$Id: cmps112-2018q4-midterm.mm,v 1.118 2018-11-05 15:37:38-08 - - $

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. ***Smalltalk.*** Write a program that prints the numbers 1 through 1000 inclusive, one number per line. **[1✔]**

```
1 to: 1000 do: [:i | stdout << i << Character nl].
```

2. Define **sum** using fold left, not recursion.

   (a) ***Ocaml.*** See the next page for the type of fold left. **[1✔]**

```
let sum = List.fold_left (+) 0
```

   (b) ***Scheme.*** The arguments to **foldl** are in the same order. **[1✔]**

```
(define (sum list) (foldl + 0 list))
```

3. Define the Fibonacci function **fib** recursively so that it runs in $O(n)$ time for argument n.
   Mathematically : $F_0 = 0$; $F_1 = 1$; $F_n = F_{n-1} + F_{n-2}$.

   (a) ***Ocaml.*** **[2✔]**

```
let fib n =
   let rec fib' n a b =
      if n = 0 then a
               else fib' (n - 1) b (a + b)
   in fib' n 0 1;;
```

   (b) ***Scheme.*** **[2✔]**

```
(define (fib n)
    (define (fibb n a b)
        (if (= n 0) a
            (fibb (- n 1) b (+ a b))))
    (fibb n 0 1))
```

4. ***C.*** Define Fibonacci using Binet's formula. Temporary local values are **double**. **[1✔]**

$$F_n = \frac{\phi^n - \psi^n}{\sqrt{5}}$$

where $\phi = \frac{1+\sqrt{5}}{2}$ and $\psi = \frac{1-\sqrt{5}}{2}$

```
long fib (long n) [
double root5 = sqrt (5);
double phi = (1 + root5) / 2;
double psi = (1 - root5) / 2;
double fib = (pow (phi, n) - pow (psi, n)) / root5;
return round (fib);
```

5. ***Smalltalk.*** Define a block **sum2** which takes a single **value:** array argument and returns its sum. Use the **to:do:** message to iterate over the array and use **at:** to select individual array elements. **[2✔]**

```
st> sum2.
a BlockClosure
st> sum2 value: #(10 20 30 40 50).
150
st> sum2 value: #().
0
```

```
sum2 := [:a|
  |s| s := 0.
  1 to: a size do: [:i| s := s + (a at: i)].
  s.
].
```

6. ***Java.*** For the example of polymorphism in each of the boxes, describe it using one of the following words: ***conversion***, ***inclusion***, ***overloading***, ***parametric***. Also, in each box write the more general classification using one of the words: ***ad hoc***, ***universal***. Score: 1 point if both words in all four boxes are correct, 1/2 point if both words in three of the boxes are correct, else 0. **[1✔]**

`class bar extends foo {`
`  ...`
`}`
universal, inclusion
	`class stack<Item> {`
	`  ...`
	`}`
	universal, parametric

`void f (int i) {`
`}`
`void f (String s) {`
`}`
ad hoc, overloading
	`int value = 6;`
	`double sq = Math.sqrt (value);`
	ad hoc, conversion

7. Without using a higher-order function, and being sure to use tail recursion where required, define the function **`max`** which takes an inequality operator and a list and returns its maximum value. In each case return a special value (see below) for an empty list and use an inner worker function otherwise. See examples.

   (a) ***Scheme.*** Return **`#f`** for an empty list. **[3✔]**

```
> (max > '(3 1 4 1 5 9 2 6 5 3 5))
9
> (max < '(3 1 4 1 5 9 2 6 5 3 5))
1
> (max string>? '("foo" "bar" "baz"))
"foo"
> (max < '())
#f
```

```
(define (max gt list)
  (define (maxx big listt)
    (if (null? listt) big
      (let ((a (car listt))
            (d (cdr listt)))
        (if (gt big a) (maxx big d)
                       (maxx a d)))))
  (if (null? list) #f
    (maxx (car list) (cdr list))))
```

   (b) ***Ocaml.*** Return **`None`** for an empty list and a **`Some`** otherwise. **[2✔]**

```
# max;;
- : ('a -> 'a -> bool) -> 'a list -> 'a option = <fun>
# max (>) [3;1;4;1;5;9;2;6];;
- : int option = Some 9
# max (<) [3;1;4;1;5;9;2;6];;
- : int option = Some 1
# max (>) [];;
- : 'a option = None
```

```
let max gt list = match list with
  | [] -> None
  | x::xs -> let rec max' x xs = match xs with
               | [] -> x
               | y::ys -> max' (if gt x y then x else y) ys
             in Some (max' x xs);;
```

8. ***Ocaml.*** Define **`maxfl`**, as above, using the function **`List.fold_left`**. Do not use recursion. **[2✔]**

```
# List.fold_left;;
- : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>
# maxfl;;
- : ('a -> 'a -> bool) -> 'a list -> 'a option = <fun>
# maxfl (<) [2;7;8;1;8;2;8];;
- : int option = Some 1
# maxfl (>) ["foo";"bar";"baz"];;
- : string option = Some "foo"
# maxfl (<) [];;
- : 'a option = None
```

```
let maxfl gt list = match list with
  | [] -> None
  | x::xs -> Some (List.fold_left
               (fun n m -> if gt n m then n else m)
               x xs);;
```

9. ***Ocaml.*** Define **`fold_left`**. **[2✔]**

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

10. ***Ocaml.*** Fill in the column at the right with the response given by the interactive toploop when the entry at the left is given. **[2✔]**

`List.map;;`	`('a -> 'b) -> 'a list -> 'b list`
`(+) 1;;`	`int -> int`
`1::2::3::[];;`	`int list = [1; 2; 3]`
`let car = function x::_ -> x` `| _ -> failwith "car";;`	`val car : 'a list -> 'a`

11. Define the function **`zipwith`**, which takes a function and two lists and uses the function to merge the two lists into one. Ensure the lists are the same length without using a length function. See the examples for expected results.

    (a) ***Scheme.*** Return the first parameter if the lists are of different lengths, otherwise return the expected result. **[3✔]**

```
> (zipwith #f + '(1 2 3 4) '(5 6 7 8))
(6 8 10 12)
> (zipwith #f - '(1 2 3) '(4))
#f
> (zipwith #f * '() '())
()
> (zipwith #f / '(1 2 3) '(4 5 6))
(1/4 2/5 1/2)
```

```
(define (zipwith error fn list1 list2)
  (cond ((null? list1) (if (null? list2) '() error))
        ((null? list2) error)
        (else (let ((tail (zipwith error fn (cdr list1) (cdr list2))))
                (if (eqv? error tail) error
                    (cons (fn (car list1) (car list2)) tail))))))
```

    (b) ***Ocaml.*** Use `failwith "zipwith"` if the lists are if different lengths. **[3✔]**

```
# zipwith;;
- : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list = <fun>
# zipwith (+) [1;2;3;4] [5;6;7;8];;
- : int list = [6; 8; 10; 12]
# zipwith (-) [1;2;3] [4];;
Exception: Failure "zipwith".
# zipwith ( * ) [] [];;
- : int list = []
# zipwith (/.) [1.;2.;3.] [4.;5.;6.];;
- : float list = [0.25; 0.4; 0.5]
```

```
let rec zipwith fn list1 list2 = match list1, list2 with
  | [], [] -> []
  | _, [] -> failwith "zipwith"
  | [], _ -> failwith "zipwith"
  | x::xs, y::ys -> fn x y :: zipwith fn xs ys;;
```

12. ***C.*** Write a function to reverse a list. Do not use **`malloc`** or **`free`**. Do not assign to any value field. Make assignment only to the **`link`** fields of the nodes. The function is destructive of the original list. Return a pointer to the reversed list. **[2✔]**

```
struct node {
   int value;
   struct node* link;
};
```

```
struct node* reverse (struct node* head) {
    struct node* newlist = NULL;
    while (head != NULL) {
      struct node* tmp = head;
      head = head->link;
      tmp->link = newlist;
      newlist = tmp;
    }
    return newlist;
```

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. ***Unix.*** What is the symbolic name for a user's home directory ?
   (A) `$`
   (B) `.`
   (C) `/`
   (D) `~`

2. ***Unix.*** What keystroke indicates end of file at the Unix command line ?
   (A) `^C`
   (B) `^D`
   (C) `^H`
   (D) `^Z`

3. ***Scheme.*** What will indicate to the parent process that an error has occurred ?
   (A) `(exit 0)`
   (B) `(exit 1)`
   (C) `exit(0);`
   (D) `exit(1);`

4. ***Unix.*** How might you locate a file called `foobar.sbir` ?
   (A) `echo $foobar.sbir | grep --plain-file`
   (B) `find $cmps-112 -name foobar.sbir`
   (C) `for i in foobar.sbir do; echo $i; done`
   (D) `grep foobar.sbir *`

5. ***Unix.*** Which environment variable must be adjusted in order to find the location of programs that are not in the standard directories ?
   (A) `$HOME`
   (B) `$PATH`
   (C) `$TERM`
   (D) `$USER`

6. Which of the following will not be written using tail recursion ?
   (A) Computing a Fibonacci number.
   (B) Finding the sum of all elements in a list.
   (C) Mapping a unary function onto a list.
   (D) Reversing a list.

7. ***Smalltalk.*** What is 7 ?
   (A) `(3 + 4) value.`
   (B) `7 value.`
   (C) `[3 + 4] value.`
   (D) `[3 + 4] value: 7.`

8. ***Scheme.*** What is `((lambda (x) x) (+ 2 3))` ?
   (A) `(+ 2 3)`
   (B) `+`
   (C) `5`
   (D) `x`

9. ***Scheme.*** What is `(3)` ?
   (A) `(caar '(1 2 3))`
   (B) `(cadr '(1 2 3))`
   (C) `(cdar '(1 2 3))`
   (D) `(cddr '(1 2 3))`

10. ***Smalltalk.*** What is 1.4142135623730951 ?
    (A) `(sqrt 2.0)`
    (B) `2.0 ** 0.5`
    (C) `2.0 sqrt`
    (D) `sqrt 2.0`

11. ***Ocaml.*** What is the type of `(+)` ?
    (A) `int * int * int`
    (B) `int * int -> int`
    (C) `int -> int * int`
    (D) `int -> int -> int`

12. ***Ocaml.*** Type checking is :
    (A) strong and dynamic
    (B) strong and static
    (C) weak and dynamic
    (D) weak and static