Course: BTech **Semester:** 6

Prerequisite: Algorithms, Data Structures, Assembly Language Program, Theory of Computation, C/C++ Programming Skills | 203105205 - Data Structure and Algorithms

Rationale: Compiler Design is a fundamental subject of Computer Engineering. Compiler design principles provide an in-depth view of translation, optimization and compilation of the entire source program. It also focuses on various designs of compiler and structuring of various phases of compiler. It is inevitable to grasp the knowledge of various types of grammar, lexical analysis, yacc, FSM(Finite State Machines) and correlative concepts of languages.

Teaching and Examination Scheme

Teaching Scheme					Examination Scheme					Total
					Internal Marks			External Marks		
Lecture Hrs/Week	Tutorial Hrs/Week	Lab Hrs/Week	Hrs/Week	Credit	T	CE	P	T	P	
3	0	0	0	3	20	20	-	60	-	100

SEE - Semester End Examination, **CIA** - Continuous Internal Assessment (It consists of Assignments/Seminars/Presentations/MCQ Tests, etc.)

Course Content

W - Weightage (%) , **T** - Teaching hours

Sr.	Topics	W	T
1	**Overview of compilation :** The structure of a compiler and applications of compiler technology; Lexical analysis - The role of a lexical analyzer, specification of tokens, recognition of tokens, hand-written lexical analyzers, LEX, examples of LEX programs.	10	8
2	**Introduction to syntax analysis** Role of a parser, use of context-free grammars (CFG) in the specification of the syntax of programming languages, techniques for writing grammars for programming languages (removal left recursion, etc.), non-context-free constructs in programming languages, parse trees and ambiguity, examples of programming language grammars.	10	7
3	**Top-down parsing** FIRST & FOLLOW sets, LL(1) conditions, predictive parsing, recursive descent parsing, error recovery. LR-parsing - Handle pruning, shift-reduce parsing, viable prefixes, valid items, LR(0) automaton, LR-parsing algorithm, SLR(1), LR (1), and LALR(1) parsing. YACC, error recovery with YACC and examples of YACC specifications.	20	7
4	**Syntax-directed definitions (attribute grammars)** Synthesized and inherited attributes, examples of SDDs, evaluation orders for attributes of an SDD, dependency graphs. S-ttributed and L-attributed SDDs and their implementation using LR-parsers and recursive-descent parsers respectively.	15	6
5	**Semantic analysis** Symbol tables and their data structures. Representation of “scope”. Semantic analysis of expressions, assignment, and control-flow statements, declarations of variables and functions, function calls, etc., using S- and L-attributed SDDs (treatment of arrays and structures included). Semantic error recovery.	15	6
6	**Intermediate code generation** Different intermediate representations –quadruples, triples, trees, flow graphs, SSA forms, and their uses. Translation of expressions (including array references with subscripts) and assignment statements. Translation of control-flow statements – it- then-else, while-do, and switch. Short-circuit code and control-flow translation of Boolean expressions. Back patching. Examples to illustrate intermediate code generation for all constructs.	15	6
7	**Run-time environments**Stack allocation of space and activation records. Access to non-local data on the stack in the case of procedures with and without nesting of procedures.	10	3
8	**Introduction to machine code generation and optimization** Simple machine code generation, examples of machine-independent code optimizations.	5	2

Reference Books

1.	**Compilers: Principles, Techniques and Tools** By Aho, Lam, Sethi, and Ullman | Pearson | Second, Pub. Year 2014

Course Outcome

After Learning the Course the students shall be able to:

1. Understand the basic concepts; ability to apply automata theory and knowledge on formal languages.
2. Ability to identify and select suitable parsing strategies for a compiler for various cases. Knowledge in alternative methods (top-down or bottom-up, etc.).
3. Understand backend of compiler: intermediate code, Code optimization Techniques and Error Recovery mechanisms
4. Understand issues of run time environments and scheduling for instruction level parallelism.

Course: BTech **Semester:** 6

Prerequisite: Algorithms, Data Structures, Assembly Language Program, Theory of Computation, C/C++ Programming Skills | 203105205 - Data Structure and Algorithms

Rationale: Compiler Design is a fundamental subject of Computer Engineering. Compiler design principles provide an in-depth view of translation, optimization and compilation of the entire source program. It also focuses on various designs of compiler and structuring of various phases of compiler. It is inevitable to grasp the knowledge of various types of grammar, lexical analysis, yacc, FSM (Finite State Machines) and correlative concepts of languages.

Teaching and Examination Scheme

Teaching Scheme					Examination Scheme					Total
					Internal Marks			External Marks		
Lecture Hrs/Week	Tutorial Hrs/Week	Lab Hrs/Week	Hrs/Week	Credit	T	CE	P	T	P	
0	0	2	0	1	-	-	20	-	30	50

SEE - Semester End Examination, **CIA** - Continuous Internal Assessment (It consists of Assignments/Seminars/Presentations/MCQ Tests, etc.)

Course Outcome

After Learning the Course the students shall be able to:
After learning the course, the students should be able to 1. Understand the basic concepts; ability to apply automata theory and knowledge on formal languages. 2. Ability to identify and select suitable parsing strategies for a compiler for various cases. Knowledge in alternative methods (top-down or bottom-up, etc.). 3. Understand backend of the compiler: intermediate code, Code Optimization Techniques and Error Recovery mechanisms 4. Understand issues of run time environments and scheduling for instruction level parallelism

List of Practical

1.	**Program to implement Lexical Analyzer.**
2.	**Program to count digits, vowels and symbols in C.**
3.	**Program to check validation of User Name and Password in C.**
4.	**Program to implement Predictive Parsing LL (1) in C.**
5.	**Program to implement Recursive Descent Parsing in C.**
6.	**Program to implement Operator Precedence Parsing in C.**
7.	**Program to implement LALR Parsing in C.**
8.	**To Study about Lexical Analyzer Generator (LEX) and Flex (Fast LexicalAnalyzer)**
9.	**Implement following programs using Lex.** a. Create a Lexer to take input from text file and count no of characters, no. of lines & no. of words. b. Write a Lex program to count number of vowels and consonants in a given input string.
10.	**Implement following programs using Lex.** a. Write a Lex program to print out all numbers from the given file. b. Write a Lex program to printout all HTML tags in file. c. Write a Lex program which adds line numbers to the given file and display the same onto the standard output.