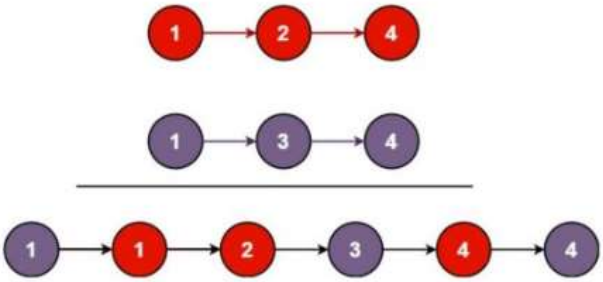
DSA LAB 02

Name: *Syed Wahaaj Ali*
Roll No: *CT-24035*
Course Code: *CT-159*
CSIT
Section: *A*

Q1:

1. You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.
   Example: **Input:** list1 = [1,2,4], list2 = [1,3,4], **Output:** [1,1,2,3,4,4]



Code:

```
/*
    Syed Wahaaj Ali
    CT-24035
*/

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = nullptr;
    }
};

class LinkedList {
public:
    Node* mergeTwoLists(Node* list1, Node* list2) {
        Node dummy(0); // dummy node to simplify logic
        Node* tail = &dummy;

        while (list1 && list2) {
            if (list1->data <= list2->data) {
                tail->next = list1;
                list1 = list1->next;
            } else {
                tail->next = list2;
                list2 = list2->next;
```

```
            }
            tail = tail->next;
        }

        // attach the remaining nodes
        if (list1) {
            tail->next = list1;
        } else {
            tail->next = list2;
        }

        return dummy.next;
    }

    void printList(Node* head) {
        while (head) {
            cout << head->data << " ";
            head = head->next;
        }
        cout << endl;
    }
};

int main() {
    // First sorted list: 1 -> 3 -> 5 -> 7
    Node* list1 = new Node(1);
    list1->next = new Node(3);
    list1->next->next = new Node(5);
    list1->next->next->next = new Node(7);

    // Second sorted list: 2 -> 4 -> 6 -> 8 -> 10
    Node* list2 = new Node(2);
    list2->next = new Node(4);
    list2->next->next = new Node(6);
    list2->next->next->next = new Node(8);
    list2->next->next->next->next = new Node(10);

    LinkedList obj;
    Node* merged = obj.mergeTwoLists(list1, list2);

    cout << "Merged Sorted List: ";
    obj.printList(merged);

    return 0;
}
```

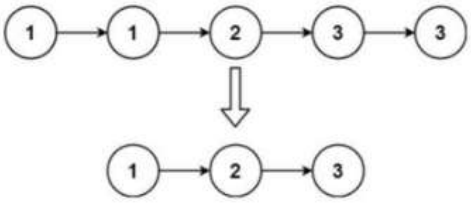
Output:

```
Merged Sorted List: 1 2 3 4 5 6 7 8 10


=== Code Execution Successful ===
```

Q2:

2. Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.



Code:

```cpp
/*
    Syed Wahaaj Ali
    CT-24035
*/

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = nullptr;
    }
};

class LinkedList {
public:
    Node* deleteDuplicates(Node* head) {
        Node* current = head;

        while (current != NULL && current->next != NULL) {
            if (current->data == current->next->data) {
                // Duplicate found? remove next node
                Node* duplicate = current->next;
                current->next = current->next->next;
                delete duplicate; // free memory
            } else {
                // Move forward if no duplicate
                current = current->next;
            }
        }

        return head;
    }

    void printList(Node* head) {
        while (head) {
            cout << head->data << " ";
            head = head->next;
        }
        cout << endl;
```

```
    }
};

int main() {
    // Sorted list with duplicates: 1 -> 1 -> 2 -> 3 -> 3 -> 4 -> 4 -> 4 -> 5
    Node* head = new Node(1);
    head->next = new Node(1);
    head->next->next = new Node(2);
    head->next->next->next = new Node(3);
    head->next->next->next->next = new Node(3);
    head->next->next->next->next->next = new Node(4);
    head->next->next->next->next->next->next = new Node(4);
    head->next->next->next->next->next->next->next = new Node(4);
    head->next->next->next->next->next->next->next->next = new Node(5);

    LinkedList obj;

    cout << "Original List: ";
    obj.printList(head);

    head = obj.deleteDuplicates(head);

    cout << "After Removing Duplicates: ";
    obj.printList(head);

    return 0;
}
```

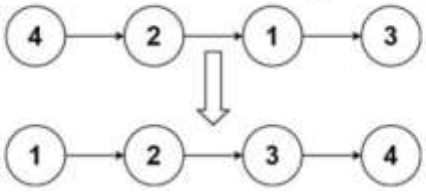
Output:

```
Original List: 1 1 2 3 3 4 4 4 5
After Removing Duplicates: 1 2 3 4 5


=== Code Execution Successful ===
```

Q3:

3. Given the head of a linked list, return the list after sorting it in ascending order. Use mergesort.



Code:

```
/*
    Syed Wahaaj Ali
    CT-24035
*/

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = nullptr;
    }
};

class LinkedList {
public:
    // Merge two sorted lists (from Ex1)
    Node* mergeTwoLists(Node* list1, Node* list2) {
        Node dummy(0);
        Node* tail = &dummy;

        while (list1 && list2) {
            if (list1->data <= list2->data) {
                tail->next = list1;
                list1 = list1->next;
            } else {
                tail->next = list2;
                list2 = list2->next;
            }
            tail = tail->next;
        }

        if (list1)
            tail->next = list1;
        else
            tail->next = list2;

        return dummy.next;
    }

    // Sort list using Merge Sort
    Node* sortList(Node* head) {
        // Base case: 0 or 1 node
```

```
        if (head == nullptr || head->next == nullptr)
            return head;

        // Find middle using slow-fast pointer
        Node* slow = head;
        Node* fast = head->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        // Split list into two halves
        Node* rightHalf = slow->next;
        slow->next = nullptr;
        Node* leftHalf = head;

        // Recursively sort both halves
        Node* leftSorted = sortList(leftHalf);
        Node* rightSorted = sortList(rightHalf);

        // Merge sorted halves
        return mergeTwoLists(leftSorted, rightSorted);
    }

    // Print list
    void printList(Node* head) {
        while (head) {
            cout << head->data << " ";
            head = head->next;
        }
        cout << endl;
    }
};

int main() {
    // Unsorted list: 4 -> 2 -> 1 -> 3 -> 5
    Node* head = new Node(4);
    head->next = new Node(2);
    head->next->next = new Node(1);
    head->next->next->next = new Node(3);
    head->next->next->next->next = new Node(5);

    LinkedList obj;

    cout << "Original List: ";
    obj.printList(head);

    head = obj.sortList(head);

    cout << "Sorted List: ";
    obj.printList(head);

    return 0;
}
```

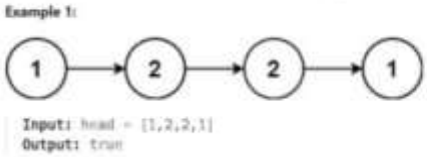
Output:

```
Original List: 4 2 1 3 5
Sorted List: 1 2 3 4 5


=== Code Execution Successful ===
```

Q4:

4. Given the head of a singly linked list, return true if it is a palindrome or false otherwise.



Code:

```
/*
    Syed Wahaaj Ali
    CT-24035
*/

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = nullptr;
    }
};

class LinkedList {
public:
    // reverse a linked list
    Node* reverseList(Node* head) {
        Node* prev = nullptr;
        Node* curr = head;
        while (curr) {
            Node* nextNode = curr->next;
            curr->next = prev;
            prev = curr;
            curr = nextNode;
        }
        return prev; // new head
    }

    // Check if list is palindrome
    bool isPalindrome(Node* head) {
        if (!head || !head->next)
            return true;

        // 1. Find middle
        Node* slow = head;
        Node* fast = head;
        while (fast->next && fast->next->next) {
            slow = slow->next;
            fast = fast->next->next;
        }

        // 2. Reverse second half
```

```
        Node* secondHalfStart = reverseList(slow->next);

        // 3. Compare both halves
        Node* firstPointer = head;
        Node* secondPointer = secondHalfStart;
        bool result = true;
        while (secondPointer) {
            if (firstPointer->data != secondPointer->data) {
                result = false;
                break;
            }
            firstPointer = firstPointer->next;
            secondPointer = secondPointer->next;
        }

        // 4. Restore the list
        slow->next = reverseList(secondHalfStart);

        return result;
    }

    // Print list
    void printList(Node* head) {
        while (head) {
            cout << head->data << " ";
            head = head->next;
        }
        cout << endl;
    }
};

int main() {
    // Palindrome list: 1 -> 2 -> 3 -> 2 -> 1
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(2);
    head->next->next->next->next = new Node(1);

    LinkedList obj;

    cout << "Original List: ";
    obj.printList(head);

    if (obj.isPalindrome(head))
        cout << "The list is a palindrome." << endl;
    else
        cout << "The list is NOT a palindrome." << endl;

    cout << "List after check (restored): ";
    obj.printList(head);

    return 0;
}
```

Output:

```
Original List: 1 2 3 2 1
The list is a palindrome.
List after check (restored): 1 2 3 2 1
```

Q5:

5. Implement class of a Circular Queue using a Linked List.

Code:

```
/*
    Syed Wahaaj Ali
    CT-24035
*/

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = nullptr;
    }
};

class CircularQueue {
private:
    Node* front;
    Node* rear;

public:
    CircularQueue() { front = rear = nullptr; }

    // Enqueue: add at rear
    void enqueue(int val) {
        Node* newNode = new Node(val);
        if (!front) { // empty queue
            front = rear = newNode;
            rear->next = front; // circular link
        } else {
            rear->next = newNode;
            rear = newNode;
            rear->next = front; // keep it circular
        }
        cout << val << " enqueued\n";
    }

    // Dequeue: remove from front
    void dequeue() {
        if (!front) {
            cout << "Queue is empty!\n";
            return;
        }
        if (front == rear) { // only one element
            cout << front->data << " dequeued\n";
            delete front;
            front = rear = nullptr;
        } else {
            Node* temp = front;
```

```
            cout << front->data << " dequeued\n";
            front = front->next;
            rear->next = front; // maintain circular
            delete temp;
        }
    }

    // Peek front element
    int peek() {
        if (!front) {
            cout << "Queue is empty!\n";
            return -1;
        }
        return front->data;
    }

    // Display
    void display() {
        if (!front) {
            cout << "Queue is empty!\n";
            return;
        }
        Node* temp = front;
        cout << "Queue: ";
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != front);
        cout << endl;
    }
};

int main() {
    CircularQueue q;
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.display();
    cout << "Front element = " << q.peek() << endl;
    q.dequeue();
    q.display();
    q.dequeue();
    q.dequeue();
    q.dequeue(); // extra dequeue test
    return 0;
}
```

Output:

```
10 enqueued
20 enqueued
30 enqueued
Queue: 10 20 30
Front element = 10
10 dequeued
Queue: 20 30
20 dequeued
30 dequeued
Queue is empty!


=== Code Execution Successful ===
```

Q6:

6. Implement class of a Stack using a Linked List.

Code:

```cpp
/*
    Syed Wahaaj Ali
    CT-24035
*/

#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = nullptr;
    }
};

class Stack {
private:
    Node* top; // top of stack
public:
    Stack() { top = nullptr; }

    // Push = insert at head
    void push(int val) {
        Node* newNode = new Node(val);
        newNode->next = top;
        top = newNode;
        cout << val << " pushed\n";
    }

    // Pop = remove from head
    void pop() {
        if (!top) {
            cout << "Stack is empty!\n";
            return;
        }
        Node* temp = top;
        cout << top->data << " popped\n";
        top = top->next;
        delete temp;
    }

    // Peek = see top
    int peek() {
        if (!top) {
            cout << "Stack is empty!\n";
            return -1;
        }
        return top->data;
```

```
    }

    // Display
    void display() {
        if (!top) {
            cout << "Stack is empty!\n";
            return;
        }
        cout << "Stack: ";
        Node* temp = top;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    Stack s;
    s.push(5);
    s.push(10);
    s.push(15);
    s.display();
    cout << "Top element = " << s.peek() << endl;
    s.pop();
    s.display();
    s.pop();
    s.pop();
    s.pop(); // extra pop test
    return 0;
}
```

Output:

```
5 pushed
10 pushed
15 pushed
Stack: 15 10 5
Top element = 15
15 popped
Stack: 10 5
10 popped
5 popped
Stack is empty!


=== Code Execution Successful ===
```