CS 1102, A05

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. 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? string? string=?
boolean? **and or** *not*
printf

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

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

You are not required to use map and filter in your answers.

---

1. (Language Design – 35 points) You want to develop a language to automate certain tasks in an instant messaging system. A messaging user can write programs that declare other users as their buddies and automatically send messages to buddies depending on criteria such as the time of day or whether a buddy is logged in or has unread messages. Each automatic message can be sent daily or only on certain days of the week.

   Here's a sample instant messaging program. Buddies have a nickname and a username. Times are given in 24-hour format.

```
buddy Maria mgalvez
buddy Jack boxedin
buddy Pat pat46

at 21:00 {friday, saturday} send Jack "off to party" ;
at 14:30 {tuesday, thursday} if Pat logged-in send "play ball?" ;
for-all-buddies logged-in at 11:45 daily send "lunch anyone?" ;
at 21:00 saturday if Maria has-unread-msgs send "U R 2 quiet" ;
```

   (a) (25 points) The following data definition captures one command needed to define the messaging language. Extend the data definition to create a messaging language that can capture the program shown above. **Include data definitions for programs and data**, as well as any other data definitions that you need.

;; A cmd is one of
;; - (*make-send string string*)

(**define-struct** *send* (*username message*))

(exam continues next page)

(b) (10 points) Write the example program from the previous page in your data definition. **Do not write an interpreter, just write the example program.**

(exam continues next page)

2. (Macros – 30 points) You are creating a system for organizing web scripts into web pages. The following program builds a homework submission system like turnin. The programmer declares which scripts the system uses, specifying a name, description, and file containing the code for each script. Each script may also be restricted to certain types of users. The programmer then specifies how the scripts are organized into pages (in the layout section). Each page enables (provides access to) certain scripts and contains links to other pages.

```
(webapp (declare-scripts (password "Change Password" "password.ss" ())
                         (create "New Assignment" "create-assign.ss" (faculty))
                         (submit "Submit" "submit-file.ss" (student))
                         (view "View Submitted" "view-file.ss" (student faculty)))
        (layout (page "index.html" "Turnin Home"
                      (enable password)
                      (link "assignments.html"))
                (page "assignments.html" "Assignments"
                      (enable create view submit)
                      (link "index.html"))))
```

Write a macro to convert the above syntax into the following code:

```
(let [(scripts (list (make-script 'password "Change Password" "password.ss" (list))
                     (make-script 'create "New Assignment" "create-assign.ss" (list 'faculty))
                     (make-script 'submit "Submit" "submit-file.ss" (list 'student))
                     (make-script 'view "View Submitted" "view-file.ss" (list 'student 'faculty))))]
  (begin (write-page-to-file ((lambda (script-data)
                                (make-htmlpage "index.html" <--- etc --->))
                              scripts))
         (write-page-to-file ((lambda (script-data)
                                (make-htmlpage "assignments.html" <--- etc --->))
                              scripts))))
```

Assume you have already written a macro for *page* that generates the **lambda** expressions shown. Assume that function *write-page-to-file* is defined. The beginning of the page macro appears on the next page for reference (**do not edit the page macro – just use it to write the webapp macro**).

(exam continues next page)

(The page macro for reference – do not edit)

```
(define-syntax page
  (syntax-rules ()
    [(page filename title (enable action ...) (link apage ...))
     (lambda (script-data)
       (make-htmlpage title <--- etc --->))]))
```

(rest of this page is for work or scratch space)

(exam continues next page)

3. (Script Position – 35 points) Consider the following programs for a simple discussion board.

```
(define-struct msg (author subject text))
(define board empty)

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

(define (post-msg username)
  (set! board (cons (make-msg username
                              (prompt-read "message subject: ")
                              (prompt-read "message body: "))
                    board)))

(define (show-messages alom)
  (cond [(empty? alom) (begin (printf "No more unread messages~n")
                              empty)]
        [(cons? alom)
         (begin (display-message (first alom))
                (cond [(symbol=? 'yes (prompt-read "Read another? "))
                       (cons (first alom) (show-messages (rest alom)))]
                      [else alom]))]))

(define (choose-action username)
  (let ([choice (prompt-read "Do you want to post or read messages? ")])
    (begin
      (cond [(symbol=? choice 'post) (post-msg username)]
            [(symbol=? choice 'read) (set! board (show-messages board))]
            [else (printf "Unknown action.  Try again~n")])
      (choose-action username))))

(choose-action "sandy")
```

(a) (10 points) On the above code, circle all calls to the defined functions (*choose-action*, *show-messages*, *post-msg* and *prompt-read*) that are **NOT** in script position (don't write anything else for this part).

(b) (25 points) Another copy of the code appears on the next page. Edit it so that all four defined functions work as scripts (i.e., with all calls to them moved into script position – do this manually rather than through macros). Do not move calls to any other functions. 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 parens or quotation marks, copy just enough to show what you are copying. Use the back of the page or extra paper if you need more space.

(exam continues next page)

```
(define-struct msg (author subject text))
(define board empty)

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

(define (post-msg username)
  (set! board (cons (make-msg username
                              (prompt-read "message subject: ")
                              (prompt-read "message body: "))
                    board)))

(define (show-messages alom)
  (cond [(empty? alom) (begin (printf "No more unread messages~n")
                              empty)]
        [(cons? alom)
         (begin (display-message (first alom))
                (cond [(symbol=? 'yes (prompt-read "Read another? "))
                       (cons (first alom) (show-messages (rest alom)))]
                      [else alom]))]))

(define (choose-action username)
  (let ([choice (prompt-read "Do you want to post or read messages? ")])
    (begin
      (cond [(symbol=? choice 'post) (post-msg username)]
            [(symbol=? choice 'read) (set! board (show-messages board))]
            [else (printf "Unknown action.  Try again~n")])
      (choose-action username))))

(choose-action "sandy")
```

(exam ends here)