Course content

This course covers fundamental principles, technologies, and methods closely related to many other areas of computer science. The course is essential for software professionals, helping to improve their skills and knowledge. It introduces the basic principles, methods, and technologies involved in designing and implementing a compiler. The course provides students with an understanding of how compilers for high-level programming languages are constructed, and it aims to cultivate their ability to analyze and design solutions for problems in the field of compilers.

Key topics covered in the course include:

1. High-Level Languages and Grammar Descriptions: Formal definitions, regular expressions, and finite state automata.
2. Lexical Analysis: Techniques for lexical analysis, tokenization, and regular expressions.
3. Syntax Analysis: Top-down and bottom-up parsing methods.
4. Attribute Grammars and Syntax-Directed Translation: Methods for implementing attribute evaluation in syntax analysis and semi-formal methods for expressing complex compiler problems.
5. Semantic Analysis and Intermediate Code Generation: Techniques for semantic analysis, intermediate code generation, and runtime memory organization.
6. Symbol Table: Design and implementation of symbol tables for managing variables, functions, and other identifiers in a program.
7. Runtime Storage Organization: Memory management during program execution, including stack and heap organization.
8. Code Optimization: Methods for improving the efficiency of the generated code.

Course objectives

Knowledge

1. Understand the fundamental principles and methods of compiler construction, including lexical analysis, syntax analysis, semantic analysis, and code generation.
2. Master techniques for using formal language definitions, finite state automata, and syntax analysis tools in compiler design.

Skills

1. Develop the ability to describe, analyze, solve, and design solutions for problems in compiler construction.
2. Use attribute grammars and syntax-directed translation to handle complex translation problems.
3. Design and implement intermediate code generation and runtime storage management methods.

Competencies

1. Apply the knowledge of compiler construction to design and implement parts of a compiler for a high-level programming language.
2. Solve complex compiler-related problems, such as code optimization and runtime memory management.