CS3501 Compiler Design

QUESTION BANK

## UNIT 1 INTRODUCTION TO COMPILERS & LEXICAL ANALYSIS

Introduction- Translators- Compilation and Interpretation-Language processors -The Phases of Compiler – Lexical Analysis– Role of Lexical Analyzer – Input Buffering – Specification of Tokens– Recognition of Tokens – Finite Automata – Regular Expressions to Automata NFA, DFA –Minimizing DFA – Language for Specifying Lexical Analyzers – Lex tool**.**

PART A

1. Define Compiler and what are the phases of compiler.
2. What are the functions performed in synthesis phase.
3. Differentiate analysis and synthesis phase.
4. Differentiate token,pattern,lexeme .
5. Define the following i) Preprocessor ii) Assembler iii) Loader and Linker
6. Define Lexical Analysis.
7. Rules of Minimizing DFA.
8. Structure of Lex tool.
9. Declaration of Lex tool.
10. Explain FA.
11. Define Input Buffering.
12. Role of LA.

PART B

1. What is Compiler? Design the Analysis and Synthesis Model of Compiler.
2. Write down the five properties of compiler.
3. What is translator? Write down the steps to execute a program.
4. Discuss all the phases of compiler with a with a diagram.
5. Write a short note on:
   a. YACC
   b. Pass
   c. Bootstrapping
   d. LEX Compiler
   e. Tokens, Patterns and Lexemes
6. Write the steps to convert Non-Deterministic Finite Automata (NDFA) into Deterministic Finite Automata (DFA).
7. Let M=({q0,q1}, {0,1}, $\delta$, q0, {q1}).
   Be NFA where $\delta$(q0,0)={q0,q1}, $\delta$(q1,1) = {q1}
   $\delta$(q1, 0)=$\varphi$, $\delta$(q1, 1)={q0, q1}
   Construct its equivalent DFA.

8. Convert the given NFA to DFA:

| Input/State | 0 | 1 |
|---|---|---|
| → q0 | {q0, q1} | q0 |
| q1 | q2 | q1 |
| q2 | q3 | q3 |
| q3 (final state) | $\varphi$ (null | q2 |

| | | |
|---|---|---|
| | character) | |

9. What is Regular Expression? Write the regular expression for:
    a. R=R1+R2 (Union operation)
    b. R=R1.R2 (concatenation Operation)
    c. R=R1* (Kleen Clouser)
    d. R=R+ (Positive Clouser)
    e. Write a regular expression for a language containing strings which end with “abb” over
       Σ= {a,b}.
    f. Construct a regular expression for the language containing all strings having anynumber of a’s and b’s except the null string.
10. Construct Deterministic Finite Automata to accept the regular expression :(0+1)* (00+11) (0+1)*

11. Derivation and Parse Tree:
    a. Let G be a Context Free Grammar for which the production Rules are given

       below: S -> aB|bA

       A -> a|aS|bAA

       B -> b|bS|aBB

Drive the string aaabbabbba using the above grammar (using Left Most Derivation and Right most Derivation).

## UNIT II SYNTAX ANALYSIS

Role of Parser – Grammars – Context-free grammars – Writing a grammar Top Down Parsing – General Strategies – Recursive Descent Parser Predictive Parser-LL(1) – Parser-Shift Reduce Parser-LR Parser- LR (0)Item Construction of SLR Parsing Table – Introduction to LALR Parser – Error Handling and Recovery in Syntax Analyzer-YACC tool – Design of a syntax Analyzer for a Sample Language**.**

### PART A

1. What is phrase level error recovery
2. Write down the necessary algorithms for FIRST and FOLLOW 4.
3. Explain the error recovery in predictive parsing
4. Define operator precedence grammer
5. Define augmented grammar Compare the LR Parsers.
6. Compare and contrast LR and LL Parsers Differentiate between top down parsers
7. Define Dead code elimination?
8. Mention the types of LR parser
9. Explain bottom up parsing method

### PART B

1. Explain the parsing techniques with a hierarchical diagram.
2. What are the problems associated with Top Down Parsing?
3. Write the production rules to eliminate the left recursion and left factoring problems.
4. Consider the following

   Grammar: A-> ABd|Aa|a

   B-> Be|b

   Remove left recursion.

5. Do left factoring in the following

   grammar: A-> aAB|aA|a

   B-> bB|b

6. Write a short note on:
   a. Ambiguity (with example)
   b. Recursive Descent Parser
   c. Predictive LL(1) parser (working)
   d. Handle pruning
   e. Operator Precedence Parser
7. Write Rules to construct FIRST Function and FOLLOW Function.
8. Consider Grammar:

   E-> E+T|T

   T-> T*F|F

F-> (E)|id

9. Write the algorithm to create Predictive parsing table with the scanning of input string.
10. Show the following

    Grammar: S->

    AaAb|BbBa

    A-> €

    B-> €

    Is LL(1) and parse the input string "ba".

11. Consider the grammar:

    E-> E+E

    E-> E*E

    E->id

    Perform shift reduce parsing of the input string "id1+id2+id3".

12. Write the properties of LR parser with its structure. Also explain the techniques of LR parser.
13. Write a short note on:
    a. Augmented grammar
    b. Kernel items
    c. Rules of closure operation and goto operation
    d. Rules to construct the LR(0) items
14. Consider the following

    grammar: S->

    Aa|bAc|Bc|bBa

    A-> d

    B-> d

    Compute closure and goto.

15. Write the rules to construct the SLR parsing table.
16. Consider the following

    grammar: E-> E+T|T

    T-> TF|F

    F-> F*|a|b

Construct the SLR parsing table and also parse the input “a*b+a”

17. Write the rules to construct the LR(1) items.

18. What is LALR parser? Construct the set of LR(1) items for this

    grammar: S-> CC

    C-> aC

    C->d

19. Show the following

    grammar S-

    >Aa|bAc|Bc|bBa

    A->d

    B->d

    Is LR(1) but not LALR(1).

20. Write the comparison among SLR Parser, LALR parser and Canonical LR Parser.

## UNIT III SYNTAX DIRECTED TRANSLATION & INTERMEDIATE CODE GENERATION

Syntax directed Definitions-Construction of Syntax Tree-Bottom-up Evaluation of S-Attribute Definitions- Design of predictive translator – Type Systems-Specification of a simple type Checker Equivalence of Type Expressions-Type Conversions. Intermediate Languages: Syntax Tree, Three Address Code, Types and Declarations, Translation of Expressions, Type Checking, Back patching.

### PART A

1. Define Type Equivalence
2. Explain the role of intermediate code generator in compilation process
3. Define left most derivation and right most derivation with example
4. What are the various types of intermediate code representation?
5. Write a note on the specification of a simple type checker.
6. Explain intermediate code representations?
7. Define type expression with an example?
8. State general activation record?
9. Explain type expression and type systems

### PART B

1. What is syntax directed translation (SDD)?
2. Write short note on:
    a. Synthesized attributes
    b. Inherited attributes
    c. Dependency graph
    d. Evaluation order
    e. Directed Acyclic Graph (DAG)
3. Draw the syntax tree and DAG for the following

    expression:(a*b)+(c-d)*(a*b)+b

4. Differentiate between synthesized translation and inherited translation.
5. What is intermediate code and write the two benefits of intermediate code generation.
6. Write the short note on:
    a. Abstract syntax tree
    b. Polish notation
    c. Three address code
    d. Backpatching
7. Construct syntax tree and postfix notation for the following expression:

(a+(b*c)^d-e/(f+g)

8. Write quadruples, triples and indirect triples for the expression:

   -(a*b)+(c+d)-(a+b+c+d)

9. Write the three address statement with example for:
   a. Assignment
   b. Unconditional jump (goto)
   c. Array statement (2D and 3D)
   d. Boolean expression
   e. If-then-else statement
   f. While, do-while statement
   g. Switch case statement

## UNIT IV RUN-TIME ENVIRONMENT AND CODE GENERATION

Runtime Environments – source language issues – Storage organization – Storage Allocation Strategies: Static, Stack and Heap allocation – Parameter Passing-Symbol Tables – Dynamic Storage Allocation – Issues in the Design of a code generator – Basic Blocks and Flow graphs – Design of a simple Code Generator – Optimal Code Generation for Expressions– Dynamic Programming Code Generation**.**

### PART A

1. Write the quadruple for the following expression $(x + y)*(y + z) + (x + y + z)$ .
2. What is a DAG? Mention its applications.
3. What are Abstract Syntax trees?
4. Define address descriptor and register descriptor
5. Discuss about common sub expression elimination
6. What is a Flow graph? Define constant folding?
7. Define reduction in strength?
8. Explain Lazy-code motion problem with an algorithm

### PART B

1. Write the definition of symbol table and procedure to store the names in symbol table.
2. What are the data structures used in symbol table?
3. What are the limitations of stack allocation?
4. Write two important points about heap management.
5. Write the comparison among Static allocation, Stack allocation and Heap Allocation with theirmerits and limitations.
6. What is activation record? Write the various fields of Activation Record.
7. What are the functions of error handler?
8. Write a short note on Error Detection and Recovery.
9. Classify the errors and discuss the errors in each phase of Compiler.
10. Illustrate loop optimization with suitable example.
11. Explain various code optimization techniques in detai

## UNIT V CODE OPTIMIZATION

Principal Sources of Optimization – Peep-hole optimization – DAG- Optimization of Basic Blocks – Global Data Flow Analysis – Efficient Data Flow Algorithm – Recent trends in Compiler Design.

### PART A

1. What are the induction variables?
2. Explain about code motion.
3. What are induction variables?
4. What is induction variable elimination?
5. What is machine independent code optimization?
6. Write a short note on copy Propagation
7. What are the induction variables?
8. Write a short note on Flow graph.

### PART B

1. What are the properties of code generation phase? Also explain the Design Issues of this phase.
2. What are basic blocks? Write the algorithm for partitioning into Blocks.
3. Write a short note on:
    a. Flow graph (with example)
    b. Dominators
    c. Natural loops
    d. Inner loops
    e. Reducible flow graphs

4. Consider the following program

```
code:Prod=0;

I=1;

Do{

Prod=prod+a[i]*b[

i];I=i+1;

}while (i<=10);
```

   a. Partition in into blocks
   b. Construct the flow graph
5. What is code optimization? Explain machine dependent and independent code optimization.
6. What is common sub-expression and how to eliminate it? Explain with example.
7. Write a short note with example to optimize the code:
   a. Dead code elimination
   b. Variable elimination
   c. Code motion
   d. Reduction in strength
8. What is control and data flow analysis? Explain with example.