CS2135, C04

Final Exam

Name:

Problem	Points	Score
1	35	
2	30	
3 or 4	35	
	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 let local define-syntax define-script begin

and the following primitive operations:

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

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

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

Unless a problem states otherwise, you are not required to use map and filter in your answers.

Everyone should do questions 1 and 2, plus **ONE** of questions 3 or 4. Question 3 is the review question and Question 4 covers web programming. If you attempt both, circle the problem number for the one we should grade.

1. (35 points) A manufacturing company is developing new robot control software. Each robot has a movable arm with a hand on the end and two (radar-based) sensors: one at the front that indicates distance to objects ahead, and one on the hand that indicates distance to the floor. Robots are able to move forward and backward, turn left and right (90 degrees), raise and lower their arm, and grab or release the hand at the end of the arm. The following sequence is an example set of instructions for a robot:

   - Move forward until 2 feet from an object
   - Lower arm until it is 3 feet from the ground
   - Grab object (assume hand is in the right position for this)
   - Raise arm 1 foot
   - Move backwards 2 feet
   - Turn left
   - Move forward 6 feet
   - Lower arm 1 foot
   - Release object

   (a) (10 points) Explain briefly (a few words will suffice) how you would go about identifying both the commands and the data in this language.

   (b) (15 points) Provide data definitions for robot control programs. You should have one definition for the data, one for the commands, and one for programs in this language. Use the variables *front-sensor* and *floor-sensor* to refer to the distances indicated by each sensor.

(exam continues next page)

(c) (10 points) Write the example set of instructions in your language.

(exam continues next page)

2. (30 points) The registrar's office needs new software for specifying and querying course schedules. They choose the following notation for course schedules (for each course, they list the course number, hour it meets in 24-hour format, and instructor's name).

```
(define lookup
  (schedule-query (CS (2135 1400 Fisler)
                      (2011 1500 Hamel)
                      (1005 0100 Elliott))
                  (EE (2201 0800 Papageorgiou)
                      (2801 0900 Lou))))
```

   (a) (25 points) Write a *schedule-query* macro that produces a function that takes a dept name, course number and one of the symbols 'time or 'prof and returns either the time that the indicated course meets or the name of the instructor teaching the course (depending on the given symbol). For example, the call (*lookup* 'CS 2135 'time) should return 1400. You may assume that all requested courses are in the schedule.

   (b) (5 points) If we wrote *schedule-query* as a function instead of a macro and used it to define *lookup* as shown above, what would happen when we hit Execute (and why)?

(exam continues next page)

3. (**Do this or question 4** – 35 points) A company maintains information on its employees, how their wages are computed, and who they supervise using the following data structure:

   ;; An emp is a (make-emp symbol (number $\rightarrow$ number) list[emp])
   (**define-struct** *emp* (*name calc-wage supervises*))

   The *calc-wage* function component is meant to consume the number of hours the person has worked and return the wage they should be paid based on the given number of hours.

   (a) (10 points) Assume the company consists of the following three people:

   - Billy : paid \$10 an hour, supervises nobody
   - Pat : paid \$12 an hour up to 30 hours (no pay for extra hours), supervises nobody
   - Jackie : paid \$30 an hour, supervises Billy and Pat

   Write the example for this collection of employees using *make-emp*.

(exam continues next page)

(b) (12 points) Write a program *compute-wage* that takes the name of an employee, the number of hours they worked, and an emp structure (for the boss of the company), and returns the wage that the employee should be paid, or *false* if the employee is not in the tree. **Assume you have a helper function** *find-emp* **that takes a name and an emp (the boss) and returns either an emp or false** (where the returned emp is the emp struct with the given name; false means that there is no employee with that name in the tree).

;; compute-wage : symbol number emp $\rightarrow$ number or false
;; computes wage due to named employee for given number of hours

(c) (13 points) Write the *find-emp* helper function as used/described in the previous part.

;; find-emp : symbol emp $\rightarrow$ emp or false
;; returns emp struct for named employee or false if no such name in tree

(review exam ends here)

4. (**Do this or question 3** – 35 points) A computer sales company uses a program to guide customers through choosing and configuring the machine they want to order. They capture machine models and their options with the following structures:

```
;; (make-model symbol list[trait])
(define-struct model (name traits))
;; (make-trait symbol list[symbol])
(define-struct trait (descr options))
(define available-machines
  (list (make-model 'dell (list (make-trait 'memory (list '256M '512M))
                                (make-trait 'harddisk (list '20GB '40GB '80GB))
                                (make-trait 'battery (list 'light 'heavy))
                                (make-trait 'screen (list '10.1in '12in))))
        (make-model 'old-n-cheap (list (make-trait 'memory (list '800K '128M))
                                       (make-trait 'harddisk (list '20MB '4GB))
                                       (make-trait 'drive (list 'floppy 'tape))))))
```

Here is the current (non-web) version of their customer software:

```
(define (prompt-read promptstr)
  (begin (printf promptstr)
         (read)))

(define (purchase-machine models)
  (let ([choice (prompt-read (format "Choose a model ~a: " (map model-name models)))])
    (let ([matches (filter (lambda (mod) (symbol=? (model-name mod) choice)) models)])
      (cond [(empty? matches) (begin (printf "Invalid model chosen~n")
                                     (purchase-machine models))]
            [else (configure (first matches))]))))

(define (configure model)
  (printf "You ordered model ~a with ~a~n" (model-name model) (select (model-traits model))))

(define (select traits)
  (cond [(empty? traits) empty]
        [(cons? traits) (let ([promptstr (format "Choose a ~a ~a: "
                                                 (trait-descr (first traits))
                                                 (trait-options (first traits)))])
                          (let ([wants (prompt-read promptstr)])
                            (cond [(member wants (trait-options (first traits)))
                                   (cons (list (trait-descr (first traits)) wants)
                                         (select (rest traits)))]
                                  [else (select traits)])))]))
```

(a) (8 points) On the above code, circle all calls to the (four) defined functions that are **NOT** in script position (you don't have to write anything else for this part).

(exam continues next page)

(b) (15 points) Here is another copy of the **SAME** program. Edit this code so that all four defined functions work as scripts (with all calls to them moved into script position). You do not need to move calls to any other functions (map, filter, etc). If the edits are simple, mark them on the original program (without recopying code). If you are copying a whole expression without changes between the open and close paren, you may copy just enough to show which expression you are copying.

```
(define (prompt-read promptstr)
  (begin (printf promptstr)
         (read)))

(define (purchase-machine models)
  (let ([choice (prompt-read (format "Choose a model ~a: " (map model-name models)))])
    (let ([matches (filter (lambda (mod) (symbol=? (model-name mod) choice)) models)])
      (cond [(empty? matches) (begin (printf "Invalid model chosen~n")
                                     (purchase-machine models))]
            [else (configure (first matches))]))))

(define (configure model)
  (printf "You ordered model ~a with ~a~n" (model-name model) (select (model-traits model))))

(define (select traits)
  (cond [(empty? traits) empty]
        [(cons? traits) (let ([promptstr (format "Choose a ~a ~a: "
                                                 (trait-descr (first traits))
                                                 (trait-options (first traits)))])
                          (let ([wants (prompt-read promptstr)])
                            (cond [(member wants (trait-options (first traits)))
                                   (cons (list (trait-descr (first traits)) wants)
                                         (select (rest traits)))]
                                  [else (select traits)])))]))
```

(exam continues next page)

(c) (12 points) Going back to the original program, assume we defined the *select* function using **define-script** instead of **define** (but didn't move any calls to script position). [Recall that **define-script** makes the function return its answer but terminate all pending computation]. Assume a customer wanted to order a dell machine with 256M memory, a 40GB disk, light battery and 12in screen. What would the customer's interaction with the software look like starting from the call (*purchase-machine available-machines*)? (ie, show statements printed by program with responses entered by customer)

(exam ends here)