Fundamentele limbajului LISP (2)

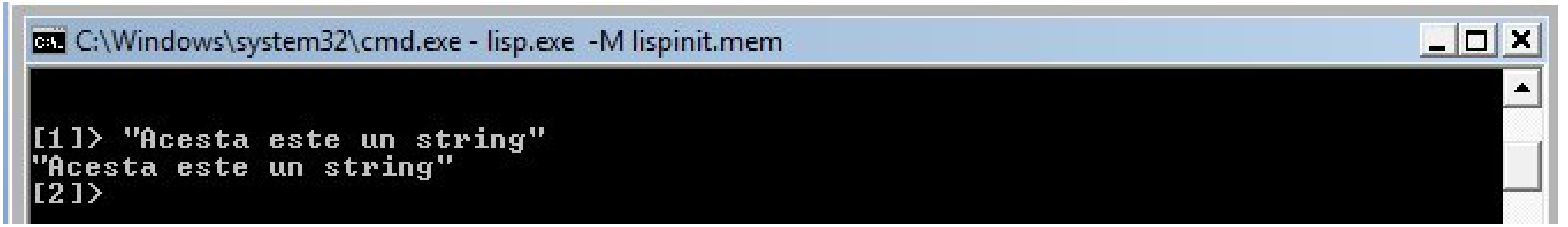
Ruxandra Stoean
http://inf.ucv.ro/~rstoean
ruxandra.stoean@inf.ucv.ro

Stringuri si caractere

- Un string este un vector de caractere.
- Este scris de catre Lisp ca secventa caracterelor sale inconjurata de ghilimele.
- Ca si numerele, stringurile sunt evaluate in ele insele.
- Sa scriem un string la prompterul de Lisp.

> “Acesta este un string.”

Exemplu


C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[1]> "Acesta este un string"
"Acesta este un string"
[2]>

Stringuri

- Un string poate fi oricat de lung si poate contine caractere precum ENTER.
- In afara numerelor, Common Lisp are de asemenea functii care opereaza si asupra obiectelor de alt tip.
- De exemplu, pentru a afla numarul de caractere dintr-un string, se foloseste functia length.

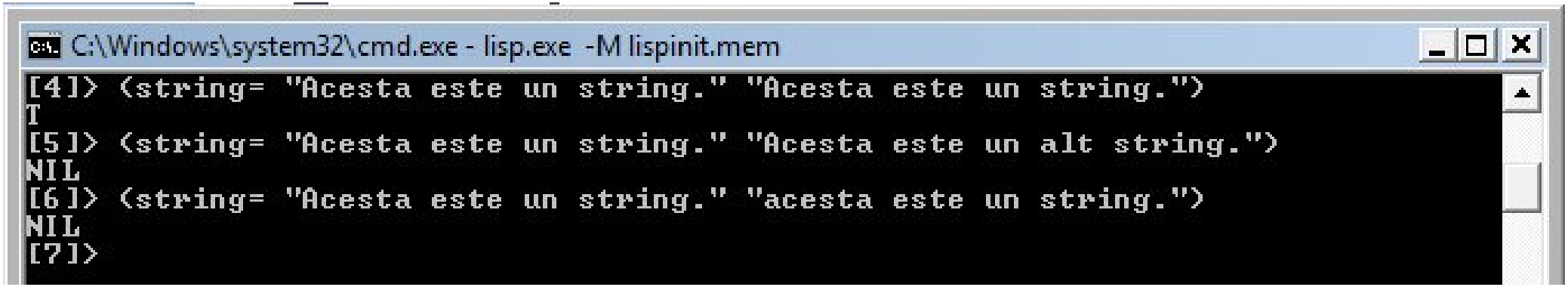
Exemplu


C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[3]> (length "Acesta este un string.")
22
[4]> _

Stringuri

- O alta functie predefinita este string=.
- Aceasta functie intoarce TRUE daca cele doua stringuri date ca argumente sunt alcatuite din aceleasi caractere si FALSE, in caz contrar.

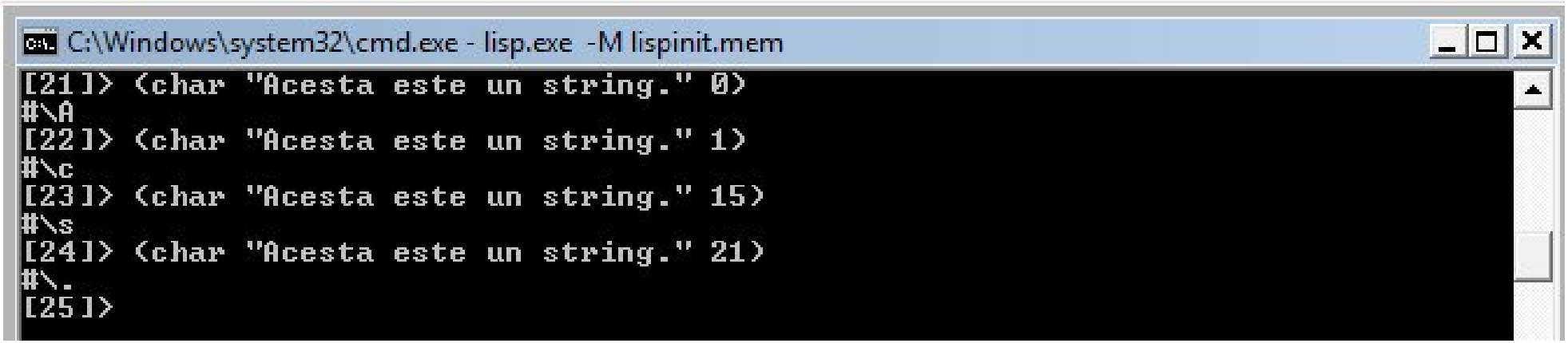
Exemple


C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[4]> (string= "Acesta este un string." "Acesta este un string.")
T
[5]> (string= "Acesta este un string." "Acesta este un alt string.")
NIL
[6]> (string= "Acesta este un string." "acesta este un string.")
NIL
[7]>

Stringuri

- Pentru a accesa un anumit caracter in string, se utilizeaza formularea (char *string index*).
- char este predefinit, string reprezinta sirul dorit iar index pozitia caracterului care va fi intors.
- Index-ul primului caracter din string e 0.
- Index-ul nu trebuie sa depaseasca lungimea sirului.

Exemple


C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[21]> (char "Acesta este un string." 0)
#\A
[22]> (char "Acesta este un string." 1)
#\c
[23]> (char "Acesta este un string." 15)
#\s
[24]> (char "Acesta este un string." 21)
#\.
[25]>

Caractere

- Se poate observa ca un caracter este scris de catre Lisp cu prefixul #\.
- Tot in acelasi mod va da si utilizatorul caracterele.
- Un caracter este evaluat in el insusi.

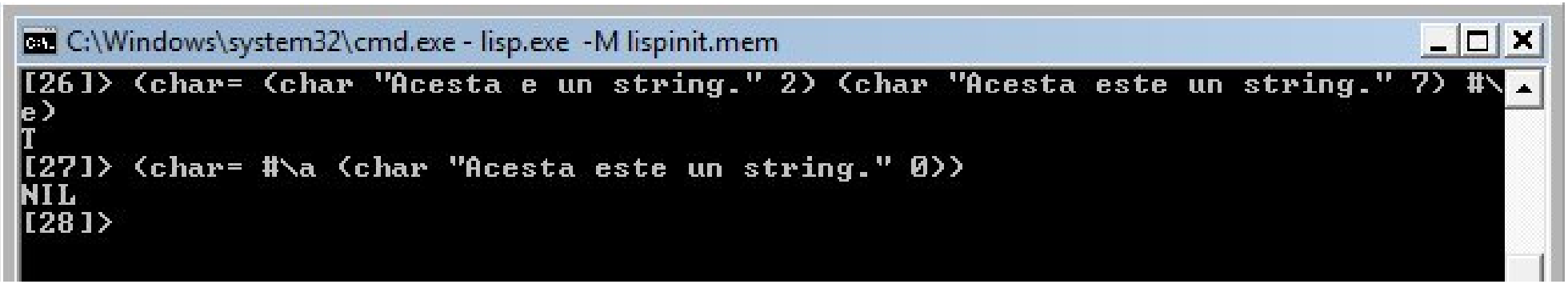
Exemple

```
[2]> #\r
#\r
[3]> #\T
#\T
[4]> #\.
#\.
```

Caractere

- Pentru testarea faptului ca doua caractere sunt identice, se foloseste functia char=.
- La fel ca la testarea egalitatii pentru numere, dar diferit de aceeasi testare pentru stringuri, aceasta functie poate lua orice numar de argumente.

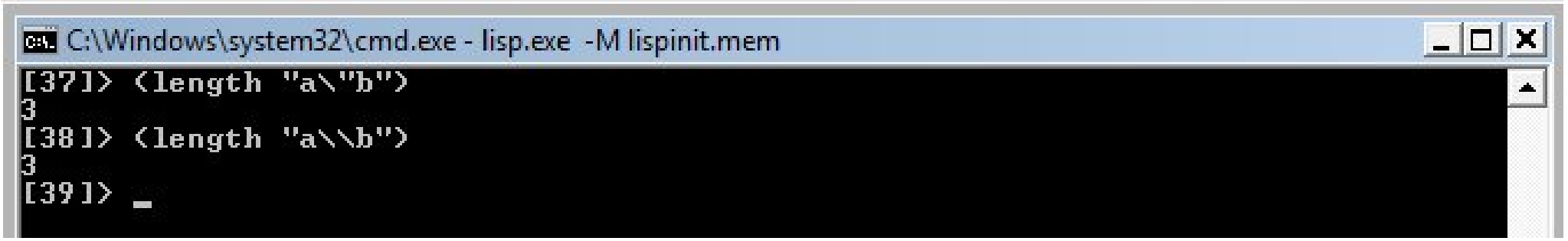
Exemple


C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[26]> (char= (char "Acesta e un string." 2) (char "Acesta este un string." 7) #\e)
T
[27]> (char= #\a (char "Acesta este un string." 0))
NIL
[28]>

Stringuri si caractere

- Pentru a utiliza simbolul “ ca parte a unui string, va trebui sa folosim caracterul \.
- Pentru a utiliza caracterul \ apoi ca parte a unui string, trebuie sa mai adaugam inca unul in fata sa.

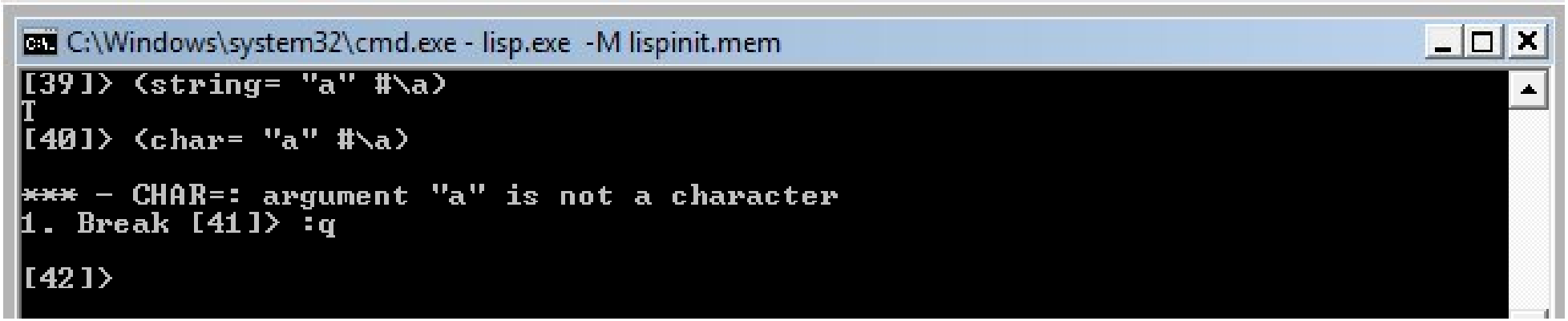
Exemple



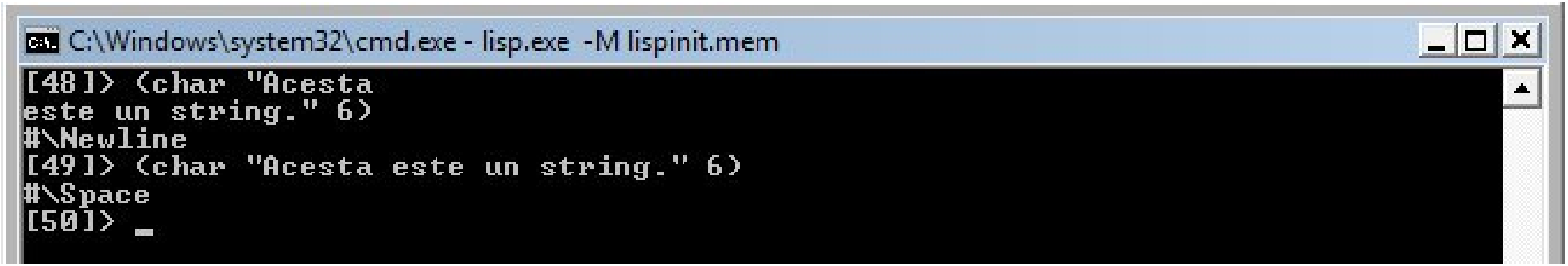
- Observati ca lungimea stringurilor nu este influentata de caracterul \.

Exemple

- Pentru testarea egalitatii a doua stringuri, a doua poate fi si caracter.
- Pentru acelasi lucru in cazul caracterelor, amandoua trebuie sa fie de acest fel.



Caracterele spatiu si ENTER



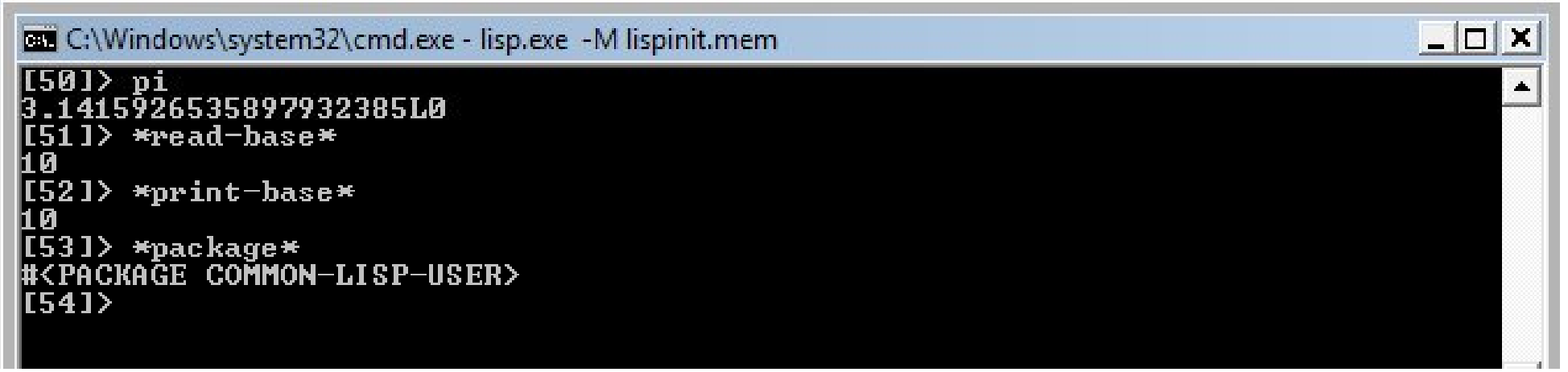
Simboluri

- Simbolurile sunt un alt tip de data in Lisp.
- Pentru a reprezenta un simbol, se folosesc secvente de litere si caracterele * si -:
  - De exemplu: paul, pi, *read-base*
- Un simbol poate reprezenta ceva pentru care dorim sa stocam informatii:
  - De exemplu, paul poate reprezenta o persoana.

Simboluri

- Simbolurile sunt de asemenea folosite drept variabile.
- Astfel, un simbol poate avea o valoare: se spune ca este legat, sau este, dimpotriva, nelegat (fara valoare).
- Unele simboluri legate sunt: pi, *read-base*, *print-base* si *package*.

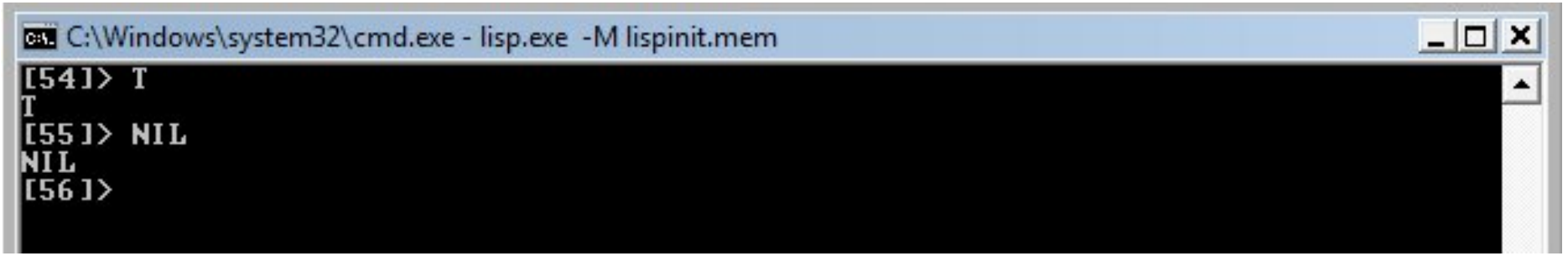
Exemple



- pi reprezinta valoarea simbolului matematic.
- *read-base* si *print-base* specifica in ce baza vor fi scrise numerele de catre utilizator, respectiv de Lisp.
- *package* specifica pachetul in care ne aflam curent.

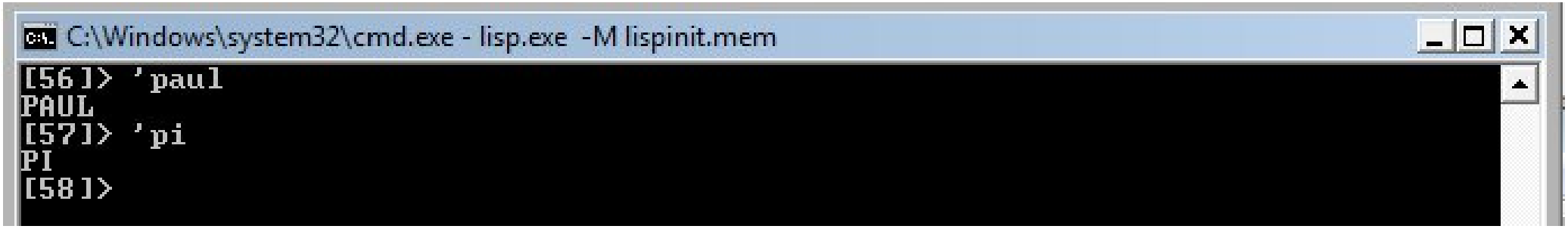
Simboluri

- Cele mai importante simboluri in Lisp sunt T si NIL.
- T reprezinta true, iar NIL desemneaza false si lista vida.
- Aceste simboluri sunt legate chiar la ele insele.



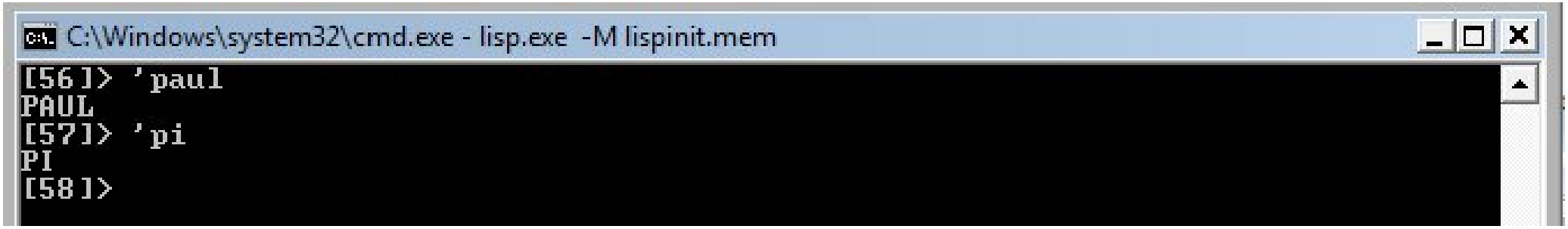
Simboluri

- Atunci cand vrem sa utilizam un simbol si nu valoarea sa, punem ‘ in fata acestuia.



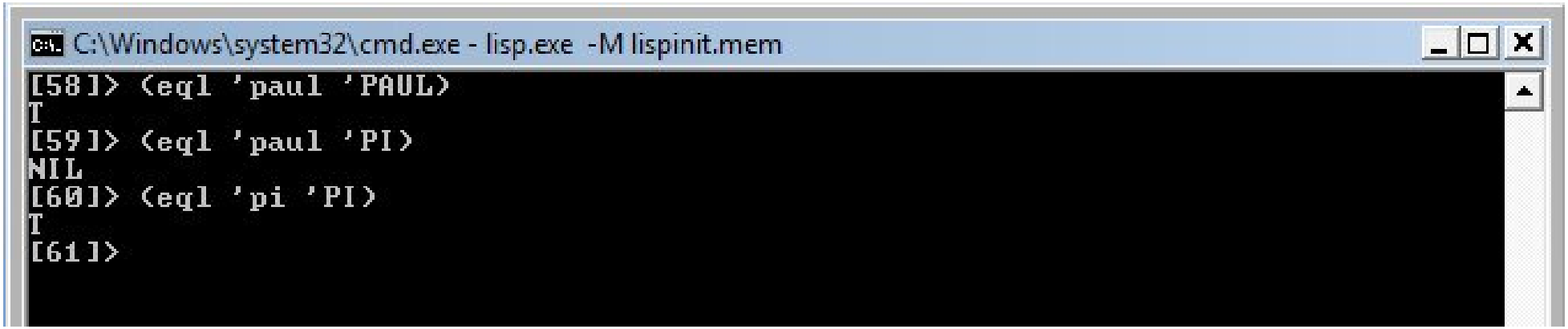
Simboluri

- Putem scrie simbolurile cu litere mici, Lisp le converteste la litere mari.



Simboluri

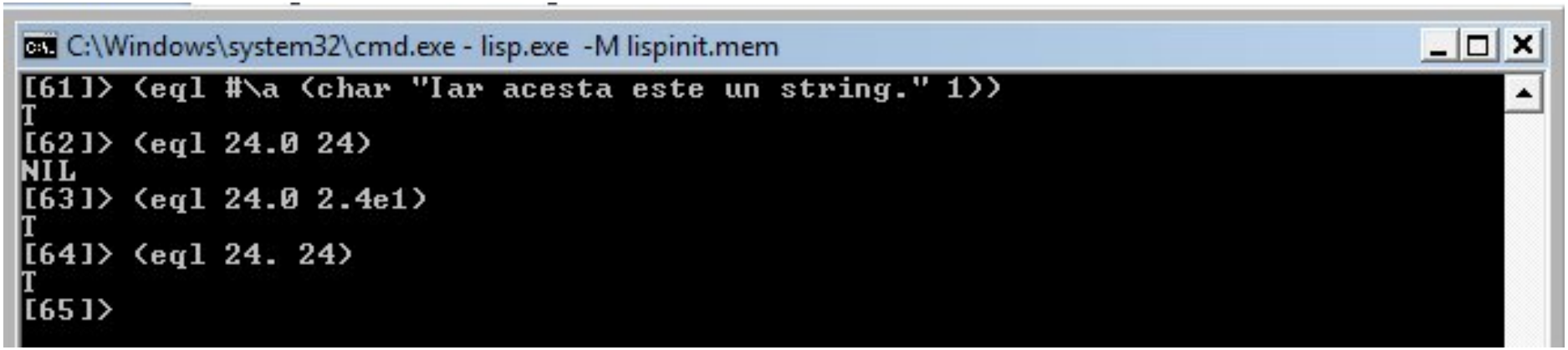
- Pentru a testa egalitatea dintre doua simboluri, se foloseste functia eql.



Functia eql

- Aceasta functie e mai generala chiar, testand daca sunt identice oricare doua obiecte Lisp:
  - Simboluri
  - Caractere
  - Numere de acelasi tip

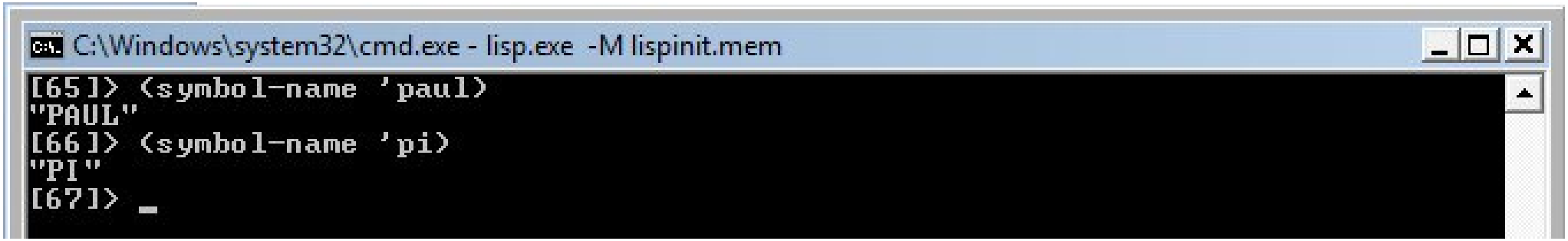
Exemple


C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[61]> (eql #\a (char "Iar acesta este un string." 1))
T
[62]> (eql 24.0 24)
NIL
[63]> (eql 24.0 2.4e1)
T
[64]> (eql 24. 24)
T
[65]>

Simboluri

- Orice simbol are un nume reprezentat de un string.
- Putem afla acest nume utilizand functia symbol-name.

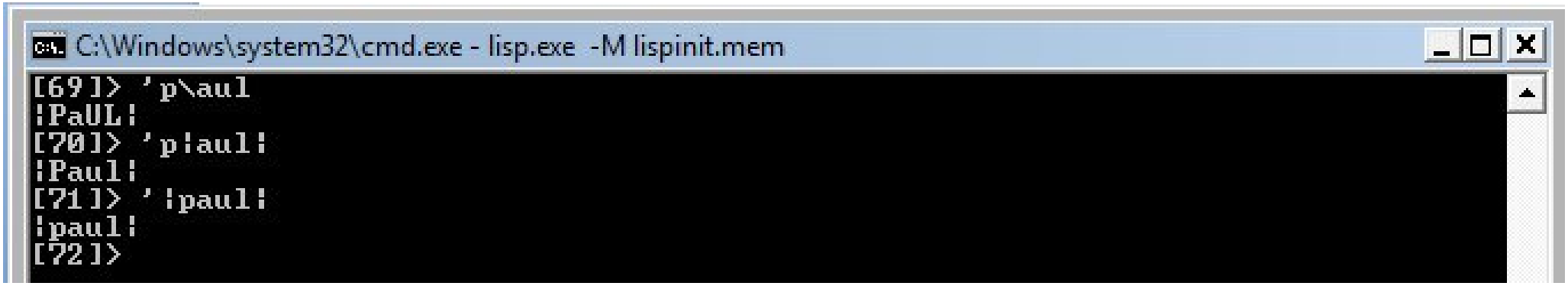
Exemplu


C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[65]> (symbol-name 'paul)
"PAUL"
[66]> (symbol-name 'pi)
"PI"
[67]> _

Simboluri

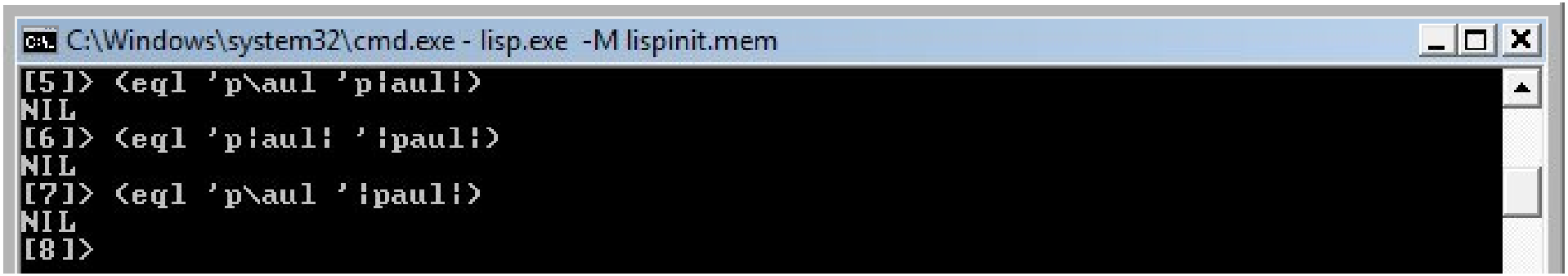
- Daca se doreste ca un caracter sa ramana scris cu litera mica in cadrul unui simbol, se va folosi \.
- Daca vrem ca Lisp sa pastreze literele exact cum le dam, le vom scrie intre ||.

Exemplu



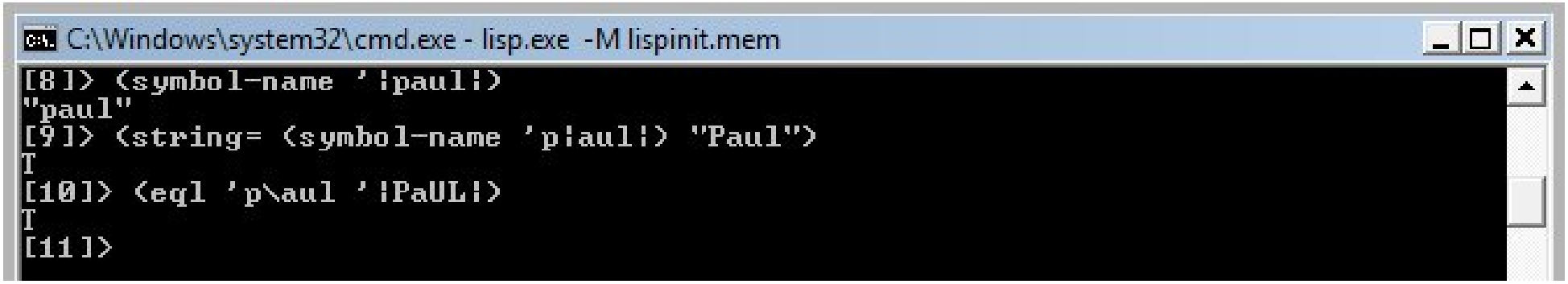
- Pentru a scrie un simbol, Lisp foloseste si ||.
- Acestea nu fac parte din simbol sau din numele sau.

Exemplu



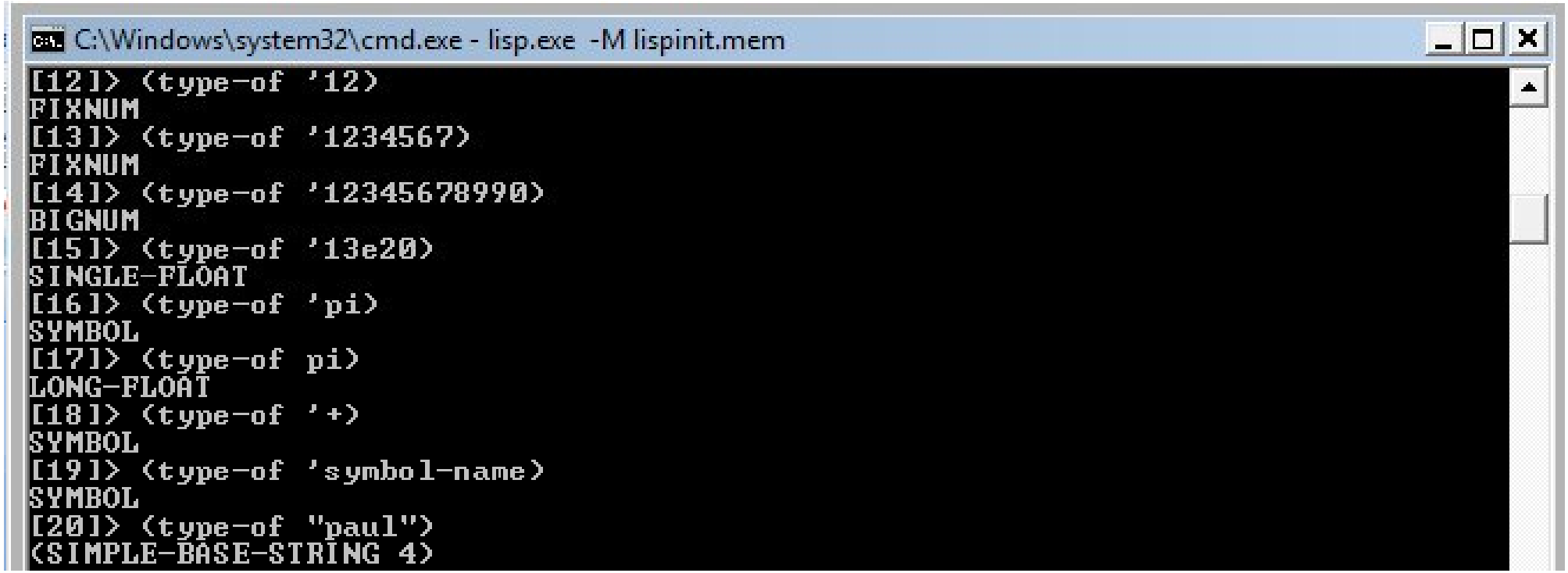
- Simbolurile cu litere diferite ca marime sunt la randul lor diferite.

Mai multe exemple



Tipul unui obiect

- Pentru a afla care este tipul unui anume obiect, se foloseste functia type-of.



Mai multe exemple

```
[27]> (type-of '#\t)
BASE-CHAR
[28]> (type-of *package*)
PACKAGE
[29]> (type-of '10/3)
RATIO
[30]> (type-of '(1 2 3))
CONS
[31]>
```

Pachete

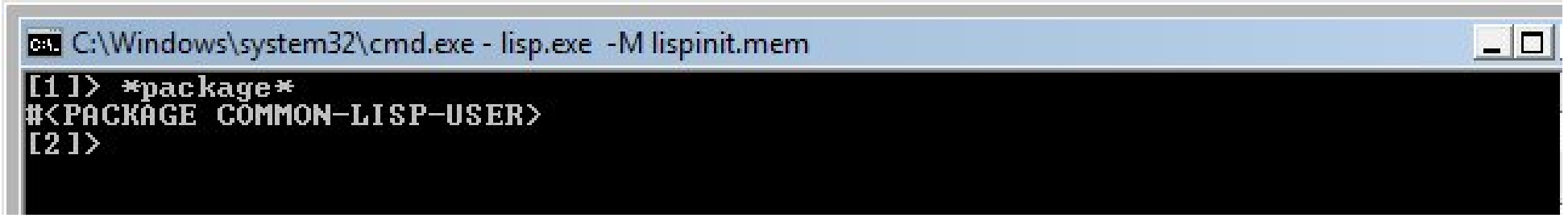
- Fiecare multime de simboluri este pastrata intr-un pachet Common Lisp.
- Utilizatorul isi poate crea propriul pachet si il poate exporta pentru ca altii sa il poata utiliza.
- Un pachet poate fi evident importat in alt pachet.

Pachete

- Am vazut ca un simbol poate avea diverse reprezentari si totusi sa ramana acelasi simbol, cu acelasi nume.
- In continuare vom vedea ca simboluri diferite pot avea acelasi nume daca sunt in pachete diferite.

Pachete

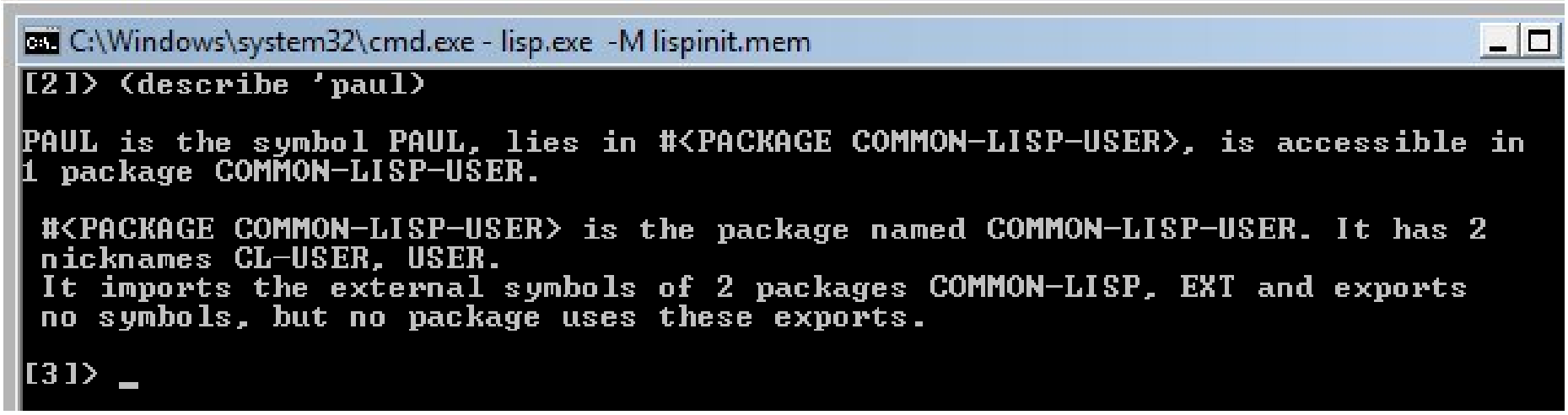
- Atunci cand interactionam cu Lisp, ne aflam deja intr-un pachet.
- Putem vedea pachetul curent verificand valoarea simbolului *package*.



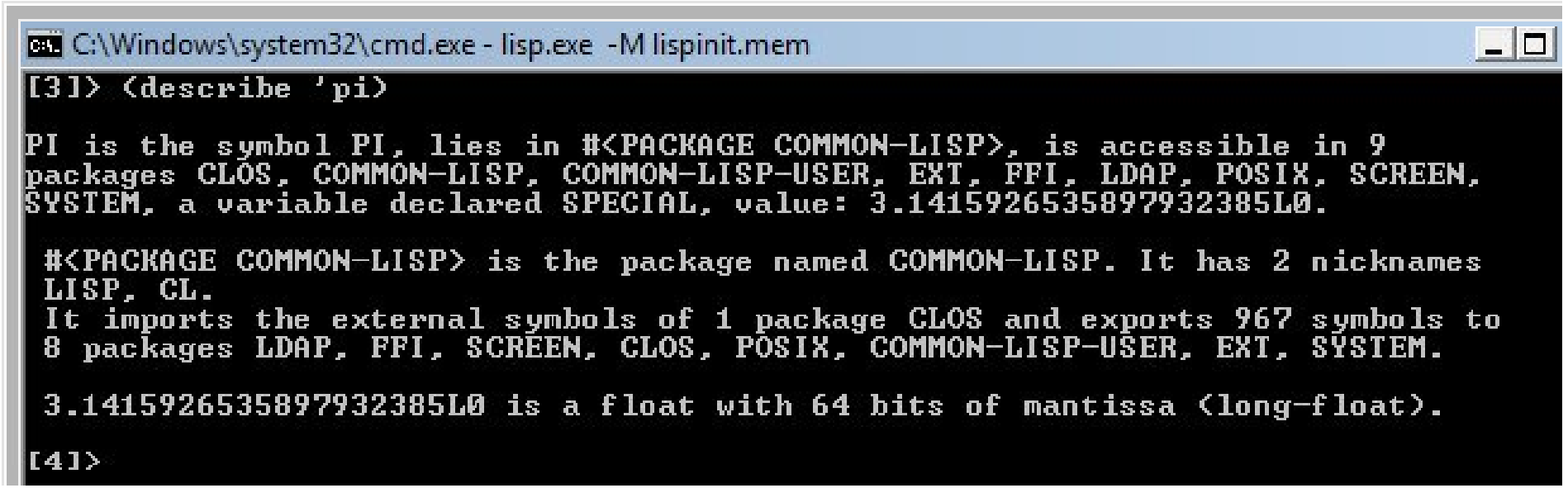
Functia describe

- Prin apelul acestei functii Lisp putem afla diverse proprietati despre obiecte.

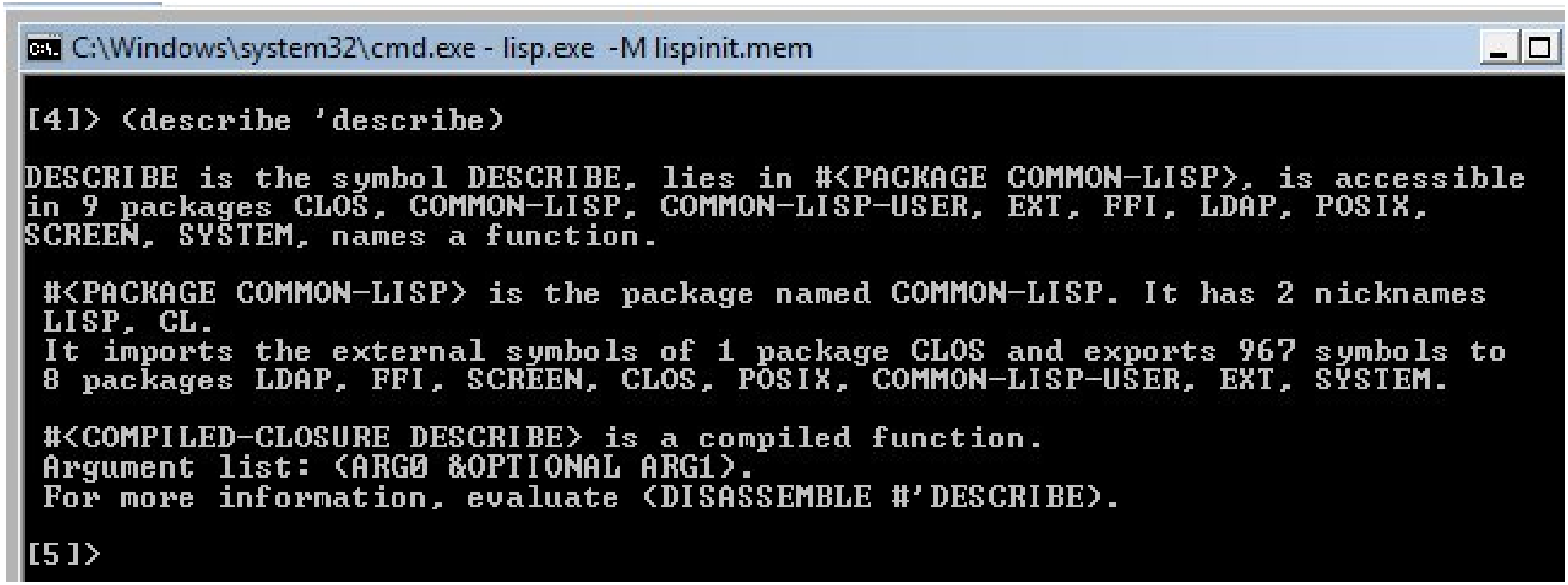
- Printre altele, putem vedea pachetul din care fac parte diferite simboluri.



Exemple


C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[3]> (describe 'pi)
PI is the symbol PI, lies in #<PACKAGE COMMON-LISP>, is accessible in 9
packages CLOS, COMMON-LISP, COMMON-LISP-USER, EXT, FFI, LDAP, POSIX, SCREEN,
SYSTEM, a variable declared SPECIAL, value: 3.14159265358979323851.
#<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
8 packages LDAP, FFI, SCREEN, CLOS, POSIX, COMMON-LISP-USER, EXT, SYSTEM.
3.14159265358979323851 is a float with 64 bits of mantissa (long-float).
[4]>

Exemple


C:\Windows\system32\cmd.exe - lisp.exe -M lispinit.mem
[4]> (describe 'describe)
DESCRIBE is the symbol DESCRIBE, lies in #<PACKAGE COMMON-LISP>, is accessible
in 9 packages CLOS, COMMON-LISP, COMMON-LISP-USER, EXT, FFI, 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
8 packages LDAP, FFI, SCREEN, CLOS, POSIX, COMMON-LISP-USER, EXT, SYSTEM.
#<COMPILED-CLOSURE DESCRIBE> is a compiled function.
Argument list: (ARG0 &OPTIONAL ARG1).
For more information, evaluate (DISASSEMBLE #'DESCRIBE).
[5]>

Pachete

- Ne putem muta in alt pachet apeland functia in-package.
- Acolo putem referi simboluri existente sau unele noi create de utilizator.

Exemplu

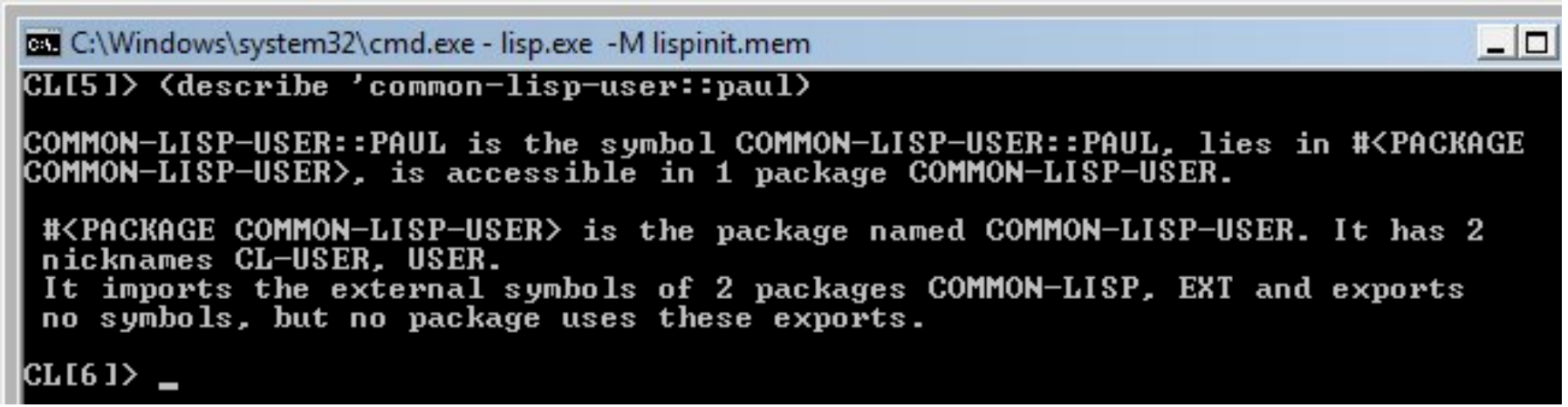
```
[1]> (in-package common-lisp)
#<PACKAGE COMMON-LISP>
CL[2]>
(describe 'pi)

PI is the symbol PI, lies in #<PACKAGE COMMON-LISP>, is accessible in 9
packages CLOS, COMMON-LISP, COMMON-LISP-USER, EXT, FFI, LDAP, POSIX, SCREEN,
SYSTEM, a variable declared SPECIAL, value: 3.14159265358979323851L0.

 #<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
 8 packages LDAP, FFI, SCREEN, CLOS, POSIX, COMMON-LISP-USER, EXT, SYSTEM.

 3.14159265358979323851L0 is a float with 64 bits of mantissa (long-float).

CL[3]> (describe 'paul)

** - Continuable Error
INTERN("PAUL"): #<PACKAGE COMMON-LISP> is locked
If you continue (by typing 'continue'): Ignore the lock and proceed
1. Break CL[4]> continue

PAUL is the symbol PAUL, lies in #<PACKAGE COMMON-LISP>, is accessible in 1
package COMMON-LISP.

 #<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
 8 packages LDAP, FFI, SCREEN, CLOS, POSIX, COMMON-LISP-USER, EXT, SYSTEM.

CL[5]>
```

Pachete

- Pentru a referi un acelasi simbol din alt pachet, folosim exprimarea:

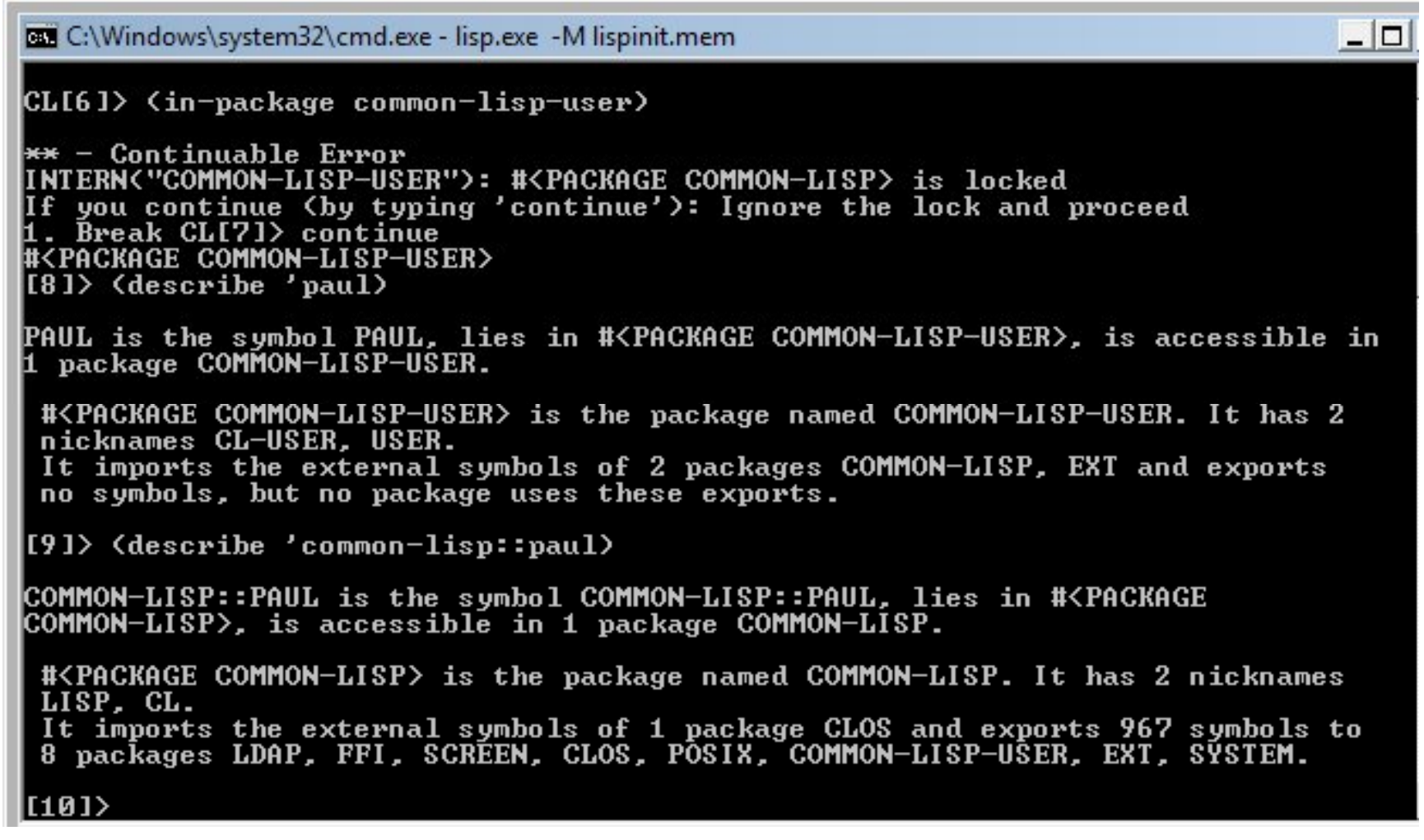
  nume_pachet::nume_simbol

- Cele doua simboluri sunt diferite.



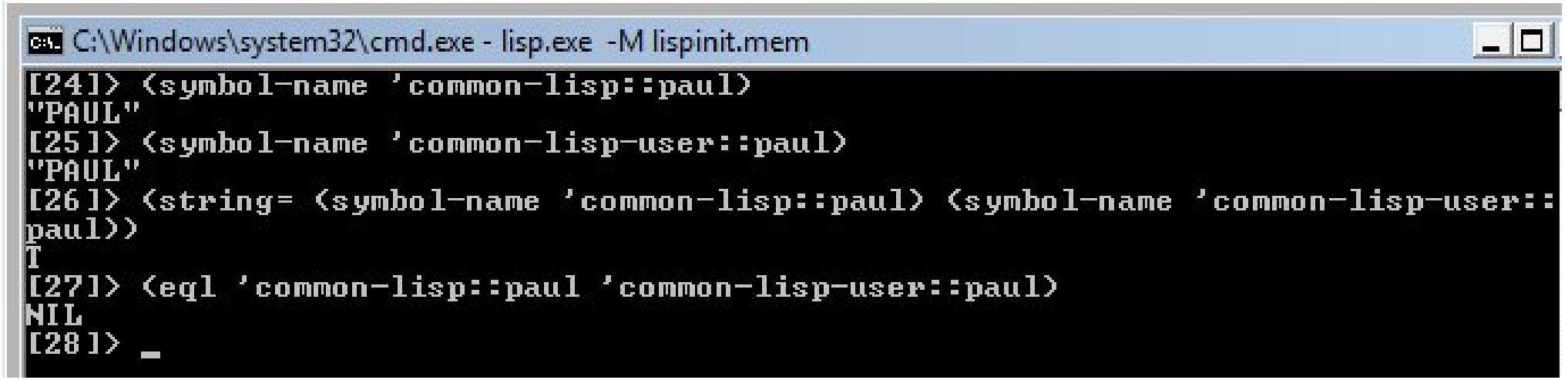
Pachete

- Sa ne intoarcem acum la pachetul common-lisp-user si sa aflam informatii despre simbolul ‘paul.



Pachete

- Cele doua simboluri din pachete diferite nu sunt identice decat ca nume.



Pachete

- Un simbol poate fi exportat din pachetul sau prin apelarea functiei export.

- Numele unui simbol extern este de forma:

nume_pachet:nume_simbol

Exemplu

```
[31]> (export 'paul)
T
[32]> (in-package common-lisp)
#<PACKAGE COMMON-LISP>
CL[33]> (describe 'common-lisp-user:paul)

COMMON-LISP-USER:PAUL is the symbol COMMON-LISP-USER:PAUL, lies in #<PACKAGE
COMMON-LISP-USER>, is accessible in 1 package COMMON-LISP-USER.

 #<PACKAGE COMMON-LISP-USER> is the package named COMMON-LISP-USER. It has 2
 nicknames CL-USER, USER.
 It imports the external symbols of 2 packages COMMON-LISP, EXT and exports 1
 symbol, but no package uses these exports.

CL[34]>
```

Pachete

- Pentru a afla daca un simbol a fost exportat sau este inca intern intr-un anumit pachet, se poate proceda precum in cele ce urmeaza.

```
CL[34]> 'common-lisp-user::paul
COMMON-LISP-USER:PAUL
CL[35]> (in-package common-lisp-user)
#<PACKAGE COMMON-LISP-USER>
[36]> 'common-lisp::paul
COMMON-LISP::PAUL
[37]> _
```

Pachete

- Putem de asemenea defini pachete noi.

```
[57]> (defpackage test)
#<PACKAGE TEST>
[58]> (in-package test)
#<PACKAGE TEST>
TEST[59]> 'common-lisp-user::paul
COMMON-LISP-USER:PAUL
TEST[60]> 'common-lisp::paul
COMMON-LISP::PAUL
TEST[61]> (describe 'common-lisp::paul)

COMMON-LISP::PAUL is the symbol COMMON-LISP::PAUL, lies in #<PACKAGE
COMMON-LISP>, is accessible in 1 package COMMON-LISP.

 #<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 TEST, LDAP, FFI, SCREEN, CLOS, POSIX, COMMON-LISP-USER, EXT,
 SYSTEM.

TEST[62]> (describe 'common-lisp-user::paul)

COMMON-LISP-USER:PAUL is the symbol COMMON-LISP-USER:PAUL, lies in #<PACKAGE
COMMON-LISP-USER>, is accessible in 1 package COMMON-LISP-USER.

 #<PACKAGE COMMON-LISP-USER> is the package named COMMON-LISP-USER. It has 2
 nicknames CL-USER, USER.
 It imports the external symbols of 2 packages COMMON-LISP, EXT and exports 1
 symbol, but no package uses these exports.
```

Pachete

- Daca dorim sa importam un simbol extern dintr-un pachet in altul folosim functia import.

```
TEST[67]> (import 'common-lisp-user:paul)
T
TEST[68]> (describe 'paul)

PAUL is the symbol PAUL, lies in #<PACKAGE COMMON-LISP-USER>, is accessible in
2 packages COMMON-LISP-USER, TEST.

 #<PACKAGE COMMON-LISP-USER> is the package named COMMON-LISP-USER. It has 2
 nicknames CL-USER, USER.
 It imports the external symbols of 2 packages COMMON-LISP, EXT and exports 1
 symbol, but no package uses these exports.

TEST[69]> (eql 'paul 'common-lisp-user::paul)
T
TEST[70]> (eql 'paul 'common-lisp::paul)
NIL
TEST[71]> 'common-lisp-user::paul
PAUL
TEST[72]> _
```

Alte observatii

- Atunci cand referim prima data un simbol, Lisp il si construieste; deci, un simbol nou nu trebuie declarat inainte.
- Daca vom incerca sa suprascriem un simbol care deja exista intr-un pachet, vom primi mesaj de eroare.

Exemplu

```
TEST[72]> (in-package common-lisp-user)
#<PACKAGE COMMON-LISP-USER>
[73]> (describe 'ion)

ION is the symbol ION, lies in #<PACKAGE COMMON-LISP-USER>, is accessible in 1
package COMMON-LISP-USER.

 #<PACKAGE COMMON-LISP-USER> is the package named COMMON-LISP-USER. It has 2
 nicknames CL-USER, USER.
 It imports the external symbols of 2 packages COMMON-LISP, EXT and exports 1
 symbol, but no package uses these exports.

[74]> (export 'ion)
T
[75]> (in-package test)
#<PACKAGE TEST>
TEST[76]> (describe 'ion)

ION is the symbol ION, lies in #<PACKAGE TEST>, is accessible in 1 package
TEST.

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

TEST[77]> (import 'common-lisp-user:ion)

** - Continuable Error
importing COMMON-LISP-USER:ION into #<PACKAGE TEST> produces a name conflict wit
h ION.
If you continue (by typing 'continue'): You may choose how to proceed.
1. Break TEST[78]>
```

Alte observatii

- Simbolurile standard din pachetul lisp sunt externe si importate automat in alte pachete.

```
TEST[79]> 'common-lisp::pi
PI
TEST[80]> 'common-lisp::describe
DESCRIBE
TEST[81]>
```

Pachetele ca tip de data

- Ca tip de data in Lisp, putem afla mai multe informatii despre pachetul curent.

```
TEST[81]> *package*
#<PACKAGE TEST>
TEST[82]> (type-of *package*)
PACKAGE
TEST[83]> (package-name *package*)
"TEST"
TEST[84]> (string= (package-name *package*) (symbol-name 'test))
T
TEST[85]> (eql *package* 'test)
NIL
```

Pachetele ca tip de data

- Pe langa functiile deja cunoscute, find-package ne spune pachetul al carui nume prescurtat il stim.

```
TEST[81]> *package*
#<PACKAGE TEST>
TEST[82]> (type-of *package*)
PACKAGE
TEST[83]> (package-name *package*)
"TEST"
TEST[84]> (string= (package-name *package*) (symbol-name 'test))
T
TEST[85]> (eql *package* 'test)
NIL
TEST[86]> (find-package "TEST")
#<PACKAGE TEST>
TEST[87]> (find-package "LISP")
#<PACKAGE COMMON-LISP>
```

Procesarea de baza a listelor

- Pana acum am discutat despre evaluarea S-expresiilor care erau date sub forma de liste.
- In continuare, vom discuta despre liste ca tip de baza in Lisp.
- Pentru a crea o lista, se foloseste functia de baza: (cons *obiect lista*), unde:
  - primul argument poate fi orice obiect Lisp
  - al doilea este o lista
  - intoarce o lista cu primul argument inserat ca prim membru si restul listei fiind al doilea argument

Exemplu

```
[1]> (cons 'a '(b c))
(A B C)
[2]> (cons 2 (cons 4 (cons 6 '(8))))
(2 4 6 8)
[3]> (cons 'c '())
(C)
[4]> (cons 'b (cons 'c '()))
(B C)
[5]> (cons 'a (cons 'b (cons 'c '())))
(A B C)
```

Primul element si restul listei

- Pentru a accesa primul element al unei liste si lista ramasa, se folosesc predicatele

(first *list*) si (rest *list*)

```
[6]> (first '(1 2 3))
1
[7]> (rest '(1 2 3))
(2 3)
[8]> (first (cons 'a '(b c)))
A
[9]> (rest (cons 'a '(b c)))
(B C)
[10]> (first '())
NIL
[11]> (rest '())
NIL
```

Mai multe exemple

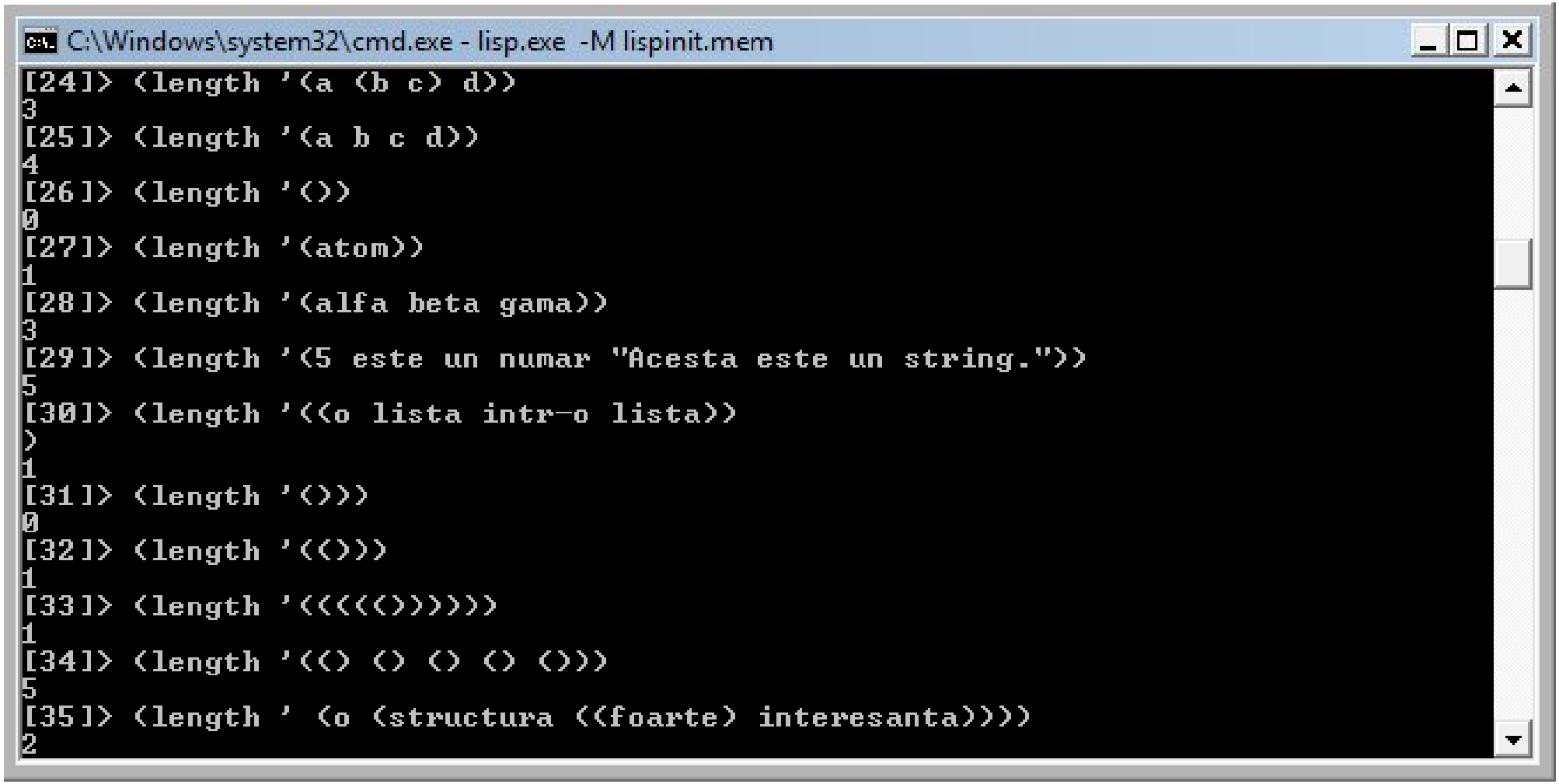
```
[12]> (first (rest '(1 2 3)))
2
[13]> (cons (first (rest '(1 2 3))) (rest (rest '(1 2 3))))
(2 3)
[14]> (first '(((((()))))))
(((NIL)))
[15]> (first (first '((A B) C)))
A
[16]> (cons '() '(A B C))
(NIL A B C)
[17]> (cons '(a b c) '())
((A B C))
[18]> (rest '(a))
NIL
[19]> (cons nil nil)
(NIL)
```

Procesarea listelor

- Functia equal spune daca elementele a doua liste sunt egale doua cate doua sau nu.

```
[20]> (equal '(a (b c) d) '(a (b c) d))
T
[21]> (equal '(a (b c) d) '(a b c d))
NIL
[22]> (equal '(a) '((a)))
NIL
[23]> (equal '(a) (first '((a))))
T
```

Functia de determinare a lungimii unei liste



Alte observatii

- Dupa ce am tastat o forma in Lisp, o putem imediat reapela cu *; cu ** putem reapela penultima forma.

```
[36]> '(a b c)
(A B C)
[37]> (first *)
A
[38]> **
(A B C)
[39]> (first (rest *))
B
[40]> **
(A B C)
[41]> (first (rest (rest *)))
C
[42]> (first **)
A
```

Pe saptamana viitoare…