

Homework #1

1) Create your own WWW page. Explain your background and present your interest in robotics. What kind of project you want to do and why. What you expect to learn in this class. Your favorite robots from Internet? Send me by email the address of your page so I will link it for all class students to see. These pages will help in our group communication and especially in projects.

The webpage is at <http://web.pdx.edu/~tzewen/ece578>.

2) Write a test of as many lisp functions as you can do in one week. You should invent tests for the following functions: car, cdr, caddr, cons, list, set, setq, prog, progn, cond, atom, number, greaterp, plus, minus, minusp, numberp, go, nconc, reverse, nreverse, print, trace, traceset, debug. Additional tests of other functions are very welcome and will increase your grade. You should be acquainted with LISP environment and interactive use after this homework.

Function	Test Cases	Result
car	(car NIL)	NIL
	(car `NIL)	NIL
	(car `a)	**A is not a LIST
	(car `())	NIL
	(car `(NIL))	NIL
	(car `(()))	NIL
	(car `(h))	H
	(car `(h t))	H
	(car `((h t) t))	(HH TT)
	(car `(h (h r)))	H
	(car `(NIL h (hh tt)))	NIL
(car `(() h (hh tt)))	NIL	
cdr	(cdr `(h))	NIL
	(cdr `(h t))	(T)
	(cdr `((h t) t))	(T)
	(cdr `(h (h r)))	((HH TT))
	(cdr `(h (hh tt) NIL))	((HH TT) NIL)
	(cdr `(h (hh tt) ()))	((HH TT) NIL)
caddr	(caddr `(h))	NIL
	(caddr `(h t))	NIL
	(caddr `((h t) t))	NIL
	(caddr `(h (h r)))	NIL

	(caddr `(h (hh tt) t))	T
	(caddr `(h m (hh tt) t))	(HH TT)
	(caddr `(h (hh tt) NIL))	NIL
	(caddr `(h (hh tt) ()))	NIL
	(caddr `(NIL NIL a))	(A)
cons	(cons NIL NIL)	(NIL)
	(cons NIL `(NIL))	(NIL NIL)
	(cons `(NIL) NIL)	((NIL))
	(cons `(NIL) `(NIL))	((NIL) NIL)
	(cons `NIL `t)	(NIL . T)
	(cons `NIL `(t))	(NIL T)
	(cons `(NIL) `t)	((NIL) . T)
	(cons `(NIL) `(t))	((NIL) T)
	(cons `h `t)	(H . T)
	(cons `h `(hh tt))	(H (HH TT))
	(cons `(h) `t)	((H) . T)
	(cons `(h) `(t))	((H) T)
	(cons `(h) `(hh tt))	((H) (HH TT))
list	(list NIL)	(NIL)
	(list NIL NIL)	(NIL NIL)
	(list `(NIL) NIL)	((NIL) NIL)
	(list `a)	(A)
	(list `h `t)	(H T)
	(list `(hh tt) `t)	((HH TT) T)
	(list `h `(hh tt))	(H (HH TT))
	(list NIL `h `t)	(NIL H T)
setq	(setq L `a)	A
	(setq L `(a b))	(A B)
	(setq L NIL)	NIL
	(setq L `(NIL))	(NIL)
	(setq L `(a NIL))	(A NIL)
	(cdr (setq L `(a NIL)))	(NIL)
	(car (setq L `(a NIL)))	A
prog	(prog NIL (car `(a b)) (car `(a b c)))	NIL
	(progn (let (var)) (setf var `local) (print var))	var is bound to local
progn	(progn NIL (car `(a b)) (car `(a b c)))	A

	(progn (let (var)) (setf var `local) (print var)) var))	var is bound to global
cond	(cond (eq 0 0) `YES) (T `NO))	YES
atom	(atom NIL)	T
	(atom `())	T
	(atom `a)	T
	(atom `(a))	NIL
number	0	0
	0.00001	1.0E-5
	-0.00001	-1.0E-5
	+0.00001	1.0E-5
	0.00001	0.00001
numberp	(numberp NIL)	NIL
	(numberp `(0))	NIL
	(numberp 0)	T
	(numberp -0)	T
	(numberp 1.0E-5)	T
	(numberp 0.001)	T
plus	(+ 1 0.00001)	1.00001
	(+ 1000 1.0E-5)	1000.0
minus	(- 1 0.1)	0.9
	(- 1000 1.0E-5)	1000.0
minusp	(minus 0)	NIL
	(minus -0)	NIL
	(minus -1)	T
	(minus 1)	NIL
greaterp	<i>undefined function</i>	
go	(tagbody a (print "hi") (go a)))	hi hi hi hi ...
	(go nil)	error
	(go t)	error
nconc	(tagbody F (setf L `(a b)) (print L) (nconc L `(c d)) (print L))	(A B) (A B C D)
reverse	(reverse `(a b c d))	(D C B A)
	(reverse `(a (b c) d e))	(E D (B C) A)
nreverse	(tagbody F (setf L `(a b)) (print L) (nreverse L) (print L))	(A B) (B A)

print	(print `hello)	hello
	(print "hello")	"hello"
	(print (cdr `(a b c)))	(B C)
	(print NIL)	NIL
trace	(trace copy)	Tracing function COPY
	(trace car)	error
traceset	<i>undefined function</i>	
debug	<i>undefined function</i>	

3) Write a LISP program that inverts the order of atoms in a list given as its argument.

```
;; inv(L)
;; Take a inverse of a list of atoms.
;;
;; Parameters:
;; L - The list to be replicated.
;;
(defun inv (L)
  (cond ((eq (cdr L) NIL) L)
        (T (append (inv (cdr L)) (list (car L)))))
  )
)
```

4) Write a LISP program that copies an arbitrary list data structure on all levels but atoms. Show on examples how it works.

```
;; copy(S)
;; Clone an arbitrary structure.
;;
;; Parameters:
;; S - The structure to be replicated.
;;
(defun copy (S)
  (cond ((atom S) S)
        (T (cons (copy (car S)) (copy (cdr S)))))
  )
)

(setq S `(a (b (c ())) d))
```

From the trace below, line 1 is the top level, initial call to the COPY function. It has the entire list L1 as an argument. The condition falls true on T-clause. So the function is invoked with car of the list L1 which is the second level recursion on line 2 and returns an atom. Then it invokes the second level recursion again on line 3 with the tail of the list L1. On the second level recursion, the copy function retrieves a list L2 which is the tail of L1 and proceeds the same procedure on the first level recursion recursively. The recursion stops when a tail of a list L_n is a NIL. The NIL itself is also an atom. Consequentially, the entire recursion terminates. This example is a 7-depth of recursion.

```
1: 1. Trace: (COPY '(A (B (C NIL)) D))
2: 2. Trace: (COPY 'A)
```

```
3: 2. Trace: COPY ==> A
4: 2. Trace: (COPY '((B (C NIL)) D))
5: 3. Trace: (COPY '(B (C NIL)))
6: 4. Trace: (COPY 'B)
7: 4. Trace: COPY ==> B
8: 4. Trace: (COPY '((C NIL)))
9: 5. Trace: (COPY '(C NIL))
10: 6. Trace: (COPY 'C)
11: 6. Trace: COPY ==> C
12: 6. Trace: (COPY '(NIL))
13: 7. Trace: (COPY 'NIL)
14: 7. Trace: COPY ==> NIL
15: 7. Trace: (COPY 'NIL)
16: 7. Trace: COPY ==> NIL
17: 6. Trace: COPY ==> (NIL)
18: 5. Trace: COPY ==> (C NIL)
19: 5. Trace: (COPY 'NIL)
20: 5. Trace: COPY ==> NIL
21: 4. Trace: COPY ==> ((C NIL))
22: 3. Trace: COPY ==> (B (C NIL))
23: 3. Trace: (COPY '(D))
24: 4. Trace: (COPY 'D)
25: 4. Trace: COPY ==> D
26: 4. Trace: (COPY 'NIL)
27: 4. Trace: COPY ==> NIL
28: 3. Trace: COPY ==> (D)
29: 2. Trace: COPY ==> ((B (C NIL)) D)
30: 1. Trace: COPY ==> (A (B (C NIL)) D)
31: (A (B (C NIL)) D)
```