HW1

Homework 1

Due: 11:59pm, Thursday, Sep 08, 2022

100 points

For all Racket assignments in this course, set the DrRacket Language to **Intermediate Student with lambda** (under **How to Design Programs**). Your assignment will be graded using the specified language level. If you use a different language level, your code may not work when it is graded.

- Carefully follow the **Sample Solution to a Programming Problem** in the **Racket HW Guide**. Only half of the credit for each programming problem is based on the the correctness of your code as determined by our test cases. Much of the grade is based on how well you follow the *design recipe*. For a crisp example of what an ideal solution looks like look at the **Sample Solution to a Programming Problem** that sorts **list-of-number** at the end of **Sample Solution to a Programming Problem** in the the **Racket HW Guide**. This process may appear painful but it shows "in the small" how program design should be done "in the large". It is not difficult. If you carefully inspect the sample program, it shows the level of detail in describing your program design (and its derivation!) that we want.

- **Do the following programming problems:**
  1. [10 pts] Develop the function **`contains?`** that consumes a symbol and a list of symbols and determines whether or not the symbol occurs in the list.
  2. [10 pts] Develop the function **`count-symbols`** that consumes a list of symbols and produces the number of items in the list. [Note: the function merely works on inputs that are lists of symbols; it may blow up on anything else].
  3. [10 pts] Develop the function **`count-numbers`** that counts how many numbers are in a list of numbers. [Note: the function merely works on inputs that are lists of numbers; it may blow up on anything else].
  4. [20 pts] Develop the function **`avg-price`**. It consumes a list of toy prices and computes the average price of a toy. The average is the total of all prices divided by the number of toys. toy prices are numbers. [Hint: develop a few auxiliary functions (following the design recipe to develop each auxiliary function) that you can use to make the definition of **`average-price`** very easy. If the list of toy prices is empty, the function **`avg-price`** produces an error message as described in Guidance below.
  5. [10pts] Develop the function **`elim-exp`** to eliminate expensive toys. The function consumes a number, called **`mp`** (short for "maximum price") and a list of toy prices, called **`lotp`**, and produces a list of all those prices in **`lotp`** that are below or equal to **`mp`**. For example

     ```
     (check-expect (elim-exp 1.0 (list 2.95 .95 1.0 5) (list .95 1.0)) = #true
     ```

  6. [10pts] Develop the function **`delete`** to eliminate specific toys from a list. The function consumes the name of a toy, called **`ty`**, and a list of names, called **`lon`**, and produces a list of names that contains all components of **`lon`** with the exception of `ty`. For example,

     ```
     (check-expect (delete 'robot (list 'robot 'doll 'dress)) (list 'doll 'dress)) = #true
     ```

  7. [30pts] A list can be used to represent a finite set. For example,

     ```
     (list 'c 'o 'm 'p)
     ```

     represents the set of symbols {'c , 'o, 'm, 'p}. In such a representation, we assume all elements in the list are unique; there are no duplicates. Develop the function **`power`** that consumes a list of symbols **`los`** (representing a set) and produces a list of list of symbols representing the power set (set of all subsets) of **`los`**. Hint: write an auxiliary function **`cons-all`** that consumes a symbol **`sym`** and a list of list of symbols **`lolos`** and inserts symbol **`sym`** at the front of each list in **`lolos`**.

     For example, `(check-expect (cons-all 'a (list (list 'c) (list 'o) (list 'm) (list 'p)) (list (list 'a 'c) (list 'a 'o) (list 'a 'm) (list 'a 'p))) = #true`

- **Guidance:**

  1. Follow the design recipe imitating **Sample Solution to a Programming Problem** in the **Racket HW Guide**.
  2. Problem 4 asks you to write a function that checks for the empty list as an input error and throws an aborting error in the case. (The purpose statement should document this behavior!) In DrRacket, the **`error`** function takes a single argument *not two arguments as documented in the book*. We recommend using a string (text enclosed in double quotation marks) like `"An empty list of toy prices triggers this aborting error"` as the argument. You can test the error-throwing behavior of a function using **`check-error`** which is documented in the DrRacket Help Desk.
  3. Study Figure 26 in 9.4 for a detailed description of the design recipe and how it is documented in the program text that you develop.

     To follow the design recipe, you must write down the *type contract, purpose*, provide at least 3 well-chosen examples (more for complex functions like `power`), write the template for the function (trivial when no recursion is involved), write the code for the function, and include illustrative test cases for the function (using at least the examples you developed ahead of time). You should bundle the examples and test cases together as a block of **`check-expect`** invocations, which was not part of DrRacket when the First Edition of the book was written. These tests should *precede* your template and code. Your chosen examples should illustrate the output you expect, and the test cases should produce the actual output (leave them uncommented). If the function processes an inductive type, make sure that your examples include the base case(s) and sample inductive cases.