Book

A Simplified Approach
to

Data Structures

Prof.(Dr.)Vishal Goyal, Professor, Punjabi University Patiala

Dr. Lalit Goyal, Associate Professor, DAV College, Jalandhar

Mr. Pawan Kumar, Assistant Professor, DAV College, Bhatinda

Shroff Publications and Distributors

Edition 2014

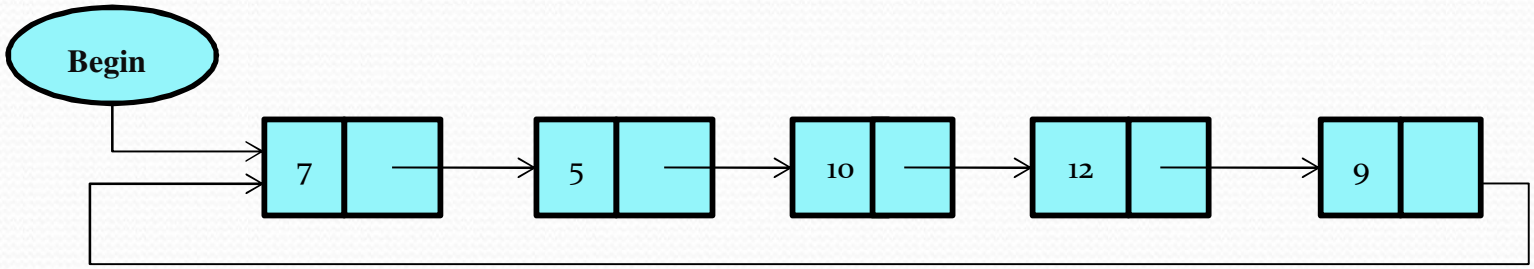
Circular Linked List
and
Doubly Linked List

CONTENT

- **Circular Linked list**
- Traversal in Circular Linked List
- Insertion operation
- Deletion operation
- Applications of Circular Linked List
- **Two-Way Linked List**
- Traversal in two way linked list
- Searching in two way linked list
- Insertion operation
- Deletion Operation

Circular Linked List(continued..)

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. The circular linked list can be shown diagrammatically.



A Circular Linked List

Circular linked list(conti..)

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

- Looking for the end of the linked list –
- In the case of one way singular linked list, the next part of the last node will contain **Null** address.
- In the case of circular linked list, the next part of the last node consist of address of the first node i.e. **Begin.**

Circular linked list(conti..)

For reaching at the end of the circular linked list,we will compare the address of the first node that is **Begin** with the address stored in the **Next** part of each node. If both the addresses come out to be same, then we have reached at the circular list

When a new node is to be inserted at the end of the circular linked list, it's next part will contain the address of the first node instead of null as is in the case of one-way singular linked list.

Algorithm : Traversal in a circular linked list

The traversal of circular linked list having list pointer variable **Begin** and a pointer variable **Pointer** to traverse the linked list from begin to end.

Step1: If **Begin = Null** Then
Print: "Circular linked list is empty"
Exit
[End If]
Step2: Process **Begin → Info**

Traversal in Circular Linked List

Step3: Set **Pointer = Begin → Next**

Step4: Repeat steps 5 and 6 while **Pointer ≠ Begin**

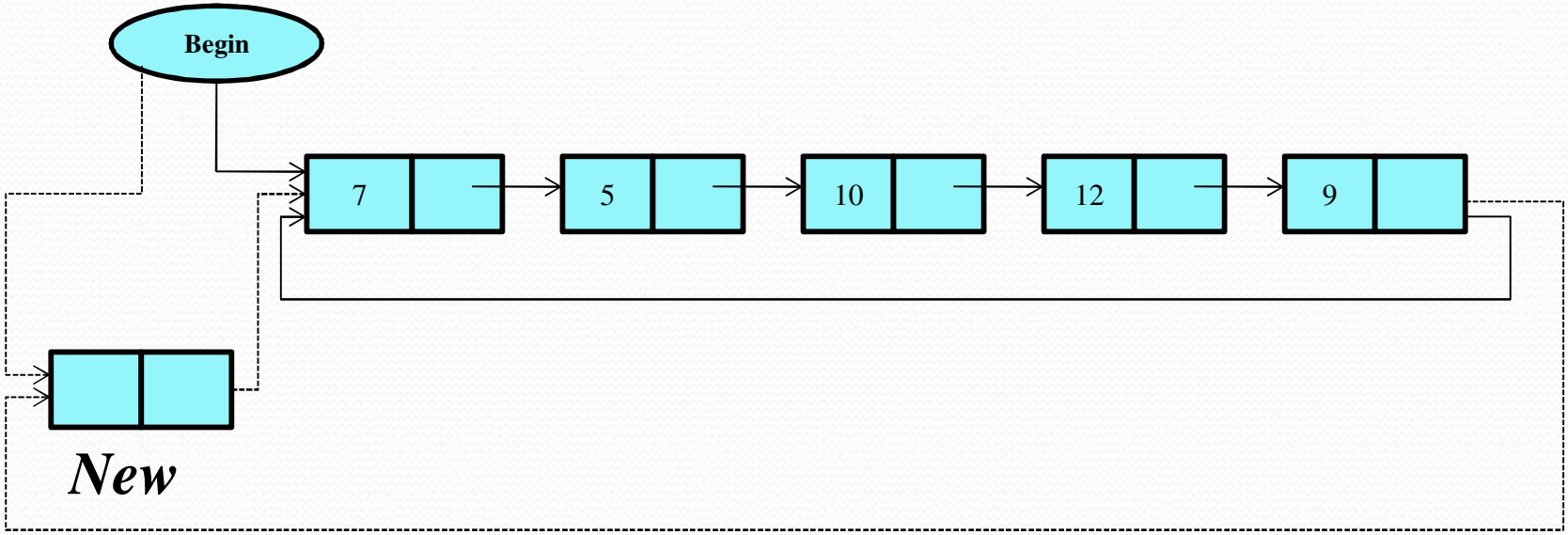
Step5: Process **Pointer → Info**

Step6: Set **Pointer = Pointer→ Next**
[End Loop]

Step7: Exit

Insertion at the Beginning of Circular Linked List

In this case, the **New** node is inserted as the first node and the **next** part of the last node is changed and now it points to the newly inserted node as shown below in figure:



Insertion of a New Node at the Beginning of a Circular Linked List

Algorithm: Insertion at the Beginning

Step1: If **Free = Null** Then
Print: "No free space available for insertion"
Exit
[End If]

Step2: Set **New = Free** And **Free = Free → Next**

Step3: Set **New → Info = Data**

Insertion at the Beginning(cont…)

Step4: If **Begin = Null** Then

Set **Begin = New**

Set **New → Next = Begin**

Exit

[End If]

Step5: Set **Pointer = Begin**

Step6: Repeat while **Pointer →Next ≠ Begin**

Set **Pointer = Pointer → Next**

[End Loop]

Insertion at the Beginning (cont…)

Step7: Set **New → Next = Begin**

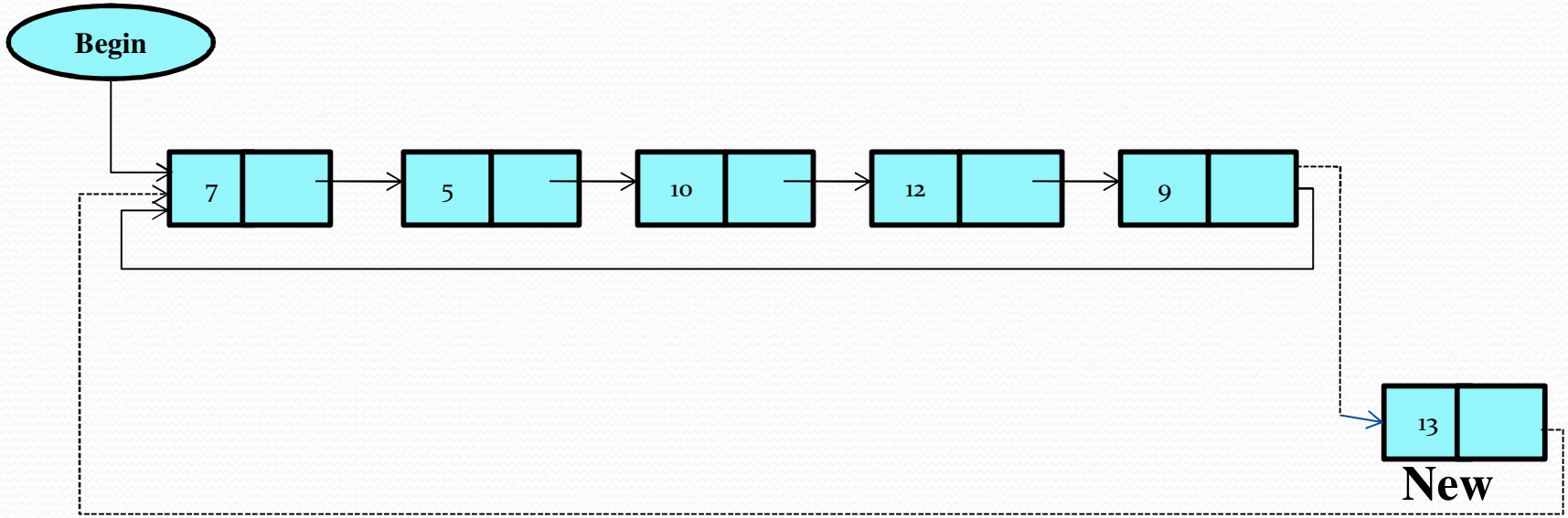
Step8: Set **Begin = New**

Step9: Set **Pointer → Next = Begin**

Step10: Exit

Insertion at the End of Circular Linked List

In the process of inserting an element at the end of the circular linked list, the address stored in the **Next** part of the last node and next part of **New** node need to be changed as shown below:



Insertion of a Node 'New' at the End of the circular linked list

Algorithm: Insertion at the End

Step1: If **Free = Null** Then
Print: "No Free space available for Insertion"
Exit
[End If]

Step2: Allocate memory to node **New**
Set **New = Free** and **Free = Free → Next**

Step3: Set **New→ Info = Item**

Insertion at the End (continued…)

Step4: If **Begin = Null** Then
Set **Begin = New** And **New → Next = Begin**
Exit
[End If]

Step5: Set **Pointer = Begin**

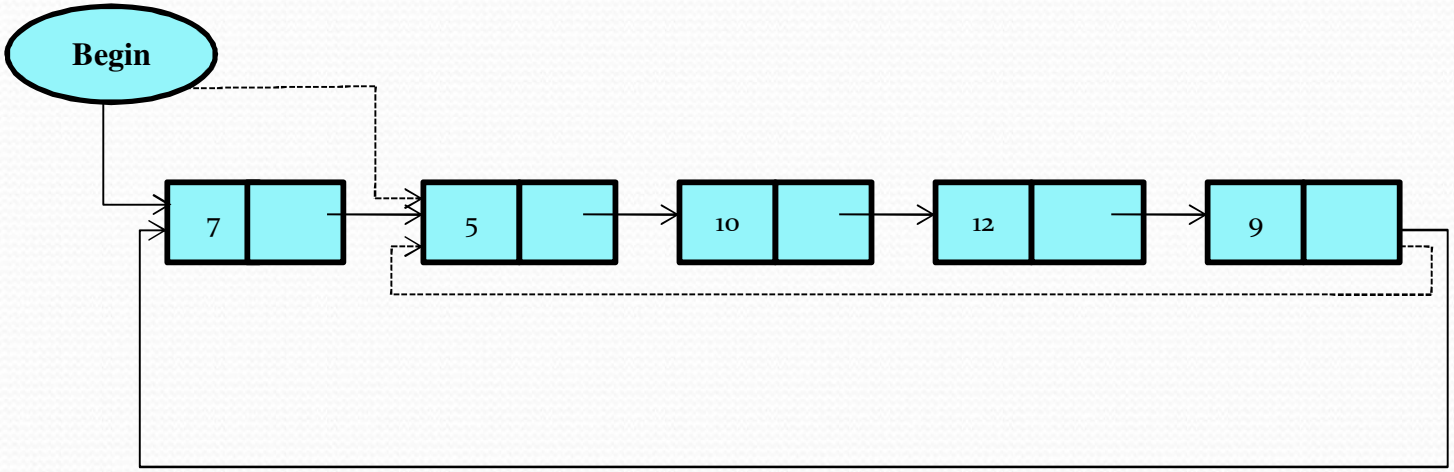
Step6: Repeat while **Pointer → Next ≠ Begin**
Set Pointer = Pointer → Next
[End Loop]

Step7: Set **Pointer → Next = New** and **New → Next = Begin**

Step8: Exit

Delete First Node in Circular Linked List

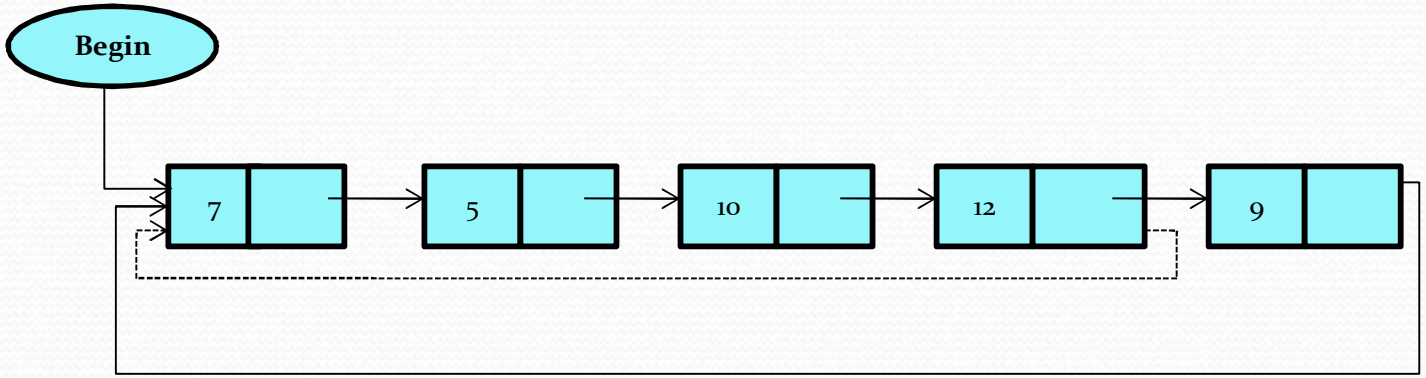
To delete the first node from the circular linked list,**Begin** will point to info part of second node and **Next** part of last node will point to info part of second node.



Deletion of First Node in Circular Linked List

Delete Last Node in Circular Linked List

To delete the last node from the circular linked list the **Next** part of Second last node will point to the **Info** part of first node and last node will be deleted.



Deletion of Last Node in Circular Linked List

Applications of Circular Linked List

Circular linked list can be used for:-

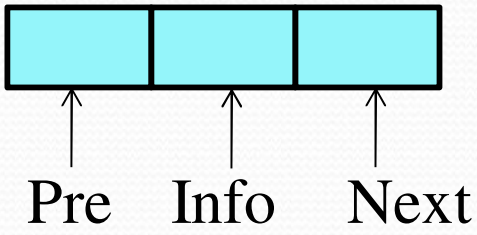
Implementing a time sharing problem of the operating system. In The operating system must maintain a list of executing processes and must alternately allow each process to use a slice of CPU time, one process at a time.The operating system will pick a process; let it use a small amount of CPU time and then move to next process. For this application there should be **no Null** Pointer unless there is no process requesting CPU time.

Two-Way Linked List(Doubly Linked List)

In Two-Way Linked List, we traverse the list in both the directions

- Forward direction (from beginning to end)
- Backward direction (from end to beginning).

The Two-Way Linked List is also known as **Doubly Linked List**. In Two-Way Linked List, each node is divided into three parts: **Pre, Info, Next.** The structure of a node used in Two-Way Linked List is as shown below:



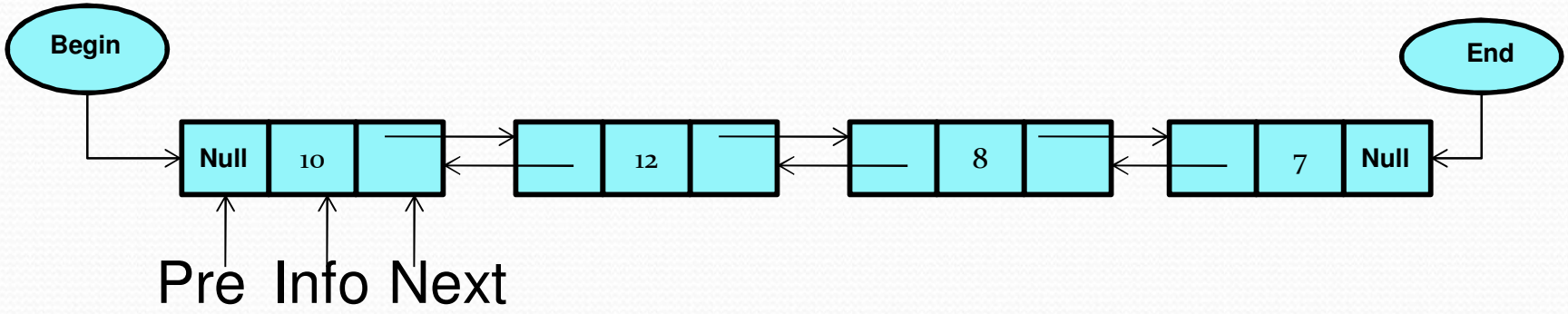
Structure of a Node used in a Two-Way Linked List

Two-Way Linked List (conti…)

- **Pre** part contains the address of the preceding node.
- **Info** part contains the element.
- **Next** part contains the address of the Next node.

Two-Way Linked List (conti…)

Here in Two-Way Linked List, two list Pointer variables i.e. **Begin** and **End** are used which contains the address of the first node and last node of the Linked List respectively . Two-Way Linked List can be shown diagrammatically as:



The **Pre** part of the first node of a Two-Way Linked List will contain **Null** as there is no node preceding the first node and the **Next** part of last node will contain **Null** as there is no node following the last node.

Two-Way Linked List (conti…)

Various operations performed on Two-Way Linked List or double linked list:

- Traversing
- Searching
- Insertion
- Deletion

Traversing a Two-Way Linked List

A Two-Way Linked List can be traversed in both the directions:

•**Forward direction**, the Pointer variable will be assigned with the address stored in the **Begin** pointer variable and reach at node whose **Next** part contains **Null** i.e. we reach at the end of list.

• **Backward direction**, the Pointer variable will be assigned with the address stored in the **End** pointer variable and move backward tiill we reach at node whose **Pre** part contains **Null** i.e. we reach at the beginning of the list. The variable **Pointer** keeps track of the address of the current node.

Algorithm: Traversing a two-way linked list

Step1: If **End = Null** Then
Print: "Linked List is empty"
Exit
[End If]

Step2: Set **Pointer = End**

Step3: Repeat while **Pointer ≠ Null**
Process **Pointer → Info**
Set **Pointer = Pointer → Pre**
[End Loop]
Step4: Exit

Searching in a Two-Way Linked List

To find the location of a given item in linked list:

•Traverse the list either from the end or from beginning of the linked list.

•Keep comparing the element stored in each node with the desired item.

•If desired item is found then further traversing is stopped and address of the node containing the desired element is returned.

Algorithm: Searching in Double Linked List

To find the position of a given element 'data' in a Two-Way Linked List by traversing it from end to beginning.

Step1: If **End = Null** Then
Print: "Linked List is empty"
Exit
[End If]

Step2: Set **Pointer = End**

Searching in Double Linked List

Step3: 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]

Step4: Print: "Element **Data** is not found in the linked list"

Step5: Exit

Insertion of an element in Two Way Linked List

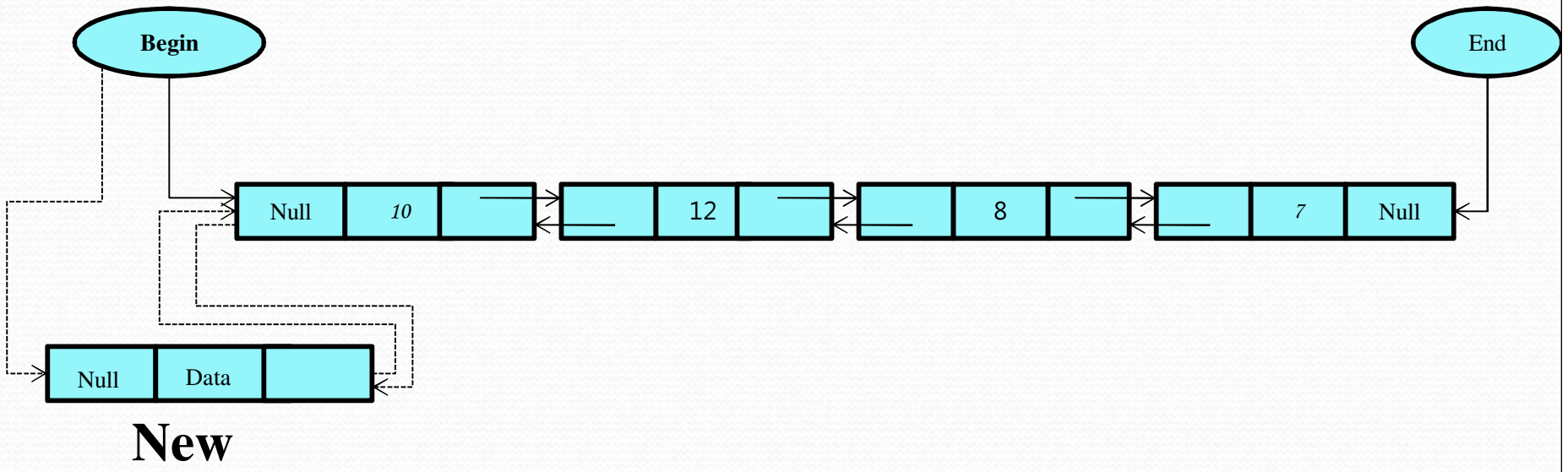
Insertion can take place at various positions in a linked list such as:

•At beginning of linked list

•At the end of linked list or after any particular node in a linked list. Insertion at the beginning or at the end of the linked list can be accomplished after changing a few pointers.
•Insertion after a particular node in the linked list requires finding the location of the node after which new node is to be inserted.

Insertion of node at Beginning of doubly linked list

An element **Data** is to be inserted at the beginning of the doubly linked list.



Insertion of a node at the Beginning in a Doubly linked List

Algorithm: Insertion of a New node at Beginning

Step1: If **Free = Null** Then
Print: "Free space not available"
Exit
[End If]

Step2: Allocate memory to node **New**
(Set **New = Free** And **Free = Free → Next**)

Step3: Set **New → Pre = Null** And **New → Info = Data**

Insertion of a New node at Beginning

Step4: If **Begin = Null** Then

Set **New → Next = Null** And **End = New**

Else

Set **New → Next = Begin** And **Begin → Pre = New**

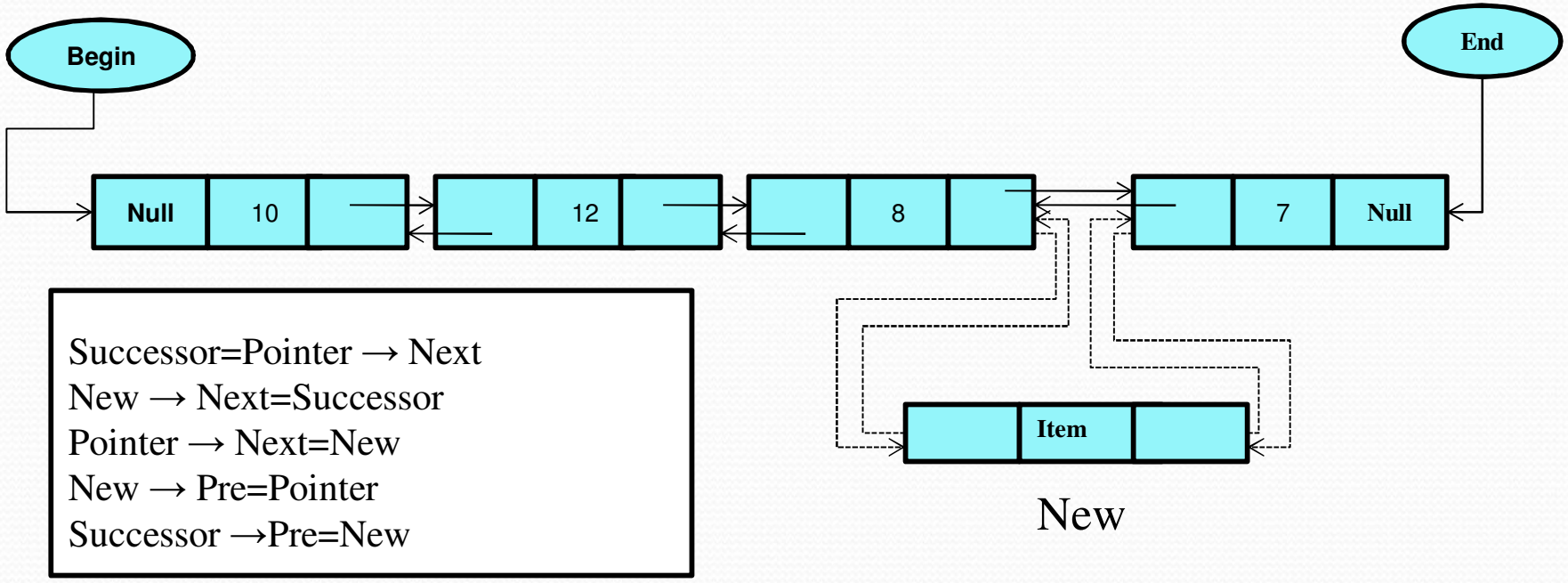
[End If]

Step5: Set **Begin = New**

Step6: Exit

Insertion of New node after particular Data

Insertion after a particular node requires **finding the location of the node after which new node is to be inserted.** After finding the desired node, the New node can be inserted easily by changing few pointers as shown:



Insertion of a node 'New' in the Linked List after a particula element 'Data'

Algorithm: Insertion after particular node

Insert a New node 'Item' after a given element 'Data' in the Two-Way Linked List

Step1: If **Free = Null** Then
Print: "Free space not available"
Exit
[End If]

Step2: **Begin = Null** Then
Print: "List is Empty, No insertion will take place"
Exit
[End If]

Insertion after particular node(conti..)

Step3: Set **Pointer = Begin**

Step4: Repeat while **Pointer → Next ≠ Null**
And **Pointer→ Info ≠ Data**
Pointer = Pointer → Next
[End Loop]

Step5: If **Pointer → Next = Null** And **Pointer → Info ≠ Data**
Print: "Item cannot be inserted as element Data is not present"
Exit
[End If]

Insertion after particular node(conti..)

Step6: Allocate memory to node **New**
(Set **New=Free** and **Free = Free → Next**)
Set **New → Info = Item**

Step7: If **Pointer → Next ≠ Null** Then
Successor = Pointer → Next
New → Next = Successor
Pointer → Next = New
New → Pre = Pointer
Successor → Pre = New

Insertion after particular node(conti..)

Else

New → Next = Null

New → Pre = Pointer

Pointer → Next = New

End = New

[End If]

Step8: Exit

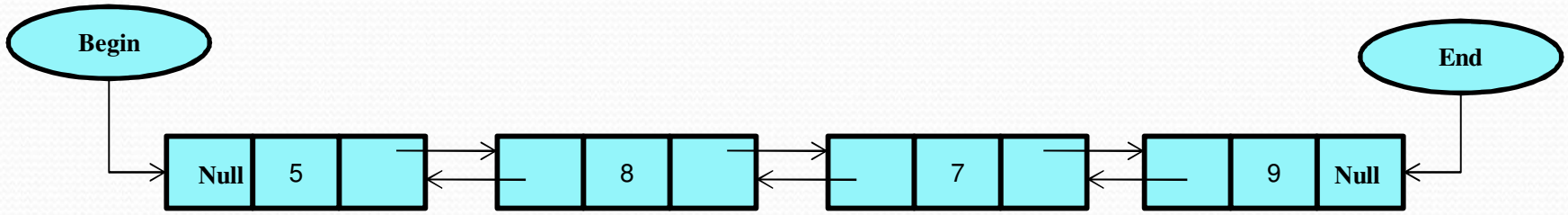
Deleting a node with given Item

For deleting a particular node:

•Traverse the list either in forward or backward direction to locate the node containing the element to be deleted

•If the desired node is not found and we reach at the end of a list then an appropriate message is displayed.

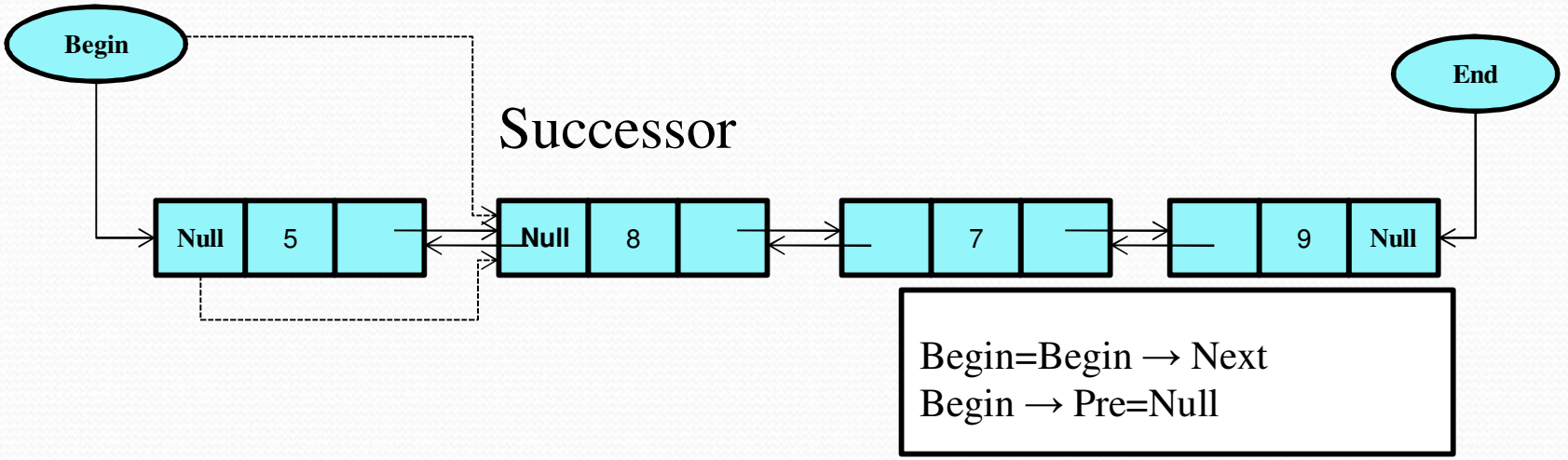
•If the desired node is found then it can be removed from the linked list by changing few pointers as shown:



A Two-Way Linked List with 4 nodes

Deleting 1st node with given Item

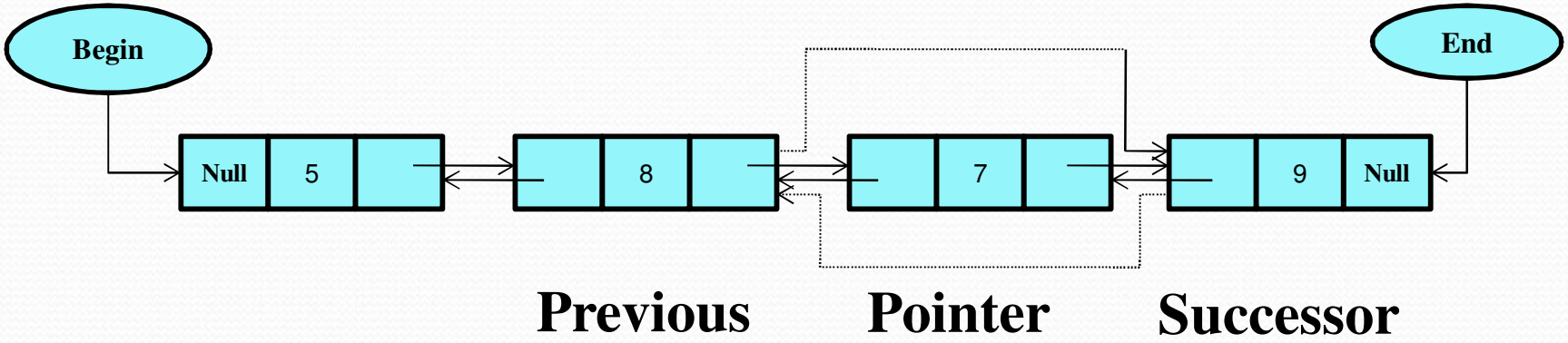
CASE1: Suppose we want to delete an element 5, which is contained in the first node of the list. Deletion will be performed as shown in the figure below:



Deleting the 1st node of a Two-Way linked List

Deleting a particular node

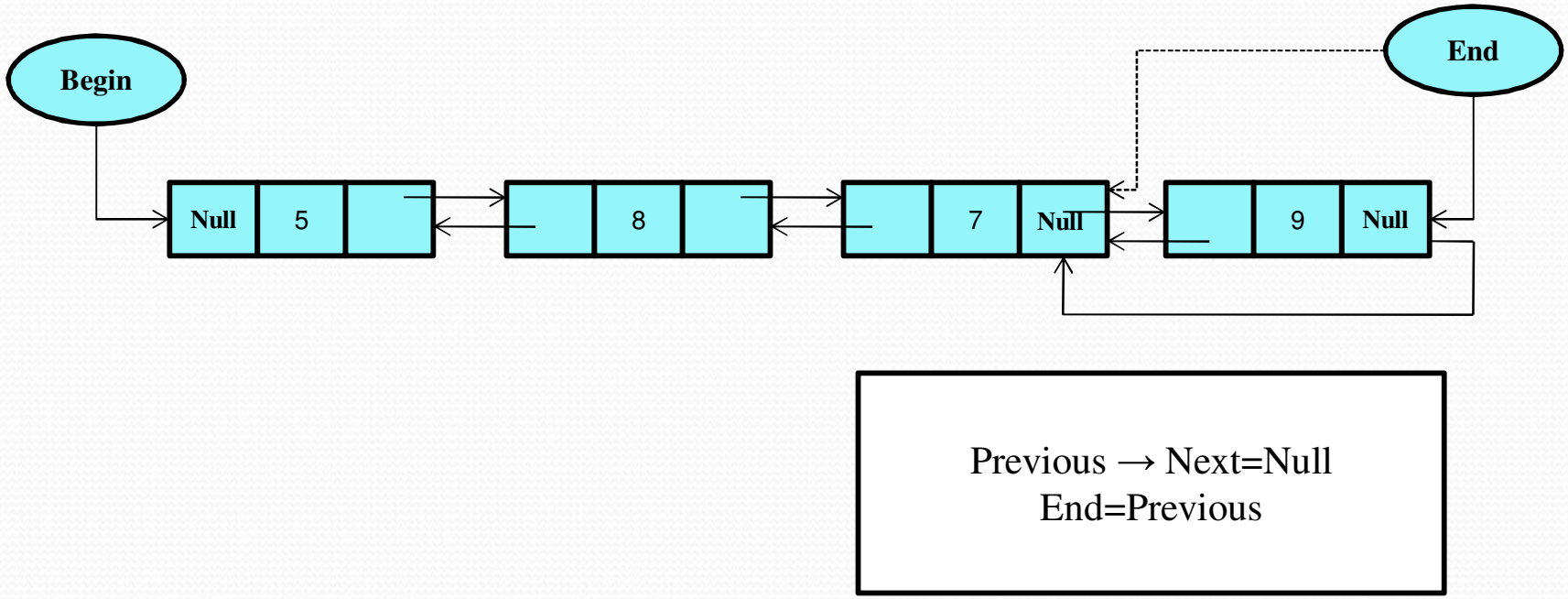
CASE2: Suppose we want to delete an element 7, which is contained in a node that lies between the first and last node of the list then deletion will be performed as shown in the figure below:



Deleting a Node present between the first and the last node of a Two-Way Linked List.

Deleting last node with given item

CASE3: Now we will delete an element 9 which is encountered in the node which is last node of the list as shown:



Deleting the last node from a Two-Way Linked List

Algorithm: Deleting a node with given Item(conti..)

Step1: If **Begin = Null** Then
Print: "Linked List is already empty"
Exit
[End If]

Step2: If **Begin → Info = Item** Then //First node to be deleted
Pos = Begin
Begin = Begin → Next
Begin → Pre = Null

Deleting a node with given Item(continue..)

//Deallocate memory held by **Pos**
Pos → Next = Free, Free = Pos
Exit
[End If]

Step3: Set **Pointer = Begin → Next**

Step4: Repeat while **Pointer → Next ≠ Null**
And **Pointer → Info ≠ Item**
Set **Pointer = Pointer → Next**
[End Loop]

Deleting a node with given Item (continue..)

Step5: If **Pointer → Next = Null** And **Pointer → Info ≠ Item**

Print: " Item to be deleted not found"

Exit

[End If]

Step6**:** If **Pointer → Next = Null** Then //last node to be deleted

Set **Previous = Pointer → Pre**

Set **Previous → Next = Null**

Set **End = Previous**

Else

Set **Previous = Pointer → Pre**

Set **successor = pointer → Next**

Set **Previous → Next = successor**

Set **successor → pre = Previous**

[End if]

Deleting a node with given Item

Step 7: Deallocate memory helld by **pointer**

Pointer → Next = Free, Free = Pointer

Step 8: Exit