CMSC 131A, Midterm 1
SOLUTION

Fall 2018

NAME:_____

UID: _____

Question	Points
1	10
2	10
3	15
4	15
5	40
Total:	90

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.

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 (swap s i)
  (string-append (substring s i)
                 (substring s 0 i)))
```

Problem 4 (15 points). Design a program that takes a list of numbers and produces the sum of positive numbers in the list. For example, if the list contains 3, 4, -1, 0, -2, and 5, the sum of positive numbers (3+4+5) is 12. You may use the following data definition:

```
;; A LoN (list of numbers) is one of:
;; - '()
;; - (cons Number LoN)
;; Interp: an arbitrarily long list of numbers
```

SOLUTION:

```
;; sum-pos : LoN -> Number
;; Count the number of positive numbers in list
(check-expect (sum-pos '()) 0)
(check-expect (sum-pos (list 1 -2 8 0)) 9)
(define (sum-pos lon)
  (cond [(empty? lon) 0]
        [(cons? lon)
         (if (< (first lon) 0)
             (sum-pos (rest lon))
             (+ (first lon) (sum-pos (rest lon))))]))
```

Problem 5 (40 points). You've been hired by Nintendo to design their next blockbuster game: Dots. It consists of, well, dots: lots and lots of dots, each of which occupies a coordinate in space. You come up with the following data definitions for representing a single dot and a collection of dots:

```
;; A Dots is one of:
;; - empty
;; - (cons Dot Dots)
;; Interp: collection of an arbitrary number of dots

;; A Dot is a (make-posn Number Number)
;; Interp: a dot's position in space
```

Problem 5(a) [10 points]. Write templates for `Dot` and `Dots` functions.

SOLUTION:

```
(define (dot-template d)
  (... (posn-x d)
       (posn-y d)
       ...))

(define (dots-template ds)
  (cond [(empty? ds) ...]
        [(cons? ds) (... (dot-template (first ds))
                         (dots-template (rest ds)) ...)]))
```

Problem 5(b) [10 points]. Write a stub for the following function:

```
;; nw : Dots -> Dots
;; Move given dots in the northwest direction one unit
(check-expect (nw empty) empty)
(check-expect (nw (cons (make-posn 3 4) empty))
              (cons (make-posn 4 5) empty))
```

SOLUTION:

```
(define (nw ds) empty)
```

Problem 5(c) [20 points]. Design two functions: `dot-flip` reflects a given dot over the y-axis, for example, $(3, 4)$ becomes $(-3, 4)$, and `dots-flip` reflects a collection of dots over the y-axis.

```
;; SOLUTION:

;; dot-flip : Dot -> Dot
;; Flip dot over y-axis
(check-expect (dot-flip (make-posn 3 4)) (make-posn -3 4))
(define (dot-flip d)
  (make-posn (- (posn-x d)) (posn-y d)))

;; dots-flip : Dots -> Dots
;; Flip dots over y-axis
(check-expect (dots-flip empty) empty)
(check-expect (dots-flip (cons (make-posn 3 4) empty))
              (cons (make-posn -3 4) empty))
(define (dots-flip ds)
  (cond [(empty? ds) empty]
        [(cons? ds)
         (cons (dot-flip (first ds))
               (dots-flip (rest ds)))]))
```

[Extra space, should you need it.]