CSSE 304 Exam 2 Jan 22, 2019 (day 26.5) Name_____ Section 01(9:00) 02(9:55) 03(10:50)
PART 1

You may work on any part of the exam in any order, **but you must turn in Part 1 before you use your computer.** During the exam you may not use email, IM, or other chat tools, cell phone, PDA, headphones, ear buds, or any other communication device or software. If you have accommodations that allow one or more of these, you may use it/them.

Part 1: You will not lose points for minor syntax errors in your code.

Part 1 Suggestion: Spend no more than 45 minutes on this part. No resources allowed other a writing implement at eraser.

Parts 2 and 3, computer. For these parts, you may use the resources from Part 1, plus TSPL and EoPL, your notes, and a Scheme programming environment plus the PLC grading program and any materials that I provided online for the course. You may not use any other web or network resources. You are allowed to look at and use any Scheme code that **you** have previously written.

Caution! It is possible to get so caught up in getting all of the points for one problem that you do not get to the other problems. Don't do that! I will give partial credit if you have the main ideas, even if a procedure does not produce correct answers for any test cases.

Problem	Max score	Your score
1	10	
2	7	
3	12	
Total	**29**	

```
(lambda (a c e)
  (lambda (b c d)
   (lambda (a d)
      (a b c d e))))
```

1. (10 points) Consider the above lambda-calculus expression. Fill in the table for the occurrences of each variable:

	a	b	c	d	e
Lexical depth					
Lexical position					

```
(define-datatype bintree bintree?
  [leaf-node
    (datum number?)]
  [interior-node
    (key symbol?)
    (left bintree?)
    (right bintree?)])
```

2. (7 points) Consider the datatype definition in the box at the right. What are the names of the procedures that are defined when that code is executed?

`interior-node?` is not one of those procedures, but it could be useful. Using `cases` in your code, define that function so that `(interior-node? obj)` returns `#t` if `obj` is a valid interior-node and `#f` otherwise. Note that `obj` can be any Scheme value. For full credit, your code must be representation-independent (i.e., don't use `car`, `cadr`, etc.). Some of the credit will be for having short and simple code.

```
(define interior-node?
   (lambda (obj)
```

3. (12 points) A stack ADT could be defined to have 5 operations (you should implement each of them here). Note that this is a different interface than the "OO" stack that we saw earlier.

```
new-stack     ; creates a new stack
empty?        ; tests for empty stack
push!         ; pushes an object onto stack.
pop!          ; removes top element and returns it.
top           ; returns top element without removing it.
```

A Scheme list seems like a simple enough way to represent a stack. The `car` of the list is the top of the stack. Show the implementations of each of the five operations. If you can't do this, do your best to explain why.

A transcript using such stacks might go like this:

```
> (define s1 (new-stack))
> (define s2 (new-stack))
> (empty? s1)
#t
> (push! 'a s1)
> (push! 'b s1)
> (push! 'c s1)
> (push! 'd s2)
> (top s1)
c
> (pop! s1)
c
> (push! (pop! s1) s2)
> (top s2)
b
> (top s1)
a
```

new-stack

empty?

push!

pop!

top