

# Rajalakshmi Engineering College

Name: Sudharsan K  
Email: 240801341@rajalakshmi.edu.in  
Roll no:  
Phone: null  
Branch: REC  
Department: I ECE AF  
Batch: 2028  
Degree: B.E - ECE

Scan to verify results



## NeoColab\_REC\_CS23231\_DATA STRUCTURES

### REC\_DS using C\_Week 1\_PAH\_modified

Attempt : 2  
Total Mark : 5  
Marks Obtained : 4.5

#### Section 1 : Coding

##### 1. 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;
```

```
// Function to create the linked list
```

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

```
// Function to display the list
```

```

void display() {
    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");
}

```

// Insert at beginning

```

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");
    display();
}

```

// Insert at end

```

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");
    display();
}

```

// Insert before a specific value

```

void insertBefore(int value, int data) {

```

```

if (head == NULL) {
    printf("Value not found in the list\n");
    printf("The linked list after insertion before a value is:\n");
    display();
    return;
}
if (head->data == value) {
    insertAtBeginning(data);
    return;
}

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

if (temp->next == NULL) {
    printf("Value not found in the list\n");
    printf("The linked list after insertion before a value is:\n");
    display();
} else {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = temp->next;
    temp->next = newNode;
    printf("The linked list after insertion before a value is:\n");
    display();
}
}

// Insert after a specific value
void insertAfter(int value, int data) {
    struct Node* temp = head;
    while (temp != NULL && temp->data != value)
        temp = temp->next;

    if (temp == NULL) {
        printf("Value not found in the list\n");
        printf("The linked list after insertion after a value is:\n");
        display();
    } else {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

```

```

        newNode->data = data;
        newNode->next = temp->next;
        temp->next = newNode;
        printf("The linked list after insertion after a value is:\n");
        display();
    }
}

```

```

// Delete from beginning
void deleteFromBeginning() {
    if (head == NULL) {
        printf("The 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");
    display();
}

```

```

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

```

```

// Delete before a specific value

```

```

void deleteBefore(int value) {
    if (head == NULL || head->next == NULL || head->data == value) {
        printf("Deletion not possible\n");
        return;
    }
    struct Node* prev = NULL;
    struct Node* curr = head;
    while (curr->next != NULL && curr->next->data == value) {
        prev = curr;
        curr = curr->next;
    }
    if (curr->next == NULL) {
        printf("Value not found in the list\n");
        printf("The linked list after deletion after a value is:\n");
        display();
    } else {
        if (prev == NULL) {
            head = curr->next;
        } else {
            prev->next = curr->next;
        }
        free(curr);
        printf("The linked list after deletion before a value is:\n");
        display();
    }
}

```

// Delete after a specific value

```

void deleteAfter(int value) {
    struct Node* temp = head;
    while (temp != NULL && temp->data != value)
        temp = temp->next;

    if (temp == NULL || temp->next == NULL) {
        printf("Deletion not possible\n");
    } else {
        struct Node* toDelete = temp->next;
        temp->next = toDelete->next;
        free(toDelete);
        printf("The linked list after deletion after a value is:\n");
        display();
    }
}

```

```
}
```

```
// Main function
```

```
int main() {  
    int choice, data, value;  
    while (1) {  
        scanf("%d", &choice);  
        switch (choice) {  
            case 1:  
                createList();  
                break;  
            case 2:  
                display();  
                break;  
            case 3:  
                scanf("%d", &data);  
                insertAtBeginning(data);  
                break;  
            case 4:  
                scanf("%d", &data);  
                insertAtEnd(data);  
                break;  
            case 5:  
                scanf("%d %d", &value, &data);  
                insertBefore(value, data);  
                break;  
            case 6:  
                scanf("%d %d", &value, &data);  
                insertAfter(value, data);  
                break;  
            case 7:  
                deleteFromBeginning();  
                break;  
            case 8:  
                deleteFromEnd();  
                break;  
            case 9:  
                scanf("%d", &value);  
                deleteBefore(value);  
                break;  
            case 10:  
                scanf("%d", &value);
```



```

        deleteAfter(value);
        break;
    case 11:
        exit(0);
    default:
        printf("Invalid option! Please try again\n");
    }
}
return 0;
}

```

**Status :** Partially correct

**Marks :** 0.75/1

## 2. 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>
```

```
// Define the Node structure
```

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

```
// Function to create a new node
```

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

```
// Function to append a node at the end
```

```
void append(struct Node** head_ref, int order_id) {  
    struct Node* newNode = createNode(order_id);  
    if (*head_ref == NULL) {  
        *head_ref = newNode;  
        return;  
    }  
    struct Node* temp = *head_ref;  
    while (temp->next != NULL)  
        temp = temp->next;  
    temp->next = newNode;
```

```
}
```

```
// Function to merge two lists: morning followed by evening
struct Node* mergeLists(struct Node* morning, struct Node* evening) {
    if (morning == NULL) return evening;
    struct Node* temp = morning;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = evening;
    return morning;
}
```

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

```
int main() {
    int n, m, order_id;

    // Read morning list
    scanf("%d", &n);
    struct Node* morning = NULL;
    for (int i = 0; i < n; i++) {
        scanf("%d", &order_id);
        append(&morning, order_id);
    }
}
```

```
// Read evening list
scanf("%d", &m);
struct Node* evening = NULL;
for (int i = 0; i < m; i++) {
    scanf("%d", &order_id);
    append(&evening, order_id);
}
```

```
// Merge and print result
struct Node* merged = mergeLists(morning, evening);
```

```
    printList(merged);  
  
    return 0;  
}
```

**Status :** Correct

**Marks :** 1/1

### 3. 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  $x^2$ :  $13 * 12 = 13$ .

Calculate the value of  $x^1$ :  $12 * 11 = 12$ .

Calculate the value of  $x^0$ :  $11 * 10 = 11$ .

Add the values of  $x^2$ ,  $x^1$  and  $x^0$  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  $x_2$ .

The third line consists of the coefficient of  $x_1$ .

The fourth line consists of the coefficient  $x_0$ .

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***

```
// You are using GCC
```

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

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

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

```
// Define the Node structure
```

```
struct Node {
```

```
    int coeff;
```

```
    struct Node* next;
```

```
};
```

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

```
// Function to append a node to the end of the linked list
void append(struct Node** head_ref, int coeff) {
    struct Node* newNode = createNode(coeff);
    if (*head_ref == NULL) {
        *head_ref = newNode;
        return;
    }
    struct Node* temp = *head_ref;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}
```

```
// Function to evaluate the polynomial
int evaluate(struct Node* head, int x, int degree) {
    int result = 0;
    struct Node* temp = head;
    while (temp != NULL) {
        result += temp->coeff * (int)pow(x, degree);
        degree--;
        temp = temp->next;
    }
    return result;
}
```

```
int main() {
    int degree, coeff, x;

    // Input
    scanf("%d", &degree);

    struct Node* head = NULL;

    for (int i = 0; i <= degree; i++) {
```

```

        scanf("%d", &coeff);
        append(&head, coeff);
    }

    scanf("%d", &x);

    // Output
    int result = evaluate(head, x, degree);
    printf("%d\n", result);

    return 0;
}

```

**Status :** Correct

**Marks :** 1/1

#### 4. 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***

```
// You are using GCC
```

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

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

```
// Define a node structure
```

```
struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
};
```



```
// Create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

```
// Insert at head (for reversing the even list)
void insertAtHead(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}
```

```
// Insert at tail (for keeping odd list in order)
void insertAtTail(struct Node** head, 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;
    }
}
```

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

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

    struct Node* evenHead = NULL;
```

```

struct Node* oddHead = NULL;

for (int i = 0; i < n; i++) {
    scanf("%d", &val);
    if (val % 2 == 0)
        insertAtHead(&evenHead, val); // Reversed even
    else
        insertAtTail(&oddHead, val); // Ordered odd
}

// Merge even and odd lists
struct Node* temp = evenHead;
if (temp == NULL) {
    printList(oddHead);
} else {
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = oddHead;
    printList(evenHead);
}

return 0;
}

```

**Status :** Correct

**Marks :** 1/1

## 5. 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**

// You are using GCC

#include <stdio.h>

#include <stdlib.h>

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

```
struct Node* head = NULL;
```

// Create the linked list

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

```

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

```

// Display the list

```

void display() {
    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");
}

```

// Insert at beginning

```

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");
    display();
}

```

// Insert at end

```

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");
}

```

```

    display();
}

// Insert before specific value
void insertBefore(int value, int data) {
    if (head == NULL || head->data == value) {
        insertAtBeginning(data);
        return;
    }

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

    if (temp->next == NULL) {
        printf("Value not found in the list\n");
        printf("The linked list after insertion before a value is:\n");
        display();
    } else {
        struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->next = temp->next;
        temp->next = newNode;
        printf("The linked list after insertion before a value is:\n");
        display();
    }
}

// Insert after specific value
void insertAfter(int value, int data) {
    struct Node* temp = head;
    while (temp != NULL && temp->data != value)
        temp = temp->next;

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

```

```

        temp->next = newNode;
        printf("The linked list after insertion after a value is:\n");
        display();
    }
}

// Delete from beginning
void deleteFromBeginning() {
    if (head == NULL) {
        printf("The 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");
    display();
}

```

```

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

```

```

// Delete before specific value
void deleteBefore(int value) {
    if (head == NULL || head->next == NULL || head->data == value) {

```

```

    printf("Deletion not possible\n");
    return;
}

struct Node* prev = NULL;
struct Node* curr = head;
while (curr->next != NULL && curr->next->data == value) {
    prev = curr;
    curr = curr->next;
}

if (curr->next == NULL) {
    printf("Value not found in the list\n");
    printf("The linked list after deletion before a value is:\n");
    display();
} else {
    if (prev == NULL) {
        head = curr->next;
    } else {
        prev->next = curr->next;
    }
    free(curr);
    printf("The linked list after deletion before a value is:\n");
    display();
}
}

// Delete after specific value
void deleteAfter(int value) {
    struct Node* temp = head;
    while (temp != NULL && temp->data != value)
        temp = temp->next;

    if (temp == NULL || temp->next == NULL) {
        printf("Invalid option! Please try again\n");
    } else {
        struct Node* toDelete = temp->next;
        temp->next = toDelete->next;
        free(toDelete);
        printf("The linked list after deletion after a value is:\n");
        display();
    }
}

```



```
}
```

```
int main() {  
    int choice, data, value;  
  
    while (1) {  
        if (scanf("%d", &choice) != 1) break;  
  
        switch (choice) {  
            case 1:  
                createList();  
                break;  
            case 2:  
                display();  
                break;  
            case 3:  
                scanf("%d", &data);  
                insertAtBeginning(data);  
                break;  
            case 4:  
                scanf("%d", &data);  
                insertAtEnd(data);  
                break;  
            case 5:  
                scanf("%d %d", &value, &data);  
                insertBefore(value, data);  
                break;  
            case 6:  
                scanf("%d %d", &value, &data);  
                insertAfter(value, data);  
                break;  
            case 7:  
                deleteFromBeginning();  
                break;  
            case 8:  
                deleteFromEnd();  
                break;  
            case 9:  
                scanf("%d", &value);  
                deleteBefore(value);  
                break;  
            case 10:
```

```
        scanf("%d", &value);
        deleteAfter(value);
        break;
    case 11:
        exit(0);
    default:
        printf("Invalid option! Please try again\n");
    }
}
return 0;
}
```

**Status :** Partially correct

**Marks :** 0.75/1