CS 565 — Out: February 5, 1998
Jens Palsberg — Due: February 12, 1998

Homework 3:

Implement Beta-reduction

Here is the grammar for a subset of Scheme:

Exp	::=	*VariableReference*
	|	`(lambda (` *Variable* `)` *Exp* `)`
	|	`(` *Exp Exp* `)`
	|	*Constant*
	|	`(` *UnaryOp Exp* `)`
	|	`(` *BinaryOp Exp Exp* `)`
	|	`(if` *Exp Exp Exp* `)`
Constant	::=	`#t` | `#f` | *IntegerNumeral*
UnaryOp	::=	`zero?`
BinaryOp	::=	`+` | `-`

1. Write in Scheme a procedure `parse` and a procedure `unparse` in the style of the procedures in *EOPL* p.85. Use the file `recscm.scm` on the CS 565 homepage to provide definitions of `define-record`, and other record operations. See Lecture Note 1 for an example of how to use such records.

2. Write in Scheme a procedure `substitute` that takes two expressions, e and m, and a variable, x, and returns $e[m/x]$. See Lecture Note 3, p.22.

3. Write in Scheme a procedure `execute-cbv` which takes a parsed expression in the above language and repeatedly does one step of call-by-value (applicative-order) computation, as long possible, possibly entering an infinite loop. If the procedure terminates, then it prints the resulting expression. Notice that the operations on numbers and Booleans give rise to a kind of redexes, the so-called δ-redexes, for example, `(+ 4 5)`, and corresponding δ-reduction rules, for example, `(+ 4 5)` $\rightarrow$ `9`. The program should apply both β-reduction and δ-reduction until no more call-by-value reductions are possible.