**Prof :elie Hamouche année : 2017**

**Structures de données**

**TD5 : Gestion des listes doublement chainées**

Les listes doublement chaînées sont des structures de données semblables aux [listes simplement chaînées](http://www.commentcamarche.net/faq/sujet-7444-liste-simplement-chainee)  ou la liaison entre les éléments se fait grâce à 2 pointeurs (un qui pointe vers l'élément précédent et un qui pointe vers l'élément suivant).

Ecrire un programme C/C++ permettant la gestion d’une liste doublement chainee illustrée comme suit **:**

NULL

8

NULL

U3

U2

U1

U0 =1

elem prev next

START

NULL

8

U5

U4

NULL

elem prev next

**dl1  : Liste doublement chainée de la suite Un**

dl1 : contient les termes de de la suite définie par**: Un = Un-1+ 2\*Un-2**

**Avec U0=1 U1=2**

dl2 : contient les termes de Newton des racines carres √a définis par :

**U0= 1 si n ==0**

**U1 = (U0+a/U0)/ 2 si n = 1**

**Un +1= (Un+a/Un)/2 si n>1**

1. Définir une cellule  **(node)** contenant une donnée de type int et 2 pointeurs \*prev et \*next sont chacun de type **node .**
2. Construire ensuite une structure doublement chainée **Dlist** définie par 2 pointeurs \*first, \*last sont chacun de type **node**  et d’une taille Length de type int ?
3. Ecrire des fonctions récursives qui évaluent **Un et NEWTON** ?

***Ecrire les opérations(fonctions ou procédures) permettant* :**

1. La création d’une liste double dl vide **Dlist\* dlist\_creation()** ?
2. de vérifier si une liste **dl** est vraiment vide d’après:**bool estvide(Dlist \*dl){..}?**
3. d’ajouter les termes de **Un et Racine de Newton**  a une liste doublement chainée **dl**  en utilisant la récursivité  **?**
4. d’insérer une cellule contenant **une donnée en tête de la liste dl** : **Dlist\* dlist\_AjoutTete(Dlist \*dl, int data){..}** ?
5. d’insérer une cellule contenant une **donnée à la fin** de la liste dl :
   1. **Dlist\* dlist\_AjoutQueue(Dlist \*dl, int d nata){..}?**
6. de supprimer un élément de la liste selon 3 possibilités :

* **L’élément se trouve en tete de liste,**
* **l' élément se trouve en fin de liste ,**
* **l’ élément se touve aune position donnée .**

1. de rechercher un élément selon sa valeur dans dl  **Dlist\* dlist\_find(Dlist \*dl, int od)**

1. d’inverser les éléments de la liste dl : **Dlis**t**\* dlist\_reverse(Dlist \*dl){..}** ?
2. d’afficher par **itération/recursivité** les éléments de la liste dl : **void dlist\_displayI(Dlist \*dl){..}?**

**void dlist\_displayR(Dlist \*dl){..}?**

1. d’afficher les éléments inverse de la liste double ?
2. d’additionner la somme les valeurs dans les cellules paires et impaires ?
3. Ecrie une fonction ou procédure permettant de remplacer les valeurs des éléments pairs de la liste  **dl1**  **par 2 éléments dont les valeurs sont égales à la moitié de cette valeur** ?
4. Ecrire une fonction qui permet de **supprimer les nombres dupliqués** ?
5. Ecrire une fonction qui va **chercher les éléments en doublons** et les remplacer par un élément dont la **valeur est le double** ?
6. Ecrire une fonction permettant **de diviser(éclater)** la liste double en 2 autres . **la 1ere contenant les nombres <10 et la 2eme contenant les nombres >=10 ? compter-les** ?
7. **libérer une liste double** de la mémoire ?
8. Ecrire le **programme principal main()** qui fait appel aux fonctions indiquées ci-dessus?

**Corrigé C++**

#include "stdafx.h"

#include <iostream.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

using namespace std;

// question a et b Définition structure doublement chainée

typedef struct node

{

int data;

struct node \*next;

struct node \*prev;

}node;

typedef struct dl

{

size\_t length;

struct node \*last;

struct node \*first;

} Dlist;

// dl1: contient les termes de la suite Un

int SuiteR(int n) {

int res = (n == 0) ? 1 : (n == 1) ? 2 : SuiteR(n - 1) + 2 \* SuiteR(n - 2);

return res;

}

// dl2: contient les termes des racines carrés de Newton

float Newton(int a, int n) {

int res;

if (n == 0)

{

res = 1;

}

else if (n == 1) {

res = (1 + a / 1) / 2;

}

else if (n>1) {

res = (Newton(a, n - 1) + a / Newton(a, n - 1)) / 2;

}

else

{

return -1;

}

return res;

}

//1. Allocation une nouvelle liste

Dlist\* dlist\_creation()

{

Dlist \*nouv = (Dlist\*)malloc(sizeof(Dlist));

if (nouv != NULL)

{

nouv->length = 0;

nouv->first = NULL;

nouv->last = NULL;

}

return nouv;

}

//2. Ajout en début de liste

Dlist\* dlist\_AjoutTete(Dlist \*dl, int od)

{

if (dl != NULL)

{

node \*nouv = (node\*)malloc(sizeof(node));

nouv->data = od;

nouv->prev = NULL;

if (dl->last == NULL)

{

nouv->next = NULL;

dl->first = nouv; // on fait pointer la 1ere cellule vers nouv

dl->last = nouv; // on fait pointer la dernière vers nouv

}

else

{

dl->first->prev = nouv; // On relie le 1er élément de la liste vers nouv

nouv->next = dl->first; // On fait pointer next de nouv vers le 1er élément

dl->first = nouv; // On fait pointer le début de liste vers le nouv

}

dl->length++;

}

return dl;

}

/\*3. Ajout en fin de liste : 2 possibilités s'offrent à nous :

• Si liste est vide Alors on fait pointer prev vers NULL

puis on fait pointer la tête et la fin de liste vers le nouvel élément

• Sinon

- Nous rattachons le dernier élément de dl à nouvel élément

- on fait pointer prev vers le dernier élément de notre liste

- on fait pointer la fin de liste vers notre nouvel élément (fin du chaînage)

\*/

Dlist\* dlist\_AjoutQueue(Dlist \*dl, int od)

{

if (dl != NULL) // On vérifie si la liste a été allouée

{

node \*nouv = (node\*)malloc(sizeof(node)); // Création d'un nouveau node

nouv->data = od; // On enregistre la donnée

nouv->next = NULL; // On fait pointer next vers NULL

if (dl->last == NULL)

{

nouv->prev = NULL; // On fait pointer prev vers NULL

dl->first = nouv; // On fait pointer la tête de liste vers le nouvel élément

dl->last = nouv; // On fait pointer la fin de liste vers le nouvel élément

}

else

dl->last->next = nouv; // On relie le dernier élément de la liste vers notre nouv

nouv->prev = dl->last; // On fait pointer prev vers le dernier élément de la liste

dl->last = nouv; // On fait pointer la fin de liste vers notre nouvel élément

}

dl->length++; /\* Incrémentation de la taille de la liste \*/

}

return dl;

}

/\* 4. Insérer un élément, 3 cas se présentent :

- Soit nous sommes en fin de liste

- Soit nous sommes en début de liste

- Soit nous sommes en milieu de liste

- \*/

Dlist\* dlist\_insertPosition(Dlist \*dl, int od, int pos)

{

if (dl != NULL)

{

node \*tmp = dl->first;

int i = 1;

while (tmp != NULL && i <= pos)

{

if (pos == i)

{

if (tmp->next == NULL)

{

dl = dlist\_AjoutQueue(dl, od);

}

else if (tmp->prev == NULL)

{

dl = dlist\_AjoutTete(dl, od);

}

else

{

node \*nouv = (node\*)malloc(sizeof(node));

nouv->data = od;

tmp->next->prev = nouv;

tmp->prev->next = nouv;

nouv->prev = tmp->prev;

nouv->next = tmp;

dl->length++;

}

}

else

{

tmp = tmp->next;

}

i++;

}

}

return dl;

}

/\*5. Supprimer un élément d'une liste. 3 possibilités s'offriront à nous:

- l'élément se trouve en fin de liste

- l'élément se trouve en début de liste

- l'élément se trouve en milieu de liste

\*/

Dlist\* dlist\_removeElem(Dlist \*dl, int od)

{

if (dl != NULL)

{

node \*tmp = dl->first;

int found = 0;

while (tmp != NULL && !found)

{

if (tmp->data == od)

{

if (tmp->next == NULL)

{

dl->last = tmp->prev;

dl->last->next = NULL;

}

else if (tmp->prev == NULL)

{

dl->first = tmp->next;

dl->first->prev = NULL;

}

else

{

tmp->next->prev = tmp->prev; // on avance le pointeur

tmp->prev->next = tmp->next; // on avance le pointeur

}

delete(tmp);

dl ->length--;

found = 1;

}

else

{

tmp = tmp->next;

}

}

}

return dl;

}

//6.Supprimer un élément selon sa pos

Dlist\* dlist\_removePos(Dlist \*dl, int pos)

{

if (dl != NULL)

{

node \*tmp = dl->first;

int i = 1;

while (tmp != NULL && i <= pos)

{

if (pos == i)

{

if (tmp->next == NULL)

{

dl->last = tmp->prev;

dl->last->next = NULL;

}

else if (tmp->prev == NULL)

{

dl->first = tmp->next;

dl->first->prev = NULL;

}

else

{

tmp->next->prev = tmp->prev;

tmp->prev->next = tmp->next;

}

delete(tmp);

dl->length--;

}

else

{

tmp = tmp->next;

}

i++;

}

}

return dl;

}

//7.Recherche un élément selon sa valeur

Dlist\* dlist\_find(Dlist \*dl, int od)

{

Dlist \* ret = NULL;

if (dl != NULL)

{

node \*tmp = dl->first;

int found = 0;

while (tmp != NULL && !found)

{

if (tmp->data == od)

{

ret = dlist\_creation();

ret = dlist\_AjoutQueue(ret, od);

found = 1;

}

else

{

tmp = tmp->next;

}

}

}

return ret;

}

//8.fonction permettant "d'inverser" une liste chaînée.

Dlist\* dlist\_reverse(Dlist \*dl)

{

Dlist \*ret = NULL;

if (dl != NULL)

{

node \* tmp = dl->last;

ret = dlist\_creation();

while (tmp != NULL)

{

ret = dlist\_AjoutQueue(ret, tmp->data);

tmp = tmp->prev;

}

}

return ret;

}

//9. affichage liste doublement chainée itérative

void dlist\_display(Dlist \*dl)

{

if (dl != NULL)

{

node \*tmp = dl->first;

while (tmp != NULL)

{

cout << "la donnee :" << tmp->data << "\n";

tmp = tmp->next;

}

}

cout << "\n";

}

//9. affichage liste doublement chainée Récursive

void dlist\_displayR(Dlist \*dl)

{

node \* tmp = dl->first;

if (tmp != NULL)

{

cout << "la donnee :" << tmp->data << "\n";

dlist\_displayR(tmp ->next);

}

}

//10. Affichage inverse des éléments de DList

Dlist\* DisplayReverse(Dlist\* dl) {

if (dl == NULL) return NULL;

else {

cout << "la donnée est " << dl->last->data << "\n";

DisplayReverse(dl->last->prev);

}

}

// 11. Addition des valeurs des cellules paires te impaires

void dlist\_sumPairImpairR(Dlist \*dl)

{

int si, sp;

si = sp = 0;

if (dl != NULL)

{

node \* tmp = dl->first;

int i = 1;

if (tmp != NULL)

{

if (i % 2 == 0) {

sp += tmp->data;

}

else {

si += tmp->data;

dlist\_sumPairImpairR(tmp ->next);

i++;

}

}

cout << "la somme des valeurs dans cellules paires :" << sp << "la somme des valeurs dans cellules impaires :" << si << "\n";

}

//12. fonction ou procédure permettant de remplacer les valeurs des éléments pairs de la liste dl1 par 2 éléments dont les valeurs sont égales à la moitié de cette valeur

Dlist\* ReplacePairNumber(Dlist \*dl) {

node \* tmp = dl->first;

node \* newNode = new node;

while (tmp != NULL) {

if (tmp->data % 2 == 0) {

if (tmp == dl->first) {

newNode->data = tmp->data / 2;

newNode->next = tmp;

newNode->prev = NULL;

tmp->data = tmp->data / 2;

tmp->prev = newNode;

dl->first = newNode;

}else if (tmp == dl->last) {

tmp->data = tmp->data / 2;

tmp->next = newNode;

tmp->prev = NULL;

newNode->data = tmp->data;

newNode ->prev = tmp;

newNode ->next = NULL;

dl->last = newNode;

}else {

node \* tmp1 = (node\*)malloc(sizeof(node));

tmp->data = tmp->data / 2;

tmp1->data = tmp->data;

tmp1->next = tmp->next;

tmp1->prev = tmp;

tmp1->next->prev = tmp;

tmp->next = tmp1;

}

}

}

tmp = tmp->next;

}

return dl;

}

// 13. fonction / procédure permettant d’éliminer les doublons de la liste Dlist, c.à.d.supprimer

// les valeurs entières qui sont répétées 2 ou plusieurs fois et ne garder qu’une seule occurrence de chaque nombre.

void deleteDoublons(Dlist \*dl){

node \*prev = new node;

node \*cur = new node;

if (dl != NULL) {

prev = dl->first;

prev->data = dl->first->data;

cur = dl->first->next;

while (cur != NULL){

if (cur->data == prev->data){

prev->next = cur->next; // avancer pointeur

delete(cur);

cur = prev->next;

}

else{

prev = cur;

}

cur = cur->next;

}

}

cout << "les doublons sont supprimés " << "\n";

}

//14. Fonction chercher les éléments en doublons et les remplacer par un élément dont la valeur est le double

Dlist \* getNumberTogether(Dlist \*dl) {

node \* tmp = dl->first;

while (tmp != NULL && tmp->next != NULL)

{

if (tmp->data == tmp->next->data) {

if (tmp == dl->first) { //1er cas

(tmp->next)->next->prev = tmp;

tmp->next = tmp->next->next;

tmp->data = 2 \* tmp->data;

}

else if (tmp->next == dl->last) { //2eme cas

tmp->next = NULL;

tmp->data = 2 \* tmp->data;

}

else { //3eme cas

tmp->next->next->prev = tmp;

tmp->next = tmp->next->next;

tmp->data = 2 \* tmp->data;

}

}

}

tmp = tmp->next;

}

return dl;

}

//15. fonction permettant de diviser(éclater) les produits en une pile P et une file F de nœuds.la pile contiendra les produits ayant une masse <10 et la File les masses >= 10.vous pouvez utiliser les fonctions et procédures

// de gestion de la Pile(empiler, dépiler, ..) et de la file F(enfiler, defiler, ..)

void Eclater(Dlist \*dl) {

Pile \* P = new Pile;

File \* F = new File;

if (dl != NULL) {

node\* tmp = dl->first;

while (tmp != NULL) {

if (tmp->d < 10)

{

Empiler(P, d);

P = P->tete->next;

}

else if (tmp->d >= 10){

Enfiler(F, d);

F = F->tail->next;

}

else{

cout << "Nothing " << "\n";

}

tmp = tmp->next;

}

}

// displayPile(P);

// dispayFile(F);

}

//16. Libérer une liste

void dlist\_liberer(Dlist \*\*dl)

{

if (\*dl != NULL)

{

node \*tmp = (\*dl)->first;

while (tmp != NULL)

{

node \*del = tmp;

tmp = tmp->next;

delete(del);

}

delete(\*dl);

\*dl = NULL;

}

}

//programme principal

int main()

{

Dlist \* dl1, \*dl2;

// ajout en tete

int n, x;

dl1 = dlist\_creation();

dl2 = dlist\_creation();

cout << "donner le dernier terme n :" << "\n";

cin >> n;

cout << "donner le valeur de x comme entier :" << "\n";

cin >> x;

for (int i = 0; i<n; i++)

{

int res = SuiteR(x, i);

dl1 = dlist\_AjoutTete(dl1, SuiteR(i);

dl2 = dlist\_AjoutTete(dl2, Newton(x, i));

}

cout << "les listes doubles dl1,dl2 contiennent :" << "\n";

cout << "------------------------------" << "\n";

dlist\_display(dl1);

dlist\_display(dl2);

// Insertion aune position donnée de L2

dl1 = dlist\_insert(dl1, 100, 5);

cout << "la liste double dl1 contient après insertion :" << "\n";

cout << "------------------------------" << "\n";

dlist\_display(dl1);

// Supprime un élément de dl

dl2 = dlist\_removeElem(dl1, 1);

cout << "la liste double dl1 contient après la suppression d'un élément:" << "\n";

ob.dlist\_display(dl1);

cout << "------------------------------" << "\n";

// Recherche d'un élément de dl1

dl1 = dlist\_find(dl1, 12);

cout << "la recherche d'un élément dans dl1 donne :" << "\n";

//10.sum paires impaires de dl1

dlist\_sumPairImpairR(dl1);

// Inverse de la liste dl1

dlist\_reverse(dl1);

cout << "l'inverse de la liste dl1 donne :" << "\n";

dlist\_display(dl1);

cout << "------------------------------" << "\n";

cout << "la liste liberee de dl1 :" << "\n";

dlist\_liberer(dl1);

cout << "programme terminée ...." << "\n";

system("PAUSE");

return 0;

}