```c
// Name : Ayush Gupta
// Rollno : MNS2025027


// 1. Program 4-5, Lab-3 is a menu-based program of
creation, insertion/deletion at desired
// location and traversing the singly linked list. Add an
option where the user can search an
// element in the singly linked list.
#include <stdio.h>
#include <stdlib.h>

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

void insert_at_begin(Node*(*head))
{
    int val;
    printf("Data for the first node : ");
    scanf("%d", &val);

    Node* newNode = malloc(sizeof(Node));
    newNode->data = val;

    if (!(*head))
        (*head) = newNode;
    else
    {
        newNode->next = (*head);
        *head = newNode;
    }
}

void insert_at_pos(Node** head)
{
    if (!*head)
    {
        printf("Sorry the list is empty...Please try
insert at begining..");
        return;
    }

    int val, pos;
    printf("Enter the position and the value: ");
    scanf("%d%d", &pos, &val);

    Node* newNode = malloc(sizeof(Node));
    newNode->data = val;
    newNode->next = NULL;

    Node* temp = *head;

    if (pos == 1)
    {
        *head = newNode;
    }

    for (int i = 0; i < pos - 2; i++)
    {
        if (!temp)
        {
            printf("Invalid Postition!..");
            free(temp);
            return;
        }
        temp = temp->next;
    }

    if (!temp)
    {
        printf("Invalid Postition!..");
        free(temp);
        return;
    }

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

void insert_at_end(int n, Node*(*head), Node** tail)
{
    int val;

    for (int i = 0; i < n; i++)
    {
        printf("Data for the %d node : ", i + 1);
        scanf("%d", &val);

        Node* newNode = malloc(sizeof(Node));
        newNode->data = val;
        newNode->next = NULL;

        if (!(*head))
```

```
            (*head) = *tail = newNode;
        else
        {
            (*tail)->next = newNode;
            *tail = newNode;
        }
    }
}

void printList(Node* head)
{
    Node* temp = head;

    while (temp)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL");
}

void delete_head(Node** head)
{
    if (!(*head))
    {
        printf("Empty List...");
        return;
    }
    Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}
```

```
void delete_tail(Node** head, Node** tail)
{
    if (!(*head)) return;

    Node* temp = *head;

    if (!temp->next)
    {
        printf("%d is deleted..", temp->data);
        free(temp);
        *head = *tail = NULL;
        return;
    }

    while (temp->next->next)
    {
        temp = temp->next;
    }
    Node* deleteNode = temp->next;
    *tail = temp;
    temp->next = NULL;
    printf("%d is deleted from the end...",
deleteNode->data);
    free(deleteNode);
}

void delete_at_pos(Node** head)
{
    if (!(*head))
    {
        printf("List is empty.\n");
```

```c
        return;
    }

    int pos;
    printf("Enter the position to delete..: ");
    if (scanf("%d", &pos) != 1 || pos < 1)
    {
        printf("Invalid position.\n");
        return;
    }

    if (pos == 1)
    {
        delete_head(head);
        return;
    }

    Node* temp = *head;
    for (int i = 1; temp && i < pos - 1; i++)
    {
        temp = temp->next;
    }

    if (!temp || !(temp->next))
    {
        printf("Position out of bounds.\n");
        return;
    }

    Node* deleteNode = temp->next;
    temp->next = deleteNode->next;
    printf("%d deleted...\n", deleteNode->data);
    free(deleteNode);
}

// Lab-4 Q1: Search an element
int search_element(Node* head)
{
    int key, pos = 1;
    printf("\nEnter the element to search: ");
    scanf("%d", &key);

    if (!head)
    {
        printf("Empty List\n");
        return -1;
    }

    Node* p = head;
    while (p)
    {
        if (p->data == key)
        {
            printf("Element found at %d position\n",
pos);
            return pos;
        }
        p = p->next;
        pos++;
    }

    printf("Element not found\n");
    return -1;
}
```

```
int main()
{
    Node *head = NULL, *tail = NULL;
    int n, val, choice;

    do
    {
        printf("\n-----------Menu----------\n");
        printf("Press 1 to insert at the begining: \n");
        printf("Press 2 to insert at the end: \n");
        printf("Press 3 to insert at the position: \n");
        printf("Press 4 to delete the begining element:
\n");
        printf("Press 5 to delete the end element: \n");
        printf("Press 6 to delete the element from any
position: \n");
        printf("Press 7 to view the complete list: \n");
        printf("Press 8 to search an element: \n");
        printf("Press 9 to exit: \n");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                insert_at_begin(&head);
                break;
            case 2:
                insert_at_end(n, &head, &tail);
                break;
            case 3:
                insert_at_pos(&head);
                break;
            case 4:
                delete_head(&head);
                break;
            case 5:
                delete_tail(&head, &tail);
                break;
            case 6:
                delete_at_pos(&head);
                break;
            case 7:
                printList(head);
                break;
            case 8:
                search_element(head);
                break;
            default:
                break;
        }
    } while (choice != 9);

    return 0;
}

/*
Lab-4 Question 2:
In Program 1, add an option where the user can delete
every alternative element from
the singly linked list.
*/

#include <stdio.h>
```

```
#include <stdlib.h>

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

void insert_at_begin(Node*(*head))
{
    int val;
    printf("Data for the first node : ");
    scanf("%d", &val);

    Node* newNode = malloc(sizeof(Node));
    newNode->data = val;

    if (!(*head))
        (*head) = newNode;
    else
    {
        newNode->next = (*head);
        *head = newNode;
    }
}

void insert_at_pos(Node** head)
{
    if (!*head)
    {
        printf("Sorry the list is empty...Please try
insert at begining..");
```

```c
        return;
    }

    int val, pos;
    printf("Enter the position and the value: ");
    scanf("%d%d", &pos, &val);

    Node* newNode = malloc(sizeof(Node));
    newNode->data = val;
    newNode->next = NULL;

    Node* temp = *head;

    if (pos == 1)
    {
        *head = newNode;
    }

    for (int i = 0; i < pos - 2; i++)
    {
        if (!temp)
        {
            printf("Invalid Postition!..");
            free(temp);
            return;
        }
        temp = temp->next;
    }

    if (!temp)
    {
        printf("Invalid Postition!..");
        free(temp);
        return;
    }

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

void insert_at_end(int n, Node*(*head), Node** tail)
{
    int val;

    for (int i = 0; i < n; i++)
    {
        printf("Data for the %d node : ", i + 1);
        scanf("%d", &val);

        Node* newNode = malloc(sizeof(Node));
        newNode->data = val;
        newNode->next = NULL;

        if (!(*head))
            (*head) = *tail = newNode;
        else
        {
            (*tail)->next = newNode;
            *tail = newNode;
        }
    }
}

void printList(Node* head)
```

```
{
    Node* temp = head;

    while (temp)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL");
}

void delete_head(Node** head)
{
    if (!(*head))
    {
        printf("Empty List...");
        return;
    }
    Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

void delete_tail(Node** head, Node** tail)
{
    if (!(*head)) return;

    Node* temp = *head;

    if (!temp->next)
    {
        printf("%d is deleted..", temp->data);
        free(temp);
        *head = *tail = NULL;
        return;
    }

    while (temp->next->next)
    {
        temp = temp->next;
    }
    Node* deleteNode = temp->next;
    *tail = temp;
    temp->next = NULL;
    printf("%d is deleted from the end...",
deleteNode->data);
    free(deleteNode);
}

void delete_at_pos(Node** head)
{
    if (!(*head))
    {
        printf("List is empty.\n");
        return;
    }

    int pos;
    printf("Enter the position to delete..: ");
    if (scanf("%d", &pos) != 1 || pos < 1)
    {
        printf("Invalid position.\n");
        return;
    }
```

```c
    if (pos == 1)
    {
        delete_head(head);
        return;
    }

    Node* temp = *head;
    for (int i = 1; temp && i < pos - 1; i++)
    {
        temp = temp->next;
    }

    if (!temp || !(temp->next))
    {
        printf("Position out of bounds.\n");
        return;
    }

    Node* deleteNode = temp->next;
    temp->next = deleteNode->next;
    printf("%d deleted...\n", deleteNode->data);
    free(deleteNode);
}

// Lab-4 Q2: Delete alternate elements
void delete_alternate(Node** head)
{
    if (!*head) return;

    Node* p = *head;
    while (p && p->next)
    {
        Node* temp = p->next;
        p->next = temp->next;
        free(temp);
        p = p->next;
    }
}

int main()
{
    Node *head = NULL, *tail = NULL;
    int n, val, choice;

    do
    {
        printf("\n-----------Menu----------\n");
        printf("Press 1 to insert at the begining: \n");
        printf("Press 2 to insert at the end: \n");
        printf("Press 3 to insert at the position: \n");
        printf("Press 4 to delete the begining element: 
\n");
        printf("Press 5 to delete the end element: \n");
        printf("Press 6 to delete the element from any 
position: \n");
        printf("Press 7 to view the complete list: \n");
        printf("Press 8 to delete alternate elements: 
\n");  // ✅ Added for Lab-4 Q2
        printf("Press 9 to exit: \n");              
// ✅ Exit always last
        scanf("%d", &choice);

        switch (choice)
```

```c
        {
            case 1:
                insert_at_begin(&head);
                break;
            case 2:
                insert_at_end(n, &head, &tail);
                break;
            case 3:
                insert_at_pos(&head);
                break;
            case 4:
                delete_head(&head);
                break;
            case 5:
                delete_tail(&head, &tail);
                break;
            case 6:
                delete_at_pos(&head);
                break;
            case 7:
                printList(head);
                break;
            case 8:
                delete_alternate(&head);
                break;
            default:
                break;
        }
    } while (choice != 9);

    return 0;
}
```

```c
/*
Lab-4 Question 3:
In Program 2, add an option where the user can search and
delete that element from
the singly linked list.
*/

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

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

void insert_at_begin(Node*(*head))
{
    int val;
    printf("Data for the first node : ");
    scanf("%d", &val);

    Node* newNode = malloc(sizeof(Node));
    newNode->data = val;

    if (!(*head))
        (*head) = newNode;
    else
    {
        newNode->next = (*head);
        *head = newNode;
    }
}

void insert_at_pos(Node** head)
{
    if (!*head)
    {
        printf("Sorry the list is empty...Please try 
insert at begining..");
        return;
    }

    int val, pos;
    printf("Enter the position and the value: ");
    scanf("%d%d", &pos, &val);

    Node* newNode = malloc(sizeof(Node));
    newNode->data = val;
    newNode->next = NULL;

    Node* temp = *head;

    if (pos == 1)
    {
        *head = newNode;
    }

    for (int i = 0; i < pos - 2; i++)
    {
        if (!temp)
        {
            printf("Invalid Postition!..");
```

```c
            free(temp);
            return;
        }
        temp = temp->next;
    }

    if (!temp)
    {
        printf("Invalid Postition!..");
        free(temp);
        return;
    }

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

void insert_at_end(int n, Node*(*head), Node** tail)
{
    int val;

    for (int i = 0; i < n; i++)
    {
        printf("Data for the %d node : ", i + 1);
        scanf("%d", &val);

        Node* newNode = malloc(sizeof(Node));
        newNode->data = val;
        newNode->next = NULL;

        if (!(*head))
            (*head) = *tail = newNode;
        else
        {
            (*tail)->next = newNode;
            *tail = newNode;
        }
    }
}

void printList(Node* head)
{
    Node* temp = head;

    while (temp)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL");
}

void delete_head(Node** head)
{
    if (!(*head))
    {
        printf("Empty List...");
        return;
    }
    Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}
```

```c
void delete_tail(Node** head, Node** tail)
{
    if (!(*head)) return;

    Node* temp = *head;

    if (!temp->next)
    {
        printf("%d is deleted..", temp->data);
        free(temp);
        *head = *tail = NULL;
        return;
    }

    while (temp->next->next)
    {
        temp = temp->next;
    }
    Node* deleteNode = temp->next;
    *tail = temp;
    temp->next = NULL;
    printf("%d is deleted from the end...",
deleteNode->data);
    free(deleteNode);
}

void delete_at_pos(Node** head)
{
    if (!(*head))
    {
        printf("List is empty.\n");
        return;
    }

    int pos;
    printf("Enter the position to delete..: ");
    if (scanf("%d", &pos) != 1 || pos < 1)
    {
        printf("Invalid position.\n");
        return;
    }

    if (pos == 1)
    {
        delete_head(head);
        return;
    }

    Node* temp = *head;
    for (int i = 1; temp && i < pos - 1; i++)
    {
        temp = temp->next;
    }

    if (!temp || !(temp->next))
    {
        printf("Position out of bounds.\n");
        return;
    }

    Node* deleteNode = temp->next;
    temp->next = deleteNode->next;
    printf("%d deleted...\n", deleteNode->data);
    free(deleteNode);
```

```
}

// ✅ Lab-4 Q3: Search and delete an element
int search_element(Node* head)
{
    int key, pos = 1;
    printf("\nEnter the element to search: ");
    scanf("%d", &key);

    if (!head)
    {
        printf("Empty List\n");
        return -1;
    }

    Node* p = head;
    while (p)
    {
        if (p->data == key)
        {
            printf("Element found at %d position\n",
pos);
            return pos;
        }
        p = p->next;
        pos++;
    }

    printf("Element not found\n");
    return -1;
}
```

```c
void delete_element(Node** head)
{
    if (!*head) return;

    int pos = search_element(*head);
    if (pos == -1)
    {
        printf("Element not found!\n");
        return;
    }

    Node* temp = *head;
    if (pos == 1)
    {
        *head = temp->next;
        printf("\n%d deleted\n", temp->data);
        free(temp);
        return;
    }

    Node* p = *head;
    for (int i = 1; i < pos - 1 && p; i++) p = p->next;

    if (!p || !p->next) return;

    Node* todelete = p->next;
    p->next = todelete->next;
    printf("\n%d deleted\n", todelete->data);
    free(todelete);
}

int main()
```

```c
{
    Node *head = NULL, *tail = NULL;
    int n, val, choice;

    do
    {
        printf("\n-----------Menu----------\n");
        printf("Press 1 to insert at the begining: \n");
        printf("Press 2 to insert at the end: \n");
        printf("Press 3 to insert at the position: \n");
        printf("Press 4 to delete the begining element:
\n");
        printf("Press 5 to delete the end element: \n");
        printf("Press 6 to delete the element from any
position: \n");
        printf("Press 7 to view the complete list: \n");
        printf("Press 8 to search and delete an element:
\n");   // ✅ Added for Lab-4 Q3
        printf("Press 9 to exit: \n");
// ✅ Exit always last
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                insert_at_begin(&head);
                break;
            case 2:
                insert_at_end(n, &head, &tail);
                break;
            case 3:
                insert_at_pos(&head);
                break;
            case 4:
                delete_head(&head);
                break;
            case 5:
                delete_tail(&head, &tail);
                break;
            case 6:
                delete_at_pos(&head);
                break;
            case 7:
                printList(head);
                break;
            case 8:
                delete_element(&head);
                break;
            default:
                break;
        }
    } while (choice != 9);

    return 0;
}

/*
Lab-4 Question 3:
In Program 2, add an option where the user can search and
delete that element from
the singly linked list.
*/

#include <stdio.h>
```

```
#include <stdlib.h>

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

void insert_at_begin(Node** head)
{
    int val;
    printf("Data for the first node : ");
    scanf("%d", &val);

    Node* newNode = malloc(sizeof(Node));
    newNode->data = val;

    if (!(*head))
        (*head) = newNode;
    else
    {
        newNode->next = (*head);
        *head = newNode;
    }
}

void insert_at_pos(Node** head)
{
    if (!*head)
    {
        printf("Sorry the list is empty...Please try
insert at begining..");
```

```c
        return;
    }

    int val, pos;
    printf("Enter the position and the value: ");
    scanf("%d%d", &pos, &val);

    Node* newNode = malloc(sizeof(Node));
    newNode->data = val;
    newNode->next = NULL;

    Node* temp = *head;

    if (pos == 1)
    {
        *head = newNode;
    }

    for (int i = 0; i < pos - 2; i++)
    {
        if (!temp)
        {
            printf("Invalid Postition!..");
            free(temp);
            return;
        }
        temp = temp->next;
    }

    if (!temp)
    {
        printf("Invalid Postition!..");
        free(temp);
        return;
    }

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

void insert_at_end(int n, Node** head, Node** tail)
{
    int val;

    for (int i = 0; i < n; i++)
    {
        printf("Data for the %d node : ", i + 1);
        scanf("%d", &val);

        Node* newNode = malloc(sizeof(Node));
        newNode->data = val;
        newNode->next = NULL;

        if (!(*head))
            (*head) = *tail = newNode;
        else
        {
            (*tail)->next = newNode;
            *tail = newNode;
        }
    }
}

void printList(Node* head)
```

```c
{
    Node* temp = head;

    while (temp)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL");
}

void delete_head(Node** head)
{
    if (!(*head))
    {
        printf("Empty List...");
        return;
    }
    Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

void delete_tail(Node** head, Node** tail)
{
    if (!(*head)) return;

    Node* temp = *head;

    if (!temp->next)
    {
        printf("%d is deleted..", temp->data);
        free(temp);
        *head = *tail = NULL;
        return;
    }

    while (temp->next->next)
    {
        temp = temp->next;
    }
    Node* deleteNode = temp->next;
    *tail = temp;
    temp->next = NULL;
    printf("%d is deleted from the end...",
deleteNode->data);
    free(deleteNode);
}

void delete_at_pos(Node** head)
{
    if (!(*head))
    {
        printf("List is empty.\n");
        return;
    }

    int pos;
    printf("Enter the position to delete..: ");
    if (scanf("%d", &pos) != 1 || pos < 1)
    {
        printf("Invalid position.\n");
        return;
    }
    if (pos == 1)
    {
        delete_head(head);
        return;
    }

    Node* temp = *head;
    for (int i = 1; temp && i < pos - 1; i++)
    {
        temp = temp->next;
    }

    if (!temp || !(temp->next))
    {
        printf("Position out of bounds.\n");
        return;
    }

    Node* deleteNode = temp->next;
    temp->next = deleteNode->next;
    printf("%d deleted...\n", deleteNode->data);
    free(deleteNode);
}

// ✅ Lab-4 Q3: Search and delete an element
int search_element(Node* head)
{
    int key, pos = 1;
    printf("\nEnter the element to search: ");
    scanf("%d", &key);
    if (!head)
    {
        printf("Empty List\n");
        return -1;
    }

    Node* p = head;
    while (p)
    {
        if (p->data == key)
        {
            printf("Element found at %d position\n",
pos);
            return pos;
        }
        p = p->next;
        pos++;
    }

    printf("Element not found\n");
    return -1;
}

void delete_element(Node** head)
{
    if (!*head) return;

    int pos = search_element(*head);
    if (pos == -1)
    {
        printf("Element not found!\n");
        return;
```

```c
    }

    Node* temp = *head;
    if (pos == 1)
    {
        *head = temp->next;
        printf("\n%d deleted\n", temp->data);
        free(temp);
        return;
    }

    Node* p = *head;
    for (int i = 1; i < pos - 1 && p; i++) p = p->next;

    if (!p || !p->next) return;

    Node* todelete = p->next;
    p->next = todelete->next;
    printf("\n%d deleted\n", todelete->data);
    free(todelete);
}

// 🔁 Your reversal functions (unchanged)
void reverse(Node** head)
{
    Node *prev = NULL, *curr = *head, *next = NULL;
    while (curr)
    {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    *head = prev;
}

void reverse_pos(Node** head)
{
    int pos;
    printf("Enter the position to reverse from: ");
    scanf("%d", &pos);

    if (pos <= 1)
    {
        reverse(head);
        return;
    }

    Node* p = *head;
    while (--pos && p)
    {
        p = p->next;
    }

    if (!p || !p->next)
    {
        printf("Position out of range!\n");
        return;
    }

    Node *prev = NULL, *next = NULL, *curr = p->next;

    while (curr)
    {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    p->next = prev;
    printf("\nReversed!");
}

int main()
{
    Node *head = NULL, *tail = NULL;
    int n, val, choice;

    do
    {
        printf("\n-----------Menu----------\n");
        printf("Press 1 to insert at the begining: \n");
        printf("Press 2 to insert at the end: \n");
        printf("Press 3 to insert at the position: \n");
        printf("Press 4 to delete the begining element:
\n");
        printf("Press 5 to delete the end element: \n");
        printf("Press 6 to delete the element from any
position: \n");
        printf("Press 7 to view the complete list: \n");
        printf("Press 8 to search and delete an element:
\n");
        printf("Press 9 to reverse entire list: \n");
// new
        printf("Press 10 to reverse from position: \n");
// new
```

```
        printf("Press 11 to exit: \n");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                insert_at_begin(&head);
                break;
            case 2:
                insert_at_end(n, &head, &tail);
                break;
            case 3:
                insert_at_pos(&head);
                break;
            case 4:
                delete_head(&head);
                break;
            case 5:
                delete_tail(&head, &tail);
                break;
            case 6:
                delete_at_pos(&head);
                break;
            case 7:
                printList(head);
                break;
            case 8:
                delete_element(&head);
                break;
            case 9:
                reverse(&head);
                break;
```

```c
            case 10:
                reverse_pos(&head);
                break;
            default:
                break;
        }
    } while (choice != 11);

    return 0;
}
```