Name_____ NetID _____

CS 314 – Principles of Programming Languages

Spring, 2016

Midterm Exam 2 - Answers

- **Do open this exam** until everyone has an exam and the instructor tells you to begin.
- There are 7 pages in this exam, including this one. When you are told to start, make sure you have them all.
- This exam is closed book – closed notes – closed electronics.
- You must put your cellphone, PDA, Ipod, or other electronic devices in a backpack, etc, and leave it out of reach. The only exception is that you can use a watch that only has time-related functions (e.g. not a calculator watch, not a smart watch).
- Write clearly – if we can't read or can't find your answer your, answer is wrong.
- Make clear what is your answer versus intermediate work.

Please circle the section that you are registered for.

1: Tuesdays: 1:55 pm **2: Tuesdays, 5:15 pm**

3: Fridays, 1:55 pm **4: Fridays, 10:35 pm**

I do not know **None of the above**

1		/20
2		/20
3		/20
4		/20
5		/20
6		/20
7		/20
8		/20
9		/20
Total		/180

1. Finish the definition below for the scheme macro `for-all`, which should work as follows: In `(for-all var vals body)` `var` should be a variable, `vals` should be an expression that evaluates to a list of values, and `body` should be an expression which refers to `var`. E.g., (for-all x '(a 4 6 b) (and (number? x) (* x 2))). The macro call should first bind `var` to the first element of `vals` and evaluate `body`. Then it should bind var to the second element of `vals`, and evaluate `body`, etc. It should return a list of the values of `body`. For instance,
(for-all x '(a 4 6 b) (and (number? x) (* x 2))) should return (#f 8 12 #f).

```
(define-syntax for
  (syntax-rules ( )
   ((_ var vals body)
    (map (lambda (var) body)
         vals))))

or  (let ((body-fn (lambda (var) body)))
       (letrec ((fn (lambda (lst)
                      (if (null? lst) '( )
                          (cons (body-fn (car lst))
                                (fn (cdr lst)))))))
         (fn vals))))))
```

missing parens are ok
0 points for random lispish symbols and parens, e.g.
(macro (body vals (list)
for second version, -5 points for let instead of letrec

2. Suppose we represent a sequence of characters, e.g, *abcd* by a three-element list: (seq length function), where seq is just the symbol seq, length is the number of characters in the sequence, and function is a function from non-negative integers to characters in the sequence. E.g., the sequence *ab* might be represented by
(seq 2 <fn>)
where <fn> is a closure that takes one argument and returns the character #\a if argument is 0 and #\b if the argument is 1. The closure may assume that its argument is always greater than or equal to 0 and less than the length of the sequence.

   Write the function (seq-reverse seq) whose argument seq is a sequence represented as above and whose value is a sequence representing the same characters as in seq but in reverse order. E.g., if seq1 represents the sequence abc then (seq-reverse seq1) represents the sequence cba.

   You may assume the following functions are already defined (you may use them, but you do NOT have to define them):
   (**seq-length** seq) and (**seq-fn** seq) return the length and closure (the <fn>) of sequence seq,

respectively.
(**make-seq** length fn) returns a sequence with the given length and closure.

Hint: think about Project 1, but 1-dimensional instead of 2-dimensional, and without range-check.

```
(define (seq-reverse seq)
  (make-seq (seq-length seq)
          (lambda (pos)
            ((seq-fn seq)
             (- (- (seq-length seq) 1) pos)))))
```

```
or … ((seq-fn seq)
          (- (seq-length seq) (+ pos 1))))))
```

missing parents ok
0 points for random symbols and parens,
-5 for .. (- (seq-length seq) pos)

3. What is the value of the following scheme expression?

```
((((lambda (a)
     (lambda (b)
       (lambda (c)
         (* b (- a c)))))
   3)
  5)
 2)
```

5 0 points for -5, 4, 9, or anything else

4. For each of the following pairs of prolog fact and goal, say whether they match, and if so what bindings will be made.

no partial credit – each part (A thru E) is 4 points or 0 points

	fact	goal	
A.	bar(A, X).	foo(a, b).	_____no match____
B.	bar(A, X).	bar(a, a).	_____A=a, X=a_____
C.	foo(x, y).	foo(R, R).	_____no match_____
D.	foo([X, Y, Y]).	foo([a, b, b]).	_____X=a, Y=b_____
E.	foo([X | Y]).	foo([a, b, c]).	_____X=a, Y=[b, c]_

5. What is the translation into predicate calculus (logic) of the following prolog clause:

eats(P, F):- food(F), color(F, C), dayglow(C).

∀P ∀F eats(P, F) if ∃ C food(F) and color(F, C) and dayglow(C)

∀ and ∃ can be "for all" and "there exists", if can be <= or ← , and can be ^

6.

A. Given the clauses below, what will the query pet_for(P, joe) print? Don't forget to print the variable bindings, if any are made. The predicate write always succeeds and prints its argument. Assume the user **does** type a semicolon to ask for another answer as many times as Prolog allows for it to be typed.

[domesticated,elephant][safe,dog][domesticated,dog][safe,elephant]
[domesticated,elephant][safe,cat][domesticated,cat][joe,cat]

P = cat ; optional -1 for each additional (wrong) or missing item
joe
P = dragon

B. The same as above but for the query pf2(P, joe). Note the cut (!) in the second clause of pf2. Note that for this query, prolog does **not** allow the user to type a semicolon at all.

[domesticated, elephant]
false optional -3 points for each additional (wrong) or missing item

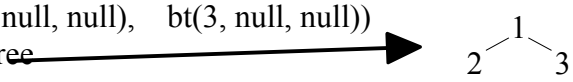
Clauses:

```
pet_for(whale, ahab):- write(whale).
pet_for(Pet, Person):- domesticated(Pet), write([domesticated, Pet]), likes(Person, Pet).
pet_for(dragon, joe):- write(joe).

pf2(whale, ahab):- write(whale).
pf2(Pet, Person):- domesticated(Pet), !, write([domesticated, Pet]), likes(Person, Pet).
pf2(dragon, joe):- write(joe).

domesticated(elephant).
domesticated(Pet):- trainable(Pet), safe(Pet).

trainable(dog).
trainable(elephant).
trainable(cat).
trainable(mouse).

safe(cat):- write([safe, cat]).
safe(dog):- write([safe, dog]).
safe(goldfish):- write([safe, goldfish]).
```

```
safe(elephant):- write([safe, elephant]).

likes(joe, cat):-write([joe, cat]).
likes(joe,  goldfish):-write([joe, goldfish]).
```

Write the following prolog predicates.

- You only need to worry about the first answer prolog will return, not about what will happen if the user types a **;**
- You may use any built-in predicates, including write(X), which prints its argument X.

7. Suppose we represent a binary tree using the functor bt with 3 arguments: data at this node, the left subtree, the right subtree. The empty tree is represented by null. So
   bt(1, bt(2, null, null), bt(3, null, null))
   represents this tree 

   preorder(Tree) should print the data in Tree in pre-order: the data at a node, then all the data in the node's left subtree (in pre-order), then all the data in the node's right subtree (in pre-order). If the tree is empty, nothing should be printed. For example, preorder(bt(1, bt(2, null, null), bt(3, null, null))) should print 1 2 3. Write preorder.

```
preorder(null).                  base case 5 points
preorder(bt(Val, LST, RST)):- write(Val), preorder(LST), preorder(RST).
                                 recursive case:  15 points.
                                       Wrong order of subgoals: -10 points
                                       wrong head of clause: -5 points
                                       missing bt in head:  -3 points
```

8. insert(N, L, LI) takes a number N and a sorted list of numbers L, in increasing order. LI is a version of L but with N inserted so as to keep LI in increasing order. E.g., insert(3, [2, 4, 5], LI) binds LI to [2, 3, 4, 5]. Write insert.

```
insert(N, [ ], [N]).                                  [6 points]
insert(N, [H | T], [H | TI]):- N>H, insert(N, T, TI).  [7 points ]
insert(N, [H | T], [ N | [H | T]]):- N < H.            [7 points ]
```

9. [HARD – leave this one for last.] Write a version of insertionSort that uses accumulator-style recursion to work through the unsorted list in front-to-back order. E.g., sorting [5, 7, 3, 4] first inserts 5 into [ ], giving [5], then inserts 7 into that, giving [5, 7], then inserts 3, giving [3, 5, 7], then inserts 4 giving [3, 4, 5, 7].

   You probably need to use a helper predicate – call it is_help. Finish insertionSort and is_help below. You may use additional clauses if you wish. You may USE the insert predicate as specified by the previous question, even if you have left that question blank. **Do not** define insert here.

```
insertion_sort( L, LS ):- is_help(L, [ ], LS).

is_help( [ ], Result, Result ).

is_help([H | T], A, Res):- insert(H, A, A1), is_help(T, A1, Res).
```