# Unit 2 Linked List

**Structure:**

2.1 Introduction
Objectives
2.2 Linked List and its representation in memory
2.3 Traversing a Linked List
2.4 Searching a Linked List
2.5 Memory Allocation and Garbage Collection
2.6 Insertion into Linked list
Insertion Algorithm
2.7 Deletion from a Linked list
Deletion Algorithm
2.8 Types of Linked List
2.9 Summary
2.10 Terminal Questions
2.11 Answers

## 2.1 Introduction

In the previous unit, you learnt about the structuring of problem and how the task of formatting a solution to a problem is made simpler if the problem can be analyzed in terms of sub problems. We also discussed algorithm complexity and time–space tradeoff that may occur in choosing the algorithm and data structure for a given problem.

One way to store data is by means of array. In this unit we will discuss on another way of storing a list in memory is to have each element in the list contain a field, called link, which contains the address of the next element in the list. Thus successive elements in the list need not occupy adjacent space in memory. This type of data structure is called linked list. In this unit we are going to discuss the various operations that we can perform on linked list like insertion, deletion, searching and traversing. We also are going to discuss how the memory will be allocated while defining the linked data structure and the role of garbage collection to de allocate the memory after its use.

**Objectives:**

After studying this unit, you should be able to:

- discuss the representation of linked list in memory
- explain some of the operations of linked list
- discuss memory allocation and garbage collection
- describe the types of linked list

## 2.2 Linked List and its representation in memory

*Linked list* is a linear collection of data elements called nodes. Each node is divided into two parts: the first part is called data field where the data are stored and the second part is called the link field or next field, contains the address of the next node in the list. The figure 2.1 shows a node with data and link fields.

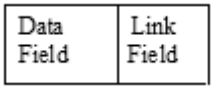



**Figure 2.1: Node of a linked list**

The figure 2.2 shows the linked list with 6 nodes. The arrow drawn represents the link between two nodes. The link field of last node contains a special value, called null pointer which indicated the end of the list. The linked list also contains a list pointer variable called START which contains the address of the first node. The list with no node is called null list or empty list and it is denoted by the null pointer in the variable START.

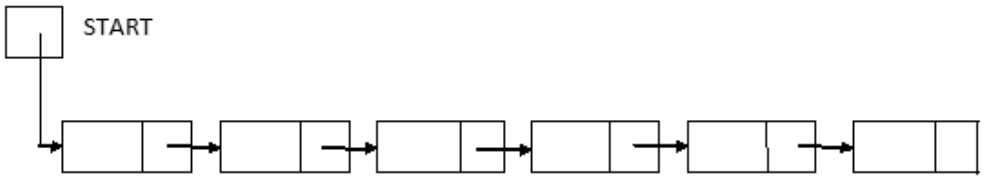



**Figure 2.2: Linked list with 6 Nodes**

If we represent linked lists in memory it requires two arrays one store the data part and one for the address part such that DATA [K] and LINK [K] respectively. It also requires a variable name START which contains the address of the first node and a next pointer sentinel denoted by NULL which indicates the end of the list. The figure 2.3 shows how the linked list is stored in memory. The nodes of a list need not occupy adjacent elements in the array DATA and LINK.

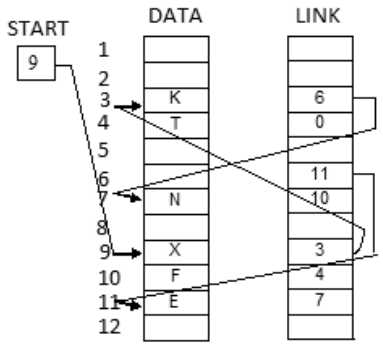



**Figure 2.3: Linked List Storage in Memory**

Generally the data field of a node may be a record with more than one data item. In such case, the data must be store in some type of record structure or in a collection of parallel arrays.

## 2.3 Traversing a Linked List

Traversing a linked list means processing each node of list exactly once. The linked list in memory is stored as linear array DATA and LINK with START pointing to the first element and NULL indicating the end of the list. Let us see the algorithm for traversing a linked list.

**Algorithm:** Traversing a linked list

1. Set P: = START. [ Initializes pointer PTR]
2. Repeat Step 3 and 4 while P ≠ NULL.
3. Apply PROCESS to DATA [P].
4. Set P := LINK [P] .[P points to the next node]
   [End of Step 2 loop]
5. Exit

In the algorithm P is the pointer variable which points to the node that is currently being processed and LINK [P] points to the next node to be processed.

P := LINK [P]

Thus the pointer moves to the next node in the list as shown in the figure 2.4

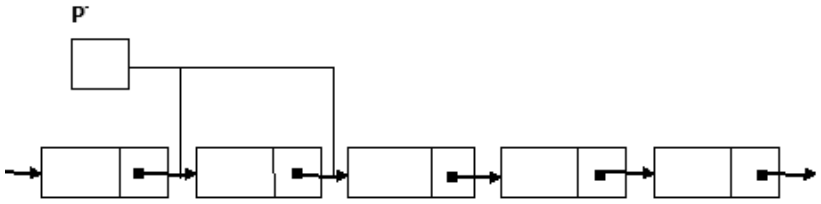



**Figure 2.4: P: = LINK [P]**

In the algorithm initially the pointer is assigned to the start variable P := START then it process the data at the first node of the list DATA [P]. Then the P is incremented to next node P := LINK [P] and then it process the data of second node. Then the P will again get incremented to next node and so on. This will be continued until the P = NULL which indicates the end of the list.

**Self Assessment Questions**

1. _____ is a linear collection of nodes.
2. _____ and _____ are the two fields of linked list.
3. Processing each node of linked list exactly once is called as _____.

## 2.4 Searching a Linked List

Searching a linked list is to find the location of an ITEM in the list. We have two searching algorithm first algorithm does not assume that the data of list are sorted, whereas the second algorithm assumes that the list is sorted.

**List is unsorted**

The data in the list are not sorted. The search is carried out by traversing through the list and comparing each data in the list with the ITEM to be searched. The following is the algorithm to search an unsorted list.

```
Algorithm: SEARCH (DATA, LINK, START, ITEM, LOC)
Search a unsorted list
        1. Set P := START
        2. Repeat Step 3 while P ≠ NULL:
        3.      If ITEM = DATA [P], then:
Set LOC: = P, and Exit.
                Else:
                    Set P := LINK [P]. [P points to the net node]
                [End of If structure]
             end of Step 2 loop]
        4. [Search is unsuccesful] Set LOC := NULL
        5. Exit
```

In the algorithm first the P is assigned to START variable and it compares the ITEM with the data of first node ITEM = DATA [P], if it matches then it set the location to be first node address LOC: = P, else the P will be incremented to next node P: = LINK [P]. Then it compares the ITEM with the second node data if it matches then it set the location to be second node address LOC := P, else the P will be incremented to next node P := LINK [P] and so on. This will be continued until P = NULL.

The complexity of this search algorithm for worst case it is ‘n’ and for average case it is ‘n/2’ same as linear search.

**List is sorted**

In this the list is sorted and the search is carried out by traversing through the list and comparing each data in the list with the ITEM to be searched. Except here when the ITEM exceeds the DATA [P] the search is stopped. The following is the algorithm to search an unsorted list.

**Algorithm:** SEARCH1 (DATA, LINK, START, ITEM, LOC)
Search a sorted list

1. Set P := START
2. Repeat Step 3 while P ≠ NULL:
3. If ITEM < DATA [P], then:
   Set P := LINK [P]. [P points to next node]
   Else if ITEM = DATA [P], then:
   Set LOC := P and Exit. [Search is successful]
   Else:
   Set LOC := NULL, and Exit.[ITEM exceeds DATA[P]]
   [End of If structure]
   [End of Step 2 loop]
4. Set LOC := NULL
5. Exit.

The complexity of this search algorithm for worst case it is 'n' and for average case it is 'n/2' same as linear search.

## 2.5 Memory Allocation and Garbage Collection

The maintenance of linked lists in memory assumes the possibility of inserting new nodes into the lists and hence requires some mechanism which provides unused memory space for the new nodes. Analogously, some mechanism is required whereby the memory space of deleted nodes becomes available for the future use.

Along with the linked list in memory, a special list is maintained which consists of unused memory cells. This list, which has its own pointer, is called the list of available space or the free storage list or the free pool.

Suppose our linked lists are implemented by parallel arrays as described in the preceding sections, and suppose insertions and deletions are to be performed on our linked lists. Then the unused memory cells in the arrays will also be linked together to form a linked list using AVAIL as its list pointer variable. Such a data structure will frequently be denoted by writing

LIST (DATA, LINK, START, AVAIL)

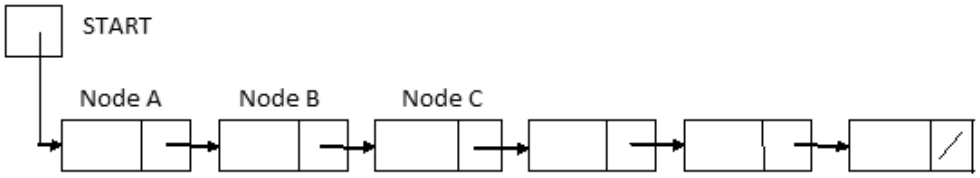
### Garbage collection

Suppose some memory space becomes reusable because a node is deleted from a list or an entire list is deleted from a program. Clearly, we want the space to be available for future use. One way to bring this about is to immediately reinsert the space into free- storage list. This is what we will do when we implement linked lists by means of linear arrays. However, this method may be too time- consuming for the operating system of a computer, which may choose an alternative method as follows.

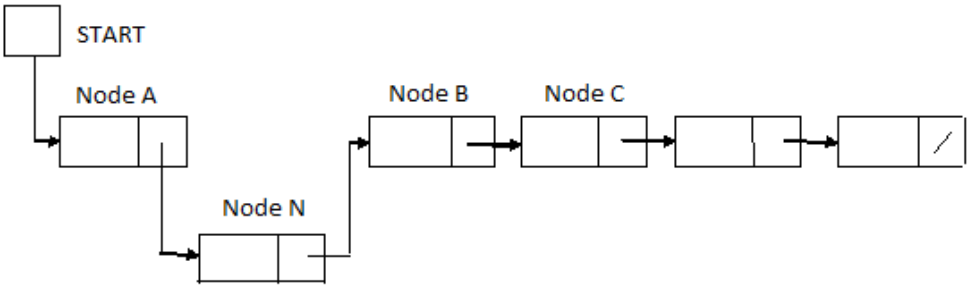
The operating system of a computer may periodically collect all the deleted space onto the free storage list. Any technique which does this collection is called garbage collection. It usually takes place in two steps. First the computer runs through all lists, tagging those cells which are currently in use, and then the computer runs through memory, collecting all untagged space onto the free- storage list. The garbage collection may take place when there is only some minimum amount of space or no space at all left in the free- storage list, or when the CPU is idle and has time to do the collection.

## 2.6 Insertion into a Linked List

Insertion is the process of adding a new node to the linked list. It requires a new node and executing two next pointer maneuvers. Suppose consider a new node N is to be inserted into list between node A and B. The figure 2.5(a) and (b) explains the insertion process.




**(a) Before Insertion**




**(b) After Insertion**

**Figure 2.5 (a) and (b): Insertion into Linked List**

The figure 2.5 does not take into account that the memory space for the new node N will come from the AVAIL list. So, for easier processing, the first node in the AVAIL will be used as the new node N. The exact diagram of insertion is shown in figure 2.6.

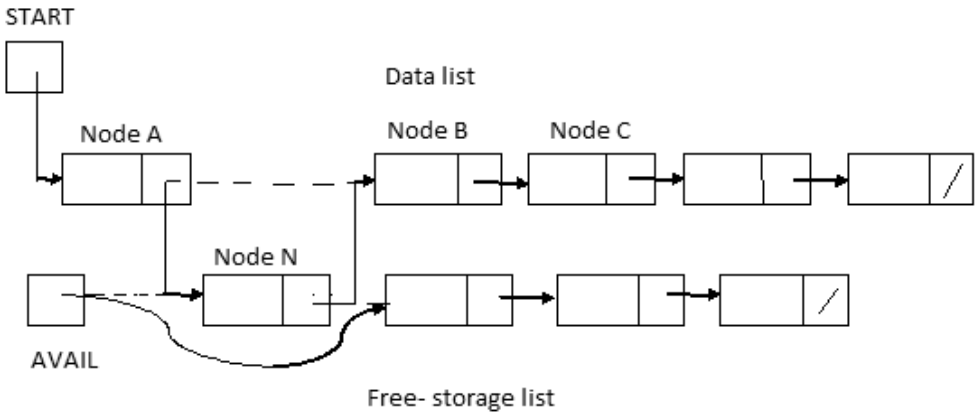



**Figure 2.6: Exact Insertion into Linked List**

The next pointer field of node A now points to the new node N, to which AVAIL previously pointed. AVAIL now points to the second node in the free-storage list, to which node N previously pointed. The next pointer field of node N now points to node B, to which node A previously pointed.

### 2.6.1 Insertion Algorithms

We have algorithms for various situation of inserting an ITEM into linked list. In this section we will see two of them. The first one is inserting a node at the beginning of a list and the second one is inserting after the node with a given location. All over the algorithm ITEM contains the new data to be added to the list.

All insertion algorithms we are going to see will use new node from AVAIL list, the following are the steps which is included in all the algorithms.

- Checking to see if space is available in the AVAIL list. If not then AVAIL = NULL, then the algorithm will print the message OVERFLOW.
- Removing the first node from the AVAIL list. Using the variable N to keep track of the location of the new node, this step can be implemented by the pair of assignments

  N := AVAIL, AVAIL := LINK [AVAIL]
- Copying new data into the new node.

  DATA [N] := ITEM

The following figure 2.7 shows the how the last two steps are implemented in free- storage list.

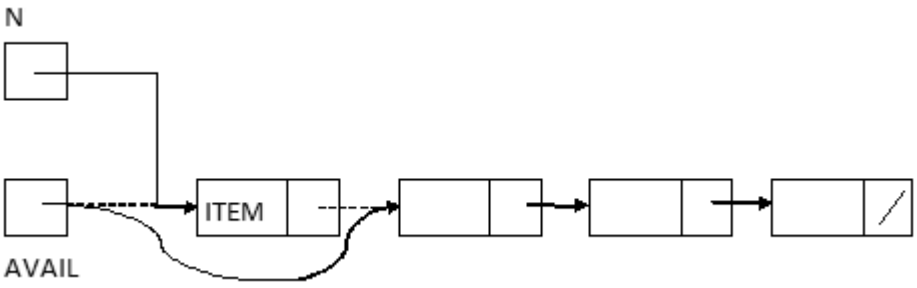



**Figure 2.7: Availing new node from AVAIL List**

**Inserting at the beginning of a list**

The simplest way of inserting a node is by inserting a node at the beginning of a list. So the algorithm is as follows.

**Algorithm:** INSERT (DATA, LINK, START, VAIL, ITEM)
Inserting at the beginning of list

1. [OVERFLOW?] if AVAIL = NULL, then: Write: OVERFLOW, and Exit.
2. [Remove first node from AVAIL list]
   Set N := AVAIL and AVAIL := LINK [AVAIL].
3. Set DATA [N] := ITEM. [Copies new data into new node]
4. Set LINK [N] := START.[New node points to the original first node]
5. Set START := N. [changes START so it points to the new node]
6. Exit.

First the OVERFLOW condition is checked and then the first node is removed from AVAIL list and marked as N. The data to be inserted is assigned to the new node data field, DATA [N] := ITEM. Now the pointers are changed by making the N nodes address field is pointed to first node of the list and the START is made to point the N node. Thus a node is inserted at the beginning of the list. The figure 2.8 shows the insertion at the beginning of a list.

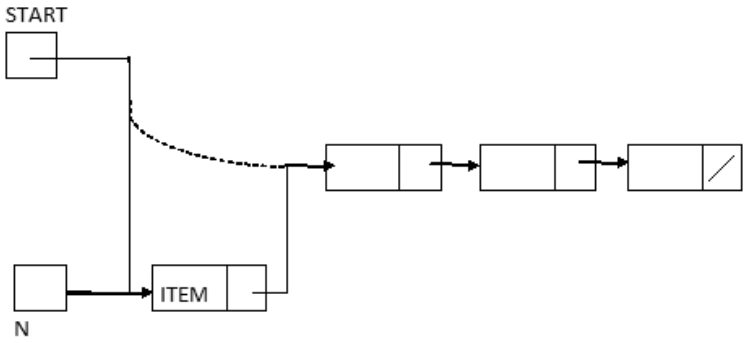



**Figure 2.8: Insertion at the Beginning of a List**

**Inserting after a given node**

In this method they will give the location LOC of a node or the location will be null, LOC = NULL. So, here if the location of some node is given we insert the ITEM next to that node and if location is null then the ITEM will be the first node. The following is the algorithm to insert after a given node.

**Algorithm:** INSERT1 (DATA, LINK, START, AVAIL, LOC, ITEM)
Inserting after a given node.

1. [OVERFLOW?] If AVAIL = NULL, then: Write: OVERFLOW, and Exit
2. [Remove first node from AVAIL list]
   Set N := AVAIL and AVAIL := LINK [AVAIL].
3. Set DATA [N] := ITEM. [Copies new data into new node]
4. If LOC = NULL, then: [Insert as first node]
   Set LINK [N] := START and START := N.
   Else: [Insert after node with location LOC]
   Set LINK [N] := LINK [LOC] and LINK [LOC] := N.
   [End of if structure.]
5. Exit.

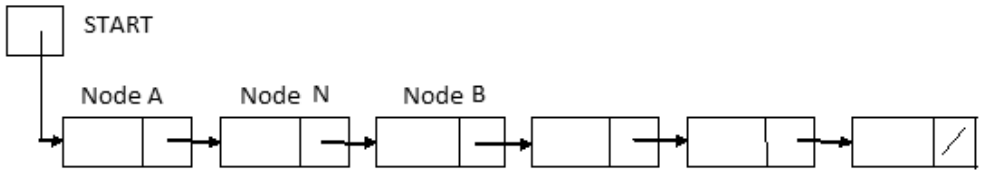
First the OVERFLOW condition is checked and then the first node is removed from AVAIL list and marked as N. The data to be inserted is assigned to the new node data field, DATA [N] := ITEM. If location is null, LOC = NULL then the N will be set as first node and the START variable will point this N node. If location is not null that means some other nodes location will be given as LOC, then the address of that LOC will point to the N node, LINK [LOC] := N and the address of new node will point to the address where LOC was previously pointing, LINK [N] := LINK [LOC].

**Self Assessment Questions**

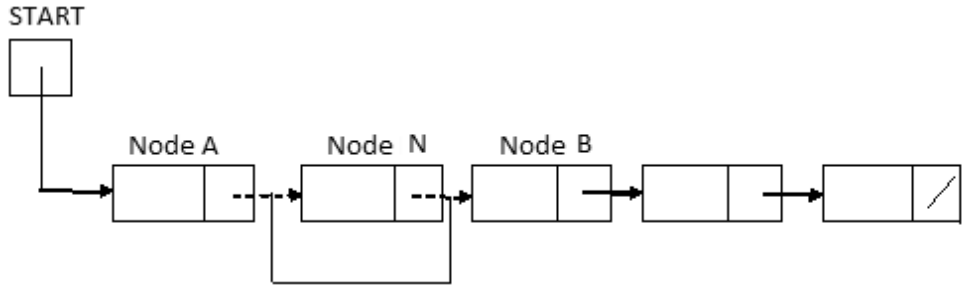
4. _____ is used to find the location of an item in a linked list.
5. _____ is used to store the unused memory cells.
6. _____ technique is used to collect all the free cells and store that in free pool.

## 2.7 Deletion from a Linked List

Deletion is the process of removing a node from the linked list. The deletion of a node is made just by a pointer change. Suppose a node has to be deleted between node A and B, the figure 2.9 (a) and (b) explains it.




**(a) Before Deletion**




**(b) After Insertion**

**Figure 2.9 (a) and (b): Deletion from Linked List**

The figure 2.9 does not take into account the fact that if a node is deleted from a list it will immediately return its memory space to the AVAIL list. So, for easier processing, it will return to the beginning of the AVAIL list. The exact diagram of deletion is shown in figure 2.10.

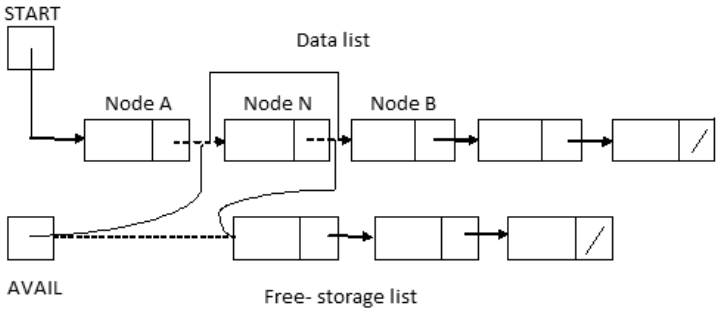



**Figure 2.10: Exact Deletion into Linked List**

- The next pointer field of node A now points to node B, where node N previously pointed.
- The next pointer field of N now points to the original first node in the free storage list, where AVAIL previously pointed.
- AVAIL now points to the deleted node N.

### 2.7.1 Deletion Algorithm

We have algorithms for deletion from linked list in various situations. In this section we will see two of them. The first one deletes the node following a given node and the second one deletes the node with a given ITEM of information.

All of our deletion algorithms will return the memory space of the deleted node N to the beginning of the AVAIL list. Accordingly, all of our algorithms will include the following pair of assignments, where LOC is the location of the deleted node N:

LINK [LOC] := AVAIL and AVAIL := LOC

Figure 2.11 shows the two operations.

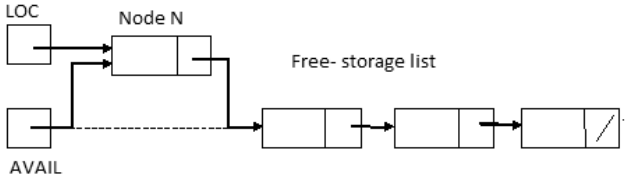



**Figure 2.11: Deleted Node added in the AVAIL List**

### Deleting the node following a given node

Let N be the node to be deleted. Suppose we are given the location LOC of node N in the list. Furthermore, suppose we are given the location LOC1 of the node preceding N or when N is the first node, we are given LOC1 = NULL. The following algorithm deletes N from the list.

Algorithm: DELETE (DATA, LINK, START, AVAIL, LOC, LOC1)

Deleting the node following a given node.

1. If LOC1 = NULL, then:
   Set START := LINK [START]. [Deletes first node]
   Else:
   Set LINK [LOC1] := LINK [LOC]. [Deletes node N]
   [End of If structure]
2. [Return deleted node to the AVAIL list]
   Set LINK [LOC] := AVAIL and AVAIL := LOC.
3. Exit.

When LOC1 = NULL then the N will be the first node and it is deleted just by assigning the START variable to the link of START variable, START := LINK [START]. The figure 2.12 explains this assignment.

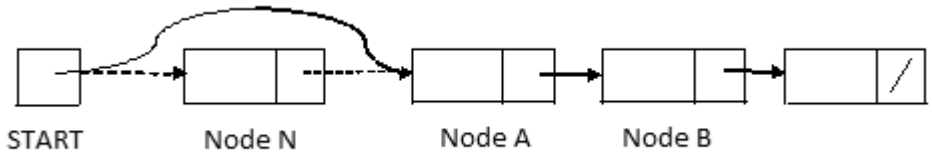



**Figure 2.12: Deletion of Node N from the List**

When LOC1 is not NULL then the deletion is done by assigning the link of LOC1 to link of LOC, LINK [LOC1] := LINK [LOC]. The figure 2.13 explains this assignment.

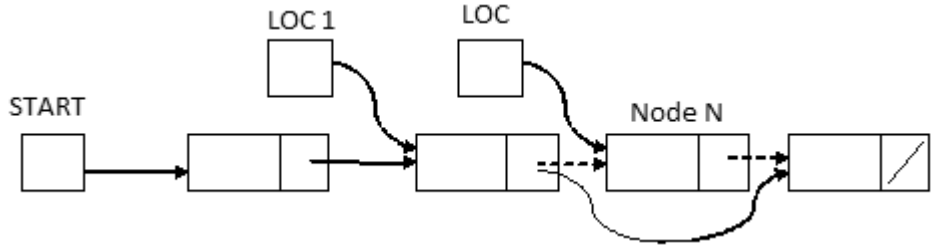



**Figure 2.13: Assigning link of LOC1 to LOC**

**Deleting the node with a given ITEM of information**

In this deletion method the ITEM information will be given and we have to delete the first node N which contains the ITEM. So, to delete the node N from the list we need to know the location of the node preceding N. Accordingly, first we give a procedure which finds the location LOC of the node N containing the ITEM and the location LOC1 of the node preceding

the node N. suppose the N is the first node then LOC1 = NULL and if the ITEM does not exist in the list then LOC = NULL.

In the procedure, first we traverse the list using a pointer variable P and comparing ITEM with DATA [P] at each node. While traversing, keep track of the preceding node by using a pointer variable P1 as shown in figure 2.14.

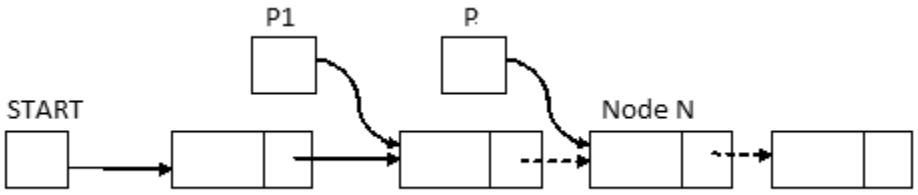



**Figure 2.14: Deletion of node for given item**

The traversing of the list continues until we get the ITEM, DATA [P] = ITEM or ITEM ≠ DATA [P]. Then the pointer variables P contains the location LOC of the node N and P1 contains the location LOC1 of the node preceding N. The following is the procedure to find the locations.

**Procedure:** FIND (DATA, LINK, START, ITEM, LOC, LOC1)

Find the location LOC and LOC1 of the node N and the node preceding N respectively.

1. [List empty?] if START = NULL, then:
   Set LOC := NULL and LOC1 := NULL, and Return.
   [End of If structure.]
2. [ITEM in first node?] If DATA [START] = ITEM, them:
   Set LOC := START and LOC1 := NULL, and Return.
   [End of If structure.]
3. Set P1 := START and P := LINK [START]. [Initializes pointers.]
4. Repeat Steps 5 and 6 while P ≠ NULL.
5. If DATA [P] = ITEM, then:
   Set LOC := P and LOC1 := P1, and Return.
   [End of If structure.]
6. Set P1 := P and P := LINK [P]. [Updates pointers]
   [End of Step 4 loop]
7. Set LOC := NULL. [Search Unsuccessful]
8. Return.

Now let us see the algorithm to delete the node N which contains the ITEM information. In this algorithm we just call the procedure to find the location of N and the node preceding N.

**Algorithm:** DELETE1 (DATA, LINK, START, AVAIL, ITEM)
Deleting the node with a given ITEM of information

1. [Use the Procedure to find LOC and LOC1]

Call FIND (DATA, LINK, START, ITEM, LOC, LOC1)

2. If LOC = NULL, then: Write: ITEM not in the list, and Exit.
3. [Deletion of node.]

If LOC1 = NULL, then:
    Set START := LINK [START]. [Deletes the first node.]
Else:
    Set LINK [LOC1] := LINK [LOC].
[End of If structure.]

4. [Return deleted node to the AVAIL list.]

Set LINK [LOC] := AVAIL and AVAIL := LOC.

5. Exit.

## 2.8 Types of Linked List

In this section we will discuss on the types of linked list they are:

- Doubly linked list
- Circular Linked list

### Doubly linked list

In some situation we need to traverse both forward and backward of a linked list. The linked list with this property needs two link field one to point the next node is called next link field and another to point the previous node is called previous link field. The linked list containing this type of nodes is called doubly linked list or two- way list. Here first nodes previous link field and the last nodes next link field are marked as null. The figure 2.15 shows the node structure.

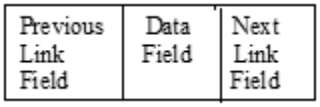



**Figure 2.15: Node Structure**

The figure 2.16 shows the structure of doubly linked list.

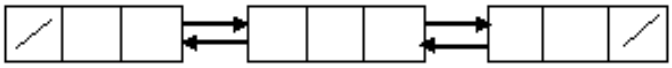


**Figure 2.16: Doubly Linked List**

The operations of doubly linked list are same as one- way list they are traversing, searching, insertion and deletion.

### Circularly linked list

Circular linked list is the one where the null pointer in the last node is replaced with the address of the first node so that it forms a circle. The advantage of this circular linked list is easy accessibility of node i.e. every node is accessible from a given node. The circular linked list can be implemented both in one- way list and two- way lists.

The figure 2.17 shows a singly circular list where the link of the last node is points to the first node.

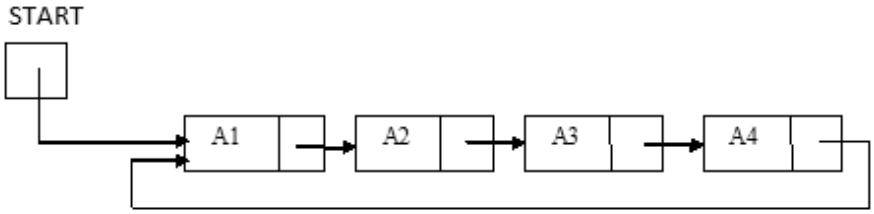



**Figure 2.17: A Singly Circular Linked List**

The figure 2.18 shows a doubly circular list where the last nodes next link points to the first node and first nodes previous link points to the last node and makes a circle.

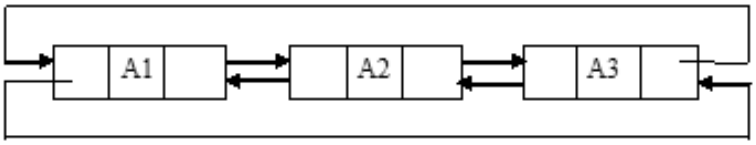



**Figure 2.18: A Doubly Circular Linked List**

### Self Assessment Questions

7. _____ and _____ are the types of linked list.
8. Doubly linked list are also called as _____.

## 2.9 Summary

This unit describes the one more way of storing the data in memory is linked list. It discusses on various operations of linked list like traversing, searching, inserting and deleting and also discuss on concept of memory allocation and garbage collection. Also in this unit we saw the two types of linked list- doubly and circular linked list.

## 2.10 Terminal Questions

1. Explain in detail the traversing and searching of linked list.
2. Define linked list and it types. How to represent linked list in memory?
3. Discuss the insertion operation of linked list in detail.
4. Explain the deletion operation of linked list in detail
5. Write a brief note on memory allocation and garbage collection.

## 2.11 Answers

### Self Assessment Questions

1. Linked list
2. Data and Link fields
3. Traversing
4. Searching
5. Free- storage list
6. Garbage Collection
7. Doubly and Circular linked list
8. Two- way list

### Terminal Questions

1. Traversing a linked list means processing each node of list exactly once. (Refer sections 2.3 and 2.4)
2. Linked list is a linear collection of data elements called nodes. Each node is divided into two parts data field and link filed. (Refer sections 2.1, 2.2 and 2.8)
3. Insertion is the process of adding a new node to the linked list. It requires a new node and executing two next pointer maneuvers. (Refer section 2.6)

4. Deletion is the process of removing a node from the linked list. The deletion of a node is made just by a pointer change. (Refer section 2.7)
5. Linked list has various operations like traversing, searching, insertion and deletion. (Refer sections 2.3, 2.4, 2.5 and 2.6)
6. Proper memory allocation and collecting the free space will improve the efficiency of your program. (Refer section 2.5)