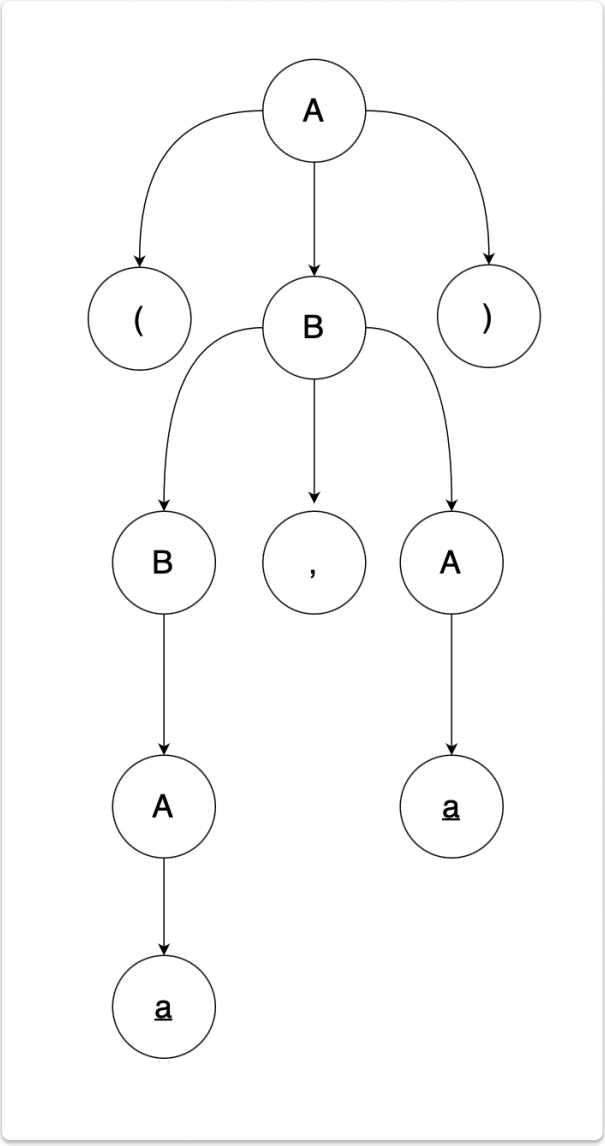
Homework 4

Reagan McFarland
NETID - rpm141
March 28th, 2020

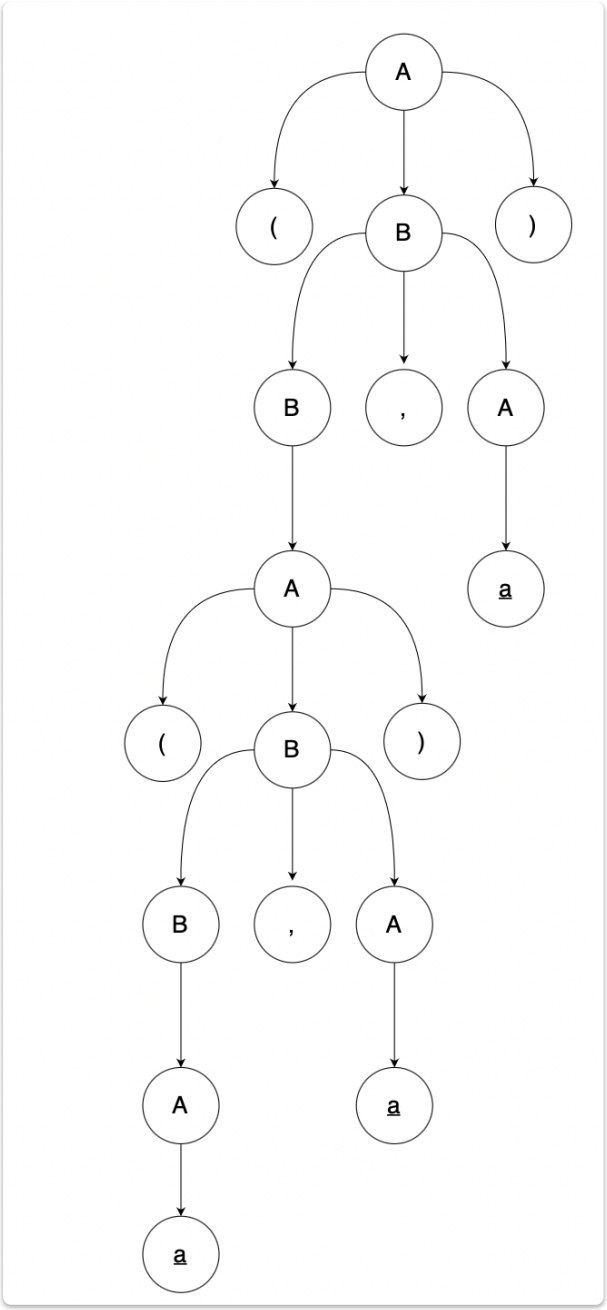
Problem 1

1.1) Give parse trees for the sentences `(a,a)` and `((a,a),a)`

`(a,a)`:



`((a,a),a)`:



1.2) Construct a leftmost and a rightmost derivation for the sentence `((a,a),a)`

Leftmost:

```
A -> (B) -> (B,A) -> ((B),A) -> ((B,A),A) -> ((A,A),A) -> ((a,A),A) -> ((a,a),A) -> ((a,a),a)
```

Rightmost:

```
A -> (B) -> (B,A) -> (B,a) -> -> ((B),a) -> ((B,A),a) -> ((B,a),a) -> ((A,a),a) -> ((a,a),a)
```

Problem 2

2.1) Compute the ***FIRST*** and ***FOLLOW*** sets for the grammar

- FIRST(ICONST) = {1,2,3,4,5}
- FIRST(ID) = {a, b, c}

- FIRST(Expr) = {+, -, *} ∪ FIRST(ICONST) = {+, -, *, 1, 2, 3, 4, 5}
- FIRST(Print) = {!}
- FIRST(Assign} = FIRST(ID) = {a, b, c}
- FIRST(Stmt) = FIRST(Assign) ∪ FIRST(Print) = {!, a, b, c}
- FIRST(NextStmt) = {;, ε}
- FIRST(Stmtlist) = FIRST(Stmt) = {!, a, b, c}
- FIRST(Program) = FIRST(Stmtlist) = {!, a, b, c}
- FOLLOW(Program) = { eof }
- FOLLOW(Stmtlist) = { . }
- FOLLOW(NextStmt) = FOLLOW(Stmtlist) = { . }
- FOLLOW(Stmt) = FIRST(NextStmt) = { ; }
- FOLLOW(Assign) = FOLLOW(Stmt) = { ; }
- FOLLOW(Print) = FOLLOW(Stmt) = { ; }
- FOLLOW(Expr) = FIRST(Expr) ∪ FOLLOW(Assign) = {+, -, *, 1, 2, 3, 4, 5, ;}
- FOLLOW(ICONST) = ⌀
- FOLLOW(ID) = ⌀

2.2) Compute the LL(1) parse table for the resulting grammar. Is the grammar LL(1) or not? Justify your answer

- FIRST+(Program) = {!, a, b, c }
- FIRST+(StmtList) = {!, a, b, c }
- FIRST+(NextStmt) = {;} ∪ FOLLOW(NextStmt) = { ;, . }
- FIRST+(Stmt) = { !, a, b, c }
- FIRST+(Assign) = {a, b, c}
- FIRST+(Print) = {!}
- FIRST+(Expr) = {+, -, *, 1, 2, 3, 4, 5}
- FIRST+(ID) = {a, b, c}
- FIRST+(ICONST) = {1, 2, 3, 4, 5}

Rule	**.**	**;**	**!**	**=**	**+**	**-**	*****	**a**	**b**	**c**	**1**
Program			Stmtlist .					Stmtlist .	Stmtlist .	Stmtlist .	
StmtList			Stmt NextStmt					Stmt NextStmt	Stmt NextStmt	Stmt NextStmt	
NextStmt	**ε**	; Stmtlist									
Stmt			Print					Assign	Assign	Assign	
Assign								ID = Expr	ID = Expr	ID = Expr	
Print			! ID								
Expr					+ Expr Expr	- Expr Expr	* Expr Expr				ICONST
ID								a	b	c	
ICONST											1

The grammar **IS LL(1)** because there are no entries defined multiple times in the same column. You can always derive the grammar by only having to look ahead one token at a time.

2.3) If the resulting grammar is LL(1), show the behavior of the LL(1) skeleton table-driven parser as a sequence of states [stack content, remaining input, next action to be taken] on sentence `c=3;!c.`

```
([eof, Program], c=3;!c., Stmtlist .) =>
([eof, ., Stmtlist], c=3;!c., Stmt NextStmt) =>
([eof, ., NextStmt, Stmt], c=3;!c., Assign) =>
([eof, ., NextStmt, Assign], c=3;!c., ID = Expr) =>
([eof, ., NextStmt, Expr, =, ID], c=3;!c., next input + pop) =>
([eof, ., NextStmt, Expr, =], =3;!c., next input + pop) =>
([eof, ., NextStmt, Expr], 3;!c., ICONST) =>
([eof, ., NextStmt, ICONST], 3;!c., next input + pop) =>
([eof, ., NextStmt], ;!c., ; StmtList) =>
([eof, ., StmtList, ;], ;!c., next input + pop) =>
([eof, ., StmtList], !c., Stmt NextStmt) =>
([eof, ., NextStmt, Stmt], !c., Print) =>
([eof, ., NextStmt, Print], !c., ! ID) =>
([eof, ., NextStmt, ID, !], !c., next input + pop) =>
([eof, ., NextStmt, ID], c., next input + pop) =>
([eof, ., NextStmt], ., pop) =>
([eof, .], ., next input + pop) =>
([eof], accept)
```

3.1) Write an interpreter for the language above

Attached with my submission. The only implementation specific thing in this program is that `ICONST()` and `Expr()` return the values of the number / expression on success, and -1 on failed parse. This makes it much easier to recursively deal with evaluations.

3.2) Write a compiler for the langauge above

Attached with my submission. The only implementation specific thing in this program is that `ICONST()` and `Expr()` return the register number that the constant was loaded into / the register number the expr was loaded into. This makes it much easier to recursively deal with nested expressions.

Also, I am using `r0, 4`, `r0, 8`, and `r0, 12` for the memory locations of variables `a`, `b`, and `c`, respectively.