Problem Statement 1

Nandha wants to delete a range of nodes between two given positions in a singly linked list. Write a program that takes input for the size of the linked list, the elements of the linked list, the start position, and the end position. The program should delete the nodes between the specified positions and display the updated linked list.

Company Tags: TCS

Input format :

The first line contains an integer '**n**' representing the size of the linked list.

The second line contains '**n**' space-separated integers representing the elements of the linked list.

The third line contains an integer '**S**' representing the start position.

The fourth line contains an integer '**E**' representing the end position.

Output format :

The first line of output displays "Linked List before deletion: " followed by the linked list before deletion.

The second line of output displays "Linked List after deletion: " followed by the linked list after deletion.

Refer to the sample output for formatting specifications.

Code constraints :

1 ≤ n ≤ 100

-50000 ≤ elements ≤ 50000

1 ≤ S ≤ 100

1 ≤ E ≤ 100

Sample test cases :

Input 1 :

```
5

1 2 3 4 5

1

3
```

Output 1 :

Linked List before deletion: 1 2 3 4 5

Linked List after deletion: 4 5

Input 2 :

5

-50000 50000 4000 3676 7263

1

5

Output 2 :

Linked List before deletion: -50000 50000 4000 3676 7263

Linked List after deletion: all the elements are deleted

```
// You are using GCC
#include <iostream>
using namespace std;

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

void insertNode(Node** head, int data) {
  Node* newNode = new Node();
  newNode->data = data;
  newNode->next = nullptr;

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

```
        temp->next = newNode;
    }
}

void deleteNodesInRange(Node** head, int start, int end) {
    if(*head == nullptr || start > end) return;

    Node* temp = *head;
    Node* prev = nullptr;

    int pos = 1;

    while(temp != nullptr && pos < start){
        prev = temp;
        temp = temp->next;
        pos++;
    }

    while(temp != nullptr && pos <= end){
        Node* del = temp;
        temp = temp->next;
        delete del;
        pos++;
    }

    if(prev == nullptr){
        *head = temp;
    }else{
        prev->next = temp;
    }
}
```

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

void deleteLinkedList(Node* head) {
   Node* temp;
   while (head != nullptr) {
      temp = head;
      head = head->next;
      delete temp;
   }
}

int main() {
   Node* head = nullptr;
   int size;

   cin >> size;

   for (int i = 0; i < size; i++) {
      int value;
      cin >> value;
      insertNode(&head, value);
   }
```

```
    int start, end;
    cin >> start;
    cin >> end;

    cout << "Linked List before deletion: ";
    displayLinkedList(head);

    deleteNodesInRange(&head, start, end);

    if (head == nullptr) {
        cout << "Linked List after deletion: all the elements are deleted" << endl;
    } else {
        cout << "Linked List after deletion: ";
        displayLinkedList(head);
    }

    deleteLinkedList(head);

    return 0;
}
```

Problem Statement 2

Daniel has a singly linked list and wants to delete the middle node from the list. If there are two middle nodes (in the case of an even number of nodes), the second one should be considered the middle node and deleted. Otherwise, the single middle node is deleted. Write a program to help Daniel delete this middle node.

For example, if the given linked list is 1->2->3->4->5 then the linked list should be modified to 1->2->4->5.

For example, if the given linked list is 1->2->3->4->5->6, then it should be modified to 1->2->3->5->6.

Company Tags: CTS

Input format :

The first line of input consists of an integer **n**, representing the number of elements in the linked list.

The second line contains **n** space-separated integers, each representing an element of the linked list.

Output format :

The first line of output displays "Original Linked List: " followed by the elements of the linked list before deletion.

The second line displays "Updated Linked List: " followed by the elements of the linked list after deletion of the middle node.

Refer to the sample output for format specifications.

Code constraints :

In this scenario, the test cases fall under the following constraints:

1 ≤ n ≤ 100

-50000 ≤ values of nodes ≤ 50000

Sample test cases :

Input 1 :

5

1 2 3 4 5

Output 1 :

Original Linked List: 1 2 3 4 5

Updated Linked List: 1 2 4 5

Input 2 :

6

1 2 3 4 5 6

Output 2 :

Original Linked List: 1 2 3 4 5 6

Updated Linked List: 1 2 3 5 6

```
// You are using GCC

#include <iostream>

using namespace std;
```

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

void insertNode(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;

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

void deleteMiddleNode(Node* head) {
    if(head == nullptr || head->next == nullptr){
        delete head;
        head = nullptr;
        return;
    }

    Node* temp = head;
    int count = 0;
    while(temp != NULL){
        temp = temp->next;
        count++;
    }

    int mid = count/2 ;
    //int pos = 1;

    temp = head;
    Node* prev = nullptr;

    for(int i=0; i< mid; i++){
        prev = temp;
        temp = temp->next;
    }

    prev->next = temp->next;
    delete temp;
}

void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    int size;
    cin >> size;
    for (int i = 0; i < size; i++) {
        int value;
        cin >> value;
        insertNode(&head, value);
    }

    cout << "Original Linked List: ";
    displayList(head);

    deleteMiddleNode(head);

    cout << "Updated Linked List: ";
    displayList(head);

    Node* temp = head;
    while (head != nullptr) {
        head = head->next;
        delete temp;
        temp = head;
    }

    return 0;
}
```

Problem Statement 3

Elsa is organizing a sequence of data stored in a singly linked list and aims to streamline the list by removing nodes positioned at even indices. The linked list nodes are inserted at the end by getting the input from the user.

Your task is to implement a program that reads the list data, removes the nodes at even positions, and outputs the modified list.

Company Tags: TCS

Input format :

The first line of input consists of an integer **n**, representing the number of elements in the linked list.

The second line contains **n** space-separated, each representing an element of the linked list.

Output format :

The first line of output displays "Original Linked List: " followed by the elements of the linked list before deletion.

The second line displays "Final Linked List: " followed by the elements of the linked list after the deletion of the even-positioned nodes.

Refer to the sample output for format specifications.

Code constraints :

In this scenario, the test cases fall under the following constraints:

1≤ n ≤ 100

-10000 ≤ elements ≤ 10000

Sample test cases :

Input 1 :

5

100 200 300 10000 9999

Output 1 :

Original Linked List: 100 200 300 10000 9999

Final Linked List: 100 300 9999

Input 2 :

3

-10000 -2000 -3000

Output 2 :

```
Original Linked List: -10000 -2000 -3000
Final Linked List: -10000 -3000
```

```
// You are using GCC
#include <iostream>
using namespace std;

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

void insertNode(Node** head, int data) {
   Node* newNode = new Node();
   newNode->data = data;
   newNode->next = nullptr;

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

void deleteEvenPositionNodes(Node** head) {
   if(head == nullptr) return;

   Node* curr = *head;
```

```
    Node* prev = nullptr;
    int pos = 1;

    while(curr != nullptr){
        if(pos%2 == 0){
            Node* toDelete = curr;
            prev->next = curr->next;
            curr = curr->next;
            delete toDelete;
        }else{
            prev = curr;
            curr = curr->next;
        }
        pos++;
    }

}
void displayLinkedList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    int size;

    cin >> size;
    for (int i = 0; i < size; i++) {
        int value;
        cin >> value;
        insertNode(&head, value);
    }

    cout << "Original Linked List: ";
    displayLinkedList(head);

    deleteEvenPositionNodes(&head);

    cout << "Final Linked List: ";
    displayLinkedList(head);

    Node* temp = head;
    while (head != nullptr) {
        head = head->next;
        delete temp;
        temp = head;
    }

    return 0;
}
```

Problem Statement 4

Madhav wants to remove all nodes with values greater than a specified value 'x' from a singly linked list. He will first insert new nodes at the end of the list and then remove the nodes with values exceeding 'x'.

Your task is to write a program that takes the size of the linked list, the elements of the linked list, and the value 'x' as input and outputs the modified list after the removal of the required nodes.

Company Tags: Capgemini

Input format :

The first line of input consists of an integer n, representing the number of elements in the linked list.

The second line contains n space-separated integers, each representing an element of the linked list.

The third line contains an integer x, the threshold value for node removal.

Output format :

The first line of output displays "Original Linked List: " followed by the elements of the linked list before deletion.

The second line displays "Modified Linked List: " followed by the elements of the linked list after deleting nodes with values greater than 'x'.

Refer to the sample output for format specifications.

Code constraints :

In this scenario, the test cases fall under the following constraints:

1 ≤ n ≤ 100

-10 ≤ elements ≤100

0 ≤ x ≤ 100

Sample test cases :

Input 1 :

5

8 7 5 3 2

5

Output 1 :

Original Linked List: 8 7 5 3 2

Modified Linked List: 5 3 2

Input 2 :

5

-8 7 -5 -3 -2

0

Output 2 :

```
Original Linked List: -8 7 -5 -3 -2

Modified Linked List: -8 -5 -3 -2
```

```
// You are using GCC
#include<iostream>
using namespace std;

struct Node{
   int data;
   Node* next;

   Node(int val){
      data = val;
      next = nullptr;
   }
};

void insertAtEnd(Node*&head, int val){
   Node* newNode = new Node(val);

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

```
Node* removeGreaterThanX(Node* head, int x){
    while(head != nullptr && head->data > x){
        Node* temp = head;
        head = head->next;
        delete temp;
    }

    Node* curr = head;

    while(curr != nullptr && curr->next != nullptr){
        if(curr->next->data > x){
            Node* temp = curr->next;
            curr->next = curr->next->next;
            delete temp;
        }else{
            curr = curr->next;
        }
    }

    return head;
}

void printLL(Node* head){
    Node* temp = head;
    while(temp != nullptr){
        cout<<temp->data<<" ";
        temp = temp->next;
    }
}

int main(){
    int n;
    cin>>n;
    Node* head = nullptr;

    int num;
    for(int i=0; i<n; i++){
        cin>>num;
        insertAtEnd(head, num);
    }

    int x;
    cin>>x;

    cout<<"Original Linked List: ";
    printLL(head);

    head = removeGreaterThanX(head, x);

    cout<<"Modified Linked List: ";
    printLL(head);

    return 0;
}
```

Problem Statement 5

Malik is working with a singly linked list and wants to remove the second last node from the list to meet specific requirements for his project. The linked list nodes are inserted at the end based on user input.

Malik needs a program that will efficiently identify and delete the second last node which takes the size of the linked list and its elements as input, performs the deletion operation, and outputs the modified list.

Company Tags: Wipro

Input format :

The first line of input consists of an integer **n**, representing the number of elements in the linked list.

The second line of input contains **n** space-separated integers, each representing an element of the linked list.

Output format :

The first line of output displays "Original Linked List: " followed by the elements of the linked list before deletion.

The second line of output displays "Updated Linked List: " followed by the elements of the linked list after deleting the second last node.

Refer to the sample output for formatting specifications.

Code constraints :

In this scenario, the test cases fall under the following constraints:

1 ≤ n ≤ 100

-50000 ≤ elements ≤ 50000

Sample test cases :

Input 1 :

```
5
1 2 3 4 5
```

Output 1 :

```
Original Linked List: 1 2 3 4 5
Updated Linked List: 1 2 3 5
```

Input 2 :

```
6
-50000 50000 7976 9887 4000 7765
```

Output 2 :

```
Original Linked List: -50000 50000 7976 9887 4000 7765
Updated Linked List: -50000 50000 7976 9887 7765
```

```
// You are using GCC
#include <iostream>
```

```
using namespace std;

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

void insertNode(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;

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

void deleteSecondLastNode(Node** head) {
    if(*head == nullptr || (*head)->next == nullptr){
        return;
    }

    if((*head)->next->next == nullptr){
        Node* temp = *head;
        *head = (*head)->next;
        delete temp;

        return;

    }


    Node* prev = nullptr;

    Node* current = *head;

    while(current->next->next != nullptr){

        prev = current;

        current = current->next;

    }


    prev->next = current->next;

    delete current;


}


void displayLinkedList(Node* head) {

    Node* temp = head;

    while (temp != nullptr) {

        cout << temp->data << " ";

        temp = temp->next;

    }

    cout << endl;

}


void deleteLinkedList(Node* head) {

    Node* temp;

    while (head != nullptr) {

        temp = head;

        head = head->next;

        delete temp;
    }
}

int main() {
    Node* head = nullptr;
    int size;

    cin >> size;

    for (int i = 0; i < size; i++) {
        int value;
        cin >> value;
        insertNode(&head, value);
    }

    cout << "Original Linked List: ";
    displayLinkedList(head);

    deleteSecondLastNode(&head);

    cout << "Updated Linked List: ";
    displayLinkedList(head);

    deleteLinkedList(head);

    return 0;
}
```

Problem Statement 6

Hema wants to create a program to delete nodes in a linked list that appear more than once. The program should take user input to construct a linked list, insert nodes at the end of the list, and then delete any duplicate nodes. Finally, it should display the updated linked list.

Input format :

The first line of input consists of an integer **n**, representing the number of elements in the linked list.

The second line contains **n** space-separated integers, each representing an element of the linked list.

Output format :

The first line of output displays "Original List: " followed by the elements of the linked list before removing duplicates.

The second line displays "Updated List: " followed by the elements of the linked list after removing duplicate nodes.

Refer to the sample output for format specifications.

Code constraints :

In this scenario, the test cases fall under the following constraints:

1 ≤ n ≤ 100

-10000 ≤ elements ≤ 10000

Sample test cases :

Input 1 :

5

50 60 70 80 50

Output 1 :

Original List: 50 60 70 80 50

Updated List: 50 60 70 80

Input 2 :

6

-10000 -10000 -10000 10000 10000 10000

Output 2 :

Original List: -10000 -10000 -10000 10000 10000 10000

Updated List: -10000 10000

Input 3 :

3

```
4 5 6
```

Output 3 :

```
Original List: 4 5 6

Updated List: 4 5 6
```

```cpp
// You are using GCC
#include<iostream>
#include<unordered_set>
using namespace std;

struct Node{
  int data;
  Node* next;

  Node(int val){
    data = val;
    next = nullptr;
  }
};

void insertAtEnd(Node*&head, int val){
  Node* newNode = new Node(val);

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

```
}

void removeDuplicates(Node*&head){
    if(!head) return;

    unordered_set<int> seen;
    Node* current = head;
    Node* prev = nullptr;

    while(current){
        if(seen.find(current->data) != seen.end()){
            prev->next = current->next;
            delete current;
            current = prev->next;
        }else{
            seen.insert(current->data);
            prev = current;
            current = current->next;
        }
    }
}

void printLL(Node* head){
    Node* temp = head;
    while(temp != NULL){
        cout<<temp->data<<" ";
        temp = temp->next;
    }
}

int main() {
    int n;
    cin>>n;
    Node* head = nullptr;

    for(int i=0; i<n; i++){
        int num;
        cin>>num;
        insertAtEnd(head, num);
    }

    cout<<"Original List: ";
    printLL(head);
    cout<<endl;

    removeDuplicates(head);

    cout<<"Updated List: ";
    printLL(head);


    return 0;
}
```

Problem Statement 7

Tim has a singly linked list and needs to remove every K-th node from the list. The task is to delete nodes that are positioned at every K-th place in the list, starting from the head of the list. Your program should read the list data, remove every Kth node, and then display the modified list.

Input format :

The first line of input consists of an integer n, representing the number of elements in the linked list.

The second line contains n space-separated integers, each representing an element of the linked list.

The third line contains an integer K, the position of nodes to be deleted.

Output format :

The first line of output displays the elements of the linked list before deletion separated by space.

The second line displays the elements of the linked list after deleting every K th node separated by space.

Refer to the sample output for formatting specifications.

Code constraints :

In this scenario, the test cases fall under the following constraints:

1 ≤ n ≤ 25

10 ≤ elements ≤ 50

1 ≤ K ≤ 10

Sample test cases :

Input 1 :

```
5
1 2 3 4 6
2
```

Output 1 :

```
1 2 3 4 6
1 3 6
```

Input 2 :

```
3
-50000 50000 1000
3
```

Output 2 :

```
-50000 50000 1000
-50000 50000
```

```
// You are using GCC
#include<iostream>
using namespace std;

struct Node{
    int data;

    Node* next;


    Node(int val){

        data = val;

        next = nullptr;

    }

};


void insertAtEnd(Node*&head, int val){

    Node* newNode = new Node(val);


    if(head == nullptr){

        head = newNode;

    }else{

        Node* temp = head;

        while(temp->next != nullptr){

            temp = temp->next;

        }

        temp->next = newNode;

    }

}


void deleteEveryKthNode(Node*& head, int K){

    if(K <= 0 || !head) return;


    if(K == 1){

        while(head){

            Node* temp = head;

            head = head->next;

            delete temp;
        }
        return;
    }

    Node* current = head;
    int count = 1;

    while(current && current->next){
        if((count+1)%K == 0){
            Node* temp = current->next;
            current->next = current->next->next;
            delete temp;
        }else{
            current = current->next;
        }
        count++;
    }
}


void printLL(Node* head){
    Node* temp = head;
    while(temp != nullptr){
        cout<<temp->data<<" ";
        temp = temp->next;
    }
}

int main() {
    int n;
    cin>>n;
```

```
  Node* head = nullptr;

  int num;
  for(int i=0; i<n; i++){
    cin>>num;
    insertAtEnd(head,num);
  }

  printLL(head);
  cout<<endl;

  int k;
  cin>>k;

  deleteEveryKthNode(head, k);

  printLL(head);

  return 0;
}
```

Problem Statement 8

Dharun is working on a program to manipulate linked lists. He wants to write a function that takes two linked lists as input, inserts nodes at the end, and deletes all the nodes from the first list that also appear in the second list.

Dharun needs your help implementing this function. The function should take two linked lists, **list1,** and **list2,** as input, where each list is represented by its head node.

Input format :

The first line contains an integer n, denoting the number of nodes in list1.

The second line contains n space-separated integers, representing the values of the nodes in list1.

The third line contains an integer m, denoting the number of nodes in list2.

The fourth line contains m space-separated integers, representing the values of the nodes in list2.

Output format :

The first line of output displays "Before deletion: " followed by the elements of the first linked list before the deletion, separated by a space.

The second line of output displays "After deletion: " followed by the elements of the first linked list after the deletion, separated by a space.

If all elements in the first linked list are the same after deletion, the third line displays, "Elements in both lists are same".

Refer to the sample output for formatting specifications.

Code constraints :

In this scenario, the test cases fall under the following constraints:

1 ≤ n,m ≤ 100

-5000 ≤ values of nodes ≤ 5000

Sample test cases :

Input 1 :

5

2 3 4 5 1

5

1 6 2 3 8

Output 1 :

Before deletion: 2 3 4 5 1

After deletion: 4 5

Input 2 :

5

1 2 3 4 5

5

1 2 3 4 5

Output 2 :

Before deletion: 1 2 3 4 5

After deletion:

Elements in both lists are same

```
// You are using GCC
#include <iostream>
#include<unordered_set>
using namespace std;

struct Node {
  int data;
  Node* next;

  Node(int val){
    data = val;
    next = nullptr;
  }
};

void insertNode(Node** head, int data) {
  Node* newNode = new Node(data);

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

```
void deleteNodesInSecondList(Node** head1, Node* head2) {
   unordered_set<int> values;
   Node* temp = head2;

   while(temp){
      values.insert(temp->data);
      temp = temp->next;
   }

   Node* dummy = new Node(0);
   dummy->next = *head1;
   Node* prev = dummy;
   Node* curr = *head1;

   while(curr){
      if(values.find(curr->data) != values.end()){
      prev->next = curr->next;
      delete curr;
      curr=prev->next;
      }else{
         prev = curr;
         curr = curr->next;
      }
   }

   *head1 = dummy->next;
   delete dummy;

}

void displayLinkedList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

void deleteLinkedList(Node* head) {
    Node* temp;
    while (head != nullptr) {
        temp = head;
        head = head->next;
        delete temp;
    }
}

bool areAllElementsSame(Node* head) {
    return head == nullptr;

}

int main() {
    Node* first = nullptr;
    Node* second = nullptr;
    int size1, size2;

    cin >> size1;

    for (int i = 0; i < size1; i++) {
        int value;
        cin >> value;

        insertNode(&first, value);

    }


    cin >> size2;


    for (int i = 0; i < size2; i++) {

        int value;

        cin >> value;

        insertNode(&second, value);

    }


    cout << "Before deletion: ";

    displayLinkedList(first);


    deleteNodesInSecondList(&first, second);


    cout << "After deletion: ";

    displayLinkedList(first);


    if (areAllElementsSame(first)) {

        cout << "Elements in both lists are same";

    }


    deleteLinkedList(first);

    deleteLinkedList(second);


    return 0;

}
```

Problem Statement 9

You need to create a program that manages a singly linked list of integers. Initially, read an integer **n** indicating the number of nodes to create. Populate the linked list with **n** integers provided as input. Finally, remove the first node (head) of the list and print the updated list.

Input format :

The first line of input consists of an integer **n**, representing the number of nodes.

The second line consists of **n** space-separated integers, representing the values of each node in the linked list.

Output format :

The output prints the space-separated values of the linked list after deleting the first node.

Refer to the sample output for formatting specifications.

Code constraints :

The given test cases fall under the following constraints:

$1 \leq n \leq 20$

$1 \leq$ node value $\leq 10^5$

Sample test cases :

Input 1 :

```
5
2 3 6 9 8
```

Output 1 :

```
3 6 9 8
```

Input 2 :

```
6
45 87 80 16 63 15
```

Output 2 :

```
87 80 16 63 15
```

```
// You are using GCC
#include<iostream>
using namespace std;

struct Node{
    int data;
    Node* next;

    Node(int val){
        data = val;
        next = nullptr;
    }
};

void insertAtEnd(Node*&head, int val){
    Node* newNode = new Node(val);

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

void removeFirstNode(Node*&head){
    Node* temp = head;
    head = head->next;
    delete temp;
}

void printLL(Node* head){
    Node* temp = head;
    while(temp != nullptr){
        cout<<temp->data<<" ";
        temp = temp->next;
    }
}

int main() {
    int n;
    cin>>n;
    Node* head = nullptr;

    for(int i=0; i<n; i++){
        int num;
        cin>>num;
        insertAtEnd(head, num);
    }

    removeFirstNode(head);

    printLL(head);

    return 0;
}
```

Problem Statement 10

Your task is to write a program that takes input for the number of elements in a singly linked list and the corresponding string values for each element. Based on this input, your program should create a linked list and then delete the alternate nodes from it.

Input format :

The first line contains an integer **n,** the number of elements in the linked list.

The second line contains **n** space-separated strings representing the elements of the linked list.

Output format :

If the linked list is empty, output "List is empty".

If the linked list is not empty, output the following:

1. The first line should display the elements of the original linked list, separated by a space.
2. The second line should display the elements of the linked list after deleting the alternate nodes, separated by a space.

Refer to the sample output for the exact text and format.

Code constraints :

The given test cases fall under the following constraints:

0 ≤ n ≤ 10

Sample test cases :

Input 1 :

```
2
Apple Banana
```

Output 1 :

```
Linked list data: Apple Banana
After deleting alternate node:Apple
```

Input 2 :

```
5
Red Green Blue Yellow Orange
```

Output 2 :

```
Linked list data: Red Green Blue Yellow Orange
After deleting alternate node:Red Blue Orange
```

Input 3 :

```
0
```

Output 3 :

```
List is empty
```

```
// You are using GCC
#include<iostream>
#include<string>
using namespace std;
```

```
struct Node{
   string data;
   Node* next;

   Node(string val){
      data = val;
      next = nullptr;
   }
};

void insertAtEnd(Node*&head, string val){
   Node* newNode = new Node(val);

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

void deleteAlternateNode(Node*&head){
   Node* curr = head;
   while(curr != nullptr && curr->next != nullptr){
      Node* temp = curr->next;
      curr->next = temp->next;
      delete temp;
```

```
        curr = curr->next;
    }
}

void printLL(Node* head){
    Node* temp = head;
    while(temp != nullptr){
        cout<<temp->data<<" ";
        temp = temp->next;
    }
    cout<<endl;
}

int main() {
    int n;
    cin>>n;
    Node* head = nullptr;

    if(n == 0){
        cout<<"List is empty"<<endl;
        return 0;
    }

    for(int i=0; i<n; i++){
        string color;
        cin>>color;
        insertAtEnd(head, color);
    }

    cout<<"Linked list data: ";
    printLL(head);
    deleteAlternateNode(head);


    cout<<"After deleting alternate node:";
    printLL(head);



    return 0;
}
```

Problem Statement 11

Udhay wants to implement a program that creates a singly linked list, deletes a node at a specified position, and prints the updated list. Your task as his friend is to help him complete the program.

Input format :

The first line of input consists of an integer **n**, the number of nodes in the linked list.

The second line consists of **n** integers.

The third line consists of an integer **x**, representing the position(1-based) of the node to be deleted.

Output format :

The output prints the linked list after deleting the node at position x.

Refer to the sample output for formatting specifications.

Code constraints :

The given test cases fall under the following constraints:

1 ≤ n ≤ 300

$-10^9 \leq$ value of each node $\leq 10^9$

x ≤ n

Sample test cases :

Input 1 :

```
7
72 53 29 42 73 79 68
7
```

Output 1 :

```
72 53 29 42 73 79
```

Input 2 :

```
6
-14 25 37 -49 54 61
4
```

Output 2 :

```
-14 25 37 54 61
```

```
// You are using GCC
#include<iostream>
using namespace std;

struct Node{
    int data;
    Node* next;

    Node(int val){
        data = val;
        next = nullptr;
    }
};

void insertAtEnd(Node*&head, int val){
    Node* newNode = new Node(val);

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

void deleteMthNode(Node*&head, int m){

    if(m == 1){
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    Node* curr = head;
    for(int i=1; curr != nullptr &&  i<m-1; i++){
        curr = curr->next;
    }

    if(curr == nullptr || curr->next == nullptr) return;

    Node* temp = curr->next;
    curr->next = temp->next;
    delete temp;

}

void printLL(Node* head){
    Node* temp = head;
    while(temp != nullptr){
        cout<<temp->data<<" ";
        temp = temp->next;
    }
}

int main() {
    int n;
    cin>>n;
    Node* head = nullptr;

    for(int i=0; i<n; i++){
        int num;
        cin>>num;
        insertAtEnd(head, num);
    }

    int m;
    cin>>m;

    deleteMthNode(head, m);

    printLL(head);

    return 0;
}
```

Problem Statement 12

Imagine you are given an integer **n** representing the number of nodes in a singly linked list.

Your task is to implement a program that creates a singly linked list with the given number of nodes and string values and then deletes the last node from the list. Finally, you need to print the contents of the modified linked list.

Input format :

The first line of input consists of an integer **n,** representing the number of nodes in the singly linked list.

The next **n** lines of input consist of **n** strings, where each line represents the string value of a node in the linked list.

Output format :

The output should print the elements of the singly linked list after deleting the last node. The elements should be separated by space.

Refer to the sample output for formatting specifications.

Code constraints :

The given test cases fall under the following constraints:

1 ≤ n ≤ 20

Sample test cases :

Input 1 :

```
4
Hello
World
I
Am
```

Output 1 :

```
Hello World I
```

Input 2 :

```
3
Hello
Hello
Hello
```

Output 2 :

```
Hello Hello
```

```
// You are using GCC
```

```
#include <iostream>

#include <string>

using namespace std;


struct Node {

    string data;

    Node* next;

};


void append(Node** head, const string& data) {

    Node* tmp = new Node;

    tmp->data = data;

    tmp->next = nullptr;

    if (*head == nullptr) {

        *head = tmp;

    } else {

        Node* curr = *head;

        while (curr->next != nullptr) {

            curr = curr->next;

        }

        curr->next = tmp;

    }

}


void print(Node* head) {

    if (head == nullptr) {

        return;

    }

    Node* curr = head;

    cout << curr->data << " ";

    print(curr->next);
```

```
}

void delete_last(Node** head) {
  if(*head == nullptr) return;

  if((*head)->next == nullptr){
    delete *head;
    *head = nullptr;
    return;
  }

  Node* temp = *head;
  while(temp->next->next != nullptr){
    temp = temp->next;
  }

  delete temp->next;
  temp->next = nullptr;

}

int main() {
  int num_of_nodes;
  string data;
  cin >> num_of_nodes;
  Node* myList = nullptr;
  for (int i = 0; i < num_of_nodes; i++) {
    cin >> data;
    append(&myList, data);
  }
  delete_last(&myList);
```

```
    print(myList);

    return 0;

}
```