AMITY UNIVERSITY
UTTAR PRADESH

Course Title: **Compiler Construction**
Credit Units: **04**
Course Level: **UG**
Course Code: **CSE304**

L	T	P/S	SW/FW	TOTAL CREDIT UNITS
3		2		4

Course Objectives:

The objective of this course module is to describe the utilization of formal Grammar using Parser representations, especially those on bottom-up and top-down approaches and various algorithms; to learn techniques for designing parser using appropriate software.

The theory and practice of programming language translation, compilation, and run-time systems, organized around a significant programming project to build a compiler for a simple but nontrivial programming language. Modules, interfaces, tools. Data structures for tree languages. Lexical analysis, syntax analysis, abstract syntax. Symbol tables, semantic analysis. Translation, intermediate code. To understand, design and implement a parser. To understand, design code generation schemes. To understand optimization of codes and runtime environment

Pre-requisites: Computer architecture) or equivalent, (Data structures and algorithms) or equivalent, (Systems programming) or equivalent

- Familiarity with Java

Course Contents/Syllabus:

	Weightage (%)
Module I :	22%
Compilers – Analysis of the source program – Phases of a compiler – Cousins of the Compiler – Grouping of Phases – Compiler construction tools - Lexical Analysis - Role of Lexical Analyzer – Input Buffering – Specification of Tokens. Text formatter, Text Editors, Phases and Passes, FSM & RE's and their Analysis, application to Lexical Implementation of Lexical Analyzers, Lexical-	

Analyzer Generator, Lex – Compiler including case study, Formal Grammar and their application to Syntax Analysis, BNF Notation, YACC including case study. The Syntactic specification of Languages: CFG, Derivation and Parse Trees, Capabilities of CFG.	
Module II :	22%
Role of the parser –Writing Grammars –Context-Free Grammars – Top Down parsing - Recursive Descent Parsing - Predictive Parsing – Bottom-up parsing - Shift Reduce Parsing – Operator Precedent Parsing - LR Parsers - SLR Parser - Canonical LR Parser .	
Module III:	22%
LR Parsers, the canonical collection of LR(0) items, constructing SLR Parsing Tables, Constructing canonical LR Parsing tables and LALR parsing tables, An Automatic Parser Generator, Implementation of LR parsing tables, Constructing LALR sets of items.	
Module IV :	19%
Issues in the design of code generator – The target machine – Runtime Storage management – Basic Blocks and Flow Graphs – Next-use Information – A simple Code generator – DAG representation of Basic Blocks – Peephole Optimization Syntax directed Translation Schemes, Implementation of Syntax directed translators, Intermediate Code, Postfix notation, Parse Trees and Syntax Trees, Three address Code, Quadruple & Triples, Translation of Assignment Statements, Boolean expressions, Control Statements, Postfix Translation, Translation with a Top Down Parser, Array references in Arithmetic expressions, Procedure Calls, Declarations and Case statements Translations. Data Structure for Symbol Tables, representing scope information. Run Time Administration: Implementation of simple Stack allocation scheme, storage allocation in block structured language.	
Module V :	15%
Principal Sources of Optimization – Optimization of basic Blocks – Introduction to Global Data Flow Analysis – Runtime Environments – Source Language issues – Storage Organization – Storage Allocation strategies – Access to non-local names – Parameter Passing. Lexical phase errors, syntax phase errors, semantic errors Code Optimization: Loop optimization, the DAG representation of basic blocks, value numbers and Algebraic Laws, Global Data – Flow Analysis	

Compiler Construction Lab

1 Consider the following regular expressions:
a) (0 + 1)* + 0*1*
b) (ab*c + (def)+ + a*d+e)+
c) ((a + b)*(c + d)*)+ + ab*c*d
Write separate programs for each of the regular expressions mentioned above.
2 Design a Lexical analyzer for identifying different types of token used in C language.
4 Write a program which accepts a regular expression from the user and generates a regular grammar which is equivalent to the R.E. entered by user. The grammar will be printed to a text file, with only one production rule in each line. Also, make sure that all production rules are displayed in compact forms e.g. the production rules:
S--> aB, S--> cd
S--> PQ
Should be written as
S--> aB | cd | PQ
And not as three different production rules. Also, there should not be any repetition of production rules.
5 Write a program to eliminate left recursion
6 Write a program for Recursive Descent Calculator.
7 Write that recognizes different a program types of English words
8 Consider the following grammar:
S --> ABC
A--> abA | ab
B--> b | BC
C--> c | cC
Following any suitable parsing technique(prefer top-down), design a parser which accepts a string and tells whether the string is accepted by above grammar or not.
10 Write a program which accepts a regular grammar with no left-recursion, and no null-production rules, and then it accepts a string and reports whether the string is accepted by the grammar or not.

11 Design a parser which accepts a mathematical expression (containing integers only). If the expression is valid, then evaluate the expression else report that the expression is invalid.
[Note: Design first the Grammar and then implement using Shift-Reduce parsing technique. Your program should generate an output file clearly showing each step of parsing/evaluation of the intermediate sub-expressions.
12 Open Ended program: Designing of various type parser

Student Learning Outcomes: After completion of this course student will be able to

- Apply knowledge of mathematics, science, engineering and computing appropriate to the discipline.
- Analyze a problem, and identify and define the computing requirements appropriate to its solution. Apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.
- Apply design and development principles in the construction of software systems.

Pedagogy for Course Delivery:

1. Classroom teaching using White board and Presentations.
2. Assignments and Tutorials for continuous assessment.

Assessment/ Examination Scheme:

Theory L/T (%)	**Lab/Practical/Studio (%)**	**Total**
75	25	100

Theory Assessment (L&T):

Continuous Assessment/Internal Assessment					End Term Examination
Components (Drop down)	**Attendance**	**Class Test**	**Home Assignment**	**Presentation/Viva**	
Weightage (%)	5	10	7	8	70

Lab Assessment

Continuous Assessment/Internal Assessment					End Term Examination
Components (Drop down)	**Attendance**	**Lab Record**	**Performance**	**Viva**	
Weightage (%)	5	10	10	5	70

Text & References for Course Lab:

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman Compilers: Principles, Techniques, & Tools, Second Edition Boston: Addison-Wesley, 2007.

REFERENCES Books:

1. Compiler Construction A.A.Puntambekar - 2009 - 572 pages
2. Compiler Construction Principles And Practice By Kenneth C Louden Turbo Codes: ... Compiler Design 2013.
3. "Introduction to Compiler Techniques", TATA McGraw Hill Publishing Co. ..... J.P. Bennet Second Edition aug. 7, 2013