SinglyLinkedList

May 15, 2025

[19]:
```
class SListNode:
    def __init__(self,data=0,next=None):
        self.Data=data
        self.Next=next

# node1=SListNode(10,None)
# node2=SListNode(20,None)
# node3=SListNode()

# # print('node2: ', node2)
# # print('Before: ', node1.Next)
# node1.Next=node2
# # print('After: ',node1.Next)
# node2.Next=node3
# head=node1

# print(head)
# print(head.Data,head.Next)
# print(head.Next.Data,head.Next.Next)
```

[8]:
```
class SListNode:
    def __init__(self,data=0,next=None):
        self.Data=data
        self.Next=next

head=SListNode(1,None)
last_node=head
for i in range(2,101):
    last_node.Next=SListNode(i,None)
    last_node=last_node.Next
tmp=head
while tmp:
    print(tmp.Data,end='->')
    tmp=tmp.Next
print()

tmp=head
```

```
for i in range(49):
    tmp=tmp.Next
print(tmp.Data)
```

```
1->2->3->4->5->6->7->8->9->10->11->12->13->14->15->16->17->18->19->20->21->22->2
3->24->25->26->27->28->29->30->31->32->33->34->35->36->37->38->39->40->41->42->4
3->44->45->46->47->48->49->50->51->52->53->54->55->56->57->58->59->60->61->62->6
3->64->65->66->67->68->69->70->71->72->73->74->75->76->77->78->79->80->81->82->8
3->84->85->86->87->88->89->90->91->92->93->94->95->96->97->98->99->100->
50
```

[17]:
```
class SList:
    class SListNode:
        def __init__(self,data=0,next=None):
            self.Data=data
            self.Next=next
    def __init__(self):
        self.head=None

    def show(self):
        tmp=self.head
        if tmp==None:
            print('list is empty')
        else:
            while tmp:
                print(tmp.Data,end='->')
                tmp=tmp.Next
            print()


    def is_empty(self):
        return self.head==None

    # def add2first(self,data):
    #     if self.is_empty(): # if self.head==None:
    #         self.head=self.SListNode(data,None)
    #     else:
    #         tmp=self.SListNode(data,None)
    #         tmp.Next=self.head
    #         self.head=tmp

    # def add2first(self,data):
    #     if self.is_empty():
    #         self.head=self.SListNode(data,None)
    #     else:
    #         self.head=self.SListNode(data,self.head)
```

```
    def add2first_by_value(self,data):
        self.head=self.SListNode(data,self.head)

    # def add2first_by_ref(self,node):
    #     if self.is_empty():
    #         node.Next=None
    #         self.head=node
    #     else:
    #         node.Next=self.head
    #         self.head=node
    def add2first_by_ref(self,node):
        node.Next=self.head
        self.head=node

    def add2first(self,inp):
        if type(inp)==type(self.SListNode()):
            inp.Next=self.head
            self.head=inp
        else:
            self.head=self.SListNode(inp,self.head)

    def Add2Last(self,x):
        if self.is_empty():
            self.Add2First(x)
        else:
            tmp=self.head
            while tmp.Next!=None:
                tmp=tmp.Next
            last_node=tmp
            last_node.Next=self.SinglyLinkedListNode(x,None)

    def AddNodeAfterNodeByValue(self,x,y):
        if self.head==None:
            return -1
        else:
            tmp=self.head
            while tmp!=None:
                if tmp.Data==x:
                    break
                else:
                    tmp=tmp.Next
            else:
                return 0
            tmp.Next=self.SinglyLinkedListNode(y,tmp.Next)
            self.size+=1
            return 1
```

```
    def RemoveFirstNode(self):
        if not self.is_empty():
            tmp=self.head
            self.head=self.head.Next
            return 1,tmp
        else:
            return 0,None
    def RemoveLastNode(self):
        if not self.is_empty():
            if self.head.Next==None:
                 return 1,self.RemoveFirstNode()[1]
            before_node=self.head
            while before_node.Next.Next:
                before_node=before_node.Next
            tmp=before_node.Next
            before_node.Next=None
            return 1,tmp
        else:
            return 0,None
    def RemoveAfterNode(self,x):
        before_node=self.head
        while before_node:
            if before_node.Data==x:
                break
            before_node=before_node.Next
        if before_node:
            if before_node.Next:
                tmp=before_node.Next
                before_node.Next=before_node.Next.Next #before_node.Next=tmp.
↪Next
                return 1,tmp
            else:
                return 0,None    # last node
        else:
            return 0,None  #  self.head==None   there is not a node with␣
↪data==x
    def RemoveAfterNodeByRef(self,node):
        before_node=self.head
        while before_node:
            if before_node==node:
                break
            before_node=before_node.Next
        if before_node:
            if before_node.Next:
                tmp=before_node.Next
                before_node.Next=before_node.Next.Next #before_node.Next=tmp.
↪Next
```

```
                return 1,tmp
            else:
                return 0,None
        else:
            return 0,None
    def RemoveNodeByVal(self,x):
        delete_node=self.head
        while delete_node:
            if delete_node.Data==x:
                break
            delete_node=delete_node.Next
        if delete_node:
            if delete_node==self.head:
                return self.RemoveFirstNode()
            before_node=self.head
            while before_node:
                if before_node.Next==delete_node:
                    break
                before_node=before_node.Next
            before_node.Next=delete_node.Next
            return 1,delete_node
        else:
            return 0,None

    def RemoveNodeByRef(self,node):
        delete_node=self.head
        while delete_node:
            if delete_node==node:
                break
            delete_node=delete_node.Next
        if delete_node:
            if delete_node==self.head:
                return self.RemoveFirstNode()
            before_node=self.head
            while before_node:
                if before_node.Next==delete_node:
                    break
                before_node=before_node.Next
            before_node.Next=delete_node.Next
            return 1,delete_node
        else:
            return 0,None

lst1=SList()
for i in reversed(range(0,10)):
    node=lst1.SListNode(i+1,None)
    lst1.add2first(node)
```

```
# lst1.add2first(5)
    lst1.show()

# out=lst1.RemoveFirstNode()
# if out[0]==1:
#     print(out[1].Data)
# else:
#     print('list is empty')

# flag,node=lst1.RemoveFirstNode()
# if flag==1:
#     print(node.Data)

# lst1.show()
# while not lst1.is_empty():
#     lst1.RemoveFirstNode()
#     lst1.show()
# while not lst1.is_empty():
#     lst1.RemoveLastNode()
#     lst1.show()
# flag,node=lst1.RemoveLastNode()
# if flag==1:
#     print(node.Data)
# lst1.show()
# lst1.RemoveAfterNode(1)
# lst1.show()
# lst1.RemoveNodeByVal(1)
node=lst1.head.Next.Next.Next
lst1.RemoveNodeByRef(node)
lst1.show()
```

```
10->
9->10->
8->9->10->
7->8->9->10->
6->7->8->9->10->
5->6->7->8->9->10->
4->5->6->7->8->9->10->
3->4->5->6->7->8->9->10->
2->3->4->5->6->7->8->9->10->
1->2->3->4->5->6->7->8->9->10->
1->2->3->5->6->7->8->9->10->
```

[15]:
```
# print(list(reversed(range(0,100))))
```

[19]:
```
class StackList:
    class StackNode:
```

```
        def __init__(self,data=0,next=None):
            self.Data=data
            self.Next=next
    def __init__(self):
        self.head=None

    def show(self):
        tmp=self.head
        if tmp==None:
            print('list is empty')
        else:
            while tmp:
                print(tmp.Data,end='->')
                tmp=tmp.Next
            print()

    def is_empty(self):
        return self.head==None

    def push(self,inp):
        if type(inp)==type(self.StackNode()):
            inp.Next=self.head
            self.head=inp
        else:
            self.head=self.StackNode(inp,self.head)

    def pop(self):
        if not self.is_empty():
            tmp=self.head
            self.head=self.head.Next
            return 1,tmp
        else:
            return 0,None

lst1=StackList()
for i in reversed(range(0,10)):
    node=lst1.StackNode(i+1,None)
    lst1.push(node)
    lst1.show()

out=lst1.pop()
if out[0]==1:
    print(out[1].Data)
else:
    print('list is empty')
```

```
lst1.show()
```

```
10->
9->10->
8->9->10->
7->8->9->10->
6->7->8->9->10->
5->6->7->8->9->10->
4->5->6->7->8->9->10->
3->4->5->6->7->8->9->10->
2->3->4->5->6->7->8->9->10->
1->2->3->4->5->6->7->8->9->10->
1
2->3->4->5->6->7->8->9->10->
```

[25]:
```
class QueueList:
    class QueueNode:
        def __init__(self,data=0,next=None):
            self.Data=data
            self.Next=next
    def __init__(self):
        self.front=None
        self.rear=None

    def show(self):
        tmp=self.front
        if tmp==None:
            print('list is empty')
        else:
            while tmp:
                print(tmp.Data,end='<-')
                tmp=tmp.Next
            print()

    def is_empty(self):
        return self.front==None

    def push(self,x):
        node=self.QueueNode(x,None)
        if self.is_empty():
            self.front=self.rear=node
        else:
            self.rear.Next=node
            self.rear=node
    def pop(self):
        if not self.is_empty():
            tmp=self.front
```

```
            self.front=self.front.Next
            if self.front==None:
                self.rear=None
        else:
            return 0,None

q1=QueueList()
for i in range(5):
    q1.push(i)
    q1.show()
while not q1.is_empty():
    q1.pop()
    q1.show()
```

```
0<-
0<-1<-
0<-1<-2<-
0<-1<-2<-3<-
0<-1<-2<-3<-4<-
1<-2<-3<-4<-
2<-3<-4<-
3<-4<-
4<-
list is empty
```

[ ]: