Gabriel Arnell

**Lab 3**

**Crafting a Compiler**

**4.7 (Derivations)**

A. **Leftmost derivation**

Input: num plus num times num plus num $

S. E $
2. T plus E $
4. F plus E $
7. num plus E $
2. num plus T $
4. num plus T times F $
5. num plus F times F $
7. num plus num times F $
6. num plus num times E $
2. num plus num times T plus F $
4. num plus num times F plus F $
7. num plus num times num plus F $
7. num plus num times num plus num $

B. **Rightmost derivation**

Input: num times num plus num times num $

S. E $
2. T $
4. T times F $
6. T times E $
2. T times T plus E $
3. T times T plus T $
4. T times T plus T times F $
7. T times T plus T times num $
5. T times T plus F times num $
7. T times T plus num times num $
5. T times F plus num times num $
7. T times num plus num times num $
5. F times num plus num times num $
6. 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.**

This grammar is interesting because unlike the Grammar we are coding our compiler with, the operator associativity of this actually does not matter much. At first it may seem like it is left most because all of the operator productions start with T and end with F, and T also produces F, but production 6 allows F to produce an E (which in turn produces a T) which means that the left-or-right associativity actually does not matter much in this case.

**5.2c (recursive descent parser - pseudo code only)**

Construct a recursive-descent partser based on the grammar

```
parseStart(){
parseValue()
match( $ )
}

parseValue(){
If (nextToken is num){
match(num)
}
Else{
match(lparen)
match(Expr)
match(rparen)
}
}

parseExpr(){
If (nextToken is plus){
match( plus )
parseValue()
parseValue()
}else{
match( prod )
parseValues()
}
}

parseValues(){
If (nextToken is num or lparen){
parseValue()
parseValues()
}else{
// empty epsilon
}
}
```

**Dragon**

**4.2.1 a,b, and c (derivations and a parse tree)**

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

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

Input: aa+a*

a. **Leftmost Derivation**

S
SS*
SS+S*
aS+S*
aa+S*
aa+a*

b. **Rightmost Derivation**

S
SS*
Sa*
SS+a*
Sa+a*
aa+a*

c. **Parse Tree, of left most**