



SYNTAX ANALYSER

Under the guidance of
Dr. J Jeyasudha

By
Mahi Prasad(RA2011033010112)
Garima Goel(RA2011033010133)



INTRODUCTION TO SYNTAX ANALYSER

- A syntax analyzer, also known as a parser, is a key component of a compiler or interpreter.
- Its primary purpose is to analyze the syntactic structure of the source code written in a programming language and determine whether it conforms to the grammar rules defined by the language.
- The syntax analyzer takes the stream of tokens generated by the lexical analyzer as input and verifies whether the sequence of tokens forms valid statements or expressions according to the language grammar.
- It ensures that the code is well-formed and adheres to the syntactic rules specified by the language specification.
- The process of syntactic analysis involves creating a parse tree or syntax tree, which represents the hierarchical structure of the code. This tree provides a structural representation of the program, showing how different language constructs and expressions are related to each other.



PROBLEM STATEMENT

The problem statement of a syntax analyzer, or parser, is to analyze the syntactic structure of the source code and determine whether it conforms to the grammar rules defined by the programming language.

- Token Recognition: The parser needs to recognize individual tokens in the input source code.
- Error Detection and Reporting: The syntax analyzer is responsible for identifying syntax errors in the code. It detects situations where the input code violates the grammar rules or contains syntax-related issues
- Parse Tree or Abstract Syntax Tree (AST) Construction: As part of the analysis, the parser constructs a parse tree or an abstract syntax tree (AST) that represents the hierarchical structure of the code


By solving these problem, the syntax analyzer ensures the syntactic correctness and structural understanding of the source code, facilitating subsequent phases in the compilation process.



SOLUTION OVERVIEW

To address the problem statement of a syntax analyzer, the following solution overview outlines the key components and steps involved:

- **Tokenization:** The first step is to tokenize the source code using a lexical analyzer (lexer). The lexer scans the input code and identifies individual tokens based on predefined rules.
- **Grammar Specification:** Define the grammar rules for the programming language using a formal notation
- **Parsing Algorithms:** Implement parsing algorithms to analyze the token stream and validate it against the grammar rules. Popular parsing algorithms include Recursive Descent, LL(1), LR(1), LALR(1), and more

- 
- Parse Tree or AST Construction: As the parser processes the tokens, it constructs a parse tree or AST that represents the hierarchical structure of the code.
 - Error Detection and Reporting: During parsing, the syntax analyzer identifies syntax errors by detecting violations of the grammar rules. When an error is encountered, the parser generates appropriate error messages that provide information about the location and nature of the error.
 - Abstract Syntax Tree (AST) Generation: Optionally, transform the parse tree into an abstract syntax tree (AST) to remove irrelevant syntactic details.
 - Intermediate Representation: Provide the resulting parse tree or AST as an intermediate representation for subsequent compiler phases, such as semantic analysis or code generation.



KEY OBJECTIVES

- **Syntactic Validity:** The primary objective is to determine whether the source code conforms to the grammar rules of the programming language
- **Error Detection and Reporting:** The syntax analyzer aims to identify and report syntax errors in the code. It detects violations of the grammar rules and generates meaningful error messages that pinpoint the location and nature of the errors.
- **Structural Representation:** Another objective is to construct a parse tree or abstract syntax tree (AST) that represents the hierarchical structure of the code.
- **Parsing Efficiency:** The syntax analyzer aims to parse the code efficiently, minimizing the time and resources required for the analysis.
- **Error Recovery:** In the event of encountering syntax errors, the syntax analyzer may attempt to recover from those errors and continue parsing the remaining code.