Cycle-2

Unit – III

Short Questions:

1. Explain language acceptance of a PDA.
2. What is decidability?
3. Differentiate between acceptance by *final state* and *empty stack* in a PDA.
4. Differentiate between Deterministic and Non-Deterministic Turing Machines
5. Explain the concept of *instantaneous description* (ID) of a PDA.
6. Explain the role of the *tape* and *head* in a Turing Machine.
7. Explain "language acceptance" by a Turing Machine
8. Explain the concept of *undecidability* with example.
9. What types of languages are accepted by a PDA
10. Differentiate Tape and Head of a FA/PDA vs Tape and Head of Turing Machine

Long:

1. Construct a PDA that accepts the language $L = \{\{a, b\}^* \mid na = nb\}$.
2. Design a PDA to accept the following language L={0^n1^n | n>0}
3. Construct a PDA that accepts the language $L = \{a^n b^n | n \geq 1\}$
4. Construct a PDA from the grammar S□aAA , A□aS|bS|a
5. Construct a PDA, M equivalent to the following CFG S□ 0BB, B□0S|1S|0 test the string w=01000 in L(M)
6. Construct that L={$a^nb^nc^n$ | n>=1} is not a Context free language using pumping lemma
7. Explain the formal definition and working of a Turing Machine and Design Turing machine Instantaneous Description with example
8. Show with an example that there exists a language, which is recursively enumerable, but not recursive.
9. Describe the Halting Problem
10. Give examples of decidable and undecidable problems.
11. Give an example of an undecidable problem that is **Recursively Enumerable (RE)**. Explain carefully how it is RE but not decidable, with reference to the Halting Problem

UNIT-IV

1. What is input buffering and why is it needed.
2. Explain Lexical errors
3. Describe the primary role of a lexical analyzer in a compiler?
4. Explain Left recursion how to eliminate it
5. Differentiate between Interpreter and Compiler
6. Define Bootstrapping
7. Discuss about Ambiguous grammars
8. Differentiate between lexical analysis and syntax analysis.
9. Define token, lexeme, and pattern with examples

Long:

1. Construct LL1 parser and table for the following grammar

S□aAB|bA| ε, A□aAb| ε, B□bB| ε

2. Differentiate LL1 model and LR model with neat diagram
3. Show that the following grammar S□ Aa|bAc|Bc|bBa, A□d, B□d is LR(1) but not LALR(1)
4. Discuss the problems in Top-Down Parsing with examples
5. Write the rules to compute First and Follow for the given grammar
6. Construct the collection of LR item sets and parsing table for the given grammar E□E+T|T, T□TF|F, F□F*|a|b parse the input string w= a*b+a
7. Illustrate the following statement on all phases of the compiler
   position: =initial + rate * 60
8. Calculate First and Follow and parsing table for the given grammar and check the given string is parse successfully (w=id+id*id). E□TE' , E'□+TE'|ε, T□FT' , T'□*FT'| ε, F□(E)|id
9. Explain the role of lexical analyzer and issues of lexical analyzer in compiler design
10. Construct the collection of LR item sets and parsing table for the given grammar E□E+T|T, T□TF|F, F□F*|a|b parse the input string w= a*b+a

UNIT –V

1. What are the **different variants of syntax trees** used in intermediate-code generation?
2. Define **three-address code (TAC).**
3. Differentiate between synthesized and inherited attributes
4. What is a run-time environment?
5. What is a Syntax-Directed Translation Scheme?
6. What is stack allocation in runtime environments?
7. How does heap allocation differ from stack allocation?
8. How can a function access nonlocal variable?
9. Define Activation record briefly
10. Design the dependency graph with example

Long:

1. Explain what a Syntax-Directed Definition is. Describe how synthesized and inherited attributes are used in SDDs with example
2. Describe the organization of a typical run-time environment
3. Explain why heap management is required in programming languages that support dynamic memory allocation.
4. Explain the challenges and techniques for accessing Nonlocal Data on the Stack
5. Explain the concept of Three-Address Code. Describe its general format and different forms (quadruples, triples, indirect triples for the expression x:= -(a*b)+(c+d)-(a+b+c+d)
6. Define different storage organizations briefly
7. Define variants of syntax tree and its construction for the expression x*y-5+z
8. Generate the **three-address code** for the expression: a = b * (c + d) and draw the corresponding syntax tree.
9. Analyze the differences between **stack allocation** and **heap allocation** and give scenarios where each is preferable.
10. Discuss how **evaluation order** is determined for an SDD and why it is important in compiler design
11. Given a simple expression grammar, construct an **L-attributed SDD** for computing the value of expressions.