```c
// ------------------------------  SINGLY CIRCULAR LINKED LIST
--------------------------------


/*
Title- Circular Singly Linked List Operations
Author- Bhakare Mahesh Santosh
ID- 492
Batch- TechnOrbit(PPA-8)
*/
#include<stdio.h>
#include<stdlib.h>

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


// -------------------- FUNCTION TO CREATE NODE -----------------

struct node* CreateNode()
{
    struct node* newnode = NULL;
    newnode = (struct node*)malloc(sizeof(struct node));
    if(newnode != NULL)
    {
        printf("Enter the data for newnode: ");
        scanf("%d",&(newnode->data));
        newnode->next = NULL;
    }
    else
    {
        printf("Memory not allocateed .....\n");
    }
    return newnode;
}
// ------------ FUNCTION TO COUNT NODES --------------------------

int CountNode(struct node* head)
{
    struct node* tempnode = head;
    int count = 0;
    if(head != NULL)
    {
        do
        {
            count++;
            head = head->next;
        }while(head != tempnode);
    }
    return count;

}


// ---------------------- FUNCTION TO CREATE LINKED LIST(MAKES LINKING OF NODES)
--------------------

void CreateLinkedList(struct node** head)
{
    struct node* newnode;
    struct node* tempnode = *head;
    newnode = CreateNode();
    if(*head == NULL)
```

```
    {
        *head = newnode;
        newnode->next =*head;
    }
    else
    {
        while(tempnode->next != *head)
        {
            tempnode = tempnode->next;
        }
        tempnode->next = newnode;
        newnode->next = *head;

    }
}


// ------------------------- FUNCTION TO DISPLAY LINKED LIST -----------

void DisplayLinkedList(struct node* tempnode)
{
    struct node* head = tempnode;
    printf("Our Linked List is: ");
    if(tempnode != NULL)
    {
        do
        {
            printf(" -> %d",tempnode->data);
            tempnode = tempnode->next;
        }while(tempnode != head);
    }
    printf("\n");

}


// ---------------------- FUNCTION TO INSERT NODE AT FIRST ----------------------

void InsertAtFirst(struct node** head)
{
    struct node* tempnode = *head;
    struct node* newnode = NULL;
    newnode = CreateNode();
    if(*head == NULL)
    {
        *head = newnode;
        newnode -> next = *head;
    }
    else
    {
        while(tempnode->next != *head)
        {
            tempnode = tempnode->next;
        }
        tempnode->next = newnode;
        newnode -> next = *head;
        *head = newnode;
    }

}


// -------------------- FUNCTION TO INSERT NODE AT LAST ------------------
void InsertAtLast(struct node** head)
{
    CreateLinkedList(head); //if we call CreateLinkedList then we create node and att it
```

```
to the last of previiously created linked list...
}

// ------------------- FUNCTION TO INSERT AT POSITION -------------------------------

void InsertAtPosition(struct node** head)
{
    struct node* newnode = NULL;
    struct node* tempnode = *head;
    int pos,n,i;
    n = CountNode(*head);
    printf("Enter the position where you want to add new node: ");
    scanf("%d",&pos);
    if(pos == 1)
    {
        InsertAtFirst(head);
    }
    else
    {
        if(pos == n+1)
        {
            InsertAtLast(head);
        }
        else
        {
            if(pos < 1 || pos > n+1)
            {
                printf("Enter valid position .....\n");
                InsertAtPosition(head);
            }
            else
            {
                if(pos > 1 && pos < n+1)
                {
                    newnode = CreateNode();
                    for(i=1;i<pos-1;i++)
                    {
                        tempnode = tempnode->next;
                    }
                    newnode->next = tempnode->next;
                    tempnode->next = newnode;
                }
            }
        }
    }
}

// ------------------- FUNCTION TO DELETE AT FIRST -------------------------------

 void DeleteAtFirst(struct node** head)
{
    struct node* tempnode = *head;
    if(*head == NULL)
    {
        printf("Linked List not created , Please create Linked list....\n");
    }
    else if((*head)->next == *head)
    {
        free(*head);
        *head = NULL;
    }
    else
    {
        while(tempnode->next != *head)
        {
            tempnode = tempnode->next;
```

```
        }
        tempnode->next = (*head)->next;
        tempnode = *head;
        *head = (*head) -> next;
        free(tempnode);
        tempnode = NULL;
    }
}

// ------------------- FUNCTION TO DELETE AT LAST -------------------------------

void DeleteAtLast1(struct node** head)
{
    struct node* tempnode = *head;
    if(*head == NULL)
    {
        printf("Linked List not created, Please create Linked List....\n");
    }
    else
    {
        if((*head) -> next == *head)
        {
            free(*head);
            *head = NULL;
        }
        else
        {
            while((tempnode -> next) -> next != *head)
            {
                tempnode = tempnode -> next;
            }
            free(tempnode -> next);
            tempnode -> next = *head;
        }

    }
}

// ------------------- FUNCTION TO DELETE AT LAST -------------------------------

void DeleteAtLast2(struct node** head)
{
    struct node* tempnode1 = *head;
    struct node* tempnode2 = *head;
    if(*head == NULL)
    {
        printf("No liked List created, Please create Linked List....\n");
    }
    else
    {
        if((*head)->next == *head)
        {
            free(*head);
            *head = NULL;
        }
        else
        {
            if((*head)->next->next == *head)
            {
                free((*head)->next);
                (*head)->next = *head;
            }
            else
            {
                while(tempnode2->next != *head)
                {
```

```
                    tempnode1 = tempnode1->next;
                    tempnode2 = tempnode1->next;
                }
                free(tempnode2);
                tempnode1->next = *head;
            }
        }
    }
}

// ------------------- FUNCTION TO DELETE AT POSITION -------------------------------

void DeleteAtPosition(struct node** head)
{
    struct node* tempnode1 = *head;
    struct node* tempnode2 = NULL;
    int n,pos,i;
    n = CountNode(*head);
    printf("Enter the position from where you want to delete node: ");
    scanf("%d",&pos);
    if(pos == 1)
    {
        DeleteAtFirst(head);
    }
    else
    {
        if(pos == n)
        {
            DeleteAtLast1(head);
        }
        else
        {
            if(pos < 1 || pos > n)
            {
                printf("Please Enter the Valid Position....\n");
                DeleteAtPosition(head);
            }
            else
            {
                if(pos > 1 && pos < n)
                {
                    for(i=1;i<pos-1;i++)
                    {
                        tempnode1 = tempnode1 -> next;
                    }
                    tempnode2 = tempnode1 -> next;
                    tempnode1 -> next = (tempnode1 -> next) -> next;
                    free(tempnode2);
                    tempnode2 = NULL;
                    tempnode1 = NULL;
                }
            }
        }
    }
}

// ------------------------ MAIN FUNCTION ---------------------------------

void main()
{
    int choice;
    struct node* first=NULL;
    do
    {
        printf(" -------------------------------- *****************
--------------------------\n\n");
```

```
        printf("1) Create Link List\n2) Display Link List\n3) Insert At First\n4) Insert
At Last\n5) Insert At Position\n6) Delete At First\n7) Delete At Last1\n8) Delete At
Last2\n9) Delete At Position\n0) Exit\nEnter your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: CreateLinkedList(&first); // &first = to make change at the address
of first
                    break;
            case 2: DisplayLinkedList(first); // first = to make copy of first
                    break;
            case 3: InsertAtFirst(&first);
                    break;
            case 4: InsertAtLast(&first);
                    break;
            case 5: InsertAtPosition(&first);
                    break;
            case 6: DeleteAtFirst(&first);
                    break;
            case 7: DeleteAtLast1(&first);
                    break;
            case 8: DeleteAtLast2(&first);
                    break;
            case 9: DeleteAtPosition(&first);
                    break;
        }
    }while(choice != 0);
}



// -------------------------------- DOUBLY CIRCULAR LINKED LIST
----------------------------------------



/*
Title- Circular Doubly Linked List Operations
Author- Bhakare Mahesh Santosh
ID- 492
Batch- TechnOrbit(PPA-8)
*/

#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node* prev;
    struct node* next;
};
int CountNode(struct node*, struct node*);
struct node* CreateNode();
void CreateLinkedList(struct node**,struct node**);
void DisplayLinkedList(struct node*,struct node*);
void ReverseDisplay(struct node*,struct node*);
void InsertAtFirst(struct node**,struct node**);
void InsertAtLast(struct node**,struct node**);
void InsertAtPosition(struct node**,struct node**);
void DeleteAtFirst(struct node**,struct node**);
void DeleteAtLast(struct node**,struct node**);
void DeleteAtPosition(struct node**,struct node**);

void main()
{
```

```c
    struct node *first = NULL, *last = NULL;
    int choice;
    do
    {
        printf("\n---------------------------------- **********************
----------------------------------------\n");
        printf("\n1) Create Linked List\n2) Display Linked List\n3) Reversed Linked
List\n4) Insert at First\n5) Insert At Last\n6) Insert At Position\n7) Delete At
First\n8) Delete At Last\n9) Delete At Position\n0) Exit\nEnter Your Choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: CreateLinkedList(&first,&last);
                    break;
            case 2: DisplayLinkedList(first, last);
                    break;
            case 3: ReverseDisplay(first, last);
                    break;
            case 4: InsertAtFirst(&first,&last);
                    break;
            case 5: InsertAtLast(&first,&last);
                    break;
            case 6: InsertAtPosition(&first,&last);
                    break;
            case 7: DeleteAtFirst(&first,&last);
                    break;
            case 8: DeleteAtLast(&first,&last);
                    break;
            case 9: DeleteAtPosition(&first,&last);
                    break;
        }
    }while(choice!=0);
}
// -------------------- FUNCTION TO COUNT NODE -----------------
int CountNode(struct node* first, struct node* last)
{
    int count = 0;
    if(first != NULL)
    {
        do
        {
            count++;
            first = first->next;
        }while(first != (last->next));
    }
    return count;
}

// -------------------- FUNCTION TO CREATE NODE -----------------

struct node* CreateNode()
{
    struct node* newnode =  NULL;
    newnode = (struct node*)malloc(sizeof(struct node));
    if(newnode == NULL)
    {
        printf("Memory not allocated\n");
    }
    else
    {
        printf("Enter the data: ");
        scanf("%d",&(newnode->data));
        newnode->prev = NULL;
        newnode->next = NULL;
    }
    return newnode;
```

```
}


// -------------------- FUNCTION TO CREATE LINKED LIST (JOINING OF NODES)
-----------------

void CreateLinkedList(struct node** first, struct node** last)
{
    struct node* newnode = NULL;
    newnode = CreateNode();
    if(*first == NULL)
    {
        *first = *last = newnode;
        (*first)->prev = (*first)->next = newnode;
    }
    else
    {
        newnode->prev = *last;
        (*last)->next = newnode;
        *last = newnode;
        (*last)->next = *first;
        (*first)->prev = *last;
    }
}

// -------------------- FUNCTION TO DISPLAY IN FORWARD -----------------

void DisplayLinkedList(struct node* first, struct node* last)
{
    printf("Linked List In Forward Order: ");
    if(first != NULL)
    {
        do
        {
            printf(" -> %d",first->data);
            first = first->next;
        }while(first != (last->next));
    }

}

// -------------------- FUNCTION TO DISPLAY IN REVERSE -----------------

void ReverseDisplay(struct node* first, struct node* last)
{
    printf("Linked List In Backward Order: ");
    if(last != NULL)
    {
        do
        {
            printf(" -> %d",last->data);
            last = last->prev;
        }while(last != (first->prev));
    }

}

// -------------------- FUNCTION TO INSERT AT FIRST -----------------

void InsertAtFirst(struct node** first, struct node** last)
{
    struct node* newnode = NULL;
    newnode =CreateNode();
    if(*first == NULL)
    {
        *first = *last = newnode;
```

```
        (*first)->prev = (*first)->next = newnode;
    }
    else
    {
        newnode->next = *first;
        (*first)->prev = newnode;
        *first = newnode;
        (*last)->next = *first;
        (*first)->prev = *last;
    }

}

// -------------------- FUNCTION TO INSERT AT LAST -----------------

void InsertAtLast(struct node** first, struct node** last)
{
    CreateLinkedList(first, last);
}

// -------------------- FUNCTION TO INSERT AT POSITION -----------------

void InsertAtPosition(struct node** first, struct node** last)
{
    struct node* tempnode = *first;
    struct node* newnode = NULL;
    int n,pos,i;
    n= CountNode(*first, *last);
    printf("enter the position where you want to insert a new node: ");
    scanf("%d",&pos);
    if(pos == 1)
    {
        InsertAtFirst(first, last);
    }
    else if(pos == n+1)
    {
        InsertAtLast(first, last);
    }
    else if(pos < 1 || pos > n+1)
    {
        printf("Invalid Position....Please Enter Position Again...\n");
        InsertAtPosition(first, last);
    }
    else if(pos > 1 && pos < n+1)
    {
        newnode = CreateNode();
        for(i = 1; i<pos;i++)
        {
            tempnode= tempnode->next;
        }
        newnode->next = tempnode;
        newnode->prev = tempnode->prev;
        tempnode->prev->next = newnode;
        tempnode->prev = newnode;
    }
}

// -------------------- FUNCTION TO DELETE AT FIRST -----------------

void DeleteAtFirst(struct node** first, struct node** last)
{
    if(*first == NULL)
    {
        printf("Linked List not Available....\n");
    }
    else if((*first)->next == *first)
```

```
    {
        free(*first);
        *first = *last = NULL;
    }
    else
    {
        *first= (*first)->next;
        free((*first)->prev);
        (*first)->prev = *last;
        (*last)->next = *first;
    }
}
// -------------------- FUNCTION TO DELETE AT LAST -----------------

void DeleteAtLast(struct node** first , struct node** last)
{
    if(*first == NULL)
    {
        printf("Linked List not Available...\n");
    }
    else if((*first)->next == *first)
    {
        free(*first);
        *first = *last = NULL;
    }
    else
    {
        *last = (*last)->prev;
        free((*last)->next);
        (*last)->next = *first;
    }
}

// -------------------- FUNCTION TO DELETE AT POSITION -----------------

void DeleteAtPosition(struct node** first, struct node** last)
{
    struct node* tempnode = *first;
    int n,pos,i;
    printf("Enter the position from where you want to delete element: ");
    scanf("%d",&pos);
    n = CountNode(*first, *last);
    if(pos == 1)
    {
        DeleteAtFirst(first, last);
    }
    else if(pos == n)
    {
        DeleteAtLast(first, last);
    }
    else if(pos > 1 && pos < n)
    {
        for( i = 1; i<pos;i++)
        {
            tempnode = tempnode->next;
        }
        tempnode->next->prev = tempnode->prev;
        tempnode->prev->next = tempnode->next;
        free(tempnode);
        tempnode = NULL;
    }
    else if(pos < 1 || pos > n)
    {
        printf("Please enter the valid position...\n");
        DeleteAtPosition(first, last);
    }
```

```
}
```