Sample Mid-Term Exam 2

CS 5510, Fall 2015

October 29

Name: ______________________________

Instructions: You have eighty minutes to complete this open-book, open-note, closed-interpreter exam. Please write all answers in the provided space, plus the back of the exam if necessary.

1) Which of the following produce different results in a eager language and a lazy language? Both produce the same result if they both produce the same number or they both produce a procedure (even if the procedure doesn't behave exactly the same when applied), but they can differ in errors reported.

a) `{{lambda {y} 12} {1 2}}`

b) `{lambda {x} {{lambda {y} 12} {1 2}}}`

c) `{+ 1 {lambda {y} 12}}`

d) `{+ 1 {{lambda {x} {+ 1 13}} {+ 1 {lambda {z} 12}}}}`

e) `{+ 1 {{lambda {x} {+ x 13}} {+ 1 {lambda {z} 12}}}}`

2) Here is an outline of the `lambda-k.rkt` interpreter:

```
     #lang plai-typed
     (require plai-typed/s-exp-match)

1    (define-type Value
|      ....)

2    (define-type ExprC
|      ....)

3    (define-type Binding
|      ....)

4    (define-type-alias Env (listof Binding))
|    (define mt-env empty)
|    (define extend-env cons)

5    (define-type Cont
|      ....)

     ;; parse ----------------------------------------
6    (define (parse [s : s-expression]) : ExprC
|      ....)

7    (module+ test ....)

     ;; interp & continue ----------------------------------------
8    (define (interp [a : ExprC] [env : Env] [k : Cont]) : Value
|      ....)

9    (define (continue [k : Cont] [v : Value]) : Value
|      ....)

10   (module+ test ....)

     ;; num+ and num* ----------------------------------------
11   (define (num-op [op : (number number -> number)] [l : Value] [r : Value]) : Value
|      ....)
|    (define (num+ [l : Value] [r : Value]) : Value
|      (num-op + l r))
|    (define (num* [l : Value] [r : Value]) : Value
|      (num-op * l r))

12   (module+ test ....)

     ;; lookup ----------------------------------------
13   (define (lookup [n : symbol] [env : Env]) : Value
|      ....)

14   (module+ test ....)
```

Below are possible additions and changes to the language that the interpreter implements. Beside each change, indicate (using numbers from the left margin above) the parts of the program that should be modified to implement that change, and very briefly describe the change at that part.

1. Add a divide operator

2. Add a single-binding `letrec` form

3. Remove the `let` form

4. Make the arguments to `+` evaluated right-to-left instead of left-to-right

3) Given the following expression:

```
{{lambda {x} {x x}}
 {lambda {y} 12}}
```

Describe a trace of the evalaution in terms of arguments to `interp` and `continue` functions for every call of each in the `lambda-k.rkt` interpreter. (There will be 7 calls to `interp` and 5 calls to `continue`.) The `interp` function takes three arguments — an expression, an environment, and a continuation — so show all three for each `interp` call. The `continue` function takes two arguments — a continuation and a value — so show both for each `continue` call. Represent continuations using records.

Answers

1) *a* and *d*.

2)

1. 2: add a new divide expression variant
   5: add two new continuation variants: eval second argument, do divide
   6: parse divide form
   7: test parsing of divide form
   8: handle new expression variant
   9: handle two new continuation variants
   10: add a test for divide
   11: add a divide function for use in `continue`
   12: test divide function

2. 2: add a new `letrec` expression variant
   4: add recursive-binding variant for environments
   5: add one continuation variants: bind recursive and eval body
   6: parse `letrec` form
   7: test parsing of `letrec` form
   8: handle new expression variant
   9: handle new continuation variant
   10: add a test for `letrec`
   13: handle recursive-bidning lookup
   14: test recursive-bidning lookup

3. 6: remove parsing of `let` form
   7: remove test of parsing `let` form
   10: adjust any tests that used `let`

4. 8: interp second expression for addition, instead of first
   10: adjust any error tests affected by the change; add error test to check right-to-left

   It would also be acceptable to adjust 5 and 9 to change the variant name `addSecondK` to `addFirstK`.

3)

interp	expr	=	`{{lambda {x} {x x}} {lambda {y} 12}}`
	env	=	`mt-env`
	k	=	`(doneK)`
interp	expr	=	`{lambda {x} {x x}}`
	env	=	`mt-env`
	k	=	`(appArgK {lambda {y} 12} mt-env (doneK))` $= k_1$
cont	k	=	`(appArgK {lambda {y} 12} mt-env (doneK))` or k_1
	val	=	`(closV 'x {x x} mt-env)` $= v_1$
interp	expr	=	`{lambda {y} 12}`
	env	=	`mt-env`
	k	=	`(doAppK` v_1 `(doneK))` $= k_2$
cont	k	=	`(doAppK` v_1 `(doneK))` or k_2
	val	=	`(closV 'y 12 mt-env)` $= v_2$

interp	expr	=	$\boxed{\texttt{\{x x\}}}$
	env	=	(extend-env (bind 'x v_2) mt-env) $= e_1$
	k	=	(doneK)
interp	expr	=	$\boxed{\texttt{x}}$
	env	=	e_1
	k	=	(appArgk $\boxed{\texttt{x}}$ e_1 (doneK)) $= k_3$
cont	k	=	(appArgK $\boxed{\texttt{x}}$ e_1 (doneK)) or k_3
	val	=	v_2
interp	expr	=	$\boxed{\texttt{x}}$
	env	=	e_1
	k	=	(doAppK v_2 (doneK)) $= k_4$
cont	k	=	(doAppK v_2 (doneK)) or k_4
	val	=	v_2
interp	expr	=	$\boxed{\texttt{12}}$
	env	=	(extend-env (bind 'y v_2) mt-env)
	k	=	(doneK)
cont	k	=	(doneK)
	val	=	(numV 12)