CS2135, C02

Midterm Exam

Name:

Problem	Points	Score
1	30	
2	30	
3	40	
	Total	

You have 50 minutes to complete the problems on the following pages. There should be sufficient space provided for your answers. You do not need to show templates, but you may receive partial credit if you do. You also do not need to show test cases or examples of data models, but you may develop them if they will help you write the programs.

Your programs may contain only the following Scheme syntax:

define define-struct cond else lambda

and the following primitive operations:

empty? cons? cons first rest list map filter
number? $+ - * / = < > <= >=$ *zero?*
symbol? symbol=? equal?
boolean? **and or** *not*

and the functions introduced by **define-struct**.

You may, of course, use whatever constants are necessary.

1. (30 points) For each of the following programs, either

   - Rewrite it with *map* and/or *filter* if appropriate (ie, if the body of the rewritten function could be a call to *map* or *filter*), or
   - Explain why *map* and *filter* aren't appropriate (based on their purposes, contracts, internal structure, etc).

   (a) (10 points)

```
;; insert : number list[number] → list[number]
;; inserts number into position in list of sorted (increasing) numbers
(define (insert anum alon)
  (cond [(empty? alon) (cons anum empty)]
        [(cons? alon)
         (cond [(> anum (first alon))
                (insert anum (rest alon))]
               [else (cons anum alon)])]))
```

(b) (10 points)

```
;; flip : list[symbol] → list[symbol]
;; changes all 'left in list to 'right and vice-versa, leaving other symbols intact
(define (flip alos)
  (cond [(empty? alos) empty]
        [(cons? alos)
         (cons (cond [(symbol=? (first alos) 'right) 'left]
                     [(symbol=? (first alos) 'left) 'right]
                     [else (first alos)])
               (flip (rest alos)))]))
```

Example: (*flip* (*list* 'left 'straight 'right)) produces (*list* 'right 'straight 'left)

(c) (10 points)

```
;; product : list[number] → number
;; computes the product of a list of numbers
(define (product alon)
  (cond [(empty? alon) 1]
        [(cons? alon)
         (* (first alon) (product (rest alon)))]))
```

2. (30 points) WPI has decided to automate housing selection. Rather than having students gather in person to select rooms, students will submit a function that takes the number of rooms available in each dorm and returns the name of the dorm they want a room in. For this problem, we'll assume all rooms are doubles in Morgan and Daniels.

   A housing request consists of the names of two students (to share a double) and a function that takes two numbers (the number of rooms available in Morgan and Daniels) and returns either 'morgan or 'daniels. The following data definition captures a housing request:

   A room-request is a
   (*make-request symbol symbol* (*number number -> symbol*))
   (**define-struct** *request* (*student1 student2 choose*))

   A dorm assignment lists two students and the dorm in which they will live. Here is the data definition for a room assignment:

   A dorm-asgmt is a
   (*make-asgmt symbol symbol symbol*)
   (**define-struct** *asgmt* (*student1 student2 dorm*))

   (a) (10 points) Beavis and Butthead would choose a room in Daniels if there were at least two rooms open there (so their friends, drawing right after them, could also live there); otherwise, they'll take Morgan. Write the *make-request* example that captures this request.

(b) (20 points) Write a program *draw-rooms* that consumes two numbers (the available rooms in Morgan and Daniels, respectively) and a list of requests and returns a list of dorm assignments. The program should process the requests in order and return the list of resulting dorm assignments. Processing a request should decrease the number of available rooms in the appropriate dorm. Assume there are enough rooms that every request will yield either 'morgan or 'daniels.

For example, if *BB-request* refers to your answer from part (a) (and assuming the same request were processed twice):

```
  (draw-rooms 9 1 (list BB-request))
= (list (make-asgmt 'Beavis 'Butthead 'morgan)

  (draw-rooms 9 2 (list BB-request BB-request))
= (list (make-asgmt 'Beavis 'Butthead 'daniels)
        (make-asgmt 'Beavis 'Butthead 'morgan))
```

3. (40 points) Consider the following definition of ancestor trees:

   An *ftree* is one of
   - 'unknown,
   - (*make-person symbol ftree free*)

   (**define-struct** *person* (*name father mother*))

   A medical researcher wants to augment and use this model to study disease patterns in families.

   (a) (5 points) Edit the data definition to store two additional pieces of information per person: blood type (one of A, B, AB, or O) and one major illness (such as diabetes, cancer, etc). Make sure you also add or edit the **define-struct**s as necessary (you may edit the existing data definition text; you do not need to recopy it).

   (b) (15 points) Our researcher has a theory that all people with diabetes have blood type A (in logic terms, diabetes $\rightarrow$ blood-type is A). In other words, every person either has both diabetes and blood type A, or they don't have diabetes. Write a program *diabetes-A?* that consumes a family tree and returns a boolean indicating whether all people in the family tree satisfy the theory. (*diabetes-A?* 'unknown) should return true.

(c) (15 points) Our researcher wants to write programs to test several theories (like *diabetes-A?*) on family trees. All of these programs check a predicate against each person in the tree and return true if everyone satisfies the predicate (in each program, 'unknown returns true). Write a program *test-theory* that consumes a function from ftree (person or 'unknown) to boolean and a ftree and returns true if the function holds of every person in the tree.

;; test-theory : (ftree → bool) ftree → boolean
;; determine whether each person in tree satisfies given predicate

(d) (5 points) Rewrite your *diabetes-A?* function using *test-theory*.