

```
1: $Id: 2014q4-soln1,v 1.3 2015-10-27 14:56:55-07 - - $
2: Answers to cmps112-2015q4-exam1, page 1
3:
4:
5: Question 1. [2]
6: (a) Return a value (or void).
7: (b) Throw an exception.
8: (c) Exit the program.
9: (d) Go into an endless loop or recursion.
10:
11:
12: Question 2(a). [2]
13: (define (filter p? list)
14:   (if (null? list) '()
15:       (let ((a (car list))
16:             (fd (filter p? (cdr list))))
17:         (if (p? a) (cons a fd) fd))))
18:
19:
20: Question 2(b). [2]
21: let rec filter p list = match list with
22:   | [] -> []
23:   | car::cdr -> if p car then car::filter p cdr
24:                 else filter p cdr
25:
26:
27: Question 3(a). [1]
28: (define (length x)
29:   (define (len x n)
30:     (if (null? x) n
31:         (len (cdr x) (+ n 1))))
32:   (len x 0))
33:
34:
35: Question 3(b). [1]
36: let length x =
37:   let rec len x n = match x with
38:     | [] -> n
39:     | _::cdr -> len cdr (n + 1)
40:   in len x 0
41:
42:
43: Question 4. [2]
44: let rec sub' num1 num2 carry = match (num1, num2, carry) with
45:   | list1, [], 0 -> list1
46:   | list1, [], carry -> sub' list1 [- carry] 0
47:   | [], _::_, _ -> raise (Invalid_argument "sub'")
48:   | h1::t1, h2::t2, carry ->
49:     let diff = h1 - h2 + carry + 10
50:     in diff mod 10 :: sub' t1 t2 (diff / 10 - 1)
51:
```

```
52:
53: Answers to cmpls112-2015q4-exam1, page 2
54:
55:
56: Question 5(a). [2]
57: (define (fold_left fn unit list)
58:   (if (null? list) unit
59:       (fold_left fn (fn unit (car list)) (cdr list))))
60:
61:
62: Question 5(b). [2]
63: let rec fold_left fn unit list = match list with
64:   | [] -> unit
65:   | car::cdr -> fold_left fn (fn unit car) cdr
66:
67:
68: Question 6(a). [2]
69: ---- The following is 2 points, if correct.
70: (define (reverse list)
71:   (define (rev list m)
72:     (if (null? list) m
73:         (rev (cdr list) (cons (car list) m))))
74:   (rev list '()))
75: ---- Either of the following for 3 points, if correct.
76: (define (snoc cdr car) (cons car cdr))
77: (define (reverse list) (fold_left snoc '() list))
78: (define (reverse list) (fold_left (lambda (d a) (cons a d)) '() list))
79:
80:
81: Question 6(b). [2]
82: ---- The following is 2 points, if correct.
83: let reverse list =
84:   let rec rev' src out = match src with
85:     | [] -> out
86:     | h::t -> rev' t (h::out)
87:   in rev' list []
88: ---- Either of the following for 3 points, if correct.
89: let reverse = List.fold_left (fun tl hd -> hd::tl) []
90: let reverse = fold_left (fun tl hd -> hd::tl) []
91:
92:
93: Question 7. [2]
94: node* reverse (node* head) {
95:   node* out = NULL;
96:   while (head != NULL) {
97:     node* t = head;
98:     head = head->link;
99:     t->link = out;
100:    out = t;
101:   }
102:   return out;
103: }
104:
```

```
105:
106: Answers to cmps112-2015q4-exam1, page 3
107:
108: 1.      (D) parametric
109:
110: 2.      (C) 3
111:
112: 3.      (D) int -> int -> int
113:
114: 4.      (D) 4
115:
116: 5.      (B) int list
117:
118: 6.      (A) strong and dynamic
119:
120: 7.      (B) strong and static
121:
122: 8.      (D) ||
123:
124: 9.      (B) (cadr '(1 2 3))
125:
126: 10.     (A) \lambda-calculus
127:
128: 11.     (C) loops
129:
130: 12.     (B) goto
131:
```