Name of the Course: Computer Science Practical 3

Sr. No.	Heading	Particulars
1	**Description the course:**	**Introduction:** This course is a practical extension of theoretical concepts covered in Operating Systems and Data Structures. It introduces students to essential programming constructs required for system-level programming and structured data organization. Through carefully curated lab exercises, students gain insights into concurrency, resource management, scheduling, and structured data processing. **Relevance:** Operating Systems and Data Structures form the foundation of all computing systems and applications. Understanding how data is stored, accessed, manipulated, and how system resources are scheduled and synchronized is vital for every computer science graduate. This practical course equips students with essential skills that are applicable across all domains of software development and system programming. **Usefulness:** This course builds a strong foundation in system-level programming and abstract data handling, essential for any computer science graduate. It enhances the ability to understand and implement core concepts like scheduling, synchronization, and data abstraction through real-time coding tasks. Students gain valuable skills for debugging, memory management, and efficient algorithm design. These skills are crucial for both academic excellence and industry readiness. **Application:** The practical skills developed in this course are directly applicable in the design of operating system modules, file systems, and memory management tools. Data structures like trees, heaps, and graphs are widely used in building compilers, databases, and networking software. The understanding of multi-threading and process synchronization enables students to contribute to applications requiring concurrency, such as games, real-time simulations, and cloud systems. These applications form the backbone of modern software development. **Interest:** Students often find this course engaging due to its interactive and logic-based approach to solving real-world

		problems. Concepts like circular queues for task scheduling or graphs for social network analysis spark curiosity and hands-on involvement. The implementation of real-time synchronization and visual data traversals makes the learning process both stimulating and rewarding. This course encourages creative problem-solving through code. **Connection with Other Courses:** This practical course closely aligns with theoretical subjects like Operating Systems, Data Structures and Algorithms, and Object-Oriented Programming. It also forms the basis for advanced topics such as Distributed Systems, Artificial Intelligence, and Systems Programming. Concepts learned here are frequently reused and expanded upon in courses like Database Systems, Compiler Design, and Software Engineering. Hence, it serves as a vital link across the curriculum. **Demand in the Industry:** Industry consistently seeks professionals proficient in data structures and operating system fundamentals. Skills like process synchronization, memory optimization, and graph traversal are essential for roles in software development, cloud computing, and backend engineering. These competencies are tested during technical interviews at leading firms, including product-based and service-based companies. The ability to translate theoretical knowledge into efficient code is highly valued across tech domains. **Job Prospects:** Completing this course prepares students for roles such as Software Developer, System Programmer, Backend Engineer, or Technical Consultant. It opens opportunities in IT services, product development, and tech startups. With further specialization, students can also pursue careers in AI/ML, cybersecurity, and data engineering. The practical exposure provided here serves as a strong launchpad for core technical positions in the industry.
2	**Vertical:**	Major
3	**Type:**	Practical
4	**Credits:**	2 credits (1 credit = 30 Hours of Practical work in a semester)
5	**Hours Allotted:**	60 hours
6	**Marks Allotted:**	50 Marks

7	**Course Objectives(CO):** **CO 1.** To develop hands-on skills in implementing core concepts of operating systems and data structures. **CO 2.** To simulate and solve real-world problems using process management, synchronization, and memory management. **CO 3.** To strengthen students' understanding of data abstraction and manipulation using linked structures, trees, graphs, and hashing. **CO 4.** To enable students to analyze and compare algorithmic strategies for CPU scheduling, buffer control, and structured data operations. **CO 5.** To foster problem-solving abilities through coding, debugging, and testing of system-level and data structure-oriented programs.
8	**Course Outcomes (OC):** After successful completion of this course, students would be able to - **OC 1.** Design and implement solutions using inter-process communication techniques such as shared memory and message passing. **OC 2.** Apply multithreading, synchronization mechanisms, and scheduling algorithms to solve operating system-related problems. **OC 3.** Construct and manipulate linear and non-linear data structures using custom implementations. **OC 4.** Demonstrate effective use of stack, queue, trees, graphs, and hash tables in algorithm development. **OC 5.** Analyze and evaluate the performance of memory and disk management techniques and abstract data operations. **OC 6.** Apply practical programming knowledge to develop efficient, real-time, and scalable system-level applications.
9	**Modules:-** **Module 1 (30 hours):** **Practical based on Principles of Operating Systems** **Process Communication using Shared Memory** • Understand shared memory concepts in inter-process communication. • Implement producer-consumer synchronization using shared memory and semaphores. • Explore issues of race conditions and how to avoid them. **Process Communication using Message Passing** • Use message queues/pipes to solve the producer-consumer problem. • Compare and contrast shared memory vs. message-passing approaches. • Analyze blocking vs. non-blocking communication. **Threading and Single Thread Control Flow** • Practice thread creation and basic thread lifecycle using standard libraries (e.g., pthreads or Java threads).

	• Observe execution order, thread joining, and delays. • Measure execution time for sequential vs threaded execution. **Multi-threading and Fibonacci Generation** • Implement multi-threading to generate and print Fibonacci sequences. • Explore thread safety, synchronization when accessing shared variables. • Introduce concepts of thread pooling and task delegation. **Process Synchronization and Bounded Buffer Problem** • Simulate producer-consumer bounded buffer using mutex and semaphores. • Implement buffer control with synchronized access. • Introduce circular queue techniques for managing shared buffers. **Readers-Writers Problem – Synchronization in Shared Access** • Implement reader and writer prioritization. • Use semaphores to allow multiple readers or exclusive writer access. • Extend to fairness in access and deadlock prevention. **CPU Scheduling Algorithms (Part 1) – FCFS and Non-preemptive Scheduling** • Simulate First-Come First-Serve scheduling. • Extend implementation to general non-preemptive scheduling. • Analyze waiting time, turnaround time, and Gantt chart generation. **CPU Scheduling Algorithms (Part 2) – Round Robin** • Implement Round Robin scheduling with configurable time quantum. • Compare with FCFS: fairness, turnaround, response time. • Track context switches and improve queue management. **Memory Management Techniques** • Simulate FIFO and LRU page replacement using page reference strings. • Measure hit/miss ratios under different reference patterns. • Extend to include frames and memory constraints. **Disk Scheduling and Simple File System Design** • Simulate FCFS, SSTF, C-SCAN, C-LOOK, RSS for disk head movement. • Design a basic file system structure with block allocation, directory management, and file operations (create, read, delete).
	Module 2 (30 hours):
	Practical based on Data Structures
	Exploring Abstract Data Types (ADT) & Custom Structures • Create and manipulate structures to model ADTs like Student, Book, or

	Employee. • Implement basic operations (create, update, delete) using structures. • Reflect on differences between primitive and abstract data types. **Building and Using Singly Linked Lists** • Construct a dynamic singly linked list with basic operations. • Apply linked lists to simulate scenarios such as managing a playlist or to-do list. • Compare static (array) vs dynamic (linked) approaches. **Polynomial Operations Using Linked Lists** • Represent polynomials using linked lists. • Perform polynomial addition and subtraction by merging lists. • Use structured representation to reinforce node manipulation. **Working with Doubly Linked Lists** • Create a doubly linked list with forward and backward traversal. • Implement insertion/deletion at head, tail, and specific positions. • Use in scenarios like browser history or undo-redo features. **Implementing and Using Stack ADT** • Implement push, pop, peek using arrays or linked lists. • Solve problems like delimiter matching or undo mechanism. • Convert expressions from prefix to postfix and evaluate them. **Understanding Queues and Circular Queues** • Develop linear and circular queues to simulate task scheduling. • Perform enqueue and dequeue with wrap-around logic. • Discuss memory utilization in linear vs circular queues. **Tree Traversals and Binary Search Trees** • Create a binary search tree (BST) from a dataset. • Perform and visualize in-order, pre-order, and post-order traversals. • Use traversal results to derive sorted sequences. **Balanced Trees & Priority Queues** • Insert values and observe AVL tree rebalancing. • Construct min-heaps or max-heaps and simulate priority queues. • Use priority queues to manage task priorities (e.g., patient triage, job

	scheduling). **Graph Representations and Traversals** • Represent graphs using adjacency matrices and lists. • Implement BFS and DFS to explore graph components. • Use graphs for mapping routes or exploring social networks. **Hashing Concepts and Collision Handling** • Implement a hash table with chaining or linear probing. • Simulate insertion, deletion, and search with collisions. • Discuss practical hashing applications (e.g., dictionary lookup, indexing).	
10	**Text Books** 1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2022). Operating system concepts (10th ed.). Wiley. 2. Aho, A. V., Ullman, J. D., & Lam, M. S. (2021). Data Structures and Algorithms in Python (Adapted by R. Rao). Pearson India.	
11	**Reference Books** 1. Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Java (6th ed.). Wiley India. 2. Kanetkar, Y. (2020). Data Structures Through Python (1st ed.). BPB Publications.	
12	**Internal Continuous Assessment: 40%**	**Semester End Examination: 60%**