CS1102, A04

Midterm Exam

Name:

Problem	Points	Score
1	35	
2	35	
3	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 definitions, 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?
string? string=? string-length
boolean? **and or** *not*

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

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

---

1. (35 points) The turnin system we've been using to submit homeworks stores information about the assignments and files submitted. Each assignment has a name, a boolean indicating whether it is open for submissions, and up to one submission per student in the class. The following definitions capture turnin info:

```
;; A asgmt is a (make-asgmt symbol boolean list[studentfiles])
(define-struct assignment (name open? submissions))

;; A studentfile is a (make-studfile symbol string)
(define-struct studfile (username code))

;; Example of above definitions
(make-asgmt 'hwk1 true (list (make-studfile 'kfisler "kathi code")
                             (make-studfile 'zellers "adam code")))))
```

   (a) (11 points) Write a program *count-submissions* that consumes an assignment name and a list of asgmt and produces the number of students who have submitted code for the named assignment (assume all students in the submissions list have submitted code). You may use Scheme's built-in function *length* that consumes a list and produces the number of items in that list. **You do not need to use map or filter for this function** (but you may if you wish).

```
;; count-submissions : symbol list[asgmt] → number
;; returns number of students who have submitted code for named assignment
```

(exam continues next page)

(b) (12 points) Write a program *close-asgmt* that consumes an assignment name and a list of asgmt and produces a list of asgmt. The contents of the returned list should be identical to the input list except that the given assignment should have its *open?* field set to false. **Use map and/or filter if possible**.

;; close-asgmt : symbol list[asgmt] → list[asgmt]
;; sets open? field of named assignment to false

(c) (12 points) Write a program *open-names* that consumes an assignment name and a list of asgmt and produces a list of the names (symbols) of all the assignments for which the *open?* field is true. **Use map and/or filter if possible**.

;; open-names : symbol list[asgmt] → list[symbol]
;; return list of names of all open assignments

(exam continues next page)

2. (35 points) Many applications allow users to specify times at which certain actions or tasks should occur (such as "send a reminder at 11am on Tuesday"). In this spirit, operating systems allow users to schedule tasks to perform on their filesystems. Each timed task specifies of the hour and day of week to perform the task and a function that consumes and produces a filesystem (as the task to perform).

   (a) (10 points) Propose a data definition for timed tasks (you may use 24-hour times to avoid having to capture AM/PM). Assume you already have a data definition for filesystems.

   (b) (10 points) Using your data definition, write the example of a timed task that removes all files with size larger than 1000 from the filesystem on Sundays at 3am. Assume you have written a function *rem-large-files* that consumes a file size and a filesystem and removes the files larger than the given size from the filesystem (**you do not need to write** *rem-large-files*).

(exam continues next page)

(c) (15 points) Write a function *run-tasks* that consumes the current day, current hour, a filesystem, and a list of timed tasks and returns a filesystem. The function should run all tasks that are scheduled for the current day and hour on the filesystem. (The final returned filesystem should reflect all of the tasks that ran, not just some of them.)

(exam continues next page)

3. (30 points) Access policies specify what actions different groups of people can perform on an organization's files. For example, a course policy for reading files might state that teachers can read all files and students can read lecture notes (but not grades). We can capture these policies as a tree of and/or operations on requirements like "person is a teacher" and "file is a gradebook" (where each requirement gives an expected value for either the type of person or the type of file):

```
;; A decision-tree is one of
;; - (make-require symbol string) [symbol is 'person or 'file]
;; - (make-andnode decision-tree decision-tree)
;; - (make-ornode decision-tree decision-tree)

(define-struct require (object has-value))
(define-struct andnode (left right))
(define-struct ornode (left right))

;; Example of decision tree for above example
(define read-rules (make-ornode (make-require 'person "teacher")
                                (make-andnode (make-require 'person "student")
                                              (make-require 'file "lecture notes"))))
```

(a) (15 points) Write a program *action-allowed?* that consumes two strings and a decision tree and produces a boolean. The two strings represent a kind of person and type of file that they wish to access. The boolean indicates whether the given person can access the given kind of file. For example, (*action-allowed?* "student" "exam" *read-rules*) should return *false*, while (*action-allowed?* "teacher" "exam" *read-rules*) should return *true*.

```
;; action-allowed? : string string decision-tree → boolean
;; determine whether given person allowed to access given type of file
```

(exam continues next page)

(b) (15 points) A university decides to add TAs as a category of people. They must update certain policies so that TAs get the same access rights as teachers. Write a function *add-TA* that consumes a decision tree and produces a decision tree. In the returned tree, all requirements in which the person is a teacher is expanded to an ornode in which the person is either a teacher or a TA.

;; add-TA : decision-tree → decision-tree
;; produce decision tree in which every requirement about teachers is expanded to allow teachers or TAs

(end of exam)