Lab 08: Structural recursion on numbers

Create a separate file for each question. Keep them in your "Labs" folder, with the name `liiqj` for Lab ii, Question j.

Download the headers for each function from the file `labinterface08.rkt` linked off the "Labs" page on the course Web site.

After you have completed a question (except class exercises), including creating tests for it, you can obtain feedback by submitting it and requesting a public test. Follow the instructions given in the Style Guide.

Language level: Beginning Student with List Abbreviations

1. *[Class exercise with lab instructor assistance]* Create a function *is-prime?* which consumes a positive natural number *n* and produces *true* if *n* is a prime number.

2. [Modified from HtDP Exercise 11.2.1] Create the function *repeat*, which consumes a natural number *n* and a symbol *symb* and produces a list with *n* occurrences of the symbol.

3. [Modified from HtDP Exercise 11.5.3] Recall that x^n means multiplying x with itself n times. Create the function *exponent*, which consumes a natural number n and a number x and computes x^n without using the built-in exponentiation function *expt*.

4. Now develop the function *exponent-without-mult*, in which the only built-in arithmetic functions allowed are the functions *add1* and *sub1*, and both x and n are natural numbers. Be prepared for it to be very slow to run!

5. Create a function *largest-prime* that consumes two positive natural numbers, *bottom* and *top*, and produces either the largest prime number in the range from *bottom* and *top* (inclusive) or *false* if there is no prime in that range.

6. Create a function *make-sqr-al* that consumes two numbers, *n* and *b*, and creates an association list, where keys go from *n* up to *b* and each value is the square of the key. For example, (*make-sqr-al* 1 2) will yield (*list* (*list* 1 1) (*list* 2 4)).

7. *Optional open-ended questions* In the game "war", two players compete to collect cards. Each player has a stack of cards. In each turn, the topmost cards in each stack are compared, with the winner (the player with the higher card value) collecting both. In a simplified version, each player's cards are stored as a list of card structures (recall that a card is a structure (*make-card v s*), where *v* is an integer in the range 1 to 10 and *s* is a symbol from the set 'hearts, 'diamonds, 'spades and 'clubs; the structure definition is (*define-struct card* (*value suit*))). If there is a tie between card values, each player keeps their own card. Create a function *war* that consumes two lists of cards, hand1 and hand2, and produces the contents of hand1 at the end of a round. You can make further enhancements, or consider other games played in a similar manner.