THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 110: MIDTERM 1 – October 6, 2014

05

Last Name: TAYLOR First Name: BRIAN

Signature: Brian Taylor UBC Student #: 5,2,3,1,1,8,5,9

Important notes about this examination

1. You have 120 minutes to complete this exam.
2. This exam will be graded largely 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!

Student Conduct during Examinations

1. Each examination candidate must be prepared to produce, upon the request of the invigilator or examiner, his or her UBCcard for identification.
2. No questions will be answered in this exam. If you see text you feel is ambiguous, make a reasonable assumption, write it down, and proceed to answer the question.
3. No examination 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. Should the examination run forty-five (45) minutes or less, no examination candidate shall be permitted to enter the examination room once the examination has begun.
4. Examination candidates must conduct themselves honestly and in accordance with established rules for a given examination, which will be articulated by the examiner or invigilator prior to the examination commencing. Should dishonest behaviour be observed by the examiner(s) or invigilator(s), pleas of accident or forgetfulness shall not be received.
5. Examination candidates suspected of any of the following, or any other similar practices, may be immediately dismissed from the examination by the examiner/invigilator, and may be subject to disciplinary action:
   i. speaking or communicating with other examination candidates, unless otherwise authorized;
   ii. purposely exposing written papers to the view of other examination candidates or imaging devices;
   iii. purposely viewing the written papers of other examination candidates;
   iv. using or having visible at the place of writing any books, papers or other memory aid devices other than those authorized by the examiner(s); and,
   v. using or operating electronic devices including but not limited to telephones, calculators, computers, or similar devices other than those authorized by the examiner(s)—(electronic devices other than those authorized by the examiner(s) must be completely powered down if present at the place of writing).
6. Examination candidates must not destroy or damage any examination material, must hand in all examination papers, and must not take any examination material from the examination room without permission of the examiner or invigilator.
7. Notwithstanding the above, for any mode of examination that does not fall into the traditional, paper-based method, examination candidates shall adhere to any special rules for conduct as established and articulated by the examiner.
8. Examination candidates must follow any additional examination rules or directions communicated by the examiner(s) or invigilator(s).

Please do not write in this space:

Question 1:	15	TP
Question 2:	8	
Question 3:	4	mc
Question 4:	22	
Question 5:	5	MP
Question 6:	23	JX
Question 7:	12	JP

0 236366 453321

NOTE: In this exam, if you need to use a primitive but can't remember the name, then just use some reasonable name. So if you forgot that place-image was called place-image, you could use something like put-image, add-image, image-goes-here or anything else that a reasonable grader will recognize. do-what-I-want is not going to be good enough though. The same is true for the order of arguments to complex functions like place-image or mouse-handlers.

PROBLEM 1:

Design a function that consumes two strings and produces true if the first string is longer than the second string. Follow the HtDF recipe, and include signature, purpose, stub, one or more check-expects as needed, template and complete function definition. It's ok to put your check-expects beneath your stub.

```
;; String String -> Boolean
;; produces true if first string is longer than second string

(check-expect (longer? "" "") false)
(check-expect (longer? "hello" "") true)
(check-expect (longer? "" "world") false)
(check-expect (longer? "hello" "world") false)

; (define (longer? s1 s2) false)   ; stub

#;
(define (longer? s1 s2)
        (... s1 s2))              ; template


(define (longer? s1 s2)
        (> (string-length s1) (string-length s2)))
```

awesome c-c!

3
2
2
1
2
1
1
1
1
1
1
1
15

great job i!

Problem 2 - Mechanisms

Given

```
(define x 2)

(define (example-function x)
  (if (> (* x 2) 3)
      (+ x 1)
      (- x 1)))

(example-function (* x 3))
```

Show the step-by-step evaluation of the following expression. Include the original expression and the final result. There should be 8 steps.

```
(example-function (* x 3))
```

1. (example-function (* x 3))
2. (example-function (* 2 3))
3. (example-function 6)
4. (if (> (* 6 2) 3)
       (+ 6 1)
       (- 6 1)))
5. (if (> 12 3)
       (+ 6 1)
       (- 6 1)))
6. (if true
       (+ 6 1)
       (- 6 1)))
7. (+ 6 1)
8. 7

8

PROBLEM 3 - Form of Data

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 choose the form of data that would be appropriate to represent this information.

Mark your choice by circling the most appropriate form of data.

(A) A car with its make and model. (e.g. Honda Civic, Honda Accord, Toyota Corolla)
[simple atomic] [interval] [enumeration] [itemization] [compound] [arbitrary-sized]

(B) The name of your CPSC 110 instructor (Gregor, Meghan or Ron)
[simple atomic] [interval] [enumeration] [itemization] [compound] [arbitrary-sized]

(C) Student number of every student who graduated from UBC in the last few years.
[simple atomic] [interval] [enumeration] [itemization] [compound] [arbitrary-sized]

(D) The population of Canada.
[simple atomic] [interval] [enumeration] [itemization] [compound] [arbitrary-sized]

4

;PROBLEM 4: Data Definitions

In this problem we are giving you a written description of a world program and a complete domain analysis for the world program. You must write the data definition(s) to represent the changing information. You will not continue working on this world program past the data definitions. Detailed instructions for what your solution must include are at the bottom of this comment box.

Program Description: When the mouse is clicked, a parachuter appears at that position on the screen. The parachuter falls down and gets bigger as it gets closer to the bottom of the screen. Each time you click the mouse, a new parachuter is created at that position. The parachuters keep falling off the bottom of the screen.

;--Constants--

WIDTH
HEIGHT

INITIAL-SCALE
SCALE-CHANGE-RATE

SPEED

PARACHUTER-IMAGE
MTS

;--Changing Information--

x-pos of parachuter
y-pos of parachuter
scale of parachuter
number of parachuters

;--big-bang options--

on-tick
to-draw
on-mouse

Design appropriate data definitions to represent the changing information of this program. Use one or more data definitions as you see fit. You may use constants that are included in the domain analysis.

For each data definition provide a type comment, interpretation, examples, a template function and the template rules used.

; BLANK PAGE FOR YOUR ANSWER TO PROBLEM 4

```
(define-struct (parachuter x y s))
;; Parachuter is (make-parachuter Integer[0,WIDTH] Integer[0,HEIGHT] Number[>0])
;; interp. a parachuter image centered at (x,y) with a scale factor of s

(define P1 (make-parachuter 20 30 0.5))
(define P2 (make-parachuter 100 50 0.8))

#;
(define (fn-for-parachuter p)
        (... (parachuter-x p)     ; Integer[0,WIDTH]
             (parachuter-y p)     ; Integer[0,HEIGHT]
             (parachuter-s p)))   ; Number[>0]

; template rules used
; compound: 3 fields


;; ListOfParachuter is one of:
;; - empty
;; - (cons Parachuter ListOfParachuter)
(define LOP1 empty)
(define LOP2 (cons (make-parachuter 20 30 0.5)
                   (cons (make-parachuter 100 50 0.8) empty)))

#;
(define (fn-for-lop lop)
        (cond [(empty? lop) (...)]
              [else
               (... (fn-for-parachuter (first lop))
                    (fn-for-lop (rest lop)))]))

;; Template rules used:
;; - one of 2 cases:
;; - atomic distinct: empty
;; - compound: (cons Parachuter ListOfParachuter)
;; - reference: (first lop) is Parachuter
;; - self-reference: (rest lop) is ListOfParachuter
```

cond?

2
2
1
1
1
1
1
1
0
0
1
1
3
1
2
1
0
4

READ THESE PARAGRAPHS VERY CAREFULLY BEFORE PROCEEDING ON TO THE REMAINING PROBLEMS.

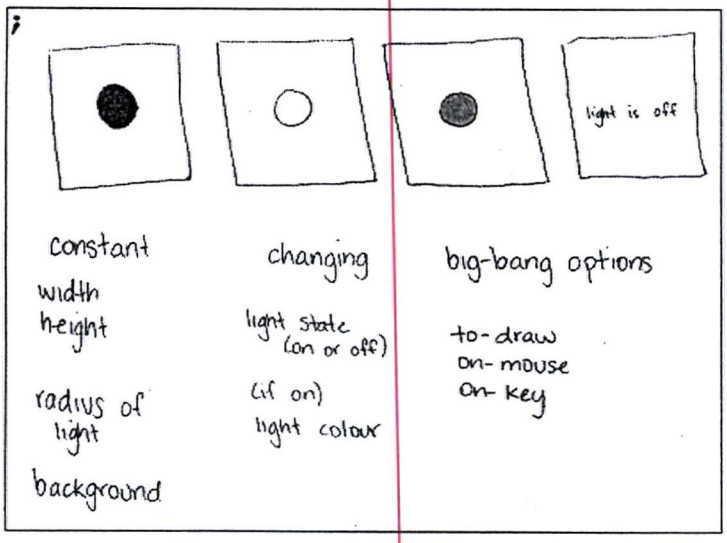
In the rest of the exam you will be working on parts of the below world program. We have provided a program description, complete domain analysis, data definitions and the functions through the wish-list entries for the world program.

Remember, you must follow the design recipes and the relevant helper rules. Read each question carefully as you must provide all the parts of the function design that we ask for. If you need a helper function, you must design it completely.

Program Description: This program displays a single traffic light. When the program starts the light is green (on). When the spacebar is pressed, the traffic light colour changes (from green to yellow, yellow to red, red to green, etc.) When the mouse is clicked the light turns off. Once the light is off, it cannot come back on again.

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

;; traffic light world program
```




```
;; ==========
;; Constants:

(define WIDTH 120)
(define HEIGHT 100)

(define LIGHT-RADIUS 20)

(define MTS (empty-scene WIDTH HEIGHT))
```

Problem 5:

On the two data definitions below you should do four things:
- on the type comments, draw any appropriate reference or self-reference arrows
- on the templates, draw any appropriate natural helper or natural recursion arrows
- label each arrow with R, SR, NH, or NR
- number each arrow with a number like 1, 2, 3 to show which arrows amongst the type comments correspond to which arrows amongst the templates.

Remember, drawing reference arrows on the data definitions help you think about the form of the functions that operate on this data.

```
;; ================
;; Data definitions:

;; LightColour is one of:
;; - "red"
;; - "yellow"
;; - "green"
;; interp. the colour of a traffic light
;; <examples are redundant for enumerations>

(define (fn-for-light-colour lc)
  (cond [(string=? "red" lc) (...)]
        [(string=? "yellow" lc) (...)]
        [(string=? "green" lc) (...)]))

;; Template rules used:
;; - one of: 3 cases
;; - atomic distinct: "red"
;; - atomic distinct: "yellow"
;; - atomic distinct: "green"



;; TrafficLight is one of:
;; - false
;; - LightColour
;; interp. a traffic light: false means the light is off, otherwise it is the given colour
(define TL1 false)
(define TL2 "red")
(define TL3 "green")

(define (fn-for-traffic-light tl)
  (cond [(false? tl) (...)]
        [else
         (... (fn-for-light-colour tl))]))

;; Template rules used:
;; - one of: 2 cases
;; - atomic distinct: false
;; - reference: LightColour
```

R1

NH1

```
;; ==================
;; Functions:

;; TrafficLight -> TrafficLight
;; start the world with (main "green")
;; no tests for main functions
(define (main tl)
  (big-bang tl                          ; TrafficLight
            (to-draw    render-tl)      ; TrafficLight -> Image
            (on-mouse   handle-mouse)   ; TrafficLight Integer Integer MouseEvent
                                        ; -> TrafficLight
            (on-key     handle-key)))   ; TrafficLight KeyEvent -> TrafficLight
```

PROBLEM 6 - Function Design

Complete the design of the render-tl function based on the wish-list entry below. You must write examples, a note stating where you took the template from and a complete function definition. You do not need to repeat the signature, purpose and stub.

```
;; TrafficLight -> Image
;; place rendering of the traffic light in the middle of MTS
;; !!!
(define (render-tl tl) empty-image)

(check-expect (render-tl false) MTS)
(check-expect (render-tl "red")
              (place-image (circle LIGHT-RADIUS "solid" "red") (/ WIDTH 2) (/ HEIGHT 2) MTS))

; < took template from TrafficLight >
(define (render-tl tl)
  (cond [(false? tl) MTS]
        [else (render-light tl)]))
```

2

2

; BLANK PAGE FOR YOUR ANSWER TO PROBLEM 6

```
;; LightColor -> Image
;; produces an image of a traffic light given a LightColor
(check-expect (render-light "red")
              (place-image (circle LIGHT-RADIUS (/ WIDTH 2) (/ HEIGHT 2) MTS)))

; (define (render-light lc) MTS) ; stub

; < took template from LightColor >

(define (render-light lc)
  (cond [(string=? "red" lc)
         (place-image (circle LIGHT-RADIUS (/ WIDTH 2) (/ HEIGHT 2) "solid" "red" MTS))]
        [(string=? "yellow" lc)
         (place-image (circle LIGHT-RADIUS (/ WIDTH 2) (/ HEIGHT 2) "solid" "yellow" MTS))]
        [(string=? "green" lc)
         (place-image (circle LIGHT-RADIUS (/ WIDTH 2) (/ HEIGHT 2) "solid" "green" MTS))]))
```

1

2

1

1

1

1

1

1

Problem 7

Complete the design of the handle-mouse function based on the wish-list entry below. You must write examples, a note stating where you took the template from and a complete function definition. You do not need to repeat the signature, purpose and stub.

```
;; TrafficLight Integer Integer MouseEvent -> TrafficLight
;; turns the light off when the mouse is clicked
;; !!!
(define (handle-mouse tl x y me) tl)    ; stub
```

```
(check-expect (handle-mouse false 20 30 "button-down") false)
(check-expect (handle-mouse false 30 40 "move") false)
(check-expect (handle-mouse "red" 40 50 "button-down") false)
(check-expect (handle-mouse "red" 50 60 "move") "red")
```

;< template from MouseEvent> ~~and large enumeration~~ >

```
(define (handle-mouse tl x y me)
  (cond [(mouse-event=? me "button-down") false]
        [else tl]))
```

1
1
2
1
1
2
1
1
1
1
1

12

```
; YOU DO NOT NEED TO COMPLETE THE HANDLE-KEY FUNCTION DESIGN. THE WISH-LIST ENTRY IS PROVIDED
 FOR COMPLETENESS OF THE PROGRAM ONLY.
```

```
;; TrafficLight KeyEvent -> TrafficLight
;; when the spacebar is pressed, if the light is on it changes colour (red > green > yellow
> red, etc.)
;; !!!
(define (handle-key tl ke) tl)
```


```
; YOU DO NOT NEED TO COMPLETE THE HANDLE-KEY FUNCTION DESIGN. THE WISH-LIST ENTRY IS PROVIDED
 FOR COMPLETENESS OF THE PROGRAM ONLY.
```