Create a new node

Step1: if first=NULL

Write linked list is empty. First= NULL.

Step2: [create a new node]

1. [create node] New_node<-avail Avail<-link[avail]
2. [assign data to information part of node] Info(new_node)<-x

Step3: [assign NULL link to end of the list] Link[new_node]<-NULL

Step4: [assign add of first node to first variable] First <- new_node.

Step5: finished

- Exit

Insert a new node at beginning:

Step1: [check for new node]

If (avail == Null) Then

Print “availability stack is underflow”

Exit

Step2:[if there is no node in linked list]

if first=null New_node<- avail

Avail<-link(avail)

set info (new_node)<- xset

link (new_node)<- Null

Step3: [if there is node in link list]

if (first = null) new_node <- avail Avail<- link(avail)

info (new_node)<-x

link (new_node)<-first

set first<- new_node

Step4:exit

Insert a node at a last position of link-list:

Step1: [check for new node]

If avail = null

Then

Print “availability stack under flow” Exit

Step2: [create node and set data and link portion]

1.New_node<-avail Avail=link(avail)

2.set info (new_node)<-x

3.set link (new_node)<- null

Step3**:** [if there is no node in list] If(first==null)

Then First<- new_node

Step4: [if node is available in link-list]

1.assign add of first node to ptr ptr<- first

2.[traveling]

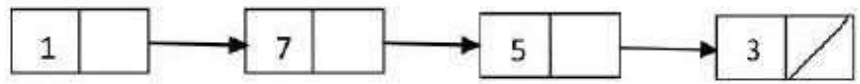
While (link (ptr)! = null) ptr<- link(ptr)

3.link (ptr)<-new_node

- Step5: Exit

FIRST

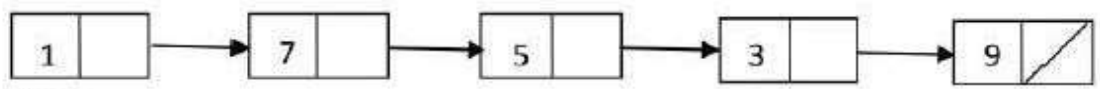
We take a **PTR** which will initially point to FIRST.



FIRST, PTR

FIRST **PTR**

Move **PTR** so that it points to the last node of the list.



FIRST

Add the new node after the node pointed by **PTR**. This is done by storing the address of the new node in the **LINK** part of **PTR**.

Insert a node in given position

Step1:[check availability list underflow]
If avail=null

Write list is underflow Exit

Step2: [create a new node]

1.[create node] New_node<-avail
Avail<-link[avail]

2.[assign data to information part of node] Info(new_node)<-x

Step3: [Read location] Read N

Step4: [set pointer ptr]

Ptr<-first Preptr<-ptr

Step5: [traverse the list and meet specific location N]

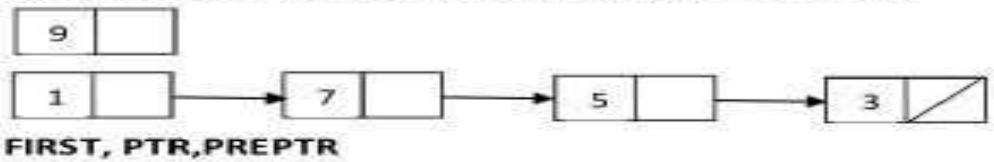
Repeat while (info [preptr]! = N)

preptr<-ptr ptr<- Link(ptr)

Step6: [insert node at location N]
Link(preptr)<- new_node
Link(new_node)<-ptr

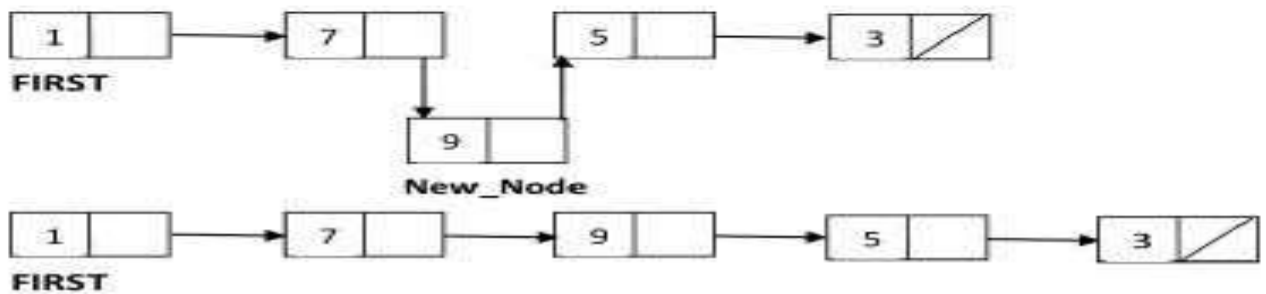
Step7: Exit

Take two pointer variable **PTR** and **PREPTR** and initialize them with **FIRST** so that **FIRST** ,**PTR** and **PREPTR** point to the first node of the list.

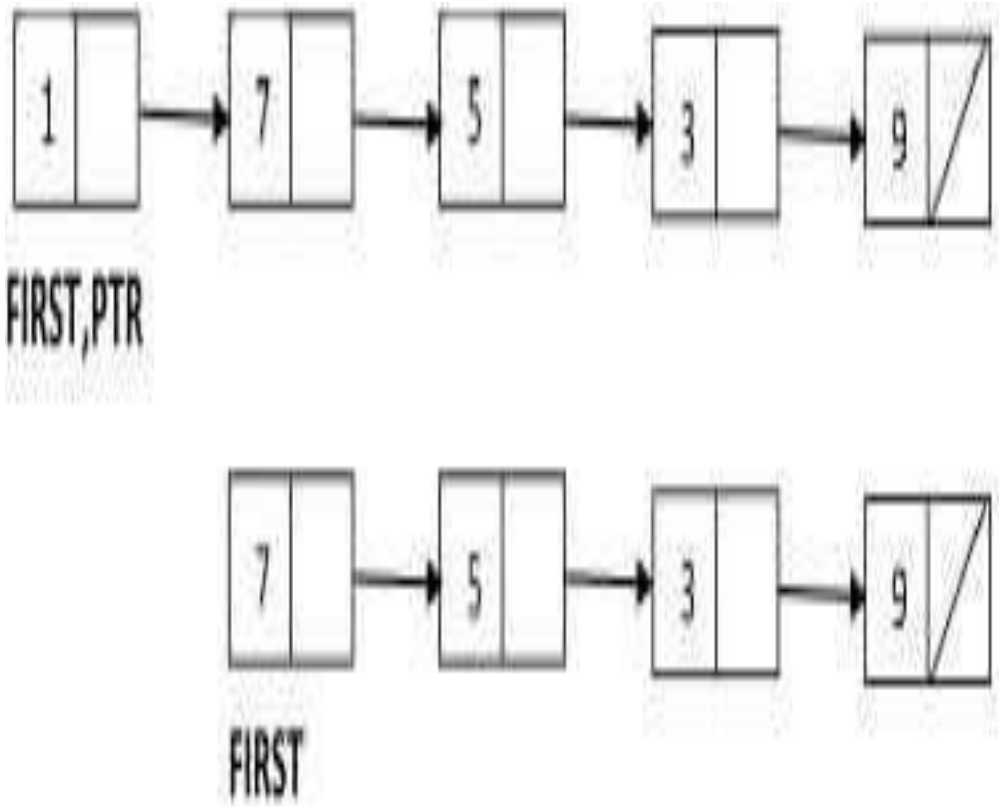


Move **PTR** and **PREPTR** until the data part of the **PREPTR** = **Value** of the node after which insertion has to be done. **PREPTR** will always point to the node just before **PTR**.

Add new node in between the nodes pointed by **PREPTR** and **PTR**.



Deleting 1st node from the list



- Step1: if first== null
- Then linked list is empty Exit
- Step2: [delete 1st node]
- if(link(first)==null) free(first)
- Step3: [more than one node exist] ptr=first
- first=link(ptr) free(ptr)
- Step4: exit

Deleting last node from the list:

Step1: if first== null

Then linked list is empty Exit

Step2: ptr=first

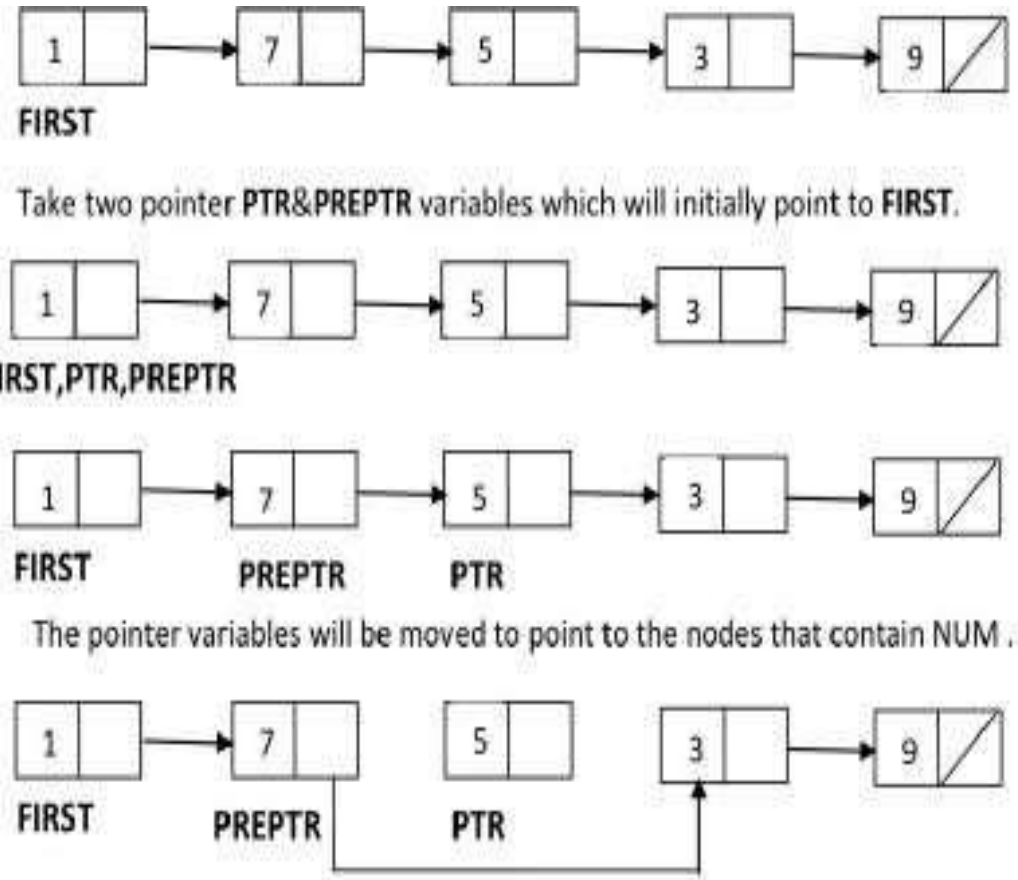
Step3: [traverse up to last node] repeat while(link(ptr)!=null)

preptr=ptr ptr=link(ptr)

Step4: link(preptr)=null Free(ptr)

Step5: exit

Deleting a given node from the list:



Step1: if first== null

Then linked list is empty Exit

Step2: read data of node N to be deleted

Step3: ptr<-first

Preptr<-ptr

Step4: repeat while(info(ptr)!=N)

Preptr<-ptr

Ptr<-link(ptr)

Step5: link(preptr)=link(ptr) Free (ptr)

Step6: exit

Circular link-list:

- A circular link list is a list in which last node is connected with first node. Therefore, first node and last node connected to each other. This concept is known as circular linked list. To delete the last node, we can use one special node called as head node.