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. Be able to submit your work via the MIMIR site for CSE1729

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 -> 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 7.7 [3m].
Language: R5RS; memory limit: 2048 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. Racket provides help in pretty printing in both the lower (interaction) and upper (definition) panes. Type (or copy-paste) the one-line version of the expression into the definition pane. Now add line

breaks by placing the cursor where you want the new line to start and hitting enter - notice how Racket lines things up. If you edit the indentation might get out of line, at which point you can indent an individual line relative to its predecessor using the tab key, or indent the whole definition pane by using the "reindent all" command, found under the Racket menu. SCHEME does not care if you indent or not, but properly-indented code is easier to read, and easier to understand.

Using Mimir and Submitting Assignments

All SCHEME assignments in this course will be submitted using Mimir - you should be able to load Mimir automatically through the assignment link on the Moodle site for CSE1729. Once your work in the Definitions pane has been written and saved as the appropriate name (lab0.rkt) in the case of this lab, you may submit it by hitting the Submit button and choosing your file. Saving the definitions pane is done by selecting File -> Save Definitions and giving your solution file the correct name. You can reload your work by selecting File -> Open from the Racket menu bar and selecting the definitions file you've previously saved, at which point you can do more editing and testing. All assignments for this course must be submitted via Mimir. No assignments will be accepted via email. A nice feature of Mimir is that when you submit your file, it will run tests on your work (in future labs and problem sets). This is used in grading your work, but also provides you with immediate feedback about what works and what doesn't.

Making it easy to read your results

We will be using a specific labeling convention with Racket definition files to make it easier for you and your instructors to verify that your code works. This will involve the use of strings. A string in SCHEME is a sequence of characters within quotation marks. We will use strings later for various things, but for now it is sufficient to know that a string evaluates to a string that looks like itself. So if you put a string in a Racket definition pane, then hit Run, that string will show up in the lower window as the result of its evaluation. We will use this as follows: before the code from a problem, we will identify which problem it is, and for each test case we will put in a string identifying the test case. Here is a simple example: suppose question 3 asks you to write a square function in SCHEME. What you put in the Racket definitions will be something like the following:

```
"problem 3a."
(define (square x)
  (* x x))
3
"tests"
"(square 4)"
(square 4)
"(square -5)"
(square -5)
"(square (square 3))"
(square (square 3))
```

What you (and we) will see in the lower window when you hit the Run button is:

```
"problem 3a."
"tests"
"(square 4)"
16
"(square -5)"
25
"(square (square 3))"
81
>
```

Now it is easy to see what your test results are. Your definitions should be in the same order as in the assignment, with tests within each question.

Lab Assignment Activities

1. This is a sample question so you can practice submitting your solutions on Mimir. Don't worry if it doesn't make total sense. We haven't covered function definition in SCHEME in lecure. But, we need a testable function to verify our code is correct and see the results of the test cases in Mimir. Type the following code into the Racket Definitions window (top frame).

```
"Problem 1"
(define (square x)
  (* x x))
"tests"
"(square 0)"
(square 0)
"(square -2)"
(square -2)
"(square 4)"
(square 4)
```

Next, press the "Run" button at the top of the Racket window. You should see the results of evaluating the strings and applications of the square function in the Racket Interactions (lower) window.

```
"Problem 1"
"tests"
"(square 0)"
0
"(square -2)"
4
"(square 4)"
16
>
```

Once you have finished:

1. Save your work (the definitions) to a file named lab0.rkt

2. Submit your lab solutions for grading via Mimir. [You may paste just the function definition in the proper box on the Mimir page for Lab 0.

3. If you haven't already, read the Honor Code Pledge on the CSE1729 Moodle site completely.

4. If you haven't already, upload your signed Honor Code Pledge in lab today.

Please note: assignments will not be graded for credit until your Honor Code Agreement is filed-see it under Course Content in Moodle.