NAME:

CS 1101
Exam 3
A-Term 2013

Question 1:	_____	(55)
Question 2:	_____	(20)
Question 3:	_____	(25)
TOTAL:	_____	(100)

You have 50 minutes to complete this exam. 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.

You should provide a signature/purpose for any helper function you define.

Your programs may contain only the following DrRacket constructs:

define define-struct cond if else local begin

and the following primitive operators:

empty? cons? cons first rest list append

`+ - * / = < > <= >=`

string=? string-length symbol=? error format
filter map
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.)

1. (55 points) The housing office at a university keeps track of information about each dorm room on campus with the following struct:

```
;; Dorm is a (make-dorm String Number Number Number)
(define-struct dorm (building room sqft beds))
;; interp:  represents a dorm room where
;;          building is the name of the building
;;          room is the room number
;;          sqft is the area of the room in square feet
;;          beds is the number of beds in the room

;; ListOfDorm is one of
;;  empty
;;  (cons Dorm ListOfDorm)
```

A variable called `Rooms` has been defined. Assume that the variable `Rooms` has been populated with information about each dorm room on campus:

```
;; Rooms:  ListOfDorm
;; remembers information about each dorm room on campus
(define Rooms (list (make-dorm ...)
                    (make-dorm ...)
                    ...))
```

   (a) (10 points) For each of the following operations, state whether a solution could be written using set!, set-structure!, or both (your answer to each part should be one of **set!**, **set-structure!**, or **both**.

      i. add a new dorm room to `Rooms`

      ii. add another bed to every room in the list that has at least 300 square feet

   (b) (5 points) Provide the signature for the mutator `set-dorm-building!`.

(c) (20 points) Write a function that satisfies the following signature and purpose:

```
;; add-bed-to-room:  Number String ListOfDorm -> void
;; Adds 1 to the number of beds in the room with the given room number
;; in the given building.  (You may assume that room numbers are
;; unique within a building.)
;; EFFECT:  modifies one of the dorm rooms in the given list
```

Use one of the set-structure! operators (don't use set! to solve this problem).

(d) (20 points) One of the buildings on campus is scheduled to be demolished, so all dorm rooms in that building are to be removed from the `Rooms` list. Write a function that satisfies the following signature and purpose:

```
;; remove-from-Baxter:  -> void
;; removes all rooms in the Baxter building from the list of dorm rooms
;; EFFECT:  changes the Rooms variable
```

Use set! (don't use a set-structure! operator to solve this problem).

2. (20 points) Here are the data definitions from Problem 1 again:

```
;; Dorm is a (make-dorm String Number Number Number)
(define-struct dorm (building room sqft beds))
;; interp:  represents a dorm room where
;;          building is the name of the building
;;          room is the room number
;;          sqft is the area of the room in square feet
;;          beds is the number of beds in the room

;; ListOfDorm is one of
;;  empty
;;  (cons Dorm ListOfDorm)
```

Using map and/or filter, as appropriate, write a program that satisfies the following contract and purpose:

```
;; large-rooms:  ListOfDorm String -> ListOfNumber
;; consumes a list of dorm rooms and the name of a building and produces
;; a list of the room numbers in the given building that are more than
;; 300 square feet in area
```

3. (25 points) Here are the data definitions from Problem 1 again:

```
;; Dorm is a (make-dorm String Number Number Number)
(define-struct dorm (building room sqft beds))
;; interp:  represents a dorm room where
;;          building is the name of the building
;;          room is the room number
;;          sqft is the number of square feet in the room
;;          beds is the number of beds in the room

;; ListOfDorm is one of
;;  empty
;;  (cons Dorm ListOfDorm)
```

Re-write the program `large-rooms` from Problem 2, but this time, instead of using map and/or filter, use accumulator-style programming. Here's the signature and purpose for `large-rooms` again:

```
;; large-rooms:  ListOfDorm String -> ListOfNumber
;; consumes a list of dorm rooms and the name of a building and produces
;; a list of the room numbers in the given building that are more than
;; 300 square feet in area
```

(Hint: you need to write at least two functions.) You may continue your answer on the next page if you need more space.

(extra page if you need additional space for your answers)