**Compiled by: Prof. Shajil Kumar P A shajil.kumar@vsit.edu.in**

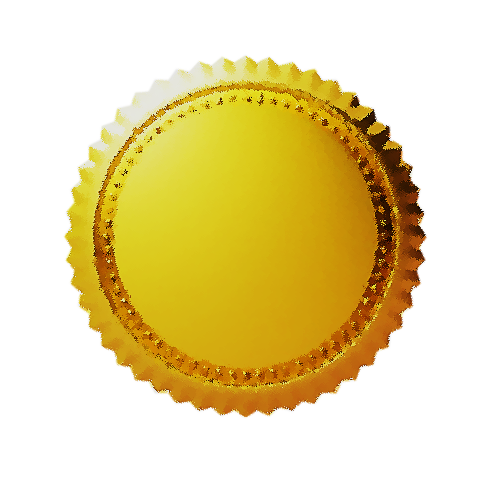


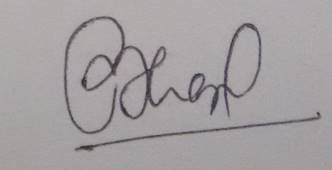
|  |  |
| --- | --- |
| VSIT logo NEW small | **Vidyalankar School of Information Technology Wadala (E), Mumbai**  [**www.vsit.edu.in**](http://www.vsit.edu.in) |

***MODULE-2: Linked List***



**Certificate**

This is to certify that the e-book titled “Data STRUCTURES” comprises all elementary learning tools for a better understating of the relevant concepts. This e-book is comprehensively compiled as per the predefined eight parameters and guidelines.



Signature Date: 31-07-2019

Mr. Shajil Kumar

Assistant Professor

Department of IT

* *DISCLAIMER: The information contained in this e-book is compiled and distributed for educational purposes only. This e-book has been designed to help learners understand relevant concepts with a more dynamic interface. The compiler of this e-book and Vidyalankar Institute of Technology give full and due credit to the authors of the contents, developers and all websites from wherever information has been sourced. We acknowledge our gratitude towards the websites YouTube, Wikipedia, and Google search engine*. *No commercial benefits are being drawn from this project.*

**Unit II**

* **Contents**
* **Linked List:** Linked List, One-way Linked List, Traversal of Linked List, Searching, Memory Allocation and De-allocation, Insertion in Linked List, Deletion from Linked List, Copying a List into Other List, Merging Two Linked Lists, Splitting a List into Two Lists, Reversing One way linked List, Circular Linked List, Applications of Circular Linked List, Two way Linked List, Traversing a Two way Linked List, Searching in a Two way linked List, Insertion of an element in Two way Linked List, Deleting a node from Two way Linked List, Header Linked List, Applications of the Linked list, Representation of Polynomials, Storage of Sparse Arrays, Implementing other Data Structures.
* **Recommended Books**
* **Text Books And Reference Books**
* A Simplified Approach to Data Structures , Lalit Goyal, Vishal Goyal, Pawan Kumar , SPD , 1st , 2014
* An Introduction to Data Structure with Applications , Jean – Paul Tremblay and Paul Sorenson , Tata MacGraw Hill , 2nd , 2007
* Data Structure and Algorithm , Maria Rukadikar , SPD , 1st , 2017
* Schaum’s Outlines Data structure , Seymour Lipschutz , Tata McGraw Hill , 2nd , 2005
* Data structure – A Pseudocode Approach with C , AM Tanenbaum, Y Langsam and MJ Augustein , Prentice Hall India , 2nd , 2006
* Data structure and Algorithm Analysis in C , Weiss, Mark Allen , Addison Wesley , 1st , 2006

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit V | Prerequisites | | | Linkage | |
| Sem I | Sem. II | Sem. III | Sem IV | Sem. VI |
| Hashing Techniques, Graphs | Imperative Programming | Object Oriented Programming |  | Java | Project |

**LINKED LIST**

A linked list can be defined as the linear collection of elements where each element is stored in a node and the linear order between elements is given by the means of pointers instead of sequential memory locations. In linked list or one way list, each node is divided into two parts. The first part of the node contain the element itself and the second part which is termed as next field or pointer field contain the address of the next node in the list. The reason for maintaining the address of the next node of the linked list in the second part of the node is that linked lists do not store the lists elements at contiguous memory locations. The structure of the node is

Info Next

While maintaining the list into the memory a pointer variable Begin is used which contains the address of the first node in the linked list. The Next field of the last node of the list contains the Null value which indicates the end of the list

**Singular or One-way Linked List**

One-way linked list is also known as singular linked list. In this each node has atleast two parts. The first part is known as **Info** part which hold the element and the second part is known as **Next** part which hold the address of next node. The address of first node is stored in as special variable known as **Begin** and the **Next** part of last node contains Null indicating the end of the linked list.

104 4001

102 2001

103 3001

101 1001

**Operations Performed On One-way Linked List**

There are various operations that can be applied on the linked list,

* Traversing a linked list
* Searching an element in the linked list
* Inserting an element in the linked list
* Deleting an element in the linked list
* Copying a linked list
* Merging two linked list
* Splitting a linked list into two or more list

**Source :** https://www.youtube.com/watch?v=hAzTqg8jmWA

**Traversal Of Linked List**

Traversing a linked list refers to visiting each node of the list in order to process the elements stored in nodes

**Algorithm: Traversing a one-way linked list to print the elements of the linked list**

* Step 1: If Begin = Null Then

Print “Linked List is Empty”

Exit

[End if]

* Step 2: Set Pointer = Begin
* Step 3: Repeat While Pointer != Null

Print Pointer ---> Info

Assign Pointer = Pointer ---> Next

[End Loop]

* Step 4: Exit

**Algorithm: Traversing a one-way linked list to find the largest element in the list.**

* Step 1: If Begin = Null Then

Print “Linked List is Empty”

Exit

[End if]

* Step 2:Set Max =Begin ---> Info and Pointer = Begin--> Next
* Step 3: Repeat While Pointer != Null

If Pointer ---> Info > Max Then

Set Max = Pointer ---> Info

[End If]

Set Pointer = Pointer ---> Next

[End Loop]

* Step 4: Print : Max
* Step 5: Exit

**Searching**

Only Linear Search is possible because elements of linked do not occupy contiguous memory locations

**Algorithm :To find the location of a desired element ‘Data’ in an unsorted linked list.**

* Step 1: If Begin = Null Then

Print “Linked List is Empty”

Exit

[End if]

* Step 2: Set Pointer = Begin
* Step 3: Repeat While Pointer != Null

If Pointer ---> Info = Data Then

Print : “Element is found at address”: Pointer

Exit

Else

Set Pointer = Pointer ---> Next

[End If]

[End Loop]

* Step 4: Print : “Element not found in the linked list”
* Step 5: Exit

**Algorithm: To find the location of a desired element ‘Data’ in an sorted (Ascending Order) linked list.**

* Step 1: If Begin = Null Then

Print “Linked List is Empty”

Exit

[End if]

* Step 2: Set Pointer = Begin
* Step 3: Repeat While Pointer != Null

If Pointer ---> Info = Data Then

Print : “Desired Element is found at address”: Pointer

Exit

Else If Pointer ---> Info > Data Then

Print : “Desired Element is not present in the linked list”

Exit

Else

Set Pointer = Pointer ---> Next

[End If]

[End Loop]

* Step 4: Print : “Desired Element is not present in the linked list”
* Step 5: Exit

**Memory Allocation and Deallocation**

To insert an element in the linked list the first requirement is to get a free Node. To delete a Node from a linked list it is desirable to return the memory taken by deleted Node for its reusability. The task of obtaining an empty Node for insertion and returning free Node after deletion is accomplished by maintaining a separate list of free Nodes that begin with pointer **Free** which points to the first available free Node. Whenever a new Node is to be inserted in to the linked list then free memory is checked for the availability. If the Node is available then the node is added to the linked list and pointer **Free** will point to the next available node. If there is no free node then it is indicated by **Free = Null** and this condition is known as **Overflow**. Whenever a node is deleted from the linked list it is inserted as the first node of the free storage. The pointer **Free** will point to the recently added node

**Insertion in Linked List**

While inserting an element into the linked list the first requirement is to get a free node. Once the space for the new node becomes available the element to be inserted is placed into info part of the node and pointers are set to add the new node at the desired insertion position of the linked list. The new node can be inserted

* + At the beginning of the list
  + At the end of the list
  + At a particular position
  + In the sorted linked list

**Insertion at the Beginning of the Linked List**

While inserting a new node at the beginning of linked list we need to check whether the list is initially empty or not. If the list is empty then we will store the **Null** value in the **Next** part of the new node and the address of the new node will be stored in the variable **Begin**. But if the list is not empty then the address stored in the variable **Begin** will be stored in the **Next** part of the new node and the address of the new node will be stored in variable **Begin** as shown below

8

7

12 Null

3

5

New Node

**Algorithm for inserting a new element at the beginning of the linked list**

* Step 1: If Free = Null Then

Print : “Overflow: No free space available for insertion”

Exit

[End If]

* Step 2:Allocate space to node New

(Set New = Free And Free= Free ---> Next)

* Step 3:Set New ---> Info = Data
* Step 4:Set New ---> Next = Begin And Begin = New
* Step 5:Exit

**Insertion At The End Of The Linked List**

Before inserting at the end of the linked list first we will check whether the linked list is initially empty or not. If the list is empty then the new node is the first node in the list. If the list is not empty then we will traverse the entire linked list from beginning to end until we reach the last node of the linked list. On reaching the last node we will store the address of the new node into the Next part of the last node of the linked list and the Next part of the new node will be set to Null

14 Null

12 Null

7

8

5

**Algorithm For Inserting a node at the end of the linked list**

* Step 1: If Free = Null Then

Print: “Overflow: No free space available for insertion”

Exit

[End If]

* Step 2:Allocate space to node New

(Set New = Free And Free= Free ---> Next)

* Step 3:Set New ---> Info = Data, New ---> Next =Null
* Step 4: If Begin = Null Then

Begin = New

Exit

[End If]

* Step 5:Set Pointer = Begin
* Step 6:Repeat While Pointer ---> Next != Null

Set Pointer = Pointer ---> Next

[End Loop]

* + Step 7: Set Pointer ---> Next = New
* Step 8:Exit

**Inserting At A Particular Position Of The Linked List**

For inserting a node into a linked list after a particular node we will locate the position of the node after which we want to insert a new node. If the desired node is found the new node will be inserted after it. If the desired node is not found we will reach at the end of the list and then the new node will not be inserted.

12 Null

7

8

5

14 Null

**Algorithm Inserting an element at the particular position**

* Step 1: If Free = Null Then

Print : “Overflow: No free space available for insertion”

Exit

[End If]

* Step 2: Set Pointer = Begin
* Step 3: Repeat While Pointer != Null AND Pointer--> Info !=Data

Set Pointer = Pointer --> Next

[End Loop]

* Step 4: If Pointer = Null Then

Print: “The node containing element Data is not present, insertion not possible.”

Else

(Allocate space to node New)

Set New = Free And Free= Free ---> Next, New ---> Info = Item

Set New ---> Next =Pointer --> Next

Set Pointer --> Next = New

[End If]

* Step 5:Exit

**Insertion In A Sorted Linked List**

To insert an element into a sorted linked list we need to find the position of the node after which new node has to be inserted. We start comparing the new element **Item** with the first element of the linked list. If the **Item** is smaller than the first element then the new node will be inserted in the first position of the linked list. Otherwise the list will be traversed to find the position of the node after which the new node is to be inserted and if we do not find any node whose element is larger than the element then this new node will be inserted at the end of the linked list. After finding the exact position for the new node we insert the new node at that position

22 Null

17

10

5

14 Null

**Algorithm To Insert a new element in a sorted linked list**

* Step 1:If Begin = Null

(Allocate space to node New)

Set New = Free And Free= Free ---> Next

Set New ---> Info = Item

Set New---> Next = Begin And Begin = New

[End If]

* Step 2: If Item < Begin --> Info Then

(Allocate space to node New)

Set New = Free And Free= Free ---> Next

Set New ---> Info = Item

Set New---> Next = Begin And Begin = New

Exit

[End If]

* Step 3 : Set Pointer1 = Begin And Pointer2 = Begin --> Next
* Step 4 : Repeat While Pointer2 != Null And Item > Pointer2 --> Info

Set Pointer1 = Pointer2 And Pointer2 = Pointer2 --> Next

[End Loop]

* Step 5: If Free = Null Then

Print : “No free space available for insertion”

Exit

[End If]

* Step 6: (Allocate space to node New)

Set New = Free And Free= Free ---> Next

* Step 7: Set New ---> Info = Item
* Step 8: If Pointer2 = Null Then

Set Pointer1 --> Next = New And New --> Next = Null

Else

Set New ---> Next =Pointer1 --> Next

Set Pointer1 --> Next = New

[End If]

* Step 9 : Exit

**Deletion from Linked List**

The deletion operation can also be performed at various positions in the linked list.

* + Deleting a node at the beginning of the linked list
  + Deleting a node at the end of the linked list
  + Deleting a particular node of the linked list

If any three cases, after deletion of a node, the memory space occupied by the deleted node will be returned to the free storage list for its reusability.

**Deleting A Node At The Beginning Of The Linked List**

Deleting a node at the beginning of the list is very simple operation which can be accomplished by changing the list pointer variable **Begin**. Now **Begin** will point to the second node in the linked list. The space occupied by the deleted node is returned to the free storage list

22 Null

17

10

5

**Algorithm : Deletion of the first node of the linked list.**

* Step 1: If Begin = Null Then

Print : “Linked List is already empty”

Exit

[End If]

* Step 2: Set Item =Begin --> Info And Pos=Begin
* Step 3: Set Begin = Begin --> Next
* Step 4: Set Pos --> Next = Free And Free = Pos
* Step 5: Exit

**Deleting A Node From The End Of Linked List**

For deleting the last node from the given linked list it is necessary to traverse the entire linked list and find the address of the preceding node of the last node i.e. address of the second last node. After finding the address of the second last node we will store the address stored in the next part of the last node into the next part of the second last node i.e. **Null** will be stored in the next part of the second last node

22 Null

17

10

5

**Algorithm : Deletion the last node of the linked list.**

* Step 1: If Begin = Null Then

Print : “Linked List is empty”

Exit

[End If]

* Step 2:If Begin --> Next = Null Then

Set Data = Begin --> Info

Deallocate memory held by Begin

(Begin --> Next = Free And Free = Begin)

Set Begin = Null

Exit

[End If]

* Step 3: Set Pointer1 = Begin And Pointer2 = Begin --> Next
* Step 4: Repeat While Pointer 2--> Next != Null

Set Pointer1 = Pointer2 And Pointer2 = Pointer2 --> Next

[End Loop]

* Step 5: Set Pointer1 --> Next = Pointer2 --> Next
* Step 6: Set Data = Pointer2 --> Info
* Step 7: Deallocate memory held by Pointer2

(Pointer2 --> Next = Free And Free = Pointer2)

* Step 8: Exit

**Deleting a Particular node from the Linked List**

For deleting a node from the linked list the first task is to find the address of the preceding node of the node to be deleted. To accomplish this task traverse the linked list from the beginning and compare the information stored with the given element. Two pointer variables will be used while traversing the list for locating the address of the node to be deleted and the address of its preceding node

22 Null

17

10

5

**Algorithm: Deleting a particular node containing the given element ‘Item’ from the linked list.**

* Step 1: If Begin = Null Then

Print : “Linked List is empty”

Exit

[End If]

* Step 2: If Begin --> Info = Item Then

Set Pos = Begin

Set Begin = Begin --> Next

Pos --> Next = Free And Free = Pos

Exit

[End If]

* Step 3: Set Pointer1 = Begin And Pointer2 = Begin --> Next
* Step 4: Repeat While Pointer2 != Null AND Pointer --> Info != Item

Set Pointer1 = Pointer2 And Pointer2 = Pointer2 --> Next

[End Loop]

* Step 5: If Pointer2 = Null Then

Print : “Node containing element Item not found”

Exit

Else

Set Pointer1 --> Next = Pointer2 --> Next

[End If]

* Step 6: Deallocate memory held by Pointer2

(Set Pointer2 --> Next = Free And Free = Pointer2)

* Step 7: Exit

**Copying a Linked List into Other Linked List**

For copying the given linked into another linked list we use two pointers **Begin1** and **Begin2.** Initially we store **Null** in the list variable Begin2 and start the pointer Begin1 with list one. Then we will traverse the entire source list from beginning to end by copying the content at each node to the new target list.

12 Null

7

8

5

5

12 Null

7

8

**Algorithm : Copying the elements of a linked list to another linked list.**

* Step 1: If Begin1 = Null Then

Print : “Linked List is empty”

Exit

[End If]

* Step 2: Begin2 = Null
* Step 3: If Free = Null Then

Print: “Free space is not available”

Exit

Else

Set New = Free and Free = Free --> Next

[End If]

* Step 4:Set New --> Info = Begin1 --> Info and New --> Next = Null
* Step 5: Begin2 = New
* Step 6: Set Pointer1 = Begin1 --> Next And Pointer2 = Begin2
* Step 7: Repeat While Pointer1 ! = Null And Free != Null

a. Set New = Free

b. Set New --> Info = Pointer1 --> Info And New -->Next = Null

c. Set Pointer2 🡪 Next = New

d. Set Pointer1 = Pointer1 -- > Next And Pointer2 = New

[End Loop]

* Step 8: If Pointer1 = Null Then

Print : “List Copied Successfully”

Exit

Else

Print : “No Enough space available to perform Copy Operations”:

[End If]

* Step 9: Exit

**Merging Two Linked List**

There are number of applications where there is a need to merge two or more lists into a single list. Merging operation refers to putting elements of two or more lists into one list. The list may be either sorted or unsorted

32 Null

20

15

12

10

8

3

20 Null

10

3

32 Null

15

12

8

**Algorithm: Merging two sorted linked list into a single sorted linked list**

* Step 1: If Begin1 = Null OR Begin2 = Null

Print “One of the given linked list is empty”

Exit

[End If]

* Step 2: If Free = Null Then

Print “No Free Space is not available”

Exit

Else

Set New = Free And Free = Free --> Next

[End If]

* Step 3: Set Begin = Null
* Step 4: If Begin1 --> Info >= Begin2 --> Info Then

Set New --> Info = Begin2 --> Info And New --> Next = Null

Set Pointer1 = Begin1 And Pointer2 = Begin2--> Next

Else

Set New --> Info = Begin1 --> Info And New --> Next = Null

Set Pointer1 = Begin1 --> Next And Pointer2 = Begin2

[End If]

* Step 5: Set Begin = New And Pointer = New
* Step 6: Repeat Steps 7 and 8 While Pointer1 != Null And Pointer2 != Null
* Step 7: If Free = Null

Print “No Free space available”

Exit

Else

Set New = Free And Free = Free --> Next

[End If]

* Step 8: If Pointer1 -->Info >= Pointer2 --> Info Then

Set New --> Info = Pointer2 --> Info

Set New --> Next = Null

Set Pointer --> Next = New

Set Pointer = New And Pointer2 = Pointer2 --> Next

Else

Set New --> Info = Pointer1 --> Info

Set New --> Next = Null

Set Pointer --> Next = New

Set Pointer = New And Pointer1 = Pointer1 --> Next

[End If]

[End Loop]

* Step 9: If Pointer1 = Null And Free != Null

Repeat While Pointer2 != Null

a. Set New = Free And Free = Free --> Next

b. Set New --> Info = Pointer2 --> Info And New --> Next = Null

c. Set Pointer --> Next = New

d. Set Pointer = New And Pointer2 = Pointer2--> Next

[End Loop]

Else

Repeat While Pointer1!= Null

1. Set New = Free And Free = Free --> Next
2. Set New --> Info = Pointer1 --> Info And New --> Next = Null
3. Set Pointer --> Next = New

d. Set Pointer = New And Pointer1 = Pointer1 --> Next

[End Loop]

[End If]

* Step 10: If Pointer1 = Null And Pointer2 = Null Then

Print “The given Linked List merged successfully”

Else

Print “No enough Space”

[End If]

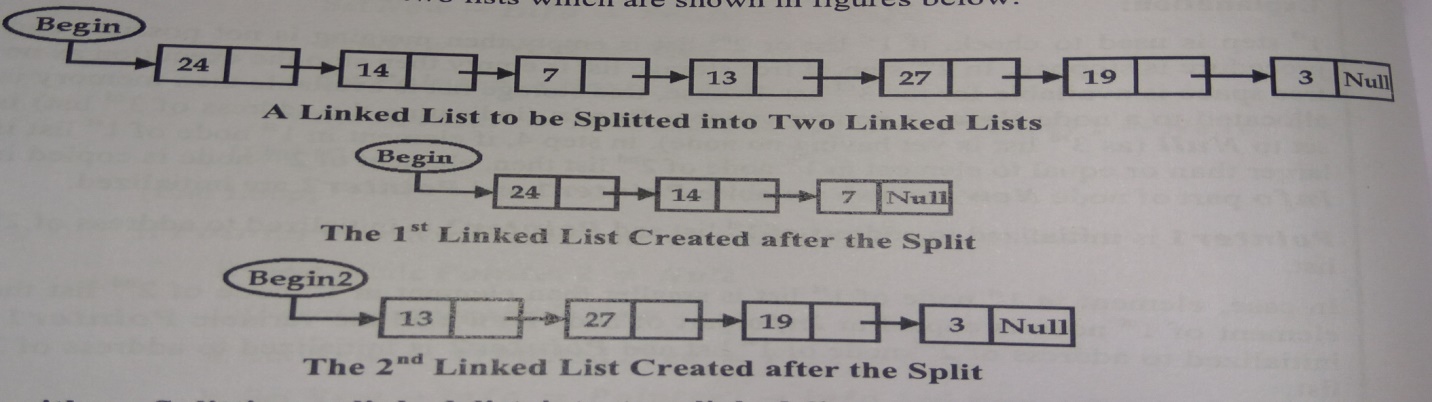
* Step 11: Exit

**Source :** https://www.youtube.com/watch?v=f3MYNbrqOys

**Splitting a Linked List into Two Linked List**

Suppose we have a linked list which we want to split into two linked lists of almost equal size.

To accomplish this task first of all we will traverse the list to count the total number of nodes in. it and then we will find the address of the (n/2)th and (n/2+1)th node. After finding those addresses we will store **Null** value in the next part of the (n/2)th node and the address of the (n/2+1)th node will be stored in the new list pointer variable **Begin2.** Now our list is splitted into two lists of sizes n/2 and (n-n/2) with the list pointers **Begin** and **Begin2** respectively



**Algorithm: Splitting a linked list into two lists with lists with list pointer variables ‘Begin’ and ‘Begin2 respectively.**

* Start 1: If Begin = Null

Print “Splitting cannot be performed on empty list”

Exit

[End If]

* Step 2: Set Pointer = Begin And Count =0
* Step 3: Repeat Steps 4 and 5 While Pointer != Null
* Step 4: Set Count = Count +1
* Step 5: Set Pointer = Pointer --> Next

[End Loop]

* Step 6: Set Mid = Integer(Count/2)
* Step 7: Set Begin2 = Null And Pointer = Begin And i=1
* Step 8: Repeat Step 9 While i< Mid
* Step 9: Set Pointer = Pointer --> Next

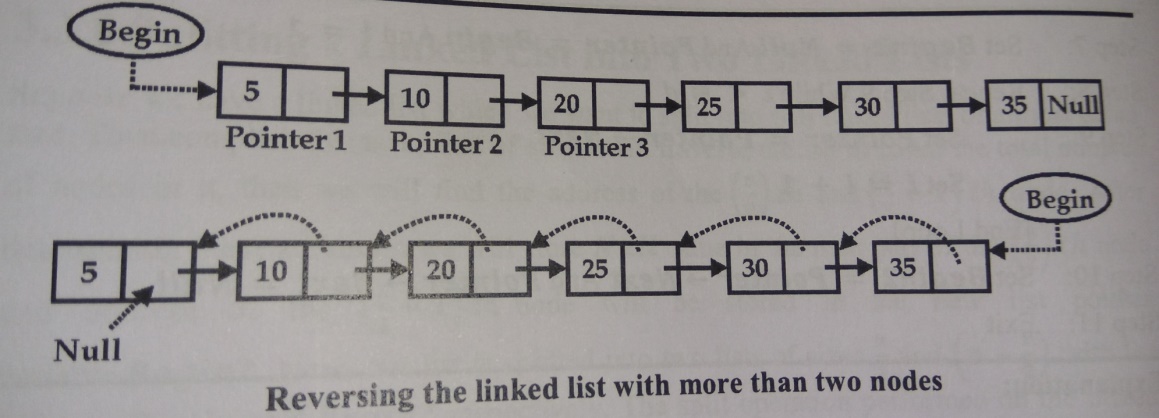
Set i=i+1

[End Loop]

* Step 10: Set Begin2 = Pointer --> Next And Pointer --> Next = Null
* Step 11: Exit

**Reversing a One–Way Linked List**

To reverse a linked list we need to use three pointer variables. One pointer variable is used to store the address of current node, second pointer variable is used to store the address of the next node and the third pointer variable is used to store the address of next to next of current node



**Algorithm: Reversing the one-way Linked List.**

* Step 1: If Begin = Null

Print : “No Node is present in the linked list ”

Exit

[End If]

* Step 2: If Begin --> Next = Null Then

Print : “ Linked List is having only one node”

Exit

[End If]

* Step 3: If Begin --> Next != Null Then

Set Pointer1 = Begin And Set Pointer2 = Begin --> Next

Set Pointer3 = Pointer2 --> Next

[End If]

* Step 4: If Pointer3 = Null Then

Set Pointer2 --> Next = Pointer1 And Set Pointer1 --> Next = Null And Set Begin = Pointer2

Exit

[End If]

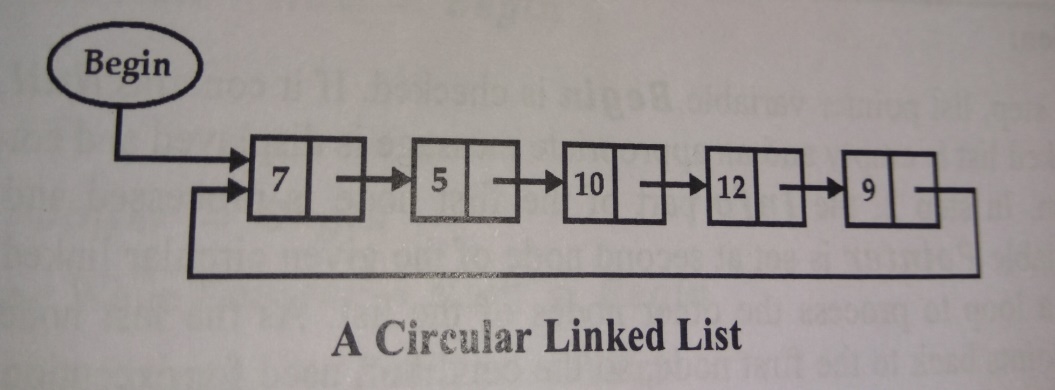
* Step 5: Set Pointer1 --> Next = Null
* Step 6 : Repeat Step 7 to 10 while Pointer3 --> Next != Null
* Step 7: Set Pointer2 --> Next = Pointer1 And Set Pointer1 = Pointer2
* Step 8: Set Pointer2 = Pointer3
* Step 9: Set Pointer3 = Pointer3 --> Next

[End Loop]

* Step 10: Set Pointer2 --> Next = Pointer1
* Step11 : Set Pointer3 --> Next = Pointer2
* Step12: Set Begin = Pointer3
* Step 13: Exit

**Circular Linked List**

A circular linked list is a list in which last node points back to the first node instead of containing the **Null** pointer in the next part of the last node.



All the operations which can be performed on ordinary singular linked list can easily be performed on circular linked list with the following changes

* + In case of 1-way singular linked list the next part of the last node will contain Null address but in case of circular linked list the next part of the last node consist of address of the first node i.e. Begin. Thus for reaching at the end of the circular linked list we will compare the address of the first node i.e. Begin with the address stored in Next part of each node. If both the addresses come out to be same then we have reached at the end of the circular list
  + When a new node is to be inserted at the end of the circular linked list its Next part will contain the address of the first node instead of Null as in the case of singular linked list

**Source:** https://www.youtube.com/watch?v=iit8UiNkKPE

**Traversal in Circular Linked List**

**Algorithm: Traverse a circular linked list with pointer variable ‘Begin’**

* Step 1: If Begin = Null then

Print: “Circular Linked List is empty”

Exit

[End If]

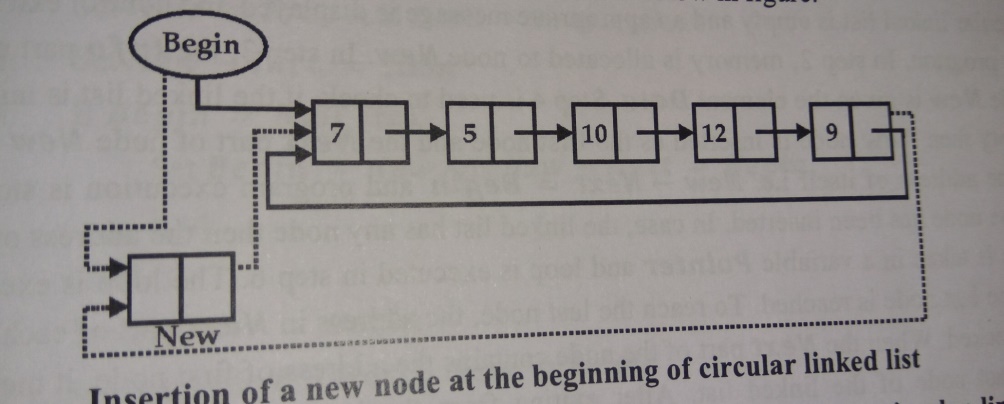
* Step 2: Process Begin --> Info
* Step 3: Pointer = Begin --> Next
* Step 4: Repeat Step 5 and 6 While Pointer != Begin
* Step 5: Process Pointer--> Info
* Step 6: Set Pointer = Pointer--> Next

[End Loop]

* Step 7 :Exit

**Insertion At The Beginning Of Circular Linked List**

In the insertion at the beginning the **New** node is inserted as the first node and the **Next** part of the last node is changed to point to newly inserted node



**Algorithm :Insertion of an element ‘Data’ at the beginning of the circular linked list**

* Step 1: If Free = Null then

Print: “No Free Space Available”

Exit

[End If]

* Step 2: Set New = Free And Free = Free --> Next
* Step 3: Set New --> Info = Data
* Step 4: If Begin = Null Then

Set Begin = New And New --> Next = Begin

Exit

[End If]

* Step 5: Pointer = Begin
* Step 6: Repeat While Pointer --> Next != Begin

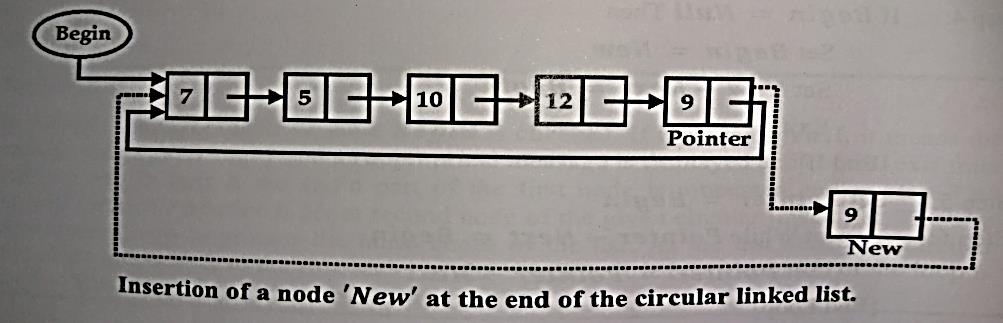
Set Pointer = Pointer --> Next

[End Loop]

* Step 7: Set New --> Next = Begin
* Step 8: Set Begin = New And Set Pointer --> Next= Begin
* Step 10: Exit

**Insertion At The End Of Circular Linked List**

The process of inserting an element at the end of the circular linked list is bit easier. Only the address stored in the **Next** part of the last node and next part of the **New** node needs to be changed



**Algorithm :Insertion of an element ‘Item’ at the end of the circular linked list**

* Step 1: If Free = Null then

Print: “No Free Space Available”

Exit

[End If]

* Step 2: (Allocate memory to node New)

Set New = Free And Free = Free --> Next

* Step 3: New --> Info = Item
* Step 4: If Begin = Null Then

Set Begin = New And New --> Next = Begin

Exit

[End If]

* Step 5: Pointer = Begin
* Step 6: Repeat While Pointer --> Next != Begin

Set Pointer = Pointer --> Next

[End Loop]

* Step 7: Set Pointer --> Next= New And New --> Next = Begin
* Step 8: Exit

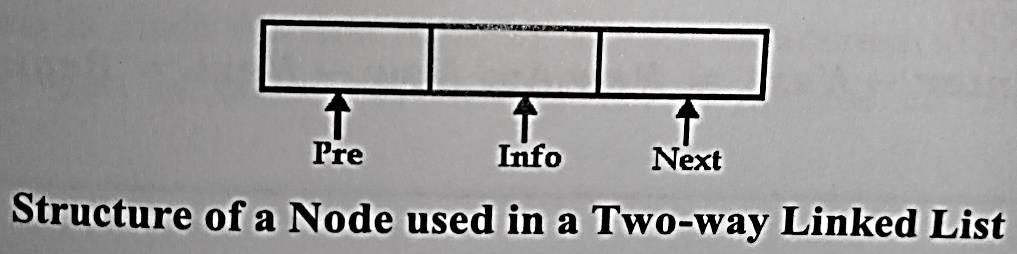
**Application of Circular linked list**

* A circular linked list can be used for implementing a timesharing problem of the operating system.
* In timesharing environment, the operating system must maintain a list of executing processes and must alternately allow each process to use a slice of CPU time, one process at a time.
* The operating system will pick a process; let it use a small amount of CPU time and then move on to the next process
* For this application there should be no **Null** pointer unless there is absolutely no process requesting CPU time.

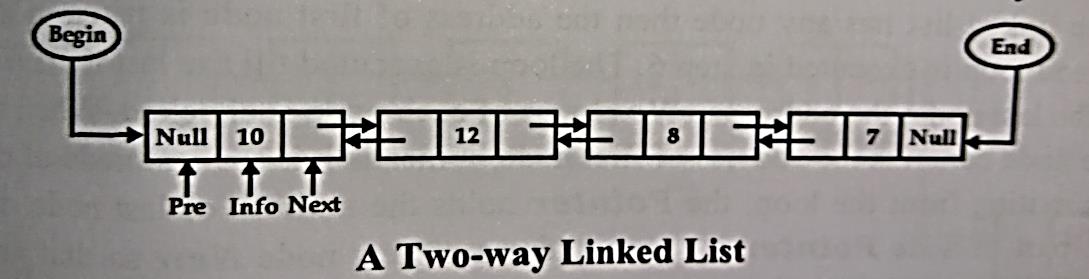
**Two-Way Linked List (Doubly Linked List)**

In a 1-way linked list we traverse the list only in one direction i.e. from beginning to end. But in certain applications it is required to traverse the list in both directions i.e. in forward direction (from beginning to end) and in backward direction (from end to beginning). This can be accomplished with the help of two-way linked list or doubly linked list. In two-way linked list each node is divided into three parts **Pre, Info, Next** where

* + **Prev** contains the address of the preceding node
  + **Info** contains the element
  + **Next** contains the address of the next node



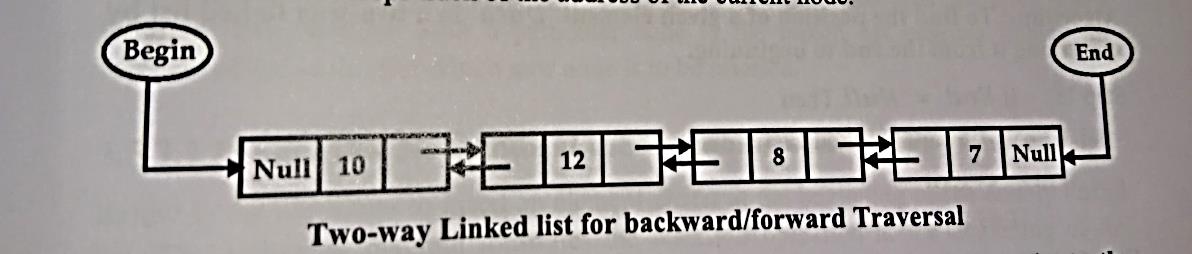
In 2-way linked list two list pointers are used **Begin** and **End** which contains the address of the first and the last node respectively.The Pre part of the first node of a 2-way linked list will contain Null as there is no node preceding the first node and the Next part of the last node of a 2-way linked list will contain Null as there is no node following the last node. Various operations that can be performed on a 2-way linked list are Traversing, Searching, Insertion and Deletion



**Source**: https://www.youtube.com/watch?v=JdQeNxWCguQ

**Traversing a Two-way Linked List**

A 2-way linked list can be traversed in both directions i.e. in forward as well as in backward direction. If a 2-way linked list is traversed in the forward direction the pointer variable will be assigned with the address stored in the **Begin** pointer variable and move forward till we reach at a node whose **Next** part contains **Null**. While traversing the list in backward direction the pointer variable will be assigned with the address stored in the **End** pointer variable and move backward till we reach at a node whose **Pre** contains **Null**. The variable **Pointer** keeps track of the address of the current node



**Algorithm: Traverse a two-way linked list starting from the end of the list to the beginning.**

* Step 1: If End = Null then

Print: “Linked List Empty”

Exit

[End If]

* Step 2: Set Pointer = End
* Step 3: Repeat While Pointer != Null

Process Pointer --> Info

Set Pointer = Pointer --> Pre

[End Loop]

* Step 4: Exit

**Searching in a Two-way Linked List**

In a 2-way linked list to find the location of the given item we can traverse the list starting either from the beginning or from end. While traversing we compare the element stored in each node with the desired item. If the desired item is found then further traversing is stopped and the address of the node containing the desired element is returned

**Algorithm : To find the position of a given element ‘Data’ in a two-way linked list by traversing it from the end to beginning**

* Step 1: If End = Null then

Print: “Linked List Empty”

Exit

[End If]

* Step 2: Set Pointer = End
* Step 3: Repeat While Pointer != Null

If Pointer --> Info = Data Then

Print : “Element Data is found at address ”: Pointer

Exit

Else

Set Pointer = Pointer --> Pre

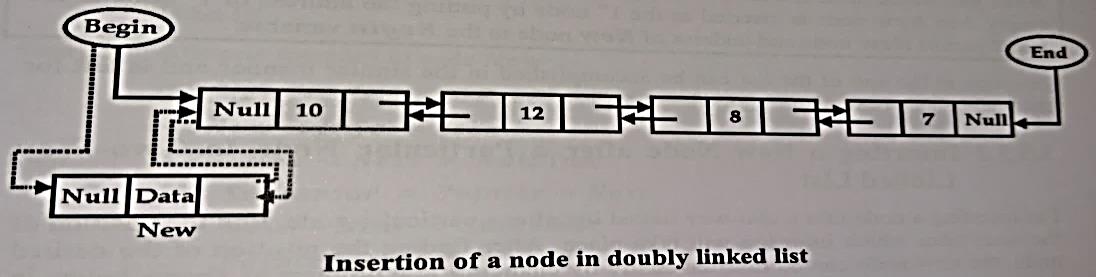
[End If]

[End Loop]

* Step 4: Print: “Element Data is not found in the linked list”
* Step 5: Exit

**Insertion Of An Element In A Two-Way Linked List**

Insertions can take place at various positions into a 2-way linked list. Insertion at the beginning or at the end of the linked list is a simple operation as it can be accomplished by changing a few pointers, but insertion after a particular node in the linked list requires finding the location of the node after which new node is to be inserted. The following diagram illustrates the insertion operation at the beginning of the linked list.



**Insertion A New Node At The Beginning Of Two-Way Linked List**

**Algorithm: To insert a new node at the beginning of a two-way linked list.**

* Step 1: If Free = Null Then

Print: “Free space not available”

Exit

[End If]

* Step 2: Allocate memory to node New

(Set New = Free And Free = Free --> Next)

* Step 3:Set New --> Pre = Null And New --> Info = Data
* Step 4: If Begin = Null Then

Set New --> Next = Null And End = New

Else

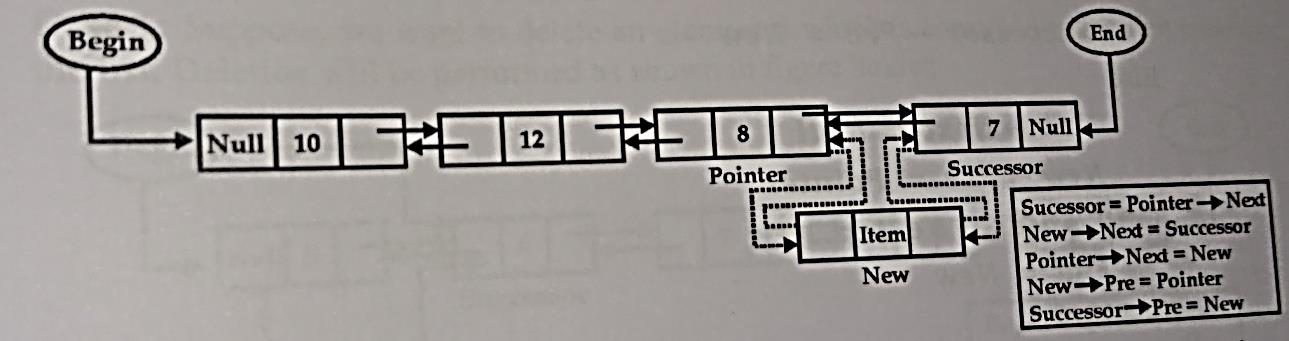
Set New --> Next = Begin And Begin --> Pre = New

[End If]

* Step 5: Set Begin = New
* Step:6 Exit

**Insertion A New Node After A Particular Node In Two-Way Linked List**

For inserting a node into a 2-way linked list after a particular node finds the position of the node after which insertion will take place. After finding the position of the desired node, the new node can be inserted easily by changing few pointers as shown below



**Algorithm: To insert a new node after a given element in a two-way linked list.**

* Step1 : If Free = Null Then

Print: “Free space not available”

Exit

[End If]

* Step 2: If Begin = Null

Print: “List is empty, No insertion will take place”

Exit

[End If]

* Step 3: Set Pointer = Begin
* Step 4: Repeat Pointer --> Next != Null And Pointer --> Info != Data

Pointer = Pointer --> Next

[End Loop]

* Step 5: If Pointer --> Next = Null And Pointer --> Info != Data

Print : “Element Data is not present”

Exit

[End If]

* Step 6: Allocate memory to node New

(Set New = Free And Free = Free --> Next)

Set New --> Info =Item

* Step 7: If Pointer --> Next != Null Then

Successor = Pointer --> Next

New --> Next = Successor

Pointer --> Next = New

New --> Pre = Pointer

Successor --> Pre = New

Else

New --> Next = Null

New --> Pre = Pointer

Pointer --> Next = New

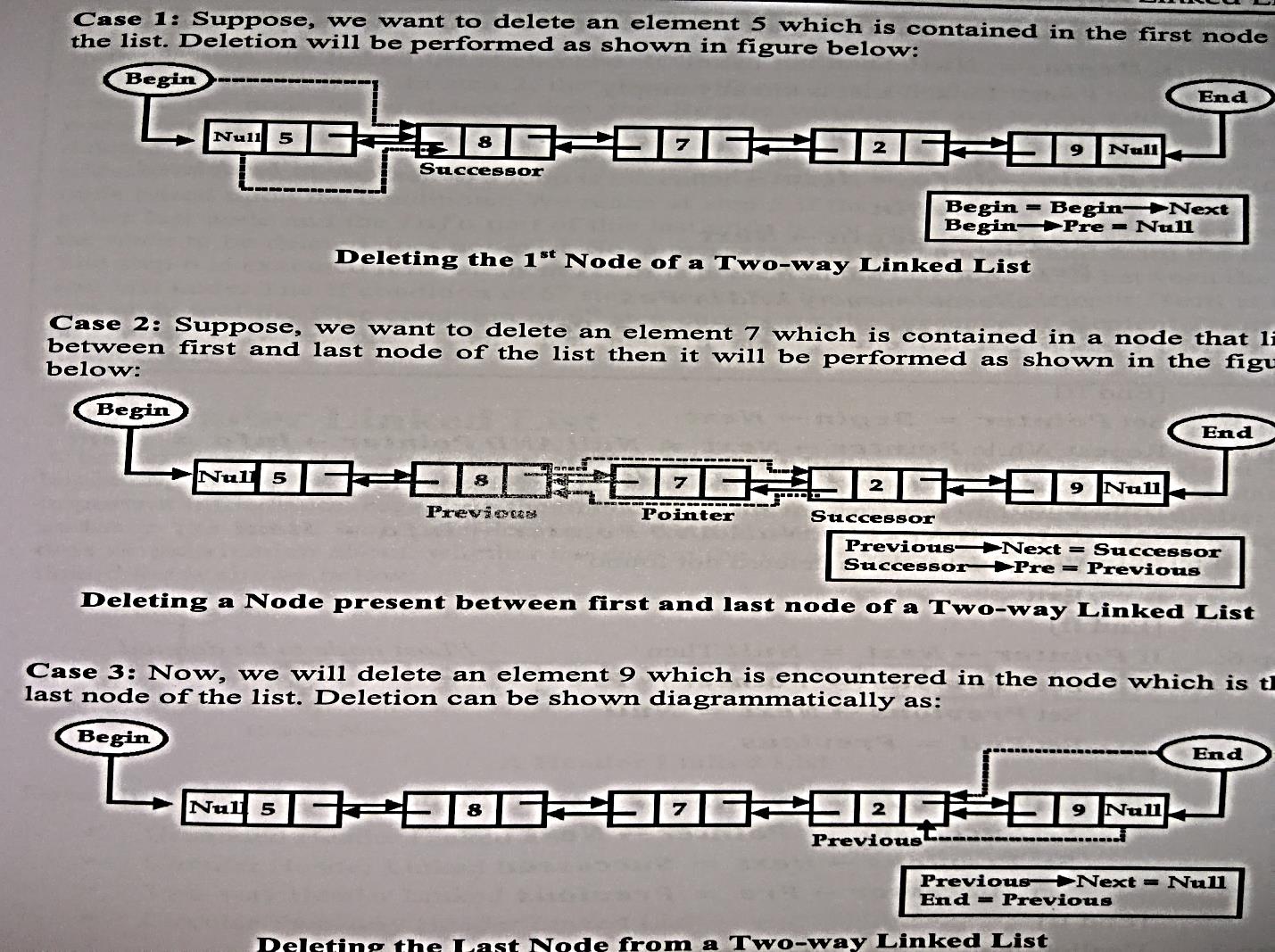
End = New

[End If]

* Step 8:Exit

**Deleting A Node With Given Item From 2-Way Linked List**

For deleting a particular node we will traverse the list either in forward or backward direction to locate the node containing the element to be deleted. While traversing the list if the node containing the item to be deleted is found then it can be removed from the linked list by changing a few pointers. If the desired node is not found and we reach at the end of the list then an appropriate message is displayed



**Algorithm: Delete a node containing an element ‘Item’ from a two-way linked list**

* Step 1: If Begin = Null

Print: “Linked List is already empty”.

Exit

[End If]

* Step 2: If Begin --> Info = Item Then

Pos = Begin

Begin =Begin --> Next

Begin --> Pre = Null

//Deallocate Memory held by Pos

Pos --> Next = Free , Free = Pos

Exit

[End If]

* Step 3: Set Pointer = Begin --> Next
* Step 4: Repeat While Pointer --> Next != Null And Pointer --> Info != Item

Set Pointer = Pointer --> Next

[End Loop]

* Step 5: If Pointer --> Next = Null And Pointer --> Info != Item

Print: “Item to be deleted not Found”

Exit

[End If]

* Step 6:If Pointer --> Next = Null Then

Set Previous = Pointer --> Pre And Previous --> Next = Null

Set End = Previous

Else

Set Previous = Pointer --> Pre And Successor = Pointer --> Next

Set Previous --> Next = Successor

Set Successor --> Pre = Previous

[End If]

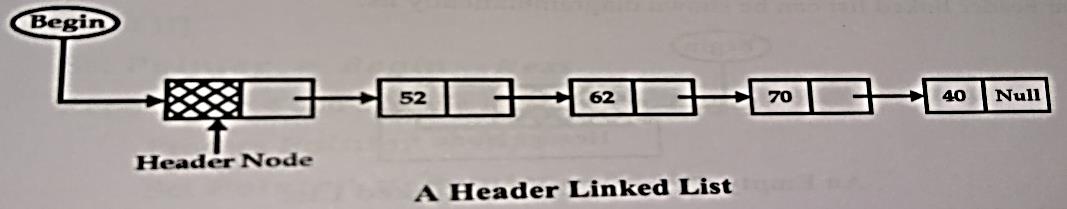
* Step 7: Deallocate memory help by Pointer

Pointer --> Next = Free , Free = Pointer

* Step 8: Exit

**Header Linked List**

A header linked list is a special kind of linked list which contains a special node at the beginning of the list. This is special node is known as head node. The head node contains information regarding the linked list. This information may be **total number of nodes** in the linked list, some description for the user like **creation date, modification date**, whether the data in the list is **sorted or unsorted**.



**Categories Of Header Linked List**

* **Grounded Header Linked List** – It is a list in which last node of the list contains the **Null** in its **Next** pointer field. If grounded header linked list is empty then the Null value will be stored in the Next pointer field of the head node
* **Circular Header Linked List** – It is a list in which the last node of the list points back to the header node i.e. **Next** pointer field of the last node contains the address of the header node. If circular header linked list is empty then address of the head node is stored in the Next pointer field of the head node itself
* **Two-way Header Linked List** – It is a list in which the **Pre** part of the header node contains the address of the first node and the **Next** part contains the address of the last node
* **Two-way Circular Header Linked List** – It is a list in which the **Pre** part of the header node contains the address of the first node and the **Pre** part of the first node contains the address of the header node. The **Next** part of the header node contains the address of the last node and the **Next** part of the last node contains the address of the header node

Source: https://www.youtube.com/watch?v=\_i0spiTbCZQ

**Algorithm : Traverse a circular header linked list.**

* Step 1: If Begin --> Next =Begin Then

Print : “Circular Header Linked List is empty”

Exit

[End If]

* Step 2: Set Pointer = Begin --> Next
* Step 3: Repeat Step 4 and 5 While Pointer != Begin

Process Pointer --> Info

* Step 4: Set Pointer = Pointer --> Next

[End Loop]

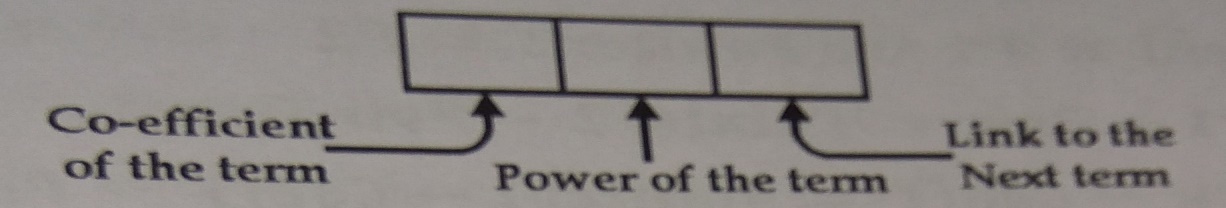
* Step 5: Exit

Linked List can be used in wide range of applications. Some of the applications are

* + To represent polynomials
  + To represent sparse matrices
  + To implement other data structures like Tree, Graph, Stack, Queue etc.

**Representation Of Polynomials**

Polynomials are frequently used in mathematical as well as scientific applications. General purpose languages do not have any built in data structure for storage and manipulation of polynomials and hence linked list can be used to represent the polynomials. Each node of the linked list will have 3 parts where first part of the node will contain the co-efficient of the variable and the second part of the node will contain the power and the third part will contain the address of the next node of the linked list. The structure of the node will be as shown below



Consider two polynomials P1 and P2 in which we want to subtract P2 from P1

P1: 3x4 – 8x3 + 6x + 9

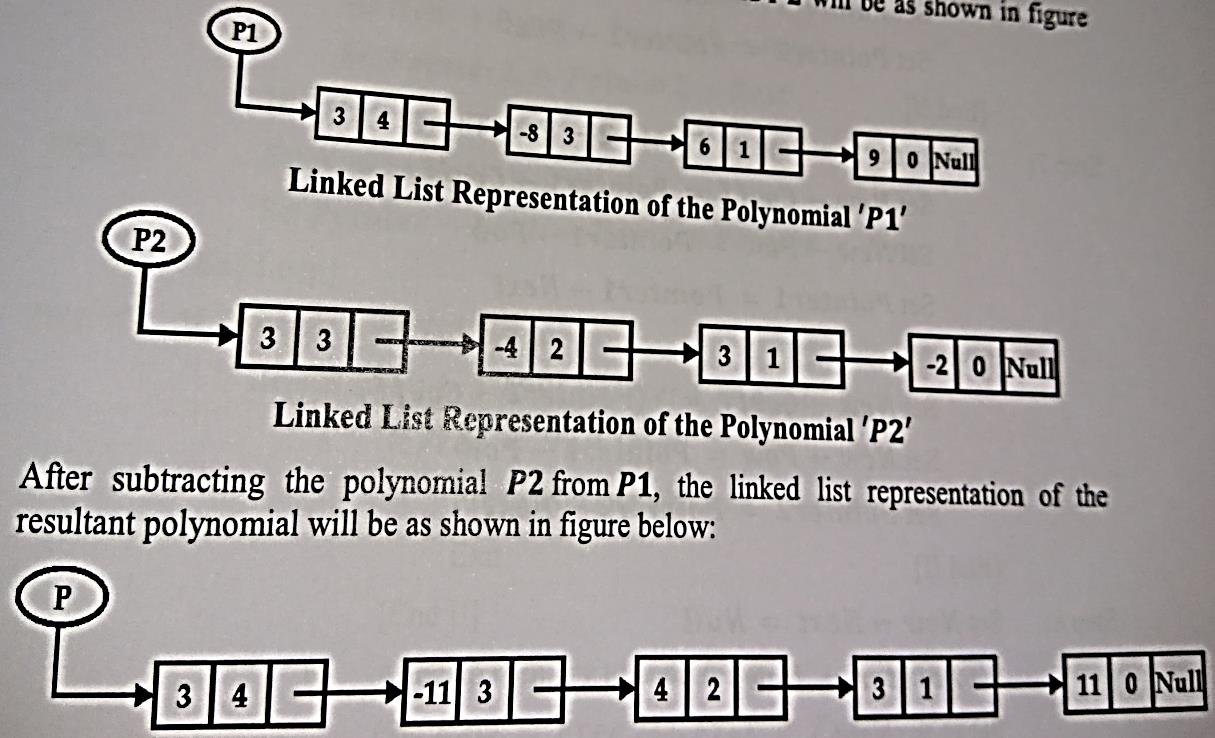
P2: 3x3 – 4x2 + 3x – 2

Their subtraction takes place as follows:

P1: 3x4 – 8x3 + 6x + 9

P2: 3x3 – 4x2 + 3x – 2

Subtraction (P1 – P2) is 3x4 – 11x3 + 4x2 + 3x + 11



**Algorithm: To subtract a given polynomial ‘P2’ from another polynomial ‘P1’**

* Step 1: If P1 = Null or P2 = Null Then

Print : “One or Both the polynomial are Null”

Exit

[End If]

* Step 2: If Free = Null Then

Print : “No free space available”

Exit

[End If]

* Step 3: Set P= Null
* Step 4: Allocate memory to node New

(Set New = Free And Free = Free --> Next)

* Step 5: Set Pointer1 = P1 And Pointer2 = P2
* Step 6: If Pointer1 --> Pow = Pointer2 --> Pow Then

Set New --> Coeff = Pointer1--> Coeff – Pointer2--> Coeff

Set New --> Pow = Pointer1--> Pow

Set Pointer1 = Pointer1 --> Next

Set Pointer2 = Pointer2 --> Next

[End If]

* Step 7: If Pointer1 --> Pow > Pointer2 --> Pow Then

Set New --> Coeff = Pointer1--> Coeff

Set New --> Pow = Pointer1--> Pow

Set Pointer1 = Pointer1 --> Next

Else

Set New --> Coeff = – (Pointer2--> Coeff)

Set New --> Pow = Pointer2--> Pow

Set Pointer2 = Pointer2 --> Next

[End If]

* Step 8: Set New --> Next = Null And Set Pointer = New And Set P = New
* Step 9: Repeat Steps 10 to 13 While Pointer1 != Null And Pointer2 != Null
* Step 10: If Free = Null Then

Print : “No free space available”

Exit

[End If]

* Step 11: Allocate memory to node New

(Set New = Free And Free = Free --> Next)

* Step 12: If Pointer1 --> Pow = Pointer2 --> Pow Then

Set New --> Coeff = Pointer1--> Coeff – Pointer2--> Coeff

Set New --> Pow = Pointer1--> Pow

Set Pointer1 = Pointer1 --> Next

Set Pointer2 = Pointer2 --> Next

Else If Pointer1 --> Pow > Pointer2 --> Pow Then

Set New --> Coeff = Pointer1--> Coeff

Set New --> Pow = Pointer1--> Pow

Set Pointer1 = Pointer1 --> Next

Else

Set New --> Coeff = – (Pointer2--> Coeff)

Set New --> Pow = Pointer2--> Pow

Set Pointer2 = Pointer2 --> Next

[End If]

* Step 13: Set New --> Next = Null And Set Pointer --> Next = New And Pointer = New

[End Loop]

* Step 14: If Pointer1 = Null Then

Repeat Step a to g While Pointer2 != Null

* + - 1. If Free = Null Then

Print : “Not Enough Space”

Exit

[End If]

b. Allocate memory to node New

(Set New = Free And Free = Free --> Next)

c. Set New --> Coeff = – (Pointer2--> Coeff)

d. Set New --> Pow = Pointer2--> Pow

e. Set Pointer2 = Pointer2 --> Next

f. Set New --> Next = Null

g. Set Pointer --> Next = New And Pointer = New

[End Loop]

Else

Repeat Step a to g While Pointer1 != Null

* + - 1. If Free = Null Then

Print : “Not Enough Space”

Exit

[End If]

b. Allocate memory to node New

. (Set New = Free And Free = Free --> Next)

c. Set New --> Coeff = Pointer1--> Coeff

d. Set New --> Pow = Pointer1--> Pow

e. Set Pointer1 = Pointer1 --> Next

f.Set New --> Next = Null

g. Set Pointer --> Next = New And Pointer = New

[End Loop]

[End If]

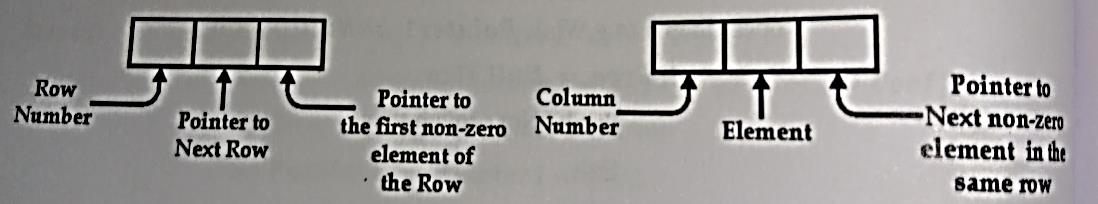
Step 15: Exit

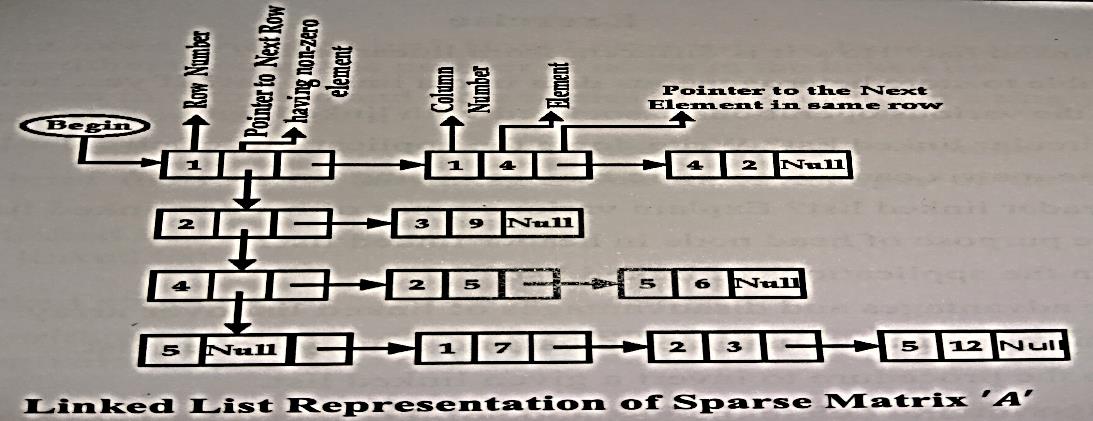
**Storage Of Sparse Array**

Sparse array is an array in which most of its elements are zero. The main problem in the array representation of sparse array is that it requires a lot of data movement while insertion and deletion of elements. This data movement can be avoided if linked list representation is used to store the sparse array. Consider a 2-dimensional array A of 5 X 5 as shown below

A =

While storing a sparse matrix in linked list format only non-zero elements are considered. Sparse Matrix represented as linked list and matrix A





* **Graded Questions**

1. Define what is linked list?

2. What are the two parts of the linked list?

3. Write a short note on Header list

4. Write and explain an algorithm to split a link list into two linked list

5. Explain how to represent a sparse array using an array and a linked list with an example

6. What is circular linked list? How to traverse a circular linked list?

7. Compare and contrast circular linked list and doubly linked list

7. Write an algorithm to traverse a linked list to find the largest item in the list

8. Write an algorithm to find the location of the element in a sorted linked list

9. Explain how to insert an item in a sorted linked list? Write an algorithm to do the same

10. Write an algorithm to delete a particular node from the linked list

11. Write and explain an algorithm to copy a linked list

12. Write an algorithm to remove all the nodes containing duplicate values

13. Write and explain an algorithm to merge two sorted list into a single sorted linked list

14. What are the different applications of a circular linked list?

15. Write a short note on two way linked list

16. Write an algorithm to count the number of nodes in the linked list

17. Explain how are polynomials represented using linked list?

18. Write an algorithm to delete a node at the kth position in a linked list

19. Write an algorithm to interchange mth and nth elements of a doubly linked list

20. For a circular linked list and doubly linked list write down the algorithm to swap a node with its successor node. Do this by adjusting the pointers and not the data

* **Multiple Choice Questions**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | The \_\_\_\_\_\_\_ field of the last node of the list contains the Null value which indicates the end of the list | | |
| a | First | b | **Next** |
| c | End | d | Last |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | In any list the address of the first node is stored in a special variable known as \_\_\_\_\_\_\_ | | |
| a | First | b | Next |
| c | Top | d | **Begin** |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | \_\_\_\_\_\_\_ is a process in a linked list which refers to visiting each node of linked list in order to process them | | |
| a | Copying | b | Merging |
| c | **Traversing** | d | Splitting |

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | \_\_\_\_\_\_\_\_\_\_ search is only possible in linked list because elements of linked list do no occupy contagious memory location | | |
| a | **Linear** | b | Selection |
| c | Binary | d | Insertion |

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | If there is no free node available for a new node to be allocated this condition is called as \_\_\_\_\_\_\_\_\_\_\_ | | |
| a | Freeflow | b | Underflow |
| c | Directflow | d | **Overflow** |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | \_\_\_\_\_\_\_\_operation refers to putting of elements of two or more lists into one list | | |
| a | Copying | b | Traversing |
| c | **Merging** | d | Splitting |

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | To reverse a linked list we need to use \_\_\_\_\_ pointer variables | | |
| a | **Three** | B | One |
| c | Two | d | Zero |

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | A \_\_\_\_\_\_\_\_\_\_ is a list in which the last node points back to the first node | | |
| a | **Circular** | b | Single |
| c | Doubly Circular | d | Double |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | In a 2-way linked list \_\_\_\_\_\_ part of the node contains the address of the previous node | | |
| a | None Of These | b | **Prev** |
| c | Info | d | Next |

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | A \_\_\_\_\_\_\_ is a special kind of linked list which contains a special node at the beginning of the list which stores the information regarding the linked list | | |
| a | Two-Way Linked List | b | Single Linked List |
| c | Circular Linked List | d | **Header Linked List** |

|  |  |  |  |
| --- | --- | --- | --- |
| 11 | For representation of the polynomials using linked list the first part of the node contains the \_\_\_\_\_\_\_\_\_\_ | | |
| a | Power of the item | b | Link to the next item |
| c | **Coefficient of the variable** | d | None of these |

|  |  |  |  |
| --- | --- | --- | --- |
| 12 | \_\_\_\_\_\_\_\_\_\_\_ is a list in which the Pre part of the header node contains the address of the first node and the next part contains the address of the last node | | |
| a | Circular header linked list | b | **Two-way header linked list** |
| c | Two-way circular header linked list | d | Grounded header linked list |

|  |  |  |  |
| --- | --- | --- | --- |
| 13 | \_\_\_\_\_\_\_ is a list in which the last node of the list contains the null in its Next pointer field | | |
| a | Circular header linked list | b | Two-way header linked list |
| c | Two-way circular header linked list | d | **Grounded header linked list** |

|  |  |  |  |
| --- | --- | --- | --- |
| 14 | \_\_\_\_\_\_\_ is a list in which the last node of the list points back to the header node | | |
| a | **Circular header linked list** | b | Two-way header linked list |
| c | Two-way circular header linked list | d | Grounded header linked list |

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | In case, list pointer variable is Null, indicates | | |
| a | Full linked list | b | **Empty linked list** |
| c | None of these | d | Linked list with one element only |

|  |  |  |  |
| --- | --- | --- | --- |
| 16 | The first part of the single linked list is known as \_\_\_\_\_\_ which holds the element | | |
| a | Begin | b | Next |
| c | Null | d | **Info** |

|  |  |  |  |
| --- | --- | --- | --- |
| 17 | A separate node of free nodes is maintained for allocating an empty node that begins with a pointer \_\_\_\_\_\_\_ | | |
| a | Next | b | Null |
| c | **Free** | d | Begin |

|  |  |  |  |
| --- | --- | --- | --- |
| 18 | The new node can be inserted at \_\_\_\_\_\_\_\_\_\_ | | |
| a | The Beginning | b | The particular position |
| c | The End | d | **All of these** |

|  |  |  |  |
| --- | --- | --- | --- |
| 19 | The node can be deleted at \_\_\_\_\_\_\_\_\_\_ | | |
| a | The End | b | The particular position |
| c | **All of these** | d | The Beginning |

|  |  |  |  |
| --- | --- | --- | --- |
| 20 | To reverse a 1-way linked list we require \_\_\_\_\_\_ pointers | | |
| a | 1 | b | **3** |
| c | 2 | d | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| 21 | A circular linked list can be used for implementing \_\_\_\_\_\_\_\_\_ problem of the operating system | | |
| a | **Timesharing** | b | Multitasking |
| c | Multiprogramming | d | Job Scheduling |

|  |  |  |  |
| --- | --- | --- | --- |
| 22 | While representing a polynomial each node in the linked list will have \_\_\_\_ parts | | |
| a | **3** | b | 4 |
| c | 2 | d | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| 23 | While reversing the linked list the first pointer contains the address of \_\_\_\_\_\_ node | | |
| a | Next | b | Next-To-Next |
| c | **Current** | d | None Of These |

|  |  |  |  |
| --- | --- | --- | --- |
| 24 | While reversing the linked list the second pointer contains the address of \_\_\_\_\_\_ node | | |
| a | **Next** | b | Next-To-Next |
| c | Current | d | None of These |

|  |  |  |  |
| --- | --- | --- | --- |
| 25 | While reversing the linked list the third pointer contains the address of \_\_\_\_\_\_ node | | |
| a | Next | b | **Next-To-Next** |
| c | Current | d | None of These |