AICTE Report

Model Curriculum

for

Undergraduate Degree Courses

in

Computer Science and Engineering
(Engineering & Technology)

Report submitted by the committee constituted by AICTE for model design.
March 2022

TABLE OF CONTENT (Page nos are approximate)

Framework for CSE Curriculum Design

Context and Background

Computer Science has changed considerably in the last few years with areas like Machine Learning and Cloud computing becoming much more important. At the same time, the technologies and underlying computing systems have also evolved, considerably easing executing some of the tasks that earlier took much more training and experience. These changes require that teaching of computer science ought to suitably adapt – to reflect the changed nature of the discipline, as well as to update courses with the more recent technology platforms. And this ought to be done while providing some flexibility to the HEIs to address their specific constraints and focus. With these in mind, the committee for model curriculum design for CSE established some of the guiding principles for the exercise:

- The focus of curriculum design is the 4 yr BE/BTech program in CSE and the target audience of the curriculum is the vast majority of universities/institutions, rather than the premier institutes (who design their own curriculum and have the expertise for it.)
- The existing CSE curriculum of AICTE will be the starting point. Suitable enhancements/modifications will be made.
- **Flexibility**. Even within the large non-top tier education system, which is the primary target, some have better infrastructure and capability, so it is desirable to provide a limited degree of flexibility to the HEIs on the curriculum.

  To provide this flexibility, for each course, the learning outcomes will be grouped in two – essential, and desired/advanced. The core courses will also be identified as essential and desired/advanced.

  This simple framework of grouping outcomes and courses as essential and desired provides limited flexibility to institutions to design their curriculum depending on their capabilities, resources, goals, etc, while still providing guidance for a sound curriculum. The flexibility can be leveraged by institutions to provide different pathways to students, and multiple exits.
- **Early Exits**. The new education policy (NEP) aims to provide multiple exit points to students. To support more than one exits for students, it is important to develop employability skills early, and not take the approach of first focusing on foundations and then develop practical skills. Also, we believe that all but a few students will go for full degree, hence while providing flexibility for early exit, the outcomes of the 4 yr degree should not be compromised. We propose:
  - Eliminate the separation of theory and labs, instead break courses topic or theme-wise. This will allow both basic theory and basic skills to be taught together and build skills early.
  - Introduce discipline courses early, so disciplinary skills can be developed early. For this have one sem common program, and allow some discipline courses in 2nd sem – this will also help students in getting internships / part-time jobs in summer. This will require branch change to be done after one semester. (For lateral entry, some make-up courses may be needed.)

  With this, and the flexibility provided in the curriculum, an HEI can design suitable exits.

- **Provide Guidance for labs/assignments.** A small analysis the committee did on usage of AICTE curriculum indicated that while the courses and topics specified in the curriculum are widely used, HEIs differ considerably in types of assignments/labs/projects they give. Also, this is where there is a perceived deficiency in education – while theory is covered suitable assignments/labs are not give for students to develop skills. To help in this, it was decided that for each course design, experts will also recommend nature of labs and assignments for each module in the course.
- **Pedagogy suggestions**. With the emergence of a host of online resources that can be used to support teaching, there may be different ways to teach a topic. To help teachers in pedagogy, for each module in a course, pedagogy suggestions have also been provided.
- **Multiple Pathways**. For supporting multiple pathways within the academic program, we propose to provide for micro specializations through thematic course streams. These can be further enhanced by HEIs with programs like honors for advanced students with more credits or advanced learning outcomes,etc.

Graduate Attributes

Curriculum of a program is finally a network of credit units – courses (core, disciplinary core, disciplinary elective, open), internships, practice, projects, etc. which help achieve program goals. Program goals can be stated as attributes the students should possess on graduation, i.e. statements about the learning, values, capabilities etc. of graduates. These are called Graduate Attributes (GAs). A program typically has:

- General GAs: which are often common across many similar programs (e.g. BTechs) and focus on generalized skills and capabilities in the graduate.
- Discipline GAs: are discipline specific attributes, which focus on understanding of different concepts and systems related to the discipline, and on competencies and skills in that discipline.

Together the GAs define the goals of the program. The aim of curriculum design is to evolve a curriculum that can develop in students the stated graduate attributes. While specifying the GAs and designing a curriculum for it, we must keep a basic constraint in mind: a full BTech program has 8 semesters, each with about 5 full courses. GAs should specify only what can be taught and absorbed in this timebox (i.e. we cannot push more simply by adding more.)

Desired Graduate Attributes for the CSE program are given below. The CSE curriculum design will focus more on delivering the discipline GAs, while strengthening the general GAs, where possible. Feedback on these Graduate Attributed was taken from many representatives from industry, as well as from HEIs. (GAs should be read by adding this at the start of each: *At graduation time, a student should have:*

General Graduate Attributes	**Discipline Graduate Attributes (for CSE)**

G1 Ability to identify a problem, analyze using design thinking techniques, and evolve innovative approaches for solving it.	CS1 Proficiency in writing in at least two dissimilar programming languages programs of modest complexity which are: readable, tested for correctness, efficient, and secure.
G2 Ability to apply mathematical concepts and techniques in problem solving.	CS2 Ability to design and apply appropriate algorithms and data structures for evolving efficient computing based solutions for new problems.
G3 Ability to function effectively in multi- cultural teams to accomplish a common goal.	CS3 Understanding of computing systems at computer architecture, operating systems, and distributed- computing levels, and how they affect the performance of software applications.
G4 Ability to communicate effectively with a wide range of audience.	CS4 Understanding of theoretical foundations, fundamental principles, and limits of computing.
G5 Ability to self-learn and engage in life-long learning and upgrade technical skills	CS5 Ability to analyse large volumes of data employing a variety of techniques for learning, better prediction, decision making, etc
G6 An understanding of professional and ethical responsibility	**ADVANCED/OPTIONAL GAs** CS6 Ability to design, implement, and evaluate computer based system or application to meet the desired needs using modern tools and methodologies
G7 Ability to undertake small research tasks and projects.	CS7 Ability to develop full stack applications using one commonly used tech-stack and modern tool.
G8 An entrepreneurial mindset for opportunities using technology and innovations.	CS8 Understanding of and ability to use advanced techniques and tools in a few different domain areas (e.g. parallel processing, image processing, IR, …)
G9 An understanding of impact of solutions on economic, societal, and environment context.	CS9 Exposure to emerging technologies such as Cloud Computing, IoT, etc
G10 Strong emotional intelligence, human and cultural values	

CSE Core Courses

For identifying the core courses for a robust CSE BTech program, the following assumptions have been made:

- All CSE program have the first course on programming, generally called “Introduction to Programming”, with different institutions choosing different languages. This course is generally common across all engineering disciplines. It is assumed that this course is part of the common core (and hence is not included in the CSE core courses)

- It is assumed that there are at least two Math courses that are part of common core which all CSE students will do – a course on Probability and Statistics, and a course on Linear Algebra.

In addition to these, to support multiple exits, it is also suggested that In first year, the common program should have an engineering design course where a project to build a system for some purpose is the main goal for students, and lectures support the project.

With these assumptions, the list of core courses recommended for CSE are given below. As explained above, the core has been split into two – essential and additional. The essential core is what all programs must have. The additional core list those courses which HEIs can choose to make core or an elective, depending on their education approach and resources.

Suggested Year	**Course Name**	**CS GAs it Directly Contributes to (GAs it supports)**
	Professional Core Courses (Essential)	
2nd year	Data structures and Algorithms (DS)	CS1, CS2, CS4
2nd year	Discrete Mathematics (DM)	CS4 (CS2, CS4, CS6)
2nd year	Computer Organization and Architecture (COA)	CS3 (CS4, CS6)
2nd year	Advanced Programming (in lieu of OO prog.) (AP)	CS1, CS3, CS7, CS8 (CS2)
3rd year	Operating Systems (OS)	CS3 (CS6, CS8)
3rd year	Design and Analysis of Algorithms (Algo)	CS1, CS2, CS4
3rd year	Database Systems (DB)	CS3, CS5, CS6 (CS1, CS7, CS9)
3rd/4th year	Computer Networks (NW)	CS3 (CS4, CS6, CS8)
3rd year	Machine Learning (New) (ML)	
3rd year	Security (New)	CS1, CS3, CS6
	Extended Professional Core (Additional)	
3rd/4th year	Compiler Design	
3rd/4th year	Theory of Computation	CS4 (CS2)

Detailed syllabus of each of these courses follows – a separate subsection for each course. A common template has been used to specify the course design. Each course design lists essential learning outcomes for the course, and desired/advanced learning outcomes. As the terms suggest, all HEIs should ensure that the essential learning outcomes are achieved. The desired/advanced learning outcomes are those which those HEIs which have the capabilities and resources to deliver may include in their courses.

Coverage of CS Graduate Attributes by the Core Courses

This table summarizes which CS GAs different courses contribute to. For each GA, list of main courses is given – these are the courses which directly contribute to the GA. Also mentioned are the courses which support the GA (though perhaps less directly):

CS1 Proficiency in writing in at least two dissimilar programming languages programs of modest complexity which are: readable, tested for correctness, efficient, and secure	Main Courses: DS, AP, Algo, Security Supporting Courses: DB
CS2 Ability to design and apply appropriate algorithms and data structures for evolving efficient computing based solutions for new problems	Main: DS, Algo Supporting: DM, AP, ToC
CS3 Understanding of computing systems at computer architecture, operating systems, and distributed- computing levels, and how they affect the performance of software application	Main: CO/CA, AP, OS, DB, NW, Security Supporting:
CS4 Understanding of theoretical foundations, fundamental principles, and limits of computing	Main: DS, DM, Algo, ToC Supporting: NW, CO/CA
CS5 Ability to analyse large volumes of data employing a variety of techniques for learning, better prediction, decision making,	Main: DB Supporting: DM
ADVANCED/OPTIONAL CS6 Ability to design, implement, and evaluate computer based system or application to meet the desired needs using modern tools and methodologies	Main: AP, DB, Security Supporting: DM, CO/CA
CS7 Ability to develop full stack applications using one commonly used tech-stack and modern tools	Main: AP Supporting: OS, NW, Security
CS8 Understanding of and ability to use advanced techniques and tools in a few domain areas (e.g. parallel processing, image processing, IR, …)	Main: Supporting: DB
CS9 Exposure to emerging technologies such as Cloud Computing, IoT, etc	Main: Supporting: DB, Security, NW

Micro Specializations (and Professional Electives)

Besides the core courses, programs normally have professional elective courses, which HEIs decide. It is possible to use the electives to provide a limited specialization in some sub-area of CSE to a BTech student. We call these as micro-specializations. These also allow multiple pathways to students, as different students can graduate with different specializations (or not). A micro specialization is a set of a few full or half courses, which build upon the core CSE program. The report gives possible structure of a few micro specializations in:

- Software Engineering
- Machine Learning
- Distributed and Cloud Systems
- Human Computer Interaction (HCI)

More specializations definition may be added (e.g. in Security, High Performance Computing, Algorithms). Micro specializations are optional for HEIs – i.e. they can decide to offer them or not, and if they do, which ones they want to. If they decide not to have these, the list of courses mentioned in the micro specializations can be used as a list of suggested professional electives.

Syllabus for CSE Core Courses

This section gives the detailed syllabus of all the core courses for the model CSE program. Each course design states the learning outcomes – both essential and desirable – and then the modules in the course and topics in each of the modules. The design also suggests for each module suitable pedagogy approach and suitable assignments/labs.

Data Structures and Algorithms

CS301	**Data Structure & Algorithms**	**3L:0T: 4P**	**5 credits**	**Pre-Reqs: ESC201**

Learning Outcomes of the course (i.e. statements on students' understanding and skills at the end of the course the student shall have):

Essential (<=6):

1. Understanding abstract specification of data-structures and their implementation.
2. Understanding time and space complexity of programs and data-structures.
3. Knowledge of basic data-structures, their applications and relative merits.
4. Ability to convert an algorithmic solution to a program using suitable data-structures and analyze the trade-offs involved in terms of time and space complexity.

Desirable/Advanced (<= 3):

1. Amortized Complexity
2. Use of randomization in data-structures

Detailed contents for Essential Learning Outcomes:

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Introduction and basic terminology (1 week)	(i) Notion of data-structures and algorithms. (ii) $log n, n, 2^n$: understanding growth of these functions, and applications (binary search and extensions to similar problems)	**T1: Chapter 3** (i) Explain the interplay between algorithms and data-structures (ii) Explain the meaning of worst and average case (iii) Examples of $O(\quad)$, Motivation behind asymptotic analysis	(i) Worst/average case analysis for small pseudo-codes (ii) Prove/disprove why a function $f(n)$ is $O(g(n))$ (and similarly for Ω). (iii) Variations on binary search with applications, recursive

	(ii) Worst-case, average-case time/space complexity and their relative merits. (iii) Asymptotic Notation: $O(\quad), \Omega(\quad)$	(large n and ignoring constants). (iv) Discuss recurrence relation for binary search.	and iterative implementation of binary search with applications to problems.
Module 2: Abstract Data-types, Arrays, Linked Lists, Stacks, Queues Dictionary ADT, Trees, Binary Trees (2.5 weeks)	(i)Abstract data-type (ADTs): arrays and linked list ADTs. (ii) Stacks, Queues: ADTs and implementations using arrays, linked lists. (iii) Doubly linked lists: ADT and implementation (iv) Dictionary ADT: implementation using array, linked lists, binary search. (v) Tree ADT and examples (vi) Implementation of trees and basic traversal algorithms (vii) Binary trees and inorder traversal	**T1: chapter 4, Chapter 6** (i) Explain the difference between specification and implementation of ADTs. (ii) Discuss stack implementation using arrays of fixed size and linked list, and relative merits. (iii) Circular array and linked list implementation of queues, and relative merits. (iv) Discuss some applications of stacks: post-fix notation, matching parentheses etc. (v) Dictionary ADT: discuss why neither arrays nor linked list is a good implementation, but sorted arrays are good for searching. (vi) Discuss situations where we need to store hierarchical data. (vii) Discuss pre-order and post-order , traversals and applications in finding height and similar problems.	(i) Implementation of stacks with application to a problem. (ii) Implementation of queues with application to a problem. (i) Implementation of trees with applications for storing and accessing hierarchical data.

Module 3: Priority Queues and Heaps (1 week)	(i) Priority Queue ADT (ii) Definition of heaps (iii) Implementation of Priority Queues using heaps and running time analysis (iv) Implementation of heaps using arrays. (v) Heap-sort	**T1: Chapter 7.1-7.3** (i) Start with FindMin and Insert, and a simple $O(1)$ time and $O(1)$ space algorithm. Buildup towards DeleteMin. (ii) Discuss algorithm for heaps: inserting an element, modifying an element, and deleting the minimum element. (iii) Discuss why array implementation is more efficient than balanced binary trees	(i) Array implementation of heaps and application to a problem. (ii) k-ary heaps: compare with binary heaps (both in theory and practice)
Module 4: Binary Search Trees, AVL Trees, 2-4 trees (3 weeks)	(i) Binary Search Trees: definition and some basic algorithms. (ii) Implementation of Dictionary ADTs using Binary Search trees and running time analysis (iii) AVL trees: height balance condition, rotations, and implementation of dictionary ADT (iv) 2-4 Trees: Multi-way search trees, implementation of dictionary ADT, Informal discussion of extension to B-trees.	**T1: Chapter 9.1, 9.2, 9.4** (i) Discuss algorithms for finding predecessor or successor, and similar problems in Binary Search Trees. (ii) Explain why height of a Binary Search Tree may not remain $O(logn)$. (v) Explain how the height balance condition ensures that the height is $O(logn)$, and how rotation changes the structure of a tree. (vi) Explain why rotations in AVL trees restore height balance condition (vii) Explain why 2-4 trees have $O(logn)$height and the running time of insert/delete operations. (viii) Discuss how balanced binary tree data-structures can	(i) Implementation of AVL trees with search, insert, delete operations and application to a problem. Comparison with unbalanced Binary Search Trees. (ii) Implementation of 2-4 trees with search, insert, delete operations and application to a problem. (iii) Comparison of the two implementations above.

		implement a priority queue	
Module 5: Hash tables, tries (2 weeks)	(i) Map ADT (ii) Hash Tables and implementation of Map using Hash Tables (iii) Design of hash functions (iv) Collision resolution schemes: chaining, open addressing schemes like linear probing, quadratic probing, double hashing. (v) Applications of Hashing: finding duplicates, set intersection, etc. (vi) Tries: implementation of Map ADT using tries. (vii) Compressed tries and suffix tries.	**T1: Chapter 8.1-8.3, Chapter 11.3** (i) Explain the difference between Map and Dictionary ADT. (ii) Discuss how hash functions can have non-numeric keys as input. (iii) Discuss the relative merits of hash tables and balanced binary search trees. (iv) Discuss how hashing can be a substitute for sorting in many cases. (v) Explain why tries can be better than balanced binary search trees in some settings. (vi) Explain how compressed tries save space (vii) Discuss real-life applications of tries.	(i) Implementation of hash tables with applications to a problem. (ii) Implementation of tries and applications to a problem.
Module 6: Sorting, Selection (1.5 weeks)	(i) bubble sort, insertion sort, selection sort. (ii) Merge sort and divide and conquer paradigm (iii) Quick sort: average and worst case analysis, randomized quicksort (intuitive explanation) (iv) Selection based on partitioning ideas used in QuickSort.	**T1: Chapter 10.1, 10.2, 10.4, 10.5, 10.7** (i) Discuss why $O(n^2)$ time algorithms can be useful sometimes (small data, data nearly sorted etc.) (ii) Only the recurrence for merge sort and mention that divide-and-conquer paradigm will be explored more in algorithms course. (ii) Discuss the randomized splitting	Implementation of sorting algorithms and comparison of running times on large data-sets.

		algorithm for quicksort and selection and explain intuitively the expected running time.	
Module 7: Graphs, representations and traversal algorithms, applications of BFS, DFS (2.5 weeks)	(i) Graph ADTs and applications (ii) Adjacency list and adjacency matrix representations and relative merits (iii) Basic graph definitions: paths, cycles, trees, spanning trees, connected components, Euler's formula. (iv) Depth First Search Traversal algorithm for directed graphs: classification of edges into forward, back and cross edges. Applications to cycle finding, topological sort in directed acyclic graphs, finding connected components. Running time analysis. (v) Breadth first search algorithm: implementation using queues, shortest path tree property. Running time analysis	**T1: Chapter 12.1-12.4** (i) Discuss the wide applicability of graphs including social networks, internet. (ii) Discuss time and space complexity of basic operations using adjacency list and adjacency matrix. (iii) Discuss why trees have $n - 1$edges. (iv) Discuss how DFS can be thought of as exploration with backtracking. Explain the role of stack in DFS. (v) Explain how BFS can be thought of as traversal along shortest paths and implementation using queues. (vi) Formal proof of why BFS yields a shortest path tree.	(i) Graph implementation using adjacency list and DFS/BFS traversal with applications.

Detailed Contents for Desirable Learning Outcomes (optional, <= 3 modules):

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**

Module 1: Amortized Complexity (1 week)	(i) Binary counter (ii) Binomial Heaps (iii) Extendible arrays	**T1: Chapter 5.1.3,** **R2: Chapter 17.1** (i) Explain the motivation behind amortized analysis. (ii) Analyze amortized complexity by explicit calculation of total number of operations after $n$steps.	(i) Exercises on amortized time complexity (e.g., a queue using two stacks etc.) (ii) Implementation of extendible arrays.
Module 2: Randomization in data-structures (1 week)	(i) Skip Lists (ii) randomized quick-sort	**T1: Chapter 8.4, 10.2** (i)Explain the idea of expectation of a random variable. (ii) How does expected running time translate to actual running time (iii)Recurrence for expected running time randomized quicksort	(i)Exercises on calculation of expectation of a random variable. (ii) Implementation of skip lists and comparison with AVL trees.

Suggested text books / Online lectures or tutorials:

1. "Data Structures and Algorithms in Java", by Michael T. Goodrich and Roberto Tamassia, John Wiley & Sons; 3rd Edition.

2. "Data Structures and Algorithms in Python", by Michael T. Goodrich and Roberto Tamassia, Wiley, 1st Edition.

Suggested reference books / Online resources:

1. NPTEL video series, Data-structures and Algorithms, Instructor: Naveen Garg.

2. Introduction to Algorithms, 4TH Edition, Thomas H Cormen, Charles E Lieserson, Ronald L Rivest and Clifford Stein, MIT Press/McGraw-Hill.

The Discipline Graduate Attributes (GAs) to which this course contributes significantly: CS1, CS2, CS4

Other discipline GAs to which this course may contribute somewhat:

Discrete Mathematics

CS-X	**Discrete Mathematics**	**3L:1T: 0P**	**5 credits**	**Pre-Reqs: 0**

Prerequisites remark:
Familiarity with some mathematical notation, ideas and concepts covered at the pre-college levels.

Learning Outcomes of the course (i.e. statements on students' understanding and skills at the end of the course the student shall have):

Essential (<=6):

1. Understand examples in Computer Science through mathematical terminology and notation.
2. Construct direct, and indirect, proofs of basic theorems.
3. Understand the differences between a mathematical proof, a heuristic, and a conjecture.
4. Learn how to divide a problem, or a proof, into smaller cases.
5. Formulate mathematical claims and be able to construct counterexamples.
6. Apply the knowledge of mathematics to solve real-world problems.

Desirable/Advanced (<= 3):

1. Identify formal algebraic structures in computer science.
2. Work with probability & statistics in a rigorous way.
3. Use this course topics to design, and rigorously analyze, real-life algorithms.

Detailed contents for Essential Learning Outcomes:

Module	**Topics**	**Pedagogy / teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Set, Relations, Functions. (~1.5 wks)	Operations and Laws of Sets, Cartesian Products, Binary Relation, Partial Ordering Relation, Equivalence Relation, Image of a Set, Sum and Product of Functions, Bijective functions, Inverse and Composite Function, Size of a Set, Finite and infinite Sets, Countable and uncountable Sets, Cantor's diagonal	- Chapter 1,3 of T1. - Chapter 2 of T2 - Discuss some examples from computer science: +Uncountability of Reals +Uncomputability by the diagonal argument (ref. Chap.6 of T1). +Relational database	Make assignments using the books. To test: (1) what was done in the class (2) whether the student can think and apply the concepts.

	argument and The Power Set theorem.	(ref. Chap.3 of T1. Refer R8).	
Module 2: Proof strategies. (1.5 wks)	Proof Methods and Strategies: Forward Proof, Proof by Contradiction, Proof by Contraposition, Proof of Necessity and Sufficiency, Case analysis, Induction.	- Chapter 1 of T1. - Chapter 1 of T2. - Discuss some examples from computer science: +Examples like above. +$\sqrt{2}$ is irrational. +Prime numbers are infinite (Chap.4.3 of T2).	Make assignments using the books. To test: (1) what was done in the class (2) whether the student can think and apply the concepts.
Module 3: Number theory. (1.5 wks)	Extended Euclid's Greatest Common Divisor algorithm, The Fundamental Theorem of Arithmetic, Modular arithmetic, Coprimality (or Euler's totient function), Chinese Remainder Theorem.	- Chapter 4 of T2. Chapter 1.20 of T1. - References T5 & R5. - Discuss some examples from computer science: +Diffie-Hellman key-exchange. +RSA Cryptosystem.	Make assignments using the books. To test: (1) what was done in the class (2) whether the student can think and apply the concepts.
Module 4: Combinatorics. (1.5 wks)	Permutation & Combination, Inclusion-Exclusion, Pigeon-hole principle, Generating functions, Recurrence.	- Chapter 2, 8, 9 of T1. Chapter 6 of T2. Chapter 2, 3, 4, 5, 6 of T4. Also refer to R3, R4. - Discuss some examples from computer science: +Count binary trees. +Count matched parentheses. +Count matrix chain multiplication.	Make assignments using the books. To test: (1) what was done in the class (2) whether the student can think and apply the concepts.

Module 5: Graphs. (1.5 wks)	Connected components, Paths, Cycles, Trees, Hamiltonian/ Eulerian Walks, Coloring, Planarity, Matching.	- Chapter 4, 5 of T1. Chapter 10, 11 of T2. - Discuss some examples from computer science: +Shortest-distance (Chapter 7 of T1). +Minimum spanning tree. +Prefix codes. +Graph isomorphism problem.	Make assignments using the books. To test: (1) what was done in the class (2) whether the student can think and apply the concepts.
Module 6: Logic. (1.5 wks)	Languages of Propositional logic and First-order logic, expressing natural language sentences in languages of propositional and first-order logic, expressing natural language predicates in the language of first-order logic. What is a proof system; like natural deduction or analytical tableau. Semantics of Propositional logic and First-order logic: evaluation of formulas making use of interpretations. Semantic entailment, Validity and Satisfiability. Notions of Consistency and Completeness of a logic.	- Chapter 1, 11 of T1. Chapter 1, 12 of T2. Also refer to R2. - Discuss some examples from computer science (also, refer T3): +SAT solvers and why these are so useful. +Relational Calculus. +Language for specification like Z. +Hoare logic for program verification.	Make assignments using the books. To test: (1) what was done in the class (2) whether the student can think and apply the concepts.

Detailed Contents for Desirable Learning Outcomes (optional, <= 3 modules):

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**
Module 7: Algebra. (1.5 wks)	Group, Permutation Groups, Cosets, Normal Subgroups, Ring, Field, Finite fields, Fermat's little theorem, Homomorphisms, Isomorphisms.	- Chapter 10 of T1. Also, refer R5, R6. - Discuss some examples from computer science (ref. T3): +RSA Cryptosystem. +Reed-Solomon error-correction encoding.	Make assignments using the books. To test: (1) what was done in the class (2) whether the student can think and apply the concepts.
Module 8: Discrete probability. (1.5 wks)	Discrete Sample Space, Probability Distribution, Random variables, Expectation, Variance, Bernoulli trials, Conditional probability & independence (Bayes' Theorem).	- Chapter 2 of T1. Chapter 7 of T2. - Discuss some examples from computer science (also refer R7): +Randomized algorithms. +Heuristics.	Make assignments using the books. To test: (1) what was done in the class (2) whether the student can think and apply the concepts.

Suggested text books / Online lectures or tutorials:

T1. Liu, C. L., & Mohapatra, D. P. (2008). Elements of Discrete Mathematics. Tata McGraw-Hill.

T2. Rosen, K. H. (2012). Discrete Mathematics and Its Applications.

T3. Huth, M., & Ryan, M. (2004). Logic in Computer Science: Modelling and Reasoning about Systems (2nd ed.). Cambridge University Press.

T4. Cohen, D. I. A. (1978). Basic techniques of combinatorial theory. John Wiley.

T5. Niven, I., Zuckerman, H. S., & Montgomery, H. L. (1991). An introduction to the theory of numbers. John Wiley & Sons.

Suggested reference books / Online resources:

R1. Norman L. Biggs, Discrete Mathematics, (2nd ed. 2002), Oxford University Press.
R2. Smullyan, R. M. (1995). First-order logic. Courier Corporation.
R3. Bóna, M. (2006). A walk through combinatorics: an introduction to enumeration and graph theory.
R4. Cameron, P. J. (1994). Combinatorics: topics, techniques, algorithms. Cambridge University Press.
R5. Shoup, V. (2009). A computational introduction to number theory and algebra. Cambridge University Press.
R6. Herstein, I. N. (2006). Topics in algebra. John Wiley & Sons.
R7. Mitzenmacher, M., & Upfal, E. (2017). Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis. Cambridge University Press.
R8. C. J. Date (2019). Database Design and Relational Theory. Normal Forms and All That Jazz.

Prepared by:

1. Somenath Biswas, Computer Science & Engineering Department, IIT Goa (retired from IIT Kanpur)
2. Partha Mukhopadhyay, Theoretical Computer Science Group, Chennai Mathematical Institute
3. Nitin Saxena, Computer Science & Engineering Department, IIT Kanpur
4. Bhabani P. Sinha, Advanced Computing & Microelectronics Unit, Indian Statistical Institute Kolkata

The Discipline Graduate Attributes (GAs) to which this course contributes significantly: CS4

Other discipline GAs to which this course may contribute somewhat: CS2, CS5, CS6

Computer Organization & Architecture

CS402	**Computer Organization & Architecture**	**3L:0T: 4P**	**5 credits**	**Pre-Reqs: ESC302**

Learning Outcomes of the course (i.e. statements on students' understanding and skills at the end of the course the student shall have):

Essential (<=6):

1. The key components of a basic computer
2. The key components of a CPU and how the instructions are executed
3. Execution and time taken by instructions in a pipelined processor
4. The need for memory hierarchy and efficiency achieved due to the use of cache
5. How the data is stored and input-output is performed in computers

Desirable/Advanced (<= 3):

1. Super-scalar and multi-core architectures for crossing one clock per instruction barrier
2. Cache data coherence related challenges in multi-core processors

Detailed contents for Essential Learning Outcomes:

Module (appx dur in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Introduction (Lectures 6) (Weeks 2)	Role of abstraction, basic functional units of a computer, Von-Neumann model of computation, A note on Moore's law, Notion of IPC, and performance. Data representation and basic operations.		
Module 2: Instruction Set Architecture (RISC-V) (Lectures 8/9) (Weeks 3)	CPU registers, instruction format and encoding, addressing modes, instruction set, instruction types, instruction decoding and execution, basic instruction cycle, Reduced Instruction Set Computer (RISC), Complex Instruction Set Computer (CISC), RISC-		Lab Modules 1 and 4

	V instructions; X86 Instruction set.		
Module 3: The Processor (Lectures 6) (Weeks 2)	Revisiting clocking methodology, Amdahl's law, Building a data path and control, single cycle processor, multi-cycle processor, instruction pipelining, Notion of ILP, data and control hazards and their mitigations.		Lab Modules 2 and 4
Module 4: Memory hierarchy (Lectures 8/9) (Weeks 3)	SRAM/DRAM, locality of reference, Caching: different indexing mechanisms, Trade-offs related to block size, associativity, and cache size, Processor-cache interactions for a read/write request, basic optimizations like write-through/write-back caches, Average memory access time, Cache replacement policies (LRU), Memory interleaving.		Lab Module 2
Module 5: Storage and I/O (Lectures 6) (Weeks 2)	Introduction to magnetic disks (notion of tracks, sectors), flash memory. I/O mapped, and memory mapped I/O. I/O data transfer techniques: programmed I/O, Interrupt-driven I/O, and DMA.		Lab Modules 3 and 5

Detailed Contents for Desirable Learning Outcomes (optional, <= 3 modules):

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**
Module 6: Superscalar processors and multicore systems	Limits of ILP, SMT processors, Introduction to		

(Lectures 6) (Weeks 2)	multicore systems and cache coherence issues		

Laboratory Modules

The laboratory component consists of 5 modules out of which three can be chosen based on the overall curriculum and its emphasis. Modules 4 and 5 assume that students have a background in using FPGA kits in their Digital Electronics course (ESC302) and have also been introduced in programming in one of the HDLs (VHD or Verilog).

Possible three options are:

1. Module 1 + Module 2 + Module 3

   Instructions & assembly language + basic performance + advanced performance analysis
2. Architecture + Module 1 + Module 2 + Module 4

   Instructions & assembly language + basic performance + basic processor design

   Module 1 + Module 4 + Module 5
3. Instructions & assembly language + basic processor design + I/O and architecture enhancements.

Detailed contents for Essential Learning Outcomes:

Module (appx dur in wks)	**Topics**	**Comments**
Module 1: (Weeks 4) Objective: Understanding architecture and instructions through assembly programming	Write programs in ARM/RISC V assembly language and test these on an instruction set simulator. Typical examples are given below. Some of these are dependent on I/O facilities provided by the simulator. ● Generate some interesting numbers (example - Happy numbers, Autonomic numbers, Hardy-Ramanujan numbers etc.) ● Implement a 4-function calculator ● Sort an integer array using merge sort (recursive) ● Evaluate an arithmetic expression specified as a	Essential component

	string (using recursive functions) ● Implement a simple game	
Module 2: Understanding performance issues related to pipelining and cache using architectural simulator (Weeks 4/5)	Usage of a instruction pipeline visualization tool like RIPES Write or generate sequence of instructions and observe the overall pipeline stalls with and without data hazards, control hazards, and with/without data forwarding. Rearrange the sequence of instructions or the program so that the pipeline stalls will be minimized.	Optional
Module 3: Understanding memory access patterns and changing basic cache memory parameters to analyze the impact of standard programs or benchmarks using architectural simulators. (Weeks 3)	Configure the simulator [gem5 is preferred] to operate on the binaries of the benchmark as the input. Run the program and examine the IPC, cache hit rate, number of conflict misses and block replacements. Vary the cache size, block size, and associativity and analyze the metrics and reason the changes observed. Modify the block replacement algorithms and see the impact at cache memory performance Calculate the access time, power and are associated with a given cache configuration. Vary the cache size, block size, and associativity and analyze the metrics and reason the changes observed.	Optional: Familiarity with tools like GEM5, CACTI and PIN, and access to benchmarks like SPEC, PARSEC, SPLASH. Any other tools/simulators that can support memory pattern analysis is also fine.
Module 4: (Weeks 4/5)	Design a simple ARM/RISC V processor for a small subset of	Optional Prerequisite:

Objective: Understanding computer architecture by designing a CPU on FPGA kit	instructions and implement on FPGA board. This is done in stages as follows. ● Design CPU for the instruction subset {add, sub, cmp, mov, ldr, str, beq, bne, b}, with each instruction executing in a single cycle. No sequential control required. For each instruction, only a limited set of variants are considered. ● Use memory generator to add program and data memories. ● Include circuit for single step execution and for displaying signals of interest. ● Modify the design to allow multi-cycle execution of instructions. ● Enhance the design to include all DP instructions and all variants of the second operand.	Use of FPGA kits as well as exposure to VHDL/Verilog
Module 5: Objective: Understanding computer architecture performance issues as well as I/O through architectural enhancements (on FPGA) (Weeks 4/5)	Extension of the CPU design and I/O programming ● Enhance the design to include all variants of DT instructions. ● Implement multiply group of instructions. ● Enhance the design to implement "branch and link" instruction and include full predication. ● Include limited exception handling. ● Interface 7-segment display and 4x4 keypad. ● Demonstrate execution of simple programs.	Optional Prerequisite: Use of FPGA kits as well as exposure to VHDL/Verilog

Suggested text books / Online lectures or tutorials:

1. "Computer Organization and Design: The Hardware/Software Interface", David A. Patterson and John L. Hennessy, 5th Ed, Elsevier,

Suggested reference books / Online resources:

2. "Computer Organisation & Architecture", Smruti Ranjan Sarangi, McGraw Hill
3. "Computer System Architecture", Mano M. Morris, Pearson.
4. "Computer Organization and Embedded Systems", 6th Edition by Carl Hamacher, McGraw Hill Higher Education
5. "Computer Architecture and Organization", 3rd Edition by John P. Hayes, WCB/McGraw-Hill
6. "Computer Organization and Architecture: Designing for Performance", 10th Edition by William Stallings, Pearson Education.
7. "Computer System Design and Architecture", 2nd Edition by Vincent P. Heuring and Harry F. Jordan, Pearson Education.

Online simulators and tools (for labs):

RIPES: https://freesoft.dev/program/108505982

GEM5: https://www.gem5.org/documentation/learning_gem5/introduction/

CACTI: https://github.com/HewlettPackard/cacti

PIN: https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-binary-instrumentation-tool-downloads.html

TEJAS: https://www.cse.iitd.ac.in/~srsarangi/archbooksoft.html

XILINX (VHDL/Verilog tools): https://www.xilinx.com/support/university/students.html

Prepared by:

1. Prof. Anshul Kumar, IIT Delhi
2. Prof. Anupam Basu, IIT Kharagpur/NIT Durgapur
3. Prof. Indranil Sengupa, IIT Kharagpur
4. Prof. Biswabandan Panda, IIT Bombay
5. Prof. John Jose, IIT Guwahati
6. Prof. M. Balakrishnan, IIT Delhi

The Discipline Graduate Attributes (GAs) to which this course contributes significantly: CS3 (and also to some graduate attributes for ECE)

Other discipline GAs to which this course may contribute somewhat: CS4, CS6

Advanced Programming

CS-X	**Advanced Programming**	**3L:1T: 0P**	**4 credits**	**Pre-Reqs: 0**

Familiarity with data structures, and introduction to programming.

Learning Outcomes of the course (i.e., statements on students' understanding and skills at the end of the course the student shall have):

Essential (<=6):

1. Understanding with the build system, IDE, tools for testing, debugging, profiling, and source code management
2. Students are able to demonstrate proficiency in object-oriented programming
3. Identify and abstract the programming task involved for a given programming problem
4. Learning and using language libraries for building large programs
5. Ability to apply defensive programming techniques (e.g., assertions, exceptions)

Desirable/Advanced:

1. Ability to implement basic event-driven programming
2. Understanding with the fundamentals of parallel programming
3. Understanding of the basics of cloud computing

Detailed contents for Essential Learning Outcomes:

Module (appx dur in wks)	**Topics**	**Pedagogy/teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Familiarity with the programming environment (~1 wks)	Understanding the build system, IDE, debugging, profiling (Eclipse TPTP / gprof / VTune etc.), and source code management	R5;	Familiarity with terminal/command prompt, using git commands and github to pull/commit/push/merge code, writing, compiling and running simple programs, debugging by setting breakpoints
Module 2: Basic principles of the object-oriented	Introduction to Object-Oriented Paradigm: data encapsulation, modularity,	Chapters 2-3, T1;	Importing pre-written classesusing the **this** keyword, calling and defining methods,

development process
 	
(~1.5 wks)
	code reuse, identifying classes, attributes, methods and objects, class relationships	Chapter 10, T2;
		Chapter 7, R;
		Chapters 11-12, R1;
			writing and instantiating classes, setter/getter methods, instance variables, returning values, debugging using print function, containment and association, scope and parameter passing

Module 3: Advanced features of OOP
 	
(~ 3 wks)
	Interfaces, inheritance, polymorphism, abstract classes, immutability, copying and cloning objects	Chapters 2-3, T1;
		Chapter 10, T2;
		Chapter 7, R4;
		Chapters 11-12, R1;
			Parameter polymorphism, method resolution, declared v/s actual type, partially and fully overriding methods, calling superclass constructor from child class constructor, protected fields and methods, using an abstract parent class v/s an interface with default and abstract methods, object equality check, object comparison (Comparable/Comparator interface), Cloneable interface/copy constructor

Module 4: Unit testing
 	
(~0.5 wks)
	Unit testing, developing test suite	R6;
		R7;
			JUnit/Boost.test testing framework, assertion methods, testcase timeout, testing for exceptions, test suite

Module 5: Using language APIs (~1 wks)	Language supported libraries for handling advanced data structures	Section 13.2, T2;
		Section 13.7, R1;
			Big-O notation, Java collection framework (or Boost libraries), sorting objects, iterating over objects

Module 6: Defensive programming
 	
(~1 wks)
	Exception handling, assertions	Section 9.4, T2;
		Chapter 14, R1;
		Section 2.7, R4;
			Exception handling using try/catch block, nesting try/catch blocks, throw and throws keywords, rethrowing exceptions, handling checked

			exception, user defined exceptions
Module 7: UML and Design patterns (~2 wks)	UML diagrams, introduction to design patterns: iterator, singleton, flyweight, adapter, strategy, template, prototype, factory, façade, decorator, composite, proxy, chain of responsibility, observer, state)	Chapter 5, T1; Chapters 3-9, R3	Create detailed UML class diagrams and use case diagrams for a given problem description, relationships in use case diagrams.

Suggested text books:

T1. Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, Kelli A. Houston. Object-Oriented Analysis and Design with Applications.
T2. M. Scott. Programming Language Pragmatics. 4th edition.

Suggested reference books / Online resources:

R1. R. Sebesta. Concepts of Programming Languages. 10th edition
R2. J. Rumbaugh et al. The Unified Modeling Language Reference Manual.
R3. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Grady Booch. Design Patterns: Elements of Reusable Object-Oriented Software.
R4. P. Van Roy and S. Haridi. Concepts, Techniques, and Models of Computer Programming.
R5. https://missing.csail.mit.edu/
R6. https://www.baeldung.com/junit
R7. https://www.tutorialspoint.com/junit/index.htm

Detailed Contents for Desirable Learning Outcomes (optional, <= 3 modules):

Option-1: Ability to implement basic event-driven programming

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**
Module-1: Introduction to event-driven programming	Adding UI controls, GUI events, event handling	O1; Chapter 10, O2;	Create GUI forms/application with different GUI components (buttons, text boxes, etc.),

(~1 wks)			responding to user input by creating event handlers, productivity in creating event handlers using anonymous classes and lambda methods
Module-2: Advanced GUI programming (~2 wks)	Creating animations, connection with the database	O1	Implement simple 2-D game that uses animations (timelines, moving shapes), client-server programming (multiplayer game), object serialization/deserialization (save/reload game)

Suggested text books / Online lectures or tutorials:

[O1] Oracle documentation: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm

[O2] P. Van Roy and S. Haridi. Concepts, Techniques, and Models of Computer Programming.

Option-2: Understanding with the fundamentals of parallel programming

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**
Module-1: Introduction to multithreading (~1 wks)	Ahmdal's law, speedup, parallel efficiency, thread creation	Chapters 2 and 4, T2;	Creating and joining threads, concurrency decomposition, assigning tasks to threads, calculate parallel speedup and efficiency, debugging multithreaded programs
Module-2: Thread pool (~1 wks)	Task parallelism and thread pools (e.g., Java ForkJoin framework, OpenMP/CilkPlus/TBB in C++)	Section 13.2, T1; Section 13.7, R1;	Asynchronous task creation and joining, task parallelism for recursive parallelism, task dependency, controlling task granularity, comparing performance/productivity of

			explicit multithreading v/s using thread pools
Module-3: Mutual exclusion (~1 wks)	Race conditions, deadlocks, producer-consumer problem	Chapter 13, T1; Chapter 4, T2; Chapter 13, R1; Chapter 8, R2;	Multithreaded program to push/pop items from shared queue - using monitor/Mutex locks, conditional wait and signalling, volatile keyword

Suggested text books / Online lectures or tutorials:

T1. M. Scott. Programming Language Pragmatics. 4th edition.
T2. P. Pacheco. An Introduction to Parallel Programming.

Suggested reference books / Online resources:

R1. R. Sebesta. Concepts of Programming Languages. 10th edition
R2. P. Van Roy and S. Haridi. Concepts, Techniques, and Models of Computer Programming.

Option-3: Understanding of the basics of cloud computing

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Basic concepts plus a distributed programming framework such as MR/Spark (2 weeks)	Basics, Need, Advantages/Benefits, models such as PaaS/Saas/IaaS/, distributed/cloud computing architectures Programming frameworks such as Map-Reduce or Spark, how to write programs in these programs, features such as fault-tolerance, using commodity hardware,	T1, Ch. 1, 2, 3 Illustrate the differences between M-R style programming and C-style programming	Set up some pre installed programs and infrastructure to see how to run programs in the cloud. Programming in Map-Reduce/Spark to study scalability. Typical problems are from matrix multiplication, text processing, graphs , web crawling, and the like

Module 2: (1 week)	Key challenges in cloud/distributed computing: communication vs computation/ problem decomposition/ failures/ etc. and analyze the communication required of basic algorithms such as matrix multiplication	T2, Ch 3. T3, Ch 1	Redo exercises from week 2 to measure communication time

Suggested text books / Online lectures or tutorials:

T1: Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer, Morgan & Claypool Publishers, 2010.
T2: Parallel Computer Architecture, David Culler, J. P. Singh, and A. Gupta, Elsevier, 1998.
T3. Distributed Computing: Principles, Algorithms, and Systems, A.D. Kshemkalyani, M. Singhal, Cambridge University Press, March 2011.

Prepared by:

1) Vivek Kumar, Computer Science & Engineering Department, IIIT Delhi
2) Kishore Kothapalli, Computer Science & Engineering Department, IIIT Hyderabad
3) Swarnendu Biswas, Computer Science & Engineering Department, IIT Kanpur

The Discipline Graduate Attributes to which this course contributes significantly: CS1, CS3, CS7, CS8
Other discipline GAs to which this course may contribute somewhat: CS2

Operating Systems

PCC- CS403	**Operating Systems**	**3L:0T: 4P**	**5 credits**	**Pre-Reqs: PCC–CS402**

Prerequisites details:

- Familiarity with the C/C++ programming language
- PCC 402: Computer Organization and Computer Architecture
- CS403 to be scheduled in semester after CS402
  (both courses not to be scheduled in the same semester)

Learning Outcomes of the course (i.e. statements on students' understanding and skills at the end of the course the student shall have):

Essential (<=6):

- To understand the role, functionality and layering of the systems software components
- To understand the design and usage of the OS API and OS mechanisms
- To understand the details of the abstractions and interfaces provided by the OS for program execution and execution requirements --- processes, threads, memory management, files.
- To understand problems arising due to concurrency and related synchronization based solutions.
- Hands-on and practical experience with usage of the OS API and basics of OS mechanisms

Desirable/Advanced (<= 3):

1. To gain an in-depth understanding of the design and implementation of OS internals via a teaching OS
2. To be able to implement incremental changes to the functionality of a teaching OS

Detailed contents for Essential Learning Outcomes:

Module (appx dur in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Introduction to Operating Systems (1 week)	Application requirements The systems stack and role of OS, resources, abstractions and interfaces Components overview of an OS Examples of different types of OSes (RTOS vs. desktop vs. mobile etc.), OS and OS distributions.	T1: Chapter 2 T2: Chapters 1, 2 T3: Chapter 1 R1: Chapter 1 R2: Chapters 1, 2 R4: Chapter 1 R7:Chapter 1 R8: Chapters 1 to 8	1. Usage of tools --- unix shell commands (file commands, ps, ls, top), text editor (nano, vi, gedit, emacs) *2. C programming language* refresher --- header files, compilation and linking using gcc,

		O1: Bash Guide for Beginners	program execution, functions, argument passing, structures, pointers, file handling.
Module 2: Computer organization and computer architecture refresher (1 week)	Basic organization of hardware components Role of OS relative to hardware functionality with examples related to the von Neumann architecture	T2: Chapter 1 T3: Chapter 1 R1: Chapter 1 R2: Chapters 1, 2 O1: Bash Guide for Beginners	1. Usage of tools --- gcc, gdb, objdump, shell scripts
Module 3: Processes (2 weeks)	Process abstraction --- program vs. process, Process Control Block (PCB) Design of system calls --- invocation and basic OS handling Process control system calls --- fork, wait, exec, getpid, getppid and variants The limited direct execution model	T1: Chapters 4,5,6 T2: Chapters 6, 7 T3: Chapters 7, 8	1. Simple strace usage to showcase different interfaces (stdlib, system call) 2. Tools usage --- ps, pstree, top 3. Usage of process control system calls to identity process identifiers, create process hierarchies, launch new executables, control exit sequence of parent and child processes. 4. Familiarity with files in the /proc/<pid>/ directory
Module 4: Memory management (3 weeks)	Address bus and memory access Memory view of a process ---- heap, stack, code, data Process memory usage requirements	T1: Chapters 13,14,15,16, 17,18,19, 20 T2: Chapter 9	1. (Virtual) addresses of variables and initialized pointers. 2. Use of malloc() and demonstration of

	The address space abstraction and system calls (mmap, munmap, sbrk, mprotect) Address translation mechanisms --- static mapping, segmentation,paging Page faults, page sharing, read/write permissions, swapping, process vs. OS memory Memory bookkeeping and management --- motivation and mechanisms (process and OS) Case studies --- (i) malloc and (ii) role of OS for program to process	R8: Chapters 5,6	per-process virtual addresses 3. Tools usage --- strace, free, top, htop, vmstat, /proc/<pid>/maps 4. Free memory statistics correlated with malloc(). Number of system calls and malloc() usage. 5. Implement a custom memory allocator using system calls
Module 5: Process management (2 weeks)	The process lifecycle---source code to execution The OS mode of execution --- limited direct execution recap, interrupts, system calls The process context switch mechanism and PCB state Scheduling policies --- set of scheduling metrics, goals and examples (interactive vs. realtime, priority)	T1: Chapters 7,8 T2: Chapter 8	1. User mode programs to demonstrate LDE 2. Demonstration of process execution interleaving in different orders 3. Simulation based analysis of scheduling policies 4. Tools usage --- nice, /proc/<pid>/status
Module 6: Concurrency and Synchronization (3 weeks)	Motivation --- application, process and OS use cases. Introduction to threads and the pthread API Synchronization primitives --- limitations of software solutions, atomic instructions, test-and-set, spinlocks, mutexes, condition variables, semaphores	T1: Chapters 26, 27, 28, 29, 30, 31, 32 T3: Chapters 11, 12	1. Creation of threads using the pthread API and modification of shared variables with and without synchronization 2. Using spinlock, mutexes and condition variables to implement

	Introduction to the pthread synchronization API Case studies --- producer-consumer, reader-writers, barriers Discussion on issues with concurrency		semaphores, barriers 3. Implement solutions to the producer-consumer, reader-writers problems using the different synchronization primitives 4. Develop synchronization solutions for applications that use shared data (e.g., ordering of threads, concurrent hash tables, etc.)
Module 7: File systems (2 weeks)	Persistence and the File abstraction Hardware view: Hard disk architecture and its interfacing Process view: System calls for file handling Roles and responsibilities of file system File system design details---file and file system metadata, directory structure, caching optimizations File System case study (the unix file system etc.)	T1: Chapters 36,37,39, 40 T2: Chapters 4,5 T3: Chapter 3	1. Tools usage --- state, file, du, df, fsck, ... 2. Implementation of file utilities (e.g., find, grep, …) using the system call API. 3. Implement a simple file system to handle files on an emulated disk (via a large file) --- file system API, superblock, inode and data block management.

Detailed Contents for Desirable Learning Outcomes (optional, <= 3 modules):

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**

1. Introduction to setup and usage of a teaching OS **(2 weeks)**	Setup, configuration and usage of a teaching OS Understanding ISA details of the teaching OS Basics of interrupt handling, end-to-end execution of a system call and context switching mechanisms	T4	1. Configuration and setup of a teaching OS and writing of simple user mode programs (e.g., xv6) 2. Simple logging style modification to system calls (e.g., #times a system call was invoked, histogram of number of system call invocations, …) 3. Implement new system calls (e.g., read PCB elements to user space of a process) (xv6 based)
2. Basic modifications to the teaching OS functionality saved	OS context of execution --- address space, stack Address translation mechanism in the teaching OS The interrupt handling mechanism (context save and restore, privilege level change, interrupt dispatch) Example usage of locks in the teaching OS File system state in the operating system (file descriptors, file objects, inodes, page cache …) and in the teaching OS	T4	1. Write a system to call to output per process address space details 2. Write a system to call to determine physical address of a virtual address 3. Observe the stack pointers, privilege level registers in user and OS modes 4. Modifying/profiling behaviour of exception handlers 5. Observe process file table entries and file objects across parent and child processes
3. Advanced modifications to teaching OS functionality	Context switch design and implementation in the teaching OS	T4	1. Write a system call to allocate the same physical block to

	Lazy allocation, page fault management and swapping/demand paging Understand implementation of the synchronization primitives in the teaching OS		different virtual addresses 2. Implement lazy allocation of physical memory to processes 3. System call to print saved state of any process 4. Write a system call to induce page faults 5. Design and implement a shared message queue between processes to be used via the system call interface

Suggested text books / Online lectures or tutorials:

T1. *Operating Systems: Three Easy Pieces*
Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
Arpaci-Dusseau Books, LLC
https://pages.cs.wisc.edu/~remzi/OSTEP/ (online version)

T2. *Design of the UNIX Operating System*
Maurice J. Bach
Pearson Education India; First edition

T3. *Advanced Programming in the UNIX® Environment*
W. Richard Stevens, Stephen A. Rago
Pearson Education India; Third edition

T4. *Xv6, a simple Unix-like teaching operating system*
Frans Kaashoek, Robert Morris, and Russ Cox
[T4-R] https://github.com/mit-pdos/xv6-riscv (RISC-V version)
[T4-X] https://github.com/mit-pdos/xv6-public (x86 version)

Suggested Online content:

O1. The Linux Documentation Project
www.tldp.org

Suggested reference books / Online resources:

R1. *Modern Operating Systems*, Andrew S. Tannenbaum and Herbert Bos, Pearson Education India; 4th edd

R2. *Operating System Concepts*, Avi Silberschatz, Peter Baer Galvin, Greg Gagne, Wiley India; 9th, ed

R3. Operating System courses offered on NPTEL, https://nptel.ac.in/

R4. *Think OS, A Brief Introduction to Operating Systems.* Allen B. Downey https://www.greenteapress.com/thinkos/index.html

R5. *Linux Kernel Development* , Robert Love, Pearson Education India; 3rd edition

R6. *Operating Systems: Principles and Practice*, Thomas Anderson, Michael Dahlin, Recursive Books; 2nd Edition, https://ospp.cs.washington.edu/index.html

R7. *Computer Systems: A Programmer's Perspective*, Randall E. Bryant, David R. O'Hallaron, Pearson Education India; 3rd edition

R8. *The C Programming Language,* Brian Kernighan, Dennis Ritchie, Pearson Education India; 2nd ed

Prepared by:

1. Purushottam Kulkarni, Department of Computer Science and Engineering
   Indian Institute of Technology Bombay
2. Chester Rebeiro, Department of Computer Science and Engineering
   Indian Institute of Technology Madras
3. Debadatta Mishra, Department of Computer Science and Engineering
   Indian Institute of Technology Kanpur

The Discipline Graduate Attributes (GAs) to which this course contributes significantly: CS3

Other discipline GAs to which this course may contribute somewhat: CS6, CS8

Design and Analysis of Algorithms

CS404	**Design and Analysis of Algorithms**	**3L:0T: 4P**	**5 credits**	**Pre-Reqs: ESC201, CS301**

Learning Outcomes of the course (i.e. statements on students' understanding and skills at the end of the course the student shall have):

Essential (<=6):

1. Analyze the asymptotic performance of algorithms.
2. Establish the correctness of algorithms.
3. Demonstrate familiarity with major algorithms and data structures.
4. Apply important algorithmic design paradigms and methods of analysis.
5. Synthesize efficient algorithms in common engineering design situations.
6. Understanding limits of efficient computation.

Desirable/Advanced (<= 3):

1. Intuitive understanding of what makes a problem NP-hard.
2. Apply network flow techniques to algorithm design.

Detailed contents for Essential Learning Outcomes:

Module (appx dur in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Applications of Graph Search (~1.5 wks)	(i) Review of BFS/DFS (ii) Checking if an undirected graph is 2-edge connected. (iii) Checking if a directed graph is strongly connected. Strongly connected components in graphs.	**T1: Chapter 3, T2: Chapter 3, R1: Chapter 22.** Review breadth first search (BFS) and depth first search (DFS) in undirected and directed graphs; Introduce notion of a bridge and cut-vertex.	Checking if a graph is biconnected

Module 2: Greedy algorithms (2 wks)	(i) Introduction to the greedy paradigm (ii) Examples of activity selection, deadline scheduling, fractional knapsack, Kruskal's algorithm for minimum spanning trees, Huffman coding.	**T1: Chapter 4, T2: Chapter 5.** Exchange arguments are a useful recipe for proving correctness of greedy algorithms. Illustrate these through examples. Show examples where greedy does not give an optimum solution	(i) Prim's algorithm for minimum spanning trees. (ii) Examples where greedy algorithms are not optimal.
Module 3: Divide and Conquer (2 wks)	(i) Explain why the divide and conquer paradigm is useful. (ii) Illustrate the paradigm through integer multiplication. (iii) Writing recurrence relations and solving them. (iv) Further examples from geometry – domination number of a set of points, identifying maximal points, closest pair of points. (iv) Linear time algorithm for finding median.	**T1: Chapter 5, T2: Chapter 2.** Solve recurrences by building the recurrence tree. Motivate choices of parameters in median finding algorithm by showing how recurrence tree changes.	(i) Solve some recurrence relations. (ii) Modify discussed algorithms (e.g., dividing into three parts instead of two parts, or two unequal parts, etc.) and analyse using recurrences.
Module 4: Dynamic Programming and shortest paths (2.5 wks)	(i) Computing Fibonacci numbers and why divide-and-conquer is not a good idea. Idea of storing function calls, tables. (ii) Notion of subproblems and optimal substructure.	**T1: Chapter 6, T2: Chapter 4, Chapter 6.**	Exercises on dynamic programming (textbook problems)

	(iii) Illustration through subset sum, (integer) knapsack, longest increasing subsequence, longest common subsequence, matrix chain multiplication. Dijkstra's algorithm for single-source shortest paths, Bellman-Ford for SSSP with -ve weights, Floyd Warshall for APSP.	Discuss why Dijkstra's algorithm is an example of dynamic programming. Extending Bellman-Ford to APSP and to find negative cycles. Discuss how dynamic programming problems can often be cast as longest paths in acyclic graphs.	
Module 5: Network flows (2 wks)	The maximum s-t flow problem in capacitated networks. Ford Fulkerson algorithm or maximum flow. Max-flow min-cut theorem and integrality of maximum flow for integral capacities. Applications of max flow to maximum bipartite matching, max disjoint paths	**T1: Chapter 7. 1-7.2, 7.5, 7.6** Notion of residual capacities and residual graphs and how this allows us to correct greedy decisions made in earlier steps of FF algorithm	(i) Some simple examples. (ii) Implementation of Ford Fulkerson algorithm.
Module 6: Intractability (2wks)	(i) Notion of polynomial time computation. (ii) Polynomial time reductions. Yes and No instances of decision problems. Decision vs optimization. (iii) NP as a class of problems with Yes certificates which can be efficiently checked. (iv) NP-hardness and Cook-Levin theorem (just the statement). (v) NP-completeness. Examples of Reductions.	**T1: Chapter 8, T2: Chapter 8**	Exercises on reductions, NP-completeness.

		(i) Emphasize how reduction can be used to solve a problem using an algorithm for a different problem. (ii) Emphasize the asymmetry in the definition of NP: No efficiently checkable NO certificates for problems in NP. (iii) Problems which are NP-hard but not in NP. (iv) Examples of poly time reductions. Polytime as a measure of efficiency. Hardness only for the general instance, whereas special instances can be efficiently solvable.	

Suggested books:

1. Algorithm Design, 1ST Edition, Jon Kleinberg and ÉvaTardos, Pearson.
2. Algorithms, Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani

Suggested reference books

1. Introduction to Algorithms, 4TH Edition, Thomas H Cormen, Charles E Lieserson, Ronald L Rivest and Clifford Stein, MIT Press/McGraw-Hill.
2. Algorithm Design: Foundations, Analysis, and Internet Examples, Second Edition, Michael T Goodrich and Roberto Tamassia, Wiley.

Detailed Contents for Desirable Learning Outcomes (optional, <= 3 modules):

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**
Module1: FFT	Polynomial multiplication using FFT and DFT.	**T1: Chapter 5.5, T2: Chapter 2.5** (i) Explain the interplay between the two representations of a polynomial. (ii) Explain why complex numbers arise naturally here.	Take two polynomials and multiply them by first computing the point-value representation using FFT and then recovering product using inverse FFT
Module2:	Beyond NP-completeness: approximations, exponential algorithms, popular heuristics.	**T1: Chapter 10, T2: Chapter 9** (i) Show that TSP can be solved in $2^n\, poly(n)$ time (rather than n! Time), and vertex cover in $2^k\, poly(n)$, where $k$is the size of the optimal vertex cover. (ii) Give examples of some simple approximation algorithms, e.g., bin packing, vertex cover. (iii) Popular heuristics for satisfiability.	(i) Implementation of some exponential time algorithms and heuristics and see how they scale with large n.

The Discipline Graduate Attributes (GAs) to which this course contributes significantly: CS1, CS2, CS4

Other discipline GAs to which this course may contribute somewhat:

Database Systems

Course code: ?	Introduction to Database Systems	3L:0T: 4P	Credits: ?	Pre-Reqs: Data Structures and Algorithms

Learning Outcomes of the course (i.e. statements on students' understanding and skills at the end of the course the student shall have):

Essential (<=6):

1. Ability to design and implement database schema for an application using RDBMS concepts.
2. Ability to write SQL queries for tasks of various complexities.
3. Ability to write an application program that uses a database system as the backend.
4. Understanding of internal working of a DBMS including data storage, indexing, query processing, transaction processing, concurrency control and recovery mechanisms.
5. Awareness of non-relational and parallel/distributed data management systems with a focus on scalability.

Desirable/Advanced (<= 3):

- Nil

Detailed contents for Essential Learning Outcomes:

Module (appx dur in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Introduction (2 hours)	(i) Motivation (ii) Introduction to Data Models (Relational, Semistructured, ER)		

Module 2: Relational Databases (4 hours)	(i) Relational Data Model (ii) Relational Algebra (iii) Relational Calculus or Connection to First Order Logic (Optional)		(i) Simple pen+paper, and using Relax Relational Algebra calculator in browser
Module 3: SQL (7 hours)	(i) DDL (ii) Insert/Delete/Update (iii) Simple Queries (select/project/join/ aggregate queries) (iv) Complex queries (With Clause, Nested Subqueries, Views)		(i) Laboratory exercises where students write SQL queries for various tasks. Platform can be PostgreSQL preferably, or MySQL. W3Schools/SQLite in web browser can also be used but beware of non-standard SQL features.
Module 4: Big Data: Key-value Stores and Semi-structured Data (JSON) (2 hours)			(i) Small exercises on MongoDB
Module 5: Database Design (6 hours)	(i) Introduction to ER model (ii) Mapping from ER to relational model (iii)Functional Dependencies (iv) Normalization (BCNF, Optionally 3NF)		(i) Exercise in ER design for an application starting with natural language description (ii) Convert ER design to tables (iii) Pen-and-paper exercises with FDs and normalization
Module 6: Physical Design (5 hours)	(i) Overview of Physical Storage (Hard Disks, Flash/SSD/RAM), sequential vs random I/O, Reliability via RAID		(i) Use a B+-tree visualization system to understand how B+-trees work

	(ii) Storage Organization (Records, Pages and Files), Database Buffers, Database Metadata (iii) Indexing, B+-Trees		
Module 7: Query Processing and Optimization (5 hours)	(i) Query Processing: External sort, Joins using nested loops, indexed nested loops (ii) Overview of Query Optimization: equivalent expressions, and concept of cost-based optimization		(i) Examine query plans for sample queries by using the Explain feature of database systems. (ii) Small exercises to show benefit of indices.
Module 8: Transaction Processing (8 hours)	(i) Concept of transactions and schedules, ACID properties (ii) Conflict-serializability (iii) Concurrency control: locks, 2PL, Strict 2PL, optional: isolation levels (iv) Recovery using undo and redo logs		(i) Pen-and-paper exercises on conflicts, cycles, conflict serializability, recoverability, etc.

Suggested text books / Online lectures or tutorials:

1. Database System Concepts, 7th Ed, Silberschatz, Korth and Sudarshan, McGraw-Hill. Indian Edition released 2021
2. Fundamentals of Database Systems, 7th Ed, Elmasri and Navathe, Pearson Pubs, 2017
3. Principles of Database Management, Lemahieu, Broucke and Baesens, Cambridge University Press, 2018

Suggested reference books / Online resources:

- Software
    - Relax Relational algebra calculator: https://dbis-uibk.github.io/relax/landing
    - SQL: PostgreSQL/MySQL/MariaDB, or SQLite in browser
    - B+-tree visualization: https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html
    - MongoDB
    - Various DB systems playground: https://www.pdbmbook.com/playground

The Discipline Graduate Attributes (GAs) to which this course contributes significantly: CS3, CS5, CS6

Other discipline GAs to which this course may contribute somewhat: CS1, CS7, CS9

Computer Networks

CS602	**Computer Networks**	**3L:0T: 2P**	**4 credits**	**Pre-Reqs: CS402 Computer Organisation, CS403 Operating Systems**

Pedagogy and Learning Outcomes

This course introduces students to the fundamental principles of computer networks, and to their use in the Internet. Lab assignments cover network programming, network tools, applications, and simulation. Through these assignments, students get a broad understanding by building, operating and tuning components of the Internet. The top-down, application-driven design of the course motivates students with familiar uses and problems of their digital world and enables them to work with real-world applications from early in the course.

At the end of the course the student shall:

Essential **(<=6):**

1. Understand the architecture principles that have enabled the orders of magnitude expansion of the Internet
2. Understand networked applications and their protocols, their installation, operation and performance tuning
3. Understand layering as a means of tackling complexity, layering applied to the Internet
4. Understand protocols as a structured means of reliable communications
5. Be conversant with network programming using the socket API
6. Be familiar with tools for configuring, monitoring and tuning the Internet and hosts

Desirable/Advanced **(<= 3):**

1. Understand basics of data centre networks and software-defined networks (SDN)
2. Understand cellular networks, mobility and the impact on applications
3. Understand the interaction between streaming media applications and the underlying network infrastructure.

Detailed contents for Essential Learning Outcomes:

Module (approx duration in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Lab / assignment / practice**
Module 1: Introduction to the Internet (1 week)	- Overview of how the Internet works. Understand at a high level what happens when we browse a website. Understand basic terminology like browser, web server, URL, domain name, IP address, packets. (1 hour) - Overview of the design principles of the Internet: packet switching vs circuit switching, store-and-forward networks, layering for modularity. Introduction to the various layers in the Internet. (1 hour) - Introduction to performance metrics like end-to-end throughput, delay, jitter and drop rates in a network. Statement of Little's Law. How performance is measured. Confidence intervals. (1 hour)	**Kurose & Ross**: Chap 1 (sec. 1.1-1.5). **Raj Jain**: Sec. 3.2 - 3.4 Top-down sequence is recommended. Performance metrics and measurement concepts are introduced in Module 1 and used in all modules (lectures and labs) as appropriate.	- Use Linux tools like ifconfig, dig, ethtool, route, netstat, nslookup, and ip to understand the networking configuration of the computer that the student is working on. - Use Wireshark to capture packets when browsing the Internet. Examine the structure of packets: the various layers, protocols, headers, payload. This resource is useful for many of the lab assignments: **https://gaia.cs.umass.edu/kurose_ross/about.php**
Module 2: Application layer (2 weeks)	- Internet names, how DNS works. (1 hour) - Application layer protocols: HTTP, SMTP, SNMP, web applications. (3 hours) - Peer-to-peer applications. P2P file distribution. (1 hour) - Audio and video streaming. Challenges of streaming over best effort IP. (1 hour)	**Kurose and Ross**: Chap 2 (sec. 2.1-2.5), Sec. 7.1.3	- Install and configure some network applications, eg. Apache, Bind (DNS), etc. - Understand various header fields and their usage in different application layer protocols using Wireshark packet capture.

Module (approx duration in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Lab / assignment / practice**
Module 3: Linux Network Programming (1 week)	- Introduction to socket programming in Linux. Understand how to build a simple client-server application using TCP/UDP sockets. (2 hours)	**Kurose and Ross**: Sec 2.7.	- Socket programming: write a simple client-server program using TCP and UDP sockets. **Optional**: Modify server to handle multiple clients concurrently.
Module 4: Transport Layer (2 wks)	- Importance of the transport layer; end-to-end principle. Transport layer protocols: basics of TCP and UDP, process-to-process delivery, multiplexing, port numbers, header structure. (2 hours) - Reliable transmission of packets over an unreliable network: sequence numbers, ACKs, timeout, retransmissions. Stop and wait, Go-back-N and sliding window. (2 hours) - TCP reliability. TCP connection setup and teardown. (1 hour) - Flow control and congestion control at the transport layer. Differences between the two. (1 hour) - TCP congestion control: Slow start; congestion avoidance using loss-based and delay-based control. (1 hour)	**Kurose & Ross**: Chap. 3 (sec. 3.1-3.6) **Lab**: Optional -- https://www.isi.edu/nsnam/ns/tutorial/nsscript1.html#second Validation tests and demos: https://www.isi.edu/nsnam/ns/ns-tests.html	- Measure TCP throughput between two hosts in a network using tools like iperf. Modify TCP configuration parameters. Use the **tc** Linux utility or similar to control bandwidth, delay, loss. Observe impact on measured throughput. - Experiment with multiple applications running concurrently to generate congestion. **Optional**: Observe the behavior of congestion control protocols in ns-2/ns-3, change various network parameters and observe evolution of the TCP congestion window.

Module (approx duration in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Lab / assignment / practice**
Module 5: The IP Layer (2 wks)	**[A] Network architecture and performance** - Network topology; Router architecture: queueing and switching. (2 hour) - Performance analysis of a single link: Basics of M/M/1 queues, traffic characteristics, performance measures, Kendall's notation. (1 hour) **[B] IP Protocol** - Need for an Internet address, and its design. Hierarchical IP addressing, IPv4 and IPv6, structure of IP datagram, IP forwarding. (1 hour) - Static and dynamic address allocation, private IP addresses, NATs, DMZ, firewalls. (2 hour)	**Kurose & Ross**: Chap. 1.4; Chap. 4 (Sec. 4.1, 4.3, 4.4) **S. Bose:** Chap. 1; Chap. 2.3	- Use tools like ping and traceroute to explore various Internet paths to popular servers. - Use web-based tools like the **whois** utility to query Internet registries, and understand which IP addresses are allocated to the student's network. Find out which are the major ISPs, and which is the ISP of the student's network.
Module 6: Routing protocols and Internet architecture (2 wks)	-Routing protocols: Link state routing. Distance vector routing: count-to-infinity, routing convergence. (3 hours) - Understand the structure of the Internet: end-user organizations and ISPs. Understand the difference between intra-domain and inter-domain routing. Intra-domain routing: OSPF. (2 hours) - Inter-domain routing: brief overview of BGP. (1 hour)	**Kurose & Ross**: Chap 4 (Sec. 4.5 - 4.6)	- Configure a simple mesh network using computers in the lab, or using Mininet. Setup static routes to conform to the desired mesh topology. - Use NS-2/NS-3 to simulate a mesh of at least 4 nodes and 3 links to evaluate performance under various conditions

Module (approx duration in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Lab / assignment / practice**
Module 7: Data Link Layer (2 wks)	- Mechanisms for error detection/recovery: Parity checks, CRC (1 hour) - Medium access protocols: Polling vs. contention-based: TDM, FDM, slotted Aloha, Aloha, CSMA/CD (3 hours) - Switched LANs: L2 addressing and ARP, Ethernet frame structure, learning switches. (2 hours)	**Kurose & Ross**: Chap. 5 (Sec. 5.1 - 5.4)	- Use Linux network tools like ethtool to observe and analyze link layer packet statistics and errors. **Optional**: Use NS-2/NS-3 to simulate medium access protocols. Observe contention, collisions and packet loss in medium access protocols. Observe the working of error detection/recovery mechanisms.
Module 8: Wireless Networks (1 wk)	- Wireless physical layer: signal-to-noise ratio, bit error rate, modulation, multipath, interference (1 hour) - Wireless LANs: 802.11 architecture (access points, SSID, channels, beacons, scanning, association), Hidden terminal problem, 802.11 CSMA-CA protocol; summary of 802.11 variants (2 hours)	**Kurose & Ross**: Chap 6 (Sec. 6.1, 6.2, 6.3)	- Use cellphone to measure WiFi signal strength (RSS) at various places in the campus. Draw a contour map with access points and RSS levels. Correlate with upload/download speed using tools like Measurement Lab speedtest. **Optional**: Understand the behavior of WiFi using NS-2/NS-3.

Detailed Contents for Desirable Learning Outcomes (optional, <= 3 modules):

Module (approx duration in wks)	**Topics**	**Pedagogy/ teaching suggestions**	**Lab / assignment / practice**
Module D1: SDN and data centre networking	Data centre network design and topology. Transport protocols optimized for data centres. Introduction to software defined networking.	**Kurose and Ross:** Chap. 5.6 Refs [5], [6]	Simulate transport protocols optimized for data centres in NS-2/NS-3.
Module D2: Mobile data networks	Cellular Internet access. Mobility management and handovers. Impact of mobility on higher layers.	**Kurose and Ross:** Chap. 6.4-6.8	Use cellphone to measure cellular signal strength (RSS) at various places in the campus. Draw a contour map with cellphone towers and RSS levels. Correlate with upload/download speed using tools like Measurement Lab speedtest.
Module D3: Streaming media protocols and services	Multimedia applications. Audio and video streaming over UDP, HTTP. Adaptive streaming. Voice over IP. Recovering from packet loss and jitter. QoS. Protocols for real time applications.	**Kurose and Ross:** Chap. 7	Implement a streaming audio/video server using open-source software.

Suggested text books / Online lectures or tutorials:

1. J.F. Kurose and K.F. Ross, *Computer networking: a top-down approach*, 6th ed, Pearson, 2017. (*6th ed. is low-cost Indian edition. 7th ed. is high-cost, may be used if available.)*

Suggested reference books / Online resources:

1. R. Jain, *The art of computer systems performance analysis,* Wiley India, 1991.
2. S.K. Bose, *An Introduction to Queueing Systems*, Springer Science+Business Media New York, 2012.
3. A.S. Tanenbaum and D.J. Wetherall, *Computer Networks*, 5th ed., Pearson, 2013.
4. Larry Peterson and Bruce Davie, *Computer Networks: A Systems Approach*, 6th Ed., available at https://book.systemsapproach.org/
5. N. Feamster, J. Rexford, Ellen Zegura, "The Road to SDN", *ACM Queue*, 2013.
6. Alizadeh et al., "Data Center TCP", *ACM SIGCOMM 2010*.
7. *The Network Simulator - ns-2*, https://www.isi.edu/nsnam/ns/
8. E. Altman and T. Jimenez, *NS2 Simulator for Beginners*, 2003.
9. *ns-3 network simulator*. https://www.nsnam.org/

Prepared by (alphabetical order)**:**

1. Timothy A Gonsalves, School of Computing & Electrical Engg, IIT Mandi
2. BN Jain, Department of Computer Science & Engg, IIIT Delhi
3. Vinay Ribeiro, Department of Computer Science & Engg, IIT Bombay
4. PM Sreelakshmi, School of Computing & Electrical Engg, IIT Mandi
5. Mythili Vutukuru, Department of Computer Science & Engg, IIT Bombay

The Discipline Graduate Attributes (GAs) to which this course contributes significantly: CS3

Other discipline GAs to which this course may contribute somewhat: CS4, CS6, CS8

Machine Learning

CSXXX	**Machine Learning**	**3L:1T: 0P**	**4 credits**	**Pre-Reqs: Engg. mathematics (LinAlg, Prob), Algorithms**

Learning Outcomes of the course (i.e., statements on students' understanding and skills at the end of the course the student shall have):
Essential (<=6):

- Understanding popular ML algorithms with their associated mathematical foundations.
- Capability to implement basic algorithms using basic (python) libraries. Have hands-on experience in applying ML to problems encountered in various domains. Have exposure to high level ML libraries or frameworks such as TF, pytorch.
- Make aware of the role of data in the future of computing and solving real-world problems.
- Helping them connect/map real-world problems to the appropriate ML algorithm(s) to solve them
- Appreciate the mathematical background behind the popular ML algorithms
- Have awareness about the importance of core CS principles such as algorithmic thinking and systems design in ML

Desirable/Advanced (<= 3):

- Have a solid mathematical understanding of the popular ML algorithms
- Preparedness to use state of the art machine learning algorithms in formulating and solving new problems.
- Capability to train (or solve optimization problems) ML models with applications in real world use cases.

Detailed contents for Essential Learning Outcomes:

Module (appx duration of 3 weeks or 9-12 Hrs)	**Topics**	**Pedagogy / teaching suggestions**	**Nature of lab / assignment / practice**
Introduction to ML	(i) Motivation and role of machine Learning in computer science and problem solving, (ii) Representation (features), linear transformations, Appreciate linear	(i)Connect Machine Learning to the broad Computer Science (ii)Expose the broad canvas of Machine Learning. Brief History and Importance (iii)Role of data, Connection to the knowledge /experience in learning.	(i)Experiments/Notebooks/Code that refresh Python, Programming frameworks used for the course (ii)Experiments/Code that expose allows students to appreciate mathematics and

	transformations and matrix vector operations in the context of data and representation. (iii) Problem formulations (classification and regression). (iv) Appreciate the probability distributions in the context of data, Prior probabilities and Bayes Rule. (v) Paradigms of Learning (Supervised, Unsupervised, and a brief overview of others)	(iv)Show successful examples of machine learning in Industry/working (v)Motivate students by Showing how ML and Data driven solutions could help in our day to day problems around (interdisciplinary such as agriculture, healthcare, education, living etc.) (vi)Refresh the basic mathematical notions that students may know (vectors, matrices, probabilities, etc.) with examples in ML	data manipulation. Appreciate (a) Features, Representation of the data/real-world phenomena (b) mathematica operations or transformations that manipulate the data (c) plot/visualise the data distributions (say in 2D). (iii)Lab/Experiments that appreciate the problem of Classification and Problem of Regression (iv)Lab/Experiments that appreciates the notions related to “Training” and “Testing”
Fundamentals of ML	(i) PCA and Dimensionality Reduction, (ii) Nearest Neighbours and KNN. (iii) Decision Tree Classifiers (iv) Generalization and overfitting (v) Notion of Training, Validation and Testing	(i)Focus on mathematical and algorithmic precise description of the content. (ii)Insights into these algorithms, why? When? What are the limitations? Why multiple algorithms exists for a specific problem (iii)Insights into the notion of generalization. Challenges for generalization. Assumptions to make (iv)Practical insights and tops on avoiding overfitting.	(i)Dimensionality Reduction using PCA and its applications in (a) removing irrelevant features (b) compression /compaction (c) efficient ML pipeline (ii)Experiment related to Nearest neighbour classifier, (a) visualise the decision boundaries (b) appreciate the role of hyperparameter K. Role of validation data in choice of hyper parameters (iii)Decision Tree as a classifier and see the overfitting with “deep” trees. How the overfitting can be controlled by seeing validation performance during the training.
Selected Algorithms	(i) Ensembling and RF (ii) Linear SVM, (iii) K Means,	(i)Make students appreciate the role of optimization in machine learning. Challenges in	(i)Experiments related to K-Means, by varying in “K”, “initialization”. How the

	(iv) Linear Regression (vi) Naive Bayes	optimization and why we are sometimes happy with sub-optimal solutions. How assumptions make the algorithms simple/tractable. (ii)Make students appreciate the role of uncertainty in data and machine learning problems/solutions. Give probabilistic insights into Loss functions (em MSE, cross entropy) (iii)Introduce iterative algorithms, convergence, role of initialization etc. in a class of ML solutions. (iv)Connect the geometric view of Margin (eg, linear SVM) and Probabilistic View of Margin (Logistic Regression) and the need of Generalization	“analysis of the algorithm” can be seen in the lab (eg. change of objective across iterations). Try multiple datasets. Appreciate that “unsupervised discovery” makes sense in the problem under consideration. (ii)An experiment that demonstrates how SVM can yield a solution better than a simple linear separating solution. Appreciate the role of support vectors. Appreciate how SVMs extend to problems even if data is not linearly separable. (iii)An experiment that makes students appreciate the utility of naive Bayes classifier in practice (say designing a text classifier).
Neural Network Learning	(i) Role of Loss Functions and Optimization, (ii) Gradient Descent and Perceptron/Delta Learning, (iii) MLP, (iv) Backpropagation (v) MLP for Classification and Regression, (vi) Regularisation, Early Stopping (vii) Introduction to Deep Learning (viii) CNNs	(i)Appreciate (a) the neuron model (b) the neural network and its utility in modelling and solving the problem. Connect to the biological motivations and parallelism. (ii)Expose the simple elegant optimization scheme of gradient descent with associated mathematical rigour and insights. (iii)Expose the practical issues in extending GD to multiple layers and how the backpropagation algorithm efficiently computes the gradients. (iv)Expose the practical challenges in training a neural network (such as non-convexity, initialization, size of data, number of parameters) and how they are taken care of in	(i)Experiment that exposes the GD and BP in simple neural networks. Show the learning process (graphs) and performances. (ii)Experiment that use a modern library and implementation of a deep neural network, expose computational graphs, expose the generalized way of appreciating BP as a learning algorithm in Deep Neural Networks (iii)Experiment that uses a popular CNN architecture for practical application (say image classification). (iv)Experiments that strengthen the empirical skills in training

		the practical implementations of today. (v)Appreciate the need for empirical skills in training neural networks.	with (a) initializations (b) update strategies (c) regularisation (d) multi fold validation on a small/medium size deep neural network that can be trained in 5 minutes.

Detailed Contents for Desirable Learning Outcomes (optional, <= 3 modules):

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**
Key Concepts from ML	Kernels (with SVM), Bayesian Methods, Generative Methods, HMM, EM	Focus on mathematical and analytical skills. Also expose the intuition behind these algorithms	Python notebooks that demonstrate the use of these algorithms on public datasets
Deep Learning Architecture s	Popular CNN Architectures, RNNs, GANS and Generative Models,	Introduce popular architectures, models, and the use of it in various settings.	Use of popular architectures for pretrained features and transfer learning Use of RNNs in learning “language models” in large text corpus (charRNN) Capability and practical challenged in working with GANS
Training Todays Neural Networks	Advances in Backpropagation and Optimization for Neural Networks Adverserial Learning	Appreciate the challenges in large non-convex optimization and how many of todays design choices have helped.	See how (i) initialization (ii) momentum (iii) update rules have helped in getting better minima/soln. Experience how regularisation helps in avoiding overfitting and getting better solutions

Suggested text books / Online lectures or tutorials:

1. Ian Goodfellow, Yoshoua Bengio, and Aaron Courville Deep Learning MIT Press Ltd,
2. *Illustrated edition*
3. Christopher M. Bishop Pattern Recognition and Machine Learning - Springer, *2nd edition*
4. Tom M. Mitchell- Machine Learning - McGraw Hill Education, *International Edition*
5. Trevor Hastie, Robert Tibshirani, and Jerome Friedman - The Elements of
6. Statistical Learning: Data Mining, Inference, and Prediction - Springer, *2nd edition*
7. *Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong., , Mathematics for Machine Learning, Cambridge University Press.*

Prepared by: Mitesh Khapra, Vineeth N Balasubramanian, P. K. Biswas, Piyush Rai, Preethi Jyothi, Pabitra Mitra, Chetan Arora, P J Narayanan, C. V. Jawahar

Cyber Security

CSXXX	**Introductory Cyber Security**	**3L:0T: 4P**	**5 credits**	**Pre-Reqs: Introduction to Programming, Data Structures & Algorithms, Operating Systems, Networking**

Learning Outcomes of the course (i.e. statements on students' understanding and skills at the end of the course the student shall have):

Having completed this course, a student should be able to:

Essential (<=6):

1. Understand the importance of cyber security (data confidentiality, Integrity, and Availability) and various recent attacks on important digital systems such as banking, e-commerce systems, e-governance systems etc.
2. Understand basic cryptography concepts – symmetric vs asymmetric cryptography, Public Key Crypto Infrastructure (PKI), Symmetric Ciphers, Hashing, Digital Signatures
3. Understand methods and tools for authentication, authorization, privilege, and their needs in securing an organization's IT system
4. Understand the common vulnerabilities in applications, web applications, network, and the Internet Infrastructure
5. Understand the methods and tools for Intrusion Detection (network and host intrusion detection) and perimeter security (firewall)
6. Understand basic malware functions and indicators of compromise

Desirable/Advanced (<= 3):

1. Understand basic mobile application security issues and android platform architecture for securing app execution
2. Understand wireless LAN security issues

Detailed contents for Essential Learning Outcomes:

Module (appx dur in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Introduction and basic terminology (~1 wks)	Cyber Security and CIA Triad, basic cyber threats to CIA, cyber-attack surfaces, recent cyber-security incidents and their high-level analysis	Example Driven Lectures with examples drawn from most	None

		recent incidents	
Module 2: Basic Cryptography (1 - 2 wks)	Role of Cryptography in ensuring confidentiality for data at rest, data in motion, and data in process. Symmetric and Asymmetric Cryptography, their needs as complementary of each other, some basic symmetric and asymmetric algorithm outlines (RSA, DH, DES, AES) Role of cryptography in data integrity, non-repudiation Hashing and Digital Signature and some example hash function outlines (MD5, SHA-256), understanding digital signature and its role. Digital Certificate and PKI. Importance of the role of a proper Pseudo Random Number Generator	Provide good intuition than nitty gritty of algorithms, or going through ciphers in details. The intuition behind PKI, Digital Certificate.	Using library functions to use RSA, AES, SHA-256 and show the result of encryption, Hashing etc. Taking apart a digital certificate and show the various components and their significance
Module 3: Authentication, Authorization and Privilege (1 week)	Importance of strong Authentication, distinction between authorization and authorization, importance of authorization, access control, Mandatory and Discretionary Access control, role based authorization, privilege and privilege escalation	Intuition of distinguishing between authorization from authentication, access control lists, MAC vs DAC with examples, importance of distinct privileges, principle of least privilege, show example of privilege escalation	Lab on 2 factor authentication, Lab on privilege escalation example

Module 4: Application Security (4-5 weeks)	Basic application vulnerabilities (Buffer overflow, Integer Overflow, format string vulnerability) , Basic mitigations of buffer overflow – platform bases, compiler based, secure programming practice Web Client Security, Same Origin Principle, DOM, Java Script Vulnerability, Cookies and Cookie attributes Secure, httponly, Concept of session and session ID, Session hijacking vulnerability, http vs. https and SSL/TLS and version issue Web Server Security – XSS, CSRF, SQL Injection, Command Injection concepts, examples of each and mitigation techniques Vulnerabilities in DNS, Routing and IP protocols especially in IPv4 and suggested remedies with DNSSEC, S-BGP, and IPSec	Intuition of why these vulnerabilities happen, and how various mitigation techniques have been developed, why the mitigation techniques are not enough and can be escaped by determined attackers, and examples from real attacks	Lab1: Buffer overflow, integer overflow and format string vulnerability testing in vulnerable applications Lab 2: DVWA based command injection. SQL injection, XSS and CSRF
Module 5: Perimeter protection and Intrusion Detection (2 weeks)	Host Intrusion Detection techniques, what are the indicators to look for and how an SIEM tool can consolidate such indicators into a management console Network Intrusion Detection – signature based vs. behavior based, Snort Firewall vs. Intrusion Detection tool, Firewall rules and customization techniques	Intuition about indicators that could indicate a host as compromised, and how multiple hosts/endpoints can be monitored Intuition behind signature vs. behavior based network monitoring and detection of intrusion Hands on with Snort installation	Lab1: Students are asked to install Wazuh and monitor a host Lab 2: Students are asked to install snort and monitor a network on their local network

Module 6: Basic Malware Analysis (1 week)	Various malware classes and their characteristics Difference between static analysis and dynamic analysis Signature vs. behavioral detection techniques	Intuition about various malware classes and their modus operandi, discuss why static and dynamic analysis complement each other, what kind of information is obtainable from static vs. dynamic analysis, demonstrate some static analysis tools	Lab 1: US static analysis tools to find how an executable can be analyzed.

Detailed Contents for Desirable Learning Outcomes (optional, <= 3 modules):

Module	**Topics**	**Pedagogy teaching suggestions**	**Nature of lab / assignment / practice**
Module 7: Mobile Application Security (1 week)	Basic mobile attack surface and the ideas of permissions, and their abuse Execution model of mobile apps in Android (Sandboxing) and communication	Provide intuition on mobile malware and how they work, give example of mobile malware attacks, provide intuition of execution model of Android and demonstrate Mandatory Access Control	None

		idea in action, SE Linux being part of Android	
Module 8: WLAN Security (1 week)	Some common ways WLAN are compromised including weak cipher such as WEP, evil twin attack, unauthorized access point based attacks (rogue wlan) etc	Provide students idea about how to look for signs of these rogue wlans, evil twins, public wifi etc.	None

Suggested text books / Online lectures or tutorials:

1. Ross J. Anderson, Security Engineering, Third Edition, Wiley, Nov 2020

Suggested reference books / Online resources:

1. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws 2nd Edition by D Stuttard and M Pinto
2. Cryptography and Network Security by William Stallings
3. The Hacker Playbook: Practical Guide To Penetration Testing (vol. 1 and 2) by Peter Kim.

Prepared by: Vinod Ganapathy (IISc), Sambuddho Chakravarty (IIITD), S. Venkatesan (IIITA), Sandeep K. Shukla (IITK)

The Discipline Graduate Attributes (GAs) to which this course contributes significantly: CS1, CS3, CS6

Other discipline GAs to which this course may contribute somewhat:

Core Courses (Additional)

The two additional core courses, which are addional.

Compiler Construction

Prerequisite: C/C++/Java programming language. Data structures and algorithms. Automata theory.

Guiding Principles: The course should benefit four categories of students:
Students who will improve their programming skills significantly (via labs).
students who will write language translators in different contexts,
Students who will seek jobs in the compilers domain or computer systems area in general.
Students who will pursue research in the compilers domain or computer systems area in general.

Learning Outcomes, Essential (<=6):

1. To understand the role, functionality and structure of program translation and interpretation in software development.
2. To understand the difference between abstraction levels of a high level language and a machine language.
3. To understand the role of a sequence of intermediate representations in lowering the level of abstractions in the process of language translation.
4. To get a firsthand experience of a practical application of elegant data structures, algorithms, and other core CS concepts such as automata theory.
5. To make effective use of tools such as lex and yacc.
6. To become a much better programmer by appreciating all that happens behind the scenes in making an HLL program run.

Desirable/Advanced (<= 3):

1. To understand some of the critical aspects of machine code generation.
2. To understand the issues in efficient code generation.

Syllabus for Essential Learning Outcomes

Module (appx dur in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Nature of lab / assignment / practice**
Module 1: Introduction to Compilers (one week)	Comparing abstractions of a high level language and a low level language; compilation as a series of steps for lowering the abstraction level through stepwise refinement; phases of compilation; bootstrapping; cross-compilation.	Sections 1.1 and 1.2 from the textbook, online resources [2] and [3].	Viewing the intermediate representations and the final assembly code generated by GCC/LLVM, relating them to the input program Programming Assignment #A0.
Module 2: Lexical Analysis (two weeks)	The role of lexical analysis; Token, lexemes, and token codes; Regular Expressions (RE) to represent tokens, Deterministic finite	Sections 3.1, 3,2, 3.4.4, 3.5, 3.6, 3.8.1, 3.8.3	Writing lex specifications and generating tokens for a given language,

	automata (DFA),Traversing a DFA for recognising tokens; Generating a lexical analyzer using lex/Flex.	from the textbook.	Programming Assignment #A1.
Module 3: Syntax Analysis (three weeks)	Context Free Grammars (CFG), Concept of parsing, sentences and sentential forms, leftmost and rightmost derivations, parse trees, ambiguous grammars; Overview of top-down and bottom-up parsing; Option1: Introduction to shift reduce parsing; viable prefixes and valid items, Constructing LR(0) sets of items; Constructing SLR parsing tables; Generating a parser using a parser generator such as Yacc/Bison. Option 2:Top-down parsing,Left factoring, Elimination of left-recursion, predictive parsing, recursive descent parsing, LL(1) parsing. Generating a parser using a parser generator such as ANTLR, JavaCC, etc.	This module is driven by the chosen parser generator. If Yacc/Bison is chosen, the module should cover bottom up parsing. If Antlr or JavaCC is used, the module should cover top down parsing. Sections 4.1, 4.2, 4.3 from the textbook. Option1: Sections 4.5, 4.6 from the textbook. Option2: Section 4.4 from the textbook.	Writing yacc specifications, generating a parser by using the scanner generated in module 2. Precise error reporting using yytext and yylineno. Using a command line switch to optionally print token details. Programming assignment #A2.
Module 4: Semantic Analysis (one weeks)	The need of semantic analysis; abstract syntax trees for expressions, assignment statements and control flow statements; attribute evaluation, syntax directed translation schemes (STDS);	Sections 5.1, 5.2, 5.3, 5.4, and 6.1 from the textbook.	Writing a type checker to ensure that a syntactically correct MMC program is type-safe. Programming Assignment #A3.
Module 5: Applications of Semantic Analysis (three weeks)	Applications of SDTS for (a) declaration processing and type checking, (b) generating three-address code	Sections 6.2, 6.3, 6.4, 6.5.1, 6.5.2, 6.9 from the textbook.	Write a translator to translate a type-checked MMC program to equivalent three-address code. Programming Assignment #A4.
Module 6: Run time support (one week)	Parameter passing by value, reference, and name; activation records, stack and static allocation of activation records; translating a function call, allocating offsets to	Sections 1.6.6, 7.1, 7.2.2, 7.2.3 from the textbook, Chapter 6.1	Not Applicable

	variables, generating code for function prologue, function epilogue, call sequence, and return sequence.	from the reference book and online material [4].	
Module 7: Introduction to Code Optimization (One week)	Control flow graphs; Local optimizations (common subexpression, copy propagation, dead code elimination).	The focus of this module is on knowing the optimizations and not on the techniques of performing them. The desirable part would need an additional one week. Section 9.1 from the textbook..	Not Applicable
Module 8: Code Generation (two weeks)	Generating assembly code from three address codes using simple register allocation and instruction selection.	By simple register allocation, we mean that all the values of temporaries are held in registers across three address code statements but not those of source variables. Sections 8.1, 8.2, 8.6 in the textbook.	

Syllabus for Desirable/Advanced topics.

Module (appx dur in wks)	**Topics**	**Pedagogy / teaching suggestions**	**Nature of lab / assignment / practice**
Module 2: Lexical Analysis (two weeks)	4-arrays representation, observing the data structures in the scanner generated by lex.	Section 3.9.8 from the textbook.	
Module 3: Syntax Analysis (three weeks)	LR(1) and LALR(1) parsing and the option not chosen in the default offering	Section 4.7 from the textbook.	Error recovery using the error token in yacc and emitting meaningful error messages.

Module 7: Introduction to Code Optimization (One week)	Global optimization (constant propagation, common subexpression elimination, copy propagation, dead code elimination,strength reduction)	Sections 9.2.5, 9.2.6, 9.4.1, 9.4.2, 9.4.3, 9.4.6, in the textbook.	
Module 8: Code Generation (two weeks)	Register allocation using graph colouring, Optimal code generation for expression trees, Sethi Ullman algorithm, Aho Johnson algorithm.	Sections 8.8.1, 8.8.4, 8.10, 8.11 in the textbook.	Write a translator to translate code in 3-address-code form to assembly code. Programming Assignment #A5

Lab/Assignment Details

- A0: Write 5 simple test-cases in MMC and then inspect the generated code.
- A1: Write a lexer to recognize valid tokens.
- A2: Write a parser to parse the given input MMC program.
- A3: Write a type-checker for a syntactically correct input MMC program.
- A4: Write a Translator that takes a type-checked MMC program and generates equivalent IR code in TACoC format; TACoC is a subset of MMC such that code is in a form similar to three address code. Details of TACoC is given towards the end of this section.
- A5: Generate MIPS code. Use the SPIM simulator to run the code.

Suggested Pedagogy for the lab/assignments:

- The instructor is expected to prepare a set of test-cases for assignments A1-A5.
- For A1, A2, and A3 – the test cases should include some test-cases which throw lexical, syntactical and type errors, respectively.
- For input test-case design the expected output and write a script to verify the output generated by the student assignment.
- For each assignment, the student should use a script to build their code (using a Makefile/ant build script/shell script). The instructor must create some sample scripts and teach the same to the students.
- The instructor should create a reference implementation for A1-A5.
- The implementation can be done in Flex+Bison or Antlr or JavaCC.

Details of MMC

MMC is a simple variant of C, and has many simplifications. This along with the simplified nature of TACoC ensures that the compiler can be written within a short span of time. Further details about this can be obtained from: http://www.cse.iitm.ac.in/~krishna/aicte-compiler-design/Lab-Syllabus.docx, or from publicly available sources on MMC.

Texts and Other Material

- [Text book] Aho, Lam, Sethi, and Ullman. Compilers: Principles, Techniques, and Tools. 2/e, Addison-Wesley, 2006.
- [Reference book] Andrew Appel and Jens Palsberg. Modern Compiler Implementation in Java. 2/e, Cambridge University Press, 2002.

- https://en.wikipedia.org/wiki/Cross_compiler.
- https://en.wikipedia.org/wiki/Bootstrapping_(compilers)
- https://en.wikipedia.org/wiki/Function_prologue_and_epilogue

Theory of Computation

CSXXX	**Theory of Computation**	**?L:?T: ?P**	**??? credits**	**Pre-Reqs: ???**

Prerequisites
Familiarity with basic data structures and algorithm design
Must have done a Discrete Mathematics course

Essential Learning Objectives:
Understand models and abstractions: automata as a basic model of computation
Link between languages, automata, and decision problems.
How to build new models from old ones: product, union, closure properties.
Argue about limitations of computational models.
Understand algebraic formalisms of languages such as regular expressions, context-free grammar.
Understand algorithms and computability through the lens of Turing machines.
Existence of unsolvable problems and what that means.
Relations between the various computational models.

Desirable Learning Outcomes
None

Module (appx dur in wks)	**Topics**	**Teaching Suggestions**	**Learning outcomes**

Module 1: Finite Automaton (4-5 wks)	-Why automata theory? -Alphabets, formal languages, and problems. -What are regular languages and automata models for them: Deterministic Finite automaton, Formal argument of correctness, Regular languages -Properties of regular languages-Closure, properties, product construction -Limitations of Automata Non-regularity, Pumping Lemma -Non-deterministic Finite Automaton, Subset construction, Equivalence with DFAs. -Regular expressions. Equivalence with regular languages. -Algorithms for regular languages, Minimization and its algorithm. -(suggested) Myhill-Nerode relations, Characterization of regular languages	-Sec 1.1 of T2 -Sec 1.2, 1.5 of T2 -Sec 1.1 of T1 -Sec 1.1 of T1 -Sec 1.4 of T1 -Sec 1.2 of T1 -Sec 1.3 of T1 -Sec 4.3, 4.4 of T2 -Lecture 15,16 of R1 +applications of automata to text search and NLP +applications of regular expressions for text search in UNIX. **Advanced Topics:** - 2DFAs, Equivalence with DFAs using Myhill-Nerode Relations (Lecture 17,18 of R1)	F. Familiarity with notations. U. Give examples of languages, regular languages. U. Design finite automata, both deterministic and nondeterministic for a given language. R. Write formal proof of correctness of a DFA U. Give examples of non-regular languages and prove that language is non-regular using pumping lemma F. Understand the difference between determinism and nondeterminism U. Use closure properties to show non-regularity U. Design regular expressions U. Use the minimization algorithm to minimize a given DFA U. (suggested) Apply Myhill-Nerode Theorem to show that a language is regular or non-regular

Module 2: Grammars, Context-free Languages and machine models. (4-5 wks)	-Grammars and the motivation from language theory. -Context-free grammars, closure properties. Chomsky Normal Form for CFGs. -PDAs. Empty-stack vs Final state acceptance conditions. Equivalence of PDAs and CFGs. -Limitations of PDA computation, non context-free language. Pumping Lemma for CFLs. -Deterministic CFLs and PDAs. -(suggested) CYK Algorithm for parsing of CFLs.	-Sec 2.1 of T1 -Sec 2.1 of T1 -Sec 2.2 of T1 -Sec 2.3 of T1 -Sec 2.4 of T1 -Sec 7.4 of T2 +applications to parsers and compilers. **Advanced Topics:** - Ogden's Lemma.	U. Design CFGs and PDAs for CFLs R. Prove correctness of CFGs F. Understand that regular languages are a subset of CFLs. R. Prove equivalence of CFGs and PDAs U. Argue a language is non-CFL using pumping lemma F. Familiarity with DPDAs U.(suggested) Construction of DPDAs U. (suggested) Parsing using CYK algorithm

Module 3: *Turing machines and Computability*, (4-5 wks)	-Modeling computation using Turing Machines. Equivalent models. Church Turing Hypothesis. -Decidability and Turing recognizability (i.e., recursive and recursively enumerable). Closure properties. -Undecidability by diagonalization. -Reductions to show undecidability. Examples of reductions. -Resource bounded Turing machines & Intro to Complexity. Basic complexity classes. Time bounded classes: P, NP, EXP. -(suggested) Post's correspondence problem and other undecidable problems -(suggested) Polytime reductions, NP-completeness, Cook-Levin Theorem without proof	-Sec 3.1, 3.2, 3.3 of T1 -Sec 4.1 of T1 -Sec 4.2 of T1 -Sec 9.3 of T2. Sec 5.1, 5.3 of T1. -Sec 7.1 of T1 -Sec 5.2 of T1 -Sec 7.3, 7.4, 7.5 **Advanced Topics:** - Rice's Theorem - Space bounded computations and complexity, PSPACE	F. Understand relation between the various classes such as decidable, Turing recognizable., co-Turing recognizable. F. Give examples of decidable languages, undecidable languages, Turing recognizable languages. U. Prove a language is undecidable by reduction from a known undecidable problem F. Relation between basic complexity classes F. (suggested) Scenarios in which the reductions are used R. (suggested) Proving languages are NP-complete using reductions

Notations:

- Topic Categorization:
  Compulsory - Topics that should be covered.
  Suggested - Optional topics that the instructor can choose from given availability of time.
  Advanced - Advanced topics in each module that an instructor can teach depending on the interest of the class.

- + indicates applications that could be mentioned in the class.

- Learning Outcome Categorization:
  Familiarity - Student should be able to identify and comprehend *what* the topic is about. This corresponds to the cognitive levels of *knowledge* and *comprehension* of Bloom's taxonomy (see e.g., https://en.wikipedia.org/wiki/Bloom's_taxonomy).
  Usability - Student should be able to understand *how* a particular idea/topic can be used, to solve problems, design examples, etc. This corresponds to the cognitive levels of *application* and *synthesis* of Bloom's taxonomy.
  Reasoning - Student should have a deeper understanding of a particular concept and *why* it works. This corresponds to the cognitive levels of *analysis* and *synthesis* of Bloom's taxonomy.

Nature of lab / assignment / practice / tutorial:

1. Make assignments using the books. To test:
   - what was done in the class
   - whether the student can think and apply the concepts.

2. Tutorials: Weekly problem-solving sessions.

Suggested textbooks:

1. Introduction to the Theory of Computation, 3rd edition. Michael Sipser, Cengage Publications (Low-cost Indian edition available).
2. Introduction to Automata, Theory, Languages and Computation. Third Edition. John Hopcroft, Rajeev Motwani, Jeffrey D. Ullmann, Pearson Publications (Low-cost Indian edition available).

Additional Reference Material:

1. Automata and Computability, Dexter C. Kozen. Part of the Undergraduate Texts in Computer Science book series (UTCS), Springer.
2. Elements of the Theory of Computation, 2nd edition. Harry Lewis, Christos Papadimitriou, Prentice Hall.

Prepared by: Jayalal Sarma, IIT Madras. S Akshay, IIT Bombay. Raghunath Tewari, IIT Kanpur.

The Discipline Graduate Attributes (GAs) to which this course contributes significantly: CS4

Other discipline GAs to which this course may contribute somewhat: CS2

Micro Specializations (and Professional Electives)

Besides the core courses, programs normally have professional elective courses. Each HEI decides the electives it can or wishes to offer. These electives can be a set of disconnected professional courses. However, in the recent times, as needs for specializations is increasing, it is possible to use the electives to provide a limited specialization in some sub-area of CSE to a BTech student. We call these as micro-specializations. Micro specializations are a response to the need of having deeper knowledge in some sub-area. These also allow multiple pathways to students, as different students can graduate with different specializations (or not).

A micro-specialization is expected to be in a sub-area of CSE. The goal of the micro specialization is to provide deeper understanding and skill development in that area. The areas in which micro specialization are offered should be aligned to industry careers or higher studies.

A micro specialization for CSE is defined as follows:

- It has a core course as the head (starting) course for the micro specialization
- It has a clearly defined goal, and learning outcomes for the goal
- It can have 2 +/- 0.5 additional courses (besides the head course) in the sub-area aligned to the goal. These courses can be full course (4-credits) or half-course (2 credit), and can be taken as electives by students (or extra credits.)

Based on the discussions and inputs, a few desired micro-specializations were identified. For a sub-set of these, a possible design of the micro-specialization is provided below. It should be pointed out that the list of courses in a specialization is illustrative – an Institution can replace or add a course aligned to the micro specialization goal. Institutions can also define a set of courses for a micro specialization and require that a subset be taken, with perhaps one being compulsory.

It should be added that HEIs are completely free to decide whether to offer micro specializations or not, and if they decide to offer, which areas to provide the specialization in. How the micro specialization is to be reflected in a student's records/certificates is also to be decided entirely by HEIs based on their policies and practices.

Note that the list of additional courses mentioned in the micro specializations also provide a list of suggested professional electives. However, there can be electives which are not a part of any specialization.

Software Engineering Micro Specialization

Goal: The aim of this specialization is to provide an understanding of the importance and role of software engineering, an understanding of the important sub-areas of software engineering (SE), and to provide a deeper understanding of some of them.

Learning outcomes of the specialization:

- Understand the role and importance of software engineering in developing industrial strength software, and the important tasks involved in engineering such software.

- Ability to apply proper SE practices to develop in a team a working software system to solve some users' problem.
- Understand and apply concepts in some sub-areas of SE like testing, maintenance, open source software, model based development, requirements engineering, etc.

Base Course for this specialization (core course): Advance Programming

Courses (Electives) Proposed for the Specialization:

(The courses for a micro specialization can be full course (4-credits) or half-course (2 credit). The total number of courses suggested is 2 +/- 0.5.) For this specialization, it is suggested that the first course should be required, followed by 2 half courses in some sub-areas of SE.

Course Name (full / half)	Purpose/Goal	Topics
Software Engineering (full)	To explain the iterative software development process and different aspects of it (e.g. short cycles, agile approaches, test-driven-development, etc), and to apply the concepts to a team project to develop software	Iterative software development process, basic project planning for such a process, requirements including user interface, architecture and design, coding using modern IDEs, testing, integration and deployment
Software testing (half)	Go deeper in software testing, including some modern tools	Unit testing and the test frameworks, test case design (incl. black box and white box), non-functional testing, test automation and tools, related metrics (e.g. coverage, performance, e.g.)
Open Source Software (half)	To expose to students open source software practices and ecosystem, and to expose them to using them	OSS evolution, current situation, general OSS practices, common OSS platforms and how to use them, small project on some OSS platform (e.g. GitHub)
Software Maintenance (half)	To explain the software maintenance cycle and different aspects of it, and develop capabilities in them	Defect cycle and bug management, regression testing, change management, refactoring, software evolution, related metrics, etc.

Prepared by: Pankaj Jalote (IIIT-Delhi), Raghu Reddy (IIIT Hyderabad), Vinay Kulkarni (TCS), Meenakshi Dsouza (IIIT Bangalore)

Machine Learning Micro Specialization

Goal: The aim of this specialization is to provide an understanding of the importance of machine learning in computer science, expose the advances in machine learning, and also to demonstrate the practical role of machine learning in related domains in artificial intelligence. This specialization also aims at providing deeper understanding of some of the sub areas, deeper analytical concepts, and the recent advances.

Learning outcomes of the specialization:

- Appreciate the role of machine learning in computer science and data-driven problem solving.
- Appreciate the role of recent and advanced machine learning algorithms and formulations in solving problems of practical importance. Expose students to principled ways of practising machine learning in many unstructured real world problem settings.
- Ability to apply principles of ML to develop practical solutions to problems seen around in industry, society and research, focusing on problem formulation, solution design, implementation and experiencing the empirical design process.
- Providing practical experience in use of popular machine learning libraries and implementations. Get the students trained in the empirical science behind the design of machine learning based solutions.

Base Course for this specialization (core course):

- Machine Learning

Courses (Electives) Proposed for the Specialization:

(*Note: The courses for the specialization can be full course (4-credits) or half-course (2 credit). The total number of courses suggested should be 2 +/- 0.5.*)

Course Name (full / half)	**Purpose/Goal**	**Topics**
Advanced Machine Learning (full)	To introduce and be familiar with the popular deep learning architectures. To expose the popular problem formulations beyond simple supervised learning used in situations of data scarcity, domain/distribution shift etc. using the popular ML/DL algorithms.	CNNs, RNNs, Auto Encoders, Loss Functions and Training. Transfer learning, domain adaptation, semi-supervised learning, active learning, self-supervised learning, incremental learning, few-shot learning Laboratory experiments to appreciate the utility

Computer Vision or Natural Language Processing (Full)	To introduce a sub-area of Artificial Intelligence (perception) where machine learning formulations get extensively used. Appreciate how ML has influenced the problems in this area, and how the type of data/problems in this area demanded newer solutions in ML.	Formulations based on classification, regression, structured prediction in sub-tasks of CV/NLP. Role of data and learning in Feature/Embedding. Laboratory experiments to appreciate the utility Encourage Course Project
Programming for Machine Learning (half; Lab Course)	To expose students to popular ML libraries and frameworks such as PyTorch and Tensorflow). To develop empirical skills in design and implementation of ML solutions To strengthen the hands-on skill of the student.	Programming in popular libraries, Use of Cloud, GPUs, Debugging/Visualization. Working on large datasets. Programming intensive. Use of popular (in industry, research) frameworks and tools expected.

Notes:

1. To strengthen the micro specialization, students may be encouraged to do the BTech Project (or Internship or a summer project) in this area if the regulations (credit system) allow.
2. Second course may be adapted to the strength of the department as an “Applied Machine Learning Course '' in an area of expertise in the department. However, the focus is on the use of ML formulations in this course.

Prepared by: Prof. C. V. Jawahar (IIIT Hyderabad), Prof. P. J. Narayanan (IIIT Hyderabad)

Distributed and Cloud Systems Micro Specialization

Goal: The aim this specialization is to provide: an understanding of the importance and role of distributed computing and data systems, an understanding of the important sub-areas of these systems such as cloud computing and big data platforms, a deeper understanding of their concepts, and the practical aspects of building applications for such systems. The aim of this specialization is to introduce important topics and themes of distributed systems and cloud computing, both from a theoretical and a practical perspective.

Learning outcomes of the specialization:

- Understand the need for and the models of distributed systems, and ability to access remote applications and data over the network.
- Understand key concepts for building scalable, reliable and consistent distributed applications.
- Understand the design, implementation, deployment and use of distributed computing and data platforms
- Understand the models of Cloud computing, enabling technologies and how to build Cloud native applications

Base Course for this specialization (core course):

Computer Networks and/or Operating Systems

Courses (Electives) Proposed for the Specialization:

The courses for the specialization can be full course (4-credits) or half-course (2 credit). The total number of courses suggested should be 2 +/- 0.5.

Course Name (full / half)	**Purpose/Goal**	**Topics**
Principles of Distributed Systems (Half or Full)+	Understand the need for and the models of distributed systems, and ability to access remote applications and data over the network. Understand key concepts for building scalable, reliable and consistent distributed applications.	Need for distributed systems, models for coordination among multiple machines over the network. Models of distributed systems, remote invocation and remote storage. Concepts of performance, scalability, reliability, consistency and correctness.
Building Cloud and Big Data Applications (Full)	Understand the models of Cloud computing, enabling technologies and how to build Cloud native applications. Understand the design and use of distributed computing and data platforms.	Designing distributed applications/algorithms, Data-Intensive Computing and Data-Oriented Programming. Distributed execution and runtimes. Streaming data management and processing. Remote and Distributed file systems, Key Value Storage and NoSQL Columnar Store. Resource abstraction and Service oriented architecture, Computing Abstractions on the Cloud, and Building Cloud-native applications.

Cloud Systems Engineering (Half or Full)+	Understand the models of Cloud computing, enabling technologies and how to build Cloud native applications.	Cloud services models, Data center architecture and management. Virtualization and Containerization. Big data computing platforms, REST based web services.
Mini-Project/Hands-on Cloud Computing/Big Data (Half)*	Understand the design, implementation, deployment and use of cloud systems to help build distributed, scalable and reliable applications.	Hands-on mini programming project that makes use of the concepts and technologies learnt in the theory courses.

Notes:

- Either separate or as a lab-component of one of the other theory courses
- The course can be offered as half or full course depending on faculty availability

Pedagogical References

1. Cristina L. Abad, Eduardo Ortiz-Holguin, and Edwin F. Boza. 2021. Have We Reached Consensus? An Analysis of Distributed Systems Syllabi. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 1082–1088. DOI:https://doi.org/10.1145/3408877.3432409
2. Joshua Adams, Brian Hainey, Laurie White, Derek Foster, Narine Hall, Mark Hills, Sara Hooshangi, Karthik Kuber, Sajid Nazir, Majd Sakr, Lee Stott, and Carmen Taglienti. 2020. Cloud Computing Curriculum: Developing Exemplar Modules for General Course Inclusion. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR '20)*. Association for Computing Machinery, New York, NY, USA, 151–172. DOI:https://doi.org/10.1145/3437800.3439206 DOI:https://doi.org/10.1145/3304221.3325536

Prepared by: Yogesh Simmhan (IISc Bangalore), Purushottam Kulkarni (IIT Bombay)

Human Computer Interaction (HCI) Micro Specialization

Goal: The aim of this specialization is to provide an understanding of the importance and role of user-centered design and associated techniques.

Learning outcomes of the specialization:

- Understand the user-centered design process and associated techniques to use it for designing and developing computing based solutions
- Develop an ability to start with an observable problem in real-life settings and to develop a technology-led solution for the same.
- To develop soft skills such as empathy for societal problems, the ability to think from another perspective, learning to work in a group
- To learn methods related to observation, interviewing, problem identification, ideation, prototyping, and evaluation.
- To apply the design process in a hands-on scenario.

Base Course for this specialization (core course): Advanced Programming

Courses (Electives) Proposed for the Specialization:

Course Name (full / half)	**Purpose/Goal**	**Topics**
Introduction to HCI (full)	To provide an understanding of the importance and role of user-centered design and techniques like observation, interviewing, problem identification, ideation, prototyping, and evaluation.	Contextual Inquiry, Interviews, Surveys, Focus Groups, sketching, low-fidelity prototyping, usability evaluation
HCI semester project (half)	Give an opportunity to apply the concepts learnt in the theory course in a hands-on project	Students will work on a project from real-life-like settings should be chosen and a group of students should be guided through the design process
Interactive Systems (half)	To develop a new interaction technique to solve a specific problem in HCI	Using any of the modern-day technologies, like, Virtual Reality, Augmented Reality, Speech technologies, Mobile programming, Drones, Haptic systems, Eye-trackers, etc. students will develop and test new interaction modalities.

Notes:

- If there is faculty to offer further courses, then one/two other half courses can be added to this specialization – e.g. HCI for Development (HCI4D), Inclusive Design, Learning Technologies,...,

Appendix: Recommendations for Using Online Content in Courses at Colleges and Universities

Background

A lot of online high quality content is available today either free or at a low cost. Besides the government supported NPTEL, we have companies like Coursera and EdX who aggregate courses from several universities (and even private commercial organizations) and offer them to students all over the world. There are other companies like Unacademy who offer courses designed and developed by them. Further, a lot of companies have online content available to students. These include IBM, Microsoft, Amazon, Infosys, Google, Linkedin, Cisco, and so on.

On the other hand, most of the Computer Science (and related) departments face serious shortage of faculty, particularly in areas where there is a significant demand in the industry.

So, on one hand, we have quality content available for free or low cost, and on the other hand, we don't have faculty to teach such courses. The natural solution is to find ways to use online content for the courses in the curriculum (with credits). This way, either a knowledgeable faculty can "teach" a much larger class, or a faculty member with inadequate background in the topic can still "teach" the course better than what s/he would have done without the support of such online content.

Issues

The online content has been around for several years. The regulatory bodies have also been encouraging use of such content (particularly, NPTEL). And yet, the online content has not been integrated with the curriculum in most colleges. The pandemic has allowed people to take a fresh look at online content and the mental barriers to using such content in the curriculum have been breached. At this time, it is felt that a lot of colleges would want to use this content. However, there are two primary academic issues that need to be addressed (besides logistics, financial, and HR issues). These are:

- How do we decide what material to use. This has two sub-issues. One, what content would be equivalent to the content that is mentioned in our curriculum (course mapping). Note this is a challenge because there may not be a single module which covers all parts of the curriculum. So we may need to select more than one module. Two, given the plethora of content, which content is of reasonable quality.
- How would a college do evaluation of students in order to assign marks/grades.

Modes of Learning

There are several ways of using the online content in the curriculum.

The simplest mode (Mode 1) is to use online content as additional reference material. In this mode, the normal teaching is anyway being done and the faculty is referring to online content in the same way he/she would refer to a book. This mode requires no change or suggestions and hence is outside the scope of this document.

The next mode (Mode 2) is the flipped classroom model where the students go through the online content (including writing programs, small quizzes, etc.), and the local faculty takes discussion sessions and does all the evaluation. This mode still requires a knowledgeable faculty member to be the instructor. However, given that we can now reduce the contact hours for the students and faculty, the same faculty can handle a much larger class (or multiple sections in case there is an upper limit on the size of the class).

The next mode (Mode 3) is that the students are studying only through the online mode and there is very little interaction with the faculty at the college. May be there can be some sessions once in a while, but mostly, the role of a local faculty is only to handle evaluation (exams, assignments, projects, etc.). This partially addresses the issue of lack of faculty in certain areas since the expertise required for handling evaluation is arguably lesser than the expertise required to teach the course.

The last mode (Mode 4) is where the online provider does everything, including evaluations. We don’t need any faculty member at the college to offer this course. We only need to decide how to translate the evaluation done by an outsider to an equivalent grade/marks on the college transcript.

Course Mapping

Each course in the curriculum has course outcomes and the content defined by the university. Typically, we may have some course outcomes which are important ones, and may be some outcome which is desirable or optional. Similarly, the curriculum may also define the rough duration of each topic. Sometimes the curriculum will also include the kind of projects or assignments that the student shall be asked to do in the course.

The key to course mapping is to realize that any two individuals defining a course will have some differences and we must be flexible to accept differences to some extent. The committee trying to do course mapping should have an understanding of what are the important components (in all three: outcomes, topics, and projects) and should ensure that the online content meets all the important requirements. But looking for identical course would be usually futile.

Also, what is noticed is that typical online courses are often available in smaller modules than a typical 4-credit course in our curriculum. And therefore, one may need to consider more than one online courses together to be equivalent to one course in the college. It may also be noted that just like some small aspects of the course may not be present in the modules chosen, there may be some aspects of the online modules which were not part of the college course. This amount of flexibility should be acceptable to the college. Typically, if the online content covers 80% of the college course, it may be accepted.

While autonomous colleges and universities do have this flexibility, the affiliated colleges may not have the flexibility of not teaching even 20% of the content. In case of affiliated colleges, one will have to either be stricter in course mapping, or find a way to cover the gaps through a local faculty or a visiting faculty. It is assumed that it is easier to find a visiting faculty for a small part of the course and hence it is still a useful mode.

It may also be noted here that for Core courses, the overlap needs to be significantly higher while for the elective courses, the overlap could be relaxed somewhat. It is because the core courses typically are pre-requisites for other courses. Also, core courses have been defined to be such because it is assumed that that content is more important for the graduates than what is taught in electives.

However, course mapping is still not an easy thing to do. It requires an understanding of important versus less important components, and quality of content among the plethora of options available. It may require someone to go through the content patiently.

And hence it is recommended that for standard courses recommended in the AICTE model curriculum, a course mapping may be suggested for the benefit of colleges and universities. A few example course mappings are, therefore, attached with this report.

Evaluation

In Modes 1, 2, and 3, the complete evaluation is local, and hence there is no issue. In Mode 4, we need to consider an external evaluation and use that internally. This is a challenge. There is a difference in how to handle this in a university versus an affiliating college. In a university, a simple way of handling this would be to assign a Pass/Fail grade to the student. In case of an affiliating college, where only the internal marks need to be forwarded to the university, one could consider the external evaluation since it impacts only 30% of the marks, and the university will anyway have its own exam of 70% marks.

There are other models like normalizing the external evaluation to the college's internal policy or average distribution of marks in other courses.

It is assumed that in Mode 4, there is a formal way of communicating performance of the student by the provider of the online courses since colleges and universities will not accept screen shots, emails, etc.

Other Issues

Financial: If there is a fee to be paid for online content, the college should have a policy on that. Typically, if the savings due to reduced faculty requirements are significant, then the fees for the online provider may be reimbursed by the college.

Teaching Load: Faculty member supporting the course whether by taking a few discussion sessions (in Mode 2) or by evaluating the students (in Mode 3) is still putting in substantial effort in managing the course, and an appropriate credit should be given to the faculty member when his/her teaching load is computed.

Training the teacher: When a course is being done in Mode 3, the local faculty member should also be expected to register for the course and go through the course (with load being appropriately counted). After a faculty member has gone through the course in two academic sessions, s/he would be well prepared to teach the course in the class in a much better way. Even if the course is being offered in Mode 4, there is no harm in asking a faculty member to register for the course and go through it. Some responsible person in the college would know the level of the course and what exactly students have done, and again, after two such sessions, the faculty member would be well prepared to teach the course. Hence this mode will also lead to better training of the teachers.

Limits on Credits: The committee believes that there should be a limit on the number of credits students can earn through online courses. In case of Mode 4, where even the evaluation is done by the online course provider, the proposed limit is 8 credits only. It is felt by the committee that the evaluation by online providers is still not fully trustworthy. As the technology for online exams or the processes for evaluation by online providers improve over a period of time, this limit may be increased. In case of Mode 3 where the content is delivered online but evaluation is local, the limit can be high. For Mode 3 and Mode 4 combined, the limit can be what the regulatory bodies like UGC have announced for online courses, which is currently 40% of the total credits. In Modes 1 and 2 where the online content is really the reference material, there is no need for any limit.

Another constraint the committee would want the colleges to consider is that in a sequence of courses in one stream of Computer Science, at least one course should be in class. For example, if we consider the sequence of systems courses – Operating Systems, Databases, Networks, Architecture, at least one course should be in class. This is to ensure that if there were some gaps in online courses, the faculty in the face to face class can try to cover that to some extent.

Faculty Incentive: There is a need to provide some incentive to faculty members who would manage the course that is being taught in the online mode. If a course is being taught in Mode 2, the load on the faculty is only marginally less than the load of teaching an in person course. So the full teaching load should be considered for the faculty. In Mode 3, the load is much less, and in Mode 4, the load is only that we are asking the faculty to also go through the course along with the students. In these two modes, the college may consider this as reduced load. However, their learning the course may be treated as equivalent to having done a Faculty Development Program when it comes to their appraisal and promotions.

Consideration in NBA Accreditation: One of the prime reason why online courses haven't become popular with colleges is that they must recruit faculty with a certain faculty to student ratio for accreditation and ranking. And once they have recruited faculty, one would always want the faculty to teach and not keep them under-loaded. If one can consider online courses as equivalent to faculty strength while deciding faculty-to-student ratio, then colleges would be attracted to online courses. A typical faculty member teaches about 100 students in a semester (across 2-3 courses). If 100 students do a course in Mode 4, we may consider this as equivalent to having one additional full time equivalent (FTE) faculty member on the rolls of the college for that semester. Similarly, if 200 students do a course in Mode 3, we may consider this as equivalent to having one additional FTE faculty member for that semester.

Committee and Area Experts

Experts appointed by AICTE:

- Pankaj Jalote, Distinguished Professor and founding Director, IIIT-Delhi (Chair)
- Manoj Singh Gaur, Director IIT Jammu
- Nutan Limaye, IIT Bombay
- Ramkumar, Pro Vice-Chancellor at Krea University
- Dheeraj Sanghi, Vice-Chancellor, JK Lakshmipat University, Jaipur
- Amit Aggarwal, NASSCOM

Other Experts in the Committee:

- Kishore Kothapalli, Professor, IIIT Hyderabad
- Sudeshna Sarkar, Professor, IIT Karaghpur
- Sukumar Nandi, Professor, IIT Gauhati
- Suchismita Roy, Professsor, NIT Durgapur
- Ashalatha Nayak, Professor, Manipal Institute of Technology
- RBV Subramanyam, Professor, NIT Warangal
- Sanjiva Prasad, Professor, IIT Delhi
- Venkatesh R, TCS Pune
- Viraj Kumar, ACM India Education Committee
- Vishram Thatte, Amazon India
- Vinnie Jauhari, Microsoft India
- R Latha, IBM India
- Gaurav Aggarwal, Google India
- Vinayaka Ram Gururajan, TCS
- Thirumala and Sundar K S, Infosys
- P.B. Kotur, Wipro
- Ishvinder Singh, Cisco Systems, Inc.
- Rahul Suresh Ghali, Accenture

Expert Committees for Different Courses

Course	Experts in the Committee
Data Structures	Madhavan Mukund (CMI), Manindra Agrawal (IIT Kanpur), Naveen Garg (IIT Delhi), Amit Kumar (IIT Delhi), Venkatesh Raman (IMSC)
Discrete Mathematics	Nitin Saxena (IIT Kanpur), Somenath Biswas (IIT Goa), Partha Mukhopadhyay (CMI), Bhabani P Sinha (ISI Calcutta)
Algorithm Design and Analysis	Madhavan Mukund (CMI), Manindra Agrawal (IIT Kanpur), Naveen Garg (IIT Delhi), Amit Kumar (IIT Delhi), Venkatesh Raman (IMSC)
Computer Architecture/Organization	M. Balakrishnan (IIT Delhi) , John Jose (IIT Gauwhati), Biswadip Panda (IIT Bombay), Anupam Basu (IIT Kgp), Indranil Sengupta (IIT Kgp), Anshul Kumar (IIT Delhi)
Advanced Programming	Kishore Kothapalli (IIIT Hyderabad), Vivek Kumar (IIIT Delhi), Swarnendu Biswas (IIT Kanpur)
Operating Systems	Purushottam (Puru) Kulkarni (IIT Bombay), Debadatta Mishra (IIT Kanpur), Chester Rebeiro (IIT Madras)
Databases	Arnab Bhattacharya (IIT Kanput), Sreenivasa Kumar (IIT Madras), Vikram Goel (IIIT-Delhi), S Sudarshan (IIT Bombay)
Computer Networks	BN Jain (IIIT Delhi), Timothy Gonsalves (IIT Mandi), Vinay Ribero (IITB), Mythili Vutukury (IIT Bombay), PM Sreelakshmi (IIT Mandi)
Machine Learning	CJ Jawahar (IIIT Hyderabad), PJ Narayanan (IIIT Hyderabad), Vineeth Balasubramanian (IIT Hyderabad), P. K. Biswas (IIT Kgp), Mitesh Khapra (IIT Madras), Piyush Rai (IIT Kanpur), Preethi Jyothi (IIT Bombay), Pabitra Mitra (IIT Kgp), Chetan Arora (IIT Delhi)

Security	Sandeep Shukla (IIT Kanpur), Sambuddho Chakravarty (IIIT Delhi), S.Venkatesan (IIIT Allahabad), Vinod Ganapathy (IISc)
Theory of Computing	Raghunath Tewari (IIT Kanpur), Jayalal Sarma (IIT Madras), S Akshay (IIT Bombay)
Compilers	Uday P. Khedker (IIT Bombay), V. Krishna Nandivada (IIT Madras), Dibyapran Sanyal (nvidia)

Expert Committees for Micro Specializations

To be added