CD – M1 – T1 – KPK – Section 10 Module Bank

1.

A. Consider the different phases involved in developing a machine understandable code for a C program that verifies whether the sum of the cubes of its individual digits is equal to the original number? Describe each phase's role in transforming the C source code into executable code.

i. Analyze the outputs of the analysis phases in the compilation process.
ii. Create optimized intermediate code from the abstract syntax tree.
iii. Develop the final machine code from the optimized intermediate representation.

Introduction - Compiler Phases	Understand, Analyze	CO1, CO4	PO1, PO2, PO3, PO4

B. Formulate a parsing table for the simple LL(1) grammar whose rules are:

$S' \rightarrow S\#$
$S \rightarrow xA \mid y \mid zB \mid w$
$A \rightarrow xA \mid y$
$B \rightarrow zBd$

Using this parsing table, give a trace of the parse for each of the inputs:

xxxy#, zzww#

Syntax Analysis - LL(1) Grammar	Apply, Analyze	CO2	PO1, PO2, PO4, PO5

2.

A. Analyze the concept of regular expressions and how they are used in lexical analysis.
Provide the regular expressions for
i. Identifying common language constructs like identifiers, literals, and keywords.
ii. Avoiding the comments and white spaces in the program.

Lexical Analysis - Regular Expressions	Understand, Apply	CO1	PO1, PO2, PO4, PO5

B.
Consider the Grammar G: $S \rightarrow m \mid mn \mid mnp \mid mnpr$.

i. Develop the supportive grammar for TDP.
ii. Construct the RDP for the above G.

Syntax Analysis - Grammar for TDP/RDP	Remember, Apply and Evaluate	CO2	PO1, PO2, PO3, PO5

3.

A. How does the occurrence of invalid tokens and misspelled keywords impact the lexical analysis phase of a compiler? Explain the role of lexical error recovery techniques in ensuring smooth transitions to subsequent compilation phases. Additionally, provide an example of a program that multiplies two numbers but includes lexical errors such as misspelled keywords (e.g., innt instead of int) and illegal identifiers. Perform lexical analysis on this code, identify the errors, and rewrite the corrected version of the program.

Lexical Analysis - Error Handling	Understand, Analyze	CO1	PO1, PO2, PO4

B. Describe the production rules for the "Nested If Else" Statement of C.

i. Design CFG

ii. Construct LL(k) parse table

iii. Verify the string "if (E) S E" is acceptable or not.

Syntax Analysis - Nested If Else	Analyze, Evaluate	CO2	PO1, PO2, PO4, PO5

4.

A. Count the number of tokens in the following C code snippets and classify each token by its type (keyword, identifier, operator, literal, delimiter, etc.) and construct the Symbol and litral table for the following :

```
i. int main() {
       int a = 10, b = 20;
      int max = (a > b) ? a : b;
   }
ii. Result = a * b + (c / d) + (a / b);
iii. #define SIZE 100
     int arr[SIZE];
     arr[0] = 10;
```

Lexical Analysis - Tokens & Symbol Table	Understand, Analyze	CO1	PO1, PO2, PO4

B. **Consider the Grammar G:**

stmt → if (expr) stmt else stmt | while (expr) stmt | { stmt_list } | id = expr ;

stmt_list → stmt stmt_list | ε

expr → expr + term | term

term → id | num | (expr)

i. Transform the grammar to remove ambiguity and make it suitable for top-down parsing.

ii. Construct the LR(0) parsing table.

iii. Parse the string if(id)id = id + num ; else { id = id ; } and explain how the conflicts are resolved.

Syntax Analysis - LR(0) Parsing	Apply and Evaluate	CO2	PO1, PO2, PO3, PO5

5.

A. **Design a simple input buffering mechanism for a C program that sorts an array of integers using the Bubble Sort algorithm.** Describe the data structures and buffering logic used for reading source code and handling tokens efficiently.

i. Implement the input buffering mechanism for the Bubble Sort logic in C.

ii. Develop a symbol table that can store and categorize tokens such as keywords, user-defined function names, loop constructs, integer constants, and logical operators used in the above program.

Symbol Table Management	Apply	CO1	PO1, PO2, PO4, PO5

B. Design a context-free grammar (CFG) for arithmetic expressions involving integer literals, the addition operator (+), and parentheses. Your grammar should be able to generate expressions such as the following:

- 25
- (24 + 25)
- 22 * (24 + 25)
- (21 + (22 + 23))

i. Write the CFG in a form suitable for LL(1) parsing.

ii. Implement a simple LL(1) parser for the grammar you designed.

iii. Test your parser with valid and invalid expressions to demonstrate its correctness.

Syntax Analysis - CFG for Arithmetic	Apply, Evaluate	CO2	PO1, PO2, PO3, PO5

6.

Design an input buffering mechanism for a C program that performs matrix multiplication. Describe how the buffer handles multi-dimensional array inputs, nested loops, and identifies operator precedence and nested brackets during lexical scanning.

i. Implement a buffered input system in C that reads two 2D matrices and performs multiplication, tracking positions of arithmetic and bracketed expressions.

ii. Construct a symbol table that classifies matrix variables, loop variables, arithmetic operators (*, +), brackets ([,], (,)), and invalid array access patterns (e.g., wrong indices or undeclared variables)

Compiler Phases - Machine Code	Understand, Analyze and Evaluate	CO1, CO4	PO1, PO2, PO3, PO4, PO5

B. Consider the Grammar G: S → Sa | ε | bB | bD

i. Develop the supportive Grammar for TDP.

ii. Evaluate the G.

iii. Construct the LL(k) parse table

iv. Construct the parse tree for one acceptable input string according to the parsing actions.

Syntax Analysis - LL(k) Parsing	Understand, Evaluate	CO2	PO1, PO2, PO4, PO5

7.

A. Consider the following Context-Free Grammar (CFG) for a simple programming language.

<program> → <statement_list>
<statement_list> → <statement> | <statement_list> <statement>
<statement> → <assignment> | <if_statement> | <loop_statement>
<assignment> → <identifier> "=" <expression> ";"
<if_statement> → "if" "(" <condition> ")" <statement> "else" <statement>
<loop_statement> → "while" "(" <condition> ")" <statement>
<expression> → <term> | <expression> "+" <term> | <expression> "-" <term>
<term> → <factor> | <term> "*" <factor> | <term> "/" <factor>
<factor> → <identifier> | <number> | "(" <expression> ")"
<condition> → <expression> "<" <expression> | <expression> ">" <expression> | <expression>"==" <expression>.

i. Compute the First(A) and Follow(A) sets.
ii. Construct the parse tree for the “a = b + c”
iii. Construct the annotated Parse tree for evaluation of
“x = 4 * 4 + 5 * 5 – 50”

Syntax Analysis – CFG & Parse Trees	Understand, Apply	CO2	PO1, PO2, PO3, PO5

B. **Consider the different phases involved in converting a C program that performs matrix multiplication.** Explain how each phase of the compiler—lexical analysis, syntax analysis, semantic analysis, intermediate code generation, and code optimization contributes to transforming the high-level code into intermediate representation. Discuss how nested loops, array indexing, and arithmetic expressions are processed and optimized in the intermediate code.

Compiler Phases - Intermediate Code	Understand, Apply	CO1, CO4	PO1, PO2, PO3

8.

A. Consider the following code snippet in a programming language similar to C:

```
int main() {
 int a = 5;
 if (a > 0) {
        a = a * 2;
}
return 0; }
```

i. Perform lexical analysis on the given code and generate a token stream.
ii. Construct the symbol tanle for the given code.

Lexical Analysis - Tokens & Symbol Table	Apply, Analyze	CO1	PO1, PO2, PO4

B. Consider the G:

$S \rightarrow A\ B\ \ ; A \rightarrow p\ A \mid q\ ; B \rightarrow r\ B\ s \mid t$

i. Create the LL(1) parsing table for the given grammar.
ii. Apply the parsing table to determine whether the input string "ppqrrtss" can be successfully parsed. Apply, Analyze

Syntax Analysis - LL(1) Parsing	Apply, Analyze	CO2	PO1, PO2, PO3, PO5

9.

A. How does the lexical analyzer handle the removal of comments, where comments are defined as follows:
i. A comment begins with // and includes all characters until the end of that line.
ii. A comment begins with / and includes all characters through the next occurrence of the character sequence /.

Lexical Analysis - Comment Removal	Understand, Analyze	CO1	PO1, PO2, PO4

B. **Consider the Grammar G:**

S → if E then S else S
  | if E then S
  | stmt
E → id
stmt → id = id ;

i. Construct the **SLR(1) parsing table**, and address the ambiguity in nested if-else.
ii. Parse the input string:
if id then if id then id = id ; else id = id ;

Syntax Analysis - SLR(0) Parsing	Apply, Evaluate	CO2	PO1, PO2, PO3, PO5

10.

A. Consider a programming language that allows variable names to contain alphanumeric characters and underscores, starting with a letter. Design regular expressions for the language's identifiers and literals. Also, identify any reserved keywords and explain how they are recognized and stored by the lexical analyser.

Lexical Analysis – Identifiers/Keywords	Understand, Analyze	CO1	PO1, PO2, PO4

B. Consider the CFG:

S → Aa | Bb
A → Ac | d
B → Bd | e

i. Construct the **LR(0) parsing table**
ii. Construct the **SLR parsing table**
iii. Identify and explain any conflicts that arise in both tables
iv. Compare the parsing capabilities and conflict resolution mechanisms of **LR(0)** and **SLR** parsers
v. Highlight the **limitations of LR(0)** parsing and describe how SLR parsing improves upon it.

Syntax Analysis - LR/SLR Comparison	Understand, Analyze	CO2	PO1, PO2, PO3, PO5

a