`$Id: cmps112-2018q1-midterm.mm,v 1.59 2018-02-05 13:29:32-08 - - $`

page 1	page 2	page 3	page 4	Total / 42

Please print clearly :

Name :
CruzID : `@ucsc.edu`

No books ; No calculator ; No computer ; No email ; No internet ; No notes ; No phone. Do your scratch work elsewhere and enter only your final answer into the spaces provided. Points will be deducted for messy answers. Unreadable answers will be presumed incorrect.

1. Without using any higher-order functions, code the function **`reverse`**, which will reverse a list.

   (a) ***Scheme.*** **[2✔]**

```
> (reverse '(1 2 3 4))
(4 3 2 1)
```

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

   (b) ***Ocaml.*** **[2✔]**

```
# reverse [1;2;3;4];;
- : int list = [4; 3; 2; 1]
```

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

2. ***Ocaml.*** What is the output from each of the following ? **[2✔]**

`reverse;;`	val reverse : 'a list -> 'a list = <fun>
`(*);;`	Warning: this is the start of a comment
`1::2::3::[];;`	int list = [1; 2; 3]

`let car = function x::_ -> x`
`      | _ -> failwith "car";;`
	val car : 'a list -> 'a = <fun>

3. Without using any higher-order functions, code the function **`fold_left`**, which will take a function, a unit, and a list, and fold the list.

   (a) ***Scheme.*** **[2✔]**

```
> (fold_left + 0 '(1 2 3))
6
```

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

   (b) ***Ocaml.*** **[2✔]**

```
# fold_left;;
- : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
# fold_left (+) 0 [1;2;3];;
- : int = 6
```

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

4. Without using any higher-order functions, code the function **find**, which will return a value associated with a given key. Indicate not found as shown here. Use the sample interactions to figure out the structure and arguments to this function.

   (a) ***Ocaml.*** **[3✔]**

```
# find;;
- : ('a -> 'b -> bool) -> 'a -> ('b * 'c) list -> 'c option = <fun>
# find (=) 3 [(1,2);(3,4);(5,6)];;
- : int option = Some 4
# find (=) 3 [(5,6);(7,8)];;
- : int option = None
```

```
let rec find cmp key list = match list with
  | [] -> None
  | (k,v)::xs -> if cmp key k then Some v
                 else find cmp key xs;;
```

   (b) ***Scheme.*** Use **cond** and not **if**. Return **#f** if not found. **[3✔]**

```
> (find = 3 '((1 2) (3 4) (5 6)))
4
> (find = 3 '((5 6) (7 8)))
#f
```

```
(define (find cmp key list)
   (cond ((null? list) #f)
         ((cmp key (caar list)) (cadar list))
         (else (find cmp key (cdr list)))))
```

5. ***Ocaml.*** Define **merge** which takes a comparison function and two sorted lists and merges the two lists into a single sorted list. **[3✔]**

```
# merge;;
- : ('a -> 'a -> bool) -> 'a list -> 'a list -> 'a list = <fun>
# merge (>) [9;5;1] [4;8;2];;
- : int list = [9; 5; 4; 8; 2; 1]
# merge (<) [1;5;9] [2;4;7];;
- : int list = [1; 2; 4; 5; 7; 9]
# merge (<) [] [];;
- : 'a list = []
```

```
let rec merge less ls1 ls2 = match ls1, ls2 with
   | [], ls2 -> ls2
   | ls1, [] -> ls1
   | x::xs, y::ys ->
     if less x y then x :: merge less xs ls2
           else y :: merge less ls1 ys;;
```

6. ***Ocaml.*** Write a function **trim** which trims leading zeros from a list. **[1✔]**

```
# trim;;
- : int list -> int list = <fun>
# trim [1;2;3];;
- : int list = [1; 2; 3]
# trim [0;0;5];;
- : int list = [5]
# trim [0;0;0];;
- : int list = []
```

```
let rec trim list = match list with
   | [] -> []
   | 0::xs -> trim xs
   | list -> list;;
```

7. ***Ocaml.*** Define `sub′` according to the programming project. It takes two `int lists` representing multiprecise numbers and subtracts the second from the first. Assume the first number is not less than the second number. **[5✔]**

```
# sub′;;
- : int list -> int list -> int list = <fun>
# sub′ [1;2;3;4] [4;3;2;1];;
- : int list = [7; 8; 0; 3]
# sub′ [0;0;0;9] [2];;
- : int list = [8; 9; 9; 8]
# sub′ [3;3;3] [3;3;3];;
- : int list = []
```

```
let rec sub'' n1 n2 borrow = match n1, n2, borrow with
  | n1, [], 0 -> n1
  | n1, [], borrow -> sub'' n1 [borrow] 0
  | h1::t1, h2::t2, borrow ->
    let diff = h1 - h2 - borrow
    in  if diff < 0 then diff + 10 :: sub'' t1 t2 1
                 else diff :: sub'' t1 t2 0
  | _, _, _ -> failwith "sub''"
in trim (sub'' n1 n2 0);;
```

Multiple choice. To the ***left*** of each question, write the letter that indicates your answer. Write ***Z*** if you don't want to risk a wrong answer. Wrong answers are worth negative points. **[5✔]**

number of correct answers		× 1 =	= a
number of wrong answers		× ½ =	= b
number of missing answers		× 0 =	0
column total $c = \max(a - b, 0)$	5		= c

1. `((lambda (x) (* x x)) (+ 2 3))`
   (A) `(+ 2 3)`
   (B) `12`
   [✓ (C) `25`]
   (D) `arity mismatch`

2. In a garbage collected language like Java, which does not have a `free` function, which is possible ?
   (A) dangling pointers but not memory leak
   (B) memory leak and dangling pointers
   [✓ (C) memory leak but not dangling pointers]
   (D) neither memory leak nor dangling pointers

3. In C++ and Java, what kind of statement allows an immediate transfer of control from the current function up several levels, unwinding the function call stack ?
   (A) `break`
   (B) `goto`
   (C) `return`
   [✓ (D) `throw`]

4. Smalltalk's type system is :
   [✓ (A) strong and dynamic]
   (B) strong and static
   (C) weak and dynamic
   (D) weak and static

5. A partially parameterized function, such as `(+)` in Ocaml, is said to be :
   [✓ (A) curried]
   (B) raised
   (C) thunked
   (D) tupled

Multiple choice. To the ***left*** of each question, write the letter that indicates your answer. Write ***Z*** if you don't want to risk a wrong answer. Wrong answers are worth negative points. **[12✓]**

number of correct answers		× 1 =	= a
number of wrong answers		× ½ =	= b
number of missing answers		× 0 =	0
column total $c = \max(a - b, 0)$	12		= c

1. In an eager functional language, which function can not be written in a tail-recursive manner without reversing a list?
   (A) fibonacci
   (B) fold_left
   [✓ (C) fold_right]
   (D) length

2. What will print:
   `5`
   (A) `((lambda (a b) (+ a b)) '(2 3))`
   [✓ (B) `((lambda (a b) (+ a b)) 2 3)`]
   (C) `(lambda (a b) (+ a b)) (2 3)`
   (D) `(lambda (a b) (+ a b)) 2 3`

3. Output of:
   `# (-);;`
   (A) `- : int * int * int = <fun>`
   (B) `- : int * int -> int = <fun>`
   (C) `- : int -> int * int = <fun>`
   [✓ (D) `- : int -> int -> int = <fun>`]

4. Which language has a strong dynamic type system?
   (A) C and C++
   (B) Java
   (C) Ocaml
   [✓ (D) Scheme]

5. What will print:
   `(1 2 7)`
   (A) `'(1 2 ,(+ 3 4))`
   (B) ``,(1 2 `(+ 3 4))``
   (C) ``` `(1 2 '(+ 3 4)) ```
   [✓ (D) ``` `(1 2 ,(+ 3 4)) ```]

6. What is the running time of:
```
let rec fib n =
    if n <= 1
        then n
        else fib (n - 1) + fib (n - 2);;
```
   (A) $O(n)$
   (B) $O(n \log n)$
   (C) $O(n^2)$
   [✓ (D) $O(2^n)$]

7. What are the first two characters in a Unix script file?
   [✓ (A) `#!`]
   (B) `%!`
   (C) `/*`
   (D) `$<`

8. What will print:
   `- : float = 5.`
   (A) `(+ 2. 3.);;`
   (B) `(+) 2. 3.;;`
   (C) `2. + 3.;;`
   [✓ (D) `2. +. 3.;;`]

9. What are the operator precedences is Smalltalk, with the highest at the left and lowest at the right?
   (A) binary, keyword, unary
   (B) keyword, binary, unary
   (C) keyword, unary, binary
   [✓ (D) unary, binary, keyword]

10. Of these languages, which one is the oldest?
    (A) Algol 60
    (B) C++
    [✓ (C) Fortran]
    (D) Java

11. What produces **x** itself as a result of this expression, assuming **x** is a list of at least one element?
    [✓ (A) `(cons (car x) (cdr x))`]
    (B) `(cons (cdr x) (car x))`
    (C) `(list (car x) (cdr x))`
    (D) `(list (cdr x) (car x))`

12. Ocaml's type system is:
    (A) strong and dynamic
    [✓ (B) strong and static]
    (C) weak and dynamic
    (D) weak and static