# ENUNTURI 1

OBSERVATIE: Desigur, voi va trebui sa scrieti in Maude modulele pe care vi le voi cere la lucrarea de control. Enunturile de mai jos au fost date la unele lucrari de control in anii trecuti; limbajul facut atunci la laboratoarele de programare logica era CafeObj-ul.

1) Semigrupa 2412, marti 16-18:

Sa se scrie un modul CafeObj pentru liste de numere naturale, care sa includa modulul NAT predefinit si sa contina doua operatii e2 si nfact, definite ca mai jos, alaturi de operatiile ajutatoare necesare pentru crearea sortului lista de numere naturale (fie acesta denumit Lista) si cele necesare pentru implementarea operatiilor e2 si nfact.
Operatiile e2 si nfact vor avea ca input o lista de numere naturale si ca output tot o lista de numere naturale, deci vom avea:
ops e2 nfact : Lista -> Lista

Pentru orice lista finita L de numere naturale:
e2(L) = lista formata din exponentii maximi ai lui 2 care divid elementele nenule ale lui L, in ordinea lor din L, de exemplu: e2(4 0 12 3 6 0 24 5) = 2 2 0 1 3 0;
nfact(L) = lista formata din elementele lui L de forma N! (= N factorial), cu N natural nenul, de exemplu: nfact(0 1 2 3 4 5 6 7) = 1 2 6 .

2) Semigrupa 2432, marti 18-20:
Sa se scrie un modul CafeObj pentru liste de numere naturale, care sa includa modulul NAT predefinit si sa contina doua operatii nsn si cresc, definite ca mai jos, alaturi de operatiile ajutatoare necesare pentru crearea sortului lista de numere naturale (fie acesta denumit Lista) si cele necesare pentru implementarea operatiilor nsn si cresc.
Operatiile nsn si cresc vor avea ca input o lista de numere naturale si ca output tot o lista de numere naturale, deci vom avea:
ops nsn cresc : Lista -> Lista

Pentru orice lista finita L de numere naturale:
nsn(L) = lista formata din elementele lui L de forma N*(N+1), cu N natural, de exemplu: nsn(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14) = 0 2 6 12;
cresc(L) = sublista crescatoare a lui L formata din: primul element al lui L, fie acesta E1, apoi urmatorul element al lui L care este mai mare sau egal cu E1, fie acesta E2, apoi urmatorul element al lui L care este mai mare sau egal cu E2 s. a. m. d., de exemplu: cresc(4 0 1 5 4 2 3 4 5 1 7) = 4 5 5 7 .

3) Semigrupa 2421, miercuri 14-16:

Sa se scrie un modul CafeObj pentru liste de numere naturale, care sa includa modulul NAT predefinit si sa contina doua operatii inccresc si cp, definite ca mai jos, alaturi de operatiile ajutatoare necesare pentru crearea sortului lista de numere naturale (fie acesta denumit Lista) si cele necesare pentru implementarea operatiilor inccresc si cp.

Operatiile inccresc si cp vor avea ca input o lista de numere naturale si ca output tot o lista de numere naturale, deci vom avea:

ops inccresc cp : Lista -> Lista

Pentru orice lista finita L de numere naturale:

inccresc(L) = sublista crescatoare de lungime maxima a lui L cu care incepe lista L si care este formata din elemente consecutive din L, de exemplu: inccresc(2 5 5 7 10 3 15 20) = 2 5 5 7 10 ;

cp(L) = lista formata din cuburile perfecte din L, de exemplu: cp(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30) = 0 1 8 27 .

4) Grupa 311, vineri 10-12:

Sa se scrie un modul CafeObj pentru liste de numere naturale, care sa includa modulul NAT predefinit si sa contina doua operatii lp3 si cdivu, definite ca mai jos, alaturi de operatiile ajutatoare necesare pentru crearea sortului lista de numere naturale (fie acesta denumit Lista) si cele necesare pentru implementarea operatiilor lp3 si cdivu.

Operatiile lp3 si cdivu vor fi definite astfel:

ops lp3 cdivu : Lista -> Lista

Pentru orice lista finita L de numere naturale:

lp3(L) = lista formata din puterile naturale ale lui 3 din L (de exemplu: lp3(1 14 12 9 27 28 0 1 5 9 1 0) = 1 9 27 1 9 1);

cdivu(L) = sublista K a lui L de lungime maxima care verifica urmatoarele proprietati: lista L incepe cu sublista K, K este formata din elemente consecutive (ca pozitie) din L si, pentru orice element N al listei K diferit de ultimul element al lui K, N divide elementul care ii succede in lista K.

Mentiuni: in modulul predefinit NAT exista:

op _divides_ : NzNat Nat -> Bool

_divides_ din NAT este relatia "divide pe" dintre un natural nenul si un natural. Vom adauga la relatia _divides_ cazurile in care primul argument al relatiei "divide pe" este 0.

Stim ca:

orice numar natural N divide pe 0, inclusiv N = 0 (cazul N = 0 nu este inclus in relatia _divides_ din NAT);

0 nu divide niciun numar natural nenul (nici acest caz nu este inclus in relatia _divides_ din NAT).

In scrierea operatiei cdivu se vor aplica si aceste cazuri. De exemplu:

cdivu(1 1 2 2 4 12 100 1.000 10.000 100.000 1.000.000 10.000.000 100.000.000 1.000.000.000 10.000.000.000 100.000.000.000 1.000.000.000.000) = 1 1 2 2 4 12 (desigur, punctele din scrierea numerelor nu apar in CafeObj);

cdivu(1 1 2 6 60 120 0 0 0 5 1) = 1 1 2 6 60 120 0 0 0.

5) Sa se scrie un modul CafeObj pentru liste de numere naturale, care sa includa modulul NAT predefinit si sa contina doua operatii nrdivcresc si nrdivdescresc, definite ca mai jos, alaturi de operatiile ajutatoare necesare pentru crearea sortului lista de numere naturale (fie acesta denumit Lista) si operatiile ajutatoare necesare pentru implementarea operatiilor nrdivcresc si nrdivdescresc.

Operatiile nrdivcresc si nrdivdescresc vor avea ca input o lista de numere naturale si ca output tot o lista de numere naturale, deci vor fi declarate astfel:

ops nrdivcresc nrdivdescresc : Lista -> Lista

Pentru orice lista finita L de numere naturale:

nrdivcresc(L) = sublista C a lui L de lungime maxima care indeplineste urmatoarele conditii: lista L incepe cu sublista C (cu alte cuvinte C este un prefix al lui L, adica exista o sublista T a lui L astfel incat L = C T = C concatenata cu T) si oricare doua elemente consecutive M si N din lista C (cu M urmat de N in lista C) au proprietatea ca numarul divizorilor naturali ai lui M este mai mic sau egal cu numarul divizorilor naturali ai lui N;

nrdivdescresc(L) = sublista C a lui L de lungime maxima care indeplineste urmatoarele conditii: lista L incepe cu sublista C (cu alte cuvinte C este un prefix al lui L, adica exista o sublista T a lui L astfel incat L = C T = C concatenata cu T) si oricare doua elemente consecutive M si N din lista C (cu M urmat de N in lista C) au proprietatea ca numarul divizorilor naturali ai lui M este mai mare sau egal cu numarul divizorilor naturali ai lui N.

OBSERVATIE: Numarul divizorilor naturali ai lui 0 este infinit.

OBSERVATIE: Orice N natural nenul are doar un numar finit de divizori naturali, care, asadar, pot fi numarati cu ajutorul unei operatii nrdiv : NzNat -> Nat.

OBSERVATIE: Modulul predefinit NAT contine relatia “divide pe”, declarata astfel:
op _divides_ : NzNat Nat -> Bool
si definita prin: pentru orice natural nenul Z si orice natural N, Z divides N = true daca si numai daca Z | N (adica daca si numai daca Z divide pe N).

EXEMPLE:
nrdivcresc(1 3 3 5 7 3 10 12 5 120 1200 24000) = 1 3 3 5 7 3 10 12;
nrdivcresc(1 3 10 0 0 0 100 1000) = 1 3 10 0 0 0;
nrdivdescresc(0 0 10 5 1 3 7) = 0 0 10 5 1;
nrdivdescresc(nil) = nil, unde nil este lista vida.

6) Sa se scrie un modul CafeObj pentru liste de numere naturale, care sa includa modulul NAT predefinit si sa contina doua operatii x2 si nrapmin, definite ca mai jos, alaturi de operatiile ajutatoare necesare pentru crearea sortului lista de numere naturale (fie acesta denumit Lista) si operatiile ajutatoare necesare pentru implementarea operatiilor x2 si nrapmin.
Operatiile x2 si nrapmin vor fi declarate astfel:
op x2 : Lista -> Lista
op nrapmin : Lista -> Nat

Pentru orice lista finita L de numere naturale:

x2(L) = sublista C a lui L obtinuta astfel: primul element al lui C, fie acesta notat E1, coincide cu primul element al lui L; urmatorul element al lui C, fie acesta notat E2, este primul element al lui L care satisface urmatoarele proprietati: ii urmeaza lui E1 ca pozitie in lista L si are valoarea egala cu dublul valorii lui E1; urmatorul element al lui C, fie acesta notat E3, este primul element al lui L care satisface urmatoarele proprietati: ii urmeaza lui E2 ca pozitie in lista L si are valoarea egala cu dublul valorii lui E2; s. a. m. d. pana la terminarea listei L sau a elementelor lui L care verifica proprietatea de mai sus;

nrapmin(L) = numarul de aparitii in lista L ale minimului din lista L.

EXEMPLE: Daca nil este lista vida:
x2(nil) = nil
x2(3 5 2 6 10 0 12 4 17 12 4 6) = 3 6 12
x2(0 3 2 5 7 0 1 9 2 0 2 5 0) = 0 0 0 0
nrapmin(nil) = 0
nrapmin(2 3 1 7 4 3 2) = 1
nrapmin(5 2 6 10 2 5 2 3 7) = 3

7) Sa se scrie un modul CafeObj pentru liste de numere naturale, care sa includa modulul NAT predefinit si sa contina doua operatii nrap si nrapcresc, definite ca mai jos, alaturi de operatiile ajutatoare necesare pentru crearea sortului lista de numere naturale (fie acesta denumit Lista) si operatiile ajutatoare necesare pentru implementarea operatiilor nrap si nrapcresc.
Operatiile nrap si nrapcresc vor fi declarate astfel:
ops nrap nrapcresc : Lista -> Lista

Pentru orice lista finita L de numere naturale:

nrap(L) = lista obtinuta din lista L prin inlocuirea fiecarui element al lui L cu numarul de aparitii ale valorii acelui element in L;

nrapcresc(L) = sublista C a lui L obtinuta astfel: primul element al lui C, fie acesta notat E1, coincide cu primul element al lui L; urmatorul element al lui C, fie acesta notat E2, este primul element al lui L care satisface urmatoarele proprietati: ii urmeaza lui E1 ca pozitie in lista L si are numarul de aparitii in lista L strict mai mare decat numarul de aparitii ale lui E1 in lista L; urmatorul element al lui C, fie acesta notat E3, este primul element al lui L care satisface urmatoarele proprietati: ii urmeaza lui E2 ca pozitie in lista L si are numarul de aparitii in lista L strict mai mare decat numarul de aparitii ale lui E2 in lista L; s. a. m. d. pana la terminarea listei L sau a elementelor lui L care verifica proprietatea de mai sus.

EXEMPLE: Daca nil este lista vida:
nrap(nil) = nil
nrap(0 3 2 3 0 3 1 5 1 3 0) = 3 4 1 4 3 4 2 1 2 4 3
nrapcresc(nil) = nil
nrapcresc(5 1 4 4 2 1 6 12 2 1 7 2 7 10 2) = 5 1 2

8) Sa se scrie un modul CafeObj pentru liste de numere naturale, care sa includa modulul NAT predefinit si sa contina doua operatii valenrap si esumanter, definite ca mai jos, alaturi de operatiile ajutatoare necesare pentru crearea sortului lista de numere naturale (fie acesta denumit Lista) si operatiile ajutatoare necesare pentru implementarea operatiilor valenrap si esumanter.

Operatiile valenrap si esumanter vor fi declarate astfel:
ops valenrap esumanter : Lista -> Lista

Pentru orice lista finita L de numere naturale:

valenrap(L) = sublista lui L formata din elementele lui L care sunt egale cu numarul aparitiilor lor in lista L;

esumanter(L) = sublista lui L formata din elementele lui L care sunt egale cu suma elementelor care le preceda in lista L.

EXEMPLE: Daca nil este lista vida, atunci:
valenrap(nil) = nil
valenrap(5 2 5 1 3 5 7 4 5 0 8 5 3 3) = 5 5 1 3 5 5 5 3 3
esumanter(nil) = nil
esumanter(0 2 5 7 1 7 0 22 3 47 7) = 0 7 22 47
esumanter(0 0 0 3 3 4 7 0 17 1 35) = 0 0 0 3 17 35
esumanter(1 1 1 2 5 3 0) = 1 5
esumanter(1 3 6 10 4) = 10
esumanter(2 0 7 14 5 0 1 2 100) = nil

9) Sa se scrie un modul CafeObj pentru liste de numere naturale, care sa includa modulul NAT predefinit si sa contina doua operatii inloc si elimdupapoz, definite ca mai jos, alaturi de operatiile ajutatoare necesare pentru crearea sortului lista de numere naturale (fie acesta denumit Lista) si operatiile ajutatoare necesare pentru implementarea operatiilor inloc si elimdupapoz.

Operatiile inloc si elimdupapoz vor fi declarate astfel:
ops inloc elimdupapoz : Lista -> Lista

Pentru orice lista finita L de numere naturale:

inloc(L) = lista obtinuta din L prin inlocuirea fiecarui element par P al lui L cu P quo 2 (= catul impartirii lui P la 2, predefinit in modulul NAT) si a fiecarui element impar I al lui L cu (3 * I) + 1;

elimdupapoz(L) = sublista lui L formata prin eliminarea din L a fiecarui element a carui valoare este strict mai mica decat numarul care indica pozitia lui in lista initiala L, unde pozitiile in L ale elementelor lui L sunt numarate de la 1 la lungimea lui L.

Amintesc ca in modulul NAT predefinit exista relatia “divide pe”, definita sub forma:
op _divides_ : NzNat Nat -> Bool

# ENUNTURI 2

Cerintele din exercitiile de mai jos au fost date la lucrari de control in anii anteriori. Unele dintre ele au fost rezolvate in modlectia9.maude si modlectia10.maude.

In a doua lucrare de control pe care o veti da, veti avea un exercitiu cu arbori binari si unul cu arbori oarecare (desigur, cele doua exercitii vor avea mai putine cerinte decat exercitiile (I) si (II) de mai jos).

In exercitiile de mai jos, sorturile pentru arbori binari, arbori oarecare si liste de arbori oarecare, precum si operatiile care construiesc aceste sorturi, vor fi denumite si declarate la fel ca in modlectia9.maude.

(I) Sa se scrie in Maude un modul pentru arbori binari cu numere intregi ca informatii in noduri, care sa includa modulul INT predefinit si sa contina urmatoarele operatii, definite ca mai jos.

```
  op maxdif : Arbbin -> Nat .
  op ecomplet : Arbbin -> Bool .
  op invers : Arbbin -> Arbbin .
```

Pentru orice arbore binar A:

maxdif(A) = maximul modulului diferentei dintre numarul de frunze al subarborelui stang si numarul de frunze al subarborelui drept ale unui nod din arborele binar A;
ecomplet(A) = true daca A este arbore binar complet, si false in caz contrar;
invers(A) = arborele binar care se obtine din A prin inversarea subarborelui stang cu subarborele drept al fiecarui nod al lui A.

(II) Sa se scrie in Maude un modul pentru arbori oarecare cu numere naturale ca informatii in noduri, care sa includa modulul NAT predefinit si sa contina urmatoarele operatii, definite ca mai jos.

```
  ops nrmaxfii nrminfiinenul nrnodint nrfrunze nrvalpare nrcunrimparfii : Arbore -> Nat .
  ops estebin descrescniv : Arbore -> Bool .
  ops addist radpara infoh : Arbore -> Arbore .
```

Pentru orice arbore A:

nrmaxfii(A) = numarul maxim de fii ai unui nod din arborele A;
nrminfiinenul(A) = numarul minim de fii ai unui nod intern (adica nod care nu e frunza) din arborele A, cu conventia: nrminfiinenul(null) = 0 si nrminfiinenul(F) = 0 daca F este o frunza;
nrnodint(A) = numarul de noduri interne (adica noduri care nu sunt frunze) ale arborelui A;
nrfrunze(A) = numarul de frunze ale arborelui A;
nrvalpare(A) = numarul de noduri din arborele A care au informatia para;
nrcunrimparfii(A) = numarul de noduri din arborele A care au un numar impar de fii;
estebin(A) = true daca A este arbore binar (cu aceeasi radacina ca aceea din reprezentarea sa ca arbore oarecare), si false in caz contrar;
descrescniv(A) = true daca informatia din fiecare nod intern (adica nod care nu e frunza) al lui A este mai mare sau egala cu informatia din fiecare fiu al sau, si false in caz contrar;
addist(A) = arborele obtinut din A prin inlocuirea fiecarei informatii I din fiecare nod al sau cu I + D, unde D este distanta de la radacina arborelui A la acel nod, adica numarul de niveluri care despart radacina lui A de acel nod (D = 0 in cazul radacinii, D = 1 pentru fiecare fiu al radacinii etc.);
radpara(A) = arborele obtinut din A prin eliminarea tuturor subarborilor sai cu informatia din radacina data de un numar natural impar;
infoh(A) = arborele obtinut din A prin inlocuirea informatiei din fiecare nod cu inaltimea subarborelui lui A de radacina acel nod.

# REZOLVARI 1

OBSERVATIE: Modulele de mai jos sunt scrise in CafeObj. Asemanarea de sintaxa cu Maude-ul este evidenta. Poate ar fi bine ca fiecare dintre voi sa transpuna aceste module in Maude, ca sa va obisnuiti cu sintaxa Maude pana la lucrarea de control.

```
mod! REZOLVARE1{

extending (NAT)

[Nat < Lista]

op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

ops e2 nfact : Lista -> Lista
op e2pt : NzNat -> Nat
op isnfact : Nat -> Bool
op estenfact : NzNat NzNat -> Bool

var N : Nat
vars X Z : NzNat
var L : Lista

ceq e2pt(Z) = 0 if not(2 divides Z) .
ceq e2pt(Z) = s(e2pt(Z quo 2)) if 2 divides Z .

eq e2(nil) = nil .
eq e2(0 L) = e2(L) .
eq e2(Z L) = e2pt(Z) e2(L) .

eq isnfact(0) = false .
eq isnfact(Z) = estenfact(Z,1) .

eq estenfact(Z,Z) = true .
ceq estenfact(X,Z) = false if (Z =/= X) and not (Z divides X) .
ceq estenfact(X,Z) = estenfact(X quo Z,s(Z)) if (Z =/= X) and (Z divides X) .

eq nfact(nil) = nil .
ceq nfact(N L) = N nfact(L) if isnfact(N) .
ceq nfact(N L) = nfact(L) if not isnfact(N) .
}


mod! REZOLVARE2{
```

```
protecting (NAT)

[Nat < Lista]

op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

ops nsn cresc : Lista -> Lista
op ensn : Nat -> Bool
op ensncun : Nat Nat -> Bool
op crescdela : Nat Lista -> Lista

vars N M : Nat
var L : Lista

eq ensn(N) = ensncun(N,0) .

ceq ensncun(N,M) = true if N == M * s(M) .
ceq ensncun(N,M) = false if N < M * s(M) .
ceq ensncun(N,M) = ensncun(N,s(M)) if N > M * s(M) .

eq nsn(nil) = nil .
ceq nsn(N L) = N nsn(L) if ensn(N) .
ceq nsn(N L) = nsn(L) if not ensn(N) .

eq crescdela(N,nil) = nil .
ceq crescdela(N,M L) = M crescdela(M,L) if N <= M .
ceq crescdela(N,M L) = crescdela(N,L) if N > M .

eq cresc(L) = crescdela(0,L) .
}
```

** Solutie data de un student: implementarea lui cresc fara operatii auxiliare:

```
mod! REZOLVARE2STUD{

protecting (NAT)

[Nat < Lista]

op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

op cresc : Lista -> Lista
```

```
vars N M : Nat
var L : Lista

eq cresc(nil) = nil .
eq cresc(N) = N .
ceq cresc(N M L) = N cresc(M L) if M >= N .
ceq cresc(N M L) = cresc(N L) if M < N .
}


mod! REZOLVARE3{

protecting (NAT)

[Nat < Lista]

op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

ops inccresc cp : Lista -> Lista
op ecp : Nat -> Bool
op ecpcun : Nat Nat -> Bool

vars N M : Nat
var L : Lista

eq inccresc(nil) = nil .
eq inccresc(N) = N .
ceq inccresc(N M L) = N inccresc(M L) if N <= M .
ceq inccresc(N M L) = N if N > M .

eq ecp(N) = ecpcun(N,0) .

ceq ecpcun(N,M) = true if N == M * M * M .
ceq ecpcun(N,M) = ecpcun(N,s(M)) if N > M * M * M .
ceq ecpcun(N,M) = false if N < M * M * M .

eq cp(nil) = nil .
ceq cp(N L) = N cp(L) if ecp(N) .
ceq cp(N L) = cp(L) if not ecp(N) .
}


mod! REZOLVARE4{
```

```
protecting (NAT)

[Nat < Lista]

op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

ops lp3 cdivu : Lista -> Lista

op ep3 : Nat -> Bool
op ep3aux : Nat Nat -> Bool

op _divide-pe_ : Nat Nat -> Bool

vars N M : Nat
var Z : NzNat
var L : Lista

eq ep3(N) = ep3aux(N,1) .

ceq ep3aux(N,M) = ep3aux(N,3 * M) if N > M .
eq ep3aux(N,N) = true .
ceq ep3aux(N,M) = false if N < M .

eq lp3(nil) = nil .
ceq lp3(N L) = N lp3(L) if ep3(N) .
ceq lp3(N L) = lp3(L) if not ep3(N) .

eq Z divide-pe N = Z divides N .
eq 0 divide-pe 0 = true .
eq 0 divide-pe Z = false .

eq cdivu(nil) = nil .
eq cdivu(N) = N .
ceq cdivu(N M L) = N cdivu(M L) if N divide-pe M .
ceq cdivu(N M L) = N if not (N divide-pe M) .
}


mod! REZOLVARE5{

extending (NAT)

[Nat < Lista]
```

```
op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

ops nrdivcresc nrdivdescresc : Lista -> Lista

op nrdiv : NzNat -> Nat
op nrdivdela : NzNat NzNat -> Nat
ops nrdivmmic nrdivmmare : Nat Nat -> Bool

vars N M : Nat
vars Z Y : NzNat
var L : Lista

eq nrdiv(Z) = nrdivdela(Z,1) .

ceq nrdivdela(Z,Y) = s(nrdivdela(Z,s(Y))) if Y <= Z and Y divides Z .
ceq nrdivdela(Z,Y) = nrdivdela(Z,s(Y)) if Y <= Z and not (Y divides Z) .
ceq nrdivdela(Z,Y) = 0 if Y > Z .

eq nrdivmmic(N,0) = true .
eq nrdivmmic(0,Z) = false .
eq nrdivmmic(Z,Y) = nrdiv(Z) <= nrdiv(Y) .

eq nrdivmmare(M,N) = nrdiv(M) == nrdiv(N) or not nrdivmmic(M,N) .

eq nrdivcresc(nil) = nil .
eq nrdivcresc(N) = N .
ceq nrdivcresc(M N L) = M if not nrdivmmic(M,N) .
ceq nrdivcresc(M N L) = M nrdivcresc(N L) if nrdivmmic(M,N) .

eq nrdivdescresc(nil) = nil .
eq nrdivdescresc(N) = N .
ceq nrdivdescresc(M N L) = M if not nrdivmmare(M,N) .
ceq nrdivdescresc(M N L) = M nrdivdescresc(N L) if nrdivmmare(M,N) .
}


mod! REZOLVARE6{

extending (NAT)

[Nat < Lista , PerNat]

op nil : -> Lista
```

```
op __ : Lista Lista -> Lista {assoc id: nil}

op [_;_] : Nat Nat -> PerNat
op minsinrapmin : Lista -> PerNat
op aldoilea : PerNat -> Nat
op minper : PerNat PerNat -> PerNat

op x2 : Lista -> Lista
op nrapmin : Lista -> Nat

vars N M X Y : Nat
var L : Lista

eq x2(nil) = nil .
eq x2(N) = N .
ceq x2(N M L) = N x2(M L) if M == N + N .
ceq x2(N M L) = x2(N L) if M =/= N + N .

eq nrapmin(nil) = 0 .
eq nrapmin(N L) = aldoilea(minsinrapmin(N L)) .

eq aldoilea([N ; M]) = M .

eq minsinrapmin(N) = [N ; 1] .
eq minsinrapmin(N M L) = minper([N ; 1],minsinrapmin(M L)) .

ceq minper([N ; M],[X ; Y]) = [N ; M] if N < X .
eq minper([N ; M],[N ; Y]) = [N ; M + Y] .
ceq minper([N ; M],[X ; Y]) = [X ; Y] if N > X .
}


mod! REZOLVARE6VER2{

extending (NAT)

[Nat < Lista]

op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

op min : Nat Nat -> Nat
op minlist : Lista -> Nat
op nrap : Nat Lista -> Nat
```

```
op x2 : Lista -> Lista
op nrapmin : Lista -> Nat

vars N M : Nat
var L : Lista

eq x2(nil) = nil .
eq x2(N) = N .
ceq x2(N M L) = N x2(M L) if M == N + N .
ceq x2(N M L) = x2(N L) if M =/= N + N .

eq nrapmin(nil) = 0 .
eq nrapmin(N L) = nrap(minlist(N L),N L) .

ceq min(N,M) = N if N <= M .
ceq min(N,M) = M if N > M .

eq minlist(N) = N .
eq minlist(N M L) = min(N,minlist(M L)) .

eq nrap(N,nil) = 0 .
eq nrap(N,N L) = s(nrap(N,L)) .
ceq nrap(N,M L) = nrap(N,L) if N =/= M .
}


mod! REZOLVARE7{

extending (NAT)

[Nat < Lista]

op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

ops nrap nrapcresc : Lista -> Lista

op nrapar : Nat Lista -> Nat
ops nrapculinit nrapcrescculinit : Lista Lista -> Lista

vars N M : Nat
vars L Linit : Lista

eq nrapar(N,nil) = 0 .
eq nrapar(N,N L) = s(nrapar(N,L)) .
```

```
ceq nrapar(N,M L) = nrapar(N,L) if N =/= M .

eq nrapculinit(nil,Linit) = nil .
eq nrapculinit(N L,Linit) = nrapar(N,Linit) nrapculinit(L,Linit) .

eq nrap(L) = nrapculinit(L,L) .

eq nrapcrescculinit(nil,Linit) = nil .
eq nrapcrescculinit(N,Linit) = N .
ceq nrapcrescculinit(N M L,Linit) = nrapcrescculinit(N L,Linit) if nrapar(N,Linit) >= nrapar(M,Linit) .
ceq nrapcrescculinit(N M L,Linit) = N nrapcrescculinit(M L,Linit) if nrapar(N,Linit) < nrapar(M,Linit) .

eq nrapcresc(L) = nrapcrescculinit(L,L) .
}


mod! REZOLVARE7VER2{

extending (NAT)

[Nat < Lista]

op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

ops nrap nrapcresc : Lista -> Lista

op nrapar : Nat Lista -> Nat
ops nrapculinit nrapcresccul : Lista Lista -> Lista

vars N M X Y : Nat
vars L L1 : Lista

eq nrapar(N,nil) = 0 .
eq nrapar(N,N L) = s(nrapar(N,L)) .
ceq nrapar(N,M L) = nrapar(N,L) if N =/= M .

eq nrapculinit(nil,L1) = nil .
eq nrapculinit(N L,L1) = nrapar(N,L1) nrapculinit(L,L1) .

eq nrap(L) = nrapculinit(L,L) .

eq nrapcresccul(nil,nil) = nil .
eq nrapcresccul(N,X) = N .
ceq nrapcresccul(N M L,X Y L1) = nrapcresccul(N L,X L1) if X >= Y .
```

```
ceq nrapcresccul(N M L,X Y L1) = N nrapcresccul(M L,Y L1) if X < Y .

eq nrapcresc(L) = nrapcresccul(L,nrap(L)) .
}


mod! REZOLVARE8{

extending (NAT)

[Nat < Lista]

op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

ops valenrap esumanter : Lista -> Lista

op nrap : Nat Lista -> Nat
op valenrapculinit : Lista Lista -> Lista
op esum : Lista Nat -> Lista

vars N M : Nat
vars L L1 : Lista

eq nrap(N,nil) = 0 .
eq nrap(N,N L) = s(nrap(N,L)) .
ceq nrap(N,M L) = nrap(N,L) if N =/= M .

eq valenrap(L) = valenrapculinit(L,L) .

eq valenrapculinit(nil,L1) = nil .
ceq valenrapculinit(N L,L1) = N valenrapculinit(L,L1) if N == nrap(N,L1) .
ceq valenrapculinit(N L,L1) = valenrapculinit(L,L1) if N =/= nrap(N,L1) .

eq esumanter(L) = esum(L,0) .

eq esum(nil,N) = nil .
eq esum(N L,N) = N esum(L,N + N) .
ceq esum(N L,M) = esum(L,N + M) if N =/= M .
}


mod! REZOLVARE9{

protecting (NAT)
```

```
[Nat < Lista]

op nil : -> Lista
op __ : Lista Lista -> Lista {assoc id: nil}

ops inloc elimdupapoz : Lista -> Lista
op elimcupozinit : Lista Nat -> Lista

vars N M : Nat
var L : Lista

eq inloc(nil) = nil .
ceq inloc(N L) = (N quo 2) inloc(L) if 2 divides N .
ceq inloc(N L) = s(N + N + N) inloc(L) if not (2 divides N) .

eq elimdupapoz(L) = elimcupozinit(L,1) .

eq elimcupozinit(nil,M) = nil .
ceq elimcupozinit(N L,M) = N elimcupozinit(L,s(M)) if N >= M .
ceq elimcupozinit(N L,M) = elimcupozinit(L,s(M)) if N < M .
}
```

# REZOLVARI 2

```
fmod EXERCITIUL-I is

  protecting INT .

  sorts Lista Arbbin .
  subsort Int < Lista .
***> Lista e sortul pentru liste de numere intregi
***> Arbbin e sortul pentru arbori binari

  op nil : -> Lista . ***> lista de numere intregi vida
  op __ : Lista Lista -> Lista [assoc id: nil] .

  op null : -> Arbbin . ***> arborele binar vid
  op _{_,_} : Int Arbbin Arbbin -> Arbbin .
***> operatiile care construiesc sortul Arbbin

  ops maxdif nrfrunze absdif h : Arbbin -> Nat .
  ops ecomplet : Arbbin -> Bool .
  op invers : Arbbin -> Arbbin .
```

```
  op plinlaniv : Arbbin Nat Nat -> Bool .

  ops ex1 ex2 ex3 ex4 ex5 ex6 ex7 ex8 ex9 ex10 : -> Arbbin .

  vars H L M N : Nat .
  vars I J K : Int .
  vars A B C D : Arbbin .

  eq nrfrunze(null) = 0 .
  eq nrfrunze(I{null,null}) = 1 .
  eq nrfrunze(I{null,J{A,B}}) = nrfrunze(J{A,B}) .
  eq nrfrunze(I{J{A,B},null}) = nrfrunze(J{A,B}) .
  eq nrfrunze(I{J{A,B},K{C,D}}) = nrfrunze(J{A,B}) + nrfrunze(K{C,D}) .

  eq absdif(null) = 0 .
  eq absdif(I{A,B}) = abs(nrfrunze(B) - nrfrunze(A)) .

  eq maxdif(null) = 0 .
  eq maxdif(I{A,B}) = max(absdif(I{A,B}),max(absdif(A),absdif(B))) .

***> max si abs sunt predefinite in modulul INT.

  eq invers(null) = null .
  eq invers(I{A,B}) = I{invers(B),invers(A)} .

  eq h(null) = 0 .
  eq h(I{A,B}) = s max(h(A),h(B)) .

  eq ecomplet(A) = plinlaniv(A,h(A),1) .

  ceq plinlaniv(I{A,null},H,L) = false if L < H .
  ceq plinlaniv(I{null,A},H,L) = false if L < H .
  ceq plinlaniv(I{J{A,B},K{C,D}},H,L) = plinlaniv(J{A,B},H,s L) and plinlaniv(K{C,D},H,s L) if L < H .
  ceq plinlaniv(I{A,B},H,L) = true if L >= H .
  eq plinlaniv(null,H,L) = true .

  eq ex1 = 1{2{null,null},-3{null,4{null,null}}} .
  eq ex2 = 1{2{null,null},3{-6{null,null},4{7{null,null},5{null,null}}}} .
  eq ex3 = 1{null,3{6{null,null},4{7{null,null},5{null,null}}}} .
  eq ex4 = 10{7{null,-8{null,null}},15{-12{null,null},-20{18{null,null},null}}} .
  eq ex5 = 10{null,25{null,25{null,null}}} .
  eq ex6 = 10{10{null,null},25{null,25{null,null}}} .
  eq ex7 = 10{7{null,30{null,null}},15{12{null,null},20{18{null,null},null}}} .
  eq ex8 = 1{2{null,null},3{null,4{null,null}}} .
  eq ex9 = 1{-2{null,null},-3{6{null,null},-4{7{null,null},5{null,null}}}} .
```

```
  eq ex10 = 1{2{4{null,null},5{null,null}},3{6{null,null},7{null,null}}} .

endfm


fmod EXERCITIUL-II is

  protecting NAT .

  sorts Lista Arbore ListArb .
  subsort Nat < Lista .
  subsort Arbore < ListArb .
***> Lista e sortul pentru liste de numere naturale
***> Arbore e sortul pentru arbori oarecare
***> ListArb e sortul pentru liste de arbori oarecare

  op nil : -> Lista . ***> lista de numere naturale vida
  op __ : Lista Lista -> Lista [assoc id: nil] .

  op null : -> Arbore . ***> arborele vid
  op _{_} : Nat ListArb -> Arbore [prec 20] .
***> operatiile care construiesc sortul Arbore

  op frunza : -> ListArb . ***> lista de arbori vida
  op _;_ : ListArb ListArb -> ListArb [assoc id: frunza prec 30] .
***> concatenarea de liste de arbori

  ops nrmaxfii nrminfiinenul nrnodint nrfrunze nrvalpare nrcunrimparfii : Arbore -> Nat .
  ops estebin descrescniv : Arbore -> Bool .
  ops addist radpara infoh : Arbore -> Arbore .

  op lungime : ListArb -> Nat .
  ops maxlist minlist : Lista -> Nat .
  op listfara0 : Lista -> Lista .
  op nrfii : Arbore -> Nat .
  op nrfiilist : ListArb -> Lista .
  ops nrnodintlist nrfrunzelist nrvalparelist nrcunrimparfiilist : ListArb -> Nat .
  op descrescnivcuval : Arbore Nat -> Bool .
  op descrescnivcuvallist : ListArb Nat -> Bool .
  op addistcuniv : Arbore Nat -> Arbore .
  op addistcunivlist : ListArb Nat -> ListArb .
  ops listaradpara listainfoh : ListArb -> ListArb .
  op h : Arbore -> Nat .
  op listah : ListArb -> Lista .
  ops ex1 ex2 ex3 ex4 ex5 ex6 ex7 ex8 ex9 : -> Arbore .
```

```
vars M N P : Nat .
var L : Lista .
vars A B C : Arbore .
var LA LA1 : ListArb .

eq maxlist(nil) = 0 .
eq maxlist(N L) = max(N,maxlist(L)) .

eq minlist(N) = N .
eq minlist(N M L) = min(N,minlist(M L)) .

eq lungime(frunza) = 0 .
eq lungime(A ; LA) = s lungime(LA) .

eq listfara0(nil) = nil .
eq listfara0(0 L) = listfara0(L) .
ceq listfara0(N L) = N listfara0(L) if N =/= 0 .

eq nrfii(null) = 0 .
eq nrfii(N{LA}) = lungime(LA) .

eq nrfiilist(frunza) = nil .
eq nrfiilist(A ; LA) = nrfii(A) nrfiilist(LA) .

eq nrmaxfii(null) = 0 .
eq nrmaxfii(N{LA}) = max(nrfii(N{LA}),maxlist(nrfiilist(LA))) .

eq nrminfiinenul(null) = 0 .
eq nrminfiinenul(N{frunza}) = 0 .
eq nrminfiinenul(N{A ; LA}) = minlist(nrfii(N{A ; LA}) listfara0(nrfiilist(A ; LA))) .

eq nrnodint(null) = 0 .
eq nrnodint(N{frunza}) = 0 .
eq nrnodint(N{A ; LA}) = s(nrnodintlist(A ; LA)) .

eq nrnodintlist(frunza) = 0 .
eq nrnodintlist(A ; LA) = nrnodint(A) + nrnodintlist(LA) .

eq nrfrunze(null) = 0 .
eq nrfrunze(N{frunza}) = 1 .
eq nrfrunze(N{A ; LA}) = nrfrunzelist(A ; LA) .

eq nrfrunzelist(frunza) = 0 .
eq nrfrunzelist(A ; LA) = nrfrunze(A) + nrfrunzelist(LA) .
```

```
  eq estebin(null) = true .
  eq estebin(N{frunza}) = true .
  eq estebin(N{A}) = estebin(A) .
  eq estebin(N{A ; B}) = estebin(A) and estebin(B) .
  eq estebin(N{A ; B ; C ; LA}) = false .

  eq nrvalpare(null) = 0 .
  ceq nrvalpare(N{LA}) = nrvalparelist(LA) if not (2 divides N) .
  ceq nrvalpare(N{LA}) = s nrvalparelist(LA) if 2 divides N .

  eq nrvalparelist(frunza) = 0 .
  eq nrvalparelist(A ; LA) = nrvalpare(A) + nrvalparelist(LA) .

  eq nrcunrimparfii(null) = 0 .
  ceq nrcunrimparfii(N{LA}) = nrcunrimparfiilist(LA) if 2 divides lungime(LA) .
  ceq nrcunrimparfii(N{LA}) = s nrcunrimparfiilist(LA) if not (2 divides lungime(LA)) .

  eq nrcunrimparfiilist(frunza) = 0 .
  eq nrcunrimparfiilist(A ; LA) = nrcunrimparfii(A) + nrcunrimparfiilist(LA) .

  eq descrescniv(null) = true .
  eq descrescniv(N{frunza}) = true .
  eq descrescniv(N{A ; LA}) = descrescnivcuval(N{A ; LA},N) .

  eq descrescnivcuval(null,N) = true .
  eq descrescnivcuval(M{frunza},N) = N >= M .
  eq descrescnivcuval(M{P{LA1} ; LA},N) = N >= M and descrescnivcuval(P{LA1},M) and
descrescnivcuvallist(LA,M) .

  eq descrescnivcuvallist(frunza,N) = true .
  eq descrescnivcuvallist((A ; LA),N) = descrescnivcuval(A,N) and descrescnivcuvallist(LA,N) .

  eq addist(A) = addistcuniv(A,0) .

  eq addistcuniv(null,N) = null .
  eq addistcuniv(M{LA},N) = (M + N){addistcunivlist(LA,s N)} .

  eq addistcunivlist(frunza,N) = frunza .
  eq addistcunivlist((A ; LA),N) = addistcuniv(A,N) ; addistcunivlist(LA,N) .

  eq radpara(null) = null .
  ceq radpara(N{LA}) = null if not (2 divides N) .
  ceq radpara(N{LA}) = N{listaradpara(LA)} if 2 divides N .
```

```
  eq listaradpara(frunza) = frunza .
  ceq listaradpara(N{LA1} ; LA) = radpara(N{LA1}) ; listaradpara(LA) if 2 divides N .
  ceq listaradpara(N{LA1} ; LA) = listaradpara(LA) if not (2 divides N) .

  eq infoh(null) = null .
  eq infoh(N{LA}) = h(N{LA}){listainfoh(LA)} .

  eq listainfoh(frunza) = frunza .
  eq listainfoh(A ; LA) = infoh(A) ; listainfoh(LA) .

  eq h(null) = 0 .
  eq h(N{LA}) = s maxlist(listah(LA)) .

  eq listah(frunza) = nil .
  eq listah(A ; LA) = h(A) listah(LA) .

  eq ex1 = 1{2{3{frunza} ; 4{frunza}} ; 5{6{frunza}} ; 7{frunza}} .
  eq ex2 = 1{2{3{frunza} ; 4{frunza}} ; 5{6{frunza}}} .
  eq ex3 = 1{2{3{frunza} ; 4{frunza}} ; 5{6{frunza} ; 7{8{frunza} ; 9{frunza} ; 10{frunza}}}} .
  eq ex4 = 1{2{3{frunza} ; 4{frunza}} ; 5{6{frunza}} ; 7{frunza}} .
  eq ex5 = 10{2{1{frunza} ; 0{frunza}} ; 5{3{frunza}} ; 7{frunza}} .
  eq ex6 = 10{2{1{frunza} ; 2{frunza}} ; 5{3{frunza}} ; 7{frunza}} .
  eq ex7 = 3{4{frunza} ; 0{6{frunza} ; 8{2{frunza} ; 10{frunza} ; 20{frunza}}}} .
  eq ex8 = 12{5{frunza} ; 0{13{frunza} ; 8{2{frunza} ; 7{frunza} ; 20{frunza}}} ; 22{25{frunza}}} .
  eq ex9 = 12{5{frunza} ; 0{8{2{frunza} ; 7{frunza} ; 20{frunza}}} ; 22{25{frunza}}} .

endfm
```