

KOÇ UNIVERSITY
COLLEGE OF ENGINEERING

COMP 301: PROGRAMMING LANGUAGE CONCEPTS

FINAL EXAM
JAN 12 2016
15:00-18:00
SOS B08

INSTRUCTOR: T. METIN SEZGIN
TIME ALLOWED: 120 MINUTES

Name: _____

Student Number: _____

NOTE: EXPLAIN YOUR ANSWERS IN FULL. THE GOAL IS TO DEMONSTRATE YOUR UNDERSTANDING OF THE MATERIAL, THEREFORE AVOID CONTENT-FREE ANSWERS. PROVIDE ALL THE WORK IN YOUR EXAM PAPER, BUT MAKE SURE THE ANSWER BOXES HAVE NOTHING BUT YOUR FINAL ANSWER TO THE QUESTIONS. INCLUDE CONTRACTS FOR ALL SCHEME FUNCTIONS THAT YOU DEFINE.

I PLEDGE ON MY HONOR THAT I HAVE NEITHER GIVEN NOR RECEIVED UNAUTHORIZED ASSISTANCE ON THIS EXAM.

Signature: _____

Question	Worth	Grade
1	20	
2	30	
3	20	
4	20	
5	10	
Bonus	10	
Total	110	

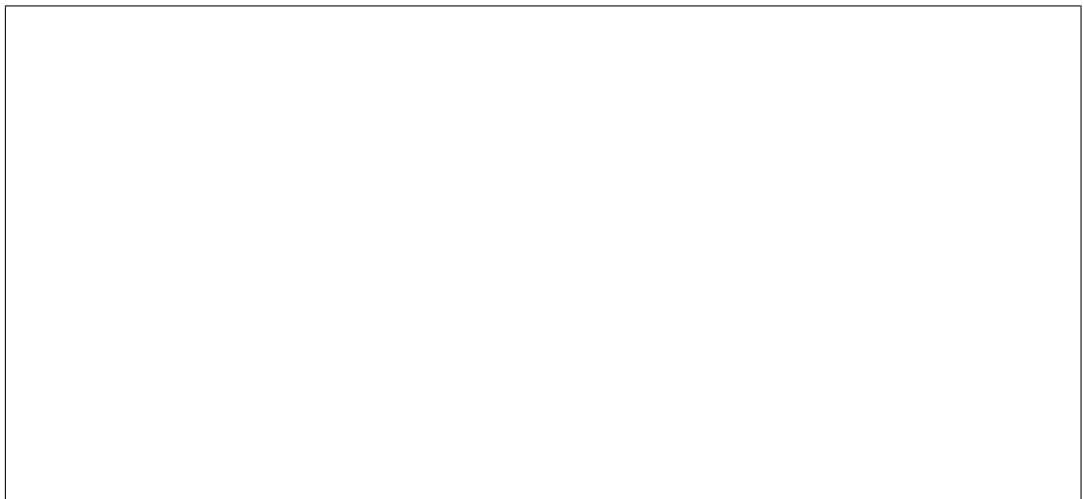
1. (20 points) Consider the program below.

```
let x = 5
in let f = proc(x) -(x,-(0,x))
   in let g = proc(y) begin set y=-(x,1); x end
   in (f (g x))
```

- (a) (10 points) What is the value of the program under call-by-reference? Explain, and show all your work. No points for correct result without convincing explanation.



- (b) (10 points) What is the value of the program under call-by-value? Explain, and show all your work. No points for correct result without convincing explanation.



2. (30 points)

- (a) (10 points) Identify all variable declarations (circle, and label). Each missing declaration and non-declarations identified as declarations lose 1 point.

```
let x = 10

in let f = proc(x) -(x,-(0,x))

in let y = 1

in let g = proc(x) begin set x=5; x; y end

in -( (f begin set x=-(x,-(0,x)); y; x end), (g (g y)) )
```

- (b) (10 points) Identify all variable references (circle, and label). Each missing reference and non-references identified as references lose 1 point.

```
let x = 10

in let f = proc(x) -(x,-(0,x))

in let y = 1

in let g = proc(x) begin set x=5; x; y end

in -( (f begin set x=-(x,-(0,x)); y; x end), (g (g y)) )
```

- (c) (10 points) Write down the lexical depth of each variable reference. Missing and incorrect depths lose 1 point each.

```
let x = 10

in let f = proc(x) -(x,-(0,x))

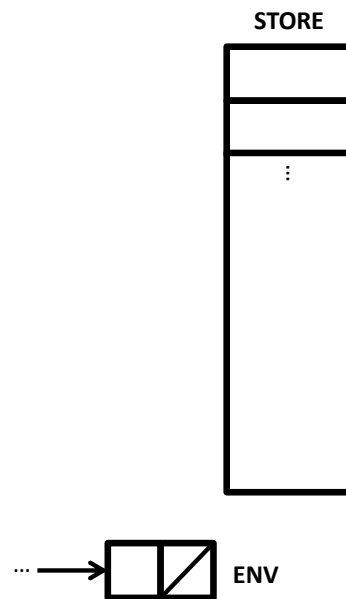
in let y = 1

in let g = proc(x) begin set x=5; x; y end

in -( (f begin set x=-(x,-(0,x)); y; x end), (g (g y)) )
```

3. (20 points) Consider the program below. Draw the environment and the store (memory) when the **value-of** function of the interpreter receives the expression $-(x, (g\ x))$ assuming an implementation of mutable pairs that stores pointers to both items in the pair. Note that you will have to improvise a way of depicting pairs visually. You can ignore the default variables created during the initialization of the interpreter.

```
let x = 20
  in let p = newpair(x, 10)
    in let g = proc (x) -(x, 5)
      in -(x, (g x))
```



4. (20 points) Again, consider the program below (same as the previous question). Compute the number of total memory allocations that we would expect to see after the program terminates under the assumptions noted in parts a and b. You can ignore the default variables created during the initialization of the interpreter.

```
let x = 20
  in let p = newpair(x,10)
    in let g = proc (x) -(x,5)
      in -(x,(g x))
```

- (a) Compute the number of memory allocations assuming call by value. Clearly list and number each point in the program where a memory allocation occurs. Explain. No explanation, no points. Max 50 words.

- (b) Compute the number of memory allocations assuming call by reference. Clearly list and number each point in the program where a memory allocation occurs. Explain. No explanation, no points. Max 50 words.

5. (10 points)

- (a) (5 points) I was running a simple expression `foo` on my computer. The last call to `value-of/k`, looked like:

```
(value-of/k
<<let x = proc(y) -(y, -(0,y)) in (x (x 3))>>
env
#(struct:diff2-cont (num-val 35)
#(struct:diff1-cont <<10>> init-env
#(struct:end-cont))))
```

What is the value of the program that I was running?

Answer:

- (b) (5 points) Write down the code corresponding to the simple expression `foo`.

6. (10 points) Bonus: I tried to use the trampolining technique to write a version of the factorial procedure that would not grow the call stack. The new version returns a zero-argument procedure every once in a while (roughly once every 10 recursive calls) to pop back the call stack. However, when I call this function, with x less than 10 works ok. If I call it with $10 < x < 20$, then I find myself having to wrap the result with parantheses to get a result:

```
> (fact 15)
#<procedure:...polined-fact.rkt:14:12>
> ((fact 15))
1307674368000
```

If $20 < x < 30$, then I need additional parantheses:

```
> (fact 25)
#<procedure:...polined-fact.rkt:14:12>
> ((fact 25))
#<procedure:...polined-fact.rkt:14:12>
> (((fact 25)))
15511210043330985984000000
```

It is obvious that there is something requiring a simple fix. Here is my code below. Show what needs to be fixed, and how. Explain.

```
(define fact
  (lambda (x)
    (fact-i x 1)))

(define fact-i
  (lambda (x pa)
    (if (= x 1)
        pa
        (if (= (remainder x 10) 0)
            (lambda () (fact-i (- x 1) (* pa x)))
            (fact-i (- x 1) (* pa x))))))

(define trampoline
  (lambda (x)
    (if (procedure? x)
        (trampoline (x))
        x)))
```