PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER 2015 EXAMINATIONS

CSC 104 H1F
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 7 questions on 17 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).

You will earn 20% for any question you leave blank or you write "I cannot answer this question".

The page after the last question has reminders about the meaning of some Intermediate Student Language functions, including the intermediate steps for `map` and `apply`.

Marking Guide

# 1:	/ 8
# 2:	/10
# 3:	/17
# 4:	/12
# 5:	/10
# 6:	/12
# 7:	/ 6
TOTAL:	/75

Good Luck!

Question 1. [8 marks]

Part (a) [3 marks]

Convert the binary representation 1001111 to its decimal representation, briefly showing your steps.

Part (b) [3 marks]

The number 6000 has the binary representation 1011101110000.
Use that to determine the binary representations of 3000, 12000, and 12003, briefly showing/describing your reasoning.

Part (c) [2 marks]

The binary representation of 16383 is 11111111111111 [that has fourteen 1s in it].
Use that to determine the binary representation of 16384, briefly showing/describing your reasoning.

Question 2. [10 marks]

Consider the function `c`:

```
; c : number -> image
(define (c n)
  (cond [(zero? n) (circle 10 "outline" "black")]
        [else (above (c (sub1 n))
                     (beside (c (sub1 n))
                             (square (image-width (c (sub1 n))) "outline" "black")
                             (c (sub1 n)))
                     (c (sub1 n)))]))
```

Part (a) [4 marks]

Draw the values of `(c 0)` and `(c 1)`:

Part (b) [6 marks]

Draw the values of `(c 2)` and `(c 3)`:

Question 3. [17 MARKS]

Consider the following `check-expect`s describing two functions `prepend-0` and `prepend-1`:

```
(check-expect (prepend-0 (list 1 0 1)) (list 0 1 0 1))
(check-expect (prepend-0 (list 0 0 1)) (list 0 0 0 1))

(check-expect (prepend-1 (list 1 0 1)) (list 1 1 0 1))
(check-expect (prepend-1 (list 0 0 1)) (list 1 0 0 1))
```

Part (a) [4 MARKS]

Define those functions `prepend-0` and `prepend-1`, including their contracts.

Part (b) [4 MARKS]

Consider the following `check-expect`s describing a function `count`:

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

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

(check-expect (count 4) (list (list 0 0)
                              (list 0 1)
                              (list 1 0)
                              (list 1 1)))

(check-expect (count 8) (list (list 0 0 0)
                              (list 0 0 1)
                              (list 0 1 0)
                              (list 0 1 1)
                              (list 1 0 0)
                              (list 1 0 1)
                              (list 1 1 0)
                              (list 1 1 1)))
```

Complete the following as a "design" `check-expect` by using `(count 4)` and the functions `prepend-0` and `prepend-1`:

```
(check-expect (count 8)













)
```

Part (c) [9 marks]

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

Question 4. [12 MARKS]

Consider the function `d`:

```
(define (d LLL)
  (cond [(list? LLL) (+ 1 (apply max (map d LLL)))]
        [else 0]))
```

Part (a) [3 MARKS]

Show the values of the following expressions:

```
(d "a")
```

```
(d (list "b" "c" "d"))
```

Part (b) [5 MARKS]

Show the value of the following expression.
Show [at least] every intermediate step immediately before and after `apply` or `map` is used, except you may use the result values from Part (a) without showing their steps here.

```
(d (list "a" (list "b" "c" "d")))
```

Part (c) [4 marks]

Determine the value of the following expression.
Include a [possibly brief] explanation for how you determined the value, for example by showing some intermediate steps, or explaining it in words.

```
(d (list (list "a"
               (list "b"
                     "c"))
         (list "d"
               (list (list "e"
                           "f")
                     "g"))))
```

Question 5. [10 MARKS]

Consider the following definitions:

```
; C : number -> image
(define (C n)
  (cond [(= n 0) (circle 10 "outline" "black")]
        [else (circle 10 "solid" "black")]))

(define-struct status (a b))

; draw : status -> image
(define (draw s)
  (beside (C (status-a s)) (C (status-b s))))

; update : status -> status
(define (update s)
  (cond [(= (status-a s) 0) (make-status 1 0)]
        [(= (status-b s) 1) (make-status 0 0)]
        [else (make-status (status-a s)
                           (+ 1 (status-b s)))]))
```

Part (a) [6 MARKS]

Show the values of each of the following expressions:

```
(C 0)
```

```
(C 1)
```

```
(draw (make-status 0 0))
```

```
(update (make-status 0 0))
```

```
(update (make-status 1 0))
```

Part (b) [4 marks]

For the following `big-bang` expression show four frames of the animation.
Include a brief explanation of how you determined each value, for example by showing some intermediate steps, or explaining it in words.

```
(big-bang (make-status 0 0)
          [to-draw draw]
          [on-tick update])
```

Question 6. [12 marks]

Consider the following definitions.

```
(define image
  (beside (square 2 "solid" "black") (square 2 "solid" (make-color 128 128 128 255))))

; List of the 'color's in 'image', starting with the first row, then the second row, etc.
(define pixels (image->color-list image))
```

Part (a) [6 marks]

Show the values of each of the following expressions:

`image`

`(length pixels)`

`(first pixels)`

`(first (reverse pixels))`

`pixels`

Part (b) [6 MARKS]

Determine the values of each of the following expressions.
Include a brief explanation of how you determined each value, for example by showing some intermediate steps, or explaining it in words.
Draw the results large enough to clearly show the pixels.

```
(color-list->bitmap (append pixels pixels) 4 4)
```

```
(color-list->bitmap (append pixels pixels) 8 2)
```

```
(color-list->bitmap (append pixels pixels) 2 8)
```

```
(color-list->bitmap (apply append (map list pixels pixels)) 8 2)
```

Question 7. [6 MARKS]

Show the values of the three non-definition expressions below:

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

```
(apply append
       (map list
            (list 1 2 3)
            (list 4 5 6)))
```

```
(define LLL (list "a" (list "b" "c")))

(apply append
       (map list
            LLL
            LLL))
```

Reminders.

The intermediate steps for `map` and `apply`:

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

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


(map f
    (list    a-0          b-0          c-0       ...)
    (list        a-1          b-1          c-1  ...))

    (list (f a-0 a-1) (f b-0 b-1) (f c-0 c-1) ...)


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

            (f a b c ...)
```

The function `max` has contract:

```
  max : number number ... -> number
```

`(max a b c ...)` is the maximum of the numbers `a`, `b`, `c`, ...

The function `sub1`:

```
  (check-expect (sub1 104) 103)
```

The function `make-color` has contract:

```
  make-color : number number number number -> color
```

`(make-color red green blue opacity)` is the color with the given `red`, `green`, and `blue` intensities, and amount of `opacity`. Those amounts are numbers in `(range 0 256 1)`.

The function `color-list->bitmap` has contract:

```
  color-list->bitmap : list-of-colors number number -> image
```

`(color-list->bitmap a-list-of-colors a-width a-height)` is the image made by putting the colors from `a-list-of-colors` into a rectangle of width `a-width` and height `a-height`, row by row.

CONT'D...

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.