.TermProject1A

Code:

```python
from LinkedStack import LinkedStack

1 usage
class DequeWithTwoStacks:

    def __init__(self):
        self.stack1 = LinkedStack()
        self.stack2 = LinkedStack()

    3 usages (2 dynamic)
    def is_empty(self):
        return self.stack1.is_empty() and self.stack2.is_empty()

    def __len__(self):
        return len(self.stack1) + len(self.stack2)

    2 usages
    def add_first(self, e):
        self.stack1.push(e)

    2 usages
    def add_last(self, e):
        self.stack2.push(e)

    1 usage
    def remove_first(self):
        if self.stack1.is_empty():
            while not self.stack2.is_empty():
                self.stack1.push(self.stack2.pop())
        if self.stack1.is_empty():
            raise Exception("Deque is empty!")
        return self.stack1.pop()

    1 usage
    def remove_last(self):
        if self.stack2.is_empty():
            while not self.stack1.is_empty():
                self.stack2.push(self.stack1.pop())
        if self.stack2.is_empty():
            raise Exception("Deque is empty!")
        return self.stack2.pop()
```

```
    1 usage
    def first(self):
        if self.stack1.is_empty():
            while not self.stack2.is_empty():
                self.stack1.push(self.stack2.pop())
        if self.stack1.is_empty():
            raise Exception("Deque is empty!")
        return self.stack1.top()

    1 usage
    def last(self):
        if self.stack2.is_empty():
            while not self.stack1.is_empty():
                self.stack2.push(self.stack1.pop())
        if self.stack2.is_empty():
            raise Exception("Deque is empty!")
        return self.stack2.top()

    1 usage
    def print_deque(self):
        # Gather elements from stack1 and stack2
        stack1_elements = []
        while not self.stack1.is_empty():
            stack1_elements.append(self.stack1.pop())
        stack2_elements = []
        while not self.stack2.is_empty():
            stack2_elements.append(self.stack2.pop())

        # Refill stacks with elements
        for element in reversed(stack2_elements):
            self.stack2.push(element)
        for element in reversed(stack1_elements):
            self.stack1.push(element)

        # The final deque is stack1 elements followed by stack2 elements
        return stack1_elements + stack2_elements
```

```python
try:
    deque = DequeWithTwoStacks()
    deque.add_first(10)
    deque.add_last(20)
    deque.add_first(5)
    deque.add_last(30)

    print("First element:", deque.first())
    print("Last element:", deque.last())

    print("Removed first:", deque.remove_first())
    print("Removed last:", deque.remove_last())

    print("Length:", len(deque))

    print("Is empty?", deque.is_empty())

    print("Deque list:", deque.print_deque())

except Exception as e:
    print(f"Error: {e}")
```

Output:

.TermProject1B

Code:

```
1 usage
class LinkedQueue:
    class _Node:
        __slots__ = '_element', '_next'

        def __init__(self, element, next=None):
            self._element = element
            self._next = next

    def __init__(self):
        self._head = None
        self._tail = None
        self._size = 0

    def __len__(self):
        return self._size

    9 usages (2 dynamic)
    def is_empty(self):
        return self._size == 0

    1 usage
    def first(self):
        if self.is_empty():
            raise Exception('Queue is empty')
        return self._head._element

    1 usage
    def enqueue(self, e):
        newest = self._Node(e, next: None)
        if self.is_empty():
            self._head = newest
        else:
            self._tail._next = newest
        self._tail = newest
        self._size += 1

    1 usage
    def dequeue(self):
        if self.is_empty():
            raise Exception('Queue is empty')
```

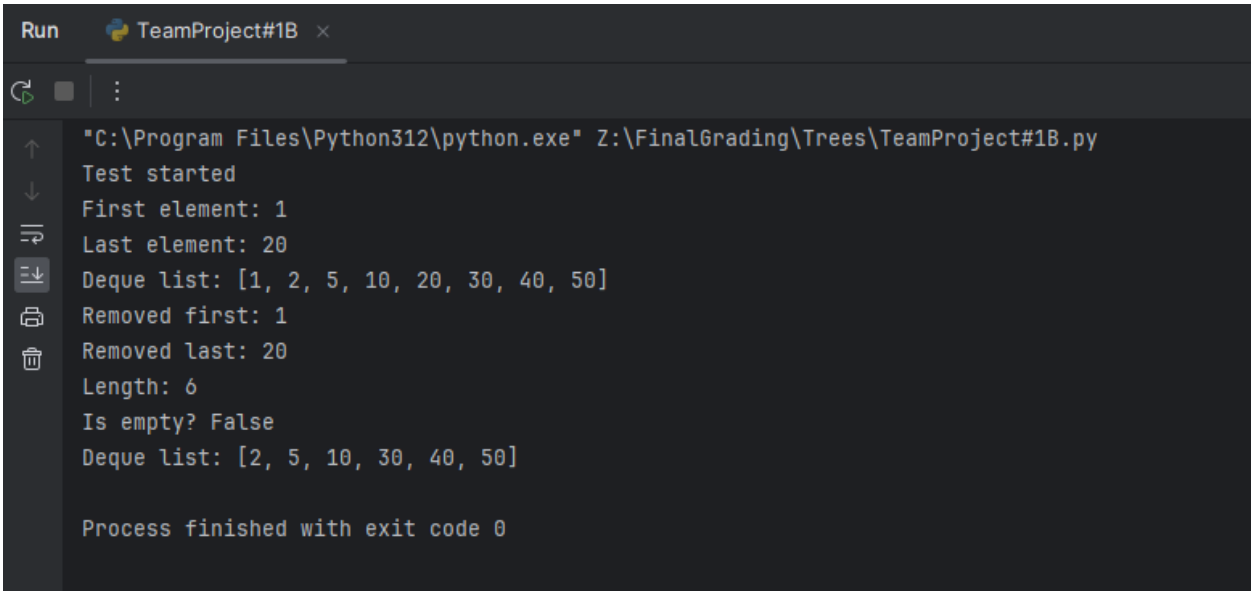
```
        answer = self._head._element
        self._head = self._head._next
        self._size -= 1
        if self.is_empty():
            self._tail = None
        return answer


1 usage
class LinkedStack:
    class _Node:
        __slots__ = '_element', '_next'

        def __init__(self, element, next=None):
            self._element = element
            self._next = next

    def __init__(self):
        self._head = None
        self._size = 0

    def __len__(self):
        return self._size

    8 usages (2 dynamic)
    def is_empty(self):
        return self._size == 0

    2 usages
    def push(self, e):
        self._head = self._Node(e, self._head)
        self._size += 1

    1 usage
    def top(self):
        if self.is_empty():
            raise Exception('Stack is empty')
        return self._head._element
```

```
72          def pop(self):
73              if self.is_empty():
74                  raise Exception('Stack is empty')
75              answer = self._head._element
76              self._head = self._head._next
77              self._size -= 1
78              return answer
79
80
        1 usage
81      class DequeWithStackAndQueue:
82          def __init__(self):
83              self.stack = LinkedStack()
84              self.queue = LinkedQueue()
85
            3 usages (2 dynamic)
86          def is_empty(self):
87              return self.stack.is_empty() and self.queue.is_empty()
88
89          def __len__(self):
90              return len(self.stack) + len(self.queue)
91
            4 usages
92          def add_first(self, e):
93              self.stack.push(e)
94
            4 usages
95          def add_last(self, e):
96              self.queue.enqueue(e)
97
            1 usage
98          def remove_first(self):
99              if self.stack.is_empty():
100                 raise Exception('Deque is empty')
101             return self.stack.pop()
102
            1 usage
103         def remove_last(self):
104             if self.queue.is_empty():
105                 raise Exception('Deque is empty')
106             return self.queue.dequeue()
```

```
        1 usage
    def first(self):
        if self.stack.is_empty():
            raise Exception('Deque is empty')
        return self.stack.top()

        1 usage
    def last(self):
        if self.queue.is_empty():
            raise Exception('Deque is empty')
        return self.queue.first()

        2 usages
    def print_deque(self):
        stack_elements = []
        while not self.stack.is_empty():
            stack_elements.append(self.stack.pop())
        queue_elements = []
        current = self.queue._head
        while current:
            queue_elements.append(current._element)
            current = current._next
        for element in reversed(stack_elements):
            self.stack.push(element)

        return stack_elements + queue_elements


try:
    print("Test started")
    deque = DequeWithStackAndQueue()

    deque.add_first(10)
    deque.add_last(20)
    deque.add_first(5)
    deque.add_last(30)
    deque.add_first(2)
    deque.add_last(40)
    deque.add_first(1)
    deque.add_last(50)

```

```
145
146         print("First element:", deque.first())
147         print("Last element:", deque.last())
148
149         print("Deque list:", deque.print_deque())
150
151         print("Removed first:", deque.remove_first())
152         print("Removed last:", deque.remove_last())
153
154         print("Length:", len(deque))
155
156         print("Is empty?", deque.is_empty())
157         print("Deque list:", deque.print_deque())
158     except Exception as e:
159         print(f"Error: {e}")
160
```

Output:


Run
TeamProject#1B
"C:\Program Files\Python312\python.exe" Z:\FinalGrading\Trees\TeamProject#1B.py
Test started
First element: 1
Last element: 20
Deque list: [1, 2, 5, 10, 20, 30, 40, 50]
Removed first: 1
Removed last: 20
Length: 6
Is empty? False
Deque list: [2, 5, 10, 30, 40, 50]
Process finished with exit code 0