Explain formal grammar and its application to syntax analyzer.

1. Formal Grammar and its Application to Syntax Analyzer:
    - Formal grammar is like a set of rules that define how sentences in a language are constructed. In the context of programming languages, it specifies how valid code should be written using production rules.
    - Think of it like a recipe. Just as a recipe tells you the steps to make a dish, formal grammar tells you the rules to write code.
    - Syntax analyzer, a part of a compiler, checks if the code follows these rules correctly.
2. Syntax Analyzer Checks Syntax:
    - Syntax analyzer is like a grammar checker for code.
    - It makes sure that the code is written in a way that the programming language understands.
3. Grouping Tokens:
    - Before checking the code, the syntax analyzer takes individual parts of the code (like words in a sentence) called tokens from the lexical analyzer.
    - Then it groups these tokens together in a way that makes sense for the programming language’s structure.
4. Generating Syntactic Error:
    - If the syntax analyzer can’t make sense of the grouped tokens based on the rules defined by the formal grammar, it signals a syntactic error.
    - It’s like when a sentence doesn’t make sense in human language; the grammar checker highlights the mistake.
5. Syntax Checking:
    - The whole process of the syntax analyzer checking if the code follows the rules of the programming language is called syntax checking.
6. Checking in Compiler with Specifications:
    - When we talk about checking syntax in a compiler, it means that the compiler

uses the formal grammar rules specified for the programming language.
- These rules, or specifications, guide the compiler on how to understand the code’s structure.

7. Specifications Tell the Compiler How Syntax Should Be:
    - Specifications are like instructions for the compiler, explaining how the programming language’s syntax should look.
    - They define the patterns and structures that are allowed in the language.

Related posts:

1. What are the types of passes in compiler ?
2. Discuss the role of compiler writing tools. Describe various compiler writing tools.
3. What do you mean by regular expression ? Write the formal recursive definition of a regular expression.
4. How does finite automata useful for lexical analysis ?
5. Explain the implementation of lexical analyzer.
6. Write short notes on lexical analyzer generator.
7. Explain the automatic generation of lexical analyzer.
8. Explain the term token, lexeme and pattern.
9. What are the various LEX actions that are used in LEX programming ?
10. Describe grammar.
11. Define parse tree. What are the conditions for constructing a parse tree from a CFG ?
12. Describe the capabilities of CFG.
13. What is parser ? Write the role of parser. What are the most popular parsing techniques ? OR Explain about basic parsing techniques. What is top-down parsing ? Explain in detail.
14. What are the common conflicts that can be encountered in shift-reduce parser ?

15. Differentiate between top-down and bottom-up parser.Under which conditions predictive parsing can be constructed for a grammar ?
16. Differentiate between recursive descent parsing and predictive parsing.
17. What is the difference between S-attributed and L-attributed definitions ?
18. What is intermediate code generation and discuss benefits of intermediate code ?
19. Define parse tree. Why parse tree construction is only possible for CFG ?
20. Discuss symbol table with its capabilities ?
21. What are the symbol table requirements ? What are the demerits in the uniform structure of symbol table ?