Sample Mid-Term Exam 2

CS 3520/6520, Fall 2018

November 6

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.
Note on actual exam: The exam will refer to the `lambda-k.rkt` interpreter. If you need the interpreter for reference to answer the questions, please bring a copy (paper or electronic) with you.

1) [20 pts] 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) [20 pts] Suppose a garbage-collected interepreter uses the following three kinds of records:

- Tag **1**: a record containing two pointers
- Tag **2**: a record containing one pointer and one integer
- Tag **3**: a record containing one integer

The interpreter has one register, which always contains a pointer, and a memory pool of size 22. The allocator/collector is a two-space copying collector, so each space is of size 11. Records are allocated consecutively in to-space, starting from the first memory location, 0.

The following is a snapshot of memory just before a collection where all memory has been allocated:

- Register: 8
- To space: 1 3 8 3 0 2 3 7 2 0 8

What are the values in the register and the new to-space (which is also addressed starting from 0) after collection? Assume that unallocated memory in to-space contains 0.

- Register:

- To space:

3) [60 pts] Given the following expression:

```
{{lambda {x} {x x}}
 {lambda {y} {+ 5 7}}}
```

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 9 calls to `interp` and 7 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.

Use the extra exam page for additional space, and use the following abbreviations to save time:

X_0 = the whole expression
X_1 = {`lambda {x} {x x}`}
X_2 = {`x x`}
X_3 = {`lambda {y} {+ 5 7}`}
X_4 = {`+ 5 7`}

Answers

1) *a* and *d*.

2) Register: 0, To space: 2 3 8 1 6 0 3 0 0 0 0

3)

interp	expr	=	$\boxed{X_0}$
	env	=	`mt-env`
	k	=	`(doneK)`
interp	expr	=	$\boxed{X_1}$
	env	=	`mt-env`
	k	=	`(appArgK` $\boxed{X_3}$ `mt-env (doneK))` $= k_1$
cont	k	=	k_1
	val	=	`(closV 'x` $\boxed{X_2}$ `mt-env)` $= v_1$
interp	expr	=	$\boxed{X_3}$
	env	=	`mt-env`
	k	=	`(doAppK` v_1 `(doneK))` $= k_2$
cont	k	=	k_2
	val	=	`(closV 'y` $\boxed{X_4}$ `mt-env)` $= v_2$
interp	expr	=	$\boxed{X_2}$
	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	=	k_3
	val	=	v_2
interp	expr	=	$\boxed{\texttt{x}}$
	env	=	e_1
	k	=	`(doAppK` v_2 `(doneK))` $= k_4$
cont	k	=	k_4
	val	=	v_2
interp	expr	=	$\boxed{X_4}$
	env	=	`(extend-env (bind 'y` v_2`) mt-env) =` e_2
	k	=	`(doneK)`
interp	expr	=	$\boxed{5}$
	env	=	e_2
	k	=	`(plusSecondK` $\boxed{7}$ e_2 `(doneK))` $= k_5$

cont	k	=	k_5
	val	=	`(numV 5)`
interp	expr	=	`7`
	env	=	e_2
	k	=	`(doPlusK (numV 5) (doneK))` $= k_6$
cont	k	=	k_6
	val	=	`(numV 7)`
cont	k	=	(doneK)
	val	=	`(numV 12)`

Same answer, but not expanding many abbreviations (which isn't recommended when you're writing them by hand):

interp	expr	=	`{{lambda {x} {x x}} {lambda {y} {+ 5 7}}}` or X_0
	env	=	`mt-env`
	k	=	`(doneK)`
interp	expr	=	`{lambda {x} {x x}}` or X_1
	env	=	`mt-env`
	k	=	`(appArgK {lambda {y} {+ 5 7}} mt-env (doneK))` $= k_1$
cont	k	=	`(appArgK {lambda {y} {+ 5 7}} mt-env (doneK))` or k_1
	val	=	`(closV 'x {x x} mt-env)` $= v_1$
interp	expr	=	`{lambda {y} {+ 5 7}}` or X_3
	env	=	`mt-env`
	k	=	`(doAppK` v_1 `(doneK))` $= k_2$
cont	k	=	`(doAppK` v_1 `(doneK))` or k_2
	val	=	`(closV 'y {+ 5 7} mt-env)` $= v_2$
interp	expr	=	`{x x}` or X_2
	env	=	`(extend-env (bind 'x` v_2`) mt-env)` $= e_1$
	k	=	`(doneK)`
interp	expr	=	`x`
	env	=	`(extend-env (bind 'x` v_2`) mt-env)` or e_1
	k	=	`(appArgk x` e_1 `(doneK))` $= k_3$
cont	k	=	`(appArgK x` e_1 `(doneK))` or k_3
	val	=	v_2
interp	expr	=	`x`
	env	=	`(extend-env (bind 'x` v_2`) mt-env)` or e_1

	k	=	`(doAppK` v_2 `(doneK))` $= k_4$
cont	k	=	`(doAppK` v_2 `(doneK))` or k_4
	val	=	v_2
interp	expr	=	$\boxed{\texttt{\{+ 5 7\}}}$ or X_4
	env	=	`(extend-env (bind 'y` v_2`) mt-env)` $= e_2$
	k	=	`(doneK)`
interp	expr	=	$\boxed{\texttt{5}}$
	env	=	`(extend-env (bind 'y` v_2`) mt-env)` or e_2
	k	=	`(plusSecondK` $\boxed{\texttt{7}}$ e_2 `(doneK))` $= k_5$
cont	k	=	`(plusSecondK` $\boxed{\texttt{7}}$ e_2 `(doneK))` or k_5
	val	=	`(numV 5)`
interp	expr	=	$\boxed{\texttt{7}}$
	env	=	`(extend-env (bind 'y` v_2`) mt-env)` or e_2
	k	=	`(doPlusK (numV 5) (doneK))` $= k_6$
cont	k	=	`(doPlusK (numV 5) (doneK))` or k_6
	val	=	`(numV 7)`
cont	k	=	(doneK)
	val	=	`(numV 12)`