<u>**Operations on Linked List**</u>

1. Traversal
2. Insertion
3. Deletion
4. Searching
5. Reversing
6. Copying
7. Merging
8. Splitting

1. Traversal of Singly Linked List

// Function to print all elements of the list from start to end.

```
void displayList(Node* head) {
   if (head == NULL) {
      cout << "List is empty.\n";
      return;
   }
   cout << "Linked List: ";
   Node* temp = head;
   while (temp != NULL) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}
```

2. INSERTION AT THE END

// Function to insert a node at the end

```
void insertEnd(Node*& head, int value) {
   Node* newNode = createNode(value);
```

*/*If the list is empty, the new node becomes the head else it travels to the last node and adds the new node.*/*

```
   if (head == NULL) {
      head = newNode;
   } else {
        Node* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
}
```

3. **DELETION AT THE END**

// Function to delete a node at the end

```
void deleteAtEnd(Node*& head) {
    if (head == NULL) {
        cout << "List is already empty.\n";
        return;
    }

    // If there is only one node
    if (head->next == NULL) {
        delete head;
        head = NULL;
        return;
    }

    // Traverse to the second last node
    Node* temp = head;
    while (temp->next->next != NULL) {
        temp = temp->next;
    }

    delete temp->next;      // Delete last node
    temp->next = NULL;      // Set new end of list
}
```

4. **SEARCHING A SPECIFIC VALUE(KEY) IN THE LIST**

```
void searchList(Node* head, int key) {
    Node* temp = head;
    int pos = 1;
    bool found = false;
    while (temp != NULL) {
        if (temp->data == key) {
            cout << "Element " << key << " found at position " << pos << ".\n";
            found = true;
        }
        temp = temp->next;
        pos++;
    }

    if (!found)
        cout << "Element " << key << " not found in the list.\n";
}
```

5. **REVERSE DISPLAY (Recursive)**

   // Function to display list in reverse using recursion

```
void displayReverse(Node* head) {
    if (head == NULL)
        return;
    displayReverse(head->next); // Go to the end
    cout << head->data << " "; // Print during unwinding
}
```

6. **COPYING A LIST**

```
Node* copyList(Node* head) {
    if (head == NULL) return NULL;

    Node* newHead = new Node{head->data, NULL};
    Node* tempNew = newHead;
    Node* tempOld = head->next;

    while (tempOld != NULL) {
        tempNew->next = new Node{tempOld->data, NULL};
        tempNew = tempNew->next;
        tempOld = tempOld->next;
    }
    return newHead;
}
```

7. **MERGING TWO LISTS**

```
Node* mergeLists(Node* head1, Node* head2) {
    if (head1 == NULL) return head2;
    if (head2 == NULL) return head1;

    Node* merged = head1;
    while (head1->next != NULL)
        head1 = head1->next;
    head1->next = head2;

    return merged;
}
```

8. **SPLITTING LIST INTO TWO HALVES**

```
void splitList(Node* head, Node*& firstHalf, Node*& secondHalf) {
    if (head == NULL || head->next == NULL) {
        firstHalf = head;
        secondHalf = NULL;
        return;
    }

    Node* slow = head;
    Node* fast = head->next;

    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }

    firstHalf = head;
    secondHalf = slow->next;
    slow->next = NULL;
}
```