Insertion Operations of linkedlist

Insertion at Last Position

Steps for Insertion at last position

In [ ]:
```
#Insertion at the last position
1.Create a node
2.use while loop to go till tail node
3.tail.next=newNode
4.return head
```

Implementation

In [2]:
```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None

def printll(head):
    while head is not None:
        print(str(head.data)+"-->",end="")
        head=head.next
    print("None")


def insert_at_last(head,data):
    curr=head
    while curr.next is not None:
        curr=curr.next
    newNode=Node(data)
    curr.next=newNode
    return head


x=Node(10)
y=Node(20)
z=Node(30)
a=Node(40)
y.next=x
x.next=z
z.next=a
printll(y)

ll3 = insert_at_last(y,500)
printll(ll3)
```

```
20-->10-->30-->40-->None
20-->10-->30-->40-->500-->None
```

Deletions Operations of linkedlist

Deletion from Beginning

Implementation

In [4]:
```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
        self.prev=None

def printll(head):
    while head is not None:
        print(str(head.data)+"-->",end="")
        head=head.next
    print("None")


def deletion_at_begin(head):
    head=head.next
    return head



x=Node(10)
y=Node(20)
z=Node(30)
a=Node(40)
y.next=x
x.next=z
z.next=a
printll(y)
ll4=deletion_at_begin(y)
printll(ll4)
```

```
20-->10-->30-->40-->None
10-->30-->40-->None
```

Deletion from the Given Position

Implementation

In [6]:
```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
        self.prev=None

def printll(head):
    while head is not None:
        print(str(head.data)+"-->",end="")
        head=head.next
    print("None")


def deletion_at_pos(head,pos):
    prev=None
    curr=head
    i=0
    while i<pos:
        prev=curr
        curr=curr.next
        i=i+1
    prev.next=curr.next
    #prev.next=prev.next.next
    return head


x=Node(10)
y=Node(20)
z=Node(30)
a=Node(40)
y.next=x
x.next=z
z.next=a
printll(y)
ll4=deletion_at_pos(y,2)
printll(ll4)
```

```
20-->10-->30-->40-->None
20-->10-->40-->None
```

Deletion from last position

Implementation

In [7]:
```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
        self.prev=None
def printll(head):
    while head is not None:
        print(str(head.data)+"-->",end="")
        head=head.next
    print("None")
def deletion_at_end(head):
    curr=head
    prev=None
    while curr.next is not None:
        prev=curr
        curr=curr.next
    prev.next=None
    return head


x=Node(10)
y=Node(20)
z=Node(30)
a=Node(40)
y.next=x
x.next=z
z.next=a
printll(y)
ll4=deletion_at_end(y)
printll(ll4)
```

```
20-->10-->30-->40-->None
20-->10-->30-->None
```

Applications of Linkedlist

In [ ]:
```
Applications of linked list in computer science:

    --> Implementation of stacks and queues
    --> Implementation of graphs: Adjacency list representation of graphs is the most popular which uses a linked list to
        store adjacent vertices.
    --> Dynamic memory allocation: We use a linked list of free blocks.
    --> Maintaining a directory of names
    --> Performing arithmetic operations on long integers
    --> Manipulation of polynomials by storing constants in the node of the linked list representing sparse matrices

Applications of linked list in the real world:

    --> Image viewer – Previous and next images are linked and can be accessed by the next and previous buttons.
    --> Previous and next page in a web browser – We can access the previous and next URL searched in a web browser by
        pressing the back and next buttons since they are linked as a linked list.
    --> Music Player – Songs in the music player are linked to the previous and next songs. So you can play songs either
        from starting or ending of the list.
    --> GPS navigation systems- Linked lists can be used to store and manage a list of locations and routes, allowing
        users to easily navigate to their desired destination.
    --> Robotics- Linked lists can be used to implement control systems for robots, allowing them to navigate and interact
        with their environment.
```

Variations/Types of Linkedlist

In [ ]:
```
There are four key types of linked lists:

    Singly linked lists --> Move in only Forward Direction
    Doubly linked lists --> Move in Forward as well as in Backward direction
    Circular linked lists --> Move in forward  and last node is connected with first node.
    Circular doubly linked lists --> Move in forward as well as backward direction and last node is connected with first node.
```

Stacks

In [ ]:
```
--> Stack is a linear Data structure. It follows the principle of LIFO(Last in first out):
--> Whatever is coming last will move out first.
--> In Stack Insertion and deletion in stack will be done from one end only and that end is known as Top of stack
```

Terminologies/Operations of Stack

In [ ]:
```
Terminology/Operations of stack:

    1.push() --> Insert an element in stack  --> Insertion will be done from Top of Stack
    2.pop() --> Delete an elememnt from stack --> Deletion will be done from Top of Stack
    3.peek() --> return the top value of stack
```

How to Implement a Stack

In [ ]:
```
Stack can be implemented by Two Ways:
    1. With linkedlist
    2. With Array
```

Implementation of stack using array

In [8]:
```
#Implementation of stack using array
class Stack:
    def __init__(self):
        self.array=[]

    def push(self,data):
        self.array.append(data)

    def pop(self):
        if len(self.array)==0:
            return "Stack is Empty"
        return self.array.pop()

    def peek(self):
        if len(self.array)==0:
            return "Stack is Empty"
        return self.array[-1]
c=Stack()
c.push(1)
c.push(2)
c.push(3)
print(c.pop())#3
print(c.pop()) #2
print(c.peek()) #1
c.pop()
c.pop()
```

```
3
2
1
```

Out[8]:
```
'Stack is Empty'
```

Implementation of Stack Using Linkedlist

Steps for Implementation of Stack Using Linkedlist

In [ ]:
```
Initlize a Variable count=0

for Push :
    1.Create  a node
    2.Make Connections
    3.Change head
    4.Count=count+1

for pop:
    1.head=head.next
    2.count=count-1

for peek:
    1.return head.data

for checking stack is empty or not:
    if count==0:
        return True
```

Implementation of Stack

In [12]:
```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None

class Stack:
    def __init__(self):
        self.head=None
        self.count=0

    def push(self,data):
        newNode=Node(data)
        newNode.next=self.head
        self.count+=1
        self.head=newNode

    def pop(self):
        if self.count==0:
            return "Stack is empty we cannot perform pop operation"
        ele = self.head.data
        self.head=self.head.next
        self.count-=1
        return ele

    def peek(self):
        if self.count==0:
            return "Stack is Empty we cannot perform peek operations"
        return self.head.data


c=Stack()
c.push(1)
c.push(2)
c.push(3)
print(c.pop())#3
print(c.pop()) #2
print(c.peek()) #1
c.pop()
c.pop()
```

```
3
2
1
```

Out[12]:
```
'Stack is empty we cannot perform pop operation'
```

Applications of Stacks

In [ ]:
```
--> Evaluation of Airthmetic expression --> 1+2//22+22/22
--> Reverse the element
--> BackTracking algorithms
--> Stack memeory management and funtion calling
--> Tower of Hanoi --> Recusrion
--> Balanced Parenthesis
--> Expression Evalution --> Prefix expression  -->+ab
                         --> infix exprssion     -->a+b
                         --> Postfix expression --> ab+
--> Back Button Implementation
--> Stack is used for evaluating expression with operands and operations.
--> Matching tags in HTML and XML
--> Undo function in any text editor.
--> Infix to Postfix conversion.
--> Stacks are used for backtracking and parenthesis matching.
--> Stacks are used for conversion of one arithmetic notation to another arithmetic notation.
--> Stacks are useful for function calls, storing the activation records and deleting them after returning from the function.
    It is very useful in processing the function calls.
--> Stacks help in reversing any set of data or strings.


Real Time Application of Stacks:
    --> CD/DVD stand.
    --> Stack of books in a book shop.
    --> Undo and Redo mechanism in text editors.
    --> The history of a web browser is stored in the form of a stack.
    --> Call logs, E-mails, and Google photos in any gallery are also stored in form of a stack.
    --> YouTube downloads and Notifications are also shown in LIFO format(the latest appears first ).
```