answers

Problem 1

(a) a is 0.
(b) a is 1.

Problem 2

Call By Name

(a)

BEFORE

```
#(struct:let-exp
  a
  #(struct:const-exp 12)
  #(struct:let-exp
    f
    #(struct:proc-exp
      x
      #(struct:begin-exp
        #(struct:var-exp x)
        (#(struct:var-exp x) #(struct:var-exp x))))
    #(struct:call-exp
      #(struct:var-exp f)
      #(struct:begin-exp
        #(struct:assign-exp
          a
          #(struct:diff-exp #(struct:var-exp a) #(struct:const-exp -13)))
        (#(struct:var-exp a))))))
env =
()
store =
()

entering let a
> (value-of (const-exp 12) (empty-env))
```

AFTER

```
#(struct:const-exp 12)
env =
()
```

```
store =
()

< (num-val 12)
entering body of let a with env =
((a 0))
store =
((0 #(struct:num-val 12)))

>(value-of
  (let-exp
   'f
   (proc-exp 'x (begin-exp (var-exp 'x) (list (var-exp 'x) (var-exp 'x))))
   (call-exp
    (var-exp 'f)
    (begin-exp
     (assign-exp 'a (diff-exp (var-exp 'a) (const-exp -13)))
     (list (var-exp 'a)))))
  (extend-env 'a 0 (empty-env)))
```

(b)

BEFORE

```
#(struct:const-exp -13)
env =
((f 1) (a 0))
store =
((0 #(struct:num-val 12))
 (1 (procedure x ... ((a 0))))
 (2 (thunk ... ((f 1) (a 0)))))

< < < (num-val -13)
< < <(num-val 25)
< < (num-val 27)
> > (value-of (var-exp 'a) (extend-env 'f 1 (extend-env 'a 0 (empty-env))))
```

AFTER

```
#(struct:var-exp a)
env =
((f 1) (a 0))
store =
((0 #(struct:num-val 25))
 (1 (procedure x ... ((a 0))))
 (2 (thunk ... ((f 1) (a 0)))))

< < (num-val 25)
```

```
< <(num-val 25)
< (num-val 25)
> (value-of (var-exp 'x) (extend-env 'x 2 (extend-env 'a 0 (empty-env))))
```

Call By Need

(a)

BEFORE

```
#(struct:let-exp
  a
  #(struct:const-exp 12)
  #(struct:let-exp
    f
    #(struct:proc-exp
      x
      #(struct:begin-exp
        #(struct:var-exp x)
        (#(struct:var-exp x) #(struct:var-exp x))))
    #(struct:call-exp
      #(struct:var-exp f)
      #(struct:begin-exp
        #(struct:assign-exp
          a
          #(struct:diff-exp #(struct:var-exp a) #(struct:const-exp -13)))
        (#(struct:var-exp a))))))
env =
()
store =
()

entering let a
> (value-of (const-exp 12) (empty-env))
```

AFTER

```
#(struct:const-exp 12)
env =
()
store =
()

< (num-val 12)
entering body of let a with env =
((a 0))
store =
```

```
((0 #(struct:num-val 12)))

>(value-of
  (let-exp
   'f
   (proc-exp 'x (begin-exp (var-exp 'x) (list (var-exp 'x) (var-exp 'x))))
   (call-exp
    (var-exp 'f)
    (begin-exp
     (assign-exp 'a (diff-exp (var-exp 'a) (const-exp -13)))
     (list (var-exp 'a)))))
  (extend-env 'a 0 (empty-env)))
```

(b)

BEFORE

```
< (proc-val
   (procedure
    'x
    (begin-exp (var-exp 'x) (list (var-exp 'x) (var-exp 'x)))
    (extend-env 'a 0 (empty-env))))
entering body of let f with env =
((f 1) (a 0))
store =
((0 #(struct:num-val 12)) (1 (procedure x ... ((a 0)))))

>(value-of
  (call-exp
   (var-exp 'f)
   (begin-exp
    (assign-exp 'a (diff-exp (var-exp 'a) (const-exp -13)))
    (list (var-exp 'a))))
  (extend-env 'f 1 (extend-env 'a 0 (empty-env))))
```

AFTER

```
#(struct:var-exp a)
env =
((f 1) (a 0))
store =
((0 #(struct:num-val 25))
 (1 (procedure x ... ((a 0))))
 (2 (thunk ... ((f 1) (a 0)))))

< < (num-val 25)
< <(num-val 25)
```

```
< (num-val 25)
> (value-of (var-exp 'x) (extend-env 'x 2 (extend-env 'a 0 (empty-env))))
```

2.2

A) In IREF the program would enter an infinite loop.

B) We think the function behaves the same in both paradigms.

C) p = [i=1, v=1, x=2].