Q1
2 Points

Q1.1
1 Point

Consider the following program, written in the language of `lecture6c.rkt` from class:

```
(let f (let b (box 0)
             (lambda x (begin (set-box! b (+ 1 (unbox b))) (unbox b))))
    (begin e
           (f 0)))
```

where `e` is some arbitrary expression in the language.

Suppose that executing the above snippet does not trigger an error. Which of the following are possible values that could be returned (just the value part of the result, not the store part)? Select all that apply.

- [x] `(numV 1)`
- [x] `(numV 3)`
- [x] `(numV 2)`
- [ ] `(numV 0)`

Q1.2

1 Point

Consider the following program, written in the language of `lecture6c.rkt` from class:

```
(let b (box 0)
  (let f (lambda x (begin (set-box! b (+ 1 (unbox b))) (unbox b)))
       (begin e
              (f 0))))
```

where `e` is some arbitrary expression in the language.

Suppose that executing the above snippet does not trigger an error. Which of the following are possible values that could be returned? Select all that apply.

- [x] `(numV 3)`
- [x] `(numV 1)`
- [x] `(numV 2)`
- [x] `(numV 0)`

Q2
1 Point

In `lecture6c.rkt` we implemented a language with mutable state (through boxes) and `begin` statements (represented with the seqC constructor in our AST) for sequencing commands. Consider the following incorrect variant of our implementation of `seqC` in that interpreter:

```
(define (eval-env (env : Env) (sto : Store) (e : Expr)) : Result
  (type-case Expr e
             ...
             [seqC (e1 e2)
                   (type-case Result (eval-env env sto e1)
                     [res (v1 sto-1)
                          (eval-env env sto e2)])]
             ...))
```

Suppose `b` is a variable bound to a box that initially contains the value 0. On which of the following examples does the above implementation of `seqC` give incorrect behavior, in the sense that calling `eval` will yield a different result than calling `eval` in a correct implementation. (Assume that the rest of the implementation of `eval-env` is the same as in `lecture6c.rkt`, aside frrom `seqC`.)

- (●) `(begin (set-box! b 1) (+ (set-box! b 2) (unbox b)))`
- ( ) `(begin (unbox b) (unbox b))`
- ( ) `(begin (unbox b) (+ (set-box! b 2) (unbox b)))`
- ( ) `(begin (set-box! b 1) (+ (unbox b) (unbox b)))`

Q3

1 Point

Consider the following program:

```
(let [(x 1)]
  (let [(f (lambda (y) (+ x y)))]
    (let ([x 6])
      (f 3))))
```

Let `v1` be the result of evaluating this program in a language where `let` and `lambda` have lexical scope, and let `v2` be the result of evaluating this program in a lanuage where `let` and `lambda` have dynamic scope.

Which of the following is true?

- ○ `v1 = 9`, `v2 = 9`
- ◉ `v1 = 4`, `v2 = 9`
- ○ `v1 = 4`, `v2 = 4`
- ○ `v1 = 9`, `v2 = 4`

Q4
2 Points

Consider the following expression written in the language used in the programming part of problem set 3:

```
(let fact (lambda n (if (equal? 0 n) 1 (* n (fact (+ n -1)))))
     (fact 5))
```

Evaluating this expression should raise an unbound identifier error for the recursive occurrence of `fact` in the definition of `fact`. We saw how this happens by considering the behavior of this example with a substitution-based interpreter with our environment-based intepreter that had lexical scope.

What happens if we change the semantics of the language to use dynamic scope instead and then execute this example?

- (●) It returns 120.
- ( ) It returns 5.
- ( ) It returns 1.
- ( ) It returns an unbound identifier error.

Q5
2 Points

Consider the following two lambda expressions written in the language of problem set 3:

```
(lambda p
    (let b1 (fst p)
      (let b2 (snd p)
        (begin
          (set-box! b1 1)
          (set-box! b2 2)
          (+ (unbox b1) (unbox b2))))))

(lambda p
    (let b1 (fst p)
      (let b2 (snd p)
        (+ (begin (set-box! b1 1) (unbox b1))
           (begin (set-box! b2 2) (unbox b2))))))
```

call the first lambda expression `e1`, and call the second `e2`.

Let `e` be a closed expression in the same language, such that evaluating the expression `(pair (e1 e) (e2 e))` does not return an error and returns the value `v`.
Which of the following could `v` possibly be (select all that apply)?

- [ ] `(pairV (numV 4) (numV 4))`
- [x] `(pairV (numV 4) (numV 3))`
- [ ] `(pairV (numV 3) (numV 4))`
- [x] `(pairV (numV 3) (numV 3))`

Q6
2 Points

Q6.1
1 Point

In the programming assignment, you are asked to implement an interpreter for a language with if statements. Here is an incorrect implementation of `ifC` statements for that language using a store-passing style like we saw in class:

```
(define (eval-env (env : Env) (sto : Store) (e : Expr)) : Result
  (type-case Expr e
             ...
             [ifC (guard e1 e2)
                  (let* ([rguard (eval-env env sto guard)]
                         [re1 (eval-env env (res-s rguard) e1)]
                         [re2 (eval-env env (res-s rguard) e2)])
                        (cond
                          [(equal? (res-v rguard) (boolV #true)) re1]
                          [(equal? (res-v rguard) (boolV #false)) re2]
                          [else (error 'eval-env "guard non-bool")]))]
             ...))
```

where `env` and `sto` and `Result` are as in `lecture6c.rkt` from class.

On which of the following examples does the above implementation of `ifC` give incorrect behavior, in the sense that calling `eval` will yield a different result than calling `eval` in a correct implementation.

- (●) `(if #t (+ 1 1) (+ 1 (lambda x 1))`
- ( ) `(if #t (+ 1 1) (+ 2 2)`
- ( ) `(if #f (+ 1 1) (+ 2 2)`
- ( ) `(if #f (+ 1 1) (+ 1 (lambda x 1))`

Q6.2
1 Point

Here is another incorrect implementation of `ifC`.

```
(define (eval-env (env : Env) (sto : Store) (e : Expr)) : Result
  (type-case Expr e
             ...
             [ifC (guard e1 e2)
                  (type-case Result (eval-env env sto guard)
                        [res (vguard sto-new)
                             (if (equal? vguard (boolV #true))
                                 (eval-env env sto e1)
                                 (eval-env env sto e2))])]
             ...))
```

Suppose the variable `b` is bound to a box initially containing the value 0. On which of the following examples does the above implementation of `ifC` give incorrect behavior?

- ○ `(if #f (set-box! b 10) (unbox b))`
- ○ `(if #t (set-box! b 10) (unbox b))`
- ○ `(if (equal? (unbox b) 0) (unbox b) (+ 1 (unbox b)))`
- ◉ `(if (begin (set-box! b 1) #t) (unbox b) 2)`

Pset 3 - Comprehension

● Ungraded

2 Hours, 18 Minutes Late

Student
Kal Sastra

Total Points
- / 10 pts

Question 1

(no title) 2 pts

1.1 (no title) 1 pt

1.2 (no title) 1 pt

Question 2

(no title) 1 pt

Question 3

(no title) 1 pt

Question 4

(no title) 2 pts

Question 5

(no title) 2 pts

Question 6

(no title) 2 pts

6.1 (no title) 1 pt

6.2 (no title) 1 pt