MANIPAL INSTITUTE OF TECHNOLOGY
BENGALURU
(A constituent unit of MAHE, Manipal)

III SEMESTER B. TECH (CSE, AI, CY, DS, IT), Assignment Questions, October 2025
Data Structures (CSS_2101)

Due Date: 31-10-2025 [5 Marks]

1. You are given a sorted array of distinct integers, arr, and a target integer, target. Your task is to implement a recursive binary search algorithm to find the index of target within arr. If target is not present, the function should return -1.
   Implement the same on the given array to find the element 23.

   arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]

   Note: Use dynamic memory allocation functions and pointers to create memory and access the elements of an array.

2. Design and implement a C program to efficiently represent and perform basic operations on a sparse matrix using a suitable data structure.
   Case Study: Consider a 5x5 sparse matrix where the non-zero elements are:
   (5, 5, 5), (0, 2, 5), (1, 1, 9), (2, 4, 3), (3, 0, 7), (4, 3, 1)

Implementation: Implement the following functions in C:

- createSparseMatrix(int rows, int cols, int numNonZero): Initializes the sparse matrix structure.
- insertElement(SparseMatrix* matrix, int row, int col, int value): Inserts a non-zero element into the sparse matrix.
- displaySparseMatrix(SparseMatrix* matrix): Displays the sparse matrix in a user-friendly format (e.g., triplet form or a reconstructed 2D matrix).

3. Write recursive C functions to perform the following:
   i. Compute the factorial of a given number.
   ii. Evaluate x^n (power of a number).
   iii. Find the GCD of two numbers using the Euclidean algorithm.
   iv. Display the intermediate results of each recursion.

4. The Registrar's office at a university needs to design a system to manage student records. This system must efficiently handle various tasks related to student data, including enrollment, grade management, and student information lookup. The core of this system will involve storing a collection of student records, where each record contains multiple pieces of related information.

   Task

   Your task is to design and implement a data structure for a student records system. You must define a struct to represent a single student and use it to build a data management system that meets the following requirements:

   Part 1: struct definition

   Define a struct named Student.

   The Student struct must contain the following members:

   a. student_id: A unique integer to identify each student.
   b. first_name: A string for the student's first name.
   c. last_name: A string for the student's last name.
   d. gpa: A floating-point number for the student's grade point average.
   e. major: A string indicating the student's major.

   Part 2: Dynamic data storage

   The system will handle an unknown number of students over time. An array with a fixed size is not an ideal solution, as it would be inefficient for memory use and inflexible for managing a changing number of students. You must choose and implement a dynamic data structure to hold a collection of Student structs.

   Part 3: Required operations

   Implement the following functions to manage the student records:

   add_student(): A function that takes student information as input and adds a new Student struct to your collection.

   delete_student(): A function that removes a student's record from the collection based on their student_id.

   find_student_by_id(): A function that searches for and displays a student's information, given their student_id.

   print_all_students(): A function that displays the information for all students currently in the system.

   get_gpa_average(): A function that calculates and returns the average GPA of all students in the collection.

5. Consider the expression: ({[a + b] * (c - d)} / e)
   Explain how your stack-based algorithm would process this expression to determine its validity. Provide a step-by-step trace of the stack's state (elements pushed/popped) for each character encountered. Your explanation should clearly demonstrate how the LIFO principle ensures correct pairing of opening and closing delimiters.
   Constraints:
   The input expression will only contain alphanumeric characters, operators, and the delimiters (,), {, }, [,].
   You should handle cases of mismatched delimiters (e.g., {[}), unclosed delimiters (e.g., ({)), and extra closing delimiters (e.g., ())).
   Your implementation should return True if the expression is valid and False otherwise.

6. Imagine a print server with a circular queue of capacity 5 to manage incoming print jobs.
   Initial State: The queue is empty. front = -1, rear = -1.
   Enqueue Operations:
   a. Job A arrives: enQueue(Job A). Queue: [Job A, _, _, _, _], front = 0, rear = 0.
   b. Job B arrives: enQueue(Job B). Queue: [Job A, Job B, _, _, _], front = 0, rear = 1.
   c. Job C arrives: enQueue(Job C). Queue: [Job A, Job B, Job C, _, _], front = 0, rear = 2.
   d. Job D arrives: enQueue(Job D). Queue: [Job A, Job B, Job C, Job D, _], front = 0, rear = 3.
   e. Job E arrives: enQueue(Job E). Queue: [Job A, Job B, Job C, Job D, Job E], front = 0, rear = 4. (Queue is full)
   Dequeue Operations:
   f. Print server processes Job A: deQueue(). Queue: [_, Job B, Job C, Job D, Job E], front = 1, rear = 4.
   g. Print server processes Job B: deQueue(). Queue: [_, _, Job C, Job D, Job E], front = 2, rear = 4.
   Enqueue with Wrap-around:
   h. Job F arrives: enQueue(Job F). Since the queue is circular, Job F is placed at index 0. Queue: [Job F, _, Job C, Job D, Job E], front = 2, rear = 0.
   i. Job G arrives: enQueue(Job G). Queue: [Job F, Job G, Job C, Job D, Job E], front = 2, rear = 1.

Your Task:

Implement the Circular Queue: Provide a detailed implementation of the circular queue data structure (using an array) in C programming language. Include methods for enQueue, deQueue, isFull, isEmpty, and peek (to view the front element without removing).

7. Consider a music player application where users can create playlists, navigate through songs, and repeat the entire playlist. Design a data structure to efficiently manage the songs within a single playlist, allowing for seamless forward and backward navigation, as well as continuous playback (looping).

   Case Study:

   You are tasked with implementing a Playlist class using a Circular Doubly Linked List. Each node in the list will represent a Song object, containing attributes such as title, artist, and duration.

   Tasks:

   Implement the Song Class: Define a Song class with appropriate attributes and a constructor.

   Implement the Playlist Class:

   a. **Node Structure:** Define the node structure for the circular doubly linked list, including data (a Song object), next pointer, and prev pointer.
   b. addSong(song): Add a new song to the end of the playlist.
   c. removeSong(title): Remove a song by its title from the playlist. Handle cases where the song is not found or the playlist becomes empty.
   d. playNext(): Advance to the next song in the playlist. If at the end, loop back to the beginning.
   e. playPrevious(): Go back to the previous song in the playlist. If at the beginning, loop to the end.
   f. displayPlaylist(): Print the details of all songs in the playlist in their current order.

   Demonstrate Functionality:

   g. Create a Playlist instance.
   h. Add at least five Song objects to the playlist.
   i. Demonstrate playNext() and playPrevious() operations multiple times, showing the circular nature.
   j. Remove a song from the middle of the playlist and demonstrate navigation again.
   k. Display the final state of the playlist.

   Example Scenario:

   Imagine a playlist with songs A, B, C, D, E.

Playing next from E should lead to A.
Playing previous from A should lead to E.
Removing song C should result in a playlist A, B, D, E, and playing next from B should lead to D.

8. Given the following sequence of ISBNs to be inserted into an initially empty Binary Search Tree: [50, 30, 70, 20, 40, 60, 80, 10, 25, 55, 65, 75, 85], perform the following operations and illustrate the state of the BST after each step:
   Insertion: Construct the BST by inserting all ISBNs from the given sequence.
   Search: Demonstrate the search operation for ISBN 60 and 90. Clearly indicate the path taken for each search.
   Deletion: Delete the node with ISBN 70. Describe the steps involved in handling this deletion, especially considering if the node has one child, two children, or no children.
   Traversal: Perform an in-order traversal of the final BST and list the ISBNs in the order they are visited.

9. Consider a binary tree data structure. Demonstrate and analyze the implementation of Inorder, Preorder, and Postorder traversals using both recursive and iterative approaches.
   Instructions:
   Construct a Binary Tree: Create a sample binary tree with at least 7-9 nodes, ensuring it has both left and right children for some nodes and some leaf nodes. Represent the tree structure clearly (e.g., using a diagram or a textual representation).
   Recursive Traversal Implementation:
   a. Provide the pseudocode or code snippets for recursive Inorder, Preorder, and Postorder traversals.
   b. Execute these recursive traversals on your sample binary tree and present the output for each.
   Iterative Traversal Implementation:
   c. Provide the pseudocode or code snippets for iterative Inorder, Preorder, and Postorder traversals (e.g., using stacks).
   d. Execute these iterative traversals on your sample binary tree and present the output for each.
   Comparative Analysis and Observations:

e. Compare the recursive and iterative implementations in terms of code complexity, readability, and efficiency (discussing potential stack overflow issues with recursion for very deep trees).
f. Discuss the specific use cases or scenarios where each traversal method (Inorder, Preorder, Postorder) is most appropriate, providing examples.
g. Analyze any challenges encountered during the implementation of either approach and how they were addressed.

10. Consider a social network represented as an undirected graph where individuals are nodes and a friendship between two individuals is an edge. You are tasked with analyzing this network to understand connectivity and influence.
    Question:
    Given the following social network graph:
    Code
    Nodes: A, B, C, D, E, F, G
    Edges: (A, B), (A, C), (B, D), (C, E), (D, F), (E, G), (F, G)
    Illustrate the traversal order of both Breadth-First Search (BFS) and Depth-First Search (DFS) starting from node 'A'. Clearly indicate the data structure used (Queue for BFS, Stack or Recursion for DFS) and the state of the data structure at each major step of the traversal.