Gyanmanjari Innovative University

Course Syllabus
Gyanmanjari Institute of Technology
Semester-7 (B. Tech)

Subject: Compiler Design – BETCE17334

Type of course: Professional Core

Prerequisite: Basic knowledge of programming, data structures, discrete mathematics, theory of computation, and operating system concepts.

Rationale:

The Compiler Design syllabus is designed to provide students with a comprehensive understanding of how high-level programming languages are translated into machine-executable code, covering all major phases such as lexical, syntax, and semantic analysis, intermediate code generation, optimization, and final code generation. It builds a strong theoretical foundation using concepts from automata, grammars, and data structures while also emphasizing practical implementation using tools like LEX and YACC. This course equips students with essential skills in language processing, problem-solving, and system-level thinking, which are crucial for developing compilers, interpreters, and other advanced software systems.

Teaching and Examination Scheme:

Teaching Scheme			Credits	Examination Marks					Total Marks
				Theory Marks		Practical Marks		CA	
CI	T	P	C	ESE	MSE	V	P	ALA	
4	0	2	5	60	30	10	20	30	150

Legends: CI-ClassRoom Instructions; T – Tutorial; P - Practical; C – Credit; ESE - End Semester Examination; MSE- Mid Semester Examination; V – Viva; CA - Continuous Assessment; ALA- Active Learning Activities.

GYANMANJARI INNOVATIVE UNIVERSITY BHAVNAGAR

Course Content:

Sr. No	Course Content	Hrs.	% Weightage
1	**Introduction and Lexical Analysis** Compilers vs. Interpreters, The structure of a compiler (Phases and Passes), Front-end and Back-end, Compiler construction tools (Lex, Yacc). Role of the Lexical Analyzer, Input buffering, Specification of tokens (Regular Expressions), Recognition of tokens, Finite Automata (DFA, NFA), Conversion of Regular Expression to NFA and NFA to DFA, Minimization of DFA. Panic mode recovery.	12	20%
2	**Syntax Analysis (Parsing):** Context-Free Grammar (CFG), Derivations, Parse trees, Ambiguity, Left recursion elimination, Left factoring. Recursive descent parsing, Predictive parsing, LL (1) grammars and parsing tables. Shift-reduce parsing, Operator-precedence parsing. The canonical collection of LR (0) items, SLR parsing, Canonical LR parsing, LALR parsing tables. Error recovery in parsing (Panic mode, Phrase-level recovery, Error productions).	15	25%
3	**Syntax-Directed Translation and Semantic Analysis** Inherited and synthesized attributes, Evaluating an SDD at the nodes of a parse tree, Evaluation orders for SDD. Implementation of SDTS, Bottom-up evaluation of S-attributed definitions, L-attributed definitions. Type systems, Specification of a simple type checker, Equivalence of type expressions, Type conversions, Symbol Table Organization and Management. Source language issues, Storage organization, Storage allocation strategies (Static, Stack, Heap), Activation records, Access to non-local names, Parameter passing mechanisms.	12	20%
4	**Intermediate Code Generation** Graphical representations (Syntax trees, Abstract Syntax Tree (AST), Directed Acyclic Graphs), Three-address code, Types of three-address statements (Quadruples, Triples, Indirect triples). Declarations, Assignment statements, Boolean expressions, Control flow statements, Backpatching, Switch statements, Procedure calls.	09	15%

5	**Code Optimization and Code Generation** Introduction to principal sources of optimization (Machine-independent and Machine-dependent), Loop optimization (Code motion, Induction variable elimination, Reduction in strength), Basic blocks and flow graphs. Introduction to data-flow routing, Reaching definitions, Live variable analysis, Available expressions. Issues in the design of a code generator, The target machine, Run-time storage management, Next-use information, A simple code generator, Register allocation and assignment, The DAG representation of basic blocks, Peephole optimization, Introduction to modern compilers like LLVM and GCC.	12	20%

Continuous Assessment:

Sr. No	Active Learning Activities	Marks
1	**Mini Language Grammar and Parser Design Analysis:** In this task, students will design a small mini language of their own containing identifiers, keywords, operators, expressions, and simple statements. Students will prepare lexical rules, regular expressions, token classification, finite automata representation, CFG, parse tree, and suitable parsing method selection such as LL(1), SLR, CLR, or LALR with justification. The activity should be submitted individually on the GMIU Web Portal.	10
2	**Source Code to Compiler Representation Mapping:** In this task, students will select any small program written in C, Java, Python, or any familiar programming language and analyze it from the compiler point of view. Students will identify variables, data types, scope, symbol table entries, semantic checking requirements, syntax tree/AST structure, and possible three-address code representation. The activity should be submitted individually with diagrams and explanation on the GMIU Web Portal.	10
3	**Code Optimization Review of Real Program Logic:** In this task, students will select any small code snippet from their own previous programs or open-source examples and identify optimization opportunities. Students will analyze redundant statements, repeated expressions, unnecessary variables, loop inefficiencies, dead code, strength reduction possibilities, and register usage considerations. They must prepare before-optimization and after-optimization logic with proper justification. The activity should be submitted individually on the GMIU Web Portal.	10
	Total	30

Suggested Specification table with Marks (Theory):60

Distribution of Theory Marks (Revised Bloom's Taxonomy)						
Level	**Remembrance (R)**	**Understanding (U)**	**Application (A)**	**Analyze (N)**	**Evaluate (E)**	**Create (C)**
Weightage %	15%	20%	25%	20%	10%	10%

Course Outcome:

After learning the course the students should be able to:	
CO1	Explain compiler structure, phases, and lexical analysis techniques.
CO2	Apply CFG and parsing techniques for syntax analysis.
CO3	Analyze semantic rules, type checking, and symbol table management.
CO4	Generate intermediate code using AST, DAG, and three-address code.
CO5	Optimize code and design efficient target code generation.

List of Practical:

Sr. No	Description	Unit No	Hrs.
1	Develop a program that reads a source code file and identifies tokens such as keywords, identifiers, constants, and operators.	01	02
2	Write a program to recognize tokens using Regular Expressions for identifiers, numbers, and symbols.	01	02
3	Implement a lexical analyzer using Finite Automata to validate input strings.	01	02
4	Write a program to compute FIRST and FOLLOW sets for a given Context-Free Grammar.	02	02
5	Develop a program to implement LL (1) parsing technique for a given grammar.	02	02
6	Implement a recursive descent parser for arithmetic expressions.	02	02

7	Develop a program to simulate shift-reduce parsing for a given input string.	02	02
8	Write a program to perform operator precedence parsing for arithmetic expressions.	02	02
9	Develop a program to construct an Abstract Syntax Tree (AST) for a given expression.	03	02
10	Develop a program to create and manage a symbol table using suitable data structures (including scope handling).	03	02
11	Write a program to generate three-address code for arithmetic expressions.	03	02
12	Implement a program to perform type checking for given expressions.	04	02
13	Develop a program to perform code optimization techniques such as constant folding, dead code elimination, and common subexpression elimination.	05	02
14	Write a program to construct a control flow graph and perform data flow analysis (live variables or reaching definitions).	05	02
15	Develop a program to generate simple target machine code (assembly-like instructions) from intermediate code with basic register allocation.	05	02
		Total	30

Instructional Method:

The course delivery method will depend upon the requirement of content and need of students. The teacher in addition to conventional teaching method by black board, may also use any of tools such as demonstration, role play, Quiz, brainstorming, MOOCs etc.

From the content 10% topics are suggested for flipped mode instruction.

Students will use supplementary resources such as online videos, NPTEL/SWAYAM videos, e-courses, Virtual Laboratory.

The internal evaluation will be done on the basis of Active Learning Assignment.

Practical/Viva examination will be conducted at the end of semester for evaluation of performance of students in laboratory.

Reference Books:

[1] Compiler Design by Rajesh Moraiya, DreamTech
[2] Compiler Design by Punpambekar. A. A, Technical Publisher.
[3] Compiler Design by Dr. O.G. Kakde , Laxmi Publications Pvt Ltd
[4] Principles of Compiler Desgin by Alfred V. Aho, Narosa Publishing House
[5] Compiler Design by Ikvindorpal Singh & Sapandeep Kaur Dhillon, Khanna Publishing House