**Prof : elie Hamouche année : 2019**

**Structures de données**

**TD3\_bis : listes simplement chainées des couples**

Soi L une Liste simplement chainée qui continent les couples de termes **(clé,valeur) = (i,i/i!**  selon l’ordre illustré comme suit :

3 /3!

1 /1!

2 /2!

4 4/4! /979!C!C

5 5/5! !!9!C!C

**List simplement chainée des couples**

On envisage construire une liste des couples (i,) indiqué ci-haut .

1. définir une structure liste simplement **chainée L** contenant une donnée couple(i, ) à parir de la fonction récursive et un pointeur vers le couple suivant ***.***

***Ecrire les fonctions et procédures permettant :***

1. d’ évaluer par itération la **fact(n)** définie par la relation :

**n! = 1 si n=0 ou n = 1**

**n\* (n-1)! sinon**

1. de donner la **taille** par recursivité d’une liste simplement chainée **?**
2. calculer par récursivité la **somme Σ i/i!**  dans la liste ? déduire la valeur e? calculer la valeur Σ i/i! associée au **clés pairs ?**
3. soit **p(x) =**

ou i : représente la clé

ai : représente la valeur

* Construire une structure polynôme pouvant être remplacée par une liste chaînée ou chaque cellule de la liste correspond à un monôme, avec son coefficient et son exposant

P(x) = **5/5! \*x^5 + 4/4! \*x^4 + 3/3! \*x3+ 2/2! \*x^2 + 1/1! \*x**

|  |  |
| --- | --- |
| 2/2 ! | 2 |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |  |
| --- | --- |
| 5/5 ! | 5 |

|  |  |
| --- | --- |
| 4/4 ! | 4 |

|  |  |
| --- | --- |
| 3/3 ! | 3 |

|  |  |
| --- | --- |
| 1/1 ! | 1 |

ploynôme **P**

* Écrire une fonction de saisie et d’affichage d’un polynôme.

1. on suppose que la liste L n’est pas triée dans l’ordre, **ajouter un couple nouveau (i,i/i!)** à la liste triée par ordre croissant ?
2. **de vérifier par recusivité** qu’un couple i/ i! appartient à la liste **L** ?
3. **de renvoyer le kieme** couple de la liste ? déduire le couple a **l’avant-dernier** de la liste **L** ?
4. **d’ incrémenter chaque couple** de la liste **L** de son rang (la clé en position 0 n’est pas modiﬁé, celle en position 1 est incrémentée de 1 et en position 2 de 2, etc.). Par exemple, sur la liste 9; 7; 5, 3, 1 la fonction doit renvoyer comme clé nouveaux 9,8; 6; 4; 2 ?copier cette nouvelle liste obtenue dans une autre ?
5. **de renverser (Mirror)** les couples de L par la récursivité ?
6. **de supprimer par recusivité une occurrence** qui correspond au couple (i/i!) ?
7. **séparer la liste L en 2 autres L1 et L2**  contenant respectivement des termes clés pairs et impairs ?
8. **d’échanger toutes les clés impairs (swap)** par des clés paires ?
9. **Afficher** par récursivité **les couples de la list L ?**

**Coorigé**

#include "stdafx.h"

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

using namespace std;

//1 structres couple et List

typedef struct c {

int cle;

float val;

}couple;

typedef struct List

{

couple c;

List \* next;

}List;

typedef struct m{

float x; //coeff

int n; // degre

}monome;

typedef struct cellule{

monome m;

struct cellule \*next;

}Polynome;

class TD4

{

private:

public:

TD4(){}

int factR(int n)

{

TD4 ob;

int res = (n == 0 || n == 1) ? 1 :n\*ob.factR(n - 1) ;

return res;

}

int rapport(int n)

{

TD4 ob;

return (n/ob.factR(n));

}

// 3. la taille par recursivité d’une liste simplement chainée ? calculer par récursivité la somme Σ (i,k) dans la liste ?

int TailleR(List \* L)

{

TD4 ob;

int res = (L == NULL) ? 0 : 1 + ob.TailleR(L->next);

return res;

}

//4. calculer par récursivité la somme Σ i/i! dans la liste ? déduire la valeur e

float SumR(List \* L,int i)

{

TD4 ob;

couple c;

float s = 0;

float res = (L == NULL) ? 0 : s +=c.cle / ob.factR(i) + ob.SumR(L->next,i);

return res;

}

// sum itérative

float SumI(List \* L)

{

TD4 ob;

List \* pl = L;

couple c;

float s = 0;

int i = 1;

while (pl != NULL){

s += i / ob.factR(i);

pl = pl->next;

}

return s;

}

//déduire la valeur e

float expo(List \*L)

{

TD4 ob;

float res = ob.SumI(L);

return res;

}

// fonction saisie d'un monôme

monome saisirMonome(){

monome om;

cout << "enter coef : "<<"\n";

cin >> om.x;

cout << "enter degre : "<<"\n";

cin >> om.n;

return om;

}

//fonction qui crée un monôme à partir d’un coefficient et d’un degré

Polynome \* CreePolynome(int coeff, int degre)

{

Polynome \*op = new Polynome;

op->m.x= coeff;

op->m.n = degre;

op->next = NULL;

return(op);

}

// fonction qui affiche Monôme

void afficherMonome(monome \*om){

cout << om->x << "x^" << om->n ;

}

// fontion qui affiche polynôme

void afficherPolynome(Polynome\* l){

TD4 ob;

Polynome\* pl;

pl = l;

while (pl != NULL){

ob.afficherMonome(&pl->m);

if (pl->next != NULL)

cout << "+";

else

cout << "\n";

pl = pl->next;

}

}

// calcul la valeur Σ i/i! associée aux clés pairs

float SumRpair(List \* L,int i)

{

TD4 ob;

float s = 0;

couple c;

if (L == NULL) return 0;

else if (L->c.cle % i == 0)

{

c.val = ob.factR(i);

s += L->c.cle /c.val ;

return s;

}

else

{

return(ob.SumRpair(L->next,i));

}

}

//5 d’ajouter un couple nouveau (i,!i/i) à la liste triée par ordre croissant

List \* insertElement\_trie(List \* L, couple oc)

{

List \* suiv, \*pl;

List \* nouv = new List;

nouv->c = oc;

nouv->next = NULL;

if (L == NULL) return nouv;

pl = L;

suiv = pl->next;

if (pl->c.cle < oc.cle) { // si la 1ere donnée inferieure a la donnée a insérer

if (suiv == NULL) {

pl->next = nouv;

return L;

}

}

else { // si la donnée > a la donnée a insérer

nouv->next = pl;

return nouv;

}

while (suiv->next != NULL && suiv->c.cle < oc.cle){

pl = pl->next;

suiv = pl->next;

}

if (suiv->c.cle >oc.cle)

{

pl->next = nouv;

nouv->next = suiv;

}

else{

suiv->next = nouv;

}

return L;

}

//6 de vérifier qu’un couple i/i! appartient à la liste L

bool appartientR(List \*L, couple oc)

{

TD4 ob;

List \*pl = L;

if (pl == NULL) return false;

else if (pl->c.cle == oc.cle) return true;

else

return(ob.appartientR(pl->next, oc));

}

//7. de renvoyer le kieme couple de la liste ? Déduire le couple a l’avant-dernier de la liste L

List \* KIemeElement(List \*l, int k){

List \*pl;

pl = l;

int i = 1;

while (pl != NULL && i < k) {

i = i + 1;

pl = pl->next;

}

return pl;

}

// avant dernier

List \* Avantdernier(List \* L)

{

List \* pl = L->next;

List \* avant = L;

while (pl->next != NULL)

{

pl = pl->next;

avant = avant->next;

}

return avant;

}

//8.Incrémenter chaque couple de la liste L de son rang (clé en position 0 n’est pas modiﬁé, celui en position 1 est incrémenté de 1,

//celui en position 2 de 2, etc.).Par exemple, sur la liste 9; 7; 5, 3, 1 la fonction doit renvoyer comme clé nouveaux 9, 8; 6; 4; 2 ?

//copier cette nouvelle liste obtenue dans une autre ?

List \* Incremente(List \* L)

{

List \* pl = L->next;

while (pl != NULL)

{

pl->c.cle = ((pl->c.cle) + 1);

pl = pl->next;

}

return L;

}

//9. de renverser(Mirror) les couples de L par la récursivité

List \* ReverseR(List \* L)

{

TD4 ob;

if (L->next == NULL)

{

return L;

}

else

{

List \* tmp;

tmp = ob.ReverseR(L->next);

L->next->next = L;

L->next = NULL;

return(tmp);

}

}

//10. supprimer une occurrence correspondant au couple (i, i/i!) Par la récursivité

List \* SupprimeOccR(List \* L, couple oc)

{

TD4 ob;

if (L == NULL)

{

return NULL;

}

else

{

if (L->c.cle == oc.cle )

{

return (ob.SupprimeOccR(L->next, oc));

}

else

{

L->next = ob.SupprimeOccR(L->next, oc);

return L;

}

}

}

// Ajout couple en queue de la liste

List \* insertQueue(List \*L, couple oc){

TD4 ob;

List \*nouv = new List;

List \*pl;

nouv->c = oc; // donnee de la nouvelle cellule

nouv->next = NULL;

if (L == NULL) { // cas si la liste est vide

L = nouv;

}

else

{

// Recherche de la dernière cellule

for (pl = L; pl->next != NULL; pl = pl->next){}

pl->next = nouv; // chainage

}

return L;

}

//11. Séparer la liste L en 2 autres L1 et L2 contenant respectivement des termes clés pairs et impairs

void SeparerList(List \* L)

{

TD4 ob;

List \*pl = L;

List \*L1, \*L2;

L1 = NULL;

L2 = NULL;

while (pl->next != NULL)

{

if (pl->c.cle % 2 != 0)

{

L1 = ob.insertQueue(L1, pl->c);

}

else

{

L2 = ob.insertQueue(L2, pl->c);

}

pl = pl->next;

}

ob.affiche\_listR(L);

}

//12. Échanger toutes les clés impaires (swap) par des clés paires

List \* Swap(List \* L)

{

List \* pl = L;

while (pl != NULL)

{

if (((pl->c.cle) % 2) != 0)

{

pl->c.cle = ((pl->c.cle) + 1);

}

pl = pl->next;

}

return L;

}

//13. Afficher par récursivité les couples de la List L

void affiche\_listR(List \*l) {

TD4 ob;

if (l != NULL) {

cout << "cle : "<< l->c.cle <<" val: "<< l->c.val << "\n";

ob.affiche\_listR(l->next);

}

cout << "\n";

}

}; // fin class

int main(int argc, char \*argv[])

{

TD4 ob;

int n;

couple c;

List \* L = new List;

L = NULL;

List \* Lt = NULL;

List \* Lk = NULL;

List \* La = NULL;

List \* Li = NULL;

List \* Lr = NULL;

List \* Ls = NULL;

List \* Lp = NULL;

List \* Le = NULL;

couple c1 = { 1, 1 / ob.factR(1) };

couple c2 = { 2, 2 / ob.factR(2) };

couple c3 = { 3, 3 / ob.factR(3) };

couple c4 = { 4, 4 / ob.factR(4) };

couple c5 = { 5, 5 / ob.factR(5) };

L = ob.insertQueue(L, c1) ;

L = ob.insertQueue(L, c2);

L = ob.insertQueue(L, c3);

L = ob.insertQueue(L, c4);

L = ob.insertQueue(L, c5);

List \* pl = L;

cout << "\n\nLa liste simplement chaine :\n\n ";

int i = 1;

while (pl != NULL)

{

c.val = ob.rapport(i);

cout << "cle :"<< pl->c.cle <<"valeur :"<< pl->c.val << "\n";

pl = pl->next;

}cout << "\n\n";

n = ob.TailleR(L);

cout << "\n\nLa taille de la liste est : " << n << "\n\n";

//somme Σ i/i!

float res1 = ob.SumR(L, 5);

cout << "\n\n La somme Σ i/i! récursive: " << res1 << "\n\n";

//somme Σ i/i!

float s = ob.SumI(L);

cout << "\n\nLa somme Σ i/i! itérative: " << s << "\n\n";

float exp = ob.expo(L);

cout << "la valeur expo : "<<exp<< "\n";

// ajouter un couple trie

cout << "\n\Ajout et tri : " << "\n\n";

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

couple c\_n;

c\_n.cle = 6;

c.val = ob.rapport(6);

Lt= ob.insertElement\_trie(L, c);

ob.affiche\_listR(Lt);

//vérifier c(cle,val) appartient a la liste L

cout << "\n\ verif c(cle,val) appartenance a la liste L " << "\n\n";

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

bool res2 = ob.appartientR(L, c);

cout << "\n\n le couple(6,6.61) existe" << res2 << "\n\n";

//de renvoyer le kieme couple

cout << "\n\ kieme couple la liste L " << "\n\n";

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

Lk= ob.KIemeElement(L, 4);

ob.affiche\_listR(Lk);

//de renvoyer l'avant dernier de L

cout << "\n\ l'avant dernier de L " << "\n\n";

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

La = ob.Avantdernier(L);

ob.affiche\_listR(La);

//incrémenter chaque couple

cout << "\n\incrémenter chaque couple " << "\n\n";

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

Li= ob.Incremente(L);

ob.affiche\_listR(Li);

//renverser (Mirror)

cout << "\n\(Mirror) " << "\n\n";

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

Lr = ob.ReverseR(L);

ob.affiche\_listR(Lr);

// de supprimer par récursivité une occurrence

cout << "\n\supprimer par récursivité " << "\n\n";

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

Ls= ob.SupprimeOccR(L, c);

ob.affiche\_listR(Ls);

//séparer la liste L en 2 autres L1 et L2

cout << "\n\séparer la liste L en 2 autres L1 et L2 " << "\n\n";

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

ob.SeparerList(L);

// échanger toutes les clés impairs (swap)

cout << "\n\échanger toutes les clés impairs (swap) " << "\n\n";

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

Le = ob.Swap(L);

ob.affiche\_listR(Le);

getchar();

}