리스트

Contents

주요 내용

- 01 리스트란
- 02 배열 리스트(파이썬 기본 제공)
- 03 연결 리스트
- 04 배열 리스트와 연결 리스트의 비교
- 05 연결 리스트의 개선 및 확장

학습목표

- 리스트의 직관적 의미를 이해한다.
- 파이썬이 기본으로 제공하는 리스트를 이해한다.
- 연결 리스트로 만든 리스트를 이해한다.
- 배열 리스트와 연결 리스트의 장단점을 이해하고, 상황에 따라 이들 중 선택할 수 있는 판단력을 기른다

01 리스트란

리스트

- '줄 세워져 있는 데이터' 또는 '죽 늘어선 데이터

그림 5-1 리스트 개념의 일상 예

ADT Abstract Data Type

- 행하는 작업의 목록으로 데이터의 타입을 나타낸 것
- Implementation detail에 신경 쓰지 않고
- 추상적 레벨에서 데이터 타입을 정의함
- "어떻게 구현할까"가 아니라 "어떻게 사용할까"에 focusing

리스트의 작업

i번 자리에 원소 x를 삽입한다

i번 원소를 삭제한다

원소 x를 삭제한다

i번 원소를 알려준다

원소 x가 몇 번 원소인지 알려준다

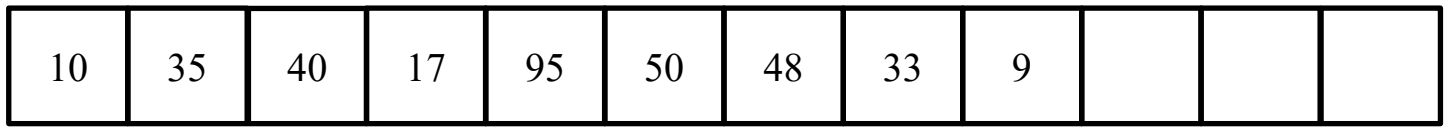
리스트의 사이즈(원소의 총 수)를 알려준다

✓ A data structure
is also a data type

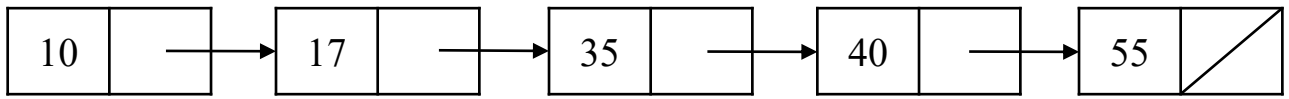
그림 5-2 ADT 리스트

리스트의 구현

- 배열로 구현할 수도 있고



연결 리스트로 구현할 수도 있다



02 배열 리스트(파이썬 내장 리스트)

리스트(배열 리스트) 객체 구조

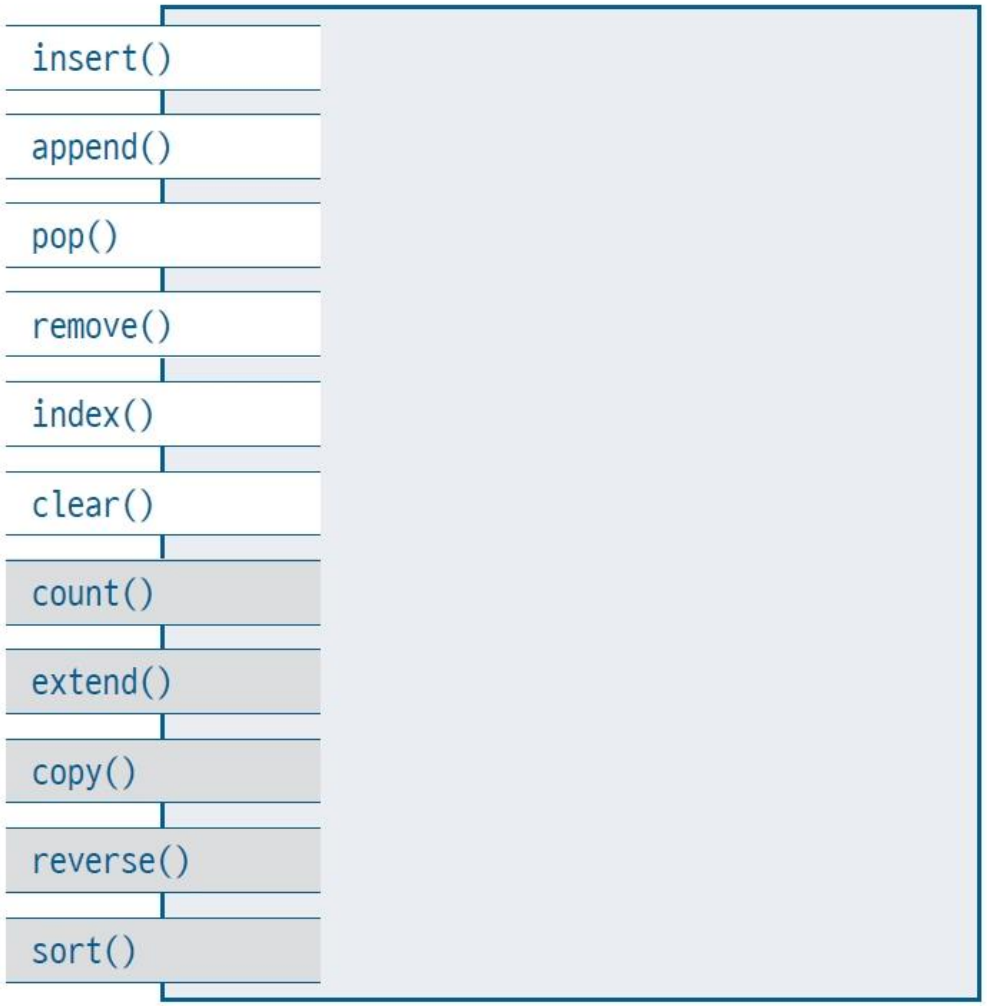


그림 5-3 리스트 객체 구조

insert(i, x)	◀ x를 리스트의 i번 원소로 삽입한다.(맨 앞자리는 0번)
append(x)	◀ 원소 x를 리스트의 맨 뒤에 추가한다.
pop(i)	◀ 리스트의 i번 원소를 삭제하면서 알려준다.
remove(x)	◀ 리스트에서 (처음으로 나타나는) x를 삭제한다.
index(x)	◀ 원소 x가 리스트의 몇 번 원소인지 알려준다.
clear()	◀ 리스트를 깨끗이 청소한다.
count(x)	◀ 리스트에서 원소 x가 몇 번 나타나는지 알려준다.
extend(a)	◀ 리스트에 나열할 수 있는 객체(예 리스트) a를 풀어서 추가한다.
copy()	◀ 리스트를 복사한다.
reverse()	◀ 리스트의 순서를 역으로 뒤집는다.
sort()	◀ 리스트의 원소들을 정렬한다.

삽입

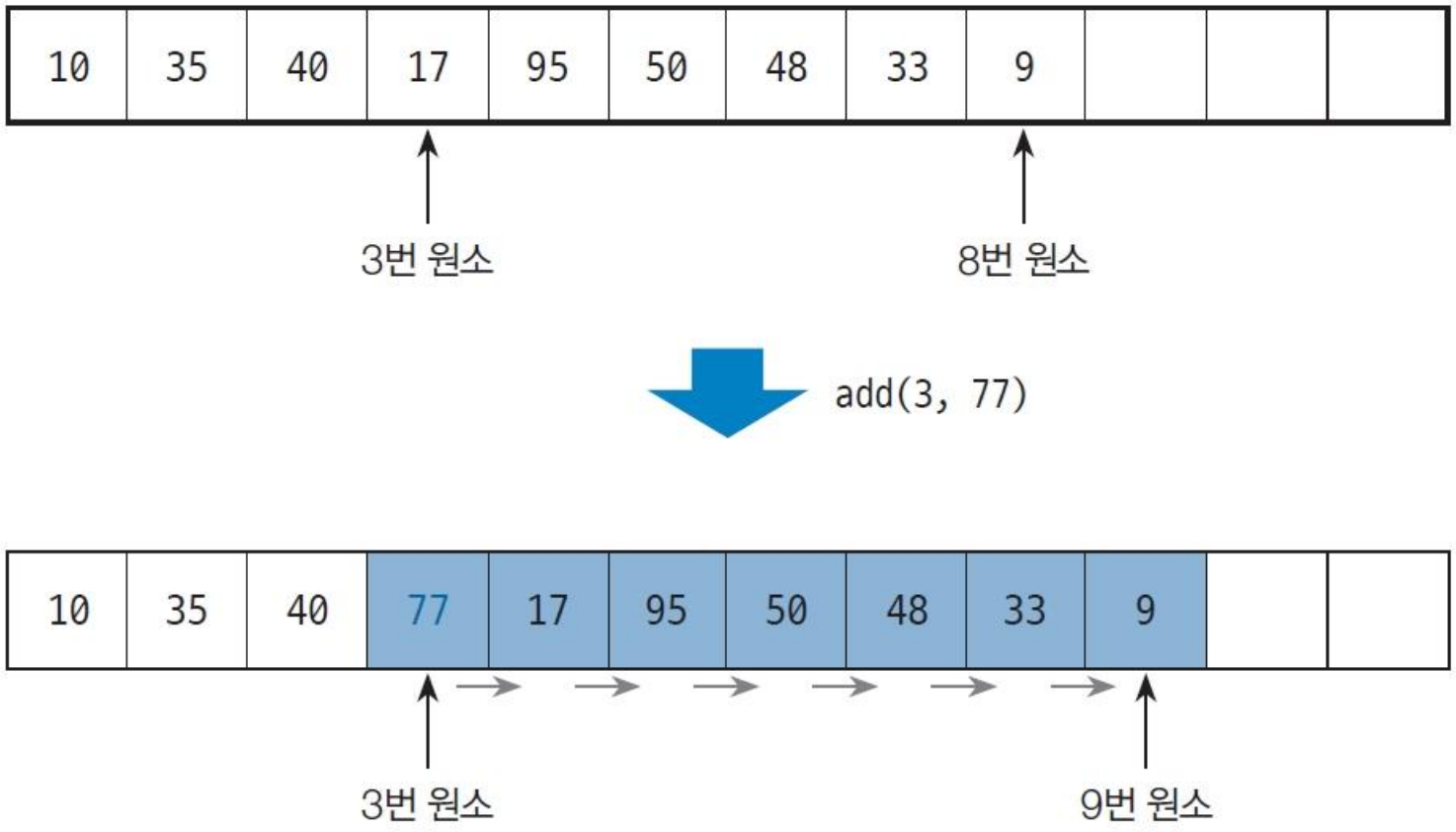
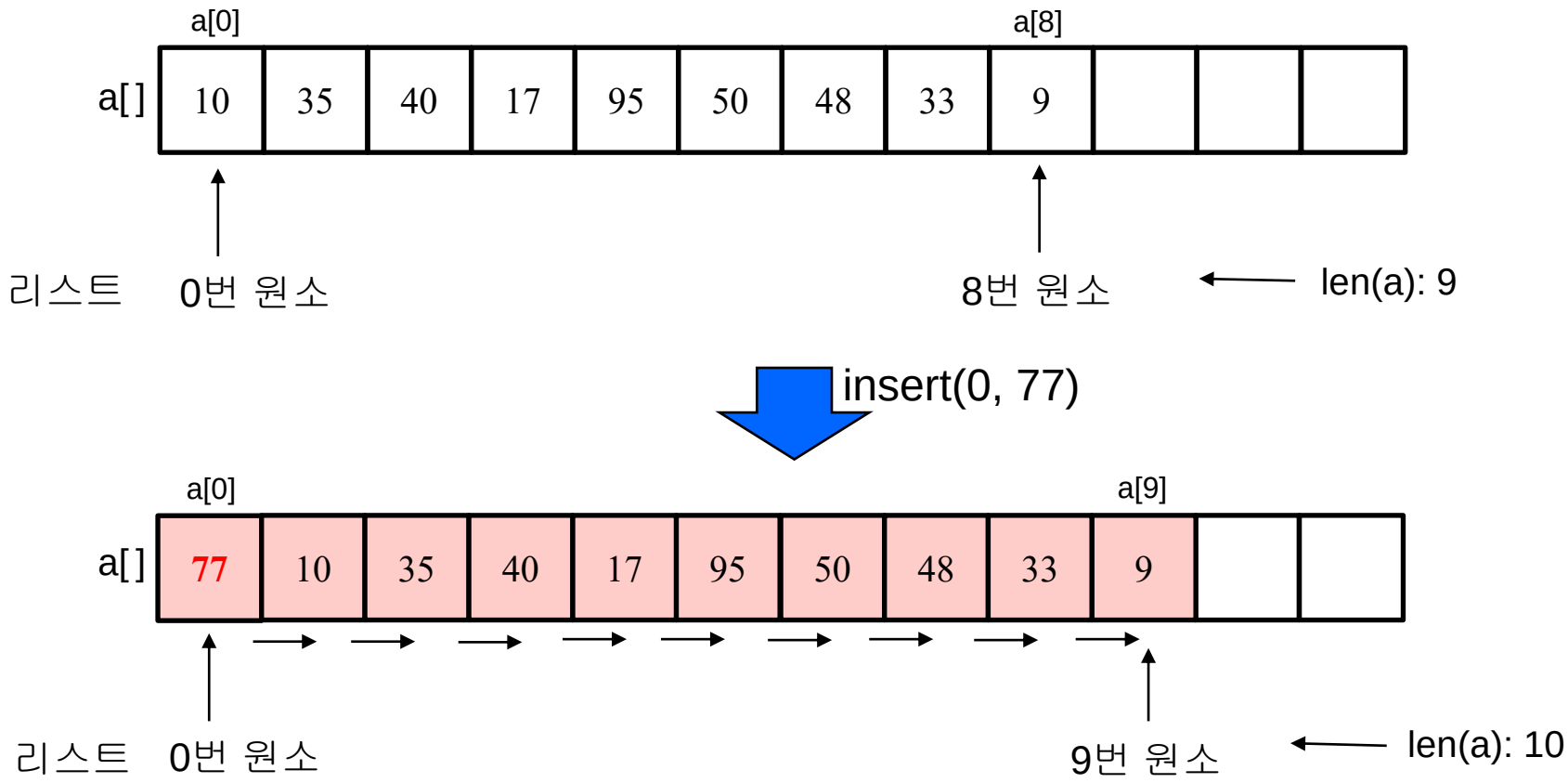
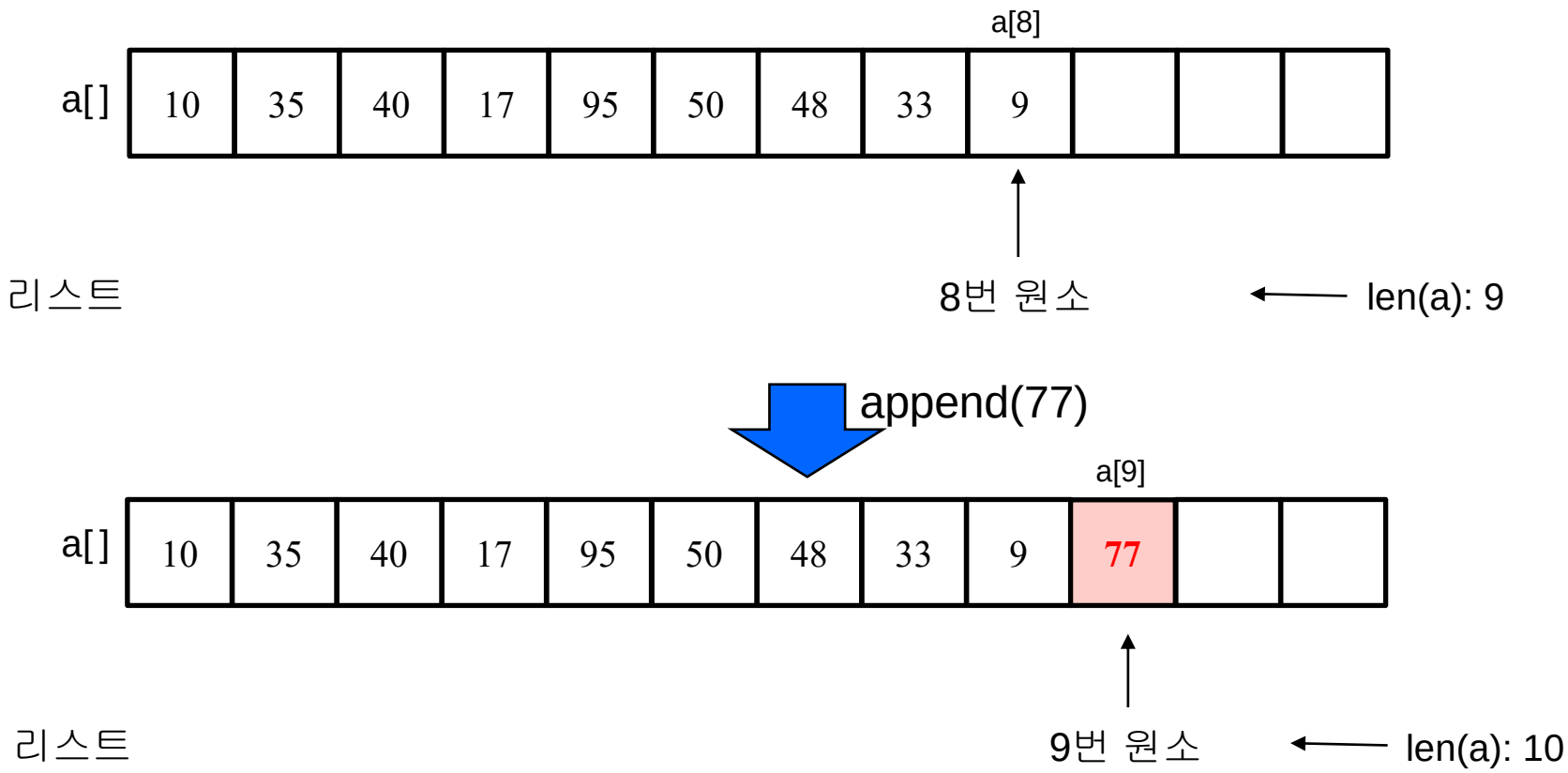


그림 5-4 배열에서 원소 삽입 후 원소들을 시프트하는 예

삽입 효율이 가장 나쁜 예


a[0]
a[8]
a[ ]
10
35
40
17
95
50
48
33
9
리스트
0번 원소
8번 원소
len(a): 9
insert(0, 77)
a[0]
a[9]
a[ ]
77
10
35
40
17
95
50
48
33
9
리스트
0번 원소
9번 원소
len(a): 10

삽입 효율이 가장 좋은 예


a[8]
a[]
10
35
40
17
95
50
48
33
9
리스트
8번 원소
len(a): 9
append(77)
a[9]
a[]
10
35
40
17
95
50
48
33
9
77
리스트
9번 원소
len(a): 10

삭제

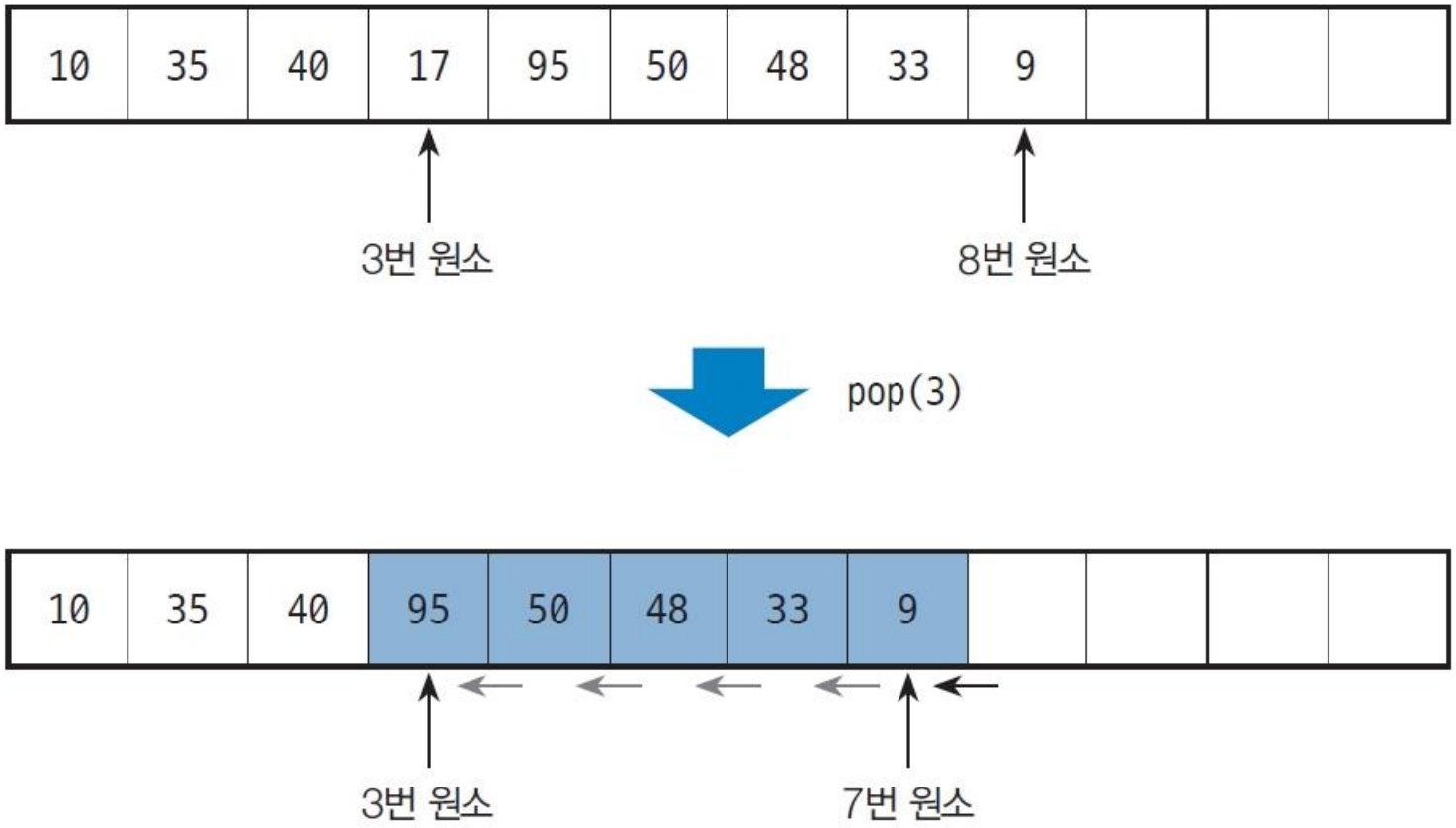
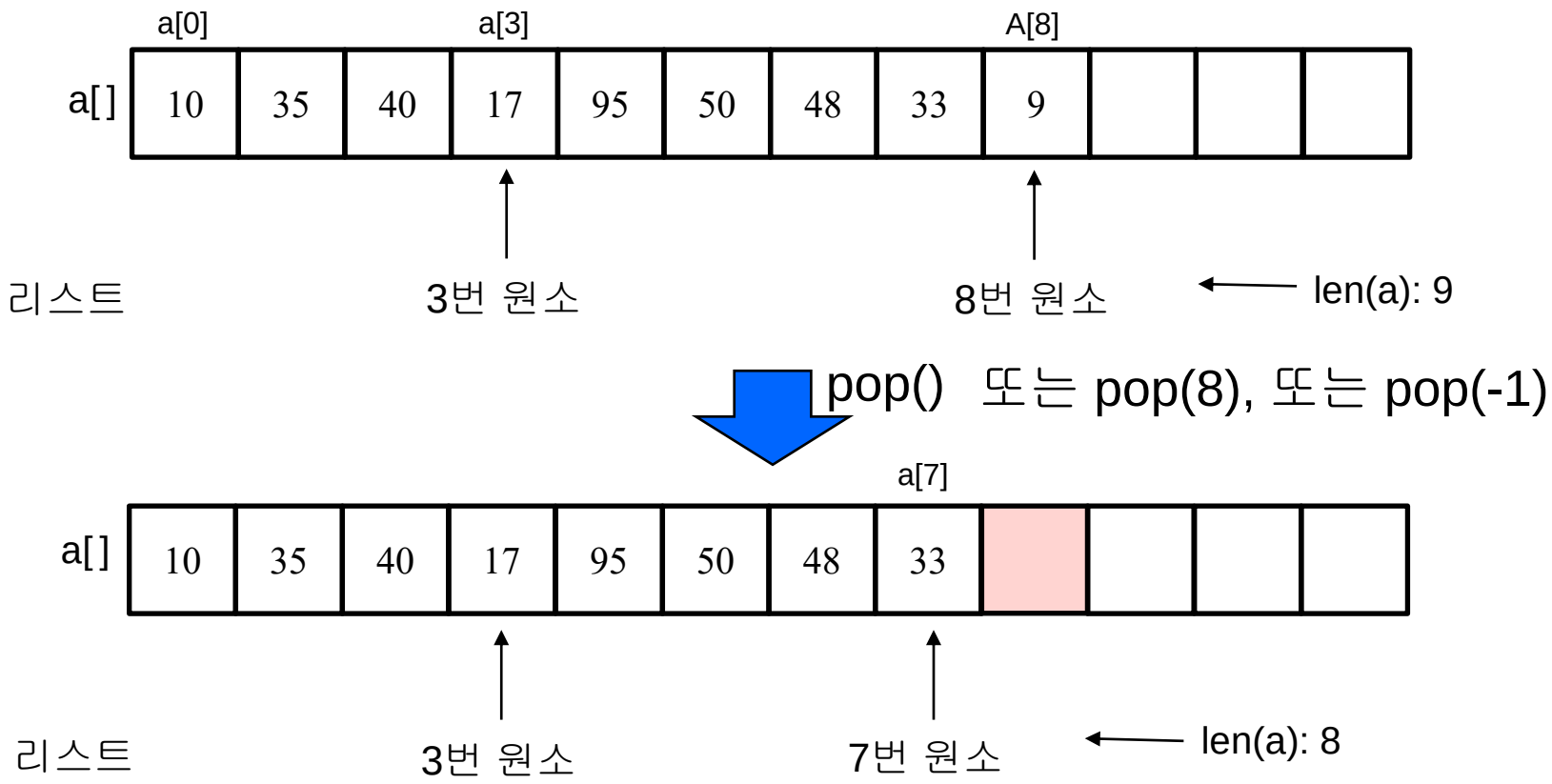


그림 5-5 배열에서 원소 삭제 후 원소들을 시프트하는 예

맨 끝 원소 삭제 방법은 3가지



메서드 사용 방법

원소 삽입

❶ [1, 3, 5, 6, 7] → a.insert(1, 'Mon') → [1, 'Mon', 3, 5, 6, 7]

❷ [1, 3, 5, 6, 7] → a.append('Mon') → [1, 3, 5, 6, 7, 'Mon']

원소 삭제

❶ [1, 3, 5, 6, 7] → a.pop(3) → [1, 3, 5, 7]

❷ [1, 3, 5, 6, 7] → a.pop(4) → [1, 3, 5, 6]

❸ [1, 3, 5, 6, 7] → a.pop() → [1, 3, 5, 6]

❹ [1, 3, 5, 6, 7] → a.pop(-1) → [1, 3, 5, 6]

❺ [1, 3, 5, 6, 7] → a.pop(-2) → [1, 3, 5, 7]

❻ [1, 3, 5, 6, 7] → a.remove(5) → [1, 3, 6, 7]

메서드 사용 방법

- 기타 작업

```
[1, 3, 5, 6, 7] → a.index(5) → 2

[1, 3, 5, 6, 7] → a.index(1) → 0

[1, 3, 5, 6, 7] → a.clear() → []

[1, 3, 5, 6, 7] → a.extend([10, 20]) → [1, 3, 5, 6, 7, 10, 20]

[1, 3, 5, 6, 7] → a.reverse() → [7, 6, 5, 3, 1]

[2, 4, 1, 5, 7] → a.sort() → [1, 2, 4, 5, 7]
```

파이썬 내장 리스트의 한계

하부가 배열로 구현됨

배열의 단점

- 연속된 공간에 저장되므로 삽입이나 삭제 시 시프트 작업이 필요하다
- 미리 크기를 정해야 하므로 오버플로우 시 배열을 새로 배정받아 내용을 복사해야 한다

But, 파이썬에서 자동으로 관리해주어 사용자가 이를 신경 쓸 필요는 없다

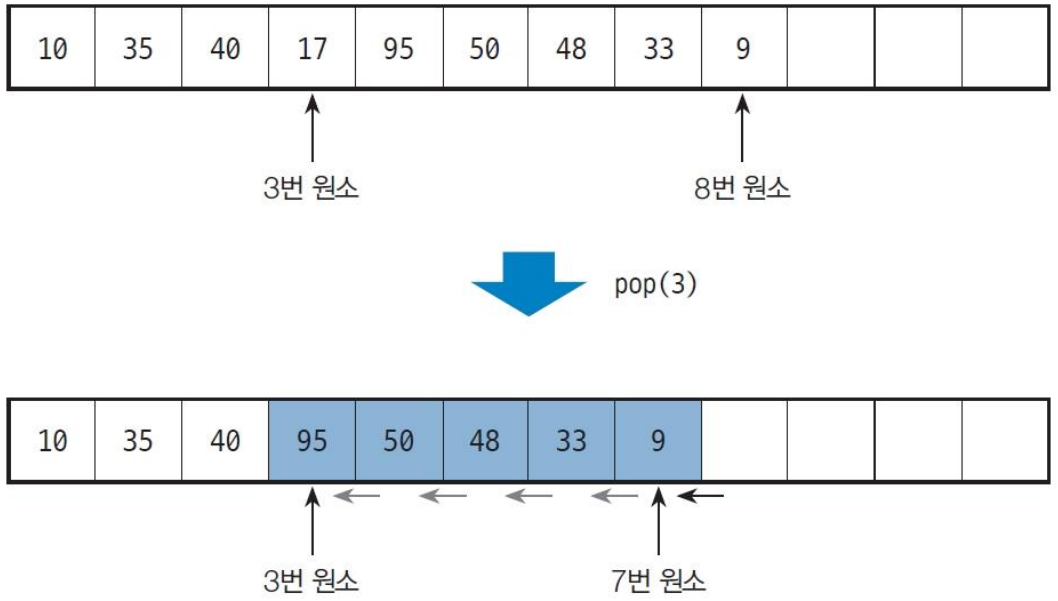


그림 5-5 배열에서 원소 삭제 후 원소들을 시프트하는 예

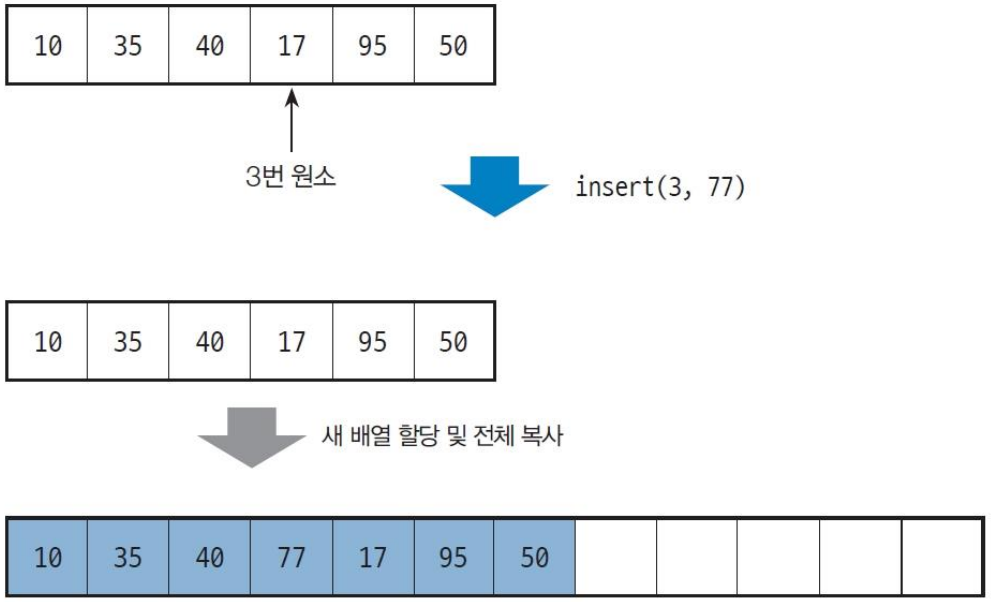
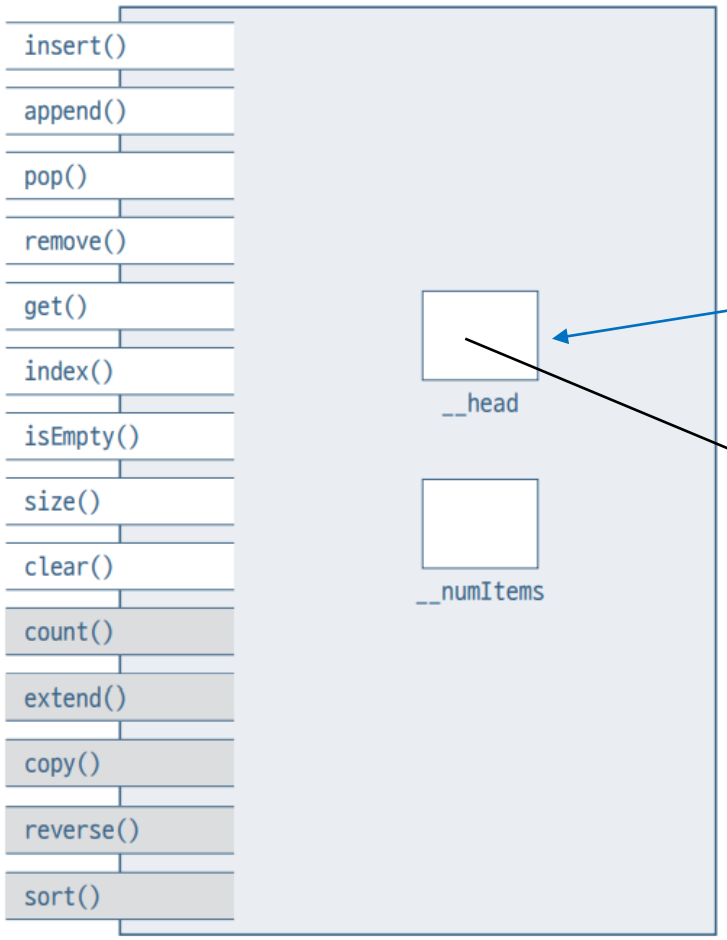


그림 5-6 배열이 꽉 찬 상태에서 삽입이 시도될 때 새 배열에 원소들을 모두 복사하는 예

03 연결 리스트

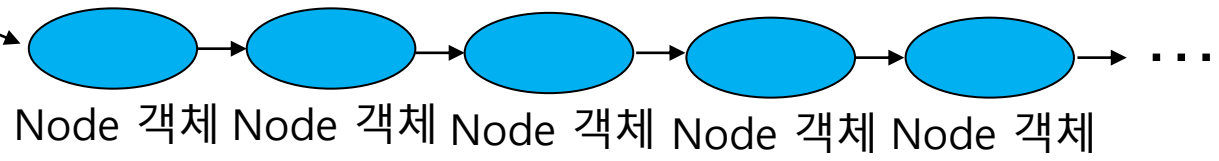
Linked List 객체 구조

강의 노트에서 문맥상 혼동의 여지가 없으므로
__head와 head, __numItems와 numItems를 섞어서 씀



Linked List 객체

Linked list의
시작 노드 레퍼런스만 있으면 된다



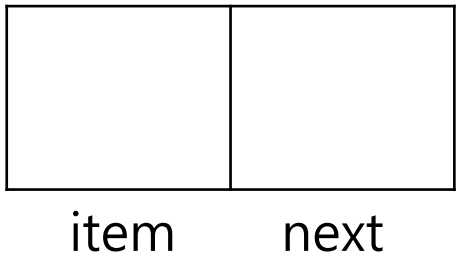
Linked List

각 작업의 의미

__head	◀ 첫 번째 노드에 대한 레퍼런스
__numItems	◀ 연결 리스트에 들어 있는 원소의 총 수
insert(i, x)	◀ x를 연결 리스트의 i번 원소로 삽입한다.(맨 앞자리는 0번)
append(x)	◀ 연결 리스트의 맨 뒤에 원소 x를 추가한다.
pop(i)	◀ 연결 리스트의 i번 원소를 삭제하면서 알려준다.
remove(x)	◀ 연결 리스트에서 (처음으로 나타나는) x를 삭제한다.
get(i)	◀ 연결 리스트의 i번 원소를 알려준다.
index(x)	◀ 원소 x가 연결 리스트의 몇 번 원소인지 알려준다.
isEmpty()	◀ 연결 리스트가 빈 리스트인지 알려준다.
size()	◀ 연결 리스트의 총 원소 수를 알려준다
clear()	◀ 연결 리스트를 깨끗이 청소한다.
count(x)	◀ 연결 리스트에서 원소 x가 몇 번 나타나는지 알려준다.
extend(a)	◀ 연결 리스트에 나열할 수 있는 객체(예 리스트, 튜플 등) a를 풀어서 추가한다.
copy()	◀ 연결 리스트를 복사해서 새 연결 리스트를 리턴한다.
reverse()	◀ 연결 리스트의 순서를 역으로 뒤집는다.
sort()	◀ 연결 리스트를 정렬한다.

ListNode 객체 구조

보통은 이렇게 그린다



노드 클래스

```
class ListNode:
        def __init__(self, newItem, nextNode:'ListNode'):
                self.item = newItem
                self.next = nextNode
```

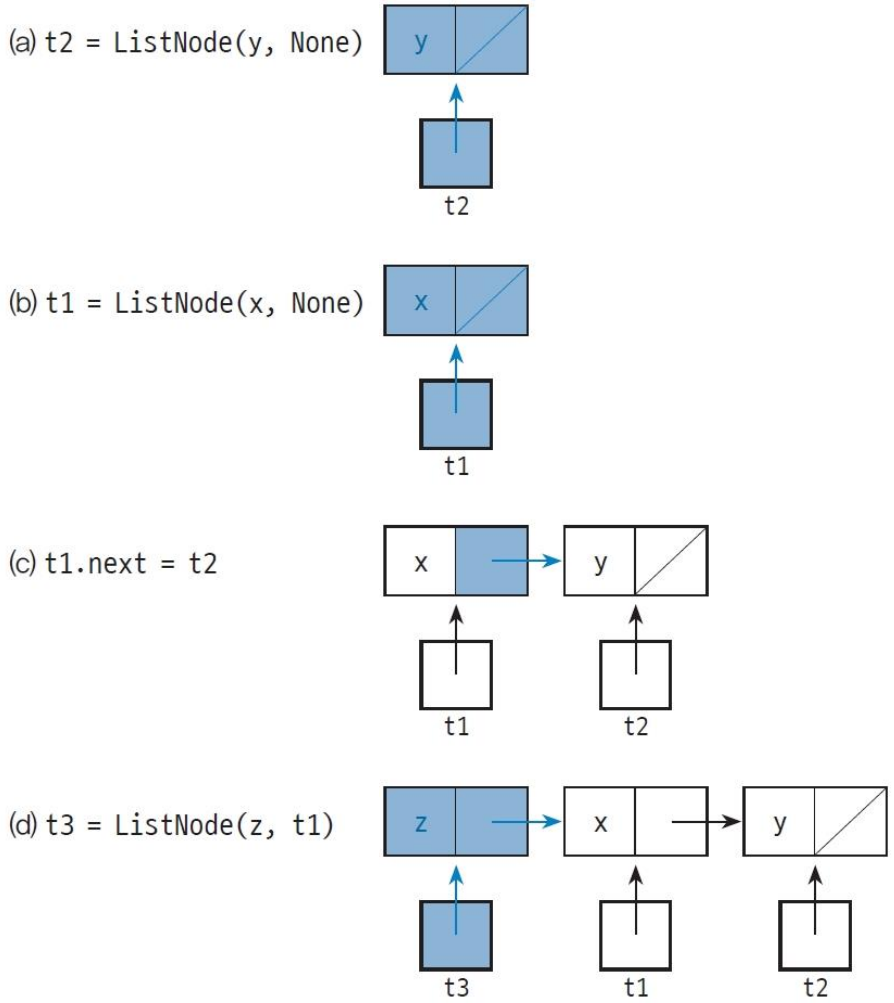
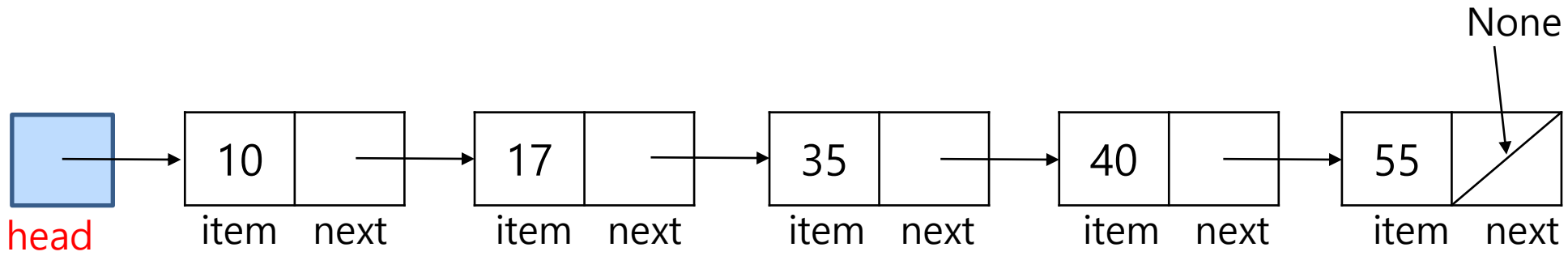


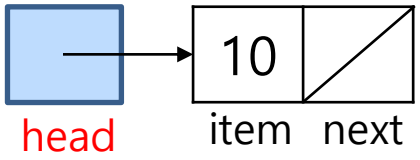
그림 5-19 클래스 ListNode의 사용 예

헤드 노드Head Node

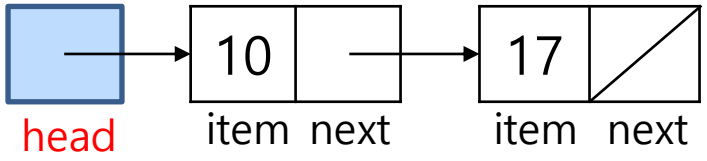
- 연결 리스트는 흔히 첫 노드에 대한 레퍼런스(여기서는 head)를 갖고 있다




```
head = ListNode(10, None)
```



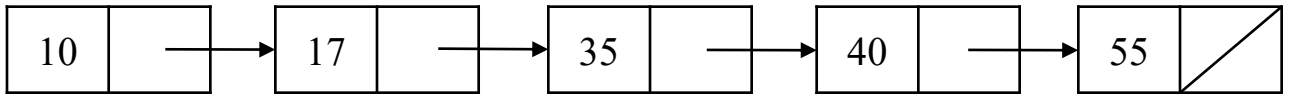

```
head.next = ListNode(17, None)
```



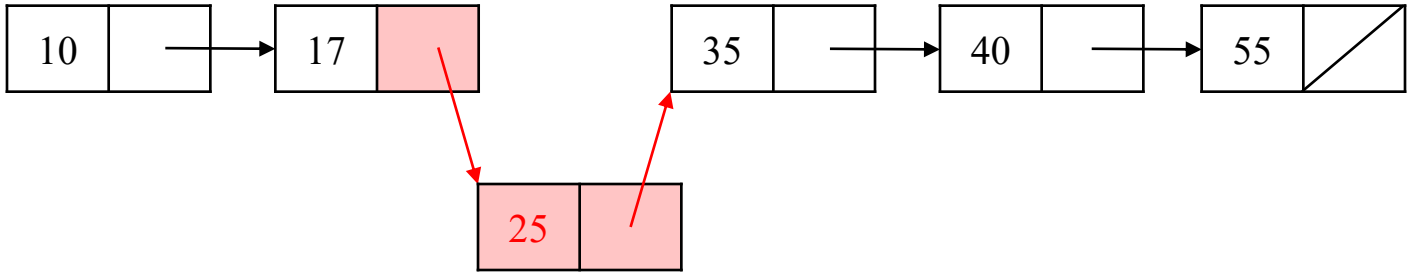
✓ 여기서, head는 단순한 레퍼런스 변수

핵심 작업

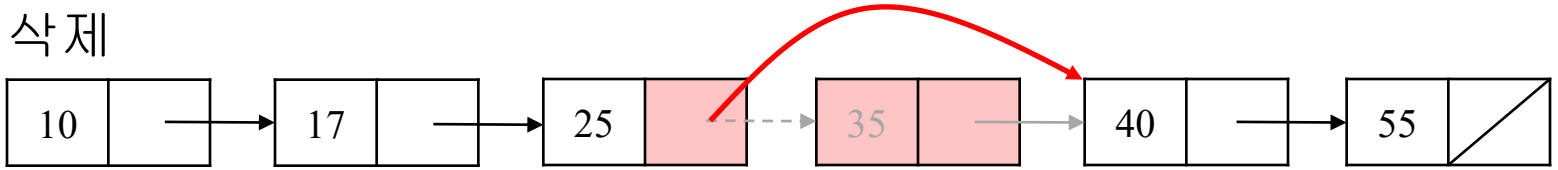
정수의 연결 리스트



삽입

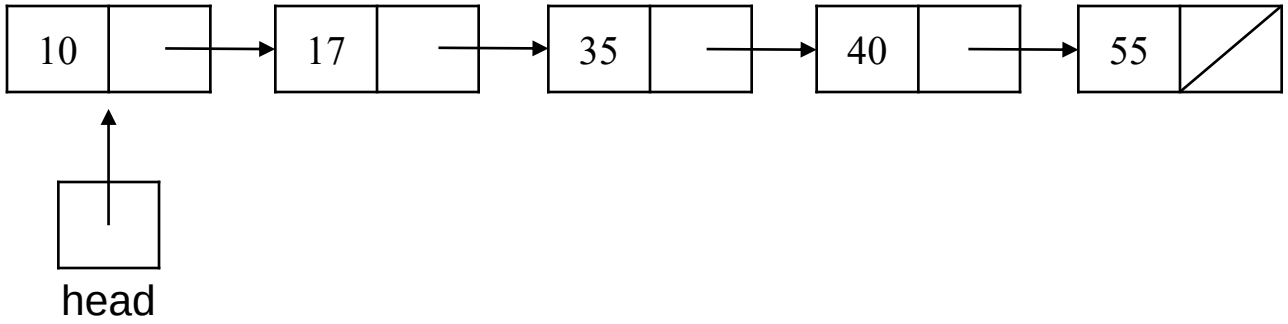


삭제



전형적 모양

대표적 상태



초기 상태

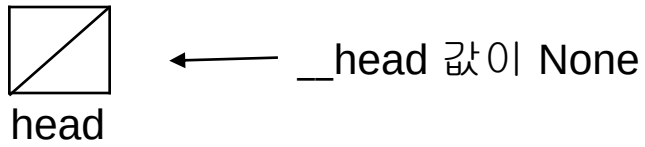


그림 **5-8** 연결 리스트의 일반 형태

삽입Insertion

prev 다음에 새 노드를 삽입한다

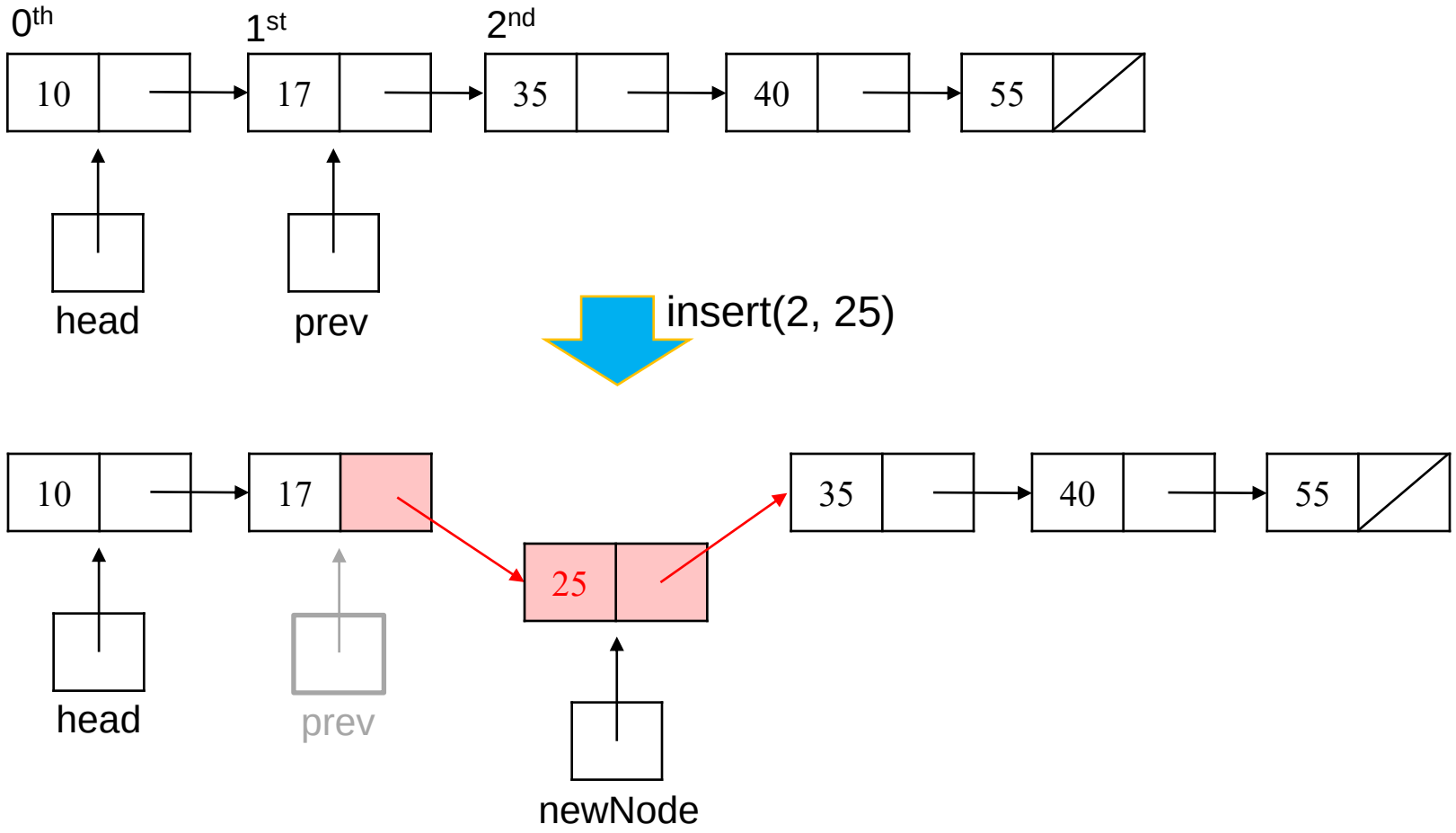
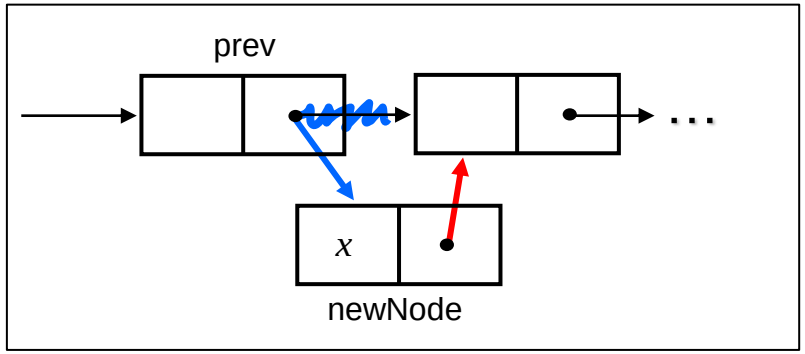


그림 **5-10** 연결 리스트에서 중간에 원소(25)를 삽입하는 예

insert()

```
newNode.item ← x
newNode.next ← prev.next
prev.next ← newNode
__numItems += 1
```

중간 삽입과 맨 끝 삽입은 문제 없다.
But, 맨 앞 삽입은 작동하지 않는다.



Animation

중간에 삽입: Okay

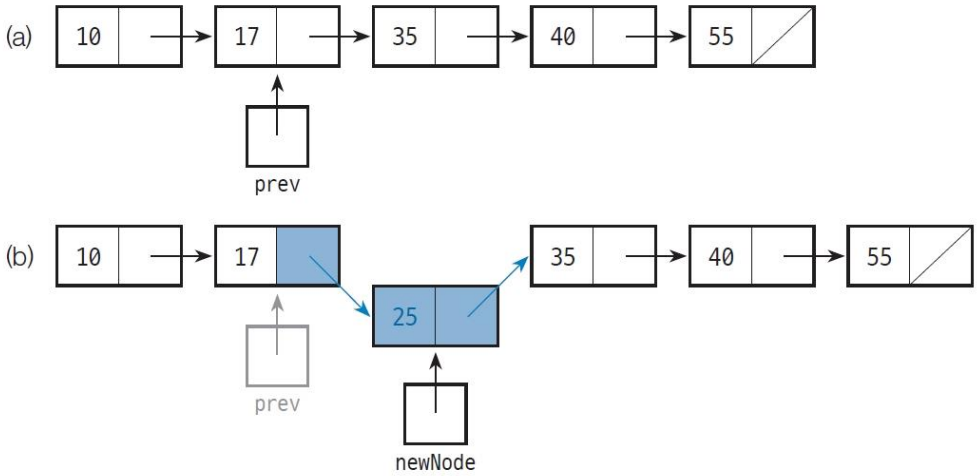


그림 5-10 연결 리스트에서 중간에 원소(25)를 삽입하는 예

맨 끝에 삽입: Okay

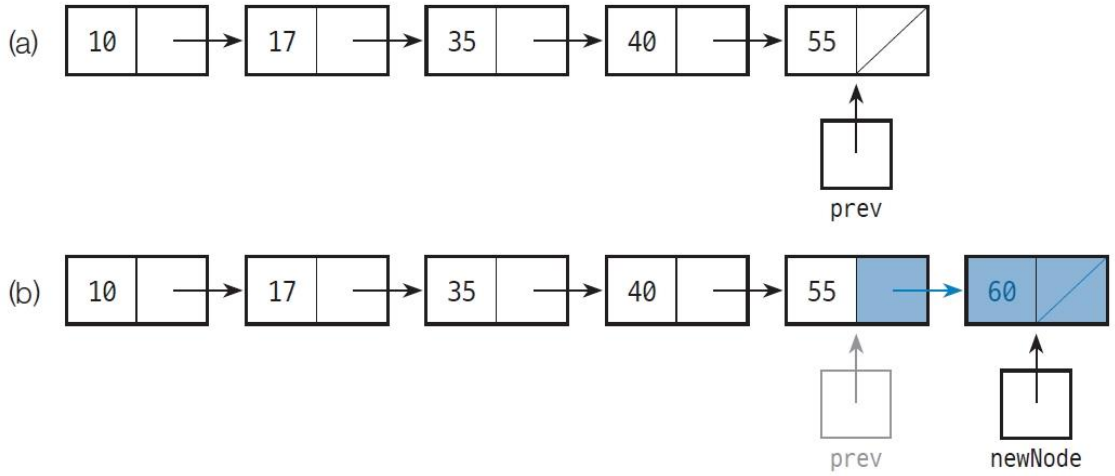
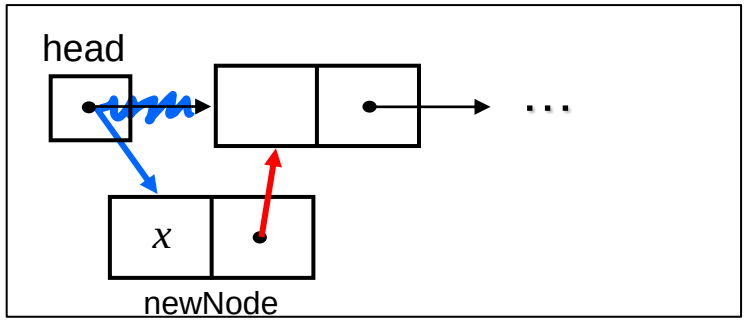


그림 5-11 연결 리스트의 맨 뒤에 원소(60)를 삽입하는 예

newNode.item ← *x*
newNode.next ← head
head ← newNode
numItems++

맨 앞에 삽입할 때

prev가 존재하지 않아 prev.next와 같은 표현 사용불가



Animation

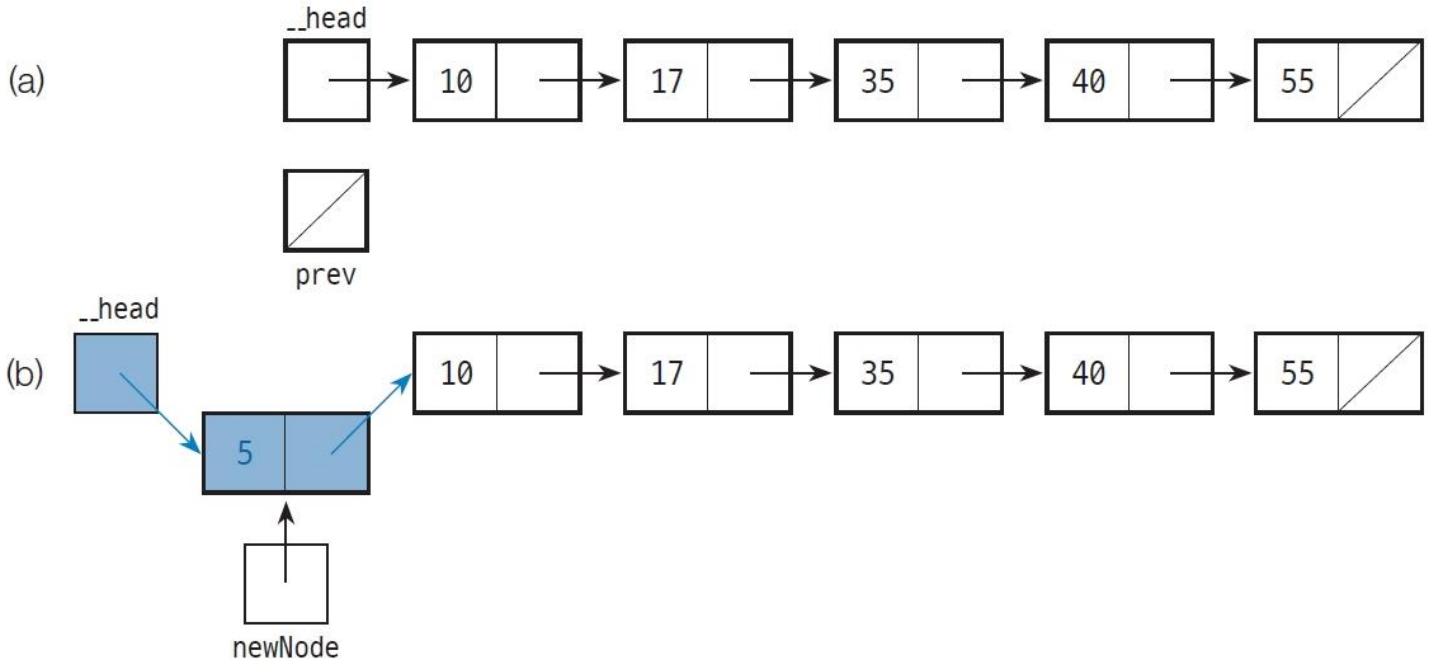


그림 5-12 연결 리스트의 맨 앞에 원소(5)를 삽입하는 예

알고리즘 5-1 연결 리스트에 원소 삽입하기

```
if i == 0:
    newNode.item = x
    newNode.next = __head
    __head = newNode
    __numItems += 1
else:
    newNode.item = x
    newNode.next = prev.next
    prev.next = newNode
    __numItems += 1
```

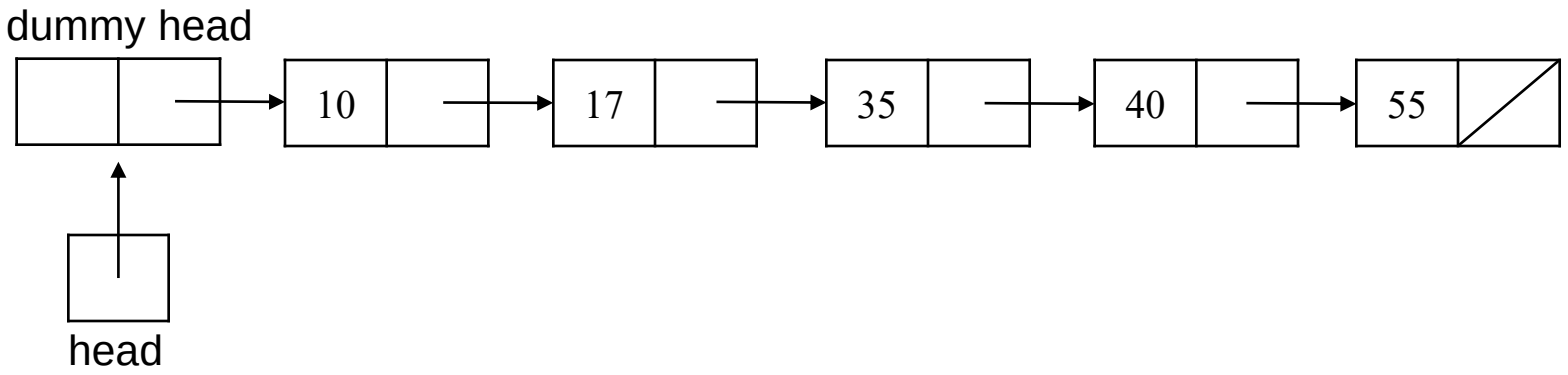
← 앞의 두 가지 경우를 다 고려한

자주 나오지 않는 "맨 앞 삽입" 때문에
이렇게 항상 두 가지 경우로 나누어 처리해야 하는가?

하나로 처리할 수 있는 방법이 있다 → 더미 헤드 노드dummy head node

더미 헤드를 둔 연결 리스트

대표적 상태



초기 상태

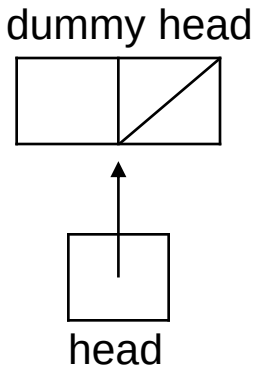


그림 **5-13** 더미 헤드 노드를 둔 연결 리스트

insert(): 더미 헤드가 있는 버전

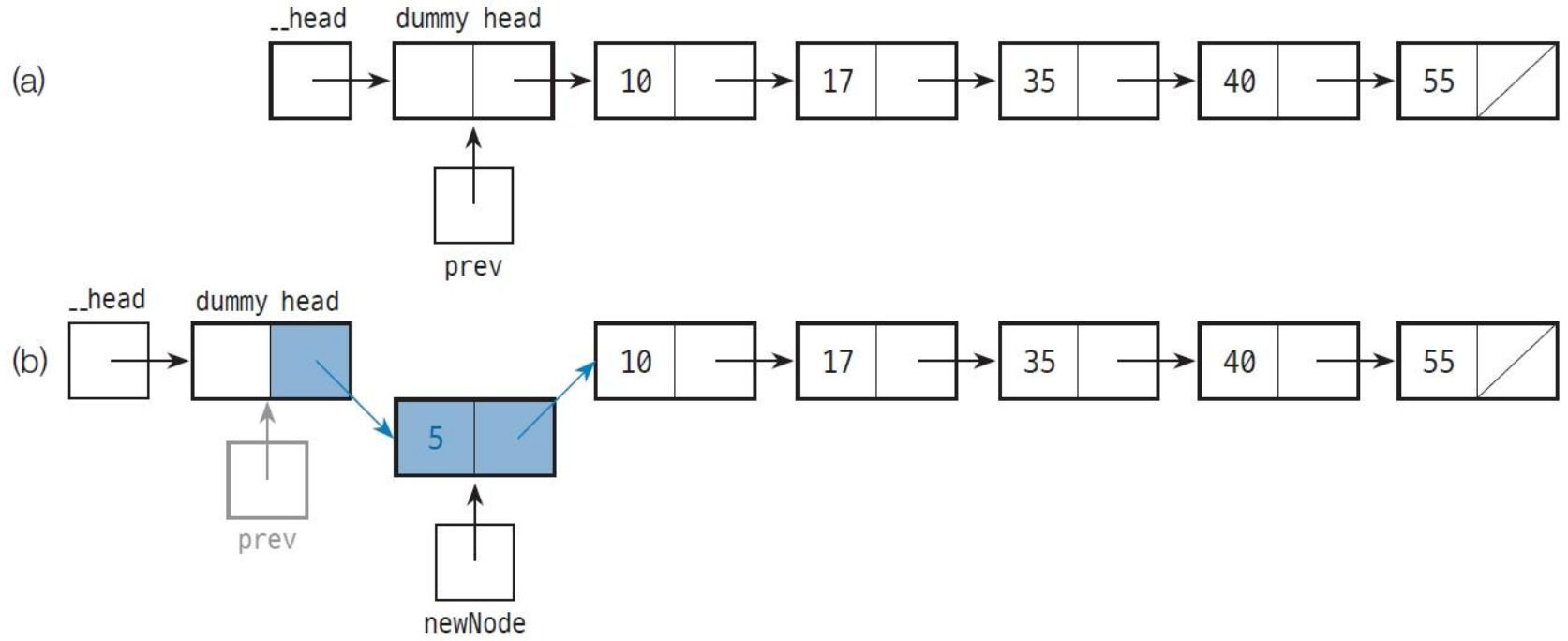


그림 5-14 더미 헤드 노드가 있는 연결 리스트에서 맨 앞에 원소를 삽입하는 예

알고리즘 5-2 연결 리스트에 원소 삽입하기(더미 헤드를 두는 대표 버전)

```
newNode.item = x
newNode.next = prev.next
prev.next = newNode
__numItems += 1
```

← Dummy head를 두면 모든 삽입은 이것으로 충분

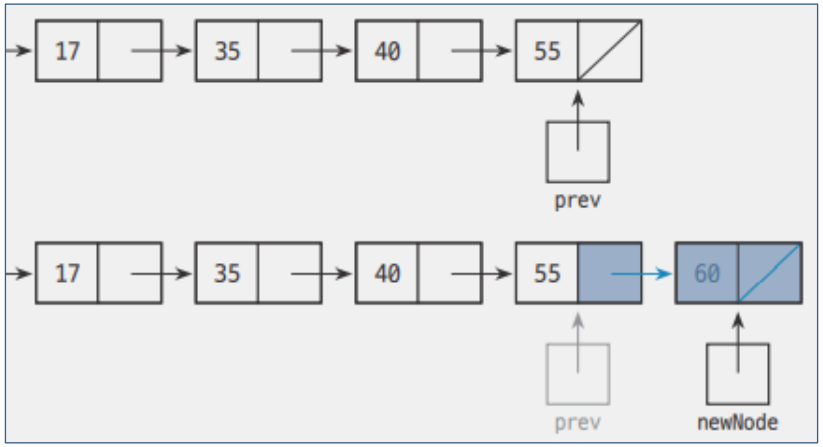
항상 prev가 존재하기 때문

append()

맨 뒤에 원소 x를 추가

```
newNode.item ← x
newNode.next ← prev.next
prev.next ← newNode
__numItems += 1
```

prev가 정해지면
insert()의 코드와 동일



삭제Deletion

prev 다음 노드를 삭제한다

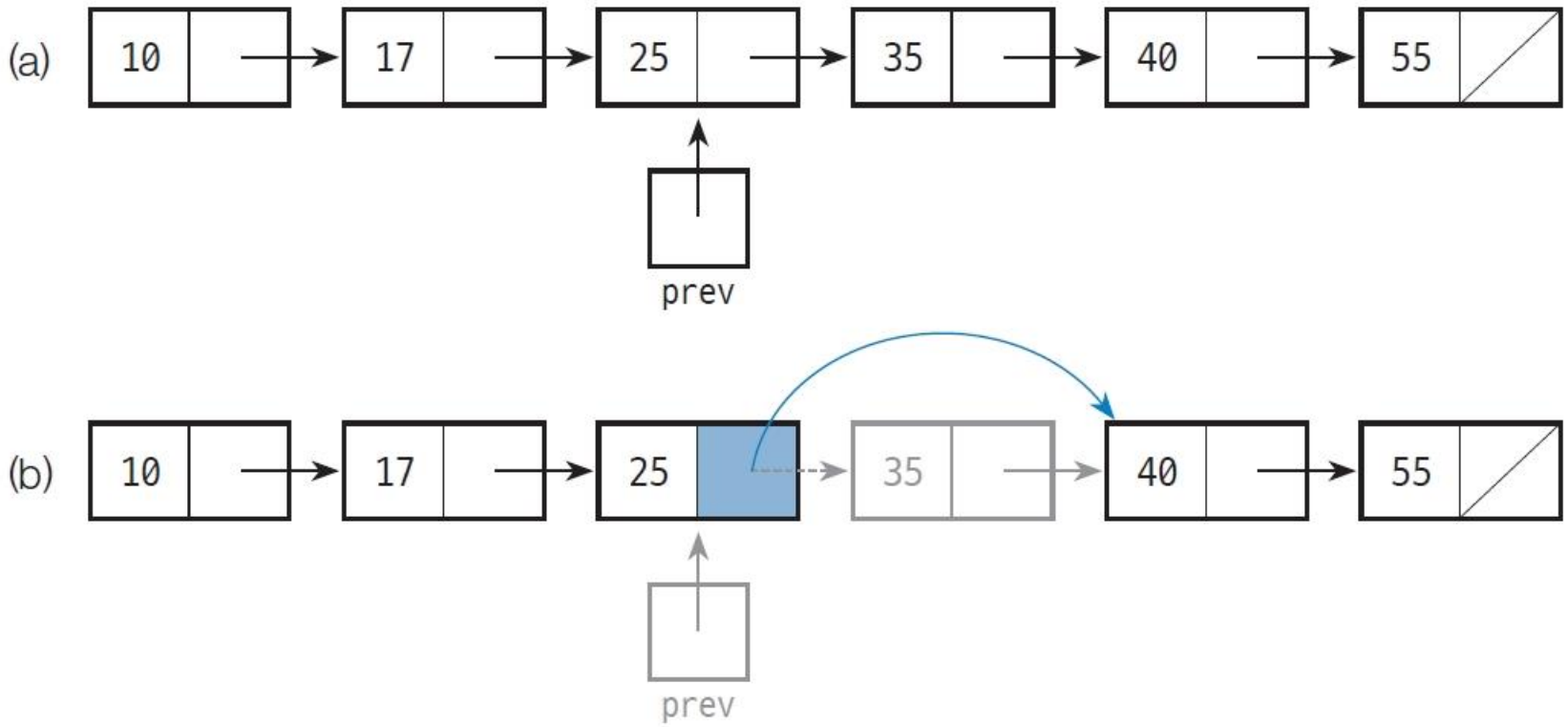
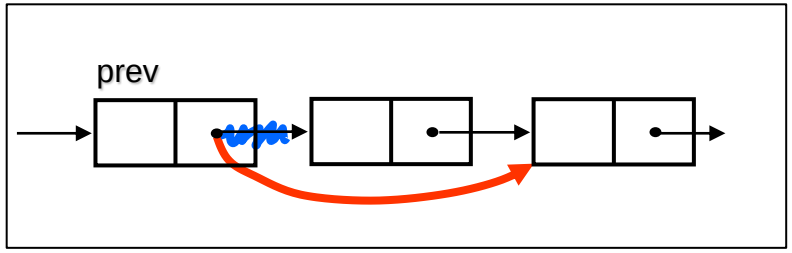


그림 5-15 연결 리스트에서 원소(35)를 삭제하는 예

pop()

prev.next ← prev.next.next
numItems--

중간 노드 삭제와 맨 끝 노드 삭제는 Okay
But, 맨 앞 노드 삭제는 작동하지 않는다



Animation

중간 노드 삭제: Okay

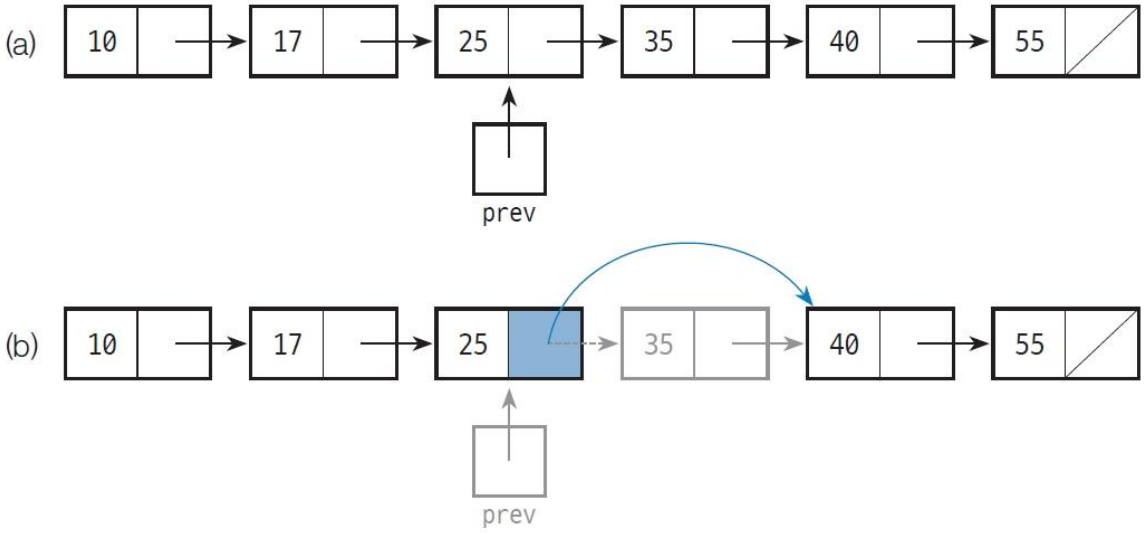


그림 5-15 연결 리스트에서 원소(35)를 삭제하는 예

맨 끝 노드 삭제: Okay

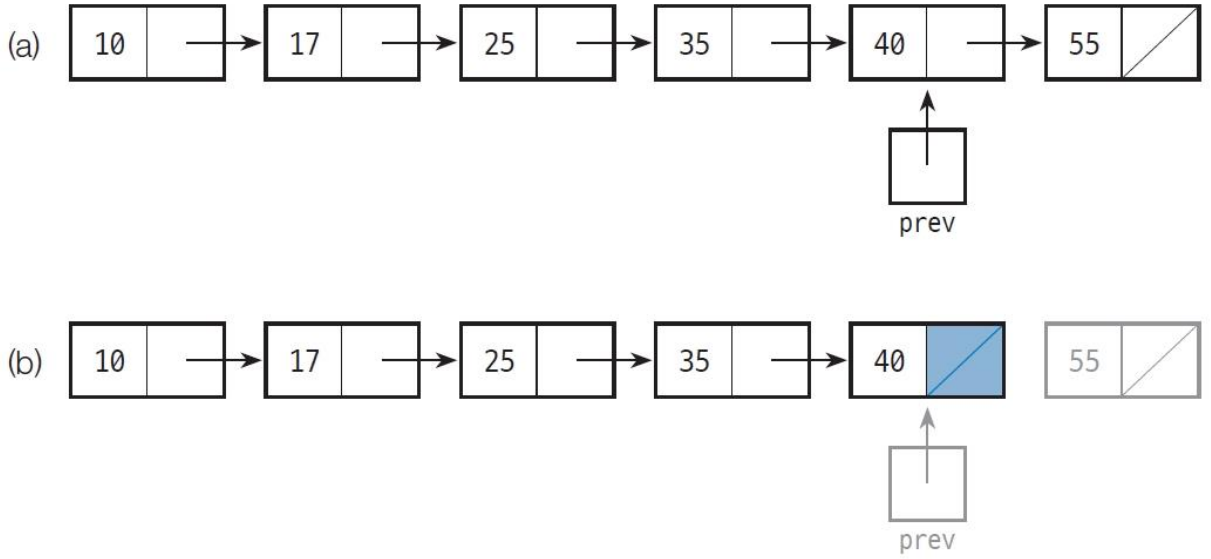
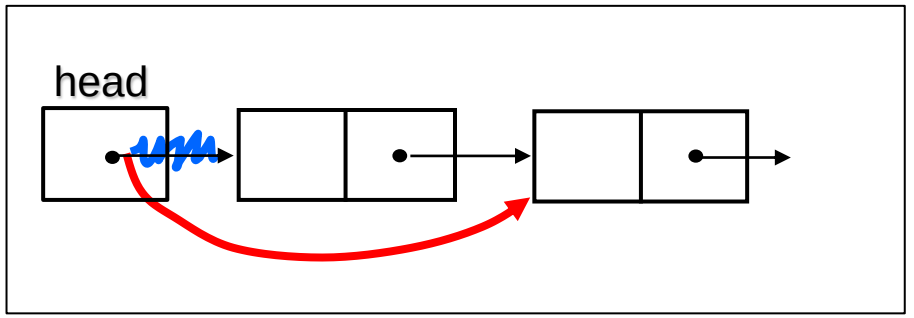


그림 5-16 연결 리스트의 마지막 원소를 삭제하는 예



Animation

head ← head.next
__numItems –= 1

맨 앞 노드를 삭제할 때

prev가 존재하지 않아 prev.next와 같은 표현 사용불가

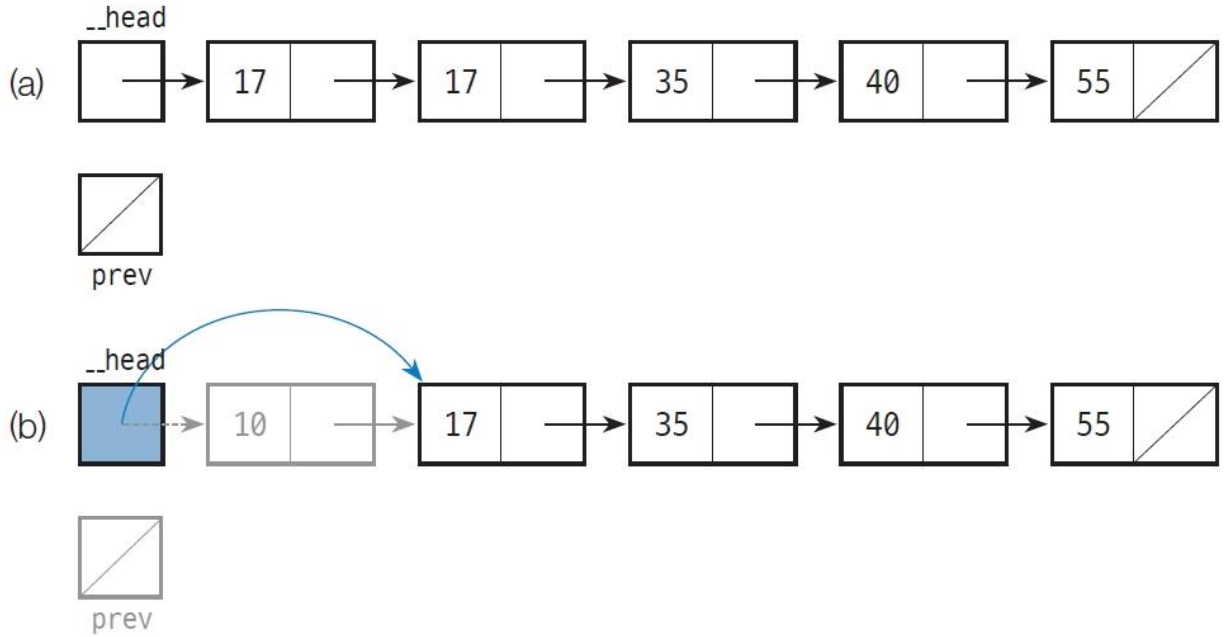


그림 5-17 연결 리스트의 맨 앞쪽 원소를 삭제하는 예

알고리즘 5-3 연결 리스트의 원소 삭제하기

```
if i == 0:
    __head.next = __head.next.next
    __numItems -= 1
else:
    prev.next = prev.next.next
    __numItems -= 1
```

← 앞의 두 가지 경우를 다 고려한

자주 나오지 않는 "맨 앞 삭제" 때문에
이렇게 항상 두 가지 경우로 나누어 처리해야 하는가?

더미 헤드를 두면 역시 하나로 처리된다

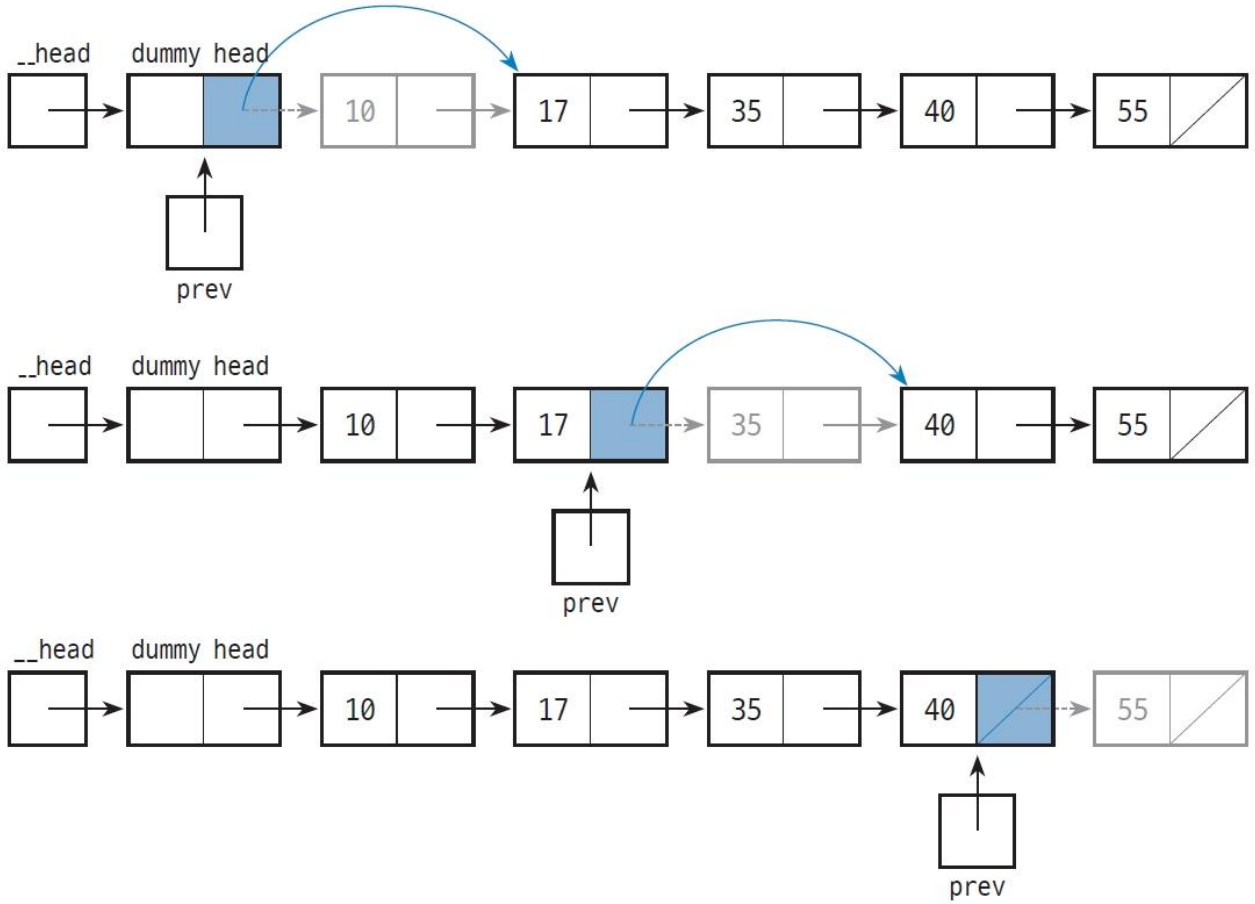


그림 5-18 더미 헤드 노드가 있는 연결 리스트의 세 가지 삭제 예

알고리즘 5-4 연결 리스트의 원소 x 삭제하기(더미 헤드를 두는 대표 버전)

```
prev.next = prev.next.next
__numItems -= 1
```

← Dummy head를 두면 이것으로 충분

항상 prev가 존재하기 때문

기타 작업

알고리즘 5-5 연결 리스트의 i번 원소 알려주기

```
get(i):
    if i >= 0 and i <= __numItems - 1:
        return __getNode(i).item
    else:
        print("error in get(", i, ")")
```

알고리즘 5-6 연결 리스트의 i번 노드 알려주기

```
__getNode(i):
    curr = __head
    for index in range(i+1):
        curr = curr.next
    return curr
```

알고리즘 5-7 x가 연결 리스트의 몇 번 원소인지 알려주기

```
index(x):
    curr = __head  # 더미 헤드
    for index in range(__numItems):
        curr = curr.next
        if curr.item == x:
            return index
    return -12345  # x가 없을 때
```

알고리즘 5-8 기타 작업들

```
isEmpty():                ◀ 리스트가 비었는지 알려주기
    return __numItems == 0

size():                   ◀ 리스트의 총 원소 수 알려주기
    return __numItems

clear():                  ◀ 리스트 깨끗이 청소하기
    newNode.next = None   ◀ 더미 헤드 노드
    __head = newNode
    __numItems = 0
```

연결 리스트 클래스 구조

```
from DS.list.listNode import ListNode # 클래스 ListNode가 다른 파일에 있을 경우

class LinkedListBasic:
    def __init__(self):
        self.__head = ListNode('dummy', None)
        self.__numItems = 0

    ❶ def insert(self, i, newItem):
        ...
    ❷ def append(self, newItem):
        ...
    ❸ def pop(self, i):
        ...
    ...
```

그림 5-20 연결 리스트 클래스 구조 (linkedListBasic.py)

파이썬 코드


```
def insert(self, i:int, x):
    if i >= 0 and i <= self. __numItems:
        prev = self.__getNode(i - 1)
        newNode = ListNode(x, prev.next)
        prev.next = newNode
        self.__numItems += 1
    else:
        print("index", i, ": out of bound in insert()") # 필요 시 에러 처리
```

```
def __getNode(self, i:int) -> ListNode:
    curr = self.__head  # dummy head, index: -1
    for index in range(i+1):
        curr = curr.next
    return curr
```

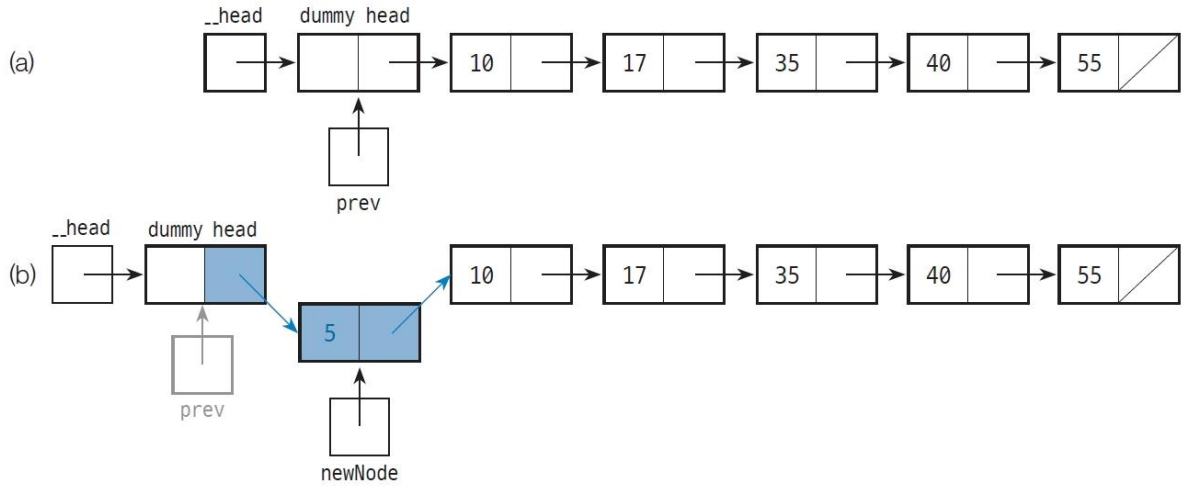


그림 5-14 더미 헤드 노드가 있는 연결 리스트에서 맨 앞에 원소를 삽입하는 예

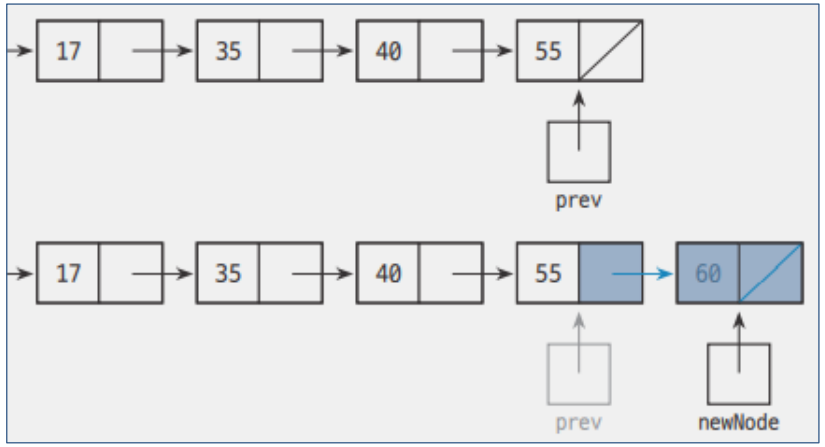
알고리즘 5-2 연결 리스트에 원소 삽입하기(더미 헤드를 두는 대표 버전)

```
newNode.item = x
newNode.next = prev.next
prev.next = newNode
__numItems += 1
```

핵심부 (유사 코드)

맨 끝 추가

```
def append(self, newItem):
        prev = self.__getNode(self.__numItems - 1)
        newNode = ListNode(newItem, prev.next)
        prev.next = newNode
        self.__numItems += 1
```



알고리즘 5-2 연결 리스트에 원소 삽입하기(더미 헤드를 두는 대표 버전)

```
newNode.item = x
newNode.next = prev.next
prev.next = newNode
__numItems += 1
```

핵심부 (유사 코드)

파이썬 코드

삭제

```
def pop(self, i:int):   # i번 노드 삭제
        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
```

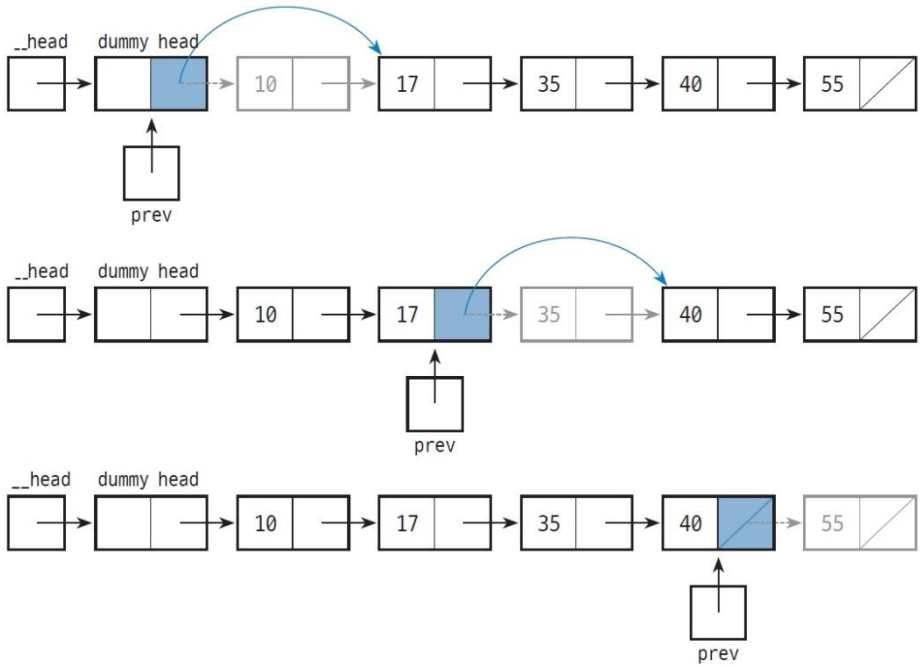


그림 5-18 더미 헤드 노드가 있는 연결 리스트의 세 가지 삭제 예

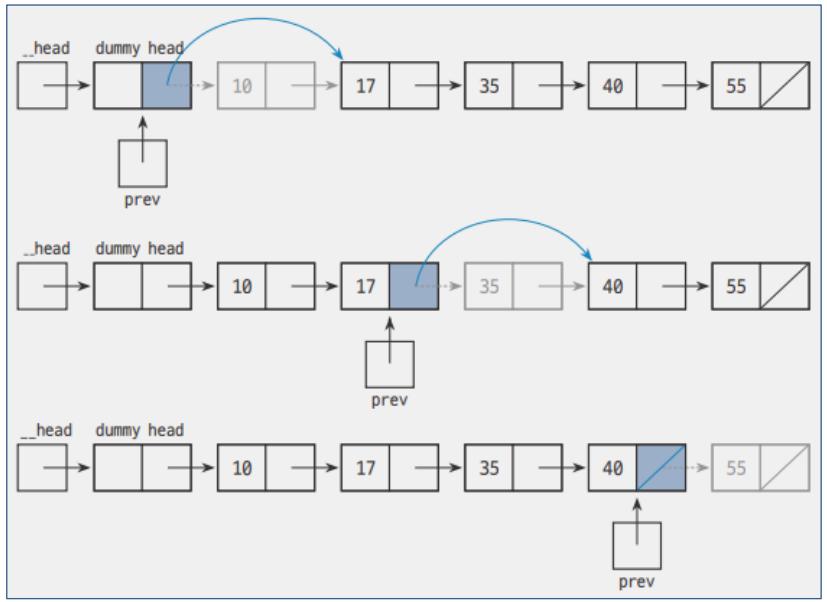
알고리즘 5-4 연결 리스트의 원소 x 삭제하기(더미 헤드를 두는 대표 버전)

```
prev.next = prev.next.next
__numItems  -= 1
```

핵심부 (유사 코드)

```
def remove(self, x):
        (prev, curr) = self.__findNode(x)
        if curr != None:
                prev.next = curr.next
                self.__numItems -= 1
                return x
        else:
                return None
```

```
def __findNode(self, i:int) -> (ListNode, ListNode):
        prev = self.__head  # dummy head
        curr = prev.next  # 0번 노드
        while curr != None:
                if curr.item == x:
                        return (prev, curr)
                else:
                        prev = curr; curr = curr.next
        return (None, None)
```



알고리즘 5-4 연결 리스트의 원소 x 삭제하기(더미 헤드를 두는 대표 버전)

```
prev.next = prev.next.next
__numItems -= 1
```

핵심부 (유사 코드)

파이썬 코드

기타 작업

```
def get(self, i:int):
        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:
        curr = self.__head.next  # 0번 노드: 더미 헤드 다음 노드
        for index in range(self.__numItems):
                if curr.item == x:
                        return index
                else:
                        curr = curr.next
        return -12345  # 안쓰는 인덱스
```

```
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  # 0번 노드
        while curr != None:
                if curr.item == x:
                        cnt += 1
                curr = curr.next
        return cnt
```

```
def extend(self, a):   # 여기서 a는 self와 같은 타입의 리스트
        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()
        for index in range(len(a)):
                self.append(a[index])
```

```
def printList(self):
        curr = self.__head.next   # 0번 노드: 더미 헤드 다음 노드
        while curr != None:
                print(curr.item, end=" ")
                curr = curr.next
        print()
```

04 배열 리스트와 연결 리스트의 비교

배열 리스트

- 직관적으로 간명하다
- 인덱스가 주어지면 즉시(상수 시간에) 접근 가능
- 연속된 공간에 저장되므로 삽입이나 삭제 시 시프트 작업 필요
- 미리 크기를 정해야 하므로 오버플로우 시 배열을 새로 배정받아 내용을 복사해야 한다
- 배열이 크면 효율이 떨어진다

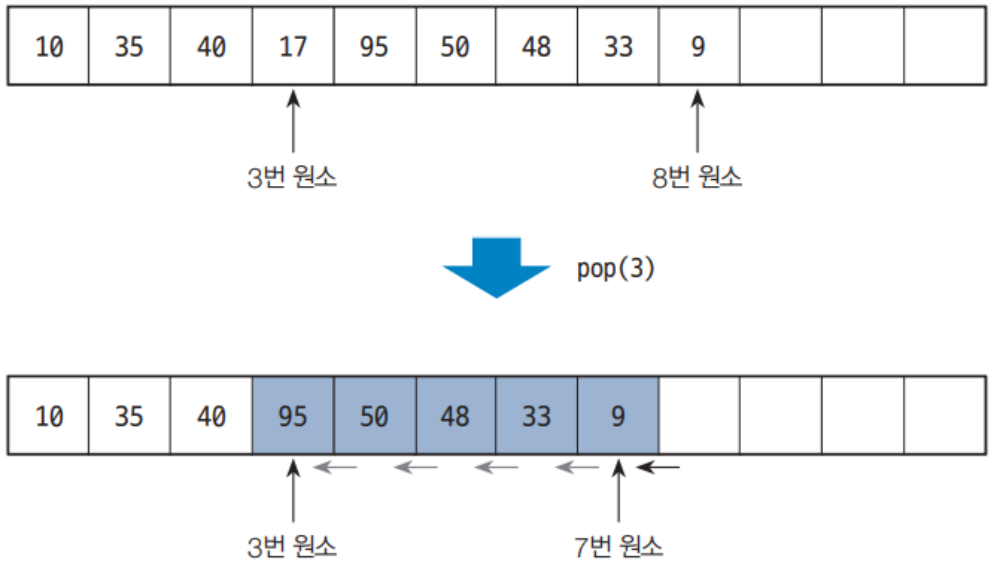


그림 5-5 배열에서 원소 삭제 후 원소들을 시프트하는 예

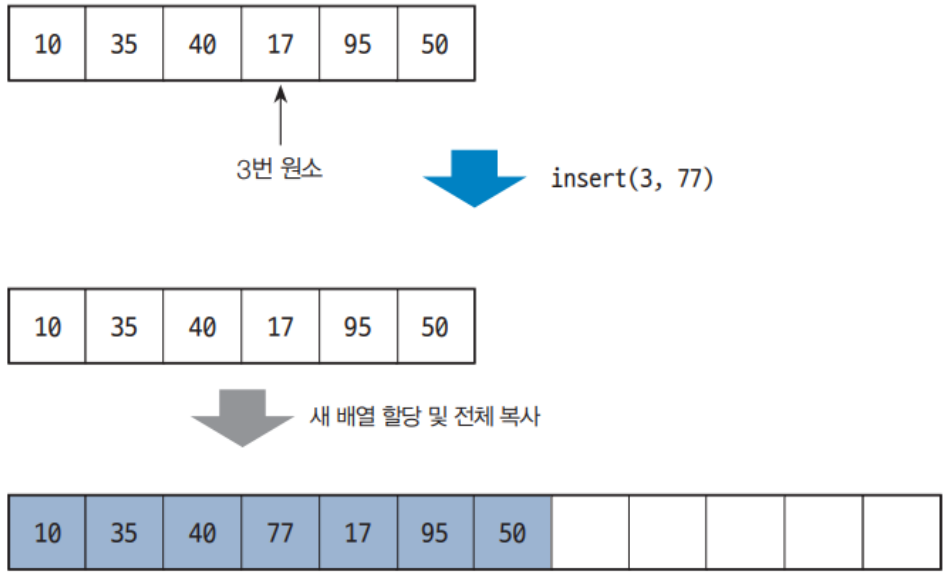
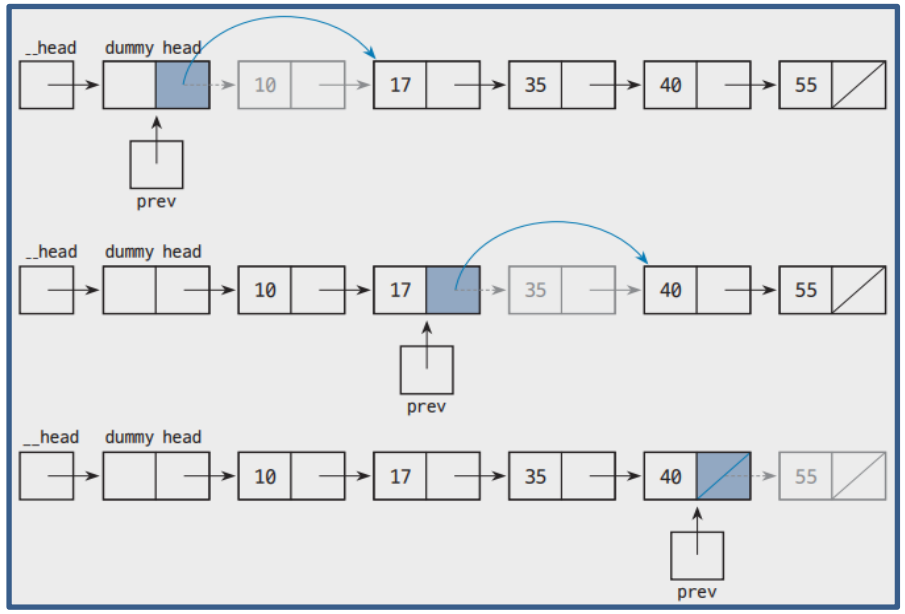


그림 5-6 배열이 꽉 찬 상태에서 삽입이 시도될 때 새 배열에 원소들을 모두 복사하는 예

연결 리스트

- 연속되지 않은 공간에 저장되어 링크를 관리하는 부담이 있다
- 인덱스가 주어진 접근도 링크를 따라가는 부담이 있다
- 삽입이나 삭제 시 시프트 작업이 필요없다
- 원소가 추가될 때 동적으로 공간을 할당 받으므로 원소의 수에 비례하는 공간만이 소요
- 오버플로우로부터 자유롭다



작업 시간 비교

작업	배열 리스트	연결 리스트
insert(i)	위치 접근 Θ(1), 삽입 작업 O(n)	위치 접근 O(n), 삽입 작업 Θ(1)
pop(i)	위치 접근 Θ(1), 삭제 작업 O(n)	위치 접근 O(n), 삭제 작업 Θ(1)
remove(x)	원소 찾기 O(n), 삭제 작업 O(n)	원소 찾기O(n), 삭제 작업 Θ(1)
get(i)	Θ(1)	O(n)

05 연결 리스트의 개선 및 확장

개선 1: 원형 연결 리스트Circular Linked List

끝 노드(tail)의 next가 null 값을 갖는 대신 첫 노드를 링크한다

맨 앞과 맨 뒤의 접근성 차이가 없어짐

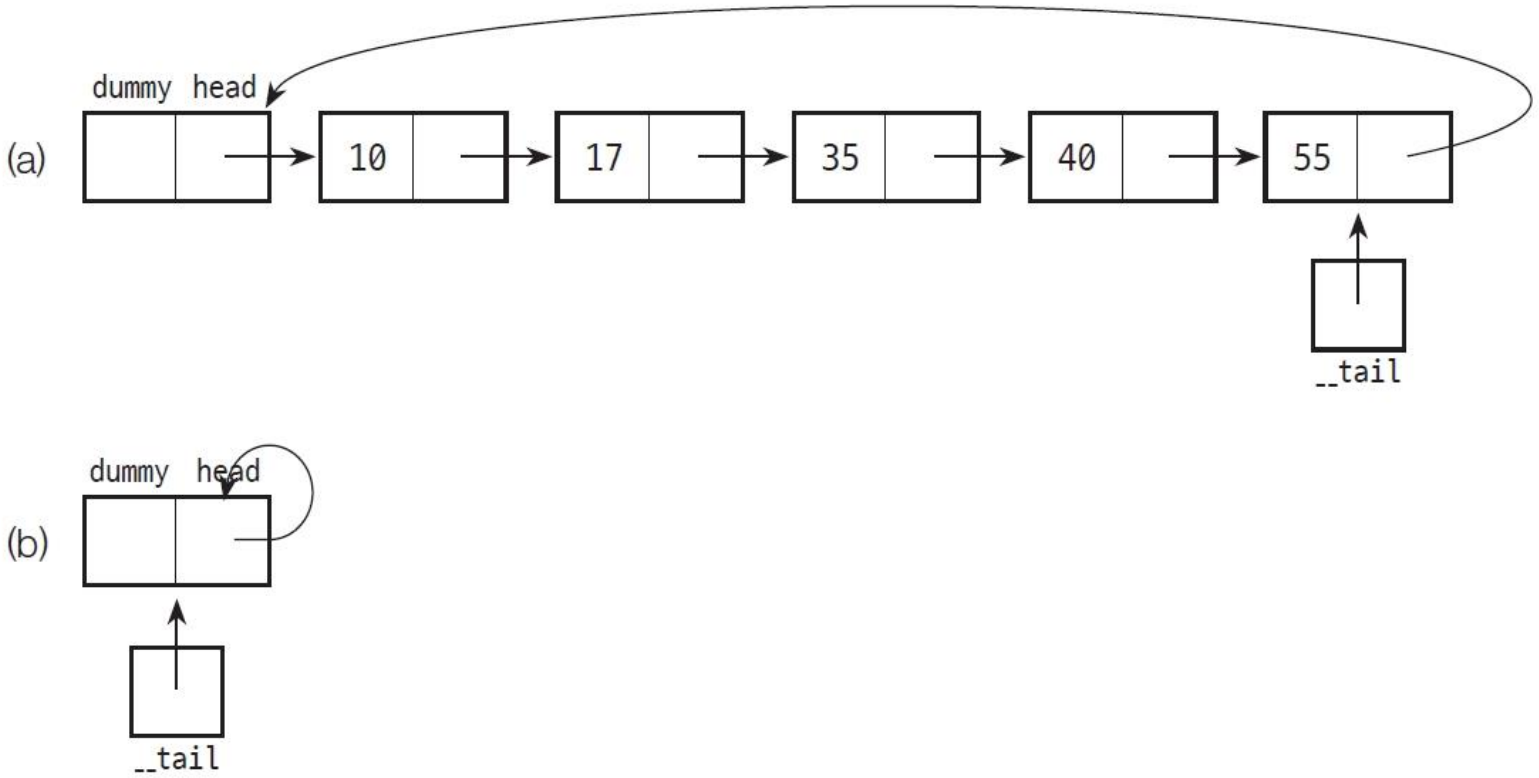


그림 5-21 더미 헤드 노드를 가진 원형 연결 리스트의 예

강의 노트에서 혼동의 여지가 없으므로
__tail과 tail을 섞어서 씀

더미 헤드를 가진 원형 연결 리스트에서의 삽입

```
def insert(self, i:int, newItem) -> None:
    if (i >= 0 and i <= self.__numItems):
        prev = self.getNode(i - 1)
        newNode = ListNode(newItem, prev.next)
        prev.next = newNode
        if i == self.__numItems:
            self.__tail = newNode
        self.__numItems += 1
    else:
        print("index", i, ": out of bound in insert()") # 필요시 에러 처리
```

코드는 삽입 위치에 무관하게 동일

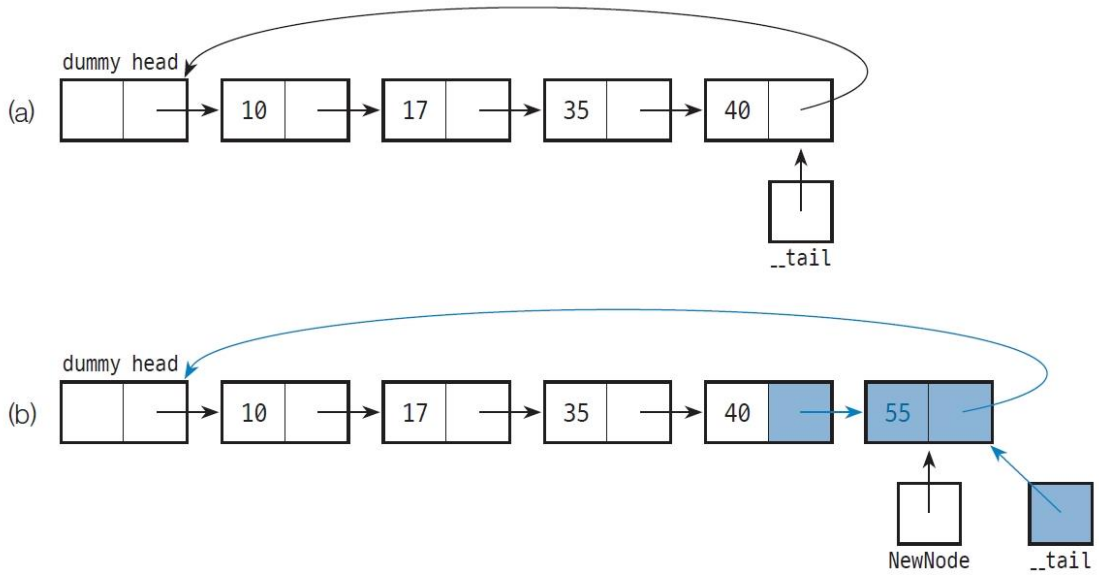


그림 5-22 더미 헤드 노드를 가진 원형 연결 리스트에서 끝에 원소 삽입하기

개선 2: 가변 파라미터

Reminder: 앞에서 pop(i)은 항상 삭제 위치를 명시했다

개선: 삭제 위치를 명시하지 않아도 된다

pop(i): i번 원소 삭제
pop(): 맨 끝 원소 삭제
pop(-1): 맨 끝 원소 삭제

```
def pop(self, *args):
        # 가변 파라미터. 인자가 없거나 -1이면 마지막 원소로 처리하기 위함.
        # 파이썬 리스트 규칙 만족
        if self.isEmpty():
                return None
        # 인덱스 i 결정
        if len(args) != 0: # pop(k)과 같이 인자가 있으면 i=k 할당
                i = args[0]
        if len(args) == 0 or i == -1:  # pop()에 인자가 없거나 pop(-1)이면 i에 맨 끝 인덱스 할당
                i = self.__numItems - 1
        # i번 원소 삭제. 이후는 앞절의 pop(i)와 같음.
        if (i >= 0 and i <= self.__numItems - 1):
                prev = self.getNode(i - 1)
                retItem = prev.next.item
                prev.next = prev.next.next
                if i == self.__numItems - 1:
                        self.__tail = prev
                self.__numItems -= 1
                return retItem
        else:
                return None
```

개선 3: 순회자

대표적 디자인 패턴

객체의 원소들을 손쉽게 순회할 수 있게 한다

원소들을 순회하는 작업의 예
count(), extend(), copy()

순회자 객체 생성

```
class CircularLinkedList:
    …
    def __iter__(self):  # generating iterator and return
        return CircularLinkedListIterator(self)
```

다음 원소 가져오기

순회자 클래스

```
class CircularLinkedListIterator:
    def __init__(self, alist):
        self.__head = alist.getNode(-1)  # dummy head
        self.iterPosition = self.__head.next  # 0번 노드
    def __next__(self):
        if self.iterPosition == self.__head: # 순환 끝
            raise StopIteration
        else: # 현재 원소 리턴하면서 다음 원소로 이동
            item = self.iterPosition.item
            self.iterPosition = self.iterPosition.next
            return item
```

파이썬 순회자에 필요한 것

1. 순회자 클래스
2. 순회자 객체 생성
   대상 클래스가 __iter__() 메서드를 갖고 있어야 한다
3. 다음 원소 리턴
   순회자 클래스가 __next()__ 메서드를 갖고 있어야 한다

사용 예

```
def printList(self):
    for element in self:
        print(element, end=' ')
    print()
```

전체 코드: 원형 연결 리스트

```
from DS.list.listNode import ListNode

class CircularLinkedList:
    def __init__(self):
        self.__tail = ListNode("dummy", None)
        self.__tail.next = self.__tail
        self.__numItems = 0

    def insert(self, i:int, newItem) -> None:
        if (i >= 0 and i <= self.__numItems):
            prev = self.getNode(i - 1)
            newNode = ListNode(newItem, prev.next)
            prev.next = newNode
            if i == self.__numItems:
                self.__tail = newNode
            self.__numItems += 1
        else:
            print("index", i, ": out of bound in insert()") # 필요시 에러 처리

    def append(self, newItem) -> None:
        newNode = ListNode(newItem, self.__tail.next)
        self.__tail.next = newNode
        self.__tail = newNode
        self.__numItems += 1
```

1/5

```
    def pop(self, *args):
        # 가변 파라미터. 인자가 없거나 -1이면 마지막 원소로 처리하기 위함. 파이썬 리스트 규칙 만족
        if self.isEmpty():
            return None
        # 인덱스 i 결정
        if len(args) != 0: # pop(k)과 같이 인자가 있으면 i=k 할당
            i = args[0]
        if len(args) == 0 or i == -1:  # pop()에 인자가 없거나 pop(-1)이면 i에 맨 끝 인덱스 할당
            i = self.__numItems - 1
        # i번 원소 삭제
        if (i >= 0 and i <= self.__numItems - 1):
            prev = self.getNode(i - 1)
            retItem = prev.next.item
            prev.next = prev.next.next
            if i == self.__numItems - 1:
                self.__tail = prev
            self.__numItems -= 1
            return retItem
        else:
            return None

    def remove(self, x):
        (prev, curr) = self.__findNode(x)
        if curr != None:
            prev.next = curr.next
            if curr == self.__tail:
                self.__tail = prev
            self.__numItems -= 1
            return x
        else:
            return None
```

2/5

```
    def get(self, *args):
        # 가변 파라미터. 인자가 없거나 -1이면 마지막 원소로 처리하기 위함. 파이썬 리스트 규칙 만족
        if self.isEmpty():
            return None
        # 인덱스 i 결정
        if len(args) != 0: # pop(k)과 같이 인자가 있으면 i=k 할당
            i = args[0]
        if len(args) == 0 or i == -1:  # pop()에 인자가 없거나 pop(-1)이면 i에 맨 끝 인덱스 할당
            i = self.__numItems - 1
        # i번 원소 리턴
        if (i >= 0 and i <= self.__numItems - 1):
            return self.getNode(i).item
        else:
            return None

    def index(self, x) -> int:
        cnt = 0
        for element in self:
            if element == x:
                return cnt
            cnt += 1
        return -12345

    def isEmpty(self) -> bool:
        return self.__numItems == 0

    def size(self) -> int:
        return self.__numItems

    def clear(self):
        self.__tail = ListNode("dummy", None)
        self.__tail.next = self.__tail
        self.__numItems = 0
```

3/5

```
    def count(self, x) -> int:
        cnt = 0
        for element in self:
            if element == x:
                cnt += 1
        return cnt

    def extend(self, a): # a는 순환가능한 모든 객체
        for x in a:
            self.append(x)

    def copy(self) -> 'CircularLinkedList':
        a = CircularLinkedList()
        for element in self:
            a.append(element)
        return a

    def reverse(self) -> None:
        head = self.__tail.next  # dummy head
        prev = head; curr = prev.next; next = curr.next
        curr.next = head; head.next = self.__tail; self.__tail = curr
        for i in range(self.__numItems - 1):
            prev = curr; curr = next; next = next.next
            curr.next = prev

    def sort(self) -> None:
        a = []
        for element in self:
            a.append(element)
        a.sort()
        self.clear()
        for element in a:
            self.append(element)
```

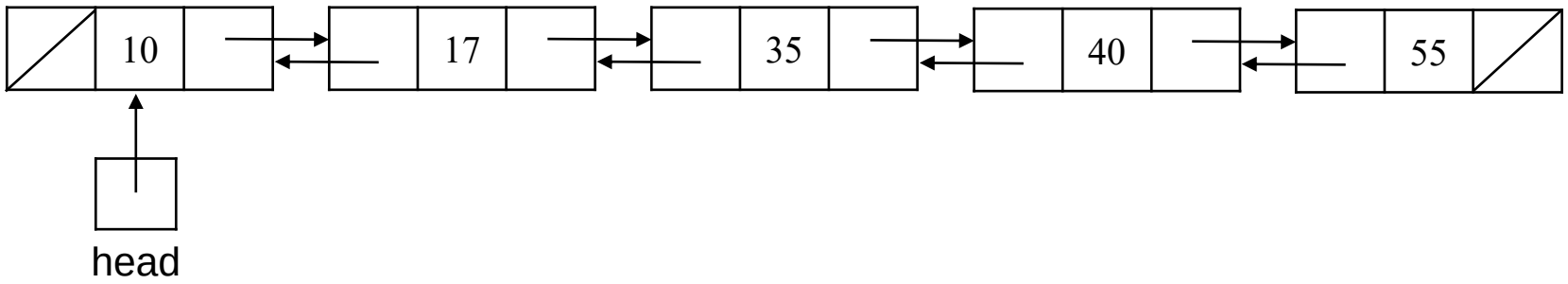
4/5

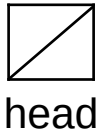
```
    def __findNode(self, x) -> (ListNode, ListNode):
        head = prev = self.__tail.next  # dummy head
        curr = prev.next  # 0번 노드
        while curr != head:
            if curr.item == x:
                return (prev, curr)
            else:
                prev = curr; curr = curr.next
        return (None, None)

    def getNode(self, i:int) -> ListNode:
        curr = self.__tail.next  # dummy head, index: -1
        for index in range(i+1):
            curr = curr.next
        return curr

    def printList(self) -> None:
        for element in self:
            print(element, end=' ')
        print()

    def __iter__(self):  # generating iterator and return
        return CircularLinkedListIterator(self)

class CircularLinkedListIterator:
    def __init__(self, alist):
        self.__head = alist.getNode(-1)  # dummy head
        self.iterPosition = self.__head.next  # 0번 노드
    def __next__(self):
        if self.iterPosition == self.__head: # 순환 끝
            raise StopIteration
        else: # 현재 원소 리턴하면서 다음 원소로 이동
            item = self.iterPosition.item
            self.iterPosition = self.iterPosition.next
            return item
```

5/5

양방향 연결 리스트Doubly Linked List

양방향 연결 리스트의 예





초기 상태: Empty list

노드 구조


```
class BidirectNode:
        def __init__(self, newItem, prevNode:'BidirectNode', nextNode:'BidirectNode'):
                self.item = newItem
                self.prev = prevNode
                self.next = nextNode
```

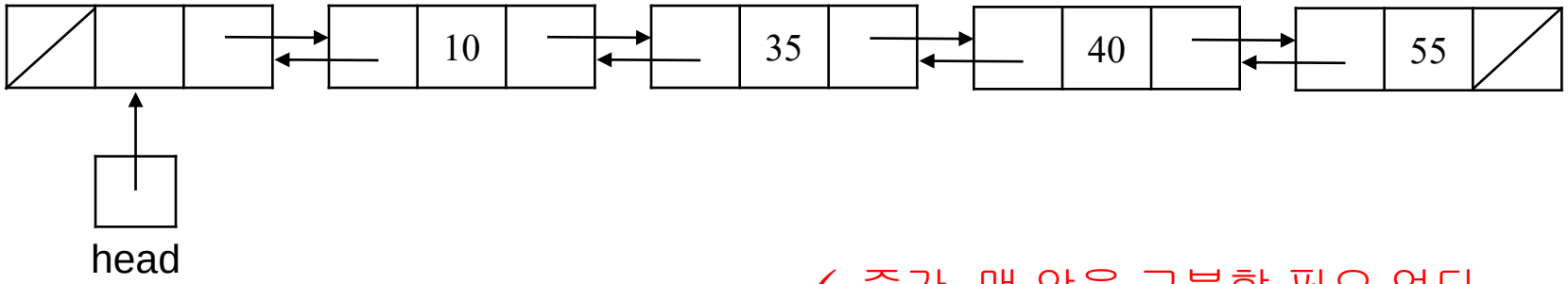
더미 헤드를 가진 양방향 연결 리스트

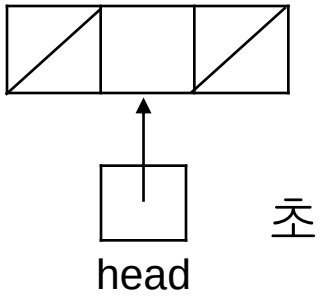
삽입

```
newNode ← BidirectNode(newItem, prev, prev.next)
newNode.next.prev ← newNode
prev.next ← newNode
numItems++
```

삭제

```
curr.prev.next ← curr.next
curr.next.prev ← curr.prev
numItems--
```





초기 상태: Empty list

✓ 중간, 맨 앞을 구분할 필요 없다
✓ 맨 앞 작업도 중간 작업과 동일

더미 헤드를 가진 원형 양방향 연결 리스트

연결 리스트의 모든 연결 방식이 포함된 버전

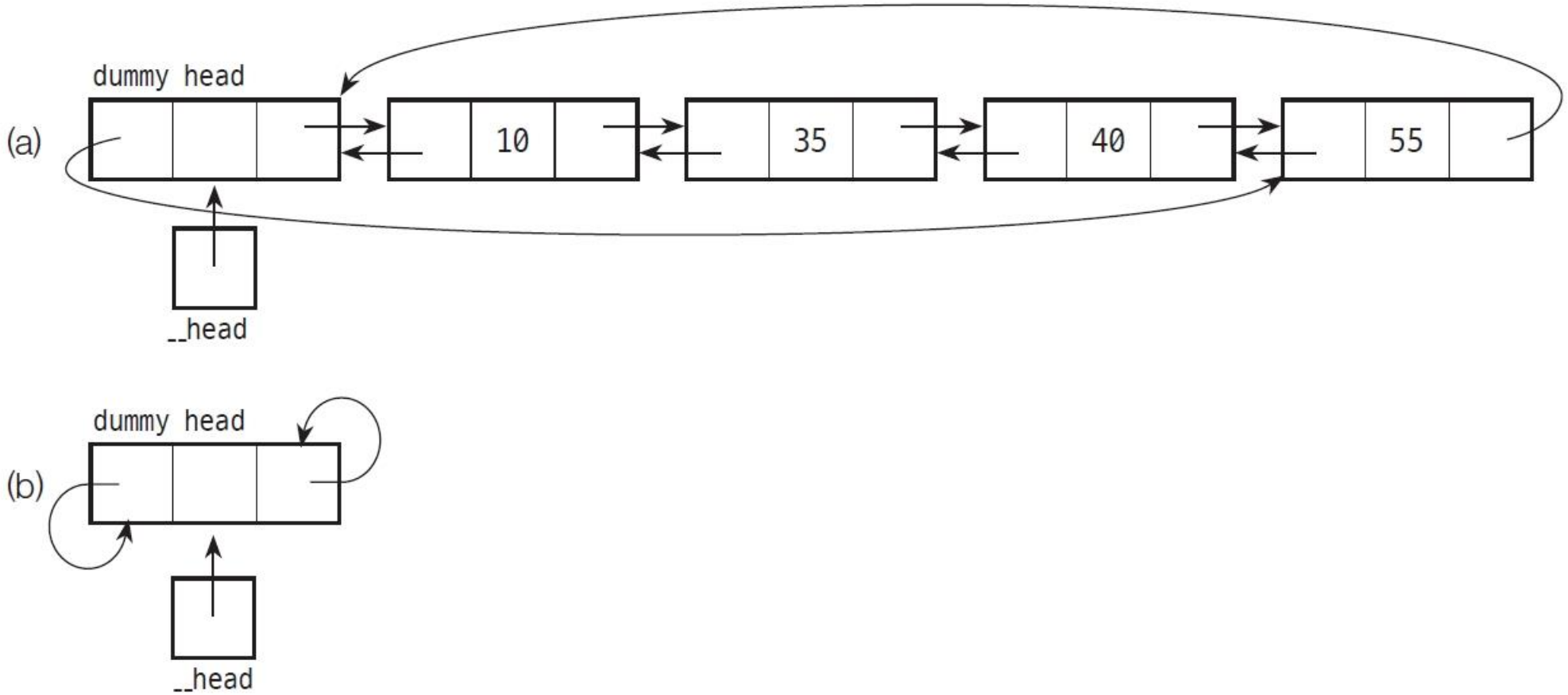


그림 5-24 더미 헤드를 가진 원형 양방향 연결 리스트의 예와 빈 리스트의 모양

삽입

```
newNode ← BidirectNode(newItem, prev, prev.next)
newNode.next.prev ← newNode
prev.next ← newNode
numItems++
```

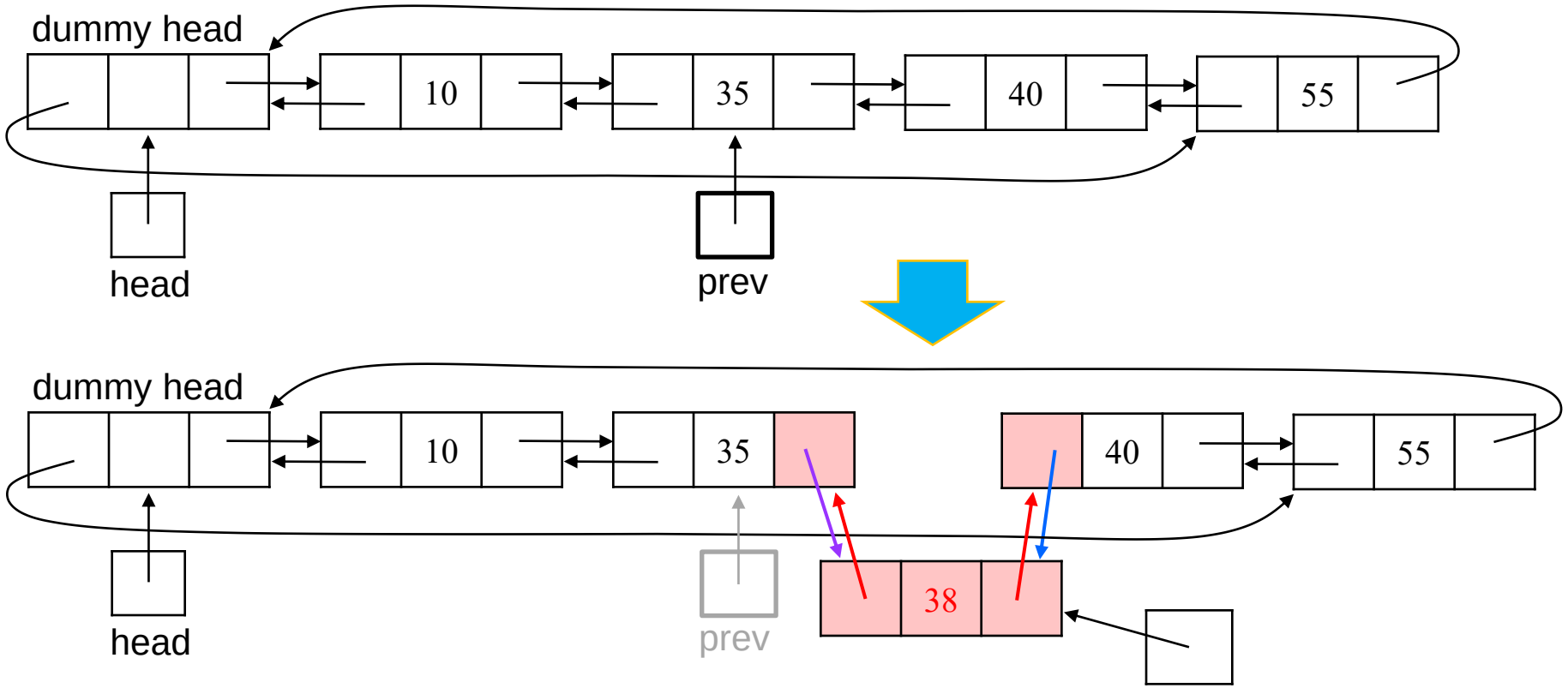
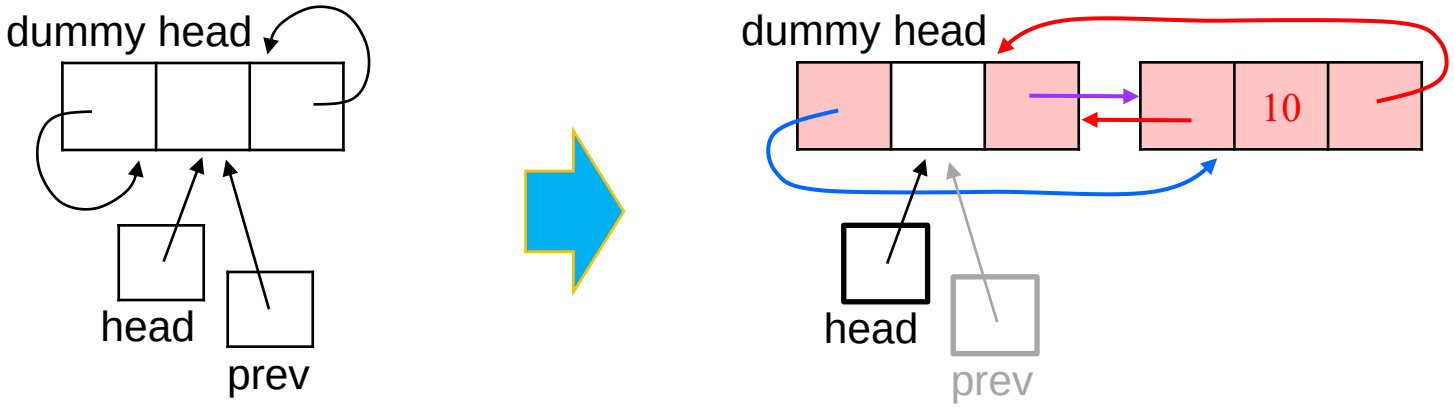


그림 5-25 양방향 연결 리스트에 원소를 삽입하는 예

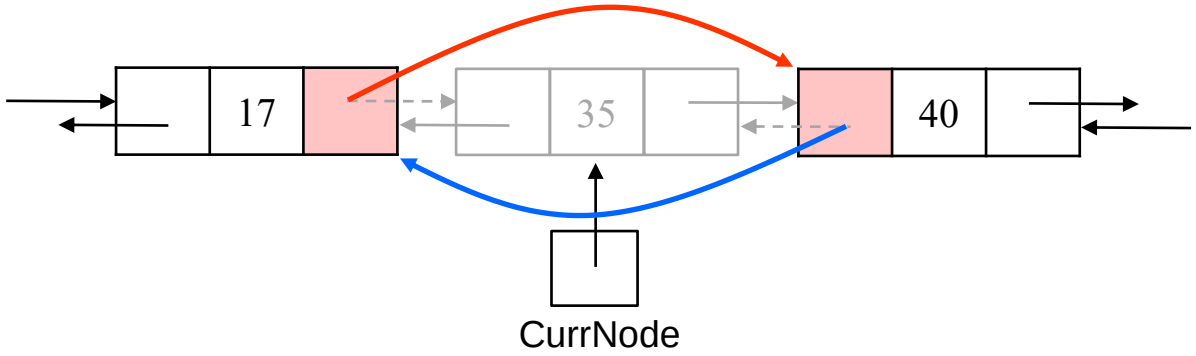
확인: 맨 앞 삽입도 이걸로 Okay

```
newNode ← BidirectNode(newItem, prev, prev.next)
newNode.next.prev ← newNode
prev.next ← newNode
numItems++
```

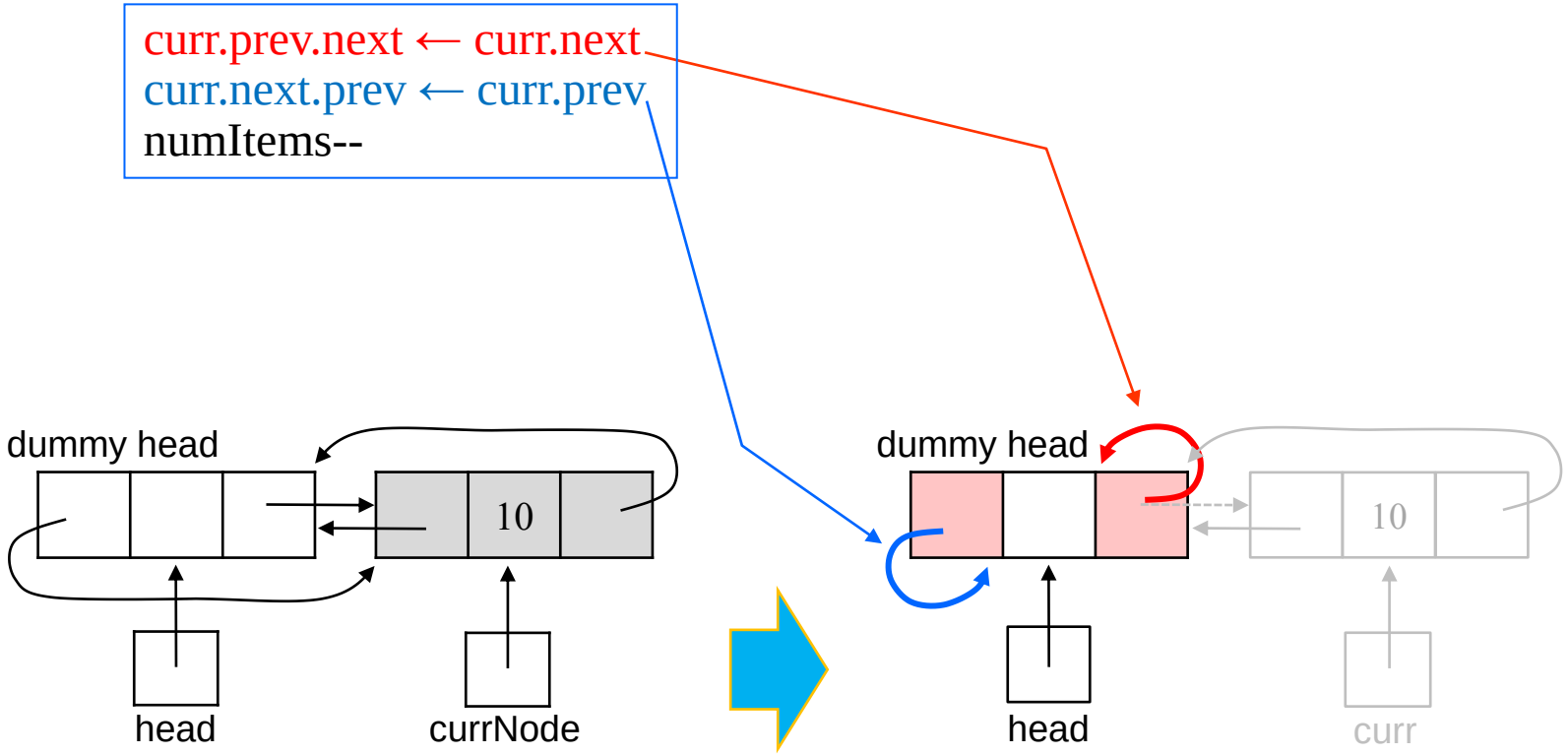


삭제

curr.prev.next ← curr.next
curr.next.prev ← curr.prev
numItems--



확인: 맨 앞 삭제도 이걸로 Okay



전체 코드: 원형 양방향 연결 리스트

```
from bidirectNode import BidirectNode

class CircularDoublyLinkedList:
    def __init__(self):
        self.__head = BidirectNode("dummy", None)
        self.__head.prev = self.__head
        self.__head.next = self.__head
        self.__numItems = 0

    def insert(self, i:int, newItem) -> None:
        if (i >= 0 and i <= self.__numItems):
            prev = self.getNode(i - 1)
            newNode = BidirectNode(newItem, prev, prev.next)
            newNode.next.prev = newNode
            prev.next = newNode
            self.__numItems += 1
        else:
            print("index", i, ": out of bound in insert()") # 필요시 에러 처리

    def append(self, newItem) -> None:
        prev = self.__head.prev
        newNode = BidirectNode(newItem, prev, self.__head)
        prev.next = newNode
        self.__head.prev = newNode
        self.__numItems += 1
    …
```

1/5

```
    def pop(self, *args):
        # 가변 파라미터. 인자가 없거나 -1이면 마지막 원소로 처리하기 위함. 파이썬 리스트 규칙 만족
        if self.isEmpty():
            return None
        # 인덱스 i 결정
        if len(args) != 0: # pop(k)과 같이 인자가 있으면 i=k 할당
            i = args[0]
        if len(args) == 0 or i == -1:  # pop()에 인자가 없거나 pop(-1)이면 i에 맨 끝 인덱스 할당
            i = self.__numItems - 1
        # i번 원소 삭제
        if (i >= 0 and i <= self.__numItems - 1):
            prev = self.getNode(i)
            retItem = curr.item
            curr.prev.next = curr.next
            curr.next.prev = curr.prev
            self.__numItems -= 1
            return retItem
        else:
            return None

    def remove(self, x):
        curr = self.__findNode(x)
        if curr != None:
            curr.prev.next = curr.next
            curr.next.prev = curr.prev
            self.__numItems -= 1
            return x
        else:
            return None
```

```
    def get(self, *args):
        … class CircularLinkedList와 동일 …

    def index(self, x) -> int:
        … class CircularLinkedList와 동일 …

    def isEmpty(self) -> bool:
        … class CircularLinkedList와 동일 …

    def size(self) -> int:
        … class CircularLinkedList와 동일 …

    def clear(self):
        self.__head = BidirectNode("dummy", None, None)
        self.__head.prev = self.__head
        self.__head.next = self.__head
        self.__numItems = 0
```



```
    def count(self, x) -> int:
        … class CircularLinkedList와 동일 …

    def extend(self, a): # a는 순환가능한 모든 객체
        … class CircularLinkedList와 동일 …

    def copy(self) -> 'CircularDoublyLinkedList':
        a = CircularDoublyLinkedList()
        … 이하 class CircularLinkedList와 동일 …

    def reverse(self) -> None:
        prev = self.__head; curr = prev.next; next = curr.next
        self.__head.next = prev.prev; self.__head.prev = curr
        for i in range(self.__numItems):
            curr.next = prev; curr.prev = next
            prev = curr; curr = next; next = next.next

    def sort(self) -> None:
        … class CircularLinkedList와 동일 …
```

```
    def __findNode(self, x) ->BidirectNode:
        curr = self.__head.next  # 0번 노드
        while curr != self.__head:
            if curr.item == x:
                return curr
            else:
                curr = curr.next
        return None

    def getNode(self, i:int) -> BiDirectNode:
        curr = self.__head  # dummy head, index: -1
        for index in range(i+1):
            curr = curr.next
        return curr

    def printList(self) -> None:
        for element in self:
            print(element, end=' ')
        print()

    def __iter__(self):  # generating iterator and return
        return CircularDoublyLinkedListIterator(self)

class CircularDoublyLinkedListIterator:
    def __init__(self, alist):
        self.__head = alist.getNode(-1)  # dummy head
        self.iterPosition = self.__head.next  # 0번 노드
    def __next__(self):
        if self.iterPosition == self.__head: # 순환 끝
            raise StopIteration
        else: # 현재 원소 리턴하면서 다음 원소로 이동
            item = self.iterPosition.item
            self.iterPosition = self.iterPosition.next
            return item
```