UNIVERSITY OF TORONTO
Faculty of Arts and Science

TERM TEST #2
CSC 324H
Duration — 50 minutes

LAST/FAMILY NAME: ____________________________________________
FIRST/GIVEN NAME: ____________________________________________

Do not turn this page until you have received the signal to start.
(In the meantime, please fill out the identification section above.)

This test consists of 4 questions on 6 pages (including this one).
When you receive the signal to start, please make sure that your copy
of the test is complete.

Good Luck!
# Question 1. [2 + 7 = 9 Marks] #

# Show the result values [2 Marks] and final memory model diagram [7 Marks] for: #

(define x 1)

(define f (λ (x)
    ; You may write this as “BODY1” in your diagram:
    (λ ()
        ; You may write this as “BODY2” in your diagram:
        (set! x (+ x 20))
        x))
)

((f 300))

((f 300))

(define g (f 4000))

(g)

(g)
Question 2. [8 Marks]

(A) [1 Mark] Write an example of an anonymous unary function:

```
(check-equal? ((inserter 'x) '(a b c)) '((x a b c)
             (a x b c) (a b x c) (a b c x)))
```

(B) Understand and design a function ‘inserter’.

```
(check-equal? ((inserter 'x) '(a b c))
             (a x b c) (a b x c) (a b c x)))
```

1 Mark] What is the arity of ‘inserter’, based on just that test case? Circle one:

- unary
- binary
- ternary
- variadic / variable-arity

1 Mark] What is the datatype of the return value of ‘inserter’, based on just that test case? Circle the most precise answer that applies:

- symbol
- list
- list of lists
- unary function
- unary predicate

- variadic / variable-arity function
- variadic / variable-arity predicate

1 Mark] What is the datatype of (inserter 'x) '(a b c) ?

Circle the most precise answer that applies:

- symbol
- list
- list of lists

[4 Marks] Based on the partial design below, write a full design for ‘inserter’. You may assume you know the list is non-empty. Use higher-order functions, including ‘fix-1st’, wherever appropriate. You may include more partial design that gets you closer to a full design, which can be worth partial credit if your full design is incorrect.

```
(check-equal? ((inserter 'x) '(a b c)) (list* (list* 'x '(a b c))
             (list (list* 'a '(x b c))
                   (list* 'a '(b x c))
                   (list* 'a '(b c x))))

(define (fix-1st f 1st) (λ (2nd) (f 1st 2nd)))
```
#| Question 3. [4 Marks] |#

#| Implement a macro / syntactic form for a short-circuiting ‘neither’.

It's also called ‘nor’, and here is the propositional logic definition:

\[
A \text{ nor } B \text{ nor } C \text{ nor } \cdots \text{ nor } Z \equiv \neg A \land \neg B \land \neg C \land \cdots \land \neg Z.
\]

For full marks use ‘...’ appropriately. |#

(check-true (neither))

(check-true (neither (= 123 324)
#false))

(check-false (neither (= 123 324)
  (= 324 (+ 1 323))
  (/ 1 0))))

#| Question 4. [4 Marks] |#

#| Implement ‘expand’ to turn shorthand unary function definition into core naming
   and function creation. Use pattern-matching and quasiquotation appropriately. |#

(check-equal? (expand '(define (f x) y))
  '(define f (\ (x) y)))

(check-equal? (expand '(define (g a) (define (f x) y)))
  '(define g (\ (a) (define (f x) y))))