CS1102, A07

Final Exam

Name:

Problem	Points	Score
1	35	
2	30	
3	35	
	Total	

You have 50 minutes to complete the problems on the following pages. There should be sufficient space provided for your answers. Test cases and function contracts are not required, but comments describing the types of data are required for data definitions.

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 andmap ormap member length
number? $+ - * / = < > <= >=$ *zero?*
symbol? symbol=? string? string=? equal? eq?
boolean? **and or** *not*
printf format

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

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

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 open or close the hand at the end of the arm (to grab or release objects). 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
   - Close hand
   - Raise arm 1 foot
   - Move backwards 2 feet
   - Turn left
   - Move forward 6 feet
   - Lower arm 1 foot
   - Open hand

   (a) (23 points) A robot control program is a list of commands. The following code starts to define the commands for robot programs (by defining the *open-hand* command). Add enough commands to capture the sample program. Include all needed define-structs (you can simply indicate the field types within the cases of the command data definition). Use the variables *front-sensor* and *floor-sensor* to refer to the distance from the robot to each sensor. **Write only data definitions and define-structs for this part.**

   ;; A command is one of
   ;; - (*make-open-hand*)

   (**define-struct** *open-hand* ())

(exam continues next page)

(b) (12 points) Write the list of commands corresponding to the example program in your language. **Just write the example program; do not write an interpreter or other code.**

(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 (1102 1300 Fisler)
                      (2022 0900 Sarkozy)
                      (3113 1000 Wills))
                  (EE (2011 0900 McNeill)
                      (2801 1600 Jarvis))))
```

   (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 1102 'time) should return 1300. 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, defined *lookup* as shown above, then ran (*lookup* 'CS 1102 'time), what would Scheme produce and why? Describe any outputs or errors (instead of just saying "it runs fine" or "we get an error").

(exam continues next page)

3. (35 points) A computer sales company uses a program to guide customers through choosing and configuring the machine they want to order. Here is the current (non-web) version of their customer software:

```
(define MODELS (list 'dell 'hp))
(define ALL-OPTIONS (list 'extra-memory 'bluetooth 'monitor))

(define (web-read promptstr)
  (begin (printf promptstr)
         (read)))

(define (purchase-laptop)
  (let ([choice (web-read (format "Choose a model ~a: " MODELS))])
    (cond [(member choice MODELS) (configure choice)]
          [else (begin (printf "Invalid model chosen~n")
                       (purchase-laptop))])))

(define (configure modelname)
  (printf "You ordered one ~a laptop with ~a~n" modelname (select ALL-OPTIONS)))

(define (select options)
  (cond [(empty? options) empty]
        [(cons? options)
         (let ([wants (web-read (format "Do you want ~a?: " (first options)))])
           (cond [(eq? wants 'yes)
                  (cons (first options) (select (rest options)))]
                 [else (select (rest options))]))]))
```

(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) (12 points) 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 terminates all pending computation]. Assume a customer wanted to order a dell laptop with bluetooth and a monitor. What would the customer's interaction with the software look like starting from the call (*purchase-machine*)? (ie, show statements printed by program with responses entered by customer)

(exam continues next page)

(c) (15 points) Here is another copy of the **SAME** code, but using **define-script** instead of **define**. Move all calls to the four scripts into script position. Do not move calls to any other functions. Do this manually, **NOT** using *let/cc*. 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 MODELS (list 'dell 'hp))
(define ALL-OPTIONS (list 'extra-memory 'bluetooth 'monitor))

(define-script (web-read promptstr)
  (begin (printf promptstr)
         (read)))

(define-script (purchase-laptop)
  (let ([choice (web-read (format "Choose a model ~a: " MODELS))])
    (cond [(member choice MODELS) (configure choice)]
          [else (begin (printf "Invalid model chosen~n")
                       (purchase-laptop))])))

(define-script (configure modelname)
  (printf "You ordered one ~a laptop with ~a~n" modelname (select ALL-OPTIONS)))

(define-script (select options)
  (cond [(empty? options) empty]
        [(cons? options)
         (let ([wants (web-read (format "Do you want ~a?: " (first options)))])
           (cond [(eq? wants 'yes)
                  (cons (first options) (select (rest options)))]
                 [else (select (rest options))]))]))
```

(exam ends here)