

<b>Semester: VII</b>				
<b>Programme :</b> B.Sc. Computer Science (Hons)				
<b>Course :</b> COMPILER DESIGN				
<b>Paper code:</b> C4CS230721T				<b>Credits: 4</b>
<b>Hours/week :</b> Theory: 4				
<b>Category:</b> Core/MDC/SEC/VAC : Core				
<b>Theory / Practical / Composite :</b> Theory				
<b>No of Modules :</b> 1				
<p><b>Course Overview:</b> 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.</p>				
<b>Course Outcome:</b>				
1. <b>Explain</b> fundamental concepts of compiler architecture, including translator types (assemblers, compilers, interpreters), language recognition mechanisms, and the multi-phase structure of compilation.				
2. <b>Design</b> lexical analyzers using finite state machines and <b>implement</b> syntax analyzers (LL(1), SLR(1), recursive descent) for context-free grammars with operator precedence handling.				
3. <b>Construct</b> intermediate code representations (three-address code, quadruples, triples) and <b>apply</b> code optimization techniques (loop optimization, DAG-based analysis) to improve program efficiency.				
4. <b>Develop</b> code generation strategies for object programs, <b>implement</b> symbol table management systems, and <b>design</b> error handling and runtime storage management (static/dynamic allocation) for compiler backends.				
5. <b>Analyze</b> trade-offs in compiler design approaches and <b>evaluate</b> their impact on performance, complexity, and applicability to modern programming language features (e.g., polymorphism, concurrency).				
<b>Prerequisites:</b>				
<ul style="list-style-type: none"> <li>• Discrete Structures &amp; Graph Algorithms</li> <li>• Theory of Computation</li> <li>• Data Structures and Algorithms</li> <li>• Programming Languages</li> <li>• Computer Organisation</li> <li>• Operating System</li> </ul>				
<b>SYLLABUS</b>				
<b>UNIT/Module</b>	<b>CONTENT</b>	<b>HOURS or NUMBER OF CLASSES</b>	<b>CO Mapping</b>	<b>COGNITIVE LEVEL</b>
I.	<b>Introduction:</b> 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. <b>Different phases of compilation.</b>			
II.	<b>Lexical Analysis:</b> Concepts, Tokens, Schemas, Design using FSM	5	CO2	Apply (K3)
III.	<b>Syntax Analysis:</b> Top down and Bottom-up parser; Operator precedence; Recursive descent; LL (1); SLR (1)	16	CO2	Apply (K3)
IV.	<b>Intermediate Code Generation:</b> 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.	<b>Code Optimization:</b> Basic blocks, loop optimization, flow graph, DAG representations of basic blocks.	6	CO3	Apply, Analyse (K3, K4)
VI.	<b>Code Generation:</b> Object Programs, Problems in Code generation.	3	CO4	Apply, Create (K3, K6)
VII.	<b>Error handling:</b> detection, reporting, recovery and repair	3	CO4	Apply (K3)
VIII.	<b>Table management:</b> Symbol table, Organization and management techniques.	4	CO4	Apply, Create (K3, K6)
IX.	<b>Runtime storage management:</b> 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: 25  
Attendance: 5  
Semester Exam: 70

#### Paper Structure for Theory Semester Exam Module:

Answer 5 out of 7 of 14 marks each

### Course outcomes (COs) and Cognitive Level Mapping

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