CSC 416

Programming Challenge 1

Jacob Peck

A set of small challenges designed to familiarize one with the basics of LISP. Includes three "sample sessions": one on LISP fundamentals, one on LISP numerics, and one dealing with basic LISP list processing.

Listing of `opening_session.text`:

```
$ clisp

{…snip…}

[1]> 15
15
[2]> "Common Lisp with Objects"
"Common Lisp with Objects"
[3]> pie

*** - SYSTEM::READ-EVAL-PRINT: variable PIE has no value
The following restarts are available:
USE-VALUE      :R1      Input a value to be used instead of PIE.
STORE-VALUE    :R2      Input a new value for PIE.
ABORT          :R3      Abort main loop
Break 1 [4]> :a
[5]> pi
3.14159265358979323 85L0
[6]> (+ 2 3 5 7)
17
[7]> (* (+ 3 6 9) (- 8 5))
54
[8]> (3 + 2)

*** - EVAL: 3 is not a function name; try using a symbol instead
The following restarts are available:
USE-VALUE      :R1      Input a value to be used instead.
ABORT          :R2      Abort main loop
Break 1 [9]> :a
[10]> (double 5)

*** - EVAL: undefined function DOUBLE
The following restarts are available:
USE-VALUE      :R1      Input a value to be used instead of (FDEFINITION
'DOUBLE).
RETRY          :R2      Retry
STORE-VALUE    :R3      Input a new value for (FDEFINITION 'DOUBLE).
ABORT          :R4      Abort main loop
Break 1 [11]> :a
[12]> (quote pie)
PIE
[13]> (quote (double 5))
(DOUBLE 5)
[14]> 'pie
PIE
[15]> ''pie
'PIE
[16]> '(double 5)
(DOUBLE 5)
[17]> (setf pie 'cherry)
CHERRY
[18]> pie
CHERRY
```

```
[19]> (setf pie apple)

*** - SETQ: variable APPLE has no value
The following restarts are available:
USE-VALUE      :R1      Input a value to be used instead of APPLE.
STORE-VALUE    :R2      Input a new value for APPLE.
ABORT          :R3      Abort main loop
Break 1 [20]> :a
[21]> (setf dozen 12)
12
[22]> dozen
12
[23]> (defin double (n) (* 2 n))

*** - EVAL: undefined function DEFIN
The following restarts are available:
USE-VALUE      :R1      Input a value to be used instead of (FDEFINITION
'DEFIN).
RETRY          :R2      Retry
STORE-VALUE    :R3      Input a new value for (FDEFINITION 'DEFIN).
ABORT          :R4      Abort main loop
Break 1 [24]> :a
[25]> (defun double (n) (* 2 n))
DOUBLE
[26]> (double 5)
10
[27]> (double pi)
6.283185307179586477L0
[28]> (double dozen)
24
[29]> (double pie)

*** - *: CHERRY is not a number
The following restarts are available:
USE-VALUE      :R1      Input a value to be used instead.
ABORT          :R2      Abort main loop
Break 1 [30]> :a
[31]> (bye)
Bye.
```

Listing of `numeric_session.text`:

```
$ clisp

{…snip…}

[1]> (+ 1 2 3 4 5 6 7 8 9 10)
55
[2]> (* 1 2 3 4 5 6 7 8 9 10)
3628800
[3]> (- 2 2 2)
-2
[4]> (/ 3 0.5)
6.0
[5]> (/ 3 5)
3/5
[6]> (sqrt 100)
10
[7]> (expt 7 50)
1798465042647412146620280340569649349251249
[8]> ;circumfrence of a circle of radius 10
(* 2 PI 10)
62.83185307179586477L0
[9]> ;area of a circle of radius 15
(* pi (expt 15 2))
706.8583470577034787L0
[10]> ;area of a circle of radius 17.2
(* pi (expt 17.2 2))
929.4089
[11]> ;area of the ring bounded by concentric circles of radii 15 and 17.2\
(- (* pi (expt 17.2 2)) (* pi (expt 15 2)))
222.55052
[12]> ;distance between the points (x1, x2) and (y1, y2)
(setf x1 5)
5
[13]> (setf x2 4)
4
[14]> (setf y1 -14)
-14
[15]> (setf y2 -12)
-12
[16]> (defun distance (a1 a2 b1 b2) (sqrt (+ (expt (- b1 a1) 2) (expt (- b2
a2) 2))))
DISTANCE
[17]> (distance x1 x2 y1 y2)
24.839485
[18]> (bye)
Bye.
```

Listing of `lp_session.text`:

```
CAR and CDR session:

$ clisp

{…snip…}

[1]> (car '(blue red yellow))
BLUE
[2]> (cdr '(blue red yellow))
(RED YELLOW)
[3]> (car '((1 2) buckle my shoe))
(1 2)
[4]> (cdr '((1 2) buckle my shoe))
(BUCKLE MY SHOE)
[5]> (car '("Sunshine"))
"Sunshine"
[6]> (cdr '("Sunshine"))
NIL
[7]> (bye)
Bye.

=========
CONS session:

$ clisp

{…snip…}

[1]> (cons 'espresso '(latte capucino))
(ESPRESSO LATTE CAPUCINO)
[2]> (cons '(a b c) '(1 2 3))
((A B C) 1 2 3)
[3]> (cons 'friday '())
(FRIDAY)
[4]> (bye)
Bye.

=========
Referencers session:

$ clisp

{…snip…}

[1]> (setf oo-languages '(simula smalltalk java clos))
(SIMULA SMALLTALK JAVA CLOS)
[2]> oo-languages
(SIMULA SMALLTALK JAVA CLOS)
[3]> (car oo-languages)
SIMULA
[4]> (cdr oo-languages)
(SMALLTALK JAVA CLOS)
```

```
[5]> (car (cdr oo-languages))
SMALLTALK
[6]> (cdr (cdr oo-languages))
(JAVA CLOS)
[7]> (cadr oo-languages)
SMALLTALK
[8]> (cddr oo-languages)
(JAVA CLOS)
[9]> (caddr oo-languages)
JAVA
[10]> (first oo-languages)
SIMULA
[11]> (second oo-languages)
SMALLTALK
[12]> (nth 2 oo-languages)
JAVA
[13]> (bye)
Bye.

=========
Constructors session:

$ clisp

{…snip…}

[1]> (setf letters '(a b c))
(A B C)
[2]> (setf numbers '(1 2 3))
(1 2 3)
[3]> letters
(A B C)
[4]> numbers
(1 2 3)
[5]> (cons letters numbers)
((A B C) 1 2 3)
[6]> (list letters numbers)
((A B C) (1 2 3))
[7]> (append letters numbers)
(A B C 1 2 3)
[8]> (list letters (cdr letters) (cddr letters))
((A B C) (B C) (C))
[9]> (append letters (cdr letters) (cddr letters))
(A B C B C C)
[10]> (bye)
Bye.

=========
"colors" session:

$ clisp

{…snip…}

[1]> (setf colors '(purple blue orange))
```

```
(PURPLE BLUE ORANGE)
[2]> 'colors
COLORS
[3]> colors
(PURPLE BLUE ORANGE)
[4]> (describe 'colors)

COLORS is the symbol COLORS, lies in #<PACKAGE COMMON-LISP-USER>, is
accessible in 1 package COMMON-LISP-USER, a variable, value:
(PURPLE BLUE ORANGE).

 #<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
 no symbols, but no package uses these exports.

 (PURPLE BLUE ORANGE) is a list of length 3.

[5]> (describe colors)

(PURPLE BLUE ORANGE) is a list of length 3.

[6]> (type-of 'colors)
SYMBOL
[7]> (type-of colors)
CONS
[8]> (bye)
Bye.

=========
"moderately imaginitive forms" session:

$ clisp

{…snip…}

[1]> (defun increment-first (n) (+ 1 (car n)))
INCREMENT-FIRST
[2]> (increment-first '(1 3 2))
2
[3]> (defun increment-second (n) (+ 1 (car (cdr n))))
INCREMENT-SECOND
[4]> (increment-second '(1 3 2))
4
[5]> (defun merge-all-but-firsts (n m) (cons (cdr n) (cdr m)))
MERGE-ALL-BUT-FIRSTS
[6]> (merge-all-but-firsts '(1 2 3) '(a b c))
((2 3) B C)
[7]> (defun merge-firsts (n m) (list (car n) (car m)))
MERGE-FIRSTS
[8]> (merge-firsts '(1 2 3) '(a b c))
(1 A)
[9]> (defun append-all-but-firsts (n m) (append (cdr n) (cdr m)))
APPEND-ALL-BUT-FIRSTS
[10]> (append-all-but-firsts '(1 2 3) '(a b c))
```

```
(2 3 B C)
[11]> (setf crazy-2 (car (cdddar '((1 2 3 2 ) 5 ))))
2
[12]> crazy-2
2
[13]> (bye)
Bye.
```