Problem Statement 1

Lisa is organizing a list of attendees for two consecutive events. She wants to split a list of attendee names into two halves for the two events. You are tasked with helping Lisa by creating a program that handles this task efficiently.

Write a program to manage a two-way linked list of attendee names to split it into two halves.

Company Tags: Microsoft

Input format :

The first line of input consists of a single integer **N,** representing the number of attendees.

The following **N** lines contain the names of the attendees, one name per line.

Output format :

The first line of output displays the names of the attendees assigned to the first event.

The second line displays the names of the attendees assigned to the second event.

Refer to the sample output for formatting specifications.

Code constraints :

1 ≤ N ≤ 20

Sample test cases :

Input 1 :

```
4
John
Emma
Michael
Sophia
```

Output 1 :

```
John Emma
Michael Sophia
```

Input 2 :

```
5
```

```
Alice
Bob
Claire
David
Emily
```

Output 2 :

```
Alice Bob Claire
David Emily
```

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

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

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

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

  if(head == nullptr){
    head = tail = newNode;
  }else{
    tail->next = newNode;
    newNode->prev = tail;
    tail = newNode;
  }
}

void printFirstHalf(Node* head, int mid){
  int count =0;
  Node* temp = head;

  while(temp != nullptr && count<mid){
    cout<<temp->data<<" ";
    temp = temp->next;
    count++;
  }

  cout<<endl;
}

void printSecondHalf(Node * head, int mid){
  Node* temp = head;
  int count =0;

  while(temp != nullptr && count<mid){
    temp = temp->next;
    count++;
  }

  while( temp != nullptr){
    cout<<temp->data<<" ";
    temp = temp->next;
  }
  cout<<endl;
```

```
}

int main(){
  int n;
  cin>>n;
  cin.ignore();

  Node* head = nullptr;
  Node* tail = nullptr;

  for(int i=0; i<n; i++){
    string name;
    getline(cin, name);
    insertAtEnd(head, tail, name);
  }

  int mid = (n+1)/2;

  printFirstHalf(head, mid);
  printSecondHalf(head, mid);

  return 0;
}
```

Problem Statement 2

A healthcare provider wants to analyze a list of patient records to determine if the sequence of records is a palindrome. A sequence is considered a palindrome if it reads the same forward and backwards.

You need to write a program using a two-way linked list to help the provider check if the given list of medical records is a palindrome.

Company Tags: Capgemini

Input format :

The first line contains an integer **n,** representing the number of records.

The second line contains **n** space-separated integers, each representing a medical record identifier.

Output format :

If the list of records is a palindrome, print "The patient's medical history is a palindrome".

If it is not a palindrome, print "The patient's medical history is not a palindrome".

Refer to the sample output for formatting specifications.

Code constraints :

1 ≤ n ≤ 20

1 ≤ identifier ≤ 100

Sample test cases :

Input 1 :

```
5

1 2 3 2 1
```

Output 1 :

```
The patient's medical history is a palindrome
```

Input 2 :

```
5

1 2 3 4 5
```

Output 2 :

```
The patient's medical history is not a palindrome
```

```
#include<iostream>

#include<vector>

using namespace std;


struct Node{

int data;

Node* prev;

Node* next;
```

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

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

  if(head == nullptr){
    head = tail = newNode;
  }else{
    tail->next = newNode;
    newNode->prev = tail;
    tail = newNode;
  }
}

Node* reverseLL(Node* head){
  Node* prev = nullptr;
  Node* curr = head;
  Node* next = nullptr;

  while(curr){
    next = curr->next;
    curr->next = prev;
    prev = curr;
    curr = next;
  }
  return prev;
}

bool checkPalindrome(Node* head){
  if(head == nullptr || head->next == nullptr){
    return true;
  }

  Node* slow = head;
  Node* fast = head;

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

  Node* newHead = reverseLL(slow);

  Node* first = head;
  Node* second = newHead;

  while(second != nullptr){
    if(first->data != second->data){
      return false;
    }

    first = first->next;
    second = second->next;
  }

  return true;
```

```
}

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

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

 bool isPalindrome = checkPalindrome(head);

 if(isPalindrome){
  cout<<"The patient's medical history is a palindrome";
 }else{
  cout<<"The patient's medical history is not a palindrome";
 }

 return 0;
}
```

Problem Statement 3

In a customer service system, interactions between users and the service team are recorded in a log. Each interaction is represented by a unique identifier. Your task is to analyze the log and determine the number of unique user interactions.

Utilize a two-way linked list to solve the program.

Company Tags: Amazon

Input format :

The first line of input consists of an integer **N,** representing the number of interactions.

This is followed by **N** lines of strings, each representing an interaction identifier.

Output format :

The output prints a single line containing the number of unique user interactions.

Refer to the sample output for formatting specifications.

Code constraints :

1 ≤ N ≤ 20

Sample test cases :

Input 1 :

```
3
ABC123
DEF456
GHI789
```

Output 1 :

```
Number of unique user interactions: 3
```

Input 2 :

```
5
ABC123
DEF456
ABC123
JKL987
DEF456
```

Output 2 :

```
Number of unique user interactions: 3
```

Input 3 :

```
2
XYZ789
```

```
XYZ789
```

Output 3 :

```
Number of unique user interactions: 1
```

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

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

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

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

  if(head == nullptr){
    head = tail = newNode;
  }else{
    tail->next = newNode;
    newNode->prev = tail;
    tail = newNode;
  }
}

int uniqueData(Node* head){
  Node* curr = head;
  int count =0;

  while(curr){
    Node* checker = head;
    bool seen = false;

    while(checker != curr){
      if(curr->data == checker->data){
        seen = true;
        break;
      }
      checker = checker->next;
    }

    if(!seen){
      count++;
    }
    curr = curr->next;
  }

  return count;
}

int main(){
  int n;
  cin>>n;
  cin.ignore();

  Node* head = nullptr;
  Node* tail = nullptr;
  for(int i=0; i<n; i++){
    string data;
    getline(cin, data);
    insertAtEnd(head, tail, data);
  }

  cout<<"The number of unique user interactions: "<<uniqueData(head);

  return 0;
}
```

Problem Statement 4

Tom manages a library system where books are categorized by their ID numbers in ascending order. He wants to maintain the order of books using a two-way linked list. Each time a new book is added, it should be placed in the correct position to keep the list sorted. Tom also wants to insert a final book after all the initial books have been added.

Write a program to help Tom maintain the sorted order of books in the library.

Company Tags: Capgemini

Input format :

The first line of input contains an integer **n** representing the number of initial books.

The second line consists of **n** integers representing the ID of each book to be added.

The third line contains an integer representing the ID of the final book to be added.

Output format :

The output prints the sorted order of book IDs after all books have been added.

Refer to the sample output for formatting specifications.

Code constraints :

1 ≤ n ≤ 25

1 ≤ book IDs ≤ 100

Sample test cases :

Input 1 :

3

2 4 1

3

Output 1 :

1 2 3 4

Input 2 :

1

5

2

Output 2 :

2 5

```
#include<iostream>

using namespace std;


struct Node{

int data;

Node* next;

Node* prev;


Node(int val){

  data = val;

  next = nullptr;

  prev = nullptr;

}

};


void insertSorted(Node*& head, Node* & tail, int num){

  Node* newNode = new Node(num);
  if(!head){
    head = tail = newNode;
    return;
  }

  if(num < head->data){
    newNode->next = head;
    head->prev = newNode;
    head = newNode;
    return;
  }

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

  newNode->next = temp->next;
  newNode->prev = temp;
  temp->next = newNode;

  if(newNode->next != nullptr){
    newNode->next->prev = newNode;
  }else{
    tail = newNode;
  }
}

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

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

  int num;
  Node* head = nullptr;
  Node* tail = nullptr;

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

  int m;
  cin>>m;
  insertSorted(head, tail, m);

  printLL(head);

  return 0;
}
```

Problem Statement 5

Ashok, a retail store manager wants to keep track of the items in inventory using a two-way linked list. Each item has an ID, name, quantity, and price. The manager needs to be able to insert new items at the beginning of the list and search for an item by its ID to check its presence.

Your task is to create a program to help Ashok manage the inventory and perform searches efficiently.

Company Tags: Amazon

Input format :

The first line of input contains an integer **n** representing the number of items to be added to the inventory.

The next **n** lines each contain:

1. An integer representing the item ID.
2. A string representing the item name.
3. An integer representing the item quantity.
4. A double value representing the item price.

The last line contains an integer representing the ID of the item to be searched.

Output format :

Print a message indicating whether the item with the given ID is present in the list.

Refer to the sample output for the exact format.

Code constraints :

1 ≤ n ≤ 20

Sample test cases :

Input 1 :

```
3
101
Pen
10
5.99
102
Notebook
5
12.99
103
```

```
Pencil
20
1.99
101
```

Output 1 :

```
Item with ID 101 is present in the list.
```

Input 2 :

```
2
201
Chair
8
49.99
202
Table
3
99.99
203
```

Output 2 :

```
Item with ID 203 is not found in the list.
```

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

struct Node{
int id, qty;
string name;
double price;
Node* next;
Node* prev;

Node(int id1, int qty1, string name1, double price1){
  id = id1;
  qty = qty1;
  name = name1;
  price = price1;
  next = nullptr;
  prev = nullptr;
}
};

void insertAtEnd(Node*& head, Node*& tail, int id, int qty, string name, double price){
  Node* newNode = new Node(id, qty, name, price);

  if(head == nullptr){
    head = tail = newNode;
  }else{
    tail->next = newNode;
    newNode->prev = tail;
    tail = newNode;
  }
}

bool searchByID(Node* head, int searchID){
  Node* temp = head;
  while(temp){
    if(temp->id == searchID){
      return true;
    }
    temp = temp->next;
  }
  return false;
}
```

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

 Node* head = nullptr;
 Node* tail = nullptr;

 for(int i=0; i<n; i++){
  int id, qty;
  string name;
  double price;

  cin>>id;
  cin.ignore();
  getline(cin, name);
  cin>>qty>>price;

  insertAtEnd(head, tail, id, qty, name, price);
 }

 int searchID;
 cin>>searchID;

 bool IDexists = searchByID(head, searchID);

 if(IDexists){
  cout<<"Item with ID "<<searchID<<" is present in the list.";
 }else{
  cout<<"Item with ID "<<searchID<<" is not found in the list.";
 }
```

```
  return 0;
}
```

Problem Statement 6

Sara is organizing a fun game for her friends where they have to rotate a sequence of characters in a specific way. She wants to use a two-way linked list to perform this rotation efficiently.

Write a program to help Sara by implementing the following operations:

1. Insert characters at the end of a two-way linked list.
2. Rotate the linked list by a given number of positions to the left.
3. Display the rotated linked list.

Company Tags: Infosys

Input format :

The first line of input contains an integer **n,** representing the number of characters to be added to the list.

The second line consists of **n** characters, representing the data to be added to the list.

The third line contains an integer **k,** representing the number of positions by which the list should be rotated.

Output format :

The output prints the characters in the two-way linked list after rotating it by k positions.

Refer to the sample output for formatting specifications.

Code constraints :

1 ≤ n ≤ 15

Sample test cases :

Input 1 :

```
5
a b c d e
2
```

Output 1 :

```
c d e a b
```

Input 2 :

```
8
a b c d e f g h
4
```

Output 2 :

```
e f g h a b c d
```

Input 3 :

```
2
E T
0
```

Output 3 :

```
E T
```

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

struct Node{
char ch;
Node* next;
Node* prev;

Node(char ch1){
  ch = ch1;
  next = nullptr;
  prev = nullptr;
}
};

void insertAtEnd(Node*& head,Node* &tail, char ch){
  Node* newNode = new Node(ch);

  if(head == nullptr){
    head = tail = newNode;
  }else{
    tail->next = newNode;
    newNode->prev = tail;
    tail = newNode;
  }
}

void rotateLeft(Node*& head, Node* tail, int k, int n){
  if( k == 0 || head == nullptr || k%n == 0) return;

  k = k%n;
  Node* curr = head;

  for(int i=0; i<k; i++){
    curr = curr->next;
  }

  tail->next = head;
  head->prev = tail;

  head = curr;
  tail = curr->prev;

  tail->next = nullptr;
  head->prev = nullptr;
}

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

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

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

  int k;
  cin>>k;

  rotateLeft(head, tail, k, n);

  printLL(head);

  return 0;
}
```

Problem Statement 7

Create a program to delete all the nodes with even values in a two-way linked list and print only the nodes with odd values. The program should take user input for the number of nodes and the values of each node and then perform the deletion.

Input format :

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

The second line contains **n** integers, representing the data to be added to the list.

Output format :

The output prints the values of the nodes that have odd values after deleting all nodes with even values.

Refer to the sample output for formatting specifications.

Code constraints :

1 ≤ n ≤ 15

1 ≤ data ≤ 100

Sample test cases :

Input 1 :

```
3
47 78 32
```

Output 1 :

```
47
```

Input 2 :

```
8
23 76 70 56 97 58 25 74
```

Output 2 :

```
23 97 25
```

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

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

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

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

  if(head == nullptr){
    head = tail = newNode;
  }else{
    tail->next = newNode;
    newNode->prev = tail;
    tail = newNode;
  }
}

void deleteEvenNodes(Node*& head){
  Node* temp = head;

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

    if(temp->data % 2 == 0){
      if(temp == head) {
        head = temp->next;
        if(head) head->prev = nullptr;
        delete temp;
      }else{
        temp->prev->next = temp->next;
        if (temp->next) {
          temp->next->prev = temp->prev;
        }
        delete temp;
      }
    }
    temp = nextNode;
  }
}

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

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

  Node* head = NULL;
  Node* tail = NULL;

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

```
 }

 deleteEvenNodes(head);

 printLL(head);
 return 0;
}
```

Problem Statement 8

Jenna is organizing her bookshelf and wants to track the order of books she places on the shelf. She decides to use a two-way linked list for this purpose. Jenna can insert a book at the beginning of the shelf. Once all books are placed, she wants to reverse the order of the books on the shelf.

Write a program to help Jenna manage her bookshelf.

Input format :

The first line of input contains an integer **n,** representing the number of books Jenna places on the shelf.

The second line contains **n** space-separated integers, representing the book IDs.

Output format :

The first line of output prints the original list.

The second line prints the reversed list.

Refer to the sample output for formatting specifications.

Code constraints :

1 ≤ n ≤ 25

1 ≤ books ID ≤ 100

Sample test cases :

Input 1 :

```
3

12 34 65
```

Output 1 :

```
Original List: 65 34 12
```

Reversed List: 12 34 65

Input 2 :

6

12 34 56 78 31 45

Output 2 :

Original List: 45 31 78 56 34 12

Reversed List: 12 34 56 78 31 45

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

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

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

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

  if(head == nullptr){
    head = tail = newNode;
  }else{
    newNode->next = head;
    head->prev = newNode;
    head = newNode;
  }
}

void printForward(Node* head){
  Node* temp = head;

  while(temp != nullptr){
    cout<<temp->data<<" ";
    temp = temp->next;
  }

  cout<<endl;
}

void printBackward(Node* tail){
  Node* temp = tail;

  while(temp != nullptr){
    cout<<temp->data<<" ";
    temp = temp->prev;
  }

  cout<<endl;
}

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

  Node* head = nullptr;
  Node* tail = nullptr;
```

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

 cout<<"Original List: ";
 printForward(head);

 cout<<"Reversed List: ";
 printBackward(tail);

 return 0;
}
```

Problem Statement 9

Vijay is tasked with implementing a system that maintains a record of student scores in a course. The system should allow for the insertion of scores at the front of a two-way linked list and display the updated list after each insertion.

Your task is to assist Vijay in designing the system.

Company Tags: Capgemini

Input format :

The input consists of a series of integer scores to be inserted into the list. The input is terminated when -1 is entered.

Output format :

After each score insertion, the program should display the updated list of scores.

The scores should be displayed in the order they were inserted.

Refer to the sample output for formatting specifications.

Code constraints :

The given test cases fall under the following constraints:

1 ≤ scores ≤ 100

Sample test cases :

Input 1 :

```
81 75 94 -1
```

Output 1 :

```
81

75 81

94 75 81
```

Input 2 :

```
95 83 72 98 -1
```

Output 2 :

```
95

83 95

72 83 95

98 72 83 95
```

```
#include<iostream>

using namespace std;


struct Node{

int data;

Node* next;

Node* prev;


Node(int val){

  data = val;

  next = nullptr;

  prev = nullptr;

}

};
```

```
void insertAtBeginning(Node*& head, Node*& tail, int n){
  Node* newNode= new Node(n);

  if(head == nullptr){
    head = tail = newNode;
  }else{
    newNode->next = head;
    head->prev = newNode;
    head = newNode;
  }
}

void printLL(Node* head){
  Node* temp = head;

  while(temp != nullptr){
    cout<<temp->data<<" ";
    temp = temp->next;
  }
  cout<<endl;
}

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

  while(true){
    int n;
    cin>>n;
    if(n == -1){
      break;
    }
    insertAtBeginning(head, tail, n);
    printLL(head);
  }

  return 0;
}
```

Problem Statement 10

Deepak, your professor, has given you the task of developing a program for a management system that utilizes a two-way linked list to track and manage tasks. Each node in the linked list represents a task with a unique ID. The program should allow for inserting tasks at the front, deleting tasks from the front and end of the list, and displaying the updated list of tasks after each operation.

Implement a two-way linked list where elements can be inserted at the front and deleted from the front and end of the list.

Company Tags: Infosys

Input format :

The first line contains an integer **n**, representing the number of elements to insert.

The next line contains **n** space-separated integers representing the elements to be inserted at the front of the linked list.

Output format :

The output consists of three parts:

After inserting all the elements at the front, display the elements in the linked list.

After deleting the node at the front, display the updated linked list.

After deleting the node at the end, display the final linked list.

Refer to the sample output for formatting specifications.

Code constraints :

The given test cases fall under the following constraints:

3 ≤ n ≤ 20

-1000 ≤ elements ≤ 1000

Sample test cases :

Input 1 :

```
5

1 2 3 4 5
```

Output 1 :

```
5 4 3 2 1

4 3 2 1

4 3 2
```

Input 2 :

```
3

9 -8 7
```

Output 2 :

```
7 -8 9

-8 9

-8
```

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

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

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

```
void insertAtBeginning(Node*& head, Node*& tail, int num){
  Node* newNode = new Node(num);

  if(head == nullptr){
    head = tail = newNode;
  }else{
    newNode->next = head;
    head->prev = newNode;
    head = newNode;
  }
}

void deleteFromFront(Node*&head, Node*&tail){
  if(head == nullptr){
    return;
  }

  if(head == tail){
    delete head;
    head = tail = nullptr;
    return;
  }

  Node* del = head;
  head = head->next;
  head->prev = nullptr;
  delete del;
}

void deleteFromEnd(Node*&head, Node*&tail){
  if(tail == nullptr){
    return;
  }

  if(head == tail){
    delete tail;
    head = tail = nullptr;
    return;
  }

  Node* del = tail;
  tail = tail->prev;
  tail->next = nullptr;
  delete del;
}

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;
  Node* tail = nullptr;

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

  printLL(head);
  deleteFromFront(head, tail);
  printLL(head);
  deleteFromEnd(head, tail);
  printLL(head);

  return 0;
}
```

Problem Statement 11

Imagine you are given a two-way linked list.

You need to implement the following operations:

1. Insert an element at the end of the list.
2. Insert an element at a given position in the list.

Company Tags: TCS

Input format :

The input begins with an integer **n**, denoting the number of elements to be inserted at the end of the list.

This is followed by **n** space-separated integers representing the elements to be inserted at the end of the list.

Next, there are two integers **data** and **position** representing the element to be inserted and the position(1-based) at which it should be inserted in the list.

Output format :

The first line of output prints the linked list after inserting elements at the end.

The second line prints the linked list after inserting an element at the given position.

Refer to the sample output for the exact text and formatting specifications.

Code constraints :

1 ≤ n ≤ 100

1 ≤ elements ≤ 100

Sample test cases :

Input 1 :

```
1

10

2

1
```

Output 1 :

```
List after inserting at the end: 10

List after inserting at position 1: 2 10
```

Input 2 :

```
3

1 2 3

10

4
```

Output 2 :

```
List after inserting at the end: 1 2 3

List after inserting at position 4: 1 2 3 10
```

```
#include<iostream>

using namespace std;


struct Node{

int data;

Node* next;

Node* prev;


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

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

  if(head == nullptr){
    head = tail = newNode;
  }else{
    tail->next = newNode;
    newNode->prev = tail;
    tail = newNode;
  }
}

void insertAtPosition(Node*& head, Node*& tail, int val, int pos){
  Node* newNode = new Node(val);

  if(pos == 1){
    newNode->next = head;
    if(head != nullptr){
      head->prev = newNode;
      head = newNode;
    }
    if(tail == nullptr){
      tail = newNode;
    }
    return;
  }

  Node* temp = head;
  int count =1;

  while(temp && count< pos -1){
    temp = temp->next;
    count++;
  }

  if(temp == nullptr && temp->next == nullptr){
    insertAtEnd(head, tail, val);
    return;
  }

  newNode->next = temp->next;
  newNode->prev = temp;
  if(temp->next != nullptr){
    temp->next->prev = newNode;
  }
  temp->next = newNode;
}

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;
 Node* tail = nullptr;

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

 int data, pos;
 cin >> data >> pos;

 cout << "List after inserting at the end: ";
 printLL(head);

 insertAtPosition(head, tail, data, pos);

 cout << "List after inserting at position " << pos << ": ";
 printLL(head);

 return 0;
}
```

Problem Statement 12

Dinesh is learning linked lists and wants to create a program that manages a two-way linked list and performs the deletion of a node at a specified position. The program should allow the user to input the number of nodes, their values, and the position of the node to be deleted. After the deletion, the program should display the original list and the updated list, both in reverse order.

Guide Dinesh in completing the program.

Input format :

The input begins with an integer **n**, representing the number of nodes in the two-way linked list.

The next line contains **n** space-separated integers representing the data values of each node.

Finally, an integer **p** is provided, representing the position(1-based) of the node to be deleted.

Output format :

The program should output the two-way linked list before and after the deletion of the specified node.

The linked list should be displayed in reverse order as a space-separated list of integers.

Refer to the sample output for formatting specifications.

Code constraints :

The given test cases fall under the following constraints:

1 ≤ n ≤ 20

1 ≤ elements ≤ 1000

Sample test cases :

Input 1 :

```
4
16 24 39 47
1
```

Output 1 :

```
Before deletion: 47 39 24 16
After deletion: 39 24 16
```

Input 2 :

```
5
```

```
5 2 4 8 10
7
```

Output 2 :

```
Before deletion: 10 8 4 2 5
After deletion: 10 8 4 2 5
```

Input 3 :

```
8
12 34 76 91 28 31 94 57
7
```

Output 3 :

```
Before deletion: 57 94 31 28 91 76 34 12
After deletion: 57 94 31 28 91 76 12
```

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

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

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

void insertAtEnd(Node*& head, Node*& tail, int val) {
    Node* newNode = new Node(val);
    if (!head) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
}

void printReverse(Node* tail) {
    Node* temp = tail;
    while (temp) {
        cout << temp->data << " ";
        temp = temp->prev;
    }
    cout << endl;
}

int getOriginalPositionFromReverse(Node* tail, int revPos) {
    int length = 0;
    Node* temp = tail;
    while (temp) {
        length++;
        temp = temp->prev;
    }
    // Reverse position 1 means original position 'length'
    return length - revPos + 1;
}

void deleteAtPosition(Node*& head, Node*& tail, int pos) {
    if (pos <= 0) return;
```

```
Node* temp = head;
int count = 1;

while (temp && count < pos) {
    temp = temp->next;
    count++;
}

if (!temp) return;

// Single node
if (head == tail) {
    delete temp;
    head = tail = nullptr;
}
// Delete head
else if (temp == head) {
    head = head->next;
    head->prev = nullptr;
    delete temp;
}
// Delete tail
else if (temp == tail) {
    tail = tail->prev;
    tail->next = nullptr;
    delete temp;
}
// Middle
else {
    temp->prev->next = temp->next;
    temp->next->prev = temp->prev;
```

```
        delete temp;
    }
}

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

    Node* head = nullptr;
    Node* tail = nullptr;

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

    int revPos;
    cin >> revPos;

    cout << "Before deletion: ";
    printReverse(tail);

    int originalPos = getOriginalPositionFromReverse(tail, revPos);
    deleteAtPosition(head, tail, originalPos);

    cout << "After deletion: ";
    printReverse(tail);

    return 0;
}
```