README - Programme de Gestion des Listes Chaînées en C

Table des matières

Introduction

Ce programme implémente quatre types de listes chaînées en langage C, chacune avec ses opérations spécifiques. Il s'agit d'un outil pédagogique démontrant les concepts fondamentaux des structures de données dynamiques.

Types de listes implémentées

- **Liste simple** : chaînage unidirectionnel
- **Liste double** : chaînage bidirectionnel
- **Liste circulaire simple** : dernier élément pointe vers le premier
- **Liste circulaire double** : chaînage bidirectionnel circulaire

Objectif pédagogique

Ce programme permet de comprendre et manipuler différentes structures de listes chaînées, leurs avantages et cas d'usage respectifs.

Structures de données

1. Liste Simple (Node)

```c
typedef struct Node {
    int data;          // Données stockées
    struct Node *next;  // Pointeur vers le nœud suivant
} Node;
```

Caractéristiques : parcours unidirectionnel, suppression et insertion efficaces.

2. Liste Double (DNode)

```c
typedef struct DNode {
    int data;            // Données stockées
    struct DNode *prev;  // Pointeur vers le nœud précédent
    struct DNode *next;  // Pointeur vers le nœud suivant
} DNode;
```

Caractéristiques : parcours bidirectionnel, suppression plus facile.

3. Liste Circulaire Simple (CNode)

```c
typedef struct CNode {
    int data;
    struct CNode *next;  // Le dernier nœud pointe vers le premier
} CNode;
```

Caractéristiques : parcours cyclique, accès rapide à la tête et à la queue.

4. Liste Circulaire Double (DCNode)

```c
typedef struct DCNode {
    int data;
    struct DCNode *prev;  // Chaînage circulaire bidirectionnel
    struct DCNode *next;
} DCNode;
```

Caractéristiques : parcours bidirectionnel cyclique, maximum de flexibilité.

Fonctionnalités

Liste Simple

supprimerOccurrences(Node **head, int val)

Supprime toutes les occurrences d'une valeur donnée dans la liste.

Algorithme :

- Parcourt la liste en maintenant deux pointeurs (actuel et précédent)
- Libère la mémoire des nœuds correspondants
- Réajuste les liens entre les nœuds restants

Complexité : O(n) où n est le nombre d'éléments

insertionTrieeSimple(Node **head, int val)

Insère une valeur en maintenant l'ordre croissant de la liste.

Algorithme :

- Trouve la position d'insertion appropriée
- Crée un nouveau nœud
- Insère le nœud en préservant l'ordre

Complexité : O(n) dans le pire cas

Liste Double

insertionTrieeDouble(DNode **head, int val)

Insère une valeur en ordre croissant dans une liste doublement chaînée.

Avantages :

- Maintient les liens bidirectionnels
- Permet un parcours dans les deux sens
- Facilite les opérations de suppression

Complexité : O(n)

Liste Circulaire Simple

insertionTeteCirculaire(CNode **last, int val)

Insère un élément au début de la liste circulaire.

Particularité : utilise un pointeur sur le dernier élément pour un accès O(1) à la tête.

insertionQueueCirculaire(CNode **last, int val)

Insère un élément à la fin de la liste circulaire.

Avantage : insertion en queue en temps constant O(1).

Liste Circulaire Double

insertionTeteDoubleCirculaire(DCNode **head, int val)

Insère un élément en tête avec maintien des liens circulaires bidirectionnels.

insertionQueueDoubleCirculaire(DCNode **head, int val)

Insère un élément en queue dans la liste circulaire double.

Avantage : combine la flexibilité du chaînage double avec l'efficacité circulaire.

Guide d'utilisation

Menu interactif

Le programme présente un menu avec 6 options :

```
=== MENU LISTES CHAINEES ===
1. Supprimer occurrences (liste simple)
2. Insertion triee (liste simple)
3. Insertion triee (liste double)
4. Insertion tete/queue (liste circulaire simple)
5. Insertion tete/queue (liste circulaire double)
0. Quitter
```

Initialisation des listes

Au démarrage, quatre listes sont pré-initialisées avec 5 éléments chacune, incluant une valeur répétée :

- **Liste simple** : 1 → 2 → 3 → 3 → 4
- **Liste double** : 10 ↔ 10 ↔ 15 ↔ 20 ↔ 25
- **Liste circulaire simple** : 5 → 7 → 5 → 9 → 11 → (retour)
- **Liste circulaire double** : 30 ↔ 40 ↔ 35 ↔ 30 ↔ 45 → (retour)

Déroulement d'une opération

1. Sélectionnez une option du menu (1-5)
2. Entrez les valeurs demandées
3. Le programme affiche l'ancienne liste
4. L'opération est effectuée
5. Le programme affiche la nouvelle liste

Exemples d'exécution

Exemple 1 : Suppression d'occurrences

```
Choix : 1
Valeur a supprimer : 3
l'ancienne liste etait: 1 -> 2 -> 3 -> 3 -> 4 -> NULL
la nouvelle liste est: 1 -> 2 -> 4 -> NULL
```

Résultat : toutes les occurrences de 3 ont été supprimées.

Exemple 2 : Insertion triée dans liste simple

```
Choix : 2
Valeur a inserer : 2
l'ancienne liste etait : 1 -> 2 -> 4 -> NULL
Nouvelle liste : 1 -> 2 -> 2 -> 4 -> NULL
```

Résultat : la valeur 2 est insérée en maintenant l'ordre croissant.

Exemple 3 : Insertion dans liste double

```
Choix : 3
Valeur a inserer : 12
l'ancienne liste etait : 10 <-> 10 <-> 15 <-> 20 <-> 25 <-> NULL
la nouvelle liste est : 10 <-> 10 <-> 12 <-> 15 <-> 20 <-> 25 <-> NULL
```

Résultat : insertion triée avec maintien des liens bidirectionnels.

Exemple 4 : Insertion en tête (liste circulaire)

```
Choix : 4
1. Inserer en tete, 2. Inserer en queue : 1
Valeur a inserer : 3
l'ancienne liste etait : 5 -> 7 -> 5 -> 9 -> 11 -> (retour tête)
la nouvelle liste est : 3 -> 5 -> 7 -> 5 -> 9 -> 11 -> (retour tête)
```

Exemple 5 : Insertion en queue (liste circulaire double)

```
Choix : 5
1. Inserer en tete, 2. Inserer en queue : 2
Valeur a inserer : 50
l'ancienne liste etait : 30 <-> 40 <-> 35 <-> 30 <-> 45 <-> (retour tête)
la nouvelle liste est: 30 <-> 40 <-> 35 <-> 30 <-> 45 <-> 50 <-> (retour tête)
```

Compilation et exécution

Prérequis

- Compilateur C (gcc, clang, etc.)
- Système d'exploitation : Linux, macOS, Windows (avec MinGW)

Compilation

```bash
gcc tp2.c -o tp2
```

Ou avec des options de débogage :

```bash
gcc -Wall -Wextra -g tp2.c -o tp2
```

Exécution

```bash
./tp2
```

Sous Windows :

```cmd
tp2.exe
```

Points importants

- Le programme libère automatiquement la mémoire lors de la suppression de nœuds
- Les listes sont initialisées automatiquement au démarrage
- Pour quitter le programme, sélectionnez l'option 0
- Les valeurs sont affichées avant et après chaque opération pour vérification

Gestion de la mémoire

Le programme utilise `malloc()` pour l'allocation dynamique et `free()` pour la libération. Dans un environnement de production, il faudrait ajouter des fonctions de libération complète de chaque liste avant la fin du programme.

Notes techniques

Complexité temporelle des opérations

Opération	Liste Simple	Liste Double	Liste Circulaire
Insertion triée	O(n)	O(n)	-
Suppression	O(n)	O(n)	-
Insertion tête	O(1)	O(1)	O(1)
Insertion queue	O(n)	O(1)	O(1)

Complexité spatiale

Toutes les listes ont une complexité spatiale de O(n) où n est le nombre d'éléments stockés.

Auteur : Programme pédagogique TP2
Date : 2025
Version : 1.0