Semester: 1	
Programme: M.Sc. Computer Science	
Course : COMPILER DESIGN	
Paper code: M2C4CS26012T	**Credits: 6**
Hours/week : Theory: 4	
Category: Core/MDC/SEC/VAC : Core	
Theory / Practical / Composite : Theory	
No of Modules : 1	

Course Overview: This course provides a rigorous foundation in the theory and practice of compiler construction, transforming high-level programming languages into efficient machine code. Students explore the complete compilation pipeline—from lexical analysis to runtime storage management—while developing hands-on skills in implementing language processors. The curriculum bridges theoretical concepts (grammar formalisms, parsing algorithms) with practical implementation (symbol table management, code optimization). Through this course, students gain critical expertise for careers in programming language design, systems programming, and software engineering, with direct relevance to modern compiler infrastructure (LLVM, GCC) and domain-specific language development.

Course Outcome:

1. **Explain** fundamental concepts of compiler architecture, including translator types (assemblers, compilers, interpreters), language recognition mechanisms, and the multi-phase structure of compilation.
2. **Design** lexical analyzers using finite state machines and **implement** syntax analyzers (LL(1), SLR(1), recursive descent) for context-free grammars with operator precedence handling.
3. **Construct** intermediate code representations (three-address code, quadruples, triples) and **apply** code optimization techniques (loop optimization, DAG-based analysis) to improve program efficiency.
4. **Develop** code generation strategies for object programs, **implement** symbol table management systems, and **design** error handling and runtime storage management (static/dynamic allocation) for compiler backends.
5. **Analyze** trade-offs in compiler design approaches and **evaluate** their impact on performance, complexity, and applicability to modern programming language features (e.g., polymorphism, concurrency).

Prerequisites:

- Discrete Structures & Graph Algorithms
- Theory of Computation
- Data Structures and Algorithms
- Programming Languages
- Computer Organisation
- Operating System

SYLLABUS

UNIT/Module	CONTENT	HOURS or NUMBER OF CLASSES	CO Mapping	COGNITIVE LEVEL
I.	**Introduction:** Concepts and Types of Translators: Assembler, Compiler, Cross Assemblers and Compilers, Interpreter,	4	**CO1**	**Understand (K2)**

	Loader; Linker. Grammars, Languages – types of grammars and their recognizers. **Different phases of compilation.**			
II.	**Lexical Analysis:** Concepts, Tokens, Schemas, Design using FSM	5	**CO2**	**Apply (K3)**
III.	**Syntax Analysis:** Top down and Bottom-up parser; Operator precedence; Recursive descent; LL (1); SLR (1)	16	**CO2**	**Apply (K3)**
IV.	**Intermediate Code Generation:** Three Address Code, Representation of three address code – Quadruples, Triples and Indirect Triples. Syntax directed translation: Attributes, Semantic Actions, Translation schemes.	6	**CO3**	**Apply (K3)**
V.	**Code Optimization:** Basic blocks, loop optimization, flow graph, DAG representations of basic blocks.	6	**CO3**	**Apply, Analyse (K3, K4)**
VI.	**Code Generation:** Object Programs, Problems in Code generation.	3	**CO4**	**Apply, Create (K3, K6)**
VII.	**Error handling:** detection, reporting, recovery and repair	3	**CO4**	**Apply (K3)**
VIII.	**Table management:** Symbol table, Organization and management techniques.	4	**CO4**	**Apply, Create (K3, K6)**
IX.	**Runtime storage management:** static allocation; dynamic allocation, activation records; heap allocation, recursive procedures	5	**CO5**	**Analyse, Evaluate (K4, K5)**

Text Books

1. Alfred V. Aho and Jeffrey D. Ullman, Principles of Compiler Design, Narossa Publication
2. Aho, Sethi and Ullman, Compilers – Principles, Techniques and Tools, Narossa Publication
3. Peter Linz, Formal Language and Automata Theory, Narossa Publication
4. Systems Programming and Operating System, D. M. Dhamdhere, Tata McGraw Hills
5. Systems Programming, John J Donovan, Tata McGraw Hills

Evaluation

Theory CIA: 20 Semester Exam: 80	

Paper Structure for Theory Semester Exam Module:
Answer 8 out of 12 of 10 marks each

Course outcomes (COs) and Cognitive Level Mapping

COs	CO Description	Cognitive levels
CO1	**Explain** fundamental concepts of compiler architecture, including translator types (assemblers, compilers, interpreters), language recognition mechanisms, and the multi-phase structure of compilation.	Understand (K2)
CO2	**Design** lexical analyzers using finite state machines and **implement** syntax analyzers (LL(1), SLR(1), recursive descent) for context-free grammars with operator precedence handling.	Apply (K3)
CO3	**Construct** intermediate code representations (three-address code, quadruples, triples) and **apply** code optimization techniques (loop optimization, DAG-based analysis) to improve program efficiency.	Apply, Analyse (K3, K4)
CO4	**Develop** code generation strategies for object programs, **implement** symbol table management systems, and **design** error handling and runtime storage management (static/dynamic allocation) for compiler backends.	Apply, Create (K3, K6)
CO5	**Analyze** trade-offs in compiler design approaches and **evaluate** their impact on performance, complexity, and applicability to modern programming language features (e.g., polymorphism, concurrency).	Analyse, Evaluate (K4, K5)