5.1.1 Linked List Operations

The typical operations performed on a linked list are:

1. Insert
2. Delete
3. Search
4. Print

1. Insert The insert operation adds a new element to the linked list. The following tasks are performed while adding the new element:

(a) Memory space is reserved for the new node.
(b) The element is stored in the INFO part of the new node.
(c) The new node is connected to the existing nodes in the list.

Depending on the location where the new node is to be added, there are three scenarios possible, which are:

(a) Inserting the new element at the beginning of the list
(b) Inserting the new element at the end of the list
(c) Inserting the new element somewhere at the middle of the list

Inserting a new element at the beginning or end of the list is easy as it only requires resetting the respective NEXT fields. However, if the new element is to be added somewhere at the middle of the list then a search operation is required to be performed to identify the point of insertion.

Figures 5.2 (a) and (b) show the insertion of a new element between two existing elements of a linked list.

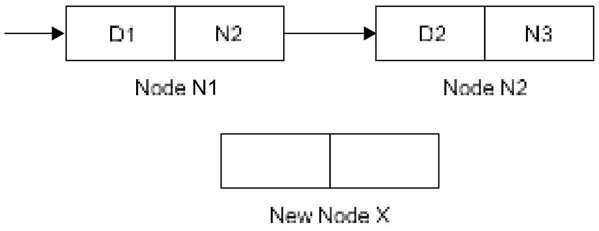

Fig 5.2 (a) *Creating a new element*

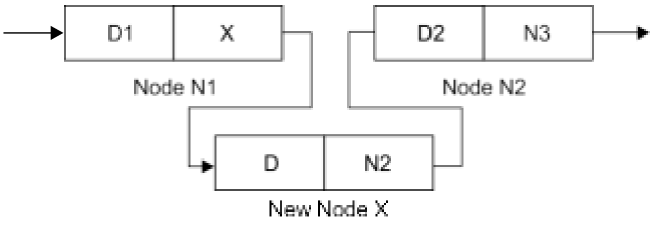

Fig 5.2 (b) *Inserting the newly created element*

Example 5.1 Write an algorithm to insert an element at the end of a linked list.

```
insert (value)
Step 1: Start
Step 2: Set PTR = addressof (New Node)
     //Allocate a new node and assign its address to the pointer PTR
Step 3: Set PTR->INFO = value;
     //Store the element value to be inserted in the INFO part of the new node
Step 4: If FIRST = NULL, then goto Step 5 else goto Step 7
     //Check whether the existing list is empty
Step 5: Set FIRST=PTR and LAST=PTR
     //Update the FIRST and LAST pointers
Step 6: Set PTR->NEXT = NULL and goto Step 8
Step 7: Set LAST->NEXT=PTR, PTR->NEXT=NULL and LAST=PTR
     //Link the newly created node at the end of the list
Step 8: Stop
```

2. Delete The delete operation removes an existing element from the linked list. The following tasks are performed while deleting an existing element:

'a* The location of the element is identified.

(b) The element value is retrieved. In some cases, the element value is simply ignored.

(c) The link pointer of the preceding node is reset.

Depending on the location from where the element is to be deleted, there are three scenarios possible, which are:

(a) Deleting an element from the beginning of the list.

(b) Deleting an element from the end of the list.

(c) Deleting an element somewhere from the middle of the list.

Deleting an element from the beginning or end of the list is easy as it only requires resetting the first and last pointers. However, if an element is to be deleted from within the list then a search operation is required to be performed for locating that element. Figures 5.3 (a) and (b) show the deletion of an element that is present between two existing elements of a linked list.

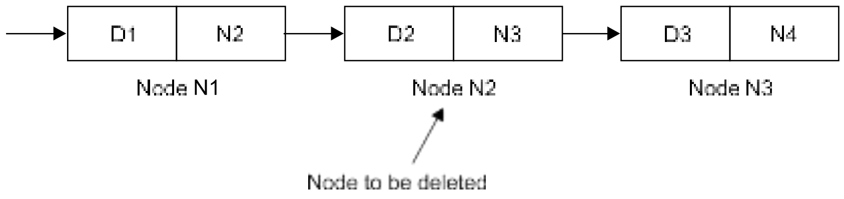

Fig 5.3 (a) *Identifying the node to be deleted*

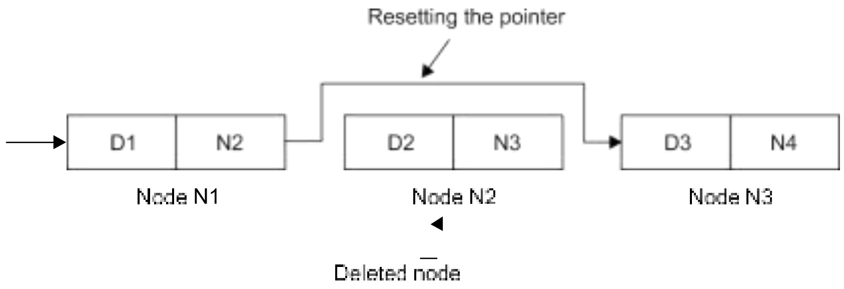

Fig 5.3 (b) *Deleting the node*

Example 5.2 Write an algorithm to delete a specific element from a linked list.

```
  delete (value)
  Step 1: Start
  Step 2: Set LOC = search (value)
      //Call the search module to search the location of the node to be deleted and assign it to LOC pointer
  Step 3: If LOC=NULL goto Step 4 else goto Step 5
  Step 4: Return ("Delete operation unsuccessful: Element not present") and Stop
  Step 5: If LOC=FIRST goto Step 6 else goto Step 10
      //Check if the element to be deleted is the first element in the list
  Step 6: If FIRST=LAST goto Step 7 else goto Step 8
      //Check if there is only one element in the list
  Step 7: Set FIRST=LAST=NULL and goto Step 9
  Step 8: Set FIRST=FIRST->NEXT
  Step 9: Return ("Delete operation successful") and Stop
  Step 10: Set TEMP=LOC-1
      //Assign the location of the node present before LOC to temporary pointer TEMP
  Step 11: Set TEMP->NEXT=LOC->NEXT
      //Link the TEMP node with the node being currently pointed by LOC
  Step 12: If LOC=LAST goto Step 13 else goto Step 14
      //Check if the element to be deleted is currently the last element in the list
  Step 13: Set LAST=TEMP
  Step 14: Return ("Delete operation successful")
  Step 15: Stop
```

3. Search The search operation helps to find an element in the linked list. The following tasks are performed while searching an element:

'a* Tra+erse the list sequentially starting from the first node.

(b) Return the location of the searched node as soon as a match is found.

'c* /eturn a search failure notification if the entire list is tra+ersed without any match.

The NEXT pointers help in traversing the linked list from start till end.

Example 5.3 Write an algorithm to search a specific element in the linked list.

```
  search (value)
  Step 1: Start
  Step 2: If FIRST=NULL goto Step 3 else goto Step 4
      //Check if the linked list is empty
  Step 3: Return ("Search unsuccessful: Element not present") and Stop
  Step 4: Set PTR=FIRST
  Step 5: Repeat Steps 6-8 until PTR!=LAST
      //Repeat Steps 6-8 until PTR is not equal to LAST
  Step 6: If PTR->INFO=value goto Step 7 else goto Step 8
  Step 7: Return ("Search successful", PTR) and Stop
  Step 8: Set PTR=PTR->NEXT
```

```
Step 9: If LAST->INFO=value goto Step 10 else goto Step 11
    //Check if the element to be searched is the last element in the list
Step 10: Return (“Search successful”, LAST) and Stop
Step 11: Return (“Search unsuccessful: Element not present”)
Step 12: Stop
```

4. Print The print operation prints or displays the linked list elements on the screen. To print theelements, the linked list is traversed from start till end using NEXT pointers.

Example 5.4 Write an algorithm to print all the linked

```
print ()
Step 1: Start
Step 2: If FIRST=NULL goto Step 3 else goto Step 4
    //Check if the linked list is empty
Step 3: Display (“Empty List”) and Stop
Step 4: If FIRST=LAST goto Step 5 else goto Step 6
    //Check if the list has only one element
Step 5: Display (FIRST->INFO) and Stop
Step 6: Set PTR=FIRST
Step 7: Repeat Steps 8-9 until PTR!=LAST
    //Repeat Steps 8-9 until PTR is not equal to LAST
Step 8: Display (PTR->INFO)
    //Displaying list elements
Step 9: Set PTR=PTR->NEXT
Step 10: Display (LAST->INFO)
    //Displaying last element
Step 11: Stop
```