Programming Languages
Assignment 1

Q1: Using the BNF grammar below show a **parse tree** and a **leftmost derivation** for each of the following 4 statements:

```
1)   A = ( A + B ) * C
2)   A = B + C + A
3)   A = A * (B + C)
4)   A = B + (C * (A * B))
```

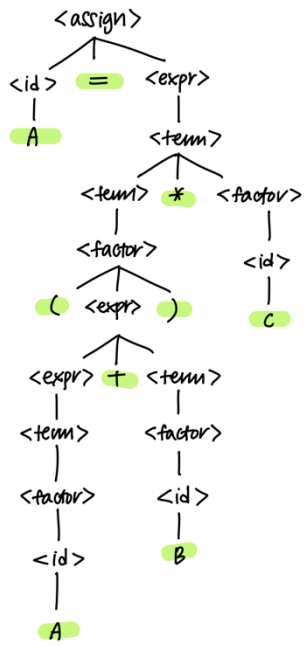
BNF grammar
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <expr> + <term> | <term>
<term> → <term> * <factor> | <factor>
<factor> → (<expr>) | <id>

1) A = (A + B) * C

Leftmost derivation:

<assign>
→ <id> = <expr>
→ A = <expr>
→ A = <term>
→ A = <term> * <factor>
→ A = <factor> * <factor>
→ A = (<expr>) * <factor>
→ A = (<expr> + <term>) * <factor>
→ A = (<term> + <term>) * <factor>
→ A = (<factor> + <term>) * <factor>
→ A = (<id> + <term>) * <factor>
→ A = (A + <term>) * <factor>
→ A = (A + <factor>) * <factor>
→ A = (A + <id>) * <factor>
→ A = (A + B) * <factor>
→ A = (A + B) * <id>
→ A = (A + B) * C

Parse Tree:



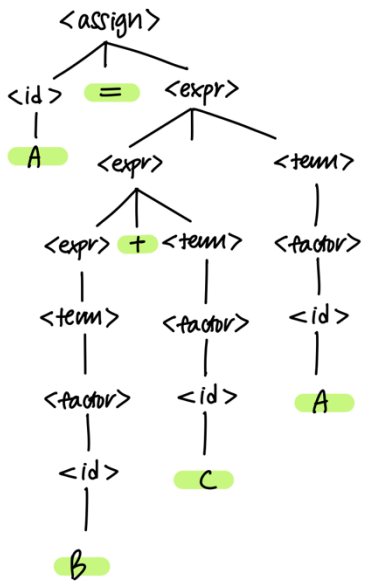
Programming Languages
Assignment 1

2) A = B + C + A

Leftmost derivation:

<assign>
→ <id> = <expr>
→ A = <expr>
→ A = <expr> + <term>
→ A = <expr> + <term> + <term>
→ A = <term> + <term> + <term>
→ A = <factor> + <term> + <term>
→ A = <id> + <term> + <term>
→ A = B + <factor> + <term>
→ A = B + <id> + <term>
→ A = B + C + <term>
→ A = B + C + <factor>
→ A = B + C + <id>
→ A = B + C + A

Parse Tree:

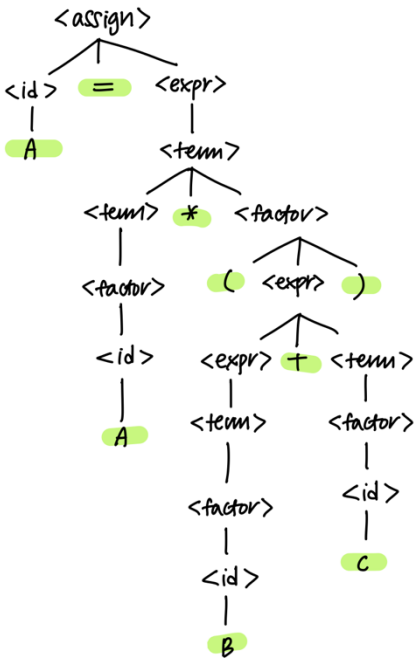


3) A = A * (B + C)

Leftmost derivation:

<assign>
→ <id> = <expr>
→ A = <expr>
→ A = <term>
→ A = <term> * <factor>
→ A = <factor> * <factor>
→ A = <id> * <factor>
→ A = A * (<expr>)
→ A = A * (<expr> + <term>)
→ A = A * (<term> + <term>)
→ A = A * (<factor> + <term>)
→ A = A * (<id> + <term>)
→ A = A * (B + <term>)
→ A = A * (B + <factor>)
→ A = A * (B + <id>)
→ A = A * (B + C)

Parse Tree:

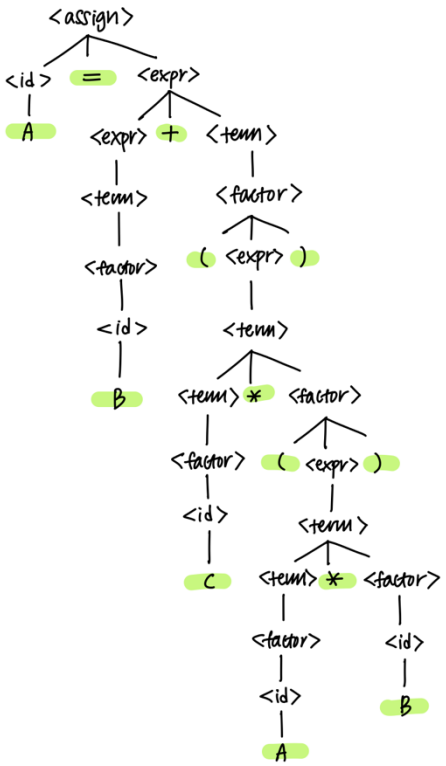


4) A = B + (C * (A * B))

Leftmost derivation:

<assign>
→ <id> = <expr>
→ A = <expr>
→ A = <expr> + <term>
→ A = <term> + <term>
→ A = <factor> + <term>
→ A = <id> + <term>
→ A = B + <term>
→ A = B + <factor>
→ A = B + (<expr>)
→ A = B + (<term>)
→ A = B + (<term> * <factor>)
→ A = B + (<factor> * <factor>)
→ A = B + (<id> * <factor>)
→ A = B + (C * <factor>)
→ A = B + (C * (<expr>))
→ A = B + (C * (<term>))
→ A = B + (C * (<term> * <factor>))
→ A = B + (C * (<factor> * <factor>))
→ A = B + (C * (<id> * <factor>))
→ A = B + (C * (<A> * <factor>))
→ A = B + (C * (<A> * <id>))
→ A = B + (C * (A * B))

Parse Tree:

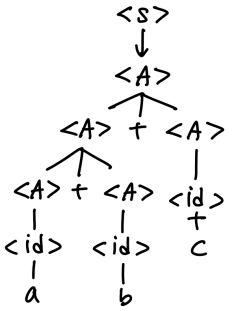


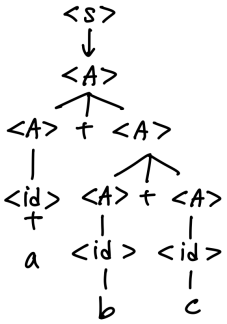
Q2: Prove that the following grammar is ambiguous:

BNF grammar
<S> → <A>
<A> → <A> + <A> | <id>
<id> → a | b | c

The BNF grammar above is ambiguous because the same expression can be interpreted in multiple ways.

For instances, the expression "a + b + c" can lead to two different parsing trees:





Having multiple parse trees for the same expression indicated that the BNF grammar is ambiguous, as the same expression does not have a single definitive way of being evaluated.

Q3: Modify the grammar below to add a unary minus — operator and ^ power operator that have higher precedence than either + or *.
Thus, the precedence of the operators in the final BNF should be ranked from the **highest** to the **lowest** as follows: ( ) , — , ^ , * , +
Also, all the operators have left associativity except the power operator ^ and unary minus — , which have right associativity.

BNF grammar
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <expr> + <term> | <term>
<term> → <term> * <factor> | <factor>
<factor> → (<expr>) | <id>

Modified grammar:

<assign> → <id> = <expr>

<id> → A | B | C

<expr> → <expr> + <term> | <term>

<term> → <term> * <power> | <power>

<power> → <unary> ^ <power> | <unary>

<unary> → - <unary> | <factor>

<factor> → (<expr>) | <id>