PROGRAM CORE

Course Code	Course Title	L	T	P	S	C
22CS102006	**COMPILER DESIGN**	3	-	2	-	4

Pre-Requisite Theory of Computation

Anti-Requisite -

Co-Requisite -

COURSE DESCRIPTION: Lexical analysis; Parsers; Run Time Environments; Syntax Directed Translation; Type checking; Code Optimization; Code Generation and Compiler tools.

COURSE OUTCOMES: After successful completion of the course, students will be able to:

CO1. Demonstrate knowledge on the phases involved in design of compilers..

CO2. Analyze code optimization Techniques..

CO3. Design experiments for implementing parsing techniques.

CO4. Synthesize rules in compiler to demonstrate semantic attribution during Parsing.

CO5. Use compiler construction tools such as LEX and YACC for designing a Parser.

CO-PO-PSO Mapping Table:

Course Outcomes	Program Outcomes												Program Specific Outcomes		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3	-	-	-	-	-	-	-	-	-	3	-	3
CO2	3	3	-	-	-	-	2	-	-	-	-	-	3	-	3
CO3	3	3	3	-	-	-	-	-	-	-	2	-	3	-	3
CO4	2	3	3	-	-	-	-	2	-	-	-	-	3	-	-
CO5	3	3	3	-	-	-	-	-	-	-	-	-	3	-	-
Course Correlation Mapping	**3**	**3**	**3**	**-**	**-**	**-**	**2**	**2**	-	**-**	**2**	**-**	**3**	-	**3**

Correlation Levels: 3: High; 2: Medium;1: Low

COURSE CONTENT

MODULE I– INTRODUCTION TO COMPILER AND LEXICAL ANALYSIS (09 Periods)

Structure of a compiler, Interpretation- Interpreters, Recursive interpreters, Iterative interpreters. Lexical Analysis: The Role of the Lexical Analyzer, Input Buffering, Specification of Tokens, The Lexical-Analyzer Generator LEX.

MODULE II – SYNTAX ANALYSIS (09 Periods)

The Role of the Parser, Eliminating Ambiguity, Eliminating of Left Recursion and Left Factoring. Top-Down Parsing: Recursive descent parsing, Non Recursive Predictive parsing, LL (1) Grammars, A traditional top-down parser generator—YACC

Bottom-Up Parsing: Shift reduce parsing, LR parsers – Simple LR parser, Canonical LR parser, LALR parser, Using Ambiguous Grammars.

MODULE III – SYNTAX DIRECTED TRANSLATION AND TYPE CHECKING (09 Periods)

Syntax directed definition, S-attributed and L-attributed definitions, Construction of syntax trees. Type Checking: Type Expressions, Type Equivalence, Rules for Type Checking, Type Conversions, Overloading of Functions and Operators.

MODULE IV – INTERMEDIATE CODE GENERATOR AND RUN TIME ENVIRONMENTS (09 Periods)

Preprocessing the intermediate code, Preprocessing of expressions, Preprocessing of if-statements and goto statements, Preprocessing of routines, Variants of Syntax Trees, Three Address Code, Boolean expressions, Flow-of-Control Statements, Control- Flow Translation of Boolean Expressions.

Run time Environments: Storage organization, Stack Allocation of Space, Access to Nonlocal Data on the Stack.

MODULE V – CODE OPTIMIZATION AND CODE GENERATION (09 Periods)

Basic Blocks and Flow Graphs, Optimization of Basic Blocks, The principal sources of optimization, Introduction to data flow analysis.

Code Generation: Issues in the Design of a Code Generator, The Target Language, Simple Code Generator, Peephole optimization, Register allocation and assignment.

Total Periods: 45

EXPERIENTIAL LEARNING

1. Write a LEX Program to scan reserved word & Identifiers of C Language language.
2. Implement Predictive Parsing algorithm.
3. Write a C program to generate three address code.
4. Implement SLR(1) Parsing algorithm.
5. Design LALR bottom up parser for the given language.
6. Write a C program for implementing the functionalities of predictive parser for the mini language specified in Note 1.
7. Write a C program for construction of LL (1) parsing.
8. Write a C program for constructing recursive descent parsing.

9. Implement a desk calculator using operator precedence parsing.

10. Consider the syntax of a programming language construct such as while-loop --

while (condition)

begin

statement ;

:

end

where while, begin, end are keywords; condition can be a single comparision expression (such as x == 20, etc.); and statement is the assignment to a location the result of a single arithmatic operation (eg., a = 10 * b).

Write a program that verifies whether the input follows the above syntax.

TEXT BOOK:

1. Alfred V. Aho, Monica S.Lam, Ravi Sethi, Jeffrey D. Ullman, Compilers–Principles, Techniques and Tools, Pearson Education, 2nd edition, 2012.

REFERENCE BOOKS:

1. Dick GruneKees van Reeuwijk Henri, Modern Compiler Design, Springer,2nd edition, 2012.

2. David Galles, Modern Compiler Design, Pearson Education Asia, 2007.

SOFTWARE/TOOLS:

1. Software: Turbo C++/Dev C++

VIDEO LECTURES:

1. https://nptel.ac.in/courses/106104123
2. https://www.reddit.com/r/Compilers/comments/10dpnky/compiler_design_theory_course_with_video_lectures/?rdt=44592

WEB RESOURCES:

1. https://nitsri.ac.in/Department/Computer%20Science%20&%20Engineering/CD-LEC-NOTES.pdf
2. http://www2.cs.uidaho.edu/~jeffery/courses/nmsu/370/lecture.html