```
(-1 - -2) * -(3 / 5)
```

Method *evaluate*(**In:** *pExpr* as an infix expression) **Returns** *Double*
    Create *operatorStack* -- Stores *Operators*
    Create *operandStack* -- Stores *Operands*
    While end of *pExpr* has not been reached **Do**
        Scan next *token* in *pExpr* -- The type of *token* is *Token*
        If *token* is an operand **Then**
            Convert *token* to *Operand* object named *number*
            operandStack.push(*number*)
        ElseIf *token* is an **InstanceOf** *LeftParen* **Then**
            Convert *token* to *LeftParen* object named *paren*
            operatorStack.push(*paren*)
        ElseIf *token* is an **InstanceOf** *RightParen* **Then**
            While not *operatorStack.peek*() is an **InstanceOf** *LeftParen* **Do** *topEval*()
            operatorStack.pop() -- Pops the *LeftParen*
        ElseIf *token* is *Negation*, *Addition*, *Subtraction*, *Multiplication*, or *Division* **Then**
            Convert *token* to *Operator* object named *operator*
            While *keepEvaluating*() returns **True Do** *topEval*()
            operatorStack.push(*op*)
    End While
    While not *operatorStack.isEmpty*() **Do** *topEval*()
    Return *operandStack.pop*()
End Method *evaluate*

Method *keepEvaluating*() **Returns True** or **False**
    If *operatorStack.isEmpty*() **Then Return False**
    Else Return *stackPrecedence*(*operatorStack.peek*()) **≥** *precedence*(*operator*)
End Method *keepEvaluating*

Method *topEval*() **Returns Nothing**
    right ← *operandStack.pop*()
    operator ← *operatorStack.pop*()
    If *operator* is *Negation* **Then** *operandStack.push*(-*right*)
    Else *left* ← *operandStack.pop*()
        If *operator* is *Addition* **Then** ***o****perandStack.push*(*left* + *right*)
        ElseIf *operator* is *Subtraction* **Then** *operandStack.push*(*left* - *right*)
        ElseIf *operator* is *Multiplication* **Then** *operandStack.push*(*left* * *right*)
        Else *operandStack.push*(*left* / *right*)
    End If
End Method *topEval*

Method *precedence*(**In:** *Operator pOperator*) **Returns Int**
    If *pOperator* is *LeftParen* **Then Return** 5
    ElseIf *pOperator* is *Negation* **Then Return** 4
    ElseIf *pOperator* is *Multiplication* or *Division* **Then Return** 3
    ElseIf *pOperator* is *Addition* or *Subtraction* **Then Return** 2
    Else Return 1
End Method *precedence*

Method *stackPrecedence*(**In:** *Operator pOperator*) **Returns Int**

If *pOperator* is *LeftParen* **Then Return** 0

ElseIf *pOperator* is *Negation* **Then Return** 4

ElseIf *pOperator* is *Multiplication* or *Division* **Then Return** 3

ElseIf *pOperator* is *Addition* or *Subtraction* **Then Return** 2

Else Return 1

End Method *stackPrecedence*