Write a C program for the following:

ARRAY

1. **Frequency Counter:** Find all elements of an array that have the highest frequency, and print them along with their count.
2. **Array Compression:** Given an array of integers, remove all duplicate values *in-place* without using an auxiliary array.
3. **Missing & Repeated Number Finder:** An array contains numbers from 1 to *n*, but one number is missing and one number is repeated. Find both.
4. **Subarray Sum Range:** Given an array and a target sum *S*, print all continuous subarrays whose sum equals *S*.
5. **Array Rotation by Reversing Method:** Implement left rotation of an array by *k* positions using only the reverse () operation.
6. **Majority Element:** Determine the element that appears more than *n*/2 times (if any).
7. **Maximum Product of Any Three Numbers:** Handle both negative and positive values.
8. **Second Largest Unique Number:** Do not sort the array; find the second largest **distinct** element.
9. **Wave Rearrangement:** Rearrange an array into a wave form: a1 ≥ a2 ≤ a3 ≥ a4 …

STACK

10. **Undo Operation Simulator:** Implement undo functionality (push operations to stack, pop to revert).
11. **Minimum Stack:** Implement a stack that supports **push, pop, and getMin()**.
12. **Two Stacks in a Single Array:** Implement two independent stacks in one shared array.
13. **Stack Reversal Using Only Push/Pop:** Reverse a stack using only stack operations and one auxiliary stack.
14. **Next Greater Element:** Use stack to find the next greater element for each array element.
15. **Evaluate Prefix Expression:** Implement prefix evaluation using stacks (not postfix or infix conversion).
16. **Factorial of a number:** Simulate recursive factorial using stack

QUEUE

16. **Interleave First Half with Second Half:** For a queue of even length, interleave its first half with its second half (e.g., 1 2 3 4 5 6 → 1 4 2 5 3 6).
17. **Queue Using Two Stacks:** Implement enqueue and dequeue using two stacks.
18. **Reverse First K Elements of a Queue:** Keep remaining elements in the same order.
19. **Job Queue with Time Slicing:** Given job arrival times and durations, simulate execution in cyclic order.
20. **Check If a Queue Is Palindrome:** Use only queue operations and at most one stack.
21. **First Non-Repeating Character Stream:** Given a stream of characters, print the first non-repeating character at each point using a queue.
22. **Queue Rotation:** Rotate queue by *k* places (circular behavior without using circular queue implementation).
23. **Inter-Queue Swap:** Swap the contents of two queues using only queue operations.
24. **Queue Sorting Using Only One Additional Queue:** No array or stack allowed.

LINKED LIST

25. **Intersection Point of Two Linked Lists:** If two lists merge at a node and find that node.
26. **Sort elements represented by Linked Lists:** Digits stored in reverse or forward order.
27. **Palindrome Check Without Extra Array:** Use pointer and reverse second half.
28. **Remove Duplicates from an Unsorted Linked List:** Without using extra buffer.
29. **Merge Two Sorted Linked Lists:** Without creating new nodes.
30. **Rotate Linked List:** Rotate right by *k* positions.
31. **Sort a Linked List (merge-sort on linked list):** Implement merge sort for linked lists.
32. **Find Middle Element in One Pass:** Use pointer.
33. **Pairwise Swap Nodes:** Swap nodes (not data) in pairs.
34. **Split Linked List into Two Alternating Lists:** L1 gets odd positions; L2 gets even positions.

SCENARIO-BASED

35. Imagine a determined young athlete named Alex who is training for a big competition. As part of his training, he must climb a set of n stairs to reach the top of a training tower. Each time he practises, he can either take a single step (climb 1 stair) or take a bigger leap (climb 2 stairs). Alex is curious about how many different ways he can reach the top of the stairs based on his climbing patterns. Alex decides to record his attempts to find out how many unique sequences of steps he takes to reach the top. Task Given an integer n representing the number of stairs, determine the number of distinct ways Alex can reach the top.
    Input Format: An integer n, the total number of stairs.
    Output Format: Print the number of distinct ways to reach the top
    Example 1: Input: n = 1 Output: 1
    Example 2: Input: n = 2 Output: 2
    Example 3: Input: n = 4 Output: 5

36. A librarian has a collection of n books arranged in a special way. To find a particular book, you must first look at the book on the top, then recursively search the rest of the pile. If you find the book, you need to count how many books you had to look at before finding it.
    Task: Write a recursive function that takes the number of books n and the position of the target book and counts how many books were looked at.
    Input Format: 2 integers: n (total books) & target (the position of the book you are looking for).
    Output Format: Print the number of books looked at.
    Sample Input: n = 5, target = 3
    Sample Output: 3 (You look at 1, then 2, then 3)

37. Mahesh has a collection of n number of marbles of different weights. Find the unique marbles, Mahesh needs to count ignoring the duplicate marbles, and update new count of marbles collection with new size.
    Task: Write a function that removes duplicate marbles from a smaller to the largest weight.
    Input Format: n (total marbles)
    Output Format: Print the number of marbled that are unique.

Sample Input: n = 11
{2, 1, 3, 2, 1, 4, 3, 4, 2, 5, 1}
Sample Output: m = 5
{1, 2, 3, 4, 5}

38. Anna works in a library, and she has a huge pile of books that need to be sorted by their titles. The pile is too large for her to handle all at once, so she decides to use a strategy where she breaks the pile into smaller stacks, sorts each stack individually, and then merges the stacks together to form the final sorted pile. Anna uses the merge sort technique to do this.
The merge sort process works by dividing the pile of books into two halves, recursively sorting each half, and then merging the sorted halves back together.
Problem: Anna needs your help to implement the merge sort algorithm to sort the titles of the books. Write a program to sort an array of book titles using the merge sort algorithm.
Input Format: An array of n book titles (as strings) to be sorted.
Output Format: Print the sorted list of book titles.
Input: War and Peace, Anna Karenina, Moby Dick, Great Expectations, The Odyssey
Output: Anna Karenina, Great Expectations, Moby Dick, The Odyssey, War and Peace

39. In a community garden, a gardener keeps track of various plants. Each plant is represented by its ID in a linked list. The gardener notices that the plants are listed in reverse order and wants to swap the positions of two specific plants.
Input Format: The first line contains the number of plants Ls(n). The second line contains nnn plant IDs (space-separated). The last line contains two plant IDs **aaa** and **bbb** that need to be swapped.
Output Format: Print the modified list of plant IDs after reversing and swapping.
Input: 6
101, 102, 105, 103, 104, 106, 105, 106
Output:106, 104, 103, 105, 102, 101

40. In an online coding platform like HackerRank, LeetCode, or a real-time code editor, users often write programs in various languages such as C++, Java, or Python. One of the most common syntax mistakes is forgetting to properly close parentheses, braces, or brackets ((), {}, []). This results in syntax errors and makes the code unable to compile or run.
To help users avoid such issues, the online platform provides a tool that automatically checks whether the parentheses in the user's code are balanced as they write. If the parentheses are unbalanced, the tool highlights the error in real-time and prevents the code from being compiled.
The goal is to develop a function that checks whether an expression or piece of code has balanced parentheses, ensuring that every opening parenthesis/bracket/brace has a corresponding closing one, and they are properly nested.
Input: s = "[()]{}{[()()]()}"
Output: True
Input: s = "[(])"
Output: False

41. As part of a computer science competition, participants are given a task to manipulate queues efficiently. One of the challenges is to reverse the order of the first K elements in a queue. The participants are required to implement a solution that achieves this task using a queue and basic programming concepts.
    Implement a function reverseFirstK that takes a queue and an integer K as input and reverses the order of the first K elements in the queue. The remaining elements in the queue should maintain their original order.
    Input: Queue: {1, 2, 3, 4, 5} and K: 3
    Output: Modified Queue: {3, 2, 1, 4, 5}

42. You are developing a search engine and need to analyze the web graph, where each web page is represented as a node and hyperlinks between pages are represented as directed edges.
    Given the adjacency list of web pages, count the total number of indexed web pages and the total number of hyperlinks connecting them.
    Question: Given an adjacency list representing a web graph, how many web pages are indexed, and how many directed hyperlinks connect them?
    Input:
    ```
    {
    {0, 1, 1, 0, 0},
    {0, 0, 1, 0, 0},
    {1, 0, 0, 0, 0},
    {0, 0, 1, 1, 0},
    {0, 0, 0, 0, 0}
    };
    ```
    Output:
    ```
    Number of web pages: 5
    Number of hyperlinks: 6
    ```

43. Heap Sort

Lab cycle programs

44. Array traversal using row major addressing
45. Array traversal using column major addressing
46. Sort array element row major addressing
47. Sort array element row major addressing
48. Sort 2D array elements
49. Searching Algorithms: Linear Search
50. Searching Algorithms: Binary Search
51. Sorting Algorithms: Bubble sort
52. Sorting Algorithms: Insertion Sort
53. Sorting Algorithms: Selection Sort
54. Sorting Algorithms: Shell sort
55. Singly Linked List operations: Insertion, deletion, search and traversal

56. Circular Linked List operations: Insertion, deletion, search and traversal
57. Doubly Linked List operations: Insertion, deletion, search and traversal
58. Implementation of stack using arrays and linked list
59. Implementation of stack using linked list
60. Infix to postfix conversion
61. Postfix evaluation
62. Sorting Algorithms: Merge sort & Quick sort
63. Sorting Algorithms: Quick sort
64. Implementation of queue using arrays
65. Implementation of queue using linked list
66. Circular queue operation
67. Double-ended queue operation
68. Priority queue operation
69. Graph Traversal: BFS and DFS
70. Binary tree traversal: pre, in, and post-order
71. Binary search tree