THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 110: MIDTERM EXAMINATION

Name: ________________________ Student #:_________________ CS Dept. ID #:__________

Signature: _____________________ **Lab Section:_____**

Important notes about this examination

1. You have 90 minutes to write this examination.
2. Except for question 1, this exam will be graded significantly on how well you **follow the design recipes**. You have been given a copy of the Recipe Summary and Template Rules. Use them!
3. Put away your books, notebooks, laptops, cell phones... everything but pens, pencils, erasers and this exam.
4. Good luck!

Rules Governing Formal Examinations

1. Each candidate must be prepared to produce, upon request, a UBCcard for identification.
2. Candidates are not permitted to ask questions of the invigilators, except in cases of supposed errors or ambiguities in examination questions.
3. No candidate shall be permitted to enter the examination room after the expiration of one-half hour from the scheduled starting time, or to leave during the first half hour of the examination.
4. Candidates suspected of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action:
    - having at the place of writing any books, papers or memoranda, calculators, computers, sound or image players/recorders/transmitters (including telephones), or other memory aid devices, other than those authorized by the examiners;
    - speaking or communicating with other candidates; and
    - purposely exposing written papers to the view of other candidates or imaging devices. The plea of accident or forgetfulness shall not be received.
5. Candidates must not destroy or mutilate any examination material; must hand in all examination papers; and must not take any examination material from the examination room without permission of the invigilator.
6. Candidates must follow any additional examination rules or directions communicated by the instructor or invigilator.

Please do not write in this space:

Question	Mark
1	
2	
3	
4	
5	
6	

Problem 1 - Mechanisms [6 points]

(A) What is the result of evaluating

```
(string-append "x" "y")

“xy”   [1 point for exact answer w/ string quotes]
```

(B) What is the result of evaluating

```
(cons (+ 2 3) (cons 2 empty))

(cons 5 (cons 2 empty))  [1 point for exact answer]
```

(C) Given

```
(define (zow x)
  (if (< x 3)
      (+ 1 2)
      (+ 3 4)))
```

Show the hand evaluation for the following expression. Be sure to show every step.

```
(zow 4)  [1 pt for each of 4 following steps]

(if (< 4 3)
    (+ 1 2)
    (+ 3 4))

(if false
    (+ 1 2)
    (+ 3 4))

(+ 3 4)

7
```

Problem 2 - Types Comments

In this problem you will be given small fragments of problem descriptions. Each fragment describes some information in a problem domain that must be represented using data in a program. In each case you need to write a types comment that would form the basis of a data definition for representing this information. If you use compound data also write the appropriate `define-struct` before the types comment. You do not need to write a complete data definition.

You may find it helpful in some cases to assume you are working on a typical world program, in which typical constants like WIDTH, HEIGHT, etc. have been defined. But you do not need to know the specific values of any of those constants. You may make any other reasonable assumptions you wish, but if they are essential to the correctness of your answer write your assumption down.

(A) a message to display, that has both text and color [4 points]

```
(define-struct msg (txt color))
;; Message is (make-message String Color)

[1 point for define-struct w 2 fields]
[1 point if struct and field names are reasonable]
[2 points for properly formed compound type comment]
[-1 point for any of:
[   improperly cased names
[   unreasonable type names
```

(B) the state of a crossing gate, which is either up or down

```
;; Gate is one of:
;;  - "open"
;;  - "closed"

[1 point for enumeration ]
[1 point for using strings rather than 0/1 ]
[ Gate is Boolean is also ok for 2 points]
```

Problem 3 - Completing Data Definitions

Given the following (nonsensical) types comments, complete the data definition with example data, a template and a list of the template rules used. You do not need to provide an interpretation.

(A) [4 points]

```
[1 point each for correct:
[  1 example
[  template (with proper name, param, body)
[  template rule

;; Farfoozle is String
(define F1 "a")

(define (fn-for-farfoozle f)
  (... f))

;; Template rules used:
;;  - atomic non distinct: String
```

(B) [7 points]

```
[1 point each for correct:
[  2 example (1 point each)
[  template (with proper name, param, body)
[  each of 3 template rules, part after : is optional

;; Harumph is one of:
;;  - Integer[7, 10]
;;  - "mumble"
(define H1 7)
(define H2 "mumble")

(define (fn-for-harumph h)
  (cond [(integer? h) (... h)]
        [else     ; could also be (string? h)
                  ; or (and (string? h) (string=? h "mumble"))
         (...)]))

;; Template rules used:
;;  - one of: 2 cases
;;  - atomic non distinct: (Integer[7, 10])
;;  - atomic distinct: "mumble"
```

Problem 4 - Designing Data Definitions

Given the following fragment of a problem description you should design a complete data definition for representing this information.

This question will be graded on how well you follow the design recipe for data definitions.
(A)
the name of a city [5 points]

```
[1 point each for correct:
[  type comment
[  interp.
[  1 example
[  template (with proper name, param, body)
[  template rule
;; CityName is String
;; interp. the name of a city
(define CN1 "Vancouver")

(define (fn-for-city-name cn)
  (... cn))

;; template rules used:
;;  - atomic non-distinct: String
```

(B)
the first name, last name, and CS department id of a student [9 points]

```
[1 point each for correct:
[  type comment
[  interp
[  2 examples (1 point each)
[  template (with proper name, param, body)
[  each of 4 template rules, part after colon is optional

(define-struct student (fn ln id))
;; Student is (make-student String String String)
;; interp. a student, with a first name, last name and id number
(define S1 (make-student "Harry" "Potter" "A1B2"))

(define (fn-for-student s)
  (... (student-fn s)
       (student-ln s)
       (student-id s)))

;; Template rules used:
;;  - compound: 3 fields
;;  - atomic non-distinct: String
;;  - atomic non-distinct: String
;;  - atomic non-distinct: Integer
```

Problem 5 - Designing Functions

Consider the following data definition:

```
(define-struct box (w h))
;; Box is (make-box Integer Integer)
;; A box with width and height.
(define B1 (make-box 2 3))
#;
(define (fn-for-box b)
  (... (box-w b)
       (box-h b)))
```

Design a function called `tippy?` to compute whether a box might be prone to tipping over. If the width is greater than or equal to the height, the box is stable; but if the height is greater than the width the box is tippy. This question will be graded on how well you follow the design recipe for functions. Be sure to show ALL the parts of the function design the recipe calls for. Please write down the stub and label it. You do not need to write a separate copy of the template.

```
[ 1 point each for
[   signature (must produce Boolean)
[   each of 3 tests
[   proper stub
[   function clearly follows template
[   function is correct

;; Box -> Boolean
;; produce true if b's height is > width
(check-expect (tippy? (make-box 10 20)) true)
(check-expect (tippy? (make-box 20 20)) false)
(check-expect (tippy? (make-box 20 10)) false)
#;
(define (tippy? b) true)  ; stub

(define (tippy? b)
  (> (box-h b)
     (box-w b)))
```

Problem 6 - Completing a World Program

Together with this exam you have been given a partially completed world program. The intent for this world program is to animate balloons floating upwards in a box, clicking the mouse adds a new balloon at that position. Before attempting this problem you will need to review the partially complete program.

As you can see the partial program is missing both the `render-lob` and `tick-lob` functions, which are specified as the to-draw and on-tick handlers for big-bang.

Choose <u>either</u> `render-lob` or `tick-lob` and provide a complete design for the function. The helper function guidelines may cause you to need a helper function, in which case you should design that helper function also. This question will be graded on how well you follow the design recipe for functions. Note, `render-lob` with be worth marginally more points than `tick-lob`. Do all your work on the exam sheets, not on the attached partial program. Do not design both hoping the grader will choose the best. If you want to design both `render-lob` and `tick-lob` and choose one then you need to mark clearly which one you want us to grade.

```
;; 1 pt each for:
;;  signature
;;  purpose (must be specific)
;;  empty test first
;;  test at least 2 long
;;  fn clearly follows template
;;    (2 case cond, natural recursion undamaged etc.)
;;  base case follows base test
;;  recursive case correct
;; 2 extra points for proper function composition in 2nd cond
clause
;; ListOfBalloon -> Image
;; produce image of all balloons in lob on MTS
(check-expect (render-lob empty)
              MTS)
(check-expect (render-lob (cons (make-balloon 3 5)
                                (cons (make-balloon 5 6)
                                      empty)))
              (place-image BALLOON-IMG
                           3
                           5
                           (place-image BALLOON-IMG
                                        5
                                        6
                                        MTS)))

(define (render-lob lob)
  (cond [(empty? lob) MTS]
        [else
         (place-balloon (first lob)
                        (render-lob (rest lob)))]))

;; 1 pt each for:
;;  signature
;;  purpose (must be specific)
;;  1 good test
;;  fn clearly follows template
;;  fn clearly follows example

;; Balloon Image -> Image
;; Render b onto img
(check-expect (place-balloon (make-balloon 110 120) MTS)
              (place-image BALLOON-IMG 110 120 MTS))

(define (place-balloon b img)
  (place-image BALLOON-IMG
               (balloon-x b)
               (balloon-y b)
               img))
```

```
;; 1 pt each for:
;;  signature
;;  purpose (must be specific)
;;  empty test first
;;  test at least 2 long
;;  fn clearly follows template
;;    (2 case cond, natural recursion undamaged etc.)
;;  base case follows base test
;; ListOfBalloon -> Image
;; produce image of all balloons in lob on MTS
;; ListOfBalloon -> ListOfBalloon
;; advance each balloon's height by BSPEED
(check-expect (tick-lob empty) empty)
(check-expect (tick-lob (cons (make-balloon 3 4)
                              (cons (make-balloon 90 100)
                                    empty)))
              (cons (make-balloon 3 (- 4 BSPEED))
                    (cons (make-balloon 90 (- 100 BSPEED))
                          empty)))

(define (tick-lob lob)
  (cond [(empty? lob) empty]
        [else
         (cons (tick-balloon (first lob))
               (tick-lob (rest lob)))]))

;; 1 pt each for:
;;  signature
;;  purpose (must be specific)
;;  1 good test
;;  fn clearly follows template
;;  fn clearly follows example

;; Balloon -> Balloon
;; advance balloon's height by BSPEED (so decrease its y)
(check-expect (tick-balloon (make-balloon 6 9))
              (make-balloon 6 (- 9 BSPEED)))

(define (tick-balloon b)
  (make-balloon (balloon-x b) (- (balloon-y b) BSPEED)))
```