

CS101 Computer Science I (Spring 2001)
Lab 7: March 28 - 30, 2001

1. A Scheme procedure called "**sum-from-to**" takes three inputs. The first input is a procedure called "**fun**". The procedure **fun** takes a number as input and returns a number as output. The remaining inputs to **sum-from-to** are two integers called "**low**" and "**high**". The procedure **sum-from-to** applies **fun** to each integer from **low** to **high** (including **low** and **high**), adds up all the results, and returns the sum. If **low** is greater than **high**, then **sum-from-to** returns zero.

```
(sum-from-to identity 1 3)           ==>    6
(sum-from-to (lambda (x) (* x x)) 1 3) ==>   14
(sum-from-to identity 7 7)           ==>    7
(sum-from-to (lambda (x) (* x x)) 1 0) ==>    0
```

Write the procedure **sum-from-to** once, using the higher-order procedures **map** and **apply** in combination with the procedure **from-to**. Recall that **(from-to low high)** returns a list of the integers from **low** to **high**. Write the procedure **sum-from-to** again, using recursion, without using any higher-order procedures other than **sum-from-to** itself in the definition.

2. Write a Scheme procedure called "**each-successive-pair?**" that takes a list called "**lst**" and a binary predicate called "**pred?**" as inputs. The **each-successive-pair?** procedure applies **pred?** to each successive pair of members of **lst**, i.e., it applies **pred?** to the 1st and 2nd members of **lst**, the 2nd and 3rd members, the 3rd and 4th members, etc. The procedure **each-successive-pair?** returns **#t** if **pred?** returns **#t** each and every time. If **lst** has fewer than two members, then **each-successive-pair?** also returns **#t**. Otherwise, **each-successive-pair?** returns **#f**.

```
(each-successive-pair? '(a a a a) equal?) ==> #t
(each-successive-pair? '(a b b a) equal?) ==> #f
(each-successive-pair? '(2 5 8 9) <)      ==> #t
(each-successive-pair? '(2 8 5 9) <)      ==> #f
(each-successive-pair? '(a) equal?)       ==> #t
(each-successive-pair? '() <)             ==> #t
```

3. Write a procedure called "**increasing-by-n?**" that takes a list of integers called "**lst**" and a single integer called "**n**" as inputs. The procedure **increasing-by-n?** returns **#t** if each member of **lst** is exactly **n** greater than the previous one. If **lst** has fewer than two members, then **increasing-by-n?** returns **#t**. Otherwise **increasing-by-n?** returns **#f**. You should use **each-successive-pair?** in your definition **increasing-by-n?**.

```
(increasing-by-n? '(5 10 15 20 25) 5) ==> #t
(increasing-by-n? '(5 10 15 20 25) 3) ==> #f
(increasing-by-n? '(5 10 15 40 55) 5) ==> #t
(increasing-by-n? '(7) 5)              ==> #t
(increasing-by-n? '() 5)                ==> #t
```