StackusingLList.py

```python
# Advanced Stack Implementation Using Linked List (Conceptual for Large Data)
# Sometimes, stacks are implemented using linked lists
# To avoid resizing issues with lists and to practice pointer manipulation.

# Node class for linked list
class StackNode:
    def __init__(self, value):
        self.value = value  # Store the node's data
        self.next = None    # Pointer to the next node (below in stack)

# Stack implemented with linked list
class LinkedListStack:
    def __init__(self):
        self.top = None  # Points to the top node in the stack

    def is_empty(self):
        # Stack is empty if top pointer is None
        return self.top is None

    def push(self, value):
        # Create a new node with the value
        new_node = StackNode(value)

        # New node's next pointer should point to current top node
        new_node.next = self.top

        # Update top pointer to new node (new top of stack)
        self.top = new_node

    def pop(self):
        # If stack is empty, raise error
        if self.is_empty():
            raise Exception("Cannot pop from empty stack!")

        # Retrieve value from top node
        popped_value = self.top.value

        # Move top pointer to next node down the stack
        self.top = self.top.next

        # Return popped value
        return popped_value

    def peek(self):
        # Return top element value without removing it
        if self.is_empty():
            raise Exception("Cannot peek on empty stack!")
        return self.top.value

    def display(self):
        # Print stack from top to bottom
        current = self.top
        values = []
        while current:
            values.append(str(current.value))
            current = current.next
        print("Stack from top to bottom:", " -> ".join(values))


# Example usage:
if __name__ == "__main__":
    stack_ll = LinkedListStack()
    stack_ll.push(5)
    stack_ll.push(10)
    stack_ll.push(15)

    stack_ll.display()  # Expected: Stack from top to bottom: 15 -> 10 -> 5
```

```
68
69      print("Peek top:", stack_ll.peek())  # Expected: 15
70
71      print("Pop:", stack_ll.pop())  # Expected: 15
72      stack_ll.display()  # Expected: 10 -> 5
73
```