Exercise 1.

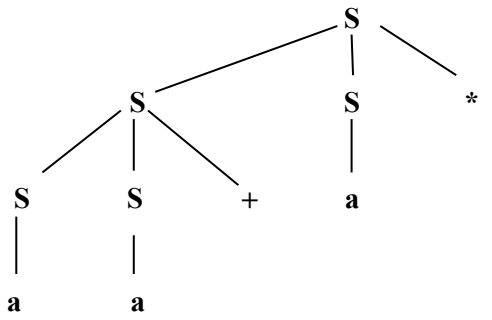
- Consider the following context-free grammar:

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

a) **Leftmost derivation** of the string **aa+a * :**
S ⇒ SS* ⇒ SS+S* ⇒ aS+S * ⇒ aa+S * ⇒ aa+a *

b) **Rightmost derivation** of the string **aa+a *:**
S ⇒ SS* ⇒ Sa* ⇒ SS+a * ⇒ Sa+a * ⇒ aa+a *

c) **Parse tree** (derivation tree) of the string **aa+a * :**



d) **The language generated by the grammar:**
The language generated by this grammar describes postfix (reverse Polish) arithmetic expressions.

Exercise 2.

- The language generated by the following grammars:
  a) **S →+SS | *SS | a** :

  The language generated by this grammar describes **prefix (Polish) arithmetic expressions**.

  b) **S →S(S)S** | ε:

  The language generated by this grammar describes **well-parenthesized (balanced-parentheses) expressions**.

  c) **S → x | y | z | S+S | SS | S*S | S/S | (S):**

  The language generated by this grammar describes **arithmetic expressions over the three variables x,y,z** with correct parentheses (i.e., properly formed/parenthesized expressions).

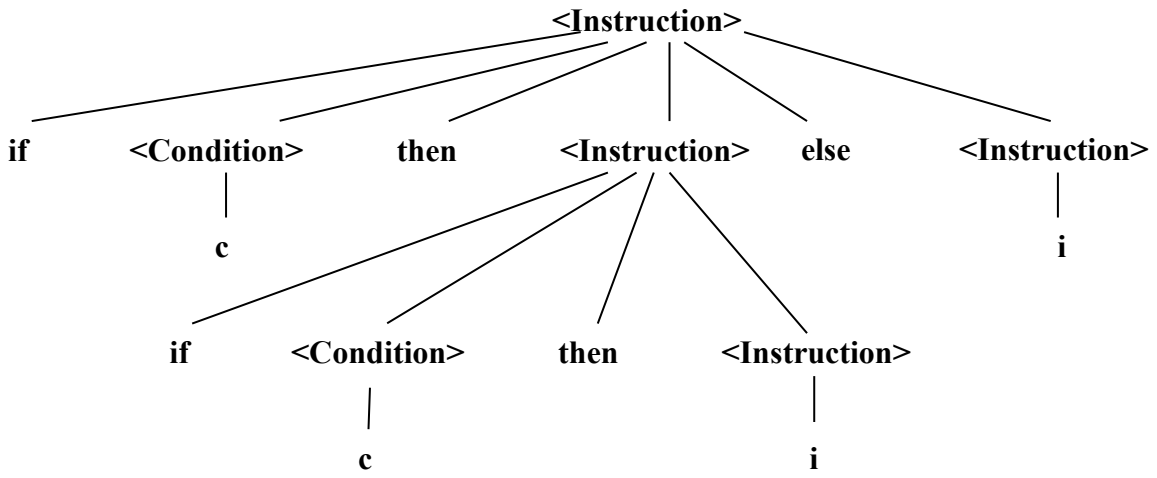
Exercise 3.

- Consider the grammar G=< N,T ,P,S> whose productions are given below:

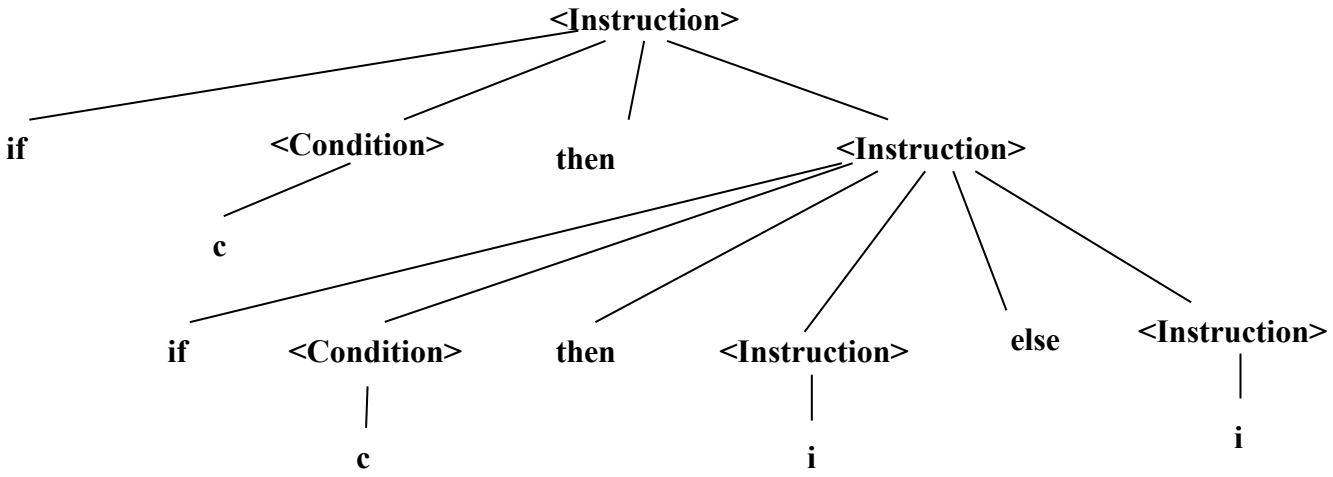
  P:
  <Instruction> → if <Condition> then <Instruction> else <Instruction>
  | if <Condition> then <Statement>

| begin <LI> end
| i
<Condition> → c
<LI> → <LI > ; i | i

a) Show that the above grammar G is **ambiguous** by providing two different parse trees for the following program fragment: **if c then if c then i else i**



parse tree 1



parse tree 2

The grammar GGG is **ambiguous** because it has two different parse trees for the following program fragment: **if c then if c then i else i**

b) It is possible to **nest** if statements. To eliminate the ambiguity, in the absence of properly placed parentheses, languages such as **C** always associate each else with the **closest unmatched if.**

Exercise 4.

- Let the grammar G= <{ A ,B,C},{ a,b },P,A> be :

  A →aB

  B →bC

C →aC | bC | ε

a) **Leftmost derivation** of the string : **abbbaab**
   A ⇒ aB ⇒ abC ⇒ abbC ⇒ abbbC ⇒ abbbaC ⇒ abbbaaC ⇒ abbbaabC ⇒ abbbaab
b) **Rightmost derivation** of the string : **abbbaab**
   A ⇒ aB ⇒ abC ⇒ abbC ⇒ abbbC ⇒ abbbaC ⇒ abbbaaC ⇒ abbbaabC ⇒ abbbaab
c) Language generated by the grammar :
   L(G) : **ab(a| b)***