COMP301: Problem Set 9

Problem 1.

In the first program anonymous procedure `g` is a `proc-val` bound to the scope of `counter = newref(0)`. Here, counter is shared among all invocations of g, which means successive calls of `g` modify the same `counter`. Here's how the environment looks for the first program:

```
|           +-- counter ----> Ref -> (num-val 2)
|           |
|-- g ----> (proc-val (procedure ...))
    |
    +-- a ----> (num-val 1)
        |
        +-- b ----> (num-val 2)
```

In the second program, `g` is a `proc-val` bound to the global scope initialized by `init-env()`. Each invocation of `g` creates a new scope with its own `counter = newref(0)` as follows:

```
| Global Environment
|
|-- g ----> (proc-val (procedure ...))
|
|-- a ----> (num-val 1)
|
|-- b ----> (num-val 1)
```

```
| Environment for <<(g 11)>>
|
|-- dummy ----> (num-vall 11)
|
|-- counter ----> Ref -> (num-val 0)
|
...
```

Thus, first program returns `a - b = 1 - 2 = -1`.

Second program returns `a - b = 1 - 1 = 0`.

Problem 2.

Modified code for `setref-exp`

```
...
(setref-exp (exp1 exp2)
  (let ((ref (expval->ref (value-of exp1 env))))
    (let ((old-value (deref ref)))
      (let ((v2 (value-of exp2 env)))
        (begin
          (setref! ref v2)
          (old-value))))))
...
```

Problem 3.

Modified code in `store.rkt`

```
...

;; newref : ExpVal -> Ref
(define newref
  (lambda (val)
    (let ((new-store (vector-enlarge the-store)))
      (begin
        (vector-set! new-store (- (vector-length new-store) 1) val)
        (set! the-store new-store)
        (- (vector-length the-store) 1)))))

;; deref : Ref -> ExpVal
(define deref
  (lambda (ref)
    (vector-ref the-store ref)))

;; setref! : Ref * ExpVal -> Unspecified
(define setref!
  (lambda (ref val)
    (vector-set! the-store ref val)))

...
```