```c
// Write a menu driven program in C to implement a Double Linked List using Dynamic Memory Allocation
// and apply various operations on it through separate functions as below:
// a)Create a Double linked list
// b)Display the list of elements (Both forward and backward)
// c)Search an element in the list
// d)Insert a node at the beginning of the list
// e)Insert a node at the end of the list
// f)Insert a node at a given position in the list
// g)Insert a node after a given node
// h)Delete the first node
// i)Delete the last node
// j)Delete a node at a given position
// k)Delete a node after a given node
// l)Sort the elements of the list in ascending order of their values

// C program for the all operations in the Doubly Linked List

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

// Linked List Node
struct node {
    int val;
    struct node *prev, *next;
};
struct node* start = NULL;

void create();
void traverse();
int search();
void insertAtFront();
void insertAtBack();
void insertAtPosition();
void insertAftNode();
void deleteFirst();
void deleteEnd();
void deletePosition();
void deleteAftNode();
void sort();
```

```c
int main()
{
    system("clear");
    int choice, k;
    while (1) {

        printf("\n\t1 To create a circular linked list\n");
        printf("\t2 Display the list of elements in forward and
backword manner\n");
        printf("\t3 Search an element in the list\n");
        printf("\t4 Insert a node at the beginning of the
list\n");
        printf("\t5 Insert a node at the end of the list\n");
        printf("\t6 Insert a node at a given position in the
list\n");
        printf("\t7 Insert a node after a given node\n");
        printf("\t8 Delete the first node\n");
        printf("\t9 Delete the last node\n");
        printf("\t10 Delete a node at a given position\n");
        printf("\t11 Delete a node after a given node\n");
        printf("\t12 Sort the elements of the list in ascending
order of their values\n");
        printf("\t13 To exit\n");
        printf("\nEnter Choice :\n");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            create();
            break;
        case 2:
            traverse();
            break;
        case 3:
            k=search();
            printf("The element found at %d position in the
list\n",k);
            break;
        case 4:
            insertAtFront();
```

```
            break;
        case 5:
            insertAtBack();
            break;
        case 6:
            insertAtPosition();
            break;
        case 7:
            insertAftNode();
            break;
        case 8:
            deleteFirst();
            break;
        case 9:
            deleteEnd();
            break;
        case 10:
            deletePosition();
            break;
        case 11:
            deleteAftNode();
        case 12:
            sort(start);
            break;
        case 13:
            exit(1);
            break;
        default:
            printf("Incorrect Choice. Try Again \n");
            continue;
        }
    }
    return 0;
}
void create()
{
    char ch;
    struct node *temp, *new;
    temp = (struct node*)malloc(sizeof(struct node));
    printf("\nEnter number to be inserted: ");
    scanf("%d", &temp->val);
    temp->prev = NULL;
```

```
    temp->next = NULL;
    start = temp;
    printf("Enter any character to continue and q for quit\n");
    scanf(" %c",&ch);
    while(ch!='q')
    {
        new=(struct node*)malloc(sizeof(struct node));
        temp->next=new;
        new->prev=temp;
        printf("\nEnter the value of node\n");
        scanf("%d",&new->val);
        new->next = NULL;
        temp=temp->next;
        printf("\nEnter q to quit or any other charachter to
continue\n");
        scanf(" %c",&ch);
    }
}
// Function to traverse the linked list
void traverse()
{
    // List is empty
    if (start == NULL) {
        printf("\nList is empty\n");
        return;
    }
    // Else print the Data
    struct node* temp;
    temp = start;
    printf("In forward manner: ");
    while(start!=NULL)
    {
        temp=start;
        printf("%d ",start->val);
        start=start->next;
    }
    printf("\nElements in Backward Direction : ");
    start=temp;
    while(start!=NULL)
    {
        printf("%d ",start->val);
        start=start->prev;
```

```
    }
    printf("%d \n",start->next);
}
int search()
{
    printf("Enter the key to search\n");
    int key, flag = 0;
    scanf("%d",&key);
    struct node *temp;
    temp = start;
    int i=1;
    while(temp != NULL)
    {
        if(temp->val==key)
        {
            return i;
        }
        temp = temp->next;
        i++;
    }
}

// Function to insert at the front
// of the linked list
void insertAtFront()
{
    int data;
    struct node* temp;
    temp = (struct node*)malloc(sizeof(struct node));
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    temp->val = data;
    temp->prev = NULL;

    // Pointer of temp will be
    // assigned to start
    temp->next = start;
    start = temp;
}

// Function to insert at the end of
// the linked list
```

```c
void insertAtBack()
{
    int data;
    struct node *temp, *trav;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    temp->val = data;
    temp->next = NULL;
    trav = start;

    // If start is NULL
    if (start == NULL) {

        start = temp;
    }

    // Changes Links
    else {
        while (trav->next != NULL)
            trav = trav->next;
        temp->prev = trav;
        trav->next = temp;
    }
}

// Function to insert at any specified
// position in the linked list
void insertAtPosition()
{
    int data, pos, i = 1;
    struct node *temp, *newnode;
    newnode = malloc(sizeof(struct node));
    newnode->next = NULL;
    newnode->prev = NULL;

    // Enter the position and data
    printf("\nEnter position : ");
    scanf("%d", &pos);
```

```
    // If start==NULL,
    if (start == NULL) {
        start = newnode;
        newnode->prev = NULL;
        newnode->next = NULL;
    }

    // If position==1,
    else if (pos == 1) {
    insertAtFront();
    }

    // Change links
    else {
    printf("\nEnter number to be inserted: ");
    scanf("%d", &data);
    newnode->val = data;
    temp = start;
        while (i < pos - 1) {
            temp = temp->next;
            i++;
        }
        newnode->next = temp->next;
        newnode->prev = temp;
        temp->next = newnode;
        temp->next->prev = newnode;
    }
}
void insertAftNode()
{
    int data, pos, i = 1;
    int n, key;
    struct node *temp, *newnode;
    temp = start;
    printf("Enter the value of node after which you want to
insert a new node\n");
    scanf("%d", &key);
    for (i = 1; temp != NULL; i++)
    {
        if (temp->val == key)
            n = i;
```

```
        temp = temp->next;
    }
    if (n == 0)
        printf("\nNode is Absent !\n");
    else
    {
        newnode = malloc(sizeof(struct node));
        newnode->next = NULL;
        newnode->prev = NULL;

        // Enter the position and data
        printf("\nEnter position of the node after whicch you
want to insert a new node: ");
        scanf("%d", &pos);

        // If start==NULL,
        if (start == NULL)
        {
            start = newnode;
            newnode->prev = NULL;
            newnode->next = NULL;
        }

        // If position==1,
        else if (pos == 1)
        {
            insertAtFront();
        }

        // Change links
        else
        {
            printf("\nEnter number to be inserted: ");
            scanf("%d", &data);
            newnode->val = data;
            temp = start;
            while (i < pos - 1)
            {
                temp = temp->next;
                i++;
            }
            newnode->next = temp->next;
```

```
            newnode->prev = temp;
            temp->next = newnode;
            temp->next->prev = newnode;
        }
    }
}

// Function to delete from the front
// of the linked list
void deleteFirst()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    else {
        temp = start;
        start = start->next;
        if (start != NULL)
            start->prev = NULL;
        free(temp);
    }
}

// Function to delete from the end
// of the linked list
void deleteEnd()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    temp = start;
    while (temp->next != NULL)
        temp = temp->next;
    if (start->next == NULL)
        start = NULL;
    else {
        temp->prev->next = NULL;
        free(temp);
    }
    free(temp);
}
```

```c
// Function to delete from any specified
// position from the linked list
void deletePosition()
{
    int pos, i = 1;
    struct node *temp, *position;
    temp = start;

    // If DLL is empty
    if (start == NULL)
        printf("\nList is empty\n");

    // Otherwise
    else {
        // Position to be deleted
        printf("\nEnter position : ");
        scanf("%d", &pos);

        // If the position is the first node
        if (pos == 1) {
            deleteFirst();
            if (start != NULL) {
                start->prev = NULL;
            }
            free(position);
            return;
        }

        // Traverse till position
        while (i < pos - 1) {
            temp = temp->next;
            i++;
        }
        // Change Links
        position = temp->next;
        if (position->next != NULL)
            position->next->prev = temp;
        temp->next = position->next;

        // Free memory
        free(position);
    }
```

```
}
void deleteAftNode()
{
    int n, key, i, pos;
    struct node *temp, *position;
    temp = start;
    printf("Enter the value of node after which you want to
insert a new node\n");
    scanf("%d", &key);
    for (i = 1; temp != NULL; i++)
    {
        if (temp->val == key)
            n = i;
        temp = temp->next;
    }
    if (n == 0)
        printf("\nNode is Absent !\n");
    else
    {
        // If DLL is empty
        if (start == NULL)
            printf("\nList is empty\n");

        // Otherwise
        else
        {
            // Position to be deleted
            printf("\nEnter position : ");
            scanf("%d", &pos);

            // If the position is the first node
            if (pos == 1)
            {
                deleteFirst();
                if (start != NULL)
                {
                    start->prev = NULL;
                }
                free(position);
                return;
            }
```

```
            // Traverse till position
            while (i < pos - 1)
            {
                temp = temp->next;
                i++;
            }
            // Change Links
            position = temp->next;
            if (position->next != NULL)
                position->next->prev = temp;
            temp->next = position->next;

            // Free memory
            free(position);
        }
    }
}
void sort(struct node *start)
{
    int tmp;
    struct node *i,*j;
    for(i=start;i!=NULL;i=i->next)
    {
        for(j=i;j!=NULL;j=j->next)
        {
            if(i->val>j->val)
            {
                tmp=i->val;
                i->val=j->val;
                j->val=tmp;
            }
        }
    }
}
```