UNIVERSITY OF GLOBAL VILLAGE

DATA STRUCTURE

COURSE PLAN

2nd Semester

Department of CSE

Instructor

Galib Jaman

Lab Instructor

Department of CSE

University of Global Village

DATA STRUCTURES

Course Learning Outcomes (CLO)

1. **Understand Fundamental Data Structures:** Demonstrate an understanding of the fundamental concepts of *data structures*, including *arrays, linked lists, stacks, queues, and trees.*

2. **Apply Recursive Techniques**: Design and implement basic *data structures* in *C* to solve computational problems.

3. **Analyze Algorithm Efficiency**: Analyze the efficiency of algorithms in terms of *time and space complexity* for operations on *data structures.*

4. **Implement Advanced Structures**: Design, implement, and optimize advanced data structures, including binary search trees, heaps, and hash tables, for real-world applications.

5. **Utilize Graph Algorithms**: Employ basic and advanced *graph* algorithms to solve problems involving connectivity, traversal, and optimization in graphs.

6. **Implement Sorting Algorithms**: Apply various sorting algorithms to organize and process data efficiently in different scenarios.

7. **Debugging & Memory Management:** Employ dynamic memory management techniques to optimize memory usage and enhance the performance of applications.

8. **Solve Real World Problems**: Work collaboratively on projects or assignments that involve implementing and using data structures.

CLO MAPPING OF

17-Class Course Plan

WEEK	TOPIC	ASSESSMENT STRATEGY	CLO
01	Introduction To Data Structures	Practice Quiz	01
02	Working With Arrays	Practice Review	01
03	Understanding Linked Lists	Practice Quiz	01
04	Working With Stacks	Practice Assignment	01
05	Working With Queues	Practice Quiz	01
06	Recursion And Backtracking	Review Assignment	02
07	Basics Of Trees	Practice Review	01
08	Binary Search Trees (BSTs)	Practice Assignment	04
09	Heap Data Structure	Practice Review	04
10	Basics Of Graph	Practice Group Work	05
11	Understanding Hash Tables	Practice Group Work	04
12	Complexity Analysis	Quiz	03
13	Overview of sorting algorithms	Quiz Practice	06
14	Dynamic Memory Management	Quiz	07
15	Advanced Graph Algorithms	Practice Review	05
16	Debugging and Optimization	Quiz Group Work	07
17	Final Project and Review	Assignment Group Work	08

WEEK-01 | CLO-01

Introduction To Data Structures

Outcome:

- Understand the need for data structures and their applications in problem-solving.

Discussion Topics:

- ☐ Definition and importance of data structures.
- ☐ Types of data structures: Linear and Non-Linear.
- ☐ Real-world examples of data structures.
- ☐ Introduction to C programming language.
- ☐ IDE installation and writing basic programs.

Questions:

1. Explain the difference between linear and non-linear data structures with examples.
2. Explain the types of data structures.
3. Draw a flowchart showing how data is stored and retrieved in a hierarchical structure.

Lab Practice:

1. Install C compiler and your favourite IDE in your machine.
2. Write a program to print “Hello, World!”
3. Write a program that takes a name as input and shows a greeting message (e.g: Hello, John Doe)

WEEK-02 | CLO-01

Working With Arrays

Outcome:

- Demonstrate the ability to use arrays for data storage and manipulation.
- Understand the advantages and disadvantages of array data structure.

Discussion Topics:

- [ ] Definition and properties of arrays.
- [ ] Static vs. dynamic arrays.
- [ ] Control flow statements: Conditionals and Loops
- [ ] Array traversal and data insertion using loop.
- [ ] Multi-dimensional arrays and their use cases.

Questions:

1. What is an array, and how is data stored in an array?
2. Given the starting memory address and the data type calculate the memory address of the n^{th} element of an array.
3. Explain why reading from an array is more efficient than insertion or deletion.

Lab Practice:

1. Write a program to declare a static array of size 10, input 10 integers from the user and print the elements.
2. Identify patterns in various shapes and write programs to draw those shapes (e.g: Rectangles, Right triangle and Pyramid).
3. Write a program to traverse an array of integers and print all even numbers along with their indices.
4. Write a program to reverse the position of the elements of an array.
5. Write a program to find the maximum and minimum values from an array.
6. Declare a multidimensional array and print each element using nested for loops.

WEEK-03 | CLO-01

Understanding Linked Lists

Outcome:

- Demonstrate the ability to use arrays for data storage and manipulation.
- Understand the advantages and disadvantages of array data structure.

Discussion Topics:

- ☐ Definition and properties of linked list.
- ☐ Singly, doubly and circular linked list.
- ☐ Node Structure in C

Questions:

1. What is a linked list, and how the data are stored in a linked list?
2. Explain the advantages and disadvantages of linked lists.
3. Explain the difference between a singly linked list and a doubly linked list.
4. Why are linked lists more suitable for dynamic memory allocation than arrays?
5. What happens when you attempt to delete the last node in a singly linked list?

Lab Practice:

1. Write a program to define a node structure for a singly linked list and initialize the head node.
2. Implement a function to add a new node at the end of a singly linked list.
3. Write a program to traverse a linked list and print all its elements.
4. Implement a function to delete a node at a specific position in a linked list.
5. Create a program to reverse a singly linked list.

WEEK-04 | CLO-01

Working With Stacks

Outcome:

- Understand the concept of stacks and their role in data management.
- Implement basic stack operations and explore their applications in programming.

Discussion Topics:

- [ ] Definition and properties of stacks (LIFO principle).
- [ ] Basic stack operations like Push, Pop, Peek, and IsEmpty.
- [ ] Implementing stacks using arrays and linked lists.
- [ ] Common applications of stacks: Expression evaluation, Backtracking, and Undo/Redo functionality.
- [ ] Limitations of stacks such as dynamic resizing and techniques for overcoming them.

Questions:

1. What is a stack, and how does the LIFO principle work in it?
2. How would you implement a stack using an array? What are its limitations?
3. What is the difference between a static and dynamic stack implementation?
4. Explain a real-world example where stacks are used.
5. How can you handle a "stack overflow" error in programming?

Lab Practice:

1. Write a program to implement a stack using an array with push, pop, and peek operations.
2. Create a stack program to check if a given string has balanced parentheses (e.g., "()", "{}", "[]").
3. Implement a function to reverse a string using a stack.
4. Write a program to simulate an "Undo" feature using a stack.

WEEK-05 | CLO-01

Working With Queues

Outcome:

- Understand the concept of queues and their role in sequential data processing.
- Implement basic queue operations and explore their real-life applications.

Discussion Topics:

- [ ] Definition and properties of queues and the FIFO principle.
- [ ] Simple, Circular, Double-ended, and Priority queues.
- [ ] Basic operations such Enqueue, Dequeue, Peek, and IsEmpty.
- [ ] Real life usage of queue.
- [ ] Advantages and Disadvantages of queue as a data structure.

Questions:

1. What is a queue, and how does it differ from a stack in terms of structure and functionality?
2. How would you implement a simple queue using an array?
3. Explain the purpose of a circular queue and its advantages over a simple queue.
4. What is a priority queue, and how does it differ from other types of queues?

Lab Practice:

1. Write a program to implement a queue using an array with enqueue and dequeue operations.
2. Create a circular queue implementation and demonstrate its advantages over a simple queue.
3. Write a program to simulate a ticket counter system using a queue, where customers are served in order of arrival.

WEEK-06 | CLO-02

Recursion and Backtracking

Outcome:

- Understand the concept of recursion and how it simplifies problem-solving through repeated self-reference.
- Apply backtracking techniques to solve complex problems systematically.

Discussion Topics:

- [ ] Definition and working of recursion (base case and recursive case).
- [ ] Recursive vs. iterative approaches: Pros and cons.
- [ ] Stack memory usage in recursion and common pitfalls (e.g., stack overflow).
- [ ] Introduction to backtracking: Definition, process, and key components (decision tree).

Questions:

1. What is recursion, and how does it differ from a loop?
2. Explain the role of the base case in a recursive function.
3. What is backtracking, and how does it differ from brute force methods?
4. How does recursion utilize stack memory for function calls?
5. Provide a real-world example where backtracking can be used effectively.

Lab Practice:

1. Write a program to find the factorial of a number using recursion.
2. Implement a recursive function to calculate the nth Fibonacci number.
3. Write a recursive program to reverse a string.
4. Create a program using recursion to solve the Tower of Hanoi problem for n disks.
5. Implement a backtracking solution to solve the N-Queens problem.
6. Write a program to find all possible paths in a maze using backtracking (maze represented as a 2D array).
7. Create a program to solve a Sudoku puzzle using backtracking.

WEEK-07 | CLO-01

Basics of Trees

Outcome:

- Understand the fundamental concepts and terminology related to trees.
- Learn the structure, types, and basic operations of trees.

Discussion Topics:

- [ ] Definition of trees and tree-related terminology such as nodes, root, leaves, height, depth, etc.
- [ ] Types of trees: Binary Trees, Binary Search Trees (BST), and N-ary Trees.
- [ ] Tree traversal methods like Inorder, Preorder, and Postorder.
- [ ] Differences between linear data structures and hierarchical structures.
- [ ] Common applications of trees in file systems and in machine learning.

Questions:

1. What is a tree, and how is it different from other data structures like arrays and linked lists?
2. How does the height of a tree impact its performance?
3. Describe the process of Inorder traversal and provide a use case for it.
4. What are some real-world scenarios where trees are used as a data structure?

Lab Practice:

1. Write a program to define a node structure for a binary tree and initialize the root node.
2. Implement functions for Inorder, Preorder, and Postorder traversals of a binary tree.
3. Write a program to find the height of a binary tree.
4. Implement a function to count the total number of nodes in a binary tree.
5. Create a program to check if two binary trees are identical.

WEEK-08 | CLO-04

Binary Search Trees (BSTs)

Outcome:

- Understand the structure and properties of Binary Search Trees (BSTs).
- Learn how to implement and perform basic operations on BSTs.

Discussion Topics:

- ☐ Definition and properties of a Binary Search Tree (BST).
- ☐ How BSTs maintain sorted order for efficient searching.
- ☐ Basic operations: Insertion, Deletion, and Search in BSTs.
- ☐ Traversal techniques in BSTs: Inorder, Preorder, Postorder (and their significance).
- ☐ Applications of BSTs: Searching, Sorting, and Range Queries.

Questions:

1. What is a Binary Search Tree, and how is it different from a regular binary tree?
2. How does the structure of a BST help in efficient searching?
3. Explain the process of inserting a node into a BST.
4. What happens when you delete a node with two children in a BST?
5. Provide real-world scenarios where BSTs are used.

Lab Practice:

1. Write a program to create a Binary Search Tree (BST) and insert nodes into it.
2. Implement a function to search for a specific key in a BST.
3. Write a program to perform Inorder, Preorder, and Postorder traversals on a BST.
4. Implement a function to find the minimum and maximum values in a BST.
5. Write a program to delete a node in a BST and handle all three cases (leaf node, one child, two children).
6. Create a program to check if a given tree is a valid BST.
7. Implement a function to find the kth smallest element in a BST.

WEEK-09 | CLO-04

Heap Data Structure

Outcome:

- Understand the structure and properties of Max-Heap and Min-Heap.
- Learn how to implement heaps and use them for efficient priority management.

Discussion Topics:

- [ ] Definition and properties of a heap.
- [ ] Types of heaps: Max-Heap and Min-Heap.
- [ ] Operations on heaps: Insertion, Deletion, and Heapify.
- [ ] Building a heap from an array (Heap Construction).
- [ ] Applications of heaps: Priority Queues, Heap Sort, and Median Finding.

Questions:

1. What is a heap, and how does it differ from a Binary Search Tree (BST)?
2. Explain the difference between a Max-Heap and a Min-Heap.
3. How is the "Heapify" process used in maintaining the heap property?
4. Provide examples of real-world problems where heaps are used effectively.

Lab Practice:

1. Write a program to implement a Max-Heap using an array.
2. Write a program to implement a Min-Heap using an array.
3. Create a function to insert an element into a heap and maintain the heap property.
4. Write a program to delete the root of a heap and perform the "Heapify" operation.
5. Implement a program to build a heap from a given array.
6. Write a program to perform Heap Sort on an array.
7. Create a priority queue implementation using a heap.
8. Implement a function to find the kth largest or kth smallest element in a given array using a heap.

WEEK-10 | CLO-05

Basics of Graph

Outcome:

- Understand the fundamental concepts of graphs and their representation.
- Learn about different types of graphs and basic graph traversal techniques.

Discussion Topics:

- [ ] Definition and key components of a graph like vertices, edges, degree, etc.
- [ ] Directed, Undirected, Weighted, Unweighted, Cyclic, and Acyclic graph.
- [ ] Adjacency Matrix and Adjacency List.
- [ ] Depth-First Search (DFS) and Breadth-First Search (BFS) algorithms.
- [ ] Real-world applications of graphs in social networks and transportation.

Questions:

1. What is a graph, and how is it different from a tree?
2. Explain the difference between an adjacency matrix and an adjacency list.
3. How does Depth-First Search (DFS) differ from Breadth-First Search (BFS)?
4. What are the advantages of using a weighted graph?
5. Provide a real-world example where graphs are used and explain its representation.

Lab Practice:

1. Implement Depth-First Search (DFS) for a given graph.
2. Write a program to perform Breadth-First Search (BFS) on a graph.
3. Write a program to calculate the shortest path between two nodes in a graph using BFS.
4. Implement a program to find all connected components in an undirected graph.