```
$Id: 2012q1-soln3,v 1.1 2012-03-16 18:50:59-07 - - $
Answers to 2012a1-test3, page 1

Note: answers which are correct, but different from the key,
still get full points.

_____________________________________________________________________
Question 1. [3]

let rec split pred list = match list with
    [] -> [], []
    | head::tail ->
      let (out1, out2) = split pred tail
      in  if pred head then (head::out1), out2
                       else out1, (head::out2)

_____________________________________________________________________
Question 2. [2]

filter( _, [], []).
filter( P, [H|T], [H|U]) :- call( P, H), filter( P, T, U).
filter( P, [_|T], U) :- filter( P, T, U).

_____________________________________________________________________
Question 3. [3]

(define (maxzip p l1 l2)
    (if (or (null? l1) (null? l2)) '()
        (let ((c1 (car l1))
              (c2 (car l2)))
         (if (p c1 c2) (cons c1 (maxzip p (cdr l1) (cdr l2)))
                       (cons c2 (maxzip p (cdr l1) (cdr l2)))))))

_____________________________________________________________________
Question 4. [2]

$lc++, $cc+= length, $wc+= @{[m/(\S+)/g]} while <>;
print "$lc $wc $cc\n";
```

```
40:
41: Answers to 2012a1-test3, page 2
42:
43: ____________________________________________________________________
44: Question 5. [1]
45:
46: universal: parametric (generic, template)
47:            inclusion (subclassing, object-oriented)
48:
49: ad hoc: conversion (coercion)
50:         overloading
51:
52: ____________________________________________________________________
53: Question 6. [2]
54:
55: (define (exclude n list)
56:     (if (or (<= n 0) (null? list)) list
57:         (exclude (- n 1) (cdr list))))
58:
59: ____________________________________________________________________
60: Question 7. [2]
61:
62: let rec exclude n list = match list with
63:     | _::tail when n > 0 -> exclude (n - 1) tail
64:     | _ -> list
65:
66: ____________________________________________________________________
67: Question 8. [2]
68:
69: exclude( _, [], []).
70: exclude( N, L, L) :- N =< 0.
71: exclude( N, [H|T], U) :- M is N - 1, exclude( M, T, U).
72:
73: ____________________________________________________________________
74: Question 9. [3]
75:
76: Object subclass: Find [
77:    Find class >> key: key array: array [
78:       1 to: array size do: [:index|
79:          (array at: index) = key ifTrue: [^ index]
80:       ].
81:       ^ nil.
82:    ]
83: ]
84:
```

```
 85:
 86: Answers to 2012a1-test3, page 3
 87:
 88: _____________________________________________________________________
 89: Question 10. [2]
 90:
 91:    static class say implements Runnable {
 92:       public void run () {
 93:          System.out.println ("Hello");
 94:       }
 95:    }
 96:
 97: _____________________________________________________________________
 98: Question 11. [2]
 99:
100: Stack s = new Stack.
101: t = s.pop(); // But pop does not set the pointer of the
102: underlying array to null.  So the array continues to point
103: at the object popped, even though it shouldn't.  It is
104: thus reachable but dead.
105: -- Many other possible explanations.
106:
107: _____________________________________________________________________
108: Question 12. [6]
109:
110: Object subclass: Stack [
111:    |array top|
112:    Stack class >> new [
113:       ^ Stack new: 10
114:    ]
115:    Stack class >> new: size [
116:       ^ super new init: size
117:    ]
118:    init: size [
119:       top := 0.
120:       array := Array new: size.
121:    ]
122:    pop [
123:       |result|
124:       result := array at: top.
125:       top := top - 1.
126:       ^ result.
127:    ]
128:    push: item [
129:       top := top + 1.
130:       array at: top put: item
131:    ]
132:    empty [
133:       ^ top = 0.
134:    ]
135: ]
136:
```

```
137:
138: Answers to 2012a1-test3, page 4
139:
140:  1.    (C) [4 + 5] value.
141:
142:  2.    (C) 2 sqrt
143:
144:  3.    (C) 'a list -> 'a list
145:
146:  4.    (C) structural
147:
148:  5.    (D) zombie
149:
150:  6.    (C) $_
151:
152:  7.    (D) map
153:
154:  8.    (B) foldl
155:
156:  9.    (B) my @a;
157:
158: 10.    (A) int *f() {int i = 6; return &i; }
159:
160: 11.    (C) throw
161:
```

```
162:
163: Answers to 2012a1-test3, page 5
164:
165:  1. overloading
166:
167:  2. overriding
168:
169:  3.    (A) currying
170:
171:  4.    (D) virtual function table
172:
173:  5.    (A) array
174:
175:  6.    (B) monad
176:
177:  7.    (B) Ocaml
178:
179:  8.    (D) p + p
180:
181:  9.    (C) heap
182:
183: 10.    (A) daemon
184:
185: 11.    (A) Algol 60
186:
```