Faculty of Engineering & Technology
Master of Technology (M. Tech)
(W. E. F.: 2023-24)

Document ID: SUTEFETM-01

Name of Faculty	:	Faculty of Engineering & Technology
Name of Program	:	Master of Technology (M. Tech)
Course Code	:	2MSE02
Course Title	:	Advanced Compiler Design (PE - II)
Type of Course	:	PE
Year of Introduction	:	2023-24

Prerequisite	:	Compiler Design Basics and Data Structure Algorithm
Course Objective	:	The course objectives for an Advanced Compiler Design course may vary depending on the institution and the instructor's preferences. However, here are some common objectives you might find in such a course: Explore Advanced Compiler Techniques, Study Compiler Optimization, Understand Language Semantics and Analysis, Investigate Just-In-Time (JIT) Compilation, Study Compiler Backends and Code Generation etc.
Course Outcomes	:	At the end of this course, students will be able to:
	CO1	Specify and analyse the lexical, syntactic and semantic structures of advanced language features.
	CO2	Separate the lexical, syntactic and semantic analysis into meaningful phases for a compiler to undertake language translation.
	CO3	Write a scanner, parser, and semantic analyser without the aid of automatic generators.
	CO4	Turn fully processed source code for a novel language into machine code for a novel computer.
	CO5	Describe techniques for intermediate code and machine code optimization.
	CO6	Design the structures and support required for compiling advanced language features.

Teaching and Examination Scheme

Teaching Scheme (Contact Hours)			Credits	Examination Marks				
				Theory Marks		Practical Marks		Total Marks
L	T	P	C	SEE	CIA	SEE	CIA	
4	0	2	5	70	30	30	20	100

*Legends: **L**-Lecture; **T**–Tutorial/Teacher Guided Theory Practice; **P** – Practical, **C** – Credit, **SEE** – Semester End Examination, **CIA** - Continuous Internal Assessment (It consists of Assignments/Seminars/Presentations/MCQ Tests, etc.))*

Course Content

Unit No.	Topics	Teaching Hours	Weightage	Mapping with CO
1	**Language Translation Overview** Overview of system software used during translation –language processors, linker, loader. Types of language processors –assembler, interpreter, compiler. Difference between interpreter, assembler and compiler. Overview and use of linker and loader, model of compilation, The Phases of a Compiler, The Grouping of Phases, Compiler-Construction Tools	06	05%	CO2
2	**Lexical Analysis** The Role of the Lexical Analyser, regular expression, regular languages, Input Buffering, Specification of Lexemes, Tokens and pattern. Recognition of Tokens, A Language for Specifying Lexical Analysers, Finite Automata, From a Regular Expression to an NFA, Design of a Lexical Analyser Generator, Optimization of DFA-Based Pattern Matchers.	09	14%	CO1
3	**Syntax Analysis** The Role of the Parser, Context-Free Grammars, writing a Grammar, Top-Down Parsing, Bottom-Up Parsing, Operator-Precedence Parsing, LR Parsers, Using Ambiguous Grammars, Parser Generators.	15	23%	CO2
4	**Syntax-Directed Translation** Syntax-Directed Definitions, Construction of Syntax Trees, Bottom- Up Evaluation of S-Attributed Definitions, L-Attributed Definitions, Top-Down Translation, Bottom-Up Evaluation of Inherited Attributes, Recursive Evaluators, Analysis of Syntax-Directed Definitions, Type Systems, Specification of a Simple Type Checker, Equivalence of Type Expressions, Type Conversions, Overloading of Functions and Operators.	05	08%	CO3
5	**Memory Allocation, Organization and Memory Management** Source Language Issues, Storage Organization, Storage-Allocation Strategies, and Access to Non local Names, Parameter Passing, and Language Facilities for Dynamic Storage Allocation, Dynamic Storage Allocation Techniques. Activation Tree, Activation Record, Parameter Passing, Symbol Table, Static, Dynamic And	07	14%	CO4

	Heap Storage Allocation, Garbage Collection.			
6	**Intermediate Code Generation** Intermediate Languages, Declarations, Assignment Statements, Boolean Expressions, Case Statements, Back patching, Procedure Calls, Types of Intermediate Forms of the Program.	05	08%	CO5
7	**Code Optimization** The Principal Sources of Optimization, Optimization of Basic Blocks, Loops in Flow Graphs, Introduction to Global Data-Flow Analysis, Iterative Solution of Data-Flow Equations, Linear optimization (peep hole) Techniques, parse optimization Techniques and structured optimization techniques. Code-Improving Transformations, Dealing with Aliases, Data-Flow Analysis of Structured Flow Graphs, Efficient Data-Flow Algorithms, A Tool for Data-Flow Analysis, Estimation of Types, Symbolic Debugging of Optimized Code	06	08%	CO5
8	**Code Generation** Issues in the Design of a Code Generator, The Target Machine, Run- Time Storage Management, Basic Blocks and Flow Graphs, Next-Use Information, A Simple Code Generator, Register Allocation and Assignment, The DAG Representation of Basic Blocks, Peephole Optimization, Generating Code from DAGs, Dynamic Programming Code-Generation Algorithm, Code-Generator Generators.	04	08%	CO6
9	**Symbol Table Management** General concepts, Symbol Table as a data structure, Various operations performed on Symbol Table, Symbol table organizations for blocked structured language and non-blocked structured language.	05	12%	CO6

Suggested Distribution of Theory Marks Using Bloom's Taxonomy						
Level	Remembrance	Understanding	Application	Analyse	Evaluate	Create
Weightage	**40**	**20**	**20**	**10**	**-**	**10**

NOTE: This specification table shall be treated as a general guideline for the students and the teachers. The actual distribution of marks in the question paper may vary slightly from above table.

Suggested List of Experiments/Tutorials

Sr. No.	Name of Experiment/Tutorial	Teaching Hours
1	Importance/Rationale behind the CD Lab.	01
2	Objectives & Outcomes.	01
3	Software / Hardware Requirements.	01
4	**Case Study:** Description of the Syntax of the source Language (mini language) for which the compiler components are designed.	02
5	Write a C Program to Scan and Count the number of characters, words, and lines in a file.	02
6	Write a C Program to implement NFAs that recognize identifiers, constants, and operators of the mini language.	02
7	Write a C Program to implement DFAs that recognize identifiers, constants, and operators of the mini language.	02
8	Design a lexical analyzer for the given language. The lexical analyzer should ignore redundant spaces, tabs and new lines, comments etc.	01
9	Implement the lexical analyzer using JLex, flex or other lexical analyzer generating tools.	01
10	Design Predictive Parser for the given language.	02
11	Design a LALR bottom up parser for the given language.	01
12	Convert the BNF rules into Yacc form and write code to generate abstract syntax tree.	01
13	A program to generate machine code from the abstract syntax tree generated by the parser.	01

Suggested Learning Websites

Sr. No.	Name of Website
1	http://compilers.iecc.com/crenshaw
2	http://www.compilerconnection.com
3	http://dinosaur.compilertools.net
4	http://pltplp.net/lex-yacc

Reference Books

Sr. No.	Name of Reference Books
1	Rich, Craig A. Advanced Compiler Design--CS 441 Lecture Notes, Spring 001(Available at Bronco Copy 'n Mail in the University Union).
2	Allen I. Holub “Compiler Design in C”, Prentice Hall of India.
3	C. N. Fischer and R. J. LeBlanc, “Crafting a compiler with C”, Benjamin Cummings.
4	J.P. Bennet, “Introduction to Compiler Techniques”, Second Edition, Tata McGraw- Hill.
5	Henk Alblas and Albert Nymeyer, “Practice and Principles of Compiler Building with C”, PHI.

6	Kenneth C. Louden, “Compiler Construction: Principles and Practice”, Thompson Learning.
7	Compiler Construction by Kenneth. C. Louden, Vikas Pub.