```c
// Name: Ayush Gupta
// Roll No: MNS2025027

1. Write a menu-based program for creation, insertion at
beginning, at end or at given
location, deletion from beginning, from end or from given
location, and traversing of
singly circular linked list.

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

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

} Node;

void insert_at_end(Node **head, Node **tail)
{
    int val;
    printf("Enter the data: ");
    scanf("%d", &val);

    Node *newNode = malloc(sizeof(Node));

    newNode->data = val;

    if (!*head)
    {
        *head = newNode;
```

```
        *tail = newNode;
        newNode->next = *head;
        return;
    }

    (*tail)->next = newNode;
    newNode->next = (*head);
    *tail = newNode;

    return;
}

void insert_at_begin(Node **head, Node **tail)
{
    int val;
    printf("Enter value: ");
    scanf("%d", &val);

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

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

```c
void insert_pos(Node **head, Node **tail)
{
    int val, pos;
    printf("Enter the value and position: ");
    scanf("%d%d", &val, &pos);

    if (pos <= 1) return;

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

    Node *curr = *head;

    int count = 0;

    while (count < pos - 1 && curr->next != *head)
    {
        curr = curr->next;
        count++;
    }

    newNode->next = curr->next;
    curr->next = newNode;

    if (curr == *tail)
    {
        *tail = newNode;
    }
}
```

```c
void delete_head(Node **head, Node **tail)
{
    if (!*head) return;
    Node *todelete = (*head);
    *head = (*head)->next;
    (*tail)->next = (*head);
    free(todelete);
}

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

    Node *p = *head;
    while (p->next != *tail)
    {
        p = p->next;
    }

    Node *temp = p->next;
    p->next = (*tail)->next;
    *tail = p;
    free(temp);
}

void delete_pos(Node **head, Node **tail)
{
    int pos;
    printf("Enter the position to delete: ");
    scanf("%d", &pos);

    if (!*head)
```

```
    {
        printf("List is empty.\n");
        return;
    }

    // Count length
    int length = 1;
    Node *curr = *head;
    while (curr->next != *head)
    {
        curr = curr->next;
        length++;
    }

    // Validate position
    if (pos > length || pos < 1)
    {
        printf("Invalid position. List has only %d
nodes.\n", length);
        return;
    }

    // Special case: only one node
    if (*head == *tail && pos == 1)
    {
        free(*head);
        *head = NULL;
        *tail = NULL;
        return;
    }

    // Delete head
```

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

    // Traverse to node before target
    Node *temp = *head;
    for (int i = 1; i < pos - 1; i++)
    {
        temp = temp->next;
    }

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

    if (todelete == *tail)
    {
        *tail = temp;
    }

    free(todelete);
}
void show(Node **head)
{
    if (!*head)
    {
        printf("NULL");
        return;
    }

    Node *p = *head;
```

```
    do
    {
        printf("%d -> ", p->data);
        p = p->next;
    } while (p != *head);

    printf("HEAD");
}

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

    do
    {
        printf("\n------------MENU------------");
        printf("\n1. Insert at begin.");
        printf("\n2. Insert at the end.");
        printf("\n3. Insert at position.");
        printf("\n4. Delete Head. ");
        printf("\n5. Delete Tail. ");
        printf("\n6. Delete Position. ");
        printf("\n7. Show List.");
        printf("\n8. Exit.\n");

        scanf("%d", &n);

        switch (n)
        {
            case 1:
```

```
                insert_at_begin(&head, &tail);
                break;
            case 2:
                insert_at_end(&head, &tail);
                break;
            case 3:
                insert_pos(&head, &tail);
                break;
            case 4:

                delete_head(&head, &tail);
                break;
            case 5:
                delete_tail(&head, &tail);
                break;
            case 6:
                delete_pos(&head, &tail);
                break;
            case 7:
                show(&head);
                break;
        }

    } while (n != 8);

    return 0;
}
```

2. Write a menu-based program for creation, insertion at beginning, at end or at given

```
location, deletion from beginning, from end or from given
location, and traversing of
doubly linked list.

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

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

void insert_at_end(Node **head, Node **tail)
{
    int val;
    printf("Enter the data: ");
    scanf("%d", &val);

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

    if (!*head)
    {
        *head = *tail = newNode;
        return;
    }

    (*tail)->next = newNode;
```

```
    newNode->prev = *tail;
    *tail = newNode;
}

void insert_at_begin(Node **head, Node **tail)
{
    int val;
    printf("Enter value: ");
    scanf("%d", &val);

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

    if (!*head)
    {
        *head = *tail = newNode;
        return;
    }

    (*head)->prev = newNode;
    *head = newNode;
}

void insert_pos(Node **head, Node **tail)
{
    int val, pos;
    printf("Enter the value and position: ");
    scanf("%d%d", &val, &pos);

    if (pos <= 1)
```

```
    {
        insert_at_begin(head, tail);
        return;
    }

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

    Node *curr = *head;
    int count = 1;

    while (curr && count < pos - 1)
    {
        curr = curr->next;
        count++;
    }

    if (!curr || !curr->next)
    {
        insert_at_end(head, tail);
        return;
    }

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

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

```
    Node *todelete = *head;
    *head = (*head)->next;

    if (*head)
        (*head)->prev = NULL;
    else
        *tail = NULL;

    free(todelete);
}

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

    Node *todelete = *tail;
    *tail = (*tail)->prev;

    if (*tail)
        (*tail)->next = NULL;
    else
        *head = NULL;

    free(todelete);
}

void delete_pos(Node **head, Node **tail)
{
    int pos;
    printf("Enter the position to delete: ");
    scanf("%d", &pos);
```

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

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

    Node *curr = *head;
    int count = 1;

    while (curr && count < pos)
    {
        curr = curr->next;
        count++;
    }

    if (!curr)
    {
        printf("Invalid position.\n");
        return;
    }

    if (curr == *tail)
    {
        delete_tail(head, tail);
        return;
```

```
    }

    curr->prev->next = curr->next;
    curr->next->prev = curr->prev;
    free(curr);
}

void show(Node **head)
{
    if (!*head)
    {
        printf("NULL\n");
        return;
    }

    Node *p = *head;
    while (p)
    {
        printf("%d <-> ", p->data);
        p = p->next;
    }
    printf("NULL\n");
}

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

    do
    {
        printf("\n------------MENU------------");
```

```
        printf("\n1. Insert at begin.");
        printf("\n2. Insert at the end.");
        printf("\n3. Insert at position.");
        printf("\n4. Delete Head.");
        printf("\n5. Delete Tail.");
        printf("\n6. Delete Position.");
        printf("\n7. Show List.");
        printf("\n8. Exit.\n");

        scanf("%d", &n);

        switch (n)
        {
            case 1:
                insert_at_begin(&head, &tail);
                break;
            case 2:
                insert_at_end(&head, &tail);
                break;
            case 3:
                insert_pos(&head, &tail);
                break;
            case 4:
                delete_head(&head, &tail);
                break;
            case 5:
                delete_tail(&head, &tail);
                break;
            case 6:
                delete_pos(&head, &tail);
                break;
            case 7:
```

```
                show(&head);
                break;
        }

    } while (n != 8);

    return 0;
}
```

3. Write a menu-based program for creation, insertion at beginning, at end or at given
location, deletion from beginning, from end or from given location, and traversing of
doubly circular linked list.

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

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

void insert_at_end(Node **head, Node **tail)
{
    int val;
    printf("Enter the data: ");
    scanf("%d", &val);
```

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

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

    newNode->next = *head;
    newNode->prev = *tail;
    (*tail)->next = newNode;
    (*head)->prev = newNode;
    *tail = newNode;
}

void insert_at_begin(Node **head, Node **tail)
{
    int val;
    printf("Enter value: ");
    scanf("%d", &val);

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

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

```
        newNode->prev = newNode;
        return;
    }

    newNode->next = *head;
    newNode->prev = *tail;
    (*head)->prev = newNode;
    (*tail)->next = newNode;
    *head = newNode;
}

void insert_pos(Node **head, Node **tail)
{
    int val, pos;
    printf("Enter the value and position: ");
    scanf("%d%d", &val, &pos);

    if (pos <= 1)
    {
        insert_at_begin(head, tail);
        return;
    }

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

    Node *curr = *head;
    int count = 1;

    while (count < pos - 1 && curr->next != *head)
    {
        curr = curr->next;
```

```
        count++;
    }

    if (curr == *tail)
    {
        insert_at_end(head, tail);
        free(newNode);
        return;
    }

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

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

    if (*head == *tail)
    {
        free(*head);
        *head = *tail = NULL;
        return;
    }

    Node *todelete = *head;
    *head = (*head)->next;
    (*head)->prev = *tail;
    (*tail)->next = *head;
    free(todelete);
```

```
}

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

    if (*head == *tail)
    {
        free(*tail);
        *head = *tail = NULL;
        return;
    }

    Node *todelete = *tail;
    *tail = (*tail)->prev;
    (*tail)->next = *head;
    (*head)->prev = *tail;
    free(todelete);
}

void delete_pos(Node **head, Node **tail)
{
    int pos;
    printf("Enter the position to delete: ");
    scanf("%d", &pos);

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

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

    Node *curr = *head;
    int count = 1;

    while (count < pos && curr->next != *head)
    {
        curr = curr->next;
        count++;
    }

    if (curr == *head)
    {
        printf("Invalid position.\n");
        return;
    }

    if (curr == *tail)
    {
        delete_tail(head, tail);
        return;
    }

    curr->prev->next = curr->next;
    curr->next->prev = curr->prev;
    free(curr);
}
```

```c
void show(Node **head)
{
    if (!*head)
    {
        printf("NULL\n");
        return;
    }

    Node *p = *head;
    do
    {
        printf("%d <-> ", p->data);
        p = p->next;
    } while (p != *head);
    printf("(HEAD)\n");
}

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

    do
    {
        printf("\n------------MENU------------");
        printf("\n1. Insert at begin.");
        printf("\n2. Insert at the end.");
        printf("\n3. Insert at position.");
        printf("\n4. Delete Head.");
        printf("\n5. Delete Tail.");
        printf("\n6. Delete Position.");
        printf("\n7. Show List.");
```

```c
        printf("\n8. Exit.\n");

        scanf("%d", &n);

        switch (n)
        {
            case 1:
                insert_at_begin(&head, &tail);
                break;
            case 2:
                insert_at_end(&head, &tail);
                break;
            case 3:
                insert_pos(&head, &tail);
                break;
            case 4:
                delete_head(&head, &tail);
                break;
            case 5:
                delete_tail(&head, &tail);
                break;
            case 6:
                delete_pos(&head, &tail);
                break;
            case 7:
                show(&head);
                break;
        }

    } while (n != 8);

    return 0;
```

```
}
```