CS 116 Winter 2013

Assignment 02

Due at 10:00am on Wednesday, January 30

- This assignment consists mostly of material from Module 02, on functional abstraction and functions as values.
- You must use abstract list functions (`map, filter, foldr, build-list`) where appropriate. You may use the function `length`.
- For this assignment, unless otherwise noted, you **cannot** use `local` or explicit recursion (i.e., you may not write a function which calls any helper functions or calls itself).
- Do **not** use `reverse`.
- For this and all subsequent assignments, you are expected to use the design recipe when writing functions from scratch, including contracts and purposes for local helper functions. For full marks, it is not sufficient to have a correct program.
- 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.
- Download the interface file from the course Web page.
- Read the course Web page for more information on assignment policies and how to organize and submit your work. Follow the instructions in the style guide. Specifically, your solutions should be placed in files `a02qY.rkt`, where `Y` is a value from 1 to 4.
- Read each question carefully for restrictions. Test data for all questions will always meet the stated assumptions for consumed values.
- Assignments will not be accepted through email. Course staff will not debug code emailed to them.
- Read the assignment carefully before asking any questions on Piazza.

Language level: Intermediate Student with Lambda
Coverage: Module 2

1. Repeat Question 1 from Assignment 1, except do not use `local` and use only one `define` function. Formally, write a function `ones` that consumes a list of natural numbers (`nats`) and produces the number of values in the list whose one's digit (i.e. the least significant or rightmost digit) is 1. For example,
   `(ones (list 82 231 1 22 1000)) => 2.`

2. Write the function `match-points` that consumes two functions (`fun1` and `fun2`, each of which consumes one value) and one list (`lst`). The function `match-points` produces a list of boolean values which has the same length as `lst`. The first value in the produced list will be `true` if the functions agree (i.e., have the same value) when they are applied on the first element of the provided list, and `false`

otherwise. The second value in the produced list will be `true` if the functions produce the same value when they are applied on the second element in the provided list, and so on. For example,

```
(match-points sqr (lambda (x) (+ x x)) (list 0 2 -2))
=> (list true true false)
```

since $0+0=0^2$ and $2+2=2^2$, but $-2+(-2)=-4$ which is not equal to $(-2)^2=4$.

3. Write the function `mult-table` which consumes two natural numbers `nr` and `nc`, and produces a list of `nr` lists, each of which contains `nc` natural numbers. The c^{th} entry of the r^{th} list (where we start numbering at 0) in the produced list should be $r*c$.
   For example,
   ```
   (mult-table 2 3) => (list (list 0 0 0) (list 0 1 2))
   ```
   and
   ```
   (mult-table 3 4) =>
      (list (list 0 0 0 0)
            (list 0 1 2 3)
            (list 0 2 4 6)).
   ```
   Hint: use `build-list` twice. The first time, try to make a list containing `nr` elements. Then, for each of these elements, make a list of `nc` elements. You may use other abstract list functions as well.

For the next problem, you will find the following definition useful:

```
(define-struct coin (name value))
;; A coin is a structure (make-coin n v),
;; that represents a valued coin, where
;;   n is a symbol, representing the name of the coin
;;   v is a num, representing the value of the coin
```

4. Write the function `money-count`, which consumes a list of coins called `coinlist`, and produces a function which consumes a list of symbols `change` and produces the value of the coins in `change`. The `coinlist` is a `(listof coin)` and `change` is a list of symbols. The `coinlist` contains the names and values of the coins, and the list `change` contains names of coins whose total value should be the value produced by the function produced by `money-count`. You can assume that all coin names in the `coinlist` are distinct, and that if a coin is not defined, it has no value. For example:
   ```
   ((money-count
     (list (make-coin 'penny 0.01) (make-coin 'quarter 0.25)))
     (list 'penny 'quarter 'nothing 'penny)) => 0.27
   ```
   You may use `local` for this question, though it is not required.
   Hint: start by dealing with very small lists (i.e., those which contain either only one coin or symbol).
   Hint 2: `foldr`, `map` and `filter` are all used in the model solution for this problem.
   Hint 3: think about finding a match between each element in `change` and the values in `coinlist`.