CODE FOR QUEUE, STACK AND DEQUE:

Activity #1 (Queue).py | Activity #1 (Stack).py | LinkedStack.py | LinkedDeque.py

```
2 usages
class LinkedStack:
    '''LIFO STack implementation using a singly linked list for storage.'''

    #---------------- nested _Node class ----------------
    class _Node:
        '''Lightweight non public class for storing a singly linked node.'''
        __slots__ = '_element', '_next' #streamline memory usage

        def __init__(self, element, next):
            self._element = element
            self._next = next

    #--------------- stack methods ---------------
    def __init__(self):
        '''Create an empty Stack'''
        self._head = None
        self._size = 0
    def __len__(self):
        '''Return the number of elements in the stack'''
        return self._size
    2 usages
    def is_empty(self):
        '''Return True if the stack is empty.'''
        return self._size == 0

    2 usages
    def push(self, e):
        '''Add element e to the top of the stack.'''
        self._head = self._Node(e, self._head)
        self._size += 1
    def top(self):
        '''Return but do not remove the element at the top of the stack'''
        '''Raise empty exception if the stack is empty!'''
        if self.is_empty():
            raise Exception('Stack is empty')
        return self._head._element #top of the stack is the head of the list
    1 usage
    def pop(self):
        '''Remove and return the elements fro mthe top of the stack (LIFO)'''
        '''Raise Empty exception if the stack is empty!'''
```

```
        if self.is_empty():
            raise Exception("The stack is empty!")
        answer = self._head._element
        self._head = self._head._next
        self._size -=1
        return answer
```

Activity #1 (Queue).py | Activity #1 (Stack).py | LinkedStack.py | LinkedDeque.py | Li

```python
from DoublyLinkedBase import _DoublyLinkedBase
3 usages
class LinkedDeque(_DoublyLinkedBase):#note the use of inheritance
    '''Double ended queue implementation based on a doubly linked list.'''
    5 usages (5 dynamic)
    def first(self):
        '''Return but do not remove the element at the front of the deque'''
        if self.is_empty():
            raise Exception("Deque is empty!")
        return self._header._next._element #real item just after header
    2 usages (2 dynamic)
    def last(self):
        '''Return but do not remove the element at the back of the deque'''
        if self.is_empty():
            raise Exception("Deque is empty!")
        return self._trailer._prev._element #real item just before trailer
    2 usages
    def insert_first(self, e):
        '''Add an element to the fron of the deque.'''
        self._insert_between(e, self._header, self._header._next)#after header
    7 usages
    def insert_last(self, e):
        '''Add an element to the back of the deque'''
        self._insert_between(e, self._trailer._prev, self._trailer)#before trailer
    4 usages
    def delete_first(self):
        '''Remove and return the element from the front of the deque.'''
        '''Raise Exception if the deque is empty.'''
        if self.is_empty():
            raise Exception("Deque is empty!")
        return self._delete_node(self._header._next)#use inherited method
    7 usages
    def delete_last(self):
        '''Remove and return the element from the back of the deque.'''
        '''Raise Exception if the deque is empty.'''
        if self.is_empty():
            raise Exception("Deque is empty!")
        return self._delete_node(self._trailer._prev)#use inherited method
```

Activity #1 (Queue).py | Activity #1 (Stack).py | LinkedStack.py | LinkedDeque.py

2 usages

```python
class LinkedQueue:
    '''FIFO queue implementation using a singly linked list for storage.'''

    #---------------- nested _Node class ----------------
    class _Node:
        '''Lightweight non public class for storing a singly linked node.'''
        __slots__ = '_element', '_next' # streamline memory usage

        def __init__(self, element, next):
            self._element = element
            self._next = next

    #--------------- queue methods ----------------
    def __init__(self):
        '''Create an empty queue'''
        self._head = None
        self._tail = None
        self._size = 0

    def __len__(self):
        '''Return the number of elements in the queue'''
        return self._size

    4 usages
    def is_empty(self):
        '''Return true if the queue is empty.'''
        return self._size == 0
    5 usages (5 dynamic)
    def first(self):
        '''Return but do not remove the element at the fron of the queue'''
        if self.is_empty():
            raise Exception('Queue is empty')
        return self._head._element #front aligned with the head of the list
    2 usages
    def dequeue(self):
        '''Remove and return the first element of the queue (FIFO)'''
        '''Raise empty exception if the queue is empty'''
        if self.is_empty():
            raise Exception('Queue is empty')
        answer = self._head._element
```

```
2 usages
32      def dequeue(self):
33          '''Remove and return the first element of the queue (FIFO)'''
34          '''Raise empty exception if the queue is empty'''
35          if self.is_empty():
36              raise Exception('Queue is empty')
37          answer = self._head._element
38          self._head = self._head._next
39          self._size -= 1
40          if self.is_empty():#special case as queue is empty
41              self._tail = None#removed head had been the tail
42          return answer
        3 usages
43      def enqueue(self, e):
44          '''Add an element to the back of queue.'''
45          newest = self._Node(e, next: None)#node will be new tail node
46          if self.is_empty():
47              self._head = newest#special case: previously empty
48          else:
49              self._tail._next = newest
50          self._tail = newest#update reference to tail node
51          self._size += 1
```

Activity #1 (Queue).py | Activity #1 (Stack).py | LinkedDeque.py

```python
from LinkedQueue import LinkedQueue as Queue
from LinkedDeque import LinkedDeque as Deque



D = Deque()
Q = Queue()

for i in range(1, 9):
    D.insert_last(i)

for i in range(3):
    Q.enqueue(D.delete_last())#[8,7,6]

D.insert_first(D.delete_last())#[5,1,2,3,4]
Q.enqueue(D.delete_last())#[8,7,6,4]
D.insert_last(D.delete_first())#[1,2,3,5]

for i in range(4):
    D.insert_last(Q.dequeue())
for i in range(4):
    Q.enqueue(D.delete_last())
for i in range(4):
    D.insert_last(Q.dequeue())

while not D.is_empty():
    print(D.delete_first())
```

Activity #1 (Queue).py | Activity #1 (Stack).py | LinkedDeque.

```python
from LinkedStack import LinkedStack as Stack
from LinkedDeque import LinkedDeque as Deque

D = Deque()
S = Stack()

for i in range(1, 9):
    D.insert_last(i)

for i in range(3):
    S.push(D.delete_last())#[8,7,6]

D.insert_first(D.delete_last())#[5,1,2,3,4]
S.push(D.delete_last())#[8,7,6,4]
D.insert_last(D.delete_first())#[1,2,3,5]

for i in range(4):
    D.insert_last(S.pop())

while not D.is_empty():
    print(D.delete_first())
```

OUTPUT FOR QUEUE PROBLEM:

```
"C:\Program Files\Python312\python.exe" "Z:\DSALGO1-IDB2\FINALS\LinkedList_Activites\Activity #1 (Queue).py"
1
2
3
5
4
6
7
8
```

OUTPUT FOR STACKS AND DEQUE:

```
"C:\Program Files\Python312\python.exe" "Z:\DSALGO1-IDB2\FINALS\LinkedList_Activites\Activity #1 (Stack).py"
1
2
3
5
4
6
7
8
```