

In [1]:

```
1 class Stack:
2
3     def __init__(self):
4         self.__data = []
5
6     def push(self,item1):
7         self.__data.append(item1)
8
9
10    def pop(self):
11        if self.isEmpty():
12            print("Hey!Stack is Empty!!")
13            return
14        return self.__data.pop()
15
16    def top(self):
17        if self.isEmpty():
18            print("Hey!Stack is Empty!!")
19            return
20        return self.__data[len(self.__data) - 1]
21
22    def size(self):
23        return len(self.__data)
24
25    def isEmpty(self):
26        return self.size() == 0
```

In [2]:

```
1 s = Stack()
2 s.push(12)
3 s.push(13)
4 s.push(14)
```

In [4]:

```
1 while s.isEmpty() is False:
2     print(s.pop())
```

```
14
13
12
```

In [4]:

```
1 s.top()
```

Hey!Stack is Empty!!

In [5]:

```
1 s.push(12)
2 s.push(13)
3 s.push(14)
4 s.push(15)
5 s.push(16)
```

In [6]:

```
1 while s.isEmpty() != True:
2     print(s.pop())
```

```
16
15
14
13
12
```

In [7]:

```
1 s.isEmpty()
```

Out[7]:

True

In [8]:

```
1 s.push(12)
2 s.push(123)
```

In [9]:

```
1 s.isEmpty()
```

Out[9]:

False

In [10]:

```
1 s.top()
```

Out[10]:

123

In [11]:

```
1 s.push(2)
2 s.push(3)
3 s.push(4)
4 s.push(6)
```

In [12]:

```
1 s.pop()
```

Out[12]:

6

In [13]:

```
1 s.top()
```

Out[13]:

4

In [14]:

```
1 while s.isEmpty() is False:  
2     print(s.pop())
```

4

3

2

123

12

Stack using Linked list

In [14]:

```
1 class Node:
2
3     def __init__(self,data):
4         self.data = data
5         self.next = None
6
7 class Stack:
8
9     def __init__(self):
10        self.__head = None
11        self.__count = 0
12
13    def push(self,element):
14        newNode = Node(element)
15        newNode.next = self.__head
16        self.__head = newNode
17        self.__count = self.__count + 1
18
19    def pop(self):
20        if self.isEmpty() is True:
21            print("Hey! Stack is empty!!")
22            return
23        data = self.__head.data
24        self.__head = self.__head.next
25        self.__count = self.__count - 1
26        data
27
28    def top(self):
29        if self.isEmpty() is True:
30            print("Hey! Stack is Empty!!")
31            return
32        data = self.__head.data
33        return data
34
35
36    def size(self):
37        return self.__count
38
39    def isEmpty(self):
40        return self.size() == 0
41
42
43
44 s = Stack()
45 s.push(3)
46 s.push(2)
47 s.push(5)
48
49 while s.isEmpty() is False:
50     print(s.pop())
51
52 s.top()
```

None

None

None

Hey! Stack is Empty!!

Built-in queue library

In [18]:

```
1 import queue  ## First in First out(FIFO)
2
3 q = queue.Queue()
4 q.put(1)
5 q.put(2)
6 q.put(3)
7 q.put(4)
8
9 while not q.empty():
10     print(q.get())
```

```
1
2
3
4
```

In [19]:

```
1 import queue  ## Last in last out (LIFO)
2
3 q = queue.LifoQueue()
4 q.put(1)
5 q.put(2)
6 q.put(3)
7
8 while not q.empty():
9     print(q.get())
```

```
3
2
1
```

Reversing a Stack using another stack

In [2]:

```
1 def reverseStack(s1,s2):
2
3     if (len(s1)<=1):
4         return
5
6     while (len(s1)!=1):
7         ele = s1.pop()
8         s2.append(ele)
9
10    lastElement = s1.pop()
11
12    while(len(s2)!=0):
13        ele = s2.pop()
14        s1.append(ele)
15
16    reverseStack(s1,s2)
17    s1.append(lastElement)
18
19 from sys import setrecursionlimit
20 setrecursionlimit(11000)
21 n = int(input())
22 s1 = [int(ele) for ele in input().split()]
23 s2 = []
24 reverseStack(s1,s2)
25 while(len(s1) != 0):
26     print(s1.pop(),end=' ')
```

```
5
1 2 3 6 4
1 2 3 6 4
```

In []:

1
