SS TUT 7 ANSWERS

1) $S \rightarrow aAb / \in$

$A \rightarrow aAb / \in$

For the given grammar, find out terminals,non terminals and start symbol.

Learning Link:-

Grammar G1 –

({S, A, B}, {a, b}, S, {S → AB, A → a, B → b})

Here,

- **S, A,** and **B** are Non-terminal symbols;
- **a** and **b** are Terminal symbols
- **S** is the Start symbol, S ∈ N
- Productions, **P : S → AB, A → a, B → b**

Example

Grammar G2 –

(({S, A}, {a, b}, S,{S → aAb, aA → aaAb, A → ε })

Here,

- **S** and **A** are Non-terminal symbols.
- **a** and **b** are Terminal symbols.
- **ε** is an empty string.
- **S** is the Start symbol, S ∈ N
- Production **P : S → aAb, aA → aaAb, A → ε**

https://www.tutorialspoint.com/automata_theory/introduction_to_grammars.htm

2) Consider the context-free grammar

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

a) Show how the string aa+a* can be generated by this grammar.

b) Construct a parse tree for this string.

c) What language does this grammar generate? Justify your answer.

3) What language is generated by the following grammars? In each case justify your answer.
a) $S \rightarrow 0\ S\ 1 \mid 0\ 1$
b) $S \rightarrow +\ S\ S \mid -\ S\ S \mid a$
d) $S \rightarrow a\ S\ b\ S \mid b\ S\ a\ S \mid$ epsilon
Also check which of the above grammar is ambiguous.

Answer Link:-

https://github.com/fool2fish/dragon-book-exercise-answers/blob/master/ch02/2.2/2.2.md

4) $E \rightarrow E + T / E - T / T$
$T \rightarrow T\ x\ F / T \div F / F$
$F \rightarrow G \uparrow F / G$
$G \rightarrow I$
$I \rightarrow 0 \mid 1 \mid \ldots \mid 9$

Draw the parse tree for the string:
1) 2 x 1 + 4 ↑ 2 ↑ 1 x 1 + 3
2) 2 ↑ 1 ↑ 4 + 3 x 5 x 6 ↑ 1 + 2 ↑ 3

https://www.gatevidyalay.com/solving-expressions-based-on-given-grammar/

5) Define Input Buffering and explain buffer pair.

Input Buffering:

Buffer Pairs:

Because of the amount of time taken to process characters and the large number of characters that must be processed during the compilation of a large source program, specialized buffering techniques have been developed to reduce the amount of overhead required to process a single input character.

Two pointers to the input are maintained:

Pointer *Lexeme Begin*, marks the beginning of the current lexeme, whose extent we are attempting to determine

Pointer *Forward*, scans ahead until a pattern match is found.

Once the next lexeme is determined, *forward* is set to character at its right end.Then, after the lexeme is recorded as an attribute value of a token returned to the parser, *Lexeme Begin* is set to the character immediately after the lexeme just found