CSEN3031	COMPILER DESIGN	L	T	P	S	J	C
		3	0	2	0	0	4
Pre-requisite	None						
Co-requisite	None						
Preferable exposure	None						

Course Description:

Compilers play a significant role in fulfilling user's computing requirements, specified in terms of programs in a high-level language, which translate into machine-understandable form. The process involved in such a transformation of a program is quite complex. This course enables the students to gain knowledge on various phases involved in designing a compiler. Theory and Formal Languages provides the basis for this course in which several automated tools help construct various phases. Advanced computer architecture, memory management, and operating systems help the compiler designer generate efficient code.

Course Educational Objectives:

- Explore the basic techniques that underlie the principles, algorithms and data structures involved in the Compiler Construction.
- Gain experience in using automated tools that helps in transforming various phases of the compiler.

UNIT 1 Introduction & Lexical Analysis 9 hours, P - 6 hours

Introduction: The Structure of Compiler, The Science of Building a Compiler in Bootstrapping and Cross compiler.

Lexical Analysis: The role of the Lexical analyzer, Input Buffering, Specification of Tokens, Recognition of Tokens, The Lexical Analyzer Generator (LEX/FLEX).

UNIT 2 Syntax Analysis (Part-I) 9 hours, P - 6 hours

Syntax Analysis (Part-I): Introduction, Context-Free Grammars, Top-Down parsing: Brute force Parsing, Recursive Descent Parsing, Non-recursive Predictive Parsing, Error Recovery in Predictive Parsing, Bottom- Up parsing - Shift Reduce Parsing, Operator-Precedence Parsing, Error Recovery in Operator Precedence Parsing.

UNIT 3 Syntax Analysis (Part-II) 9 hours, P - 6 hours

Syntax Analysis (Part-II): Introduction to LR Parsing: Simple LR Parser, More Powerful LR Parsers (CLR&LALR), Using Ambiguous grammars, Error Recovery in LR Parsers, Parser Generator (YACC).

UNIT 4 Syntax Directed Translation & Intermediate Code Generation 9 hours, P - 6 hours

Syntax Directed Translation: Syntax Directed Definitions, Types of attributes, Evaluation Orders for SDD's, Applications of Syntax-Directed Translation.

Intermediate Code Generation: Three Address codes, Types & Declarations, Translation of Arithmetic Expressions, Back patching for Boolean Expressions, and Flow of Control.

UNIT 5 Code Optimization & Code Generation 9 hours, P - 6 hours

Code Optimization: The Principal Sources of Optimization, Basic blocks and Flow Graphs, Optimization of Basic Blocks, Introduction to Data-Flow Analysis: Live Variable Analysis

Code Generation: Issues in designing a code Generator, The Target Language, A Simple Code Generator, Peephole Optimization, Register allocation, and Assignment.

List of experiments

1. Implement transition diagram for identifying an identifier and classify whether it is variable, array, or function.
2. Implement transition diagram for identifying constant and classifying whether it is integer or real.
3. Write a LEX program to count the number of characters, words, lines, and blanks in the given input file.
4. Write a LEX program to recognize all the tokens in the given input file. Here the input file is having source code.
5. Write a program which reads CFG and checks for left recursion then display the CFG after elimination of left recursion.
6. Write a program which reads CFG and if required apply the left factoring then display the resulting CFG.
7. Write a program to implement Recursive Descent Parser for the given grammar.
8. Write a program to check if a given expression is valid or not using YACC for the following grammar.

E -> E+E
E -> E-E
E -> E*E
E -> E/E
E -> E%E
E -> (E)
E -> number
E ->id

9. Write a program to evaluate the given expression using LEX and YACC for the following grammar.

E -> E+E
E -> E-E
E -> E*E

E -> E/E
E -> E%E
E -> (E)
E -> number
E -> id

10. Write a program to covert infix expression to postfix expression using LEX and YACC for the following grammar.

E -> E+T | E-T | T
T -> T*F | T/F | F
F -> (E) | -F | id

11. Write a program to generate Quadruples (Three address code) for the given expression using LEX and YACC for the following grammar.

E -> E+E
E -> E-E
E -> E*E
E -> E/E
E -> E%E
E -> (E)
E -> number
E -> id

12. Write a program to implement a simple Code Generator for the given three address code.

TextBooks:

1. Alfred.V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey. D. Ullman, Compilers Principles, Techniques and Tools, 2/e, Pearson Education, 2008.

References:

1. Alfred.V. Aho, J.D.Ullman, Principles of compiler design, Narosa Publications, 2002
2. John R. Levine, Tony Mason, Doug Brown, Lex &yacc, O'reilly ,2/e, 1992
3. Keith Cooper, Linda Torczon, Engineering a compiler, Morgan Kaufmann, 2/e, 2011
4. Samuel, M. P., Philip, P. K., 1997, "Powder metallurgy tool electrodes for electric discharge machining", International journal of machine tools and manufacture, 37, 1625–33.

Course Outcomes:

After successful completion of the course the student will be able to:

1. Define and analyse various phases involved in designing a compiler(L1)
2. Compare and contrast between bottom-up and top-down parsing techniques(L2)
3. Illustrate the usage of Syntax Directed Definition in generating intermediate code(L3)

4. Identify various machine independent optimization techniques(L4)

5. Explore techniques involved in obtaining the final code(L4)

CO-PO Mapping:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	1	1											1	1	1
CO2	1	2			2								2	1	1
CO3	2	2	2										2	2	1
CO4	1	2	1										2	2	2
CO5	2	3	2										2	2	2

Note: 1 - Low Correlation 2 - Medium Correlation 3 - High Correlation

APPROVED IN:
BOS : 06-09-2021 **ACADEMIC COUNCIL: 01-04-2022**

SDG No. & Statement: SDG 8 “Economic growth”: In an increasingly automated world with computers, writing efficient compilers can help use them to solve several problems such as logistics, planning etc., leading to economic growth.

SDG Justification: