THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 110: MIDTERM EXAMINATION  OCTOBER 5TH, 2011

Name: ________________________  UBC Student #: ___________  CS Dept. ID #: __________

Signature: ___________________________

Lab Section: ___________       Lecture Section (circle):   101 Joanna   102 Chen  103 Gregor

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 Exam Sheet. Use it!
3. Put away books, papers, laptops, cell phones... everything but pens, pencils, erasers and this exam.
4. Good luck!

VERY IMPORTANT NOTE
Each question is graded on its own separate point scale. Then we compute a percentage for each question. Finally we use the relative weight of each question to combine the per question scores.

What this means is that you should not attempt to compare points from one question with points from another question.

Please do not write in this space:

<table>
<thead>
<tr>
<th>Question</th>
<th>Grader</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Percentage of grade:
Questions 1 and 2 will total 15%
Questions 3, 4 and 5 will total 55%
Questions 6 and 7 will total 30%
Problem 1 - Mechanisms       [6 POINTS TOTAL]

(A) What is the value of

\[(\star (\+ 3 1) (\div 6 (\+ 1 2))))\]

\[8\]    [2 POINTS]

(B) Given

\[(\text{define} (\text{foo} \, x))\]
\[\begin{align*}
\text{(cond} & \left[ \begin{array}{l}
(\text{<} \, x \, 0) \, \text{"n"} \\
(\text{=} \, x \, 0) \, \text{"0"} \\
\text{else} \, \text{"p"}]
\end{array} \right)\]
\end{align*}\]

what is the value of:

\[(\text{foo} \, 6)\]

"p"    [2 POINTS, ONLY 1 IF STRING QUOTES MISSING]

(C) Given

\[(\text{define-struct} \, \text{snitch} \, (a \, b))\]

\[(\text{define} \, \text{A} \, (\text{make-snitch} \, 1 \, 2))\]
\[(\text{define} \, \text{B} \, (\text{make-snitch} \, 3 \, 4))\]

what is the value of:

\[(\+ (\text{snitch-a} \, \text{A}) \, (\text{snitch-b} \, \text{B}))\]

\[5\]    [2 POINTS]
Problem 2 - Mechanisms [12 POINTS]

Given

(define (bar x y)
 (if (< x y)
  (* x 2)
  (* y 3)))

Show the step-by-step evaluation of the following expression. Be sure to show every step.

(bar (+ 1 2) (+ 3 4))

[2 points for each correct step present in answer
steps must be completely correct except
arithmetic errors OK]

(bar 3 (+ 3 4))

(bar 3 7)

(if (< 3 7)
  (* 3 2)
  (* 7 3))

(if true
  (* 3 2)
  (* 7 3))

(* 3 2)

6
Problem 3 - Types Comments  [10 points]

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.

(A) your mood as shown in a Facebook app, which is happy, sad or asleep

;; Mood is one of:
;;    - "happy"
;;    - "sad"
;;    - "asleep"

[one of 3 atomic distinct gets    3 points]
[3 reasonable strings             2 more points]
[extra words or syntax take back 2 points]

(B) a student's age, in whole years

;; Age is Natural                 [5 points]

;; Age is Integer[>=0]            [3 points]
;; Age is Integer[0, infinity]    [3 points]
;; Age is Natural[1, 100]         [3 points]
;; Age is Number                   [1 point]

[extra words or syntax take back 2 points]
Problem 4 - Completing Data Definitions

For each of 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) [9 points]

;; Murf is Natural

(define M1 2)              [2 points for reasonable example

(define (fn-for-murf m)    [1 point for proper name
 (... m))                 [3 points for proper body

;; Template rules used:
;; - atomic non-distinct: Natural [3 points if only rule
 [ is atomic non distinct
 [ : Natural need not be there

(B) [13 points]

;; Splurf is one of:
;; - "drumping"
;; - Natural

(define M1 "drumping")              [1 each for 2 examples
(define M2 3)

(define (fn-for-splurf s)           [1 for proper fn name
 (cond [(string? s) (...)]         [1 each for 4 proper
 [else (... n))]))            [ cond questions/answers
 [ -2 if cond not used

;; Template rules used:
;; - one of: 2 cases [2 parts after : not needed
;; - atomic distinct: "Drumping" [2
;; - atomic non-distinct: Natural [2
Problem 5 - Designing Data Definitions

In this problem you will design complete data definitions. In each case you will be given fragments of a problem description. Each describes some information in a problem domain that must be represented using data in a program. In each case you need to design a complete data definition for representing this information. You should consider part A and part B to be SEPARATE programs.

This question will be graded on how well you follow the design recipe for data definitions.

(A) [10 points]
the name of a pet cow

;; Cow is String  [2 for X is String
;; interp. the name of a cow  [2 for reasonable interp.
[-1 if doesn’t say interp.

(define C1 "Bessie")  [2 for 1 good example

(define (fn-for-cow c)  [1 for proper name
  (... c))  [1 for proper body

;; Template rules used:
;; - atomic non-distinct: String  [2 for proper rule
[ : String not needed
[take away 1 for
extra improper rules
(B) [14 points]
a magic wand with two properties: its length and the wood it is made from (for example
10.75 inches, vine wood)

(define-struct wand (l w))
;; Wand is (make-wand Number String)
;; interp. A wand, with length in inches and wood

(define W1 (make-wand 10.75 "vine wood"))

(define (fn-for-wand w)
  (... (wand-l w)
       (wand-w w)))

;; Template rules used:
;; - Compound: 2 fields
;; - Atomic non-distinct: (wand-l w) is Number
;; - Atomic non-distinct: (wand-w w) is String

[Points as follows:
[ 2 for 2 field struct
[ 2 for type comment
[ 2 1 each for field types (must be Number String)
[ 1 for reasonable interp (0 if doesn’t say interp.)
[ 1 for 1 good example
[ 3 for proper template w/ two selectors
[ 3 1 for each correct template rule
[ only part before : required]
Problem 6 - Designing Functions

As part of your summer job you are working on a system that keeps track of problem reports for your company’s product. Each problem report is represented as a TroubleTicket, as described below:

```
(define-struct tckt (sub date desc))
;; TroubleTicket is (make-tckt String String String)
;; interp. A problem report where:
;;   sub   is the email address of the person who submitted ticket
;;   date  is the date ticket was submitted
;;   desc  is the text of the problem report
(define T1 (make-tckt "customer@acme.com" 
                      "10/31/2010" 
                      "Pressing the close button doesn’t close!")
#;
(define (fn-for-tckt t)
  (... (tckt-sub t) (tckt-date t) (tckt-desc t)))
```

Part of the trouble ticket system helps to categorize reports. Your boss, who knows very little about software and even less about what really makes a problem hard, asks you to: Provide a complete design for a function to determine whether a problem is complex or not. Consider it complex if its description is more than 25 characters long. (Feel free to use reasonable, but precise abbreviations in any check-expects you write.)

[12 points
[2 for signature: 1 for each correct type (must be TroubleTicket and Boolean)
[2 for purpose: 1 each for desc length > 25 and when produces true
[1 for stub with correct result type
[1 for covering each of 3 test cases (< = >)
[2 if function body includes “(tckt-desc t)” (is based on template)
[1 for > 25
[1 for string-length (or reasonable approximation)
;; TroubleTicket -> Boolean
;; produce true if desc length is > 25
(check-expect
  (complex? (make-tckt "h@b.com" "10/30/2010" "1234"))
  false)
(check-expect
  (complex? (make-tckt "h@b.com" "10/30/2010" "<string 25 long>")
  false)
(check-expect
  (complex? (make-tckt "h@b.com" "10/30/2010" "<string 26 long>")
  true)

;(define (complex? t) false) ;stub

(define (complex? t)
  (> (string-length (tckt-desc t)) 25))
```
Problem 7 - Completing a World Program

In this problem you will complete a world program. First read the description of the program's behavior below. Then read through the program, completing it in the marked places. This problem will be graded primarily on your proper use of the HtDF recipe.

The program shows an animation of a “cosmic circle” that grows from just a small dot (radius = 1) in the middle of the screen until it reaches a certain size (radius = 100), then starts out small again. Your computer science professors, who are from a different era, think this will look really “cool”.

;; Cosmic Circles

(require 2htdp/image)
(require 2htdp/universe)

;; Constants:
(define WIDTH 400)
(define HEIGHT 400)
(define SPEED 1) ; amount circle’s radius increases per tick
(define START 1) ; initial radius of circle
(define STOP 100) ; final radius of circle
(define CTR-X (/ WIDTH 2))
(define CTR-Y (/ HEIGHT 2))

(define MTS (empty-scene WIDTH HEIGHT))

;; Data definitions:

;; Circle is Natural[START, STOP]
;; interp. the radius of the cosmic circle
(define C1 START)

(A) Write the template for Circle here. You do NOT need to write the template rules used.

[no points for template, its purpose was to remind you it was needed for part B]

(define (fn-for-circle c)
  (... c))
;;; Functions:

;;; Circle -> Circle
;;; start the cosmic circle (main START)
(define (main c)
  (big-bang c
    (on-tick next-circle) ; Circle -> Circle
    (to-draw render-circle))) ; Circle -> Image

(B) Design the function next-circle. Provide a complete design including signature, purpose, stub, tests and complete function definition. You do not need to provide a copy of the template. (Note that you do not need to design render-circle.)

[[11 total]]
[2 - 1 for each correct type in Signature
 [3 - specific purpose,
    1 for increment
    1 for boundary
    1 for using (at least one) constant(s)
 [2 - 1 each for < STOP and = STOP test case
 [1 for stub with correct result type
 [1 if code clearly follows template in (A)
 [2 for correct code;
 [ -1 if test is wrong < >= etc. (= is ok, doesn’t need >= by data definition)
 [ -1 if answers are reversed
 [ -1 if constants are not used

;;; Circle -> Circle
;;; produce circle that is larger by SPEED, reset to START when >= STOP
(check-expect (next-circle START) (+ START SPEED))
(check-expect (next-circle STOP) START)

;(define (next-circle c) START) ;stub

(define (next-circle c)
  (if (>= c STOP)
      START
      (+ c SPEED)))