COMP 50 Lab: Simplifying Code

October 30–31, 2013

This lab gives you supervised practice at two completely different kinds of code simplification:

- Rewriting code to make more effective use of primitives
- Exploiting the built-in abstract functions on lists

Simplifying code fragments

Work on this part for *at most 15 minutes*.

1. Simplify each of these following code fragments. Write the simplified version directly into your lab.

   a. Simplify this expression:

   ```
   (append (list1 (first a-lon)) (nub (rest a-lon)))
   ```

   b. Simplify this expression:

   ```
   (boolean=? true (world-start? a-world))
   ```

   c. Simplify this expression:

   ```
   (boolean=? false (world-hit? a-world)
   ```

   d. Simplify this definition:

   ```
   (define (atom-element? atom)
     (cond
       [(string? atom) true]
       [(not (string? atom)) false]))
   ```

   e. Simplify this definition:

   ```
   (define (format-check sl)
     (cond
       [(symbol? (first sl)) (list-of-elements? (rest sl))]
       [(not (symbol? (first sl))) false]))
   ```

   f. Simplify this expression:

   ```
   (cond
       [(and (is-element? (first lov)) (is-loe? (rest lov))) true]
       [else false])
   ```

Exploiting built-in "loopy" list functions

Work on this part for an hour. If we got reasonably far in lecture, and if you worked on the `nub` problem last week, you can hope to finish the first two problems.

2. *Selecting Scrabble words*
   The goal in this problem is to get permissible Scrabble words from a longer list of words that includes impermissible contractions, possessives, and capital letters. Permissible Scrabble words are made from lower-case letters *only*.

   a. If we didn't finish this problem in class, design and implement a reusable function for purpose statements of the form "to produce a list of all *somethings* on this list that *satisfy property*."
      Be sure to write a signature with *type variables* (representing the unknown data that is abstracted over).

   b. If we didn't finish this problem in class, design and implement a reusable function of them "to tell if all the *somethings* on this list satisfy *property*."
      Be sure to write a signature with *type variables* (representing the unknown data that is abstracted over).

   c. Using your functions from parts a and b, together with the ISL functions `string->list` and `char-lower-case?`, *define a function* that is given a list of words and produces another list containing those words on the given list that are made from all lower-case letters.

3. *Eliminating duplicates, loosely defined.*
   This problem has four parts. You've seen the first part before.

   a. *Define a function* `nub-numbers` which removes duplicates from a list of numbers. Your function may remove any duplicate numbers it likes, but the order of non-duplicate numbers must not be disturbed.
      Here are some functional examples:

```
(check-expect (nub-numbers '(1 2 3 4 5)) '(1 2 3 4 5))
(check-expect (nub-numbers '(1 2 3 4 5 5 5 5)) '(1 2 3 4 5))
(check-expect (nub-numbers '(2 1 2 4 5 2)) '(2 1 4 5))
```

   b. *Define a function* `nub-strings` which removes duplicates from a list of strings. The order of non-duplicate strings must not be disturbed.

   c. *Define a function* `nub-images` which removes duplicates from a list of images. The order of non-duplicate images must not be disturbed.

   d. *Define a function* `nub-last-digits` which takes a list of nonnegative whole numbers and returns that same list with "last-digit duplicates" removed. Two numbers are last-digit duplicates if they have the same last digit. The order of other, non-duplicate numbers must not be disturbed.
      Here are some functional examples:

```
(check-expect (nub-last-digits '(1 2 3 4 5)) '(1 2 3 4 5))
(check-expect (nub-last-digits '(10 20 30 40 50)) '(10))
(check-expect (nub-last-digits '(11 12 17 32 39 21)) '(11 12 17 39))
```

      Domain knowledge: the last digit of a nonnegative whole number is its remainder when divided by 10.

   e. *Generalize your functions* by defining a general-purpose duplicate-removal function that can be specialized to handle all of the cases above.

   f. Using your new general function, remove "same-digits duplicates" from a list of numbers. Two numbers are same-digits duplicates if they are made from the same digits.
      Hint: use `number->string`, `explode`, and `anagrams?`.

4. *Testing for Scrabble words*
   A typical Linux distribution ships with a dictionary that contains around a half a million words. You can test your code on a list of that size, but you wouldn't want to look at the results. In this problem you'll find a way to look at a fraction of the results.

   Define a function `take` that takes as arguments a natural number and a list, and it returns the given number of elements from the start of the given list. If it runs out of elements, `take` should return the entire list.

   For `take`, use the design recipe for *multiple complex arguments*. Remember the conditions `zero?` and `positive?`, and use `sub1` as a "selector function" for positive natural numbers.

   Try out your permissible words function with

   ```
   (take 100 (produce-permissible-words (read-lines "/usr/share/dict/words")))
   ```

   To get the `read-lines` function, you will need the `batch-io` teachpack from `2htdp` (the middle list of teachpacks.)

5. *More testing for Scrabble words*
   How many words are in `/usr/share/dict/words`? What percentage (to the nearest whole percent) of those words are permissible in Scrabble?

Submitting the lab

Ten minutes before the end of the lab, put the following text at the beginning of a DrRacket file. You may use an empty file or the source code you have been working on:

```
#|
What I did during this lab:
   (you fill in this part)

What I learned during this lab:
   (you fill in this part)

|#
```

The lab staff will help you articulate what you learned.

Finally, *submit this file through the handin server* as `lab-simplify`. You will need to submit it using *two* usernames connected by a + sign, as in

```
Jane.Doe+Richard.Roe
```

You submit using Jane Doe's password.