# Previous Lab Discussion

## Stack

A stack is a linear data structure that follows the Last In First Out (LIFO)principle. This means that the last element added to the stack is the first element to be removed. Stacks are typically implemented using arrays or linked lists.

The basic operations on a stack are:

- **Push:** Adds an element to the top of the stack.
- **Pop:** Removes the top element from the stack.
- **Display:** Displays all the elements in the stack, in order from top to bottom.

# Current Lab Discussion

**Linked List**

Linked lists are dynamic data structures, meaning that they can grow and shrink as needed. This is in contrast to arrays, which have a fixed size. When adding or removing elements from a linked list, only the nodes affected by the operation need to be updated. This can make linked lists more efficient than arrays for certain operations, such as insertion and deletion.

# TASK-05

**Create a menu driven program for Linked List.**
**Include all operations in the menu, which are as follows:**
**1. Insert at Head**
**2. Insert at Tail**
**3. Insert After**
**4. Insert Before**
**5. Delete from Head**
**6. Delete from Tail**
**7. Delete Node**
**8. Traverse List**

# Code:

- **node.h File**

```
#include<iostream>
using namespace std;

class Node
{
	public:
		double data;
		Node* next;
		Node(double i=0, Node* n=0)
		{
			data = i;
			next = n;
		}
};
```

- **linkedlist.h file**

```
#include "node.h"
#include <iostream>
using namespace std;

class linkedlist
{
	private:
		Node* head;
		Node* tail;
	public:
		linkedlist()
		{
			head = 0;
			tail = 0;
```

```
                }
                void insertathead(double value);
                void insertattail(double value);
                void insertafter(double existing, double value);
                void insertbefore(double existing, double value);
                void deletefromhead();
                void deletefromtail();
                void deletenode(double value);
                void traverselist();
                bool isempty();
};
bool linkedlist::isempty()
        {
                if(head == 0 && tail == 0)
                {
                        return true;
                }
                else
                {
                        return false;
                }
        }
void linkedlist::insertathead(double value)
        {
                Node* newnode = new Node(value);
                if(head == 0 && tail == 0)
                {
                        head = tail = newnode;
                }
                else
                {
                        newnode->next = head;  // -> for accessing data
member
                        head = newnode;
```

```
                }
        }
void linkedlist::insertattail(double value)
        {
                Node* newnode = new Node(value);
                if(head == 0 && tail == 0)
                {
                        head = tail = newnode;
                }
                else
                {
                        tail->next = newnode;
                        tail = newnode;
                }
        }
void linkedlist::insertafter(double existing, double value)
        {
                if(isempty())
                {
                        cout<<"\nList is empty.";
                }
                else if(existing == tail->data)
                {
                        insertattail(value);
                }
                else
                {
                        Node* currnode = head;
                        while(currnode != 0 && currnode->data !=
existing)
                        {
                                currnode = currnode->next;
                        }
                        if(currnode==0)
```

```
                {
                        cout<<"\nInsertion is not possible in the list because existing element in not present in the list.";
                }
                else
                {
                        Node* newnode = new Node(value);
                        newnode->next = currnode->next;
                        currnode->next = newnode;
                }
        }
}
void linkedlist::insertbefore(double existing, double value)
{
        if(isempty())
        {
                cout<<"\nList is empty.";
        }
        else if(existing == head->data)
        {
                insertathead(value);
        }
        else
        {
                Node* prevnode = 0;
                Node* currnode = head;
                while(currnode != 0 && currnode->data != existing)
                {
                        prevnode = currnode;
                        currnode = currnode->next;
                }
                if(currnode==0)
                {
```

```
                cout<<"\nInsertion is not possible in the list because existing element in not present in the list.";
            }
            else
            {
                Node* newnode = new Node(value);
                newnode->next = currnode;        // newnode = currnode
                prevnode->next = newnode;        // currnode->next = currnode
            }
        }
    }
void linkedlist::traverselist()
    {
        if(isempty())
        {
            cout<<"\nList is empty.";
        }
        else
        {
            cout<<"\nValues in list are: "<<endl;
            Node* currnode = head;
            while(currnode != 0)
            {
                cout<<currnode->data<<endl;
                currnode = currnode->next;
            }
        }
    }
void linkedlist::deletefromhead()
    {
        if(isempty())
        {
```

```
                cout<<"\nList is empty.";
            }
            else if(head==tail)
            {
                Node* currnode = head;
                head = tail = 0;
                delete currnode;
            }
            else
            {
                Node* currnode = head;
                head = head->next;
                currnode->next = 0;
                delete currnode;
            }
        }
    void linkedlist::deletefromtail()
        {
            if(isempty())
            {
                cout<<"\nList is empty.";
            }
            else if(tail==head)
            {
                Node* currnode = tail;
                tail = head = 0;
                delete currnode;
            }
            else
            {
                Node* currnode = head;
                Node* prevnode = 0;
                while(currnode->next!=0)
                {
```

```
                prevnode = currnode;
                currnode = currnode->next;
            }
            tail = prevnode;
            tail->next = 0;
            delete currnode;
        }
    }
void linkedlist::deletenode(double value)
    {
        if(isempty())
        {
            cout<<"\nList is empty.";
        }
        else if(head->data == value)
        {
            deletefromhead();
        }
        else if(tail->data == value)
        {
            deletefromtail();
        }
        else
        {
            Node* currnode = head;
            Node* prevnode = 0;
            while(currnode!=0 && currnode->data!=value)
            {
                prevnode = currnode;
                currnode = currnode->next;
            }
            if(currnode==0)
            {
                cout<<"\nValue doesn't exist in the list.";
```

```
                }
                else
                {
                        prevnode->next = currnode->next;
                        currnode->next=0;
                        delete currnode;
                }
        }

}
```

- **.cpp file**

```
#include <iostream>
#include "linkedlist.h"
using namespace std;
int main()
{
        double val;
        double existing;
        char con;
        int choice;
        linkedlist list;
        do
        {

                cout<<"\tPress 1 for insert at head"<<endl;
                cout<<"\tPress 2 for insert at tail"<<endl;
                cout<<"\tPress 3 for insert after"<<endl;
                cout<<"\tPress 4 for insert before"<<endl;
                cout<<"\tPress 5 for delete from head"<<endl;
                cout<<"\tPress 6 for delete from tail"<<endl;
                cout<<"\tPress 7 for delete from specific node"<<endl;
                cout<<"\tPress 8 for traverse node"<<endl;
```

```
cout<<"Enter choice: ";
cin>>choice;
switch (choice)
{
        case 1:
        cout<<"Enter value to insert at head: ";
        cin>>val;
        list.insertathead(val);
        break;
 case 2:
        cout<<"Enter value to insert at tail: ";
        cin>>val;
        list.insertattail(val);
    break;
case 3:
        cout<<"Enter value to insert after: ";
        cin>>existing;
        cin>>val;
        list.insertafter(existing,val);
     break;
case 4:
        cout<<"Enter value to insert before: ";
        cin>>existing;
        cin>>val;
        list.insertbefore(existing,val);
    break;
case 5:
        cout<<"Enter value to delete from head: ";
        cin>>val;
        list.deletefromhead();
    break;
case 6:
        cout<<"Enter value to delete from tail: ";
        list.deletefromtail();
```

```
                break;
            case 7:
                cout<<"Enter value for specific node deletion: ";
                cin>>val;
                list.deletenode(val);
                break;
            case 8:
                list.traverselist();
               break;
            default:
                cout<<"Sorry! Wrong choise"<<endl;
                break;
            }
            cout<<"\nPress (y) for again continue the program and press any key except (y) for exit: ";
            cin>>con;

        }
        while(con == 'y');
}
```