Mid-Term Exam 2

CS 3520, Fall 2005

(take-home, but Fall 2007 exam will be in-class)

Name: ______________________________

Start time: ____________________
End time: ____________________

Instructions: You have ninety minutes to complete this open-book, open-note, closed-computer, take-home exam. Please write you start and finish times above, and write all answers in the provided space, plus the back of the exam if necessary. **The Fall 2007 Mid-Term 2 exam will be in-class.**

1) Which of the following produce different results in a call-by-value language and a call-by-name 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).

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

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

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

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

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

2) The following web servlet implementation (main handler plus helper function) uses `web-read`, which takes only a prompt and uses `let/cc` internally to obtain a continuation. Convert the servlet (both functions) to instead use `web-read/k`, which takes a prompt and an explicit continuation procedure (and does not use `let/cc` internally). You should assume that the `correct-password?` function requires no interaction with the user. **The Fall 2007 version of this question will be more difficult.**

```
(define (pw-handler base args)
  (get-pw (web-read "Name")))

(define (get-pw name)
   (local [(define pw (web-read "Password"))]
     (if (correct-password? name pw)
         (format "Hello, ~a" name)
         (get-pw name))))
```

3) Given the following expression:

```
{{fun {x} {x x}}
 {fun {y} 12}}
```

Describe a trace of the evalaution in terms of arguments to `interp` and `continue` functions for every call of each. (There will be 7 calls to `interp` and 5 calls to `continue`.) The `interp` function takes three arguments — an expression, a substitution cache, and a continuation — so show all three for each `interp` call. The `continue` function takes two arguments — a value and a continuation — so show both for each `continue` call. Represent continuations using records. **The Fall 2007 version of this question will involve the store-passing interpreter of HW6 instead of the `interp`–`continue` interpreter. The trace will be shorter than in Mid-Term 1.**

4) 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:

Answers

1) *a* and *d*.

2)

```
(define (pw-handler base args)
  (web-read/k "Name" get-pw))

(define (get-pw name)
   (web-read/k "Password"
               (lambda (pw)
                 (if (correct-password? name pw)
                     (format "Hello, ~a" name)
                     (get-pw name)))))
```

3)

interp	expr	=	$\boxed{\texttt{\{\{fun \{x\} \{x x\}\} \{fun \{y\} 12\}\}}}$
	subs	=	`(mtSub)`
	k	=	`(mtK)`
interp	expr	=	$\boxed{\texttt{\{fun \{x\} \{x x\}\}}}$
	subs	=	`(mtSub)`
	k	=	`(appArgK` $\boxed{\texttt{\{fun \{y\} 12\}}}$ `(mtSub) (mtK))`
cont	val	=	`(closureV 'x` $\boxed{\texttt{\{x x\}}}$`)` $= v_1$
	k	=	`(appArgK` $\boxed{\texttt{\{fun \{y\} 12\}}}$ `(mtSub) (mtK))`
interp	expr	=	$\boxed{\texttt{\{fun \{y\} 12\}}}$
	subs	=	`(mtSub)`
	k	=	`(doAppK` v_1 `(mtK))`
cont	val	=	`(closureV 'y` $\boxed{\texttt{12}}$`)` $= v_2$
	k	=	`(doAppK` v_1 `(mtK))`
interp	expr	=	$\boxed{\texttt{\{x x\}}}$
	ds	=	`(aSub 'x` v_2 `(mtSub))` $= ds_1$
	k	=	`(mtK)`
interp	expr	=	$\boxed{\texttt{x}}$
	ds	=	ds_1
	k	=	`(appArgk` $\boxed{\texttt{x}}$ ds_1 `(mtK))`
cont	val	=	v_2
	k	=	`(appArgK` $\boxed{\texttt{x}}$ ds_1 `(mtK))`
interp	expr	=	$\boxed{\texttt{x}}$
	ds	=	ds_1
	k	=	`(doAppK` v_2 `(mtK))`
cont	val	=	v_2

	k	=	`(doAppK` v_2 `(mtK))`
interp	expr	=	$\boxed{\texttt{12}}$
	ds	=	`(aSub 'y` v_2 `(mtSub))`
	k	=	`(mtK)`
cont	val	=	`(numV 12)`
	k	=	`(mtK)`

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