**HOMEWORK**

**Implement singly and doubly linked lists.**

**Ex. No: 1(a)**

**AIM: To implement a linked list and do all operations on it.**

**ALGORITHM:**

Step 1: Start the process.

Step 2: Initialize and declare variables.

Step 3: Enter the choice. INSERT / DELETE.

Step 4: If choice is INSERT then

a) Enter the element to be inserted.

b) Get a new node and set DATA[NEWNODE] = ITEM.

c) Find the node after which the new node is to be inserted.

d) Adjust the link fields.

e) Print the linked list after insertion.

Step 5: If choice is DELETE then

a) Enter the element to be deleted.

b) Find the node containing the element (LOC) and its preceding node (PAR).

c) Set ITEM = DATA [LOC] and delete the node LOC.

d) Adjust the link fields so that PAR points to the next element. ie

LINK [PAR] = LINK [LOC].

e) Print the linked list after deletion.

Step 6: Stop the process.

**Ex. No: 1(b)**

**AIM: To implement a doubly linked list and do all operations on it.**

**ALGORITHM:**

Step 1: Data type declarations

record Node {

data

prev

}

record List {

Node firstNode // points to first node of list; null for empty list

Node lastNode // points to last node of list; null for empty list

}

Step 2: Iterating over the nodes

Iterating through a doubly linked list can be done in either direction. In fact, direction can change many

times, if desired.

node := list.firstNode

while node ≠ null

<do something with node.data>

node := node.next

node := list.lastNode

while node ≠ null

<do something with node.data>

node := node.prev

Step 3:Inserting a node

function insertAfter(List list, Node node, Node newNode)

newNode.prev := node

newNode.next := node.next

if node.next == null

list.lastNode := newNode

else

node.next.prev := newNode

node.next := newNode

function insertBefore(List list, Node node, Node newNode)

newNode.prev := node.prev

newNode.next := node

if node.prev is null

list.firstNode := newNode

else

node.prev.next := newNode

node.prev := newNode

Function to insert a node at the beginning of a possibly-empty list:

function insertBeginning(List list, Node newNode)

if list.firstNode == null

list.firstNode := newNode

list.lastNode := newNode

newNode.prev := null

newNode.next := null

else

insertBefore(list, list.firstNode, newNode)

A symmetric function inserts at the end:

function insertEnd(List list, Node newNode)

if list.lastNode == null

insertBeginning(list, newNode)

else

insertAfter(list, list.lastNode, newNode)

Step 5:Deleting a node

Removing a node is easier, only requiring care with the firstNode and lastNode:

function remove(List list, Node node)

if node.prev == null

list.firstNode := node.next

else

node.prev.next := node.next

if node.next == null

list.lastNode := node.prev

else

node.next.prev := node.prev

destroy node