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. Points will be deducted for messy answers. Unreadable answers will be presumed incorrect. There are no questions about Haskell, Ada, Intercal, or Erlang.

1. ***Smalltalk.*** Define a block **sum** whose **value:** message returns the sum of the **Number**s of an array. **[2✔]**

```
st> sum1 value: #(1 2 3 4 5).
15
```

```
sum1 := [:a|        "one possible
answer"
   |s| s := 0.
   a do: [:n| s := s + n].
   s.
].
```

2. Without using higher-order functions, define **sum** which returns the sum of numbers in a list.

   (a) ***Ocaml.*** **[2✔]**

```
# sum;;
- : int list -> int = <fun>
# sum [1;2;3;4;5];;
- : int = 15
```

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

   (b) ***Scheme.*** **[2✔]**

```
> (sum '(1 2 3 4 5))
15
```

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

   (c) ***Prolog.*** **[2✔]**

```
| ?- sum([1,2,3,4,5],N).
N = 15
yes
```

```
sum([],0).
sum([H|T],N) :- sum(T,M), N is H + M.
```

3. Define **sum** as described above, without recursion, but using the left fold function.

   (a) ***Ocaml.*** **[1✔]**

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

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

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

```
> foldl
#<procedure:foldl>
```

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

4. Each box in the following table represents a kind of polymorphism. In each box, write two terms from the following list which describes that particular kind of polymorphism: ***ad hoc***, ***conversion***, ***inclusion***, ***overloading***, ***parametric***, ***universal***. **[2✔]**

`template <typename T>`
`class stack { }`
universal, parametric
	`int i, j; string s, t;`
	`i += j; s += t;`
	ad hoc, overloading

`class B extends A {`
`}`
universal, inclusion
	`double add (double a, double b);`
	`double x = add (3, 4);`
	ad hoc, conversion

5. ***Prolog.*** Assume a database with facts matching the following queries: `female(Person)`, `male(Person)`, `parent(Parent,Child)`. Define the following rules:

   (a) `sister(Sister,Sibling)` **[1✔]**

```
sister(X,X) :- !, fail.
sister(Sister,X) :- female(Sister), parent(Y,Sister), parent(Y,X).
```

   (b) `father(Father,Child)` **[1✔]**

```
father(Father,Child) :- male(Father), parent(Father,Child).
```

   (c) `grandmother(Grmother,Grchild)` **[1✔]**

```
grandmother(Grmother,Grchild) :-
      female(Grmother), Parent(Grmother,X), Parent(X,Grchild).
```

6. ***Smalltalk.*** Extend class `Array` with a keyword message `findpos:`, which searches an array for the first element in it that is equal to the argument. Return `nil` if not found. **[3✔]**

```
st> #(11 22 33 44 99 88 77 66) findpos: 88.
6
st> #(11 22 33 44 99 88 77 66) findpos: 102.
nil
```

```
Array extend [
  findpos: value [
    1 to: self size do: [:index|
      (self at: index) = value ifTrue: [^ index]
    ].
    ^ nil
  ]
].
```

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

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

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

8. ***Ocaml.*** Define a function `zip` which takes two lists as arguments and returns a list of tuples containing the corresponding elements. If the lists are of different lengths, ignore excess elements of the longer list. **[2✔]**

```
# zip;;
- : 'a list -> 'b list -> ('a * 'b) list = <fun>
# zip [1;2;3] [4;5;6];;
- : (int * int) list = [(1, 4); (2, 5); (3, 6)]
# zip [1;2;3;4;5;6;7;8] [3;6;9];;
- : (int * int) list = [(1, 3); (2, 6); (3, 9)]
```

```
let rec zip ls1 ls2 = match ls1, ls2 with
   | [], _ -> []
   | _, [] -> []
   | x::xs, y::ys -> (x,y) :: zip xs ys;;
```

9. ***Scheme.*** Define a function `zip` which takes two lists as arguments and returns a list of lists containing the corresponding elements. That is, for the inner lists, the `car` is taken from the first list, the `cadr` is from the second list, and the `cddr` is `'()`. Ignore excess elements from the longer list. **[3✔]**

```
> (zip '(1 2 3) '(4 5 6))
((1 4) (2 5) (3 6))
> (zip '(1 2 3 4 5 6 7 8) '(3 6 9))
((1 3) (2 6) (3 9))
```

```
(define (zip ls1 ls2)
      (if (or (null? ls1) (null? ls2)) '()
         (cons (list (car ls1) (car ls2))
              (zip (cdr ls1) (cdr ls2)))))
```

10. ***Prolog.*** Given the facts listed here, write a relation `chow_time` with a single list argument, and which will succeed if anything in the list will get eaten. Hint : The function `member(X,Y)` checks to see if the item `X` is a member of the list `Y`. Write the relation `chow_time`, which has a list argument and succeeds **[2✔]**

```
eats(fox,chicken).
eats(chicken,grain).
```

```
chow_time( List ) :-
  member( Diner, List ),
  member( Dinner, List ),
  eats( Diner, Dinner ).
```

11. ***Scheme.*** Write a function `filter` which takes a predicate and a list and returns a list whose elements are in the same order as the input list, but which contains only elements for which the predicate is true. Use a `let` form so that the functions `car` and `cdr` are not called more than once anywhere in the function. **[3✔]**

```
> (filter (lambda (x) (> x 0)) '(-3 44 72 -91 202 0 -34))
(44 72 202)
```

```
(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)))))
```

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. If a function's arguments are always evaluated before the function is called, that is ____ order evaluation.
   [✓ (A) applicative]
   (B) curried
   (C) normal
   (D) object-oriented

2. An access (static) link is needed in languages with :
   (A) a function call stack
   (B) inner classes
   (C) nested classes
   [✓ (D) nested functions]

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

4. `10`
   (A) `(apply '+ '(1 2 3 4))`
   (B) `(apply '+ (1 2 3 4))`
   [✓ (C) `(apply + '(1 2 3 4))`]
   (D) `(apply + (1 2 3 4))`

5. `# (<) 2;;`
   (A) `- : 'a -> bool = <fun>`
   (B) `- : bool -> 'a = <fun>`
   (C) `- : bool -> int = <fun>`
   [✓ (D) `- : int -> bool = <fun>`]

6. The structured program theorem says that only three programming constructs are necessary : sequence (`;`), conditional (`if`), and looping (`while`). This was proved by :
   [✓ (A) Corrado Böhm & Giuseppe Jacopini]
   (B) Donald Knuth
   (C) Edsger Dijkstra
   (D) Niklaus Wirth

7. Prolog uses ____ to set the values of variables.
   (A) pointer dereferencing
   (B) template instantiation
   (C) type inference
   [✓ (D) unification]

8. Java has [x] inheritance of classes and [y] inheritance of interfaces.
   (A) [x] = multiple, [y] = multiple
   (B) [x] = multiple, [y] = single
   [✓ (C) [x] = single, [y] = multiple]
   (D) [x] = single, [y] = single

9. What Perl statement will copy all input to the standard output ?
   (A) `print <> while;`
   [✓ (B) `print while <>;`]
   (C) `while <> print;`
   (D) `while print <>;`

10. Which C/C++/Java operator uses normal order evaluation ?
    [✓ (A) `&&`]
    (B) `++`
    (C) `--`
    (D) `==`

11. Grace Hopper, USN, lead the design team for which programming language ?
    (A) ALGOL 60
    (B) BASIC
    [✓ (C) COBOL]
    (D) FORTRAN

12. What language uses "duck typing" to determine method dispatch ?
    (A) C++
    (B) Java
    (C) Ocaml
    [✓ (D) Smalltalk]

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 Java, parametric polymorphism is implemented by:
   (A) recompiling functions from each class when they are instantiated.
   [✓ (B) representing all generic parameters as objects, and performing implicit casting operations.]
   (C) tagging the low-order bit of each field of a structure to distinguish pointers from other things.
   (D) using macro substitution when the preprocessor is run ahead of compilation.

2. When a garbage collector forms the closure of the root set, it identifies all ____ objects on the heap.
   (A) dead
   (B) live
   [✓ (C) reachable]
   (D) unreachable

3. What will make Smalltalk print 1.4142135623730951 ?
   (A) `(sqrt 2)`
   [✓ (B) `2 sqrt.`]
   (C) `X is sqrt(2).`
   (D) `sqrt 2.0;;`

4. Which function takes a function f and a list, applies f to every element of the list, and returns a new list of the same length whose values are $f(x)$?
   (A) filter
   (B) fold_left
   (C) fold_right
   [✓ (D) map]

5. What is the type of the argument of `f` in the Ocaml statement `let f () = 3`
   (A) `null`
   (B) `nullptr`
   [✓ (C) `unit`]
   (D) `void`

6. What might be a Prolog fact?
   (A) `foo(BAR,BAZ).`
   (B) `foo(BAR,baz).`
   (C) `foo(bar,BAZ).`
   [✓ (D) `foo(bar,baz).`]

7. If a function g is nested inside a function f, what does g need in order to refer to the local variables of f?
   (A) dynamic link
   (B) result register
   (C) return address
   [✓ (D) static link]

8. An Ocaml pattern match of `x::y::z` will match a list of at a minimum, ____ elements.
   (A) 1
   [✓ (B) 2]
   (C) 3
   (D) 4

9. What is `((lambda (x) x) (+ 2 3))`?
   (A) `(+ 2 3)`
   (B) `+`
   [✓ (C) `5`]
   (D) `x`

10. What is the parenthesized equivalent of the Smalltalk expression `a b c: d`?
    [✓ (A) `((a b) c: d)`]
    (B) `(a (b c: d))`
    (C) `(a b) (c: d)`
    (D) `a ((b c:) d)`

11. What is `2`?
    (A) `(caar (1 2 3))`
    [✓ (B) `(cadr (1 2 3))`]
    (C) `(cdar (1 2 3))`
    (D) `(cddr (1 2 3))`

12. Is half of two plus two equal to two or three?
    (A) two
    (B) three
    [✓ (C) yes]
    (D) no