PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL 2017 EXAMINATIONS

CSC 104 H1S
Instructor(s): G. Baumgartner

Duration — 3 hours

No Aids Allowed

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

Last (Family) Name(s): ______________________________

First (Given) Name(s): ______________________________

Do ***not*** *turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

This Final Examination paper consists of 9 questions on 18 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the paper is complete and fill in your name and student number above.*

Answer each question directly on this exam paper, in the space provided. If you need more space for one of your solutions you may also use a "blank" page at the end of the paper (in which case make sure to mention that where the question is asked).

If you leave a Question blank, or a Part of a Question blank, or clearly cross out your answer with a diagonal line, you will receive 25% of the marks allocated to that Question or Part.

Marking Guide

# 1:	______/ 6
# 2:	______/ 9
# 3:	______/ 6
# 4:	______/15
# 5:	______/13
# 6:	______/ 4
# 7:	______/ 8
# 8:	______/12
# 9:	______/ 8
TOTAL:	______/81

Good Luck!

Reminders

The **datatypes** used in **contracts**:

```
  image
  number
  string
  boolean
  list
  function
```

The **Intermediate Steps** for `map`, `apply`, and `repeated`:

```
; map :  function  list  ->  list

(map f (list    a     b     c  ...))

  =>   (list (f a) (f b) (f c) ...)


; map : function  list  list  ->  list

(map f (list    a       b       c    ...)
       (list    x       y       z    ...))

  =>   (list (f a x) (f b y) (f c z) ...)


; apply :  function  list  ->  any

(apply f (list a b c ...))

  =>     (f    a b c ...)


; repeated :  function  any  number  ->  list

(repeated f a n)

  =>  (list a (f a) (f (f a)) ...) ; with n elements.
```

Question 1. [6 marks]

Part (a) [3 marks]

Convert the number 104 into its binary representation, showing your work.

Part (b) [3 marks]

The number 4203 has the binary representation: 1000001101011 .
Use that fact to determine the binary representations of:

- 8406
- 4202
- 2101

Include a brief explanation or demonstration of how you used the representation of 4203 to produce the representations of the other three numbers.

Question 2. [9 MARKS]

Read the following definition of a function T:

```
(define (T n)
  (cond [(= n 0) (triangle 10 "solid" "black")]
        [else (above (T (- n 1))
                     (flip-vertical
                      (beside (T (- n 1))
                              (T (- n 1)))))]))
```

Part (a) [3 MARKS] Draw the values of `(T 0)` and `(T 1)`:

Part (b) [2 MARKS]

Complete this **design** `check-expect`, using "`(T 1)`" [do not draw anything]:

```
(check-expect (T 2)
```

Part (c) [4 MARKS] Draw the values of `(T 2)` and `(T 3)`:

Question 3. [6 marks]

Read the definition of function `T-count`, which produces the number of triangles used to produce the result of the function `T` from Question 2.

```
; T-count :  number  ->  number
(define (T-count n)
  (cond [(= n 0) 1]
        [else (+ (T-count (- n 1))
                 (T-count (- n 1))
                 (T-count (- n 1)))]))
```

Part (a) [3 marks]

Briefly explain why `T-count` takes a **very** long time to calculate `(T-count 100)`.

Part (b) [3 marks]

Suggest a small change in the `[else ...]` clause of the body of `T-count`, that is still recursive [it must use `(T-count (- n 1))` at least once], but produces the same result in much less time.
Briefly explain why your change allows `(T-count 100)` to produce its result quickly.

Question 4. [15 MARKS]

Part (a) [1 MARK] Read the following definition of a function `C`.

Fill in the contract for `C`..

```
; C :              ->

(define (C n)
  (circle (* 10 (+ 2 n)) "outline" "black"))
```

Part (b) [2 MARKS]

Show at least one intermediate step, and the final result value, for the following expression:

```
(C -1)
```

Part (c) [3 MARKS] Read the following definition of a function `next`.

```
; next :  list-of-two-numbers  ->  list-of-two-numbers

(define (next pair)
  (list (- (first pair))
        (+ (second pair) (first pair) 1)))
```

For each of the following expressions, show at least one intermediate step, and the final result value:

```
(next (list -1 1))
```

```
(next (list 1 1))
```

Part (d) [5 MARKS]

Show [at least] the intermediate step for `repeated` [according to the step for `repeated` listed on the second page of this exam], and the final result value, for the following expression:

```
(repeated next (list -1 1) 5)
```

Part (e) [4 MARKS] Read the following definition of a function `draw`.

```
; draw :  list-of-two-numbers  ->  image

(define (draw pair)
  (beside (C (first pair))
          (C (second pair))))
```

Show [at least] the intermediate step for `map` [according to the step for `map` listed on the second page of this exam], and the final result value, for the following expression.
You may use your final result value from Part (d) of this question.

```
(map draw (repeated next (list -1 1) 5))
```

Question 5. [13 MARKS]

Read the following check-expects documenting two functions prepend-0 and prepend-1:

```
(check-expect (prepend-0 "101") "0101") ; Puts a zero at the beginning.
(check-expect (prepend-0 "001") "0001") ; Puts a zero at the beginning.

(check-expect (prepend-1 "101") "1101") ; Puts a one at the beginning.
(check-expect (prepend-1 "001") "1001") ; Puts a one at the beginning.
```

Part (a) [4 MARKS]

Define the two functions prepend-0 and prepend-1, including their contracts.

Part (b) [4 MARKS]

Consider the following check-expects **documenting** a function count:

```
(check-expect (count 1) (list ""))

(check-expect (count 2) (list "0" "1"))

(check-expect (count 4) (list "00" "01" "10" "11"))

(check-expect (count 8) (list "000" "001" "010" "011" "100" "101" "110" "111"))
```

Complete the following as a **design** check-expect by using (count 4), and the functions prepend-0 and prepend-1, to produce the the list for (count 8) documented above.

```
(check-expect (count 8)
```

Part (c) [5 marks]

Define the function count, including its contract.
You can assume the input is a power of 2.

```
; count :            ->

; For a number n that is a power of 2 [for example: 1, 2, 4, 8, 16, ...],
;  produce a list of strings with the binary representations of 0, 1, 2, ..., n-1.

(define (count n)
```

Question 6. [4 marks]

Recall the the built-in function explode that takes a string and produces a list of each character from it.

```
; explode : string  ->  list-of-strings

(check-expect (explode "It isn't?") (list "I" "t" " " "i" "s" "n" "'" "t" "?"))
```

Read this documentation check-expect and contract for a function string-reverse to reverse a string.

```
(check-expect (string-reverse "It isn't?") "?t'nsi tI")

; string-reverse :  string  ->  string
```

Use explode to help you complete the definition of the function string-reverse:

```
(define (string-reverse a-string)
```

Question 7. [8 MARKS]

Part (a) [4 MARKS] Complete the definition of function q?.

```
; q? :  string  string  number  ->  boolean
;
; Produce #true if the length of ‘string-0’ is less than the length of ‘string-1’,
;  or the length of ‘string-1’ is more than ‘n’ [otherwise produce #false].

(define (q? string-0 string-1 n)
```

Part (b) [4 MARKS] Write the definition of function p?.

```
; p? :  image  image  ->  boolean
;
; Produce #true if the width and height of the first image are both less than the
;  width and height of the second image [otherwise produce #false].
```

Question 8. [12 marks]

Part (a) [6 marks] Read the following definition of a function `R`:

```
(define (R LoL)
  (cond [(list? LoL) (reverse (map R LoL))]
        [else LoL]))
```

Show the final result value of the following expression:

```
(R 1)
```

For the following expression, show [at least] each intermediate step before and after a `map` is performed, and show the value of the final result:

```
(R (list 3 4))
```

For the following expression, show [at least] each intermediate step before and after a `map` is performed, and show the value of the final result.
If a step uses `(R (list 3 4))` you do not need to show the steps for `(R (list 3 4))` again.

```
(R (list 1 2 (list 3 4)))
```

Part (b) [6 MARKS] Read the following definition of a function `F`:

```
(define (F LoL)
  (cond [(list? LoL) (reverse (apply append (map F LoL)))]
        [else (list LoL)]))
```

Show the final result value of the following expression:

```
(F 1)
```

For the following expression, show [at least] each intermediate step before and after a `map` or `apply` is performed, and show the value of the final result:

```
(F (list 3 4))
```

For the following expression, show [at least] each intermediate step before and after a `map` or `apply` is performed, and show the value of the final result.
If a step uses `(F (list 3 4))`, you do not need to show the steps for `(F (list 3 4))` again.

```
(F (list 1 2 (list 3 4)))
```

Question 9. [8 marks]

Part (a) [4 marks] Show the result value for each of these expressions:

```
(filter list? (list (list 1 2 3) 4 (list (list 5) 6) 7 8 (list 9 10)))
```

```
(length (filter list? (list (list 1 2 3) 4 (list (list 5) 6) 7 8 (list 9 10))))
```

Part (b) [4 marks] Show the intermediate steps and final result value for each of these expressions:

```
(map rest (list (list 1 2 3 4) (list 5 6) (list 7 8 9)))
```

```
(map +
     (list 1 2 3)
     (list 4 5 6))
```

Use the space on this "blank" page for scratch work, or for any answer that did not fit elsewhere.
Clearly label each such answer with the appropriate question and part number.

Use the space on this "blank" page for scratch work, or for any answer that did not fit elsewhere.
Clearly label each such answer with the appropriate question and part number.

Use the space on this "blank" page for scratch work, or for any answer that did not fit elsewhere.
Clearly label each such answer with the appropriate question and part number.

Please write nothing on this page.