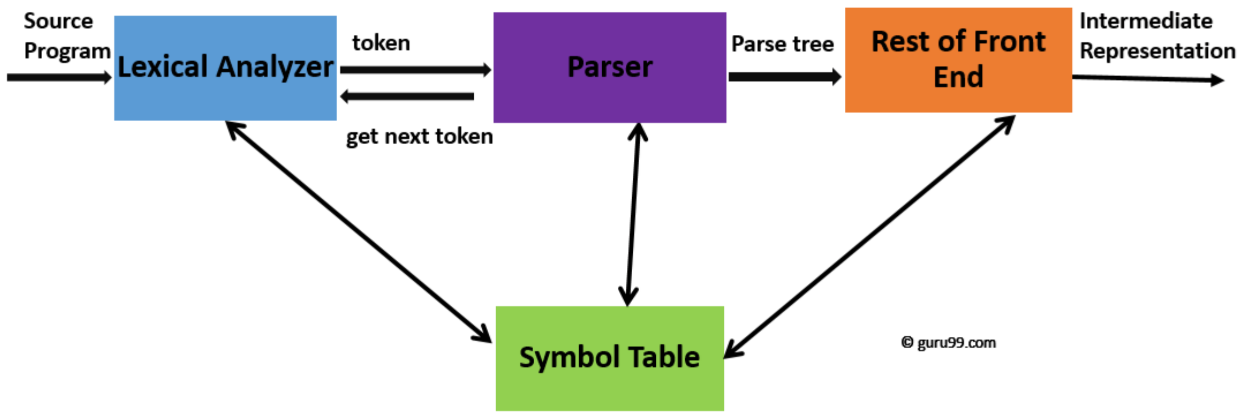
Introduction to Syntax Analysis/Parsing in Compiler Design

<u>Syntax:</u>

Syntax of a language refers to the structure of valid programs or statements of that language. CFG or ‘grammar’ specify the syntax of a language.

<u>Parsing/Syntax analysis:</u>

1. It is the second phase of compilation process
2. Parsing is the process of deriving string from a given grammar.
3. It determines if a string of token is and generate parse tree. It takes input as the stream of tokens from lexical analyzer and verify if that string of token is a valid sequence whether its structure is syntactically correct of not.

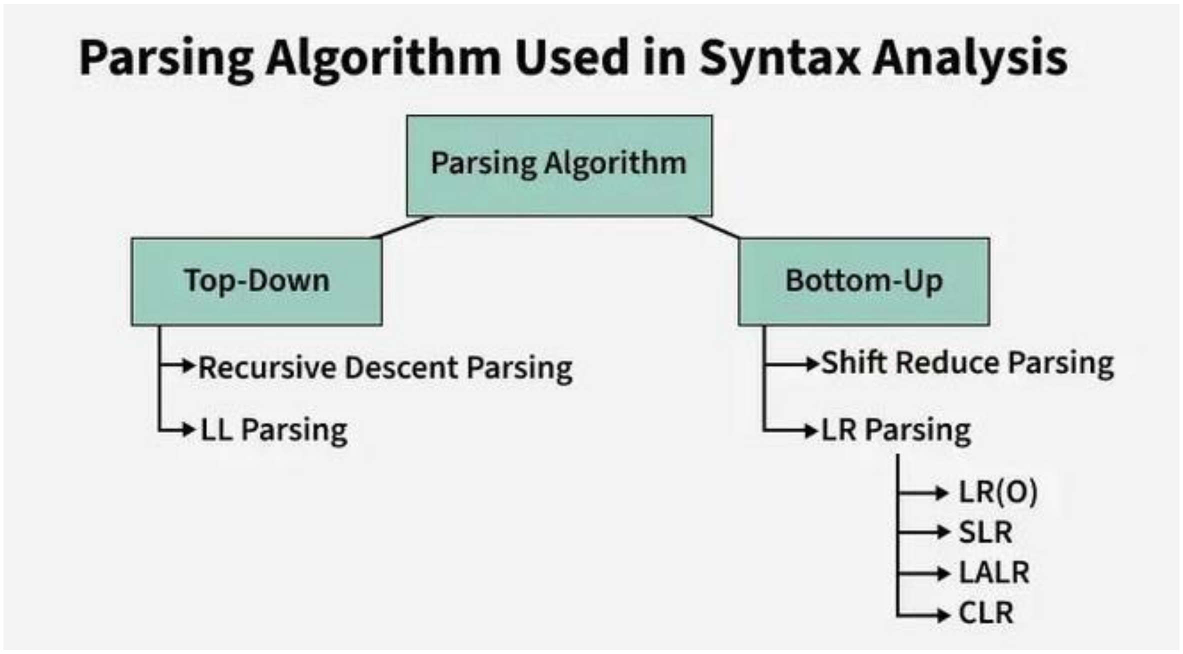


4. Syntax tree is the compressed form of parse tree
5. Syntax analysis reports all syntax error.
6. It is a two-step process: -
    a. Top-down parser
    b. Bottom-up parser

✓ Importance of Syntax Analysis

1. It is used to check if the code is grammatically correct or not.
2. It helps us to detect all types of syntax errors.
3. It gives an exact description of the error.
4. It rejects invalid code before actual compiling.

Introduction to Syntax Analysis/Parsing in Compiler Design

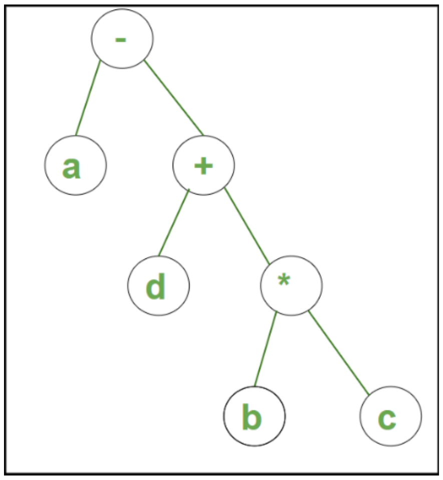


✓ Rules to Draw a Parse Tree

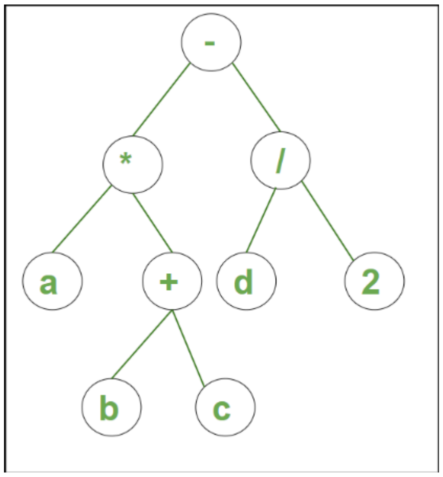
- All leaf nodes need to be terminals.
- All interior nodes need to be non-terminals.
- In-order traversal gives the original input string.

✓ Generate a **Syntax Tree/Parse Tree** based on the expression:

Example 1: Syntax Tree for the string a - b ∗ c + d

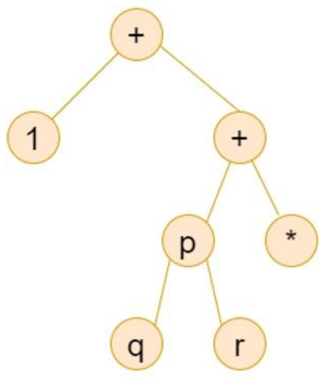


Example 2: Syntax Tree for the string a * (b + c) – d /2

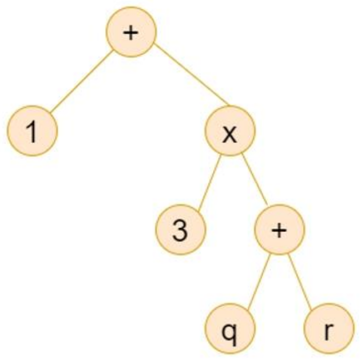


Introduction to Syntax Analysis/Parsing in Compiler Design

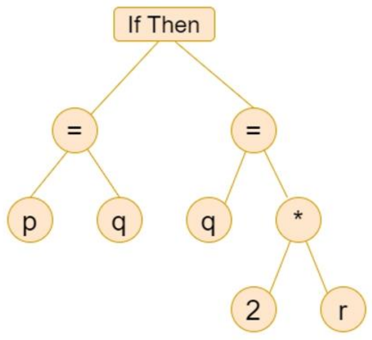
Example 3:- Draw the syntax tree for 1 + (3*4) + 1



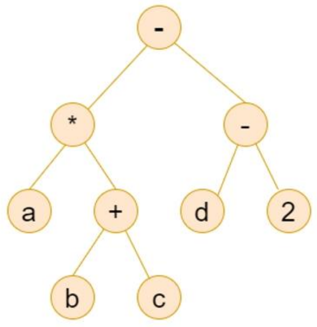
Example 4:- Draw the syntax tree for 1 + 3*(4+1)



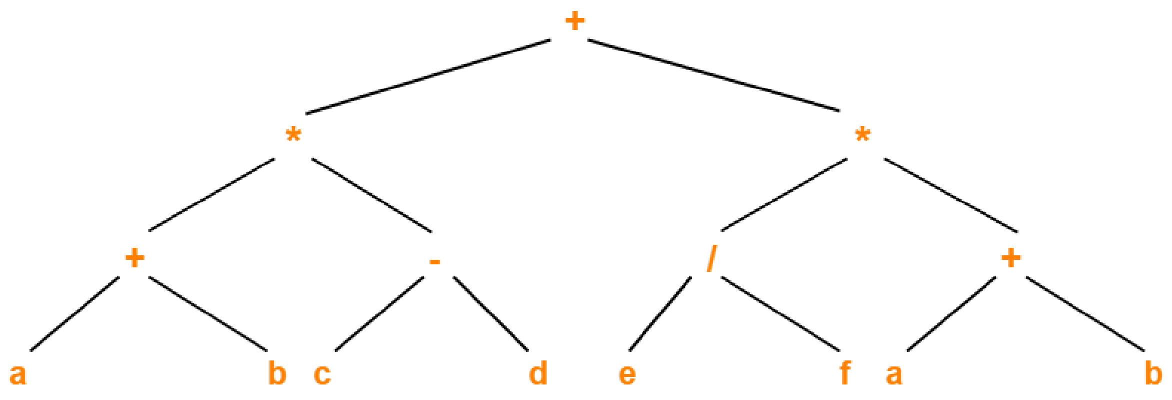
Example 5:- Construct a syntax tree for a statement.
If p = q then q = 2 * r



Example 6:- Draw syntax tree for following arithmetic expression
a * (b + c) – d /2.



Example 7:- Construct a syntax tree for the following arithmetic expression-
(a + b) * (c – d) + ((e / f) * (a + b))



Syntax Tree

Introduction to Syntax Analysis/Parsing in Compiler Design

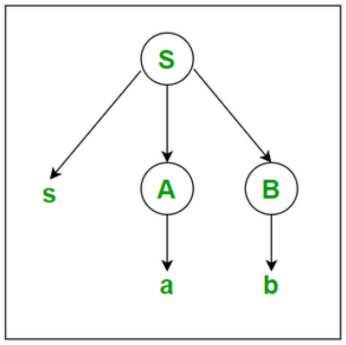
Example 8: Let us take an example of Grammar (Production Rules).

S -> sAB

A -> a

B -> b

The input string is "sab", then the Parse Tree is:



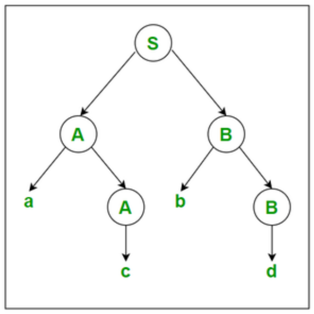
Example-9: Let us take another example of Grammar (Production Rules).

S -> AB

A -> c/aA

B -> d/bB

The input string is "acbd", then the Parse Tree is as follows:



Problem-10: Considering the following grammar-

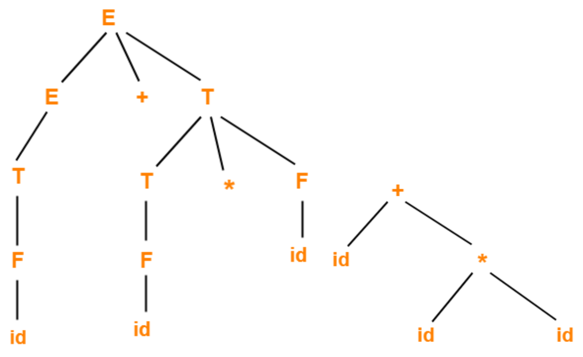
E → E + T | T

T → T x F | F

F → (E) | id

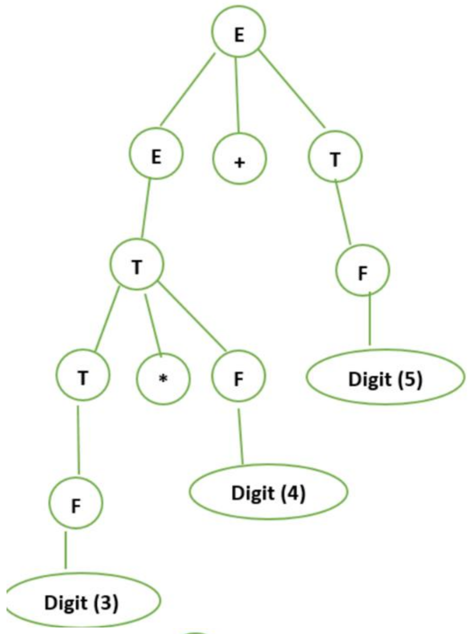
Generate the following for the string w1 = id + id x id and w2 = 3*4+5

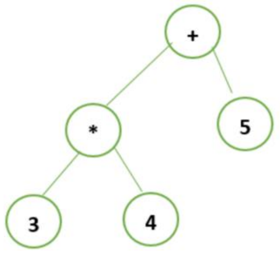
1. Parse tree
2. Syntax tree



Parse Tree

Syntax Tree





Introduction to Syntax Analysis/Parsing in Compiler Design

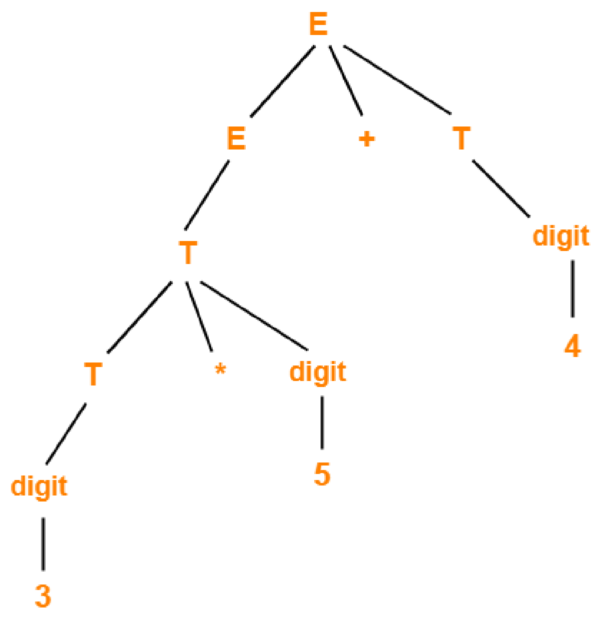
Variants of syntax tree:

A syntax tree basically has two variants which are described below:

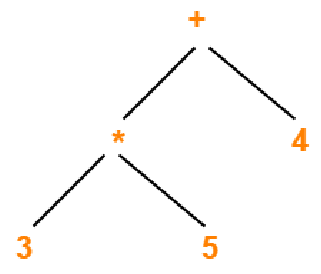
1. Directed Acyclic Graphs for Expressions (DAG)
2. The Value-Number Method for Constructing DAGs

Parse Trees Vs Syntax Trees-

Parse Tree	Syntax Tree
Parse tree is a graphical representation of the replacement process in a derivation.	Syntax tree is the compact form of a parse tree.
Each interior node represents a grammar rule. Each leaf node represents a terminal.	Each interior node represents an operator. Each leaf node represents an operand.
Parse trees provide every characteristic information from the real syntax.	Syntax trees do not provide every characteristic information from the real syntax.
Parse trees are comparatively less dense than syntax trees.	Syntax trees are comparatively denser than parse trees.



Parse Tree



Syntax Tree