# Rajalakshmi Engineering College

Name: Prajeet V
Email: 240701389@rajalakshmi.edu.in
Roll no: 240701389
Phone: 9363389322
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

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***

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

// Define the structure for a node in the doubly linked list
struct Node {
   int data;
   struct Node* next;
   struct Node* prev;
};

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

// Function to insert a node at the end of the doubly linked list
```

```
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

// Function to print the list in original order
void printOriginal(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Function to print the list in reverse order
void printReverse(struct Node* head) {
    if (head == NULL) return;
    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->prev;
    }
    printf("\n");
}

int main() {
    int n;
    scanf("%d", &n);
    struct Node* head = NULL;
    int data;
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        insertAtEnd(&head, data);
    }

    printf("List in original order:\n");
    printOriginal(head);

    printf("List in reverse order:\n");
    printReverse(head);

    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*

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>


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


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

```
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}


void insertAtPosition(struct Node** head, int data, int position) {
    if (position < 1) {
        printf("Invalid position\n");
        return;
    }
    struct Node* newNode = createNode(data);
    if (position == 1) {
        newNode->next = *head;
        if (*head != NULL) {
            (*head)->prev = newNode;
        }
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    for (int i = 1; i < position - 1; i++) {
        if (temp == NULL) {
            printf("Invalid position\n");
            return;
        }
        temp = temp->next;
    }
    newNode->next = temp->next;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

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

int main() {
    int n, m, p;
    scanf("%d", &n);
    struct Node* head = NULL;
    for (int i = 0; i < n; i++) {
        int data;
        scanf("%d", &data);
        insertAtEnd(&head, data);
    }
    scanf("%d", &m);
    scanf("%d", &p);
    if (p < 1 || p > n + 1) {
        printf("Invalid position\n");
        displayList(head);
    } else {
        insertAtPosition(&head, m, p);
        displayList(head);
    }
    return 0;
}
```

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

3. Problem Statement

Ashiq is developing a ticketing system for a small amusement park. The park issues tickets to visitors in the order they arrive. However, due to a system change, the oldest ticket (first inserted) must be revoked instead of the last one.

To manage this, Ashiq decided to use a doubly linked list-based stack, where:

Pushing adds a new ticket to the top of the stack.Removing the first inserted ticket (removing from the bottom of the stack).Printing the remaining tickets from bottom to top.

***Input Format***

The first line consists of an integer n, representing the number of tickets issued.

The second line consists of n space-separated integers, each representing a ticket number in the order they were issued.

***Output Format***

The output prints space-separated integers, representing the remaining ticket numbers in the order from bottom to top.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 7
24 96 41 85 97 91 13
Output: 96 41 85 97 91 13

***Answer***

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

// Define a doubly linked list node
typedef struct Node {
   int ticket_number;
   struct Node* prev;
```

```
    struct Node* next;
} Node;

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

// Function to push a ticket (append to the tail)
void push(int ticket_number) {
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->ticket_number = ticket_number;
    new_node->prev = tail;
    new_node->next = NULL;

    if (tail) {
        tail->next = new_node;
    } else {
        head = new_node;
    }
    tail = new_node;
}

// Function to remove the oldest ticket (delete from head)
void remove_oldest() {
    if (!head) return;

    Node* temp = head;
    head = head->next;

    if (head) {
        head->prev = NULL;
    } else {
        tail = NULL; // If list is now empty
    }

    free(temp);
}

// Function to print tickets from bottom to top
void print_tickets() {
    Node* current = head;
    while (current) {
        printf("%d ", current->ticket_number);
        current = current->next;
    }
    printf("\n");
}

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

    for (int i = 0; i < n; i++) {
        scanf("%d", &ticket);
        push(ticket);
    }

    remove_oldest(); // Remove the first inserted ticket

    print_tickets(); // Print remaining tickets from bottom to top

    return 0;
}
```

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