CS 565 Out: October 2, 1997
Jens Palsberg Due: October 9, 1997

Homework 6:

Implement CPS-conversion

1. Convert the following program to tail form, then to first-order form, then to imperative form. Submit all three versions of the program.

```
(define filter
  (lambda (p lst)
    (if (null? lst)
        '()
        (if (p (car lst))
            (cons (car lst) (filter p (cdr lst)))
            (filter p (cdr lst))))))
```

2. Convert the following program to tail form, then to first-order form, then to imperative form. Submit all three versions of the program.

```
(define product-aux
  (lambda (s los)
    (if (null? los)
        '()
        (cons (list s (car los))
              (product-aux s (cdr los))))))

(define product
  (lambda (los1 los2)
    (if (null? los1)
        '()
        (append (product-aux (car los1) los2)
                (product (cdr los1) los2)))))
```

```
(define append
  (lambda (l m)
    (if (null? l)
        m
        (cons (car l) (append (cdr l) m)))))
```

3. Here is the grammar for a subset of Scheme.

Program	::=	*FunctionDef*$_1$ ... *FunctionDef*$_n$
FunctionDef	::=	`(define` *FunctionName*
		`(lambda (` *Variable* ... *Variable* `)` *Exp* `))`
Exp	::=	*VariableReference*
	|	`(` *Exp Exp* ... *Exp* `)`
	|	*Constant*
	|	`(` *UnaryOp Exp* `)`
	|	`(` *BinaryOp Exp Exp* `)`
	|	`(if` *Exp Exp Exp* `)`
Constant	::=	`#t` | `#f` | *IntegerNumeral* | `'()`
UnaryOp	::=	`zero?` | `null?` | `car` | `cdr`
BinaryOp	::=	`+` | `-` | `cons`

Let us call this subset BabyScheme. Use JTB and JavaCC and write one or more visitors which from a MiniScheme program generate a syntax tree in the style of Lecture Note 5, Section 9. Write in Scheme a program which converts BabyScheme programs to tail form. As part of the submission, include the results of running your solution on the programs in questions 1 and 2.