Université de Yaoundé I
Faculté des Sciences
Département d'Informatique

RAPPORT DE TRAVAUX PRATIQUES

INF231 : Structure de Données II

TP2 : Programme de Manipulation de Listes Chainées en C

Responsable : Pr. Melatagia
Matière : INF231 - Structure de Données II
Année académique : 2025-2026

Groupe de TP

Nom	Prénom	Matricule
KENGNI DIEKONG	BERNARD LOIC	23V2146
PETANG	DANIEL	23V2121
MBEZELE ZOA	DANIELLE NAOMI	24G2837
ABDEL ADY TCHALLA NGANDO		23V2538
AZAMBOU	MARTHE NEFERTYTI	23V2357
TEKENG KAMWELE	JUNIOR CAMBELL	23U2686

2 octobre 2025

Table des matières

1 Introduction **1**

2 Structure du programme **2**
2.1 Compilation et exécution 2
2.2 Menu principal 2
2.3 Gestion des entrées 2

3 Dossiers algorithmiques **2**
3.1 Listes simplement chainées non circulaires 2
3.1.1 Ajout d'un élément dans une liste triée 2
3.1.2 Suppression de toutes les occurrences d'un élément 3
3.2 Listes simplement chainées circulaires 4
3.2.1 Insertion en tête 4
3.2.2 Insertion en queue 5
3.3 Listes doublement chainées non circulaires 6
3.3.1 Ajout d'un élément dans une liste triée 6
3.4 Listes doublement chainées circulaires 7
3.4.1 Insertion en tête 7
3.4.2 Insertion en queue 8

4 Conclusion **9**

Annexes **10**

1 Introduction

Ce rapport présente les travaux pratiques réalisés dans le cadre du cours de Structure de Données II (INF231) sous la direction du Professeur Melatagia. L'objectif de ce TP était de développer un programme en C permettant de manipuler différents types de listes chainées à travers diverses opérations fondamentales.

Le programme implémente huit fonctionnalités principales organisées en quatre catégories :

1. **Listes simplement chainées non circulaires** :
   - — Ajout d'un élément dans une liste triée
   - — Suppression de toutes les occurrences d'un élément
   - — Affichage de la liste
2. **Listes simplement chainées circulaires** :
   - — Insertion en tête
   - — Insertion en queue
   - — Affichage de la liste
3. **Listes doublement chainées non circulaires** :
   - — Ajout d'un élément dans une liste triée
   - — Suppression de toutes les occurrences d'un élément
   - — Affichage de la liste

4. **Listes doublement chainées circulaires** :
   — Insertion en tête
   — Insertion en queue
   — Affichage de la liste

Ce document détaille pour chaque fonctionnalité le problème abordé, le principe de résolution, le dictionnaire de données, l'algorithme utilisé et l'analyse de complexité. Il présente également la structure du programme et les choix d'implémentation.

2 Structure du programme

Le programme est organisé en trois fichiers principaux :

— `main.c` : Contient la fonction principale et le menu interactif hiérarchique
— `tp2.h` : Contient les déclarations des structures de données et des fonctions
— `tp2.c` : Contient l'implémentation des fonctions de manipulation des listes

2.1 Compilation et exécution

Le programme peut être compilé à l'aide de la commande suivante :

```
make
```

L'exécution se fait ensuite avec :

```
./main
```

2.2 Menu principal

Le programme propose un menu interactif hiérarchique permettant à l'utilisateur de sélectionner le type de liste à manipuler, puis les opérations spécifiques. Une boucle principale assure la continuité de l'exécution jusqu'à ce que l'utilisateur choisisse de quitter le programme.

2.3 Gestion des entrées

Une attention particulière a été portée à la robustesse de la gestion des entrées utilisateur. La fonction `lireEntier` assure la validation des entrées numériques en rejetant les caractères non autorisés et en vérifiant les plages de valeurs.

3 Dossiers algorithmiques

3.1 Listes simplement chainées non circulaires

3.1.1 Ajout d'un élément dans une liste triée

Problème Insérer un nouvel élément dans une liste simplement chainée tout en maintenant l'ordre croissant des éléments.

Principe Parcourir la liste jusqu'à trouver la position d'insertion appropriée où la valeur du nœud précédent est inférieure et celle du nœud suivant est supérieure à la valeur à insérer.

Dictionnaire de données

- t : pointeur vers la tête de liste
- nouveau : nouveau nœud à insérer
- courant : pointeur de parcours de la liste
- v : valeur à insérer

Algorithme

```
Allouer nouveau nœud
nouveau->val = v
nouveau->next = NULL

Si liste vide alors
    retourner nouveau

Si v <= tête->val alors
    nouveau->next = tête
    retourner nouveau

courant = tête
Tant que courant->next != NULL et courant->next->val < v
    courant = courant->next

nouveau->next = courant->next
courant->next = nouveau
Retourner tête
```

Complexité

- Temps : O(n) dans le pire cas
- Espace : O(1)
- Totale : O(n)

3.1.2 Suppression de toutes les occurrences d'un élément

Problème Supprimer toutes les occurrences d'une valeur donnée dans une liste simplement chainée.

Principe Parcourir la liste et supprimer chaque nœud contenant la valeur cible en maintenant la cohérence des liens.

Dictionnaire de données

- t : pointeur vers la tête de liste
- courant, precedent : pointeurs de parcours
- aSupprimer : pointeur vers le nœud à libérer
- v : valeur à supprimer

Algorithme

```
courant = tête
precedent = NULL
Tant que courant != NULL
    Si courant->val = v alors
        aSupprimer = courant
        Si precedent = NULL alors
            tête = courant->next
        Sinon
            precedent->next = courant->next
        courant = courant->next
        Libérer aSupprimer
    Sinon
        precedent = courant
        courant = courant->next
Retourner tête
```

Complexité

- Temps : O(n)
- Espace : O(1)
- Totale : O(n)

3.2 Listes simplement chainées circulaires

3.2.1 Insertion en tête

Problème Insérer un nouvel élément au début d'une liste simplement chainée circulaire.

Principe Créer un nouveau nœud et le lier à la tête actuelle, puis mettre à jour le dernier nœud pour qu'il pointe vers le nouveau nœud.

Dictionnaire de données

- t : pointeur vers la tête de liste
- nouveau : nouveau nœud à insérer
- last : pointeur vers le dernier nœud
- v : valeur à insérer

Algorithme

```
Allouer nouveau nœud
nouveau->val = v

Si liste vide alors
    nouveau->next = nouveau
    retourner nouveau
```

```
last = tête
Tant que last->next != tête
    last = last->next

nouveau->next = tête
last->next = nouveau
Retourner nouveau
```

- Temps : O(n) pour trouver le dernier nœud
- Espace : O(1)
- Totale : O(n)

3.2.2 Insertion en queue

Problème Insérer un nouvel élément à la fin d'une liste simplement chainée circulaire.

Principe Créer un nouveau nœud et le lier à la tête, puis mettre à jour l'ancien dernier nœud pour qu'il pointe vers le nouveau nœud.

Dictionnaire de données

- t : pointeur vers la tête de liste
- nouveau : nouveau nœud à insérer
- last : pointeur vers le dernier nœud
- v : valeur à insérer

Algorithme

```
Allouer nouveau nœud
nouveau->val = v

Si liste vide alors
    nouveau->next = nouveau
    retourner nouveau

last = tête
Tant que last->next != tête
    last = last->next

nouveau->next = tête
last->next = nouveau
Retourner tête
```

Complexité

- Temps : O(n) pour trouver le dernier nœud
- Espace : O(1)
- Totale : O(n)

3.3 Listes doublement chainées non circulaires

3.3.1 Ajout d'un élément dans une liste triée

Problème Insérer un nouvel élément dans une liste doublement chainée tout en maintenant l'ordre croissant.

Principe Parcourir la liste pour trouver la position d'insertion et mettre à jour les liens précédent et suivant des nœuds adjacents.

Dictionnaire de données

- t : pointeur vers la tête de liste
- nouveau : nouveau nœud à insérer
- courant : pointeur de parcours
- v : valeur à insérer

Algorithme

```
Allouer nouveau nœud
nouveau->val = v
nouveau->prev = NULL
nouveau->next = NULL

Si liste vide alors
    retourner nouveau

Si v <= tête->val alors
    nouveau->next = tête
    tête->prev = nouveau
    retourner nouveau

courant = tête
Tant que courant->next != NULL et courant->next->val < v
    courant = courant->next

nouveau->next = courant->next
nouveau->prev = courant
Si courant->next != NULL alors
    courant->next->prev = nouveau
courant->next = nouveau
Retourner tête
```

Complexité

- Temps : O(n) dans le pire cas
- Espace : O(1)
- Totale : O(n)

Suppression de toutes les occurrences d'un élément

Problème Supprimer toutes les occurrences d'une valeur donnée dans une liste doublement chainée.

Principe Parcourir la liste et pour chaque occurrence, mettre à jour les liens des nœuds précédent et suivant avant de libérer le nœud courant.

Dictionnaire de données

- t : pointeur vers la tête de liste
- courant : pointeur de parcours
- aSupprimer : pointeur vers le nœud à libérer
- v : valeur à supprimer

Algorithme

```
courant = tête
Tant que courant != NULL
    Si courant->val = v alors
        aSupprimer = courant
        Si courant->prev = NULL alors
            tête = courant->next
            Si tête != NULL alors
                tête->prev = NULL
        Sinon
            courant->prev->next = courant->next
        Si courant->next != NULL alors
            courant->next->prev = courant->prev
        courant = courant->next
        Libérer aSupprimer
    Sinon
        courant = courant->next
Retourner tête
```

Complexité

- Temps : O(n)
- Espace : O(1)
- Totale : O(n)

3.4 Listes doublement chainées circulaires

3.4.1 Insertion en tête

Problème Insérer un nouvel élément au début d'une liste doublement chainée circulaire.

Principe Créer un nouveau nœud et l'insérer entre le dernier nœud et la tête actuelle, puis mettre à jour tous les liens concernés.

Dictionnaire de données

- t : pointeur vers la tête de liste
- nouveau : nouveau nœud à insérer
- last : pointeur vers le dernier nœud
- v : valeur à insérer

Algorithme

```
Allouer nouveau nœud
nouveau->val = v

Si liste vide alors
    nouveau->next = nouveau
    nouveau->prev = nouveau
    retourner nouveau

last = tête->prev
nouveau->next = tête
nouveau->prev = last
tête->prev = nouveau
last->next = nouveau
Retourner nouveau
```

- Temps : O(1)
- Espace : O(1)
- Totale : O(1)

3.4.2 Insertion en queue

Problème Insérer un nouvel élément à la fin d'une liste doublement chainée circulaire.

Principe Créer un nouveau nœud et l'insérer entre le dernier nœud et la tête, puis mettre à jour les liens.

Dictionnaire de données

- t : pointeur vers la tête de liste
- nouveau : nouveau nœud à insérer
- last : pointeur vers le dernier nœud
- v : valeur à insérer

Algorithme

```
Allouer nouveau nœud
nouveau->val = v

Si liste vide alors
```

```
    nouveau->next = nouveau
    nouveau->prev = nouveau
    retourner nouveau

last = tête->prev
nouveau->next = tête
nouveau->prev = last
last->next = nouveau
tête->prev = nouveau
Retourner tête
```

Complexité

- Temps : O(1)
- Espace : O(1)
- Totale : O(1)

4 Conclusion

Ce TP a permis de mettre en œuvre diverses opérations fondamentales sur les listes chainées en langage C. Chaque algorithme a été analysé en termes de complexité temporelle et spatiale, permettant de comprendre leurs performances relatives.

Les principales difficultés rencontrées ont concerné la gestion des pointeurs dans les listes circulaires et doublement chainées, particulièrement lors des opérations de suppression où il faut maintenir la cohérence des liens entre les nœuds. La solution implémentée avec des fonctions de validation des entrées et une gestion rigoureuse de la mémoire assure une expérience utilisateur fiable.

Ce travail illustre l'importance des structures de données dynamiques et des algorithmes de manipulation dans la résolution de problèmes complexes, et démontre la capacité du langage C à implémenter efficacement ces structures de bas niveau.

Annexes

Annexe A : Code source

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tp2.h"

int main() {
    printf("\t\tBIENVENUE DANS LE MENU PRINCIPAL\n");
    char input[100];
    char n;
    ListeSChainee t1 = NULL;
    ListeDChainee t2 = NULL;

    do {
        printf("0- Quitter\n1- Effectuez des opérations dans une liste simplement chaînée non circulaire\n2- Effectuez des opérations dans une liste simplement chaînée circulaire\n3- Effectuez des opérations dans une liste doublement chaînée non circulaire\n4- Effectuez des opérations dans une liste doublement chaînée circulaire\n");
        printf("Entrez le chiffre correspondant à votre choix : ");

        if (fgets(input, sizeof(input), stdin) == NULL) {
            break;
        }

        if (strlen(input) > 2 || (strlen(input) == 1 && input[0] == '\n')) {
            printf("Erreur ! Veuillez entrer un seul caractère.\n");
            continue;
        }

        n = input[0];

        switch(n) {

            case '1':
                do {
                    printf("0- Quitter\n1- Ajouter un élément dans la liste\n2- Lire un élément et supprimer toutes les occurences dans la liste\n3- Afficher la liste\n");
                    printf("Entrez le chiffre correspondant à votre choix : ");

                    if (fgets(input, sizeof(input), stdin) == NULL) {
                        break;
                    }

                    if (strlen(input) > 2 || (strlen(input) == 1 && input[0] == '\n')) {
                        printf("Erreur ! Veuillez entrer un seul caractère.\n");
                        continue;
                    }

                    n = input[0];

                    switch(n) {
                        case '1': t1 = ajoutElementSC(t1); break;
                        case '2': t1 = suppOccurenceSC(t1); break;
                        case '3': affichageSC(t1); break;
                        case '0': printf("Au Revoir !\n"); break;
                        default: printf("Erreur ! Veuillez choisir une option valable.\n");
                    }
                    printf("\n");
                } while(n != '0');
            break;

```

```
57              case '2':
58                  do {
59                      printf("0- Quitter\n1- Ajouter un élément en tête de la
    liste\n2- Ajouter un élément en queue de la liste\n3- Afficher la liste\n")
    ;
60                      printf("Entrez le chiffre correspondant à votre choix : ");
61
62                      if (fgets(input, sizeof(input), stdin) == NULL) {
63                          break;
64                      }
65
66                      if (strlen(input) > 2 || (strlen(input) == 1 && input[0] ==
     '\n')) {
67                          printf("Erreur ! Veuillez entrer un seul caractère.\n")
    ;
68                          continue;
69                      }
70
71                      n = input[0];
72
73                      switch(n) {
74                          case '1': t1 = ajoutTeteSCC(t1); break;
75                          case '2': t1 = ajoutQueueSCC(t1); break;
76                          case '3': affichageSCC(t1); break;
77                          case '0': printf("Au Revoir !\n"); break;
78                          default: printf("Erreur ! Veuillez choisir une option
    valable.\n");
79                      }
80                      printf("\n");
81                  } while(n != '0');
82              break;
83
84              case '3':
85                  do {
86                      printf("0- Quitter\n1- Ajouter un élément dans la liste\n2-
     Lire un élément et supprimer toutes les occurences dans la liste\n3-
    Afficher la liste\n");
87                      printf("Entrez le chiffre correspondant à votre choix : ");
88
89                      if (fgets(input, sizeof(input), stdin) == NULL) {
90                          break;
91                      }
92
93                      if (strlen(input) > 2 || (strlen(input) == 1 && input[0] ==
     '\n')) {
94                          printf("Erreur ! Veuillez entrer un seul caractère.\n")
    ;
95                          continue;
96                      }
97
98                      n = input[0];
99
100                     switch(n) {
101                         case '1': t2 = ajoutElementDC(t2); break;
102                         case '2': t2 = suppOccurenceDC(t2); break;
103                         case '3': affichageDC(t2); break;
104                         case '0': printf("Au Revoir !\n"); break;
105                         default: printf("Erreur ! Veuillez choisir une option
    valable.\n");
106                     }
107                     printf("\n");
108                 } while(n != '0');
109             break;
110
111             case '4':
112                 do {
113                     printf("0- Quitter\n1- Ajouter un élément en tête de la
    liste\n2- Ajouter un élément en queue de la liste\n3- Afficher la liste\n")
    ;
114                     printf("Entrez le chiffre correspondant à votre choix : ");
115
116                     if (fgets(input, sizeof(input), stdin) == NULL) {
117                         break;
```

```
                    }

                    if (strlen(input) > 2 || (strlen(input) == 1 && input[0] == '\n')) {
                        printf("Erreur ! Veuillez entrer un seul caractère.\n");
                        continue;
                    }

                    n = input[0];

                    switch(n) {
                        case '1': t2 = ajoutTeteDCC(t2); break;
                        case '2': t2 = ajoutQueueDCC(t2); break;
                        case '3': affichageDCC(t2); break;
                        case '0': printf("Au Revoir !\n"); break;
                        default: printf("Erreur ! Veuillez choisir une option valable.\n");
                    }
                    printf("\n");
                } while(n != '0');
            break;

            case '0': printf("Au Revoir !\n"); break;
            default: printf("Erreur ! Veuillez choisir une option valable.\n");
        }
        printf("\n");
    } while(n != '0');

    return 0;
}
```

tp2.h

```
#ifndef _TP2_
#define _TP2_

typedef struct ListeSChainee{
    int val;
    struct ListeSChainee *next;
}*ListeSChainee;
typedef struct ListeDChainee{
    struct ListeDChainee *prev;
    int val;
    struct ListeDChainee *next;
}*ListeDChainee;
int lireEntier(const char* message, int min, int max);
ListeSChainee ajoutElementSC(ListeSChainee t);
ListeDChainee ajoutElementDC(ListeDChainee t);
ListeSChainee suppOccurenceSC(ListeSChainee t);
ListeDChainee suppOccurenceDC(ListeDChainee t);
void affichageSC(ListeSChainee t);
void affichageDC(ListeDChainee t);
void affichageSCC(ListeSChainee t);
void affichageDCC(ListeDChainee t);
ListeSChainee ajoutTeteSCC(ListeSChainee t);
ListeDChainee ajoutTeteDCC(ListeDChainee t);
ListeSChainee ajoutQueueSCC(ListeSChainee t);
ListeDChainee ajoutQueueDCC(ListeDChainee t);

#endif
```

tp2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
```

```c
#include <string.h>
#include <errno.h>
#include "tp2.h"

#define MAX 100

int lireEntier(const char* message, int min, int max) {
    char buffer[100];
    long valeurLong;
    char *endptr;

    do {
        printf("%s", message);
        if (fgets(buffer, sizeof(buffer), stdin) == NULL) {
            printf("Erreur de lecture.\n");
            continue;
        }

        // Supprimer le saut de ligne
        buffer[strcspn(buffer, "\n")] = '\0';

        // Convertir en long
        errno = 0;
        valeurLong = strtol(buffer, &endptr, 10);

        // Vérifications
        if (endptr == buffer) {
            printf("Erreur: Veuillez entrer un nombre entier valide.\n");
        } else if (*endptr != '\0') {
            printf("Erreur: Caractères supplémentaires non autorisés.\n");
        } else if (errno == ERANGE || valeurLong < min || valeurLong > max) {
            printf("Erreur: La valeur doit être comprise entre %d et %d (valeur saisie: %ld).\n", min, max, valeurLong);
        } else {
            return (int)valeurLong;
        }
    } while (1);
}

ListeSChainee ajoutElementSC(ListeSChainee t){
    int v = lireEntier("Entrez l'entier à ajouter: ", 0, MAX);
    ListeSChainee nouveau = (ListeSChainee) malloc(sizeof(ListeSChainee));
    if (nouveau == NULL) {
        printf("Erreur d'allocation mémoire\n");
        return t;
    }
    nouveau->val = v;
    nouveau->next = NULL;
    if (t == NULL) {
        return nouveau;
    }
    if (t->val >= v) {
        nouveau->next = t;
        return nouveau;
    }
    ListeSChainee courant = t;
    while (courant->next != NULL && courant->next->val < v) {
        courant = courant->next;
    }
    nouveau->next = courant->next;
    courant->next = nouveau;
    return t;
}

ListeDChainee ajoutElementDC(ListeDChainee t){
    int v = lireEntier("Entrez l'entier à ajouter: ", 0, MAX);
    ListeDChainee nouveau = (ListeDChainee) malloc(sizeof(ListeDChainee));
    if (nouveau == NULL) {
        printf("Erreur d'allocation mémoire\n");
        return t;
    }
    nouveau->val = v;
    nouveau->prev = NULL;
```

```
    nouveau->next = NULL;
    if (t == NULL) {
        return nouveau;
    }
    if (t->val >= v) {
        nouveau->next = t;
        t->prev = nouveau;
        return nouveau;
    }
    ListeDChainee courant = t;
    while (courant->next != NULL && courant->next->val < v) {
        courant = courant->next;
    }
    nouveau->next = courant->next;
    nouveau->prev = courant;
        if (courant->next != NULL) {
        courant->next->prev = nouveau;
    }
    courant->next = nouveau;
    return t;
}

ListeSChainee suppOccurenceSC(ListeSChainee t){
    int v = lireEntier("Entrez l'entier à supprimer: ", 0, MAX);
    ListeSChainee courant = t;
    ListeSChainee precedent = NULL;
    ListeSChainee aSupprimer;
    int suppressions = 0;
    while (courant != NULL) {
        if (courant->val == v) {
            aSupprimer = courant;
            if (precedent == NULL) {
                t = courant->next;
            } else {
                precedent->next = courant->next;
            }
            courant = courant->next;
            free(aSupprimer);
            suppressions++;
        } else {
            precedent = courant;
            courant = courant->next;
        }
    }
    if (suppressions > 0) {
        printf("%d occurrence(s) supprimée(s) avec succès !\n", suppressions);
    } else {
        printf("Aucune occurrence trouvée.\n");
    }
    return t;
}

ListeDChainee suppOccurenceDC(ListeDChainee t){
    int v = lireEntier("Entrez l'entier à supprimer: ", 0, MAX);
    ListeDChainee courant = t;
    ListeDChainee aSupprimer;
    int suppressions = 0;
    while (courant != NULL) {
        if (courant->val == v) {
            aSupprimer = courant;
            if (courant->prev == NULL) {
                t = courant->next;
                if (t != NULL) {
                    t->prev = NULL;
                }
            } else {
                courant->prev->next = courant->next;
            }
            if (courant->next != NULL) {
                courant->next->prev = courant->prev;
            }
            courant = courant->next;
            free(aSupprimer);
```

```
150             suppressions++;
151         } else {
152             courant = courant->next;
153         }
154     }
155     if (suppressions > 0) {
156         printf("%d occurrence(s) supprimée(s) avec succès !\n", suppressions);
157     } else {
158         printf("Aucune occurrence trouvée.\n");
159     }
160     return t;
161 }
162
163 void affichageSC(ListeSChainee t){
164     ListeSChainee l = t;
165     if (l == NULL) {
166         printf("Liste vide\n");
167     } else {
168         while (l != NULL) {
169             printf("\"%d\"  ", l->val);
170             l = l->next;
171         }
172         printf("\n");
173     }
174 }
175
176 void affichageDC(ListeDChainee t){
177     ListeDChainee l = t;
178     if (l == NULL) {
179         printf("Liste vide\n");
180     } else {
181         while (l != NULL) {
182             printf("\"%d\"  ", l->val);
183             l = l->next;
184         }
185         printf("\n");
186     }
187 }
188
189 void affichageSCC(ListeSChainee t){
190     if (t == NULL) {
191         printf("Liste vide\n");
192         return;
193     }
194     ListeSChainee l = t;
195     printf("\"%d\"  ", l->val);
196     l = l->next;
197     while (l != t) {
198         printf("\"%d\"  ", l->val);
199         l = l->next;
200     }
201     printf("\n");
202 }
203
204 void affichageDCC(ListeDChainee t){
205     if (t == NULL) {
206         printf("Liste vide\n");
207         return;
208     }
209     ListeDChainee l = t;
210     printf("\"%d\"  ", l->val);
211     l = l->next;
212     while (l != t) {
213         printf("\"%d\"  ", l->val);
214         l = l->next;
215     }
216     printf("\n");
217 }
218
219 ListeSChainee ajoutTeteSCC(ListeSChainee t){
220     int v = lireEntier("Entrez l'entier à ajouter: ", 0, MAX);
221     ListeSChainee m = (ListeSChainee) malloc (sizeof(ListeSChainee));
222     if(m == NULL){
```

```
223         printf("Erreur d'allocation de mémoire\n");
224         return t;
225     }
226     m->val = v;
227     if(t == NULL){
228         m->next = m;
229         return m;
230     }
231     ListeSChainee last = t;
232     while(last->next != t){
233         last = last->next;
234     }
235     m->next = t;
236     last->next = m;
237     return m;
238 }
239
240 ListeSChainee ajoutQueueSCC(ListeSChainee t){
241     int v = lireEntier("Entrez l'entier à ajouter: ", 0, MAX);
242     ListeSChainee m = (ListeSChainee) malloc (sizeof(ListeSChainee));
243     if(m == NULL){
244         printf("Erreur d'allocation de mémoire\n");
245         return t;
246     }
247     m->val = v;
248     if(t == NULL){
249         m->next = m;
250         return m;
251     }
252     ListeSChainee last = t;
253     while (last->next != t) {
254         last = last->next;
255     }
256     m->next = t;
257     last->next = m;
258     return t;
259 }
260
261 ListeDChainee ajoutTeteDCC(ListeDChainee t){
262     int v = lireEntier("Entrez l'entier à ajouter: ", 0, MAX);
263     ListeDChainee m = (ListeDChainee) malloc (sizeof(ListeDChainee));
264     if(m == NULL){
265         printf("Erreur d'allocation de mémoire\n");
266         return t;
267     }
268     m->val = v;
269     if(t == NULL){
270         m->next = m;
271         m->prev = m;
272         return m;
273     }
274     ListeDChainee last = t->prev;
275     m->next = t;
276     m->prev = last;
277     t->prev = m;
278     last->next = m;
279     return m;
280 }
281
282 ListeDChainee ajoutQueueDCC(ListeDChainee t){
283     int v = lireEntier("Entrez l'entier à ajouter: ", 0, MAX);
284     ListeDChainee m = (ListeDChainee) malloc (sizeof(ListeDChainee));
285     if(m == NULL){
286         printf("Erreur d'allocation de mémoire\n");
287         return t;
288     }
289     m->val = v;
290     if(t == NULL){
291         m->next = m;
292         m->prev = m;
293         return m;
294     }
295     ListeDChainee last = t->prev;
```

```
296     m->next = t;
297     m->prev = last;
298     last->next = m;
299     t->prev = m;
300     return t;
301 }
```

Annexe B : Makefile

```
CC = gcc
CFLAGS = -Wall -Wextra
OBJECTS = main.o tp2.o

all: main

main: $(OBJECTS)
 $(CC) $(CFLAGS) -o main $(OBJECTS)

main.o: main.c tp2.h
 $(CC) $(CFLAGS) -c main.c

tp2.o: tp2.c tp2.h
 $(CC) $(CFLAGS) -c tp2.c

clean:
 rm -f main $(OBJECTS)
```