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.

1. Define the function factorial. Note : $0\,! = 1$, $n\,! = n \times (n-1)\,!$, and is undefined for negative numbers.

   (a) ***Scheme***. Return `#f` for a negative number. Deduct 1 point if not tail recursive. **[2✔]**

```
> (fac 15)        > (fac 0)
1307674368000     1
> (fac 10)        > (fac -1)
3628800           #f
> (fac 5)
120
```

```
(define (fac n)
    (define (facc nn mm)
        (if (<= nn 1) mm
            (facc (- nn 1) (* nn mm))))
    (if (< n 0) #f (facc n 1)))
```

   (b) ***Ocaml***. Fail for a negative number. Deduct 1 point if not tail recursive. **[2✔]**

```
# fac;;                      # fac 5;;
- : int -> int = <fun>       - : int = 120
# fac 15;;                   # fac 0;;
- : int = 1307674368000      - : int = 1
# fac 10;;                   # fac (-1);;
- : int = 3628800            Exception: Failure
                             "fac n | n < 0".
```

```
let fac n =
   let rec fac' n' a' =
      if n' <= 1 then a'
              else fac' (n' - 1) (n' * a')
   in if n < 0 then failwith "fac n | n < 0"
              else fac' n 1
```

   (c) `Smalltalk`. Extend class `Number` to add a unary message `fac`. Return nil if negative. **[2✔]**

```
st> 15 fac.       st> 0 fac.
1307674368000     1
st> 10 fac.       st> -1 fac.
3628800           nil
st> 5 fac.
120
```

```
Number extend [
   fac [
      |f n|
      n := self.
      (n < 0) ifTrue: [^ nil]
           ifFalse: [
              f := 1.
              [n > 1] whileTrue: [ f := f * n. n := n - 1 ].
              ^ f
      ]
   ]
].
```

   (d) ***Prolog.*** Fail if negative. **[2✔]**

```
| ?- fac(15,M).              | ?- fac(5,M).
M = 1307674368000 ?          M = 120 ?
yes                          yes
| ?- fac(10,M).              | ?- fac(0,M).
M = 3628800 ?                M = 1 ?
yes                          yes
                             | ?- fac(-1,M).
                             no
```

```
fac(N,_) :- N < 0, !, fail.
fac(0,1).
fac(N,M) :- A is N - 1, fac(A,B), M is N * B.
```

2. ***Prolog.***

   (a) Define `sum`. **[1✔]**

```
| ?- sum([1,2,3,4,5],N).
N = 15
| ?- sum([],N).
N = 0
```

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

   (b) Define `length`. **[1✔]**

```
| ?- length([1,2,3,4,5],N).
N = 5
| ?- length([],N).
N = 0
```

```
length([],0).
length([_|T],N) :- length(T,M), N is M + 1.
```

3. Write a function that performs differentiation of a polynomial. For each term of the form kx^n, replace the term with the value knx^{n-1}. Any term with an exponent of 0 is lost. Assume all input exponents are non-negative integers. Examples: $\frac{d}{dx}\,4x^4+6x^3+7x^2+5x+8=16x^3+18x^2+14x+5$ and $\frac{d}{dx}\,4x^6+8x^3=24x^5+24x^2$

   (a) ***Ocaml.*** Represent a polynomial as a `(float*int) list`, where each float is the coefficient and each int is the corresponding exponent. **[2✔]**

```
# differentiate;;
- : (float*int) list ->
(float*int) list = <fun>
# differentiate
[(4.,4);(6.,3);(7.,2);(5.,1);(8.,0)];;
- : (float*int) list =
[(16.,3);(18.,2);(14.,1);(5.,0)]
# differentiate [(4.,6);(8.,3)];;
- : (float*int) list = [(24.,5);(24.,2)]
```

```
let rec differentiate list = match list with
   | [] -> []
   | (_,0)::xs -> differentiate xs
   | (coeff,expt)::xs -> (coeff *. float_of_int expt, expt - 1)
              :: differentiate xs;;
```

   (b) ***Scheme.*** Represent a polynomial as a list of lists of length 2. (See the examples.) The `car` of each sublist is the coefficient and the `cdr` is the exponent. Use a `let` to give meaningful names to the `caar`, `cadar`, and `cdr` of the input list. **[3✔]**

```
> (differentiate ’((4 4)(6 3)(7 2)(5 1)(8 0)))
((16 3)(18 2)(14 1)(5 0))
> (differentiate ’((4 6)(8 3)))
((24 5)(24 2))
```

```
(define (differ poly)
   (if (null? poly) '()
      (let ((coeff (caar poly))
            (expt (cadar poly))
            (rest (cdr poly)))
           (if (= 0 expt) (differ rest)
              (cons (list (* coeff expt) (- expt 1))
                    (differ rest))))))
```

4. ***Java.*** Write a class `counter` such that might be used in the observers and reporters problem. It has a private `long` field to represent a count, which has an initial value of 0. The method `click` increments the count. The method `reset` returns the current count and resets it to 0. Be sure that race conditions are not possible when its methods are called from multiple concurrent threads. **[3✔]**

```
class counter {
   private long count = 0;
   public boolean stop = false;
   synchronized void click () {
      ++count;
   }
   synchronized long reset () {
      long result = count;
      count = 0;
      return result;
   }
}
```

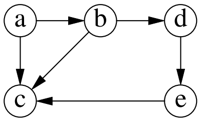
5. ***Ocaml.*** Define `merge` that takes a comparison function and two sorted lists and returns a list merged into sorted order. **[2✔]**

```
# merge;;
- : (’a -> ’a -> bool) -> ’a list -> ’a list -> ’a list = <fun>
# merge (<) [1;3;5;7;9] [2;4;6;8];;
- : int list = [1; 2; 3; 4; 5; 6; 7; 8; 9]
# merge (>) [99;88;77] [95;85;76];;
- : int list = [99; 95; 88; 85; 77; 76]
# merge (<) [1;2;3;4] [1;2;3;4];;
- : int list = [1; 1; 2; 2; 3; 3; 4; 4]
```

```
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. ***Prolog.***

   (a) Write a set of facts called `arrow` which describe this graph, where `arrow(a,b)` means that a→b. **[2✔]**



```
arrow(a,b).  ///The facts may be listed in any arbitrary order.
arrow(a,c).
arrow(b,c).
arrow(b,d).
arrow(d,e).
arrow(e,c).
```

   (b) Given `ispath` as defined here, write the relation `ispath(A,Visited,B)` which succeeds if there is a path from `A` to `B` and avoids infinite traversals around a loop. Use `Visited` to keep track of the nodes visited. **[2✔]**

Assume these relations :

```
not(X) :- X,!,fail.
not(_).
ispath(A,A).
ispath(A,B) :- ispath(A,[],B).
```

Results :

```
| ?- ispath(a,e).    yes
| ?- ispath(c,e).    no
| ?- ispath(a,c).    yes
| ?- ispath(d,c).    yes
```

```
ispath(A,_,B) :- arrow(A,B).
ispath(A,Visited,B) :-
  arrow(A,X),
  not(member(X,Visited)),
  ispath(X,[X|Visited],B).
```

7. Define the function `contains` which returns true or succeeds if the first argument is in the list which is passed as its second argument. Returns false or fails otherwise. See the examples. In all cases, just use the equals (=) operator for comparison.

   (a) ***Scheme.*** **[1✔]**

`(contains 3 '(1 2 3 4 5))`	`#t`
`(contains 3 '(4 5 6))`	`#f`
`(contains 3 '())`	`#f`

```
(define (contains x list)
      (if (null? list) #f
         (or (= x (car list)) (contains x (cdr list)))))
```

   (b) ***Prolog.*** **[1✔]**

`contains(3,[1,2,3,4,5]).`	`yes`
`contains(3,[4,5,6]).`	`no`
`contains(3,[]).`	`no`

```
contains(X,[X|_]).
contains(X,[_|Y]) :- contains(X,Y).
```

   (c) ***Ocaml.*** **[1✔]**

`contains 3 [1;2;3;4;5];;`	`- : bool = true`
`contains 3 [4;5;6];;`	`- : bool = false`
`contains 3 [];;`	`- : bool = false`

```
let rec contains x list = match list with
   | [] -> false
   | car::cdr -> x = car || contains x cdr;;
```

8. ***Smalltalk.*** Finish the following definition of class `Stack`. Write the methods `push`, `pop`, and `isempty`. **[3✔]**

```
Object subclass: Stack [
   |array top|
   Stack class >> new: size [
      ^ super new init: size
   ]
   init: size [
      top := 0.
      array := Array new: size.
   ]
```

```
pop [
  |result|
  result := array at: top.
  top := top - 1.
  ^ result.
]
push: item [
  top := top + 1.
  array at: top put: item
]
isempty [
  ^ top = 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. **[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. What Java keyword is used to ensure that a method does not allow race conditions?
   (A) `mutex`
   (B) `protected`
   (C) `runnable`
   [✓ (D) `synchronized`]

2. What interface should be implemented by a class object that should support multiple threads?
   (A) `Iterable`
   [✓ (B) `Runnable`]
   (C) `Synchronable`
   (D) `Thread`

3. How many times will the following for-loop be executed? `for (i = 1; i <= 100; i <<= 1);`
   (A) 1
   [✓ (B) 7]
   (C) 8
   (D) 100

4. `(caddr '((1 2 3) (4 5 6) (7 8 9)))`
   (A) `()`
   (B) `(1 2 3)`
   (C) `(4 5 6)`
   [✓ (D) `(7 8 9)`]

5. Type of `[(1,"a");(2,"b");(3,"c")]`
   (A) `(int * int * int) list`
   `* (string * string * string) list`
   (B) `(int * string) list list`
   [✓ (C) `(int * string) list`]
   (D) `int list * string list`

6. What will cause a list with no elements to be passed to the function `f`?
   [✓ (A) `(f '())`]
   (B) `(f ())`
   (C) `(f null)`
   (D) `(f null?)`

7. Perl, Scheme, and Prolog are languages whose type checking is:
   [✓ (A) strong and dynamic]
   (B) strong and static
   (C) weak and dynamic
   (D) weak and static

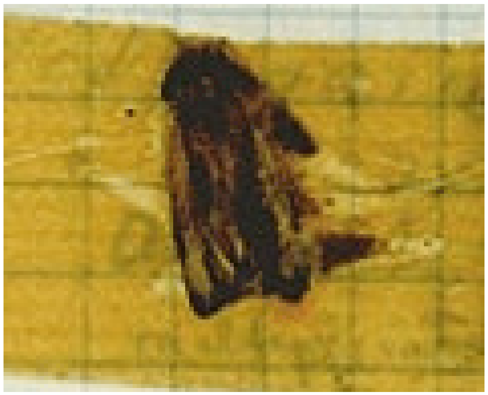
8. Ocaml. Type of `(+)`
   (A) `int * int * int`
   (B) `int * int -> int`
   (C) `int -> int * int`
   [✓ (D) `int -> int -> int`]

9. If `guess` will search a database for a possible answer and `verify` checks to see if it is acceptable, how would this be coded in Prolog?
   [✓ (A) `find(X) :- guess(X), verify(X).`]
   (B) `find(X) :- guess(X).`
   `find(X) :- verify(X).`
   (C) `find(X) :- verify(X), guess(X).`
   (D) `verify(X) :- find(X), guess(X).`

10. If ***M*** = memory leak, and ***D*** = dangling references, which is possible in Java?
   (A) ***D*** but not ***M***
   [✓ (B) ***M*** but not ***D***]
   (C) both of them
   (D) neither

11. Scheme. What will return 2?
   (A) `(if "" 1 2)`
   (B) `(if #\0 1 2)`
   [✓ (C) `(if #f 1 2)`]
   (D) `(if 0 1 2)`

12. Ocaml. What is 7?
   (A) `(+) (3, 4);;`
   [✓ (B) `(+) 3 4;;`]
   (C) `(+) 3, 4;;`
   (D) `3 (+) 4;;`



The First "Computer Bug". Moth found trapped between points at Relay #70, Panel F, of the Mark II Aiken Relay Calculator while it was being tested at Harvard University, 9 September 1947. The operators affixed the moth to the computer log, with the entry: "First actual case of bug being found". They put out the word that they had "debugged" the machine, thus introducing the term "debugging a computer program". In 1988, the log, with the moth still taped by the entry, was in the Naval Surface Warfare Center Computer Museum at Dahlgren, Virginia.

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. Which is generally considered to be a scripting language ?
   (A) Algol 60
   (B) Haskell
   (C) OpenGL
   [✓ (D) Perl]

2. Which function must be written using tail recursion ?
   (A) `filter`
   [✓ (B) `fold_left`]
   (C) `fold_right`
   (D) `map`

3. A garbage collector finds heap objects by finding all objects :
   [✓ (A) in the closure of the root set]
   (B) in the static data area
   (C) that are dead
   (D) that are live

4. Which C, C++, and Java operator uses applicative order evaluation ?
   (A) `&&`
   [✓ (B) `==`]
   (C) `? :`
   (D) `||`

5. Given this Smalltalk definition, what returns 4 ?
   `a := [:x| x + 1].`
   (A) `3 a.`
   (B) `a 3.`
   (C) `a at: 3.`
   [✓ (D) `a value: 3.`]

6. Which can be a fact in a Prolog database ?
   (A) `foo(BAR,BAZ).`
   (B) `foo(BAR,baz).`
   (C) `foo(bar,BAZ).`
   [✓ (D) `foo(bar,baz).`]

7. What is used to pass an unevaluated expression in Haskell ?
   (A) block
   (B) closure
   (C) monad
   [✓ (D) thunk]

8. A closure is :
   (A) A special field of a structure or class used to point at a base class when implementing shared multiple inheritance.
   (B) A special type declaration in Ocaml used to distinguish sum types from product types.
   [✓ (C) A structure on the heap, used to hold variables of an outer function when referenced by an inner function.]
   (D) A table used to dynamically dispatch virtual functions in an object-oriented environment.

9. Ocaml. `let f x = x /. 2.;;`
   [✓ (A) `val f : float -> float = <fun>`]
   (B) `val f : float -> int = <fun>`
   (C) `val f : int -> float = <fun>`
   (D) `val f : int -> int = <fun>`

10. Which languages uses unification to determine the values of variables ?
    (A) Ocaml
    [✓ (B) Prolog]
    (C) Scheme
    (D) Smalltalk

11. In C++, the statement `p->f(x,y);` will be translated into C as :
    (A) `p->f(p,x,y);`
    (B) `p->f(x,y);`
    [✓ (C) `p->vft->f(p,x,y);`]
    (D) `p->vft->f(x,y);`

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