1).Create a program to sort the elements in an array using pointers.

2).Write a **menu-driven C++ program** that performs **recursive operations** on numbers:

1. Find the factorial of a number
2. Find the GCD of two numbers
3. Find the LCM of two numbers (use GCD recursively)
4. Find the sum of digits of a number
5. Reverse a number
6. Exit

The program should display a menu to the user, take input, and perform the selected operation.

3).Implement a stack using arrays in C++.

- Create functions for the following operations:
    1. push() – to insert an element into the stack
    2. pop() – to remove the top element from the stack
    3. display() – to display all elements of the stack
- Use array-based implementation and handle stack overflow and underflow.
- Write a menu-driven program to test these functions.

4).Write a program to process a 2D array of integers:

- Traverse each element and check whether it is a palindrome. Replace palindromes with 1 and non-palindromes with 0.
- Print the resulting matrix of 1s and 0s.

Input:

121 23 44

56 77 89

33 12 9

Output:

1 0 1

0 1 0

1 0 1

5).Write a program to rotate a singly linked list by k positions to the right.

Input: 1 -> 2 -> 3 -> 4 -> 5->NULL

k = 2

Output: 4 -> 5 -> 1 -> 2 -> 3->NULL

6).Create a class Employee with the following private data members:

- name (string)
- id (integer)
- salary (float)

The class should have public member functions:

1. void input() — to input details of an employee
2. void display() — to display details of an employee
3. void giveRaise(float percent) — to increase the employee's salary by a given percentage

Write a program to:

1. Create an array of Employee objects.
2. Input details for all employees using input().
3. Display all employee details using display().
4. Ask the user for a salary raise percentage and apply it to all employees.
5. Sort the employees in ascending order of salary using Selection Sort implemented by recursion.

6. Display the sorted and updated employee details.

7).Write a **C++ program** to perform operations on a **singly linked list** using **recursion**.

Requirements:

1. Define a Node structure:

```
struct Node {
   int data;
   Node* next;
};
```

2. Implement the following **recursive functions**:

- Node* insert(Node* head, int value)
    - Recursively insert a new node at the **end** of the linked list.
- Node* sortLinkedList(Node* head)
    - Recursively sort the linked list in **ascending order**.
- void display(Node* head)
    - Recursively print all elements of the linked list.

3. In the main() function:

- Ask the user to input the number of elements.
- Insert elements into the linked list.
- Display the **original list**.
- Sort the linked list recursively.
- Display the **sorted list**.

8).Write a program to perform various operations on a string. The program should repeatedly display a menu to the user and perform the selected operation until the user chooses to exit (no built-in functions should be used).

The program should allow the user to:

1. Find the length of a string
2. Reverse a string
3. Check if a string is a palindrome
4. Count vowels and consonants in a string
5. Exit the program

9).Write a **C++ program** that converts a **mathematical expression in infix form** (e.g., A + B * C) into its **postfix form** (also called Reverse Polish Notation, e.g., A B C * +) using a **stack-based approach**. Implement the stack using singly linked lists.

The program should accept a infix expression from the user and output its postfix equivalent.

10).Write a **C++ program** to implement a **queue** using a **linked list** and perform basic queue operations.

Requirements

Define a **Node** structure:

```
struct Node {
   int data;
   Node* next;
};
```

Create a **Queue** class with the following **member functions**:

- void enqueue(int value) – Insert an element at the **rear** of the queue.
- int dequeue() – Remove and return the element from the **front** of the queue.
- void display() – Display all elements of the queue from **front to rear**.
- bool isEmpty() – Check if the queue is empty.

Implement **dynamic memory allocation** for nodes using new and delete.

In the main() function:

Display a **menu** to the user:

- Enqueue
- Dequeue
- Display
- Exit

Perform operations according to the user's choice.

11).Write a **C++ program** to create a **binary tree** and perform basic operations.

Requirements:

Implement a Node structure with:

int data

Node* left

Node* right

Implement a BinaryTree class with the following **member functions**:

- Node* insert(Node* root, int value) – Insert a node into the binary tree.
- void inorder(Node* root) – Print **inorder traversal**.
- void preorder(Node* root) – Print **preorder traversal**.
- void postorder(Node* root) – Print **postorder traversal**.
- int height(Node* root) – Return the height of the tree.

In the main() function:

Take input values from the user to create the tree.

Display **all traversals**, number of nodes, and height.

Use recursion for all post, pre and in order traversals.