UNIVERSITY OF TORONTO
Faculty of Arts and Science

Midterm (L0101/L2003)
CSC324H1F

October 30, 11am (**50 min.**)
Examination Aids: Aid sheet on back, detachable!

Name:

Student Number:

Please read the following guidelines carefully!

- Please write your name on the front **and back** of the exam.
- This examination has **4** questions. There are a total of **8 pages, DOUBLE-SIDED**.
- You may always write helper functions unless explicitly asked not to. You may not use mutation or any Racket iterative constructs.
- Any question you leave blank, or where you clearly cross out your work and write "I don't know," is worth **10% of the marks**.

Take a deep breath.

This is your chance to show us

How much you've learned.

We **WANT** to give you the credit

That you've earned.

A number does not define you.

Good luck!

1. **[5]** Consider the Racket function `intersperse`, which takes an item and a list, and returns a new list consisting of the items in the original list, with the new item inserted between each element. For example:

```
> (intersperse 1 '(2 3 4 5))
'(2 1 3 1 4 1 5)
> ; edge cases
> (intersperse 1 '())
'()
> (intersperse 1 '(2))
'(2)
```

Implement this function *without* using explicit recursion, but instead using higher-order list functions. (Solutions using explicit recursion will be awarded at most 3 marks.) You are encouraged to define your own helper functions. **The only list functions you may use are the ones found on the aid sheet.**

Hint: you may find it easier to generate a list like `'(1 2 1 3 1 4 1 5)` first.

2. (a) **[2]** Consider the following Racket function call expression, which contains other function call subexpressions. Assume all names refer to functions.

```
(a b (c b) (d (e f g) h (i)) (j k))
```

Using what you know about evaluation order in Racket, state the order in which the *function calls* in these expressions are evaluated. You may list either the functions which are called (e.g., "`c`"), or the entire expression (e.g., "`(c b)`") in your answer.

(b) **[3]** Consider the following function definition.

```
(define f (lambda () 10))
```

State the output of each of the following expressions in Racket. No explanation necessary.

```
> f
```

```
> (f)
```

```
> ((f))
```

(c) **[2]** Explain the meaning of **variable shadowing**, and give a brief example of it in Racket.

(d) **[2]** As you know, free identifiers are crucial to understanding the semantics of closures in Racket (and other languages). Give an example of a function definition that has at least one free and one bound identifier, and state what the free and bound identifier(s) are.

3. Consider the following Racket macro:

```
(define-syntax mymac
  (syntax-rules ()
    [(mymac (<a> <b>) ...) (list (<a> <b>) ...)]))
```

(a) **[4]** Give an example of a *correct* use of the macro which has at least two different arguments following the `mymac`. Show three things: (1) the macro expression `(mymac something ...)`, (2) what the expression *expands* into, and (3) what the expanded expression *evaluates* to.

(b) **[2]** Give two example uses of the above macro, one of which would produce a "bad syntax" error, and the other a runtime error. The one which produces a syntax error must not have any syntax errors in the individual `(<a> <b>)` subexpressions. State which one produces which, but no explanation is required.

(c) **[2]** Consider this proposed shortcut for the class macro to automatically define a Point class:

```
(define-syntax Point-class
  (syntax-rules ()
    [(Point-class) (define (Point x y)
                     (lambda (msg)
                       (cond [(equal? msg "x") x]
                             [(equal? msg "y") y]
                             [else "Unrecognized message!"]))))]))
```

Why can't we use this macro to do what we want?

4. **[6]** Consider the `class` macro provided on the aid sheet.

   We want to extend this macro to `class-private`, which requires the user to **follow** the public method declarations with zero or more *private methods*, which are accessible to the public methods, but not from outside the class (i.e., are "local" to the object).

   For simplicity, we'll treat each private method as a function that you can call using the normal syntax, without using "self." Example of the required syntax:

```
(class-private A (a)
  [(public-f x y) (+ (private-f x) y)] ; Note how private-f is called here.
  (private
    [(private-f z) (* a z)]
    [(private-other) (* a a)]
  ))

> (define obj (A 10))
> ((obj "public-f") 3 4)
34
> (obj "private-f")
"Unrecognized message!"
```

   `private` is a literal keyword which must appear at the beginning of the private method declarations.

   In the space below, define an extended class macro to enable this syntax. You should re-use most or all of the code we provided. Your macro should handle any number of private methods, but may assume that no private methods refer to each other or themselves. There can be any number of public methods and attributes, same as the regular class macro.

   Please add comments to identify what code you've added/changed for your TAs.

```
(define-syntax class-private
```

Use this page for rough work.

Use this page for rough work.

Name:

There were two proposed names for the language on Assignment 1. Circle the one you like more.

FunShake Jigglejavelin

	Q1	Q2	Q3	Q4	Total
Grade					
Out Of	5	9	8	6	28