CS 3520/6520 Fall 2023
Practice Midterm Exam 1

Name: _____

Instructions: You have eighty minutes to complete this open-book, open-note exam. Electronic devices are allowed only to consult notes, slides, or books (not videos) from local storage. Network use to look up technical information is prohibited, you can use DrRacket only as an editor (not running any programs), and use of other interpreters or large language models is not allowed. **Write only on the front side of each page**, and ask the proctor for extra pages if needed.

1. Given the following grammar: 8 points

⟨*weed*⟩ `::= leaf`
`|  ? branch` ⟨*weed*⟩ ⟨*weed*⟩ `¿`
`|  ? stem` ⟨*weed*⟩ `¿`

Provide a `type` declaration for `Weed` that is a suitable representation for ⟨*weed*⟩s.

2. Implement the function `weed_forks`, which takes a `Weed` and returns the number of `branch`es that it contains. Your implementation must follow the shape of the data definitions, and **it must include suitable and sufficient tests**. 20 points

For each of the following Moe expressions, show the store (as a Shplait expression) that would be returned with the program's value when using the `"store_reslet.rhm"` interpreter. Instead of nested `override_stores`, you can show the store as a list of `cells`. Make sure the cells appear in the store list in same left-to-right order that `"store_reslet.rhm"` produces. Recall that locations are allocated starting at `1`.

3. 9 points

```
box(box(1 + 2))
```

4. 9 points

```
let b = box(1 + 2):
  begin:
    set_box(b, 4)
    box(5)
```

5. 9 points

```
let f = (fun (x):
           box(x)):
  set_box(f(0), f(1))
```

6. 9 points

```
let f = (fun (x):
           box(x)):
  let b = f(10):
    set_box(b, b)
```

Each remaining question shows a Moe expression plus a candidate trace of `interp` using the `"lambda.rhm"` implementation. Unlike normal trace output, nesting is not shown at all here, 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 `interp` 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] = intV(2)
[4] interp(parse('3'),
           mt_env)
[5] = intV(3)
[6] = intV(5)
```

8. 9 points

```
fun (x):
  5
```

```
[1] interp(parse('fun (x): 5'),
           mt_env)
[2] = closV(#'x, parse('5'), mt_env)
```

9. 9 points

```
let f = (fun (x):
           x + 1):
  f(10)
```

```
[1]  interp(parse('let f = (fun (x): x + 1): f(10)'),
            mt_env)
[2]  interp(parse('fun (x): x + 1'),
            mt_env)
[3]  = V1 = closV(#'x, parse('x + 1'), mt_env)
[4]  interp(parse('f(10)'),
            E1 = extend_env(bind(#'f, V1), mt_env))
[5]  interp(parse('f'),
            E1)
[6]  = V1
[7]  interp(parse('10'),
            E1)
[8]  = intV(10)
[9]  interp(parse('x + 1'),
            E2 = extend_env(bind(#'x, intV(10)), mt_env))
[10] interp(parse('x'),
            E2)
[11] = intV(10)
[12] interp(parse('1'),
            E2)
[13] = intV(1)
[14] = intV(11)
[15] = intV(11)
[16] = intV(11)
```

10. 9 points

```
let f = (fun (x):
           x + 1):
  f
```

```
[1] interp(parse('let f = (fun (x): x + 1): f'),
           mt_env)
[2] interp(parse('fun (x): x + 1'),
           mt_env)
[3] = closV(#'x, parse('x + 1'), mt_env)
[4] interp(parse('x + 1'),
           mt_env)
[5] interp(parse('x'),
           mt_env)
[6] error
```

11. 9 points

```
let f = (fun (x):
           fun (y):
             x(y)):
  f(fun (z):
      z)(1)
```

```
[1]  interp(parse('let f = (fun (x): fun (y): x(y)): f(fun (z): z)(1)'),
            mt_env)
[2]  interp(parse('fun (x): fun (y): x(y)'),
            mt_env)
[3]  = V1 = closV(#'x, parse('fun (y): x(y)'), mt_env)
[4]  interp(parse('f(fun (z): z)(1)'),
            E1 = extend_env(bind(#'f, V1), mt_env))
[5]  interp(parse('f(fun (z): z)'),
            E1)
[6]  interp(parse('f'),
            E1)
[7]  = V1
[8]  interp(parse('fun (z): z'),
            E1)
[9]  = V2 = closV(#'z, parse('z'), E1)
[10] interp(parse('fun (y): x(y)'),
            E2 = extend_env(bind(#'x, V2), mt_env))
[11] = V3 = closV(#'y, parse('x(y)'), E2)
[12] = V3
[13] interp(parse('1'),
            E1)
[14] = intV(1)
[15] interp(parse('x(y)'),
            E3 = extend_env(bind(#'y, intV(1)), E2))
[16] interp(parse('x'),
            E3)
[17] = V2
[18] interp(parse('y'),
            E3)
[19] = intV(1)
[20] interp(parse('z'),
            extend_env(bind(#'z, intV(1)), E1))
[21] = intV(1)
[22] = intV(1)
[23] = intV(1)
[24] = intV(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. 9 points

```
let f = (fun (x):
           -1 * x):
  f(10) + 8
```

```
[1]   interp(parse('let f = (fun (x): -1 * x): f(10) + 8'),
             mt_env)
[2]   interp(parse('fun (x): -1 * x'),
             mt_env)
[3]   = V1 = closV(#'x, parse('-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]   = intV(10)
[10]  interp(parse('-1 * x'),
             E2 = extend_env(bind(#'x, intV(10)), E1))
[11]  interp(parse('-1'),
             E2)
[12]  = intV(-1)
[13]  interp(parse('x'),
             E2)
[14]  = intV(10)
[15]  = intV(-10)
[16]  = intV(-10)
[17]  interp(parse('8'),
             E1)
[18]  = intV(8)
[19]  = intV(-2)
[20]  = intV(-2)
```

Answers

1.

```
// Note that `?` and `¿` are just bits of concrete syntax.
// They don't need to be represented any more than the spaces
// between words or the parentheses in Moe.

type Weed
| leaf()
| stem(rst :: Weed)
| branch(left :: Weed,
         right :: Weed)
```

2.

```
fun weed_forks(w :: Weed) :: Int:
  match w
  | leaf(): 0
  | stem(rst): weed_forks(rst)
  | branch(l, r): 1
                  + weed_forks(l)
                  + weed_forks(r)

check: weed_forks(leaf())
       ~is 0
check: weed_forks(stem(leaf()))
       ~is 0
check: weed_forks(stem(branch(leaf(), leaf())))
       ~is 1
check: weed_forks(branch(branch(leaf(), leaf()), leaf()))
       ~is 2
```

3.

```
[cell(2, boxV(1)),
 cell(1, intV(3))]
```

4.

```
[cell(2, intV(5)),
 cell(1, intV(4)),
 cell(1, intV(3))]
```

5.

```
[cell(1, boxV(2)),
 cell(2, intV(1)),
 cell(1, intV(0))]
```

6.

```
[cell(1, boxV(1)),
 cell(1, intV(10))]
```

7. Step [4] should have a `1` instead of `3`: `interp(parse('1'), mt_env)`.

8. Correct.

9. Correct.

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

```
interp(parse('f'),
       extend_env(bind(#'f, V1),
                  mt_env))
```

where `V1` is the result shown at [3].

11. Correct.

12. Step 10 should have `mt_env` in place of `E1`.