CMSC 198Q, Midterm 1 (PRACTICE)
SOLUTION

Summer 2019

NAME:_____

Question	Points
1	10
2	10
3	15
4	10
5	20
Total:	65

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 60 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.

Unless specifically instructed otherwise, you may use any built-in BSL functions or data types.

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 the following program, 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.

```
(define Q 1)
(define (w z) (< (string-length z) 20))
(w (cond [(= 1 Q) "fred"]
         [else 9]))
```

SOLUTION:

```
(w (cond [(= 1 Q) "fred"] [else 9])) -->[const]
               ^
(w (cond [(= 1 1) "fred"] [else 9])) -->[arith]
          ^^^^^^^
(w (cond [#true "fred"] [else 9]))   -->[cond]
   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
(w "fred")                           -->[plug]
^^^^^^^^^^
(< (string-length "fred") 20))       -->[arith]
   ^^^^^^^^^^^^^^^^^^^^^
(< 5 20)                             -->[arith]
#true
```

Problem 2 (10 points). For the following structure definition, list the names of every function it creates. For each function, classify it as being either a constructor, accessor, or predicate.

```
(define-struct dr (who strange))
```

SOLUTION:

```
- make-dr    : Constructor
- dr-who     : Accessor
- dr-strange : Accessor
- dr?        : Predicate
```

Problem 3 (15 points). You've been hired by the CMNS development office and been given their existing software for spamming potential donors. It contains the following data definition:

```
;; A Person is a (make-name Title String String)
;; Interp: a person's title and first and last names.
(define-struct name (title first last))

;; A Title is one of:
;; - "r"      Interp: Mr.
;; - "s"              Ms.
;; - "d"              Dr.
```

Finish the design of the following function for creating letter openings:

```
;; dear : Person -> String
;; Create a letter opening for the given person.
(check-expect (dear (make-name "d" "Minnie" "Maisy"))
              "Dear Dr. Maisy:")
(check-expect (dear (make-name "r" "Fred" "Rogers"))
              "Dear Mr. Rogers:")
(check-expect (dear (make-name "s" "Miriam" "Maisel"))
              "Dear Ms. Maisel:")
```

SOLUTION:

```
(define (dear p)
  (string-append "Dear "
                 (title-abbrv (person-title p))
                 " "
                 (person-last p)
                 ":"))

;; title-string : Title -> String
;; Render title as a string abbreviation
(check-expect (title-abbrv "d") "Dr.")
(check-expect (title-abbrv "r") "Mr.")
(check-expect (title-abbrv "s") "Ms.")
(define (title-abbrv t)
  (cond [(string=? t "d") "Dr."]
        [(string=? t "r") "Mr."]
        [(string=? t "s") "Ms."]))
```

Problem 4 (10 points). Do you ever mispell words like “peice” or “acheive” (which are correctly spelled “piece” and “achieve”)? Turns out a lot of people make *transposition* errors like this when typing. Based on this insight, you decide to make a next generation messaging app that helps users by giving them a *transpose* operation. The idea is that as a user types, they can place their cursor between two letters that need to be transposed and invoke `transpose`. To implement this feature, you define the following function for transposing letters at a given position in a string that has at least 2 letters 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 5 (20 points). Growing tired of Bug, 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)))
```