# REMOVING DUPLICATES IN AN ARRAY USING LINKED LIST DATA STRUCTURE

A Presentation by Group 7

*CSC317- Data Structures and Algorithms I*

# Table of Contents

# Introduction

An array is a fundamental data structure that stores a collection of elements, typically of the same type. However, in many scenarios, arrays can contain duplicate elements, leading to issues such as inefficient memory usage, inaccurate results, performance degradation, and complex data processing.

# Problem Statement

Given an unsorted list of integer numbers in an array containing duplicates, write a program using a linked list data structure to output array without duplicates.

# Solution Analysis

Why use a linked list?

- The dynamic nature of a linked list makes size adjustment easy.
- Linked lists require les memory usage
- A linked list allows efficient deletion of nodes, which helps in removing duplicates.

# Algorithm

1. Start
2. Initialize an empty array to store user inputs.
3. Read integer inputs from the user, and store in the array.
4. Initialize an empty linked list.
5. Traverse through the array.
6. For each element in the array, create a new node in the linked list with the element as its data.
7. For each node in the linked list
    - Compare the current node's data with the data of every other node ahead of it in the linked list.
    - If a duplicate is found (i.e., a node with the same data), remove the duplicate node from the list.
    - Continue until the end of the list is reached.
8. Traverse through the modified linked list.
9. Print the data of each node, which now represents the array without duplicates.
10. Stop.

# Demonstration – Source code

```cpp
#include <iostream>
#include <vector>
using namespace std;

// Node structure for the linked list
struct Node
{
    int data;
    Node *next;
};

// Function to create a new node with given data
Node *newNode(int data)
{
    Node *temp = new Node; // Allocate new node
    temp->data = data;     // Set the data
    temp->next = NULL;     // Next pointer set to NULL
    return temp;           // Return the new node
}
```

```cpp
// Function to remove duplicates from the linked list
void removeDuplicates(Node *start)
{
    Node *ptr1, *ptr2, *dup;
    ptr1 = start;

    // Outer loop for each element in the list
    while (ptr1 != NULL && ptr1->next != NULL)
    {
        ptr2 = ptr1;

        // Inner loop to compare the element with the rest of the list
        while (ptr2->next != NULL)
        {
            // If duplicate is found
            if (ptr1->data == ptr2->next->data)
            {
                // Remove the duplicate
                dup = ptr2->next;
                ptr2->next = ptr2->next->next;
                delete dup; // Free memory of the duplicate node
            }
            else
            {
                ptr2 = ptr2->next;
            }
        }
        ptr1 = ptr1->next;
    }
}
```

```
// Function to print the elements of the linked list
void printList(Node *node)
{
    while (node != NULL)
    {
        cout << node->data << " ";
        node = node->next;
    }
}

int main()
{
    vector<int> inputs; // Vector to store user inputs
    int num;
    cout << "Enter integers (Enter -1 to end): " << endl;

    // Reading integers until -1 is entered
    while (true)
    {
        cin >> num;
        if (num == -1)
            break;
        inputs.push_back(num);
    }
```

```
    // Creating linked list from the vector of inputs
    for (int value : inputs)
    {
        if (start == NULL)
        {
            start = newNode(value);
            current = start;
        }
        else
        {
            current->next = newNode(value);
            current = current->next;
        }
    }

    cout << "Linked list before removing duplicates: " << endl;
    printList(start);
    removeDuplicates(start); // Remove duplicates from the list
    cout << "\nLinked list after removing duplicates: " << endl;
    printList(start);

    // Freeing memory used by the linked list
    Node *temp;
    while (start != NULL)
    {
        temp = start;
        start = start->next;
        delete temp;
    }
```

# Demonstration – Results

```
$ ./linkedList
Enter integers (Enter -1 to end):
20 40 2 50 60 2 30 30 50 75 60 -1
Linked list before removing duplicates:
20 40 2 50 60 2 30 30 50 75 60
Linked list after removing duplicates:
20 40 2 50 60 30 75
```

```
$ ./linkedList
Enter integers (Enter -1 to end):
1 1 4 5 6 5 7 5 8 9 1 -1
Linked list before removing duplicates:
1 1 4 5 6 5 7 5 8 9 1
Linked list after removing duplicates:
1 4 5 6 7 8 9
```

# Solution Review

1. The code effectively removes duplicates from an unsorted list of integers, meeting the primary objective of task.
2. The dynamic data structure allows for datasets where the size is not predetermined, as it can easily expand or contract.
3. It modifies the linked list in place, eliminating the need for additional memory allocation, which can help with large datasets.

# Limitations

1. Unlike arrays, linked lists do not provide random access to their elements. This can be a drawback in scenarios where such access is required.
2. Each node in a linked list requires extra memory for the pointer, in addition to the data it holds. This increases the space complexity.
3. The implementation is highly dependent on the correct management of the Node structure and pointer manipulations, which can be error-prone and requires careful handling.

# Conclusion

In conclusion, this project effectively demonstrates essential concepts in algorithms and data structures, particularly for small to medium-sized datasets.
With limitations like the time and space complexity, solutions such as hashing, sorting, or using alternative data structures like hash tables would produce better results to help handle larger data sets and efficiently handle memory usage.