

CLUSTER UNIVERSITY OF SRINAGAR

Bachelors in Information Technology

Semester: 6th

Course Type: MINOR

Programme: UG



Course Code: UGICT22N601

Credits: 4 (Theory: 3, Practical: 1)

Contact Hrs.: 75 (Theory: 45, Practical: 30)

Max. Marks: 100

Theory External: 60; Min Marks: 24

Theory Internal: 15 Marks, Min Marks: 06

Practical External: 15 Marks, Min. Marks: 06

Practical Internal: 10 Marks, Min. Marks: 04

Title: Compiler Design

COURSE OBJECTIVES:

- To introduce the fundamental concepts, structure and phases of a compiler.
- To develop an understanding of lexical, syntax and semantic analysis.
- To familiarize students with parsing techniques and syntax-directed translation.
- To introduce intermediate code generation, code optimization and code generation.
- To provide basic practical knowledge of implementing compiler components using suitable tools

COURSE CONTENT:

UNIT I: Compiler Fundamentals and Lexical Analysis

Introduction to language processors, compilers, interpreters and translators, phases of a compiler, front-end and back-end, single-pass and multi-pass compilers, compiler construction tools, lexical analysis, role of lexical analyzer, tokens, patterns and lexemes, specification and recognition of tokens, regular expressions, regular languages and finite automata, conversion of regular expressions to finite automata, lexical analyzer generators and symbol tables.

UNIT II Syntax and Semantic Analysis

Syntax analysis, context-free grammars, derivations and parse trees, ambiguity in grammars, left recursion and left factoring, top-down parsing, recursive descent parsing, predictive parsing and LL parsing, bottom-up parsing, shift-reduce parsing and LR parsing, SLR and LALR parsing, parser generators, syntax-directed definitions and translation, syntax trees, semantic analysis, type checking and error detection and recovery.

Handwritten signatures and text: "Zacka" and other illegible signatures.

Unit III: Intermediate Code, Runtime Environment, Optimization and Code Generation

Intermediate representations, syntax trees, three-address code, quadruples and triples, translation of expressions and control-flow statements, basic blocks and flow graphs, runtime environment and storage organization, activation records, code generation, instruction selection, register allocation, code optimization, local and global optimization, common optimization techniques, loop optimization and peephole optimization.

UNIT IV

- Implementation of a Lexical Analyzer to identify tokens from a given source program.
- Implementation of Regular Expression and Finite Automata Based Token Recognition.
- Implementation of a Simple Parser using recursive descent or predictive parsing.
- Construction of Parse Trees for simple grammars and expressions.
- Generation of Three-Address Code for arithmetic expressions and simple control statements.
- Implementation of Basic Code Optimization Techniques such as constant folding, common sub-expression elimination and dead-code elimination.

COURSE OUTCOMES After successful completion of the course, students will be able to:

- Explain the different phases and functions of a compiler.
- Design basic lexical analyzers using tokens, regular expressions and finite automata.
- Apply suitable parsing techniques to analyze simple grammars.
- Explain syntax-directed translation, semantic analysis and type checking.
- Generate intermediate code and apply basic code optimization techniques.
- Implement basic components of a compiler using suitable programming languages or compiler-construction tools.

Text Book

1. A. V. Aho, M. S. Lam, R. Sethi and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd Edition, Pearson/Addison-Wesley.

Reference Books

1. Andrew W. Appel and Jens Palsberg, *Modern Compiler Implementation in Java*, 2nd Edition, Cambridge University Press.
2. Keith D. Cooper and Linda Torczon, *Engineering a Compiler*, Morgan Kaufmann.

A series of handwritten signatures in blue ink, likely belonging to the faculty members of the institution. The signatures are stylized and vary in length and complexity.