$Id: solution-2017q4-midterm.txt,v 1.5 2017-11-08 13:01:48-08 - - $
Solution to cmps112-2017q4-midterm, page 1

Question 1(a). [2]

```
let sum list =
    let rec sum' list acc = match list with
        | [] -> acc
        | x::xs -> sum' xs (x + acc)
    in sum' list 0
```

... deduct 1 point if correct, but not tail recursive

Question 1(b). [2]

```
let rec fold_left fn unit list = match list with
    | [] -> unit
    | x::xs -> fold_left fn (fn unit x) xs
```

... deduct 1 point if correct, but not tail recursive

Question 1(c). [2]

```
let sumf = fold_left (+) 0
```

Question 2. [2]

```
(define (reverse list)
        (define (rev in out)
                (if (null? in) out
                    (rev (cdr in) (cons (car in) out))))
        (rev list '()))
```

... deduct 1 point if correct, but not tail recursive
ALTERNATE:

```
(define (reverse list)
        (foldl (lambda (a d) (cons a d)) '() list))
```

... add 1 bonus point if uses foldl, and if used CORRECTLY.

Question 3. [2]

```
(define (map f list)
        (if (null? list) '()
            (cons (f (car list)) (map f (cdr list)))))
```

Solution to cmps112-2017q4-midterm, page 2

Question 4. [2]

```
fac   int -> int                    Grading:
n     int                           9 correct -> 2 points
fac'  int -> int -> int             8 or 7 correct -> 1.5 points
n'    int                           6 or 5 correct -> 1 point
a'    int                           4 or 3 correct -> 0.5 points
<=    'a -> 'a -> bool              2 or fewer correct -> 0 points
1     int
-     int -> int -> int
*     int -> int -> int
```

Question 5(a). [2]

```
(define (sum list)
        (define (summ list acc)
                (if (null? list) acc
                    (summ (cdr list) (+ (car list) acc))))
        (summ list 0))
```

... deduct 1 point if correct, but not tail recursive

Question 5(b). [2]

```
(define (fold_left fn unit list)
        (if (null? list) unit
            (fold_left fn (fn unit (car list)) (cdr list))))
```

... deduct 1 point if correct, but not tail recursive

Question 5(c). [2]

```
(define (sumf list)
        (fold_left + 0 list))
```

Question 6(a). [2]

```
let rec evenlen list = match list with
    | [] -> true
    | [_] -> false
    | car::cadr::cddr -> evenlen cddr
```

Question 6(b). [2]

```
let evenlen = List.fold_left (fun t _ -> not t) true
```

Solution to cmps112-2017q4-midterm, page 3

Question 7. [2]

universal - parametric (or template or generic)
universal - inclusion (or oop)
ad hoc - conversion
ad hoc - overloading
... assign 1/2 point for each pair (left and right column)
... that are correct

Question 8. [2]

```
let reverse list =
    let rec rev inl outl = match inl with
        | [] -> outl
        | x::xs -> rev xs (x::outl)
    in rev list []
```

... deduct 1 point if correct, but not tail recursive
ALTERNATE:

```
let reverse = List.fold_left (fun t h -> h::t) [];;
```

... add 1 bonus point if uses foldl, and if used CORRECTLY.

Question 9. [2]

```
node reverse (node head) {
   node out = null;
   while (head != null) {
      node t = head;
      head = head.link;
      t.link = out;
      out = t;
   }
}
```

Question 10. [4]

```
let collatz n =
    let rec collatz' n rest =
        if n <= 1
           then 1::rest
           else if n mod 2 = 0
                   then collatz' (n / 2) (n::rest)
                   else collatz' (n * 3 + 1) (n::rest)
    in  List.rev (collatz' n [])
```

... deduct 1 point if correct, but collatz fn not tail recursive

```

Solution to cmps112-2017q4-midterm, page 4

 1.    (C) \lambda-calculus

 2.    (A) strong and dynamic

 3.    (B) strong and static

 4.    (A) ALGOL 60

 5.    (D) $ O ( 2 sup n ) $

 6.    (A) $ O ( n ) $

 7.    (A) (apply + ’(1 2 3 4))

 8.    (B) Edsger Dijkstra

 9.    (D) only M, but neither D nor U.

10.    (D) int -> int -> int

11.    (D) (cddr ’(1 2 3 4))

12.    (B) $x$ is bound and $y$ is free.

```