C211: Spring 2015
Midterm: Part II (Practical)

Name: _____

Username: _____

Lab Section: _____

Instructions:

- **Important:** Do all your work in BSL+. Do **not** use any of the following built-in procedures (or logical equivalents that you define yourself) to solve any of these problems:

  ```
  append
  reverse
  length
  remove
  ```

- Follow the Design Recipe for *all* functions that you write.
- You have until exactly 9:30pm to complete the four (4) problems on this part of the exam using the lab computer.
- At the end of the exam, you will submit your work electronically, using C211 Handin, *and* you will submit this paper to the proctor. Space is provided on this paper for scratch work, but nothing that you write here will be graded. Your grade for this part of the exam will be based entirely on what you submit electronically.

Procedures: We are following the same procedures as you practiced in lab last week and have been announced on the course website. We include these here for completeness.

- No devices, other than the lab computer, are permitted.
- The only applications you may have open are DrRacket and a browser.
- Your Ethernet cable must be disconnected.
- This part is open-book. You may use whatever printed materials you have brought with you, but you may not share materials with other students.
- You may refer to any electronic material on your local computer.
- You may open and read the online documentation (F1) at any time.

How to Submit:

- When you are done, let your proctor know that you are ready to submit.
- Reconnect your cable and turn in your work under assignment `midterm` using C211 Handin.
- If you've made a typo in a function name, then you may correct it and resubmit.
- You must turn in this paper to the proctor before leaving the exam room.

4. Design a function `bookends` that takes a natural number `n` and returns a list containing the value `'end` as the first and last elements, with `n` occurrences of the value `'book` in between.

```
(check-expect (bookends 0) '(end end))
(check-expect (bookends 4) '(end book book book book end))
```

5. A big enough list is one with two or more elements.

   (a) In comments, write a data definition for a BigEnoughListOfNum.

   (b) Design a function `widest-gap` that takes a list of two or more numbers, and returns the maximal difference between two consecutive numbers in the list.

```
(check-expect (widest-gap '(4 9)) 5)
(check-expect (widest-gap '(9 4)) 5)
(check-expect (widest-gap '(4 9 1 7 -5 0 3 8)) 12)
```

6. In this problem, you will first define a function in part (a) and then make use of it in part (b).

   (a) Design a *recursive* function `replicate-first` that takes a natural number `n` and a non-empty list, and returns the list with the first element being replicated `n` times. Notice that if `n` is zero, this results in removing the first element.

```
(check-expect (replicate-first 0 '(good bye)) '(bye))
(check-expect (replicate-first 1 '(stays the same)) '(stays the same))
(check-expect (replicate-first 3 '(warning! alien approaching))
              '(warning! warning! warning! alien approaching))
```

   Before you begin coding, go back and re-read the first instruction on the front of this exam.

   (b) Design a *recursive* function `replicate-all` that takes an item, a natural number `n`, and a list, and replaces each occurrence of the item in the list with `n` copies of itself. Make appropriate use of `replicate-first`. Note that if `n` is zero, this results in removing all occurrences of the item from the list.

```
(check-expect (replicate-all 'penny 0 '(penny nickel penny penny dime))
              '(nickel dime))
(check-expect (replicate-all 'ever 5 '(never ever do that again!))
              '(never ever ever ever ever ever do that again!))
(check-expect (replicate-all 'duck 3 '(duck goose duck goose))
              '(duck duck duck goose duck duck duck goose))
```

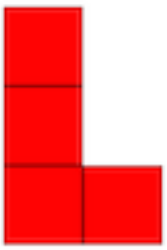
   Before you begin coding, go back and re-read the first instruction on the front of this exam.

7. This problem extends the Simple Tetris game you implemented in a7.

   (a) Define two variables, `WIDTH` and `HEIGHT`, to have the values 7 and 10, respectively. These represent the logical size of the grid of blocks.

   (b) Define a structure named `block` with three fields: `row`, `col`, and `color`.

   (c) In the actual Tetris game, the dropping piece is a cluster of several blocks, all of which are the same color. We represent a Piece with a list of Block. One such piece is shaped like the letter L. Define a variable `L-piece` corresponding to a red L-shaped piece located in the upper left corner of the grid.



   (d) Design a predicate `hit-bottom?` that takes a Piece and returns `true` if any block within the piece has reached the bottom row, and `false` otherwise.

   (e) Design a function `piece-down` that takes a Piece and returns the result of moving it down one row, if possible. However, if the piece has already hit the bottom, then just return it unchanged.