**linkedListBasic.py 코드**

from listNode import ListNode

class linkedListBasic:

def \_\_init\_\_(self):

'''

더미 헤드 \_\_head를 만듦

항목 갯수를 0으로 초기화

'''

self.\_\_head = ListNode('dummy', None)

self.\_\_numItems = 0

def insert(self, i:int, newItem):

'''

i가 list의 item 갯수보다 적으면,

i번째에 newItem 값을 넣어라

그렇지 않으면 error 출력

'''

if i >= 0 and i <= self.\_\_numItems:

prev = self.\_\_getNode(i - 1)

newNode = ListNode(newItem, prev.next)

prev.next = newNode

self.\_\_numItems += 1

else:

print("index", i, "out of bound in insert()")

def append(self, newItem):

'''

newItem을 list의 맨 끝에 삽입해라.

이전 node의 next 속성에 newNode를 부여.

\_\_numItems를 1 증가

'''

prev = self.\_\_getNode(self.\_\_numItems - 1)

newNode = ListNode(newItem, prev.next)

prev.next = newNode

self.\_\_numItems += 1

def pop(self, i:int):

'''

i가 0 이상 list 길이 이하이면

prev의 next 속성에 다음 노드(next)를 부여.

\_\_numItems를 1 감소

'''

if (i >= 0 and i <= self.\_\_numItems - 1):

prev = self.\_\_getNode(i - 1)

curr = prev.next

prev.next = curr.next

retItem = curr.item

self.\_\_numItems -= 1

return retItem

else:

return None

def remove(self, x):

'''

x를 찾음. x가 list에 없으면 아무것도 안함

x가 list에 있으면

이전 node의 next 속성에 다음 노드를 부여

\_\_numitems를 1 감소

'''

(prev, curr) = self.\_\_findNode(x)

if curr != None:

prev.next = curr.next

self.\_\_numItems -= 1

return x

else:

return None

def get(self, i:int):

'''

i번째의 item을 return하는 함수.

list가 비었다면 None을 return.

\_\_getNode(i) 함수로 i번째 노드를 가져오고,

i번째 노드의 item을 return'''

if self.isEmpty():

return None

if (i >= 0 and i <= self.\_\_numItems - 1):

return self.\_\_getNode(i).item

else:

return None

def index(self, x) -> int:

'''

x 값의 index를 return하는 함수.

curr 변수를 첫 번째 노드로 설정

노드의 갯수 동안 curr의 값이 x인지 확인

x인 curr을 찾으면 그 index를 return

아니면 -2(없는 인덱스)를 return

'''

curr = self.\_\_head.next

for index in range(self.\_\_numItems):

if curr.item == x:

return index

else:

curr = curr.next

return -2

def isEmpty(self) -> bool:

return self.\_\_numItems == 0

def size(self) -> int:

return self.\_\_numItems

def clear(self):

self.\_\_head = ListNode('dummy', None)

self.\_\_numItems = 0

def count(self, x) -> int:

cnt = 0

curr = self.\_\_head.next

while curr != None:

if curr.item == x:

cnt += 1

curr = curr.next

return cnt

def extend(self, a):

for index in range(a.size()):

self.append(a.get(index))

def copy(self):

a = linkedListBasic()

for index in range(self.\_\_numItems):

a.append(self.get(index))

return a

def reverse(self):

a = linkedListBasic()

for index in range(self.\_\_numItems):

a.insert(0, self.get(index))

self.clear()

for index in range(a.size()):

self.append(a.get(index))

def sort(self) -> None:

a = []

for index in range(self.\_\_numItems):

a.append(self.get(index))

a.sort()

self.clear()

for index in range(len(a)):

self.append(a[index])

def \_\_findNode(self, x) -> (ListNode, ListNode):

prev = self.\_\_head

curr = prev.next

while curr != None:

if curr.item == x:

return (prev, curr)

def \_\_getNode(self, i:int) -> ListNode:

curr = self.\_\_head

for index in range(i+1):

curr = curr.next

return curr

def printList(self):

curr = self.\_\_head.next

while curr != None:

print(curr.item, end = ' ')

curr = curr.next

print()

def \_\_iter\_\_(self):

self.position = 0

return self

def \_\_next\_\_(self):

if self.position >= self.\_\_numItems:

raise StopIteration

result = self.get(self.position)

self.position += 1

return result

linkedListBasic의 메서드 부분은 교재를 참고하여 작성하였습니다.

Iterator은 직접 구현하였습니다.

**linkedListBasics.py Iterator 해설**.

def \_\_iter\_\_(self):

self.position = 0

return self

def \_\_next\_\_(self):

if self.position >= self.\_\_numItems:

raise StopIteration

result = self.get(self.position)

self.position += 1

return result

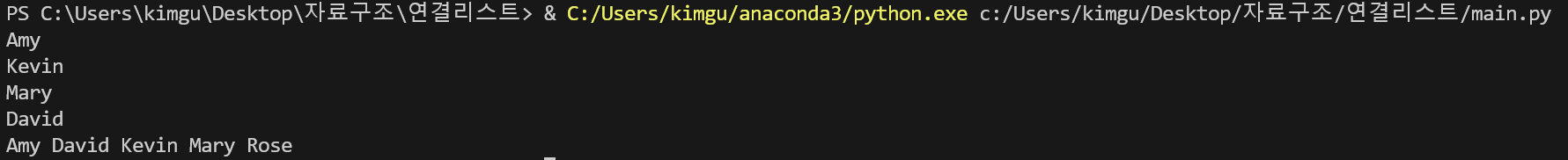
Iterator 부분은 \_\_iter\_\_(self) 메서드와 \_\_next\_\_(self) 메서드로 구성되어 있습니다.

\_\_iter\_\_(self) 메서드는 Iteration을 시작할 때 생성자로 사용됩니다. 여기에서 속성 self.position을 정의하여 Iterate할 때의 위치를 정의하였습니다.

\_\_next\_\_(self) 메서드에서 self.position이 리스트 항목 수보다 같거나 크면 StopIteration 오류를 raise하여 Iteration을 종료시킵니다.

그렇지 않은 경우, self.position번째 item을 return하고, self.position을 1 증가시킵니다.

출력 결과



**circularLinkedListBasic.py 코드**

from listNode import ListNode

class circularLinkedListBasic:

def \_\_init\_\_(self):

'''

항목 갯수를 0으로 초기화

'''

self.\_\_tail = ListNode('dummy', self.\_\_tail)

self.\_\_numItems = 0

def insert(self, i:int, newItem) -> None:

'''

i가 list의 item 갯수보다 적으면,

i번째에 newItem 값을 넣어라

그렇지 않으면 error 출력

'''

if (self.\_\_numItems == i):

self.append(i)

elif i >= 0 and i < self.\_\_numItems:

prev = self.\_\_prevNode(i)

newNode = ListNode(newItem, prev.next)

prev.next = newNode

self.\_\_numItems += 1

else:

print("index", i, "out of bound in insert()")

def append(self, newItem):

'''

newItem을 list의 맨 끝에 삽입해라.

이전 node의 next 속성에 newNode를 부여.

새롭게

\_\_numItems를 1 증가

'''

if self.\_\_numItems == 0:

self.\_\_tail.next = ListNode(newItem, self.\_\_tail.next)

self.\_\_numItems += 1

else:

first = self.\_\_tail.next.next

self.\_\_tail.next.next = ListNode(newItem, first)

self.\_\_numItems += 1

def pop(self, i:int=None):

'''

i가 0 이상 list 길이 이하이면

prev의 next 속성에 다음 노드(next)를 부여.

\_\_numItems를 1 감소

'''

if i == None or i == self.\_\_numItems - 1:

retItem = self.\_\_tail.next.item

first = self.\_\_tail.next.next

prev = self.\_\_getNode(self.\_\_numItems - 2)

prev.next = first

self.\_\_numItems -= 1

return retItem

if (i >= 0 and i < self.\_\_numItems - 1):

prev = self.\_\_getNode(i - 1)

curr = prev.next

prev.next = curr.next

retItem = curr.item

self.\_\_numItems -= 1

return retItem

else:

return None

def remove(self, x):

'''

x를 찾음. x가 list에 없으면 아무것도 안함

x가 list에 있으면

이전 node의 next 속성에 다음 노드를 부여

\_\_numitems를 1 감소

'''

first = self.\_\_tail.next.next

(prev, curr) = self.\_\_findNode(x)

if curr != None:

if curr == self.\_\_tail.next:

return self.pop()

prev.next = curr.next

self.\_\_numItems -= 1

return x

else:

return None

def get(self, i:int):

'''

i번째의 item을 return하는 함수.

list가 비었다면 None을 return.

\_\_getNode(i) 함수로 i번째 노드를 가져오고,

i번째 노드의 item을 return'''

if self.isEmpty():

return None

if (i >= 0 and i < self.\_\_numItems):

return self.\_\_getNode(i).item

else:

return None

def index(self, x) -> int:

'''

x 값의 index를 return하는 함수.

curr 변수를 첫 번째 노드로 설정

노드의 갯수 동안 curr의 값이 x인지 확인

x인 curr을 찾으면 그 index를 return

아니면 -2(없는 인덱스)를 return

'''

curr = self.tail.next.next

for index in range(self.\_\_numItems-1):

if curr.item == x:

return index

else:

curr = curr.next

return -2

def isEmpty(self) -> bool:

return self.\_\_numItems == 0

def size(self) -> int:

return self.\_\_numItems

def clear(self):

self.\_\_tail = ListNode('dummy', self.\_\_tail)

self.\_\_numItems = 0

def count(self, x) -> int:

cnt = 0

first = self.\_\_tail.next.next

curr = first

while curr != first:

if curr.item == x:

cnt += 1

curr = curr.next

return cnt

def extend(self, a):

first = self.\_\_tail.next.next

self.\_\_tail.next.next = a.copy.getfirst()

self.\_\_numItems += a.size()

curr = first.next

for index in range(self.\_\_numItems-2):

curr = curr.next

curr.next = first

self.\_\_tail.next = curr

def copy(self):

a = circularLinkedListBasic()

for index in range(self.\_\_numItems):

a.append(self.get(index))

return a

def reverse(self):

a = circularLinkedListBasic()

for index in range(self.\_\_numItems):

a.insert(0, self.get(index))

self.clear()

for index in range(a.size()):

self.append(a.get(index))

def sort(self) -> None:

a = []

for index in range(self.\_\_numItems):

a.append(self.get(index))

a.sort()

self.clear()

for index in range(len(a)):

self.append(a[index])

def \_\_findNode(self, x) -> (ListNode, ListNode):

prev = self.\_\_tail.next

curr = prev.next

while curr != self.\_\_tail.next:

if curr.item == x:

return (prev, curr)

else:

prev = curr; cur = curr.next

if curr.item == x:

return (prev, curr)

else:

return (None, None)

def \_\_getNode(self, i:int) -> ListNode:

curr = self.\_\_tail.next.next

for index in range(i):

curr = curr.next

return curr

def printList(self):

curr = self.\_\_tail.next.next

while curr != self.\_\_tail.next:

print(curr.item, end = ' ')

curr = curr.next

print(curr.item)

def getfirst(self) -> ListNode:

return self.\_\_tail.next.next

def \_\_iter\_\_(self):

self.position = 0

return self

def \_\_next\_\_(self):

if self.position >= self.\_\_numItems:

raise StopIteration

result = self.get(self.position)

self.position += 1

return result

\_\_init\_\_함수에서는\_\_tail 속성을 LinkedList(“dummy”, self.\_\_tail)로 정의하였습니다. \_\_tail.next.next 속성은 리스트의 첫 항목으로 정의하여 다른 메서드를 작성하였습니다. Iterator 부분은 LinkedListBasic 부분과 같이 작성하였습니다.

다만, 충분히 검토하지 못하여 버그가 발생하는 상황입니다.