Data Structures - 2025-1
Assignment 03 | 20241243

<list>

01.

장점
- 첫 번째 노드를 삭제하거나 삽입할 때 일반적인 노드 처리와 동일하게 예외 처리없이 구현할 수 있음.
- 헤드가 None인지 확인할 필요가 없어 불필요한 코드가 줄어듦.

단점
- 리스트의 크기의 상관없이 항상 하나의 추가적인 노드가 필요하여 메모리 사용량이 증가한다.

02.

```
if (index(x) == -1 or index(x) == None)
        return False
return True
```

03.

```
    while current and index < i:
        current = current.next
        index += 1

    while current and index <= j:
        print(current.data, end=' ')
        current = current.next
        index += 1
```

04.

```
    while index < i:
        curr = curr.next
        index += 1
        if curr == self.head:        # 순환구조
                return

    while index <= j:
        print(current.data, end=' ')
        current = current.next
        index += 1
        if curr == self.head:
                break
```

05.

(재귀 없는 버전)

```
def numItems(self):
 count = 0
 curr = self.__head.next
```

```
 while (curr != None)
        count += 1
        curr = curr.next
 return count
```

(재귀 알고리즘)

```
def numItems(self):
        def count_nodes(node):
                if (node == None)
                        return 0
                return 1 + count_nodes(node.next)
        return count_nodes(self.__head.next)
```

06.

```
def pop(self, i, k):
    if (index < 0 or index >= self.__numItems):
        return None

    retItems = []
    prev = self.__getNode(index - 1)
    curr = prev.next

    for _ in range(k):
        if curr is None:
            break
        retItems.append(curr.item)
        prev.next = curr.next
        curr = curr.next
        self.__numItems -= 1
    return retItems
```

07.

```
newNode = BidirectNode(x, None, None)
cur = self.__head.next
while ((self.cur != self.__head) and (cur.item < x)):
        prev = curr
        newNode.next.prev = newNode
        prev.next = newNode
        self.__numItems ++
```

08.

```
if (index(node1) != -12345 and index(node2) != -12345)
        return True
return False
```

09.

```
def lastIndexOf(self, x):
        while (curr.next != None)
                self.__lastIndex = index(x)
                curr = curr.next
        return lastIndex
```

10.

```
def lastIndexOf(self, x):
        while (curr.next != self.__head)
                self.__lastIndex = index(x)
                curr = curr.next
        return lastIndex
```

<stack>

01. 15 25 + 10 2 * -

1. push(15)
2. push(25)
3. '+' -> pop(25) -> pop(15) -> 15+25 = 40
4. push(40)
5. push(10)
6. push(2)
7. '*' -> pop(2) -> pop(10) -> 10*2 = 20
8. push(20)
9. '-' -> pop(20) -> pop(40) -> 40-20 = 20

02.

```
class ListStack:
 def __init__(self):
  self.__stack = []
 def push(self, x):
  self.__stack.insert(0, x)
 def pop(self):
  return self.__stack.pop(-1)
 def top(self):
  if self.isEmpty():
   print("No element in stack") return None else:
   return self.__stack[0]
 def isEmpty(self) -> bool:
  return not bool(self.__stack) #return len(self.__stack)==0
def popAll(self):
 self.__stack.clear()
def printStack(self):
 print("Stack:") for i in range(len(self.__stack)):
 print('stack[', i, ']:', self.__stack[i])
```

03.

```
n : 스택에 있는 원소의 개수
st = ListStack()
sameCount = 0
curr = st.top
while not st.isEmpty():
        if (curr == 'w' or curr == 'wR')
                sameCount++
                curr = curr.next
if sameCount == 2:
 return "yes"
else
 return "no"
```

04.

```
def copy(self):
        temp_stack = LinkedStack()
        new_stack = LinkedStack()
        curr = self.top
        while curr:      #curr이 none이 아닐때까지 반복.
                temp_stack.push(curr.item)
                curr = curr.next
        while temp_stack.isEmpty() == False:     #while not temp_stack.isEmpty():
                new_stack.push(temp_stack.pop())
        reuturn new_stack
```

05.

```
def parenBalanvce(s) -> bool:
        stack = LinkedStack()
        for char in s:
        if char == '(':
                stack.push(char)
        elif (s[i] == ')')
                if stack,isEmpty == True:
                        return False
                stack.pop()
        return len(stack) == 0
```

06.

괄호의 종류만 늘어날뿐 괄호를 검사하는 로직은 동일하다. 괄호의 짝이 맞는지에 대한 조건만 추가된다.
그 외 로직은 동일함.

07.

100개

08.

51개