VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

Version 08/06-06

Consolidated Academic Administration Plan for the Course

System Programming and Compiler Design (Core) Sem. V – Program Computer Engineering 2025-26 Odd Semester

Faculty – Pankaj Vanwari (Cluster Mentor)

The academic resources available in VIT –

VMIS (ERP)	V-Refer and V-Live	VIT Library	VAC & MOOC Courses
Institute & Department Vision and Mission	Former IA question papers and solutions (prepared by faculty)	Former IA question papers solutions - hardcopy	Value Added Courses (VAC) are conducted throughout the semester & in the semester break - Enrol for the VACs
Program Educational Objectives (PEO)	MU end semester examination question papers and solutions (prepared by faculty)	MU end semester exam question paper & solutions - by faculty, hardcopy	
Program Specific Outcome (PSO)	Class notes and Digital Content for the subject (scanned / typed by faculty)	All text books, reference books, e -books mentioned in the syllabus & AAP	Online courses from NPTEL, Coursera etc. are pursued throughout the semester - Register for the course & get certified
Program Outcome (PO)	Comprehensive question bank, EQ, GQ, PPT, Class Test papers	Technical journals and magazines for reference	
Departmental Knowledge Map	Academic Administration Plan & Beyond Syllabus Activity report	VIT library has many resources e:g :- IEEE, Nimbus, xplore, EBSCO etc.	Watch former lectures captured in LMS at VIT

1.a Course Objectives (Write in detail – as per NBA guidelines)

Cognitive	What do you want students to know?	Students should know the procedures for the designing of software systems and to understand the compilation process.
Affective	What do you want students to think / care about?	Students should take care about carrying out optimization on the immediate code generated with appropriate semantic actions and implementations.
Behavioural	What do you want students to be able to do?	Students should be able to solve problems universally encountered in designing a language translator, regardless of the source or target machine.

Advice to Students:

Attend every class!!! Missing even one class can have a substantial effect on your ability to understand the course. Be prepared to think and concentrate, in the class and outside. I will try to make the class very interactive. Participate in the class discussions. Ask questions when you don't understand something. Keep up with the class readings. Start assignments and homework early. Meet me in office hour to discuss ideas, solutions or to check if, what you understand is correct.

The v-Refer Link

http://vidyalankarlive.com/vrefer/index.php/apps/files/?dir=/vRefer/CMPN/SEM%20V/2025-26/Verticals/PC-PCC/Subjects/System%20Programming%20and%20Compiler%20Design_PCCE15/PV&fileid=1080513

Collaboration Policy:

We encourage discussion between students regarding the course material. However, no discussion of any sort is allowed with anyone on the assignment and homework for the class. If you find solution to some problems in a book or on the internet, you may use their idea for the solution; provided you acknowledge the source

(name and page in the book or the website, if the idea is found on the internet). Even though you are allowed to use ideas from another source, you must write the solution in your own words. If you are unsure whether or not certain kinds of collaboration is possible, please ask the teacher.

1.b Course Outcome (CO) Statements and Module-Wise Mapping (follow NBA guideline)

CO No.	Statements	Related Module/s
CO1	Recall the basic concepts and functions of system software components such as compilers, assemblers, loaders, and linkers.	1, 2, 3, 4, 5, 6
CO2	Explain the working of each compiler phase and system program with appropriate models and examples.	1, 2, 3, 4, 5, 6
CO3	Apply lexical, syntax, and semantic analysis techniques to construct components of a compiler.	1, 2, 3, 4, 5, 6
CO4	Analyze intermediate code representations, parsing strategies, and code generation flow to identify design trade-offs.	1, 2, 3, 4, 5, 6
CO5	Evaluate optimization techniques and compiler design choices in terms of efficiency and performance.	3, 5
CO6	Design basic system software modules such as lexical analyzers, parsers, or simple code generators using appropriate algorithms and data structures.	4, 6

1.c Course Outcome (CO) Statements and Module-Wise Mapping (follow NBA guideline)

	Mapped to Learning Outcomes
CO1	1.1, 2.1, 3.1, 4.1, 5.1, 6.1
CO2	1.2, 2.2, 3.2, 4.2, 5.2, 6.2
CO3	1.3, 2.3, 3.3, 4.3, 5.3, 6.3
CO4	1.4, 2.4, 3.4, 4.4, 5.4, 6.4
CO5	3.5, 5.5
CO6	6.5

1.d Mapping of COs with POs (mark S: Strong, M: Moderate, W: Weak, Dash '–': not mapped)

(List of POs is available in V-refer)

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
	Knowledge A	Analysis	Design	Investigation	Modern Tools	Society	Environment &sustainability	Ethics T	Teamwork C	Communication	Project Mgt	Life long learning
CO 1	S	M	-	-	-	-	-	-	-	-	-	W
CO 2	S	S	-	-	-	-	-	-	-	-	-	-

CO 3	S	S	M	-	M	-	-	-	-	-	-	-
CO 4	M	S	-	M	M	-	-	-	-	-	-	-
CO 5	M	S	-	M	M	-	-	-	-	-	-	-
CO6	S	M	S	M	S	-	-	-	M	M	M	M

1.e **Mapping of COs with PSOs (mark S: Strong, M: Moderate, W: Weak, Dash '–': not mapped)**

	PSO 1	PSO 2	PSO 3	PSO 4
CO 1	M	-	W	
CO 2	M	-	-	
CO 3	S	W	-	
CO 4	M	W	-	
CO 5	M	W	-	
CO6	S	W	M	

1.f **Teaching and Examination Scheme (As specified by the autonomous syllabus) for the Course**

Verticals	BSC/ESC	Program Courses	Multidisciplinary Courses	Skill Courses	HSSM	Experiential Learning	Liberal Learning
Tick suitable category		√					

Subject Code	Subject Name	**Teaching Scheme**			Credits Assigned			
		Theory	Practical	Tutorial	Theory	TW/Practical	Tutorial	Total
PCCE15	System Programming and Compiler Design	3	-	-	3	-	-	3

Subject Code	Subject Name				**Examination Scheme**				
		Theory			**Total (Theory)**	**Practical**			**Total (Practical)**
		ISA	MSE	ESE		ISA	ESE	ORAL	
PCCE15	System Programming and Compiler Design	20	30	50	100	-	-	-	-

Subject Code	Subject Name	**MSE-1***		
		Q, No	Module wise % Distribution	Relevant to Bloom Taxonomy
PCCE15	System Programming and Compiler Design	Q1-5*3 =15 marks	Module 1:40%	L1,L2,L3
		Q2-10*2=20 marks	Module 2:60%	L1,L2,L3,L4

	Subject Name	**MSE-2***

Subject Code		Q, No	Module wise % Distribution	Relevant to Bloom Taxonomy
PCCE15	System Programming and Compiler Design	Q1-5*3 =15 marks	Module 3:50%	L1,L2,L3
		Q2-10*2=20 marks	Module 4:50%	L1,L2,L3,L4

Subject Code	Subject Name	ESE#		
		Q, No	Distribution	Relevant to Bloom Taxonomy
PCCE15	System Programming and Compiler Design	Q1-10*2 =20 marks	Module 1: 10% Module 2: 15%	L1,L2,L3 L1,L2,L3
		Q2-10*2=20 marks	Module 3: 15% Module 4: 15%	L1,L2,L3,L4 L1,L2,L3
		Q3-10*2 =20 mark	Module 5: 25%	L1,L2,L3
		Q4-10*2=20 marks	Module 6: 20%	L1,L2,L3,L4

*** Recommended distribution: -** 30 Marks from Assignments, 40 marks based on assignments with slightly enhance difficulty /complex, 30 marks from thought provoking

 # Recommended distribution: - 30 Marks from Assignments, 40 marks based on assignments/MSE with slightly enhance difficulty /complex, 30 marks from thought provoking

1.g Faculty-Wise Distribution of all Lecture-Practical-Tutorial Hours for the Course

Divisions	Lecture (Hrs.)	Practical (Hrs.)				Tutorial (Hrs.)			
		Batch 1	Batch 2	Batch 3	Batch 4	Batch 1	Batch 2	Batch 3	Batch 4
A	PV (3)	-	-	-	-	-	-	-	-
B	PV (3)	-	-	-	-	-	-	-	-
C	PV (3)	-	-	-	-	-	-	-	-

1.h Office Hours (Faculty will be available in office in this duration for solving students' query)

Division	Day	Time (at least 1 Hr. / Division)	Venue (Office Room No.)
A	Monday	03:45 PM onwards	M-209
B	Tuesday	03:45 PM onwards	M-209
C	Wednesday	03:45 PM onwards	M-209

2.a Syllabus: Module Wise Teaching Hours and % Weightage in autonomous syllabus Question Paper

Module No.	Module Title and Brief Details	Teaching Hrs. for each module	% Weightage in autonomous syllabus Question Papers			Performance Indicator Mapping
			ISA	MSE	ESE	
1	Introduction to System Software and Compiler Concept of System Software, Goals of system software, system program and system programming, Introduction to various system programs such as Assembler, Macro processor, Loader, Linker, Compiler, Interpreter, Device Drivers, Operating system, Editors, Debuggers. Introduction to compilers, Phases of compilers	4	5	33	10	1.4.1, 2.1.1, 1.4.2, 2.1.2, 6.1.1
Learning Outcome-1.1	Define system software and list different types of system programs.					
Learning Outcome-1.2	Explain the functions of assemblers, loaders, linkers, and compilers.					
Learning Outcome-1.3	Illustrate the workflow of a compiler and describe how various system programs interact.					
Learning Outcome-1.4	Compare and contrast system programs based on their role in program execution.					
2	Lexical Analysis Role of Finite State Automata in Lexical Analysis, Design of Lexical analyzer, data structures used.	4	10	33	10	1.4.1, 2.1.1, 1.4.2, 2.1.2, 6.1.1, 1.4.3, 2.2.3, 5.2.2, 2.1.4, 2.4.4, 5.3.1
Learning Outcome-2.1	Describe the role of lexical analysis in a compiler.					
Learning Outcome-2.2	Explain how finite automata are used in lexical analysis.					
Learning Outcome-2.3	Construct simple lexical analyzers for tokenizing input strings.					
Learning Outcome-2.4	Analyze the efficiency and correctness of a given lexical analyzer design.					
3	Syntax and Semantic Analysis Syntax Analysis- Role of Context Free Grammar in Syntax analysis, Types of Parsers: Top down parser- LL(1), Bottom up parser- SR Parser, Operator precedence parser, SLR. Semantic Analysis, Syntax directed definitions	15	35	34	33	1.4.1, 2.1.1, 1.4.2, 2.1.2, 6.1.1, 1.4.3, 2.2.3, 5.2.2, 2.1.4, 2.4.4, 5.3.1, 2.1.5, 5.3.2, 7.1.2
Learning Outcome-3.1	Recall the types of parsers and grammar notations used in syntax analysis.					
Learning Outcome-3.2	Interpret parse trees and syntax-directed definitions.					
Learning Outcome-3.3	Implement LL(1) or SLR parsers for a given grammar.					

Module No.	Module Title and Brief Details	Teaching Hrs. for each module	% Weightage in autonomous syllabus Question Papers			Performance Indicator Mapping
			ISA	MSE	ESE	
Learning Outcome-3.4	Examine parsing errors and ambiguity in context-free grammars.					
Learning Outcome-3.5	Justify the choice of parser for a specific language or grammar structure.					
4	Intermediate Code Generation Types of Intermediate codes: Syntax tree, Postfix notation, three address codes: Triples and Quadruples, indirect triple. Additional: Macro Processer Design	9	20	50	20	1.4.1, 2.1.1, 1.4.2, 2.1.2, 6.1.1, 1.4.3, 2.2.3, 5.2.2, 2.1.4, 2.4.4, 5.3.1
Learning Outcome-4.1	List different intermediate code representations.					
Learning Outcome-4.2	Describe the purpose of using intermediate code in compilation.					
Learning Outcome-4.3	Generate intermediate code such as three-address code or syntax trees from a given source code snippet.					
Learning Outcome-4.4	Compare intermediate representations based on ease of translation and optimization potential.					
5	Code Optimization Need and sources of optimization, Code optimization techniques: Machine Dependent and Machine Independent.	4	10	50	10	1.4.1, 2.1.1, 1.4.2, 2.1.2, 6.1.1, 1.4.3, 2.2.3, 5.2.2, 2.1.4, 2.4.4, 5.3.1, 2.1.5, 5.3.2, 7.1.2
Learning Outcome-5.1	Identify common code optimization techniques.					
Learning Outcome-5.2	Explain the difference between machine-dependent and machine-independent optimizations.					
Learning Outcome-5.3	Apply basic optimizations like constant folding or dead code elimination.					
Learning Outcome-5.4	Analyze how different optimizations affect performance and resource usage.					
Learning Outcome-5.5	Assess the trade-offs in applying aggressive vs. conservative optimization strategies.					
6	Code Generation Issues in the design of code generator, code generation algorithm. Basic block and flow graph. Addition: Assembler Design.	9	20	-	17	1.4.1, 2.1.1, 1.4.2, 2.1.2, 6.1.1, 1.4.3, 2.2.3, 5.2.2, 2.1.4, 2.4.4, 5.3.1, 1.4.5, 2.2.5, 3.2.5
Learning Outcome-6.1	Recall the components involved in code generation.					
Learning Outcome-6.2	Explain the concept of basic blocks and flow graphs.					

Module No.	Module Title and Brief Details	Teaching Hrs. for each module	% Weightage in autonomous syllabus Question Papers			Performance Indicator Mapping
			ISA	MSE	ESE	
Learning Outcome-6.3	Design a simple code generation algorithm using intermediate code.					
Learning Outcome-6.4	Break down code generation problems into target-specific issues.					
Learning Outcome-6.5	Develop a prototype code generator for a basic programming language subset.					
	Total	**45**				

Note: - As an attachment Annexure is required for assessment criteria of learning outcomes.

2.b Prerequisite Courses

No.	Semester	Name of the Course	Topic/s
1	II	Structured Programming Approach	C programming concept
2	V	Theoretical Computer Science	Finite Automata, Grammar Concept

2.c Relevance to Future Courses

No.	Semester	Name of the Course
1	VII	Natural Language Processing

2.d

See :- Identify real life scenarios/examples which uses the knowledge of the subject ,(Discussion on how to prepare examples and case studies e.g. "Boeing Plane": C Programming Language – Intro to Computer Science – Harvard's CS50 (2018) – Bing video)

Real Life Scenario	Concept Used
Syntax Highlighting in IDEs	Integrated Development Environments (IDEs) use lexical analysis to highlight syntax, making code easier to read and debug. By breaking down the source code into tokens, the IDE can apply different colors and fonts to keywords, variables, and operators.
Code Formatting Tools	Tools like Prettier and ESLint for JavaScript use syntax analysis to enforce coding standards and automatically format code. They build parse trees to understand the structure of the code, ensuring consistent style and readability.
Cross-Compilation	Code generation enables cross-compilation, where code written on one platform is compiled to run on another. This is essential for developing software for different operating systems or hardware architectures, such as mobile apps that run on both iOS and Android.
Educational Tools	Teaching tools like online code editors and learning platforms use compilers to provide instant feedback to

	students, allowing them to learn programming concepts interactively.
Just-In-Time (JIT) Compilation	JIT compilers, used in environments like Java's JVM and JavaScript engines in browsers, compile code at runtime for improved performance and flexibility, enabling faster execution and adaptation to runtime conditions.

3 Past Results – Division-Wise

Details	Target – DEC 2025	MAY 2025	DEC 2024	MAY 2024
Course Passing % – Average of 2 Divisions	100%	99.34%	100%	100
Marks Obtained by Course Topper (mark/100)	100/100	100/100	100/100	92

	Division A		Division B	
Year	Initials of Teacher	% Result	Initials of Teacher	% Result
May 2025	PJP	100	PJP	99.34
Dec 2024	RSG	100	RSG	100
May 2024	PV	100	PV	98.6

4 All the Learning Resources – Books and E-Resources

4.a List of Textbooks (T – Symbol for Textbooks) to be Referred by Students

Sr. No	Textbook Titles	Author/s	Publisher	Edition	Module Nos.	Available in our Library
1	Systems programming	D. M Dhamdhere	Tata McGraw Hill	Second Edition	1,2,3,4	Yes
2	Compilers Principles, Techniques and Tools	A. V. Aho, R. Shethi, Monica Lam, J.D. Ulman	Pearson Education	Second Edition	5,6	Yes
3	Systems Programming	J. J. Donovan:	Tata McGraw Hill Publishing Company	Second Edition	1,2,3,4	Yes

4.b List of Reference Books (R – Symbol for Reference Books) to be Referred by Students

Sr. No	Reference Book Titles	Author/s	Publisher	Edition	Module Nos.	Available in our Library
1	Lex &yacc	John R. Levine, Tony Mason & Doug Brown	O'Reilly	II	5,6	No

2	Compiler construction	D,M.Dhamdhere	MACMILLAM	II	5,6	Yes
3	Compiler construction : principles and practices	Kenneth C.Louden	CENGAGE Learning	II	5,6	No
4	System software : An introduction to system programming	Leland L. Beck	Pearson	II	1,2,3,4	No

4.c List of E - Books (E – Symbol for E-Books) to be Referred by Students

Sr. No	E- Book Titles	Author/s	Publisher	Edition	Module Nos.	Available in our Library
1	Compiler Construction	William Waite and Gerhard Goos	Springer	II	2,3,4,5	No
2	Compiler Design in C	Allen Holub	TMH	III	All	No
3	Basic of Compiler Construction	Torben Mogensen	Springer	II	3,4,5	No

4.d Reading latest / top rated research papers (at least 5 papers)

Name of Paper	Name of Authors (Background)	Published in		Problem Statement	Available in our Library
		Date	Journal		
"Spoq: Scaling Machine-Checkable Systems Verification in Coq"	Xupeng Li, Xuheng Li, Wei Qiang, Ronghui Gu, and Jason Nieh	July 2023	17th USENIX Symposium on Operating Systems Design and Implementation (OSDI) Conference	There exists a problem of addressing the complexity and vulnerabilities of large-scale system software like operating systems and hypervisors. Traditional verification methods face high proof costs, challenges in handling complex software semantics, and gaps between verified specifications and running code. There must be a system that automates translating code, generating specifications, and proving correctness, ensuring functional correctness and higher-level security guarantees without manual retrofitting.	No
Multiple input parsing and lexical analysis	Elizabeth Scott, Adrian Johnstone, And Robert Walsh	19 July 2023	ACM Transactions on Programming Languages and Systems	The document addresses the challenges in traditional language analysis where lexical and syntax analysis are treated as separate procedures. This separation often leads to inefficiencies and difficulties in handling ambiguous cases where the interpretation of words depends on the context of sentences. It introduces MGLL, an extension of generalized parsing, enabling	No

				multiple input strings to be parsed concurrently, along with an enhanced approach to lexical analysis that utilizes this multi-parsing capability. The goal is to provide a more flexible and efficient framework for formal language specification, parsing multiple lexicalizations, and resolving ambiguities in both programming and natural languages.	
Unveiling code pre-trained models: Investigating syntax and semantics capacities	Wei Ma, Shangqing Liu, Mengjie Zhao, Xiaofei Xie, Wenhang Wang	August 2024	ACM Transactions on Software Engineering and Methodology	There is a necessity to understand both syntax and semantics of programming code. One has to analyze various code models using probing tasks to evaluate their proficiency in learning core code structures like Abstract Syntax Trees (ASTs), Control Dependency Graphs (CDGs), and Data Dependency Graphs (DDGs), highlighting their strengths, limitations, and implications for future advancements in code intelligence.	No
A Hybrid Machine Learning Model for Code Optimization	Yacine Hakimi, Riyadh Baghdadi, Yacine Challal	September 2023	International Journal of Parallel Programming	Develop a hybrid machine learning model that combines the strengths of deep learning for feature extraction with the efficiency of classical machine learning for small datasets. Evaluate this model on three tasks—device mapping, thread coarsening, and algorithm classification—demonstrating superior or comparable performance to state-of-the-art models with reduced data requirements and computational complexity.	No
C4CAM: A Compiler for CAM-based In-memory Accelerators	Hamid Farzaneh, João Paulo C. de Lima, Mengyuan Li, Asif Ali Khan, Xiaobo Sharon Hu, Jeronimo Castrillon	April 2024	Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems	Develop a novel compiler framework that automates code generation and optimization for CAM-based accelerators, enabling broader accessibility and efficient design-space exploration.	No

4.e **Based on research paper an identify the current Problem statement**

Problem Statement			Used in				
	Quiz	Assignment	Lab	Mini Project	Poster Presentation	Test	Any Other
How system software interacts with hardware and other software components, ensuring seamless and efficient operations.	√						

4.f **Identify Companies / Industries which use the knowledge of the subject and thus may provide Internships and final Placements**

Name of the Company	To be / Contacted for		
	Student Internship	Student Final Placement	Faculty Internship
NVIDIA, AWS, Qualcomm	√	--	--

4.g **Identify suitable relevant TOP Guest Speakers from Industry, Example: - (CS50 Lecture by Mark Zuckerberg - 7 December 2005 - YouTube)**

Name of the Identified Guest Speaker	Designation	Name of the Company
Amanda Silver https://devblogs.microsoft.com/	Vice President of Product, Developer Tools at Microsoft.	Microsoft
Michael Hind https://www.research.ibm.com/	Distinguished Research Staff Member and Program Director of the Future of Computing group at IBM Research.	IBM Research

4.h **Identify relevant technical competitions to participate [Competitions -Paper Presentations, Projects, Hackathons, IVs etc..]**

Name of the Relevant Technical Competition Identified to participate	Organized by	Date of the Event
HackMIT	https://hackmit.org/	September 2025 (Annually)

4.i **Identify faculty in TOP schools / Universities who are teaching same / similar subject and develop rapport e.g. Exchange Lecture Material (Assignments / Tests / Project etc..), Joint Paper Publication**

University	Name of the Course	Name of Faculty	Type of Collaboration		
			Exchange of Lecture Material	Joint Publication/ Research	Other
University of California, Berkeley	Distributed systems, cloud computing, and system software.	Professor John D. Kubiatowicz	√	--	--

4.j **Module Best Available in – Title of the best resource [from *4.a* to *4.d* in this AAP] and other details as necessary**

Module No.	Title of the Module	Textbook	Mention the Tile					
			Reference Book	E-books	Journal	E-Journal	Available in our Library	V-refer
1	Introduction to System Software and Compiler	T1, 3	R4	E2	-	-	-	-
2	Lexical Analysis	T1, 3	R4	E1, 2	-	-	-	-
3	Syntax and Semantic Analysis	T1, 3	R4	E1, 2, 3	-	-	-	-
4	Intermediate Code Generation	T1, 3	R4	E1, 2, 3	-	-	-	-
5	Code Optimization	T5, 6	R1, 2, 3	E1, 2, 3	-	-	-	-
6	Code Generation	T5, 6	R1, 2, 3	E2	-	-	-	-

4.k **Referred to any top-rated university in that subject for content**

University	Name of the Course	Name of Faculty	Date of Delivery of the Course	Remarks
University of Washington	CSE 401/M501 Compiler Construction	Hal Perkins	Autumn 2023	https://courses.cs.washington.edu/courses/cse401/18au/

4.l **Faculty received any certification related to this subject. List of Certifications Identified / Done**

Course	Certifying Agency	No. of Hours	Level of the Course		Certification		Remarks
			Introductory	Advance Skill Development	Done on	Proposed to be on	
Compiler Design	NPTEL - Swayam	12 weeks	✓		May 2024		

4.m **Completed subject wise/cluster wise training with cluster mentor.**
List of relevant Refresher Course Identified / Done

Course	Certifying Agency (As suggested by DAB/Cluster Mentor/Industry/University other than MU)	Certification		Remarks
		Done on	Proposed to be on	
Pedagogy	Udemy	May 2020	-	Outcome Based Education & Academic Quality Assurance
PBL				
Sub. Content Training	NPTEL -Swayam	May 2024		Compiler Design

4.n Best Practices Identified and adopted

No.	Item	Best Practices Identified		
		IIT Kanpur	IIT Madras	IIT Kharagpur
1	Microsite	https://www.cse.iitk.ac.in/users/swarnendu/courses/spring2023-cs335/	http://www.cse.iitm.ac.in/~krishna/cs3300/	https://cse.iitkgp.ac.in/~bivasm/compiler2013.html
2	Video Lectures		✓	
3	Assignments		✓	✓
4	Mini Project			
5	Assessment Metric			
6	Quizzes		✓	
7	Labs/ Practical (PBL)			✓
8	Tests			
9	Peer Assessment			
10	Any Other	Slides		

4.o Web Links for Online Notes/YouTube/VIT Digital Content/VIT Lecture Capture/NPTEL Videos

Students can view lectures by VIT professors, captured through LMS 'Lecture Capture' in VIT campus for previous years.

No.	Websites / Links	Module Nos.
1	http://dragonbook.stanford.edu/lecture-notes/Stanford-CS143/03-Lexical-Analysis.pdf	5
2	https://lambda.uta.edu/cse5317/notes/node6.html	5
3	http://www.montefiore.ulg.ac.be/~geurts/Cours/compil/2011/03-syntaxanalysis-3.pdf	5

4.p Recommended MOOC Courses like Coursera / NPTEL / MIT-OCW / edX/VAC etc.

Sr. No.	MOOC Course Link	Course conducted by – Person / University / Institute / Industry	Course Duration	Certificate (Y / N)

1	https://ocw.mit.edu/courses/6-004-computation-structures-spring-2017/pages/c11/	MIT - Massachusetts Institute of Technology		
2	https://www.edx.org/course/compilers	Stanford University		Y
3	https://onlinecourses.nptel.ac.in/noc21_cs07/preview	By Prof. Santanu Chattopadhyay, IIT Kharagpur	12 WEEKS	Y

5 Consolidated Course Lesson Plan

	From (date/month/year)	From (date/month/year)	Total Number of Weeks
Semester Duration	07/07/2025	18/10/2025	15

Week	Lecture no.	Module No.	Lecture Topics / MSE / BSA planned to be covered	Actual date of Completion (Handwritten)	COs Mapped	Mapping Bloom Taxonomy level	Recommended Prior Viewing / Reading	
							Lecture No. (on LMS)	Chapter No./ Books/ Web Site
1	1,2	1	Concept of System Software, Goals of system software, system program and system programming, Introduction to various system programs such as Assembler, Macro processor,		CO1	Understand		T3,Chp#1 Pg-4-7
	3	1	Loader, Linker, Compiler, Interpreter, Device Drivers, Operating system, Editors, Debuggers.		CO1	Understand		T3,Chp#1 Pg-4-7
2	4,5	1	Introduction to compilers, Phases of compilers		CO1	Understand		T2 Chp#4 Pg-122-136
	6	2	Lexical Analysis- Role of Finite State Automata in Lexical Analysis,		CO2	Understand		T2 Chp#4 Pg-122-136 T2 Chp#5 Pg-150-167
3	7,8	2	Role of Finite State Automata in Lexical Analysis,		CO2	Understand		T2 Chp#4 Pg-122-136

Week	Lecture no.	Module No.	Lecture Topics / MSE / BSA planned to be covered	Actual date of Completion (Handwritten)	COs Mapped	Mapping Bloom Taxonomy level	Recommended Prior Viewing / Reading	
							Lecture No. (on LMS)	Chapter No./ Books/ Web Site
								T2 Chp#5 Pg-150-167
	9	2	Design of Lexical analyzer, data structures used.		CO2	Apply		T2 Chp#4 Pg-122-136
4	10,11	3	Syntax Analysis- Role of Context Free Grammar in Syntax analysis,		CO3	Understand		T2 Chp#5 Pg-150-167
	12	3	Basics of Grammar		CO3	Understand		--
5	13,14	3	Top-Down Parsing- Recursive descent		CO3	Apply		T2 Chp#4 Pg-122-136
	15	3	Concepts of First & follow		CO3	Understand		T2 Chp#5 Pg-150-167
6	16,17	3	LL-1 Parser Design		CO3	Apply		T2 Chp#4 Pg-122-136
	18	3	Design Problems on Top-down parsing		CO3	Apply		T2 Chp#5 Pg-150-167
7	19,20	3	Bottom-up Parsing- SR Parser, OPP parser design		CO3	Apply		T2 Chp#5 Pg-150-167
	21	3	SLR parser design		CO3	Apply		T2 Chp#4 Pg-122-136

Week	Lecture no.	Module No.	Lecture Topics / MSE / BSA planned to be covered	Actual date of Completion (Handwritten)	COs Mapped	Mapping Bloom Taxonomy level	Recommended Prior Viewing / Reading	
							Lecture No. (on LMS)	Chapter No./ Books/ Web Site
								T2 Chp#5 Pg-150-167
8	22,23	3	LR-1 & LALR parser design		CO3	Apply		T2 Chp#4 Pg-122-136 T2 Chp#5 Pg-150-167
	24	3	Semantic Analysis, Syntax directed definitions		CO3	Understand		T2 Chp#4 Pg-122-136 T2 Chp#5 Pg-150-167
9	25,26	4	Intermediate Code Generation: Types of Intermediate codes: Syntax tree, Postfix notation,		CO4	Understand		T2 Chp#5 Pg-168-178
	27	4	three address codes: representation		CO4	Understand		T2 Chp#1 Pg- 20-22
10	28,29	4	Triples and Quadruples, indirect triple.		CO4	Understand		T2 Chp#1 Pg-1-22
	30	4	Macro Processer Design-Macro definition & concepts		CO4	Understand		T3 Chp#3Pg-62-77
11	31,32	4	Features of macro, need of macro processor,		CO4	Understand		T3 Chp#3Pg-62-77

Week	Lecture no.	Module No.	Lecture Topics / MSE / BSA planned to be covered	Actual date of Completion (Handwritten)	COs Mapped	Mapping Bloom Taxonomy level	Recommended Prior Viewing / Reading	
							Lecture No. (on LMS)	Chapter No./ Books/ Web Site
	33	4	Design of two pass macro processor		CO4	Apply		T3 Chp#3Pg-62-77
12	34,35	5	Code Optimization: Need and sources of optimization,		CO5	Understand		T2 Chp#3 Pg- 84-134
	36	5	Code optimization techniques: Machine Dependent		CO5	Understand		T2 Chp#4 Pg- 181-190
13	37,38	5	Machine Independent.		CO5	Understand		T2 Chp#4 Pg- 181-190
	39	6	Code Generation: Issues in the design of code generator		CO6	Understand		T2 Chp#4 Pg- 195-202
14	40,41	6	code generation algorithm		CO6	Understand		R1 Chp#3 Pg- 30-36
	42	6	Basic block and flow graph		CO6	Understand		T2 Chp#4 Pg- 195-202
15	43, 44	6	Introduction to Assembler-features of ALP		CO6	Understand		T3 Chp#3Pg-62-77
	45	6	Assembler Design- Two pass, Roles of Assembler in code generation		CO6	Apply		T3 Chp#3Pg-62-77

6 Rubric for Grading and Marking of Term Work (inform students at the beginning of semester)

- Activity/ies should be designed as per reference of credit structure.
- If the subject is of 2 credit, activity/ assignment should be design for 2 hours with appropriate complexity and engaging time.

Theory (ISA=20)												
Class Participation	Activity-1	Activity-2	Activity-3	Activity-4	Activity-5	Activity-6	Activity-7	Activity-8	Activity-9	Activity-10	Activity-11	Total
05	1	2	2	2	2	1	1	1	1	1	1	20

Class Participation	MSE-1	MSE-2	ESE	Total
20	30	30	50	100

7 Assignments / Tutorials Details

Assignment/ Tutorial No.	Title of the Assignments / Tutorials	CO Map	Mapping Bloom Taxonomy Level	Assignment/ Tutorials given to Students on	Assignments to be submitted back on
1	Assignment #1: Introduction to System Software and Compiler	CO1, CO2	Understand	21-07-2025	28-07-2025
2	Assignment #2: Lexical Analysis.	CO1, CO2, CO3, CO4	Apply	28-07-2025	04-08-2025
3	Assignment #3: Syntax and Semantic Analysis	CO1, CO2, CO3	Apply	04-08-2025	11-08-2025
4	Assignment #4: Parsing - I	CO2, CO3	Apply	11-08-2025	18-08-2025
5	Assignment #5: Parsing - II	CO3, CO4	Apply	18-08-2025	25-08-2025
6	Assignment #6: Parsing - III	CO4, CO5	Apply	01-09-2025	08-09-2025
7	Assignment #7: Intermediate Code Generation-I	CO2, CO3	Apply	08-09-2025	15-09-2025
8	Assignment #8: Intermediate Code Generation - II	CO3, CO4	Apply	15-09-2025	22-09-2025
9	Assignment #9: Macro Processor Design	CO1, CO6	Apply	22-09-2025	29-09-2025
10	Assignment #10: Code Optimization	CO4, CO5	Apply	29-09-2025	06-10-2025

11	Assignment #11: Code Generation	CO3, CO4, CO6	Understand	06-10-2025	13-10-2025

Analysis of Assignment / Tutorial Questions and Related Resources

Assignment / Tutorial No.	Week No.	Type* (√)			Module No.	Based on #			Question Type (√)	
		OT	CS	DTP		Textbook	Reference Book	Other Learning Resource	Real Life Assignments	Thought Provoking
1	Week 2			√	1	√	√	E-Book #1 Website #1		√
2	Week 3			√	1	√	√	E-Book #1 Website #1		√
3	Week 4			√	2	√	√	E-Book #1 Website #1		--
4	Week 7			√	3	√	√	E-Book #1 Website #1		√
5	Week 8			√	3	√	√	E-Book #1 Website #1		--
6	Week 9			√	3	√	√	E-Book #1 Website #1		√
7	Week 10			√	4	√	√	E-Book #1 Website #1		--
8	Week 11			√	5	√	√	Web Link #5		√
9	Week 12			√	2,3,4	√	√	4.d. Latest / top rated research papers		√
10	Week 13			√	6	√	√	Web Link #6		√
11	Week 14			√	ALL	√	√	Web Link #6		√

* Tick (√) the Type of the Assignment: Online Tools (OT); Collaborative Assignments (CS); Design /Thought provoking (DTP)
# Write number for textbook, reference book, other learning resource from this AAP – *from Points 4.a to 4.d*

8 In Semester Assessment (ISE) / Other Class Test / Open Book Test (OBT)/Take Home Test (THT) Details

Tests	Test Dates	Module No.	CO Map	MSE Question Paper Pattern	Policy
MSE-1	8th Week	1, 2, 3	CO1, CO2, CO3	Q.1. Solve 2 out of 3 (10 Marks) Q.2. Solve 1 out of 2 (10 Marks) Q.3. Solve 1 out of 2 (10 Marks) Q.1. Solve 2 out of 3 (10 Marks) Q.2. Solve 1 out of 2 (10 Marks) Q.3. Solve 1 out of 2 (10 Marks)	
MSE-2	14th Week	3,4,5	CO3, CO4, CO5		

Pop Quiz	5th Week 9th Week	1, 2, 3, 4	CO1. CO2, CO3, CO4	MCQ based (10 Questions)	

*** Failures of MSE (MSE-1+MSE-2) shall appear for MSE in the next semester. There is no provision for re-test in the same semester.**

9. Practical Activities

Practical No.	Module No.	Title of the **Experiments**	Type of Experiment		Topics to be highlighted	CO Map
			PBL	Newly Added		
NA						

10 Uncovering syllabus with different Activities.

No.	Type of the Activity	Activities	Number of beneficiaries	Other Details – guest profile, feedback, mark sheet, report
1	**Experiential learning/Interaction with Outside World**	1- Guest Lectures by Industry Expert		NA
		2- Workshops		NA
		3- Mini Project		NA
		4- Industrial Visit		NA
		5- Any other activity		CASE STUDY of LEX/YACC/YAJCO
2	**Collaborative & Group Activity**	6- Poster Presentation		Topic: ICG
		7- Minute Papers		Minute paper on Role of Automata theory in compiler construction (week 9 or 10)
		8- Students Seminars		NA
		9- Students Debates		NA
		10- Panel Discussion / Mock GD		NA
		11- Mock Interview		NA
		12- Any other activity		NA
3	**Co-Curricular Activity**	13- Informative videos (NPTEL/YouTube /TEDx/ MIT OW/edX)		NPTEL videos
		14- Lecture Capture Usage		Yes

		15- Any other activity		NA
4	**Tests & Assessments**	16- Class Tests/ Weekly Tests		NA
		17- Pop Quiz		NA
		18- Mobile App Based Quiz		NA
		19- Open Book Test		NA
		20- Take Home Test		1 on Design of SLR parser for a given grammar (week 14)
		21- Any other activity		NA

11 AAP

No.	Programme	Course	Uploaded on V-refer	Date
1	Computer Engineering	System Programming & Compiler Design		

12 Lecture Guide

No.	Programme	Course	Uploaded on V-refer	Date
1	Computer Engineering	System Programming & Compiler Design		

*** Do not delete any activity. Give details for planned events. Write 'NA' for activity Not Planned.**

Consolidated Academic Administration Plan Prepared by (mention all theory teaching faculty names with signature)

Please write below your name and sign with date of the external cluster mentor meeting

Pankaj Vanwari		

Dr. Jayant Dani External Industry Mentor	Dr. Manish Pote External Academic Mentor	Pankaj Vanwari VIT Cluster Mentor	Dr. Ravindra Sangle Program HOD

Annexure:

Assessment Criteria of Learning Outcomes:

Learning Outcomes: The Learner will:	**Assessment Criteria:** The Learner can:	**Evaluated under ISA/MSE/ESE/LAB**
LO 1.1: Define system software and list different types of system programs. (P.I. - 1.4.1, 2.1.1) (CO1)	• Define what system software is. • List at least 5 common types of system programs (e.g., compiler, assembler, loader, linker, macro processor). • Identify system software types from a given list. • Match system programs with their correct functions.	ISA/MSE
LO 1.2: Explain the functions of assemblers, loaders, linkers, and compilers. (P.I. - 1.4.2, 2.1.2) (CO2)	• Define what an assembler, loader, linker, and compiler is. • Describe the function of each system program in the program execution process. • Identify the role of each component given a scenario or diagram. • Match system programs with their correct function or operational stage.	ISA/MSE
LO 1.3: Illustrate the workflow of a compiler and describe how various system programs interact. (P.I. - 1.4.3, 2.2.3) (CO3)	• Draw or explain the block diagram representing the phases of a compiler. • Describe the input/output of each compiler phase (e.g., lexical analysis, syntax analysis, code generation). • Identify system programs (e.g., linker, loader, macro processor) and explain their role in the program execution process. • Match each system program with its correct function or position in the overall compilation-execution flow.	MSE/ESE
LO 1.4: Compare and contrast system programs based on their role in program execution. (P.I. - 2.1.4, 5.3.1) (CO4)	• Define the purpose of system programs such as assembler, loader, linker, and compiler. • Identify the role of each system program in the execution process. • Distinguish between the functionalities of system programs using examples. • Compare two or more system programs based on input/output behavior and dependency. • Analyze a scenario to determine which system programs are involved and how they interact.	MSE/ESE
LO 2.1: Describe the role of lexical analysis in a	• Define what lexical analysis is.	ISA/MSE

compiler. (P.I - 1.4.1, 2.1.1) (CO1)	• List the key functions of a lexical analyzer (e.g., tokenization, removing white spaces/comments). • Identify the components involved in lexical analysis (e.g., input buffer, scanner, token). • Match each component with its corresponding function in the lexical analysis phase.	
LO 2.2: Explain how finite automata are used in lexical analysis. (P.I. - 1.4.2, 2.1.2) (CO2)	• Define finite automata and its role in compiler design. • Describe the working of deterministic and non-deterministic finite automata in token recognition. • Identify finite automata components (states, transitions, accept states) from a given diagram. • Match different types of tokens (identifiers, numbers, keywords) with their corresponding finite automata patterns. • Explain how a lexical analyzer uses finite automata to scan and tokenize input.	ISA/MSE
LO 2.3: Construct simple lexical analyzers for tokenizing input strings. (P.I. - 1.4.3, 2.2.3) (CO3)	• Define what a lexical analyzer is and explain its role in a compiler. • List the basic components required to build a lexical analyzer (e.g., tokens, patterns, lexemes, finite automata). • Identify correct token classifications from given input examples. • Match token patterns with appropriate regular expressions. • Construct a simple lexical analyzer using code or pseudocode to tokenize an input string.	MSE/ESE
LO 2.4: Analyze the efficiency and correctness of a given lexical analyzer design. (P.I. - 2.1.4, 5.3.1) (CO4)	• Explain the workflow and components of a lexical analyzer. • Identify inefficiencies in a given lexical analyzer's state transition or tokenization logic. • Compare two lexical analyzer designs based on time and space complexity. • Detect incorrect or ambiguous behavior in lexical output given an input string. • Justify improvements to a lexical analyzer using analysis of performance and correctness.	MSE/ESE
LO 3.1: Recall the types of parsers and grammar notations used in syntax	• Define the term "parser" and "grammar" in the context of syntax analysis.	ISA/MSE

analysis. (P.I. - 1.4.1, 2.1.1) (CO1)	• List at least 4 types of parsers (e.g., LL(1), SLR, operator precedence, LR). • Identify types of grammar notations such as BNF and EBNF. • Match parser types with their parsing approach (e.g., top-down or bottom-up).	
LO 3.2: Interpret parse trees and syntax-directed definitions. (P.I. - 1.4.2, 2.1.2) (CO2)	• Define what a parse tree is and explain its purpose. • Identify components (non-terminals, terminals, productions) within a given parse tree. • Trace a derivation and construct the corresponding parse tree. • Match syntax-directed definitions to grammar rules and interpret their semantic rules. • Explain how attributes are computed using syntax-directed definitions.	
LO 3.3: Implement LL(1) or SLR parsers for a given grammar. (P.I. - 1.4.3, 2.2.3) (CO3)	• Define what LL(1) and SLR parsers are. • Identify FIRST and FOLLOW sets for a given grammar. • Construct a predictive parsing table for LL(1) grammar. • Construct a parsing table for SLR grammar using LR(0) items. • Implement an LL(1) or SLR parser for a provided grammar using a programming language or parsing tool. • Trace input strings through the parsing table to determine acceptance or errors.	MSE/ESE
LO 3.4: Examine parsing errors and ambiguity in context-free grammars. (P.I. - 2.1.4, 5.3.1) (CO4)	• Define what a parsing error is in the context of syntax analysis. • Identify ambiguous grammar from given production rules. • Analyze a grammar to determine whether it leads to shift-reduce or reduce-reduce conflicts. • Match types of parsing errors with example scenarios.	MSE/ESE
LO 3.5: Justify the choice of parser for a specific language or grammar structure. (P.I. - 2.1.5, 5.3.2) (CO5)	• Define different types of parsers (e.g., LL(1), SLR, operator precedence). • List characteristics and limitations of various parsing techniques. • Identify suitable parsers for given grammar examples or language features. • Justify the selection of a particular parser based on grammar structure and efficiency considerations.	ESE
LO 4.1: List different intermediate code	• Define what intermediate code is in the context of compilers.	

representations. (P.I. - 1.4.1, 2.1.1) (CO1)	• List at least 4 types of intermediate representations (e.g., syntax tree, postfix notation, three-address code, quadruples). • Identify the type of intermediate code used from a given code snippet. • Match intermediate code representations with their characteristics or structure.	
LO 4.2: Describe the purpose of using intermediate code in compilation. (P.I. - 1.4.2, 2.1.2) (CO2)	• Define what intermediate code is in the context of a compiler. • List at least three types of intermediate code representations (e.g., syntax tree, three-address code, postfix notation). • Identify intermediate code representations from sample code snippets. • Match types of intermediate code with their respective advantages or use cases.	ISA/MSE
LO 4.3: Generate intermediate code such as three-address code or syntax trees from a given source code snippet. (P.I. - 1.4.3, 2.2.3) (CO3)	• Define intermediate code and its purpose in compilation. • Identify different forms of intermediate representations (e.g., syntax tree, postfix, three-address code). • Translate a given arithmetic or logical expression into three-address code. • Generate a syntax tree from a simple source code snippet. • Match the generated intermediate code with the corresponding input code segments.	MSE/ESE
LO 4.4: Compare intermediate representations based on ease of translation and optimization potential. (P.I. - 2.1.4, 5.3.1) (CO4)	• Define what intermediate representations (IRs) are in the context of a compiler. • List various forms of IRs (e.g., three-address code, syntax tree, postfix notation, quadruples, triples). • Identify the characteristics of different IR formats. • Match IR types to their best use-cases (e.g., easy translation, good for optimization). • Compare two IRs based on translation complexity and potential for code optimization.	MSE/ESE
LO 5.1: Identify common code optimization techniques. (P.I. - 1.4.1, 2.1.1) (CO1)	• Define what code optimization means in the context of a compiler. • List at least 5 common code optimization techniques (e.g., constant folding, dead code elimination, loop unrolling).	ISA/MSE

	• Identify specific optimization techniques applied in short code snippets. • Match optimization techniques with their corresponding benefits or use-cases.	
LO 5.2: Explain the difference between machine-dependent and machine-independent optimizations. (P.I. - 1.4.2, 2.1.2) (CO2)	• Define what code optimization is. • List key features of machine-dependent and machine-independent optimizations. • Identify examples of both types from a given set of optimization techniques. • Match optimization strategies with their corresponding category (machine-dependent or independent). • Explain how each type impacts code generation and performance.	ISA/MSE
LO 5.3: Apply basic optimizations like constant folding or dead code elimination. (P.I. - 1.4.3, 2.2.3) (CO3)	• Define what code optimization means in the context of compiler design. • List common basic optimizations such as constant folding and dead code elimination. • Identify where constant folding or dead code elimination can be applied in a code snippet. • Apply constant folding to simplify expressions. • Eliminate unreachable or dead code from a sample intermediate code block.	MSE/ESE
LO 5.4: Analyze how different optimizations affect performance and resource usage. (P.I. - 2.1.4, 5.3.1) (CO4)	• Identify common compiler optimization techniques (e.g., loop unrolling, dead code elimination, constant folding). • Explain the purpose and effect of each optimization on code efficiency. • Compare optimized vs. non-optimized code in terms of execution time or memory usage. • Analyze given code snippets to determine which optimizations are applied and how they impact performance.	MSE/ESE
LO 5.5: Assess the trade-offs in applying aggressive vs. conservative optimization strategies. (P.I. - 2.1.5, 5.3.2) (CO5)	• Define aggressive and conservative optimization strategies. • List key characteristics and examples of each optimization type. • Identify suitable scenarios for applying aggressive vs. conservative optimizations. • Compare the impact of both strategies on code performance and compilation time.	ESE

	• Justify the selection of a particular strategy in a given programming context.	
LO 6.1: Recall the components involved in code generation. (P.I. - 1.4.1, 2.1.1) (CO1)	• Define what code generation is in the context of a compiler. • List at least five key components involved in code generation (e.g., basic blocks, flow graphs, instruction selection, register allocation, code emission). • Identify code generation components from a provided list. • Match each component with its function or role in the code generation process.	ISA/MSE
LO 6.2: Explain the concept of basic blocks and flow graphs. (P.I. - 1.4.2, 2.1.2) (CO2)	• Define what a basic block is in the context of compiler design. • List the characteristics or properties of a basic block. • Identify basic blocks and flow graph components in a given intermediate code segment. • Match flow graph components (e.g., nodes, edges, entry/exit points) with their definitions or functions.	ISA/MSE
LO 6.3: Design a simple code generation algorithm using intermediate code. (P.I. - 1.4.3, 2.2.3) (CO3)	• Define what code generation is in the context of compiler design. • List essential components required for a basic code generation algorithm. • Identify different types of intermediate code used for code generation (e.g., three-address code, postfix). • Match intermediate code constructs with corresponding target code patterns. • Construct a simple code generation algorithm for a given intermediate code sequence. • Implement the designed algorithm to generate assembly-level or low-level code.	MSE/ESE
LO 6.4: Break down code generation problems into target-specific issues. (P.I. - 2.1.4, 5.3.1) (CO4)	• Identify the key steps involved in code generation. • List common target-specific issues (e.g., register allocation, instruction selection). • Break down a given code generation task into basic blocks and flow graphs. • Match target machine constraints with their impact on generated code.	MSE/ESE
LO 6.5: Develop a prototype code generator for a basic programming	• Explain the structure and purpose of a code generator.	ESE

language subset. (P.I. - 1.4.5, 2.2.5) (CO6)	• Identify components required for generating target code (e.g., symbol tables, flow graphs). • Design a basic control flow for translating intermediate code into target code. • Implement a working prototype of a code generator for a simple language construct. • Demonstrate the functionality of the code generator with test inputs and outputs. • Debug and refine the prototype based on test cases.	