# Rajalakshmi Engineering College

Name: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
Branch: REC
Department: I CSE (CS) FB
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 : 19

### Section 1 : MCQ

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

***Answer***

prev

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

2. 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***

3. 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***

4. 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 <--> 5 <--> 4 <--> 3 <--> 2 <--> 1.

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

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

***Answer***

The insertion and deletion of a node take a bit longer

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

6. 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***

7. 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***

8. 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**

9. How do you reverse a doubly linked list?

***Answer***

By swapping the next and previous pointers of each node

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

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

***Answer***

2

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

11. 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 first occurrence of a given data value in a doubly linked list.

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

12. 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->next = head; if (head != NULL) { head->prev = newNode; } head = newNode; }

***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. 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->head = NULL; list->tail = NULL;

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

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

***Answer***

The previous pointer of the new node is NULL

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

16. 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**

17. 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**

18. 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**

19. 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->Bwd->Fwd = X->Fwd; X->Fwd->Bwd = X->Bwd;

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

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

***Answer***

A doubly linked list that uses bitwise AND operator for storing addresses

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

# Rajalakshmi Engineering College

Name: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
Branch: REC
Department: I CSE (CS) FB
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 : 0

### 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;
};

// You are using GCC
void insertAtEnd(struct Node** head, char item) {
   //type your code here
}
void displayForward(struct Node* head) {
    //type your code here
}

void displayBackward(struct Node* tail) {
    //type your code here
```

```
}

void freePlaylist(struct Node* head) {
    //type your code here
}

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 :*** Wrong **Marks : 0/10**

# Rajalakshmi Engineering College

Name: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
Branch: REC
Department: I CSE (CS) FB
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 : 0

### 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***

-

***Status :*** Skipped ***Marks : 0/10***

# Rajalakshmi Engineering College

Name: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
Branch: REC
Department: I CSE (CS) FB
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 : 26

## Section 1 : Coding

1. 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>

// Doubly linked list node
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to push ticket to the top
void push(struct Node** top, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = *top;

    if (*top != NULL) {
        (*top)->next = newNode;
    }

    *top = newNode;
```

```
}

// Function to remove the bottom node (first inserted)
void removeBottom(struct Node** top) {
    if (*top == NULL) return;

    struct Node* temp = *top;

    // Traverse to the bottom (head)
    while (temp->prev != NULL) {
        temp = temp->prev;
    }

    // Remove bottom node
    if (temp->next != NULL) {
        temp->next->prev = NULL;
    } else {
        *top = NULL; // Only one node
    }

    free(temp);
}

// Function to print stack from bottom to top
void printStack(struct Node* top) {
    if (top == NULL) return;

    // Traverse to the bottom
    struct Node* temp = top;
    while (temp->prev != NULL) {
        temp = temp->prev;
    }

    // Print from bottom to top
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
```

```
    int n, ticket;
    struct Node* top = NULL;

    scanf("%d", &n);

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

    // Remove the first inserted ticket (bottom)
    removeBottom(&top);

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

    return 0;
}
```

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

2. Problem Statement

Aarav is working on a program to analyze his test scores, which are stored in a doubly linked list. He needs a solution to input scores into the list and determine the highest score.

Help him by providing code that lets users enter test scores into the doubly linked list and find the maximum score efficiently.

***Input Format***

The first line 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 score to be inserted.

***Output Format***

The output prints an integer, representing the highest score present in the list.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 4
89 71 2 70

Output: 89

***Answer***

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

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

// Function to insert a node at the end of the list
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;

    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 find the maximum score in the list
int findMax(struct Node* head) {
    if (head == NULL) return -1;

    int max = head->data;
    struct Node* temp = head->next;

    while (temp != NULL) {
        if (temp->data > max)
            max = temp->data;
        temp = temp->next;
    }

    return max;
}

int main() {
    int N, score;
    struct Node* head = NULL;

    scanf("%d", &N);

    for (int i = 0; i < N; i++) {
        scanf("%d", &score);
        insertEnd(&head, score);
    }

    int maxScore = findMax(head);
    printf("%d\n", maxScore);

    return 0;
}
```

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

3. Problem Statement

Imagine you're managing a store's inventory list, and some products were accidentally entered multiple times. You need to remove the duplicate products from the list to ensure each product appears only once.

You have an unsorted doubly linked list of product IDs. Some of these product IDs may appear more than once, and your goal is to remove any duplicates.

***Input Format***

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

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

***Output Format***

The output prints the final after removing duplicate nodes, separated by a space.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 10
12 12 10 4 8 4 6 4 4 8
Output: 8 4 6 10 12

***Answer***

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

// Doubly linked list node
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to insert at end
void insertEnd(struct Node** head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
```

```
    newNode->next = NULL;

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

    struct Node* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;

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

// Function to remove duplicates
void removeDuplicates(struct Node** head) {
    struct Node *ptr1 = *head, *ptr2, *dup;

    while (ptr1 != NULL) {
        ptr2 = ptr1->next;
        while (ptr2 != NULL) {
            if (ptr1->data == ptr2->data) {
                dup = ptr2;
                ptr2 = ptr2->next;

                if (dup->prev)
                    dup->prev->next = dup->next;
                if (dup->next)
                    dup->next->prev = dup->prev;

                free(dup);
            } else {
                ptr2 = ptr2->next;
            }
        }
        ptr1 = ptr1->next;
    }
}

// Function to print list in reverse
```

```
void printReverse(struct Node* head) {
    if (head == NULL) return;

    // Move to last node
    while (head->next != NULL)
        head = head->next;

    // Print from end to start
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->prev;
    }
}

int main() {
    int n, x;
    struct Node* head = NULL;

    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &x);
        insertEnd(&head, x);
    }

    removeDuplicates(&head);
    printReverse(head);

    return 0;
}
```

***Status :*** Partially correct **Marks : 6/10**

# Rajalakshmi Engineering College

Name: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

## REC_DS using C_Week 2_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 50

## Section 1 : Coding

1. Problem Statement

Pranav wants to clockwise rotate a doubly linked list by a specified number of positions. He needs your help to implement a program to achieve this. Given a doubly linked list and an integer representing the number of positions to rotate, write a program to rotate the list clockwise.

***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 linked list elements.

The third line consists of an integer k, representing the number of places to rotate the list.

***Output Format***

The output displays the elements of the doubly linked list after rotating it by k positions.

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 5
1 2 3 4 5
1
Output: 5 1 2 3 4

***Answer***

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

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

// Function to insert a node at the end of the doubly linked list
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

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

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

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

// Function to rotate the doubly linked list by k positions
void rotateClockwise(struct Node** head, int k) {
    if (*head == NULL || k == 0) return;

    // Find the length of the list
    struct Node* temp = *head;
    int length = 1;
    while (temp->next != NULL) {
        length++;
        temp = temp->next;
    }

    // If k is greater than or equal to the length, reduce it
    k = k % length;
    if (k == 0) return;

    // Traverse to the (length - k)th node
    struct Node* newTail = *head;
    for (int i = 1; i < length - k; i++) {
        newTail = newTail->next;
    }

    // The new head will be the (length - k + 1)th node
    struct Node* newHead = newTail->next;

    // Update pointers to rotate the list
    newTail->next = NULL;
```

```
    newHead->prev = NULL;
    temp->next = *head;
    (*head)->prev = temp;
    *head = newHead;
}

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

    struct Node* head = NULL;

    // Insert elements into the doubly linked list
    for (int i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        insertEnd(&head, value);
    }

    scanf("%d", &k);

    // Rotate the list by k positions
    rotateClockwise(&head, k);

    // Print the rotated doubly linked list
    printList(head);

    return 0;
}
```

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

2. Problem Statement

Bala is a student learning about the doubly linked list and its functionalities. He came across a problem where he wanted to create a doubly linked list by appending elements to the front of the list.

After populating the list, he wanted to delete the node at the given position from the beginning. Write a suitable code to help Bala.

***Input Format***

The first line contains an integer N, the number of elements in the doubly linked list.

The second line contains N integers separated by a space, the data values of the nodes in the doubly linked list.

The third line contains an integer X, the position of the node to be deleted from the doubly linked list.

***Output Format***

The first line of output displays the original elements of the doubly linked list, separated by a space.

The second line prints the updated list after deleting the node at the given position X from the beginning.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
10 20 30 40 50
2
Output: 50 40 30 20 10
50 30 20 10

***Answer***

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

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

```
// Function to insert a node at the front of the doubly linked list
void insertFront(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    newNode->prev = NULL;

    if (*head != NULL) {
        (*head)->prev = newNode;
    }

    *head = newNode;
}

// Function to delete a node at a specific position (from the beginning)
void deleteNodeAtPosition(struct Node** head, int position) {
    if (*head == NULL) return;

    struct Node* temp = *head;

    // If the head node needs to be removed
    if (position == 1) {
        *head = temp->next;  // Move head to the next node
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
        free(temp);  // Free memory of the old head
        return;
    }

    // Traverse to the node at the given position
    for (int i = 1; temp != NULL && i < position; i++) {
        temp = temp->next;
    }

    // If the position is out of range
    if (temp == NULL) return;

    // Remove the node from the list
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
```

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

    free(temp);  // Free the memory of the node
}

// Function to print the doubly linked list
void printList(struct Node* head) {
    if (head == NULL) {
        printf("The 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, X;

    // Read the number of elements in the list
    scanf("%d", &N);

    struct Node* head = NULL;

    // Read the elements and insert them at the front
    for (int i = 0; i < N; i++) {
        int value;
        scanf("%d", &value);
        insertFront(&head, value);
    }

    // Read the position to delete from the list
    scanf("%d", &X);

    // Print the original list
    printList(head);
```

```
    // Delete the node at the given position
    deleteNodeAtPosition(&head, X);

    // Print the updated list
    printList(head);

    return 0;
}
```

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

3. Problem Statement

Rohan is a software developer who is working on an application that processes data stored in a Doubly Linked List. He needs to implement a feature that finds and prints the middle element(s) of the list. If the list contains an odd number of elements, the middle element should be printed. If the list contains an even number of elements, the two middle elements should be printed.

Help Rohan by writing a program that reads a list of numbers, prints the list, and then prints the middle element(s) based on the number of elements in the list.

***Input Format***

The first line of the input consists of an integer n the number of elements in the doubly linked list.

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

***Output Format***

The first line prints the elements of the list separated by space. (There is an extra space at the end of this line.)

The second line prints the middle element(s) based on the number of elements.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 5
20 52 40 16 18

Output: 20 52 40 16 18
40

***Answer***

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

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

// Function to insert a node at the end of the doubly linked list
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;

    if (*head == NULL) {
        *head = newNode; // The first node
        return;
    }

    // Traverse to the last node
    while (temp->next != NULL) {
        temp = temp->next;
    }

    // Insert the new node at the end
    temp->next = newNode;
```

```
    newNode->prev = temp;
}

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

// Function to print the middle element(s)
void printMiddle(struct Node* head, int n) {
    struct Node* slow = head;
    struct Node* fast = head;

    // Traverse the list with fast and slow pointers
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }

    // If n is odd, slow will be at the middle element
    if (n % 2 == 1) {
        printf("%d\n", slow->data);
    } else {
        // If n is even, print both slow and slow->prev
        printf("%d %d\n", slow->prev->data, slow->data);
    }
}

int main() {
    int n;

    // Read the number of elements in the list
    scanf("%d", &n);

    struct Node* head = NULL;

    // Read the elements and insert them into the doubly linked list
```

```
    for (int i = 0; i < n; i++) {
        int value;
        scanf("%d", &value);
        insertEnd(&head, value);
    }

    // Print the elements of the list
    printList(head);

    // Print the middle element(s)
    printMiddle(head, n);

    return 0;
}
```

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

4. Problem Statement

Tom is a software developer working on a project where he has to check if a doubly linked list is a palindrome. He needs to write a program to solve this problem. Write a program to help Tom check if a given doubly linked list is a palindrome or not.

***Input Format***

The first line 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 linked list elements.

***Output Format***

The first line displays the space-separated integers, representing the doubly linked list.

The second line displays one of the following:

1. If the doubly linked list is a palindrome, print "The doubly linked list is a palindrome".
2. If the doubly linked list is not a palindrome, print "The doubly linked list is not a palindrome".

Refer to the sample output for the formatting specifications.

***Sample Test Case***

Input: 5
1 2 3 2 1

Output: 1 2 3 2 1
The doubly linked list is a palindrome

***Answer***

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

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

// Function to insert a node at the end of the doubly linked list
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

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

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

// Function to check if the doubly linked list is a palindrome
int isPalindrome(struct Node* head) {
    if (head == NULL) {
        return 1;
    }

    struct Node* start = head;
    struct Node* end = head;

    // Traverse to the last node
    while (end->next != NULL) {
        end = end->next;
    }

    // Compare elements from both ends
    while (start != end && start->prev != end) {
        if (start->data != end->data) {
            return 0; // Not a palindrome
        }
        start = start->next;
        end = end->prev;
    }

    return 1; // Is a palindrome
}

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

    struct Node* head = NULL;
```

```
    // Insert elements into the doubly linked list
    for (int i = 0; i < N; i++) {
        int value;
        scanf("%d", &value);
        insertEnd(&head, value);
    }

    // Print the doubly linked list
    printList(head);

    // Check if the list is a palindrome
    if (isPalindrome(head)) {
        printf("The doubly linked list is a palindrome\n");
    } else {
        printf("The doubly linked list is not a palindrome\n");
    }

    return 0;
}
```

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

5. Problem Statement

Riya is developing a contact management system where recently added contacts should appear first. She decides to use a doubly linked list to store contact IDs in the order they are added. Initially, new contacts are inserted at the front of the list. However, sometimes she needs to insert a new contact at a specific position in the list based on priority.

Help Riya implement this system by performing the following operations:

Insert contact IDs at the front of the list as they are added.Insert a new contact at a given position in the list.

***Input Format***

The first line of input consists of an integer N, representing the initial size of the linked list.

The second line consists of N space-separated integers, representing the values

of the linked list to be inserted at the front.

The third line consists of an integer position, representing the position at which the new value should be inserted (position starts from 1).

The fourth line consists of integer data, representing the new value to be inserted.

***Output Format***

The first line of output prints the original list after inserting initial elements to the front.

The second line prints the updated linked list after inserting the element at the specified position.

Refer to the sample output for formatting specifications.

***Sample Test Case***

Input: 4
10 20 30 40
3
25

Output: 40 30 20 10
40 30 25 20 10

***Answer***

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

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

// Function to insert a node at the front of the doubly linked list
void insertFront(struct Node** head, int data) {
```

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

    if (*head != NULL) {
        (*head)->prev = newNode;
    }

    *head = newNode;
}

// Function to insert a node at a specific position
void insertAtPosition(struct Node** head, int position, int data) {
    if (position == 1) {
        insertFront(head, data);
        return;
    }

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    struct Node* temp = *head;

    for (int i = 1; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }

    if (temp == NULL) {
        return;
    }

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

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

    temp->next = newNode;
}

// Function to print the doubly linked list
```

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

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

    struct Node* head = NULL;

    // Insert the initial contacts at the front
    for (int i = 0; i < N; i++) {
        int value;
        scanf("%d", &value);
        insertFront(&head, value);
    }

    // Print the original list
    printList(head);

    // Insert the new contact at the specified position
    int position, data;
    scanf("%d", &position);
    scanf("%d", &data);

    insertAtPosition(&head, position, data);

    // Print the updated list
    printList(head);

    return 0;
}
```

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