PRACTICAL NO.04

SINGLY LINKED LIST

Learning Objectives

PLO	CLO	LL
4	1	C-3

By the end of this lab, students will be able to:

- Understand the concept of **dynamic memory allocation** and **pointers**.
- Implement a **singly linked list** with insertion, deletion, traversal, and searching.
- Apply linked list concepts to solve a **real-world problem**.

Background

A **singly linked list** is a linear data structure where each element (node) contains:

1. **Data** – the actual value.
2. **Pointer** – the address of the next node.

Unlike arrays, linked lists provide:

- **Dynamic size allocation** (nodes can be added/removed at runtime).
- **Efficient insertion/deletion** (no shifting of elements like arrays).

Practical Tasks

Write the complete code in your own system and check the results.

```
# include <iostream>

using namespace std;

int count =0;

struct node{

        int data;

        node *next;

};

*head,*tail,*nn,*temp;
```

//create new node

```
void create_list (int a)
{ if(head==NULL) {
nn= new node;
nn->data= a;
nn-> next = null;
head=nn;
tail=nn;
cout=count+1;   }
else{
cout<<"list already created";      }
}
```

// insertion at the beginnig

```
void insert_at_first(int a)
{ if (head==NULL) {
 cout<<"create the list first"; }
else{ nn=new node;
nn->data=a;
nn->next=head;
head=nn;
count=count+1; } }
```

//insert at the last

```
void insert_at_last (int a) {
if (head==NULL) {
```

```
cout<<"create the list"<<endl;}
else{
nn=new node;
nn->data=a;
nn->next = NULL;
tail->next = nn;
tail=nn;
count=count+1; } }
```

//insert at the specfic location

```
void insert_at_specfic_pos (int pos, int a) {
if(head==NULL) {
cout<<"create the list first"<<endl; }
else {
int first = 1;
temp=head;
while(temp!=NULL) {
if(pos-1== first) {
nn=new node;
nn->data=a;
nn->next=temp->next;
temp->next=nn; }
first++;
temp=temp->next;
}          count=count+1; } }
```

//display

```
void display() {
temp=head;
while(temp!=null) {
cout<<" "<<temp->data;
temp=temp->next; } }
```

// DELETION

//DELETE THE FIRST NODE

```
void delete_at_first () {
temp=head->next;
head= null;
head=temp;
count =count-1;}
void delete_at_last() {
temp=head;
while(temp!=NULL) {
if (temp->next==tail) {
temp->next=null;
tail=temp; }
temp=temp->next; }
count = count--; }
void delete_at_specfic_pos (int pos) {
int first=1;
node *temp2;
```

```
node *temp3;
temp = head;
while(temp!=NULL) {
if (pos == first)
{ temp2=temp->next;
temp3 = temp2->next;
temp->next= temp3; }
first++;
temp=temp->next; }
count --; }
void searching (int no) {
int count = 0;
int first =1;
temp = head;
while(temp!= NULL) {
if(temp->data == no){
count ++;
break; }
temp=temp->next;
first++; }
if (count!=0) {
cout<<"value found at the position "<<first<<endl; } else {
cout<<"value not found"<<endl; } }
```

Step 2: Basic Operations

Task 1: Create a Linked List

- Implement the function `create_list(int a)` to initialize the first node.
- Test by creating a list with the first element `10`.

Task 2: Insert Elements

- Insert elements **at the beginning, at the end, and at a specific position**.
- Example: Insert `20` at the beginning, `30` at the end, and `25` at position `2`.

Task 3: Display the List

- Print the list after each insertion to check updates.

Task 4: Delete Elements

- Delete from:
  - Beginning
  - End
  - A specific position
- Observe the updated list after each deletion.

Task 5: Search an Element

- Search for values like `25` and `100` to test found/not found cases.

Real-World Problems for Singly Linked List

Web Browser History (Singly Linked List)

Problem Statement

You are asked to implement a simplified **Web Browser History System** using **singly linked list**.

- Each visited website will be stored as a **node**.
- The system should allow you to:
  1. **Add new websites visited** (insert at the end).
  2. **Search** if a website was visited.
  3. **Delete history entries** (delete from beginning, end, or specific position).
  4. **Display all history entries**.

C++ Implementation

```
#include <iostream>

#include <string>

using namespace std;

struct Node {

    string website;   // store website name

    Node* next; };;

Node* head = NULL;

Node* tail = NULL;

// Add new website at the end

void addWebsite(string site) {

 Node* nn = new Node;

  nn->website = site;

  nn->next = NULL;

   if (head == NULL) {

        head = nn;

        tail = nn;

    } else {

     tail->next = nn;

     tail = nn; }

    cout << site << " visited and added to history." << endl; }

// Display browsing history

void displayHistory() {

  if (head == NULL) {
```

```
    cout << "History is empty." << endl;
      return; }
    cout << "Browsing History: ";
    Node* temp = head;
    while (temp != NULL) {
    cout << temp->website << " -> ";
     temp = temp->next; }
    cout << "NULL" << endl; }
// Search for a website
void searchWebsite(string site) {
    Node* temp = head;
    int pos = 1;
    while (temp != NULL) {
        if (temp->website == site) {
            cout << site << " found at position " << pos << " in history." << endl;
            return; }
        temp = temp->next;
        pos++;}
    cout << site << " not found in history." << endl; }
// Delete first entry
void deleteFirst() {
    if (head == NULL) {
        cout << "History is empty." << endl;
        return; }
```

```
    Node* temp = head;
    head = head->next;
    delete temp;
    cout << "First history entry deleted." << endl; }
// Delete last entry
void deleteLast() {
    if (head == NULL) {
        cout << "History is empty." << endl;
        return; }
    if (head == tail) {  // Only one node
        delete head;
        head = tail = NULL;
    } else {
        Node* temp = head;
        while (temp->next != tail) {
            temp = temp->next; }
        delete tail;
        tail = temp;
        tail->next = NULL; }
    cout << "Last history entry deleted." << endl; }
// Delete specific position
void deleteAtPos(int pos) {
    if (head == NULL) {
        cout << "History is empty." << endl;
```

```
        return; }
    if (pos == 1) {
        deleteFirst();
        return; }
    Node* temp = head;
    for (int i = 1; temp != NULL && i < pos - 1; i++) {
        temp = temp->next; }
    if (temp == NULL || temp->next == NULL) {
        cout << "Invalid position." << endl;
        return; }
    Node* del = temp->next;
    temp->next = del->next;
    if (del == tail) {
        tail = temp; }
    delete del;
    cout << "History entry at position " << pos << " deleted." << endl;}
// Main function
int main() {
    addWebsite("google.com");
    addWebsite("youtube.com");
    addWebsite("wikipedia.org");
    addWebsite("github.com");
    displayHistory();
```

```
searchWebsite("youtube.com");

searchWebsite("twitter.com");

deleteFirst();

displayHistory();

deleteLast();

displayHistory();

deleteAtPos(2);

displayHistory();

return 0;}
```

Sample Output

```
google.com visited and added to history.
youtube.com visited and added to history.
wikipedia.org visited and added to history.
github.com visited and added to history.

Browsing History: google.com -> youtube.com -> wikipedia.org -> github.com -> NULL

youtube.com found at position 2 in history.
twitter.com not found in history.

First history entry deleted.
Browsing History: youtube.com -> wikipedia.org -> github.com -> NULL

Last history entry deleted.
Browsing History: youtube.com -> wikipedia.org -> NULL

History entry at position 2 deleted.
Browsing History: youtube.com -> NULL
```

Programming Assignment: Music Playlist using Singly Linked List

Problem Statement

You are asked to implement a **Music Playlist System** using **singly linked list** in C++.

Each song will be stored as a **node**, containing:

- `Song Name` (string)
- `Pointer to next song`

Your program should support the following operations:

1. **Add Songs**
    - Add a song at the **beginning** of the playlist.
    - Add a song at the **end** of the playlist.
    - Add a song at a **specific position** in the playlist.
2. **Delete Songs**
    - Delete the **first song** from the playlist.
    - Delete the **last song** from the playlist.
    - Delete the song at a **specific position**.
3. **Search a Song**
    - Search for a song by its **name**.
    - Display its **position** in the playlist if found, otherwise show `"Song not found"`.
4. **Display Playlist**
    - Print all the songs in order.
    - Example:
    - `Playlist: SongA -> SongB -> SongC -> NULL`

Student Task

- Implement the above operations in C++ using a **singly linked list**.
- Create a playlist with the following sequence of operations:
    1. Add `"Shape of You"` at the beginning.
    2. Add `"Believer"` at the end.
    3. Add `"Perfect"` at the end.
    4. Insert `"Thunder"` at position 2.
    5. Display the playlist.
    6. Delete the first song.
    7. Delete the last song.
    8. Delete the song at position 2.
    9. Search for `"Believer"`.
    10. Search for `"Faded"`.
    11. Display the final playlist.

Expected Output

```
Playlist: Shape of You -> Thunder -> Believer -> Perfect -> NULL
After deleting first: Thunder -> Believer -> Perfect -> NULL
After deleting last: Thunder -> Believer -> NULL
After deleting at position 2: Thunder -> NULL
Searching Believer: Song found at position 2
Searching Faded: Song not found
Final Playlist: Thunder -> NULL
```