Code

Activity1Finals_Jacobivan.py | LinkedStack.py | LinkedDeque.py | LinkedQueue.py

```
from LinkedStack import LinkedStack as Stack
from LinkedQueue import LinkedQueue as Queue
from LinkedDeque import LinkedDeque as Deque
Q = Queue()
S = Stack()
D1 = Deque()
D2 = Deque()
print("_____")
print("|Variable   |Contents                        |")
print("-----------------------------------------------")
for i in range (1,9):
    D1.insert_last(i)
for i in range (1,9):
    Q.enqueue(D1.delete_first())
for i in range (1,9):
    D1.insert_first(Q.dequeue())
for i in range (1,4):
    D1.insert_last(D1.delete_first())
for i in range (1,7):
    Q.enqueue(D1.delete_last())
for i in range (1,7):
    D1.insert_last(Q.dequeue())
for i in range (1,4):
    D1.insert_first(D1.delete_last())
#============print elements of D====================
print("|D1 contents|", end='=')
while not D1.is_empty():
    print(D1.delete_first(), end='=|=')
#============print elements of Q ==================
print()
print("|Q contents |", end='= ')
while not Q .is_empty():
    print(Q.dequeue(), end='=|')

for i in range (1,9):
    D2.insert_last(i)
for i in range (1,9):
    S.push(D2.delete_last())
```

```python
    D1.insert_first(D1.delete_last())
#============print elements of D=====================
print("|D1 contents|", end='=')
while not D1.is_empty():
    print(D1.delete_first(), end='=|=')
#============print elements of Q =================
print()
print("|Q contents |", end='= ')
while not Q .is_empty():
    print(Q.dequeue(), end='=|')

for i in range (1,9):
    D2.insert_last(i)
for i in range (1,9):
    S.push(D2.delete_last())
for i in range (1,4):
    D2.insert_last(S.pop())
for i in range (1,3):
    D2.insert_first(S.pop())
for i in range (1,4):
    D2.insert_first(D2.delete_last())
for i in range (1,4):
    D2.insert_last(S.pop())
#============print elements of D2====================
print()
print("|D2 contents|", end='=')
while not D2.is_empty():
    print(D2.delete_first(), end='=|=')
#============print elements of S====================
print()
print("|S contents |", end='=')
while not S.is_empty():
    print(S.pop(), end='=|')
```

Output:

```
"C:\Program Files\Python312\python.exe" Z:\DSAL601-IDB2\FINALS\Activity1Finals_JacobIvan\Activity1Finals_JacobIvan.py
-----------------------------------------------
|Variable   |Contents                         |
-----------------------------------------------
|D1 contents|=1=|=2=|=3=|=5=|=4=|=6=|=7=|=8=|=
|Q contents |=
|D2 contents|=1=|=2=|=3=|=5=|=4=|=6=|=7=|=8=|=
|S contents |=
Process finished with exit code 0
```

Codes I used:

```
    def is_empty(self):
        '''Return True if the stack is empty.'''
        return self._size == 0

    1 usage
    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
    4 usages
    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
```

```
        return self._trailer._prev._element #real item just before trailer
    4 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
    6 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
    4 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
```

```

    def is_empty(self):
        '''Return true if the queue is empty.'''
        return self._size == 0
    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
    3 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
        self._head = self._head._next
        self._size -= 1
        if self.is_empty():#special case as queue is empty
            self._tail = None#removed head had been the tail
        return answer
    2 usages
    def enqueue(self, e):
        '''Add an element to the back of queue.'''
        newest = self._Node(e, next: None)#node will be new tail node
        if self.is_empty():
            self._head = newest#special case: previously empty
        else:
            self._tail._next = newest
        self._tail = newest#update reference to tail node
        self._size += 1
```