CSC 324H5 F 2018 Midterm
Duration — 50 minutes
Aids allowed: none

Student Number: |_|_|_|_|_|_|_|_|_|_|

Last Name: ____________________ **First Name:** ____________________

Lecture Section: L0101 Test Version: A Instructor: Dan Zingaro

*Do **not** turn this page until you have received the signal to start.*
(Please fill out the identification section above, **write your name on the back of the test**, and read the instructions below.)
Good Luck!

This test consists of 4 questions on 10 pages (including this page). *When you receive the signal to start, please make sure that your copy is complete.*
Comments are not required except where indicated, although they may help us mark your answers. They may also get you part marks if you can't figure out how to write the code.
If you use any space for rough work, indicate clearly what you want marked.

# 1:	______/10
# 2:	______/ 5
# 3:	______/ 4
# 4:	______/ 6
TOTAL:	______/25

Question 1. [10 marks]

Answer each of the following multiple choice questions by circling all answers that apply. There may be multiple correct answers per question, or no correct answers at all.

1. Which of the following expressions match the macro pattern `(<a> <b> ...)`?

   (a) `(list)`

   (b) `(a b c)`

   (c) `(a) (b) (c)`

   (d) `(a ((b) (c)))`

2. Which of the following definitions are tail recursive?

   (a)
   ```
   (define (f lst acc)
     (if (empty? lst)
         acc
         (f (rest lst) (cons (first lst) acc))))
   ```
   (b)
   ```
   (define (f lst acc)
     (if (empty? lst)
         (f '() acc)
         (f (rest lst) (cons (first lst) acc))))
   ```
   (c)
   ```
   (define (f lst acc)
     (if (empty? lst)
         (* 10 acc)
         (f (rest lst) (+ 1 acc))))
   ```
   (d)
   ```
   (define (f lst acc)
     (if (empty? lst)
         acc
         (reverse (f (rest lst) (cons (head lst) acc)))))
   ```

3. Consider the following Haskell code:

   ```
   g1 x y = x + y
   g2 x y z = x + (y * z)
   ```

   Which of the following expressions evaluate to a function that takes exactly one argument and returns a number?

   (a) `g1 3`

   (b) `g2 3`

   (c) `\x -> g1 3`

   (d) `\x -> g2 x 4 5`

4. Which of the following are valid expressions of the below grammar?

```
<expr> = INTEGER
       | (+ <expr> <expr>)
       | (* <expr> <expr>)
       | (- <expr> <expr>)
       | (/ <expr> <expr>)
```

(a) `(2 + 3)`

(b) `(+ (/ 1 0) (* 3 (- 6 5)))`

(c) `(+ 1 (+ 0 0) (+ 0 0))`

(d) `+ 2 3`

5. Consider the following macro:

```
(define-syntax my-macro
  (syntax-rules ()
    [(my-macro (<x> ...))
     (list 'a <x> ...)]
    [(my-macro ((<x> <y> ...)))
     (list 'b <y> ...)]
    [(my-macro ((<x> <y>) ...))
     (list 'c <x> ...)]))
```

Which macro expansions below are correct?

(a) `(my-macro (1 2 3))` expands to `(list 'b 2 3)`

(b) `(my-macro (1))` expands to `(list 'a 1)`

(c) `(my-macro (1 2))` expands to `(list 'c 1)`

(d) `(my-macro (1 2) (3 4))` expands to `(list 'c 1 3)`

Question 2. [5 marks]

Write a macro `my-rest` that skips the first element of a list defined using the `list` constructor. For example:

```
>>> (my-rest (list 1 2 3 4 5))
'(2 3 4 5)
>>> (my-rest (list 4 5))
'(5)
>>> (my-rest (list 6))
'()
>>> (my-rest (4 5))
; my-rest: bad syntax in: (my-rest (4 5))
```

Note that `my-rest` would not work with lists constructed using other means, like `cons` or `quote`.

```
(define-syntax my-rest
  (syntax-rules (                              )
```

Question 3. [4 marks]

The following Haskell data type encodes arbitrarily nested subtraction expressions.

```
data SubExpr = SInt Int
             | Sub SubExpr SubExpr
```

For example, the mathematical expression `(4 - (3 - 2))` is expressed as

```
(Sub (SInt 4) (Sub (SInt 3) (SInt 2)))
```

Write a function `evalSub` that evaluates arbitrarily nested `SubExpr` expressions. For example, the function call `evalSub (Sub (SInt 4) (Sub (SInt 3) (SInt 2)))` should return `3` since `(4 - (3 - 2)) = 3`.

```
evalSub :: SubExpr -> Int
```

Question 4. [6 MARKS]

Part (a) [2 MARKS]

Briefly (in two or three sentences) explain why we use a hash table instead of a `cond` when implementing Racket objects.

Part (b) [2 MARKS]

In Lab 5, you implemented a constructor using an explicit method named `new`. What is the benefit of this over the approach to constructors that we used in lecture?

Part (c) [2 MARKS]

Briefly explain why `fix-first` is useful for implementing `self`.

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]