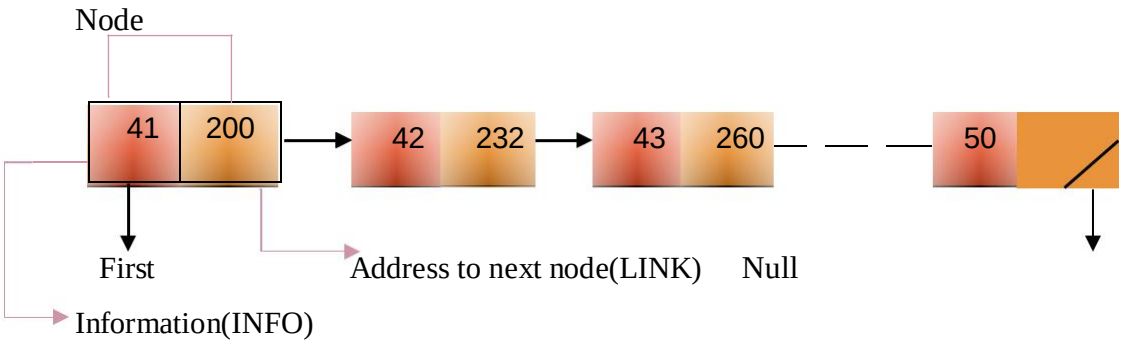
Chapter-4 Linked List

4.5 Linked List Presentation:

Link List is Non-Linear data structure because it doesn't contain the data sequentially. It is the collection of nodes which stores Data and Link. The following representation is known as one-way chain or singly linked linear list.



Node contains the information and the pointer which contains the address of next location. The link part of the last node in the list will contain a null character which indicate ending of the list.

First pointer always points to the address of the first information. In empty list First=Null, so before inserting and deleting first this condition should be check.

4.7 Availability list: -

It is a pool of free nodes used whenever a node is to be inserted in list. A free node is taken from the availability list. The deletion of the node from the list causes return to the availability list, this is the memory management.

The basic operations for linked list are: -

- To create a linked list
- Traversing a linked list
- Insert a new node at first,
- Insert a new node at end
- Insert a new node at middle
- Delete a node from beginning

- Delete a node from end,
- Delete a node from middle
- Searching element in linked list
- Count the number of nodes in linked list

4.7.1 Linked list creation algorithm:

Algorithm [To create a linked list]

Step 1: - [Initially list is empty or check whether any node exist or not]

If(first=Null)

Then write (“List is empty”)

First = null

Step 2: -[To createa new node]

[Remove free node from availability list]
[Only one node exist]
IF (first=Null)

Then

Temp avail

Avail link (avail)

Step 3: - [Assign a value to information part of node]

INFO (tmp) x

Step 4: - [Assign Null to the address part for node to indicate end of list]

Link [tmp] Null

Step 5: - [Assign address of first node to first variable]

First tmp

Step 6: - [Return at created node]

Return (first)

4.7.2 Inserting a node at the beginning of linked list :

Algorithm: insert (x, first)

Step 1: - Start

Step 2: - [Check for availability list underflow]

If(avail=Null)

Then write (“Availability stack underflow”)

Return (first)

Step 3: - [Check whether any node exist or not in list]

(A) [if there is not any node in the list then remove freed node from availability list]

(1) IF (first = null) Then Tmp avail

Availlink (avail)

(2) [assign data to node] Info (tmp) null

(3) [set link of node to null] Link(tmp) null

(B) [if there is any node available in list and then insert a new node]

(1) [assign data to node] Tmp avail

Availlink(avail)

(2) [assign data to node] Info (tmp) x

(3) [set link of code] Link (tmp) first

Step4: - [Assign the address of temporary pointer to first pointer]

First tmp

Step5: - Return

4.7.3 Insert new node at the end of the list (Append a list) :

Algorithm: insert(x, first)

Step 1: [check for availability list underflow]

if(avail=null)

then

write("availability stack is underflow ")

return(first)

Step 2: [create node and set the data and link portion of node]

tmp avail

avail link(avail)

[assign data to information part of node]

info(tmp) x

[set the link portion to null because it is last node in list]
link(tmp) null

Step 3: [if there is no node in list]

if(first=null)

then

first tmp

Step 4: [if there is any node in the list and put new node at the end]

A) [assign address of first node to tmp]
tmp first

B) [traverse the list til last node is
reached] repeat while(link(tmp)<>null)

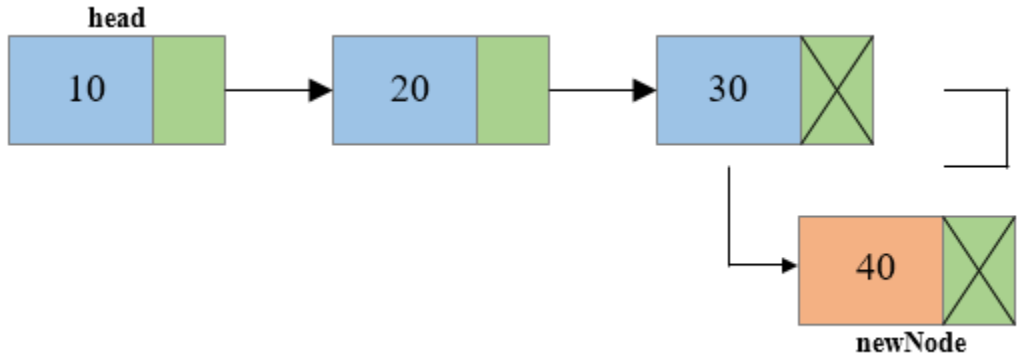
temp link(temp)

C) [set link of last no de to new node] link(temp)=tmp

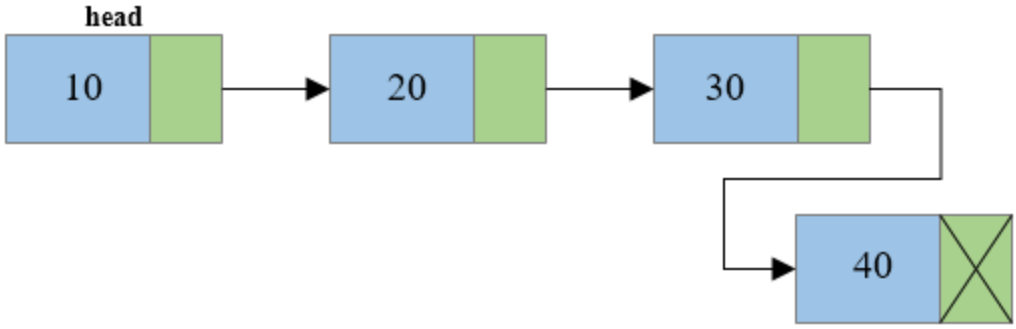
Step 5: Return

Steps to insert node at t he end of singly linked list :

- Create a new node and make sure that the address part of the new node points to null.



- Traverse to the last node of t he linked list and connect the last node of th e list with the new node, i.e. last node will now p oint to new node. (lastnode->next = newnode).



4.7.4 Insert a node at a ny location or specific location :

Algorithm: insloc (x, first)

Step 1: [check for availability list underflow]

if(avail=null)

then

("availability stack is underflow")

return(first)

Step 2: [create new node and assign data]

(1) [create node]

tmp avail avail

link [avail]

(2) [assign data to information portion of node]

info(tmp) x

Step 3: [read location]

read n

Step 4: [if there is any node and location is first(1)]

(1) [assign address of first node to tmp]

link(tmp) null

(2) [assign address of temporary pointer to first pointer to first pointer]

first tmp

Step 5: [if there is any node and insert a new node at location n]

(1) [assign address of first node to temp]

temp first

(2) [traverse the list until last node or meet at specific location

n] repeat while (link(tmp)<>null or link(tmp)=n) temp

link(temp)

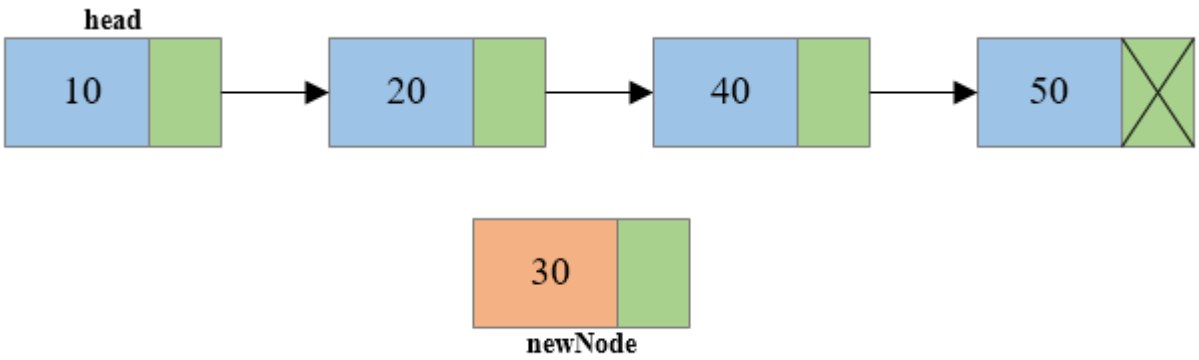
(3) [set link of this node to next node]

link(temp) temp

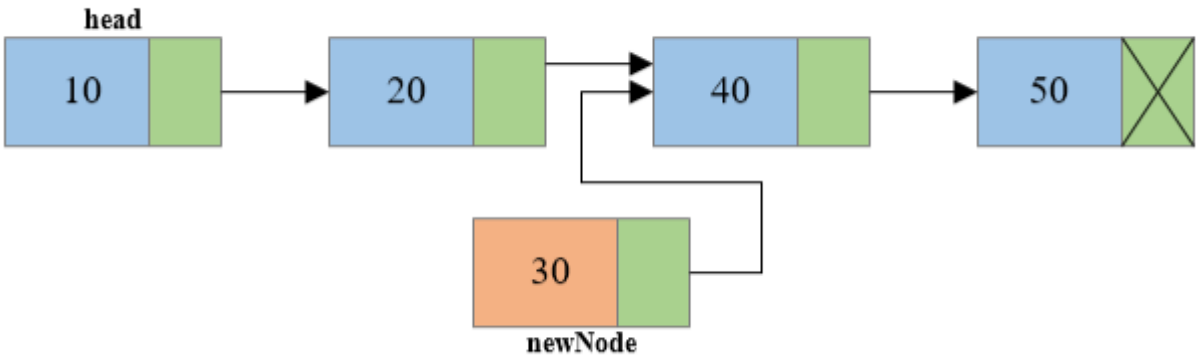
Step 6: return

Steps to insert node at t he middle of singly linked list

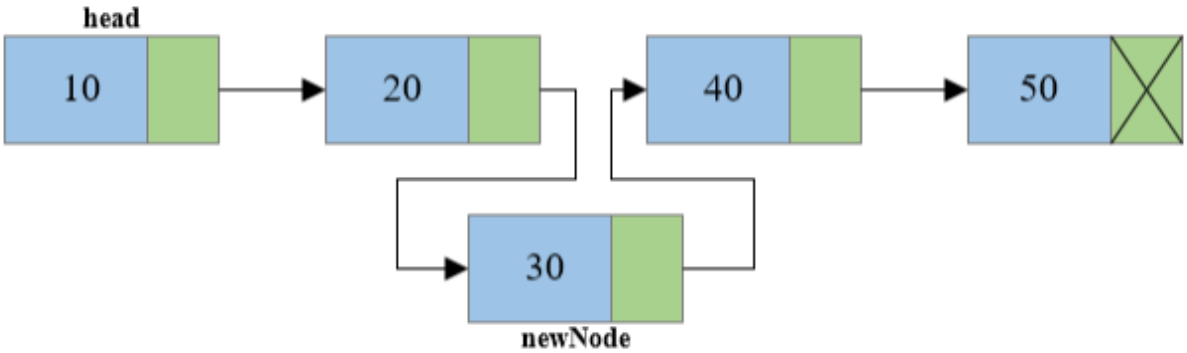
- Create a new node.



- Traverse to the n-1th position of the linked list and connect the new node with the n+1th node. Means the new node should also point to the same node that the n-1th n ode is pointing to. (newnode->next = temp->next where temp is the n-1th node).
- now at last connect the n-1th n ode with the new node i.e. the n-1th node wi ll now point to new node. (temp->next = newnode where temp is the n-1th node).



- now at last connect the n-1th n ode with the new node i.e. the n-1th node wi ll now point to new node. (temp->next = newnode where temp is the n-1th node).



4.7.5 Inserting a node to an ordered linear list :

Algorithm: insert (x, first, tmp, new)

Step 1: [check whether any node exist or not]

[if there is no node]

if (first=null)

then

tmp avail

avail link(avail)

(1) [assign data to node]

info(tmp)=x

(2) [set link of node to

null] link(tmp)=null

Step 2: [if there is only one node]

if(first=null or x<=info(first))

then

link(new)=first

first=new

Step 3: [if there is more than one node in a list]

repeat while(tmp<>null)

if(x>info(tmp) and x<=info(link(tmp)))

then

link (new) link(tmp)

link (tmp) new

else

tmp link(tmp)

Step 4: [finished]

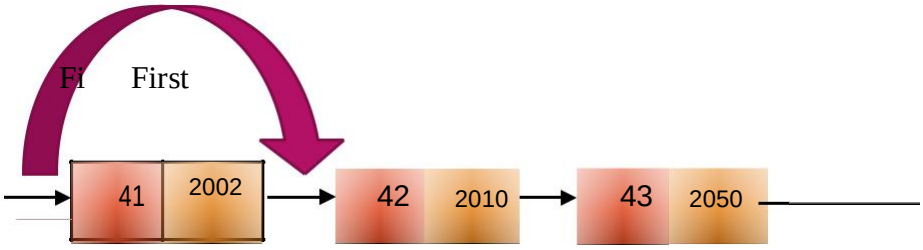
return

4.7.6 Deleting a first node from the linked list:

Before performing the deletion operation, it should be check that link list should not be empty otherwise display “List Underflow”.

If we want to delete first node from the list:

- Check for underflow
- Make link of start pointer to second node of the list
- Free the space associated with first node



Algorithm (x, first)

Step 1: - [Check for empty list]

If(first=Null)

Then write (“list is underflow”)

Return

Step 2: - [Delete first node]

A) [Only one node exists]

If(link(first)=Null)

Then free (first)

Firstnull

B) [if more than one node exist]

[Assign address to temporary

node] Tmp first

[Move first pointer to second node and free the

space] First link(tmp)

Free(tmp)

Step 3: - End

4.7.7 Deleting a node from END of linked list :

Algorithm: delete end (x,first)

Step 1: - Start

Step 2: - [Check for empty list]

If(first=Null)

Then write (“list is underflow”)

Return

Step 3: - [Assign address of first node to tmp]

Tmp first

Step4: - [Traverse the list until the last node is reached]

Repeat while(link(tmp)<>Null)

Temp link(tmp)

Step5: - [Assign link of tmp to a null which will terminate the list]

Link(tmp) Null

Free(temp)

Step 6: - End

4.7.8 Delete a node from list by specific location (position) :-

Algorithm: DELETE-POS (X , FIRST)

Step 1: [Read position]

Read N

Step 2: [Check for empty list]

If (first = NULL)

Then

Write(" Link list is empty and under-flow ")

Return

Step 3: [Node is delete at position N]

D) [Assign address of first node to tmp]
tmp first

E) [Traverse the list till specific location is reached]
Repeat for I : 1,2 to N

temp link(tmp)
link(tmp) link(temp)
free(temp)

Step 4: [Finished]

Return

4.7.9 Searching a node in linked list :-

Algorithm: SEARCH (FIRST , X)

- FIRST is a pointer which [points to the first node in the list.
- X is the element which we want to search in the list.

Step 1: [Initialize]

FLAG 0

Step 2: [Check for empty list]

If FIRST = NULL

Write (" List is empty node not found ")

Step 3: [Search entire list]

SAVE FIRST

Repeat while SAVE != NULL

If INFO(SAVE) = X

Then FLAG 1

SAVE LINK(SAVE)

Else

SAVE LINK(SAVE)

Step 4: [Node found?]

If FLAG = 1

Then

Write (" Node found ")

Else

Write (" Node not found ")

Step 5: [Finished]

Return

4.7.10 Count number of nodes in linked list :-

Algorithm: COUNT (FIRST)

This function counts number of nodes in the list.

FIRST is a pointer which contains address of first node in the list.

Step 1: [Initialize]

Count 0

SAVE FIRST

Step 2: [Process the list until end of the list]

Repeat while SAVE != NULL

Count Count + 1

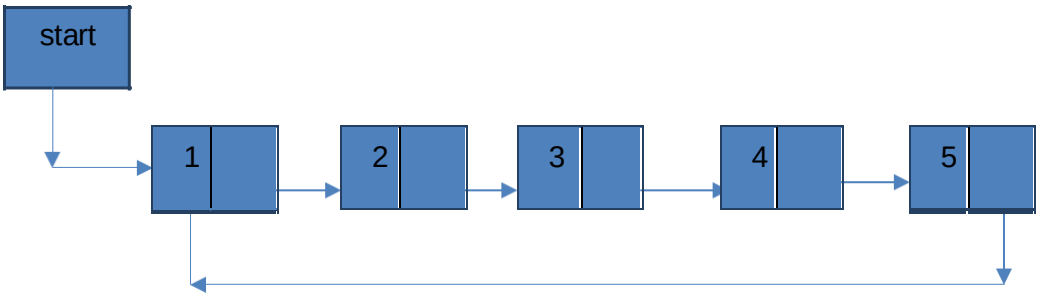
SAVE LINK(SAVE)

Step 3: [Finished]

Return (Count)

4.8 Circular Linked List

Circular linked list *is a linked list where all nodes are connected to form a circle. There isnoNULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.*



No	DATA	NEXT
1	H	4
2	NULL	
3	NULL	
4	E	7
5	NULL	
6	NULL	
7	L	8
8	L	10
9	NULL	
10	O	1

Advantages of Using Circular Linked List:

- Some problems are circular and a circular data structure would be more natural when used to represent it
- The entire list can be traversed starting from any node (traverse means visit every node just once)

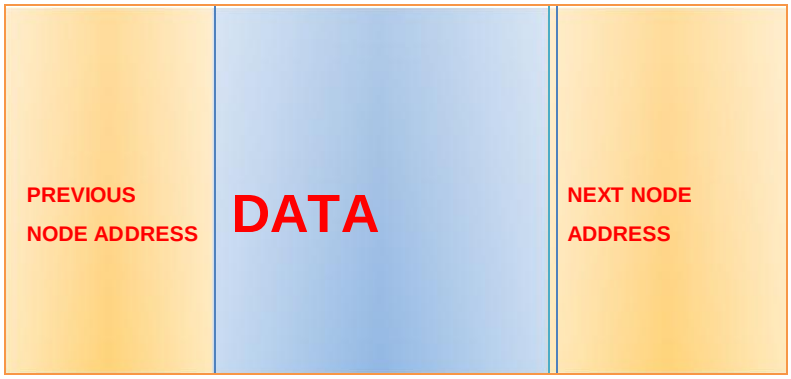
4.9 DIFFERENCE BETWEEN CIRCULAR AND LINEAR LINKED LIST

LINEAR LINKED LIST	CIRCULAR LINKED LIST
In this the nodes are in sequence till end.	In this the nodes are never ending and sequence is change from the last unique node.
The next pointer of the last node is null	The next pointer of the last node is first node

We can only set first node as starting for traversing the linked list.	We can set any node as starting node and still ca traverse whole linked list

4.10 DOUBLY LINKED LIST

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer). A sample node in a doubly linked list is shown in the figure.



OPERATIONS ON DOUBLY LINKED LIST:

4.10.1 INSERTION OF NEW NODE IN THE BEGINNING OF THE LINKED LIST

ALGORITHM:

Step 1: if ptr = null

write overflow
go to step 9
[end of if]

Step 2: set new_node = ptr

Step 3: set ptr = ptr -> next

Step 4: set new_node -> data = val

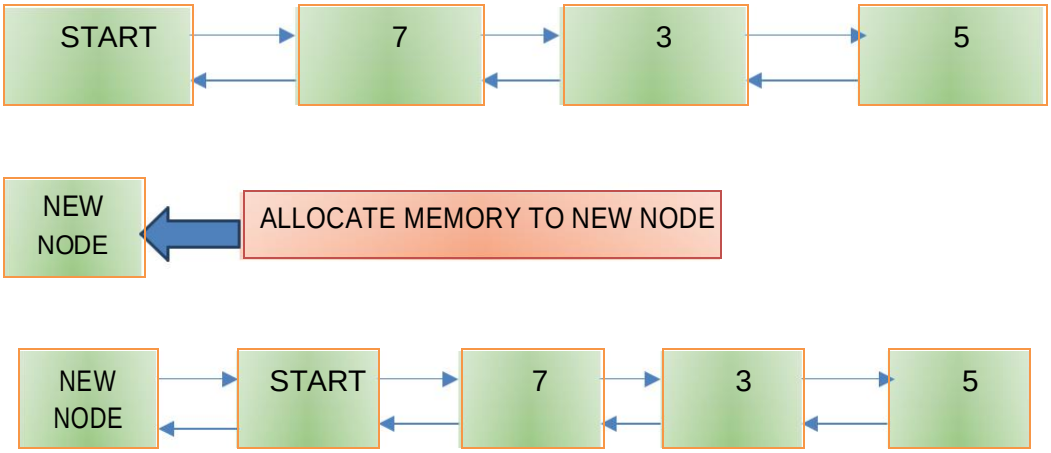
Step 5: set new_node -> prev = null

Step 6: set new_node -> next = start

Step 7: set head -> prev = new_node

Step 8: set head = new_node

Step 9: exit



4.10.2 INSERTION OF NEW NODE IN THE END OF THE LINKED LIST :

ALGORITHM:

Step 1: IF ptr = NULL

Write OVERFLOW
Go to Step 11
[END OF IF]

Step 2: SET NEW_NODE = ptr

Step 3: SET ptr = ptr -> NEXT

Step 4: SET NEW_NODE -> DATA = VAL

Step 5: SET NEW_NODE ->NEXT = NULL

Step 6: SET TEMP = START

Step 7: Repeat Step 8 while TEMP ->NEXT != NULL

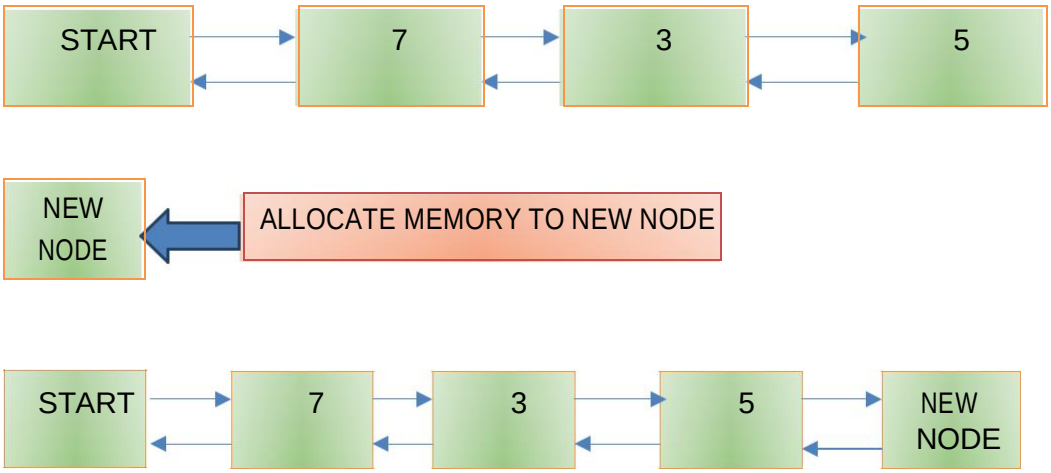
Step 8: SET TEMP = TEMP -> NEXT

[END OF LOOP]

Step 9: SET TEMP -> NEXT = NEW_NODE

Step 10: SET NEW_NODE -> PREV = TEMP

Step 11: EXIT



4.10.3 INSERTION OF NEW NODE AFTER GIVEN NODE IN THE LINKED LIST:

ALGORITHM

STEP 1 : if ptr = null

write overflow

goto step 12

end of if

Step 2 : set new_node = ptr

Step 3 : new_node → data = val

Step 4 : set temp = head

Step 5 : set I = 0

Step 6 : repeat step 5 and 6 until i

Step 7 : temp = TEMP→NEXT

Step 8 : if temp = null

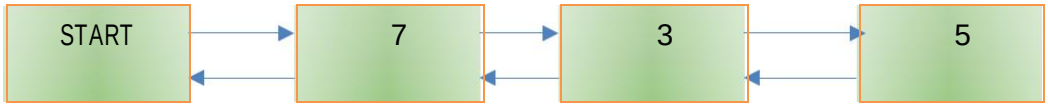
```
write "desired node not present"
    goto step 12
    end of if
end of loop
```

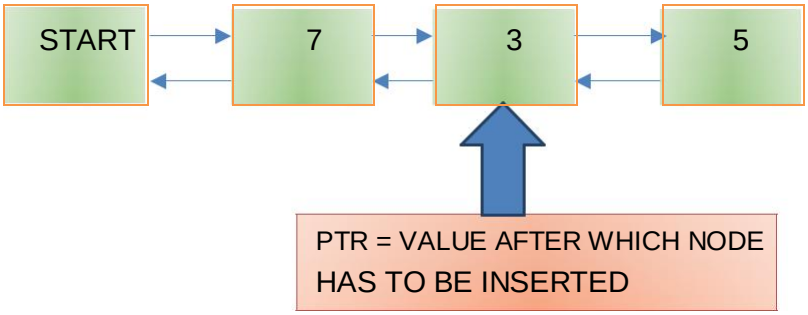
Step 9 : ptr→next = temp→next

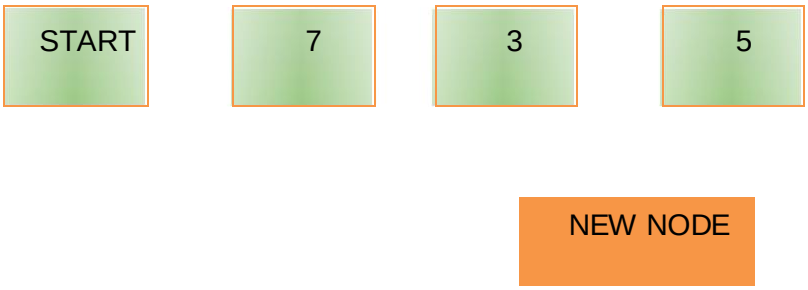
Step 10 : temp→next = ptr

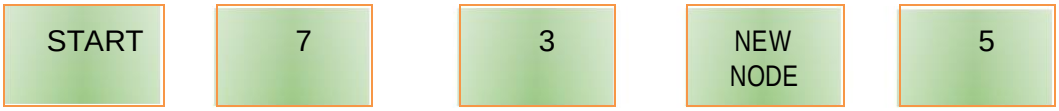
Step 11 : set ptr = new_node

Step 12 : exit









4.10.4 DELETING THE FIRST NODE OF DOUBLY LINKED LIST

ALGORITHM:

STEP 1: IF HEAD = NULL

WRITE UNDERFLOW
GOTO STEP 6

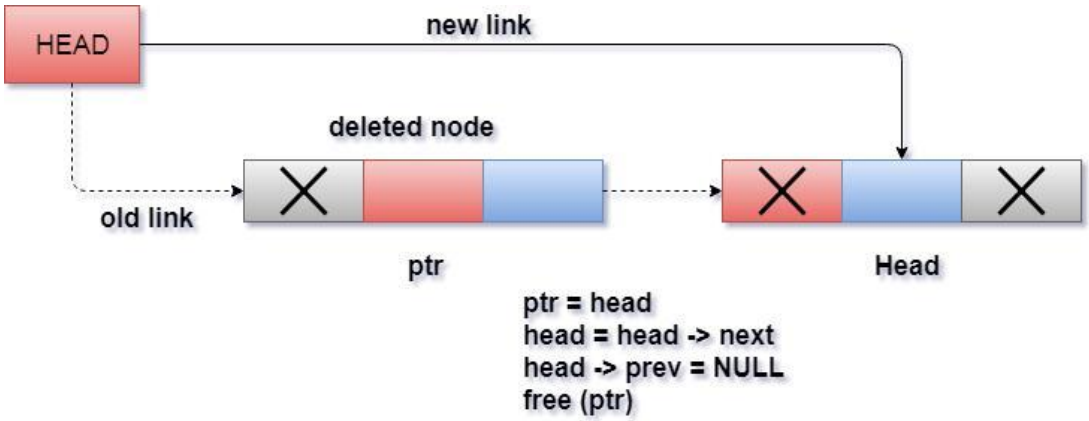
STEP 2: SET PTR = HEAD

STEP 3: SET HEAD = HEAD→NEXT

STEP 4: SET HEAD→PREV = NULL

STEP 5: FREE PTR

STEP 6: EXIT



Deletion in doubly linked list from beginning

4.10.5 DELETE THE LAST NODE FROM DOUBLY LINKED LIST :

ALGORITHM:

Step 1: IF HEAD = NULL

Write UNDERFLOW
Go to Step 7
[END OF IF]

Step 2: SET TEMP = HEAD

Step 3: REPEAT STEP 4 WHILE TEMP->NEXT != NULL

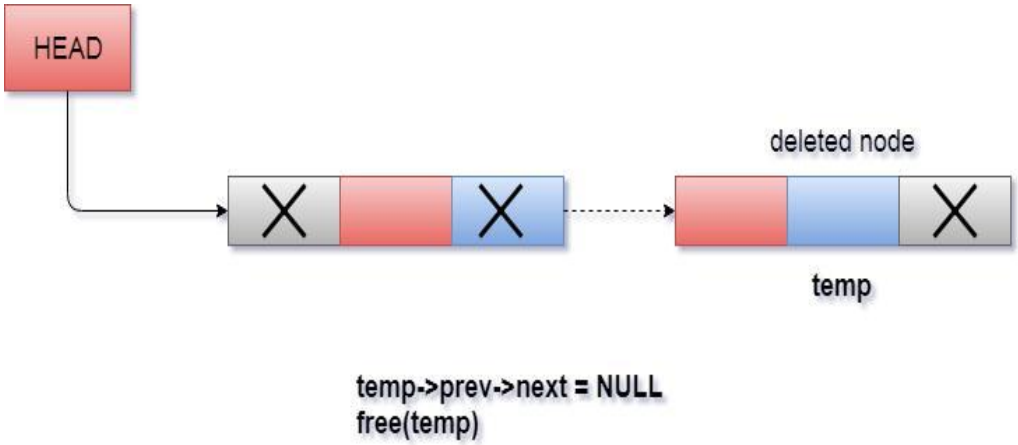
Step 4: SET TEMP = TEMP->NEXT

[END OF LOOP]

Step 5: SET TEMP ->PREV-> NEXT = NULL

Step 6: FREE TEMP

Step 7: EXIT



Deletion in doubly linked list at the end

7. SEARCHING A NODE IN LINKED LIST

ALGORITHM:

Step 1: IF HEAD == NULL

WRITE "UNDERFLOW"
GOTO STEP 8
[END OF IF]

Step 2: Set PTR = HEAD

Step 3: Set i = 0

Step 4: Repeat step 5 to 7 while PTR != NULL

Step 5: IF PTR→data = item

return i

[END OF IF]

Step 6: i = i + 1

Step 7: PTR = PTR→next

Step 8: Exit

8. COUNT NUMBER OF NODES IN LINKED LIST

ALGORITHM:

Step 1: [INITIALIZE]

Counter = 0

Step 2: POINTER = HEAD

Step 3: repeat till POINTER->NEXT != HEAD

Step4 : Counter = Counter + 1

Step 5:exit

Application of Doubly linked lists

There are various application of doubly linked list in the real world. Some of them can be listed as:

- Doubly linked list can be used in navigation systems where both front and back navigation is required.
- It is used by browsers to implement backward and forward navigation of visited web pages i.e. **back** and **forward** button.
- It is also used by various application to implement **Undo** and **Redo** functionality.
- It can also be used to represent deck of cards in games.
- It is also used to represent various states of a game.