The University of British Columbia
CPSC 110: Midterm Examination
February 2nd, 2011

Name: __SOLUTIONS______ Student #: __________________ CS Dept. ID #: __________

Signature: ___________________ Lab Section: _____ Lecture 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 UBC Card 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:

<table>
<thead>
<tr>
<th>Question</th>
<th>Mark</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>88</td>
</tr>
</tbody>
</table>
Problem 1 - Mechanisms (15 points)

In this problem you will be given a number of expressions and asked to write the value of each. In some cases there will be some definitions that precede the expressions.

<table>
<thead>
<tr>
<th>What is the value of</th>
<th>What is the value of</th>
</tr>
</thead>
<tbody>
<tr>
<td>(= 3 (+ 1 2))</td>
<td>(string-length &quot;apple&quot;)</td>
</tr>
<tr>
<td>true</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>[2 points]</td>
</tr>
<tr>
<td></td>
<td>[2 points]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What is the value of</th>
<th>Given</th>
</tr>
</thead>
<tbody>
<tr>
<td>(if (&lt;= 11 10)</td>
<td>(define (wrap x)</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>(string-append &quot;W&quot; x &quot;W&quot;))</td>
</tr>
<tr>
<td>&quot;b&quot;)</td>
<td>what is the value of:</td>
</tr>
<tr>
<td></td>
<td>(wrap &quot;apple&quot;)</td>
</tr>
<tr>
<td>&quot;b&quot;</td>
<td>&quot;WappleW&quot; [4 points]</td>
</tr>
<tr>
<td></td>
<td>[2 points]</td>
</tr>
</tbody>
</table>

Given

(define-struct foo (a b))

(define (bar f)
  (string-append (foo-b f) (foo-a f)))

what is the value of:

(bar (make-foo "d" "e"))

"ed" [5 points]
Problem 2 - Types Comments (18 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.

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) the position of a butterfly that flies around the screen, moving both up and down and left and right

```
(define-struct bfly (x y))
;; Butterfly is (make-bfly Number Number)
```
4 for 2 field structure (any name is fine)
4 for proper type comment,
subtract 2 if name of structure isn't properly cased/hyphenated
subtract 2 if name of Type isn't properly cased
subtract 3 if type comment says (make-bfly x y) instead of Number Number

(B) the health of a pet rock measured as an integer from 1 to 10 inclusive

```
;; Health is Integer[1, 10]
```
4 for proper type comment (Natural is OK)
subtract 1 if not Integer
subtract 1 if not [1, 10] (proper brackets)
subtract 1 if name is not properly cased

(C) the volume levels parents hear on your earbuds, which is either loud or too loud

```
;; Volume is one of:
;; - "loud"
;; - "too loud"
```
6 points for proper 2 case one of
subtract 2 if name of type isn't properly cased
subtract 1 if cases aren't on separate lines
Problem 3 - Completing Data Definitions (17 points)

(A) Given the following (nonsensical) types comment, 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.

;;; Murf is Natural
(define M1 1)
(define (fn-for-murf m)
  (... m))
;;; Template rules used:
;;; - atomic non distinct

2 for 1 example
2 for proper template name & 1 param
2 for proper template body
2 for proper rule (allow atomic)

(B) Given the following TWO (nonsensical) types comments, complete just the Zow data definition with example data, a template and a list of the template rules used. You do not need to provide an interpretation. You should assume that the Blimpie data definition has been completed.

;;; Blimpie is one of:
;;; - "a"
;;; - "b"
;;; - Number
(define-struct zow (s b))
;;; Zow is (make-zow String Blimpie)
(define Z1 (make-zow "A" "a"))
(define Z2 (make-zow "B" "b"))
(define Z3 (make-zow "C" 3))

(define (fn-for-zow z)
  (... (zow-s z)
    (fn-for-blimpie (zow-b z)))))
;;; Template rules used:
;;; - compound: (make-zow String Blimpie)
;;; - atomic non-distinct: s field is String  <<optional
;;; - reference: b field is Blimpie

examples
1 point each (up to 3) for covering
{string}x{a, b, number}

function
1 point for proper name and 1 param
1 point each for 2 selectors
1 point call to fn-for-blimpie

rules:
1 point compound
1 points reference
parts after : optional
Problem 4 - Designing Data Definitions (20 points)

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)
the name of a city

;; 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

1 point for xxx is String types comment
1 for reasonable interp.
1 for at least 1 example
1 for proper template name and 1 param
1 for proper template body
1 for proper rule used (allow atomic)
(B)
the status of a countdown, which either has not yet started, or is an integer between 10 and 0 inclusive

;; Countdown is one of:
;;  - false
;;  - Integer[0, 10]
;; interp. false means not started, number is point in countdown
(define CD1 false)
(define CD2 4)

(define (fn-for-countdown cd)
  (cond [(false? cd) (...)]
        [else (... cd)]))

;; Template rules used:
;;  - one of: 2 cases
;;  - atomic distinct: false
;;  - atomic non-distinct: Integer[0, 10]

1 for one of
1 for reasonable false value (empty, -1, "", "not started")
1 for Integer[0, 10]
1 if interp explains first subclass
1 if interp explains second subclass
1 for example that covers first subclass
1 for example that covers second subclass
1 point for proper template fn name
1 point for cond
1 for proper false case
1 for proper else case (number? or other test OK)
1 (3) for each of 3 template rules
-1 for extra cases, clauses and/or rules
Problem 5 - Designing Functions (12 points)

As part of your summer job working for Translink you have to redesign part of the software that runs inside the fare boxes on buses. The fare box can accept either coins or a new kind of fare card that stores a balance that is reduced each time the card is used. Lucky for you, your boss has already written a data definition, you just need to design one function.

```scheme
(define-struct card (bal))
;; FarePaid is one of:
;;  - Number[>=0]
;;  - (make-card Number[>=0])
;; interp. a number means the amount put into the coin slot
;;         a card has the balance of the card put in the reader
(define F1 1.00)             ; a loonie put in the slot
(define F2 (make-card 12.00)) ; a card w/ $12 on it put in reader#

(define (fn-for-fare-paid fp)
  (cond [(number? fp) (... fp)]
       [(card? fp)
        (... (card-bal fp))]))
```

Design a function called `enough?` to compute whether the fare paid is $\geq 2.50$. The function should consume `FarePaid` and produce `Boolean`. 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 the recipe calls for. Please write down the stub and label it. You do not need to write a separate copy of the template.

```scheme
;; FarePaid -> Boolean
;; produce true if coins or card balance is $\geq 2.50
(check-expect (enough? 2.00) false)
(check-expect (enough? 2.50) true)
(check-expect (enough? 2.51) true)
(check-expect (enough? (make-card 2.00)) false)
(check-expect (enough? (make-card 2.50)) true)
(check-expect (enough? (make-card 2.51)) true)

(define (enough? fp)
  (cond [(number? fp) (>= fp 2.50)]
       [(card? fp)
        (>= (card-bal fp) 2.50)]))
```
1 each for proper type in signature
2 if purpose comprehends both coin and card
1 each for 6 tests \{(coin, card) \times \{<, =, >\}\}
2 each for adding \(\geq 2.50 \geq 2.50\) to template to get correct function.

-2 if produces "true" and "false" rather than true and false

If trying to award partial function correctness first subtract 2 points for each edit to the template. This may produce a lower score than above, in which case use the above.
Problem 6 - Completing a World Program (18 Points)

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 handling of references from one data definition to another, in both templates and final code.

The program shows an animation of a “magic ball” that floats across the screen, changing colour as it goes. As time goes by the ball’s x coordinate increases by a fixed amount. When it gets to the right edge it just keeps going and disappears off the screen. The y coordinate is constant. At each clock tick the ball’s colour changes, from red to green on one tick, then green to blue on the next tick, then blue to red and so on.

;; A Magic Ball

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

;; Constants:
(define WIDTH 400)
(define HEIGHT 200)
(define SPEED 4) ; amount ball’s x coordinate increases per tick
(define RADIUS 10) ; radius of ball
(define CTR-Y (/ HEIGHT 2))

(define MTS (empty-scene WIDTH HEIGHT))

;; Data definitions:

;; Colour is one of:
;; - "red"
;; - "green"
;; - "blue"
;; interp. the current colour of a magic ball

Write the template for Colour here. You do NOT need to write the template rules used.

#;
(define (fn-for-colour c)
  (cond [(string=? c "red") (...)]
       [(string=? c "green") (...)]
       [(string=? c "blue") (...)]))
(define-struct ball (x colour))
;; Ball is (make-ball Number Colour)
;; interp. a ball, with a current x coordinate and current colour
(define B1 (make-ball 8 "red"))
(define B2 (make-ball 12 "green"))

Write the template for Ball here. You do NOT need to write the template rules used.

#;
(define (fn-for-ball b)
  (... (ball-x b)
       (fn-for-colour (ball-colour b)))))

;; Functions:

;; Ball -> Ball
;; start the magic ball with a call like (main B1)
(define (main b)
  (big-bang b
    (on-tick ball-next) ; Ball -> Ball
    (to-draw render))) ; Ball -> Image

Design the function ball-next. If you follow the template and helper function rules you should end up with both ball-next and a helper function, you need to provide a complete design for both.

;; Ball -> Ball
;; produce a ball moved right by SPEED and with next colour
(check-expect (ball-next (make-ball 0 "red"))
              (make-ball (+ 0 SPEED) "green"))

(define (ball-next b)
  (make-ball (+ (ball-x b) SPEED)
             (colour-next (ball-colour b))))

  1 signature
  1 purpose, which must explicitly say move by SPEED and next colour
  1 at least one good test
  3 proper template is evident in function
     no helper call for x, helper call for colour
  3 correct filling of template
;;; Colour -> Colour
;;; produce the next colour
(check-expect (colour-next "red")   "green")
(check-expect (colour-next "green") "blue")
(check-expect (colour-next "blue")  "red")

(define (colour-next c)
  (cond [(string=? c "red")   "green"
        [(string=? c "green") "blue"
        [(string=? c "blue")  "red"]]))

1 signature
1 purpose
1 each for 3 required tests
2 proper template is evident in function
2 correct filling of template

award 4 points total if wrong helper function is perfectly designed