In [1]:

```
class Stack:

    def __init__(self):
        self.__data = []

    def push(self,item1):
        self.__data.append(item1)


    def pop(self):
        if self.isEmpty():
            print("Hey!Stack is Empty!!")
            return
        return self.__data.pop()

    def top(self):
        if self.isEmpty():
            print("Hey!Stack is Empty!!")
            return
        return self.__data[len(self.__data) - 1]

    def size(self):
        return len(self.__data)

    def isEmpty(self):
        return self.size() == 0
```

In [2]:

```
s = Stack()
s.push(12)
s.push(13)
s.push(14)
```

In [4]:

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

```
14
13
12
```

In [4]:

```
s.top()
```

```
Hey!Stack is Empty!!
```

In [5]:

```
s.push(12)
s.push(13)
s.push(14)
s.push(15)
s.push(16)
```

In [6]:

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

```
16
15
14
13
12
```

In [7]:

```
s.isEmpty()
```

Out[7]:

```
True
```

In [8]:

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

In [9]:

```
s.isEmpty()
```

Out[9]:

```
False
```

In [10]:

```
s.top()
```

Out[10]:

```
123
```

In [11]:

```
s.push(2)
s.push(3)
s.push(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]:

```
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,element):
        newNode = Node(element)
        newNode.next = self.__head
        self.__head = newNode
        self.__count = self.__count + 1

    def pop(self):
        if self.isEmpty() is True:
            print("Hey! Stack is empty!!")
            return
        data = self.__head.data
        self.__head = self.__head.next
        self.__count = self.__count - 1
        data

    def top(self):
        if self.isEmpty() is True:
            print("Hey! Stack is Empty!!")
            return
        data = self.__head.data
        return data


    def size(self):
        return self.__count

    def isEmpty(self):
        return self.size() == 0



s = Stack()
s.push(3)
s.push(2)
s.push(5)

while s.isEmpty() is False:
    print(s.pop())

s.top()
```

```
None
None
None
Hey! Stack is Empty!!
```

Built-in queue library

In [18]:

```
import queue  ## First in First out(FIFO)

q = queue.Queue()
q.put(1)
q.put(2)
q.put(3)
q.put(4)

while not q.empty():
    print(q.get())
```

```
1
2
3
4
```

In [19]:

```
import queue  ## last in last out (LIFO)

q = queue.LifoQueue()
q.put(1)
q.put(2)
q.put(3)

while not q.empty():
    print(q.get())
```

```
3
2
1
```

Reversing a Stack using another stack

In [2]:

```
def reverseStack(s1,s2):

    if (len(s1)<=1):
        return

    while (len(s1)!=1):
        ele = s1.pop()
        s2.append(ele)

    lastElement = s1.pop()

    while(len(s2)!=0):
        ele = s2.pop()
        s1.append(ele)

    reverseStack(s1,s2)
    s1.append(lastElement)

from sys import setrecursionlimit
setrecursionlimit(11000)
n = int(input())
s1 = [int(ele) for ele in input().split()]
s2 = []
reverseStack(s1,s2)
while(len(s1) != 0):
    print(s1.pop(),end=' ')
```

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

In [ ]:

```

```