Harvey Mudd College
Computer Science 60
Fall 2010

Assignment 4
The Unicalc Programming Language
Due. 11:59 p.m., Wed., 29 September 2010
Pair-programming is optional on this assignment

This assignment builds on the Unicalc experience from the previous assignment. We want to extend the simple CLI to become a full programming language. To do this, we will include in it Racket-like programming constructs. It's not that we expect all of our users to become programmers, but we would like to have that option. Along the way, we learn how to implement constructs that occur in real languages.

In class, you were shown how to implement lambda expressions using closures and how to apply named user-defined functions. It is fair to use that code, with attribution course. Here are the constructs we want in our language for this assignment.

Name	Example	Definition
>, =, < operators	`(> (* 1.7 km) mile)`	Comparison operators, return a non-Quantity value #t, #f, or error (if compared values are incompatible).
lambda expressions	`(lambda ($x $y) (/ $y $x))`	An anonymous function with argument list and body expression. Imports are allowed.

let expressions	`(let (($x 5)`
	`($y meter))`
	`(* $x $y))`
		Evaluate an expression with bindings of variables.

let* expressions	`(let* (($x meter)`
	`($y (* $x kg))`
	`(/ $y second))`
		Cascaded evaluation of an expression with bindings of variables.

if expressions	`(if (> (* 1.7 km) mile)`
	`(* 5 second)`
	`(* 10 second))`
		Conditional expressions. Only one of the second or third argument expressions of the if is evaluated.

simple **recursive** definitions	`(define $mfac`
	`(lambda($x)`
	`(if (< $x 1)`
	`1`
	`(* $x`
	`($mfac (/ $x 2))))))`
		The current environment is used for bindings, in the event that there is no binding that would shadow it.