MARK LLOYD YADAO 11/22/24

IDB2/DSALGO1

1.

Activity2.py PositionalList.py LinkedStack.py

```python
#1
1 usage
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

    5 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'''
        if self.is_empty():
            raise Exception('Stack is empty')
        return self._head._element

    3 usages
```

```
37
        3 usages
38      def pop(self):
39          '''Remove and return the elements from the top of the stack (LIFO)'''
40          if self.is_empty():
41              raise Exception("The stack is empty!")
42          answer = self._head._element
43          self._head = self._head._next
44          self._size -= 1
45          return answer
46
    1 usage
47  def evaluate_postfix(expression):
48      stack = LinkedStack()
49      operators = {'+', '-', '*', '/'}
50
51      for token in expression.split():
52          if token not in operators:
53              stack.push(float(token))
54          else:
55              b = stack.pop()
56              a = stack.pop()
57
58              if token == '+':
59                  stack.push(a + b)
60              elif token == '-':
61                  stack.push(a - b)
62              elif token == '*':
63                  stack.push(a * b)
64              elif token == '/':
65                  stack.push(a / b)
66
67      return stack.pop()
68
69  # Example usage
70  postfix_expression = "5 2 + 8 3 - * 4 /"
71  result = evaluate_postfix(postfix_expression)
72  print(f"The result of the postfix expression '{postfix_expression}' is: {result}")
73
```

2.

```
#2
1 usage
class PositionalList:
    class _Node:
        __slots__ = '_element', '_prev', '_next'

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

    class Position:
        def __init__(self, container, node):
            self._container = container
            self._node = node

        def element(self):

            return self._node._element

        def __eq__(self, other):

            return type(other) is type(self) and other._node is self._node

        def __ne__(self, other):

            return not (self == other)

    def __init__(self):
        '''Create an empty list.'''
        self._header = self._Node( element: None,  prev: None,  next: None)
        self._trailer = self._Node( element: None, self._header,  next: None)
        self._header._next = self._trailer
        self._size = 0

    def __len__(self):

        return self._size

    def is_empty(self):
```

Activity2.py × PositionalList.py LinkedStack.py

```
117
118             if not isinstance(p, self.Position):
119                 raise TypeError('p must be proper Position type')
120             if p._container is not self:
121                 raise ValueError('p does not belong to this container')
122             if p._node._next is None:
123                 raise ValueError('p is no longer valid')
124             return p._node
125
        5 usages
126         def _make_position(self, node):
127
128             if node is self._header or node is self._trailer:
129                 return None  # boundary violation
130             else:
131                 return self.Position(self, node)
132
        1 usage
133         def first(self):
134
135             return self._make_position(self._header._next)
136
137         def last(self):
138
139             return self._make_position(self._trailer._prev)
140
141         def before(self, p):
142
143             node = self._validate(p)
144             return self._make_position(node._prev)
145
        1 usage
146         def after(self, p):
147
148             node = self._validate(p)
149             return self._make_position(node._next)
150
151         def __iter__(self):
152
153             cursor = self.first()
154             while cursor is not None:
```

```
            yield cursor.element()
            cursor = self.after(cursor)

    5 usages (1 dynamic)
    def _insert_between(self, e, predecessor, successor):

        node = self._Node(e, predecessor, successor)
        predecessor._next = node
        successor._prev = node
        self._size += 1
        return self._make_position(node)

    def add_first(self, e):

        return self._insert_between(e, self._header, self._header._next)

    1 usage
    def add_last(self, e):

        return self._insert_between(e, self._trailer._prev, self._trailer)

    def add_before(self, p, e):

        original = self._validate(p)
        return self._insert_between(e, original._prev, original)

    def add_after(self, p, e):

        original = self._validate(p)
        return self._insert_between(e, original, original._next)

    def delete(self, p):

        original = self._validate(p)
        element = original._element
        original._prev._next = original._next
        original._next._prev = original._prev
        original._prev = original._next = original._element = None  # deprecate node
        self._size -= 1
        return element
```

```
        return element

    def replace(self, p, e):

        original = self._validate(p)
        old_value = original._element
        original._element = e
        return old_value

2 usages
def insertion_sort(arr, ascending=True):
    pos_list = PositionalList()
    for item in arr:
        pos_list.add_last(item)

    if ascending:
        sorted_arr = sorted(pos_list)
    else:
        sorted_arr = sorted(pos_list, reverse=True)

    return sorted_arr

# Example usage
numbers = [1, 72, 81, 25, 65, 91, 11]
ascending_sorted = insertion_sort(numbers, ascending=True)
descending_sorted = insertion_sort(numbers, ascending=False)

print("Sorted in ascending order:", ascending_sorted)
print("Sorted in descending order:", descending_sorted)
```

OUTPUT

```
The result of the postfix expression '5 2 + 8 3 - * 4 /' is: 8.75
Sorted in ascending order: [1, 11, 25, 65, 72, 81, 91]
Sorted in descending order: [91, 81, 72, 65, 25, 11, 1]
```