1. Draw the diagram of the language processing system.
2. Define the terms: lexme, pattern
3. What is a language processing system?
4. What are the error recovery actions in a lexical analyzer?
5. What is recognizer?
6. What are the different types of error recovery strategies?
7. What are the issues to be considered in the design of lexical analyzer?
8. Define concrete and abstract syntax with example.
9. Write Regular expressions for identifier and number.
11. Which phases of the compiler will store the types of the identifiers in the symbol Table?
12. What is the advantage of a parser generator in the design of any compiler?
13. Draw a transition diagram to represent relational operators.
14. List four software tools that analyses the source program.
15. Distinguish between pass and phases of a compiler.
16. List the compiler construction tools.
17. Define auxiliary definitions in Lex.
18. Define Compiler
19. What is regular expression?
20. Differentiate compiler and interpreter
21. Write short notes on buffer pair
PART - B

1. Describe the various phase of compiler and trace it with the program segment Position: = initial+ rate* 60. (10)
2. Draw the transition diagram for unsigned numbers (6)
3. Describe in details about the various compiler construction tools. (8)
4. Write briefly about input buffering. What is its role in lexical analysis? (8)
5. Give the minimized DFA for the following expressions (a/b)*abb (10)
6. Explain the various error recovery strategies (4)
7. Discuss the input buffering scheme in lexical analyser. (6)
8. Construct a NFA using thompson’s construction algorithm for the regular expression (a/b)* abb (a/b)* and convert it into DFA. (6)
9. Illustrate the compiler’s internal representation of the change in the source program, as translation progress by considering the translation of the statement A:= B+C*50. (8)
10. Construct a DFA directly from the regular expression (a/b)* abb without constructing NFA. (10)
11. Explain the need for dividing the compilation process into various phases and explain its functions. (8)
12. Explain how abstract stack machines can be used as translators (8)
13. What is syntax directed translation? How it is used for translation of expressions. (16)
14. Explain the phases of compiler. (12)
15. List the advantages of table management during the compilation. (4)
16. Describe any three compiler construction tools (12)
17. Find all the tokens and their types from the statement “while (count>=0) count--; sum+=count ;) in c language (4)
18. What is a compiler? Explain the various phases of compiler in detail (10)
19. Give a detailed note on the compiler-construction tools (6)
20. Construct a minimum state DFA for the regular expression (a/b)* abc (10)
21. Explain the role of lexical DFA (6)
22. Explain in detail the front end of the compiler. (10)
23. Draw and explain how the expression a=b+c*d will be converted into a object code. (6)
24. Discuss the purposes of error handler in compilation (16)
25. Describe compiler generators.
26. Explain the various phases of compiler in detail with neat sketch (16)
27. Explain with neat sketch implementation of lexical analyser(10)
28. Describe in detail about any three compiler construction tools (6)
UNIT II SYNTAX ANALYSIS

PART - A

1. What is predictive parser?
2. What is handle pruning?
3. Eliminate left recursion from the following grammar.
   \[ S \rightarrow (L)a; L \rightarrow L,S/S. \]
4. What is CLR?
5. Derive the string and construct a syntax tree for the input string ceaedbe using the grammar
   \[ S \rightarrow saA | A, A \rightarrow AbB | B, B \rightarrow cSd | e \]
6. List the factors to be considered for top-down parsing.
7. Construct the equivalent unambiguous grammar for the grammar
   \[ S \rightarrow aSbS | bSaS | e. \]
8. Write the rule to eliminate left recursion in a grammar.
9. What is meant by panic mode of error recovery?
10. What type of grammar is suitable for constructing top-down parser without duplicate entries in the parsing table?
11. Compare NFA and DFA based on time and space complexity.
12. What are the goals of error handler in a parser?
13. What is phrase level error recovery?
14. What are the disadvantages of operator precedence parsing?
15. How will you eliminate left recursion?
16. Define handle
17. Define canonical LR(1) items
18. What are handles?
19. List out the actions involved in shift-reduce parsing.
20. Eliminate left recursion for the given grammar
    \[ E \rightarrow E+T/T \\
    T \rightarrow T*F/F \\
    F \rightarrow (E)/id \]
21. Construct a parse tree of \((a+b)^*c\) for the grammar
    \[ E \rightarrow E+E / E^*E/(E)/id. \]

PART - B

1. Give the predictive parsing table for the following grammar
   \[ E \rightarrow E+T/T \]
T\rightarrow T*F/F
F\rightarrow (E) / id

Show the moves of the parser for the input (id+id)|*id. (10)

2. Explain the Role of parser in detail. (6)

3. Give the LR parsing table for the grammar
   E\rightarrow E+T/T
   T\rightarrow T*F/F
   F\rightarrow (E) / id

4. Give and explain operator precedence algorithm (6)

5. Give the LALR parsing table for the grammar
   S\rightarrow L=R/R
   L\rightarrow *R/ id
   r\rightarrow L

6. Explain the stack implementation of Shift-Reducing Parsing. (6)

7. Give the definitions for FIRST(X) and FOLLOW (A) procedures used in construction predictive Parser. (4)

8. What is an operator grammar? Draw the precedence graph for the following table (12)

9. Write a note on error recovery in predictive parsing (4)

10. Write the LR parsing algorithm. Check whether the grammar is SLR(1) or not. Justify the answer with reasons. (12)

11. Given the following grammar S\rightarrow AS|b, A\rightarrow SA|a Construct a SLR parsing table for the string
baab (16)

12. Consider the grammar E\rightarrow E+T, T\rightarrow T*F,F\rightarrow(E)|id. Using predictive parsing parse the string
id+id*id (16)

13. Generate operator precedence parsing table for the following grammar and parse the string
(a,a). (16)

14. Construct SLR parsing table for the grammar
   E\rightarrow E+T|T
   T\rightarrow TF|F (16)
F->F*|a/b

15. What are lexical and syntax errors? Describe the methods to recover from them. (8)

16. Write regular expressions for recognizing identifiers, constant, relational operators, and unary Operators with respect to c language. (8)

17. Write the algorithm to remove the left recursion from the given grammar. (8)

18. Write the algorithm for predictive parsing. (8)

19. Explain in detail about the error recovery strategies in parsing. (8)

20. Consider the grammar E-> E+E/E*/(E)/id. Show the sequence of moves made by the shift-reduces parser on the input id1 +id2 = id3 and determine whether the given string is accepted by the parser or not. (8)

21. Draw the NFA for recognizing the keywords, IF, WHILE, FOR, ELSE, SWITCH.

22. Write short notes on Lex.

23. How will you eliminate left recursion from a grammar G? Explain. (8)

24. Consider the following grammar S-> A a|bA ->Ac|Sd|e eliminate left recursion.

25. Draw the predictive parser table for the following grammar:
   G. S->iCtS | iCtSeS | a  C->b.


27. Explain the working model of top-down parsing and bottom up parsing?

28. Describe the error detection and recovery process involved in the lexical phases. (6)

29. Write the algorithm for predictive parser and parse the input expression

30. Performs LR parsing and derive the input a(a,a(a,a)) using the below given grammar

31. Perform shift reduce parsing for the input 2*(1+3) using the grammar

32. Write an algorithm for generating LR item sets and constructing SLR parsing table. (12)

33. Write about LALR Parser. (4)
UNIT III INTERMEDIATE CODE GENERATION

PART - A

1. Give the ways of representing three address statements
2. What is back patching?
3. Translate the expression a-(b+c) into three address code.
4. List out the three functions that are used to manipulate list of labels in back patching.
5. Why is it necessary to generate intermediate code instead of generating target program itself?
6. Define back patching.
7. Translate the arithmetic expression a*(b+c) into syntax tree.
8. List the advantage of DAG.
9. What is the necessity for carrying the position information in identifying the handle for reduction in a shift-reduce parser?
10. What is the advantage of LR(1) items over LR(0) items?

PART - B

1. Explain about the different type of three address statements (8)
2. Give the semantic rules for declarations in procedure (8)
3. Discuss various ways of implementing symbol table and compare the merits and demerits. (10)
4. Give the syntax directed translation for control statements. For any one example, use the Transaction to convert to three address codes. (6)
5. What are the various data structure used for symbol table construction and explain any one in detail. (8)
6. Let A be 10 x 20 array with low 1 = low 2=1. Let w =4. Draw an annotated parse tree for the assignment statement X:A[y,z]. Give the sequence of three address statement generated. (8)
7. How would you generate the intermediate code for the flow of control statements? Explain with examples. (16)
8. Explain in detail how three address code are generated and implemented. (16)
9. Explain the role of declaration statement in intermediate code generation. (16)
10. Write the translation scheme for Boolean expression using numerical representation scheme (8)
11. Generate three address code for the boolean expression a<b or c<d and e<f. (8)

12. Generate three address code for the following statement

   while a<b do
   begin
   if c<d then
   x := y+z
   else
   x := y-z
   end

13. Describe how compilers generate code for procedure calls. (8)

14. Construct predictive parsing table for the grammar

   B -> B or B | B and B | not B | (B) | true false

15. Construct SLR parsing table for the grammars

   A -> AS | aA |
   S -> a
   A -> a
   S -> a
   A -> a

16. For the operators given below, calculate the operator-precedence relations and operator-precedence function

   id, *, $

17. Explain the LR parsing algorithm in detail (8)

18. Construct a canonical parsing table for the grammar given below

   S -> CC
   C -> cC/d

UNIT IV CODE GENERATION

PART - A

1. What is constant folding?
2. What is DAG? What are applications of DAG in compiler?
3. What are the uses of register and address descriptors in code generation?
4. Define basic blocks.
5. What is three address code (TAC) and generates TAC for the statement:
   a = 2*b+3*C
6. What is SDT?
7. Define Directed Acyclic Graph (DAG)
8. Write short notes on global data flow analysis
9. Define back patching with an example
10. Give syntax directed translation for the following statement call p1 (int a, int b)
11. How can you find the leaders in basic block?

Part - B

1. Explain peephole optimization with example (8)
2. Explain the DAG representation of the basic block with an example (8)
3. What is three address code? What are its types? How it is implemented?
4. Construct the DAG for the following basic block
   \[ D := B \times C; \quad E := A + B; \quad B := B \times C; \quad A := E - D. \]
5. Design a simple code generator and explain with example. (16)
6. Write short notes on
   (i) Peephole optimization (ii) Issues in code generation
7. Explain the structure-preserving transformation on basic blocks with example. (8)
8. Discuss the characteristics of peephole optimization. (8)
9. Generate code for the assignment \( d := (a-b)+(a-c)+(a-c) \) using the code generator algorithm (8)
10. Construct the DAG for the basic block (8)
    \[
    d := b \times c \\
    e := a + b \\
    b := b \times c \\
    a := e - d. 
    \]
11. List all the types of three address statements. (8)
12. Describe the translation scheme for handling declaration with scope information.
13. Consider the statement "while(a>b or a>c) do if (p and q) a=a+m". describe the translation rules along with their semantic actions for generating the equivalent three address statement. (16)
14. Explain procedure calls with a neat example (8)
15. Describe the method of generating syntax-directed definition for control statements.
16. Derive the syntax directed translation schema for flow control statements. (16)
17. Discuss in detail how you design a symbol table (16)
18. Brief intermediate code generation for basic block, Control flow and Boolean expressions
19. Explain the data structure used for implementing symbol table (10)
20. Write about quadruple and Triple with its structure (6)
UNIT V CODE OPTIMIZATION AND RUN TIME ENVIRONMENTS

PART - A

1. What is code elimination?
2. What are the limitations of the static allocation?
3. What is flow graph?
4. What are the criteria used for the code improving transformations?
5. List the issues in code generation.
6. Write the steps for constructing leaders in basic blocks.
7. What are the issues in static allocation?
8. What is meant by copy-restore?
9. What is meant by optimizing compiler?
10. What is meant by constant folding and copy propagation?
11. What is the necessity for identifying the next use information about variables?
12. What is meant by bootstrapping of compilers?
13. What are the basic blocks?
14. What are the properties of optimizing compilers?
15. Define flow graphs.
16. What is the purpose of next use information in the code generation phases.
17. Optimize following the code by eliminating common sub expression:
   \[ y = A + B \cdot c \cdot x + C \cdot (x^2) + D \cdot (x^3) \]
18. What is peep-hole optimization.
19. Define code motion.
20. Define basic block and flow graph.

Part - B

1. Discuss about the code generation algorithm in detail \((10)\)
2. Suggest a suitable representation for symbol tables of procedure oriented languages \((6)\)
3. Explain the various issues in the design of code generator \((10)\)
4. Describe in detail about the stack allocation in memory management \((6)\)
5. Explain about Code Generation algorithm \((8)\)
6. Explain the various storage allocation strategies \((8)\)
7. Why do we need code optimization? Explain the principal sources of optimization \((16)\)
8. Explain with an example how basic blocks are optimized \((16)\)
9. Explain the storage allocation strategies used in run time environments \((16)\).
10. Explain the three techniques for loop optimization with example (12)
11. Discuss the three different storage allocation strategies. (16)
12. Explain all the sources of optimizations (16)
13. Explain all the issues associated with the code generation process. (16)
14. Explain various code optimization techniques in detail. (10)
15. Generate target code for the given program segment (6)

```c
main()
{
    int i,j;
    i=4;
    j=i+5;
}
```
16. Explain the various issues involved in the design of code generation.
17. Describe in detail about Run-time Storage Management (6)
18. Explain the various issues involved in the design of a code generator (16)
19. What is bootstrapping? Explain the approaches to compiler development.
20. Discuss about the following:
   i ) Copy of propagation, Dead Code elimination and code motion. (6)
21. Discuss in detail the principle source of optimization. (16)
22. Describe the issues in the design of code generator (8)
23. Describe peephole optimization (8)