Programming Languages (CS 329) **Lab 2**
Computer Science and Engineering **Out: 2000-10-08**
Indian Institute of Technology Bombay **Due: 2000-10-30**

1. Here is the grammar for FLK:

L	`::=`	`#u`	[UnitLiteral]
	|	`#t` | `#f`	[BooleanLiteral]
	|	... | `-2` | `-1` | `0` | `1` | `2` | ...	[IntLiteral]
P	`::=`	`+` | `-` | `*` | `/` | `=`	[ArithOps]
	|	`and` | `or` | `not` |	[BooleanOps]
	|	...	[PrimitiveOps]
E	`::=`	L	[Literal]
	|	I	[Identifier]
	|	`(`P E^*`)`	[ApplyPrimitiveOp]
	|	`(if` E_1 E_2 E_3`)`	[Branch]
	|	`(pair` E_1 E_2`)` | `(left` E`)` | `(right` E`)`	[Pair]
	|	`(proc I` E`)` | `(call` E_1 E_2`)`	[Procedures]
	|	`(rec I` E`)`	[Recursion]

   Write a denotational interpreter for FLK using Scheme. Use Scheme's support of booleans and integers; do not use lambda calculus for these. Make provisions for printing out the environment at each call to $\mathcal{E}$.

2. Write a recursive function to find the product of integers given in a list in Scheme, using Scheme's regular `define` mechanism for recursion. Now introduce a bail-out if the number zero is found in the input list, using the `callcc` construct. Insert print statements to see how control is transfered in the event of zero being actually present in the input list. Turn in a version of code which can return out of any depth of recursion in one shot.

3. Implement a producer-consumer coroutine pair using `callcc` in Scheme. The producer will increment a counter and the consumer will print out successive values.