# Week-5:

# Single Linked List

Q1. Write a program to create a single linked list. Include methods to insert elements at the beginning, end, a specific position of the list, and a specific element of the list, delete elements, search for a value, and display the contents of the list.

Code:

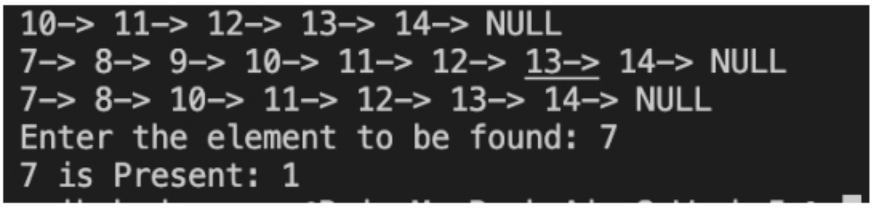
```
#include<iostream>
using namespace std;
 class
    Node{ pu
    blic : int
    data;
    Node* next;
    Node(int data){
        this->data = data;
        next = NULL;
    }
 };
  void insertAtHead(Node* &head , int data)
    { Node* newNode = new Node(data);
    newNode->next = head;
    head =newNode;
 }
  void insertAtTail(Node* &tail , int data){
    Node* newNode = new Node(data);
    tail->next = newNode;
    tail = newNode;
 }
 void insertAtPosition(Node* &head ,Node* &tail, int data,int
pos){
    if(pos==1)
        {insertAtHead(head,data
        ); return;
    }
    Node* temp = head;
    int count =1;
    while(count<pos-1){
        temp=temp->next;
        count++;
    }
    if(temp->next==NULL)
        {insertAtTail(tail ,
        data); return;
    }
     Node* newnode = new Node(data);
    newnode->next = temp->next; temp-
    >next =newnode;

}
void print(Node* head){
    Node* temp = head;
    while(temp->next!=NULL){
        cout<<temp->data<<"-> ";
        temp = temp->next;
    }cout<<"NULL";
    cout<<endl;
}
void deletenode(Node* &head, int pos ){
    if(pos==1){
        Node* temp = head;
        head= head->next;
        temp->next = NULL;
        delete temp;
    }
    Node* prev = NULL;
    Node* curr = head;
    int count =1;
    while(count < pos){
        prev =curr;
        curr= curr->next;
        count++;
    }
    prev->next = curr->next;
    curr->next= NULL;
    delete curr;
}
 bool searchElement(Node* head, int target){
    Node* temp = head;
    while(temp!=NULL){
        if(temp->data==target){

            return true ;
        }
        temp = temp->next;
    }
    return false;
 }
int main(){
    Node * node1 = new Node(10);
    Node* head =node1;
    Node* tail = node1;
    insertAtTail(tail,11);
    insertAtTail(tail,12);
    insertAtTail(tail,13);
    insertAtTail(tail,14);
    insertAtTail(tail,15);
    print(head);
    insertAtHead(head,9);
    insertAtHead(head,8);
    insertAtHead(head,7);
     print(head);
     deletenode(head,3);
    print(head);
     int k ;
    cout<<"Enter the element to be found: ";
    cin>>k;
    cout<<k<<" is Present: "<<searchElement(head,k)<<endl;

    return 0;
}
```

Output:



```
10-> 11-> 12-> 13-> 14-> NULL
7-> 8-> 9-> 10-> 11-> 12-> 13-> 14-> NULL
7-> 8-> 10-> 11-> 12-> 13-> 14-> NULL
Enter the element to be found: 7
7 is Present: 1
```

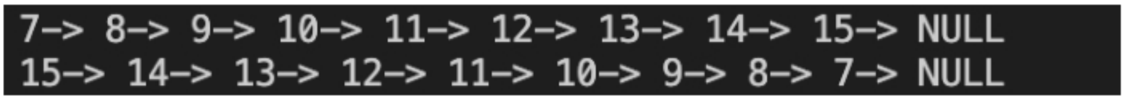
Q2. Write a function to reverse a single linked list in-place. Implement an iterative solution to reverse the list.

Code:

```
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node *next;
    Node(int data)
    {
        this->data = data;
        next = NULL;
    }
};
void insertAtHead(Node *&head, int data)
{
    Node *newNode = new Node(data);
    newNode->next = head;
    head = newNode;
}
void insertAtTail(Node *&tail, int data)
{
    Node *newNode = new Node(data);
    tail->next = newNode;
    tail = newNode;
}
void insertAtPosition(Node *&head, Node *&tail, int data, int
pos)
{
    if (pos == 1)
    {
        insertAtHead(head, data);
        return;
    }
    Node *temp = head;

    int count = 1;
    while (count < pos - 1)
    {
        temp = temp->next;
        count++;
    }
    if (temp->next == NULL)
    {
        insertAtTail(tail, data);
        return;
    }
    Node *newnode = new Node(data);
    newnode->next = temp->next;
    temp->next = newnode;
}
void print(Node *head)
{
    Node *temp = head;
    while (temp != NULL)
    {
        cout << temp->data << "-> ";
        temp = temp->next;
    }
    cout << "NULL";
    cout << endl;
}
int main()
{
    Node *node1 = new Node(10);
    Node *head = node1;
    Node *tail = node1;
    insertAtTail(tail, 11);
    insertAtTail(tail, 12);
    insertAtTail(tail, 13);
    insertAtTail(tail, 14);
    insertAtTail(tail, 15);
    insertAtHead(head, 9);
    insertAtHead(head, 8);
    insertAtHead(head, 7);
    print(head);
    Node *prev = NULL;
    Node *curr = head;
    while (curr != NULL)
    {
        Node *next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
        }
        print(prev);
    }
```

Output:



```
7-> 8-> 9-> 10-> 11-> 12-> 13-> 14-> 15-> NULL
15-> 14-> 13-> 12-> 11-> 10-> 9-> 8-> 7-> NULL
```

Q3. Write a function that takes two sorted single linked lists as input and merges them into a single sorted list. The original lists should remain unchanged.

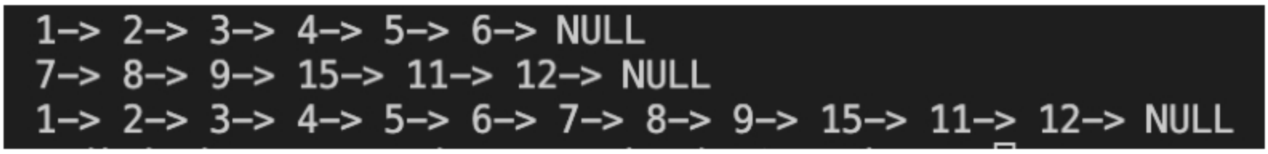
Code:

```
#include<iostream>
using namespace std;
 class
    Node{ pu
    blic : int
    data;
    Node* next;
    Node(int data){ this-
        >data = data; next
        = NULL;
    }
 };
  void insertAtHead(Node* &head , int data)
    { Node* newNode = new Node(data);
    newNode->next = head;
    head =newNode;
 }
  void insertAtTail(Node* &tail , int data){
    Node* newNode = new Node(data);
    tail->next = newNode;
    tail = newNode;
 }
 void insertAtPosition(Node* &head ,Node* &tail, int data,int
pos){
    if(pos==1)
        { insertAtHead(head,data
        ); return ;
    }
    Node* temp = head;
    int count =1;
    while(count<pos-1){
        temp=temp->next;
        count++;
    }
    if(temp->next==NULL)
        {insertAtTail(tail ,
        data); return;
    }
     Node* newnode = new Node(data);
    newnode->next = temp->next;
```

```
        temp->next =newnode;

    }
    void print(Node* head)
        { Node* temp = head;
        while(temp!=NULL){
            cout<<temp->data<<"-> ";
            temp = temp->next;
        }cout<<"NULL";
        cout<<endl;
    }
    int main(){
        Node * node1 = new Node(1);
        Node* head1 =node1;
        Node* tail1 = node1;
        insertAtTail(tail1,2);
        insertAtTail(tail1,3);
        insertAtTail(tail1,4);
        insertAtTail(tail1,5);
        insertAtTail(tail1,6);
        print(head1);
        Node * node2 = new Node(7);
        Node* head2 =node2;
        Node* tail2 = node2;
        insertAtTail(tail2,8);
        insertAtTail(tail2,9);
        insertAtTail(tail2,15);
        insertAtTail(tail2,11);
        insertAtTail(tail2,12);
        print(head2);
        Node* newhead = head1;
        Node* newTail = newhead;
        while(newTail->next!=NULL){
            newTail= newTail->next;
        }
        newTail->next = head2;
        newTail=tail2;
        print(newhead);
    }
```

Output:



```
1-> 2-> 3-> 4-> 5-> 6-> NULL
7-> 8-> 9-> 15-> 11-> 12-> NULL
1-> 2-> 3-> 4-> 5-> 6-> 7-> 8-> 9-> 15-> 11-> 12-> NULL
```

Q4. Write a function to remove the nth node from the end of a single linked list and return the head of the modified list.

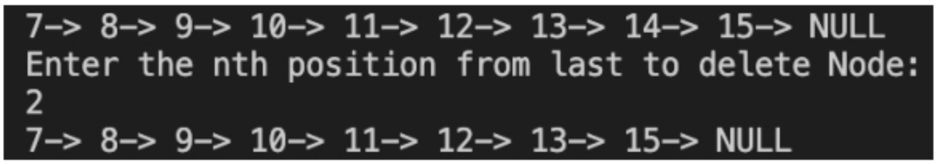
Code:

```
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node *next;
    Node(int data)
    {
        this->data = data;
        next = NULL;
    }
};
void insertAtHead(Node *&head, int data)
{
    Node *newNode = new Node(data);
    newNode->next = head;
    head = newNode;
}
void insertAtTail(Node *&tail, int data)
{
    Node *newNode = new Node(data);
    tail->next = newNode;
    tail = newNode;
}
void insertAtPosition(Node *&head, Node *&tail, int data, int
pos)
{
    if (pos == 1)
    {
        insertAtHead(head, data);
        return;
    }
    Node *temp = head;

    int count = 1;
    while (count < pos - 1)
    {
        temp = temp->next;
        count++;
    }
    if (temp->next == NULL)
    {
        insertAtTail(tail, data);
        return;
    }
    Node *newnode = new Node(data);
    newnode->next = temp->next;
    temp->next = newnode;
}
void print(Node *head)
{
    Node *temp = head;
    while (temp != NULL)
    {
        cout << temp->data << "-> ";
        temp = temp->next;
    }
    cout << "NULL";
    cout << endl;
}

int main()
{
    Node *node1 = new Node(10);
    Node *head = node1;
    Node *tail = node1;
    insertAtTail(tail, 11);
    insertAtTail(tail, 12);
    insertAtTail(tail, 13);
    insertAtTail(tail, 14);
    insertAtTail(tail, 15);
    insertAtHead(head, 9);
    insertAtHead(head, 8);
    insertAtHead(head, 7);
    print(head);
    cout << "Enter the nth position from last to delete Node:" << endl;
    int n;
    cin >> n;
    Node *start = new Node(-1);
    start->next = head;
    Node *slow = start;
    Node *fast = start;
    for (int i = 0; i < n; i++)
    {
        fast = fast->next;
    }
    while (fast != NULL && fast->next != NULL)
    {
        slow = slow->next;
        fast = fast->next;
    }
    slow->next = slow->next->next;
    start = start->next;
    print(start);
}
```

Output:



```
7-> 8-> 9-> 10-> 11-> 12-> 13-> 14-> 15-> NULL
Enter the nth position from last to delete Node:
2
7-> 8-> 9-> 10-> 11-> 12-> 13-> 15-> NULL
```

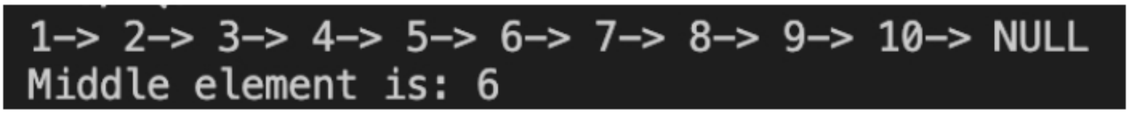
Q5. Write a function to find the middle node of a single linked list. If the list contains an even number of nodes, return the second middle node.

Code:

```
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node *next;
    Node(int data)
    {
        this->data = data;
        next = NULL;
    }
};
void insertAtHead(Node *&head, int data)
{
    Node *newNode = new Node(data);
    newNode->next = head;
    head = newNode;
}
void insertAtTail(Node *&tail, int data)
{
    Node *newNode = new Node(data);
    tail->next = newNode;
    tail = newNode;
}
void insertAtPosition(Node *&head, Node *&tail, int data, int pos)
{
    if (pos == 1)
    {
        insertAtHead(head, data);
        return;
    }
    Node *temp = head;

    int count = 1;
    while (count < pos - 1)
    {
        temp = temp->next;
        count++;
    }
    if (temp->next == NULL)
    {
        insertAtTail(tail, data);
        return;
    }
    Node *newnode = new Node(data);
    newnode->next = temp->next;
    temp->next = newnode;
}
void print(Node *head)
{
    Node *temp = head;
    while (temp != NULL)
    {
        cout << temp->data << "-> ";
        temp = temp->next;
    }
    cout << "NULL";
    cout << endl;
}
int main()
{
    Node *node1 = new Node(4);
    Node *head = node1;
    Node *tail = node1;
    insertAtTail(tail, 5 );
    insertAtTail(tail, 6 );
    insertAtTail(tail, 7 );
    insertAtTail(tail, 8 );
    insertAtTail(tail, 9 );
    insertAtTail(tail, 10 );
    insertAtHead(head, 3 );
    insertAtHead(head, 2 );
    insertAtHead(head, 1 );
    print(head);
    Node *slow = head;
    Node *fast = head;
    while (fast != NULL)
    {
        fast = fast->next;
        if (fast != NULL)
        {
            slow = slow->next;
            fast = fast->next;
        }
    }
    cout <<"Middle element is: "<< slow->data << endl;
}
```

Output:



```
1-> 2-> 3-> 4-> 5-> 6-> 7-> 8-> 9-> 10-> NULL
Middle element is: 6
```

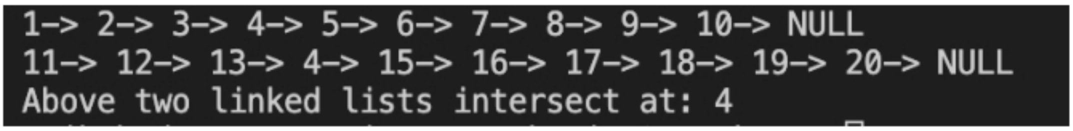
Q6. Write a function to find the intersection of two singly linked lists. If the lists do not intersect, return null.

Code:

```
#include<iostream>
#include<unordered_map>
using namespace std;
 class
    Node{ pu
    blic : int
    data;
    Node* next;
    Node(int data){
        this->data = data;
        next = NULL;
    }
 };
  void insertAtHead(Node* &head , int data)
    { Node* newNode = new Node(data);
    newNode->next = head;
    head =newNode;
 }
  void insertAtTail(Node* &tail , int data){
    Node* newNode = new Node(data);
    tail->next = newNode;
    tail = newNode;
 }
void print(Node* head)
    { Node* temp = head;
    while(temp!=NULL){
        cout<<temp->data<<"-> ";
        temp = temp->next;
    }cout<<"NULL";
    cout<<endl;
}
int main(){
    Node* node1 = new Node(4);
    Node* head =node1;
    Node* tail = node1;
    insertAtTail(tail,5);
    insertAtTail(tail,6);
    insertAtTail(tail,7);
    insertAtTail(tail,8);
    insertAtTail(tail,9);
    insertAtTail(tail,10);
    insertAtHead(head,3);
    insertAtHead(head,2);
    insertAtHead(head,1);
    print(head);
    Node* node2 = new Node(4);
    Node* head2 =node2;
    Node* tail2 = node2;
```

```
        insertAtTail(tail2,16);
        insertAtTail(tail2,17);
        insertAtTail(tail2,18);
        insertAtTail(tail2,19);
        insertAtTail(tail2,20);
        insertAtHead(head2,13);
        insertAtHead(head2,12);
        insertAtHead(head2,11);
        print(head2);
        unordered_map<int,int>mp;
        Node* temp =head;
        while(temp!=NULL){
            mp[temp->data]++;
            temp=temp->next;
        }
        temp = head2;
        while(temp!=NULL){
            if(mp[temp->data]>0){
                break;
            }
            else{
            temp= temp-
            >next; }
        }
       if(temp != NULL) {
            cout << "Above two linked lists intersect at: " <<
    temp-
        >data; }
        else {
            cout << "The two linked lists do not intersect";
        }
    }
```

Output:



```
1-> 2-> 3-> 4-> 5-> 6-> 7-> 8-> 9-> 10-> NULL
11-> 12-> 13-> 4-> 15-> 16-> 17-> 18-> 19-> 20-> NULL
Above two linked lists intersect at: 4
```

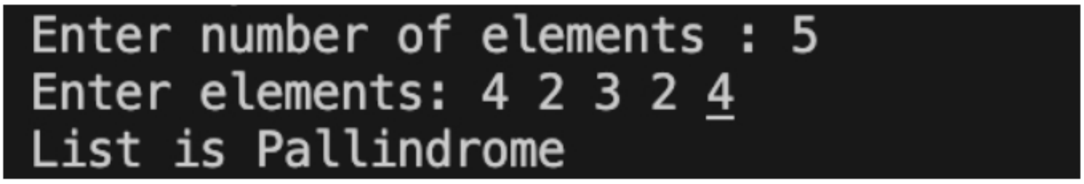
Q7. Write an O(n) time function to determine if a single linked list is a palindrome.

Code:

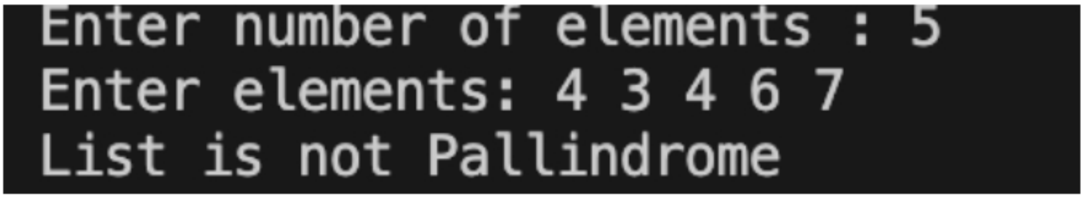
```
    #include<iostream>
    using namespace std;
    class Node{
        public:
            int data;
            Node* next;
        Node(){}
        Node(int data){
            this->data = data;
        }
    };
    void createLinkedList(Node*& head, int size) {
        Node* temp = nullptr;
        for(int i = 0; i < size; i++) {
            int d;
            cin >> d;
            if(i == 0) {
                head = new Node(d);
                temp = head;
            } else {
                temp->next = new Node(d);
                temp = temp->next;
            }
        }
    }
    void print(Node* head)
        { Node* temp = head;
        while(temp != NULL){
            cout<<temp->data<<" ";
            temp = temp->next;
        }
    }
    bool isPallindrome(Node* head){
        if(head == NULL && head->next == NULL){
            return true;
        }
        Node* slow = head;
        Node* fast = head;
        while(fast != NULL && fast->next != NULL){
            slow = slow->next;
            fast = fast->next->next;
        }
        Node* curr = slow;
        Node* prev = NULL;
        while(curr){
            Node* nxt = curr->next;
            curr->next = prev;
            prev = curr;
            curr = nxt;
        }
        slow = head;
        fast = prev;
        while(fast){
            if(slow->data != fast->data){
                return false;
            }
            slow = slow->next;
            fast = fast->next;
        }
        return true;
    }
```

```
int main(){
    Node* head;
    cout<<"Enter number of elements : ";
    int n;
    cin>>n;
    cout<<"Enter elements: ";
    createLinkedList(head, n);
    if(isPallindrome(head)){
        cout<<"List is
    Pallindrome"<<endl;}else{
        cout<<"List is not Pallindrome"<<endl;
    }
}
```

Output:



```
Enter number of elements : 5
Enter elements: 4 2 3 2 4
List is Pallindrome
```



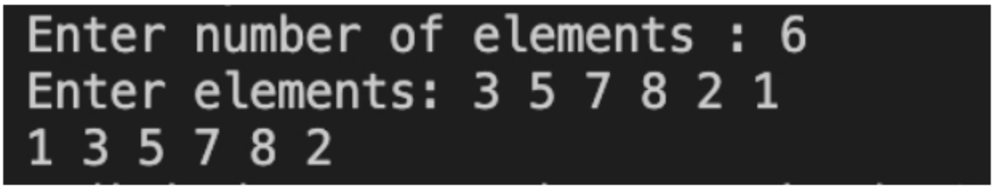
```
Enter number of elements : 5
Enter elements: 4 3 4 6 7
List is not Pallindrome
```

Q8. Define the function moveToFront(struct node *head) to move a last node to the front of a single linked list.

Code:

```
#include<iostream>
using namespace std;
class Node{
    public:
        int data;
        Node* next;
    Node(){}
    Node(int data){
        this->data = data;
        this->next = NULL;
    }
};
void createLinkedList(Node*& head, int size) {
    Node* temp = nullptr;
    for(int i = 0; i < size; i++) {
        int d;
        cin >> d;
        if(i == 0) {
            head = new Node(d);
            temp = head;
        } else {
            temp->next = new Node(d);
            temp = temp->next;
        }
    }
}
void print(Node* head)
    { Node* temp = head;
    while(temp != NULL){
        cout<<temp->data<<" ";
        temp = temp->next;
    }
}
void moveToFront(Node* &head){
    if(head==NULL || head->next==NULL)
        { cout<<"NOT  POSSIBLE"<<endl;
        return;
    }
    Node* temp = head; while(temp-
    >next->next!=NULL){
        temp = temp->next;
    }
    Node* lastNode = temp->next;
    temp->next=NULL; lastNode-
    >next = head;
    head= lastNode;
}
int main(){
    Node* head;
    cout<<"Enter number of elements : ";
    int n;
    cin>>n;
    cout<<"Enter elements: ";
    createLinkedList(head, n);
    moveToFront(head);
    print(head);
    cout<<endl;
}
```

Output:



```
Enter number of elements : 6
Enter elements: 3 5 7 8 2 1
1 3 5 7 8 2
```

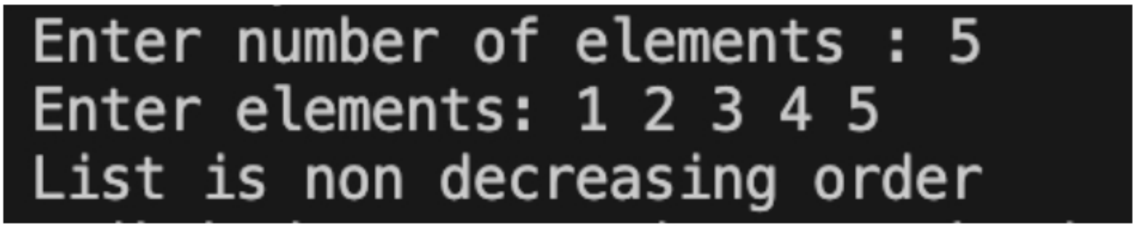
Q9. Write a program to check if the list is in non-decreasing order or not.

Code:

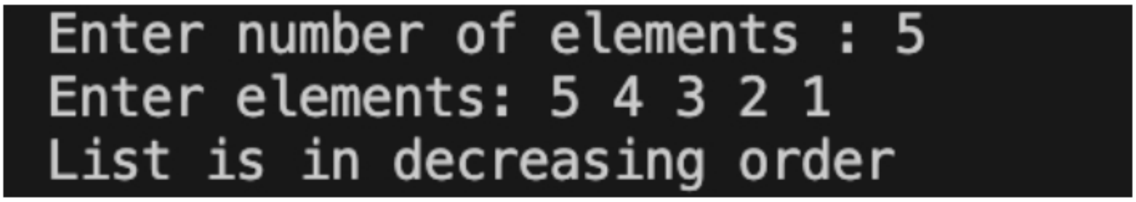
```
#include<iostream>
using namespace std;
class Node{
    public:
        int data;
        Node* next;
    Node(){}
    Node(int data){
        this->data = data;
        this->next = NULL;
    }
};
void createLinkedList(Node*& head, int size) {
    Node* temp = nullptr;
    for(int i = 0; i < size; i++) {
        int d;
        cin >> d;
        if(i == 0) {
            head = new Node(d);
            temp = head;
        } else {
            temp->next = new Node(d);
            temp = temp->next;
        }
    }
}
void print(Node* head)
    { Node* temp = head;
    while(temp != NULL){
        cout<<temp->data<<" ";
        temp = temp->next;
    }
}
bool isNonDecreasing(Node* head){
    if(head == NULL || head->next == NULL){
        return true;
    }
    Node* prev = head;
    Node* curr = head->next;
    while(curr){
        if(prev->data > curr->data){
            return false;
        }
        prev = curr;
        curr = curr->next;
    }
    return true;
}

int main(){
    Node* head;
    cout<<"Enter number of elements : ";
    int n;
    cin>>n;
    cout<<"Enter elements: ";
    createLinkedList(head, n);
    if(isNonDecreasing(head)){
        cout<<"List is non decreasing
    order";}else{
        cout<<"List is in decreasing order";
    }
    cout<<endl;
}
```

Output:



```
Enter number of elements : 5
Enter elements: 1 2 3 4 5
List is non decreasing order
```



```
Enter number of elements : 5
Enter elements: 5 4 3 2 1
List is in decreasing order
```