Explain the implementation of lexical analyzer.

Lexical analyzer can be implemented in following step :

1. Input: The lexical analyzer takes the source code of a program as input.
2. Scanning: It reads the source code character by character, typically using a technique called input buffering. This means it doesn’t process the code immediately but rather collects a bunch of characters before analyzing them.
3. Regular Expressions: These are like special search patterns that define the structure of tokens (like keywords, identifiers, operators, etc.) in the programming language.
4. NFA (Nondeterministic Finite Automaton): Regular expressions are converted into NFAs. Think of NFAs as theoretical machines that can be in multiple states at once.

5. DFA (Deterministic Finite Automaton): NFAs are converted into DFAs, which are simpler versions of NFAs where each input leads to only one possible state. Then these DFAs are minimized to make them more efficient.

6. Recognition and Lexemes: The minimized DFA is used to recognize patterns in the source

code. When a pattern is recognized, it breaks it down into smaller units called lexemes. For example, in the code `int x = 5;`, `int` would be a lexeme recognized as a keyword, `x` would be recognized as an identifier, `=` as an operator, and `5` as a constant.

7. Evaluation Phases: Each recognized lexeme is associated with a phase in the programming language's grammar. This phase evaluates what the lexeme represents in the program. For example, if `int` is recognized, it indicates a declaration of an integer variable.

8. Constructing State Table and Generating Code: Finally, the tool creates a state table based on the DFA and generates program code. This code includes the state table, the evaluation phases for recognized lexemes, and routines to use them appropriately.

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. Write short notes on lexical analyzer generator.
6. Explain the automatic generation of lexical analyzer.
7. Explain the term token, lexeme and pattern.
8. What are the various LEX actions that are used in LEX programming ?
9. Describe grammar.
10. Explain formal grammar and its application to syntax analyzer.
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 ?