**CSCI 3210 Project4**

Due: see class calendar

**Goal: To learn how to use functional programming language to solve a problem.**

**Description:**

In this assignment, you are required to write functions specified below in Scheme, one of the functional programming languages.

* Write a Scheme function ***sphereVolume(r)*** that given the radius r of a sphere, computes its volume as 4/3 \* 3.14 \* r \* r \* r.
* Write a Scheme function ***power(A, B)*** that takes two numeric parameters, A and B, and returns A raised to the B power.
* Write a Scheme function ***countZero(list)*** that returns the number of zeros in a given **simple list** of numbers.
* Write a Scheme function ***reverse(list)*** that returns the reverse of its **simple list** parameter.
* Write a Scheme function ***findEnds(list)***  that takes a **simple list** of numbers as its parameter and returns the largest and smallest numbers in the list.
* Write a Scheme function ***replace(atom1, atome2, list)*** that takes two atoms and **a list** as parameters and returns a list identical to the parameter list except all occurrences of the first given atom in the list are replaced with the second given atom, no matter how deeply the first atom is nested.

Note: A simple list is a list such that none of its element is a list. For example, ‘(1 2 3) is a simple list, while ‘((1 2) 3) is not a simple list.

To avoid typing long lists at the command line, I suggest you create a few sample arguments in files:

> (load "my\_lists")

> (countZero list1)

where the file my\_lists contains the definition

(define list1 '(1 9 0 12 0 30 50 0))

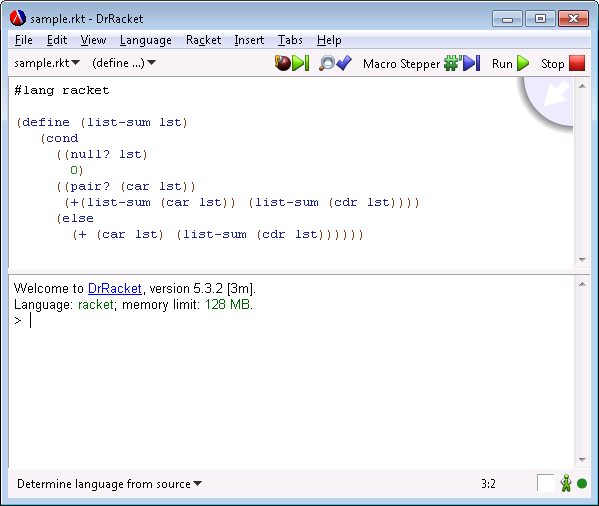
**Important**:  *you are required to use* ***only*** *the functional features of Scheme; functions with an exclamation point in their names (e.g. set!) and input/output mechanisms other than load and the regular read-eval-print loop are not allowed.  (You may find imperative features useful for debugging.  That’s ok, but get them out of your code before you hand anything in.)*

**Requirement**:

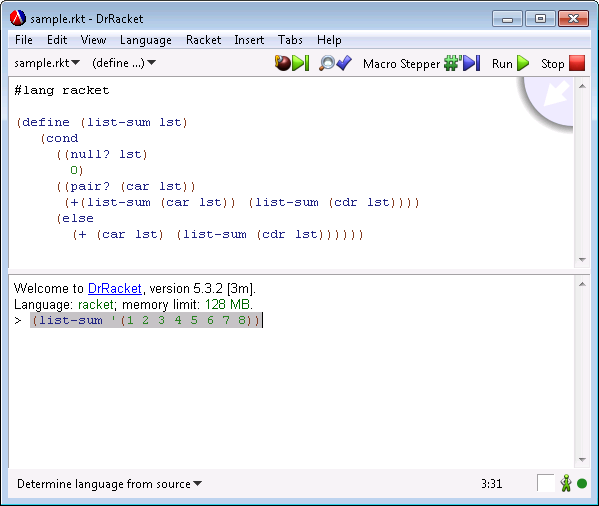
* All functions should be put within the same source file.
* The signature of each function (including function name and order of parameters) must match the given signature.

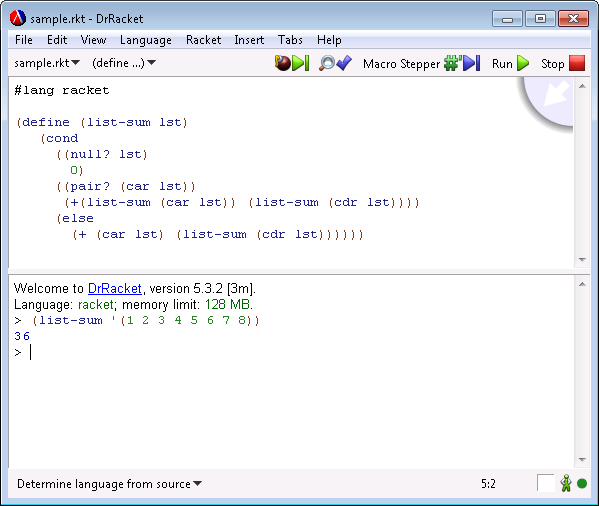
**IDE: Dr. Racket**

Dr. Racket is the IDE we are going to use. The software is free and can be downloaded from: <http://racket-lang.org/download/>. It supports different platforms such as Windows, Mac, or Linux. You can also use Dr. Racket on shemp.cs.mtsu.edu. Here is how it looks like:



The top window is the editor, and bottom half is the command line window. After completing coding, click the Run button on the menu bar, and if there is no syntax error, functions defined in the code can be invoked from the command line window. The following pictures gives an example: after typing (list-sum '(1 2 3 4 5 6 7 8)) and press ENTER, the function is called and result is diplayed.





**Project Tips**

The detailed manual of Racket can be found at:

<http://docs.racket-lang.org/reference/index.html>

You don’t need read through the whole reference, just find information you need such as how to define/invoke functions, how to manipulate lists, and how to use conditional statements.

For these two functions, using recursive solution is properly easier. Make sure you cover all base cases (for example, function arguments are lists, empty lists).

Some useful functions are provided below with a brief description:

* **(list? listvalue )**

returns true (#t) if listvalue is a list value; otherwise false (#f)

* **(null? listvalue)**

returns true if listvalue is an empty list, otherwise false

* **(equal? value1 value2)**

returns true if value1 is equal to value2, otherwise false

* **(> value1 value2)**

returns true if value1 greater than value2, otherwise false. (similar for >)

* **(- value1 value2)**

returns the result of value1 – value2 (similar for +)

* **(length listvalue)**

returns the number of elements in the listvalue.

* **(list-tail listvalue pos)**

returns the list after the first pos elements of listvalue.

* **(drop-right listvalue pos)**

returns a fresh list whose elements are the prefix of listvalue, dropping its pos-length tail.

* **(list-ref listvalue pos)**

Returns the element of listvalue at position pos, where the list’s first element is position 0.

* **(not boolvalue)**

Returns true if boolvalue is false; otherwise false.

For detailed information on list operations, please refer to the page below:

<http://docs.racket-lang.org/reference/pairs.html#%28part._.List_.Operations%29>

**How to submit?**

* Copy the rubric4.doc and your source program to the project4 folder in your individual repository. Edit the file to put your name.
* Commit the whole project2 folder to the server. Make sure the folder contains your source program.
* **Any commit of the project after the deadline is considered as cheating. If this happens, the latest version before the deadline will be graded, and you may receive up to 50 points deduction.**
* **No hard copy needed**