Scott Hansen

Lab 3:

Crafting a compiler:
Do exercises 4.7 and 5.2c (write pseudo code only for 5.2c)

7. A grammar for infix expressions follows:

```
1  Start → E  $
2  E     → T  plus  E
3        |  T
4  T     → T  times  F
5        |  F
6  F     → (  E  )
7        |  num
```

(a) Show the leftmost derivation of the following string.

num plus num times num plus num $

(b) Show the rightmost derivation of the following string.

num times num plus num times num $

(c) Describe how this grammar structures expressions, in terms of the precedence and left- or right- associativity of operators.

4.7:
a:

| **Rule** | **Start** | **End** |
|---|---|---|
| 1 | Start | E $ |
| 2 | E $ | T plus E $ |
| 5 | T plus E $ | F plus E $ |
| 7 | F plus E $ | num plus E $ |
| 2 | num plus E $ | num plus T plus E $ |
| 4 | num plus T plus E $ | num plus T times F plus E $ |
| 5 | num plus T times F plus E $ | num plus F times F plus E $ |
| 7 | num plus F times F plus E $ | num plus num times F plus E $ |

| | | |
|---|---|---|
| 7 | num plus num times F plus E $ | num plus num times num plus E $ |
| 3 | num plus num times num plus E $ | num plus num times num plus T $ |
| 5 | num plus num times num plus T $ | num plus num times num plus F $ |
| 7 | num plus num times num plus T $ | num plus num times num plus num $ |

b:

| **Rule** | **Start** | **End** |
|---|---|---|
| 1 | Start | E $ |
| 2 | E $ | T plus E $ |
| 4 | T plus E $ | T plus T times F $ |
| 7 | T plus T times F $ | T plus T times num $ |
| 5 | T plus T times num $ | T plus F times num $ |
| 7 | T plus F times num $ | T plus num times num $ |
| 4 | T plus num times num $ | T times F plus num times num $ |
| 7 | T times F plus num times num $ | T times num plus num times num $ |
| 5 | T times num plus num times num $ | F times num plus num times num $ |
| 7 | F times num plus num times num $ | num times num plus num times num $ |

C: This grammar gives precedence to addition as plus symbols are farther up in the tree and require less derivation. Multiplication is given more precedence than parenthesis, and instructions inside parenthesis execute in the precedence referenced above.

?Instructions are expanded to the right so it is a right-most associative grammar

2. Consider the following grammar, which is already suitable for LL(1) parsing:

```
1  Start  → Value $
2  Value  → num
3         | lparen Expr rparen
4  Expr   → plus Value Value
5         | prod Values
6  Values → Value Values
7         | λ
```

(c) Construct a recursive-descent parser based on the grammar.

5.2c:

```
parseStart()
{
  addBranchNode(start);
  parseValue();
  parseExpr();
  parseValues();
}

parseValue()
{
  addBranchNode(value);
  matchAndConsume(num, ( Expr ));
  kick();
}

parseExpr()
{
  addBranchNode(expr);
  matchAndConsume(+ Value Value, * Values)
  kick();
}

parseValues()
{
  addBranchNode(values);
  matchAndConsume(value Values, ‘’)
  kick();
}
```

```
matchAndConsume(params)
{
  if checkParams()
  {
    addLeadNode(params);
  }
}
```

Dragon:
Do exercises 4.2.1 a, b, and c

### 4.2.8 Exercises for Section 4.2

**Exercise 4.2.1 :** Consider the context-free grammar:

$$S \rightarrow S\,S\,+ \mid S\,S\,* \mid a$$

and the string $aa + a*$.

a) Give a leftmost derivation for the string.

b) Give a rightmost derivation for the string.

c) Give a parse tree for the string.

a: leftmost derivation

| **Rule** | **Start** | **End** |
|---|---|---|
| 2 | S | SS* |
| 1 | SS* | SS+S* |
| - | SS+S* | aS+S* |
| - | aS+S* | aa+S* |
| - | aa+S* | aa+a* |

b: rightmost derivation

| **Rule** | **Start** | **End** |
|---|---|---|

| | | |
|---|---|---|
| 2 | S | SS* |
| - | SS* | Sa* |
| 1 | Sa* | SS+a* |
| - | SS+a* | Sa+a* |
| - | Sa+a* | aa+a* |

c: parse tree