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 may refer to the "env.rkt", "lambda.rkt", and "store-with.rkt" interpreters. If you need the interpreters for reference to answer the questions, please bring a copy (paper or electronic) with you.

**1. Given the following grammar:**

     \[
     \langle \text{weed} \rangle ::= \text{leaf} \\
     \quad \mid (\text{branch} \ \langle \text{weed} \rangle \ \langle \text{weed} \rangle) \\
     \quad \mid (\text{stem} \ \langle \text{weed} \rangle)
     \]

Provide a **define-type** declaration for \texttt{Weed} that is a suitable representation for \(\langle \text{weed} \rangle\)s.
2. Implement the function `weed-forks`, which takes a `Weed` and returns the number of branches that it contains. Your implementation must follow the shape of the data definitions, and it **must include suitable and sufficient tests**.
For each of the following expressions, show the store that would be returned with the program's value when using the "store-with.rkt" interpreter. Instead of nested "override-store"s, you can show the store as a list of cells. Recall that locations are allocated starting at 1.

3. 

{box {box (+ 1 2)}}

4. 

{let {[[b {box (+ 1 2)}]]
    {begin
        {set-box! b 4}
        {box 5}}}}

5. 

{let {[[f {lambda {x}
            {box x}}]]
    {set-box! {f 0} {f 1}}}}

6. 

{let {[[f {lambda {x}
            {box x}}]]
    {let {[[b {f 10}]]
        {set-box! b b}}}}
Each remaining question shows an expression plus a candidate trace of interp using the "lambda.rkt" implementation. Nesting is not shown at all (either with boxes or indentation or leading > and <), but the trace should show all calls to interp in the right order with the right arguments, and it should show all returns from interp at the right places with the right result values. If interp eventually reports an error, the trace should show error at the end of the trace, and without omitting any calls to initerp 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 trace. For an incorrect interp 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 result value—that should appear at that position.

The actual exam will have fewer of these.

7. 9 points

{ + 2 1}

[1] (interp (parse `(+ 2 1))
   mt-env)
[2] (interp (parse `2)
   mt-env)
[3] = (numV 2)
[4] (interp (parse `3)
   mt-env)
[5] = (numV 3)
[6] = (numV 5)

8. 9 points

{lambda {x} 5}

[1] (interp (parse `(lambda {x} 5))
   mt-env)
[2] = V1 = (closV `x (parse `5) mt-env)
9. 9 points

```scheme
(let ([f (lambda [x] (+ x 1))])
  
  (interp (parse `(let ((f (lambda [x] (+ x 1))))
                  
                      {f 10}))
           mt-env)

  (interp (parse `(lambda [x] (+ x 1)))
           mt-env)

  = V1 = (closV 'x (parse `(x (+ x 1))) mt-env)

  (interp (parse `(f 10))
           
           E1 = (extend-env (bind 'f V1) mt-env))

  (interp (parse `'f)
           
           E1)

  = V1

  (interp (parse `10)
           
           E1)

  = (numV 10)

  (interp (parse `(x (+ x 1)))
           
           E2 = (extend-env (bind 'x (numV 10)) mt-env))

  (interp (parse `x)
           
           E2)

  = (numV 10)

  (interp (parse `1)
           
           E2)

  = (numV 1)

  = (numV 11)

  = (numV 11)

  = (numV 11)

  = (numV 11)
```

10. 9 points

```scheme
(let ([f (lambda [x] (+ x 1))])
  
  (interp (parse `(let ((f (lambda [x] (+ x 1))))
                  
                  f))
           mt-env)

  (interp (parse `(lambda [x] (+ x 1)))
           mt-env)

  = V1 = (closV 'x (parse `(x (+ x 1))) mt-env)

  (interp (parse `(+ x 1))
           
           mt-env)

  (interp (parse `x)
           
           mt-env)
```

5
11. 9 points

```plaintext
{let {{f {lambda {x}
    {lambda {y} {x y}}}}}
  {{f {lambda {z} z}}}
1}

[1] (interp (parse `let {f
    {lambda {x}
      {lambda {y} {x y}}}}
  {f {lambda {z} z}} 1})
mt-env)
[2] (interp (parse `lambda {x} {lambda {y} {x y}}) mt-env)
[3] = V1 = (closV 'x (parse `lambda {y} {x y}) mt-env)
[4] (interp (parse `{f {lambda {z} z}} 1})
E1 = (extend-env (bind 'f V1) mt-env))
[5] (interp (parse `f {lambda {z} z})
E1)
[6] (interp (parse `f)
E1)
[7] = V1
[8] (interp (parse `lambda {z} z))
E1)
[9] = V2 = (closV 'z (parse `z) E1)
[10] (interp (parse `{lambda {y} {x y}})
E2 = (extend-env (bind 'x V2) mt-env))
[12] = V3
[13] (interp (parse `1)
E1)
[14] = (numV 1)
[15] (interp (parse `{x y})
E3 = (extend-env (bind 'y (numV 1)) E2))
[16] (interp (parse `x)
E3)
[17] = V2
[18] (interp (parse `y)
E3)
[19] = (numV 1)
[20] (interp (parse `z)
E4 = (extend-env (bind 'z (numV 1)) E1))
[21] = (numV 1)
[22] = (numV 1)
[23] = (numV 1)
[24] = (numV 1)
```
12. This question is too mean to be on an exam, but if you check every detail, you should be able to find a mistake. Hint: the number of the step that is wrong is part of the expression for question 6.

\[
\text{let } \{[f \{\lambda \{x\} \{\ast -1 \ x}\}]\} \\
\{+ \{f \ 10\} \ 8\}
\]

[1] (interp (parse `\{let \{\{f \{\lambda \{x\} \{\ast -1 \ x}\}\}\} \\
\{+ \{f \ 10\} \ 8\}\}`)
   mt-env)
[2] (interp (parse `\{\lambda \{x\} \{\ast -1 \ x}\}`)
   mt-env)
[3] = V1 = (closV 'x (parse `\{\ast -1 \ x\}`) mt-env)
[4] (interp (parse `\{+ \{f \ 10\} \ 8\}`)
   E1 = (extend-env (bind 'f V1) mt-env))
[5] (interp (parse `\{f \ 10\}`)
   E1)
[6] (interp (parse `\f`)
   E1)
[7] = V1
[8] (interp (parse `\10`)
   E1)
[9] = (numV 10)
[10] (interp (parse `\{\ast -1 \ x\}`)
    E2 = (extend-env (bind 'x (numV 10)) E1))
[11] (interp (parse `\-1`)
    E2)
[12] = (numV -1)
[13] (interp (parse `\x`)
    E2)
[14] = (numV 10)
[15] = (numV -10)
[16] = (numV -10)
[17] (interp (parse `\8`)
    E1)
[18] = (numV 8)
[19] = (numV -2)
[20] = (numV -2)
Answers

1. 

(define-type Weed
  (leaf)
  (stem [rest : Weed])
  (branch [left : Weed]
    [right : Weed]))

2. 

(define (weed-forks [w : Weed]) : Number
  (type-case Weed w
    [(leaf) 0]
    [(stem rest) (weed-forks rest)]
    [(branch l r) (+ 1
      (+ (weed-forks l)
        (weed-forks r)))]
    ))

(test (weed-forks (leaf))
  0)
(test (weed-forks (stem (leaf)))
  0)
(test (weed-forks (stem (branch (leaf) (leaf))))
  1)
(test (weed-forks (branch (branch (leaf) (leaf)) (leaf)))
  2)

3. (list (cell 2 (boxV 1)) (cell 1 (numV 3)))
4. (list (cell 2 (numV 5)) (cell 1 (numV 4)) (cell 1 (numV 3)))
5. (list (cell 1 (boxV 2)) (cell 2 (numV 1)) (cell 1 (numV 0)))
6. (list (cell 1 (boxV 1)) (cell 1 (numV 10)))

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

8. Correct.


10. The body expression {+ x 1} should not be interp. Step [4] should be

    (interp (parse `f)
      (extend-env (bind 'f V1)
        mt-env))
11. Correct.

12. Step 10 should have `mt-env` in place of `E1`.