```
$Id: 2014q4-soln3,v 1.2 2014-12-11 13:28:47-08 - - $
Answers to 2014q4-test3, page 1

_____________________________________________________________________
Question 1. [2]

let reverse list =
    let rec rev' in out = match in with
        | [] -> out
        | h::t -> rev' t (h::out)
    in rev' list []

_____________________________________________________________________
Question 2. [3]

let reverse = List.fold_left (fun tl hd -> hd::tl) []
let sum = List.fold_left (+) 0
let length = List.fold_left (fun _ n -> n + 1) 0

_____________________________________________________________________
Question 3. [3]

let collatz n =
    let rec collatz' n rest =
        if n <= 1
           then 1::rest
           else if even n
                   then collatz' (n / 2) (n::rest)
                   else collatz' (n * 3 + 1) (n::rest)
    in  reverse (collatz' n [])

_____________________________________________________________________
Question 4. [2]

(define (takex n list)
    (if (or (null? list) (<= n 0)) '()
        (cons (car list) (takex (- n 1) (cdr list)))))
```

```
39:
40: Answers to 2014q4-test3, page 2
41:
42: ____________________________________________________________________
43: Question 5. [2]
44:
45:    It may return a result (including unit or void).
46:    It may raise (throw) an exception.
47:    It may never return (infinite loop or recursion).
48:    It may exit the program.
49:
50: ____________________________________________________________________
51: Question 6. [2]
52:
53: Applicative order    Normal order
54: (\x. * x x) 5        (* (+ 2 3) (+ 2 3))
55: (* 5 5)              (* 5 5)
56: 25                   25
57:
58: ____________________________________________________________________
59: Question 7. [2]
60:
61: fac  : int -> int
62: n    : int
63: 0    : int
64: fac' : int -> int -> int
65: n'   : int
66: m'   : int
67: -    : int -> int -> int
68: *    : int -> int -> int
69:
70: ____________________________________________________________________
71: Question 8. [2]
72:
73: edge(a,b).       % Note: the edge facts may be listed in either order.
74: edge(a,c).       % e.g., edge(a,b) and edge(b,a) mean the same thing.
75: edge(a,d).
76: edge(b,c).
77: edge(c,d).
78: adjacent(X,Y) :- edge(X,Y).
79: adjacent(X,Y) :- edge(Y,X).
80:
81: ____________________________________________________________________
82: Question 9. [2]
83:
84: (define (map f list)
85:     (if (null? list) '()
86:         (cons (f (car list)) (map f (cdr list)))))
87:
88: (define (filter p list)
89:     (if (null? list) '()
90:         (let ((hd (car list))
91:               (tl (filter p (cdr list))))
92:              (if (p hd) (cons hd tl)
93:                         (tl)))))
94:
```

```
 95:
 96: Answers to 2014q4-test3, page 3
 97:
 98: ____________________________________________________________________
 99: Question 10. [1]
100:
101: print while <>;
102:
103: ____________________________________________________________________
104: Question 11. [2]
105:
106: mother(Mother,Child) :- parents(_,Mother,Child).
107: father(Father,Child) :- parents(Father,_,Child).
108:
109: ____________________________________________________________________
110: Question 12. [3]
111:
112: let rec range n m =
113:     if n > m then []
114:              else n::range (n+1) m;;
115:
116: (define (range n m)
117:     (if (> n m) '()
118:         (cons n (range (+ n 1) m))))
119:
120: sub range {
121:    my ($n,$m) = @_;
122:    return $n..$m;
123: }
124:
125: ____________________________________________________________________
126: Question 13. [4]
127:
128: (define (pairthem l1 l2)
129:     (if (or (null? l1) (null? l2)) '()
130:         (cons (list (car l1) (car l2))
131:               (pairthem (cdr l1) (cdr l2)))))
132:
133: let rec pairthem l1 l2 = match l1, l2 with
134:    | [], _ -> []
135:    | _, [] -> []
136:    | h1::t1, h2::t2 -> (h1,h2):: pairthem t1 t2;;
137:
```

```
138:
139: Answers to 2014q4-test3, page 4
140:
141:  1.    (B) lambda
142:
143:  2.    (D) Smalltalk
144:
145:  3.    (D) fold left $ O ( n ) $;  fold right $ O ( n ) $
146:
147:  4.    (B) fold left $ O ( 1 ) $;  fold right $ O ( n ) $
148:
149:  5.    (A) find(X) :- guess(X), verify(X).
150:
151:  6.    (C) ('a -> 'b) -> 'a list -> 'b list
152:
153:  7.    (A) The message +4 is sent to the object 3.
154:
155:  8.    (D) unit -> int
156:
157:  9.    (C) ML and Ocaml
158:
159: 10.    (C) reachable
160:
161: 11.    (D) virtual function table
162:
163: 12.    (C) Donald Knuth
164:
```

```
165:
166: Answers to 2014q4-test3, page 5
167:
168:  1.    (A) C++
169:
170:  2.    (A) (*...*)
171:
172:  3.    (D) reference counting
173:
174:  4.    (D) float -> float -> float
175:
176:  5.    (D) throw
177:
178:  6.    (C) a pointer to the stack frame in which the current
179:            function is nested.
180:
181:  7.    (A) Haskell
182:
183:  8.    (D) a value: 3.
184:
185:  9.    (D) print `date`;
186:
187: 10.    (B) 0.0/0.0
188:
189: 11.    (B) [4 + 5] value.
190:
191: 12.    (A) Edsger Dijkstra
192:
```