dernier nœud

---

### 3. Insertion en fin (`insertfin`)

\*\*Algorithme :\*\*

```

FONCTION insertfin(tete, val)

nouveau ← create(val)

SI tete = NULL ALORS

// Liste vide

tete ← nouveau

SINON

dernier ← tete.precedent

// Mise à jour des 4 liens

dernier.suivant ← nouveau

nouveau.precedent ← dernier

nouveau.suivant ← tete

tete.precedent ← nouveau

FIN SI

FIN FONCTION

```

\*\*Schéma :\*\*

```

Avant: 6 <-> 3 <-> 1

↑ ↓

└───────────┘

Insérer 5 en fin:

Après: 6 <-> 3 <-> 1 <-> 5

↑ ↓

└─────────────────┘

```

\*\*Propriétés :\*\*

- Complexité : O(1)

- Pas besoin de parcourir la liste

---

### 4. Affichage direct (`affiche`)

\*\*Algorithme :\*\*

```

FONCTION affiche(tete)

SI tete = NULL ALORS

Afficher "Liste vide"

RETOURNER

FIN SI

T ← tete

FAIRE

Afficher T.val

T ← T.suivant

TANT QUE T ≠ tete

FIN FONCTION

```

\*\*Propriétés :\*\*

- Complexité : O(n)

- Parcours dans le sens suivant

---

### 5. Affichage inverse (`affiche\_inverse`)

\*\*Algorithme :\*\*

```

FONCTION affiche\_inverse(tete)

SI tete = NULL ALORS

Afficher "Liste vide"

RETOURNER

FIN SI

T ← tete.precedent // Commencer par le dernier

FAIRE

Afficher T.val

T ← T.precedent

TANT QUE T ≠ tete.precedent

FIN FONCTION

```

\*\*Propriétés :\*\*

- Complexité : O(n)

- Parcours dans le sens précédent

---

## Complexité

| Opération | Complexité temporelle | Complexité spatiale |

|-----------|----------------------|---------------------|

| Création d'un nœud | O(1) | O(1) |

| Insertion en tête | O(1) | O(1) |

| Insertion en fin | O(1) | O(1) |

| Affichage | O(n) | O(1) |

| Affichage inverse | O(n) | O(1) |

\*\*Légende :\*\*

- n = nombre d'éléments dans la liste

- O(1) = temps constant

- O(n) = temps linéaire

---

## Compilation et exécution

### Prérequis

- Compilateur GCC

- Système d'exploitation : Linux, macOS, Windows (avec MinGW)

### Commandes

```bash

# Compilation

gcc -o liste\_circulaire listspcir.c -Wall -Wextra

# Exécution

./liste\_circulaire

```

### Options de compilation recommandées

```bash

gcc -o liste\_circulaire listspcir.c -Wall -Wextra -g -std=c99

```

- `-Wall` : Active tous les avertissements

- `-Wextra` : Avertissements supplémentaires

- `-g` : Informations de débogage

- `-std=c99` : Standard C99

---

## Exemples d'utilisation

### Sortie du programme

```

Liste après insertion en tête :

Sens direct : 6 -> 3 -> 1 -> (retour à tête : 6)

Sens inverse : 1 -> 3 -> 6 -> (retour au dernier : 1)

Liste après insertion en fin :

Sens direct : 6 -> 3 -> 1 -> 5 -> 7 -> (retour à tête : 6)

Sens inverse : 7 -> 5 -> 1 -> 3 -> 6 -> (retour au dernier : 7)

```

### Trace d'exécution détaillée

#### Étape 1 : Insertions en tête

```

inserttete(1) → [1]

inserttete(3) → [3] <-> [1]

inserttete(6) → [6] <-> [3] <-> [1]

```

#### Étape 2 : Insertions en fin

```

insertfin(5) → [6] <-> [3] <-> [1] <-> [5]

insertfin(7) → [6] <-> [3] <-> [1] <-> [5] <-> [7]

```

---

## Améliorations possibles

### Fonctionnalités à ajouter

1. \*\*Suppression\*\* :

- Suppression en tête

- Suppression en fin

- Suppression d'une valeur spécifique

2. \*\*Recherche\*\* :

- Rechercher une valeur

- Compter les occurrences

3. \*\*Manipulation\*\* :

- Inversion de la liste

- Tri de la liste

- Fusion de deux listes

4. \*\*Gestion mémoire\*\* :

- Fonction pour libérer toute la liste

- Détection de fuites mémoire

### Exemple de fonction de suppression

```c

void supprime\_tete(liste \*\*tete) {

if(\*tete == NULL) return;

liste \*temp = \*tete;

if((\*tete)->suivant == \*tete) {

// Un seul élément

\*tete = NULL;

} else {

liste \*dernier = (\*tete)->precedent;

\*tete = (\*tete)->suivant;

(\*tete)->precedent = dernier;

dernier->suivant = \*tete;

}

free(temp);

}

```

---

## Comparaison avec d'autres structures

| Structure | Insertion tête | Insertion fin | Accès dernier | Bidirectionnel |

|-----------|----------------|---------------|---------------|----------------|

| Liste simple | O(1) | O(n) | O(n) | Non |

| Liste simple circulaire | O(1) | O(n) | O(n) | Non |

| Liste doublement chaînée | O(1) | O(n)\*