

Code

```
Activity1Finals_JacobIvan.py x LinkedStack.py LinkedDeque.py LinkedQueue.py
1 |
2 from LinkedStack import LinkedStack as Stack
3 from LinkedQueue import LinkedQueue as Queue
4 from LinkedDeque import LinkedDeque as Deque
5 Q = Queue()
6 S = Stack()
7 D1 = Deque()
8 D2 = Deque()
9 print("-----")
10 print("|Variable |Contents |")
11 print("-----")
12 for i in range(1,9):
13     D1.insert_last(i)
14 for i in range(1,9):
15     Q.enqueue(D1.delete_first())
16 for i in range(1,9):
17     D1.insert_first(Q.dequeue())
18 for i in range(1,4):
19     D1.insert_last(D1.delete_first())
20 for i in range(1,7):
21     Q.enqueue(D1.delete_last())
22 for i in range(1,7):
23     D1.insert_last(Q.dequeue())
24 for i in range(1,4):
25     D1.insert_first(D1.delete_last())
26 #=====print elements of D=====
27 print("|D1 contents|", end='')
28 while not D1.is_empty():
29     print(D1.delete_first(), end='|')
30 #=====print elements of Q=====
31 print()
32 print("|Q contents |", end='')
33 while not Q.is_empty():
34     print(Q.dequeue(), end='|')
35
36 for i in range(1,9):
37     D2.insert_last(i)
38 for i in range(1,9):
39     S.push(D2.delete_last())
```

```
Activity1Finals_JacobIvan.py x LinkedStack.py LinkedDeque.py LinkedQueue.py
25 D1.insert_first(D1.delete_last())
26 #####print elements of D#####
27 print("|D1 contents|", end='')
28 while not D1.is_empty():
29     print(D1.delete_first(), end='|')
30 #####print elements of Q#####
31 print()
32 print("|Q contents |", end=' ')
33 while not Q.is_empty():
34     print(Q.dequeue(), end='|')
35
36 for i in range(1,9):
37     D2.insert_last(i)
38 for i in range(1,9):
39     S.push(D2.delete_last())
40 for i in range(1,4):
41     D2.insert_last(S.pop())
42 for i in range(4,3):
43     D2.insert_first(S.pop())
44 for i in range(1,4):
45     D2.insert_first(D2.delete_last())
46 for i in range(1,4):
47     D2.insert_last(S.pop())
48 #####print elements of D2#####
49 print()
50 print("|D2 contents|", end='')
51 while not D2.is_empty():
52     print(D2.delete_first(), end='|')
53 #####print elements of S#####
54 print()
55 print("|S contents |", end='')
56 while not S.is_empty():
57     print(S.pop(), end='|')
58
```

Output:

```
Run Activity1Finals_JacobIvan x
"C:\Program Files\Python312\python.exe" Z:\DSAL601-I082\FINALS\Activity1Finals_JacobIvan\Activity1Finals_JacobIvan.py
-----
|Variable |Contents |
-----
|D1 contents|=1|=2|=3|=4|=5|=6|=7|=8|=
|Q contents |=
|D2 contents|=1|=2|=3|=4|=5|=6|=7|=8|=
|S contents |=
Process finished with exit code 0
```

Codes I used:

```

20         return self._size
21     def is_empty(self):
22         '''Return True if the stack is empty.'''
23         return self._size == 0
24
25     1 usage
26     def push(self, e):
27         '''Add element e to the top of the stack.'''
28         self._head = self._Node(e, self._head)
29         self._size += 1
30     def top(self):
31         '''Return but do not remove the element at the top of the stack'''
32         '''Raise empty exception if the stack is empty!'''
33         if self.is_empty():
34             raise Exception('Stack is empty')
35         return self._head._element #top of the stack is the head of the list
36
37     4 usages
38     def pop(self):
39         '''Remove and return the elements from the top of the stack (LIFO)'''
40         '''Raise Empty exception if the stack is empty!'''
41         if self.is_empty():
42             raise Exception("The stack is empty!")
43         answer = self._head._element
44         self._head = self._head._next
45         self._size -= 1
46         return answer

```

```

12         raise Exception("Deque is empty.")
13     return self._trailer._prev._element #real item just before trailer
4 usages
14 def insert_first(self, e):
15     '''Add an element to the front of the deque.'''
16     self._insert_between(e, self._header, self._header._next)#after header
6 usages
17 def insert_last(self, e):
18     '''Add an element to the back of the deque'''
19     self._insert_between(e, self._trailer._prev, self._trailer)#before trailer
4 usages
20 def delete_first(self):
21     '''Remove and return the element from the front of the deque.'''
22     '''Raise Exception if the deque is empty.'''
23     if self.is_empty():
24         raise Exception("Deque is empty!")
25     return self._delete_node(self._header._next)#use inherited method
4 usages
26 def delete_last(self):
27     '''Remove and return the element from the back of the deque.'''
28     '''Raise Exception if the deque is empty.'''
29     if self.is_empty():
30         raise Exception("Deque is empty!")
31     return self._delete_node(self._trailer._prev)#use inherited method

```

```

23
24 def is_empty(self):
25     '''Return true if the queue is empty.'''
26     return self._size == 0
27
28 def first(self):
29     '''Return but do not remove the element at the front of the queue'''
30     if self.is_empty():
31         raise Exception('Queue is empty')
32     return self._head._element #front aligned with the head of the list
3 usages
33 def dequeue(self):
34     '''Remove and return the first element of the queue (FIFO)'''
35     '''Raise empty exception if the queue is empty'''
36     if self.is_empty():
37         raise Exception('Queue is empty')
38     answer = self._head._element
39     self._head = self._head._next
40     self._size -= 1
41     if self.is_empty():#special case as queue is empty
42         self._tail = None#removed head had been the tail
43     return answer
2 usages
44 def enqueue(self, e):
45     '''Add an element to the back of queue.'''
46     newest = self._Node(e, next=None)#node will be new tail node
47     if self.is_empty():
48         self._head = newest#special case: previously empty
49     else:
50         self._tail._next = newest
51     self._tail = newest#update reference to tail node
52     self._size += 1

```