

Recursive List Processing and HOFs

Task 1:

Code:

```
( defun singleton-p ( l )  
  ( cond  
    ( ( null l ) nil )  
    ( ( null ( cdr l ) ) t )  
    ( t nil )  
  )  
)
```

```
( defun rac ( l )  
  ( cond  
    ( ( singleton-p l )  
      ( car l )  
    )  
    ( t  
      ( rac ( cdr l ) )  
    )  
  )  
)
```

```
( defun rdc ( l )  
  ( cond  
    ( ( singleton-p l )  
      ()  
    )  
    ( t  
      ( cons ( car l ) ( rdc ( cdr l ) ) )  
    )  
  )  
)
```

```
( defun snoc ( o l )  
  ( cond  
    ( ( null l )  
      ( list o )  
    )  
    ( t  
      ( cons ( car l ) ( snoc o ( cdr l ) ) )  
    )  
  )  
)
```

```

)

(defun palindrome-p (l)
  (cond
    ((null l)
     t)
    ((singleton-p l)
     t)
    ((equal (car l) (rac l))
     (palindrome-p (cdr (rdc l))))
    (t
     nil)
  )
)
)

```

Demo:

```
CL-USER> (load "lp.lsp")
```

```
T
```

```
CL-USER> (singleton-p '(one))
```

```
T
```

```
CL-USER> (singleton-p '(one two))
```

```
NIL
```

```
CL-USER> (singleton-p '(one two three four five six seven))
```

```
NIL
```

```
CL-USER> (trace rac)
```

```
(RAC)
```

```
CL-USER> (rac '(one))
```

```
0: (RAC (ONE))
```

```
0: RAC returned ONE
```

```
ONE
```

```
CL-USER> (rac '(one two three four))
```

```
0: (RAC (ONE TWO THREE FOUR))
```

```
1: (RAC (TWO THREE FOUR))
```

```
2: (RAC (THREE FOUR))
```

```
3: (RAC (FOUR))
```

```
3: RAC returned FOUR
```

```

    2: RAC returned FOUR
    1: RAC returned FOUR
    0: RAC returned FOUR
FOUR
CL-USER> (trace rdc)
(RDC)
CL-USER> (rdc '(one))
    0: (RDC (ONE))
    0: RDC returned NIL
NIL
CL-USER> (rdc '(one two three four five))
    0: (RDC (ONE TWO THREE FOUR FIVE))
    1: (RDC (TWO THREE FOUR FIVE))
    2: (RDC (THREE FOUR FIVE))
    3: (RDC (FOUR FIVE))
    4: (RDC (FIVE))
    4: RDC returned NIL
    3: RDC returned (FOUR)
    2: RDC returned (THREE FOUR)
    1: RDC returned (TWO THREE FOUR)
    0: RDC returned (ONE TWO THREE FOUR)
(ONE TWO THREE FOUR)
CL-USER> (untrace rac)
T
CL-USER> (untrace rdc)
T
CL-USER> (trace snoc)
(SNOC)
CL-USER> (snoc 'blue '())
    0: (SNOC BLUE NIL)
    0: SNOC returned (BLUE)
(BLUE)
CL-USER> (snoc 'blue '(red))
    0: (SNOC BLUE (RED))
    1: (SNOC BLUE NIL)
    1: SNOC returned (BLUE)
    0: SNOC returned (RED BLUE)

```

(RED BLUE)

CL-USER> (snoc 'blue '(blueish1 blueish2 blueish3 blueish4))

0: (SNOC BLUE (BLUEISH1 BLUEISH2 BLUEISH3 BLUEISH4))

1: (SNOC BLUE (BLUEISH2 BLUEISH3 BLUEISH4))

2: (SNOC BLUE (BLUEISH3 BLUEISH4))

3: (SNOC BLUE (BLUEISH4))

4: (SNOC BLUE NIL)

4: SNOC returned (BLUE)

3: SNOC returned (BLUEISH4 BLUE)

2: SNOC returned (BLUEISH3 BLUEISH4 BLUE)

1: SNOC returned (BLUEISH2 BLUEISH3 BLUEISH4 BLUE)

0: SNOC returned (BLUEISH1 BLUEISH2 BLUEISH3 BLUEISH4 BLUE)

(BLUEISH1 BLUEISH2 BLUEISH3 BLUEISH4 BLUE)

CL-USER> (untrace snoc)

T

CL-USER> (trace palindrome-p)

(PALINDROME-P)

CL-USER> (palindrome-p '())

0: (PALINDROME-P NIL)

0: PALINDROME-P returned T

T

CL-USER> (palindrome-p '(palindrome))

0: (PALINDROME-P (PALINDROME))

0: PALINDROME-P returned T

T

CL-USER> (palindrome-p '(clos sloc))

0: (PALINDROME-P (CLOS SLOC))

0: PALINDROME-P returned NIL

NIL

CL-USER> (palindrome-p '(food drink food))

0: (PALINDROME-P (FOOD DRINK FOOD))

1: (PALINDROME-P (DRINK))

1: PALINDROME-P returned T

0: PALINDROME-P returned T

T

CL-USER> (palindrome-p '(1 2 3 4 5 4 2 3 1))

0: (PALINDROME-P (1 2 3 4 5 4 2 3 1))

1: (PALINDROME-P (2 3 4 5 4 2 3))

1: PALINDROME-P returned NIL

0: PALINDROME-P returned NIL

NIL

CL-USER> (palindrome-p '(hey hey my my my my hey hey))

0: (PALINDROME-P (HEY HEY MY MY MY MY HEY HEY))

1: (PALINDROME-P (HEY MY MY MY MY HEY))

2: (PALINDROME-P (MY MY MY MY))

3: (PALINDROME-P (MY MY))

4: (PALINDROME-P NIL)

4: PALINDROME-P returned T

3: PALINDROME-P returned T

2: PALINDROME-P returned T

1: PALINDROME-P returned T

0: PALINDROME-P returned T

T

Task 2:

Code:

```
( defun select ( n l )  
  ( cond  
    ( ( = n 1 )  
      ( car l )  
    )  
    ( t ( select ( - n 1 ) ( cdr l ) ) )  
  )  
)
```

```
( defun pick ( l )  
  ( select ( random ( length l ) ) l )  
)
```

Demo:

CL-USER> (trace select)

(SELECT)

CL-USER> (select 3 '(a b c d e f))

0: (SELECT 3 (A B C D E F))

1: (SELECT 2 (B C D E F))

2: (SELECT 1 (C D E F))

2: SELECT returned C

1: SELECT returned C

0: SELECT returned C

C

CL-USER> (select 2 '(91 83 86 83 03))

0: (SELECT 2 (91 83 86 93 3))

1: (SELECT 1 (83 86 93 3))

1: SELECT returned 83

0: SELECT returned 83

83

CL-USER> (select 1 '(kieran ant pec greg cj))

0: (SELECT 1 (KIERAN ANT PEC GREG CJ))

0: SELECT returned KIERAN

KIERAN

CL-USER> (untrace select)

T

CL-USER> (trace pick)

(PICK)

CL-USER> (untrace pick)

T

CL-USER> (pick '(23 83 79 93 23))

23

CL-USER> (pick '(pizza burger cookies pancakes))

BURGER

CL-USER> (pick '(r6 overwatch madden theshow))

MADDEN

Task 3:

Code:

```
( defun sum ( l )  
  ( cond  
    ( ( null l )  
      0  
    )  
    ( t  
      ( + ( car l ) ( sum ( cdr l ) ) )  
    )  
  )  
)
```

```
( defun product ( l )  
  ( cond  
    ( ( null l )  
      1  
    )  
    ( t  
      ( * ( car l ) ( product ( cdr l ) ) )  
    )  
  )  
)
```

Demo:

```
CL-USER> ( load "lp.lsp" )
```

```
T
```

```
CL-USER> ( trace sum )
```

```
(SUM)
```

```
CL-USER> ( trace product )
```

```
WARNING: PRODUCT is already TRACE'd, untracing it first.
```

```
(PRODUCT)
```

```
CL-USER> ( sum '() )
```

```
0: (SUM NIL)
```

```
0: SUM returned 0
```

```
0
```

```
CL-USER> (product '() )
```

```
0: (PRODUCT NIL)
```

```
0: PRODUCT returned 1
```

```
1
```


CL-USER> (sum '(496))

0: (SUM (496))

1: (SUM NIL)

1: SUM returned 0

0: SUM returned 496

496

CL-USER> (sum '(1 11 111))

0: (SUM (1 11 111))

1: (SUM (11 111))

2: (SUM (111))

3: (SUM NIL)

3: SUM returned 0

2: SUM returned 111

1: SUM returned 122

0: SUM returned 123

123

CL-USER>

; No value

CL-USER> (product '(1 11 111))

0: (PRODUCT (1 11 111))

1: (PRODUCT (11 111))

2: (PRODUCT (111))

3: (PRODUCT NIL)

3: PRODUCT returned 1

2: PRODUCT returned 111

1: PRODUCT returned 1221

0: PRODUCT returned 1221

1221

CL-USER> (sum '(1 2 3 4 5 6 7 8 9 10))

0: (SUM (1 2 3 4 5 6 7 8 9 10))

1: (SUM (2 3 4 5 6 7 8 9 10))

2: (SUM (3 4 5 6 7 8 9 10))

3: (SUM (4 5 6 7 8 9 10))

4: (SUM (5 6 7 8 9 10))

5: (SUM (6 7 8 9 10))

6: (SUM (7 8 9 10))

7: (SUM (8 9 10))

8: (SUM (9 10))

9: (SUM (10))

10: (SUM NIL)

10: SUM returned 0
 9: SUM returned 10
 8: SUM returned 19
 7: SUM returned 27
 6: SUM returned 34
 5: SUM returned 40
 4: SUM returned 45
 3: SUM returned 49
 2: SUM returned 52
 1: SUM returned 54
 0: SUM returned 55
 55
 CL-USER> (product '(1 2 3 4 5 6 7 8 9 10))
 0: (PRODUCT (1 2 3 4 5 6 7 8 9 10))
 1: (PRODUCT (2 3 4 5 6 7 8 9 10))
 2: (PRODUCT (3 4 5 6 7 8 9 10))
 3: (PRODUCT (4 5 6 7 8 9 10))
 4: (PRODUCT (5 6 7 8 9 10))
 5: (PRODUCT (6 7 8 9 10))
 6: (PRODUCT (7 8 9 10))
 7: (PRODUCT (8 9 10))
 8: (PRODUCT (9 10))
 9: (PRODUCT (10))
 10: (PRODUCT NIL)
 10: PRODUCT returned 1
 9: PRODUCT returned 10
 8: PRODUCT returned 90
 7: PRODUCT returned 720
 6: PRODUCT returned 5040
 5: PRODUCT returned 30240
 4: PRODUCT returned 151200
 3: PRODUCT returned 604800
 2: PRODUCT returned 1814400
 1: PRODUCT returned 3628800
 0: PRODUCT returned 3628800
 3628800

Task 4:

Code:

```
( defun iota ( n )
  ( cond
    ( ( = n 0 )
      ()
    )
    ( t
      ( snoc n ( iota ( - n 1 ) ) )
    )
  )
)

( defun duplicate ( n lo )
  ( cond
    ( ( = n 0 )
      ()
    )
    ( t
      ( snoc lo ( duplicate ( - n 1 ) lo ) )
    )
  )
)
```

Demo:

```
CL-USER> ( load "lp.lsp" )
T
CL-USER> ( trace iota )
(IOTA)
CL-USER> ( trace duplicate )
(DUPLICATE)
CL-USER> ( iota 1 )
0: (IOTA 1)
1: (IOTA 0)
1: IOTA returned NIL
0: IOTA returned (1)
(1)
CL-USER> ( iota 10 )
```

```

0: (IOTA 10)
1: (IOTA 9)
2: (IOTA 8)
3: (IOTA 7)
4: (IOTA 6)
5: (IOTA 5)
6: (IOTA 4)
7: (IOTA 3)
8: (IOTA 2)
9: (IOTA 1)
10: (IOTA 0)
10: IOTA returned NIL
9: IOTA returned (1)
8: IOTA returned (1 2)
7: IOTA returned (1 2 3)
6: IOTA returned (1 2 3 4)
5: IOTA returned (1 2 3 4 5)
4: IOTA returned (1 2 3 4 5 6)
3: IOTA returned (1 2 3 4 5 6 7)
2: IOTA returned (1 2 3 4 5 6 7 8)
1: IOTA returned (1 2 3 4 5 6 7 8 9)
0: IOTA returned (1 2 3 4 5 6 7 8 9 10)
(1 2 3 4 5 6 7 8 9 10)
CL-USER> (duplicate 3 'boing)
0: (DUPLICATE 3 BOING)
1: (DUPLICATE 2 BOING)
2: (DUPLICATE 1 BOING)
3: (DUPLICATE 0 BOING)
3: DUPLICATE returned NIL
2: DUPLICATE returned (BOING)
1: DUPLICATE returned (BOING BOING)
0: DUPLICATE returned (BOING BOING BOING)
(BOING BOING BOING)
CL-USER> ( duplicate 9 '9)
0: (DUPLICATE 9 9)
1: (DUPLICATE 8 9)
2: (DUPLICATE 7 9)
3: (DUPLICATE 6 9)
4: (DUPLICATE 5 9)
5: (DUPLICATE 4 9)

```

6: (DUPLICATE 3 9)
7: (DUPLICATE 2 9)
8: (DUPLICATE 1 9)
9: (DUPLICATE 0 9)
9: DUPLICATE returned NIL
8: DUPLICATE returned (9)
7: DUPLICATE returned (9 9)
6: DUPLICATE returned (9 9 9)
5: DUPLICATE returned (9 9 9 9)
4: DUPLICATE returned (9 9 9 9 9)
3: DUPLICATE returned (9 9 9 9 9 9)
2: DUPLICATE returned (9 9 9 9 9 9 9)
1: DUPLICATE returned (9 9 9 9 9 9 9 9)
0: DUPLICATE returned (9 9 9 9 9 9 9 9 9)
(9 9 9 9 9 9 9 9 9)

Task 5:

Code:

```
( defun factorial ( n )  
  ( product ( iota n ) )  
)
```

```
( defun power ( n1 n2 )  
  ( product ( duplicate n2 n1 ) )  
)
```

Demo:

```
CL-USER> (load "lp.lsp")
```

```
T
```

```
CL-USER> ( factorial 5 )
```

```
120
```

```
CL-USER> (factorial 10 )
```

```
3628800
```

```
CL-USER> (power 2 16)
```

```
65536
```

```
CL-USER> (power 5 6)
```

```
15625
```

Task 6:

Code:

```
( defun filter-in ( p l )  
  ( cond  
    ( ( null l ) nil )  
    ( ( funcall p ( car l ) )  
      ( cons ( car l ) ( filter-in p ( cdr l ) ) )  
    )  
    ( t ( filter-in p ( cdr l ) ) )  
  )  
)
```

```
( defun filter-out ( p l )  
  ( cond  
    ( ( null l ) nil )  
    ( ( funcall p ( car l ) ) ( filter-out p ( cdr l ) ) )  
    ( t ( cons ( car l ) ( filter-out p ( cdr l ) ) ) )  
  )  
)
```

Demo:

```
CL-USER> ( filter-in #'palindrome-p '( ( 1 2 3 3 2 1 ) ( 1 ) ( 1 2 3 4 5 ) ) )  
((1 2 3 3 2 1) (1))
```

```
CL-USER> ( filter-in #'singleton-p '( ( blue ) ( red blue green ) ) )  
((BLUE))
```

```
CL-USER> ( defun large ( n )  
  ( > n 100 ) )
```

LARGE

```
CL-USER> ( filter-in #'large '( 101 30 435 23 56 546 ) )  
(101 435 546)
```

```
CL-USER> ( filter-out #'palindrome-p '( ( 1 2 3 3 2 1 ) ( 1 ) ( 1 2 3 4 5 ) ) )  
((1 2 3 4 5))
```

```
CL-USER> ( filter-out #'singleton-p '( ( blue ) ( blue red yellow ) ( 1 2 ) ( 3 ) ) )  
((BLUE RED YELLOW) (1 2))
```

```
CL-USER> ( filter-out #'large '( 12 4230 432 45 20 583 ) )  
(12 45 20)
```

Task 7:

Code:

```
( defun take-from ( o l )  
  ( cond  
    ( ( null l ) nil )  
    ( ( equal o ( car l ) )  
      ( take-from o ( cdr l ) )  
    )  
    ( t  
      ( cons ( car l ) ( take-from o ( cdr l ) ) )  
    )  
  )  
)
```

Demo:

```
CL-USER> ( take-from 5 '( 12 3 5 3 23 5 34 5 ) )  
(12 3 3 23 34)
```

```
CL-USER> ( take-from 'kieran '( kieran ant kieran cj paul ryan ant ) )  
(ANT CJ PAUL RYAN ANT)
```

```
CL-USER> ( take-from 'true '( true true false true false false ) )  
(FALSE FALSE FALSE)
```


Task 8:

Code:

```
( defun random-permutation ( l )  
  ( cond  
    ( ( null l ) ( ) )  
    ( t  
      ( setf r ( nth ( random ( length l ) ) l ) )  
      ( setf l ( remove r l :count 1 ) )  
      ( cond r ( random-permutation l ) )  
    )  
  )  
)
```

Demo:

```
CL-USER> ( random-permutation '( 1 2 3 4 5 6 7 8 9 ) )  
(6 2 4 7 8 3 1 5 9)  
CL-USER> ( random-permutation '( kieran ant cj pec paul ryan kyle squid ) )  
(ANT PAUL PEC RYAN KIERAN KYLE CJ SQUID)  
CL-USER> ( random-permutation '(calc ai compsys quality ) )  
(QUALITY CALC COMPSYS AI)  
CL-USER> ( trace random-permutation )  
(RANDOM-PERMUTATION)  
CL-USER> ( random-permutation '(p c 1b 2b 3b ss lf cf rf) )  
0: (RANDOM-PERMUTATION (P C |1B| |2B| |3B| SS LF CF RF))  
1: (RANDOM-PERMUTATION (P |1B| |2B| |3B| SS LF CF RF))  
2: (RANDOM-PERMUTATION (P |1B| |3B| SS LF CF RF))  
3: (RANDOM-PERMUTATION (P |1B| |3B| SS LF CF))  
4: (RANDOM-PERMUTATION (P |3B| SS LF CF))  
5: (RANDOM-PERMUTATION (P |3B| SS CF))  
6: (RANDOM-PERMUTATION (|3B| SS CF))  
7: (RANDOM-PERMUTATION (SS CF))  
8: (RANDOM-PERMUTATION (CF))  
9: (RANDOM-PERMUTATION NIL)  
9: RANDOM-PERMUTATION returned NIL  
8: RANDOM-PERMUTATION returned (CF)  
7: RANDOM-PERMUTATION returned (SS CF)  
6: RANDOM-PERMUTATION returned (|3B| SS CF)
```

5: RANDOM-PERMUTATION returned (P |3B| SS CF)
4: RANDOM-PERMUTATION returned (LF P |3B| SS CF)
3: RANDOM-PERMUTATION returned (|1B| LF P |3B| SS CF)
2: RANDOM-PERMUTATION returned (RF |1B| LF P |3B| SS CF)
1: RANDOM-PERMUTATION returned (|2B| RF |1B| LF P |3B| SS CF)
0: RANDOM-PERMUTATION returned (C |2B| RF |1B| LF P |3B| SS CF)
(C |2B| RF |1B| LF P |3B| SS CF)
CL-USER>

Task 9:

Task 10:

Task 11: