Brendan Van Allen
Lab 3

CaC 4.7

a) Leftmost derivation

```
Start

E $

T plus E $

F plus E $

num plus E $

num plus T plus E $

num plus T times F plus E $

num plus F times F plus E $

num plus num times F plus E $

num plus num times num plus E $

num plus num times num plus T $

num plus num times num plus F $

num plus num times num plus num $
```

b) Rightmost derivation

```
Start

E $

T plus E $

T plus T $

T plus T times F $

T plus T times num $

T plus F times num $

T plus num times num $

T times F plus num times num $

T times num plus num times num $

F times num plus num times num $

num times num plus num times num $
```

(c)

This grammar appears to structure expressions with same precedence and left/right associativity of the (+) and (*) operators as our normal mathematics. The operator ‘times’ has a higher precedence than ‘plus’ because if the non-terminal F in production 4 is not a num, it must be contained within parentheses. This means in the expression (with nums numbered for convenience):

```
num1 times num2 plus num3
```

num2 will belong to the ‘times’ operator, since there are no parentheses. Right-to-left associativity of ‘plus’ is evident in the fact that in production 2, non-terminal E appears on the right side of the expression, and E is the only non-terminal that can produce a ‘plus’ operator. Left-to-right associativity of ‘minus’ can be determined in the same fashion, with non-terminal T being the only one able to produce a ‘minus’ operator, and it appears on the left side of the expression in production 4.

CaC 5.2c (pseudo code only)

```
parse()
  parseStart()

parseStart()
  parse(Value)
  match($)

parseValue()
  if(currentToken is num)
    matchAndConsume(num)
  else
    matchAndConsume(lparen)
    parseExpr()
    matchAndConsume(rparen)

parseExpr()
  if(currentToken is plus)
    matchAndConsume(plus)
    parseValue()
    parseValue()
  else
    matchAndConsume(prod)
    parseValues()

parseValues()
  if(currentToken in (num, lparen))
    parseValue()
    parseValues()
  else
    // Lambda production

matchAndConsume(expectedTokens)
  if(currentToken in expectedTokens)
    addLeafNode(currentToken)
  else
    error

resetParent()
  if(currentToken != null && currentToken.parent != null)
    currentToken = currentToken.parent
```

```
Tree()
  this.root = null
  this.cur = [];

  addBranchNode(node)
    Node n = new Node(node)
    if root == null
      this.root = n
    else
      assignParent(n)
      assignChild(n)
    currentNode = n
  addLeafNode(node)
    Node n = new Node(node)
    if root == null
      this.root = n
    else
      assignParent(n)
      assignChild(n)

  Node()
    name
    parent
    children
```

Dragon 4.2.1 a, b, and c

a) Leftmost derivation

5
S S *
S S + S *
a S + S *
a a + S *
a a + a *

b) Rightmost derivation

5
S S *
S a *
S S + a *
S a + a *
a a + a *

c) Parse Tree