CS61A Notes – Week 5a: Vectors (solutions)

Just When You Were Getting Used to Lists

QUESTIONS

1. **Write a procedure `(sum-of-vector v)` that adds up the numbers inside the vector. Assume all data fields are valid numbers.**

   You can use an accumulator or a helper like this:

```
(define (sum-of-vector v)
   (define (helper index)
      (cond ((= index (vector-length v)) 0)
            (else (+ (vector-ref v index) (helper (+ index 1))))))
   (helper 0))
```

1. **Write a procedure `(vector-copy! src src-start dst dst-start length)`. After the call, `length` elements in vector `src` starting from index `src-start` should be copied into vector `dst` starting from index `dst-start`.**

```
STk> a => #(1 2 3 4 5 6 7 8 9 10)
STk> b => #(a b c d e f g h i j k)
STk> (vector-copy! a 5 b 2 3) => okay
STk> a => #(1 2 3 4 5 6 7 8 9 10)
STk> b => #(a b 6 7 8 f g h i j k)
```

```
(define (vector-copy! src src-start dst dst-start length)
   (if (> length 0)
       (begin
          (vector-set! dst dst-start (vector-ref src src-start))
          (vector-copy! src (+ src-start 1) dst (+ dst-start 1) (-
length 1)))))
```

1. **Write a procedure `(insert-at! v i val)`; after a call, vector `v` should have `val` inserted into location `i`. All elements starting from location `i` should be shifted down. The last element of `v` is discarded.**

```
STk> a => #(i'm like you #[unbound] #[unbound])
STk> (insert-at! a 1 'bohemian) => okay
STk> a => #(i'm bohemian like you #[unbound])
```

```
(define (insert-at! v i val)
   (let* ((amt-to-shift (- (vector-length v) i 1))
          (temp-v (make-vector amt-to-shift)))
      (vector-copy! v i temp-v 0 amt-to-shift)
      (vector-copy! temp-v 0 v (+ i 1) amt-to-shift)
      (vector-set! v i val)))
```

NOTE: why didn't we just do (vector-copy! v i v (+ i 1) amt-to-shift)?

1. **Write a procedure `(vector-double! v)`. After a call, vector `v` should be doubled in size, with all the elements in the old vector replicated in the second half. So,**

```
STk> a => #(1 2 3 4)
STk> (vector-double! a) => okay
STk> a => #(1 2 3 4 1 2 3 4)
```

IMPOSSIBLE! Hope you weren't fooled by this. To double the size of a vector, you'd have to allocate a new vector. However, recall that you cannot change what "a" points to from within a procedure! You can at most

return a new vector double in size.

1. **Write a procedure `reverse-vector!`. Do I have to explain what it does?**

```
;; for elements from start to stop, do body
(define (from-to-do start stop body)
   (if (> start stop)
       'done
       (begin (body start) (from-to-do (+ 1 start) stop body))))

(define (reverse-vector! v)
   (from-to-do 0 (- (quotient (vector-length v) 2) 1)
      (lambda (i)
         (let ((temp (vector-ref v i)))
            (vector-set! v i (vector-ref v (- (vector-length v) i
1)))
            (vector-set! v (- (vector-length v) i 1) temp)))))
```

1. **Write a procedure `(square-table! t)` that takes in a rectangular table and squares every element.**

```
(define (square-table! t)
   (let ((n (vector-length t))
         (m (vector-length (vector-ref t 0))))
      (from-to-do 0 (- n 1)
         (lambda (i)
            (from-to-do 0 (- m 1)
               (lambda (j)
                  (vector-set!
                     (vector-ref! t i)
                     j
                     (square (vector-ref! (vector-ref! t i)
j)))))))))
```