COMPILER DESIGN

V Semester: **CSE / IT**

Course Code	Category	Hours / WEEK			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AIT004	Core	3	1	-	4	30	70	100
Contact Classes: 45	**Tutorial Classes: 15**	**Practical Classes: Nil**				**Total Classes: 60**		

I. COURSE OVERVIEW:

This course describes the basic techniques for compiler construction and tools that can be used to perform syntax-directed translation of a high-level programming language into an executable code. It will provide deeper insights into the more advanced semantics aspects of programming languages, machine independent optimizations and code generation.

II. OBJECTIVES:

The course should enable the students to:

I	The process of translating a high-level language to machine code required forcompiler construction.
II	The Software tools and techniques used in compiler construction such as lexical analyzer and parser generators.
III	The data structures used in compiler construction such as abstract syntax trees, symbol tables, three-address code, and stack machines.
IV	The deeper insights into the syntax and semantic aspects of programming languages, dynamic memory allocation and code generation.

III. COURSE OUTCOMES:

After successful completion of the course, students should be able to:

CO 1	**Summarize** phases of a compiler in the construction of languageprocessors.	Understand
CO 2	**Make use of finite** automata for designing a lexical analyzer for a specific programming language constructs.	Apply
CO 3	**Choose** top down, bottom up parsing methods for developing a parser with representation of a parse table or tree.	Apply
CO 4	**Outline** syntax directed translations, intermediate forms for performing semantic analysis along with code generation.	Understand
CO 5	**Relate** symbol table, type checking and storage allocation strategies used in run-time environment.	Understand
CO 6	**Select** code optimization techniques on intermediate code form for generating target code.	Apply

IV. SYLLABUS:

UNIT-I	INTRODUCTION TO COMPILERS AND PARSING	Classes: 08

Introduction to compilers**:** Definition of compiler, interpreter and its differences, the phases of a compiler, role of lexical analyzer, regular expressions, finite automata, from regular expressions to finite automata, pass and phases of translation, bootstrapping, LEX-lexical analyzer generator; Parsing: Parsing, role of parser, context free grammar, derivations, parse trees, ambiguity, elimination of left recursion, left factoring, eliminating ambiguity from dangling-else grammar, classes of parsing, top-down parsing: backtracking, recursive-descent parsing, predictive parsers, LL(1) grammars.

UNIT-II	BOTTOM-UP PARSING	Classes: 09

Bottom-up parsing: Definition of bottom-up parsing, handles, handle pruning, stack implementation of shift-reduce parsing, conflicts during shift-reduce parsing, LR grammars, LR parsers-simple LR, canonical LR and Look Ahead LR parsers, error recovery in parsing, parsing ambiguous grammars, YACC-automatic parser generator.

UNIT-III	**SYNTAX-DIRECTED TRANSLATION AND INTERMEDIATE CODE GENERATION**	**Classes: 10**
Syntax-directed translation: Syntax directed definition, construction of syntax trees, S-attributed and L-attributed definitions, translation schemes, emitting a translation. Intermediate code generation: Intermediate forms of source programs– abstract syntax tree, polish notation and three address code, types of three address statements and its implementation, syntax directed translation into three-address code, translation of simple statements, Boolean expressions and flow-of-control statements.		
UNIT-IV	**TYPE CHECKING AND RUN TIME ENVIRONMENT**	**Classes: 09**
Type checking: Definition of type checking, type expressions, type systems, static and dynamic checking of types, specification of a simple type checker, equivalence of type expressions, type conversions, overloading of functions and operators; Run time environments: Source language issues, Storage organization, storage-allocation strategies, access to nonlocal names, parameter passing, symbol tables, and language facilities for dynamic storage allocation.		
UNIT-V	**CODE OPTIMIZATION AND CODE GENERATOR**	**Classes: 09**
Code optimization: The principle sources of optimization, optimization of basic blocks, loops in flow graphs, peophole optimization; Code generator: Issues in the design of a code generator, the target machine, runtime storage management, basic blocks and flow graphs, a simple code generator, register allocation and assignment, DAG representation of basic blocks.		
Text Book:		
Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, “Compilers–Principles, Techniques and Tools”, Pearson Education, Low Price Edition, 2004.		
Reference Books:		
1. Kenneth C. Louden, Thomson, “Compiler Construction– Principles and Practice”, PWS Publishing, 1st Edition, 1997. 2. Andrew W. Appel, “Modern Compiler Implementation C”, Cambridge University Press, Revised Edition, 2004.		
Web References:		
1. www.vssut.ac.in/lecture_notes/lecture1422914957.pdf 2. http://csenote.weebly.com/principles-of-compiler-design.html 3. http://www.faadooengineers.com/threads/32857-Compiler-Design-Notes-full-book-pdf-download 4. https://www.vidyarthiplus.com/vp/thread-37033.html#.WF0PhlMrLDc		
E-Text Books:		
1. http://www.e-booksdirectory.com/details.php?ebook=10166 2. http://www.e-booksdirectory.com/details.php?ebook=7400re		
Course Home Page:		