CSE1729 - Introduction to Programming | January 24, 2017

Lab Assignment 0

# Objectives

1. Become familiar with Racket, a Scheme interpreter;
2. Be able to set the Scheme mode and know the default mode we will use for the course;
3. Be able to enter expressions in Racket and understand their evaluation;
4. Practice with function abstractions
5. Be able to submit your work via the Moodle site for CSE1729 (http://courses.engr.uconn.edu).

# Background

## Racket

Scheme is the programming language we will be using for almost all programming in CSE1729. Scheme programs are "executed" by an interpreter and constructed in a "dialog" with the interpreter. Racket is an implementation of an interpreter for Scheme and the one you will be using for the course. All lab assignments and problem sets should be completed and tested using Racket before submission. Racket is available on all workstations in School of Engineering, Learning Center computer labs. If you would like to work on your own machine, you may download Racket (for free) for your operating system at http://racket-lang.org/download/.

## Starting Racket

Once installed properly, you should be able to start Racket by finding and double clicking the 'DrRacket' icon. On MS Windows machines you are likely to find it by clicking "All Programs"-> Applications->Racket->"Dr. Racket." On MacOS machines, you should click Applications -> Racket v6.6 -> DrRacket. If you prefer a command-line interface, you can also start Racket by typing 'racket' in a Terminal window on Mac machines, or command prompt on Windows machines. You should see something similar to:

```
Welcome to DrRacket, version 6.6 [3m].
Language: racket [custom]; memory limit: 8192 MB.
>
```

Racket runs a *read-interpret-print* loop. That is, it continuously repeats a cycle of reading a Scheme expression, evaluating that expression and printing the result of evaluating that expression. Right now it is just waiting for an expression to evaluate. Since Racket supports several dialects of Scheme, we can indicate the particular dialect we want the interpreter to emulate. For most of the semester, we will follow the R5RS standard which we can indicate opening the Language -> Choose Language menu item and selecting the "Other Languages" radio button and clicking on R5RS in the dialog box and clicking on the "OK" button. Go ahead and do that now. You should use this mode unless otherwise indicated for the rest of the course.

### Expressions

You will be learning more about expressions and combining expressions in lecture. For now, let's start with the simplest type of expression, a number. Go ahead and type any number that comes to mind, perhaps you have a favorite, into the upper half of the Racket window. This is the Definitions panel. Then click Run at the top of the screen. I like the numeral 9, so I'll type '99' and the interpreter responds with: `99`. So, a number evaluates to the value it represents.

A slightly more complex expression might include a mathematical operator and some operands. But, before we do that, we should note that Scheme uses *prefix notation*. So, our expression should start with the operator and be followed by all of the operands. You can add this to your Definitions window and click Run again. All expressions in the Definitions window are evaluated again and the results displayed in the bottom, Interactions, window. So if I type '+ 33 66' I might expect the interpreter to respond with `99` but instead we get:

```
#<procedure:+>
> 33
> 66
```

The interpreter evaluates each operator or operand individually as if each is a separate expression. Not quite what we had intended. To show these more complex expressions are composed of the operator and two or more operands (yes, you can include more than two) they are enclosed in parentheses and are considered a list. So, if I were to type '(+ 33 66)' in the interpreter, it would print `99` as we would expect. Next, satisfy your curiosity by entering an expression with three operands (e.g. '(+ 33 33 33)') and satisfy yourself that the operator is applied as you would expect.

In addition, we can use the result of evaluating an expression as an operand for another, larger, expression. For instance, type this expression (which can be found in your book) into the Scheme interpreter:

```
(+ (* 3 (+ (* 2 4) (+ 3 5))) (+ (- 10 7) 6))
```

and take a moment or two to work out the result so that you can "play computer" and demonstrate to yourself that you understand how the interpreter is evaluating this expression. Now, this expression as it is shown here is fairly difficult to sort out. That is, it is difficult to determine which expressions belong to which operators. In order to help us humans visualize how an expression is composed, and therefore evaluated, we can use *pretty printing*. To use pretty printing, we line up operands vertically by inserting tabs where appropriate as shown below and in your textbook. Type the expression into the Scheme interpreter using pretty printing appropriately. Be sure to line up the operands associated with the same operator in the same column.

```
(+ (* 3
      (+ (* 2 4)
         (+ 3 5)))
   (+ (- 10 7)
      6))
```

**Note: You must use pretty printing in all of your submissions for lab assignments and problem sets.**

### Submitting Assignments

Once your work in the Definitions window has been written, tested and debugged, you can save your solution by selecting File -> Save Definitions and giving your solution file a name. You may then submit your solution for credit on Moodle. You can load your work by selecting File -> Open from the Racket menu bar and selecting the definitions file you've previously saved. **All assignments for this course must be submitted via Moodle. No assignments will be accepted via email.**

# Try This!

Practice your newly developed Scheme powers with the following expressions:

1. Use the Scheme interpreter to compute the product $34,234,567 \times 23,123,456$.
2. Use the Scheme interpreter to compute $45,321,123^3$.

# Lab Assignment Activities

1. Currency Conversion
   (a) The exchange rate between U.S. dollars and euros is $1\$ = 0.93\,€$. Write a Scheme procedure to convert dollars into euros. How many euros will you get when you exchange $50?

   **Solution:**
   ```
   (define (dollars-to-euros d)
     (* d 0.93))
   (dollars-to-euros 50)
   ```

   (b) The exchange rate between euros and Japanese yen is $1\,€ = 121/73\,¥$. Write a Scheme procedure to convert euros into yen. How many yen will you get when you exchange €50?

   **Solution:**
   ```
   (define (euros-to-yen e)
     (* e 121.73))
   (euros-to-yen 50)
   ```

   (c) Write a procedure using your solutions from parts (a) and (b) to convert U.S. dollars to yen. How many yen can you receive in exchange for $50?

   **Solution:**
   ```
   (define (dollars-to-yen d)
     (euros-to-yen (dollars-to-euros d)))
   (dollars-to-yen 50)
   ```

2. A *matrix* is a rectangular grid of numbers organized into rows and columns. Matrices are an important tool in algebra and are often used to solve systems of linear equations. Below are examples of a couple of $2 \times 2$ matrices (matrices with 2 rows and 2 columns) that we will call $M$ and $N$.

$$M = \begin{pmatrix} 2 & -4 \\ -6 & 12 \end{pmatrix} \qquad N = \begin{pmatrix} -3 & 1 \\ 2 & 7 \end{pmatrix}$$

   (a) A special value associated with any $2 \times 2$ matrix is the *determinant*. Given a generic $2 \times 2$ matrix, the determinant can be computed using the following formula:

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

   Using the formula, we can compute the determinant of matrix $M$ above as $(2)(12) - (-4)(-6) = 0$. Write a Scheme procedure to compute the determinant of a generic $2 \times 2$ matrix. Assume that the matrix elements $a$. $b$, $c$ and $d$ are given as four formal parameters. Compute the determinant of $N$.

**Solution:**

```
(define (det a b c d)
  (- (* a d)
     (* b c)))

(det -3 1 2 7)
```

(b) A matrix is called *invertible* if its determinant is non-zero. Write a procedure that checks whether or not a generic $2 \times 2$ matrix is invertible. Verify that $N$ is invertible and $M$ is not invertible.

**Solution:**

```
(define (is-invertible? a b c d)
     (not (= (det a b c d) 0)))

(is-invertible? 2 -4 -6 12)
(is-invertible? -3 1 2 7)
```

(c) The determinant of a $3 \times 3$ matrix can be computed from the following formula:

$$\det\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = a \times \det\begin{pmatrix} e & f \\ h & i \end{pmatrix} - b \times \det\begin{pmatrix} d & f \\ g & i \end{pmatrix} + c \times \det\begin{pmatrix} d & e \\ g & h \end{pmatrix}$$

Fill in the body of the following procedure to find the determinant of a $3 \times 3$ matrix:

```
(define (det3x3 a b c
                d e f
                g h i)

  ; your code here

)
```

What is the determinant of $\begin{pmatrix} 0 & 5 & -6 \\ 8 & -11 & 4 \\ 5 & 1 & 1 \end{pmatrix}$?

**Solution:**

```
(define (det3x3 a b c
                d e f
                g h i)
  (+ (* a (det e f h i))
     (* (- b) (det d f g i))
     (* c (det d e g h))))

(det3x3 0 5 -6 8 -11 4 5 1 1)
```

3. Save your work to a file using an appropriate filename (using the following convention: *lastname-firstname*-CSE1729-Lab0.rkt) and preserving the .rkt file extension.

4. Read the Honor Code Pledge on the CSE1729 Moodle site completely.

5. Hand in your signed Honor Code Pledge in class in lab today.

6. Submit your lab solution file for grading via the Moodle site.

   Please note: assignments will not be graded for credit until your pledge is filed.