CS 152 / SE 152
Programming Language Paradigms

Spring Semester 2014

Department of Computer Science
San Jose State University
Prof. Ron Mak

Assignment #2

Assigned: Monday, February 17
Due: Monday, February 24 at **11**:59 pm
Team assignment, 100 points

Recursive Scheme procedures

Write the following Scheme procedures.

You may write and use helper procedures. For each problem, **at least one** of your main procedure or any helper procedures **must be recursive**.

Demonstrate the capabilities of each of your procedures with a variety of sample arguments. Be sure to test your procedures with simple lists such as **`(a b c)`**, lists with sublists such as **`(a (b (c d)) e)`**, and of course, the empty list **`()`**. Your procedures **should not throw exceptions**, such as by attempting to take the **`car`** or **`cdr`** of an empty list.

For this assignment, you do not need to validate the arguments to your procedures. For example, it an argument is supposed to be an integer whose value is nonnegative, you do not need to check that the argument is indeed a nonnegative integer.

1. Procedure **`nest`** generates a nest of **n** lists, where argument **n** is an integer whose value is 0 or greater.)

   - Examples: **`(nest 0)`** ➔ **`()`**
     `(nest 1)` ➔ **`(())`**
     `(nest 2)` ➔ **`((()))`**

2. Procedure **`alternate`** returns every other element from a list, starting with the first element.

   - Example: **`(alternate '(a b (c d) e f)) ➔ (a (c d) f)`**
     `(alternate '()) ➔ ()`

3. Predicate procedure **`all-same?`** tests whether a given list has all the same elements.

   - Examples: **`(all-same? '((a b) (a b) (a b))) ➔ #t`**
     `(all-same? '(a)) ➔ #t`
     `(all-same? '()) ➔ #t`
     `(all-same? '(x y z)) ➔ #f`

4. Procedure **`remove-leading`** has two arguments, an item **`item`** and a list **`lst`**. Remove the leading elements from **`lst`** up to but not including **`item`**.

   - Examples: **`(remove-leading 'x '(a b c x y z)) ➔ (x y z)`**
     `(remove-leading 'x '(a b c)) ➔ ()`
     `(remove-leading '(p q) '(a (p q) z))`
     `➔ ((p q) z)`

5. Procedure **`subst-first`** has three parameters: an item **`new`**, an item **`old`**, and a list. Replace the first top-level occurrence of item **`old`** in the list with item **`new`**. Items **`new`** and **`old`** can themselves be lists.

   - Example: **`(subst-first 'dog 'cat '(my cat is smart))`**
     `➔ (my dog is smart)`

6. Procedure **`translate`** has two arguments, an item **`word`** in the source language and a list **`dictionary`**. List **`dictionary`** is a list of pairs of the form **`(source target)`** where **`source`** is a word in the source language and **`target`** is a word in the target language. Translate **`word`** by looking up its equivalent in the target language.

   - Example: **`(translate 'cat '((dog chien) (cat chat)))`**
     `➔ chat`

7. Procedure **`sandwich-first`** takes two items **`a`** and **`b`** and a list as its arguments. Replace the first top-level occurrence of two successive **`b`**'s in the list with **`b a b`**.

   - Example: **`(sandwich-first 'meat 'bread`**
     `'(bread cheese bread bread mustard))`
     `➔ (bread cheese bread meat bread mustard)`

8. Procedure **`count`** has two arguments, an item **`item`** and a list **`lst`**. Count the number of occurrences of **`item`** in **`lst`**, no matter how deeply nested it is.

    - Examples: **`(count 'x '(a b x c x x d)) ➔ 3`**
      `(count '(p q) '(a (p q) (b ((p q) c)) d))`
      `➔ 2`

9. Procedure **`union`** has two list arguments. Return a list that is the union of the argument lists and which contains no duplicates.

    - Examples: **`(union '(a b (c)) '(b (c) d)) ➔ (a b (c) d)`**
      `(union '() '(a b c)) ➔ (a b c)`

10. Procedure **`intersect`** has two list arguments. Return a list that is the intersection of the argument lists and which contains no duplicates.

    - Examples: **`(intersect '(a b (c)) '(a (c) d)) ➔ (a (c))`**
      `(intersect '(a b b c) '(b c c d)) ➔ (b c)`
      `(intersect '(a b c) '((c) d e)) ➔ ()`

What to turn in

Each team turns in one zip file consisting of:

- Text file(s) containing your Scheme procedures and output from your test arguments.
- A short report that briefly explains how each procedure works (one or two paragraphs per procedure). Instead of the separate report, you can include the explanation as comments in your procedures.

Email the zip file to ron.mak@sjsu.edu.

Important: Name the zip file after your team and append the assignment number, for example **`SuperCoders-2.zip`**. Some mailers may not allow you to mail zip files, so you may have to rename the file to have the suffix other than **`.zip`**, such as **`.zzz`**. Do not include **`.class`** files and do not email executable files.

The subject line of your email should be **CS 152 Assignment #2,** ***team name***
for example **`CS 152 Assignment #2, SuperCoders`**

CC all the team members so that I can do a "Reply all" when I send out your score. This is a team assignment. Each member of the team will receive the same score.