**Batch:____C1______ Roll No.:16010122257**

**Experiment No. 6**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Title: Implementation of Linked List**

**Objective:** To understand the use of linked list as data structures for various application.

**Expected Outcome of Experiment:**

| CO | Outcome |
|---|---|
| CO 2 | Apply linear and non-linear data structure in application development. |

**Books/ Journals/ Websites referred:**

**1)**Ma'am's classroom notes
2) *https://www.geeksforgeeks.org/data-structures/linked-list/*
3)
*https://github.com/MethkupalliVasanth/Books/blob/master/Narasimha%20Karumanchi%20-%20Data%20structures%20and%20algorithms%20made%20easy%20(0%2C%20CareerMonk).pdf*

**Introduction:**

Define Linked List

Definition:An ordered collection of homogenous data items

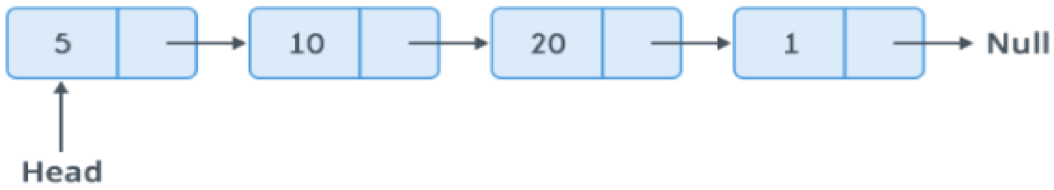
Where elements can be added anywhere and removed

from anywhere.

**Types of linked list:**

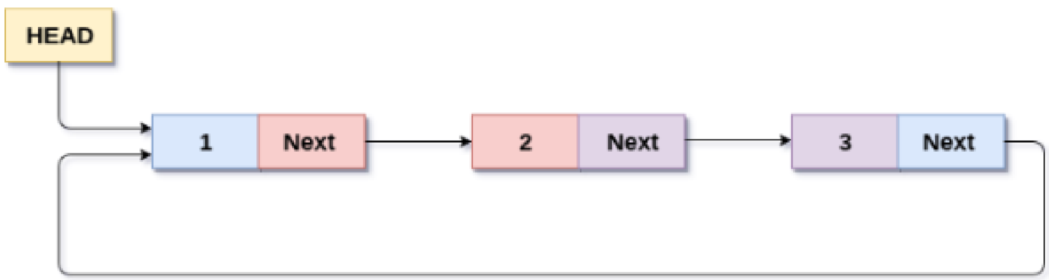
Singly linked list- can be traversed only in one

Direction



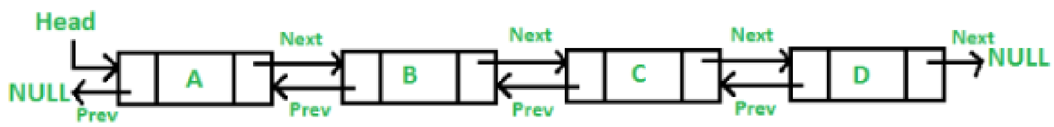

Circular linked list- last node is connected to

first node

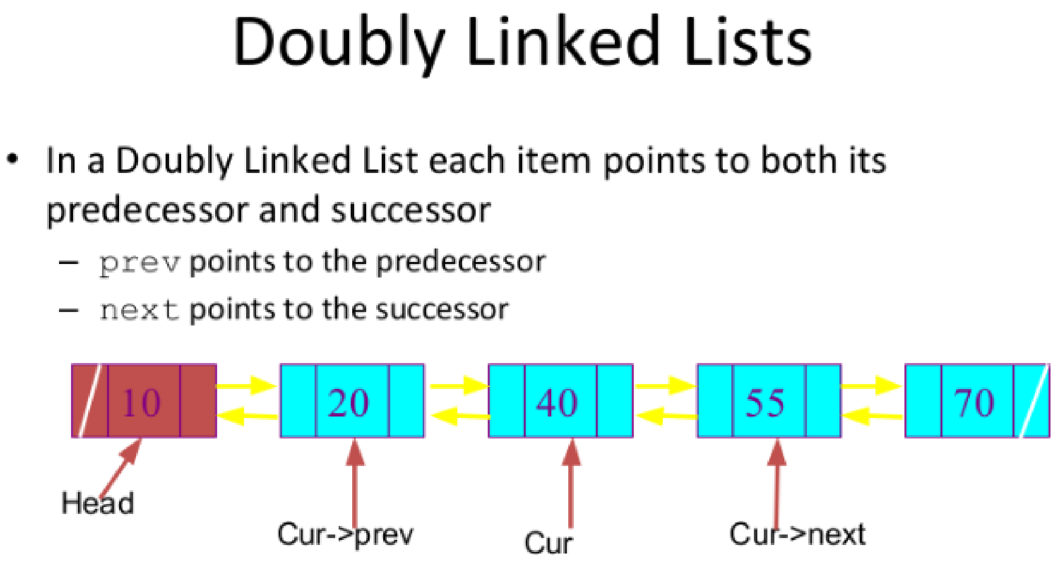



**Circular Singly Linked List**

Doubly linked list – can be traversed in both directions




**Algorithm for creation, insertion, deletion, traversal and searching an element in assigned linked list type:**




Doubly Linked List Definition

```
struct Node{

 int data;

   Node* next;

   Node* prev;

     };

     Node* Head, List1, List2;
```

Insertion into DLL

1. initialize data structure

CreateNode(Head);

Head=NULL

Search // searching in sorted list for insertion operation

temp=Head

while(NewNode->data > temp->data && temp->next!=Null)

temp=temp->next;

2. Insertion

createNode(NewNode) // CreateNode creates a node, adds data

to it and assigns both addresses NULL value

2A. insertion of first node

if(Head==NULL)

head=NewNode;

2B. insertion at the end

if(NewNode->data > temp->data && temp->next==Null)

{ NewNode->prev = temp

temp->next= NewNode

}

2C. Insert before the head node

if(NewNode->data < temp->data && temp==Head)

{ NewNode->next = temp;

temp->prev = NewNode

Head= Newnode;

}

2D. Insertion in between

{

NewNode-> Next = temp;

NewNode -> prev = temp->prev

temp-> prev->next= NewNode

temp->prev = NewNode

}

Deletion in DLL

1. Deletion from empty DLL

1A. Element doesn't exist

2. Deletion of last/only node in DLL

3. Deletion of Head node

4. Deletion of last node

5. General case

Deletion in DLL

1. if (Head==Null)

Print" Underflow.. Error“

//Search

temp=Head

while( temp!=Null && Temp->data <SearchKey)

temp=temp-> next

Deletion in DLL

1A. // unsuccessful search e.g. SearchKey ==1 or 30 or 500

if(Temp->data > SearchKey || (temp-> next ==Null && temp->data <SearchKey)) )

print "Element does not exist"

2. Deletion of only node in list

if(temp->data == SearchKey && temp=Head && temp->next ==NULL)

{ Head=NULL;

return(Temp)

}

4. //e.g. SearchKey= 100 Deletion of last node

if(temp->data == SearchKey && temp->next ==NULL)

{ temp->prev->next = NULL

return(temp)

}

3. Deletion in general

{ temp->next->prev= temp->prev

temp->prev->next= temp->next

}

5. Deletion of first node in list

if(temp==head && temp->data == SearchKey)

{ head=head->next

temp->next->prev= NULL

return(temp)

**Implementation of an application using linked list:**

**DOUBLY LINKED LIST:**

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

class Node {
public:
    int data;
    Node* prev;
    Node* next;


    Node(int d) {
        this->data = d;
        this->prev = NULL;
        this->next = NULL;
    }


    ~Node() {
```

```
        cout << "Deleting node with data " << data << endl;
    }
};

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;
    int nodeCount;

public:

    DoublyLinkedList() {
        head = NULL;
        tail = NULL;
        nodeCount = 0;
    }



    ~DoublyLinkedList() {
        Node* current = head;
        while (current != NULL) {
            Node* next = current->next;
            delete current;
            current = next;
```

```
        }
    }

    void insertAtHead(int d) {
        Node* newNode = new Node(d);
        if (head == NULL) {
            head = newNode;
            tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
        nodeCount++;
        cout << "Node with data " << d << " inserted at the head. Node count: " <<
nodeCount << endl;
    }

    void insertAtTail(int d) {
        Node* newNode = new Node(d);
        if (tail == NULL) {
            head = newNode;
            tail = newNode;
        } else {
```

```
            newNode->prev = tail;

            tail->next = newNode;

            tail = newNode;

        }

        nodeCount++;

        cout << "Node with data " << d << " inserted at the tail. Node count: " << nodeCount << endl;

    }




    void deleteNode(int d) {

        Node* nodeToDelete = search(d);

        if (nodeToDelete == NULL) {

            cout << "Node with data " << d << " not found. Node count: " << nodeCount << endl;

            return;

        }



        if (nodeToDelete == head) {

            head = nodeToDelete->next;

        } else {

            nodeToDelete->prev->next = nodeToDelete->next;

        }



        if (nodeToDelete == tail) {

            tail = nodeToDelete->prev;
```

```
        } else {
            nodeToDelete->next->prev = nodeToDelete->prev;
        }

        delete nodeToDelete;
        nodeCount--;
        cout << "Node with data " << d << " deleted. Node count: " << nodeCount <<
endl;
    }


    Node* search(int d) {
        Node* temp = head;
        while (temp != NULL) {
            if (temp->data == d) {
                return temp;
            }
            temp = temp->next;
        }
        return NULL;
    }


    void display() {
        Node* temp = head;
        while (temp != NULL) {
```

```
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList dll;

    dll.insertAtHead(5);
    dll.insertAtHead(8);
    dll.insertAtTail(4);
    dll.insertAtTail(9);
    dll.display();

    dll.deleteNode(4);
    dll.display();

    dll.deleteNode(8);
    dll.display();

    dll.deleteNode(3); // Trying to delete a node that doesn't exist
    dll.display();
```

```
    return 0;
}
```

**Output:**

```
Output                                                    Clear
/tmp/gr0chlvk7b.o
Node with data 5 inserted at the head. Node count: 1
Node with data 8 inserted at the head. Node count: 2
Node with data 4 inserted at the tail. Node count: 3
Node with data 9 inserted at the tail. Node count: 4
8 5 4 9
Deleting node with data 4
Node with data 4 deleted. Node count: 3
8 5 9
Deleting node with data 8
Node with data 8 deleted. Node count: 2
5 9
Node with data 3 not found. Node count: 2
5 9
Deleting node with data 5
Deleting node with data 9
```

**Conclusion:-Post lab questions:**

1. Compare and contrast SLL and DLL

   Singly linked list (SLL) :
   - SLL nodes contains 2 field -data field and next link field.
   - In SLL, the traversal can be done using the next node link only. Thus traversal is possible in one direction only.
   - The SLL occupies less memory than DLL as it has only 2 fields.
   - We mostly prefer to use singly linked list for the execution of stacks.
   - A singly linked list consumes less memory as compared to the doubly linked list.

   Doubly Linked List(DLL):
   - DLL nodes contains 3 fields -data field, a previous link field and a next link field.
   - In DLL, the traversal can be done using the previous node link or the next node link. Thus traversal is possible in both directions (forward and backward).
   - The DLL occupies more memory than SLL as it has 3 fields.
   - We can use a doubly linked list to execute heaps and stacks, binary trees.
   - The doubly linked list consumes more memory as compared to the singly linked list.