CMSC 131A, Midterm 1 (Practice)
SOLUTION

Fall 2017

Question	Points
1	10
2	10
3	10
4	20
5	10
6	20
7	20
Total:	100

This test is open-book, open-notes, but you may not use any computing device other than your brain and may not communicate with anyone. You have 50 minutes to complete the test.

The phrase "design a program" or "design a function" means follow the steps of the design recipe. Unless specifically asked for, you do not need to provide intermediate products like templates or stubs, though they may be useful to help you construct correct solutions.

You may use any of the data definitions given to you within this exam and do not need to repeat their definitions.

When writing tests, you may use a shorthand for writing check-expects by drawing an arrow between two expressions to mean you expect the first to evaluate to same result as the second. For example, you may write `(add1 3)` $\rightarrow$ `4` instead of `(check-expect (add1 3) 4)`.

Problem 1 (10 points). For each of the following programs, write out each step of computation. At each step, underline the expression being simplified. Label each step as being "arithmetic" (meaning any built-in operation), "conditional", "plug" (for plugging in an argument for a function parameter), or "constant" for replacing a constant with its value.

Problem 1(a).

```
(define R 9)
(define (w z) (* R z))
(add1 (cond [(= (w 2) 12) 4]
            [else 9]))
SOLUTION:
(add1 (cond [(= (w 2) 12) 4] [else 9]))  -->[plug]
                ^^^^^
(add1 (cond [(= (* R 2) 12) 4] [else 9])) -->[const]
                   ^
(add1 (cond [(= (* 9 2) 12) 4] [else 9])) -->[arith]
                ^^^^^^^
(add1 (cond [(= 18 12) 4] [else 9]))     -->[arith]
             ^^^^^^^^^
(add1 (cond [#false 4] [else 9]))        -->[cond]
      ^^^^^^^^^^^^^^^^^^^^^^^^^^
(add1 (cond [else 9]))                   -->[cond]
      ^^^^^^^^^^^^^^^
(add1 9)                                 -->[arith]
^^^^^^^^
10
```

Problem 1(b).

```
(posn-x (cond [(= (string-length "hi") 2) (make-posn 3 4)]
              [else (make-posn 8 0)]))
SOLUTION:
(posn-x (cond [(= (string-length "hi") 2) (make-posn 3 4)]
                  ^^^^^^^^^^^^^^^^^^^^
                                                          -->[arith]
              [else (make-posn 8 0)]))
(posn-x (cond [(= 2 2) (make-posn 3 4)]
               ^^^^^^^
                                                          -->[arith]
              [else (make-posn 8 0)]))
(posn-x (cond [#true (make-posn 3 4)]
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
              [else (make-posn 8 0)]))                    -->[cond]
              ^^^^^^^^^^^^^^^^^^^^^^
(posn-x (make-posn 3 4))
^^^^^^^^^^^^^^^^^^^^^^^^
                                                          -->[arith]
3
```

Problem 2 (10 points). The graphical text editor you developed in a recent assignment has garnered major attention by Bay-area venture capitalists who want to base a start-up company around your editor. They're ready to invest to help you develop a full-featured prototype. But before they write a check, they want to make sure you can handle fancy operations for expert text editor users. They ask if you can implement a *transpose* operation, which will transpose the character to the left of a cursor with the character to the right of the cursor. To satisfy your potential investors, you define the following function for transposing characters at a given position in a string that has at least 2 characters in it:

```
;; transpose : String Index -> String
;; Transpose characters at left and right of index i.
;; Assumes string has length >= 2 and 1<=i, i+1<=length.
(check-expect (transpose "ab" 1) "ba")
(check-expect (transpose "peice" 2) "piece")
(check-expect (transpose "student's" 8) "students'")
(define (transpose s i) ...)
```

Give a correct definition for `transpose`. (You only need to provide code, not the design steps.)

SOLUTION:

```
(define (transpose s i)
  (string-append (substring s 0 (sub1 i))
                 (substring s i (add1 i))
                 (substring s (sub1 i) i)
                 (substring s (add1 i))))
```

Problem 3 (10 points). A point in three dimensional space can be represented by three numbers specifying the points distance from the x-axis, y-axis, and z-axis. The distance of a point (x, y, z) can be computed as:

$$\sqrt{(x^2 + y^2 + z^2)}$$

For example, the distance of $(2, 6, 9)$ to the origin is 11; the distance of $(3, 2, 6)$ is 7. Assume the following data definition and design a program for computing the distance of a given 3D point.

```
;; A 3D is a (make-3d Number Number Number)
;; Interp: a point in three dimension space
(define-struct 3d (x y z))

SOLUTION:

;; dist3d : 3D -> Number
;; Compute distance to origin from given 3-d point
(check-expect (dist3d (make-3d 3 2 6)) 7)
(define (dist3d p)
  (sqrt (+ (sqr (3d-x p)) (sqr (3d-y p)) (sqr (3d-z p)))))
```

Problem 4 (20 points). Growing tired of Space Invaders, you decide to build a new video game. Part of the game consists of flying billiard balls that move in straight lines at varying velocities until hitting other balls or bouncing off walls and other obstacles. After making some sketches, you decide on the following data representation for billiard balls:

```
;; A BB is a:
;; (make-bb Color
;;          (make-posn Integer Integer)
;;          (make-vel Integer Integer))

;; A Color is one of:
;; - "red"
;; - "yellow"
;; - "green"

(define-struct bb (color center vel))
(define-struct vel (deltax deltay))
```

The interpretation of a `BB` is that the color is the color of the ball, the `posn` is the location of the center of the ball, and the `vel` structure describes the ball's velocity as a change along the x-axis (`deltax`) and y-axis (`deltay`) in one clock tick.

Design a function called `tock : BB -> BB` that calculates where a given billiard ball will be, based on its velocity, after one tick of the clock and assuming it does not encounter any obstacle.

```
SOLUTION:

;; tock : BB -> BB
;; Move ball one tick based on velocity
(check-expect (tock (make-bb "red" (make-posn 2 5) (make-vel 1 -2)))
              (make-bb "red" (make-posn 3 3) (make-vel 1 -2)))
(define (tock bb)
  (make-bb (bb-color bb)
           (make-posn (+ (posn-x (bb-center bb))
                         (vel-deltax (bb-vel bb)))
                      (+ (posn-y (bb-center bb))
                         (vel-deltay (bb-vel bb))))
           (bb-vel bb)))
```

Problem 5 (10 points). You've been hired by UPS, a package delivery service, to manage and improve their truck routing software. The software directs their truck drivers on how to proceed at each intersection (we assume streets are laid out on a simple grid for this problem): either continue straight, turn left, turn right, or take a u-turn (i.e. turn around and proceed in the opposite direction.

You decide on the following data definition for representing these instructions:

```
;; A Turn is one of:
;; - "S"
;; - "L"
;; - "R"
;; - "U"
;; Interp: directions for truck turns: straight (S), left (L),
;; right (R), or u-turn (U).
```

Problem 5(a). Write a template for `Turn` functions.

SOLUTION:

```
(define (turn-temp t)
  (cond [(string=? t "S") ...]
        [(string=? t "L") ...]
        [(string=? t "R") ...]
        [(string=? t "U") ...]))
```

Problem 5(b). Write a stub for the following function:

```
;; backtrack : Turn -> Turn
;; Compute the turn to go backward on route
(check-expect (backtrack "S") "S")
(check-expect (backtrack "L") "R")
(check-expect (backtrack "R") "L")
(check-expect (backtrack "U") "U")
```

SOLUTION:

```
(define (backtrack t) "R")
```

Problem 6 (20 points). Part of your new UPS job is to compute useful metrics about trucking routes, which will be used by the business office to optimize routes for improving gasoline consumption. A route consists of any number of segments and each segment consists of a turn (as in problem 2) and a distance to drive (measured in miles) after making that turn.

Define a data definition for routes and give two examples of values which are routes.

```
SOLUTION (there are many, this is just one):

;; A Route is one of:
;; - '()
;; - (cons Seg Route)
;; Interp: list of segments to follow in a truck route

;; A Seg is a (make-seg Turn Number)
(define-struct seg (turn dur))
;; Interp: turn to make immediately,
;; followed by distance dur going straight

'()
(cons (make-seg "L" 50)
      (cons (make-seg "R" 20)
            '()))
```

Problem 7 (20 points). Students seem obsessed with grades, which are calculated as numbers between 0 and 100, inclusive. At the end of the year, the professor needs to figure out how to convert a numeric grade into a letter grade, so they decide to write a small program using the following data definition:

```
;; A Grade is one of:
;; - [90,100]  ;; Interp: A
;; - [80,90)   ;;         B
;; - [70,80)   ;;         C
;; - [60,70)   ;;         D
;; - [0,60)    ;;         F
```

While students are obsessed with their individual grades, the university is more interested in knowing statistics about the class as a whole. They ask the professor questions like "how many 'C' students were in your class?," so they decide to write another program using the following data definition:

```
;; A ClassGrades is one of:
;; - '()
;; - (cons Grade ClassGrades)
```

But then, because the professor is lazy, they outsource the problem to you.

Design a program that calculates the number of 'C' grades in a `ClassGrades` collection.

```
SOLUTION:

;; cs : ClassGrades -> Natural
;; Count the number of Cs in the class, i.e. numbers in [70,80).
(check-expect (cs '()) 0)
(check-expect (cs (cons 99 (cons 72 (cons 74 (cons 50 '()))))) 2)
(define (cs gs)
  (cond [(empty? gs) 0]
        [(cons? gs)
         (+ (cond [(c? (first gs)) 1]
                  [else 0])
            (cs (rest gs)))]))

;; c? : Grade -> Boolean
;; is the given grade a C (i.e. in [70,80))?
(check-expect (c? 70) #true)
(check-expect (c? 80) #false)
(define (c? g)
  (and (<= 70 g) (< g 80)))
```