2. Lexical Analysis

01. Ans: (a)
Sol: Comments are deleted during lexical analysis, by ignoring comments.

02. Ans: (a)
Sol: The expansion of macro is done as the input tokens are generated during the lexical analysis phase.

03. Ans: (a)
Sol: As soon as an identifier identifies as lexemes the scanner checks whether it is a reserved word.

04. Ans: (c)
Sol: Type checking is a semantic feature.

05. Ans: (a)
Sol: Compiler identifies only Grammatical errors, but not logical & runtime errors.

06. Ans: (d)
Sol: A compiler that runs on one machine and generates code for another machine is called cross compiler.

07. Ans: (b)
Sol: The object code which is obtained from Assembler is in Hexadecimal, which is not executable, but it is relocated.

08. Ans: (b)
Sol: Syntax analysis can be expanded but the CFG describes the syntax becomes cumbersome.

09. Ans: (a)
Sol: The identifiers are entered into the symbol table during lexical analysis phase.

10. Ans: (a)
Sol: As I/O to an external device is involved most of the time is spent in lexical analysis.

11. Ans: (a)
Sol: The lex utility creates a DFA from the regular definition.

12. Ans: (b)
Sol: The specifications of lexical analysis we write in lex language, when it run through lex compiler it generates an output called lex.yy.c.

13. Ans: (a)
Sol: Parenthesis matching cannot be done at the lexical analysis phase.

14. Ans: 20
15. Ans: 7

16. Ans: (b)
Sol: if , (, x , > =, y, ), {, x, =, x, +, y, ;, }, else, {, x, =, x, −, y, ;, }, ;
17. Ans: (d)  
   **Sol:** All are tokens only.

18. Ans: (c)  
   **Sol:** Syntax tree is input to semantic analyzer. Character stream is input to lexical analyzer. Intermediate representation is input to code generation. Token stream is input to syntax analyzer.

19. Ans: 18

20. Ans: (b)

### 3. Parsing Techniques

01. Ans: (b)  
   **Sol:** As + is left associative the left most + should be reduced first

02. Ans: (d)  
   **Sol:**

   
   \[
   S \rightarrow S \rightarrow S \rightarrow S \rightarrow a \rightarrow a \\
   S \rightarrow S \rightarrow S \rightarrow a \\
   S \rightarrow S \rightarrow S \\
   S \rightarrow S \rightarrow S \\
   \]

   So the sentence has an infinite number of derivations.

03. Ans: (a)  
   **Sol:** The grammar which is both left and right recursive is always ambiguous grammar.

04. Ans: (d)  
   **Sol:**

   
   \[
   S \rightarrow S \rightarrow b \\
   b \rightarrow S \rightarrow a \\
   a \rightarrow S \rightarrow b \\
   \]

   Hence the option (d) is correct.

05. Ans: 2
   **Sol:**

06. Ans: (e)  
   **Sol:**

07. Ans: (a)  
   **Sol:** S \rightarrow Ad \rightarrow Sad is indirect left recursion.
08. Ans: (c)
Sol: The production of the form $A \rightarrow A \alpha/\beta$ is left recursive, and can be eliminated by replacing with
$A \rightarrow \beta A^1$
$A^1 \rightarrow \alpha A^1/\epsilon$

09. Ans: (d)
Sol: ↑ is least precedence and left associative
+ is higher precedence and right associative

10. Ans: (c)
Sol: Precedence from low to high is ↑, +, id.

11. Ans: (b)
Sol: $\rightarrow \ast$

12. Ans: 144
Sol: $3-2*4\$2*3\$2$
$1*4\$2*3\$2$
$1*16*9$
$16*9$
$= 144$

13. Ans: (b)
Sol: Rule ‘a’ evaluates to 4096
Rule ‘b’ evaluates to 65536
Rule ‘c’ evaluates to 32

14. Ans: (c)
Sol: A bottom up parsing technique builds the derivation tree in bottom up and simulates a rightmost derivation in reverse

15. Ans: (d)
Sol: Operator precedence parser is a shift reduce parser.

16. Ans: (e)
Sol: $\text{first}(s) = \text{first}(A) \cup \text{first}(a) \cup \text{first}(Bb) = \{d\} \cup \{f, a\} \cup \{e, b\} = \{a, b, d, e, f\}$

17. Ans: (d)
Sol: {$s, s$} both follow additional.

18. Ans: (e)
Sol: $\text{first}(A) = \{a, c\}, \text{follow}(A) = \{b, c\}$
$\text{first}(A) \cap \text{follow}(A) = \{c\}$

19. Ans: (d)
Sol: $\text{ Follow}(B) = \text{ First}(C) \cup \text{ First}(x) \cup \text{ Follow}(D) = \{y, m\} \cup \{x\} \cup \text{ Follow}(A) \cup \text{ First}(B) = \{y, m, x\} \cup \{\$\} \cup \{w, x\} = \{w, x, y, m, \$\}$

20. Ans: (a)
Sol: $\text{ Follow}(S) = \{\$\}$
Consider $S \rightarrow [SX]$
$\text{ Follow}(S) = \text{ First}(X)$
$= \{+,-, b\} \cup \{\}\}
= \{+,-, b, \}$
Consider $X \rightarrow + SY$
$\text{ Follow}(S) = \text{ First}(Y)$
$= \{-\} \cup \text{ Follow}(X)$
$= \{-\} \cup \{c, \}$
$= \{-, c, \}$
Consider \( Y \rightarrow - S X c \)
\[
\text{Follow}(S) = \text{First}(X) \\
= \{+, -, b\} \cup \text{First}(c) \\
= \{+, -, b, c\} \\
\therefore \text{Follow}(S) = \{+, -, b, c, ], \}$

21. **Ans:** (c)

**Sol:**
Follow \((T) = \{+, $\} \)
First \((S) = \{a, +, \varepsilon\} \)
\[
\therefore \text{Follow}(T) \cap \text{First}(S) = \{+\}
\]

22. **Ans:** (d)

**Sol:**
Follow \((A)=\text{first}(B) \cup \text{Follow}(S) \cup \text{Follow}(B) \)
\[
= \{e\} \cup \{f\} \cup \{c, d\} = \{c, d, e, $\}. \]

23. **Ans:** (d)

**Sol:**
Follow \((S) = \{\$, a, d\} \)
Follow \((A) = \{a\} \)
Follow \((B) = \{a, d\} \)
Follow \((C) = \{\$, a, d\} \)

24. **Ans:** (c)

**Sol:**
The predictive parsing table:

<table>
<thead>
<tr>
<th>&lt;expression&gt;</th>
<th>&lt;expression&gt; →</th>
<th>identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;factor&gt;</td>
<td>&lt;expression&gt; →</td>
<td>A</td>
</tr>
<tr>
<td>&lt;rest&gt;</td>
<td></td>
<td>B</td>
</tr>
</tbody>
</table>

**Sol:**
The grammar is not LL(1), as on input symbol a there is a choice.

The grammar is not LL(2), as input ab there is a choice.

25. **Ans:** (c)

**Sol:**
The grammar is not LL(1), as on input symbol a there is a choice.

The grammar is not LL(2), as input ab there is a choice.

26. **Ans:** (c)

**Sol:**
The grammar is not LL(1), as on input abc there is a choice.

27. **Ans:** (c)

**Sol:**
The grammar is not LL(2), as input ab there is a choice.

28. **Ans:** (a)

**Sol:**
A left recursive grammar cannot be LL(1).

29. **Ans:** (c)

**Sol:**
A \( \rightarrow \varepsilon \) production is added in ‘A’ row and Follow(A) column.

30. **Ans:** (d)

**Sol:**
S \( \rightarrow aSb \) and S \( \rightarrow \varepsilon \) both appear in ‘S’ row and ‘a’ column.

31. **Ans:** (b)

**Sol:**
The first 2 symbols of ‘S’ production is distinct hence the grammar is LL(2).
32. Ans: (d)
Sol: The rightmost derivation is
\[<\text{accumulated\_sum}> \rightarrow <\text{accumulated\_sum}> *<\text{number}> \rightarrow <\text{accumulated\_sum}> +<\text{number}> * \text{number} \rightarrow \text{number} + \text{number} * \text{number}\]

33. Ans: (d)
Sol: An operator grammar is \(\varepsilon\)-free grammar and no two non terminals are adjacent.

34. Ans: (c)
Sol: An operator grammar is \(\varepsilon\)-free grammar and no two non-terminals are adjacent.

35. Ans: (c)
Sol: An operator grammar is \(\varepsilon\)-free grammar and has no two adjacent non-terminals.

36. Ans: (d)
Sol: As per normal HLL rules exponentiation is right associative where as -, +, * are left associative.

37. Ans: (d)
Sol: Lead (S) = \{a\} \cup \{c\} \cup Lead (B) \cup \{d\} \\
= \{a,c,d,e\}

38. Ans: (b)
Sol: Trail (E) = \{+\} \cup Trail (T) \\
= \{+, *\} \cup Trail (F) \\
= \{+, *, \text{id}\}

39. Ans: (b)
Sol: Lead (E) \(>\) and lead (E) contains \{+, \text{id}\}

40. Ans: (d)
Sol: Possible relations with \(c\) are \(d>c\) and \(c>\$\) only.

41. Ans: (b)
Sol: The grammar \(E \rightarrow E + E/a\) can have an operator precedence parser but not an LR parser.

42. Ans: (a)
Sol: The grammar \(E \rightarrow E + T \mid T, T \rightarrow i\) is left recursive. So it is not LL(1) but is LR(0). So (a) is true & (b) is false.

The grammar
\[S \rightarrow a \mid aA\]
\[A \rightarrow b\]
has the LR(0) machine

\[S \rightarrow .a\]
\[S \rightarrow .aA\]
\[a \rightarrow S \rightarrow a.\]
\[A \rightarrow .b\]

Hence not LR(1) but is SLR(1).

43. Ans: (d)
Sol: The grammar
\[E \rightarrow E + E \mid E * E \mid i\]
Can have a shift reduce parser if we use the precedence and associativity of operations.
The operator precedence technique works with some ambiguous grammars.
44. Ans: (d)  
Sol: The grammar  
\[ S \rightarrow a \mid A, \ A \rightarrow a \]  
is neither LL(1) nor LR(0) & is ambiguous.  
No ambiguous grammar can be LL or LR.

45. Ans: (d)  
Sol: No ambiguous grammar can be LR(1).

46. Ans: (c)  
Sol: The grammar  
\[ S \rightarrow Aa \mid Bb \]  
\[ A \rightarrow \varepsilon \]  
\[ B \rightarrow \varepsilon \]  
is LL(1) but not LR(0)  
The LR(0) machine has a conflict.

The grammar is  
\[ S \rightarrow a \mid ab \]  
Is LR(2) & not LR(1).

47. Ans: (d)  
Sol: Every LR(0) grammar is SLR(1)  
Every SLR(1) grammar is LALR(1)  
Every LALR(1) grammar is LR(1)  
The grammar \( S \rightarrow a \) is both LL(2) & LR(0)  
trivially.

48. Ans: (b)  
Sol: Every LL(1) is LR(1)
52. Ans: 7
Sol:

\[
\begin{align*}
S' & \rightarrow S \\
S & \rightarrow AA \\
A & \rightarrow aA \\
A & \rightarrow b \\
A & \rightarrow b
\end{align*}
\]

53. Ans: (c)
Sol: The given grammar is LR(0) and every LR(0) is LR(1).

54. Ans: 2
Sol:

\[
\begin{align*}
E' & \rightarrow . E \\
E & \rightarrow . E \\
E & \rightarrow . T \\
T & \rightarrow . T* \\
T & \rightarrow . F \\
F & \rightarrow . id
\end{align*}
\]

55. Ans: (a)
Sol: The grammar is LL(1), LR(0), SLR(1), LALR(1) & LR(1).

56. Ans: (d)
Sol: The grammar is ambiguous.

\[
\begin{align*}
& A \\
& A + A \\
& A + A
\end{align*}
\]

There are two derivation trees for the sentence \(i + i + i\). As the grammar is ambiguous it cannot be LL or LR. So, (a), (b), (c), are ruled out. The answer is (d).

57. Ans: 2
Sol: The LR(0) items of the grammar is

\[
\begin{align*}
S' & \rightarrow . S \\
S & \rightarrow . AaAb \\
S & \rightarrow . BbBa \\
S & \rightarrow . aBe \\
S & \rightarrow . bAe, \\
S & \rightarrow . aAd, \\
S & \rightarrow . bBa, \\
S & \rightarrow . AaAd, \\
S & \rightarrow . bBd, \\
S & \rightarrow . bAd, \\
S & \rightarrow . bBe, \\
S & \rightarrow . b Ae,
\end{align*}
\]

Reduce – Reduce conflict.

58. Ans: (a)
Sol: Consider the partial LR(1) machine shown above. The states (X) & (Y) have a common core. However if we merge the states to
obtain the LALR(1) machine we will end up with conflicts. So the grammar is LR(1) but not LALR(1).

59. Ans: (a)
Sol:

Consider the partial LR(1) machine above. The states ⃝&Ⓨ have a common core but different look ahead sets. If we merge ⃝&Ⓨ so obtain the LALR(1) a conflict arise.

60. Ans: (b)
Sol: LR(1) items of the grammar is

Item 3 has Shift-Reduce conflict.

61. Ans: (d)
Sol:  

As there is no conflicts the grammar is in LALR(1).

62. Ans: (c)
Sol:  

63. Ans: (d)
Sol:

64. Ans: (c)
Sol:  

No. of conflicts=2
65. Ans: (c)
Sol: The grammar is only LR(1)

66. Ans: (d)
Sol: The grammar is LL(1)

\[
\begin{align*}
S' & \rightarrow S \\
S & \rightarrow .(S) \\
S & \rightarrow .
\end{align*}
\]

Every LL(1) is LR(1)

67. Ans: (b)

68. Ans: (b)
Sol: SLR(1) & LALR(1) have the same number of states. LR(1) may have more.

69. Ans: 10
Sol: The number of states in both SLR(1) and LALR(1) are same.

70. Ans: (c)
Sol: YACC uses LALR (1) parse table as it uses less number of states requires less space and takes less time for the construction of parse tree.

02. Ans: (c)
Sol: The attribute ‘val’ is synthesized and the SDD is S-attributed and every ‘S’-attributed is L-attributed definition

03. Ans: (c)
Sol: Given SDT counting the number of a’s and b’s in a given string.

04. Ans: (c)
Sol: For input: \( a + b - c \)

05. Ans: (c)
Sol: Bottom up traversal of the parse tree results the output: 10.

4. Syntax Directed Translation Schema

01. Ans: (c)
Sol: SDT is part of semantic Analysis
06. Ans: (b)
Sol: \( S \rightarrow S_1 S_2 c \{ S.val = S_1.val * S_2.val - 4 \} \)
\( S \rightarrow a \{ S.val = 6 \} \)
\( S \rightarrow b \{ S.val = 2 \} \)
The rightmost derivation of ‘abc’ is
\( S \Rightarrow SSc \)
\( \Rightarrow S bc \)
\( \Rightarrow a bc \)
In \( S_1 \), \( S_2 \), \( c \), \( S_1.val = 6 \), \( S_2.val = 2 \). So answer is “8”.

07. Ans: (c)
Sol: \( \neg(A \land (A \rightarrow B)) \)
\( G \rightarrow \neg(A \land (A \rightarrow B)) \)
\( F \rightarrow \neg(A \land (A \rightarrow B)) \)
\( F_1 \rightarrow A \rightarrow B \)
\( F \rightarrow \neg(A \land (A \rightarrow B)) \)
\( (F \rightarrow \neg(A \land (A \rightarrow B))) \)
\( A=F \rightarrow (F \rightarrow (A \land (A \rightarrow B))) \)
\( B=F \rightarrow (F \rightarrow (A \land (A \rightarrow B))) \)

08. Ans: (c)
Sol: The rightmost derivation is
\( E \rightarrow E + E \rightarrow E + E + E \)
\( \Rightarrow E + E + E + E \)
\( \Rightarrow E + E + E + E + E \)
\( \equiv a + b + c + d + e \)

09. Ans: (a)
Sol: The leftmost derivation for aaaa is
\( S \rightarrow aS \)
\( \rightarrow aaS \)
\( \rightarrow aaaS \)
\( \rightarrow aaaa \)
The dependency graph

10. Ans: (a)
Sol: The rightmost derivation is
\( S \rightarrow aB \rightarrow aa BB \rightarrow aa Bb \rightarrow aa bb \)

11. Ans: (c)
Sol: \( S \rightarrow aA \{ \text{print 1} \} \)
\( S \rightarrow a \{ \text{print 2} \} \)
\( A \rightarrow Sb \{ \text{print 3} \} \)
Input: aab

12. Ans: (c)
Sol: \( a_1b_1a_2b_2b_3 \)
\( S \Rightarrow a_1S \)
\( \Rightarrow a_1b_1S \)
\( \Rightarrow a_1b_1a_2S \)
\( \Rightarrow a_1b_1a_2b_2S \)
\( \Rightarrow a_1b_1a_2b_2b_3 \)
Above is rightmost derivation

\[ S \rightarrow b_3 \]
\[ S \rightarrow b_2S \]
\[ S \rightarrow a_2S \]
\[ z \]
\[ zy \]
\[ zyx \]
\[ b_1S \]
\[ a_1S \]
\[ zyxy \]
\[ zyxxy \]
13. Ans: (a)
Sol: 

14. Ans: (c)
Sol: As the grammar is ambiguous & we do not specify the precedence of operators either postfix form may result depending on the parser implementation.

15. Ans: (d)
Sol: 

16. Ans: (a)
Sol: According to the action of shift reduce parser, the parse tree constructed is

The Depth First Traversal of the above parse tree is a b * c ↑

5. Intermediate Code Generation

01. Ans: (c)
Sol: The purpose of using intermediate codes in compilers is to reuse machine independent code for other compilers.

02. Ans: (d)
Sol: The final result is the machine language code. The others are all standard intermediate forms.

03. Ans: (d)
Sol: TAC is a statement that contains atmost three memory references.

04. Ans: (d)
Sol: TAC can be implemented as a record structure with fields for operator, and arguments as Quadruples, triples and indirect triples.
05. Ans: (b)
Sol: The Quadruples is record structure with four fields.
1. (*, b, c, T1)
2. (+, a, T1, T2)
3. (–, T2, d, T3)

06. Ans: (c)
Sol: (1) (and, b, c, T1)
(2) (or, a, T1, T2, c, T3)
(3) (or, T2, c, T3)

07. Ans: (a)
Sol: 1. (+, b, c)
2. (NEG, (1))
3. (*, a, (2))

08. Ans: 10
Sol: Rewriting the given assignments
x1 = u1 – t1; → needs two new variables
y2 = x1 * v1; → needs three new variables
x3 = y2 + w1; → needs four new variables
y4 = t2 – z1; → needs five new variables
y5 = y2 + w1 + y4; → needs 10 new variables atmost

09. Ans: (b)
Sol: All assignments in SSA are to variables with distinct names
p3 = a – b
q4 = P3 * c
p4 = u * v
q5 = P4 + q4

10. Ans: (d)
Sol: Peephole optimization expression is the final code.

11. Ans: (d)
Sol: DAG for the expression a*b*b is

12. Ans: (b)
Sol: DAG is constructed based on precedence and associativity of operators, and option (b) is the correct representation.

13. Ans: 4
Sol: Number of nodes = 4

14. Ans: (b)
Sol: Nodes = 8
Edges = 10
15. **Ans: (a)**

**Sol:** In C the storage for array is row major order.

Between \( X[l] \[32\] \[8\] \& X \[l+1\] \[32\] \[8\] \)
there must be \(32 \times 8\) integer of type int i.e
\(32 \times 8 \times 4 = 1024\) bytes. So in \( X[i] \[j\] \[k\] 
for a variation of index \(i\) by 1, 1024 bytes must be skipped. So the answer must be (a)

16. **Ans: (b)**

**Sol:**

(1) \((+, c, d)\)

(2) \((-b, (1))\)

(3) \((*, e, f)\)

(4) \((+, (2), (3))\)

(5) \( (=, a, (4))\)