Assignment 5
Due 4:00 PM on Monday, November 8

For this and all subsequent assignments, you are expected to use the design recipe when writing functions from scratch. Do not copy the purpose directly from the assignment description. The purpose should be written in your own words and include reference to the parameter names of your functions. The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources. Test data for all questions will always meet the stated assumptions for consumed values.

Please read the course Web page for more information on assignment policies and how to organize and submit your work. Be sure to download the interface file from the course Web page and to follow all the instructions listed in the style guide (on the Web page). Specifically, your solutions should be placed in files a5qY.rkt, where Y is a value from 1 to 4. For full marks, it is not sufficient to have a correct program. Be sure to follow all the steps of the design recipe, including the definition of constants and helper functions where appropriate. **DO NOT use the built-in `reverse` and `make-list` functions.**

Language level: Beginning Student.
Coverage: Modules 5 & 6
Useful structure and data definitions:

```
(define-struct contact (name email-address))
;; A contact is a structure (make-contact n e) where
;;      n is a non-empty string representing a person's name and
;;      e is a non-empty string representing a person's email address.

(define-struct email (from to subject message))
;; An email is a structure (make-email f t s m) where
;;      f is a contact representing who sent the email,
;;      t is a contact representing who received the email,
;;      s is a string representing the subject of the email, and
;;      m is a non-empty string representing the text of the message.

(define-struct card (value suit))
;; A card is a structure (make-card v s) where
;;      v is an integer in the range from 1 to 10, and
;;      s is a symbol from the set 'hearts, 'diamonds, 'spades, and 'clubs.
```

1. In simplified terms, a radar station can detect incoming objects within its radius (including those at the boundary of the radius). For example, a radar station located at (0, 0) with a radius of 100 can identify the objects at (20, -10) and (-100, 0), but not the object at (100, 200). Write a function called `identify-objects` that consumes a `posn` representing the

coordinates of the radar station, a `number` representing the radius of the radar, and a list of `posn` structures representing the coordinates of objects to be identified, and produces a list of `posn` structures representing the coordinates of objects within the radius of the radar station.

For example,

```
(identify-objects (make-posn 0 0) 100
                  (cons (make-posn 20 -10)
                    (cons (make-posn -100 0)
                      (cons (make-posn 100 200)
                        (cons (make-posn -90 -89) empty))))) =>
(cons (make-posn 20 -10) (cons (make-posn -100 0) empty)).
```

2. Write a function called `flush?` that consumes a list of `card` structures and produces `true` if the list of given cards has a flush, and produces `false`, otherwise. A hand of cards has a flush if there are at least five cards with the same suit. Note that the list of cards consumed could be of any length and could contain cards with different suits.

   For example,

```
(flush? (cons (make-card 1 'diamonds)
         (cons (make-card 10 'diamonds) empty))) => false,

(flush? (cons (make-card 3 'diamonds)
         (cons (make-card 9 'diamonds)
          (cons (make-card 8 'diamonds)
           (cons (make-card 1 'clubs)
            (cons (make-card 1 'diamonds)
             (cons (make-card 10 'diamonds) empty)))))))  => true.
```

3. Write a function called `make-address-book` that consumes a `string` representing the email address of a user, a list of `email` structures representing the emails sent or received by the user, and produces an address book, which is a list of `contact` structures that are the senders/recipients of the given emails. Each contact in the address book must be unique. That is, the user might send or receive multiple emails to/from the same person, but only one copy of this person's contact information should be kept in the address book. Moreover, the address book should not contain the contact information of the user. You can assume that each email is either sent or received by the user and the user might send email to himself.

   Example #1:

```
(make-address-book "m47liu@uwaterloo.ca" empty) => empty,
```

Example #2:

```
(define my-emails
  (cons(make-email (make-contact  "M. liu" "m47liu@uwaterloo.ca")
                   (make-contact "J.P. Pretti" "jpretti@uwaterloo.ca")
                   "Hello" "I love CS115!")
       (cons (make-email (make-contact "L. Case" "lcase@uwaterloo.ca")
                         (make-contact  "M. liu" "m47liu@uwaterloo.ca")
                         "Where is Jeff?" "Out of the country.")
             (cons(make-email(make-contact "J.P. Pretti" "jpretti@uwaterloo.ca")
                             (make-contact "M. liu" "m47liu@uwaterloo.ca")
                             "Progress" "Which slide?")
                  (cons(make-email(make-contact  "M. liu" "m47liu@uwaterloo.ca")
                                  (make-contact  "M. liu" "m47liu@uwaterloo.ca")
                                  "Reminder" "Pick up the kids.") empty)))))
(make-address-book "m47liu@uwaterloo.ca" my-emails) =>
   (cons (make-contact "J.P. Pretti" "jpretti@uwaterloo.ca")
         (cons (make-contact "L. Case" "lcase@uwaterloo.ca")empty)).
```

Note that depending on the particular implementation, contacts in an address book might appear in different order. For instance, when running example #2, some implementation might produce:

```
   (cons (make-contact "L. Case" "lcase@uwaterloo.ca")
         (cons (make-contact "J.P. Pretti" "jpretti@uwaterloo.ca")empty)).
```

It contains the same set of contacts as showed in the example, but in different order. **This is also a CORRECT output**.

Hint: the built-in predicate function `equal?` can be used to check whether two structures are the same.

4. Chuckie Lucky won a million dollar, which he places in an account at 8% interest rate, compounded annually. That is, the interest is 8% of the current balance, including previous addition of interest. On the last day of each year, Chuckie withdraws $100,000. Write a program called `balance` that consumes a `number` representing the given number of years，calculates the balance after the given number of years has elapsed, and produces either a `number` representing the balance if it is zero or positive, or produces the symbol `'Overdrawn` if the balance is negative. You can assume that the interest will be added to the account before Chuckie withdraws the money.

   For example, `(balance 0)` -> `1000000`, `(balance 10)` -> `710268.75068180332544`, `(balance 21)` -> `'Overdrawn`.

   Hint: use structural recursion to calculate the balance after the given number of years has passed.