Rajalakshmi Engineering College

Name: Baviri Setty Sai Deevan
Email: 241901016@rajalakshmi.edu.in
Roll no: 241901016
Phone: 8838291380
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_MCQ_Updated

Attempt : 1
Total Mark : 20
Marks Obtained : 15

Section 1 : MCQ

1. Which code snippet correctly deletes a node with a given value from a doubly linked list?

```
void deleteNode(Node** head_ref, Node* del_node) {
   if (*head_ref == NULL || del_node == NULL) {
      return;
   }
   if (*head_ref == del_node) {
      *head_ref = del_node->next;
   }
   if (del_node->next != NULL) {
      del_node->next->prev = del_node->prev;
   }
   if (del_node->prev != NULL) {
      del_node->prev->next = del_node->next;
   }
```

```
    free(del_node);
}
```

Answer

Deletes the node at a given position in a doubly linked list.

Status : Wrong ***Marks : 0/1***

2. What is the correct way to add a node at the beginning of a doubly linked list?

Answer

void addFirst(int data){   Node* newNode = new Node(data);   newNode-&gt;next = head;   if (head != NULL) {   head-&gt;prev = newNode;   }   head = newNode;   }

Status : Correct ***Marks : 1/1***

3. Which of the following statements correctly creates a new node for a doubly linked list?

Answer

struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));

Status : Correct ***Marks : 1/1***

4. What does the following code snippet do?

```
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;
newNode->prev = NULL;
```

Answer

Creates a new node and initializes its data to 'value'

Status : Correct ***Marks : 1/1***

5. How do you delete a node from the middle of a doubly linked list?

Answer

All of the mentioned options

Status : Correct ***Marks : 1/1***

6. What is a memory-efficient double-linked list?

Answer

Each node has only one pointer to traverse the list back and forth

Status : Wrong ***Marks : 0/1***

7. What will be the output of the following code?

```
#include <stdio.h>
#include <stdlib.h>

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

int main() {
   struct Node* head = NULL;
   struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
   temp->data = 2;
   temp->next = NULL;
   temp->prev = NULL;
   head = temp;
   printf("%d\n", head->data);
   free(temp);
   return 0;
}
```

Answer

2

Status : Correct ***Marks : 1/1***

8. Which of the following is false about a doubly linked list?

Answer

Implementing a doubly linked list is easier than singly linked list

Status : Correct ***Marks : 1/1***

9. Which of the following is true about the last node in a doubly linked list?

Answer

Its next pointer is NULL

Status : Correct ***Marks : 1/1***

10. Which pointer helps in traversing a doubly linked list in reverse order?

Answer

prev

Status : Correct ***Marks : 1/1***

11. What will be the output of the following program?

```
#include <stdio.h>
#include <stdlib.h>

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

int main() {
    struct Node* head = NULL;
    struct Node* tail = NULL;
    for (int i = 0; i < 5; i++) {
        struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = i + 1;
        temp->prev = tail;
        temp->next = NULL;
        if (tail != NULL) {
            tail->next = temp;
        } else {
            head = temp;
        }
        tail = temp;
    }
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    return 0;
}
```

Answer

1 2 3 4 5

Status : Correct ***Marks : 1/1***

12. How many pointers does a node in a doubly linked list have?

Answer

2

Status : Correct ***Marks : 1/1***

13. What is the main advantage of a two-way linked list over a one-way linked list?

Answer

Two-way linked lists allow for traversal in both directions.

Status : Correct ***Marks : 1/1***

14. Which of the following information is stored in a doubly-linked list's nodes?

Answer

All of the mentioned options

Status : Correct ***Marks : 1/1***

15. How do you reverse a doubly linked list?

Answer

By changing the previous pointer of each node to the next node

Status : Wrong ***Marks : 0/1***

16. What happens if we insert a node at the beginning of a doubly linked list?

Answer

The new node does not have a next pointer

Status : Wrong ***Marks : 0/1***

17. What will be the effect of setting the prev pointer of a node to NULL in a doubly linked list?

Answer

The node will become the new head

Status : Correct ***Marks : 1/1***

18. Consider the provided pseudo code. How can you initialize an empty two-way linked list?

```
Define Structure Node
   data: Integer
   prev: Pointer to Node
   next: Pointer to Node
End Define

Define Structure TwoWayLinkedList
   head: Pointer to Node
   tail: Pointer to Node
End Define
```

Answer

struct TwoWayLinkedList* list = malloc(sizeof(struct TwoWayLinkedList)); list-&gt;head = NULL; list-&gt;tail = NULL;

Status : Correct ***Marks : 1/1***

19. Consider the following function that refers to the head of a Doubly Linked List as the parameter. Assume that a node of a doubly linked list has the previous pointer as prev and the next pointer as next.

Assume that the reference of the head of the following doubly linked list is passed to the below function 1 <--> 2 <--> 3 <--> 4 <--> 5 <-->6. What should be the modified linked list after the function call?

```
Procedure fun(head_ref: Pointer to Pointer of node)
   temp = NULL
   current = *head_ref

   While current is not NULL
      temp = current->prev
      current->prev = current->next
      current->next = temp
      current = current->prev
   End While

   If temp is not NULL
      *head_ref = temp->prev
```

```
    End If
End Procedure
```

Answer

6 &lt;--&gt; 5 &lt;--&gt; 4 &lt;--&gt; 3 &lt;--&gt; 2 &lt;--&gt; 1.

Status : Correct ***Marks : 1/1***

20. Where Fwd and Bwd represent forward and backward links to the adjacent elements of the list. Which of the following segments of code deletes the node pointed to by X from the doubly linked list, if it is assumed that X points to neither the first nor the last node of the list?

A doubly linked list is declared as

```
struct Node {
      int Value;
      struct Node *Fwd;
      struct Node *Bwd;
);
```

Answer

X-&gt;Bwd.Fwd = X-&gt;Fwd ; X.Fwd-&gt;Bwd = X-&gt;Bwd;

Status : Wrong ***Marks : 0/1***

Rajalakshmi Engineering College

Name: Baviri Setty Sai Deevan
Email: 241901016@rajalakshmi.edu.in
Roll no: 241901016
Phone: 8838291380
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Your task is to create a program to manage a playlist of items. Each item is represented as a character, and you need to implement the following operations on the playlist.

Here are the main functionalities of the program:

Insert Item: The program should allow users to add items to the front and end of the playlist. Items are represented as characters.Display Playlist: The program should display the playlist containing the items that were added.

To implement this program, a doubly linked list data structure should be used, where each node contains an item character.

Input Format

The input consists of a sequence of space-separated characters, representing the items to be inserted into the doubly linked list.

The input is terminated by entering - (hyphen).

Output Format

The first line of output prints "Forward Playlist: " followed by the linked list after inserting the items at the end.

The second line prints "Backward Playlist: " followed by the linked list after inserting the items at the front.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: a b c -

Output: Forward Playlist: a b c
Backward Playlist: c b a

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   char item;
   struct Node* next;
   struct Node* prev;
};

void insertAtEnd(struct Node** head, char item){
   struct Node* newnode = (struct Node*)malloc(sizeof(struct Node));
   newnode -> item = item;
   newnode -> next = NULL;
   newnode -> prev = NULL;
   if(*head == NULL){
      *head = newnode;
      newnode -> prev = NULL;
      return;
   }struct Node* temp = *head;
```

```
    while (temp -> next != NULL){
        temp = temp -> next;
    }
    temp -> next = newnode;
    newnode -> prev = temp;
}

void displayForward(struct Node* head){
    struct Node* temp = head;
    while(temp != NULL){
        printf("%c ",temp -> item);
        temp = temp -> next;
    }
}

void displayBackward(struct Node* tail){
    struct Node* temp = tail;
    while(temp != NULL){
        printf("%c ",temp -> item);
        temp = temp -> prev;
    }
}

void freePlaylist(struct Node* head){
    struct Node* temp = head;
    while(temp != NULL){
        struct Node* Next = temp -> next;
        free(temp);
        temp = Next;
    }
}

int main() {
    struct Node* playlist = NULL;
    char item;

    while (1) {
        scanf(" %c", &item);
        if (item == '-') {
            break;
        }
        insertAtEnd(&playlist, item);
    }
    struct Node* tail = playlist;
    while (tail->next != NULL) {
        tail = tail->next;
    }

    printf("Forward Playlist: ");
    displayForward(playlist);

    printf("Backward Playlist: ");
    displayBackward(tail);

    freePlaylist(playlist);

    return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: Baviri Setty Sai Deevan
Email: 241901016@rajalakshmi.edu.in
Roll no: 241901016
Phone: 8838291380
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Moniksha, a chess coach organizing a tournament, needs a program to manage participant IDs efficiently. The program maintains a doubly linked list of IDs and offers two functions: Append to add IDs as students register, and Print Maximum ID to identify the highest ID for administrative tasks.

This tool streamlines tournament organization, allowing Moniksha to focus on coaching her students effectively.

Input Format

The first line consists of an integer n, representing the number of participant IDs to be added.

The second line consists of n space-separated integers representing the participant IDs.

Output Format

The output displays a single integer, representing the maximum participant ID.

If the list is empty, the output prints "Empty list!".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3
163 137 155
Output: 163

Answer

```
#include <stdio.h>
#include <stdlib.h>

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

struct node* createnode(int data){
   struct node* newnode = (struct node*)malloc(sizeof(struct node));
   newnode -> data = data;
   newnode -> next = NULL;
   return newnode;
}

void insertAtEnd(struct node** head, int data){
   struct node* newnode = createnode(data);
   if (*head == NULL){
      *head = newnode;
      return;
   }else{
      struct node* temp = *head;
      while(temp -> next != NULL){
         temp = temp -> next;
      }
```

```
        temp -> next = newnode;
        newnode -> next = NULL;
    }
}

int search(struct node* head){
    struct node* temp = head;
    int max = head -> data;
    while (temp != NULL){
        if (temp -> data > max){
            max = temp -> data;
        }
        temp = temp -> next;
    }
    return max;
}

void display(struct node* head){
    if (head == NULL){
        printf("Empty list!");
        return;
    }else{
        int maximal = search(head);
        printf("%d",maximal);
    }
}

int main(){
    int n;
    scanf("%d",&n);

    struct node* head = NULL;
    for (int i = 0 ; i < n ; i++){
        int id;
        scanf("%d",&id);
        insertAtEnd(&head, id);
    }
    display(head);
    return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: Baviri Setty Sai Deevan
Email: 241901016@rajalakshmi.edu.in
Roll no: 241901016
Phone: 8838291380
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Bob is tasked with developing a company's employee record management system. The system needs to maintain a list of employee records using a doubly linked list. Each employee is represented by a unique integer ID.

Help Bob to complete a program that adds employee records at the front, traverses the list, and prints the same for each addition of employees to the list.

Input Format

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

The second line consists of N space-separated integers, representing the employee IDs.

Output Format

For each employee ID, the program prints "Node Inserted" followed by the current state of the doubly linked list in the next line, with the data values of each node separated by spaces.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
101 102 103 104

Output: Node Inserted
101
Node Inserted
102 101
Node Inserted
103 102 101
Node Inserted
104 103 102 101

Answer

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

struct node {
   int info;
   struct node* prev, * next;
};

struct node* start = NULL;

void traverse(){
   struct node* temp = start;
   printf("Node Inserted\n");
   while (temp != NULL){
      printf("%d ",temp -> info);
      temp = temp -> next;
   }
   printf("\n");
}
```

```
struct node* createnode(int data){
   struct node* newnode = (struct node*)malloc(sizeof(struct node));
   newnode -> info = data;
   newnode -> prev = NULL;
   newnode -> next = NULL;
   return newnode;
}

void insertAtFront(int data){
   struct node* newnode = createnode(data);
   newnode -> next = start;
   if (start != NULL){
      start -> prev = newnode;
   }
   start = newnode;
}

int main() {
   int n, data;
   cin >> n;
   for (int i = 0; i < n; ++i) {
      cin >> data;
      insertAtFront(data);
      traverse();
   }
   return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: Baviri Setty Sai Deevan
Email: 241901016@rajalakshmi.edu.in
Roll no: 241901016
Phone: 8838291380
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Ravi is developing a student registration system for a college. To efficiently store and manage the student IDs, he decides to implement a doubly linked list where each node represents a student's ID.

In this system, each student's ID is stored sequentially, and the system needs to display all registered student IDs in the order they were entered.

Implement a program that creates a doubly linked list, inserts student IDs, and displays them in the same order.

Input Format

The first line contains an integer N the number of student IDs.

The second line contains N space-separated integers representing the student IDs.

Output Format

The output should display the single line containing N space-separated integers representing the student IDs stored in the doubly linked list.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 20 30 40 50

Output: 10 20 30 40 50

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct node{
   int id;
   struct node* prev;
   struct node* next;
};

struct node* start = NULL;

struct node* createnode(int data){
   struct node* newnode = (struct node*)malloc(sizeof(struct node));
   newnode -> id = data;
   newnode -> prev = NULL;
   newnode -> next = NULL;
   return newnode;
}

void insertAtEnd(int data){
   struct node* newnode = createnode(data);
   if (start == NULL){
      start = newnode;
   }else{
        struct node* temp = start;
        while(temp -> next != NULL){
            temp = temp -> next;
        }
        temp -> next = newnode;
        newnode -> prev = temp;
    }
}

void traverse(){
    struct node* temp = start;
    while(temp != NULL){
        printf("%d ",temp -> id);
        temp = temp -> next;
    }
    printf("\n");
}

int main(){
    int n;
    scanf("%d",&n);
    for (int i = 0 ; i < n ; i++){
        int value;
        scanf("%d",&value);
        insertAtEnd(value);
    }
    traverse();
    return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: Baviri Setty Sai Deevan
Email: 241901016@rajalakshmi.edu.in
Roll no: 241901016
Phone: 8838291380
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Ashwin is tasked with developing a simple application to manage a list of items in a shop inventory using a doubly linked list. Each item in the inventory has a unique identification number. The application should allow users to perform the following operations:

Create a List of Items: Initialize the inventory with a given number of items. Each item will be assigned a unique number provided by the user and insert the elements at end of the list.

Delete an Item: Remove an item from the inventory at a specific position.

Display the Inventory: Show the list of items before and after deletion.

If the position provided for deletion is invalid (e.g., out of range), it should display an error message.

Input Format

The first line contains an integer n, representing the number of items to be initially entered into the inventory.

The second line contains n integers, each representing the unique identification number of an item separated by spaces.

The third line contains an integer p, representing the position of the item to be deleted from the inventory.

Output Format

The first line of output prints "Data entered in the list:" followed by the data values of each node in the doubly linked list before deletion.

If p is an invalid position, the output prints "Invalid position. Try again."

If p is a valid position, the output prints "After deletion the new list:" followed by the data values of each node in the doubly linked list after deletion.

Refer to the sample output for the formatting specifications.

Sample Test Case

```
Input: 4
1 2 3 4
5
Output: Data entered in the list:
 node 1 : 1
 node 2 : 2
 node 3 : 3
 node 4 : 4
Invalid position. Try again.
```

Answer

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct node{
   int data;
   struct node* prev;
   struct node* next;
};

struct node* head = NULL;

void display(){
   struct node* temp = head;
   int i = 1;
   while(temp != NULL){
      printf("node %d : %d\n",i,temp -> data);
      temp = temp -> next;
      i++;
   }
   printf("\n");
}

struct node* createnode(int data){
   struct node* newnode = (struct node* )malloc(sizeof(struct node));
   newnode -> data = data;
   newnode -> prev = NULL;
   newnode -> next = NULL;
   return newnode;
}

void insertAtEnd(int data){
   struct node* newnode = createnode(data);
   if (head == NULL){
      head = newnode;
   }else{
      struct node* temp = head;
      while (temp -> next != NULL){
         temp = temp -> next;
      }
      temp -> next = newnode;
      newnode -> prev = temp;
   }
}

void deleteAtPos(int pos){
    if (head == NULL){
        printf("Empty List\n");
        return;
    }
    struct node* temp = head;
    if (pos == 1){
        head = head -> next;
        free(temp);
        return;
    }
    for (int i = 0 ; i < pos - 1 && temp != NULL ; i++){
        temp = temp -> next;
    }
    if (temp == NULL){
        printf("Invalid position. Try again.");
        return;
    }
    if (temp -> next != NULL){
        temp -> next -> prev = temp -> prev;
    }
    if (temp -> prev != NULL){
        temp -> prev -> next = temp -> next;
    }
    free(temp);
    printf("After deletion the new list:\n");
    display();
}

int main(){
    int n;
    scanf("%d",&n);
    for (int i = 0 ; i < n ; i++){
        int value;
        scanf("%d",&value);
        insertAtEnd(value);
    }printf("Data entered in the list:\n");
    display();
    int pos_to_delete;
    scanf("%d",&pos_to_delete);
    deleteAtPos(pos_to_delete);
    return 0;
}
```

Status : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: Baviri Setty Sai Deevan
Email: 241901016@rajalakshmi.edu.in
Roll no: 241901016
Phone: 8838291380
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 30

Section 1 : Coding

1. Problem Statement

Sam is learning about two-way linked lists. He came across a problem where he had to populate a two-way linked list and print the original as well as the reverse order of the list. Assist him with a suitable program.

Input Format

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

The second line consists of n space-separated integers, representing the elements.

Output Format

The first line displays the message: "List in original order:"

The second line displays the elements of the doubly linked list in the original order.

The third line displays the message: "List in reverse order:"

The fourth line displays the elements of the doubly linked list in reverse order.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 4 5
Output: List in original order:
1 2 3 4 5
List in reverse order:
5 4 3 2 1

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct node{
   int data;
   struct node* prev = NULL;
   struct node* next = NULL;
}node;

struct node* head = NULL, *tail = NULL;

struct node* createnode(int data){
   struct node* newnode = (struct node* )malloc(sizeof(struct node));
   newnode -> data = data;
   newnode -> prev = NULL;
   newnode -> next = NULL;
   return newnode;
}

void insertAtEnd(int data){
   struct node* newnode = createnode(data);
```

```
    if (head == NULL){
        head = newnode;
    }else{
        struct node* temp = head;
        while(temp -> next != NULL){
            temp = temp -> next;
        }
        temp -> next = newnode;
        newnode -> prev = temp;
    }
}

void frontprint(){
    struct node* temp = head;
    printf("List in original order:\n");
    while(temp != NULL){
        printf("%d ",temp -> data);
        temp = temp -> next;
    }
    printf("\n");
}

void backprint(){
    struct node* temp = head;
    while(temp -> next != NULL){
        temp = temp -> next;
    }
    tail = temp;
    struct node* temp2 = tail;
    printf("List in reverse order:\n");
    while(temp2 != NULL){
        printf("%d ",temp2 -> data);
        temp2 = temp2 -> prev;
    }
    printf("\n");
}

int main(){
    int n;
    scanf("%d",&n);
    for (int i = 0 ; i < n ; i++){
        int value;
        scanf("%d",&value);
        insertAtEnd(value);
    }
    frontprint();
    backprint();
    return 0;
}
```

Status : Correct ***Marks : 10/10***

2. Problem Statement

You are required to implement a program that deals with a doubly linked list.

The program should allow users to perform the following operations:

Insertion at the End: Insert a node with a given integer data at the end of the doubly linked list.Insertion at a given Position: Insert a node with a given integer data at a specified position within the doubly linked list.Display the List: Display the elements of the doubly linked list.

Input Format

The first line of input consists of an integer n, representing the number of elements to be initially inserted into the doubly linked list.

The second line consists of n space-separated integers, denoting the elements to be inserted at the end.

The third line consists of integer m, representing the new element to be inserted.

The fourth line consists of an integer p, representing the position at which the new element should be inserted (1-based indexing).

Output Format

If p is valid, display the elements of the doubly linked list after performing the insertion at the specified position.

If p is invalid, display "Invalid position" in the first line and the second line prints the original list.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 25 34 48 57
35
4

Output: 10 25 34 35 48 57

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct node{
   int data;
   struct node* prev;
   struct node* next;
}node;

struct node* head = NULL;

struct node* createnode(int data){
   struct node* newnode = (struct node*)malloc(sizeof(struct node));
   newnode -> data = data;
   newnode -> prev = NULL;
   newnode -> next = NULL;
   return newnode;
}

void insertAtEnd(int data){
   struct node* newnode = createnode(data);
   if (head == NULL){
      head = newnode;
   }else{
      struct node* temp = head;
      while (temp -> next != NULL){
         temp = temp -> next;
      }
      temp -> next = newnode;
```

```
        newnode -> prev = temp;
    }
}

void insertAtbeg(int data){
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode -> prev = NULL;
    newnode -> data = data;
    newnode -> next = head;
    if (head != NULL){
        head -> prev = newnode;
        head = newnode;
    }
}

void display(){
    struct node* temp = head;
    while (temp != NULL){
        printf("%d ",temp -> data);
        temp = temp -> next;
    }
    printf("\n");
}

void insertAtPos(int data, int pos){
    if (pos < 1){
        printf("Invalid Postion\n");
        display();
        return;
    }
    if (pos == 1){
        insertAtbeg(data);
        display();
    }
    struct node* newnode = createnode(data);
    struct node* temp = head;
    for (int i = 1 ; i < pos - 1 && temp != NULL ; i++){
        temp = temp -> next;
    }
    if (temp == NULL){
        printf("Invalid Position\n");
        display();
        return;
    }
    newnode -> next = temp -> next;
    newnode -> prev = temp;
    if (temp -> next != NULL){
        temp -> next -> prev = newnode;
    }
    temp -> next = newnode;
    display();
}

int main(){
    int n;
    scanf("%d",&n);
    for (int i = 0 ; i < n ; i++){
        int data;
        scanf("%d",&data);
        insertAtEnd(data);
    }
    int newdata,newpos;
    scanf("%d",&newdata);
    scanf("%d",&newpos);
    insertAtPos(newdata, newpos);
    return 0;
}
```

Status : Correct ***Marks : 10/10***

3. Problem Statement

Imagine Anu is tasked with finding the middle element of a doubly linked list. Given a doubly linked list where each node contains an integer value and is inserted at the end, implement a program to find the middle element of the list. If the number of nodes is even, return the middle element pair.

Input Format

The first line of input consists of an integer N, representing the number of nodes in the doubly linked list.

The second line consists of N space-separated integers, representing the values of the nodes in the doubly linked list.

Output Format

The first line of output prints the space-separated elements of the doubly linked list.

The second line prints the middle element(s) of the doubly linked list, depending on whether the number of nodes is odd or even.

Refer to the sample outputs for the formatting specifications.

Sample Test Case

Input: 5
10 20 30 40 50

Output: 10 20 30 40 50
30

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct node{
    int data;
    struct node* prev;
    struct node* next;
}node;

struct node* head = NULL;

struct node* createnode(int data){
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode -> data = data;
    newnode -> prev = NULL;
    newnode -> next = NULL;
    return newnode;
}

void insertAtBeg(int data){
    struct node* newnode = createnode(data);
    if (head == NULL){
        head = newnode;
        return;
    }else{
        struct node* temp = head;
        while (temp -> next != NULL){
            temp = temp -> next;
        }
        temp -> next = newnode;
        newnode -> prev = temp;
    }
}

void display_list(){
    struct node* temp = head;
    while(temp != NULL){
        printf("%d ",temp -> data);
        temp = temp -> next;
    }
    printf("\n");
}

void display_middle(){
    struct node* slow = head;
    struct node* fast = head;
    while (fast != NULL && fast -> next != NULL){
        slow = slow -> next;
        fast = fast -> next -> next;
    }
    if (fast != NULL){
        printf("%d",slow -> data);
    }else{
        printf("%d %d",slow -> prev -> data, slow -> data);
    }
}

int main(){
    int n;
    scanf("%d",&n);
    for (int i = 0 ; i < n ; i++){
        int value;
        scanf("%d",&value);
        insertAtBeg(value);
    }
    display_list();
    display_middle();
    return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: Baviri Setty Sai Deevan
Email: 241901016@rajalakshmi.edu.in
Roll no: 241901016
Phone: 8838291380
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_PAH_modified

Attempt : 2
Total Mark : 5
Marks Obtained : 5

Section 1 : Coding

1. Problem Statement

Bharath is very good at numbers. As he is piled up with many works, he decides to develop programs for a few concepts to simplify his work. As a first step, he tries to arrange even and odd numbers using a linked list. He stores his values in a singly-linked list.

Now he has to write a program such that all the even numbers appear before the odd numbers. Finally, the list is printed in such a way that all even numbers come before odd numbers. Additionally, the even numbers should be in reverse order, while the odd numbers should maintain their original order.

Example

Input:

6

3 1 0 4 30 12

Output:

12 30 4 0 3 1

Explanation:

Even elements: 0 4 30 12

Reversed Even elements: 12 30 4 0

Odd elements: 3 1

So the final list becomes: 12 30 4 0 3 1

Input Format

The first line consists of an integer n representing the size of the linked list.

The second line consists of n integers representing the elements separated by space.

Output Format

The output prints the rearranged list separated by a space.

The list is printed in such a way that all even numbers come before odd numbers and the even numbers should be in reverse order, while the odd numbers should maintain their original order.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 6
3 1 0 4 30 12
Output: 12 30 4 0 3 1

Answer

```
#include <stdio.h>
```

```c
#include <stdlib.h>

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

struct LinkedList {
   struct Node* head;
   struct Node* lastNode;
};

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}


void append(struct LinkedList* list, int data) {
   struct Node* newNode = createNode(data);
   if (list->lastNode == NULL) {
      list->head = newNode;
      list->lastNode = newNode;
   } else {
      list->lastNode->next = newNode;
      list->lastNode = newNode;
   }
}

struct Node* getNode(struct LinkedList* list, int index) {
   struct Node* current = list->head;
   for (int i = 0; i < index; i++) {
      if (current == NULL) {
         return NULL;
      }
      current = current->next;
   }
   return current;
}
```

```
struct Node* getPrevNode(struct LinkedList* list, struct Node* refNode) {
   struct Node* current = list->head;
   while (current != NULL && current->next != refNode) {
      current = current->next;
   }
   return current;
}

void insertAtBeginning(struct LinkedList* list, struct Node* newNode) {
   if (list->head == NULL) {
      list->head = newNode;
   } else {
      newNode->next = list->head;
      list->head = newNode;
   }
}

void removeNode(struct LinkedList* list, struct Node* node) {
   struct Node* prevNode = getPrevNode(list, node);
   if (prevNode == NULL) {
      list->head = list->head->next;
   } else {
      prevNode->next = node->next;
   }
}

void display(struct LinkedList* list) {
   struct Node* current = list->head;
   while (current != NULL) {
      printf("%d ", current->data);
      current = current->next;
   }
}

void moveEvenBeforeOdd(struct LinkedList* list) {
   struct Node* current = list->head;
   while (current != NULL) {
      struct Node* temp = current->next;
      if (current->data % 2 == 0) {
         removeNode(list, current);
         insertAtBeginning(list, current);
      }
```

```
        current = temp;
    }
}

int main() {
    struct LinkedList linkedList = {NULL, NULL};
    int n, data;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        append(&linkedList, data);
    }
    moveEvenBeforeOdd(&linkedList);
    display(&linkedList);
    return 0;
}
```

Status : Correct **_Marks : 1/1_**

2. Problem Statement

John is working on evaluating polynomials for his math project. He needs to compute the value of a polynomial at a specific point using a singly linked list representation.

Help John by writing a program that takes a polynomial and a value of x as input, and then outputs the computed value of the polynomial.

Example

Input:

2

13

12

11

1

Output:

36

Explanation:

The degree of the polynomial is 2.

Calculate the value of x2: 13 * 12 = 13.

Calculate the value of x1: 12 * 11 = 12.

Calculate the value of x0: 11 * 10 = 11.

Add the values of x2, x1 and x0 together: 13 + 12 + 11 = 36.

Input Format

The first line of input consists of the degree of the polynomial.

The second line consists of the coefficient x2.

The third line consists of the coefficient of x1.

The fourth line consists of the coefficient x0.

The fifth line consists of the value of x, at which the polynomial should be evaluated.

Output Format

The output is the integer value obtained by evaluating the polynomial at the given value of x.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 2
13
12
11
1

Output: 36

Answer

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

struct PolyTerm {
   int coeff;
   int exp;
   struct PolyTerm* next;
};

struct PolyTerm* createTerm(int coeff, int exp) {
   struct PolyTerm* newTerm = (struct PolyTerm*)malloc(sizeof(struct
PolyTerm));
   newTerm->coeff = coeff;
   newTerm->exp = exp;
   newTerm->next = NULL;
   return newTerm;
}

void addTerm(struct PolyTerm** poly, int coeff, int exp) {
   struct PolyTerm* term = createTerm(coeff, exp);
   if (*poly == NULL) {
      *poly = term;
      return;
   }
   struct PolyTerm* last = *poly;
   while (last->next != NULL) {
      last = last->next;
   }
   last->next = term;
}

int evaluatePoly(struct PolyTerm* poly, int x) {
   int result = 0;
   while (poly != NULL) {
      result += poly->coeff * pow(x, poly->exp);
      poly = poly->next;
   }
   return result;
}

int main() {
   int degree, coeff, x;
   struct PolyTerm* poly = NULL;
   scanf("%d", &degree);
   for (int i = degree; i >= 0; i--) {
      scanf("%d", &coeff);
      if (coeff != 0) {
          addTerm(&poly, coeff, i);
      }
   }
   scanf("%d", &x);
   int result = evaluatePoly(poly, x);
   printf("%d\n", result);
   struct PolyTerm* temp;
   while (poly != NULL) {
      temp = poly;
      poly = poly->next;
      free(temp);
   }
   return 0;
}
```

Status : Correct ***Marks : 1/1***

3. Problem Statement

Emily is developing a program to manage a singly linked list. The program should allow users to perform various operations on the linked list, such as inserting elements at the beginning or end, deleting elements from the beginning or end, inserting before or after a specific value, and deleting elements before or after a specific value. After each operation, the updated linked list should be displayed.

Your task is to help Emily in implementing the same.

Input Format

The first line contains an integer choice, representing the operation to perform:

- For choice 1 to create the linked list. The next lines contain space-separated integers, with -1 indicating the end of input.
- For choice 2 to display the linked list.
- For choice 3 to insert a node at the beginning. The next line contains an integer data representing the value to insert.
- For choice 4 to insert a node at the end. The next line contains an integer data representing the value to insert.
- For choice 5 to insert a node before a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 6 to insert a node after a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 7 to delete a node from the beginning.
- For choice 8 to delete a node from the end.
- For choice 9 to delete a node before a specific value. The next line contains an integer value representing the node before which deletion occurs.
- For choice 10 to delete a node after a specific value. The next line contains an integer value representing the node after which deletion occurs.
- For choice 11 to exit the program.

Output Format

For choice 1, print "LINKED LIST CREATED".

For choice 2, print the linked list as space-separated integers on a single line. If the list is empty, print "The list is empty".

For choice 3, 4, 5, and 6, print the updated linked list with a message indicating the insertion operation.

For choice 7, 8, 9, and 10, print the updated linked list with a message indicating the deletion operation.

For any operation that is not possible print an appropriate error message such as "Value not found in the list".

For choice 11 terminate the program.

For any invalid option, print "Invalid option! Please try again".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
5
3
7
-1
2
11

Output: LINKED LIST CREATED
5 3 7

Answer

```
#include <stdio.h>
#include <stdlib.h>

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

struct Node* head = NULL;

void createList() {
   struct Node* newNode, * temp;
   int data;
   scanf("%d", &data);
   while (data != -1) {
      newNode = (struct Node*)malloc(sizeof(struct Node));
      newNode->data = data;
      newNode->next = NULL;
      if (head == NULL) {
         head = newNode;
      } else {
         temp = head;
         while (temp->next != NULL) {
            temp = temp->next;
         }
         temp->next = newNode;
      }
      scanf("%d", &data);
   }
```

```
    printf("LINKED LIST CREATED\n");
}

void displayList() {
    struct Node* temp = head;
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void insertAtBeginning(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
    printf("The linked list after insertion at the beginning is:\n");
    displayList();
}

void insertAtEnd(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("The linked list after insertion at the end is:\n");
    displayList();
}
```

```
void insertBeforeValue(int value, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node* temp = head;
   newNode->data = data;
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }
   if (head->data == value) {
      newNode->next = head;
      head = newNode;
      return;
   }
   while (temp->next != NULL && temp->next->data != value) {
      temp = temp->next;
   }
   if (temp->next == NULL) {
      printf("Value not found in the list\n");
   } else {
      newNode->next = temp->next;
      temp->next = newNode;
   }
   printf("The linked list after insertion before a value is:\n");
   displayList();
}

void insertAfterValue(int value, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node* temp = head;
   newNode->data = data;
   while (temp != NULL && temp->data != value) {
      temp = temp->next;
   }
   if (temp == NULL) {
      printf("Value not found in the list\n");
   } else {
      newNode->next = temp->next;
      temp->next = newNode;
   }
   printf("The linked list after insertion after a value is:\n");
   displayList();
```

```
}

void deleteFromBeginning() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
    printf("The linked list after deletion from the beginning is:\n");
    displayList();
}

void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    struct Node* prev;
    if (head->next == NULL) {
        head = NULL;
        free(temp);
        return;
    }
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
    prev->next = NULL;
    free(temp);
    printf("The linked list after deletion from the end is:\n");
    displayList();
}

void deleteBeforeValue(int value) {
    if (head == NULL || head->next == NULL) {
        printf("Operation not possible\n");
        return;
    }
    struct Node* temp = head;
    struct Node* prev = NULL;
    struct Node* prevPrev = NULL;
    while (temp->next != NULL && temp->next->data != value) {
        prevPrev = prev;
        prev = temp;
        temp = temp->next;
    }
    if (temp->next == NULL) {
        printf("Value not found in the list\n");
    } else if (prev == NULL) {
        printf("No node exists before the value\n");
    } else {
        if (prevPrev == NULL) {
            head = temp;
        } else {
            prevPrev->next = temp;
        }
        free(prev);
    }
    printf("The linked list after deletion before a value is:\n");
    displayList();
}

void deleteAfterValue(int value) {
    struct Node* temp = head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("Operation not possible\n");
    } else {
        struct Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
        free(nodeToDelete);
    }
    printf("The linked list after deletion after a value is:\n");
    displayList();
}

int main() {
    int option, data, value;
    while (1) {
```

```
scanf("%d", &option);
switch (option) {
   case 1:
      createList();
      break;
   case 2:
      displayList();
      break;
   case 3:
      scanf("%d", &data);
      insertAtBeginning(data);
      break;
   case 4:
      scanf("%d", &data);
      insertAtEnd(data);
      break;
   case 5:
      scanf("%d", &value);
      scanf("%d", &data);
      insertBeforeValue(value, data);
      break;
   case 6:
      scanf("%d", &value);
      scanf("%d", &data);
      insertAfterValue(value, data);
      break;
   case 7:
      deleteFromBeginning();
      break;
   case 8:
      deleteFromEnd();
      break;
   case 9:
      scanf("%d", &value);
      deleteBeforeValue(value);
      break;
   case 10:
      scanf("%d", &value);
      deleteAfterValue(value);
      break;
   case 11:
      exit(0);
```

```
            default:
                printf("Invalid option! Please try again\n");
        }
    }
    return 0;
}
```

Status : Correct ***Marks : 1/1***

4. Problem Statement

Write a program to manage a singly linked list. The program should allow users to perform various operations on the linked list, such as inserting elements at the beginning or end, deleting elements from the beginning or end, inserting before or after a specific value, and deleting elements before or after a specific value. After each operation, the updated linked list should be displayed.

Input Format

The first line contains an integer choice, representing the operation to perform:

- For choice 1 to create the linked list. The next lines contain space-separated integers, with -1 indicating the end of input.
- For choice 2 to display the linked list.
- For choice 3 to insert a node at the beginning. The next line contains an integer data representing the value to insert.
- For choice 4 to insert a node at the end. The next line contains an integer data representing the value to insert.
- For choice 5 to insert a node before a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 6 to insert a node after a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 7 to delete a node from the beginning.
- For choice 8 to delete a node from the end.
- For choice 9 to delete a node before a specific value. The next line contains an integer value representing the node before which deletion occurs.
- For choice 10 to delete a node after a specific value. The next line contains an integer value representing the node after which deletion occurs.
- For choice 11 to exit the program.

Output Format

For choice 1, print "LINKED LIST CREATED".

For choice 2, print the linked list as space-separated integers on a single line. If the list is empty, print "The list is empty".

For choice 3, 4, 5, and 6, print the updated linked list with a message indicating the insertion operation.

For choice 7, 8, 9, and 10, print the updated linked list with a message indicating the deletion operation.

For any operation that is not possible print an appropriate error message such as "Value not found in the list".

For choice 11 terminate the program.

For any invalid option, print "Invalid option! Please try again".

Refer to the sample output for formatting specifications.

Sample Test Case

```
Input: 1
5
3
7
-1
2
11
Output: LINKED LIST CREATED
5 3 7
```

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;
```

```
    struct Node* next;
};

struct Node* head = NULL;

void createList() {
    struct Node* newNode, * temp;
    int data;
    scanf("%d", &data);
    while (data != -1) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->next = NULL;
        if (head == NULL) {
            head = newNode;
        } else {
            temp = head;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
        scanf("%d", &data);
    }
    printf("LINKED LIST CREATED\n");
}

void displayList() {
    struct Node* temp = head;
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void insertAtBeginning(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
    printf("The linked list after insertion at the beginning is:\n");
    displayList();
}

void insertAtEnd(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("The linked list after insertion at the end is:\n");
    displayList();
}

void insertBeforeValue(int value, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = head;
    newNode->data = data;
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    if (head->data == value) {
        newNode->next = head;
        head = newNode;
        return;
    }
    while (temp->next != NULL && temp->next->data != value) {
        temp = temp->next;
    }
    if (temp->next == NULL) {
        printf("Value not found in the list\n");
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
    }
    printf("The linked list after insertion before a value is:\n");
    displayList();
}

void insertAfterValue(int value, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = head;
    newNode->data = data;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Value not found in the list\n");
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
    }
    printf("The linked list after insertion after a value is:\n");
    displayList();
}

void deleteFromBeginning() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
    printf("The linked list after deletion from the beginning is:\n");
    displayList();
}

void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    struct Node* prev;
    if (head->next == NULL) {
        head = NULL;
        free(temp);
        return;
    }
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
    prev->next = NULL;
    free(temp);
    printf("The linked list after deletion from the end is:\n");
    displayList();
}

void deleteBeforeValue(int value) {
    if (head == NULL || head->next == NULL) {
        printf("Operation not possible\n");
        return;
    }
    struct Node* temp = head;
    struct Node* prev = NULL;
    struct Node* prevPrev = NULL;
    while (temp->next != NULL && temp->next->data != value) {
        prevPrev = prev;
        prev = temp;
        temp = temp->next;
    }
    if (temp->next == NULL) {
        printf("Value not found in the list\n");
    } else if (prev == NULL) {
        printf("No node exists before the value\n");
    } else {
        if (prevPrev == NULL) {
            head = temp;
        } else {
            prevPrev->next = temp;
        }
        free(prev);
    }
    printf("The linked list after deletion before a value is:\n");
    displayList();
}

void deleteAfterValue(int value) {
    struct Node* temp = head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("Operation not possible\n");
    } else {
        struct Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
        free(nodeToDelete);
    }
    printf("The linked list after deletion after a value is:\n");
    displayList();
}

int main() {
    int option, data, value;
    while (1) {
        scanf("%d", &option);
        switch (option) {
            case 1:
                createList();
                break;
            case 2:
                displayList();
                break;
            case 3:
                scanf("%d", &data);
                insertAtBeginning(data);
                break;
            case 4:
                scanf("%d", &data);
                insertAtEnd(data);
                break;
            case 5:
                scanf("%d", &value);
                scanf("%d", &data);
                insertBeforeValue(value, data);
                break;
            case 6:
                scanf("%d", &value);
                scanf("%d", &data);
                insertAfterValue(value, data);
                break;
            case 7:
                deleteFromBeginning();
                break;
            case 8:
                deleteFromEnd();
                break;
            case 9:
                scanf("%d", &value);
                deleteBeforeValue(value);
                break;
            case 10:
                scanf("%d", &value);
                deleteAfterValue(value);
                break;
            case 11:
                exit(0);
            default:
                printf("Invalid option! Please try again\n");
        }
    }
    return 0;
}
```

Status : Correct ***Marks : 1/1***

5. Problem Statement

Imagine you are managing the backend of an e-commerce platform. Customers place orders at different times, and the orders are stored in two separate linked lists. The first list holds the orders from morning, and the second list holds the orders from the evening.

Your task is to merge the two lists so that the final list holds all orders in sequence from the morning list followed by the evening orders, in the same order

Input Format

The first line contains an integer n , representing the number of orders in the morning list.

The second line contains n space-separated integers representing the morning orders.

The third line contains an integer m , representing the number of orders in the evening list.

The fourth line contains m space-separated integers representing the evening orders.

Output Format

The output should be a single line containing space-separated integers representing the merged order list, with morning orders followed by evening orders.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 3
101 102 103
2
104 105

Output: 101 102 103 104 105

Answer

```
#include <stdio.h>
#include <stdlib.h>

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

```
};

void append(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head_ref;
    new_node->data = new_data;
    new_node->next = NULL;
    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }
    while (last->next != NULL)
        last = last->next;

    last->next = new_node;
}

void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
}

void appendList(struct Node** list1, struct Node* list2) {
    if (*list1 == NULL) {
        *list1 = list2;
        return;
    }
    struct Node* temp = *list1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = list2;
}

int main() {
    int n, m, order;
    struct Node* morningList = NULL;
    struct Node* eveningList = NULL;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &order);
        append(&morningList, order);
    }
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d", &order);
        append(&eveningList, order);
    }
    appendList(&morningList, eveningList);
    printList(morningList);
    return 0;
}
```

Status : Correct ***Marks : 1/1***