Program:
Design and implement a dynamic list (Singly linked list/ doubly linked list) to store any
information which needs a linear data structure.

Aim:
To implement a Doubly linked list

Algorithm:

Beginsert

- Firstly make a pointer and allocate some memory to it.
- If the pointer is NULL then display an memory error
- Then check if the list is full and display appropriate error
- Else set the data element of pointer to value which the user intends to insert
- Then set it's next element to head and its prev element to NULL
- Set Head's prev to the pointer
- Lastly set the head element to point to new pointer

Lstinsert

- Firstly make a pointer and allocate some memory to it.
- If the pointer is NULL then display an memory error
- Then check if the list is full and display appropriate error
- Else set the data element of pointer to value which the user intends to insert
- Now check if the list is empty
- If it is empty then set ptr's next and prev element to NULL and set head to ptr
- If it's not empty then make a new pointer and traverse to the last element in the list
- Now set it's next element to ptr and set ptr's next element to NULL
- Also set ptr's prev element to the last element

RandInsert

- Firstly make a pointer and allocate some memory to it.

- If the pointer is NULL then display an memory error
- Then check if the list is full and display appropriate error
- Else set the data element of pointer to value which the user intends to insert
- Now take input from the user specifying the location where the node is to be inserted
- Now traverse to the location
- Set ptr's next element to temp's next element
- And set ptr's prev element to temp
- Also set ptr's next element's prev to ptr
- And now set temp's next element to ptr

Begdelete

- Firstly check if the list is empty
- If its empty then print the underflow condition
- If it's not empty then set head to it's next element and free the block of memory that head was occupying previously
- Also set the head's prev element to NULL

Lstdelete

- Firstly check if the list is empty
- If its empty then print the underflow condition
- Now check if head is the only element by checking if its next element is NULL
- If it's NULL then free the memory and set head to NULL
- Else, traverse to the second last element of the list
- Now set the next element of this pointer to NULL and free the memory block from the traversing pointer

Randdelete

- Firstly take input from user about the location where the node is to be inserted
- Now traverse to the location
- If while traversing the to the location the list ends, then return the control flow and display appropriate error
- If ptr successfully traverses to the specified location, set the element's prev's next to its next element's next element

- Finally free the memory block that the traversing pointer holds

Program

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    struct node *prev;
    struct node *next;
    int data; // creating the doubly linked node
};
struct node *head;
void beginsert();
void lstinsert();
void randinsert();
void begdelete();
void lstdelete();
void randelete();
void display();
void search();
void main() // main function
{
    int choice = 0;
    while (choice != 9)
    {
        printf("\n1.Insert in beginning\n2.Insert at last\n3.Insert at any
random location\n4.Delete at beginning\n5.Delete from last \n6.Delete at
random\n7.Search\n8.Display\n9.Exit\n ");
        printf("\nEnter your choice:\n");
        scanf("\n%d", &choice);
        switch (choice)
        {
        case 1:
            beginsert();
            break;
        case 2:
```

```
            lstinsert();
            break;
        case 3:
            randinsert();
            break;
        case 4:
            begdelete();
            break;
        case 5:
            lstdelete();
            break;
        case 6:
            randelete();
            break;
        case 7:
            search();
            break;
        case 8:
            display();
            break;
        case 9:
            exit(0);
            break;
        default:
            printf("Please enter valid choice..");
        }
    }
}
void beginsert() // Function to insert data in the beginning
{
    struct node *ptr;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter Item value");
```

```
        scanf("%d", &item);
        ptr->data = item;
        if (head == NULL)
        {
            ptr->next = NULL;
            ptr->prev = NULL;
            head = ptr;
        }
        else
        {

            ptr->prev = NULL;
            ptr->next = head;
            head->prev = ptr;
            head = ptr;
        }
        printf("\nNode inserted\n");
    }
}
void lstinsert() // Function to insert data at the end
{
    struct node *ptr, *temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value");
        scanf("%d", &item);
        ptr->data = item;
        if (head == NULL)
        {
            ptr->next = NULL;
            ptr->prev = NULL;
            head = ptr;
        }
        else
```

```
        {
            temp = head;
            while (temp->next != NULL)
            {
                temp = temp->next;
            }
            temp->next = ptr;
            ptr->prev = temp;
            ptr->next = NULL;
        }
    }
    printf("\nnode inserted\n");
}
void randinsert() // function to insert at random specified location in
the node
{
    struct node *ptr, *temp;
    int item, loc, i;
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("\n OVERFLOW");
    }
    else
    {
        temp = head;
        printf("Enter the location after which you want to insert:");
        scanf("%d", &loc);

        for (i = 0; i < loc; i++)
        {
            temp = temp->next;
            if (temp == NULL)
            {
                printf("\n There are less than %d elements", loc);
                return;
            }
        }

        printf("Enter value");
```

```
        scanf("%d", &item);
        ptr->data = item;
        ptr->next = temp->next;
        ptr->prev = temp;
        temp->next = ptr;
        temp->next->prev = ptr;
        printf("\nnode inserted\n");
    }
}
void begdelete() // function to delete node at beginning
{
    struct node *ptr;
    if (head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if (head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        head = head->next;
        head->prev = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
}
void lstdelete() // function to delete last node of ll
{
    struct node *ptr;
    if (head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if (head->next == NULL)
    {
```

```
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        ptr->prev->next = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
}
void randelete() // deleting node at specified location
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the data after which the node is to be deleted : ");
    scanf("%d", &val);
    ptr = head;
    while (ptr->data != val)
        ptr = ptr->next;
    if (ptr->next == NULL)
    {
        printf("\nCan't delete\n");
    }
    else if (ptr->next->next == NULL)
    {
        ptr->next = NULL;
    }
    else
    {
        temp = ptr->next;
        ptr->next = temp->next;
        temp->next->prev = ptr;
        free(temp);
        printf("\nnode deleted\n");
```

```
    }
}
void display() // function to display all the elements in the ll
{
    struct node *ptr;

    ptr = head;
    if (head == NULL)
    {
        printf("\nUnderflow!!!!\n");
    }
    else
    {
        printf("\n printing values...\n");
        while (ptr != NULL)
        {
            printf("%d\n", ptr->data); // displaying the ll elements
            ptr = ptr->next;
        }
    }
}
void search() // function to search the required element in LL
{
    struct node *ptr;
    int item, i = 0, flag;
    ptr = head;
    if (ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d", &item);
        while (ptr != NULL)
        {
            if (ptr->data == item)
            {
                printf("\nitem found at location %d ", i + 1);
                flag = 0;
```

```
                break;
            }
            else
            {
                flag = 1;
            }
            i++;
            ptr = ptr->next;
        }
        if (flag == 1)
        {
            printf("\nItem not found\n");
        }
    }
}
```

Output

```
1.Insert in beginning
2.Insert at last
3.Insert at any random location
4.Delete at beginning
5.Delete from last
6.Delete at random
7.Search
8.Display
9.Exit

Enter your choice:
1

Enter Item value3

Node inserted

1.Insert in beginning
2.Insert at last
3.Insert at any random location
4.Delete at beginning
5.Delete from last
6.Delete at random
7.Search
8.Display
9.Exit

Enter your choice:

2

Enter value1

node inserted

1.Insert in beginning
2.Insert at last
3.Insert at any random location
4.Delete at beginning
5.Delete from last
6.Delete at random
7.Search
8.Display
9.Exit

Enter your choice:
8

 printing values...
3
1
```