CS1101, A04

Exam 3

Name:

Problem	Points	Score
1	40	
2	20	
3	40	
	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 (unless a problem states otherwise), but you may develop them if they will help you write the programs.

Your programs may contain only the following Scheme constructs:

define define-struct cond else local begin

and the following primitive operators:

empty? cons? cons first rest list append
$+ - * / = < > <= >=$ *zero? max*
string=? string-length symbol=?
and or *not*
set!

and the operators introduced by **define-struct** (including the set operators on structures).

You may, of course, use whatever constants are necessary (*empty*, *true*, *false*, 0, etc).

(If you did the advanced lab 4 and want to use *map* or *filter*, go ahead, but make sure you know what you are doing first.)

---

1. (40 points) An online bookstore uses the following data definition and variable to track customers' orders:

```
;; An order is (make-order string list-of-string symbol)
(define-struct order (name titles status))
;; Example: (make-order "Erica" (list "Hamlet" "Intro to Calculus") 'paid))

;; orderlist : list-of-order
;; stores the orders that customers have placed
```

where *status* is one of 'placed, 'paid, or 'shipped.

(a) (20 points) Write a function *change-status* that consumes an order and a symbol and changes the order's status to the given symbol. As part of your answer, fill in the EFFECT statement in the documentation provided below.

```
;; change-status : order symbol → void
;; changes status of given order to given symbol
;; EFFECT:
```

(exam continues next page)

(b) (20 points) Write a function *remove-shipped* that removes all orders with the status 'shipped from *orderlist*.

```
;; removed-shipped : → void
;; removes all orders with status 'shipped from orderlist
;; EFFECT: changes orderlist to exclude shipped orders
```

(exam continues next page)

2. (20 points) Using **accumulator style**, write a function *largest-pos* that consumes a list of numbers and produces the largest positive number in the list, or 0 if the list contains no positive number. [HINT: the *largest-pos* function should use a helper that accumulates the largest number seen so far.]

   ;; largest-pos : list-of-number → number
   ;; produces largest positive number in list or 0 if no positive number in list

(exam continues next page)

3. (40 points) A university's registration system uses the following data definitions for students and courses:

```
;; A course is (make-course symbol number list-of-student)
(define-struct course (dept number enrolled))

;; A student is (make-student string list-of-course)
(define-struct student (name schedule))
```

The list of students in each course contains all students who are taking the course, and the list of courses (schedule) in each student contains all the courses that a student is taking.

(a) (15 points) Write a function *register* that consumes a student and a course and enrolls the student in the course.

```
;; register : student course → void
;; registers the student for the course
;; EFFECT: adds student to course's enrollment and adds course to student's schedule
```

(exam continues next page)

(b) (25 points) Assume that the registration system contains the following variables and functions:

- all-students : list-of-student
- all-courses : list-of-course
- add-course : symbol number → void
  adds a course to all-courses
- register : student course → void
  enrolls a student in a course and puts the course in the student's schedule
- drop : student course → void
  removes a student from a course and eliminates the course from the student's schedule
- cancel-course : course → void
  removes a course from all-courses and from all schedules of students in the course
- enrollment : course → number par returns number of students enrolled in the course

i. (15 points) Describe in English what your test cases should check in order to guarantee that these functions work and interact as expected (refer to specific functions in your answer).

(exam continues next page)

ii. (10 points) Write a sequence of Scheme expressions that captures the tests you just described in English (in the previous part). **Write only the test cases—do not write the functions.**

(end of exam)