Course content

This course provides practical experience in designing and implementing a compiler for a simple programming language, PL/0. Students will apply the principles learned in the Compiler Theory course to real-world applications, using tools like Lex and Yacc for lexical analysis and syntax parsing. The course focuses on enhancing students' practical skills in compiler construction, error handling, semantic analysis, and code generation.

Key topics covered include:

1. Lexical Analysis: The process of converting input text into tokens.
2. Syntax Analysis: The process of analyzing the grammatical structure of source code for generating parsers.
3. Semantic Analysis: Techniques for checking the correctness of the program's logic, including type checking and scope resolution.
4. Error Handling: Approaches for detecting, reporting, and recovering from errors in the compiler design.
5. Symbol Tables: Construction and management of symbol tables for storing information about variables, functions, and other symbols.
6. Intermediate Code Generation: Creation of intermediate representations of the source code for further optimization and target code generation.
7. Target Code Generation: The process of generating assembly or machine code from intermediate representations.
8. Compiler Optimization: Basic techniques for optimizing generated code to improve performance and reduce size.
9. Testing and Debugging: Techniques for testing the compiler, including creating test cases, analyzing results, and debugging the compiler code.

Course Objectives

Knowledge

1. Understand the principles and stages of compiler construction, including lexical analysis, syntax parsing, semantic analysis, and code generation.
2. Gain familiarity with tools like Lex and Yacc for automatic code generation in the context of compiler design.
3. Develop an understanding of the process of compiler optimization and the challenges involved.

Skills

1. Design and implement a complete compiler for a simple language (PL/0), including all phases from lexical analysis to target code generation.
2. Apply error-handling techniques to detect and report various types of errors during compilation.
3. Manage symbol tables and intermediate representations effectively to support compiler functionality.

4. Use Lex and Yacc to automate parts of the compiler construction process, focusing on efficiency and reliability.

Competencies

1. Integrate various compiler components (lexical analyzer, parser, semantic analyzer, etc.) into a cohesive and functioning system.
2. Evaluate and improve compiler performance through optimization techniques.
3. Analyze and debug compiler behavior, using test cases to validate correctness and efficiency.