**Dosar de laborator**

**Ghimpu Lucian Eduard**

**Grupa 223 2/2**

**Documentatie Laborator 1 PLf (R1)**

**Nume: Ghimpu Lucian Eduard**

**Grupa: 223 2/2**

**Data: 15.10.2017**

**Cerinta R1:**

**11.**

**a.** Să se determine numărul format prin însumarea elementelor de ordin par ale unei liste, din care se scad elementele de ordin impar ale listei.

**b**. Să se determine diferența a două mulțimi reprezentate sub formă de lista.

**Model Matematic pentru Recursivitate:**

a)

rezultat(l1…ln) =

b)

cautare(e, l1…ln) =

diferenta(l1…ln, l1’…ln’) =

Cod Sursa:

linkedlist.h:

#pragma once

#include <iostream>

#include <stdlib.h>

using namespace std;

//Struct pentru Nod

template <typename T>

struct Node

{

T value;

Node\* next;

};

//Struct pentru Lista

template <typename T>

struct LinkedList

{

Node<T>\* first = nullptr;

};

//Functie wrapper pentru crearea recursiva

template <typename T>

LinkedList<T> creare()

{

LinkedList<T> temp;

temp.first = createRec<T>();

return temp;

}

//Creaza recursiv o lista, input de la utilizator

template <typename T>

Node<T>\* createRec()

{

T elem;

cout << "Type a element: " << endl;

cin >> elem;

if (elem == 0)

{

return nullptr;

}

else

{

Node<T>\* n = new Node<T>;

n->value = elem;

n->next = createRec<T>();

return n;

}

}

//Destructor de lista, wrapper function

template <typename T>

void destroyLinkedList(LinkedList<T> l)

{

destroyRec(l.first);

}

//Destructor recursiv

template <typename T>

void destroyRec(Node<T>\* n)

{

if (n != nullptr)

{

destroyRec(n->next);

delete n;

}

}

//Wrapper function pentru printare

template <typename T>

void printList(LinkedList<T> l)

{

printRec(l.first);

cout << endl;

}

//Printare recursiva

template <typename T>

void printRec(Node<T>\* n)

{

if (n != nullptr)

{

cout << n->value << " ";

printRec(n->next);

}

}

/\*

Să se determine numărul format prin însumarea elementelor de ordin par

ale unei liste, din care se scad elementele de ordin impar ale listei.

\*/

template <typename T>

int cerinta11a(LinkedList<T> l)

{

return sumaRec11a(l.first);

}

template <typename T>

int sumaRec11a(Node<T>\* n)

{

if (n == nullptr)

{

return 0;

}

else if (n->next == nullptr)

{

return n->value\*(-1);

}

else

{

return n->value\*(-1) + n->next->value + sumaRec11a(n->next->next);

}

}

//Wrapper pentru cautare

template <typename T>

bool cautare(LinkedList<T> l, T elem)

{

return cautareRec(l.first, elem);

}

//Cautare recursiva

template <typename T>

bool cautareRec(Node<T>\* n, T elem)

{

if (n == nullptr)

{

return false;

}

else if (n->value == elem)

{

return true;

}

else

{

return cautareRec(n->next, elem);

}

}

//Wrapper pentru adaugare, primeste un TElement si creaza un nod care v-a fi adaugat

template <typename T>

LinkedList<T> adaugare(LinkedList<T> l, T elem)

{

Node<T>\* nou = new Node<T>;

nou->value = elem;

nou->next = nullptr;

adaugareRec<T>(l.first, nou);

return l;

}

//Wrapper function pentru adaugare, dar returneaza nod, nu lista (folosit pentru a putea face adaugare recursiva in diferenta de multimi)

template <typename T>

Node<T>\* adaugareReturn(Node<T>\* l, Node<T>\* n)

{

if (l == nullptr)

{

l = n;

return l;

}

if (n == nullptr)

{

return l;

}

else

{

adaugareRec(l, n);

return l;

}

}

//Adaugare recursiva

template <typename T>

void adaugareRec(Node<T>\* n, Node<T>\* elem)

{

if (n == nullptr)

{

n = elem;

}

else if (n->next == nullptr)

{

n->next = elem;

}

else

{

adaugareRec(n->next, elem);

}

}

// Să se determine diferența a două mulțimi reprezentate sub formă de lista

template <typename T>

LinkedList<T> diferenta(LinkedList<T> l1, LinkedList<T> l2)

{

LinkedList<T> l3;

l3.first = diferentaRec(l1.first, l2.first);

return l3;

}

template <typename T>

Node<T>\* diferentaRec(Node<T>\* l1, Node<T>\* l2)

{

if (l1 == nullptr)

{

return nullptr;

}

else if (cautareRec(l2, l1->value) == false)

{

Node<T>\* nou = new Node<T>;

nou->value = l1->value;

nou->next = nullptr;

return adaugareReturn(nou, diferentaRec(l1->next, l2));

}

else

{

return diferentaRec(l1->next, l2);

}

}

main.cpp:

#include "linkedList.h"

#include <stdlib.h>

int main()

{

//test create, print

LinkedList<int> l1;

LinkedList<int> l2;

cout << "Introdu prima lista:" << endl;

l1 = creare<int>();

cout << "Introdu a 2 a lista:" << endl;

l2 = creare<int>();

//test cerinta 11a

printList(l1);

printList(l2);

int n11 = cerinta11a(l1);

cout << "Suma nr poz pare, dif nr poz impare pt l1: " << n11 << endl;

//test cautare

cout << cautare(l1, 99) << endl;

//test adaugare

l1=adaugare(l1, 56);

l1 = adaugare(l1, 57);

l1 = adaugare(l1, 58);

l1 = adaugare(l1, 59);

printList(l1);

printList(l2);

//test diferenta (11b)

LinkedList<int> l3;

l3 = diferenta(l1, l2);

printList(l3);

destroyLinkedList(l1);

destroyLinkedList(l2);

destroyLinkedList(l3);

return 0;

}

**Documentatie Laborator 1 PLf (L1)**

**Nume: Ghimpu Lucian Eduard**

**Grupa: 223 2/2**

**Data: 15.10.2017**

**14.**

**a.** Sa se scrie un predicat care testeaza egalitatea a doua multimi, fara sa se faca apel la diferenta a doua multimi.

**b.** Definiti un predicat care selecteaza al n-lea element al unei liste.

**Model Matematic pentru Recursivitate:**

a)

membru(E,l1…ln)=

submultime(l1…ln,l1’…ln’)=

egale(l1..ln,l1’…ln’) =

b)

nth(l1…ln,N,E) =

**Cod Sursa + explicatii problema 14 a**

%Verifica daca un element E se afla intr-o lista

%member(integer,list)

%(i,i)

member(E,[E|\_]).

member(E,[\_|T]):-member(E,T).

%verifica daca o multime M1 e submultime a lui M2

%submultime(list,list)

%(i,i)

submultime([],\_).

submultime([H1|T1],L2):-

member(H1,L2),

submultime(T1,L2).

%Verifica daca 2 multimi sunt egale

%egal(list,list)

%(i,i)

egal(L1,L2):-submultime(L1,L2),submultime(L2,L1).

**Cod Sursa + explicatii problema 14 b**

%returneaza al n-lea element dintr-o lista

%nth(list,integer,integer)

%nth(lista data, pozitia, parametru output)

%(i,i,o)

nth([H|\_],1,E):-

E=H.

nth([\_|T],N,E):-

N1 is N-1,nth(T,N1,E).

**Date de teste:**

a: egal([1,2,3],[1,2,3]) -> true

egal([1,2,3],[1,2,3,4]) -> false

b: nth([1,2,3,4,5],1,X) -> X=1

nth([1,2,3,4,5],6,X) -> false

**Documentatie Laborator P2 PLf (P2)**

**Nume: Ghimpu Lucian Eduard**

**Grupa: 223 2/2**

**Data: 31.10.2017**

**7.**

a) Definiti un predicat care determina produsul unui numar reprezentat

cifra cu cifra intr-o lista cu o anumita cifra. De ex: [1 9 3 5 9 9] \* 2

=> [3 8 7 1 9 8]

b) Se da o lista eterogena, formata din numere intregi si maximum 9 liste de

numere intregi. Sa se inlocuiasca fiecare sublista cu rezultatul

inmultirii sublistei cu numarul de ordine al sublistei (prima sublista cu

1, a 2-a cu 2, etc.). De ex:

[1, [2, 3], [4, 1, 4], 3, 6, [7, 5, 1, 3, 9], 5, [1, 1, 1], 7] =>

[1, [2, 3], [8, 2, 8], 3, 6, [2, 2, 5, 4, 1, 7], 5, [4, 4, 4], 7]

**Model Matematic pentru Recursivitate:**

a)

adaugaFinal(e, l1…ln) =

invers(l1…ln) =

inmultireRec(l1…ln,e,ct)

cerintaA(l1…ln,e) = invers(inmultireRec(invers(l1…ln),e,0)).

b)

cerintaB(l1…ln, index) =

**Cod Sursa + explicatii 7 a**

%functie pentru a adauga la finalul unei liste un element

%(I,I,o)

adaugaFinal(E,[],[E]).

adaugaFinal(E,[H|T],[H|L]):-adaugaFinal(E,T,L).

%functie pentru a inversa o lista

%(I,o)

invers([],[]).

invers([H|T],L):-invers(T,L1),adaugaFinal(H,L1,L).

%functia care inmulteste un numar scris sub forma de lista cu o cifra

%inmultireRec(list,integer,integer,list)

%inmultireRec(lista data, cifra de inmultit, cifra de transport, rezultat)

%(I,I,I,o)

inmultireRec([],\_,CT,[]):-CT is 0.

inmultireRec([],\_,CT,[CT]):-CT>0.

inmultireRec([H|T],E,CT,[H1|RL]):-

H1 is (mod((H\*E)+CT, 10)),

CT1 is ((H\*E)+CT) div 10,

inmultireRec(T,E,CT1,RL).

%functie wrapper pentru inmultirea recursiva

%(list,list,integer)

%(lista data, rezultat, cifra de inmultit)

%(I,o,I)

cerintaA(L,R,E):-invers(L,L1),inmultireRec(L1,E,0,R1), invers(R1,R).

**Cod Sursa + explicatii 7 b**

%functie care se foloseste de penctua a pe o lista eterogena

%cerintaB(list,integer,list)

%ceritaB(list data, indexul de inceput, rezultat)

%(I,I,o)

cerintaB([],\_,[]).

cerintaB([H|T],I,R):-

integer(H),

cerintaB(T,I,R1),

R=[H|R1].

cerintaB([H|T],I,R):-

\+ integer(H),

I1 is I+1,

cerintaB(T,I1,R1),

R=[H1|R1],

cerintaA(H,H1,I1).

finalB(L,R):-cerintaB(L,0,R).

**Date de test:**

7a: cerintaA([9,9],X,9) -> [8,9,1]

7b: finalB([1,[2,3],[4,1,4],5,[2]],X) -> X=[1,[2,3],[8,2,8],5,[6]]

**Documentatie Laborator P3 PLf (P3)**

**Nume: Ghimpu Lucian Eduard**

**Grupa: 223 2/2**

**Data: ?**

**6.**

Sa se genereze toate sirurile de n paranteze ce se inchid corect. Exemplu: n=4 sunt 2 solutii: (()) si ()()

**Model Matematic pentru Recursivitate:**

{ vid, daca d==0 and i==0

par(d,i,r) = { par(d-1, i+1, r+"("), daca d>0

{ par(d, i-1, r+")"), daca I>0

permutationsPar(n, r) = findall(x, par(n/2, 0, x), r)SSS

**Cod Sursa + explicatii**

%par(int d, int i, list rez)

%(i,i,0)

%returneaza toate permutarile de paranteze, pentur un N dat

%variabila d tine minte parantezele deschide, I pe cele inchise

par(0,0,[]).

par(D,I,['('|R]):-D>0, D1 is D-1, I1 is I+1, par(D1,I1,R).

par(D,I,[')'|R]):-I>0, I1 is I-1, par(D,I1,R).

%permutations(int n, list r)

%(i,o)

% returneaza toate permutarile de paranteze, pentur un N dat

permutationsPar(N, R) :- N2 is N div 2, findall(X, par(N2,0, X), R).

**Data de test:**

permutationsPar(2,X).

X = [['(', ')']].

permutationsPar(4,X).

X = [['(', '(', ')', ')'], ['(', ')', '(', ')']].

**Documentatie Laborator L1**

**Nume: Ghimpu Lucian Eduard**

**Grupa: 223 2/2**

**Data: ?**

6.

a) Sa se scrie de doua ori elementul de pe pozitia a n-a a unei liste liniare. De exemplu, pentru (10 20 30 40 50) si n=3 se va produce (10 20 30 30 40 50).

b) Sa se scrie o functie care realizeaza o lista de asociere cu cele doua liste pe care le primeste. De ex: (A B C) (X Y Z) --> ((A.X) (B.Y) (C.Z)).

c) Sa se determine numarul tuturor sublistelor unei liste date, pe orice nivel. Prin sublista se intelege fie lista insasi, fie un element de pe orice nivel, care este lista. Exemplu: (1 2 (3 (4 5) (6 7)) 8 (9 10)) => 5 (lista insasi, (3 ...), (4 5), (6 7), (9 10)).

d) Sa se construiasca o functie care intoarce numarul atomilor dintr-o lista, de la nivel superficial.

**Cod sursa + model + explicatii**

#|

a) Sa se scrie de doua ori elementul de pe pozitia a n-a a unei liste

liniare. De exemplu, pentru (10 20 30 40 50) si n=3 se va produce (10 20

30 30 40 50).

{ null, daca l1..ln = null

dublare(l1...ln, i, n) = { l1 U l1 U dublare(l2...ln, i+1, n), daca i==n

{ l1 U dublare(l2...ln, i+1, n), daca i!=n

|#

(defun dublare (l i j)

(cond

((null l) nil)

((= i j) (cons (car l) (cons (car l) (dublare (cdr l) (+ i 1) j))))

(t (cons (car l) (dublare (cdr l) (+ i 1) j)))

)

)

(defun dublareAux (l n)

(dublare l 0 n)

)

#|

Sa se scrie o functie care realizeaza o lista de asociere cu cele doua

liste pe care le primeste. De ex: (A B C) (X Y Z) --> ((A.X) (B.Y)

(C.Z)).

perechi(l1...ln l1'...ln') = { null, daca l1=null sau l2=null

{ (l1 U l1') U perechi(l2...ln, l2'...ln')

|#

(defun perechi (l1 l2)

(cond

((or (null l1) (null l2)) nil)

(t (cons (cons (car l1) (car l2)) (perechi (cdr l1) (cdr l2))))

)

)

#|

Sa se determine numarul tuturor sublistelor unei liste date, pe orice

nivel. Prin sublista se intelege fie lista insasi, fie un element de pe

orice nivel, care este lista. Exemplu: (1 2 (3 (4 5) (6 7)) 8 (9 10)) =>

5 (lista insasi, (3 ...), (4 5), (6 7), (9 10)).

{ 1, daca l=null

nrListe(l1...ln) = { nrListe(l2...ln), daca l1 nu e lista

{ nrListe(l1) + nrliste(l2...ln), daca l1 e lista

|#

(defun nrListe (l)

(cond

((null l) 1)

((atom (car l)) (nrListe(cdr l)))

(t (+ (nrListe (car l)) (nrListe (cdr l))))

)

)

#|

Sa se construiasca o functie care intoarce numarul atomilor dintr-o lista,

de la nivel superficial.

{ 0, daca l=null

nrAtomi(l1...ln) = { 1 + nrAtomi(l2...ln), daca l1 e atom

{ nrAtomi(l2...ln), daca l1 nu e atom

|#

(defun nrAtomi (l)

(cond

((null l) 0)

((atom (car l)) (+ 1 (nrAtomi (cdr l))))

(t (nrAtomi (cdr l)))

)

)

**Documentatie Laborator L2**

**Nume: Ghimpu Lucian Eduard**

**Grupa: 223 2/2**

**Data: ?**

**2**. Sa se tipareasca lista nodurilor de pe nivelul k dintr-un arbore de tipul (1).

**Cod sursa + model + explicatii**

#| MODEL MATEMATIC

{ null, daca lista vida

drept(l1...ln,noduri,muchii) = { null, daca noduri = muchii+1

{ l1 U l2 U drept(l3...ln,noduri+1,muchii+l2)

{ null, daca lista vida

stang(l1...ln,noduri,muchii) = { l1...ln, daca noduri = muchii+1

{ stang(l3...ln,nocuri+1,muchii+l2)

{ null, daca lista vida

wrapper(l1...ln, nivel) = { l1, daca nivel = 0

{ wrapper(drept(l3...ln,0,0),nivel-1) U wrapper(stang(l3...ln,0,0),nivel-1)

|#

(defun drept(lista noduri muchii)

(cond

((null lista) nil)

((= noduri (+ 1 muchii)) nil)

(t (append (list (car lista)) (list (cadr lista))

(drept (cddr lista) (+ noduri 1) (+ muchii (cadr lista))))

)

)

)

(defun stang(lista noduri muchii)

(cond

((null lista) nil)

((= noduri (+ 1 muchii)) lista)

(t (stang (cddr lista) (+ noduri 1)

(+ muchii (cadr lista)))

)

)

)

(defun wrapper(lista nivel)

(cond

((null lista) nil)

((= nivel 0) (list (car lista)))

(t (append (wrapper (drept (cddr lista) 0 0) (- nivel 1))

(wrapper (stang (cddr lista) 0 0) (- nivel 1)))

)

)

)

**Date de test:**

(wrapper `(A 2 B 0 C 2 D 0 E 0) `0) = (A)

(wrapper `(A 2 B 0 C 2 D 0 E 0) `1) = (B,C)

**Documentatie Laborator L3**

**Nume: Ghimpu Lucian Eduard**

**Grupa: 223 2/2**

**Data: ?**

5. Definiti o functie care testeaza apartenenta unui nod intr-un arbore n-ar reprezentat sub forma (radacina lista\_noduri\_subarb1... lista\_noduri\_ \_subarbn) Ex: arborelele este (a (b (c)) (d) (e (f))) si nodul este 'b => adevarat

**Cod sursa + Model**

#|

{ null, daca l1 e atom si diferit de e

membru(e,l) = { membru(e,l1) AND memebru(e,l2) ... AND membru(e,ln)

sum(l) = { l1, daca l1 e atom

{ sum(l1) + sum(l2) + ... sum(ln)

|#

(defun membru(e l)

(cond

((and (atom l) (not (equal l e))) 0)

((and (atom l) (equal l e)) 1)

(t (mapcar #'(lambda (a) (membru e a)) l))

)

)

(defun sum (x)

(cond

((atom x) x)

(t (apply '+ (mapcar #'sum x)))

)

)

(defun wrapper (l e)

(sum (membru e l))

)**Date de test:**

(wrapper `(1 (2 (3) (4) ) `2) -> 1

(wrapper `(1 (2 (3) (4) ) `5) -> 0