# Data Structures through C

**Arrays and Strings:**

1. Write a C program to find the second-largest element in an array.
2. Implement a function to rotate an array to the left by a given number of positions.
3. Explain the differences between arrays and linked lists in C.
4. Write a program to check if a string is a palindrome in C.
5. Implement a C function to remove duplicate elements from a sorted array.
6. Explain the concept of a sparse matrix and implement its addition in C.
7. Write a program to find the intersection of two arrays in C.
8. Implement a function to perform a string reversal in C without using library functions.

**Linked Lists:**

1. Describe the advantages and disadvantages of singly linked lists in C.
2. Write a C program to find the middle element of a linked list.
3. Implement a function to detect and remove a loop in a linked list in C.
4. Explain the working of a doubly linked list and its applications in C.
5. Write a program to merge two sorted linked lists in C.
6. Design and implement a stack using a linked list in C.
7. Describe the role of a dummy node in a linked list and its benefits in C.
8. Implement a function to reverse every k nodes of a linked list in C.

**Stacks and Queues:**

1. Write a C program to evaluate a postfix expression using a stack.
2. Explain the concept of a circular queue and its implementation in C.
3. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time in C.
4. Implement a queue using two stacks in C.
5. Write a C program to check for balanced parentheses using a stack.
6. Describe the applications of stacks and queues in real-world scenarios.

**Trees:**

1. Implement a C program to perform an in-order traversal of a binary tree.
2. Write a function to find the height of a binary tree in C.
3. Explain the concept of a binary search tree and its properties in C.
4. Design and implement a binary tree with functions for insertion, deletion, and search in C.
5. Write a program to check if two binary trees are identical in C.
6. Explain the working of the AVL tree and its balancing operations in C.
7. Implement a function to find the lowest common ancestor in a binary tree in C.
8. Describe the applications of trees in data structures.

**Sorting and Searching:**

1. Compare the time complexity of bubble sort and quicksort in C.
2. Write a C program to perform the insertion sort on an array.
3. Explain the working of the binary search algorithm and its time complexity in C.
4. Implement a function to perform the selection sort on a linked list in C.
5. Design and implement a hash table in C with collision handling.
6. Describe the working of the linear search algorithm and its applications in C.
7. Write a C program to find the kth smallest element in an array using the quickselect algorithm.
8. Discuss the advantages and disadvantages of various sorting algorithms in C.

**Graphs:**

1. Explain the representation of a graph using adjacency matrices and lists in C.
2. Implement depth-first search (DFS) for a graph in C.
3. Write a program to check if a graph is cyclic using depth-first search in C.
4. Describe the working of Dijkstra's algorithm for finding the shortest path in a graph in C.
5. Implement breadth-first search (BFS) for a graph in C.
6. Discuss the applications of graphs in real-world scenarios.

**Dynamic Programming:**

1. Explain the concept of dynamic programming and its advantages in C.
2. Write a C program to find the nth Fibonacci number using dynamic programming.
3. Implement the 0/1 knapsack problem using dynamic programming in C.
4. Describe the working of the Longest Common Subsequence (LCS) algorithm in C.
5. Solve the Rod Cutting problem using dynamic programming in C.

**Advanced Data Structures:**

1. Implement a priority queue using a min-heap in C.
2. Write a C program to perform topological sorting for a directed acyclic graph.
3. Explain the concept of a tree and its applications in C.
4. Design and implement a disjoint-set data structure (Union-Find) in C.
5. Implement a skip list in C.

**Advanced Concepts:**

1. Explain the concept of bit manipulation and its applications in C.
2. Write a C program to check if a number is a power of two using bitwise operations.
3. Describe the working of the Boyer-Moore algorithm for string searching in C.
4. Implement a simple garbage collector for a dynamic memory allocation in C.
5. Discuss the advantages of using function pointers in C.

**File Handling and External Sorting:**

1. Write a C program to read and write data to a text file.
2. Implement external sorting using the merge-sort algorithm for large datasets in C.
3. Describe the concept of buffered and unbuffered I/O in C.

**Memory Allocation and Pointers:**

1. Explain the difference between malloc() and calloc() functions in C.
2. Write a program to implement a basic memory allocator using malloc() and free() in C.
3. Discuss the concept of segmentation faults and memory leaks in C.
4. Implement a function to reverse a string in place using pointers in C.

**Miscellaneous:**

1. Discuss the advantages and disadvantages of using recursion in C.
2. Write a C program to implement a basic calculator with addition, subtraction, multiplication, and division operations.
3. Explain the concept of time and space complexity in algorithm analysis in C.
4. Implement a program to generate prime numbers within a given range in C.

**Interview Preparation:**

1. Solve the "Two Sum" problem using a hash table in C.
2. Implement a function to check if a linked list has a cycle using Floyd's cycle detection algorithm in C.
3. Write a C program to reverse a linked list in groups of given size.
4. Discuss the difference between BFS and DFS traversal of a graph in C.
5. Solve the "Rain Water Trapping" problem using stacks in C.
6. Implement a function to check if a binary tree is a binary search tree in C.
7. Discuss the importance of the "const" keyword in C programming.

**System Design and Object-Oriented Concepts:**

1. Describe the principles of object-oriented programming in C.
2. Implement a basic class hierarchy with inheritance and polymorphism in C++.
3. Discuss the importance of encapsulation and abstraction in C++.
4. Design a simple system for a library management system using classes and objects in C++.

**Operating System Concepts:**

1. Explain the concept of process synchronization and semaphores in C.
2. Write a program to demonstrate multithreading in C using pthreads.
3. Discuss the basics of memory management in an operating system using malloc() and free() functions in C.

**Database Management System Concepts:**

1. Describe the differences between SQL and NoSQL databases.
2. Implement basic CRUD (Create, Read, Update, Delete) operations on a table using SQL queries in C.
3. Discuss the principles of normalization in database design.

**Software Engineering:**

1. Explain the concepts of version control systems and their importance.
2. Discuss the advantages and disadvantages of using agile methodology in software development.

**Coding Patterns:**

1. Implement a function to find the longest increasing subsequence in an array using dynamic programming in C.
2. Solve the "Reverse a Linked List in K-Group" problem using recursion in C.

**Problem-Solving Techniques:**

1. Describe the sliding window technique and its applications in C.
2. Write a C program to implement the Breadth-First Search algorithm for a grid-based problem.

**Industry Practices and Trends:**

1. Discuss the importance of test-driven development (TDD) in modern software engineering.
2. Explain the concept of continuous integration and continuous deployment (CI/CD) in C.