# CMSC 131A, Midterm 1
# SOLUTION

Fall 2017

NAME:________________________________________________________

UID: __________________________________________

| Question | Points |
|---|---|
| 1 | 10 |
| 2 | 10 |
| 3 | 10 |
| 4 | 15 |
| 5 | 15 |
| 6 | 10 |
| 7 | 20 |
| 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)` → 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 R 9)
(define (w z) (= R z))
(cond [(w 2) 4]
      [else 9])
```

```
SOLUTION:

(cond [(w 2) 4] [else 9])      -->[plug]
       ^^^^^
(cond [(= R 2) 4] [else 9])    -->[const]
          ^
(cond [(= 9 2) 4] [else 9])    -->[arith]
       ^^^^^^^
(cond [#false 4] [else 9])     -->[cond]
^^^^^^^^^^^^^^^^^^^^^^^^^^
(cond [else 9])                -->[cond]
^^^^^^^^^^^^^^^
9
```

**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 run (time up score))
```

```
SOLUTION:

- make-run  : Constructor
- run-time  : Accessor
- run-up    : Accessor
- run-score : Accessor
- run?      : Predicate
```

**Problem 3 (10 points).** The graphical text editor you developed in a recent assignment has attracted the interest of investors. But before they write you a check, they want to make sure you can handle fancy operations for expert text editor users. They ask if you can implement a *swap* operation, which will swap content to the left of the cursor with the content to the right.. To satisfy your potential investors, you define the following function for swapping strings at a given position:

```
;; swap : String Integer -> String
;; Swap string to the left of i with string to the right.
;; Assume (<= 0 i (string-length s)).
(check-expect (swap "" 0) "")
(check-expect (swap "ab" 1) "ba")
(check-expect (swap "bannana" 3) "nanaban")
(define (swap s i) ...)
```

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

SOLUTION:

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

**Problem 4 (15 points).** Use the following data definition for representing names:

```
;; A Name is a (make-name String String)
;; Interp: a person's full (first and last) name.
;; Both strings must contain at least one letter.
(define-struct name (first last))
```

Write a program, `name-width`, that determines the width needed to print someone's name on a letter using a mono-typed font (meaning each letter has the same width). The program can calculate this width based on the length of the person's first and last name, plus the width of a space to go in between. Assume each letter is 10 pixels wide. For example, the name Bob Harper requires 100 pixels; the name Bo Jack requires 70 pixels. You may start from the following header:

```
;; name-width : Name -> Number
;; Computes width needed to print name in mono-type font
```

SOLUTION:

```
(check-expect (name-width (make-name "Bob" "Harper")) 100)
(check-expect (name-width (make-name "Bo" "Jack")) 70)
(define (name-width n)
  (* 10 (+ 1 (string-length (name-first n))
             (string-length (name-last n)))))
```

**Problem 5 (15 points).** Design a program that takes a list of numbers and produces the count of non-negative numbers in the list. For example, if the list contains 3, 4, -1, 0, -2, and 5, the count of non-negative numbers is 4. 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:

```
;; count : LoN -> Number
;; Count the number of non-negative numbers in list
(check-expect (count-non-neg '()) 0)
(check-expect (count-non-neg (list 1 -2 8 0)) 3)
(define (count-non-neg lon)
  (cond [(empty? lon) 0]
        [(cons? lon)
         (if (< (first lon) 0)
             (count-non-neg (rest lon))
             (add1 (count-non-neg (rest lon))))]))
```

**Problem 6 (10 points).** You've been hired by the US Department of Transportation to design traffic simulation software, which includes simulating traffic lights. Discovering the simple design from 131A is insufficient, you come up with the following representation, which makes it possible to model not just the color, but also how much longer the current light has before changing:

```
;; A Light is one of:
;; - (make-red Natural)
;; - (make-yellow Natural)
;; - (make-green Natural)
;; Interp: color and remaining duration (in ticks) of a light
(define-struct red (count))
(define-struct yellow (count))
(define-struct green (count))
```

(Recall that a `Natural` is a natural number, i.e. a non-negative integer.)

**Problem 6(a).** Write a template for `Light` functions.

SOLUTION:

```
(define (light-temp l)
  (cond [(red? l) (... (red-count l) ...)]
        [(yellow? l) (... (yellow-count l) ...)]
        [(green? l) (... (green-count l) ...)]
```

**Problem 6(b).** Write a stub for the following function:

```
;; tock : Light -> Light
;; Advance the light by one tick of time
(check-expect (tock (make-red 10)) (make-red 9))
(check-expect (tock (make-red 0)) (make-green 10))
```

SOLUTION:

```
(define (tock l) (make-red 0))
```

**Problem 7 (20 points).** Over the summer, you take an internship at the University of Maryland's Center for Bioinformatics and Computational Biology (CBCB). Your first task deals with computing information about *DNA strands*. A DNA strand is an arbitrarily long sequence of *nucleotides*, which are either: C, G, A, or T. Design a program that, given a DNA strand, computes the number of times the G nucleotide occurs in it. For example, if the strand is AGCTTTGA, your program should produce 2.

```
;; SOLUTION:

;; A Nucleotide is one of:
;; - "C"
;; - "G"
;; - "A"
;; - "T"
;; Interp: DNA nucleotide C,G,A,T.

;; A DNA is one of:
;; - '()
;; - (cons Nucleotide DNA)
;; Interp: a DNA strand of nucleotides

;; count-g : DNA -> Natural
;; Count the number of occurrence of G in given DNA strand
(check-expect (count-g '()) 0)
(check-expect (count-g (cons "A" (cons "G" (cons "G" '())))) 2)
(define (count-g dna)
  (cond [(empty? dna) 0]
        [(cons? dna)
         (+ (if (string=? (first dna) "G") 1 0)
            (count-g (rest dna)))]))
```

[Extra space, should you need it.]