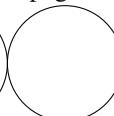
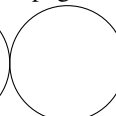


\$Id: cmps112-2019q1-final.mm,v 1.165 2019-03-14 12:47:31-07 - - \$

page 1	page 2	page 3	page 4	page 5	Total / 54	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. Answer only in the spaces provided on the exam paper. Points will be deducted for answers not in the spaces provided. Points will be deducted for messy answers. Unreadable answers will be presumed incorrect.

1. *Ocaml*. Define the following functions.

(a) [2✓]

```
# map;;
- : ('a -> 'b) -> 'a list -> 'b list = <fun>

let rec map f list = match list with
| [] -> []
| x::xs -> f x :: map f xs
```

(b) [2✓]

```
# filter;;
- : ('a -> bool) -> 'a list -> 'a list = <fun>

let rec filter f list = match list with
| [] -> []
| x::xs -> if f x then x :: filter f xs
           else filter f xs
```

(c) [2✓]

```
# fold_left;;
- : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a = <fun>

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

2. *Scheme*. Define the following functions. Each has an example of a call.

(a) [2✓]

```
> (map / '(1. 2. 3. 4. 5. 6.))
(1.0 0.5 0.3333333333333333 0.25 0.2 0.16666666666666666)

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

(b) [2✓]

```
> (filter (lambda (x) (> x 0)) '(1 2 -3 -4 5 -6))
(1 2 5)

(define (filter p? list)
  (if (null? list) '()
      (let ((a (car list))
            (d (cdr list)))
        (if (p? a) (cons a (filter p? d))
            (filter p? d)))))
```

3. Code a linear search **findpred**, given a predicate and a sequence of pairs, return the value associated with the first key that the predicate matches. Determine the not-found value from the examples. Do not use any higher-order functions. Follow the general assumptions about the particular language for which the answer is required.

- (a) **Smalltalk**. Extend class **Array**. Search an array of arrays. The inner arrays are all key and value pairs. Return **nil** if not found. [2✓]

```
st> pos := [:x | x > 0].
st> two := [:x | x = 2].
st> a := #((-3 6) #(-9 8) #(10 100) #(2 20)).
st> b := #((1 2) #(3 4) #(5 6)).
st> a findpred: pos.
100
st> b findpred: two.
nil
```

```
Array extend [
  findpred: pred [
    self do: [:a]
      (pred value: (a at: 1)) ifTrue: [^ a at: 2].
    ].
    ^ nil.
  ]
].
```

- (b) **Perl**. Return **undef** if not found. [2✓]

```
Source:
sub pos {$_[0]>0};
sub two {$_[0]==2};
sub show { my ($a) = @_;
  print defined($a) ? "$a\n" : "NONE\n";
}
show findpred \&pos, ([-3,6],[-9,8],[4,7],[10,0]);
show findpred \&two, ([1,2],[3,4],[5,6]);
Output:
7
NONE
```

```
sub findpred ($@) {
  my ($pred, @array) = @_;
  for my $pair (@array) {
    return $pair->[1] if $pred->($pair->[0]);
  }
  return undef;
}
```

- (c) **Prolog**. The term **call(P,K)** will call **P** with the argument **K**. [2✓]

Rules:

```
pos(X) :- X > 0.
two(X) :- X == 2.
```

Queries:

```
| ?- findpred(pos, [pair(-3,6),pair(-9,8),pair(4,7),pair(10,0)], X).
X = 7 ? ;
X = 0 ? ;
no
| ?- findpred(two, [pair(1,2),pair(3,4),pair(5,6)], X).
no
```

```
findpred(P, [pair(K,V)|_], V) :- call(P,K).
findpred(P, [_|L], V) :- findpred(P, L, V).
```

- (d) **Ocaml**. [2✓]

```
# findpred;;
- : ('a -> bool) -> ('a * 'b) list -> 'b option = <fun>
# let pos x = x > 0;;
# let two x = x = 2;;
# findpred pos [-3,6;-9,8;4,7;10,0;2,20];;
- : int option = Some 7
# findpred two [1,2;3,4;5,6];;
- : int option = None
```

```
let rec findpred p list = match list with
| [] -> None
| (x,y)::zs -> if p x then Some y
  else findpred p zs
```

- (e) **Scheme**. The first argument to **findpred** is the value to be returned if not found. [2✓]

```
> (define (pos x) (> x 0))
> (define (two x) (= x 2))
> (findpred 9999 pos '((-3 6) (-9 8) (10 100) (2 20)))
100
> (findpred #f two '((1 2) (3 4) (5 6)))
#f
```

```
(define (findpred none p list)
  (cond ((null? list) none)
        ((p (caar list)) (cadar list))
        (else (findpred none p (cdr list)))))
```

4. **Make.** Write a complete **Makefile** that will compile an arbitrary Java program into the appropriate class file. Assume the entire Java program is contained in a single file. For example, **make foo.class** should build the class file using the command **java foo.java**. Use proper **make** variables so that any Java program can be compiled, not just one called **foo**. [1✓]

```
%.class : %.java
javac $<
```

5. Without using any higher-order functions, define **oddden** which determines if a list has an odd number of elements or not.

(a) **Ocaml.** [2✓]

```
# oddlen [];;
- : bool = false
# oddlen [1;2;3;4];;
- : bool = false
# oddlen [1;2;3;4;5];;
- : bool = true

let rec oddlen list = match list with
| [] -> false
| [_] -> true
| _::_:xs -> oddlen xs
```

(b) **Scheme.** Use **cond**, not **if**. [2✓]

```
> (oddden '())
#f
> (oddden '(1 2 3 4))
#f
> (oddden '(1 2 3 4 5))
#t

(define (oddden list)
  (cond ((null? list) #f)
        ((null? (cdr list)) #t)
        (else (oddden (cddr list)))))
```

(c) **Prolog.** Succeeds or fails. [1✓]

```
| ?- oddlen([]).
no
| ?- oddlen([1,2,3,4]).
no
| ?- oddlen([1,2,3,4,5]).
true ? ;
no

oddden(_).
oddden(_;_) :- oddlen(T).
```

6. **Smalltalk.** Define classes **Animal**, **Cat**, and **Dog**.

- (a) Class **Animal** has an instance field called **name**. Its class method **new** sets the name. Its has instance methods **setName**, which sets the name, and **name** which returns the name. [3✓]
- (b) Class **Cat** is a subclass of **Animal** and has an instance method **noise** which returns the string **'meow'**. Class **Dog** is a subclass of **Animal** and has an instance method **noise** which returns the string **'woof'**. [1✓]

Object subclass: Animal [

```
|name|
Animal class >> new: name_ [
|result|
result := super new.
result setName: name_.
^ result
]
setName: name_ [ name := name_ ]
name [ ^ name ]
].
```

Animal subclass: Cat [

```
noise [ ^ 'meow' ]
].
Animal subclass: Dog [
noise [ ^ 'woof' ]
].
```

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		$\times 1 =$	$= a$
number of wrong answers		$\times \frac{1}{2} =$	$= b$
number of missing answers		$\times 0 =$	0
column total $c = \max(a - b, 0)$	12		$= c$

1. **Make**. What wildcard dependency might be used to specify how to compile a C program?

(A) `%.c: %.o`
 (B) `%.o: %.c`
 (C) `*.c: *.o`
 (D) `*.o: *.c`

2. **Perl**. What will print `foo`?

(A) `print "foo" if 0;`
 (B) `print "foo" if FALSE;`
 (C) `print "foo" if "";`
 (D) `print "foo" if undef;`

3. **Ocaml**. What kind of polymorphism is represented by:

```
# List.length;;
- : 'a list -> int = <fun>
```

(A) conversion
 (B) inclusion
 (C) overloading
 (D) parametric

4. **Ocaml**. What term could be used to describe the function `(+)`?

(A) `curried`
 (B) dynamically typed
 (C) overloaded
 (D) polymorphic

5. **Perl**. The first line of a script.

(A) `#!/usr/bin/perl`
 (B) `(*usr/bin/perl*)`
 (C) `//GO.SYSIN DD PERL`
 (D) `chmod ugo+x perl`

6. Normal order evaluation is also known as:

(A) dynamic dispatch
 (B) interpretive execution
 (C) `lazy evaluation`
 (D) pass by reference

7. **Smalltalk**. What is the order of evaluation of messages, listed in order with highest precedence at the left to lowest precedence at the right?

(A) binary then keyword then unary
 (B) binary then unary then keyword
 (C) keyword then binary then unary
 (D) keyword then unary then binary
 (E) `unary then binary then keyword`
 (F) unary then keyword then binary

8. **Smalltalk**. What is the value of an uninitialized variable?

(A) `nil`
 (B) `nullptr`
 (C) `undef`
 (D) There is no way of knowing because it depends on whatever random bits happen to be in that memory location.

9. **Java**. What kind of polymorphism is this?

```
interface foo {
    void bar (int x);
    void bar (String x);
}
```

(A) conversion
 (B) inclusion
 (C) `overloading`
 (D) parametric

10. **Bash**. What will run the program `foo`, redirect both its stdout and stderr to the file `bar`?

(A) `foo <bar 1>&2`
 (B) `foo <bar 2>&1`
 (C) `foo >bar 1>&2`
 (D) `foo >bar 2>&1`

11. **Bash**. What shell variable had to be exported via your profile in order to make access the the binaries of the various languages convenient?

(A) `BINDIR`
 (B) `PATH`
 (C) `SHELL`
 (D) `USER`

12. What Java and C++ statement is equivalent to Fortran's and PL/1's non-local goto, which is able to transfer control from one function into the calling context of the function that called it?

(A) `break`
 (B) `catch`
 (C) `continue`
 (D) `throw`

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		$\times 1 =$	$= a$
number of wrong answers		$\times \frac{1}{2} =$	$= b$
number of missing answers		$\times 0 =$	0
column total $c = \max(a - b, 0)$	12		$= c$

1. *Perl*. If a child process is terminated by a signal, what is the signal number?
(A) `$? & 127`
(B) `$? << 8`
(C) `$? >> 8`
(D) `$? | 127`
2. *Perl*. What is the exit status of a child process?
(A) `$? & 127`
(B) `$? << 8`
(C) `$? >> 8`
(D) `$? | 127`
3. *Ocaml*. How much stack space is used by `List.fold_left`, where n is the length of the list argument?
(A) $O(1)$
(B) $O(\log_2 n)$
(C) $O(n)$
(D) $O(n \log_2 n)$
4. *Ocaml*. How much stack space is used by `List.fold_right`, where n is the length of the list argument?
(A) $O(1)$
(B) $O(\log_2 n)$
(C) $O(n)$
(D) $O(n \log_2 n)$
5. *Ocaml*. `List.map`
(A) `('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`
(B) `('a -> 'b -> 'b) -> 'a list -> 'b -> 'b`
(C) `('a -> 'b) -> 'a list -> 'b list`
(D) `('a -> bool) -> 'a list -> 'a list`
6. What language uses “duck typing”?
(A) Java
(B) Ocaml
(C) Perl
(D) Prolog
(E) Scheme
(F) Smalltalk

7. What language uses normal order evaluation by default?
(A) Fortran
(B) Haskell
(C) Ocaml
(D) Prolog
8. *Perl*. What is used to extract an element using `$key` from a hash `%hash`?
(A) `%hash[$key]`
(B) `%hash{$key}`
(C) `$hash[$key]`
(D) `$hash{$key}`
9. *Scheme*. What is 24?
(A) `(apply '* '(1 2 3 4))`
(B) `(apply '* (1 2 3 4))`
(C) `(apply * '(1 2 3 4))`
(D) `(apply * (1 2 3 4))`
10. *Ocaml*. What is an `int list`?
(A) `(1,2,3,4)`
(B) `(1;2;3;4)`
(C) `[1,2,3,4]`
(D) `[1;2;3;4]`
11. *Ocaml*. What is `float = infinity`?
(A) `0./0.`
(B) `0./1.`
(C) `1./0.`
(D) `1./1.`
12. `Languages/prolog/Examples/einstein.pl`
The _____ owns the fish.
(This example was discussed during lecture, so you might have remembered this had you attended lecture.)
(A) brit
(B) dane
(C) german
(D) norwegian
(E) swede

