Course Name:	**Data Structures Laboratory**	**Semester:**	**III**
Date of Performance:	**11 / 08 / 2025**	**Batch No:**	**A1**
Faculty Name:	**Prof. Sushma Kadge**	**Roll No:**	**20**
Faculty Sign & Date:		**Grade/Marks:**	**/25**

Experiment No: 2
Title: Linked List

Aim and Objective of the Experiment:

To understand basics of linked list and its operations.

Write a program to -
a) Create a singly linked list of integers. 97, 53, 367, 76, 121,10
b) Insert an integer 32 in the beginning of the linked list
c) Delete any value specified by the user in the above linked list.
d) Display the contents of the above list after Deletion.
e) Search any value specified by the user in the linked list.

COs to be achieved:

CO1 : Understand and implement the different data structures used in problem solving
CO2: Apply linear and non-linear data structure in application development

Books/Journals/Websites referred:

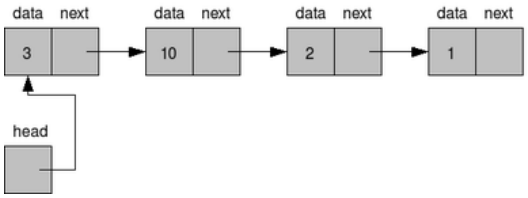
1. GeeksforGeeks.com

Tools required:

DEV C/C++ compiler/ Code blocks C compiler

Theory:

A linear list is a list where each element has a unique successor. There are four common operations associated with a linear list: insertion, deletion, retrieval, and traversal. Linear lists can be divided into two categories: general list and restricted list. In a general list the data can be inserted or deleted without any restriction whereas in a restricted list there are restrictions for these operations. Linked lists and arrays are commonly used to implement general linear lists. A linked list is simply a chain of structures which contain a pointer to the next element. It is dynamic in nature. Items may be added to it or deleted from it at will.



:

A list item has a pointer to the next element, or to NULL if the current element is the tail (end of the list). This pointer points to a structure of the same type as itself. This structure that contains elements and pointers to the next structure is called a Node.

Implementation details:

1. Enlist all the Steps followed and various options explored
2. Explain your program logic and methods used.
3. Explain the Importance of the approach followed by you

=>

1. A singly linked list was initialized with the integers **97, 53, 367, 76, 121, 10**.
2. The insertion operation added the integer **32** at the beginning of the list.
3. The deletion operation removed a user-specified value, with a message displayed if the value was not found.
4. The display operation printed the elements of the list after each modification.
5. The search operation located a user-specified value and displayed its position, or indicated that the value was not present in the list.

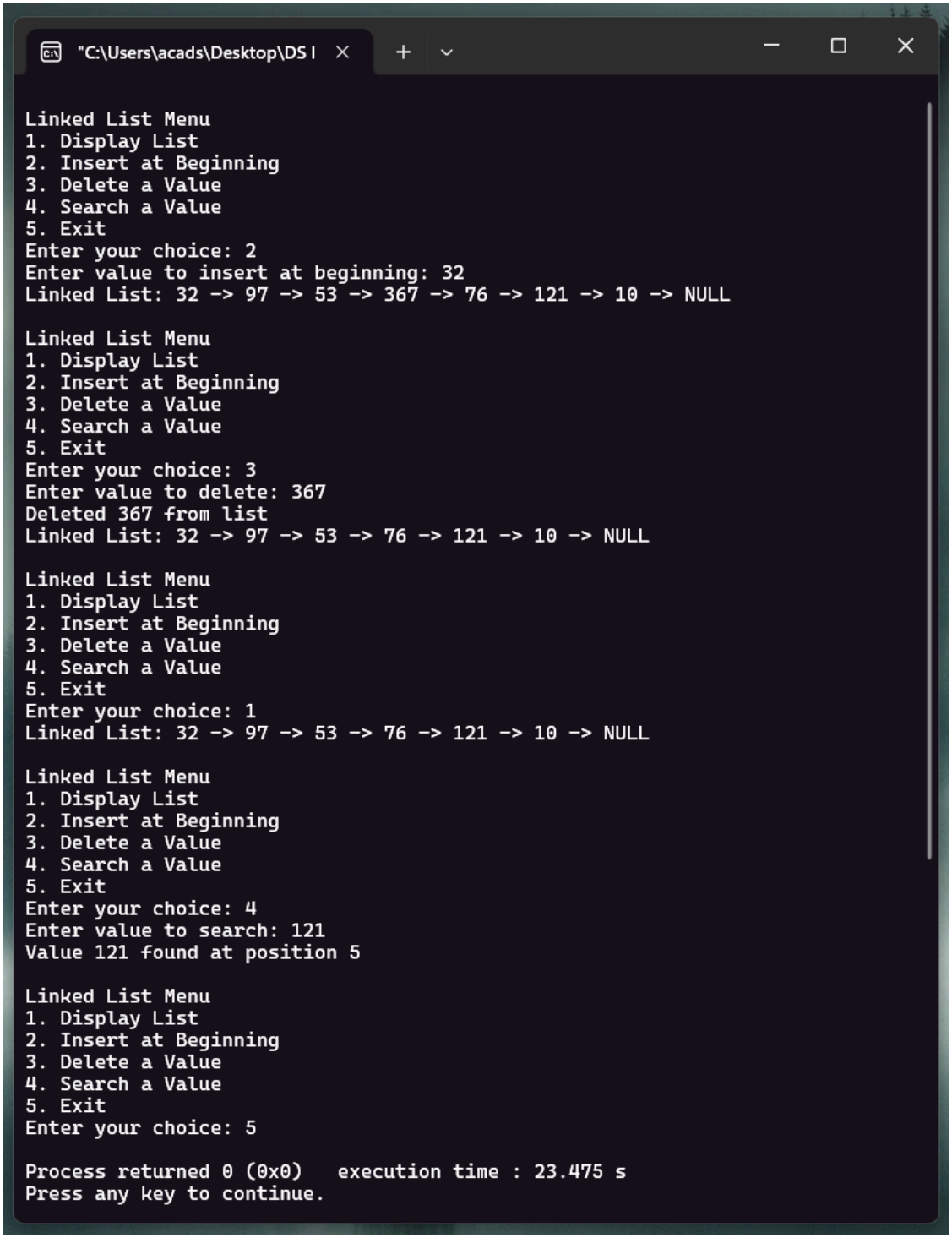
C/C++ Code implemented:

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

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

struct Node *head = NULL;

void createList()
{
    int arr[] = {97, 53, 367, 76, 121, 10};
    int n = 6, i;
    struct Node *temp, *last;
    for(i = 0; i < n; i++)
    {
        temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = arr[i];
        temp->next = NULL;
```

```
        if(head == NULL)
        {
            head = temp;
            last = head;
        }
        else
        {
            last->next = temp;
            last = temp;
        }
    }
}

void insertAtBeginning(int val)
{
    struct Node *temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = val;
    temp->next = head;
    head = temp;
}

void deleteValue(int val)
{
    struct Node *temp = head, *prev = NULL;
    while(temp != NULL && temp->data != val)
    {
        prev = temp;
        temp = temp->next;
    }
    if(temp == NULL)
    {
        printf("Value not found, cannot delete\n");
        return;
    }
    if(prev == NULL)
        head = temp->next;
    else
        prev->next = temp->next;

    free(temp);
    printf("Deleted %d from list\n", val);
}

void searchValue(int val)
{
    struct Node *temp = head;
    int pos = 1, found = 0;
    while(temp != NULL)
    {
        if(temp->data == val)
        {
            printf("Value %d found at position %d\n", val, pos);
            found = 1;
            break;
        }
        temp = temp->next;
        pos++;
    }
    if(!found)
        printf("Value %d not found in the list\n", val);
}
```

```c
void displayList()
{
    struct Node *temp = head;
    printf("Linked List: ");
    while(temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main()
{
    int ch, val;
    createList();

    do
    {
        printf("\nLinked List Menu\n");
        printf("1. Display List\n");
        printf("2. Insert at Beginning\n");
        printf("3. Delete a Value\n");
        printf("4. Search a Value\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch(ch)
        {
        case 1:
            displayList();
            break;
        case 2:
            printf("Enter value to insert at beginning: ");
            scanf("%d", &val);
            insertAtBeginning(val);
            displayList();
            break;
        case 3:
            printf("Enter value to delete: ");
            scanf("%d", &val);
            deleteValue(val);
            displayList();
            break;
        case 4:
            printf("Enter value to search: ");
            scanf("%d", &val);
            searchValue(val);
            break;
        case 5:
            break;
        default:
            printf("Invalid choice, try again\n");
        }
    } while(ch != 5);

    return 0;
}
```

Output/ program results after execution:



Post Lab Subjective/Objective type Questions:

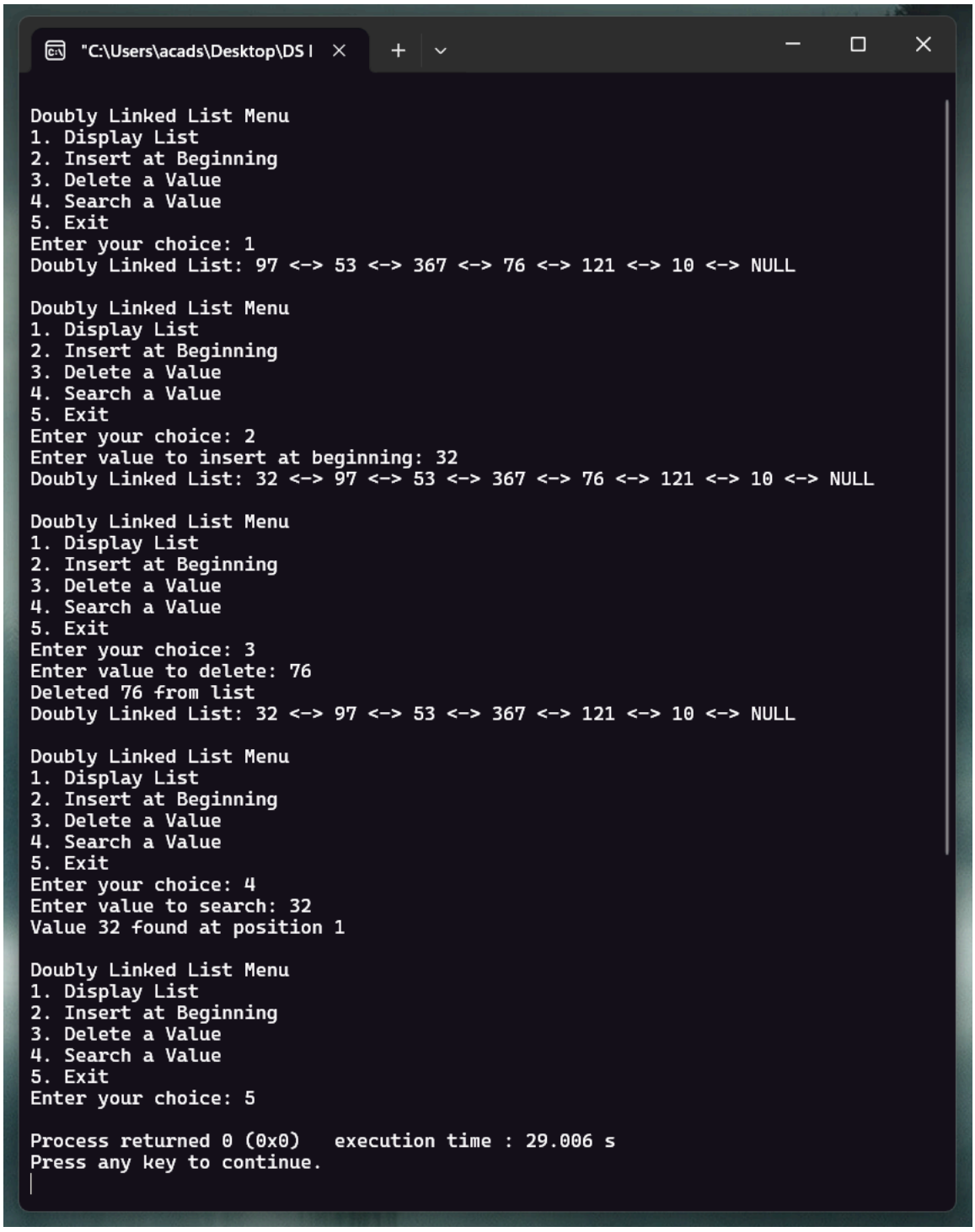
1. Modify the above code for doubly LL/ circular LL/ circular doubly LL (any one)
Program:

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

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

struct Node *head = NULL;

void createList()
{
    int arr[] = {97, 53, 367, 76, 121, 10};
    int n = 6, i;
    struct Node *temp, *last = NULL;
    for(i = 0; i < n; i++)
    {
        temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = arr[i];
        temp->prev = NULL;
        temp->next = NULL;
        if(head == NULL)
        {
            head = temp;
            last = head;
        }
        else
        {
            last->next = temp;
            temp->prev = last;
            last = temp;
        }
    }
}

void insertAtBeginning(int val)
{
    struct Node *temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = val;
    temp->prev = NULL;
    temp->next = head;
    if(head != NULL)
        head->prev = temp;
    head = temp;
}
```

```c
void deleteValue(int val)
{
    struct Node *temp = head;
    while(temp != NULL && temp->data != val)
        temp = temp->next;


    if(temp == NULL)
    {
        printf("Value not found, cannot delete\n");
        return;
    }

    if(temp->prev != NULL)
        temp->prev->next = temp->next;
    else
        head = temp->next;

    if(temp->next != NULL)
        temp->next->prev = temp->prev;

    free(temp);
    printf("Deleted %d from list\n", val);
}

void searchValue(int val)
{
    struct Node *temp = head;
    int pos = 1, found = 0;
    while(temp != NULL)
    {
        if(temp->data == val)
        {
            printf("Value %d found at position %d\n", val, pos);
            found = 1;
            break;
        }
        temp = temp->next;
        pos++;
    }
    if(!found)
        printf("Value %d not found in the list\n", val);
}

void displayList()
{
    struct Node *temp = head;
    printf("Doubly Linked List: ");
    while(temp != NULL)
    {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

```c
int main()
{
    int ch, val;
    createList();

    do
    {
        printf("\nDoubly Linked List Menu\n");
        printf("1. Display List\n");
        printf("2. Insert at Beginning\n");
        printf("3. Delete a Value\n");
        printf("4. Search a Value\n");
        printf("5. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch(ch)
        {
        case 1:
            displayList();
            break;
        case 2:
            printf("Enter value to insert at beginning: ");
            scanf("%d", &val);
            insertAtBeginning(val);
            displayList();
            break;
        case 3:
            printf("Enter value to delete: ");
            scanf("%d", &val);
            deleteValue(val);
            displayList();
            break;
        case 4:
            printf("Enter value to search: ");
            scanf("%d", &val);
            searchValue(val);
            break;
        case 5:
            break;
        default:
            printf("Invalid choice, try again\n");
        }
    } while(ch != 5);

    return 0;
}
```

Output:


"C:\Users\acads\Desktop\DS I
Doubly Linked List Menu
1. Display List
2. Insert at Beginning
3. Delete a Value
4. Search a Value
5. Exit
Enter your choice: 1
Doubly Linked List: 97 <-> 53 <-> 367 <-> 76 <-> 121 <-> 10 <-> NULL
Doubly Linked List Menu
1. Display List
2. Insert at Beginning
3. Delete a Value
4. Search a Value
5. Exit
Enter your choice: 2
Enter value to insert at beginning: 32
Doubly Linked List: 32 <-> 97 <-> 53 <-> 367 <-> 76 <-> 121 <-> 10 <-> NULL
Doubly Linked List Menu
1. Display List
2. Insert at Beginning
3. Delete a Value
4. Search a Value
5. Exit
Enter your choice: 3
Enter value to delete: 76
Deleted 76 from list
Doubly Linked List: 32 <-> 97 <-> 53 <-> 367 <-> 121 <-> 10 <-> NULL
Doubly Linked List Menu
1. Display List
2. Insert at Beginning
3. Delete a Value
4. Search a Value
5. Exit
Enter your choice: 4
Enter value to search: 32
Value 32 found at position 1
Doubly Linked List Menu
1. Display List
2. Insert at Beginning
3. Delete a Value
4. Search a Value
5. Exit
Enter your choice: 5
Process returned 0 (0x0) execution time : 29.006 s
Press any key to continue.

Conclusion:

The experiment successfully demonstrated the implementation of both **singly linked list** and **doubly linked list**. In the singly linked list, insertion at the beginning, deletion of a specified value, searching for a value, and displaying the list were carried out. In the doubly linked list, the same operations were performed with the added advantage of bidirectional traversal using previous and next pointers.

Signature of faculty in-charge with Date: