1. **Notation notation notation**

   One annoying thing about Scheme is that it can only understand arithmetic operations that are written in prefix notation. That is, if I want to evaluate an expression, the arithmetic operator must come first, which really goes against everything you were taught as a child. Let's leverage our interpreter skills to define a Scheme procedure that accepts arithmetic operations with infix notation, which places operators between operands as you're used to. You only need to support the addition and multiplication operators `*` and `+`, but you need to support order of operations. Define the `interpret` procedure so that it passes the test cases below.

```
scm> (interpret '(1))
1
scm> (interpret '(1 + 2))
3
scm> (interpret '(1 * 2))
2
; Order of operations apply
scm> (interpret '(3 + 2 * 5 + 4))
17
scm> (interpret '(5 * 3 + 2 + 4 * 9))
53
; Parentheses should be handled properly
scm> (interpret '(3 * (2 * 4)))
24
scm> (interpret '(3 + (2 + 4)))
9
scm> (interpret '((3 + 2) + 4)
9
; Parentheses are prioritized higher than order of operations
scm> (interpret '(1 + 2 * (3 + 4)))
15
scm> (interpret '(1 + 2 * (3 + 4 * (5 + 6))))
95

; Some helper procedures (optional)
(define (caar x) (car (car x)))
(define (cadr x) (car (cdr x)))
(define (cddr x) (cdr (cdr x)))

(define (interpret expr)
(cond
  (_____________________________________________________________________________ expr)

  ((null? (cdr expr)) (if _____________________________________________________________________________))

  ((_____________________________________) (interpret _____________________________________________________))

  ((_____________________________________) (interpret _____________________________________________________))

  ((_____________________________________) (+ _____________________________________________________________))))
```

2. *Don't forget to check your quiz answers, which are on the last page of discussion solutions that are posted at the end of each week.*