IFET Autonomous
College of Engineering
Permenantly Affiliated to Anna University|Approved by AICTE

ACADEMIC YEAR: 2025-26
REGULATION: 2023

DEPARTMENT OF CSE & IT
FUNDAMENTALS

Subject Code/Name : 23CS6401/ 23IT6401/ COMPILER DESIGN
YEAR / SEM : III / VI

UNIT I– INTRODUCTION TO COMPILER DESIGN

Compilers – Analysis of the Source Program – Phases of a Compiler- Types of Compiler-Role of Lexical Analyzer – Input Buffering – Specification of Tokens – Recognition of Tokens -Finite Automata – Regular Expressions to Automata – Minimizing DFA. Language for Specifying Lexical Analyzers - Design of Lexical analyzer generator(LEX) - Recent trends in Compiler Design.

Compilers

Compiler is a translator which is used to convert programs in high-level language to low-level language. It translates the entire program and also reports the errors in source program encountered during the translation.

Analysis of the Source Program

1) Linear analysis: In this type of analysis the source string is read from left to right and grouped into tokens.

2) Hierarchical analysis: In this analysis, characters or tokens are grouped hierarchically into nested collections for checking them syntactically.

3) Semantic analysis: This kind of analysis ensures the correctness of meaning of the program.

Phases of a Compiler

A compiler operates in various phases. A phase is a logically interrelated operation that takes source program in one representation and produces output in another representation. There are two phases of compilation.

- Analysis (Machine Independent/Language Dependent)
- Synthesis (Machine Dependent/Language Independent) The different phases of compiler are as follows,
    1. Lexical analysis
    2. Syntax analysis
    3. Semantic analysis
    4. Intermediate code generation
    5. Code generation
    6. Code optimization

Types of Compiler

1. Incremental Compiler

- Incremental compiler is a compiler which performs the recompilation of only modified source rather than compiling the whole source program.

2. Cross Compiler

- Source language i.e. the application program.
- Target language in which machine code is written.

- The Implementation language in which a compiler is written.

Role of Lexical Analyzer

The main task of lexical analyzer is to read the source program, scan the input characters, group them into lexemes and produce the token as output.

Processes of lexical analyzer

Scanning: It performs reading of input characters, removal of white spaces and comments

Tokenization: It is the more complex portion, where the scanner produces the sequence of tokens as output

Input Buffering

- A two-buffer input scheme that is useful when lookahead on the input is necessary to identify tokens.
- Techniques for speeding up the lexical analyzer, such as the use of sentinels to mark the buffer end.

Specification of Tokens

Regular expressions are a notation to represent lexeme patterns for a token. Regular expressions are used to represent the language for lexical analyzer.

- Strings and Languages
- Operations on Languages
- Regular Expressions

Recognition of Tokens

Recognition of token explains how to take the patterns for all needed tokens. It builds a piece of code that examines the input string and finds a prefix that is a lexeme matching one of the patterns.

Transition Diagram

As an intermediate step in the construction of a lexical analyzer, we first produce flowchart, called a Transition diagram. It depicts the actions that take place when a lexical analyzer is called by the parser to get the next token.

Finite Automata

There are two types of finite automata:

a) Nondeterministic finite automata (NFA)

b) Deterministic finite automata (DFA)

The transition function (δ) maps the finite set of state (Q) to a finite set of input symbols (Σ),$Q \times \Sigma \rightarrow Q$

Regular Expressions to Automata

Conversion of an NFA to a DFA: The Subset Construction Algorithm

The algorithm for constructing a DFA from a given NFA such that it recognizes the same language is called subset construction. The reason is that each state of the DFA machine corresponds to a set of states of the NFA.

Construction of NFA from Regular Expression

The McNaughton-Yamada-Thompson algorithm is used to convert a regular expression to an NFA.

Thompson algorithm

Input: A regular expression r over alphabet Σ
Output: An NFA N accepting (r)

Minimizing DFA

- Minimization of DFA focuses on reducing the number of states in the given finite automata
- The states which are equivalent are grouped thereby enabling the reduction of states
- Two states are said to be equivalent if on taking the same input symbol they transit to same group of states (final or non-final). Otherwise, they are distinguishable states

Language for Specifying Lexical Analyzers

A lexical specification language (like regular expressions) defines token patterns used by a lexer to recognize keywords, identifiers, literals, and operators. It allows concise expression of lexical rules that are automatically translated into efficient scanning code.

Design of Lexical analyzer generator(LEX)

LEX converts regular-expression based token specifications into a deterministic finite automaton (DFA) that scans source code and produces tokens. It outputs C code for the lexer, which works together with a parser generator like YACC.

Recent trends in Compiler Design

Modern compilers use just-in-time (JIT) compilation, machine-learning-based optimizations, and parallelism for faster execution and adaptive performance. Security-oriented compilation, energy-aware optimization, and cloud-based build systems are also emerging trends.