LINKED LIST:-

Problem Statement

Jaanu wants to create a program that allows her to create a linked list of strings. Jaanu to enter the number of strings she wants to insert into the linked list. Then, she should enter each string one by one and insert them at the end of the linked list. After that, ask Jaanu to enter a new string, which will be appended at the end of the linked list. Finally, display the contents of the linked list.

Note: This is a sample question asked in a TCS interview.
Input format :
The first line contains an integer, num_of_strings, representing the number of strings in the linked list.
The next num_of_strings lines contain the strings that constitute the linked list.
The last line contains a string, new_string, which needs to be appended at the end of the linked list.
Output format :
The insert and append at the end of the linked lists after inserting the new node are in the format "Linked List Contents: <list values separated by space>".

Refer to the sample output for format specifications.
Code constraints :
The number of strings should be a positive integer.
The length of each string should be less than or equal to 100 characters.
Sample test cases :
Input 1 :

```
3
Apple
Banana
Orange
Grapes
```

Output 1 :

```
Linked List Contents: Apple Banana Orange Grapes
```

Input 2 :

```
2
Hello
World
Space
```

Output 2 :

```
Linked List Contents: Hello World Space
```

Input 3 :

```
0
None
None
```

Output 3 :

```
Linked List Contents: None
```

```
#include <iostream>

#include <string>


using namespace std;
```

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

void insertAtEnd(Node** head, string newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = nullptr;

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

void appendToEnd(Node** head, string newString) {
    insertAtEnd(head, newString);
}

void printLinkedList(Node* head) {
    Node* current = head;
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
```

```
    }
    cout << endl;
}

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

    cin >> numStrings;

    for (int i = 0; i < numStrings; i++) {
        string str;
        cin >> str;
        insertAtEnd(&head, str);
    }

    string newString;
    cin >> newString;
    appendToEnd(&head, newString);

    cout << "Linked List Contents: ";
    printLinkedList(head);

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

    return 0;
```

}

Problem statement

Uma wants to create a program that allows her to build a linked list by inserting nodes at the end. She wants to be able to input the data for each node and specify when to stop inserting nodes (a negative value is entered, indicating the end of the input). After inserting the nodes, she wants to display the contents of the linked list.

Note: This is a sample question asked in a Wipro interview.
Input format :
The input consists of an integer value for each node to be inserted at the end of the linked list.
After inserting each node, enter a non-negative integer indicating the value of the next node to be inserted. If a negative integer is entered, it indicates the end of node insertion.
The input terminates when a negative integer is entered.
Output format :
If the linked list is empty, the output displays the message "Linked List is empty." on a new line.
If the linked list is not empty, the output displays the contents of the linked list in a space-separated format on a new line, preceded by the text "Linked List: ".
Each node's data is displayed in the order it was inserted.

Refer to the sample output for format specifications.
Code constraints :
The elements of the linked list are integers.
The input will be a series of integers, terminated by a negative value.
The input values can be positive or zero.
Sample test cases :
Input 1 :

```
1
2
3
4
-1
```

Output 1 :

```
Linked List: 1 2 3 4
```

Input 2 :

```
-2
```

Output 2 :

```
Linked List is empty.
```

Input 3 :

```
2000
-3000
4000
-5000
-1
```

Output 3 :

```
Linked List: 2000
```

```
#include<iostream>

using namespace std;
```

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

void print(struct Node *head)
{
   if (head == NULL)
   {
      cout << "Linked List is empty." << endl;
      return;
   }

   struct Node *current = head;
   cout << "Linked List: ";
   while (current != NULL)
   {
      cout << current->data << " ";
      current = current->next;
   }
   cout << endl;
}

void insert(struct Node **head, int value)
{
   Node *last = *head;
   Node *newnode = new Node;
   newnode->data = value;
   newnode->next = NULL;
```

```
    if(*head == NULL)
    {
        *head = newnode;
    }
    else
    {
        while(last->next != NULL)
        {
            last = last->next;
        }
        last->next = newnode;
    }
}

int main()
{
    struct Node *head = NULL;
    int elements;
    while(1)
    {
        cin >> elements;
        if(elements >= 0)
        {
            insert(&head, elements);
        }
        else
        {
            break;
        }
    }
```

```
    print(head);
}
```

Problem Statement

Suresh wants to create a program that allows him to create a linked list of integers. Suresh should enter the number of elements he wants to add to the linked list. Then, he should enter each element one by one and add them to the linked list. Finally, display the contents of the linked list. Suresh should note that the code handles an empty list case and prints a message if the list is empty.

Note: This is a sample question asked in an Accenture interview.
Input format :
The first line of input contains an integer numElements, representing the number of elements to be added to the linked list.
The next numElements lines contain an integer each, representing the elements to be added to the linked list.
Output format :
The output displays the elements of the linked list, separated by a space.
If the linked list is empty, the program will output "The list is empty."

Refer to the sample output for formatting specifications.
Code constraints :
1<=numElements<=20
Sample test cases :
Input 1 :

```
4
7 8 6 4
```

Output 1 :

```
7 8 6 4
```

Input 2 :

```
0
```

Output 2 :

```
The list is empty.
```

```
#include <iostream>

using namespace std;


struct Node {

    int data;

    Node* next;

};


class LinkedList {
```

```
private:
  Node* head;
public:
  LinkedList(){
    head = NULL;
  }

  void push_back(int newElement) {
    Node* newNode = new Node();
    newNode->data = newElement;
    newNode->next = NULL;
    if(head == NULL) {
      head = newNode;
    } else {
      Node* temp = head;
      while(temp->next != NULL)
        temp = temp->next;
      temp->next = newNode;
    }
  }

  void PrintList() {
    Node* temp = head;
    if(temp != NULL) {
      while(temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
      }
      cout << endl;
    } else {
      cout << "The list is empty.\n";
```

```
    }
  }
};

int main() {
 LinkedList MyList;

 int numElements;
 cin >> numElements;

 for (int i = 0; i < numElements; i++) {
  int element;
  cin >> element;
  MyList.push_back(element);
 }

 MyList.PrintList();

 return 0;
}
```

Problem Statement

Lisa wants to create a linked list sorted in ascending order. She wants to insert nodes in such a way that the linked list remains sorted after insertion.

Write a program that takes the number of nodes to be inserted, followed by their values in non-decreasing order. The program should then ask for a new value and insert a node with that value at the appropriate position to maintain the sorted order.

Finally, the program should print the updated linked list.

Example

Input:
5
1 3 5 7 9
4

Output:
1 3 4 5 7 9

Explanation:
The program first creates a sorted linked list using the given input values: 1, 3, 5, 7, and 9. After creating the initial sorted linked list, the program asks for a new value, which is 4. It then inserts a node with value 4 at the appropriate position to maintain the sorted order. Finally, the program prints the updated linked list, which is 1, 3, 4, 5, 7, and 9 in ascending order.

Note: This is a sample question asked in Wipro recruitment.
Input format :
The first line of input consists of an integer **n**, representing the number of elements in the initial sorted linked list.
The second line of input consists of **n** space-separated integers, representing the elements of the sorted linked list.
The third line of input consists of integer **data**, which represents the element to be inserted into the linked list.
Output format :
The program should output the updated linked list after inserting the new element, separated by space.
The linked list should remain sorted in ascending order.
Code constraints :
The input linked list is sorted in ascending order.
Sample test cases :
Input 1 :

```
5
1 3 5 7 9
4
```

Output 1 :

```
1 3 4 5 7 9
```

Input 2 :

```
6
-15 -10 0 5 9 10
7
```

Output 2 :

```
-15 -10 0 5 7 9 10
```

```
#include <iostream>

using namespace std;


/* Link list node */

struct Node {

   int data;

   Node* next;

};
```

```
/* Function to insert a new_node in a list. Note that this function expects a pointer to head_ref as this can modify the head of the input linked list (similar to push()) */
void sortedInsert(Node** head_ref, int new_data)
{
   Node* new_node = new Node();
   new_node->data = new_data;
   new_node->next = NULL;

   Node* current;
   /* Special case for the head end */
   if (*head_ref == NULL || (*head_ref)->data >= new_data) {
      new_node->next = *head_ref;
      *head_ref = new_node;
   }
   else {
      /* Locate the node before the point of insertion */
      current = *head_ref;
      while (current->next != NULL && current->next->data < new_data) {
         current = current->next;
      }
      new_node->next = current->next;
      current->next = new_node;
   }
}

/* Function to print linked list */
void printList(Node* head)
{
   Node* temp = head;
   while (temp != NULL) {
      cout << temp->data << " ";
```

```
        temp = temp->next;
    }
    cout << endl;
}

/* Driver program to test count function */
int main()
{
    /* Start with the empty list */
    Node* head = NULL;
    int n, data;

    cin >> n;

    for (int i = 0; i < n; i++) {
        cin >> data;
        sortedInsert(&head, data);
    }

    cin >> data;
    sortedInsert(&head, data);

    printList(head);

    return 0;
}
```

Problem Statement

You are developing a Shift Schedule Management System for a company. The system maintains a list of employees and their shift assignments. Each employee is represented as a node in a linked list, where each node contains an integer representing the shift number.

Your task is to implement a feature that allows left-shifting of the shift schedule by a given number of shifts. This is required when there are changes in the company's work schedule or staffing requirements.

Note: This is the sample question asked in Microsoft recruitment.
Input format :
The first line contains an integer 'N' representing the number of shifts in the schedule.
The next N space integer represents the shift numbers for each shift.
The last line contains an integer 'k' representing the number of shifts to left-shift the schedule.
Output format :
Tje first line of output should display the original linked list representing the shift schedule.
The next line should display the modified linked list after left-shifting the schedule by 'k' shifts.
Code constraints :
The linked list represents the employee shift schedule.
Each node in the linked list contains an integer representing the shift number.
The shift schedule can be left-shifted by a positive integer 'k', where 'k' is less than or equal to the length of the linked list.
The left-shift operation should preserve the order of shifts.

Sample test cases :
Input 1 :

```
5
2 4 7 8 9
3
```

Output 1 :

```
Original Linked List: 2 4 7 8 9
Modified Linked List after left shift: 8 9 2 4 7
```

Input 2 :

```
8
1 2 3 4 5 6 7 8
4
```

Output 2 :

```
Original Linked List: 1 2 3 4 5 6 7 8
Modified Linked List after left shift: 5 6 7 8 1 2 3 4
```

```
#include <iostream>

using namespace std;


// Node structure

struct Node {

   int data;

   Node* next;

};


// Function to create a new node

Node* createNode(int data) {
```

```
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the linked list
Node* insertNode(Node* head, int data) {
    if (head == NULL)
        head = createNode(data);
    else {
        Node* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = createNode(data);
    }
    return head;
}

// Function to left-shift the linked list by k nodes
Node* leftShiftLinkedList(Node* head, int k) {
    if (head == NULL || k == 0)
        return head;

    Node* current = head;
    int count = 1;
    while (count < k && current != NULL) {
        current = current->next;
        count++;
    }
```

```
    if (current == NULL)
        return head;

    Node* kthNode = current;
    while (current->next != NULL)
        current = current->next;

    current->next = head;
    head = kthNode->next;
    kthNode->next = NULL;

    return head;
}

// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    int N, k;
    cin >> N;

    Node* head = NULL;
    for (int i = 0; i < N; i++) {
        int value;
```

```
        cin >> value;
        head = insertNode(head, value);
    }

    cin >> k;

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

    head = leftShiftLinkedList(head, k);

    cout << "Modified Linked List after left shift: ";
    printList(head);

    return 0;
}
```

Problem Statement

You are tasked with designing a program that operates on two linked lists. Your objective is to create a new linked list that represents the union of the two given linked lists while ensuring that the elements in the union list are distinct and sorted in ascending order.

Note: This is a sample question asked in the Microsoft interview.
Input format :
The first line consists of an integer **n**, representing the number of nodes in the first linked list.
The second line consists of **n** space-separated integers, representing the nodes in the first linked list.
The third line consists of an integer **m**, representing the number of nodes in the second linked list.
The fourth line consists of **m** space-separated integers, representing the nodes in the second linked list.
Output format :
The first line of output displays the nodes of the first linked list, sorted in ascending order.
The second line displays the nodes of the second linked list, sorted in ascending order.
The third line displays the nodes of the union linked list after merging the distinct elements from the first and second linked lists.

Refer to the sample output for formatting specifications.
Sample test cases :
Input 1 :

```
6
9 6 4 2 3 8
```

```
5
1 2 8 6 2
```

Output 1 :

```
First Linked List: 2 3 4 6 8 9
Second Linked List: 1 2 2 6 8
Union Linked List: 1 2 3 4 6 8 9
```

Input 2 :

```
6
1 5 1 2 2 5
5
4 5 6 7 1
```

Output 2 :

```
First Linked List: 1 1 2 2 5 5
Second Linked List: 1 4 5 6 7
Union Linked List: 1 2 4 5 6 7
```

```
#include <iostream>

using namespace std;

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

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert data in a sorted manner
void sortedInsert(Node** head, Node* newNode) {
    if (*head == NULL || newNode->data < (*head)->data) {
```

```
        newNode->next = *head;
        *head = newNode;
    } else {
        Node* current = *head;
        while (current->next != NULL && current->next->data < newNode->data) {
            current = current->next;
        }
        newNode->next = current->next;
        current->next = newNode;
    }
}

// Function to display the linked list
void display(Node* head) {
    if (head == NULL) {
        cout << "Empty linked list" << endl;
        return;
    }

    Node* current = head;
    while (current != NULL) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

// Function to check if an element exists in the linked list
bool exists(Node* head, int data) {
    Node* current = head;
    while (current != NULL) {
```

```
        if (current->data == data) {
            return true;
        }
        current = current->next;
    }
    return false;
}

// Function to get the union of two linked lists
Node* makeUnion(Node* head1, Node* head2) {
    Node* result = NULL;

    // Traverse the first linked list and add distinct elements to the result
    Node* temp1 = head1;
    while (temp1 != NULL) {
        if (!exists(result, temp1->data)) {
            Node* newNode = createNode(temp1->data);
            sortedInsert(&result, newNode);
        }
        temp1 = temp1->next;
    }

    // Traverse the second linked list and add distinct elements to the result
    Node* temp2 = head2;
    while (temp2 != NULL) {
        if (!exists(result, temp2->data)) {
            Node* newNode = createNode(temp2->data);
            sortedInsert(&result, newNode);
        }
        temp2 = temp2->next;
    }
```

```
    return result;
}

int main() {
    int n1, n2;
    cin >> n1;

    Node* head1 = NULL;

    for (int i = 0; i < n1; i++) {
        int data;
        cin >> data;
        Node* newNode = createNode(data);
        sortedInsert(&head1, newNode);
    }

    cin >> n2;

    Node* head2 = NULL;

    for (int i = 0; i < n2; i++) {
        int data;
        cin >> data;
        Node* newNode = createNode(data);
        sortedInsert(&head2, newNode);
    }

    cout << "First Linked List: ";
    display(head1);
```

```
    cout << "Second Linked List: ";
    display(head2);

    Node* unionList = makeUnion(head1, head2);

    cout << "Union Linked List: ";
    display(unionList);

    return 0;
}
```

Problem Statement

You are working on a program for an inventory management system in a retail store. The store uses barcodes to label and track its products. Each barcode is represented as a linked list, where each digit of the barcode is stored in a separate node. Your task is to write a function that adds 1 to the barcode value and updates the linked list accordingly.

For example, 1999 is represented as (1-> 9-> 9 -> 9), and adding 1 to it should change it to (2->0->0->0)

Note: This is a sample question asked in the Flipkart interview.
Input format :
The first line of input contains an integer n, indicating the number of digits in the barcode.
The second line of input contains n space-separated integers, representing the digits of the barcode.
Output format :
The output prints the linked list representing the modified barcode, after adding 1 to its value.

Code constraints :
The barcode represents a non-negative integer.
The number of digits in the barcode can vary.
Sample test cases :
Input 1 :

```
4
1 9 9 9
```

Output 1 :

```
2 0 0 0
```

Input 2 :

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

Output 2 :

```
1 2 3 4 5 7
```

```
#include <iostream>
```

```
using namespace std;

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

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

// Function to insert data at the end of the linked list
void insert(Node** head, int data) {
    Node* newNode = createNode(data);

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

```
// Function to reverse the linked list
Node* reverseList(Node* head) {
    Node *prev = nullptr, *current = head, *nextNode = nullptr;
    while (current != nullptr) {
        nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }
    return prev;
}

// Function to add 1 to the linked list representation of a number
Node* addOne(Node* head) {
    head = reverseList(head);
    Node* current = head;
    int carry = 1;
    while (current != nullptr) {
        int sum = current->data + carry;
        current->data = sum % 10;
        carry = sum / 10;
        if (carry == 0) {
            break;
        }
        current = current->next;
    }
    if (carry != 0) {
        Node* newNode = createNode(carry);
        current->next = newNode;
    }
    head = reverseList(head);
```

```
    return head;
}

// Function to display the linked list
void display(Node* head) {
    if (head == nullptr) {
        cout << "Empty linked list" << endl;
        return;
    }

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

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

    Node* head = nullptr;

    for (int i = 0; i < n; i++) {
        int data;
        cin >> data;
        insert(&head, data);
    }

    //cout << "Original Linked List: ";
```

```
    //display(head);


    head = addOne(head);


    //cout << "Updated Linked List: ";
    display(head);


    return 0;
}
```

Problem Statement

You are tasked with creating a program that processes a linked list containing integer data and rearranges its nodes. Specifically, your program should separate the even and odd integers, placing the even integers before the odd ones while maintaining their original order within each group.

Note: The new nodes are inserted at the beginning of the linked list.

Example

Input:
linked list = 1 2 3 4

Output:
4 2 3 1

Explanation:
Initially, the linked list contains four nodes with the values: 1 2 3 4.
The new nodes are inserted at the beginning of the list. So, after insertion, the nodes will be in the order: 4 3 2 1.
Now, rearrange the linked list to position even numbers before odd numbers while preserving their order.
The resulting rearranged linked list is 4 2 3 1.

Input format :
The first line of input consists of an integer **N**, representing the number of elements in the linked list.
The second line consists of **N** space-separated integers, representing the elements in the linked list.

Output format :
The output displays the rearranged list, containing the even numbers followed by the odd numbers.

Sample test cases :

Input 1 :

```
4
1 2 3 4
```

Output 1 :

```
4 2 3 1
```

Input 2 :

```
5
12  15  13  14  16
```

Output 2 :

```
16  14  12  13  15
```

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

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

Node* head = NULL;

void push(int new_data) {
   Node* new_node = new Node;
   new_node->data = new_data;
   new_node->next = head;
   head = new_node;
}

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

void segregateEvenOdd() {
```

```
Node* end = head;
Node* prev = NULL;
Node* curr = head;
Node* new_end;

while (end->next != NULL) {
   end = end->next;
}
new_end = end;

while (curr->data % 2 != 0 && curr != end) {
   new_end->next = curr;
   curr = curr->next;
   new_end->next->next = NULL;
   new_end = new_end->next;
}
if (curr->data % 2 == 0) {
   head = curr;
   while (curr != end) {
      if (curr->data % 2 == 0) {
         prev = curr;
         curr = curr->next;
      }
      else {
         prev->next = curr->next;
         curr->next = NULL;
         new_end->next = curr;
         new_end = curr;
         curr = prev->next;
      }
   }
```

```
    }
    else {
        prev = curr;
    }
    if (new_end != end && end->data % 2 != 0) {
        prev->next = end->next;
        end->next = NULL;
        new_end->next = end;
    }
}

int main() {
    int n, A;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> A;
        push(A);
    }
    segregateEvenOdd();
    printList();
    return 0;
}
```

Problem Statement

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 to implement this function. The function should take two linked lists, **list1** and **list2,** as input, where each list is represented by its head node.

Note: This is a sample question asked in a Capgemini interview.
Input format :
The first line contains an integer n, denoting the number of nodes in list1.
The next line contains n space-separated integers, representing the values of the nodes in list1.
The next line contains an integer m, denoting the number of nodes in list2.
The next line contains m space-separated integers, representing the values of the nodes in list2.
Output format :

The first line of output displays the elements of the first linked list before the deletion, separated by a space.
The second line of output displays 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 will be displayed stating, "All elements in the first linked list are the same."

Refer to the sample output for formatting specifications.
Code constraints :
1 <= n,m <= 100
-50000 <= values of nodes <= 50000
Sample test cases :
Input 1 :

```
5
2 3 4 5 1
5
1 6 2 3 8
```

Output 1 :

```
First Linked List before deletion: 2 3 4 5 1
First Linked List after deletion: 4 5
```

Input 2 :

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

Output 2 :

```
First Linked List before deletion: 1 2 3 4 5
First Linked List after deletion:
All elements in the first linked list are the same.
```

```
#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;
    }
}

bool valueExists(Node* second, int value) {
    Node* curr = second;
    while (curr != nullptr) {
        if (curr->data == value) {
            return true;
        }
        curr = curr->next;
    }
    return false;
}

void deleteNodesInSecondList(Node** first, Node* second) {
    if (*first == nullptr || second == nullptr) {
        return;
    }

    Node* prev = nullptr;
    Node* curr1 = *first;

    while (curr1 != nullptr) {
```

```
        if (valueExists(second, curr1->data)) {
            if (prev == nullptr) {
                *first = curr1->next;
                delete curr1;
                curr1 = *first;
            } else {
                prev->next = curr1->next;
                delete curr1;
                curr1 = prev->next;
            }
        } else {
            prev = curr1;
            curr1 = curr1->next;
        }
    }
}

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) {
    if (head == nullptr) {
        return true;
    }

    int firstElement = head->data;
    Node* current = head->next;
    while (current != nullptr) {
        if (current->data != firstElement) {
            return false;
        }
        current = current->next;
    }
    return true;
}

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 << "First Linked List before deletion: ";
    displayLinkedList(first);

    deleteNodesInSecondList(&first, second);

    cout << "First Linked List after deletion: ";
    displayLinkedList(first);

    if (areAllElementsSame(first)) {
        cout << "All elements in the first linked list are the same.";
    }

    deleteLinkedList(first);
    deleteLinkedList(second);

    return 0;
}
```

Problem Statement

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.

Note: This is a sample question asked in a mPhasis interview.
Input format :
The first line of input consists of the number of nodes in the linked list, numNodes (an integer).
The second line of input consists of the values for each node, inserted at the end of the list, separated by spaces, in the order they appear in the linked list.
Output format :
The output displays the original list and the linked list in the next line after removing the nodes that appear more than once.

Refer to the sample output for format specifications.
Code constraints :
1 <= numNodes <= 100
-10000 <= values of nodes <= 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
```

```
#include <iostream>


using namespace std;




struct Node {

   int data;

   Node* next;

};


void insertNode(Node** head, int value) {
```

```
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

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

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

    Node* currNode = *head;

    while (currNode != nullptr) {
        Node* temp = currNode;
        while (temp->next != nullptr) {
            if (temp->next->data == currNode->data) {
                Node* duplicateNode = temp->next;
                temp->next = temp->next->next;
                delete duplicateNode;
            } else {
                temp = temp->next;
```

```
            }
        }
        currNode = currNode->next;
    }
}

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

int main() {
    Node* head = nullptr;
    int numNodes, value;

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

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

    deleteDuplicates(&head);

    cout << "Updated List: ";
```

```
    displayList(head);


    return 0;
}
```

Problem Statement

Madhev wants to remove all nodes with values greater than a specified value 'x' from a singly linked list. He needs your help to write a program that takes the size of the linked list, the elements of the linked list, and the value 'x' as input.

Additionally, he wants to insert new nodes at the end of the linked list. The program should then delete all nodes with values greater than 'x' from the linked list and display the modified linked list.

Write a program to solve Madhev's problem.

Note: This is a sample question asked in a Cocubes interview.
Input format :
The first line of input consists of the size of the linked list n (an integer).
The second line of input consists of the elements of the linked list arr (a sequence of space-separated integers).
The last line of input consists of the value 'x' (an integer) to compare against the nodes.
Output format :
The output displays the original linked list.
The modified linked list after removing nodes with values greater than 'x'.

Refer to the sample output for format specifications.
Code constraints :
1<=n<=1=100
-10000<=arr<=10000
-10000<=x<=10000
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
```

```
#include <iostream>
```

```
using namespace std;
struct Node {
   int data;
   Node* next;
};

Node* getNode(int data) {
   Node* newNode = new Node;
   newNode->data = data;
   newNode->next = nullptr;
   return newNode;
}

void deleteGreaterNodes(Node** head_ref, int x) {
   Node* temp = *head_ref;
   Node* prev = nullptr;



   while (temp != nullptr && temp->data > x) {
      *head_ref = temp->next;
      delete temp;
      temp = *head_ref;
   }



   while (temp != nullptr) {

      while (temp != nullptr && temp->data <= x) {
         prev = temp;
         temp = temp->next;
      }
```

```
        if (temp == nullptr)
            return;


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

        temp = prev->next;
    }
}

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

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

    cin >> size;

    for (int i = 0; i < size; i++) {
        int value;
        cin >> value;
        if (head == nullptr) {
```

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

    int x;
    cin >> x;

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

    deleteGreaterNodes(&head, x);

    cout << "\nModified Linked List: ";
    printList(head);

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

    return 0;
}
```

Problem Statement

Your task is to write a program that takes input for the number of elements in the 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.

Note: This is a sample question asked in a HCL interview.
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:
The first line should display the elements of the original linked list, separated by a space.
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 formatting specifications.
Code constraints :
$0 \leq n \leq 100$
Each string in the input is non-empty and consists of alphanumeric characters (letters and digits) only.
The total length of all strings combined does not exceed 10^2 characters.
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
```

```
#include <iostream>

using namespace std;


struct node {

   string data;

   node* nextptr;

} *stnode; // node declared
```

```
void make(int n);
void alternateDel(node* stnode);
void display();

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

   make(n);

   if (stnode == nullptr) {
      cout << "List is empty";
   } else {
      cout << "Linked list data: ";
      display();
      cout << "\nAfter deleting alternate node:";
      alternateDel(stnode);
      display();
   }

   return 0;
}

void make(int n) // function to create linked list
{
   struct node* frntNode, * tmp;

   if (n == 0) {
```

```
        stnode = nullptr;
        return;
    }

    string data;

    cin >> data;

    stnode = new node;
    stnode->data = data;
    stnode->nextptr = nullptr;
    tmp = stnode;

    for (int i = 2; i <= n; i++) {

        cin >> data;

        frntNode = new node;
        frntNode->data = data;
        frntNode->nextptr = nullptr;
        tmp->nextptr = frntNode;
        tmp = tmp->nextptr;
    }
}

void display() // function to display linked list
{
    if (stnode == nullptr) {
        cout << "List is empty";
    } else {
        struct node* tmp = stnode;
```

```
        while (tmp != nullptr) {
            cout << tmp->data << " ";
            tmp = tmp->nextptr;
        }
    }
}

void alternateDel(node* stnode) // function to delete alternate nodes
{
    if (stnode == nullptr)
        return;

    node* prev = stnode;
    node* alt_node = stnode->nextptr;

    while (prev != nullptr && alt_node != nullptr) {
        prev->nextptr = alt_node->nextptr;

        delete alt_node;

        prev = prev->nextptr;
        if (prev != nullptr)
            alt_node = prev->nextptr;
    }
}
```

Problem Statement

You are developing a program for a company to manage employee records. As part of the record management functionality, you need to implement a function that splits a grounded header-linked list of employee records into two separate lists: one for employees with even employee IDs and another for employees with odd employee IDs.

Implement a function to split a grounded header linked list into two separate lists, dividing the elements based on their parity (even and odd).

Note: This is a sample question asked in Infosys recruitment.
Input format :
The first line of input consists of the number of elements **n** in the list.
The second line of input consists of **n** elements, separated by space.
Output format :
The first line of output prints the list of even elements.
The second line of output prints the list of odd elements.

Refer to the sample output for formatting specifications.
Sample test cases :
Input 1 :

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

Output 1 :

```
Even List: 2 4
Odd List: 1 3 5
```

Input 2 :

```
8
1 2 4 6 8 10 12 14
```

Output 2 :

```
Even List: 2 4 6 8 10 12 14
Odd List: 1
```

```
#include <iostream>


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


// Function to insert a new node at the end of a linked list
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;
    }
}

// Function to display the linked list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        std::cout << temp->data << " ";
        temp = temp->next;
    }
    std::cout << std::endl;
}

// Function to split a linked list into two separate lists based on parity (even and odd)
void splitListByParity(Node* head, Node*& evenHead, Node*& oddHead) {
    Node* evenCurrent = nullptr;
    Node* oddCurrent = nullptr;

    Node* current = head->next;
    while (current != nullptr) {
        if (current->data % 2 == 0) {
            if (evenHead == nullptr) {
                evenHead = current;
                evenCurrent = current;
            } else {
                evenCurrent->next = current;
```

```
            evenCurrent = evenCurrent->next;
        }
    } else {
        if (oddHead == nullptr) {
            oddHead = current;
            oddCurrent = current;
        } else {
            oddCurrent->next = current;
            oddCurrent = oddCurrent->next;
        }
    }
    current = current->next;
  }

  // Set the last nodes of both lists to nullptr to terminate them
  if (evenCurrent != nullptr) {
    evenCurrent->next = nullptr;
  }
  if (oddCurrent != nullptr) {
    oddCurrent->next = nullptr;
  }
}

int main() {
  Node* head = new Node();
  head->next = nullptr;

  int n;
  std::cin >> n;
  for (int i = 0; i < n; i++) {
    int data;
```

```
        std::cin >> data;
        insertNode(head, data);
    }

    Node* evenHead = nullptr;
    Node* oddHead = nullptr;

    splitListByParity(head, evenHead, oddHead);

    // std::cout << "Original List: ";
    // displayList(head->next);

    std::cout << "Even List: ";
    displayList(evenHead);

    std::cout << "Odd List: ";
    displayList(oddHead);

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

    return 0;
}
```

Problem Statement

You are working on a text editing application, and you need to implement a feature that allows users to insert a character at a specific index in the text. You decide to implement this feature using a grounded header linked list to efficiently manage the text.

Note: This is a sample question asked in a Capgemini interview.

Input format :

The first line of input consists of an integer **n,** representing the number of characters.
The second line consists of **n** space-separated characters, representing the initial characters in the list.
The third line consists of an integer **index** representing the position for insertion.
The fourth line consists of a character to be inserted at the specified index.

Output format :

For each insertion operation, display the updated list after inserting the character, at the specified index. (index starts from 0)
Print "Invalid position" for invalid input numbers.

Refer to the sample output for formatting specifications.

Code constraints :

The number of characters n ($0 \leq n \leq 100$).
Each character is an alphanumeric ASCII character (a-z, A-Z, 0-9).

Sample test cases :

Input 1 :

```
5
A B C D E
2
X
```

Output 1 :

```
Updated list: A B X C D E
```

Input 2 :

```
3
P Q R
0
S
```

Output 2 :

```
Updated list: S P Q R
```

Input 3 :

```
7
p q r s t u v
10
l
```

Output 3 :

```
Invalid position.
Updated list: p q r s t u v
```

```
#include <iostream>


struct Node {

   char data;

   Node* next;

};
```

```
Node* createNode(char value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}

void insertAfterPosition(Node* head, int position, char value) {
    Node* newNode = createNode(value);

    Node* current = head;
    for (int i = 0; i < position; i++) {
        if (current->next == nullptr) {
            std::cout << "Invalid position." << std::endl;
            delete newNode;
            return;
        }
        current = current->next;
    }

    newNode->next = current->next;
    current->next = newNode;
}

void displayList(Node* head) {
    Node* current = head->next;
    while (current != nullptr) {
        std::cout << current->data << " ";
        current = current->next;
    }
```

```
    std::cout << std::endl;
}

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

int main() {
    Node* head = createNode('\0'); // Grounded header node
    int n;
    char value;

    std::cin >> n;

    for (int i = 0; i < n; i++) {
        std::cin >> value;
        insertAfterPosition(head, i, value);
    }

    int position;
    std::cin >> position;

    std::cin >> value;

    insertAfterPosition(head, position, value);
    std::cout << "Updated list: ";
```

```
    displayList(head);


    deleteList(head);


    return 0;
}
```

Problem Statement

You are developing a program for a school to manage student attendance records. As part of the attendance management functionality, you need to implement a program that deletes all occurrences of duplicate student IDs from a grounded header linked list.

In this scenario, the school maintains an attendance list represented as a grounded header-linked list. Each node in the list represents a student's attendance record and contains the student's ID as data. The first node serves as the header node and does not contain any actual attendance data. The subsequent nodes hold the student attendance records.

Note: This is a sample question asked in CTS recruitment.
Input format :
The first line of input consists of the number of elements **n** in the list.
The second line of input consists of **n** elements, separated by space.
The third line of input consists of the number to delete.
Output format :
The first line of output prints the original list.
The second line of output prints the list after deleting all occurrences of the given number.

Refer to the sample output for formatting specifications.
Sample test cases :
Input 1 :

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

Output 1 :

```
Original List: 1 2 3 3 3
List after deleting all occurrences of 3: 1 2
```

Input 2 :

```
6
1 1 1 1 1 12
1
```

Output 2 :

```
Original List: 1 1 1 1 1 12
List after deleting all occurrences of 1: 12
```

```
#include <iostream>
```

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

// Function to insert a new node at the end of a linked list
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;
    }
}

// Function to display the linked list
void displayList(Node* head) {
    Node* temp = head->next;
    while (temp != nullptr) {
        std::cout << temp->data << " ";
        temp = temp->next;
    }
    std::cout << std::endl;
}
```

```
// Function to delete all occurrences of a specific value from the linked list
void deleteValue(Node*& head, int value) {
    if (head == nullptr || head->next == nullptr) {
        return;
    }

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

int main() {
    Node* head = new Node();
    head->next = nullptr;

    int n;
    std::cin >> n;
    for (int i = 0; i < n; i++) {
        int data;
        std::cin >> data;
```

```
        insertNode(head, data);
    }

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

    int value;
    std::cin >> value;

    deleteValue(head, value);

    std::cout << "List after deleting all occurrences of " << value << ": ";
    displayList(head);

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

    return 0;
}
```

Problem Statement

You are working on a program to manage the lineup of a sports team.

As part of the lineup management functionality, you need to implement a program that rotates the positions of the team members by a given number of positions to the right. This will allow the coach to reorganize the lineup and make necessary adjustments based on player performance or strategic considerations.

The lineup is represented as a grounded header linked list, where each node contains the player's jersey number. The first node serves as the header node and does not contain any actual player data. The subsequent nodes represent the players in the lineup.

Note: This is a sample question asked in Capgemini recruitment.
Input format :
The first line of input consists of the number of elements **n** in the list.
The second line of input consists of **n** elements, separated by space.
The third line of input consists of the number of positions to rotate right.
Output format :
The first line of output prints the original list.
The second line of output prints the rotated list.

Refer to the sample output for formatting specifications.
Sample test cases :
Input 1 :

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

Output 1 :

```
Original List: 1 2 3 4 5
Rotated List: 3 4 5 1 2
```

Input 2 :

```
6
25 36 95 74 86 12
1
```

Output 2 :

```
Original List: 25 36 95 74 86 12
Rotated List: 12 25 36 95 74 86
```

```
#include <iostream>


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


// Function to insert a new node at the end of a linked list
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;
    }
}

// Function to display the linked list
void displayList(Node* head) {
    Node* temp = head->next;
    while (temp != nullptr) {
        std::cout << temp->data << " ";
        temp = temp->next;
    }
    std::cout << std::endl;
}

// Function to rotate a linked list to the right by a given number of positions
Node* rotateList(Node* head, int positions) {
    if (head == nullptr || head->next == nullptr || positions <= 0) {
        return head;
    }

    int length = 0;
    Node* current = head->next;
    Node* tail = nullptr;
```

```
    // Find the length of the list and locate the tail node
    while (current != nullptr) {
        length++;
        tail = current;
        current = current->next;
    }

    // Adjust the number of positions if it exceeds the length of the list
    positions %= length;

    // No rotation needed if the number of positions is a multiple of the list length
    if (positions == 0) {
        return head;
    }

    // Find the new head and the new tail after rotation
    int count = 0;
    current = head->next;
    while (count < length - positions - 1) {
        current = current->next;
        count++;
    }
    Node* newHead = current->next;
    current->next = nullptr;
    tail->next = head->next;
    head->next = newHead;

    return head;
}

int main() {
```

```
Node* head = new Node();
head->next = nullptr;

int n;
std::cin >> n;
for (int i = 0; i < n; i++) {
    int data;
    std::cin >> data;
    insertNode(head, data);
}

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

int positions;
std::cin >> positions;

Node* rotatedList = rotateList(head, positions);

std::cout << "Rotated List: ";
displayList(rotatedList);

// Free memory
Node* current = rotatedList;
while (current != nullptr) {
    Node* temp = current;
    current = current->next;
    delete temp;
}

return 0;
```

}

Problem Statement

Alice is a detective investigating a mysterious case involving a secret code. She discovered a circular header linked list containing characters. Alice believes that this linked list might be a palindrome, meaning it reads the same forwards and backward.

To validate her hypothesis, Alice needs your help to write a program that checks whether the circular header linked list is indeed a palindrome when considering the entire list in both forward and backward directions.

Write a program that takes the input of a circular header linked list and determines if it is a palindrome. The program should output "Palindrome" if the linked list is a palindrome and "Not Palindrome" otherwise.

Note: This is a sample question asked in Capgemini recruitment.
Input format :
The first line represents the number of elements in the linked list n.
The next n line represents the elements in a linked list.
Output format :
The output represents the palindrome or not.
Sample test cases :
Input 1 :

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

Output 1 :

```
Linked list is not a palindrome.
```

Input 2 :

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

Output 2 :

```
Linked list is a palindrome.
```

```
#include <iostream>

#include <stack>


using namespace std;


struct Node {

   int data;

   Node* next;

};
```

```
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

bool isPalindrome(Node* head) {
    if (head == nullptr)
        return true;

    stack<int> values;
    Node* current = head;
    do {
        values.push(current->data);
        current = current->next;
    } while (current != head);

    current = head;
    while (!values.empty()) {
        if (values.top() != current->data)
            return false;
        values.pop();
        current = current->next;
    }

    return true;
}

int main() {
    int n;
```

```
    cin >> n;

    Node* head = nullptr;
    Node* prev = nullptr;
    for (int i = 0; i < n; i++) {
        int value;
        cin >> value;

        Node* newNode = createNode(value);
        if (head == nullptr) {
            head = newNode;
            newNode->next = head;
        } else {
            prev->next = newNode;
            newNode->next = head;
        }
        prev = newNode;
    }

    if (isPalindrome(head))
        cout << "Linked list is a palindrome.";
    else
        cout << "Linked list is not a palindrome.";

    return 0;
}
```

Problem Statement

Your task is to create a program that handles circular linked lists and supports the insertion of nodes at the end and the deletion of nodes containing a specified value. The program should take user inputs to build the circular linked list, display it before and after the deletion operation, and manage the deletion of nodes as per the provided value.

Note: This is a sample question asked in CTS recruitment.
Input format :
The first line of input consists of an integer **n**, representing the size of the list.
The second line consists of **n** space-separated integers, representing the elements to be inserted at the end of the list.
The last line consists of the element to be deleted from the list.
Output format :
The output represents the Circular Header linked list before deletion and after deletion of the given element.
If the element is found in the list, print "Value not found in the linked list!"

Refer to the sample output for formatting specifications.
Sample test cases :
Input 1 :

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

Output 1 :

```
Linked list before deletion
1 2 3 4 5
Linked list after deletion
1 2 4 5
```

Input 2 :

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

Output 2 :

```
Linked list before deletion
1 2 3 4 5
Value not found in the linked list!
Linked list after deletion
1 2 3 4 5
```

```
#include <iostream>

using namespace std;

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

// Function to create a new node
Node* createNode(int value) {
    Node* newNode = new Node();
```

```
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}

// Function to insert a node at the end of the linked list
Node* insertNode(Node* head, int value) {
    if (head == nullptr) {
        head = createNode(value);
        head->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = createNode(value);
        temp->next->next = head;
    }
    return head;
}

// Function to delete a node at the end of the linked list
Node* deleteEndNode(Node* head, int value) {
    if (head == nullptr) {
        cout << "Linked list is empty!" << endl;
        return nullptr;
    }

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

```
    // Traverse to the last node
    while (temp->next != head) {
        if (temp->next->data == value) {
            prev = temp;
            delNode = temp->next;
        }
        temp = temp->next;
    }

    if (delNode == nullptr) {
        cout << "Value not found in the linked list!" << endl;
        return head;
    }

    // If the node to be deleted is the head
    if (delNode == head) {
        prev->next = head->next;
        delete head;
        head = prev->next;
    } else {
        prev->next = delNode->next;
        delete delNode;
    }

    return head;
}

// Function to display the linked list
void displayLinkedList(Node* head) {
    if (head == nullptr) {
```

```
        return;
    }

    Node* temp = head;
   // cout << "Linked list: ";
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

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

    for (int i = 0; i < numNodes; i++) {
        int value;
        cin >> value;
        head = insertNode(head, value);
    }
   cout << "Linked list before deletion" << endl;
    displayLinkedList(head);

    int valueToDelete;
    cin >> valueToDelete;

    head = deleteEndNode(head, valueToDelete);
    cout << "Linked list after deletion" << endl;
```

```
    displayLinkedList(head);


    return 0;
}
```

Problem Statement

A Library online form application wants to implement an available book details feature using a Circular Header Linked List. The application allows users to add the books at the end.

Write a program to add elements at the end of the circular header linked list.

Note: This is a sample question asked in Accenture recruitment.
Input format :
The first line represents the size of the list n
The next n lines represent the character elements inside the list.
Output format :
The output represents the circular Header linked list by insertion at the end.
If no elements are inserted, print "Linked List is empty."
Sample test cases :
Input 1 :

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

Output 1 :

```
a b c d e
```

Input 2 :

```
0
```

Output 2 :

```
Linked List is empty.
```

```
#include <iostream>


// Node structure for Circular Header Linked List
struct Node {
    char data;
    Node* next;
};


// Function to create a new node
```

```
Node* createNode(int newData) {
    Node* newNode = new Node;
    newNode->data = newData;
    newNode->next = nullptr;
    return newNode;
}

// Function to insert a node at the end of the circular linked list
void insertAtEnd(Node*& header, char newData) {
    Node* newNode = createNode(newData);

    if (header->next == nullptr) {
        newNode->next = header;
        header->next = newNode;
    } else {
        newNode->next = header->next;
        header->next = newNode;
        header = newNode; // Update header to maintain circularity
    }
}

// Function to display the circular linked list
void displayList(const Node* header) {
    if (header->next == nullptr) {
        std::cout << "Linked List is empty." << std::endl;
        return;
    }

    Node* current = header->next;
    do {
        if(current->data != 0)
```

```
        {
        std::cout << current->data << " ";
        }
        current = current->next;
    } while (current != header->next);

    std::cout << std::endl;
}

// Function to delete the circular linked list to prevent memory leaks
void deleteList(Node*& header) {
    if (header->next == nullptr) {
        delete header;
        header = nullptr;
        return;
    }

    Node* current = header->next;
    Node* nextNode;

    while (current != header) {
        nextNode = current->next;
        delete current;
        current = nextNode;
    }

    delete header;
    header = nullptr;
}

int main() {
```

```
    // Create the circular header node
    Node* header = new Node;
    header->next = nullptr;

    int listSize;
    std::cin >> listSize;

    Node* lastNode = header; // Keep track of the last node

    for (int i = 0; i < listSize; i++) {
        char newData;
        std::cin >> newData;

        // Insert node at the end
        Node* newNode = createNode(newData);
        newNode->next = header;

        lastNode->next = newNode;
        lastNode = newNode;
    }

    displayList(header);

    deleteList(header);

    return 0;
}
```

Problem Statement

You are given the task of developing a program that operates on a circular header linked list, a variant of the standard linked list. In this circular header linked list, the last node points back to the header node, forming a circular structure.

Your goal is to implement a program that inserts elements at the end of the circular linked list and prints the data values of alternate nodes.

Note: This is a sample question asked in TCS recruitment.
Input format :
The first line of input consists of an integer **n**, representing the size of the list.
The second line consists of **n** space-separated integers, representing the elements to be inserted at the end of the list.
Output format :
The output prints the alternate nodes of the circular header linked list.
If no elements are inserted, print "Linked List is empty".

Refer to the sample output for formatting specifications.
Code constraints :
n should be odd
Sample test cases :
Input 1 :

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

Output 1 :

```
Alternate Nodes: 1 3 5
```

Input 2 :

```
0
```

Output 2 :

```
Linked List is empty.
```

```
#include <iostream>


// Node structure for Circular Header Linked List
struct Node {
   int data;
   Node* next;
};


// Function to create a new node
Node* createNode(int newData) {
   Node* newNode = new Node;
   newNode->data = newData;
   newNode->next = nullptr;
   return newNode;
}
```

```
// Function to insert a node to the circular linked list
void insert(Node*& header, int newData) {
    Node* newNode = createNode(newData);

    if (header->next == nullptr) {
        newNode->next = header;
        header->next = newNode;
    } else {
        Node* lastNode = header->next;

        while (lastNode->next != header) {
            lastNode = lastNode->next;
        }

        newNode->next = header;
        lastNode->next = newNode;
    }
}

// Function to print alternate nodes in the circular linked list
void printAlternateNodes(const Node* header) {
    if (header->next == nullptr) {
        std::cout << "Linked List is empty." << std::endl;
        return;
    }

    Node* current = header->next;
    int count = 0;

    std::cout << "Alternate Nodes: ";
```

```
    do {

        if (count % 2 == 0) {
            std::cout << current->data << " ";
        }


        current = current->next;
        count++;
    } while (current != header->next);

    std::cout << std::endl;
}

// Function to delete the circular linked list to prevent memory leaks
void deleteList(Node*& header) {
    if (header->next == nullptr) {
        delete header;
        header = nullptr;
        return;
    }

    Node* current = header->next;
    Node* nextNode;

    while (current != header) {
        nextNode = current->next;
        delete current;
        current = nextNode;
    }
```

```
    delete header;
    header = nullptr;
}

int main() {
    // Create the circular header node
    Node* header = new Node;
    header->next = nullptr;

    int listSize;
    std::cin >> listSize;

    for (int i = 0; i < listSize; i++) {
        int newData;
        std::cin >> newData;

        // Insert node
        insert(header, newData);
    }

    printAlternateNodes(header);

    deleteList(header);

    return 0;
}
```

Problem Statement

In LinkedListia, a kingdom ruled by King LinkedList, a circular header linked list structure was established. It had a unique circular arrangement where the last node pointed back to the header node. In LinkedListia, a circular header linked list structure existed.

Advisor Intersectionia was tasked with finding common elements between two linked lists, **List 1** and **List 2**. They efficiently traversed the circular structure, ensuring no element was missed. The intersection operation succeeded, bringing joy to LinkedListia and inspiring future exploration of data structures.

Design a circular header linked list and perform an intersection operation with two linked lists to find the common elements.

Note: This is a sample question asked in Wipro recruitment.
Input format :
The first line of input consists of the number of test cases.
The first line of each test case represents the number of elements in the linked list n.
The next line of each test case represents the n elements in the linked list.
Output format :
The first line of output prints the list1 elements.
The second line of output prints the list2 elements.
The third line of output prints the intersection elements of both lists.

Sample test cases :
Input 1 :

```
2
4
3 4 5 6
6
5 6 7 8 9 10
```

Output 1 :

```
List 1: 3 4 5 6
List 2: 5 6 7 8 9 10
Intersection: 6 5
```

Input 2 :

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

Output 2 :

```
List 1: 1 2 3 4 5
List 2: 3 4 5 6
Intersection: 5 4 3
```

```
#include <iostream>

#include <unordered_set>


using namespace std;


struct Node {

   int data;

   Node* next;
```

```
};

void append(Node** headRef, int data) {
    Node* newNode = new Node();
    newNode->data = data;

    if (*headRef == nullptr) {
        *headRef = newNode;
        newNode->next = *headRef;
        return;
    }

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

    temp->next = newNode;
    newNode->next = *headRef;
}

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

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

```
    cout << endl;
}

Node* getIntersection(Node** heads, int numLists) {
    unordered_set<int> commonElements;
    Node* current = heads[0];

    // Store elements of the first linked list in the hash table
    do {
        commonElements.insert(current->data);
        current = current->next;
    } while (current != heads[0]);

    // Find common elements among the remaining linked lists
    for (int i = 1; i < numLists; i++) {
        current = heads[i];
        unordered_set<int> tempSet;

        // Find common elements with the hash table
        do {
            if (commonElements.find(current->data) != commonElements.end()) {
                tempSet.insert(current->data);
            }
            current = current->next;
        } while (current != heads[i]);

        // Update the common elements
        commonElements = tempSet;
    }

    // Create a new circular linked list with the common elements
```

```
    Node* result = nullptr;
    Node* tail = nullptr;

    for (int element : commonElements) {
        Node* newNode = new Node();
        newNode->data = element;

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

    return result;
}

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

    Node** heads = new Node*[numLists];

    for (int i = 0; i < numLists; i++) {
        heads[i] = nullptr;

        int numElements;
```

```
        cin >> numElements;

        for (int j = 0; j < numElements; j++) {
            int data;
            cin >> data;
            append(&heads[i], data);
        }
    }

    for (int i = 0; i < numLists; i++) {
        cout << "List " << i + 1 << ": ";
        printList(heads[i]);
    }

    Node* intersection = getIntersection(heads, numLists);

    cout << "Intersection: ";
    printList(intersection);

    return 0;
}
```

DOUBLY LINKED LIST:-

Problem Statement

Implement a software tool that analyzes customer feedback for a product. The tool requires a feature to search for a specific feedback entry in a linked list and replace it with an updated version. Each feedback entry is represented by a unique identifier. Implement a program that allows the user to create a doubly linked list of feedback entries with insertion at the end and perform a search operation to find a specific entry. If the entry is found, the program should replace it with an updated version provided by the user. Finally, the program should display the modified list of feedback entries.

The doubly linked list is implemented as a list with insertion at the end. New entries are appended to the end of the list.
The search operation will start from the beginning of the list and traverse forward until the entry is found.

Note: This is a sample question asked in a Paypal interview.

Input format :

The first line contains an integer, N, representing the number of feedback entries.
N lines follow, each containing a string representing a feedback entry identifier.
The (N+1)-th line contains a string, searchId, representing the feedback entry identifier to search for, and a string, updatedId, representing the updated feedback entry identifier.

Output format :

The program should output a single line displaying the modified list of feedback entries, separated by a space.

Code constraints :

The feedback entries are represented by unique identifiers.
The identifiers are alphanumeric strings.
The program should handle a list of arbitrary length.
The program should be implemented using a doubly linked list data structure.

Sample test cases :

Input 1 :

```
5
ABC123 DEF456 GHI789 JKL987 MNO654
DEF456 XYZ789
```

Output 1 :

```
Modified List: ABC123 XYZ789 GHI789 JKL987 MNO654
```

Input 2 :

```
3
PQR123 STU456 VWX789
STU456 YZA789
```

Output 2 :

```
Modified List: PQR123 YZA789 VWX789
```

Input 3 :

```
6
XYZ123 PQR123 DEF789 GHI123 PQR123 MNO789
PQR123 ABC456
```

Output 3 :

```
Modified List: XYZ123 ABC456 DEF789 GHI123 ABC456 MNO789
```

```
#include <iostream>


struct Node {

   std::string data;

   Node* prev;

   Node* next;

};


void insertAtEnd(Node** head, const std::string& newData) {

   Node* newNode = new Node;

   newNode->data = newData;
```

```
    newNode->prev = nullptr;
    newNode->next = nullptr;

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

void searchAndReplace(Node* head, const std::string& searchData, const std::string& newValue) {
    Node* current = head;
    while (current != nullptr) {
        if (current->data == searchData) {
            current->data = newValue;
        }
        current = current->next;
    }
}

void printList(Node* head) {
    Node* current = head;
    while (current != nullptr) {
        std::cout << current->data << " ";
        current = current->next;
    }
```

```
    std::cout << std::endl;
}

int main() {
    Node* head = nullptr;
    int n;
    std::string searchData, newValue;

    std::cin >> n;

    // Create the doubly linked list
    for (int i = 0; i < n; i++) {
        std::string data;
        std::cin >> data;
        insertAtEnd(&head, data);
    }

    std::cin >> searchData >> newValue;

    // Search and replace element in the list
    searchAndReplace(head, searchData, newValue);

    std::cout << "Modified List: ";
    printList(head);

    return 0;
}
```

Problem Statement

You are developing a software tool for analyzing temperature data collected from various weather stations. The tool requires a feature to reverse the order of temperature values in a linked list representing a sequence of recorded temperatures. Implement a program that allows

the user to input a list of temperatures and create a doubly linked list using the temperature values. The program should then reverse the order of temperatures in the list and display the reversed list as the output.

The program will prompt the user to enter the number of temperature values, followed by the temperature values themselves. It will create a doubly linked list using the entered temperature values, with each temperature value stored in a node. Next, the program will reverse the order of the temperature values in the list using the reverseList function. Finally, it will print the reversed list of temperature values.

This software tool will assist meteorologists and weather analysts in examining temperature trends by providing them with a reversed sequence of recorded temperatures, enabling them to gain insights into temperature fluctuations over time.

Note: This is a sample question asked in a Capgemini interview.
Input format :
The first line contains an integer 'n' representing the number of temperature values.
The next line contains 'n' space-separated integers representing the temperature values.
Output format :
The first line of output consists of the original list of temperature values.
The next line of output consists of a reversed list of temperature values.

Refer to the sample output for formatting specifications.
Code constraints :
The temperature values are integers.
The temperature values can be both positive and negative.
The size of the linked list is determined by the user input.
Sample test cases :
Input 1 :

```
3
-5 0 5
```

Output 1 :

```
Original List: 5 0 -5
Reversed List: -5 0 5
```

Input 2 :

```
5
10 20 30 40 50
```

Output 2 :

```
Original List: 50 40 30 20 10
Reversed List: 10 20 30 40 50
```

```
#include <iostream>


struct Node {

   int data;

   Node* prev;

   Node* next;

};
```

```
void insertAtBeginning(Node** head, int newData) {
    Node* newNode = new Node;
    newNode->data = newData;
    newNode->prev = nullptr;
    newNode->next = *head;

    if (*head != nullptr)
        (*head)->prev = newNode;

    *head = newNode;
}

void reverseList(Node** head) {
    Node* current = *head;
    Node* temp = nullptr;

    while (current != nullptr) {
        // Swap prev and next pointers of the current node
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;

        // Move to the next node
        current = current->prev;
    }

    // Update the head pointer
    if (temp != nullptr)
        *head = temp->prev;
}
```

```
void printList(Node* head) {
    Node* current = head;
    while (current != nullptr) {
        std::cout << current->data << " ";
        current = current->next;
    }
    std::cout << std::endl;
}

int main() {
    Node* head = nullptr;
    int n, data;

    std::cin >> n;

    // Create the doubly linked list
    for (int i = 0; i < n; i++) {
        std::cin >> data;
        insertAtBeginning(&head, data);
    }

    std::cout << "Original List: ";
    printList(head);

    // Reverse the list
    reverseList(&head);

    std::cout << "Reversed List: ";
    printList(head);

    return 0;
```

}

Problem Statement

You are tasked with implementing a system that manages a student database. The database stores student information using a doubly linked list. However, the system requires the ability to split the student database into two equal halves for better management. Your task is to write a program that splits the given doubly linked list of student names into two halves, ensuring that the student information is evenly distributed between the resulting lists. The split should be done in such a way that each resulting list maintains the original order of the student records.

Write a program that takes input in the specified format and splits the doubly linked list of student names into two halves using the "insertAtEnd" function.

Note: This is a sample question asked in Microsoft recruitment.
Input format :
The input begins with a single integer, N, representing the number of students in the database.
The following N lines contain the names of the students, one name per line.
Output format :
The program should output the following:
The first half of the student names after splitting.
The second half of the student names after splitting.
Each list should be displayed as a space-separated string of student names.
If either the first half or the second half is empty, it should be displayed as an empty string.
Code constraints :
The maximum number of students in the database is 100.
Each student's name is a string of alphanumeric characters and can have a maximum length of 100 characters.
The input list of student names cannot be empty.
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>

#include <string>
```

```
struct Node {
    std::string data;
    Node* prev;
    Node* next;
};

// Function to split a doubly linked list into two halves
void splitDoublyLinkedList(Node* head, Node** firstHalf, Node** secondHalf) {
    if (head == nullptr || head->next == nullptr) {
        *firstHalf = head;
        *secondHalf = nullptr;
        return;
    }

    Node* slow = head;
    Node* fast = head->next;

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

    *firstHalf = head;
    *secondHalf = slow->next;
    slow->next = nullptr;
    if (*secondHalf != nullptr) {
        (*secondHalf)->prev = nullptr;
```

```
    }
}

// Function to insert a node at the end of a doubly linked list
void insertAtEnd(Node** head, const std::string& newData) {
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = nullptr;

    if (*head == nullptr) {
        newNode->prev = nullptr;
        *head = newNode;
        return;
    }

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

    lastNode->next = newNode;
    newNode->prev = lastNode;
}

// Function to display a doubly linked list
void displayLinkedList(Node* head) {
    Node* currentNode = head;

    while (currentNode != nullptr) {
        std::cout << currentNode->data << " ";
        currentNode = currentNode->next;
```

```
    }

    std::cout << std::endl;
}

int main() {
    Node* head = nullptr;
    Node* firstHalf = nullptr;
    Node* secondHalf = nullptr;

    int n;
    std::string name;

    std::cin >> n;
    std::cin.ignore(); // Ignore the newline character

    for (int i = 0; i < n; i++) {
        std::getline(std::cin, name);
        insertAtEnd(&head, name);
    }

    //displayLinkedList(head);

    splitDoublyLinkedList(head, &firstHalf, &secondHalf);

    displayLinkedList(firstHalf);

    displayLinkedList(secondHalf);

    return 0;
}
```

Problem Statement:

Imagine you are a software developer working on a critical project for a medical research institute. The project involves analyzing patient data stored in a doubly linked list. One of the tasks assigned is to develop a program that can determine whether a patient's medical history, represented by a doubly linked list, is a palindrome or not.

A palindrome in the context of this project means that the sequence of medical events recorded in the linked list, when read forward or backward, remains the same. It is crucial to identify palindromes in the medical history as they may indicate recurring patterns or symptoms that require special attention.

Write a program that assists in analyzing the patient data by checking if a given doubly linked list, representing a patient's medical history, is a palindrome or not. The program should provide a reliable tool to help identify potential patterns or recurring symptoms that could aid in diagnosing and treating patients effectively.

Note: This is a sample question asked in a Capgemini interview.

Input format :

The first line contains an integer, 'n', representing the number of medical events recorded in the patient's history.

The second line contains 'n' space-separated integers, denoting the medical events in chronological order.

Output format :

If the doubly linked list is a palindrome, output "The patient's medical history is a palindrome".

If the doubly linked list is not a palindrome, output "The patient's medical history is not a palindrome".

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>


struct Node {

   int data;
```

```
    Node* prev;
    Node* next;

    Node(int data) {
        this->data = data;
        this->prev = nullptr;
        this->next = nullptr;
    }
};

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

void insertAtEnd(int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) {
        head = newNode;
        tail = newNode;
    } else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
}

bool isPalindrome() {
    if (head == nullptr) {
        return true;
    }
    Node* front = head;
    Node* back = tail;
```

```
    while (front != back && front->prev != back) {
        if (front->data != back->data) {
            return false;
        }
        front = front->next;
        back = back->prev;
    }
    return true;
}

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

    for (int i = 0; i < n; i++) {
        int data;
        std::cin >> data;
        insertAtEnd(data);
    }

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

    return 0;
}
```