Expresii conditionale

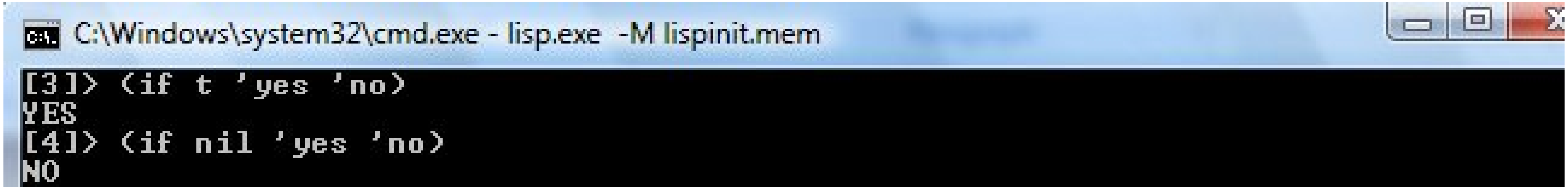
Ruxandra Stoean
http://inf.ucv.ro/~rstoean
ruxandra.stoean@inf.ucv.ro

Conditionalul

- Lisp are doua tipuri de expresii conditionale:
  - IF
  - COND

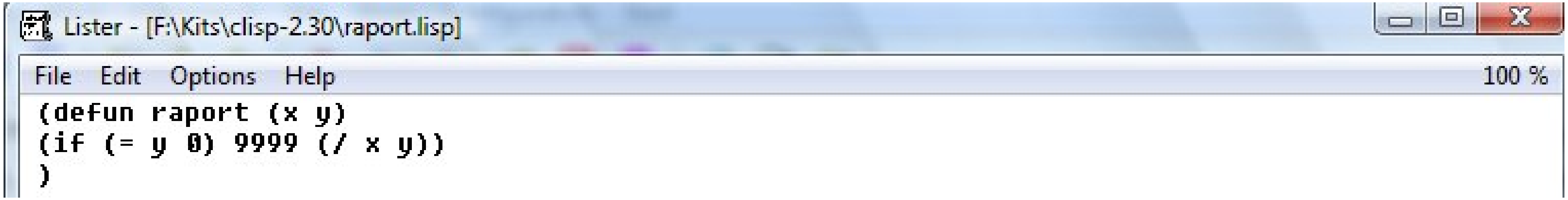
- Functia care exprima clasicul IF are formularea (if *test then else*):
  - if e cuvint cheie.
  - Daca *test* e adevarat, atunci se intoarce valoarea lui *then*; altfel, vom obtine valoarea lui *else*.

Exemple simple

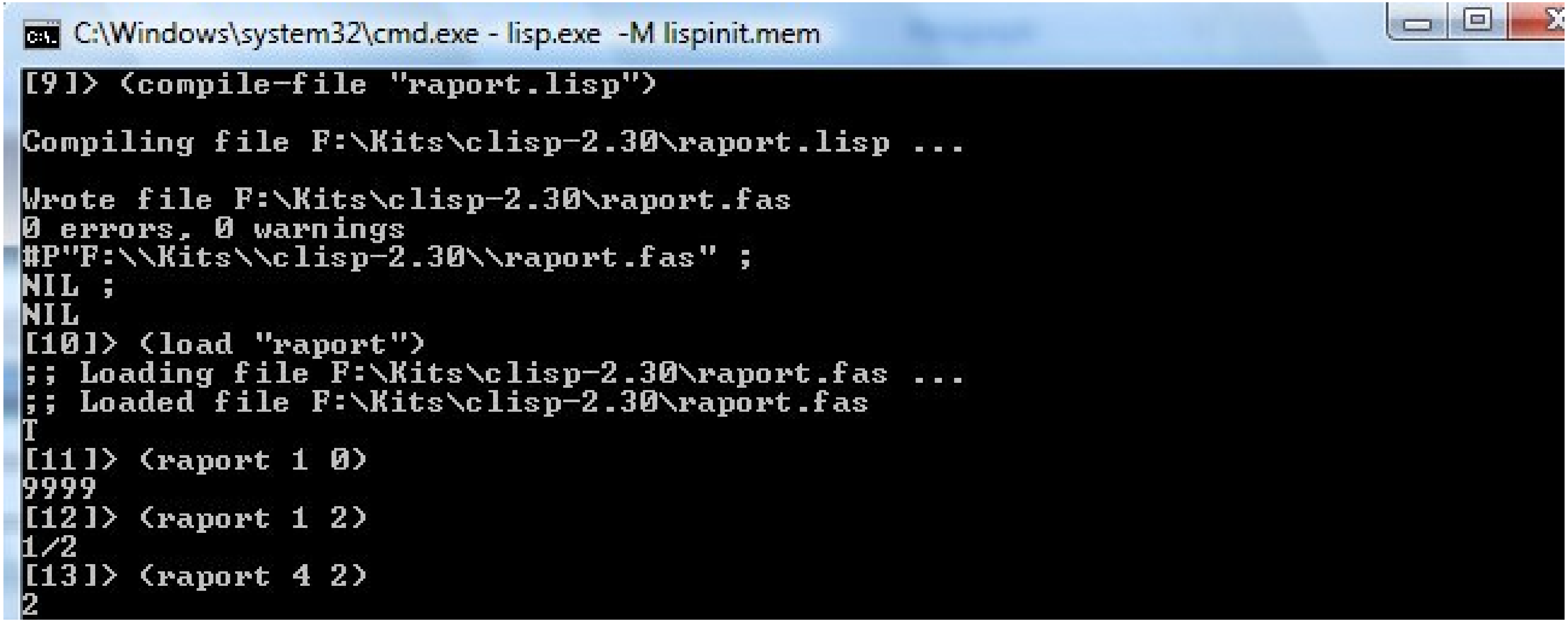


```
[3]> (if t 'yes 'no)
YES
[4]> (if nil 'yes 'no)
NO
```

Exemple – Testare raport

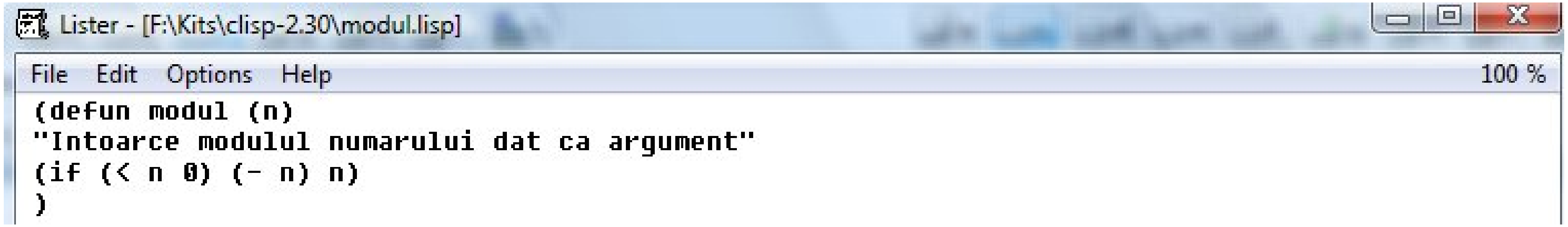


```
(defun raport (x y)
(if (= y 0) 9999 (/ x y))
)
```

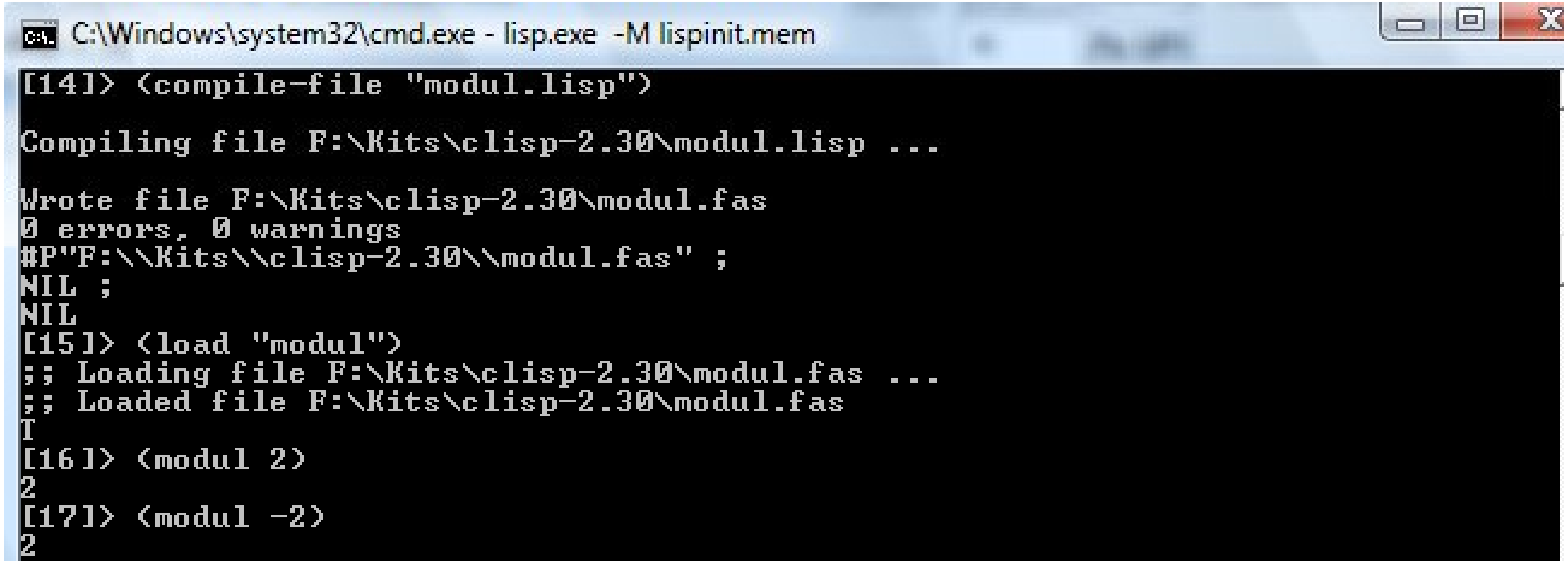


```
[9]> (compile-file "raport.lisp")

Compiling file F:\Kits\clisp-2.30\raport.lisp ...

Wrote file F:\Kits\clisp-2.30\raport.fas
0 errors, 0 warnings
#P"F:\\Kits\\clisp-2.30\\raport.fas" ;
NIL ;
NIL
[10]> (load "raport")
;; Loading file F:\Kits\clisp-2.30\raport.fas ...
;; Loaded file F:\Kits\clisp-2.30\raport.fas
T
[11]> (raport 1 0)
9999
[12]> (raport 1 2)
1/2
[13]> (raport 4 2)
2
```

Exemple – Modulul unui numar

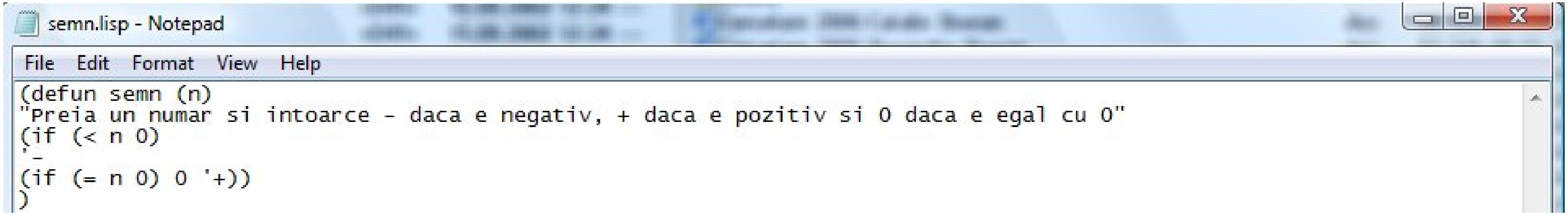


```
(defun modul (n)
"Intoarce modulul numarului dat ca argument"
(if (< n 0) (- n) n)
)
```

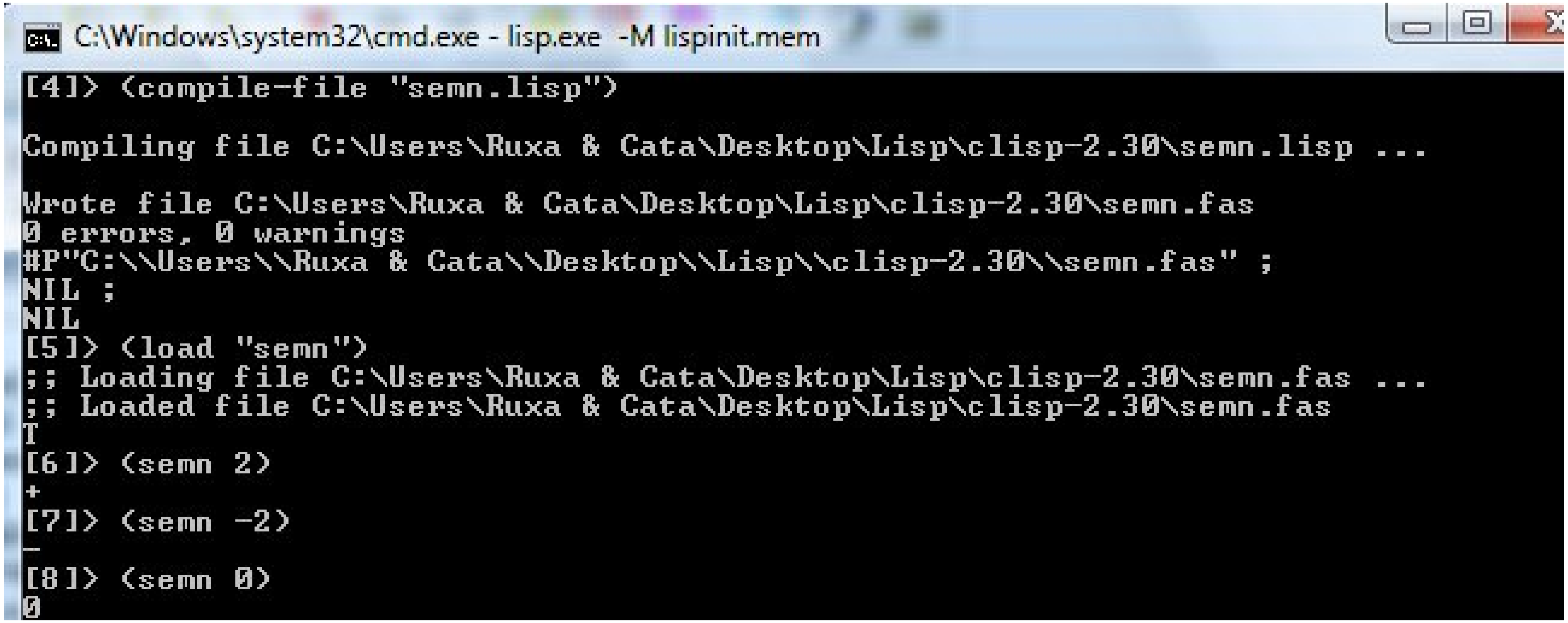


```
[14]> (compile-file "modul.lisp")

Compiling file F:\Kits\clisp-2.30\modul.lisp ...

Wrote file F:\Kits\clisp-2.30\modul.fas
0 errors, 0 warnings
#P"F:\\Kits\\clisp-2.30\\modul.fas" ;
NIL ;
NIL
[15]> (load "modul")
;; Loading file F:\Kits\clisp-2.30\modul.fas ...
;; Loaded file F:\Kits\clisp-2.30\modul.fas
T
[16]> (modul 2)
2
[17]> (modul -2)
2
```

Exemplu – Functia semn



```
(defun semn (n)
"Preia un numar si intoarce - daca e negativ, + daca e pozitiv si 0 daca e egal cu 0"
(if (< n 0)
'-
(if (= n 0) 0 '+))
)
```



```
[4]> (compile-file "semn.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\semn.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\semn.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\semn.fas" ;
NIL ;
NIL
[5]> (load "semn")
;; Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\semn.fas ...
;; Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\semn.fas
T
[6]> (semn 2)
+
[7]> (semn -2)
-
[8]> (semn 0)
0
```

Conditionalul COND

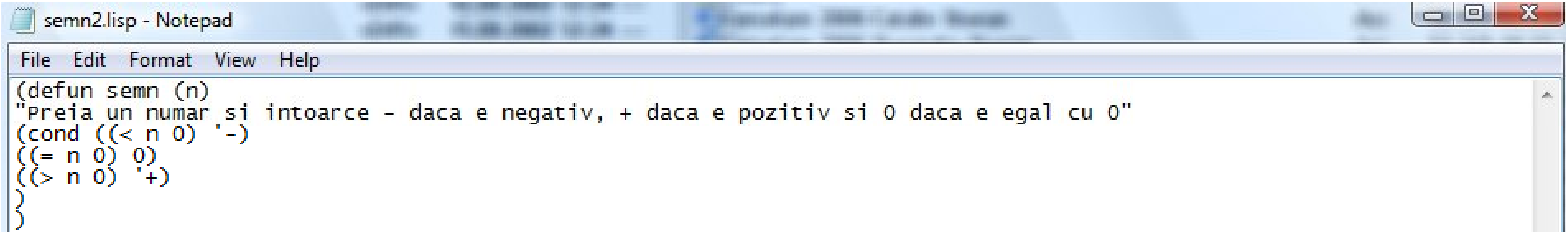
- Expresia IF este potrivita pentru a alege intre doua calcule pe baza unui singur test.
- Insa, in momentul in care avem de ales intre teste multiple, folosirea lui IF este greoaie si greselile pot aparea foarte usor.
- In aceste cazuri, vom utiliza alternativa lui IF si anume conditionalul COND.
- Evident, in cazul invers, cand avem un singur test, este mai eficient sa folosim IF.

Conditionalul COND

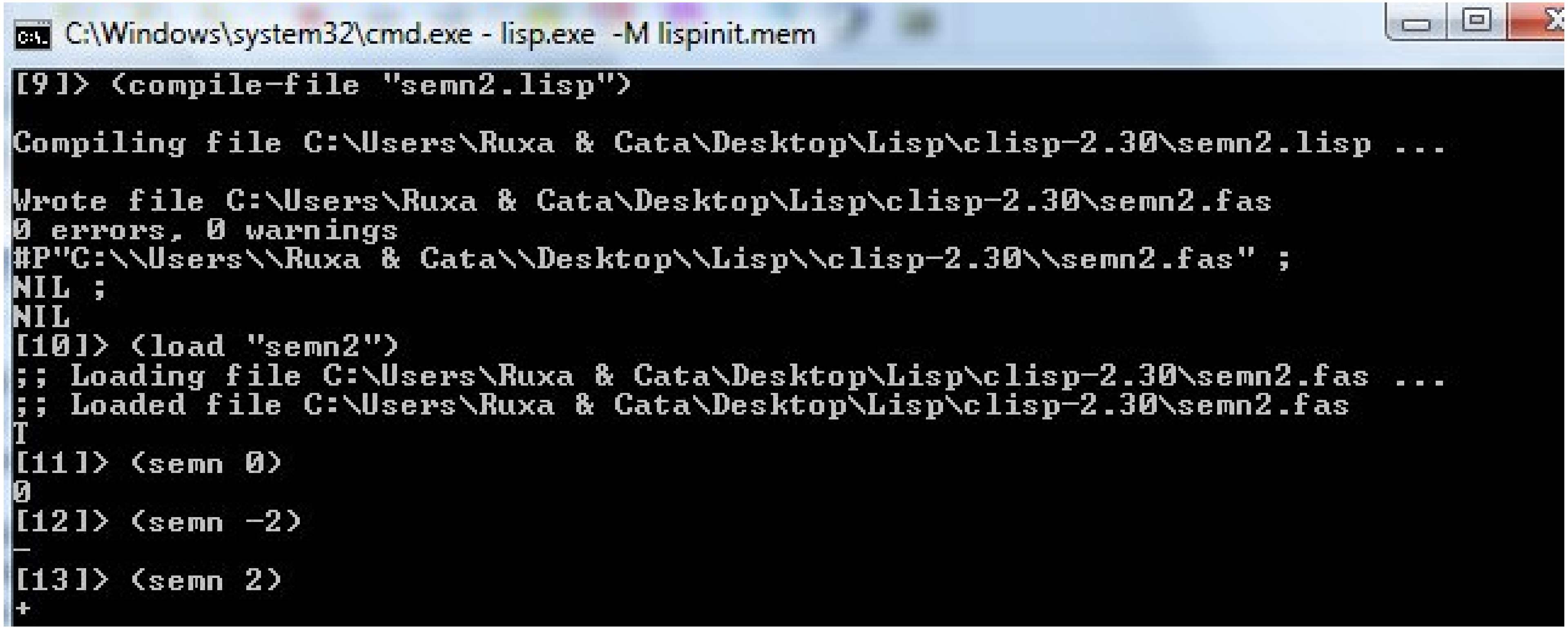
- Functia COND are sintaxa (cond $(p_1\ e_1)$ … $(p_n\ e_n)$):
  - Evalueaza p_i-urile in ordine pana cand unul dintre ele, p_j, este true.
  - Atunci intoarce e_j.
  - Daca niciun p_i nu este evaluat ca True, atunci intoarce False.

- Fiecare lista $(p_i\ e_i)$ poarta numele de pereche COND:
  - p_i este testul (conditia).
  - e_i este expresia.

Exemplu – Functia semn - Reluare



```
(defun semn (n)
"Preia un numar si intoarce - daca e negativ, + daca e pozitiv si 0 daca e egal cu 0"
(cond ((< n 0) '-)
((= n 0) 0)
((> n 0) '+)
)
)
```



```
[9]> (compile-file "semn2.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\semn2.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\semn2.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\semn2.fas" ;
NIL ;
NIL
[10]> (load "semn2")
;; Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\semn2.fas ...
;; Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\semn2.fas
T
[11]> (semn 0)
0
[12]> (semn -2)
-
[13]> (semn 2)
+
```

Asemanare cu IF-ul procedural

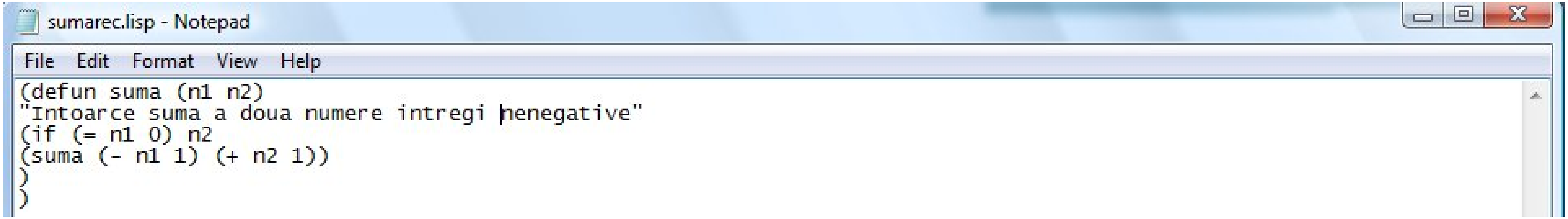
if p_1 then e_1
else if p_2 then e_2
else if p_3 then e_3
else e_4

(cond (p_1 e_1)
(p_2 e_2)
(p_3 e_3)
(t e_4))

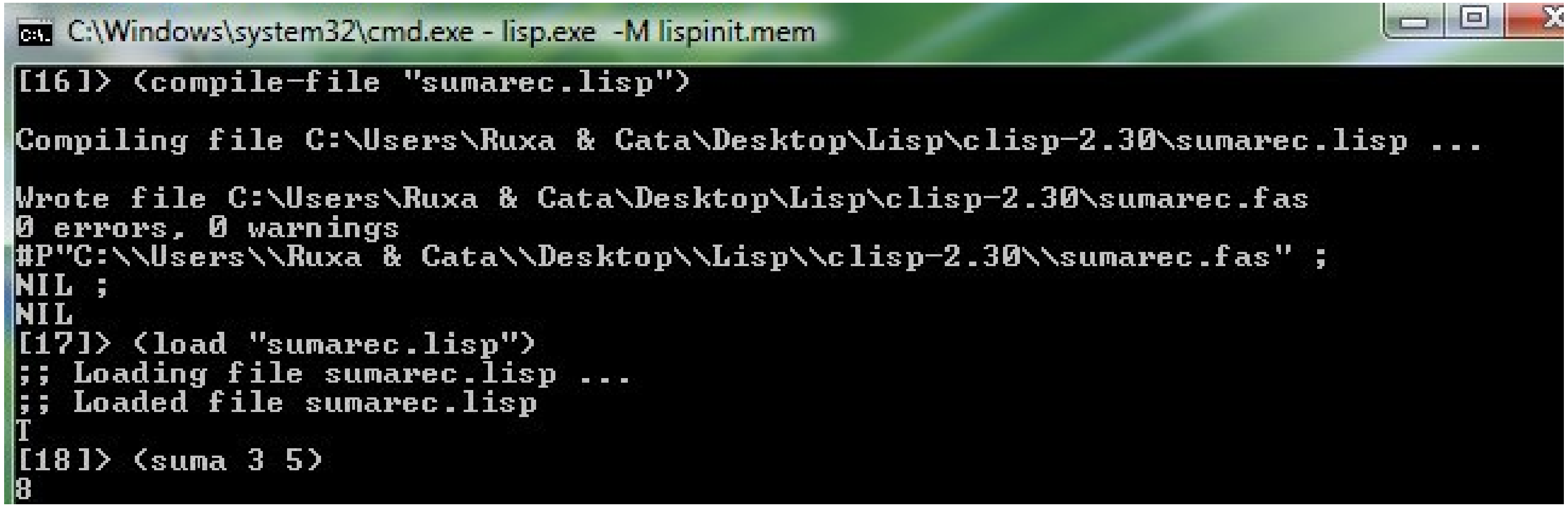
Recursivitate

Folosirea functiilor recursive

- Sa calculam recursiv suma a doua numere nenegative.

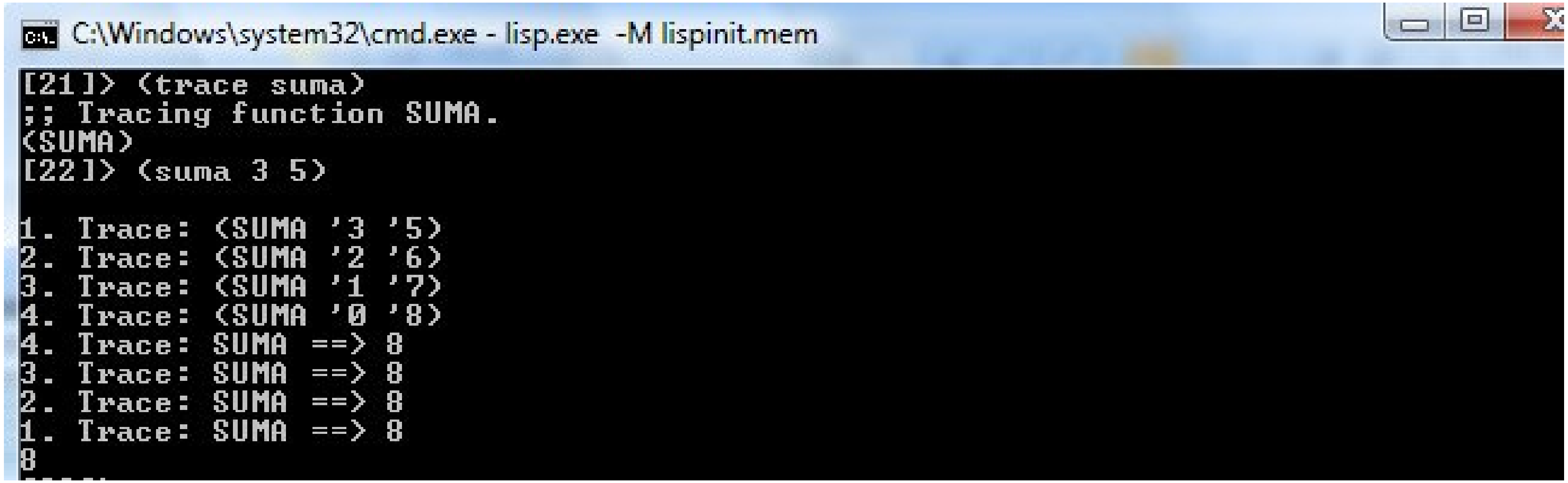


```
(defun suma (n1 n2)
"Intoarce suma a doua numere intregi nenegative"
(if (= n1 0) n2
(suma (- n1 1) (+ n2 1))
)
)
```



```
[16]> (compile-file "sumarec.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\sumarec.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\sumarec.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\sumarec.fas" ;
NIL ;
NIL
[17]> (load "sumarec.lisp")
;; Loading file sumarec.lisp ...
;; Loaded file sumarec.lisp
T
[18]> (suma 3 5)
8
```

Observarea recursivitatii



```
[21]> (trace suma)
;; Tracing function SUMA.
(SUMA)
[22]> (suma 3 5)

1. Trace: (SUMA '3 '5)
2. Trace: (SUMA '2 '6)
3. Trace: (SUMA '1 '7)
4. Trace: (SUMA '0 '8)
4. Trace: SUMA ==> 8
3. Trace: SUMA ==> 8
2. Trace: SUMA ==> 8
1. Trace: SUMA ==> 8
8
```

Definirea unei functii recursive

- Fiecare functie recursiva poate avea formularea:
  - (defun *functie lista_variabile* (cond *perechi_cond*))
  - sau (defun *functie lista_variabile* (if *test then else*)).

- In cazul unei functii recursive corect definite, un apel cu parametri nepotriviti poate genera o recursivitate infinita.

Functia ASSERT

- Pentru a evita argumentele gresite, atunci cand definim o functie putem folosi constructia assert.
- Sintaxa acesteia este:

(assert *asertie* (*variabile_de_schimbat*) *string variabile_mentionate*)

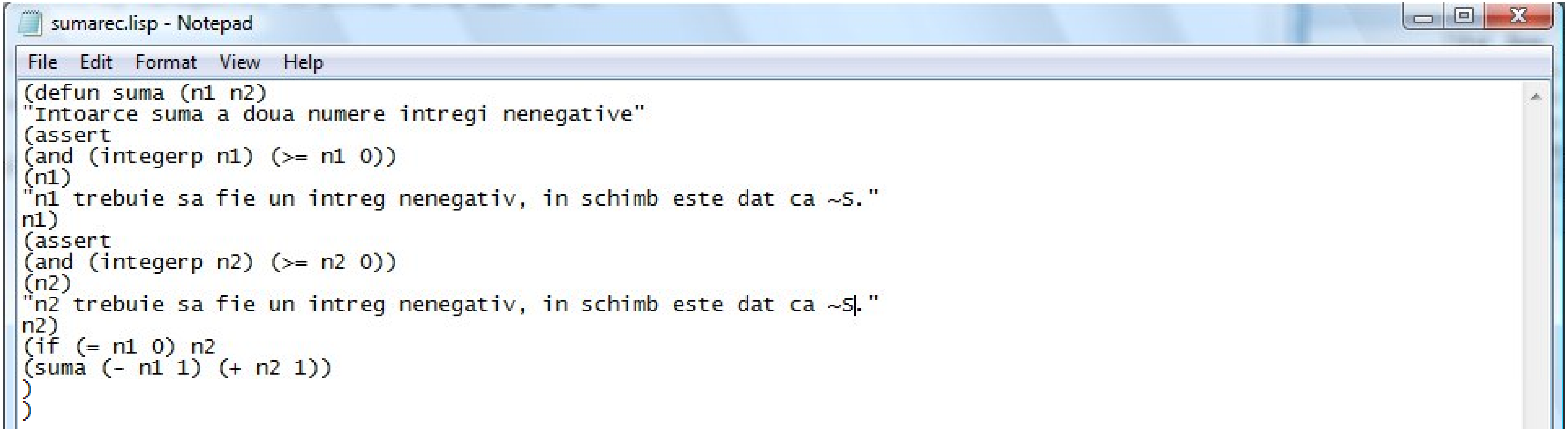
  - ▫ *Asertia* este evaluata.
  - ▫ Daca este True, functia se executa normal.

Functia ASSERT

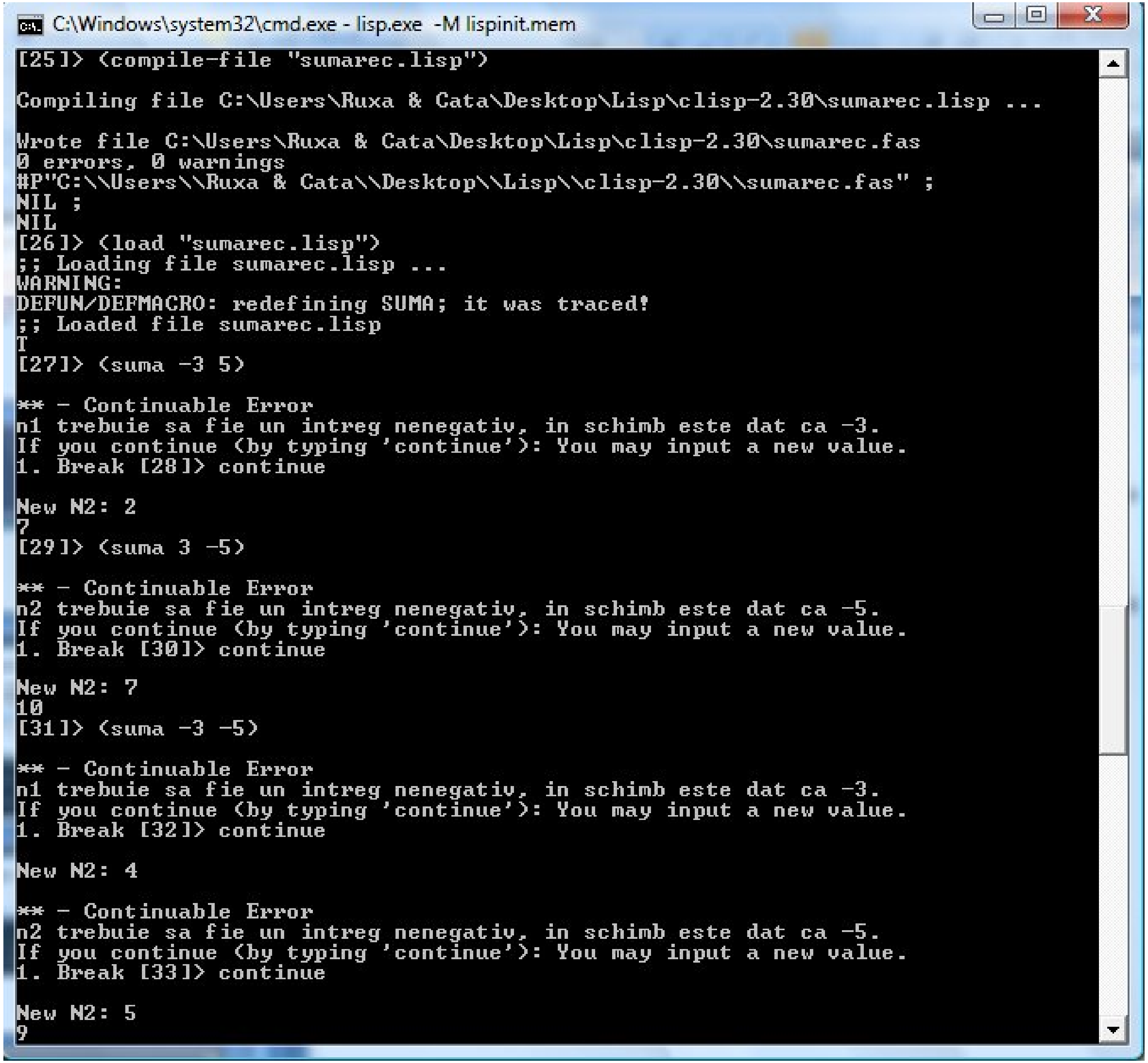
(assert *asertie* (*variabile_de_schimbat*) *string* *variabile_mentionate*)

- Daca este False, Lisp printeaza o eroare:
  - Ii da utilizatorului optiunea de a termina sau de a schimba valorile acelor *variabile_de_schimbat*.
  - Mesajul din *string* este afisat.
  - In acest string putem mentiona anumite variabile, scriind ~S pentru fiecare si trecandu-le in cadrul campului *variabile_mentionate*.

Redefinim suma a doua numere



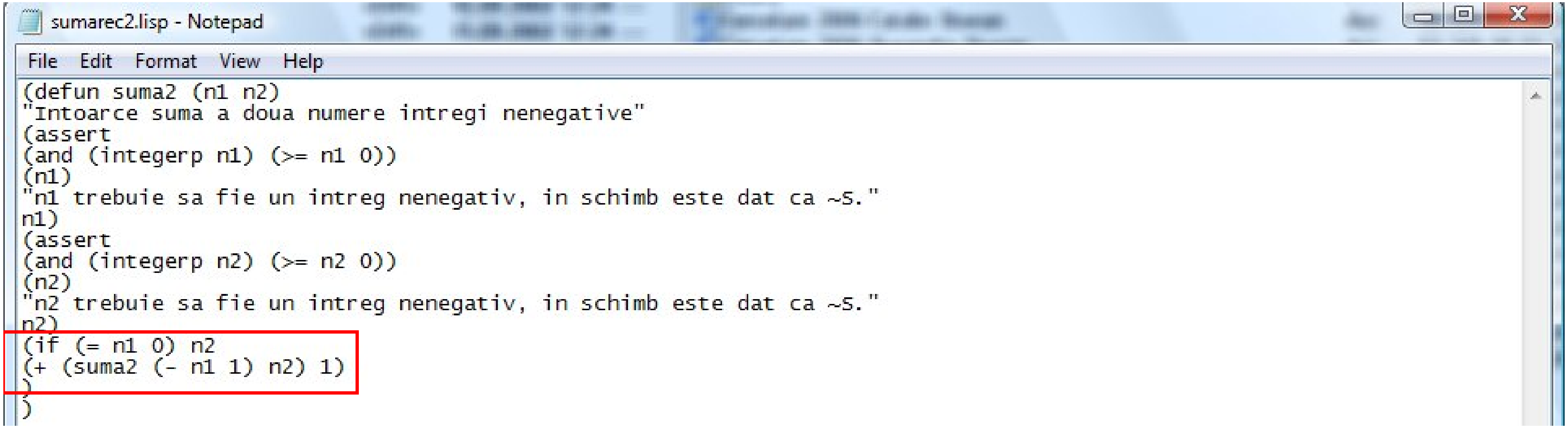
```
(defun suma (n1 n2)
"Intoarce suma a doua numere intregi nenegative"
(assert
(and (integerp n1) (>= n1 0))
(n1)
"n1 trebuie sa fie un intreg nenegativ, in schimb este dat ca ~S."
n1)
(assert
(and (integerp n2) (>= n2 0))
(n2)
"n2 trebuie sa fie un intreg nenegativ, in schimb este dat ca ~S."
n2)
(if (= n1 0) n2
(suma (- n1 1) (+ n2 1))
)
)
```



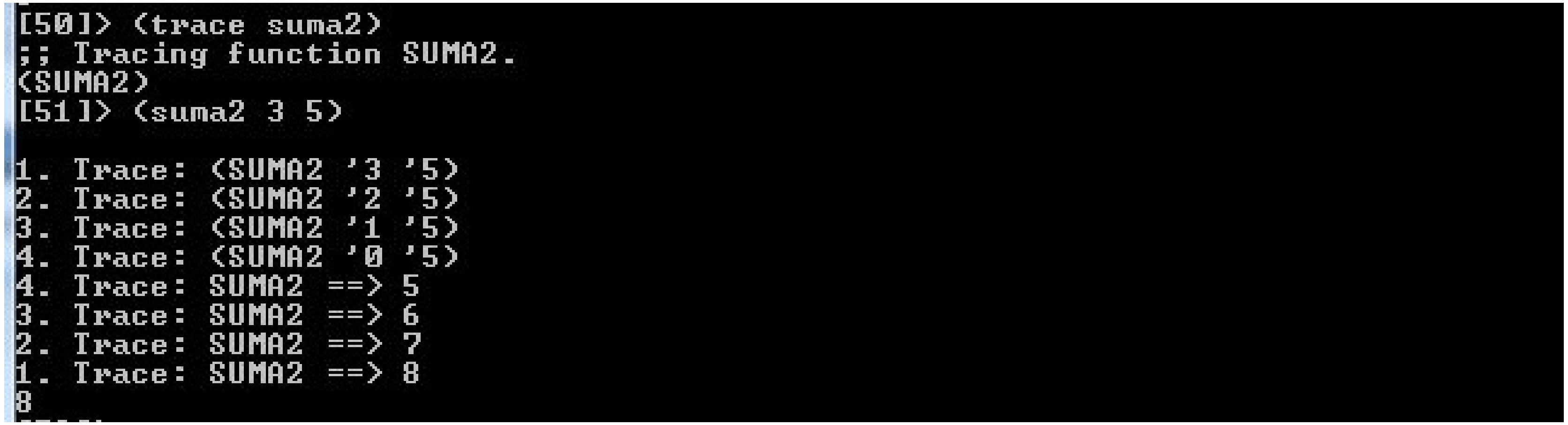
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem

```
[25]> (compile-file "sumarec.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\sumarec.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\sumarec.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\sumarec.fas" ;
NIL ;
NIL
[26]> (load "sumarec.lisp")
;; Loading file sumarec.lisp ...
WARNING:
DEFUN/DEFMACRO: redefining SUMA; it was traced!
;; Loaded file sumarec.lisp
T
[27]> (suma -3 5)

** - Continuable Error
n1 trebuie sa fie un intreg nenegativ, in schimb este dat ca -3.
If you continue (by typing 'continue'): You may input a new value.
1. Break [28]> continue

New N2: 2
7
[29]> (suma 3 -5)

** - Continuable Error
n2 trebuie sa fie un intreg nenegativ, in schimb este dat ca -5.
If you continue (by typing 'continue'): You may input a new value.
1. Break [30]> continue

New N2: 7
10
[31]> (suma -3 -5)

** - Continuable Error
n1 trebuie sa fie un intreg nenegativ, in schimb este dat ca -3.
If you continue (by typing 'continue'): You may input a new value.
1. Break [32]> continue

New N2: 4

** - Continuable Error
n2 trebuie sa fie un intreg nenegativ, in schimb este dat ca -5.
If you continue (by typing 'continue'): You may input a new value.
1. Break [33]> continue

New N2: 5
9
```

O alta versiune a sumei

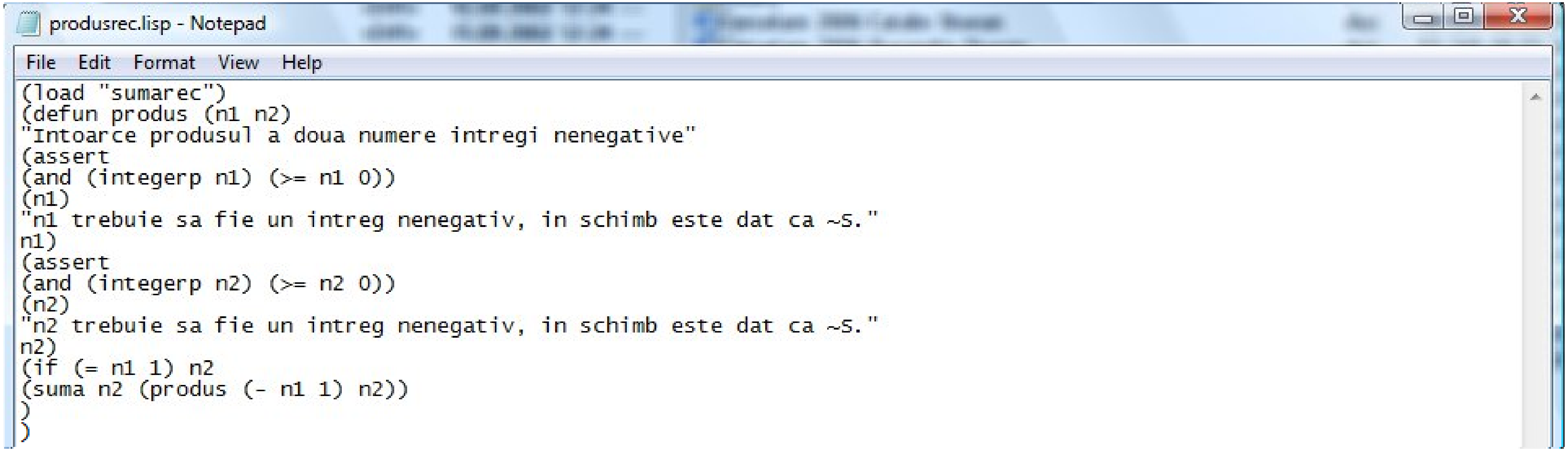


```
(defun suma2 (n1 n2)
"Intoarce suma a doua numere intregi nenegative"
(assert
(and (integerp n1) (>= n1 0))
(n1)
"n1 trebuie sa fie un intreg nenegativ, in schimb este dat ca ~S."
n1)
(assert
(and (integerp n2) (>= n2 0))
(n2)
"n2 trebuie sa fie un intreg nenegativ, in schimb este dat ca ~S."
n2)
(if (= n1 0) n2
(+ (suma2 (- n1 1) n2) 1)
)
)
```



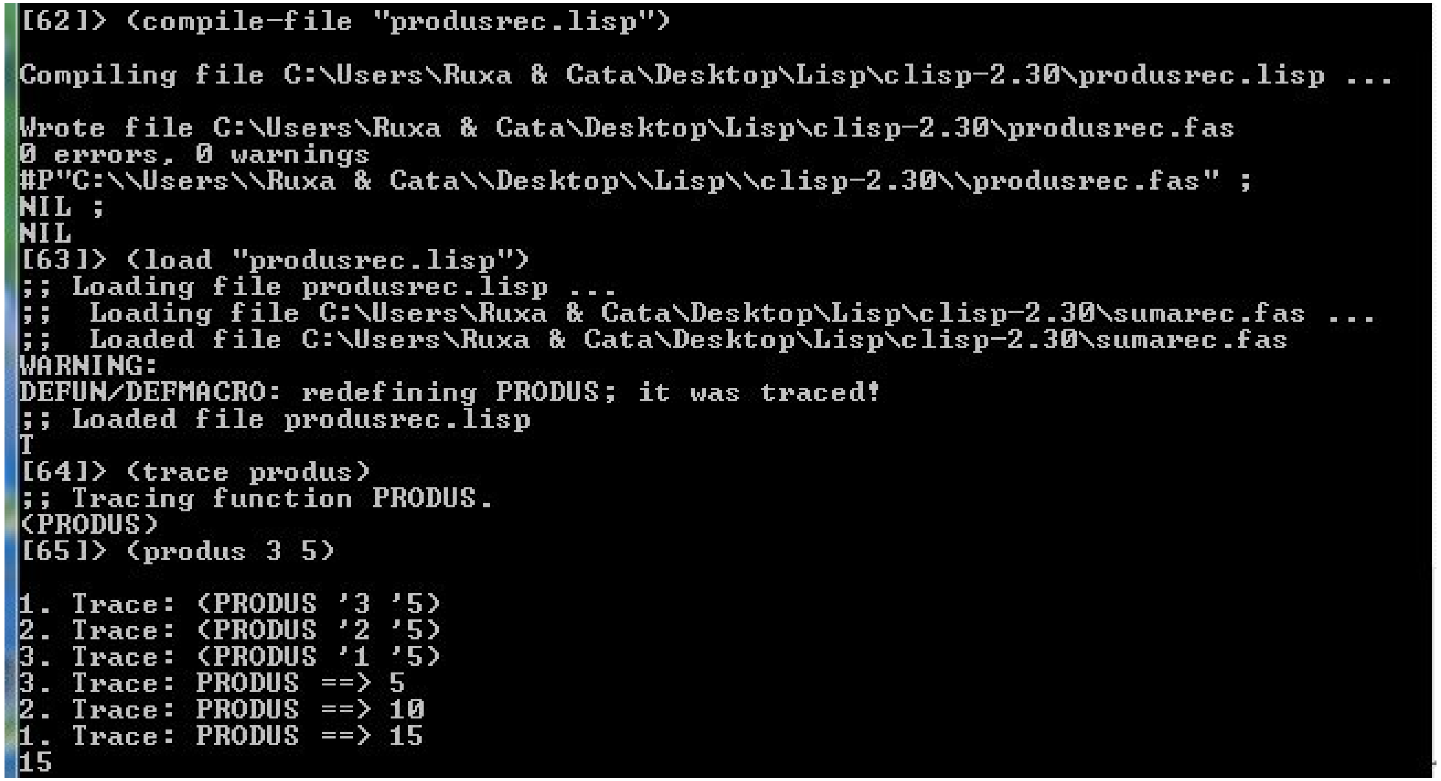
```
[50]> (trace suma2)
;; Tracing function SUMA2.
(SUMA2)
[51]> (suma2 3 5)

1. Trace: (SUMA2 '3 '5)
2. Trace: (SUMA2 '2 '5)
3. Trace: (SUMA2 '1 '5)
4. Trace: (SUMA2 '0 '5)
4. Trace: SUMA2 ==> 5
3. Trace: SUMA2 ==> 6
2. Trace: SUMA2 ==> 7
1. Trace: SUMA2 ==> 8
8
```

Produsul a doi intregi nenegativi



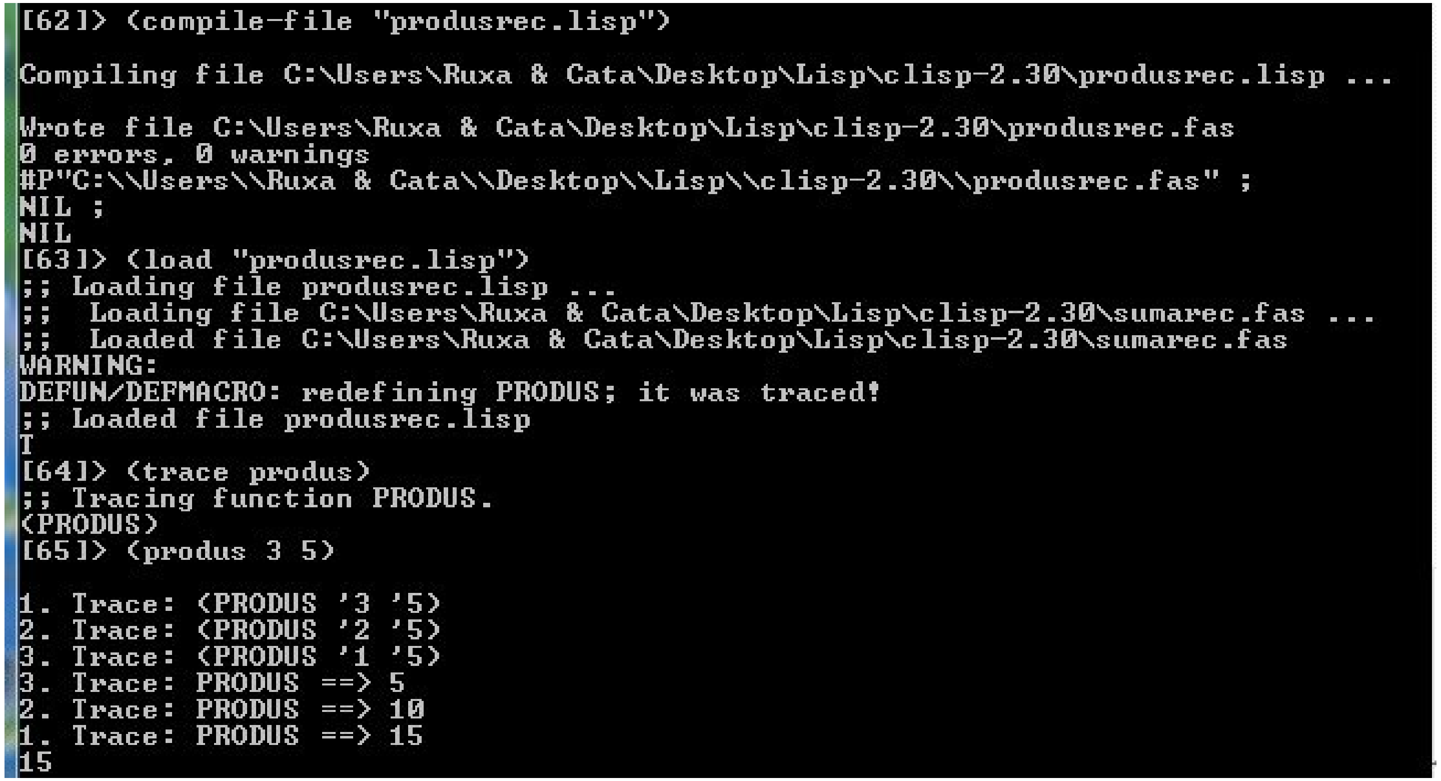
```
(load "sumarec")
(defun produs (n1 n2)
"Intoarce produsul a doua numere intregi nenegative"
(assert
(and (integerp n1) (>= n1 0))
(n1)
"n1 trebuie sa fie un intreg nenegativ, in schimb este dat ca ~S."
n1)
(assert
(and (integerp n2) (>= n2 0))
(n2)
"n2 trebuie sa fie un intreg nenegativ, in schimb este dat ca ~S."
n2)
(if (= n1 1) n2
(suma n2 (produs (- n1 1) n2))
)
)
```

Produsul a doi intregi nenegativi



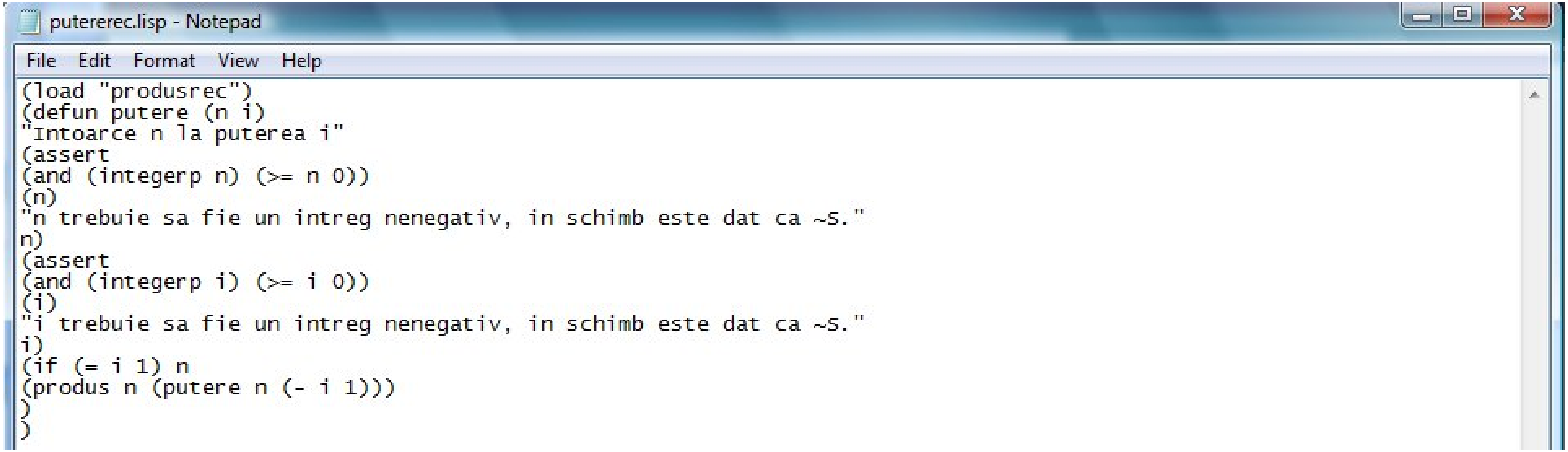
```
[62]> (compile-file "produsrec.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\produsrec.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\produsrec.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\produsrec.fas" ;
NIL ;
NIL
[63]> (load "produsrec.lisp")
;; Loading file produsrec.lisp ...
;;  Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\sumarec.fas ...
;;  Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\sumarec.fas
WARNING:
DEFUN/DEFMACRO: redefining PRODUS; it was traced!
;; Loaded file produsrec.lisp
T
[64]> (trace produs)
;; Tracing function PRODUS.
(PRODUS)
[65]> (produs 3 5)

1. Trace: (PRODUS '3 '5)
2. Trace: (PRODUS '2 '5)
3. Trace: (PRODUS '1 '5)
3. Trace: PRODUS ==> 5
2. Trace: PRODUS ==> 10
1. Trace: PRODUS ==> 15
15
```

Produsul a doi intregi nenegativi

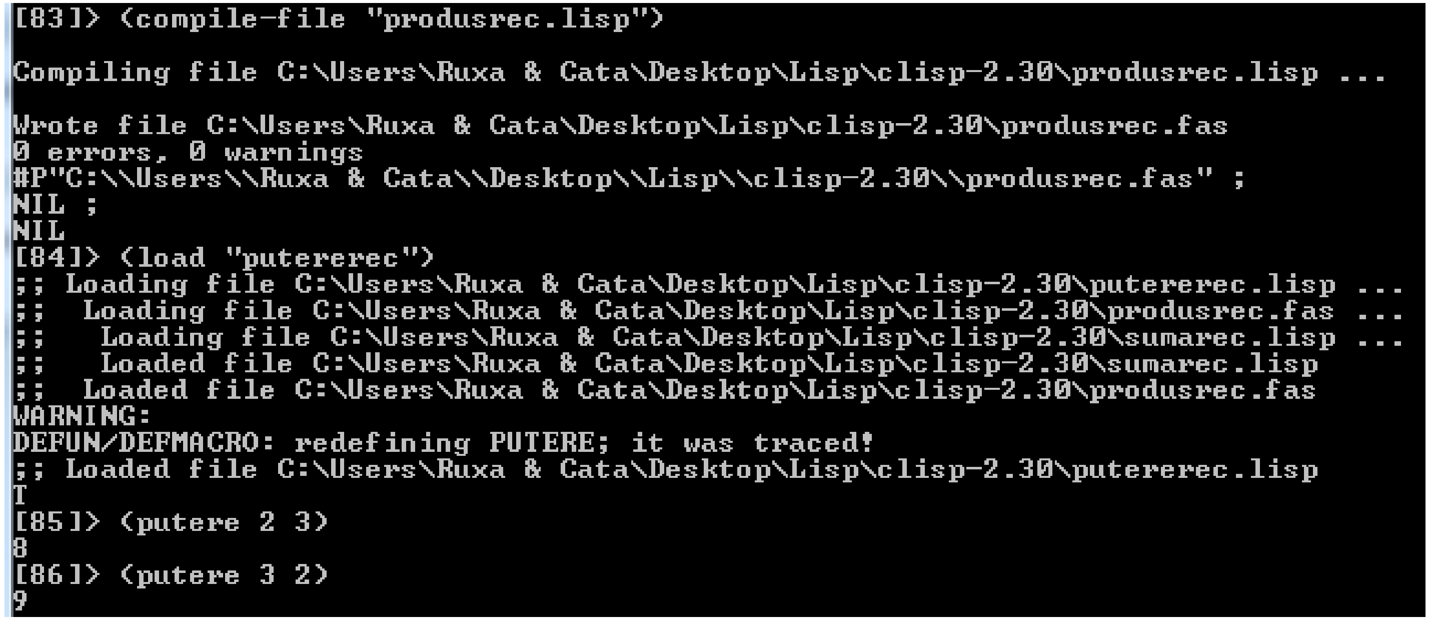


```
[62]> (compile-file "produsrec.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\produsrec.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\produsrec.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\produsrec.fas" ;
NIL ;
NIL
[63]> (load "produsrec.lisp")
;; Loading file produsrec.lisp ...
;;  Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\sumarec.fas ...
;;  Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\sumarec.fas
WARNING:
DEFUN/DEFMACRO: redefining PRODUS; it was traced!
;; Loaded file produsrec.lisp
T
[64]> (trace produs)
;; Tracing function PRODUS.
(PRODUS)
[65]> (produs 3 5)

1. Trace: (PRODUS '3 '5)
2. Trace: (PRODUS '2 '5)
3. Trace: (PRODUS '1 '5)
3. Trace: PRODUS ==> 5
2. Trace: PRODUS ==> 10
1. Trace: PRODUS ==> 15
15
```

Ridicarea unui numar la putere



```lisp
(load "produsrec")
(defun putere (n i)
"Intoarce n la puterea i"
(assert
(and (integerp n) (>= n 0))
(n)
"n trebuie sa fie un intreg nenegativ, in schimb este dat ca ~S."
n)
(assert
(and (integerp i) (>= i 0))
(i)
"i trebuie sa fie un intreg nenegativ, in schimb este dat ca ~S."
i)
(if (= i 1) n
(produs n (putere n (- i 1)))
)
)
```



```
[83]> (compile-file "produsrec.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\produsrec.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\produsrec.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\produsrec.fas" ;
NIL ;
NIL
[84]> (load "putererec")
;; Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\putererec.lisp ...
;;  Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\produsrec.fas ...
;;   Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\sumarec.lisp ...
;;   Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\sumarec.lisp
;;  Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\produsrec.fas
WARNING:
DEFUN/DEFMACRO: redefining PUTERE; it was traced!
;; Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\putererec.lisp
T
[85]> (putere 2 3)
8
[86]> (putere 3 2)
9
```

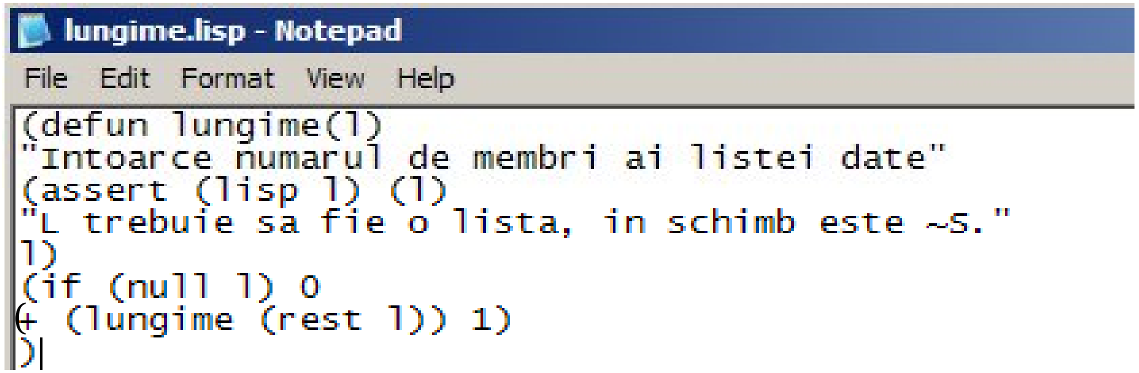
Recursivitatea la liste

- Ca prim exemplu, sa incercam definirea versiunii proprii a functiei length, care determina lungimea unei liste.

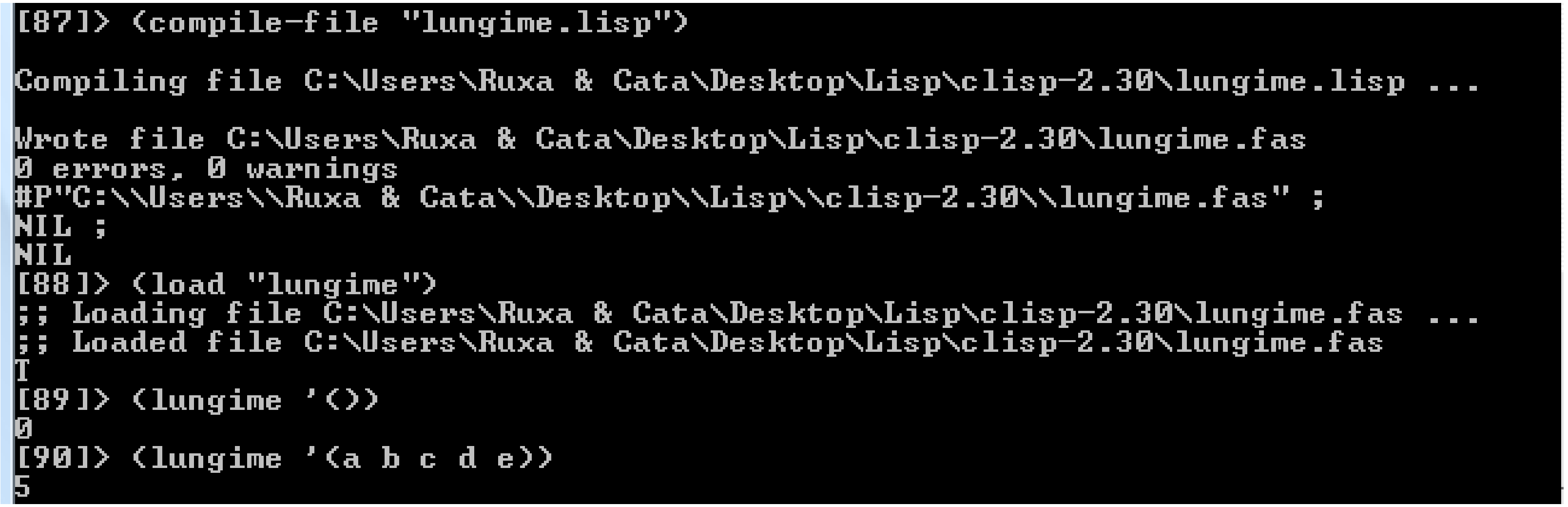
- **Partea recursiva**: Lungimea unei liste nevide este cu o unitate mai mare decat lungimea restului listei.

- **Conditia de terminare**: Lungimea listei vide () este 0.

Lungimea unei liste

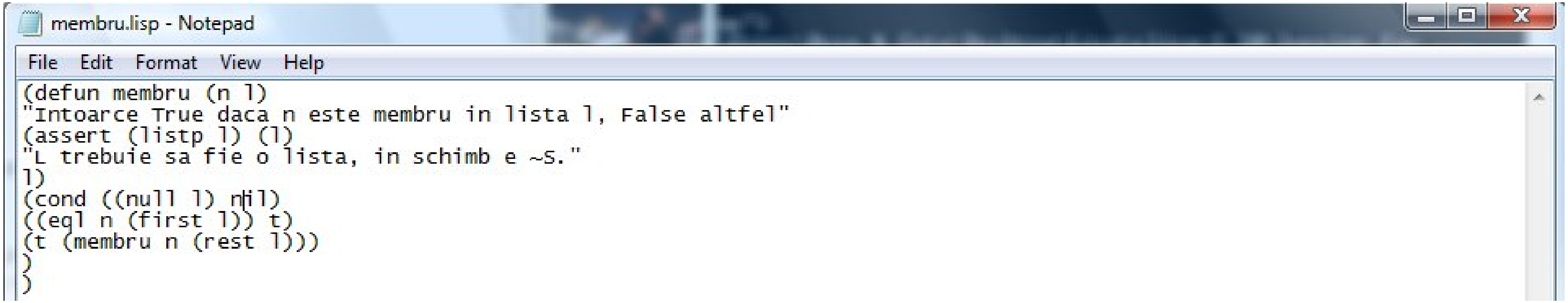


```
(defun lungime(l)
"Intoarce numarul de membri ai listei date"
(assert (lisp l) (l)
"L trebuie sa fie o lista, in schimb este ~S."
l)
(if (null l) 0
(+ (lungime (rest l)) 1)
)
```

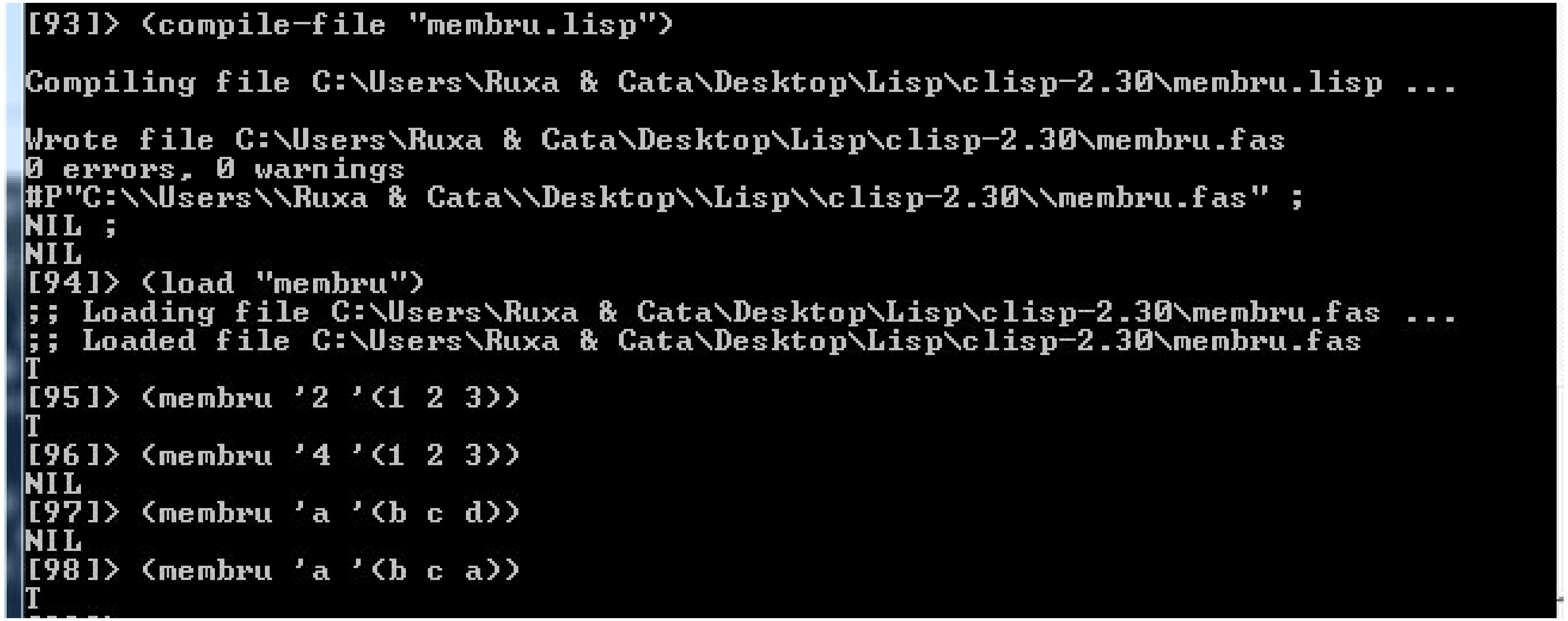


```
[87]> (compile-file "lungime.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\lungime.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\lungime.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\lungime.fas" ;
NIL ;
NIL
[88]> (load "lungime")
;; Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\lungime.fas ...
;; Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\lungime.fas
T
[89]> (lungime '())
0
[90]> (lungime '(a b c d e))
5
```

Apartenenta unui element la o lista

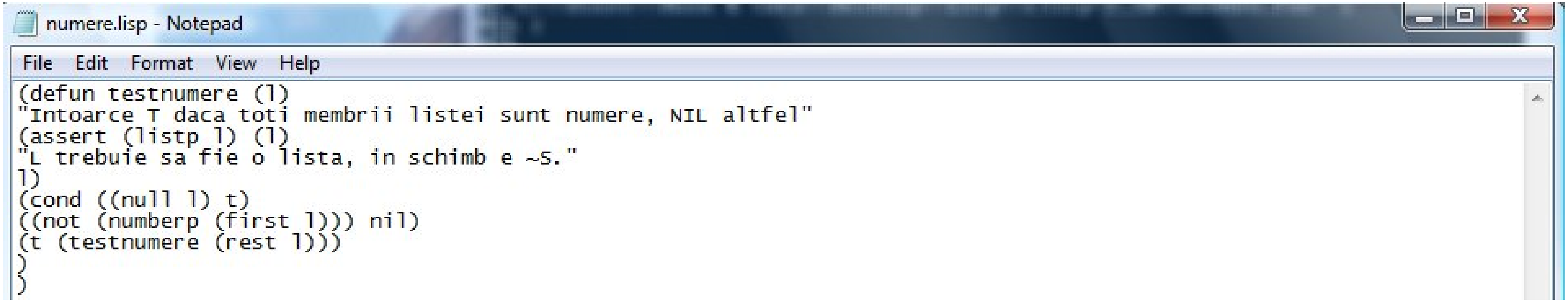


```
(defun membru (n l)
"Intoarce True daca n este membru in lista l, False altfel"
(assert (listp l) (l)
"L trebuie sa fie o lista, in schimb e ~S."
l)
(cond ((null l) nil)
((eql n (first l)) t)
(t (membru n (rest l)))
)
)
```

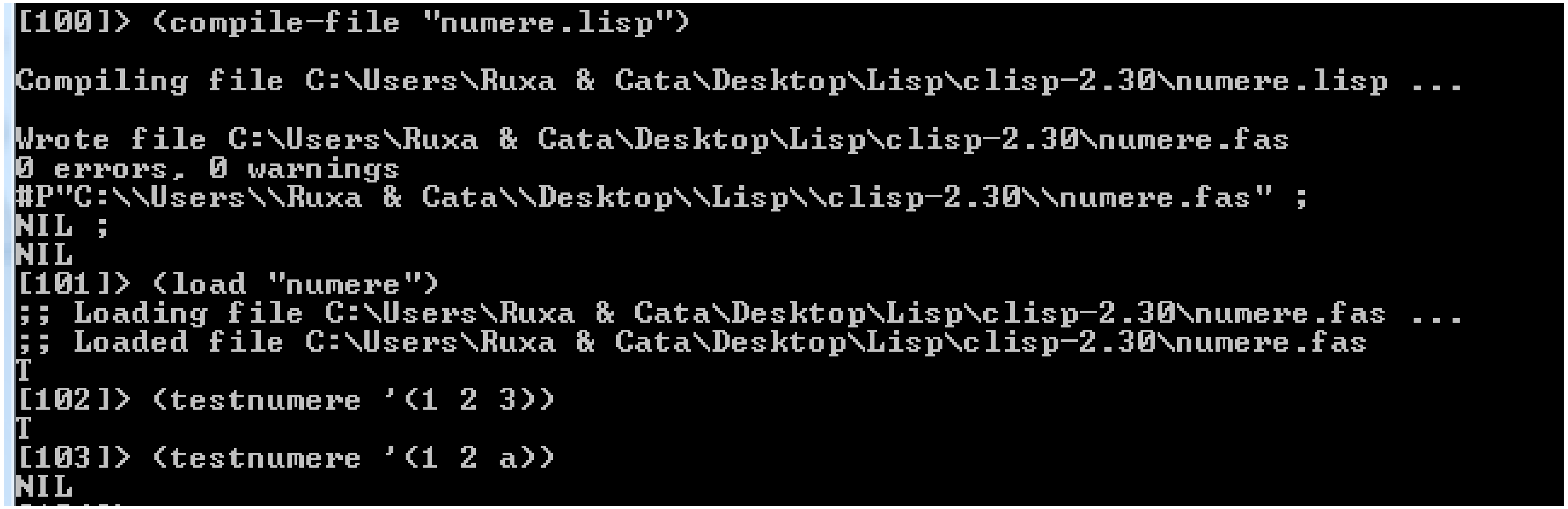


```
[93]> (compile-file "membru.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\membru.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\membru.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\membru.fas" ;
NIL ;
NIL
[94]> (load "membru")
;; Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\membru.fas ...
;; Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\membru.fas
T
[95]> (membru '2 '(1 2 3))
T
[96]> (membru '4 '(1 2 3))
NIL
[97]> (membru 'a '(b c d))
NIL
[98]> (membru 'a '(b c a))
T
```

Testarea daca o lista e formata sau nu numai din numere

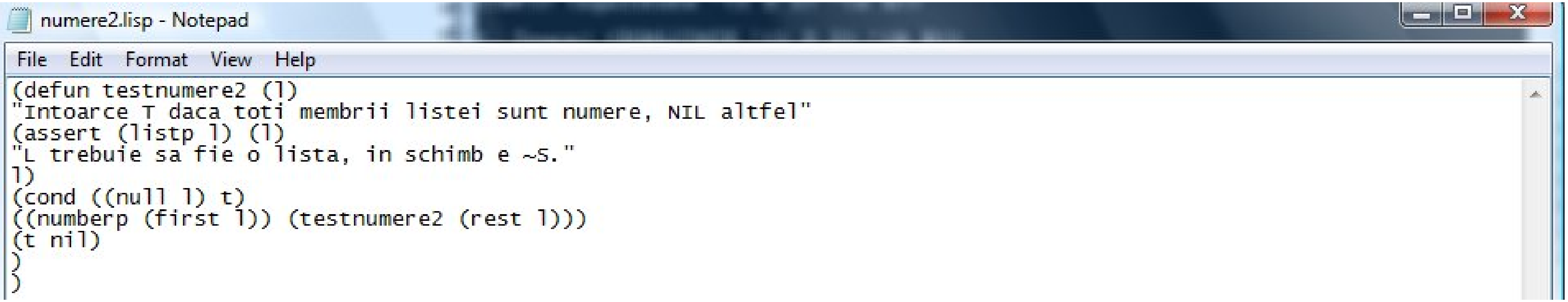


```
(defun testnumere (l)
"Intoarce T daca toti membrii listei sunt numere, NIL altfel"
(assert (listp l) (l)
"L trebuie sa fie o lista, in schimb e ~S."
l)
(cond ((null l) t)
((not (numberp (first l))) nil)
(t (testnumere (rest l)))
)
)
```

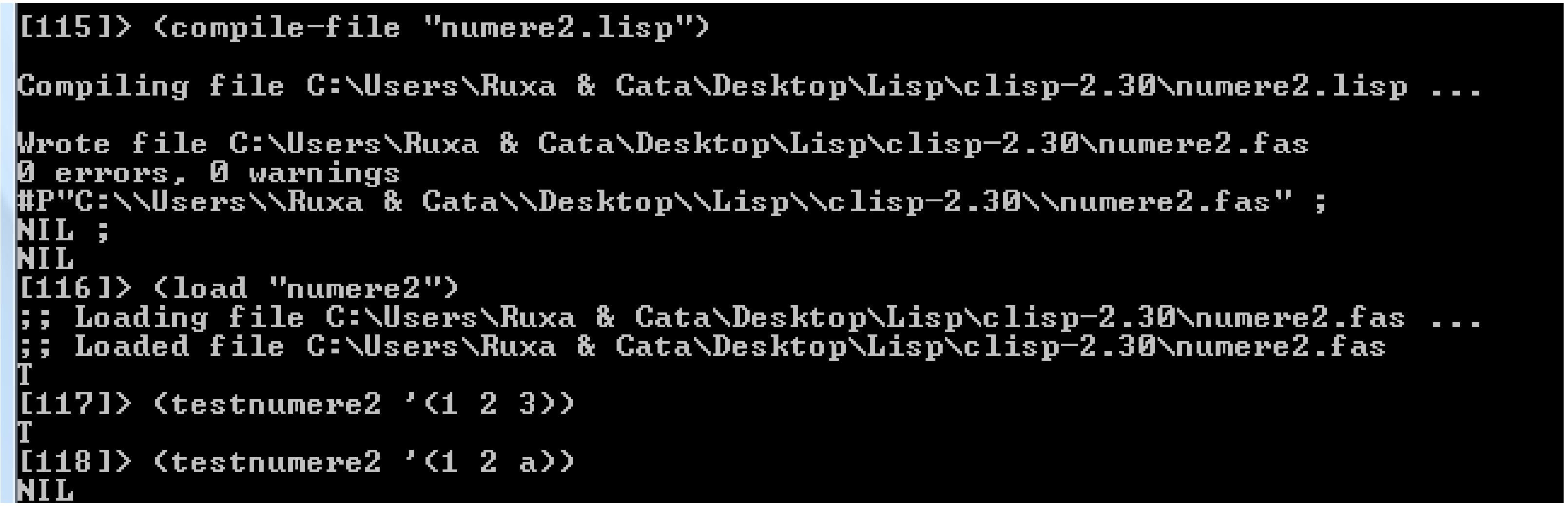


```
[100]> (compile-file "numere.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\numere.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\numere.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\numere.fas" ;
NIL ;
NIL
[101]> (load "numere")
;; Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\numere.fas ...
;; Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\numere.fas
T
[102]> (testnumere '(1 2 3))
T
[103]> (testnumere '(1 2 a))
NIL
```

Testarea daca o lista e formata sau nu numai din numere – alta versiune

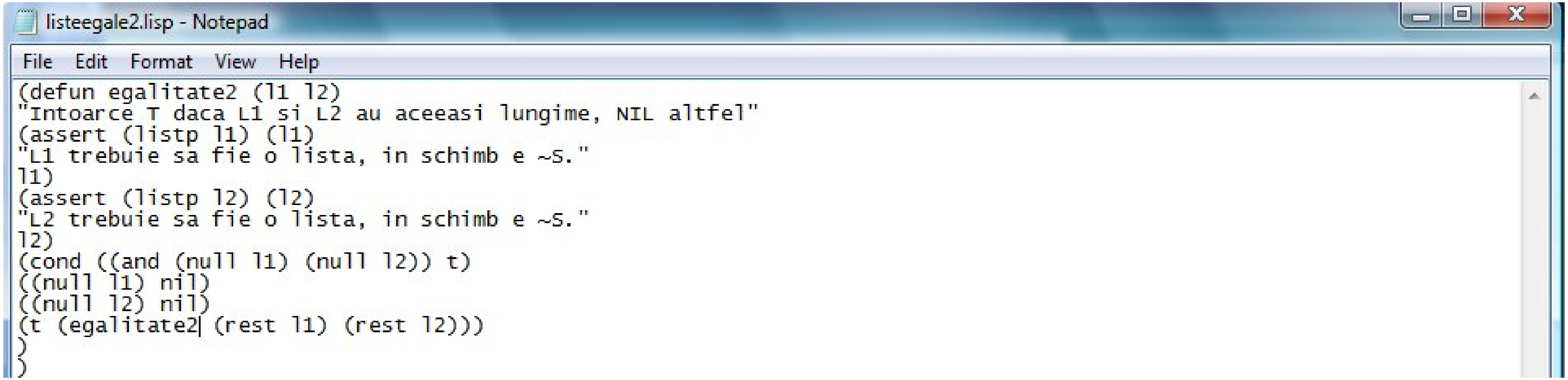


```lisp
(defun testnumere2 (l)
"Intoarce T daca toti membrii listei sunt numere, NIL altfel"
(assert (listp l) (l)
"L trebuie sa fie o lista, in schimb e ~S."
l)
(cond ((null l) t)
((numberp (first l)) (testnumere2 (rest l)))
(t nil)
)
)
```

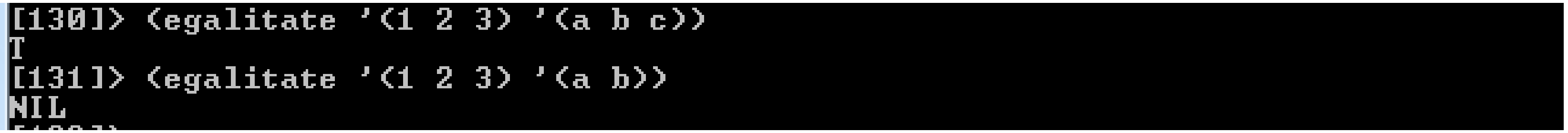


```
[115]> (compile-file "numere2.lisp")

Compiling file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\numere2.lisp ...

Wrote file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\numere2.fas
0 errors, 0 warnings
#P"C:\\Users\\Ruxa & Cata\\Desktop\\Lisp\\clisp-2.30\\numere2.fas" ;
NIL ;
NIL
[116]> (load "numere2")
;; Loading file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\numere2.fas ...
;; Loaded file C:\Users\Ruxa & Cata\Desktop\Lisp\clisp-2.30\numere2.fas
T
[117]> (testnumere2 '(1 2 3))
T
[118]> (testnumere2 '(1 2 a))
NIL
```

Verificarea egalitatii lungimii a doua liste

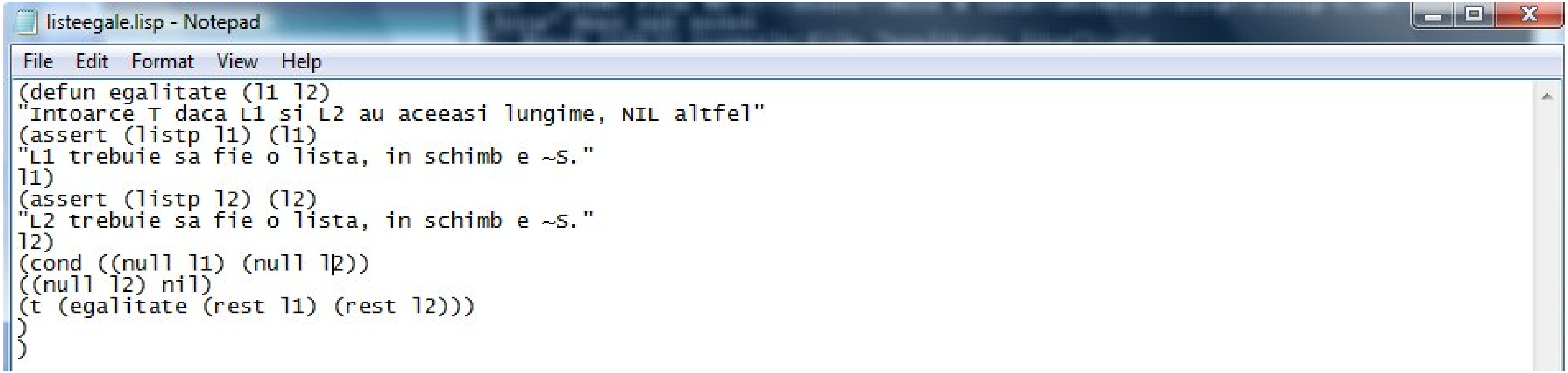


```
(defun egalitate2 (l1 l2)
"Intoarce T daca L1 si L2 au aceeasi lungime, NIL altfel"
(assert (listp l1) (l1)
"L1 trebuie sa fie o lista, in schimb e ~S."
l1)
(assert (listp l2) (l2)
"L2 trebuie sa fie o lista, in schimb e ~S."
l2)
(cond ((and (null l1) (null l2)) t)
((null l1) nil)
((null l2) nil)
(t (egalitate2 (rest l1) (rest l2)))
)
)
```



```
[130]> (egalitate '(1 2 3) '(a b c))
T
[131]> (egalitate '(1 2 3) '(a b))
NIL
```

Verificarea egalitatii lungimii a doua liste - varianta



```
(defun egalitate (l1 l2)
"Intoarce T daca L1 si L2 au aceeasi lungime, NIL altfel"
(assert (listp l1) (l1)
"L1 trebuie sa fie o lista, in schimb e ~S."
l1)
(assert (listp l2) (l2)
"L2 trebuie sa fie o lista, in schimb e ~S."
l2)
(cond ((null l1) (null l2))
((null l2) nil)
(t (egalitate (rest l1) (rest l2)))
)
)
```

Pe saptamana viitoare…