CS 1102, A04

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) Imagine that you are developing a language to capture the moves of one player in a simplified soccer game. The player can kick the ball hard (trying to score), kick it gently (to move it forward), pass the ball in some direction, or run in some direction (where a direction is forward, backward, left, or right).

   When the player's team has the ball, the player's program (the offense program) might informally look like

```
begin
  repeat-until-play-stops
    if have ball
      if near goal then kick hard     ;; to try to score
      else if near opponent then pass in direction of a teammate
      else kick easy                  ;; to move ball forward
    else run forward
  start defense program
```

   (a) (25 points) The following data definitions capture part of the language for controlling players. Extend the definitions and add define-structs as necessary so that you could write the offense program given above (in other words, add cases to the object, boolean-cmd, and cmd data definitions as needed to finish the language definition). You may add other data definitions if necessary.

   ;; A boolean-cmd is one of
   ;; - (*make-check-near game-element*)

   (**define-struct** *check-near near-what*)

   ;; A game-element is

   ;; A cmd is one of
   ;; - (*make-get-direction-to symbol*)
   ;; - (*make-start-program program*)

   (**define-struct** *get-direction-to* (*which-team*)) ; where which-team is 'same or 'other
   (**define-struct** *start-program* (*aprog*))

   ;; A program is a list[cmd]
   (**define-struct** *program* (*commands*))

(exam continues next page)

(b) (10 points) Write the example (offense) program from the previous page in your data definition. As an example of its use, your example could replace the ??? in the following Scheme fragment:

```
(local [(define defense-program (make-program <assume filled in>))
        (define offense-program ???)]
  (make-start-program offense-program))
```

(exam continues next page)

2. (Macros – 30 points) You are writing an adventure game and need a way to write down initial configurations of caves, what they contain, and which other caves they have doors to. Your data definitions are:

```
;; A cave is (make-cave symbol symbol list[cave])
(define-struct cave (name item doors))
;; A maze is (make-maze cave list[cave])
(define-struct maze (init caves))
```

Your current format for defining and connecting caves is as follows (where *set-cave-doors!* is a helper that changes the list of doors that a cave has—necessary for creating circular data).

```
(local [(define entry (make-cave 'entry 'candle empty))
        (define dungeon (make-cave 'dungeon 'sword empty))
        (define snack-bar (make-cave 'snack-bar 'cookies empty))]
  (begin (set-cave-doors! entry (list dungeon))
         (set-cave-doors! dungeon (list entry snack-bar))
         (set-cave-doors! snack-bar (list entry))
         (make-maze entry (list entry dungeon snack-bar))))
```

Write a macro for **the-maze** that will convert the following notation for specifying caves into the above code. The keywords are in bold. Your macro should work on any number and configuration of caves specified in this style. It does not need to include error checking.

```
(the-maze starts-at entry
          (the-cave entry has candle connects-to (dungeon))
          (the-cave dungeon has sword connects-to (entry snack-bar))
          (the-cave snack-bar has cookies connects-to (entry)))
```

(exam continues next page)

3. (Script Position – 35 points) Consider the following programs for handling course registration.

```
(define allcourses (list 'cs1102 'cs2102 'ma1023))

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

(define (choose numcourses)
  (cond [(symbol=? 'yes (prompt-read (format "You have ~a courses. Register for more?" numcourses)))
         (choose-new numcourses)]
        [else empty]))


(define (choose-new numcourses)
  (let ([course (prompt-read "Enter next course name: ")])
    (cond [(member course allcourses) (cons course (choose (+ 1 numcourses)))]
          [else (begin (printf "Invalid course name.  Try again~n")
                       (choose-new numcourses))])))

(printf "You have registered for ~a~n" (choose 0))
```

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

(b) (25 points) Edit the code so that all four defined functions work as scripts (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 if you need extra space.

(exam ends here)