CS2135, C04

Midterm Exam

Name:

Problem	Points	Score
1	20	
2	24	
3	26	
4	30	
	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. You do not need to use map/filter unless a problem states otherwise (you may use them if you wish).

Your programs may contain only the following Scheme syntax:

define define-struct cond else lambda local let

and the following primitive operations:

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

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

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

---

1. (20 points) A voting system tracks information about candidates for elected office, such as their name, party affiliation, favorite issue, and how many votes they have earned so far.

```
;; A candidate is a (make-candidate symbol symbol symbol number)
(define-struct candidate (name party issue votes))

;; Example: (make-candidate 'Pat 'national 'health-care 10)
```

   (a) (10 points) Write a program *find-candidate* that takes a name (symbol) and a list of candidates and returns the candidate with the given name.

```
;; find-candidate : symbol list[candidate] → candidate
;; find the candidate info for the candidate with the given name
```

   (b) (10 points) Write a program *incr-votes* that takes a candidate name and a list of candidates and returns a new list of candidates with the named candidate having one additional vote.

```
;; incr-votes : symbol list[candidate] → list[candidate]
;; returns list of candidates with one vote added to named candidate
```

(exam continues next page)

2. (24 points) For each of the following programs, either

   - Rewrite it with *map* and/or *filter* if the body of the 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) (8 points)

```
;; parity : list[number] → list[symbol]
;; return list indicating whether each input was odd or even (odd? and even? are built in)
(define (parity alon)
  (cond [(empty? alon) empty]
        [(cons? alon) (cond [(odd? (first alon)) (cons 'odd (parity (rest alon)))]
                            [else (cons 'even (parity (rest alon)))])]))
```

(b) (8 points)

```
;; snoc : alpha list[alpha] → list[alpha]
;; put element on end of list (a "backwards cons")
(define (snoc elt lst)
  (cond [(empty? lst) (cons elt empty)]
        [(cons? lst) (cons (first lst) (snoc elt (rest lst)))]))
```

(c) (8 points)

```
;; reflect : list[posn] → list[posn]
;; flips coordinates of non-diagonal positions and eliminates diagonal positions
(define (reflect alop)
  (cond [(empty? alop) empty]
        [(cons? alop) (cond [(= (posn-x (first alop)) (posn-y (first alop)))
                            (reflect (rest alop))]
                           [else (cons (make-posn (posn-y (first alop)) (posn-x (first alop)))
                                       (reflect (rest alop)))])]))
```

(exam continues next page)

3. (26 points) Absentee ballots for elections are submitted days or weeks in advance, preventing absentee voters from using last minute information when casting their votes. Thus, rather than have an absentee ballot select a fixed candidate, a more flexible ballot system would allow the person to submit a function that determines the name of who to vote for based on up-to-date information on the candidates (as people have when voting on election day). The information on candidates is identical to that from problem 1.

```
;; A candidate is a (make-candidate symbol symbol symbol number)
(define-struct candidate (name party issue votes))

;; A ballot is a (make-ballot number (list[candidate] → symbol))
(define-struct ballot (district cast-vote))
```

For the following questions, make up missing data on districts/candidates as needed. You may use the functions from question 1 as helper functions without copying them. Define all other helper functions that you use as part of your answer.

(a) (4 points) Define a ballot that votes for FrodoBaggins regardless of the current candidate info.

(b) (10 points) Write a program *make-underdog-ballot* that takes two names and returns a ballot that votes for the candidate (out of the two named) who has fewer votes.

```
;; make-underdog-ballot : symbol symbol → ballot
;; create ballot to vote for named candidate with fewer votes
```

(exam continues next page)

(c) (12 points) Write a program *tally-votes* that consumes a name of a candidate (symbol), a list of candidates, and a list of ballots and returns the number of votes cast for that candidate when the absentee ballots are cast in the order they appear in the list. (Each vote should affect the candidate information before the next ballot is checked.)

;; tally-votes : symbol list[candidate] list[ballot] → number
;; counts votes given to named candidate from list of ballots.

(exam continues next page)

4. (30 points) Chain letters tell people to send a copy of the letter to several friends (who in turn also send the letter to several friends, thus circulating the letter to many people in a short time span). We can trace how a chain letter circulates through the following data definition, where an empty sent-to list implies that the sender hasn't yet forwarded the message to anyone:

```
;; A list[sender] is either
;; - empty, or
;; - (cons sender list[sender])

;; A sender is a (make-sender string list[sender])
(define-struct sender (email sent-to))

;; Example of decision tree to recommend activities to people
(define chain (make-sender "k@host.com"
                           (list (make-sender "jj@ww.net" empty)
                                 (make-sender "bozo@circus.org"
                                              (list (make-sender "goofy@dn.com" empty))))))
```

(a) (15 points) Write a program *all-recipient-emails* that takes a sender and returns a list of all the email addresses that the letter has been sent to (do not worry about duplicates).

```
;; all-recipient-emails : sender → list[symbol]
;; return the list of all email addresses in the sender tree
```

(exam continues next page)

(b) (15 points) Chain letters usually tell the recipient how many friends to forward the letter to. Write a program *always-sent-to-6?* that takes a sender and returns a boolean indicating whether every sender in the tree who has already forwarded the letter has sent it to exactly 6 people.

;; always-sent-to-6? : sender → boolean
;; determine whether every sender who has sent the message out has sent it to exactly 6 recipients