

Programmare in Lisp pur

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

Definirea propriilor functii

- “Un program Lisp este o colectie de functii scrise de un programator Lisp” - S. C. Shapiro.
- Pentru a defini o functie, se foloseste forma **defun**:

(defun *functie lista_variabile string_documentie forma*)

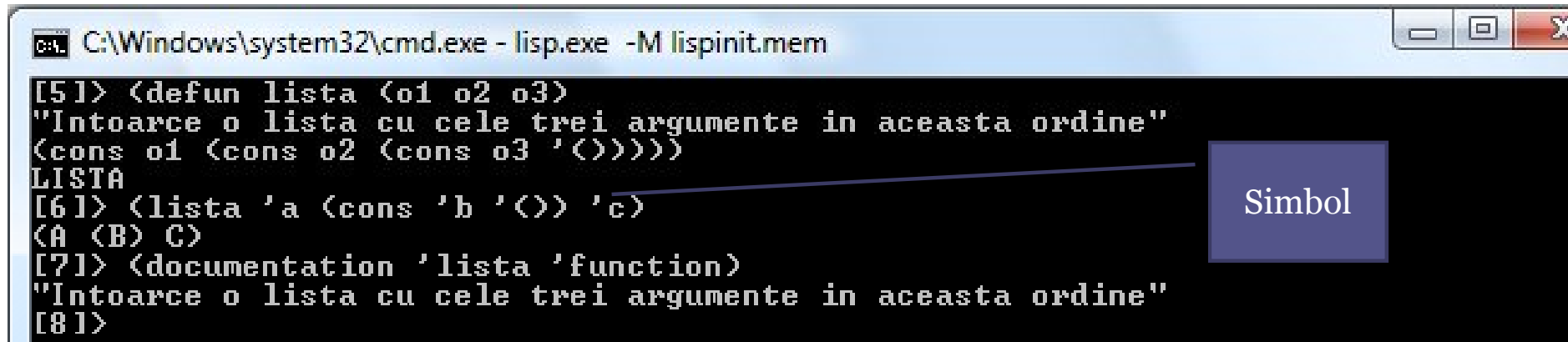
- *functie* este un simbol;
- *lista_variabile* este o lista de simboluri;
- *string_documentatie* este un string;
- *forma* este o forma Lisp.

Definirea propriilor functii

(**defun** *functie lista_variabile string_documentie forma*)

- **defun** intoarce *functie*;
 - defineste *functie* ca fiind numele unei functii;
 - argumentele (atributele) sale formale sunt simbolurile din *lista_variabile*;
 - definitia sa se afla in *forma*;
 - documentatia (explicatiile) pentru aceasta functie sunt in *string_documentatie*.
- Observatie: Argumentele formale ale functiei poarta numele de **variabile lambda**.
 - Numele provine de la **calculul lambda** al lui A. Church care sta la baza Lisp-ului.

Exemplu



```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[5]> (defun lista (o1 o2 o3)
      "Intoarce o lista cu cele trei argumente in aceasta ordine"
      (cons o1 (cons o2 (cons o3 '()))))
LISTA
[6]> (lista 'a (cons 'b '()) 'c)
(A (B) C)
[7]> (documentation 'lista 'function)
"Intoarce o lista cu cele trei argumente in aceasta ordine"
[8]>
```

Simbol

- Se definește funcția lista.
- Aceasta ia trei obiecte Lisp ca atribute actuale.
- Intoarce o lista care are ca membri cele trei obiecte Lisp date ca argumente.
- După ce funcția este evaluată, aceasta se poate folosi ca și cum ar fi una predefinită în Lisp.
- Putem afla de asemenea informații despre funcție cu expresia **documentation**.

Apelarea unei functii

- Cand o functie este apelata, se parcurg urmatoorii pasi:
 - Lisp verifica daca primul membru al listei este un simbol care reprezinta o functie;
 - Obiectele date ca argumente sunt evaluate;
 - Valorile obiectelor devin valorile atributelor formale.
 - Variabilele formale sunt **legate** la valorile date;
 - Forma care reprezinta definitia functiei este evaluata;
 - Atributele formale sunt dezlegate;
 - Valoarea formei de definitie este intoarsa.

Apelarea unei functii - exemplu

- Cand o functie este apelata, se parcurg urmatoorii pasi:
 - Lisp verifica daca primul membru al listei este un simbol care reprezinta o functie;
 - Obiectele date ca argumente sunt evaluate;
 - Valorile obiectelor devin valorile atributelor formale, adica variabilele formale sunt **legate** la valorile date;
 - Forma care reprezinta definitia functiei este evaluata;
 - Atributele formale sunt dezlegate;
 - Valoarea formei de definitie este intoarsa.
- Cand functia definita este apelata, avem urmatoorii pasi:
 - **lista** este un simbol care reprezinta o functie;
 - 'a este evaluat drept A, (cons 'b '()) ca (B) si 'c drept C;
 - o1 este legat la A, o2 la (B) si o3 la C;
 - (cons o1 (cons o2 (cons o3 '()))) este evaluata, fiecare obiect avand valorile de mai sus;
 - o1, o2 si o3 revin la valorile initiale;
 - Se intoarce (A (B) C).

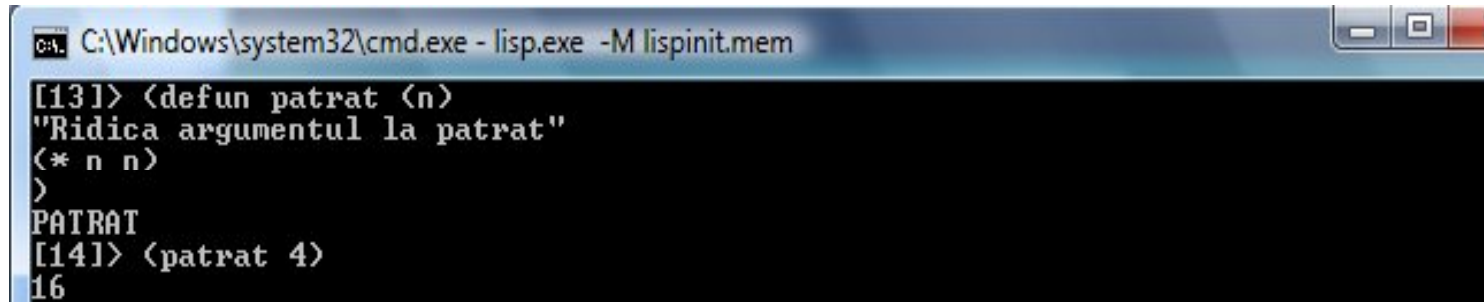
Reversul unei liste de doua numere

- Folosind functiile predefinite **first** si **second** care dau primul si cel de-al doilea element al unei liste, sa se defineasca o functie care inverseaza cei doi membri ai unei liste.

```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[1]> (defun listainv (o1 o2)
"Intoarce o lista cu cele doua argumente in ordine"
(cons o1 (cons o2 '())))
)
LISTAINV
[2]> (defun interschimba (lista)
"Dandu-se o lista de doua numere, intoarce lista cu elementele interschimbate"
(listainv (second lista) (first lista))
)
INTERSCHIMBA
[3]> (listainv (second '(a b)) (first '(a b)))
(B A)
[4]> (listainv 'b 'a)
(B A)
[5]> (interschimba '(a b))
(B A)
```

Patratul unui numar

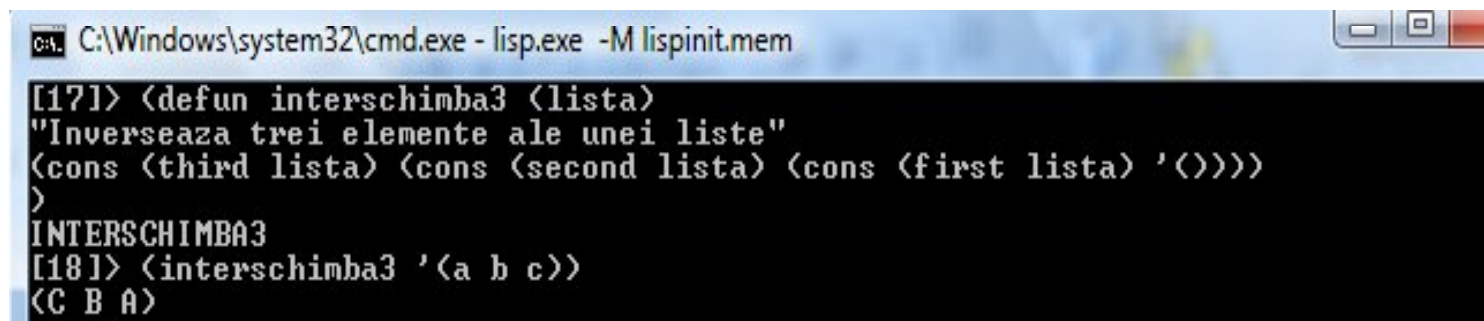
- Definiti o functie care sa calculeze patratal unui numar dat n .



```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[13]> (defun patrat (n)
"Ridica argumentul la patrat"
(* n n)
)
PATRAT
[14]> (patrat 4)
16
```


Reversul unei liste de trei numere

- Folosind functia predefinita `third` care da elementul de pe pozitia a treia dintr-o lista, sa se defineasca o functie care inverseaza cei trei membri ai unei liste.



```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[17]> (defun interschimba3 (lista)
  "Inverseaza trei elemente ale unei liste"
  (cons (third lista) (cons (second lista) (cons (first lista) '()))))
)
INTERSCHIMBA3
[18]> (interschimba3 '(a b c))
(C B A)
```

Definirea de functii in pachete

- Sa definim un nou pachet pentru a defini functiile personale.
- Ne mutam din pachetul curent in cel nou definit.
- Sa definim, de exemplu, pachetul *invatare*.
- In interiorul sau sa definim functia *fn*.

Definirea de functii in pachete

- Apelam functia **describe** pentru a vedea daca fn este un simbol mostenit din alt pachet.
- Daca acesta este cazul, apelam functia **shadow** avand ca argument functia existenta.
- Reapelam **describe** pentru a fi siguri ca fn este acum simbol al pachetului *invatare*.

Definirea de functii in pachete

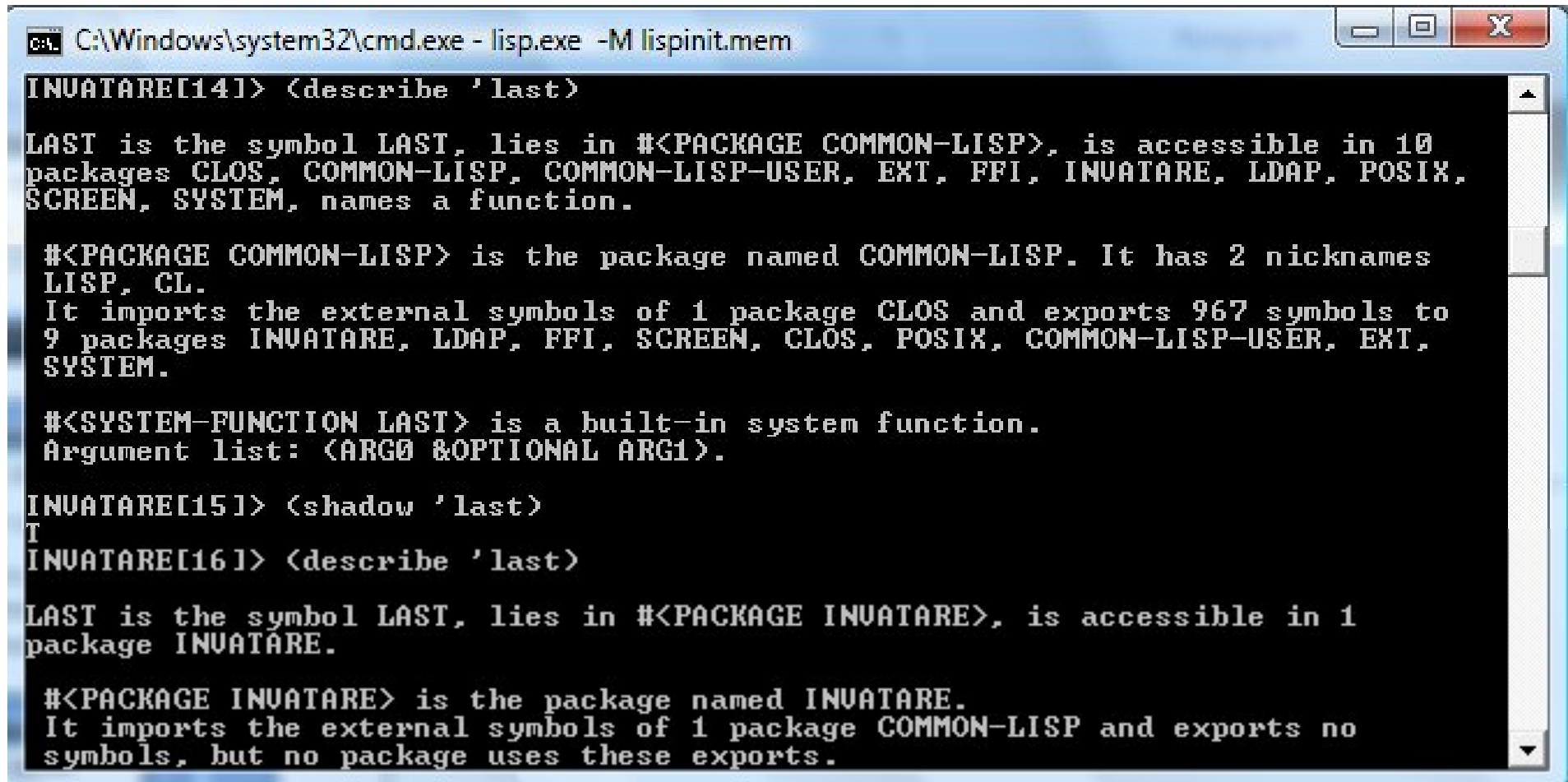
- Definim functia fn .
- Testam functia nou definita.
- Verificam daca functia fn originala se poate inca folosi, utilizand formularea ce include pachetul care o exporta.
- Exportam simbolul functional nou definit, daca dorim folosirea sa si in alte pachete.

Redefinirea functiei predefinite **last**

- last intoarce o lista formata din ultimul element dintr-o lista data.
- In versiunea noastra, va intoarce al treilea element al unei liste date.

```
[10]> *package*  
#<PACKAGE COMMON-LISP-USER>  
[11]> (defpackage invatare)  
#<PACKAGE INVATARE>  
[12]> (in-package invatare)  
#<PACKAGE INVATARE>
```

Continuare



```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
INUATARE[14]> (describe 'last)
LAST is the symbol LAST, lies in #<PACKAGE COMMON-LISP>, is accessible in 10
packages CLOS, COMMON-LISP, COMMON-LISP-USER, EXT, FFI, INUATARE, LDAP, POSIX,
SCREEN, SYSTEM, names a function.

#<PACKAGE COMMON-LISP> is the package named COMMON-LISP. It has 2 nicknames
LISP, CL.
It imports the external symbols of 1 package CLOS and exports 967 symbols to
9 packages INUATARE, LDAP, FFI, SCREEN, CLOS, POSIX, COMMON-LISP-USER, EXT,
SYSTEM.

#<SYSTEM-FUNCTION LAST> is a built-in system function.
Argument list: (ARG0 &OPTIONAL ARG1).

INUATARE[15]> (shadow 'last)
T
INUATARE[16]> (describe 'last)
LAST is the symbol LAST, lies in #<PACKAGE INUATARE>, is accessible in 1
package INUATARE.

#<PACKAGE INUATARE> is the package named INUATARE.
It imports the external symbols of 1 package COMMON-LISP and exports no
symbols, but no package uses these exports.
```

Continuare

```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
INVATARE[17]> (defun last (lista)
"Intoarce al treilea element al unei liste care are trei sau mai multe elemente"
(third lista))
LAST
INVATARE[18]> (last '(a b c))
C
INVATARE[19]> (lisp:last '(a b c))
(C)
INVATARE[20]> (export 'last)
T
INVATARE[21]> (in-package common-lisp-user)
#<PACKAGE COMMON-LISP-USER>
[22]> (invatare:last '(a b c d e))
C
[23]> (lisp:last '(a b c d e))
(E)
```

Alt exemplu

- Sa definim o functie care sa recunoasca semnul intrebarii, ?.

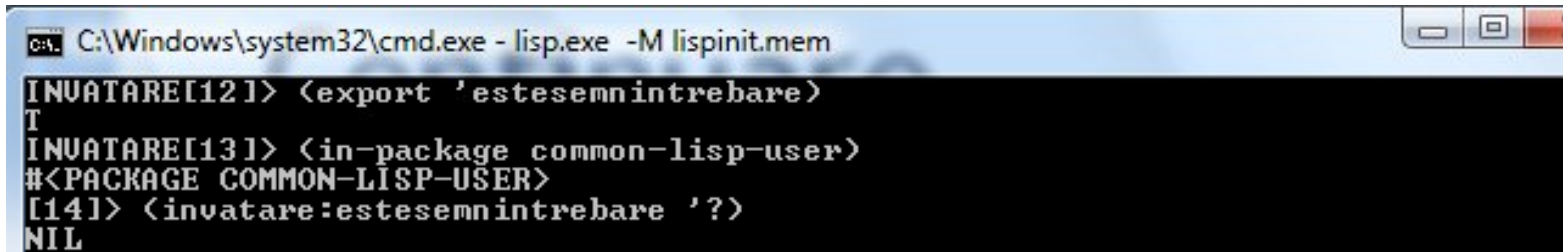
```
ca. C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[21] <in-package invatare>
#<PACKAGE INVATARE>
INVATARE[3] <describe 'estesemnintrebare>

ESTESEMNINTREBARE is the symbol ESTESEMNINTREBARE, lies in #<PACKAGE
INVATARE>, is accessible in 1 package INVATARE.

#<PACKAGE INVATARE> is the package named INVATARE.
It imports the external symbols of 1 package COMMON-LISP and exports no
symbols, but no package uses these exports.

INVATARE[4] <defun estesemnintrebare (o)
"Intoarce True daca o este simbolul de semn de intrebare"
>
ESTESEMNINTREBARE
INVATARE[5] <defun estesemnintrebare (o)
"Intoarce T daca o este simbolul de semn de intrebare; NIL, altfel"
(eql o '?)>
ESTESEMNINTREBARE
INVATARE[6] <estesemnintrebare '?)>
T
INVATARE[7] <estesemnintrebare 'ceva>
NIL
```


Problema!

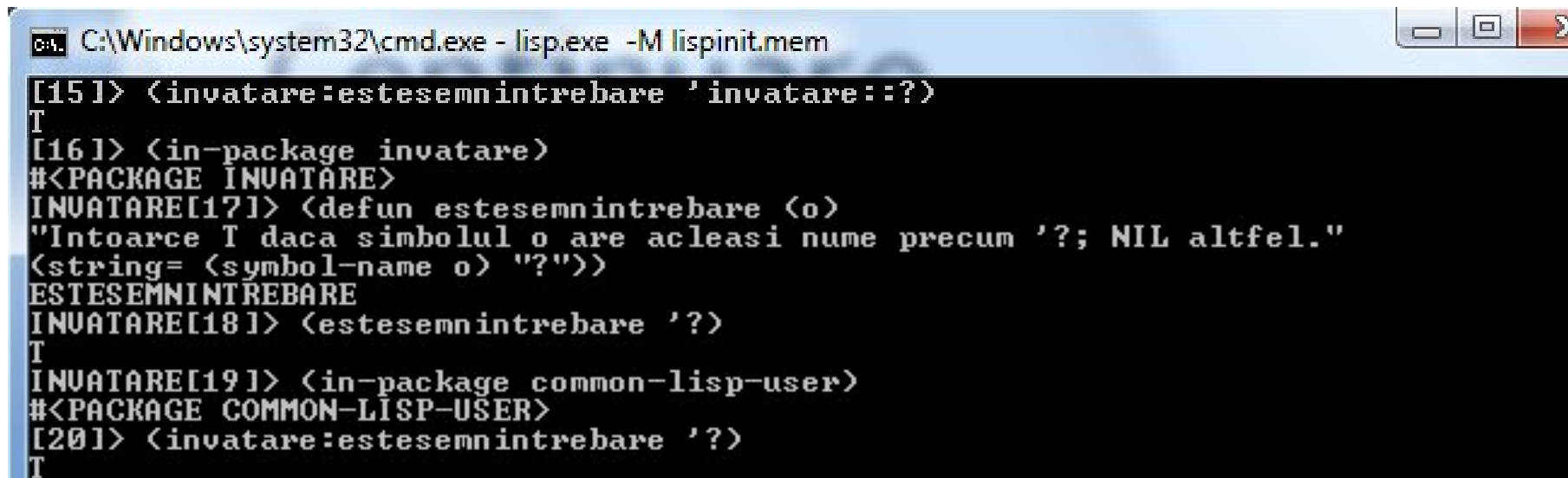


```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
INUATARE[12]> (export 'estesemnintrebare)
T
INUATARE[13]> (in-package common-lisp-user)
#<PACKAGE COMMON-LISP-USER>
[14]> (invatare:estesemnintrebare '?)
NIL
```

- Cand am definit functia, ne aflam in pachetul invatare, in care argumentul sau era invatare::?.
- Cand il testam in pachetul common-lisp-user, argumentul va fi common-lisp-user::?.
- Cele doua simboluri sunt evident diferite.

Reformulare

- Intentionam de fapt nu sa recunoastem simbolul ?, ci sa recunoastem orice simbol al carui nume de afisare este ?.



```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[15]> (invatare:estesemnintrebare 'invatare::?)
T
[16]> (in-package invatare)
#<PACKAGE INVATARE>
INVATARE[17]> (defun estesemnintrebare (o)
  "Intoarce T daca simbolul o are acelasi nume precum '?; NIL altfel."
  (string= (symbol-name o) "?"))
ESTESEMNIINTREBARE
INVATARE[18]> (estesemnintrebare '?)
T
INVATARE[19]> (in-package common-lisp-user)
#<PACKAGE COMMON-LISP-USER>
[20]> (invatare:estesemnintrebare '?)
T
```

Salvarea definitiilor intr-un fisier

- Pentru a putea salva functiile definite pentru o reapelare urmatoare, le vom stoca intr-un fisier cu extensia ***.lisp**.
- Acest fisier il putem crea in Notepad, avand grija ca, in momentul salvarii sa alegem optiunea All Files.
- Fisierul se salveaza in directorul in care avem instalat Lisp-ul.

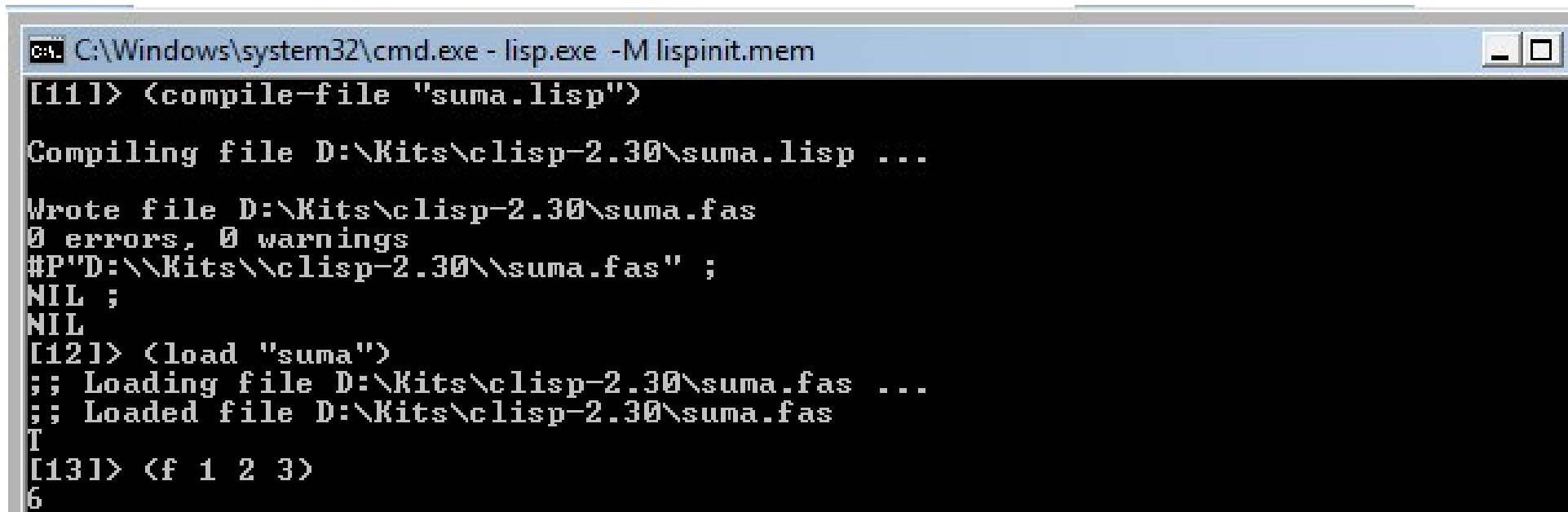
Compilarea si incarcarea definitiiilor

- Pentru a vedea eventualele erori/atentionari, vom compila fisierul rezultat, prin apelarea functiei (`compile-file "nume.lisp"`).
- Pentru incarcarea in memoria Lisp, se foloseste apelarea (`load "nume"`).
- Se apeleaza apoi functia definita in modul clasic de lucru cu Lisp.

Exemplu - Suma a trei numere



```
Lister - [D:\Kits\clisp-2.30\suma.lisp]
File Edit Options Help 100 %
(defun f (o1 o2 o3)
  "Suma a trei numere"
  (+ o1 o2 o3)
)
```



```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[11]> <compile-file "suma.lisp">
Compiling file D:\Kits\clisp-2.30\suma.lisp ...
Wrote file D:\Kits\clisp-2.30\suma.fas
0 errors, 0 warnings
#P"D:\Kits\clisp-2.30\suma.fas" ;
NIL ;
NIL
[12]> <load "suma">
;; Loading file D:\Kits\clisp-2.30\suma.fas ...
;; Loaded file D:\Kits\clisp-2.30\suma.fas
T
[13]> <f 1 2 3>
6
```

Inversarea unei liste de 4 membri

```
Lister - [D:\Kits\clisp-2.30\reverse.lisp]
File Edit Options Help 100 %
(defun inverseaza (lista)
  "Inverseaza o lista de 4 elemente"
  (cons (fourth lista) (cons (third lista) (cons (second lista) (cons (first lista)
    '())))))
)
```

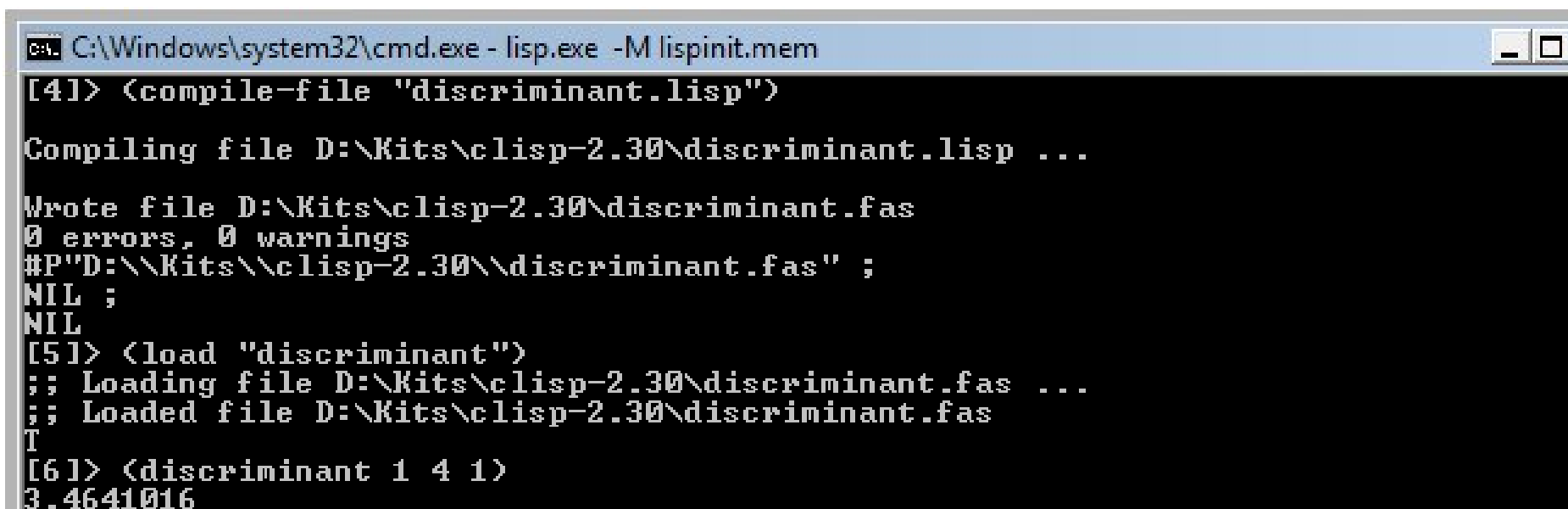
```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[1]> (compile-file "reverse.lisp")
Compiling file D:\Kits\clisp-2.30\reverse.lisp ...
Wrote file D:\Kits\clisp-2.30\reverse.fas
0 errors, 0 warnings
#P"D:\Kits\clisp-2.30\reverse.fas" ;
NIL ;
NIL
[2]> (load "reverse")
;; Loading file D:\Kits\clisp-2.30\reverse.fas ...
;; Loaded file D:\Kits\clisp-2.30\reverse.fas
T
[3]> (inverseaza '(a b c d))
(D C B A)
```

Calculul discriminantului

- Presupunem ca avem o ecuatie de gradul 2 fara radacini complexe.
- Testul complet al tuturor posibilitatilor va constitui o parte a cursului viitor.



```
Lister - [D:\Kits\clisp-2.30\discriminant.lisp]
File Edit Options Help 100 %
(defun discriminant (a b c)
  (sqrt (- (* b b) (* 4 a c))))
)
```



```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[4]> <compile-file "discriminant.lisp">
Compiling file D:\Kits\clisp-2.30\discriminant.lisp ...
Wrote file D:\Kits\clisp-2.30\discriminant.fas
0 errors, 0 warnings
#P"D:\Kits\clisp-2.30\discriminant.fas" ;
NIL ;
NIL
[5]> <load "discriminant">
;; Loading file D:\Kits\clisp-2.30\discriminant.fas ...
;; Loaded file D:\Kits\clisp-2.30\discriminant.fas
T
[6]> <discriminant 1 4 1>
3.4641016
```

Calculul radacinilor ecuatiei de gradul 2

- Presupunem ca ecuatia nu are radacini complexe si ca a este diferit de 0.

```

Lister - [D:\Kits\clisp-2.30\radacini.lisp]
File Edit Options Help 100 %
(load "discriminant")
(defun radacini (a b c)
  (cons (/ (+ (- b) (discriminant a b c)) (* 2 a)) (cons (/ (- (- b) (discriminant
a b c)) (* 2 a)) '()))
)

```

```

C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[7]> <compile-file "radacini.lisp">
Compiling file D:\Kits\clisp-2.30\radacini.lisp ...
Wrote file D:\Kits\clisp-2.30\radacini.fas
0 errors, 0 warnings
#P"D:\Kits\clisp-2.30\radacini.fas" ;
NIL ;
NIL
[8]> <load "radacini">
;; Loading file D:\Kits\clisp-2.30\radacini.fas ...
;; Loading file D:\Kits\clisp-2.30\discriminant.fas ...
;; Loaded file D:\Kits\clisp-2.30\discriminant.fas
;; Loaded file D:\Kits\clisp-2.30\radacini.fas
T
[9]> <radacini 1 -2 1>
(1 1)
[10]> <radacini 1 2 -3>
(1 -3)

```


Funcții predicat

- Acestea sunt funcțiile care întorc fie True (**T**), fie False (**NIL**).
- Pana acum, am intalnit exemple de astfel de functii, `=`, `char=`, `string=`, `eql`, `equal`.
- O multime standard de functii predicat sunt cele care verifica tipul obiectelor Lisp.
- O astfel de functie intoarce T daca tipul obiectului este cel specificat si NIL, altfel.

Exemple

```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[25]> <numberp 5>
T
[26]> <integerp 5>
T
[27]> <floatp 5>
NIL
[28]> <floatp 12.0>
T
[29]> <characterp "a">
NIL
[30]> <stringp "a">
T
[31]> <symbolp '\5>
T
[32]> <packagep (in-package common-lisp-user)>
T
[33]> <listp "a list?">
NIL
[34]> <listp '(a list?)>
T
```

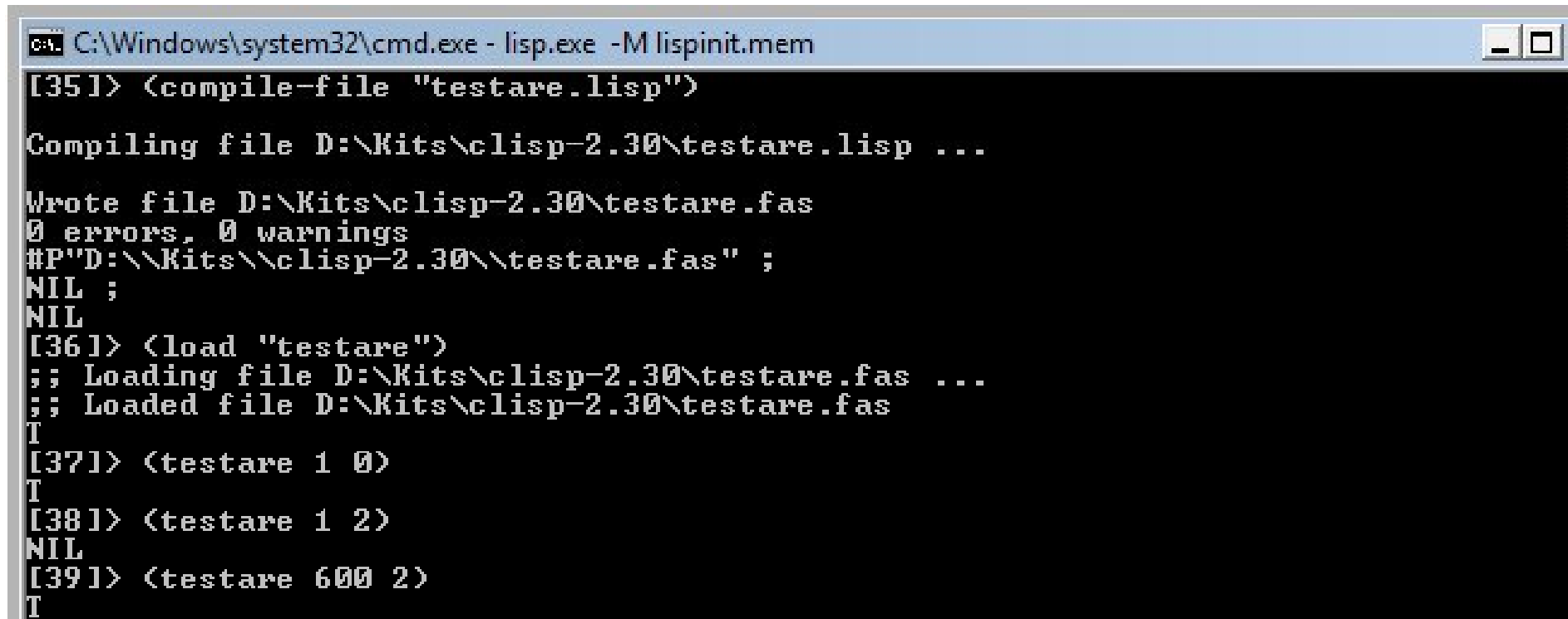
Combinarea functiilor predicat

- Pentru a alatura rezultatele functiilor predicat, Lisp utilizeaza operatorii logici **and** si **or**.
- Acesti operatori lucreaza cu un numar arbitrar de elemente.
- Fiecare se opreste atunci cand intalneste primul rezultat al unui predicat care deja conduce la raspunsul final:
 - Un T in cazul unei disjunctii;
 - Un NIL in cazul unei conjunctii.

Testarea raportului a doua numere



```
Lister - [D:\Kits\clisp-2.30\testare.lisp]
File Edit Options Help 100 %
(defun testare (x y)
(or (= y 0) (> (/ x y) 100))
)
```



```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[35]> <compile-file "testare.lisp">

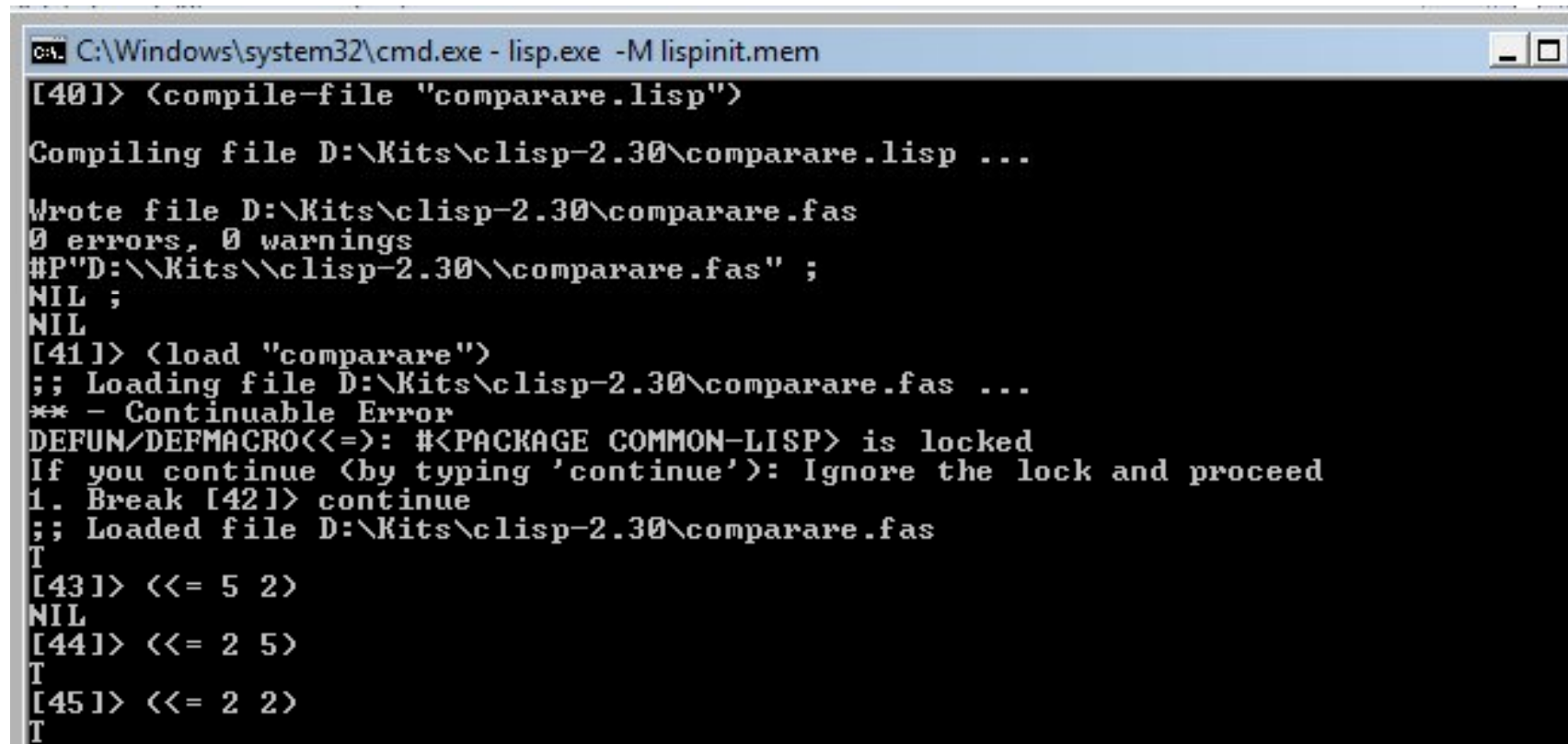
Compiling file D:\Kits\clisp-2.30\testare.lisp ...

Wrote file D:\Kits\clisp-2.30\testare.fas
0 errors, 0 warnings
#P"D:\Kits\clisp-2.30\testare.fas" ;
NIL ;
NIL
[36]> <load "testare">
;; Loading file D:\Kits\clisp-2.30\testare.fas ...
;; Loaded file D:\Kits\clisp-2.30\testare.fas
T
[37]> <testare 1 0>
T
[38]> <testare 1 2>
NIL
[39]> <testare 600 2>
T
```

Compararea a doua numere



```
Listner - [D:\Kits\clisp-2.30\comparare.lisp]
File Edit Options Help 100 %
(defun <= (x y)
  (or (< x y) (= x y))
)
```



```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[40]> <compile-file "comparare.lisp">
Compiling file D:\Kits\clisp-2.30\comparare.lisp ...
Wrote file D:\Kits\clisp-2.30\comparare.fas
0 errors, 0 warnings
#P"D:\Kits\clisp-2.30\comparare.fas" ;
NIL ;
NIL
[41]> <load "comparare">
;; Loading file D:\Kits\clisp-2.30\comparare.fas ...
** - Continuable Error
DEFUN/DEPMACRO(<=>): #<PACKAGE COMMON-LISP> is locked
If you continue (by typing 'continue'): Ignore the lock and proceed
1. Break [42]> continue
;; Loaded file D:\Kits\clisp-2.30\comparare.fas
T
[43]> <<= 5 2>
NIL
[44]> <<= 2 5>
T
[45]> <<= 2 2>
T
```

Lungimea unui string / a unei liste

```
Lister - [D:\Kits\clisp-2.30\lungime.lisp]
File Edit Options Help 100 %
(defun len (x)
  (and (or (stringp x) (listp x)) (> (length x) 5))
)
```

```
C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[46]> <compile-file "lungime.lisp">
Compiling file D:\Kits\clisp-2.30\lungime.lisp ...
Wrote file D:\Kits\clisp-2.30\lungime.fas
0 errors, 0 warnings
#P"D:\Kits\clisp-2.30\lungime.fas" ;
NIL ;
NIL
[47]> <load "lungime">
;; Loading file D:\Kits\clisp-2.30\lungime.fas ...
;; Loaded file D:\Kits\clisp-2.30\lungime.fas
T
[48]> <len "ceva">
NIL
[49]> <len "altceva">
T
[50]> <len '(1 2 3)>
NIL
[51]> <len '(a b c d e f)>
T
```

Pe saptamana viitoare...

