Sr.	Unit No.	Question	BL	CO
1.	**1**	Discuss types of translators.	**U**	**CO1**
2.	**1**	Compare compiler v/s interpreter v/s assembler	**U**	**CO1**
3.	**1**	Discuss analysis synthesis phase of compiler.	**U**	**CO1**
4.	**1**	Explain Phases of compiler.	**U**	**CO1**
5.	**1**	Describe output of all the phases of compiler for following expression: 1. Position=initial+rate*60 2. A=A+B*C*2 3. F=(9/5)*C+32 4. I=(p*n*r)/100	**U**	**CO1**
6.	**1**	Compare front end and back end	**R**	**CO1**
7.	**1**	Explain cousins of compiler.	**U**	**CO1**
8.	**1**	Discuss pass structure of compiler.	**U**	**CO1**
9.	**1**	Write the effect of reducing the number of passes.	**R**	**CO1**
10.	**1**	Explain types of Compilers.	**U**	**CO1**
11.	**2**	Discuss interaction of scanner and parser.	**U**	**CO2**
12.	**2**	Describe token, pattern, and lexemes.	**U**	**CO2**
13.	**2**	Discuss input buffering methods.	**U**	**CO2**
14.	**2**	Design regular expression for the given language.	**A**	**CO2**
15.	**2**	Discuss types of finite automata.	**U**	**CO2**
16.	**2**	Design NFA from regular expression for the given language. 1. abba 2. bb(a)* 3. (a|b)* 4. a* | b* 5. a(a)*ab 6. aa*+ bb* 7. (a+b)*abb 8. 10(0+1)*1 9. (a+b)*a(a+b) 10. (0+1)*010(0+1)* 11. (010+00)*(10)* 12. 100(1)*00(0+1)*	**A**	**CO2**
17.	**2**	Design NFA for given regular expression using Thompson's notation and then convert it into DFA. (using Subset construction) OR Explain subset construction method for constructing DFA from an NFA with an example. 1. (a+b)*a(a+b) 2. (a+b)*ab*a 3. (a | b)* ab 4. (a | b)*abb 5. (0|1)* 6. (0*|1*)* 7. (0|1)*011 8. a(a+b)*ab 9. b(a+b)*abb	**A**	**CO2**
18.	**2**	Perform optimization on given DFA.	**A**	**CO2**
19.	**2**	Design DFA without constructing NFA from given regular expression. (Tree based	**A**	**CO2**

<table>
<tr><td></td><td></td><td>method)
1. (a | b)*abb#
2. (a | b)*a#
3. a*(b* | c*)(a | c)*#
4. (a+b)*ab*a#
5. 0*1*(0/1)#</td><td></td><td></td></tr>
<tr><td>20.</td><td>2</td><td>Discuss LEX Tool.</td><td>U</td><td>CO2</td></tr>
<tr><td>21.</td><td>3</td><td>Discuss role of parser.</td><td>U</td><td>CO3</td></tr>
<tr><td>22.</td><td>3</td><td>Explain context free grammar with example.</td><td>R</td><td>CO3</td></tr>
<tr><td>23.</td><td>3</td><td>Explain leftmost and rightmost derivation with example.
Or
Perform leftmost or rightmost derivation for given string and grammar.
1. S→aSbS | bSaS | ε (string: abab)
2. S→A1B
A→0A | ϵ
B→0B | 1B | ϵ (String: 1001)
3. E→E+E | E*E | id | (E) | -E (String: id+id*id)
4. S→ aS | Sa | ϵ (string: aaaa)
5. S→SS+ | SS* | a (string: aa+a*)</td><td>U</td><td>CO3</td></tr>
<tr><td>24.</td><td>3</td><td>Discuss Ambiguity with example.
<table>
<tr><td>S →a|Sa|bSS|SSb|SbS</td><td>S→ABA,
A→aA | ε
B→bB | ε</td><td>S →ABA
A →aA | Λ
B →bB | Λ</td></tr>
<tr><td>S → A | B
A →aAb | aabb
B →abB | Λ</td><td colspan="2">S → S + S | S * S | a | b
Write the unambiguous CFG based on precedence rules for the above grammar.
Derive the parse tree for expression (a + a)*b</td></tr>
</table></td><td>A</td><td>CO3</td></tr>
<tr><td>25.</td><td>3</td><td>Explain types of parsers.</td><td>U</td><td>CO3</td></tr>
<tr><td>26.</td><td>3</td><td>Discuss backtracking with suitable example.</td><td>U</td><td>CO3</td></tr>
<tr><td>27.</td><td>3</td><td>Perform Left recursion and left factoring on given grammar.
Left recursion
1. A→Abd | Aa | a
B→Be | b
2. A→AB | AC | a | b
3. S→A | B
A→ABC | Acd | a | aa
B→Bee | b
4. Exp→Exp+term | Exp-term | term
5. S→(L) | a
L→L , S | S
Left factoring
1. S→iEtS | iEtSeS | a
2. A→ ad | a | ab | abc | x
3. S→A
A→Ad | Ae | aB | aC
B→bBC | f
C→g</td><td>A</td><td>CO3</td></tr>
<tr><td>28.</td><td>3</td><td>Explain Recursive Decent Parsing method.</td><td>A</td><td>CO3</td></tr>
<tr><td>29.</td><td>3</td><td>Design LL(1) Parsing table for the given grammar.
1. S→aBa
B→bB | ϵ
2. check that following grammar is LL(1) or not.</td><td>A</td><td>CO3</td></tr>
</table>

		S→aB | ϵ B→bC | ϵ C→cS | ϵ 3. Develop an LL(1) parser table for the following grammar and Parse the string using the parsing table : (id*id) + (id*id) E→TA A→+TA| ϵ T→VB B→*VB| ϵ V→id|(E) 4. Develop an LL(1) parser table for following: S → iCtSeS | iCtS | a C→b 5. E→BA A→&BA| ϵ B→true|false 6. S → AaAb | BbBa A→ϵ B→ϵ 7. S→ aAB | bA | ϵ A→ aAb | ϵ B→ bB | ϵ 8. S→iCtSA | a A→ eS | ϵ C→ b 9. S→ (L) | a L→ L,S | S 10. S→1AB | ε A→1AC | 0C B→0S C→1		
30.	**3**	Define: Handle pruning and Handle.	**R**	**CO3**
31.	**3**	Apply shift reduce parser for parsing given string using unambiguous grammar: 1. id-id*id–id 2. id+id*id 3. id*id+id+id 4. (id+id)*id 5. id-id-id+id	**A**	**CO3**
32.	**3**	Check the following grammar is operator grammar or not. Justify your answer. Also prepare precedence matrix, and graph. 1. E→E + T | T T→T * F | F F→(E) | id 2. E→EAE | id A→+ | * 3. E→EOE E→id O→* | + | -	**A**	**CO3**

33.	**3**	Design SLR parsing table for given grammar. 1. S→AaBa | BbBa B→Є A→Є 2. E→ E+T | T T→ TF | F F→ F* | a | b 3. E→ E+T | T T→ T*F | F F→ (E) | id	**A**	**CO3**
34.	**3**	Design CLR parsing table for given grammar. 1. S→CC C→cC | d 2. S→aSA | Є A→bS |c	**A**	**CO3**
35.	**3**	Design LALR parsing table for given grammar. 1. S→L=R S→R L→*R L →id R→L 2. S→Aa | bAc | dc | bda A→d 3. S→CC C→ cC | d	**A**	**CO3**
36.	**3**	Discuss YACC parser generator.	**U**	**CO3**
37.	**3**	Explain Syntax directed definitions.	**U**	**CO3**
38.	**3**	Compare synthesized attribute v/s inherited attribute	**U**	**CO3**
39.	**3**	Design annotated parse tree for a given string. 1. 3*5+4n; 2. (3+4)*(5+6)n; 3. 1*2*(3+4)n;	**A**	**CO3**
40.	**4**	Design Syntax directed definition to translates arithmetic expressions from infix to prefix notation.	**A**	**CO4**
41.	**4**	Explain L-Attributed definition.	**U**	**CO4**
42.	**4**	Explain syntax directed translation scheme with example.	**U**	**CO4**
43.	**4**	Explain quadruple, triple, and indirect triple with example.	**U**	**CO4**
44.	**4**	Discuss various representations of three address code for a given example. 1. (a = b * -c + b * -c) 2. a*-(b+c) 3. a+a*(b-c)+(b-c)*d 4. ans=a+b*c/2.0 5. x = -a * b + -a * b 6. -(a*b)+(c+d)-(a+b+c+d)	**U**	**CO4**
45.	**4**	Express following statement in form of DAG. 1. a=b*-c+b*-c	**U**	**CO4**

		2. a + a * (b – c) + (b – c) * d		
46.	**4**	Discuss source language issues.	**U**	**CO4**
47.	**4**	Explain activation record and activation tree.	**U**	**CO4**
48.	**4**	Explain storage allocation strategies in brief.	**U**	**CO4**
49.	**5**	What is code optimization? Discuss its types.	**R**	**CO5**
50.	**5**	Explain Machine independent code optimization techniques.	**U**	**CO5**
51.	**5**	Explain Machine dependent code optimization techniques.	**U**	**CO5**
52.	**5**	Explain peephole optimization.	**U**	**CO5**
53.	**5**	Discuss loop optimization.	**U**	**CO5**
54.	**5**	Discuss issues in code generation.	**U**	**CO5**
55.	**5**	Calculate instruction cost of given assembly code.	**A**	**CO5**
56.	**5**	What is basic block? Write algorithm to partition a basic block.	**U**	**CO5**
57.	**5**	Discuss transformation on basic block.	**U**	**CO5**
58.	**5**	Explain flowgraph with example.	**U**	**CO5**
59.	**5**	Describe code generation algorithm.	**U**	**CO5**
60.	**5**	Design target code for given assignment statement using code generation algorithm.	**U**	**CO5**