📘 Compiler Design – UNIT II

🔥 MOST IMPORTANT 14-MARK QUESTIONS

🎯 Easy + Detailed + RGPV Exam Writing Style

🔥 Q1. Explain Syntax Analysis with Diagram.

✍️ Answer:

📌 Introduction

Syntax Analysis is the second phase of a compiler.
It checks whether the sequence of tokens follows the grammar rules of the programming language.

Syntax Analyzer is also called:

```
Parser
```

🌟 Diagram

```
Tokens
   ↓
Syntax Analyzer
   ↓
Parse Tree
```

🌟 Functions of Syntax Analyzer

✅ Checks grammatical correctness

✅ Generates parse tree

✅ Detects syntax errors

✅ Sends information to semantic analyzer

🌟 Example

Correct statement:

```
int a;
```

Incorrect statement:

```
int ;
```

Parser reports syntax error.

🌟 Advantages

✅ Detects syntax errors

✅ Helps in program structure checking

✅ Simplifies semantic analysis

📌 Conclusion

Syntax Analysis is an important compiler phase that checks program structure according to grammar rules.

🔥 Q2. Explain CFG with Parse Tree.

✍️ Answer:

📌 Definition

CFG (Context Free Grammar) is used to describe the syntax of a programming language.

🌟 Components of CFG

Symbol	Meaning
V	Variables
T	Terminals
P	Production Rules
S	Start Symbol

🌟 Example of CFG

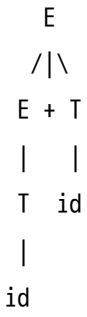
```
E → E + T
E → T
T → id
```

🌟 Parse Tree

For:

```
id + id
```

Parse Tree:



🌟 Advantages

✅ Defines language syntax

✅ Used in parser design

✅ Detects syntax errors

📌 Conclusion

CFG helps in defining the grammatical structure of programming languages.

🔥 Q3. Explain Top Down Parsing.

✍️ Answer:

📌 Definition

Top Down Parsing constructs the parse tree from:

```
Root → Leaf
```

It starts from the start symbol and tries to derive the input string.

🌟 Working

```
Start Symbol
      ↓
Production Rules
      ↓
Input String
```

🌟 Example

Grammar:

```
S → aAB
A → b
B → c
```

Input:

```
abc
```

🌟 Advantages

✅ Easy to implement
✅ Simple parsing technique

🌟 Disadvantages

❌ Backtracking problem
❌ Cannot handle left recursion

📌 Conclusion

Top Down Parsing is a simple parsing technique used to generate parse trees from root to leaf.

🔥 Q4. Explain Recursive Descent Parsing.

✍️ Answer:

📌 Definition

Recursive Descent Parsing is a top down parsing technique in which:

```
Each non-terminal has a recursive function
```

🌟 Example Grammar

```
E → T + E
E → T
T → id
```

🌟 Recursive Function

```
void E()
{
   T();
}
```

🌟 Advantages

✅ Easy coding

✅ Suitable for small grammars

🌟 Disadvantages

❌ Backtracking issue

❌ Cannot handle left recursive grammar

📌 Conclusion

Recursive Descent Parsing uses recursive procedures for parsing grammar rules.

🔥 Q5. Explain Predictive Parsing.

✍️ Answer:

📌 Definition

Predictive Parsing is a top down parsing technique that does not use backtracking.

It predicts the correct production rule using:

- FIRST set
- FOLLOW set

🌟 Diagram

```
Input String
      ↓
Predictive Parser
      ↓
Parse Tree
```

🌟 Parsing Table

Non-Terminal	Input Symbol	Production
E	id	E → TE'

🌟 Advantages

✅ Fast parsing
✅ No backtracking
✅ Efficient parsing

🌟 Disadvantages

❌ Cannot handle left recursive grammar

📌 Conclusion

Predictive Parsing improves parsing efficiency by eliminating backtracking.

🔥 Q6. Explain Bottom Up Parsing.

✍️ Answer:

📌 Definition

Bottom Up Parsing constructs parse tree from:

```
Leaf → Root
```

It reduces the input string to the start symbol.

🌟 Working

```
Input String
      ↓
Reduction Process
      ↓
Start Symbol
```

🌟 Shift Reduce Operations

Operation	Meaning
Shift	Push input into stack
Reduce	Apply grammar rule

Operation	Meaning
Accept	Successful parsing
Error	Invalid string

🌟 Advantages

✅ Efficient parsing

✅ Handles large grammars

🌟 Disadvantages

❌ Complex implementation

📌 Conclusion

Bottom Up Parsing reduces input strings into start symbols using grammar rules.

🔥 Q7. Explain Operator Precedence Parser.

✍️ Answer:

📌 Definition

Operator Precedence Parser uses operator priority and associativity for parsing expressions.

🌟 Example

Expression:

```
a + b * c
```

First evaluated:

```
b * c
```

Then:

```
a + result
```

🌟 Working

Uses precedence table to determine:

- Shift
- Reduce operations

🌟 Advantages

✅ Fast expression parsing
✅ Easy implementation

🌟 Disadvantages

❌ Cannot handle all grammars

📌 Conclusion

Operator Precedence Parser is mainly used for arithmetic expression parsing.

🔥 Q8. Explain LR Parser and its Types.

✍️ Answer:

📌 Definition

LR Parser is the most powerful bottom up parser.

🌟 Meaning of LR

Symbol	Meaning
L	Left to Right scanning
R	Rightmost derivation

🌟 Types of LR Parser

Parser	Full Form
SLR	Simple LR
LALR	Look Ahead LR
CLR	Canonical LR

🌟 SLR Parser

Simple version of LR parser.

Features:

- Small parsing table
- Easy implementation

🌟 LALR Parser

Most commonly used parser.

Features:

- Smaller table than CLR
- More powerful than SLR

🌟 CLR Parser

Most powerful LR parser.

Features:

- Large parsing table
- Better error handling

🌟 Advantages

✅ Efficient parsing

✅ Handles large grammars

✅ Detects syntax errors

🌟 Disadvantages

❌ Complex design

📌 Conclusion

LR parsers are powerful bottom up parsers used in compiler construction.

🔥 Q9. Explain Syntax Directed Definition (SDD).

✍️ Answer:

📌 Definition

Syntax Directed Definition combines:

```
Grammar + Semantic Rules
```

🌟 Uses

✅ Syntax tree construction

✅ Type checking

✅ Translation process

🌟 Example

```
E → E1 + T
E.val = E1.val + T.val
```

🌟 Advantages

✅ Supports semantic analysis

✅ Easy translation

📌 Conclusion

SDD is used for syntax directed translation and semantic processing.

🔥 Q10. Explain S-Attributed and L-Attributed Definition.

✍️ Answer:

🌟 S-Attributed Definition

📌 Definition

Uses only:

```
Synthesized Attributes
```

🌟 Features

✅ Used in bottom up parsing

✅ Easy evaluation

🌟 Example

```
E → E1 + T
E.val = E1.val + T.val
```

🌟 L-Attributed Definition

📌 Definition

Uses:

```
Inherited + Synthesized Attributes
```

🌟 Features

✅ Used in top down parsing

✅ More flexible

🌟 Difference

S-Attributed	L-Attributed
Uses synthesized attributes only	Uses inherited and synthesized attributes
Bottom up parsing	Top down parsing

📌 Conclusion

S-attributed and L-attributed definitions are used for semantic rule evaluation.

🔥 Q11. Explain Syntax Tree Construction.

✍️ Answer:

📌 Definition

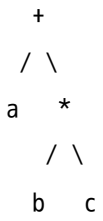
A Syntax Tree is a tree representation of program structure.

🌟 Example

Expression:

```
a + b * c
```

🌟 Syntax Tree



🌟 Construction Steps

1. Identify operators
2. Create tree nodes
3. Arrange according to precedence

🌟 Advantages

✅ Easy expression representation

✅ Used in semantic analysis

✅ Helps code generation

📌 Conclusion

Syntax Tree represents expressions in hierarchical tree form for compiler processing.