CS 3520/6520 Fall 2019
Practice Midterm Exam 2

Name: ______________________________

Instructions You have eighty minutes to complete this open-book, open-note exam. Electronic devices are allowed only to consult notes or books from local storage; network use is prohibited. **Write only on the front side of each page**, and ask the proctor for extra pages if needed.

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.

For each of the following, indicate whether the expression produces the **same** or **different** results in a eager variant of Curly and a lazy variant of Curly. Both produce the same result if they both produce the same number, they both produce a procedure (even if the procedure doesn't behave exactly the same when applied), or they both produce an error (even if the errors differ).

1. `{+ 1 2}` 5 points

2. `{{lambda {y} 12} {1 2}}` 5 points

3. `{lambda {x} {{lambda {y} 12} {1 2}}}` 5 points

4. `{+ 1 {lambda {y} 12}}` 5 points

5. `{+ 1 {{lambda {x} {+ 1 13}} {+ 1 {lambda {z} 12}}}}` 5 points

6. `{+ 1 {{lambda {x} {+ x 13}} {+ 1 {lambda {z} 12}}}}` 5 points

7. Suppose a garbage-collected interepreter uses the following three kinds of records: 15 points

- 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:

In the "`lambda-k.rkt`", what final result will the following `continue` calls produce? Show your answer as a Plait expression of type `Value`, or write *error* if the `continue` call leads to an error instead of a result `Value`.

The actual exam will have fewer of these.

8. 5 points

```
(continue (doPlusK (numV 8)
                   (doneK))
          (numV -1))
```

9. 5 points

```
(continue (doAppK (closV 'x
                         (parse `{* x x})
                         mt-env)
                  (doPlusK (numV 1)
                           (doneK)))
          (numV 3))
```

10. 5 points

```
(continue (appArgK (parse `{lambda {f} {f y}})
                   (extend-env (bind 'y (numV 5)) mt-env)
                   (doneK))
          (numV 3))
```

11. 5 points

```
(continue (appArgK (parse `{lambda {f} {f y}})
                   (extend-env (bind 'y (numV 5)) mt-env)
                   (doneK))
          (closV 'g
                 (parse `{g {lambda {q} {+ q {* -1 y}}}})
                 (extend-env (bind 'y (numV 7)) mt-env)))
```

Each remaining question shows an expression plus a candidate trace of `interp` and `continue` using the `"lambda-k.rkt"` implementation. The trace should show all calls to `interp` and `continue` in the right order with the right arguments. If `interp` or `continue` eventually reports an error, the trace should show *error* at the end of the trace, and without omitting any calls to `interp` or `continue` that are made or any result values that are produced by nested calls.

For each question, mark the trace as "correct" if it correctly shows the complete `interp` at `continue` trace. For an incorrect trace, identify the first place where the trace is wrong (which would be the end if the trace is incomplete) and provide the correct next term—either a full `interp` call or a full `continue` call—that should appear at that position.

Keep in mind that `parse` desugars `let`, so ``(parse `{let {[x 1]} x})`` is interchangeable with ``(parse `{{lambda {x} x} 1})``, for example.

The actual exam will have fewer of these.

12. 10 points

```
{+ 2 1}
```

```
[1]  (interp (parse `{+ 2 1})
             mt-env
             (doneK))
[2]  (interp (parse `2)
             mt-env
             K1 = (plusSecondK (parse `1) mt-env (doneK)))
[3]  (continue K1
               (numV 2))
[4]  (interp (parse `3)
             mt-env
             K2 = (doPlusK (numV 2) (doneK)))
[5]  (continue K2
               (numV 3))
[6]  (continue (doneK)
               (numV 5))
```

13. 10 points

```
{lambda {x} 5}
```

```
[1]  (interp (parse `{lambda {x} 5})
             mt-env
             (doneK))
[2]  (continue (doneK)
               (closV 'x (parse `5) mt-env))
```

14. 10 points

```
{let {[f {lambda {x} {+ x 1}}]}
  {f 10}}
```

```
[1]   (interp (parse `{{lambda {f} {f 10}}
                       {lambda {x} {+ x 1}}})
              mt-env
              (doneK))
[2]   (interp (parse `{lambda {f} {f 10}})
              mt-env
              K1 = (appArgK
                     (parse `{lambda {x} {+ x 1}})
                     mt-env
                     (doneK)))
[3]   (continue K1
                V1 = (closV 'f (parse `{f 10}) mt-env))
[4]   (interp (parse `{lambda {x} {+ x 1}})
              mt-env
              K2 = (doAppK V1 (doneK)))
[5]   (continue K2
                V2 = (closV 'x (parse `{+ x 1}) mt-env))
[6]   (interp (parse `{f 10})
              E1 = (extend-env (bind 'f V2) mt-env)
              (doneK))
[7]   (interp (parse `f)
              E1
              K3 = (appArgK (parse `10) E1 (doneK)))
[8]   (continue K3
                V2)
[9]   (interp (parse `10)
              E1
              K4 = (doAppK V2 (doneK)))
[10]  (continue K4
                (numV 10))
[11]  (interp (parse `{+ x 1})
              E2 = (extend-env (bind 'x (numV 10)) mt-env)
              (doneK))
[12]  (interp (parse `x)
              E2
              K5 = (plusSecondK (parse `1) E2 (doneK)))
[13]  (continue K5
                (numV 10))
[14]  (interp (parse `1)
              E2
              K6 = (doPlusK (numV 10) (doneK)))
```

```
[15]  (continue K6
                (numV 1))
[16]  (continue (doneK)
                (numV 11))
```

15. 10 points

```
{let {[f {lambda {x} {+ x 1}}]}
  f}
```

```
[1]  (interp (parse `{{lambda {f} f}
                      {lambda {x} {+ x 1}}})
             mt-env
             (doneK))
[2]  (interp (parse `{lambda {f} f})
             mt-env
             K1 = (appArgK
                    (parse `{lambda {x} {+ x 1}})
                    mt-env
                    (doneK)))
[3]  (continue K1
               V1 = (closV 'f (parse `f) mt-env))
[4]  (interp (parse `{lambda {x} {+ x 1}})
             mt-env
             K2 = (doAppK V1 (doneK)))
[5]  (continue K2
               (closV 'x (parse `{+ x 1}) mt-env))
[6]  (interp (parse `{+ x 1})
             mt-env
             (doneK))
[7]  (interp (parse `x)
             mt-env
             (plusSecondK (parse `1) mt-env (doneK)))
```

[8] *error*

16. 10 points

```
{{{{lambda {x}
     {lambda {y}
       {lambda {x}
         x}}}
   1}
  2}
 0}
```

```
[1]  (interp (parse `{{{{lambda {x}
                         {lambda {y}
                           {lambda {x} x}}}
                       1}
                      2}
                     0})
             mt-env
             (doneK))
[2]  (interp (parse `{{{lambda {x}
                        {lambda {y} {lambda {x} x}}}
                      1}
                     2})
             mt-env
             K1 = (appArgK (parse `0) mt-env (doneK)))
[3]  (interp (parse `{{lambda {x}
                       {lambda {y} {lambda {x} x}}}
                     1})
             mt-env
             K2 = (appArgK (parse `2) mt-env K1))
[4]  (interp (parse `{lambda {x}
                      {lambda {y} {lambda {x} x}}})
             mt-env
             K3 = (appArgK (parse `1) mt-env K2))
[5]  (continue K3
               V1 = (closV
                     'x
                     (parse `{lambda {y} {lambda {x} x}})
                     mt-env))
[6]  (interp (parse `0)
             mt-env
             K4 = (doAppK V1 K2))
[7]  (continue K4
               (numV 0))
[8]  (interp (parse `{lambda {y} {lambda {x} x}})
             E1 = (extend-env (bind 'x (numV 0)) mt-env)
             K2)
```

```
[9]   (continue K2
                V2 = (closV 'y (parse `{lambda {x} x}) E1))
[10]  (interp (parse `2)
              mt-env
              K5 = (doAppK V2 K1))
[11]  (continue K5
                (numV 2))
[12]  (interp (parse `{lambda {x} x})
              E2 = (extend-env (bind 'y (numV 2)) E1)
              K1)
[13]  (continue K1
                V3 = (closV 'x (parse `x) E2))
[14]  (interp (parse `1)
              mt-env
              K6 = (doAppK V3 (doneK)))
[15]  (continue K6
                (numV 1))
[16]  (interp (parse `x)
              (extend-env (bind 'x (numV 1)) E2)
              (doneK))
[17]  (continue (doneK)
                (numV 1))
```

17. 10 points

```
{let {[f {lambda {x}
           {lambda {y} {x y}}}]}
  {{f {lambda {z} z}}
   1}}
```

```
[1]   (interp (parse `{{lambda {f}
                         {{f {lambda {z} z}} 1}}
                        {lambda {x}
                          {lambda {y} {x y}}}})
              mt-env
              (doneK))
[2]   (interp (parse `{lambda {f}
                        {{f {lambda {z} z}} 1}})
              mt-env
              K1 = (appArgK
                     (parse `{lambda {x} {lambda {y} {x y}}})
                     mt-env
                     (doneK)))
[3]   (continue K1
                V1 = (closV
                       'f
                       (parse `{{f {lambda {z} z}} 1})
                       mt-env))
[4]   (interp (parse `{lambda {x} {lambda {y} {x y}}})
              mt-env
              K2 = (doAppK V1 (doneK)))
[5]   (continue K2
                V2 = (closV 'x (parse `{lambda {y} {x y}}) mt-env))
[6]   (interp (parse `{{f {lambda {z} z}} 1})
              E1 = (extend-env (bind 'f V2) mt-env)
              (doneK))
[7]   (interp (parse `{f {lambda {z} z}})
              E1
              K3 = (appArgK (parse `1) E1 (doneK)))
[8]   (interp (parse `f)
              E1
              K4 = (appArgK (parse `{lambda {z} z}) E1 K3))
[9]   (continue K4
                V2)
[10]  (interp (parse `{lambda {z} z})
              E1
              K5 = (doAppK V2 K3))
[11]  (continue K5
                V3 = (closV 'z (parse `z) E1))
```

```
[12]  (interp (parse `{lambda {y} {x y}})
              E2 = (extend-env (bind 'x V3) mt-env)
              K3)
[13]  (continue K3
                V4 = (closV 'y (parse `{x y}) E2))
[14]  (interp (parse `1)
              E1
              K6 = (doAppK V4 (doneK)))
[15]  (continue K6
                (numV 1))
[16]  (interp (parse `{x y})
              E3 = (extend-env (bind 'y (numV 1)) E2)
              (doneK))
[17]  (interp (parse `x)
              E3
              K7 = (appArgK (parse `y) E3 (doneK)))
[18]  (continue K7
                V3)
[19]  (interp (parse `y)
              E3
              K8 = (doAppK V3 (doneK)))
[20]  (continue K8
                (numV 1))
[21]  (interp (parse `z)
              (extend-env (bind 'z (numV 1)) E1)
              (doneK))
[22]  (continue (doneK)
                (numV 1))
```

18. This question is too mean to be on an exam, but if you check every detail, you should be able to find a mistake. 10 points

```
{let {[f {lambda {x} {* -1 x}}]}
  {+ {f 10} 8}}
```

```
[1]    (interp (parse `{{lambda {f} {+ {f 10} 8}}
                        {lambda {x} {* -1 x}}})
               mt-env
               (doneK))
[2]    (interp (parse `{lambda {f} {+ {f 10} 8}})
               mt-env
               K1 = (appArgK
                      (parse `{lambda {x} {* -1 x}})
                     mt-env
                      (doneK)))
[3]    (continue K1
                 V1 = (closV 'f (parse `{+ {f 10} 8}) mt-env))
[4]    (interp (parse `{lambda {x} {* -1 x}})
               mt-env
               K2 = (doAppK V1 (doneK)))
[5]    (continue K2
                 V2 = (closV 'x (parse `{* -1 x}) mt-env))
[6]    (interp (parse `{+ {f 10} 8})
               E1 = (extend-env (bind 'f V2) mt-env)
               (doneK))
[7]    (interp (parse `{f 10})
               E1
               K3 = (plusSecondK (parse `8) E1 (doneK)))
[8]    (interp (parse `f)
               E1
               K4 = (appArgK (parse `10) E1 K3))
[9]    (continue K4
                 V2)
[10]   (interp (parse `10)
               E1
               K5 = (doAppK V2 K3))
[11]   (continue K5
                  (numV 10))
[12]   (interp (parse `{* -1 x})
               E2 = (extend-env (bind 'x (numV 10)) E1)
               K3)
[13]   (interp (parse `-1)
               E2
               K6 = (multSecondK (parse `x) E2 K3))
[14]   (continue K6
                  (numV -1))
```

```
[15]  (interp (parse `x)
              E2
              K7 = (doMultK (numV -1) K3))
[16]  (continue K7
                (numV 10))
[17]  (continue K3
                (numV -10))
[18]  (interp (parse `8)
              E1
              K8 = (doPlusK (numV -10) (doneK)))
[19]  (continue K8
                (numV 8))
[20]  (continue (doneK)
                (numV -2))
```

Answers

1. Same result: `3`.

2. Different results: error and `12`.

3. Same result: a function.

4. Same result: error.

5. Different results: error and `15`.

6. Same result: error.

7. Register: `0`, To space: `2 3 8 1 6 0 3 0 0 0 0`

8. `(numV 7)`

9. `(numV 10)`

10. *error*, because 3 is not a function

11. `(numV -2)`

12. Step [4] should have a `1` instead of `3`: `(interp (parse `1) mt-env)`.

13. Correct.

14. Correct.

15. The body expression `{+ x 1}` should not be `interp`ed. Step [6] should be

```
(interp (parse `f)
        (extend-env (bind 'f (closV 'x (parse `{+ x 1}) mt-env))
                    mt-env)
        (doneK))
```

16. Starting at step [6], the expressions/values `0` and `1` are backwards. The final answer should be `(numV 0)`. Step [6] should be

```
(interp (parse `1)
        mt-env
        (doAppK V1 K2))
```

17. Correct.

18. Step 12 should have `mt-env` in place of `E1`.