1. Role of lexical analysis and its issues. OR How do the parser and scanner communicate? Explain with the block diagram communication between them.

- The lexical analyzer is the first phase of compiler. Its main task is to read the input characters and produce as output a sequence of tokens that the parser uses for syntax analysis.
- This interaction is given in figure 2.1,

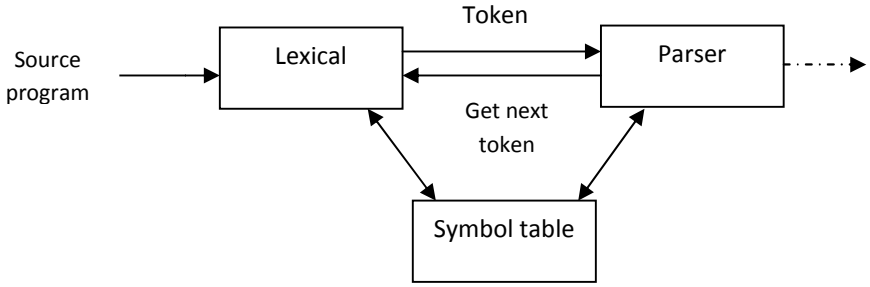

Fig. 2.1 Communication between Scanner & Parser

- It is implemented by making lexical analyzer be a subroutine.
- Upon receiving a "get next token" command from parser, the lexical analyzer reads the input character until it can identify the next token.
- It may also perform secondary task at user interface.
- One such task is stripping out from the source program comments and white space in the form of blanks, tabs, and newline characters.
- Some lexical analyzer are divided into cascade of two phases, the first called scanning and second is "lexical analysis".
- The scanner is responsible for doing simple task while lexical analysis does the more complex task.

Issues in Lexical Analysis:

There are several reasons for separating the analysis phase of compiling into lexical analysis and parsing:

- Simpler design is perhaps the most important consideration. The separation of lexical analysis often allows us to simplify one or other of these phases.
- Compiler efficiency is improved.
- Compiler portability is enhanced.

2. Explain token, pattern and lexemes.

Token: Sequence of character having a collective meaning is known as *token*.

Typical tokens are,

1) Identifiers 2) keywords 3) operators 4) special symbols 5) constants

Pattern: The set of rules called *pattern* associated with a token.

Lexeme: The sequence of character in a source program matched with a pattern for a token is

called lexeme.

Token	Lexeme	Pattern
Const	Const	Const
If	If	If
Relation	<,<=,= ,< >,>=,>	< or <= or = or < > or >= or >
Id	Pi, count, n, I	letter followed by letters and digits.
Number	3.14159, 0, 6.02e23	Any numeric constant
Literal	"Darshan Institute"	Any character between " and " except "

Table 2.1. Examples of Tokens

Example:

total = sum + 12.5

Tokens are: total (id),

= (relation)

Sum (id)

+ (operator)

12.5 (num)

Lexemes are: total, =, sum, +, 12.5

3. What is input buffering? Explain technique of buffer pair. OR Which technique is used for speeding up the lexical analyzer?

There are mainly two techniques for input buffering,

- ✓ Buffer pair
- ✓ Sentinels

1. Buffer pair:

- The lexical analysis scans the input string from left to right one character at a time.
- So, specialized buffering techniques have been developed to reduce the amount of overhead required to process a single input character.
- We use a buffer divided into two N-character halves, as shown in figure 2.2. N is the number of character on one disk block.

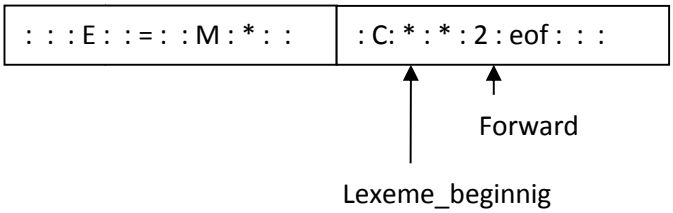

Fig. 2.2 An input buffer in two halves

- We read N input character into each half of the buffer.
- Two pointers to the input are maintained and string between two pointers is the current lexemes.

- Pointer *Lexeme Begin*, marks the beginning of the current lexeme.
- Pointer *Forward,* scans ahead until a pattern match is found.
- If forward pointer is at the end of first buffer half then second is filled with N input character.
- If forward pointer is at the end of second buffer half then first is filled with N input character.

code to advance forward pointer is given below,

```
if forward at end of first half then begin
        reload second half;
        forward := forward + 1;
end
else if forward at end of second half then begin
        reload first half;
        move forward to beginning of first half;
end
else forward := forward + 1;
```

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.

2. Sentinels:

- If we use the scheme of Buffer pairs we must check, each time we move the forward pointer that we have not moved off one of the buffers; if we do, then we must reload the other buffer. Thus, for each character read, we make two tests.
- We can combine the buffer-end test with the test for the current character if we extend each buffer to hold a sentinel character at the end. The sentinel is a special character that cannot be part of the source program, and a natural choice is the character **EOF.**

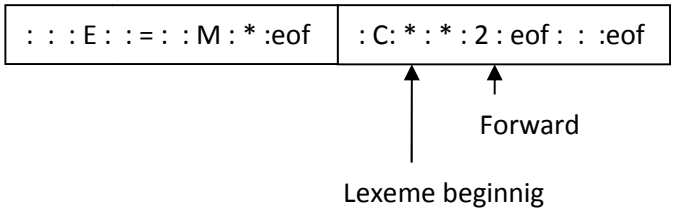

Fig.2.3. Sentinels at end of each buffer half

- Look ahead code with sentinels is given below:

```
forward  := forward + 1;
if forward  = eof then begin
        if forward at end of first half then begin
                reload second half;
                forward  := forward + 1;
        end
        else if forward at the second half then begin
                reload first half;
```

```
                    move forward to beginning of first half;
                end
                else terminate lexical analysis;
            end;
```

4. Specification of token.

Strings and languages

Terms for a part of string

Term	Definition
Prefix of S	A string obtained by removing zero or more trailing symbol of string S. e.g., ban is prefix of banana.
Suffix of S	A string obtained by removing zero or more leading symbol of string S. e.g., nana is suffix of banana.
Sub string of S	A string obtained by removing prefix and suffix from S. e.g., nan is substring of banana.
Proper prefix, suffix and substring of S	Any nonempty string x that is respectively prefix, suffix or substring of S, such that s≠x
Subsequence of S	A string obtained by removing zero or more not necessarily contiguous symbol from S. e.g., baaa is subsequence of banana.

Table 2.2. Terms for a part of a string

Operation on languages

Definition of operation on language

Operation	Definition
Union of L and M Written L U M	L U M = {s | s is in L or s is in M }
concatenation of L and M Written LM	LM = {st | s is in L and t is in M }
Kleene closure of L written L*	L* denotes “zero or more concatenation of” L.
Positive closure of L written L+	L+ denotes “one or more concatenation of” L.

Table 2.3. Definitions of operations on languages

5. Regular Expression & Regular Definition.

Regular Expression

1. 0 or 1
   0+1
2. 0 or 11 or 111
   0+11+111

3. Regular expression over ∑={a,b,c} that represent all string of length 3.
   (a+b+c)(a+b+c)(a+b+c)
4. String having zero or more a.
   a*
5. String having one or more a.
   a^+
6. All binary string.
   (0+1)*
7. 0 or more occurrence of either a or b or both
   (a+b)*
8. 1 or more occurrence of either a or b or both
   $(a+b)^+$
9. Binary no. end with 0
   (0+1)*0
10. Binary no. end with 1
    (0+1)*1
11. Binary no. starts and end with 1.
    1(0+1)*1
12. String starts and ends with same character.
    0(0+1)*0 or a(a+b)*a
    1(0+1)*1 b(a+b)*b
13. All string of a and b starting with a
    a(a/b)*
14. String of 0 and 1 end with 00.
    (0+1)*00
15. String end with abb.
    (a+b)*abb
16. String start with 1 and end with 0.
    1(0+1)*0
17. All binary string with at least 3 characters and 3rd character should be zero.
    (0+1)(0+1)0(0+1)*
18. Language which consist of exactly two b's over the set ∑={a,b}
    a*ba*ba*
19. ∑={a,b} such that 3rd character from right end of the string is always a.
    (a+b)*a(a+b)(a+b)
20. Any no. of a followed by any no. of b followed by any no. of c.
    a*b*c*
21. It should contain at least 3 one.
    (0+1)*1(0+1)*1(0+1)*1(0+1)*
22. String should contain exactly Two 1's
    0*10*10*
23. Length should be at least be 1 and at most 3.
    (0+1) + (0+1) (0+1) + (0+1) (0+1) (0+1)

24. No.of zero should be multiple of 3
    (1*01*01*01*)*+1*
25. ∑={a,b,c} where a are multiple of 3.
    ((b+c)*a (b+c)*a (b+c)*a (b+c)*)*
26. Even no. of 0.
    (1*01*01*)*
27. Odd no. of 1.
    0*(10*10*)*10*
28. String should have odd length.
    (0+1)((0+1)(0+1))*
29. String should have even length.
    ((0+1)(0+1))*
30. String start with 0 and has odd length.
    0((0+1)(0+1))*
31. String start with 1 and has even length.
    1(0+1)((0+1)(0+1))*
32. Even no of 1
    (0*10*10*)*
33. String of length 6 or less
    $(0+1+\wedge)^6$
34. String ending with 1 and not contain 00.
    $(1+01)^+$
35. All string begins or ends with 00 or 11.
    (00+11)(0+1)*+(0+1)*(00+11)
36. All string not contains the substring 00.
    (1+01)* (^+0)
37. Language of all string containing both 11 and 00 as substring.
    ((0+1)*00(0+1)*11(0+1)*)+ ((0+1)*11(0+1)*00(0+1)*)
38. Language of C identifier.
    (_+L)(_+L+D)*

Regular Definition

- A regular definition gives names to certain regular expressions and uses those names in other regular expressions.
- Here is a regular definition for the set of Pascal identifiers that is define as the set of strings of letters and digits beginning with a letters.

  letter → A | B | . . . | Z | a | b | . . . | z
  digit → 0 | 1 | 2 | . . . | 9
  id → letter (letter | digit)*

- The regular expression id is the pattern for the identifier token and defines letter and digit. Where letter is a regular expression for the set of all upper-case and lower case letters in the alphabet and digit is the regular expression for the set of all decimal digits.

6. Reorganization of Token.

- Here we address how to recognize token.
- We use the language generated by following grammar,

stmt → **if** expr **then** stmt
| **if** expr **then** stmt **else** stmt
| ∈

expr → term **relop** term
| term
term → **id** | **num**

- Where the terminals if, then, else, relop, id and num generates the set of strings given by the following regular definitions,

if → if
then → then
relop → < | <= | = | <> | > | >=
letter → A | B | . . . | Z | a | b | . . . | z
digit → 0 | 1 | 2 | . . . | 9
id → letter (letter | digit)*
num → **digit**$^+$ ()?(E(+/-)?**digit**$^+$)?

- For this language the lexical analyzer will recognize the keyword if, then, else, as well as the lexeme denoted by **relop, id** and **num**.
- num represents the unsigned integer and real numbers of pascal.
- Lexical analyzer will isolate the next token in the input buffer and produces token and attribute value as an output.

7. Transition Diagram.

- A stylized flowchart is called transition diagram.
- Positions in a transition diagram are drawn as a circle and are called states.
- States are connected by arrows called edges.
- Edge leaving state have label indicating the input character.
- The transition diagram for unsigned number is given in Fig.2.4.

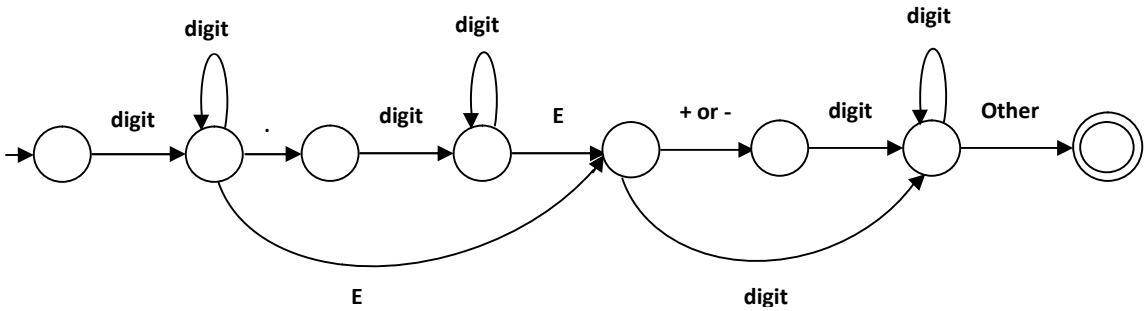

Fig. 2.4. Transition diagram for unsigned number

- Transition diagram for the token **relop** is shown in figure 2.5.

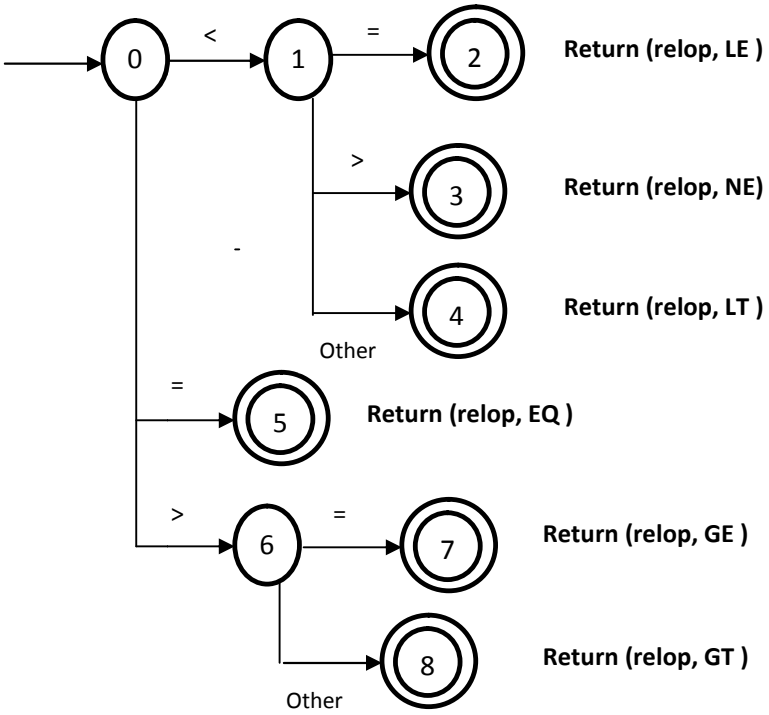

Fig. 2.5. Transition diagram for relational operator

8. Explain Finite automata. (NFA & DFA)

- We compile a regular expression into a recognizer by constructing a generalized transition diagram called a finite automaton.
- A finite Automata or finite state machine is a 5-tuple($S,\sum,S_0,F,\delta$) where
  - S is finite set of states
  - ∑ is finite alphabet of input symbol
  - $S_0 \in S$ (Initial state)
  - F (set of accepting states)
  - δ is a transition function
- There are two types of finite automata,

1. Deterministic finite automata (DFA) have for each state (circle in the diagram) exactly one edge leaving out for each symbol.
2. Nondeterministic finite automata (NFA) are the other kind. There are no restrictions on the edges leaving a state. There can be several with the same symbol as label and some edges can be labeled with ε.

9. Conversion from NFA to DFA using Thompson's rule.

Ex:1 (a+b)*abb

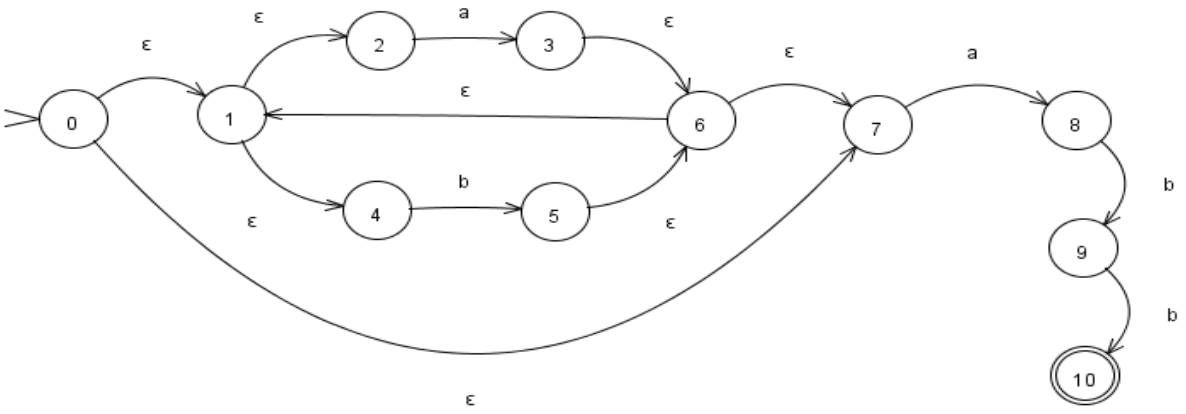

Fig. 2.6. NFA for (a+b)*abb

- ε – closure (0) = {0,1,2,4,7} ---- Let A
- Move(A,a) = {3,8}
  ε – closure (Move(A,a)) = {1,2,3,4,6,7,8}---- Let B
  Move(A,b) = {5}
  ε – closure (Move(A,b)) = {1,2,4,5,6,7}---- Let C

- Move(B,a) = {3,8}
  ε – closure (Move(B,a)) = {1,2,3,4,6,7,8}---- B
  Move(B,b) = {5,9}
  ε – closure (Move(B,b)) = {1,2,4,5,6,7,9}---- Let D

- Move(C,a) = {3,8}
  ε – closure (Move(C,a)) = {1,2,3,4,6,7,8}---- B
  Move(C,b) = {5}
  ε – closure (Move(C,b)) = {1,2,4,5,6,7}---- C

- Move(D,a) = {3,8}
  ε – closure (Move(D,a)) = {1,2,3,4,6,7,8}---- B
  Move(D,b) = {5,10}
  ε – closure (Move(D,b)) = {1,2,4,5,6,7,10}---- Let E

- Move(E,a) = {3,8}
  ε – closure (Move(E,a)) = {1,2,3,4,6,7,8}---- B
  Move(E,b) = {5}
  ε – closure (Move(E,b)) = {1,2,4,5,6,7}---- C

States	**a**	**b**
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

Table 2.4. Transition table for (a+b)*abb

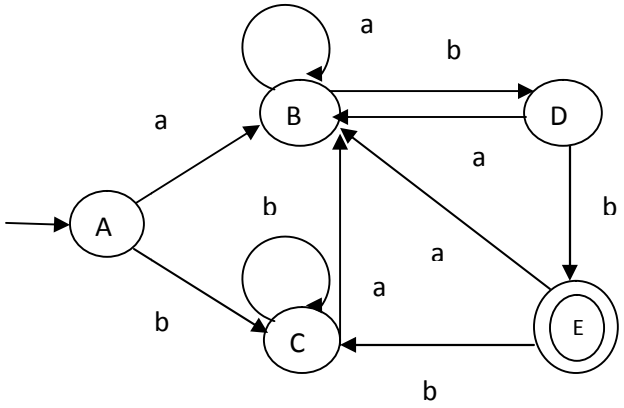

Fig.2.7. DFA for (a+b)*abb

DFA Optimization

- Algorithm to minimizing the number of states of a DFA

1. *Construct an initial partition Π of the set of states with two groups: the accepting states F and the non-accepting states S - F.*
2. *Apply the repartition procedure to Π to construct a new partition Πnew.*
3. *If Π new = Π, let Πfinal= Π and continue with step (4). Otherwise, repeat step (2) with Π = Πnew.*

for each group G of Π do begin
partition G into subgroups such that two states s and t
of G are in the same subgroup if and only if for all
input symbols a, states s and t have transitions on a
to states in the same group of Π.
replace G in Πnew by the set of all subgroups formed

4. *Choose one state in each group of the partition Πfinal as the representative for that group. The representatives will be the states of M'. Let s be a representative state, and suppose on input a there is a transition of M from s to t. Let r be the representative of t's group. Then M' has a transition from s to r on a. Let the start state of M' be the representative of the group containing start state s_0 of M, and let the accepting states of M' be the representatives that are in F.*
5. *If M' has a dead state d (non-accepting, all transitions to self), then remove d from M'. Also remove any state not reachable from the start state.*

Example: Consider transition table of above example and apply algorithm.

- Initial partition consists of two groups (E) accepting state and non accepting states (ABCD).

- E is single state so, cannot be split further.
- For (ABCD), on input a each of these state has transition to B. but on input b, however A, B and C go to member of the group (ABCD), while D goes to E, a member of other group.
- Thus, (ABCD) split into two groups, (ABC) and (D). so, new groups are (ABC)(D) and (E).
- Apply same procedure again no splitting on input a, but (ABC) must be splitting into two group (AC) and (B), since on input b, A and C each have a transition to C, while B has transition to D. so, new groups (AC)(B)(D)(E).
- Now, no more splitting is possible.
- If we chose A as the representative for group (AC), then we obtain reduced transition table shown in table 2.5,

States	**a**	**b**
A	B	C
B	B	D
D	B	E
E	B	C

Table 2.5. Optimized Transition table for (a+b)*abb

10. Conversion from Regular Expression to DFA without constructing NFA.

Ex:1 (a+b)*abb#

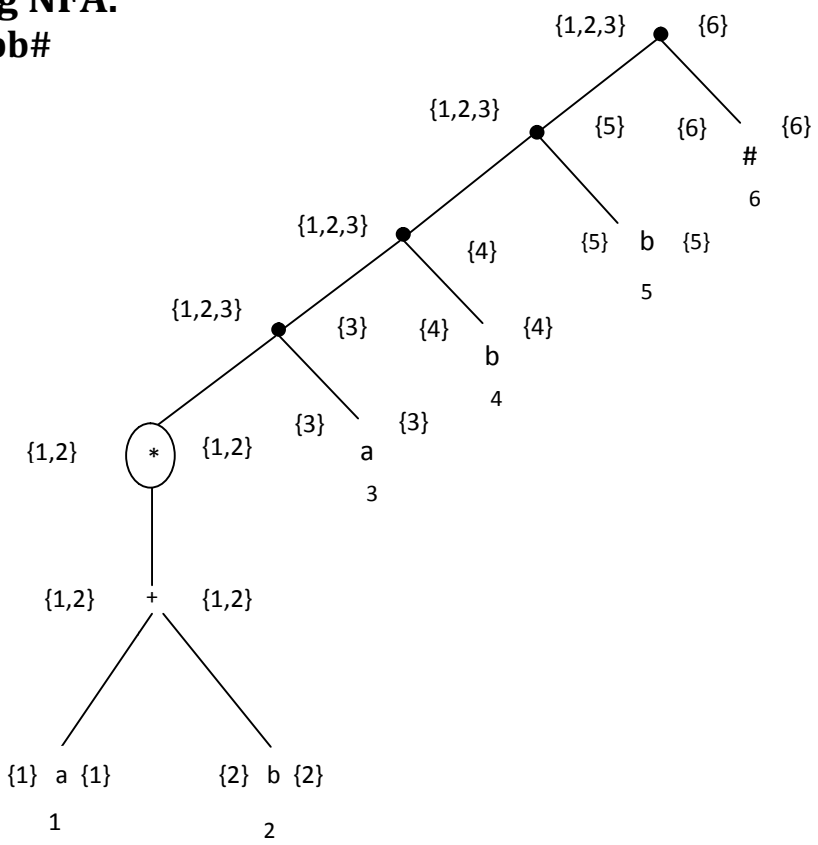

Fig.2.8. Syntax tree for (a+b)*abb#

- To find followpos traverse concatenation and star node in depth first search order.

1. i=lastpos(c1)={5} firstpos(c2)={6} followpos(i)=firstpos(c2) followpos(5)={6}	2. i=lastpos(c1)={4} firstpos(c2)={5} followpos(i)=firstpos(c2) followpos(4)={5}	3. i=lastpos(c1)={3} firstpos(c2)={4} followpos(i)=firstpos(c2) followpos(3)={4}
4. i=lastpos(c1)={1,2} firstpos(c2)={3} followpos(i)=firstpos(c2) followpos(1)={3} followpos(2)={3}	5. i=lastpos(c1)={1,2} firstpos(c1)={1,2} followpos(i)=firstpos(c1) followpos(1)={1,2} followpos(2)={1,2}	

Position	Followpos(i)
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}

Table 2.6. follow pos table

Construct DFA

Initial node = firstpos (root node)= {1,2,3} -- A

δ(A,a) = followpos(1) U followpos(3)
= {1,2,3} U {4}
= {1,2,3,4} --- B

δ (A,b) = followpos(2)
= {1,2,3} ---- A

δ (B,a) = followpos(1) U followpos(3)
= {1,2,3} U {4}
= {1,2,3,4} --- B

δ(B,b) = followpos(2) U followpos(4)
= {1,2,3} U {5}
= {1,2,3,5} --- C

δ(C,a) = followpos(1) U followpos(3)
= {1,2,3} U {4}
= {1,2,3,4} --- B

δ(C,b) = followpos(2) U followpos(5)
= {1,2,3} U {6}

= {1,2,3,6} --- D

δ(D,a) = followpos(1) U followpos(3)
= {1,2,3} U {4}
= {1,2,3,4} --- B
δ(D,b) = followpos(2) = {1,2,3}---A

Transition Table		
States	a	b
A	B	A
B	B	C
C	B	D
D	B	A

Table 2.7. Transition table for (a+b)*abb

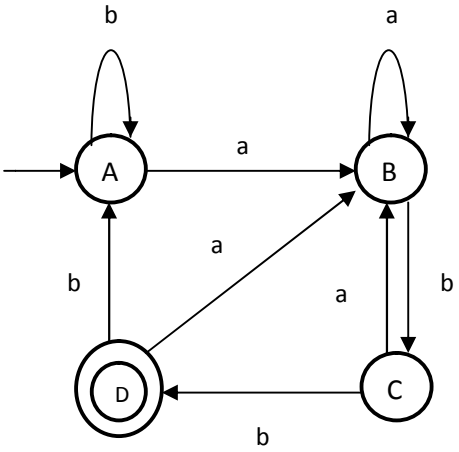

Fig.2.9. DFA for (a+b)*abb

Ex:2 a*b*a(a\b)*b*a#

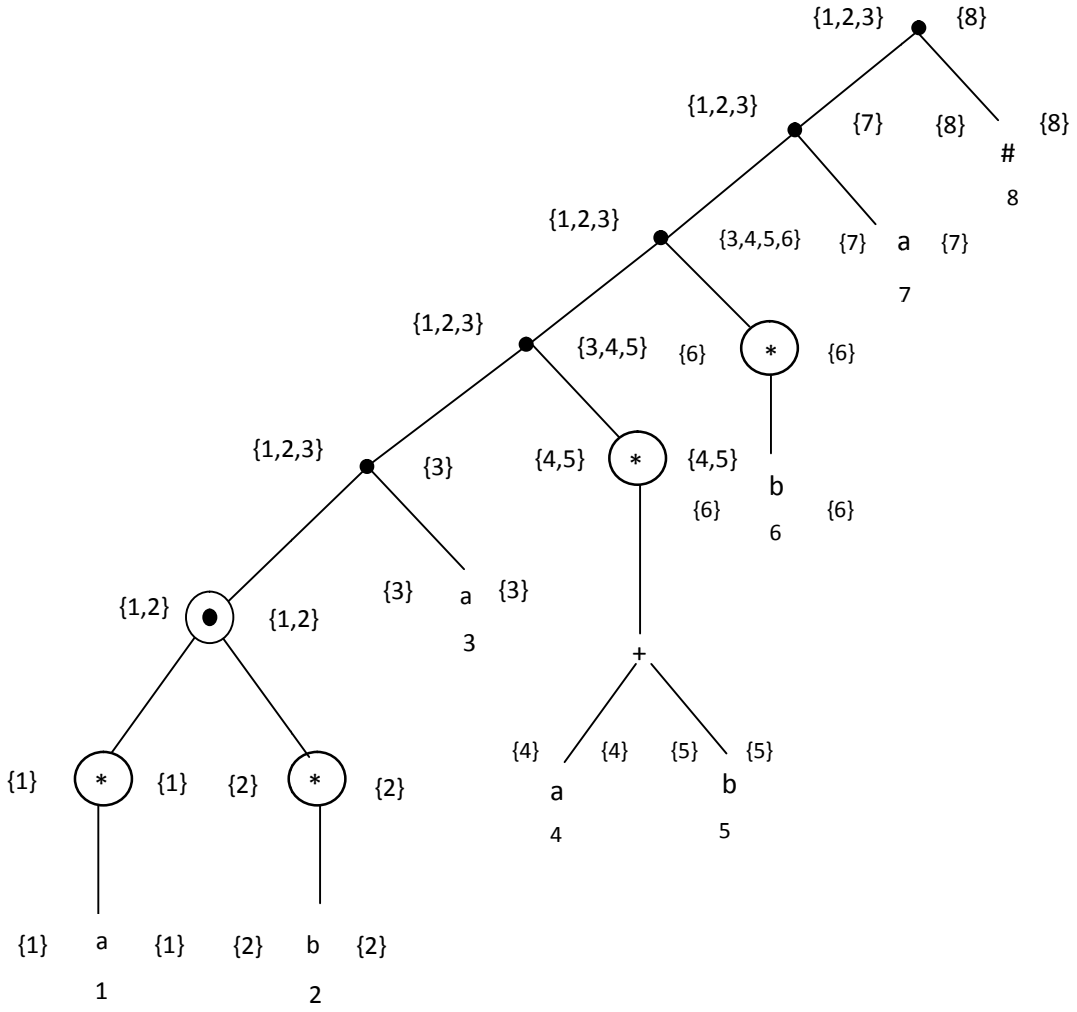

Fig.2.10. Syntax tree for a*b*a(a\b)*b*a#

- To find followpos traverse concatenation and star node in depth first search order

1. i=lastpos(c1)={7} firstpos(c2)={8} followpos(i)=firstpos(c2) followpos(7)={8}	2. i=lastpos(c1)={3,4,5,6} firstpos(c2)={7} followpos(i)=firstpos(c2) followpos(3)={7} followpos(4)={7} followpos(5)={7} followpos(6)={7}	3. i=lastpos(c1)={3,4,5} firstpos(c2)={6} followpos(i)=firstpos(c2) followpos(3)={6} followpos(4)={6} followpos(5)={6}
4. i=lastpos(c1)={3} firstpos(c2)={4,5} followpos(i)=firstpos(c2)	5. i=lastpos(c1)={1,2} firstpos(c2)={3} followpos(i)=firstpos(c2)	6. i=lastpos(c1)={1} firstpos(c2)={2} followpos(i)=firstpos(c2)

followpos(3)={4,5}	followpos(1)={3} followpos(2)={3}	followpos(1)={2}
7. i=lastpos(c1)={1} firstpos(c1)={1} followpos(i)=firstpos(c1) followpos(1)={1}	8. i=lastpos(c1)={2} firstpos(c1)={2} followpos(i)=firstpos(c1) followpos(2)={2}	9. i=lastpos(c1)={4,5} firstpos(c1)={4,5} followpos(i)=firstpos(c1) followpos(4)={4,5} followpos(5)={4,5}
9. i=lastpos(c1)={6} firstpos(c1)={6} followpos(i)=firstpos(c1) followpos(6)={6}		

n	followpos(n)
1	{1,2,3}
2	{2,3}
3	{4,5,6,7}
4	{4,5,6,7}
5	{4,5,6,7}
6	{6,7}
7	{8}

Table 2.8. follow pos table

Construct DFA

Initial node = firstpos (root node)= {1,2,3} -- A

δ (A,a) = followpos(1) U followpos(3)
= {1,2,3} U {4,5,6,7}
= {1,2,3,4,5,6,7} ---B

δ (A,b) = followpos(2)
= {2,3} ----C

δ (B,a) = followpos(1) U followpos(3) U followpos(4) U followpos(7)
= {1,2,3} U {4,5,6,7} U {8} U {4,5,6,7}
= {1,2,3,4,5,6,7,8} ---D

δ (B,b) = followpos(2) U followpos(5) U followpos(6)
={2,3} U {4,5,6,7} U {4,5,6,7}
= {2,3,4,5,6,7} ---E

δ (C,a) = followpos(3)
= {4,5,6,7} ---F

δ (C,b) = followpos(2)
= {2,3} ---C

δ (D,a) = followpos(1) U followpos(3) U followpos(7) U followpos(4)
= {1,2,3} U {4,5,6,7} U {8} U {4,5,6,7}
= {1,2,3,4,5,6,7,8} ---D
δ (D,b) = followpos(2) U followpos(5) U followpos(6)
={2,3} U {4,5,6,7} U {4,5,6,7}
= {2,3,4,5,6,7} ---E

δ (E,a) = followpos(3) U followpos(4) U followpos(7)
={4,5,6,7} U {4,5,6,7} U {8}
= {4,5,6,7,8} ---G
δ (E,b) = followpos(2) U followpos(5) U followpos(6)
={2,3} U {4,5,6,7} U {4,5,6,7}
= {2,3,4,5,6,7} ---E

δ (F,a) = followpos(4) U followpos(7)
= {4,5,6,7} U {8}
= {4,5,6,7,8} ---G
δ (F,b) = followpos(5) U followpos(6)
={4,5,6,7} U {4,5,6,7}
= {4,5,6,7} ---F

δ (G,a) = followpos(4) U followpos(7)
= {4,5,6,7} U {8}
= {4,5,6,7,8} ---G
δ (G,b) = followpos(5) U followpos(6)
={4,5,6,7} U {4,5,6,7}
= {4,5,6,7} ---F

Transition table:

Transition table		
	a	b
A	B	C
B	D	E
C	F	C
D	D	E
E	G	E
F	G	F
G	G	F

Table 2.9. Transition table for a*b*a(a\b)*b*a#

DFA:

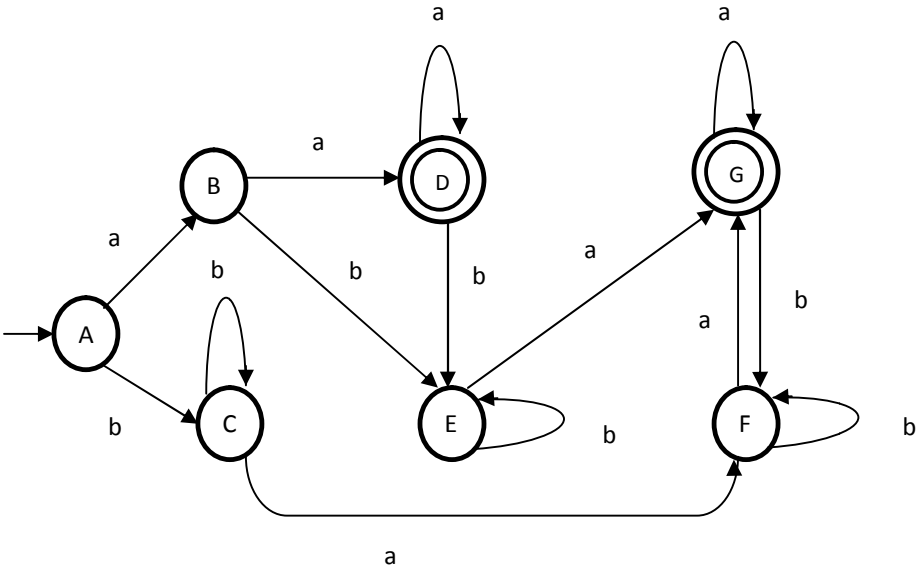

Fig.2.11. DFA for a*b*a(a\b)*b*a#