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

// 1. A magic square is a square grid of numbers where the sum of the numbers in each row,
// column, and diagonal is the same. Write a C program to implement the magic square for
// the n x n matrix, where n is an odd number and the array contains numbers 1 to n^2.
#include <stdio.h>
#include <stdlib.h>

void generateMagicSquare(int n)
{
    if (n % 2 == 0)
    {
        printf("Magic square is only exist for odd inteager n.\n");
        return;
    }

    int magic[n][n];

    // Initialize all cells to 0
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) magic[i][j] = 0;

    int num = 1;
    int i = 0, j = n / 2;

    while (num <= n * n)
    {
```

```
        magic[i][j] = num;

        int next_i = (i - 1 + n) % n;
        int next_j = (j + 1) % n;

        if (magic[next_i][next_j] != 0)
        {
            i = (i + 1) % n;
        }
        else
        {
            i = next_i;
            j = next_j;
        }

        num++;
    }

    // Print the magic square
    printf("\nMagic Square of size %d:\n", n);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++) printf("%4d ",
magic[i][j]);
        printf("\n");
    }

    // Optional: Print the magic constant
    int magicConstant = n * (n * n + 1) / 2;
    printf("\nMagic Constant Number = %d\n", magicConstant);
}
```

```c
int main()
{
    int n;
    printf("Enter an odd number for magic square matrix size:
");
    scanf("%d", &n);

    generateMagicSquare(n);
    return 0;
}
// 2. Write a C program to print a string in reverse order
using recursion. You need not store
// the characters of the string in an array or any other data
structure.
#include <stdio.h>

void reverse()
{
    char c = getchar();
    if (c == '\n') return;
    reverse();
    putchar(c);
}
int main()
{
    printf("Enter a string: ");
    reverse();
    return 0;
}
// 3. Write a C program to calculate the length of the input
string using recursion.
#include <stdio.h>
```

```c
int count()
{
    char c = getchar();
    if (c == '\n') return 0;
    if (c == ' ') return count();
    return 1 + count();
}
int main()
{
    int n;
    printf("Enter the string: ");
    n = count();
    printf("\n%d total letters!", n);
    return 0;
}
// 4. Write a C program to create a singly linked list having
number of elements as per user
// choice.
#include <stdio.h>
#include <stdlib.h>

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

int main()
{
    int n, val;
    printf("Number of node? ");
```

```
    scanf("%d", &n);

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

    for (int i = 0; i < n; i++)
    {
        printf("Value of %d node: ", i + 1);
        scanf("%d", &val);
        struct Node* newNode = malloc(sizeof(struct Node));

        newNode->data = val;
        newNode->next = NULL;

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

    struct Node* temp = head;

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

```
    printf("Null");

    temp = head;

    while (head)
    {
        temp = head;
        head = head->next;
        free(temp);
    }
    return 0;
}
// 5. Using program 4, create a menu based program to insert
an element at the beginning,
// end or at the location given by the user in the linked
list, and to delete the element from
// the beginning, end or from any user given location.
#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");
        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);
}

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 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;

            default:
                break;
        }
    } while (choice != 8);

    return 0;
}
```