Experiment-3

Student Name: Kumar Adya **UID:** 23BCS80040
Branch: BE-CSE **Section/Group:** 636/A
Semester: 6th **Date of Performance:**27/01/25
Subject Name: Advanced Programming Lab - 2 **Subject Code:** 22CSP-351

1. Aim:

1. Problem: 1.3.1: Merge Two Sorted Lists. 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.
2. Problem: 1.3.2: Remove Duplicates from Sorted List II. Given the head of a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. Return the linked list sorted as well.

2. Objective:

1. Problem 1.3.1: Merge two sorted linked lists into a single sorted linked list by combining their nodes sequentially.
2. Problem 1.3.2: Remove all duplicate nodes from a sorted linked list, leaving only nodes with distinct values in sorted order.

3. Implementation/Code:

1.)

```
#include <iostream>

using namespace std;

struct ListNode
{
```

```
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};

class Solution
{
public:
    ListNode *mergeTwoLists(ListNode *list1, ListNode *list2)
    {
        ListNode dummy(0);
        ListNode *current = &dummy;

        while (list1 != nullptr && list2 != nullptr)
        {
            if (list1->val < list2->val)
            {
                current->next = list1;
                list1 = list1->next;
            }
            else
            {
                current->next = list2;
                list2 = list2->next;
            }
            current = current->next;
        }

        if (list1 != nullptr)
            current->next = list1;
        if (list2 != nullptr)
            current->next = list2;
```

```
        return dummy.next;
    }
};

ListNode *createList()
{
    int n, val;
    cout << "Enter the number of elements: ";
    cin >> n;

    if (n == 0)
        return nullptr;

    cout << "Enter the elements: ";
    cin >> val;
    ListNode *head = new ListNode(val);
    ListNode *current = head;

    for (int i = 1; i < n; i++)
    {
        cin >> val;
        current->next = new ListNode(val);
        current = current->next;
    }

    return head;
}

void printList(ListNode *head)
{
    while (head != nullptr)
```

```
    {
        cout << head->val << " -> ";
        head = head->next;
    }
    cout << "nullptr" << endl;
}

int main()
{
    cout << "Create List 1:" << endl;
    ListNode *list1 = createList();

    cout << "Create List 2:" << endl;
    ListNode *list2 = createList();

    cout << "List 1: ";
    printList(list1);

    cout << "List 2: ";
    printList(list2);

    Solution solution;
    ListNode *mergedHead = solution.mergeTwoLists(list1, list2);

    cout << "Merged List: ";
    printList(mergedHead);

    return 0;
}
```

2.)

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct ListNode
{
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};

class Solution
{
public:
    ListNode *deleteDuplicates(ListNode *head)
    {
        if (!head)
            return nullptr;

        ListNode dummy(0);
        dummy.next = head;
        ListNode *prev = &dummy;

        while (head)
        {
            bool isDuplicate = false;
            while (head->next && head->val == head->next->val)
            {
                head = head->next;
```

```
            isDuplicate = true;
        }
        if (isDuplicate)
        {
            prev->next = head->next;
        }
        else
        {
            prev = prev->next;
        }
        head = head->next;
    }

    return dummy.next;
  }
};

ListNode *createList()
{
  int n, val;
  cout << "Enter the number of elements: ";
  cin >> n;

  if (n == 0)
    return nullptr;

  vector<int> elements;
  cout << "Enter the elements: ";
  for (int i = 0; i < n; i++)
  {
    cin >> val;
    elements.push_back(val);
```

```
    }

    sort(elements.begin(), elements.end());

    ListNode *head = new ListNode(elements[0]);
    ListNode *current = head;

    for (int i = 1; i < n; i++)
    {
        current->next = new ListNode(elements[i]);
        current = current->next;
    }

    return head;
}

void printList(ListNode *head)
{
    while (head != nullptr)
    {
        cout << head->val << " -> ";
        head = head->next;
    }
    cout << "nullptr" << endl;
}

int main()
{
    cout << "Create List:" << endl;
    ListNode *list = createList();

    cout << "User Input List: ";
```

```
    printList(list);

    Solution solution;
    ListNode *result = solution.deleteDuplicates(list);

    cout << "List After Removing Duplicates: ";
    printList(result);

    return 0;
}
```

4. Output:

1.

```
PS D:\class_problem\ap\exp_3> cd "d:\class_problem\ap\exp_3\" ; if ($?) { g++ test1.cpp -o test1 } ; if ($?) { .\test1 }
Create List 1:
Enter the number of elements: 3
Enter the elements: 1 2 3
Create List 2:
Enter the number of elements: 3
Enter the elements: 4 5 6
List 1: 1 -> 2 -> 3 -> nullptr
List 2: 4 -> 5 -> 6 -> nullptr
Merged List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> nullptr
PS D:\class_problem\ap\exp_3>
```

2.

```
PS D:\class_problem\ap\exp_3> cd "d:\class_problem\ap\exp_3\" ; if ($?) { g++ test2.cpp -o test2 } ; if ($?) { .\test2 }
Create List:
Enter the number of elements: 6
Enter the elements: 1 2 3 1 4 6
User Input List: 1 -> 1 -> 2 -> 3 -> 4 -> 6 -> nullptr
List After Removing Duplicates: 2 -> 3 -> 4 -> 6 -> nullptr
PS D:\class_problem\ap\exp_3>
```

5. **Time Complexity:**
   1. O(n+m)
   2. O(n)

6. **Space Complexity:**
   1. O(n+m)
   2. O(1)

7. **Learning Outcome:**
   1. Understand how to merge two sorted linked lists efficiently.
   2. Practice working with linked list pointers and traversal.
   3. Learn how to identify and remove duplicate nodes in a sorted linked list.
   4. Gain experience in handling edge cases in linked list operations.
   5. Improve problem-solving skills with recursion or iterative approaches.
   6. Develop the ability to write clean, modular code for linked list problems.