Collaboration policy in effect for this assignment: SEE last page before starting the assignment.

CSCI 320, Fall 2011, Homework 1

DUE: Thursday, September 8, 3pm

1. Before starting DrRacket Read through the following code and predict what the interpreter will print in response to evaluation of each of the following expressions. Assume that the sequence is evaluated in the order in which it is presented here. After jotting down all predictions, check your answers by typing the statements into Dr. Racket –record the results.

	Prediction	Actual
`(- 8 9)`		
`(- 7 2 3)` *(ops can take more than 2 operands)*		
`(/ 8 3)` *(real or int result?)*		
`(> 3.7 4.4)`		
`(> 7  3  11)`		
`(- (if (> 3 4) 7 10) (/ 16 10))`		
`(define b 13)`		
`13`		
`b`		
`(define (square x) (* x x))`		
`(square 13)`		
`(square b)`		
`(square (square (/ b 1.3)))`		
`(define a b)`		
`(= a b)`		

`(if (= (* b a) (square 13))`
`  (< a b)`
`  (- a b))`		

`(cond ((>= a 2) b)`
`  ((< (square b) (square a)) (/ 1 0))`
`  (else (abs (- (square a) b))))`		

`(* 5 (- 2 (/ 4 2) (/ 8 3)))`		
`(* 5 (- 2 (/ 4 2)) (/ 8 3))`		

What do the last two examples tell you in terms of indenting code so that you can "see" the logic?

Collaboration policy in effect for this assignment: SEE last page before starting the assignment.

Creating a file in Dr. Racket Since the Dr. Racket Interaction buffer (bottom portion of the environment window) will chronologically list all the expressions you evaluate, and since you will generally have to try more than one version of the same procedure as part of the coding and debugging process, it is usually better to keep your procedure definitions in a separate editing buffer (at the top, sometimes call the definitions window), rather than to work only in the interaction buffer. You can save this other buffer in a file so you can split your work over more than one session.

The basic idea is that you type your programs into the editor buffer, and then use the **Check Syntax** button to see if they have correct syntax. The **Check Syntax** button will also highlight your code with colors indicating which words are function names (by virtue of their position), which are variable names, etc. It will also create a *variable flow overlay* that shows where variable values come from in a function, and where they are going to. This happens whenever you pass your mouse over the variable name. You don't have to click. NOTE: the feedback that you get depends on what you have the language level set to.

After **Check Syntax**, you would then click on **Run**. This loads the editing buffer into the Scheme Interaction buffer. You can then start running your program by typing into the Interaction buffer.

To practice these ideas, go to the empty editing buffer. In this buffer, create a definition for a simple procedure, by typing in the following (verbatim):

```
;; computer square of a number
(define (square x) (*x x))
```

Now, click on **Check Syntax.**
If you actually typed in the definition *exactly* as printed above, you should see a bright red Check Syntax Error Message saying "*x" (with no space) has a problem. This is not a *syntax* error, but DrRacket is warning you that "*x" is currently undefined (it's a perfectly valid identifier name). Let's see what happens when we try to run this code. Click on **Run** to load this incorrect definition into the Racket Interpreter buffer. You will see that you get a red error message here too. Ignore it and type "`(square 4)`" and return. The interpreter prints "`reference to an undefined identifier: square`". At this point we see that the definition of square did not work because of the problem with `*x`.

Edit the definition to insert a space between **`*`** and **`x`**. Re-Syntax-Check and Re-run the definition. Try evaluating `(square 4)` again. Save this working definition along with calls to it to show that it works, in a file called **`hw1-answers.rkt`**. Each call should be preceded by a comment indicating what the correct result of the call would be.

You can put all of your later answers in this file to print out or submit.

Printing a File FYI, You can print either the definitions buffer or the interactions buffer, under the FILE menu.

Stepped Evaluation: Using The Debugger Load the file `code0.rkt` it into Racket. This simply contains a few definitions and some function calls. After doing a Check-Syntax, press the Debug button. The Debugger will automatically process definitions up until the first function call, then wait for you to "step" through the program. Press the **Step** button and watch the window and the stack change as the program is executed or more definitions are processed. Make sure that you understand exactly what is happening at each point in the evaluation!!

Another Debugging example While you work, you will often need to debug programs.
Load the code for this problem set from code1.rkt into the editor buffer. This will load definitions of the following three procedures `p1`, `p2`, and `p3`:

Collaboration policy in effect for this assignment: SEE last page before starting the assignment.

```
(define (p1 x y)
 (+ (p2 x y) (p3 x y)))
(define (p2 z w)(* z w))
```

```
(define (p3 a b)
  (+ (p2 a) (p2 b)))
```

Click on **Check Syntax**. Check out the flow of information display---move your cursor over, for example, the various occurrences of `p2` in the code. Click on **Run** to load the code into the interpreter.
In the interaction buffer, evaluate the expression `(p1 1 2)`. This should signal an error, with the message:
`Procedure p2: expects 2 arguments, given 1: 1`
You can fix the error and re-execute the code. Note: if you fix the code, and forget to click **Run** and try to re-run `(p1 1 2)` in the interpreter window, you'll see that DrRacket warns you that you've changed the code but haven't re-loaded it by printing a yellow-and black warning message "WARNING: The definitions window has changed. Click Run." Place a fixed version of the code and a call to it in your **`hw1-answers.rkt`** file. Make sure that there is a Racket comment right before the function definition and each call to it.

Paper exercise:
Suppose + were not defined, but the more primitive procedures INC and DEC were. (INC n) returns the integer one greater than n, and (DEC n) returns the number one less than n. The following are two possible definitions of +, written using INC and DEC and using + recursively.

```
; two different definitions for '+' add two numbers
(define (+ a b)
 (if (= a 0)
 b
 (inc (+ (dec a) b))))

(define (+ a b)
 (if (= a 0)
 b
 (+ (dec a) (inc b))))
```

Assignment: Write the steps of the evolution of the evaluation of (+ 4 5) for each of the <u>two</u> definitions of +. DO THIS ON A SEPARATE SHEET OF PAPER.

For example, the steps of the first version of + for (+ 3 6) are

(+ 3 6)
(inc (+ 2 6))
(inc (inc (+ 1 6)))
(inc (inc (inc (+ 0 6))))
(inc (inc (inc 6)))
(inc (inc 7))
(inc 8)
9
Note: this is a pencil and paper exercise, no online computing required.

Coding exercises - using recursion

Any positive number n can be written as a sequence of digits $d_k\ d_{k-1}\ ...d_1\ d_0$, where d_k is non-zero. Thus the number n is the sum of d_i times 10 to the i power, for $i = 0$ to k. For example, if $n = 456$, then k is 2 and d_2 is 4, d_1 is 5, and d_0 is 6. A Racket program can access the individual digits by using the following basic functions:

```
;; Returns the units digit of the integer n
(define (units-digit n)
  (remainder n 10))

;; Returns all but the units digit of the integer n
```

Collaboration policy in effect for this assignment: SEE last page before starting the assignment.

```
(define (all-but-units-digit n)
  (quotient n 10))
```

With those definitions in place, for example,

```
(unit-digit 456) --> 6 (all-but-units-digit 456) --> 45
```

By combining these functions, you can access the rightmost (units) digit, the second digit, the third digit, etc. ultimately reaching the most significant (leftmost) digit. If `(all-but-the-units-digit n)` is zero, you know n is a one-digit number.

Using these access functions, define the following functions:

1. `(decimal-length n)` returns $k+1$, where d_k is the leading digit of n. For example, `(decimal-length 348567) --> 6`
2. `(ith-digit n i)` returns d_i of n. For example `(ith-digit 671835629 3) --> 5`
3. `(leading-digit n)` returns d_k, the most significant digit. `(leading-digit 534) --> 5`
4. `(occurances d n)` returns the number of times digit d occurs in n. `(occurrances 6 549457654875498758765465547856 3) --> 4`
5. `(digit-sum n)` returns the sum of the digits in n. `(digit-sum 2354) --> 14, (digit-sum 8) --> 8`
6. `(digital-root n)` returns the result of repeatedly applying digit-sum until the result is a single digit. `(digital-root 341943) --> 6` (via 3+4+1+9+4+3=24, `then 2+4=6)`
7. `(backwards n)` returns the digits in reverse order. You do not need to handle leading or trailing zeros. For this problem you may use multiplication. For example: `(backwards 329145) --> 541923`

Add the definition for these routines along with calls that show that they work to **`hw1-answers.rkt`**. Do not forget to provide appropriate documentation of each procedure – minimally, a one-liner explaining what problem is being solved. Hint: Each of these functions should be fairly short and simple. If they get long, re-examine the problem!

Include your name on all of your work and in all of your files.

Submit `hw1-answers.rkt` via OAKS by the due date/time.
Submit a completed page 1 and separate sheet for the Paper Exercise (page 3) to Dr. McCauley or Vanathi (or you may scan and upload these as well) by due date/time – see below.
Due date/time: Thursday, September 8, 3pm

WARNING: Solutions (and the next assignment) will be posted immediately after the due date/time, so late assignments will not be accepted.

EARLY submissions are always accepted.

Collaboration policy in effect for this assignment: SEE last page before starting the assignment.

- You may only submit solutions generated by your own work.
- You may discuss DrRacket with classmates and instructors.
- You may discuss Racket with classmates and instructors.
- You may discuss these problems with classmates and instructors. If it helps you to learn from others how to solve the problem, that is fine. The objective is that you CAN and DO solve the problem.
- You MAY NOT copy the work of others.
- You MAY NOT give your work to others.
- You should not look for solutions on the internet, you should craft your own.