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

ONE-WAY LINK LIST

Contents

- Introduction To Linked list
- Introduction To One Way Linked list
- Operations on One Way Linked List

Introduction to Linked List

Linked List

- A linked list can be defined as the linear collection of elements where each element is stored in a node.
- The linear order b/w elements is given by means of pointers instead of sequential memory locations.

Types of Linked List

- Singular or one-way linked list.
- Doubly or two-way linked list.
- Circular linked list.
- Header linked list.

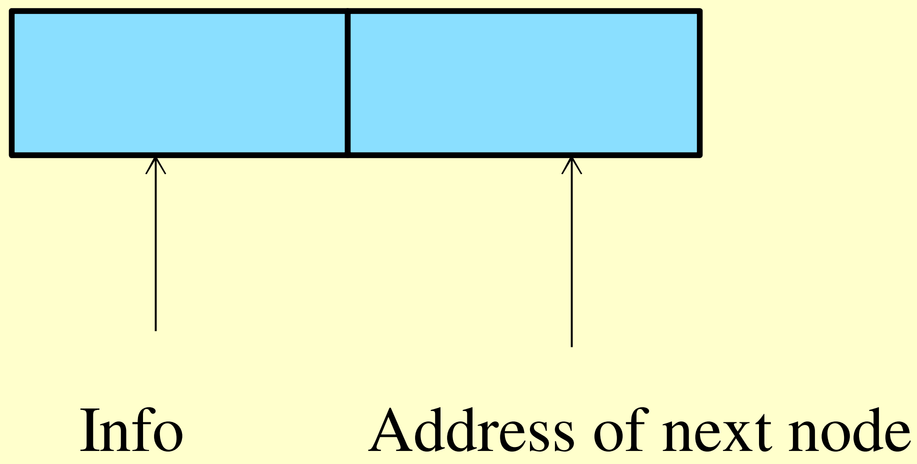
One Way Linked List

It is also known as singular linked list.

Each node has two parts:-

- The first part is known as info part which holds the element.
- Second part is known as next part which holds the address of next node.

One Way Linked List



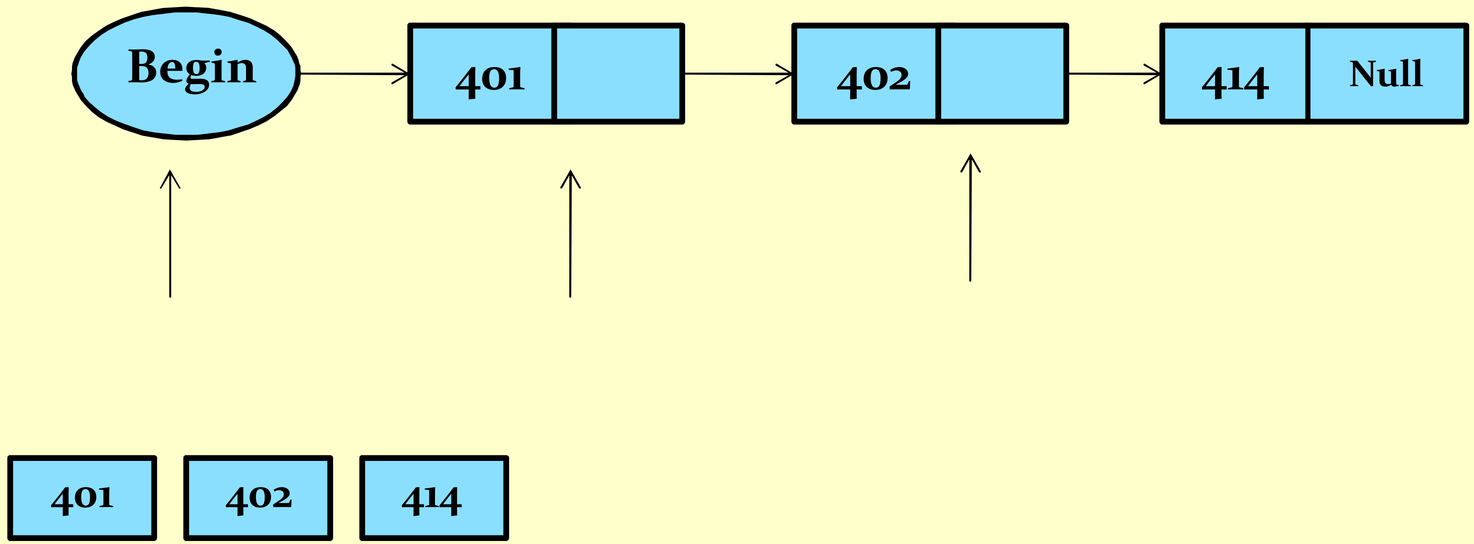
Operations of 1-Way Linked List

- Traversing a linked list.
- Searching an element in linked list
- Inserting an element in linked list
- Deleting an element in linked list
- Copying a linked list.
- Merging two linked lists
- Splitting linked list into further parts.

Traversing A Linked List

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

Example:- list of students with roll no.



Algorithm : ***Traversal of linked list***

Step1.If ***Begin=Null*** then

Print ”linked list is empty”

Exit

[End If]

Step2. Set ***Pointer =Begin***

Step3.Repeat While ***Pointer!=Null***

a. print : ***Pointer → Info***

b. Assign ***Pointer =Pointer → Next***

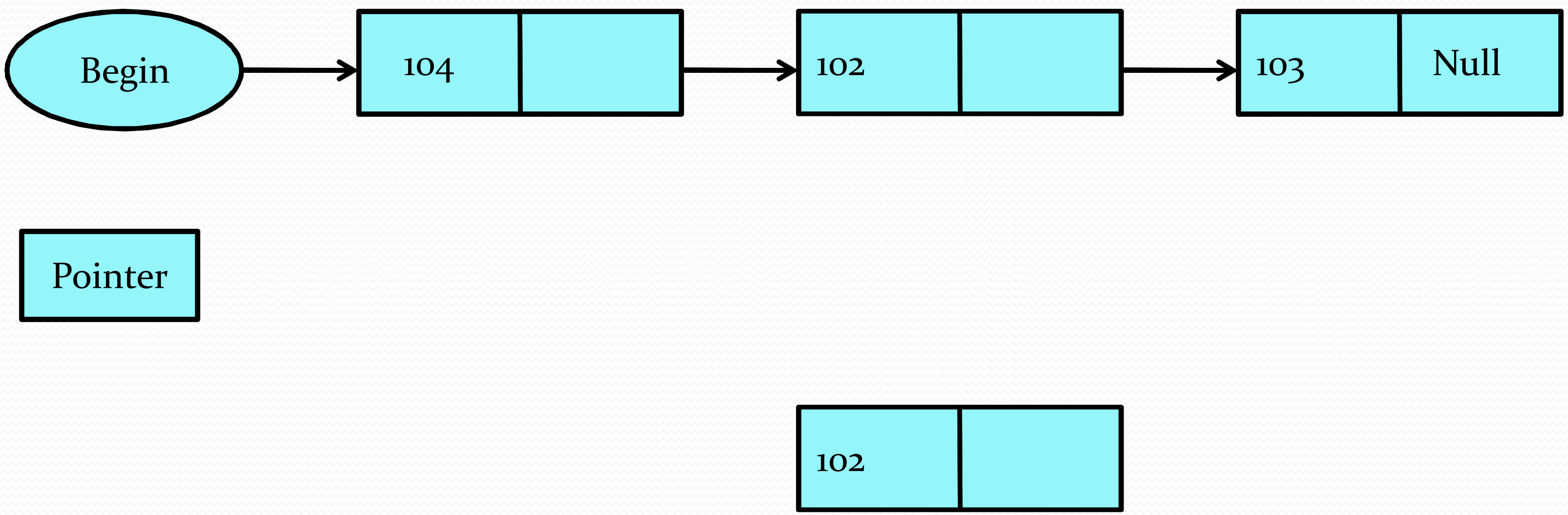
[End loop]

Step4.Exit

Searching In A Linked List

- In searching we traverse the list from begin and compare the elements stored in each node with the desired element to be searched
- If match is found then the address of the node is returned otherwise we proceed to next node
- If element not found till null then search unsuccessful

Searching In A Link List



Item found in link list

Algorithm :*Searching a Linked List*

Step1: 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 linked list"

Step 5 :Exit

Memory Allocation/Deallocation

Before we move on to insertion and deletion in a linked list part let's discuss about memory allocation and deallocation.

- To insert an element into the linked list , First we need is to get a free node.

- In case of deletion of a node, it is desired to return to memory taken by deleted node for its reusability in future.

Insertion In Linked List

- First we get a free node from the free storage list

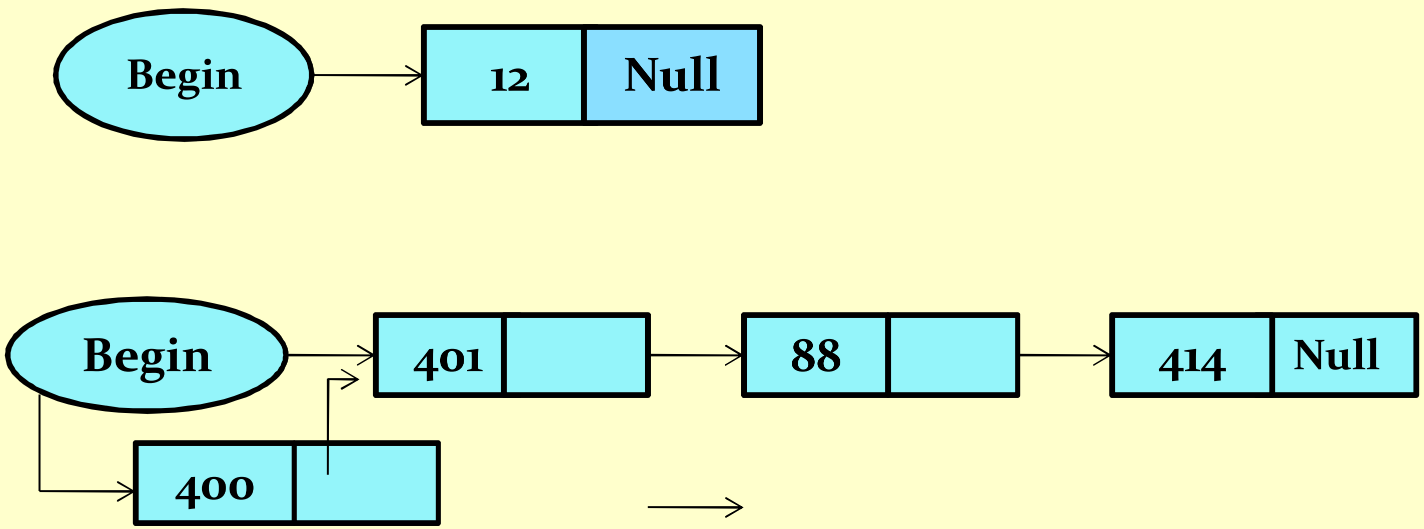
- Then element to be inserted is placed into the info part of the node and pointers are set to add the new node at desired location of list.

Insertion In Linked List (Continued)

Where we can insert element in linked list?

- At the beginning of linked list
- At end
- At a particular position in list
- In the sorted linked list

Insertion At Beginning



Algorithm : ***Insertion At Beginning***

Step 1: If ***Free* = *Null*** Then

Print : "Overflow: No free available for insertion"

Exit

[End if]

Step 2: Allocate Space to node ***New***

(set ***New* = *Free*** And ***Free* = *Free* → *Next***)

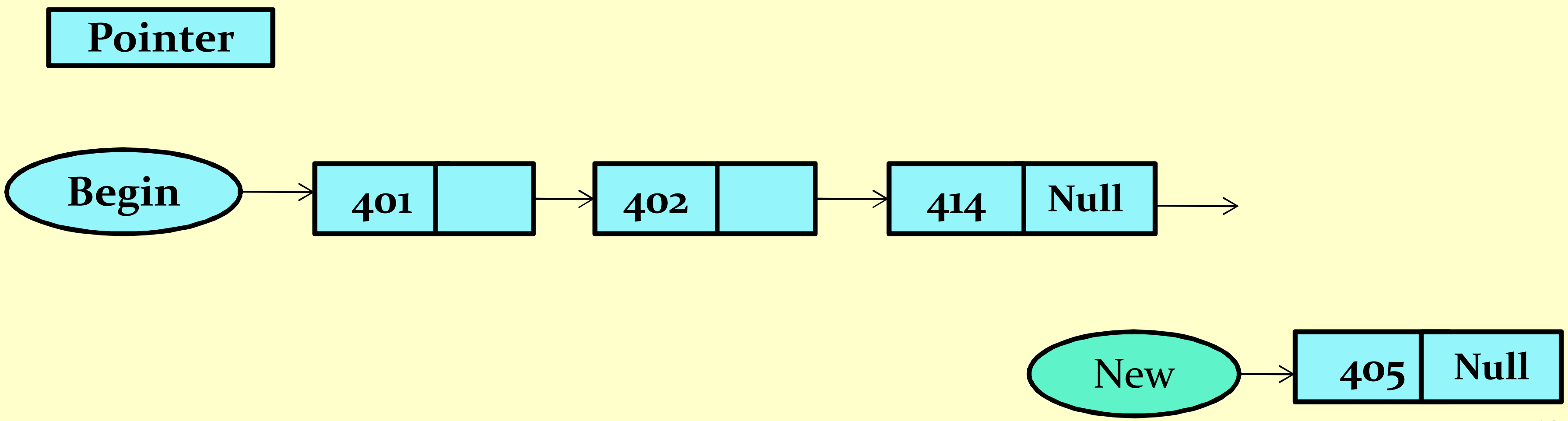
Step 3: Set ***New*→*Info* = *Data***

Step 4: Set ***New* →*Next* = *Begin*** And ***Begin* = *New***

Step 5: Exit

Insertion At End

- If list is empty, store null value in next part of new node and insert item in the info part
- If list is not empty, traverse list till end node.
- Store address of new node into next part of the last node of the linked list and the next part set to null.



<u>Algorithm</u> : *Insertion At End*

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]

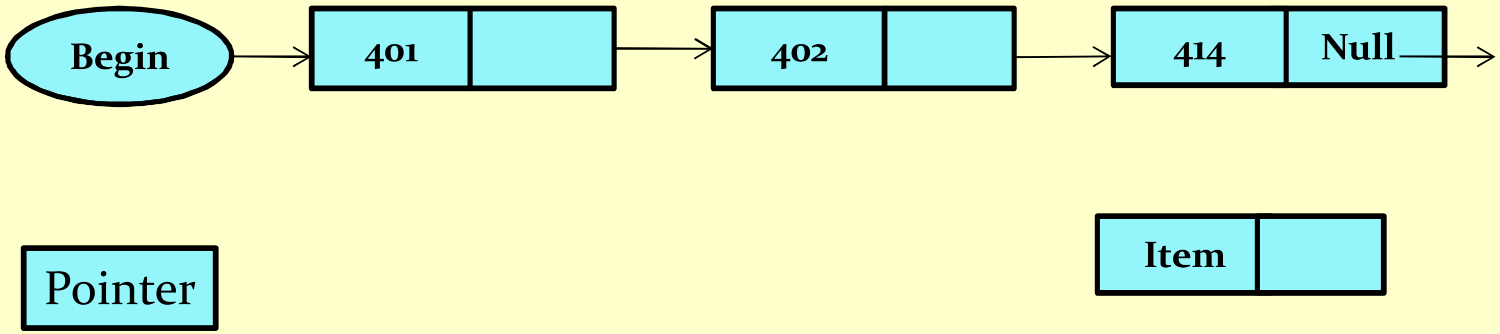
Step 7: Set ***Pointer* → *Next* = *New***

Step 8: Exit

Insertion At A Particular Position

- Locate the position of the node after which we want to insert the new node.
- 2 cases are there if location found and if not found
- Traverse till we not reach on desired loc.
- If we reach on desired loc. Then loc. Found insert element if we reach on end but not find a loc. Yet then loc. Not found.

Insertion At Any Location



New → Next=Pointer → Next

Pointer →Next=New

Algorithm : *Insertion At Any Location*

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, so insertion is not possible.”

Else

Allocate space to node ***New***

Set ***New=Free, Free=Free→ Next, New→ Info=Item***

Set ***New→Next=Pointer →Next***

Set ***Pointer → Next=New***

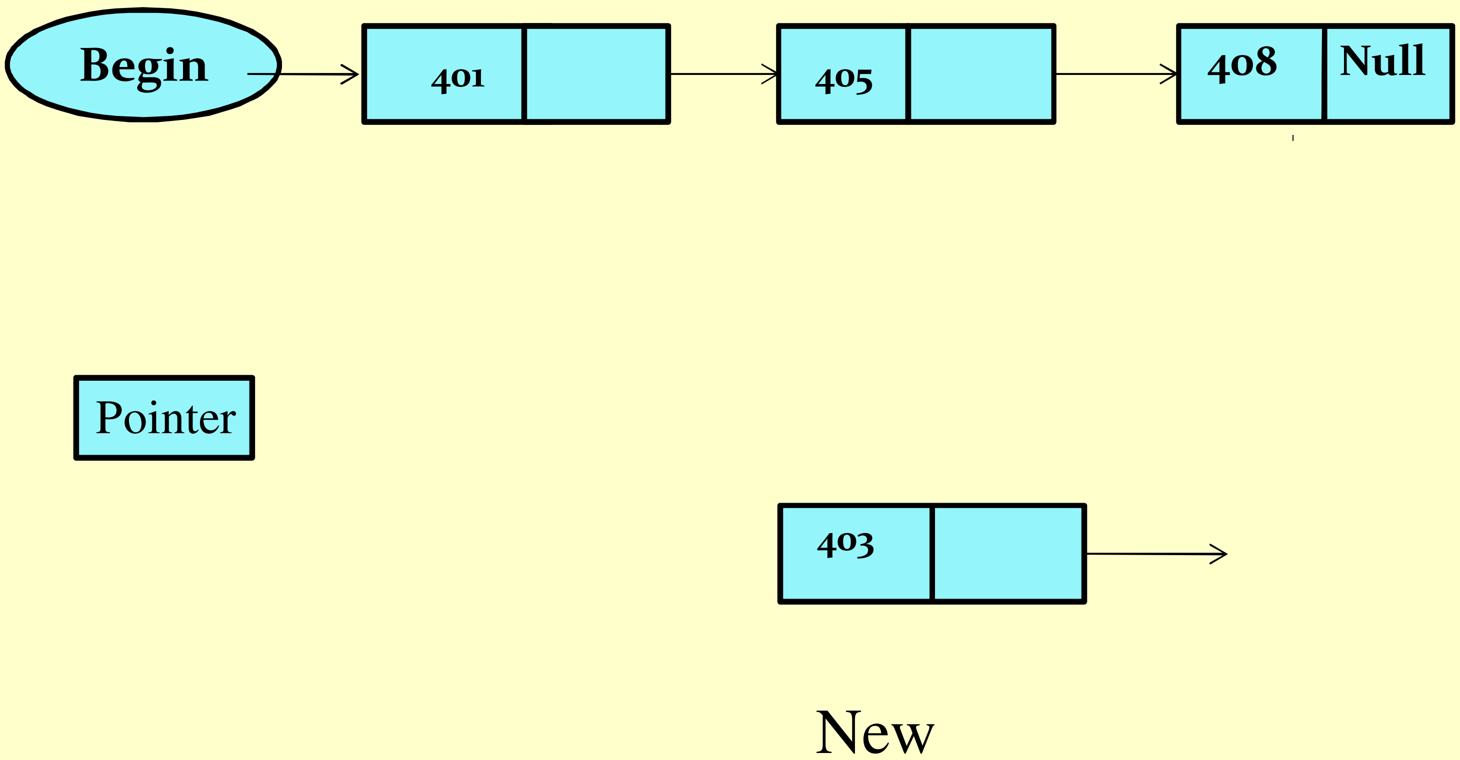
[End If]

Step 5: Exit

Insertion In Sorted Linked List

- Find the position of the node after which new node has to be inserted
- List can be sorted in ascending order and descending order
- In ascending order first we compare the element with the first element if inserted element is small then it will inserted at first position else comparing goes on in desc. Order it is opposite.

Insertion In A Sorted Linked List



Algorithm : *Insertion At Any Location In sorted Link List*

Step 1: If ***Begin = Null*** Then

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 ***Pointer = 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 space for insertion , Allocation of space to node New is not possible"

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 = Pointer → Next***

Set ***Pointer1 → Next = New***

[End If]

Step 9: Exit

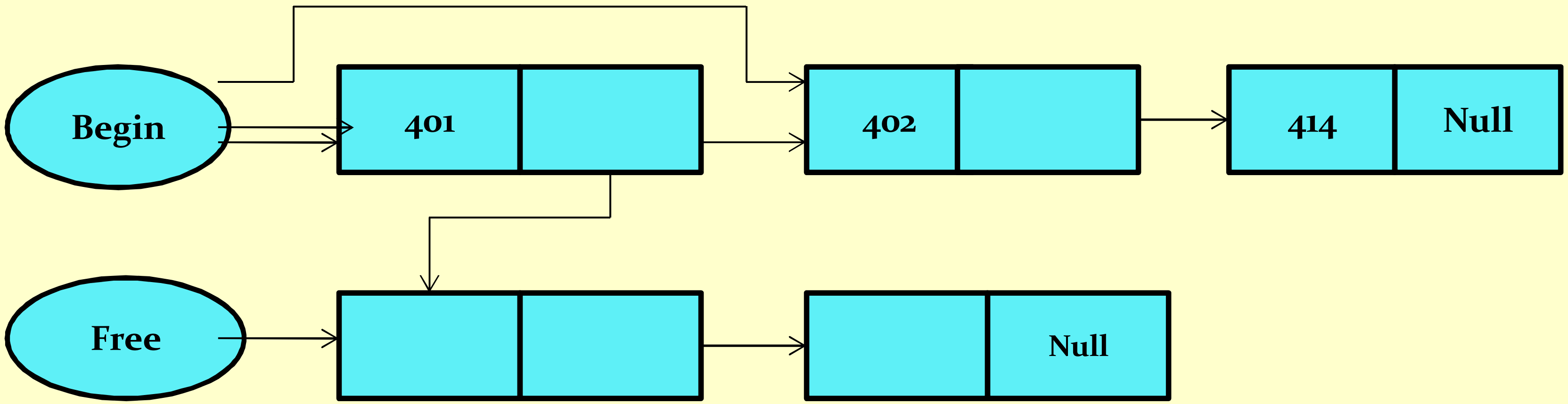
Deletion From Linked List

Deletion can be done in 3 ways:

- Deleting a node at the begin of link list
- Deleting a node at end.
- Del. A particular node in the linked list.

Deleting A Node At Begin.

- Deletion of a node at the begin of the list is a very simple operation which can be done by changing the list pointer variable begin.
- Now begin will point to next node in the list.
- The space occupied by the deleted node is returned to the free storage list.



DELETION OF NODE FROM BEGIN OF LIST

Algorithm : ***Deletion At Beginning 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: ***Begin = Begin → Next***

Step 4:***Pos → Next = Free*** and ***Free = Pos***

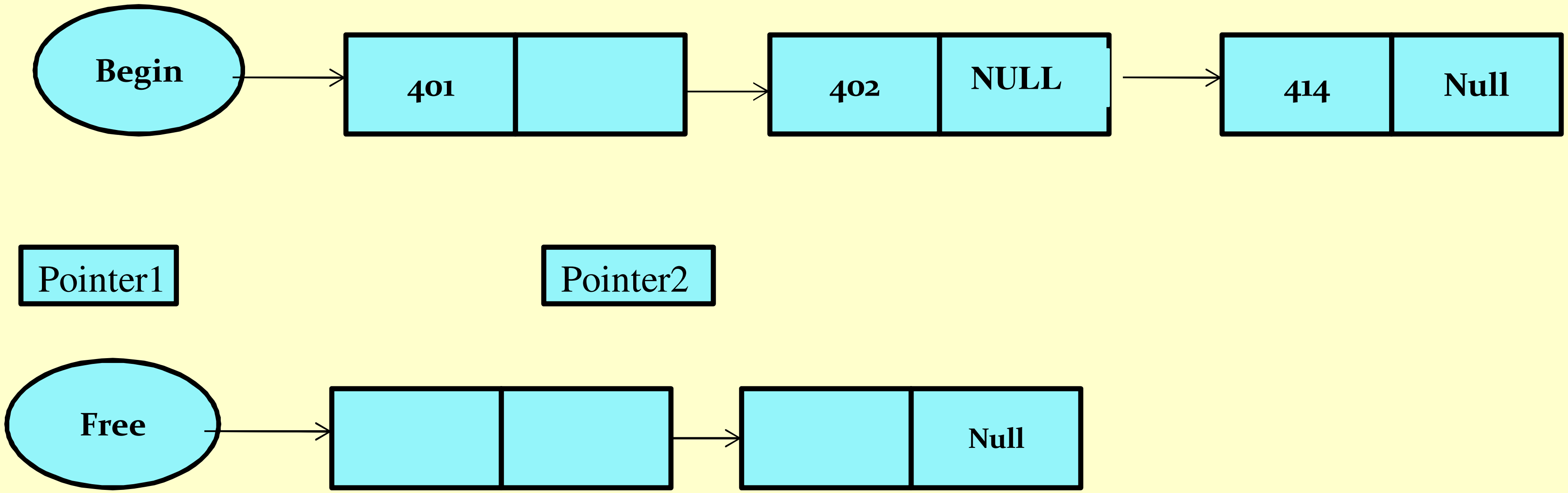
Step 5:Exit

Deleting A Node At End

- For deleting the lost node from the given plinked list, it is necessary to traverse the entire linked list for finding the address of the preceding node of the last node i.e, address of 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 2nd last node.

Deleting A Node At End



Pointer1 = Begin
Pointer2 = Begin->Next

Pointer1 = Pointer2
Pointer2 = Pointer2 -> Next

Pointer1->Next=Pointer2->Next
Pointer2->Next=Free
Free=Pointer2

Algorithm : *Deletion At End*

Step 1:If ***Begin* = *Null*** Then
    Print : "Linked List 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 ***Pointer2* → *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

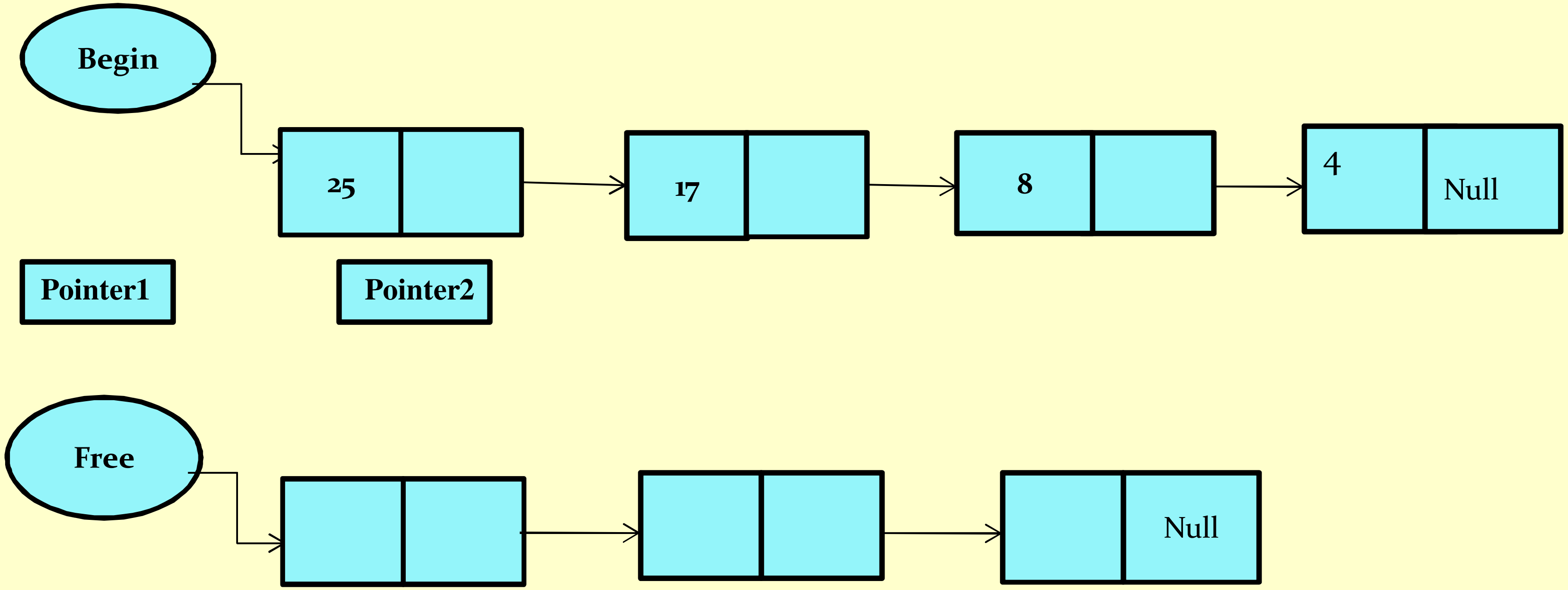
(***Pointer2*** ➞ ***Next*** = ***Free*** and ***Free*** = ***Pointer2***)

Step 8:Exit

Delete A Particular Node From Link List

- For deleting a particular node from the linked list, the first task is to find the address of the preceding node of nth node to be deleted.
- To complete the task traverse the linked list from begin and compare the info. Stored in node with item.
- Two pointers pointer1 pointer2 will be used while traversing the list for locating the address of the node to be deleted and address of it's preceding node.

Deleting A Particular Node In Link List



Pointer1 ->Next=Pointer2->Next

Pointer2->Next=Free

Free=Pointer2

<u>Algorithm</u>: ***Deletion At Any Location***

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 ***Pointer2 → Info != Item***

SET ***Pointer1 = Pointer2*** and ***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

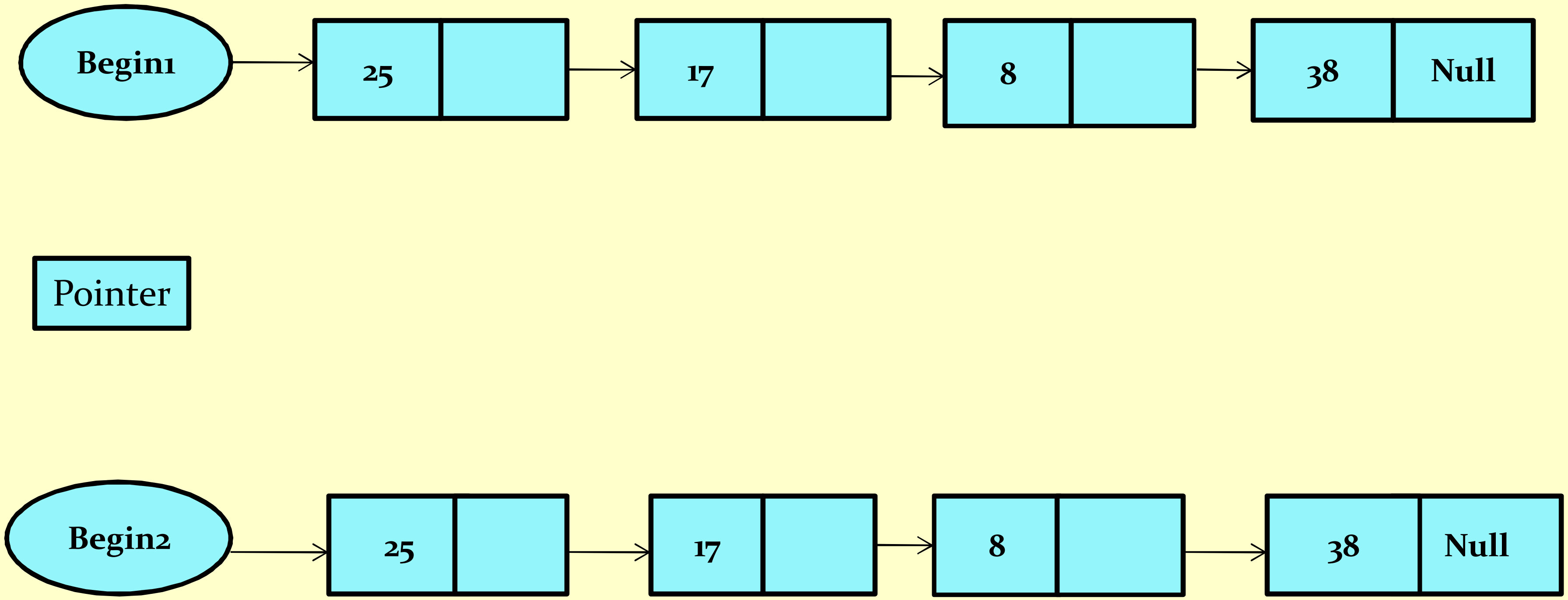
Copy A Link List Into Other Link List

- Consider the linked list with its start pointer as begin1.For copying this given linked list into another list, use a new pointer variable begin2 for the list in which source list will be copied.

- Initially we will store null in the list variable begin2.

- Now we will traverse the entire source list from begin to the end by copying the contents to the new target.

Copy A Link List Into Other Link List



A Link List Is Copied

<u>Algorithm</u>: *Copying One Link List Into Another Link List*

Step 1: If ***Begin1=Null*** Then

Print: "Source List is Empty"

Exit

[End If]

Step 2: Set ***Begin2=Null***

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

Print: "Free space not available"

Exit

Else

Allocate memory to the node ***New***

Set ***New=Free*** And ***Free=Free Next***

[End If]

Step 4: Set ***New*** $\rightarrow$ ***Info=Begin1*** $\rightarrow$ ***Info*** And ***New*** $\rightarrow$ ***Next=Null***

Step 5: Set ***Begin2=New***

Step 6: Set ***Pointer1=Begin1 → Next*** And ***Pointer2=Begin2***

Step 7: Repeat While ***Pointer1!=Null*** And ***Free!=Null***

a. Allocate memory to node ***New***

(***New=Free*** And ***Free=Free→ Next***)

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"

Else

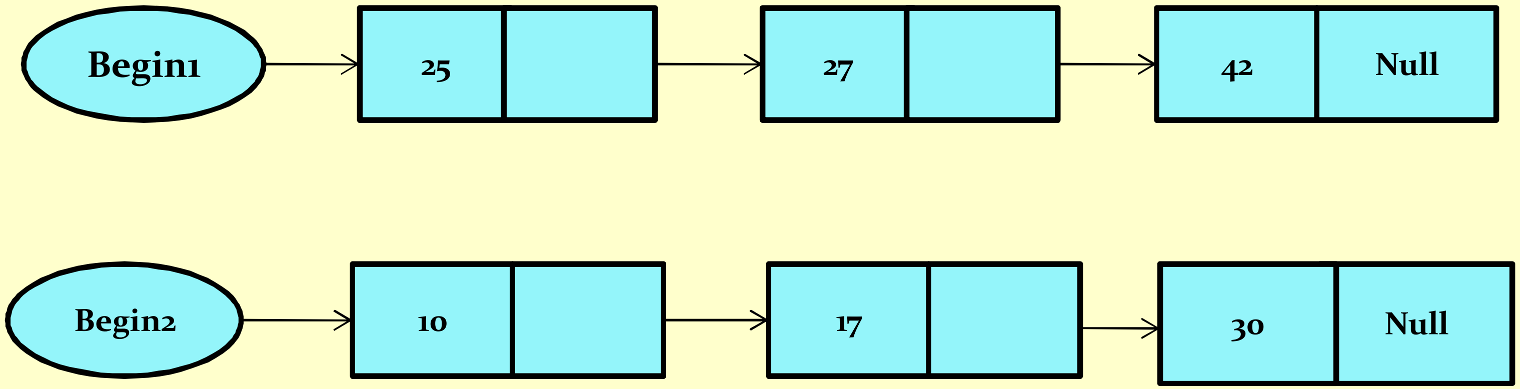
Print: "Not enough space to perform copy operation"

[End If]

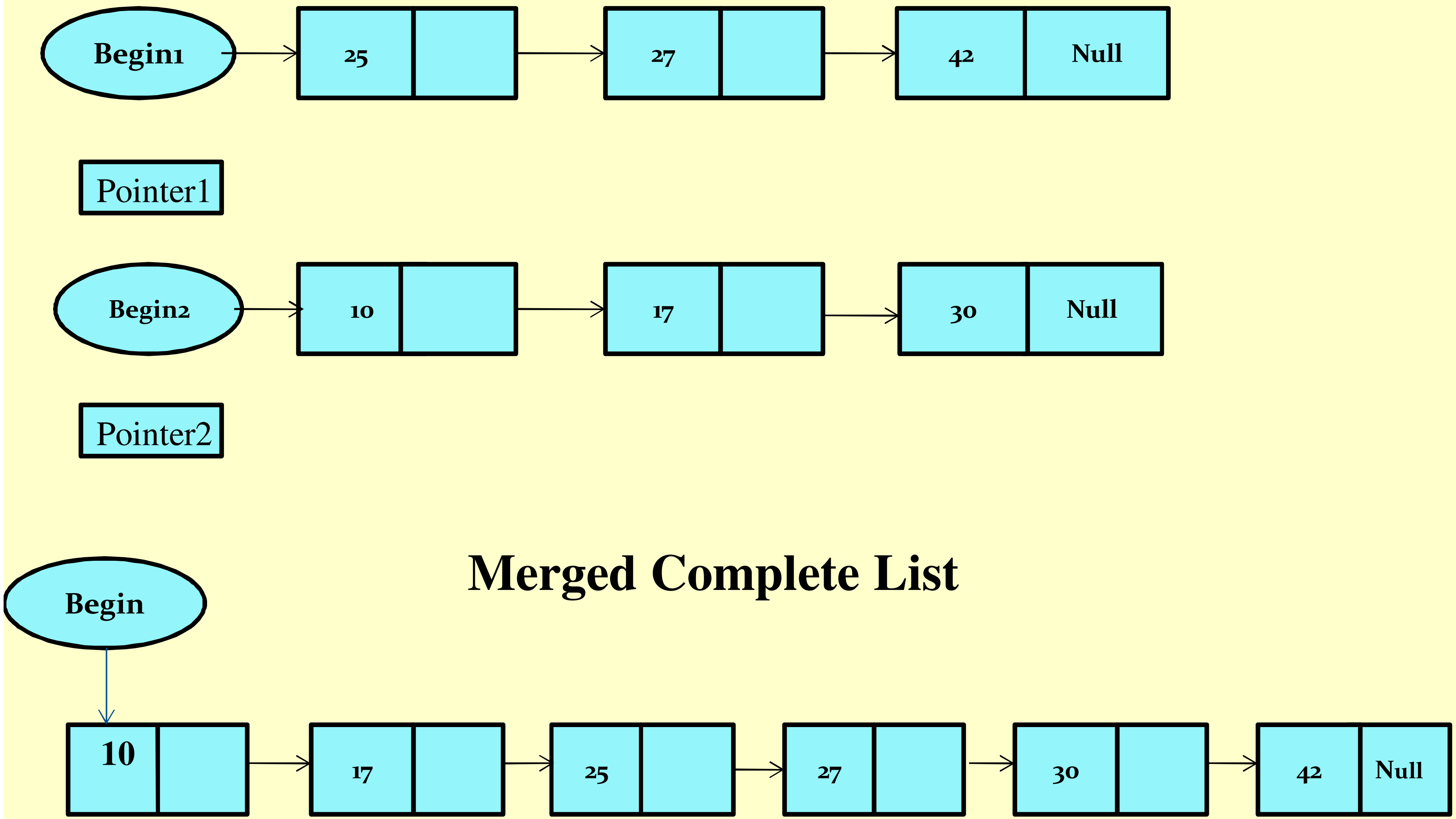
Step 9: Exit

Merging Two Linked List

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



After Merging



Algorithm : *Merging Two Sorted Linked List*

Step 1: If ***Begin1=Null*** or ***Begin2=Null*** then

Print "one of the given linked list is empty"

Exit

[end if]

Step 2: If ***Free =Null*** then

Print: "no free space available"

Exit

Else

//Allocate memory to node New

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 7and 8 while ***Pointer1!=Null*** and ***Pointer2!=Null***

Step 7: If ***Free=Null*** then

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=Pointer*** → ***Info***

Set ***New*** → ***Next=Null***

Set ***Pointer*** → ***Next=New***

Set Pointer=New and ***Pointer2=Pointer2*** → ***Next***

[End If]

[End Loop]

Step 9: If ***Pointer1=Null*** and ***Free!=Null*** then

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***

a. Set ***New=Free*** and ***Free=Free*** → ***Next***

b. Set ***New*** → ***Info=Pointer1*** → ***Info*** and ***New*** → ***Next=Null***

c. 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 link lists merged successfully”

Else

Print “Not Enough space”

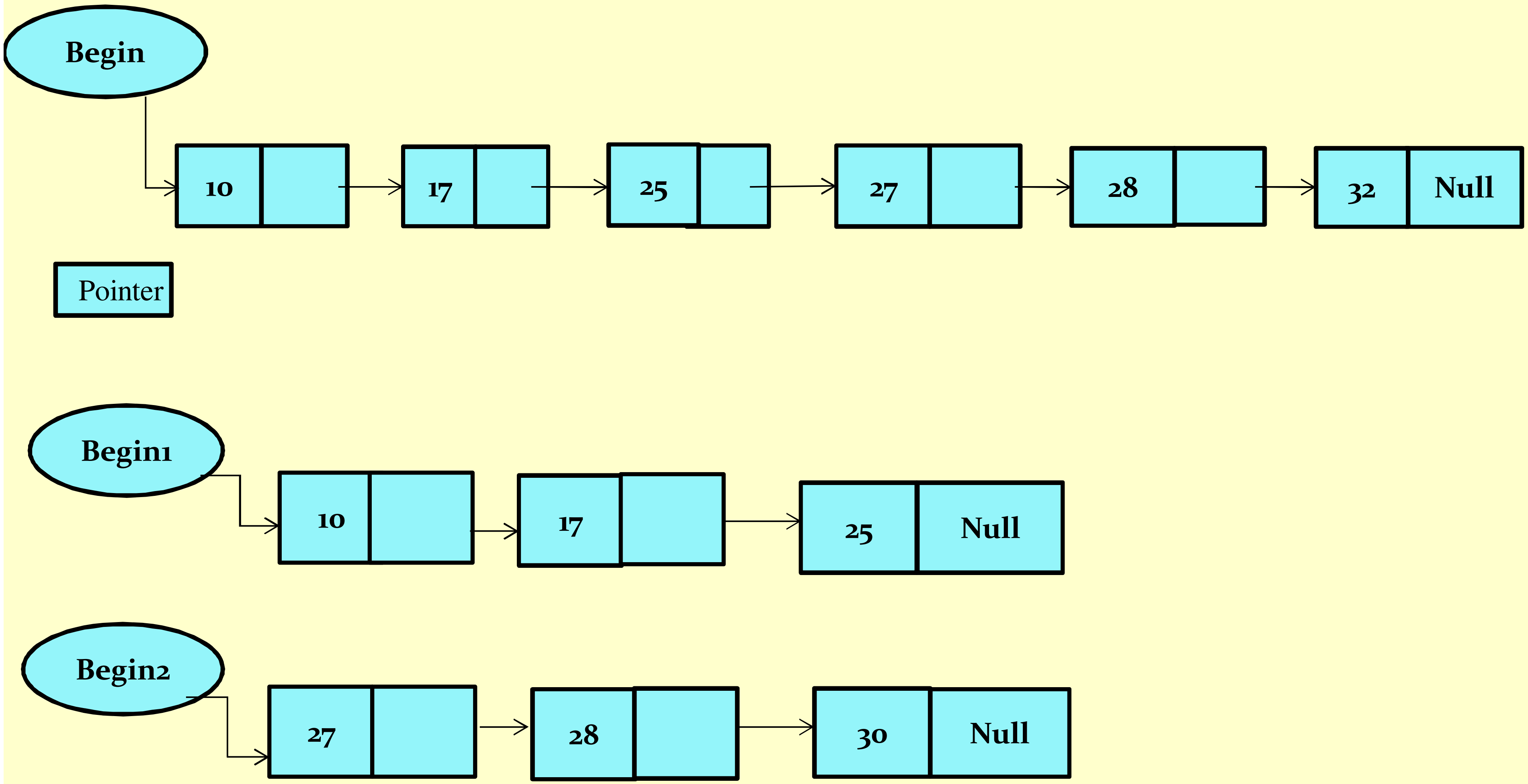
[End If]

Step 11: Exit

Splitting Two Lists

- Suppose we have a linked list which we want to split into lists.
- First we check total no. Of nodes then (N/2)th and (N/2+1)th Node.
- After finding these addresses we will store null in the next part of the (n/2)th node and address of (n/2+1)th node will be stored in the new list pointer variable begin2.
- Now our list divide into 2 parts n/2 and n-n/2 with list begin1 and begin2.

Splitted List1 And List2 .



Algorithm: *Split A Link List Into Two Link Lists.*

Step 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*** $\rightarrow$ ***Next***

Set ***i=i+1***

[End Loop]

Step 10: Set ***Begin2 = Pointer*** $\rightarrow$ ***Next*** And

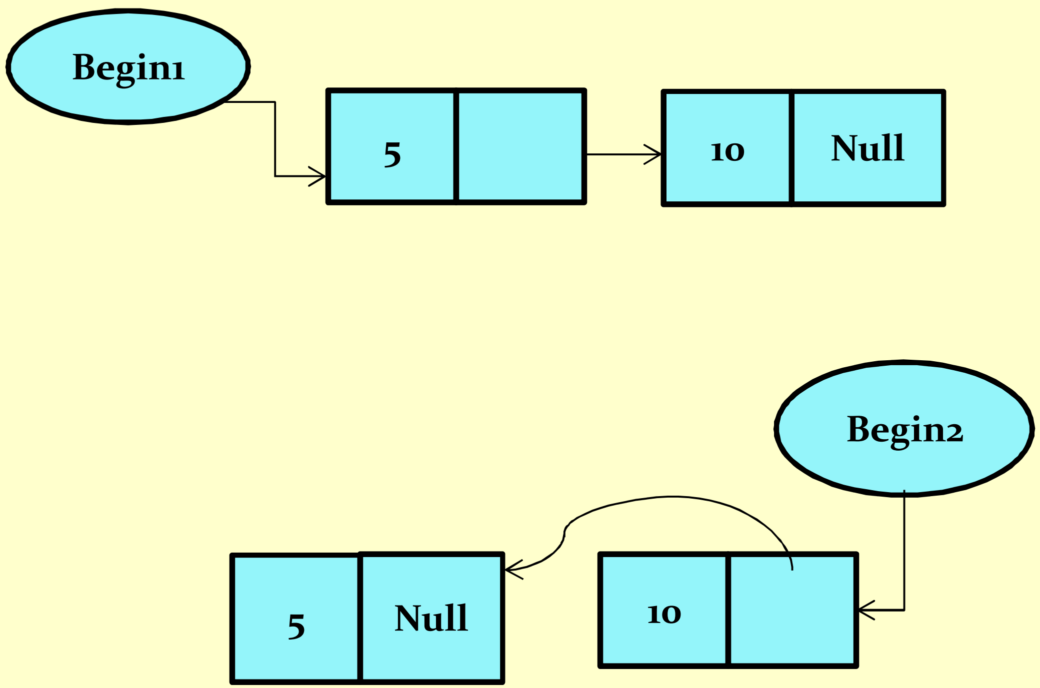
Pointer $\rightarrow$ ***Next = Null***

Step 11: Exit

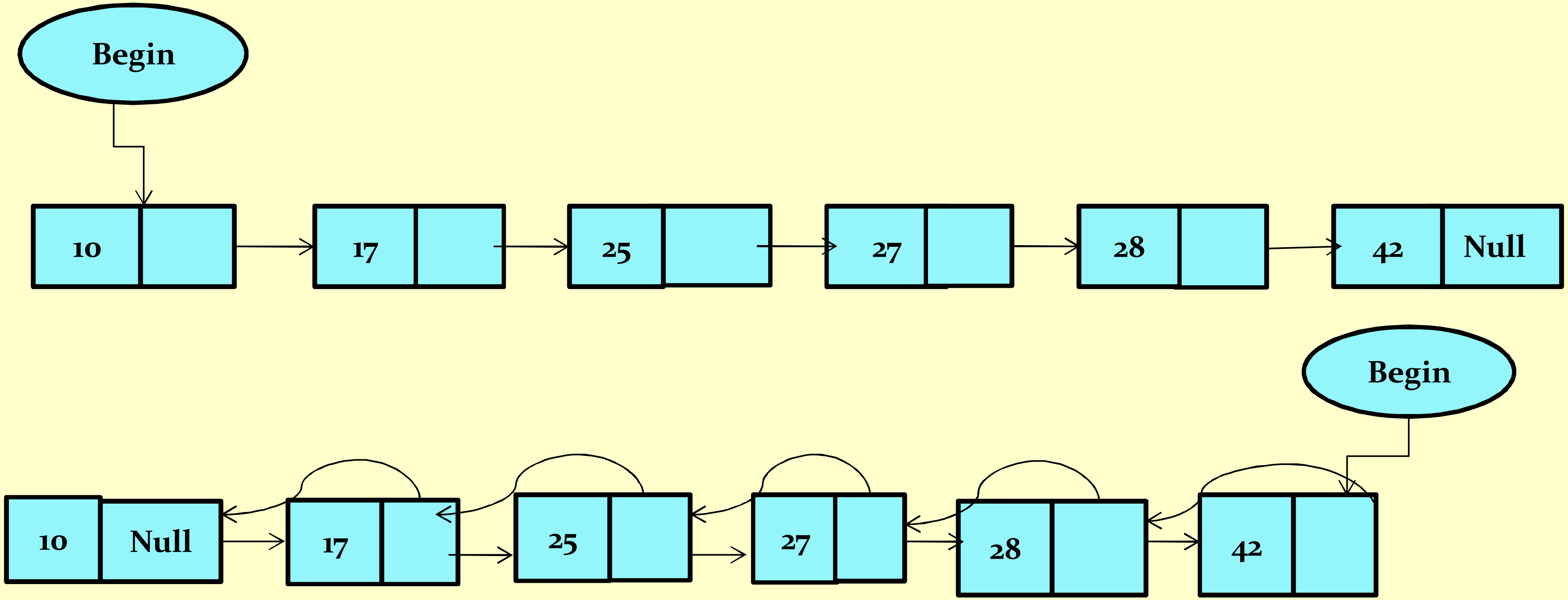
Reversing A One Way Linked List

- To reverse a linked list, we need to use three pointers variables.
- One pointer variable is used to store the address of current node.
- Second pointer variable will be used to store the address of next node.
- The third pointer variable will be used to store the address of next to next of current node.

Reversing A One Way Linked List



Reverse Link List With More Than Two Nodes



Algorithm: *Reverse The One Way Link List*

Step 1: ***If Begin = Null*** Then

Print: "No node is present in link list"

Exit

[End If]

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

Print: "link list is having only one node"

Exit

[End If]

Step 3: ***If Begin → Next != Null*** Then

Set ***Pointer1 = Begin***

Set ***Pointer2 = Begin →Next***

Set ***Pointer3 = Pointer2 → Next***

[End If]

Step 4: ***If Pointer3 = Null*** Then

Set ***Pointer2 → Next = Pointer1***

Set ***Pointer1 → Next = Null***

Set ***Begin = Pointer2***

Exit

[End If]

Step 5: Set ***Pointer1 → Next=Null***

Step 6: Repeat steps 7 to 10 while ***Pointer3 → Next!=Null***

Step 7: Set ***Pointer2 → Next=Pointer1***

Step 8: Set ***Pointer1=Pointer2***

Step 9: Set ***Pointer2=Pointer3***

Step 10: Set ***Pointer3=Pointer3 → Next***

[End Loop]

Step 11: Set ***Pointer2 →Next=Pointer1***

Step 12: Set ***Pointer3 →Next=Pointer2***

Step 13: Set ***Begin=Pointer3***

Step 14: Exit