Fundamentals of Computer Science 1 **Exam 1**
CS2500 Exam 1--Practice

This exam is designed so that it can be taken in 1 hour; to accommodate everyone's needs, the instructors will stay for 2 hours.

You may use the cards in the design recipe, along with both sides of **three** *sheet of 8.5x11 paper as a "cheat sheet." Otherwise, this exam is closed-book, closed-notes, closed-Internet, ...*

You may only use definitions, expressions, and functions found in BSL on this exam. Define everything else.

The phrase "design a data definition" or "design a function" means that you should apply the design recipe. Show all steps!

You may use a shorthand notation to write any examples or test cases: for example, you could write (string? "foo") → #true *to indicate* (check-expect (string? "foo") #true).

Some basic test-taking advice: before you start answering any problems, read every *problem so your brain can be thinking about the harder problems in the background while you knock off the easy ones.*

Name: ________________________________ NUID: ____________________

Instructor: __________________________ Lecture time: ____________________

Problem	Possible	Score
1	6	
2	10	
3	10	
4	16	
Total	42	

1. Consider the following code in BSL:

```
(define-struct thing [foo bar])

(define ABC "123")
(define DEF (string-append ABC " test"))
(define GHI (make-thing 987 DEF))

(define (get-result val)
  (cond
    [(string=? ABC val) 42]
    [(<= (string-length val) 100) (thing-foo GHI)]
    [else (string-length (string-append val val))]))

(get-result DEF)
```

If you ran it in DrRacket, what would the output be? **You must explain your work**, for example, by showing intermediate results. As a reminder of how the $<$ and $>$ functions work in BSL, (< 2 3) is #true since 2 is less than 3. (6 pts)

2. Consider the following data definition and interpretation:

```
(define-struct espresso [shots decaf])
(define-struct dripcoffee [size decaf milk])

; A CoffeeOrder is one of
; - (make-espresso PositiveInteger Boolean)
; - (make-dripcoffee PositiveInteger Boolean Boolean)
;
; Interpretation: An order at a coffee shop
; - make-espresso
;   - shots is the number of espresso shots
;   - decaf is #true if the customer wants decaf, #false otherwise
; - make-dripcoffee
;   - size is the size of the cup in ounces
;   - decaf is #true if the customer wants decaf, #false otherwise
;   - milk is #true if the customer wants milk added, #false otherwise
```

2a. Give a representative set of data examples of a CoffeeOrder. Make sure you cover all cases of the data definition. (4 pts)

2b. Create a template for CoffeeOrder. (6 pts)

3. Consider the following data definition, interpretation, and examples:

```
(define-struct flight [id seats tickets])

; A Flight is a (make-flight String NonNegativeInteger NonNegativeInteger)
;
; Interpretation: A record of an airline flight
; - id is the airline and flight number
; - seats is the number of passenger seats on the aircraft
; - tickets is the number of tickets that have been sold

(define FLIGHT-1 (make-flight "Delta 80" 139 120))
(define FLIGHT-2 (make-flight "United 777" 119 122))
```

Design the function overbooked? that accepts a Flight and returns whether or not that flight is overbooked (i.e., has more tickets sold than there are seats available on the aircraft). Be sure to show all steps of the design recipe. You do not need to make a template for Flight, but are encouraged to do so. (10 pts)

4. Consider the following data definition(s):

```
(define-struct traincar [seats wheels])

; A TrainCar is a (make-traincar PositiveInteger PositiveInteger)
;
; Interpretation: A train car in a train
; - seats is the number of passenger seats in the car
; - wheels is the number of wheels on this car

(define-struct train [currentcar remainder])

; A Train is one of:
; - "caboose"
; - (make-train TrainCar Train)
;
; Interpretation: A train of cars, ending in a caboose
; - make-train
;   - currentcar is the current car in the train
;   - remainder is the remainder of the train
; - "caboose" is the final car in the train
```

4a. Provide example(s) and template(s) for *all* data definitions above. (6 pts)

4b. Let us say that a TrainCar is *good* if it has at least 30 seats AND it has at least 4 wheels. Design the function num-good-cars that accepts a Train and returns the total number of cars that are good. Be sure to show all steps of the design recipe. You may use your data examples from problem 4a here if you wish. (10 pts)

This page is left blank for work.