

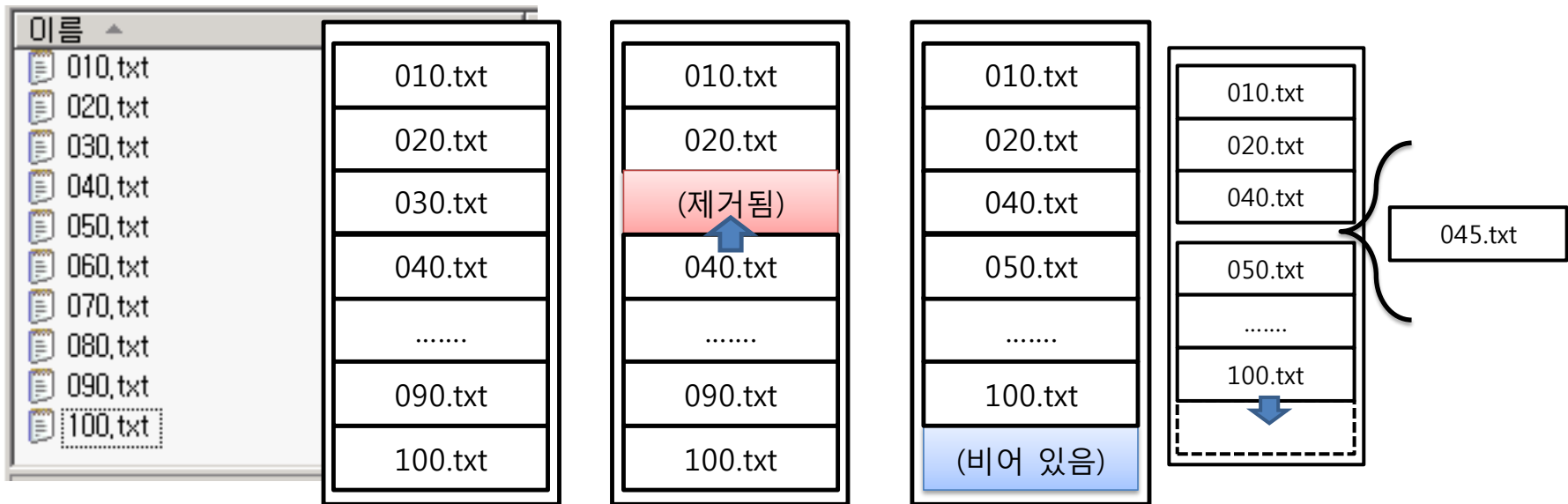
리스트

# 목차

1. 리스트의 개념
2. 리스트 추상 자료형
3. 배열 리스트
4. 연결 리스트의 개념
5. 단순 연결 리스트
6. 원형 연결 리스트
7. 이중 연결 리스트
8. 연결 리스트의 응용

# 1. 리스트의 개념 (1/2)

- 리스트(List)
  - 자료를 순서대로 저장하는 자료구조
    - 일직선으로 연결된(Sequential) 자료



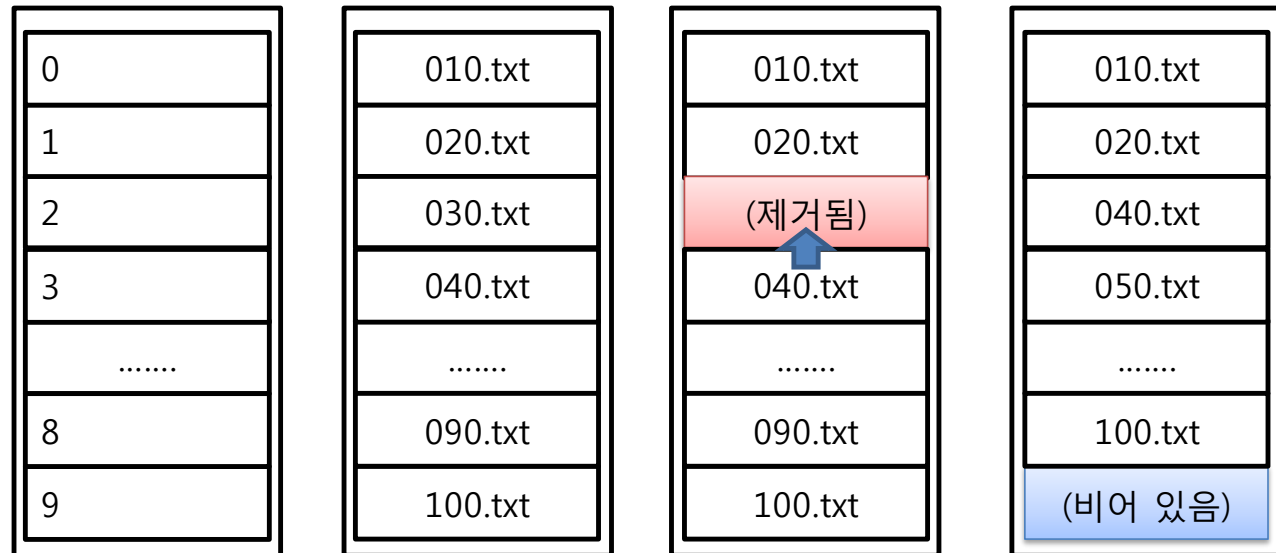
# 1. 리스트의 개념 (2/2)

- 구현 방법
  - 배열 리스트(Array List)
  - 연결 리스트(Linked List)
    - (연결) 포인터를 이용한 구현



## 2. 리스트 추상 자료형

- 리스트 생성
  - 최대 원소 개수  $n$  (배열의 크기)
- 원소 추가
  - 원소 추가 가능 여부 판단
- 원소 반환
- 원소 제거
  - 리스트 초기화
- 리스트 삭제



### 3. 배열 리스트 (1/4)

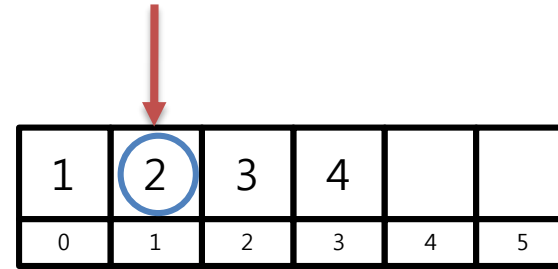
- 배열 리스트
  - 논리적 (저장) 순서와 물리적 (저장) 순서가 동일
    - Cf) 최대 원소 개수

1	2	3	4		
0	1	2	3	4	5

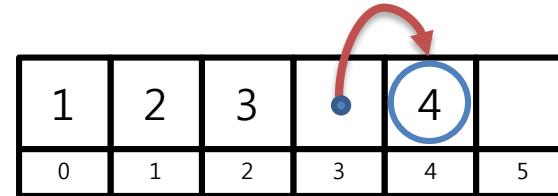
- 원소의 위치 인덱스는 0부터 시작(0-based index)
  - C 배열에서와 동일

### 3. 배열 리스트 (2/4)

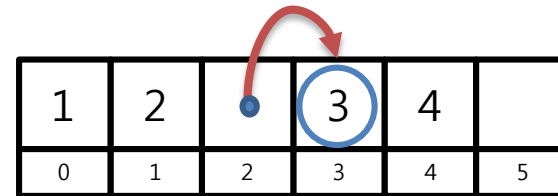
- 배열 리스트의 원소 추가
  - 추가 전
    - 시작
      - (가장 오른쪽 자료)
    - 방향
      - (왼쪽 방향)
    - 어디까지
      - (추가하려는 위치)



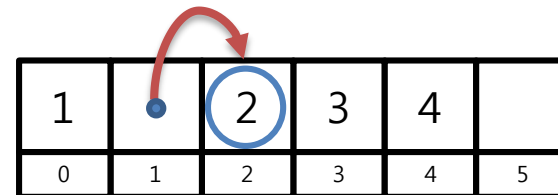
1	2	3	4		
0	1	2	3	4	5



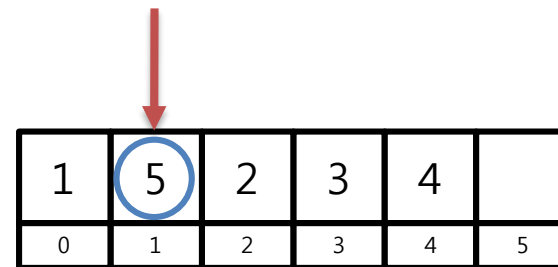
1	2	3		4	
0	1	2	3	4	5



1	2		3	4	
0	1	2	3	4	5



1		2	3	4	
0	1	2	3	4	5



1	5	2	3	4	
0	1	2	3	4	5

### 3. 배열 리스트 (3/4)

- 배열 리스트에서 원소 제거
  - 제거 후
    - 시작
      - (제거하려는 위치 오른쪽)
    - 방향
      - (오른쪽)
    - 어디까지
      - (가장 오른쪽 자료)

1	5	2	3	4	
0	1	2	3	4	5

5		2	3	4	
0	1	2	3	4	5

5	2		3	4	
0	1	2	3	4	5

5	2	3		4	
0	1	2	3	4	5

5	2	3	4		
0	1	2	3	4	5



### 3. 배열 리스트 (4/4)

- Q) 원소의 개수가 10만개인 배열 리스트에서 원소의 추가/제거가 빈번하게 발생한다면?
  - 배열 크기: 10만개
    - 위치 0의 자료 제거
    - 위치 0의 자료 추가

### 3. 배열 리스트: 구현

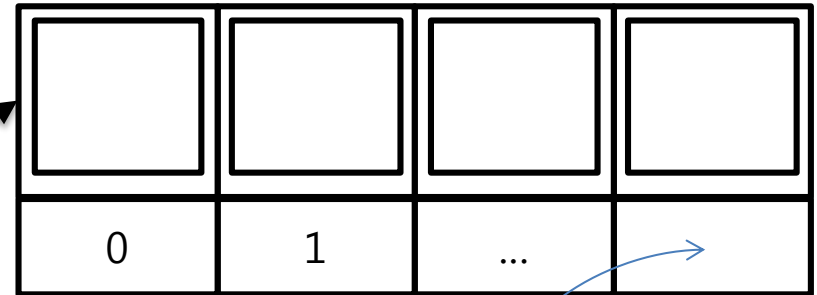
ArrayList 구조체(struct)

최대 원소 개수  
(maxElementCount)

현재 원소 개수  
(currentElementCount)

원소 저장을 위한 배열 포인터  
(pElement)

ArrayListNode 구조체



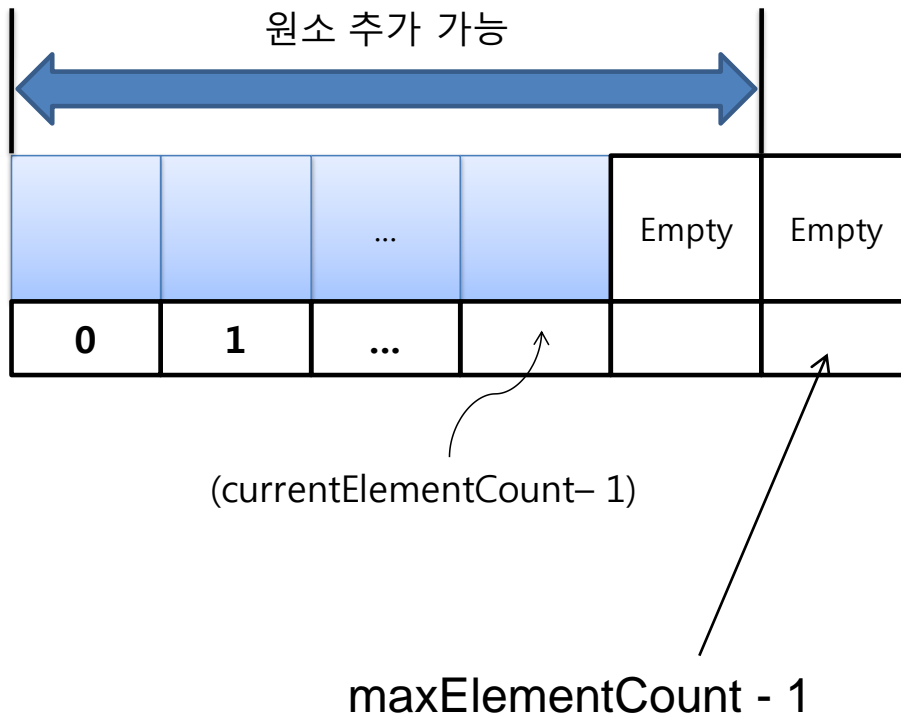
(maxElementCount - 1)

### 3. 배열 리스트: 구현

- 배열 리스트의 생성
- 원소 추가
- 원소 제거
- 기타
- 예제 프로그램

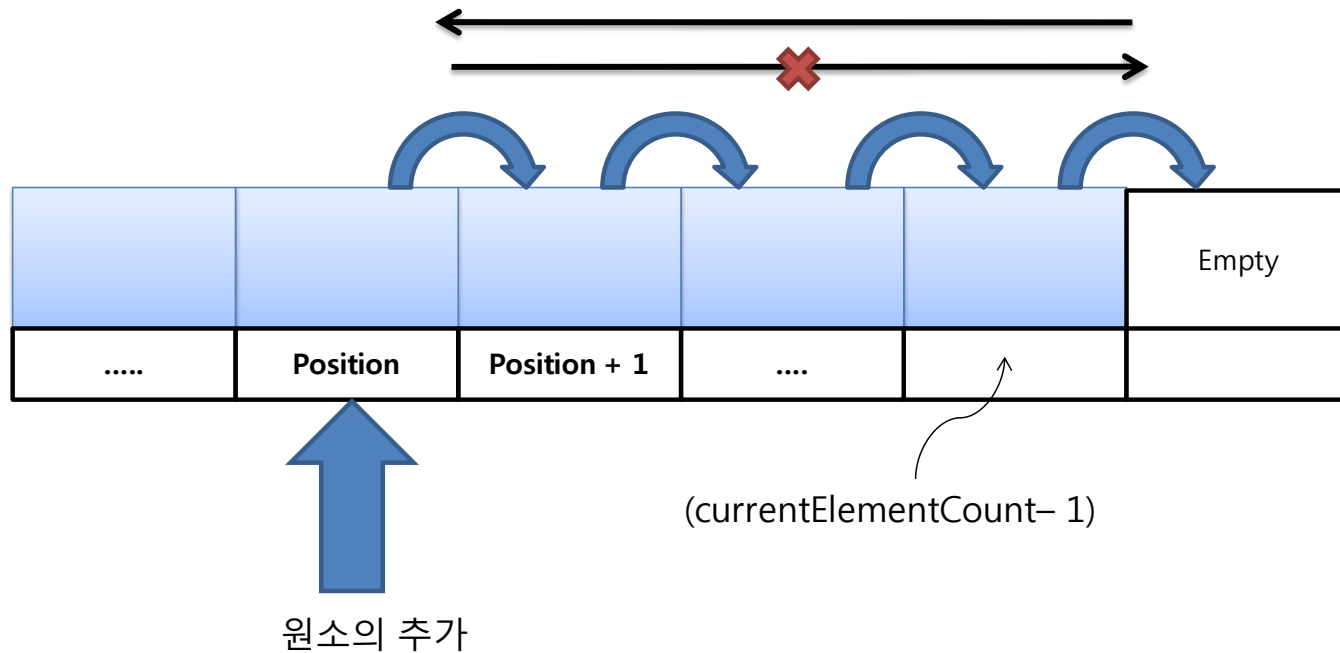
### 3. 배열 리스트: 구현

- 원소 추가
  - `addALElement()`



### 3. 배열 리스트: 구현

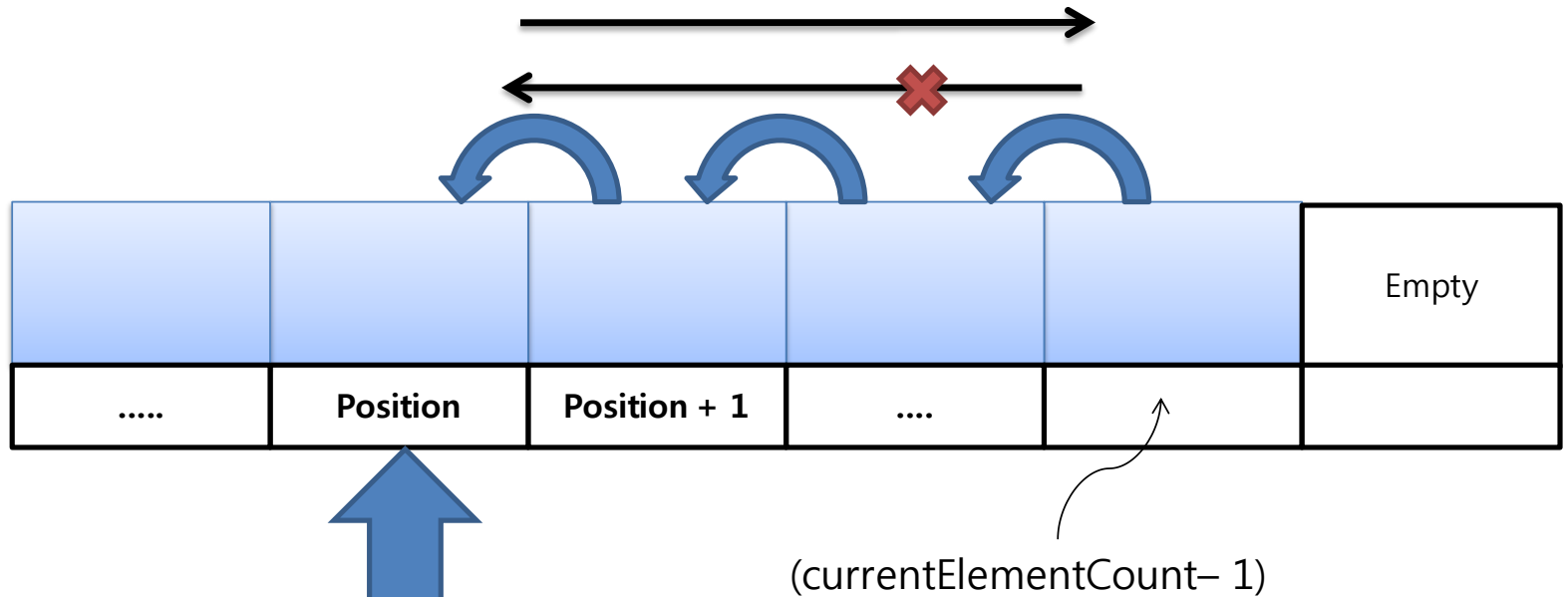
- 원소 추가 (계속)
  - addALElement()



- 시작
- 방향
- 어디까지

### 3. 배열 리스트: 구현

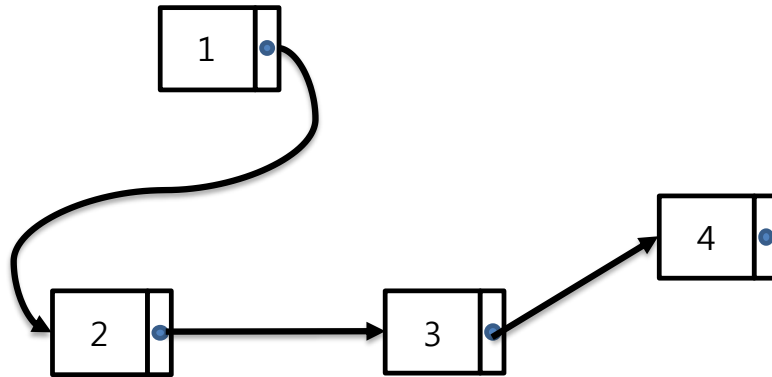
- 원소의 제거



- 시작
- 방향
- 어디까지

## 4. 연결 리스트의 개념 (1/6)

- 연결 리스트(Linked List)
  - 포인터



논리적 위치

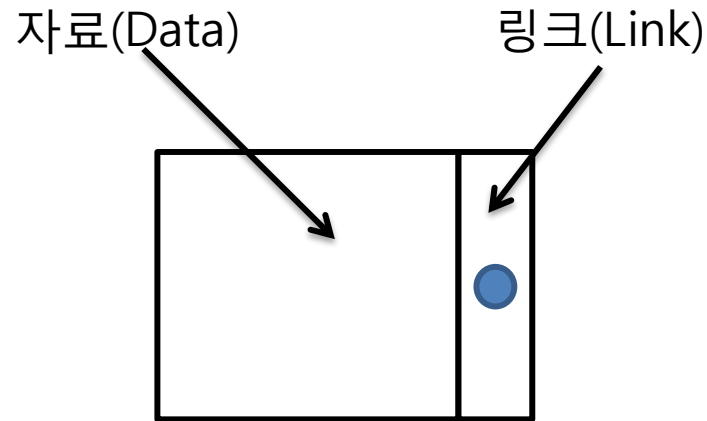
1	2	3	4		
0	1	2	3	4	5

물리적 위치

- 차이점
  - 최대 원소 개수 지정

## 4. 연결 리스트의 개념 (2/6)

- 연결 리스트의 구조
  - 노드(Node) = ?



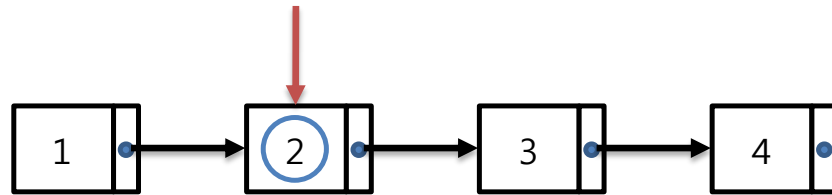
- 연결 리스트의 노드 추가/제거
  - 배열 리스트는?
    - 원소 이동 필요



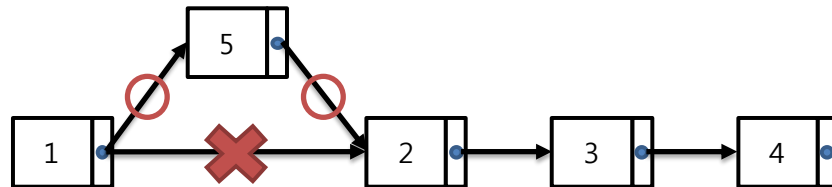
## 4. 연결 리스트의 개념 (3/6)

- 연결 리스트의 노드 추가
  - 예) '위치 인덱스' 1에 새로운 원소 5를 추가  
(2번째 자료)

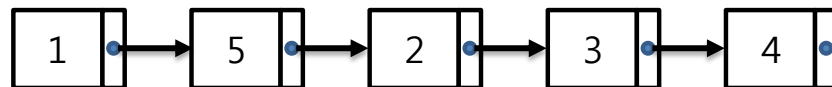
1. 새로운 노드 추가 전



2-1. 기존 링크 제거  
2-2. 새로운 링크 추가



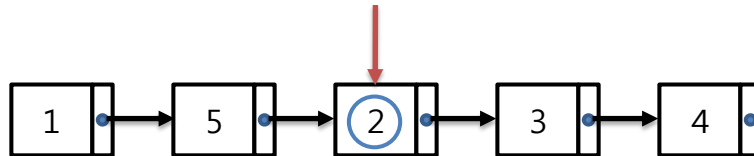
3. 새로운 노드 추가 후



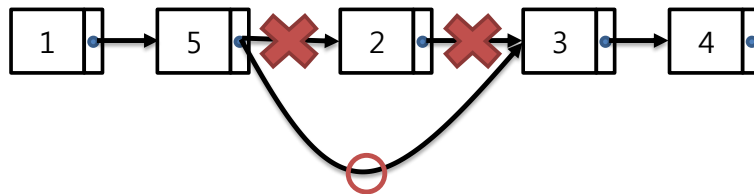
## 4. 연결 리스트의 개념 (4/6)

- 연결 리스트의 노드 제거
  - 예) '위치 인덱스' 2의 원소 제거  
(3번째 자료)

### 1. 노드 제거 전



- 2-1. 기존 링크 제거
- 2-2. 새로운 링크 추가



### 3. 노드 제거 후

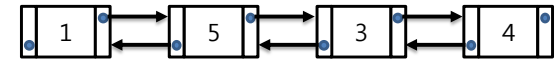
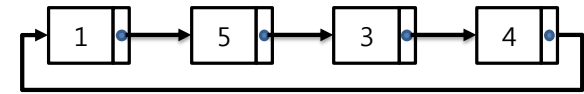
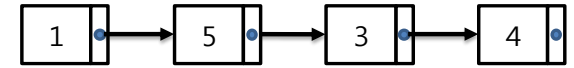


## 4. 연결 리스트의 개념 (5/6)

- 연결 리스트 VS 배열 리스트
  - 연결 리스트의 장점
    - 추가 원소 이동 연산 불필요
    - 최대 원소 개수 지정 불필요
  - 연결 리스트의 단점
    - 구현의 어려움
    - 탐색 연산의 비용 높음
      - 원소 개수가  $n$ 개:  $O(n)$

## 4. 연결 리스트의 개념 (6/6)

- 연결 리스트의 종류
  - 단순 연결 리스트(Singly Linked List)
  - 원형 연결 리스트(Circular Linked List)
  - 이중 연결 리스트(Double Linked List)



- 연결 리스트의 특성 비교
  - 이전(previous) 노드에 대한 접근 연산
    - 단순 연결 리스트: 첫번째 노드부터 새로 순회
    - 원형 연결 리스트: 전체 리스트를 한번 순회 (계속 순회하다 보면 이전 노드 접근 가능)
    - 이중 연결 리스트: 직접 접근 가능

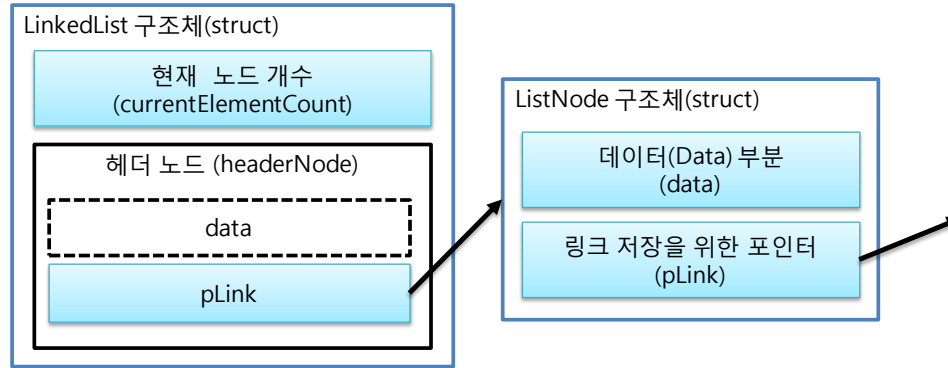
## 5. 단순 연결 리스트 (1/6)

- 소스의 구성

파일 이름	내용
03_02.vcproj	Visual Studio 프로젝트 파일
linkedlist.h	구조체와 함수 선언
linkedlist.c	함수 구현
exampl03_02.c	예제 프로그램

## 5. 단순 연결 리스트 (2/6)

- 구조체



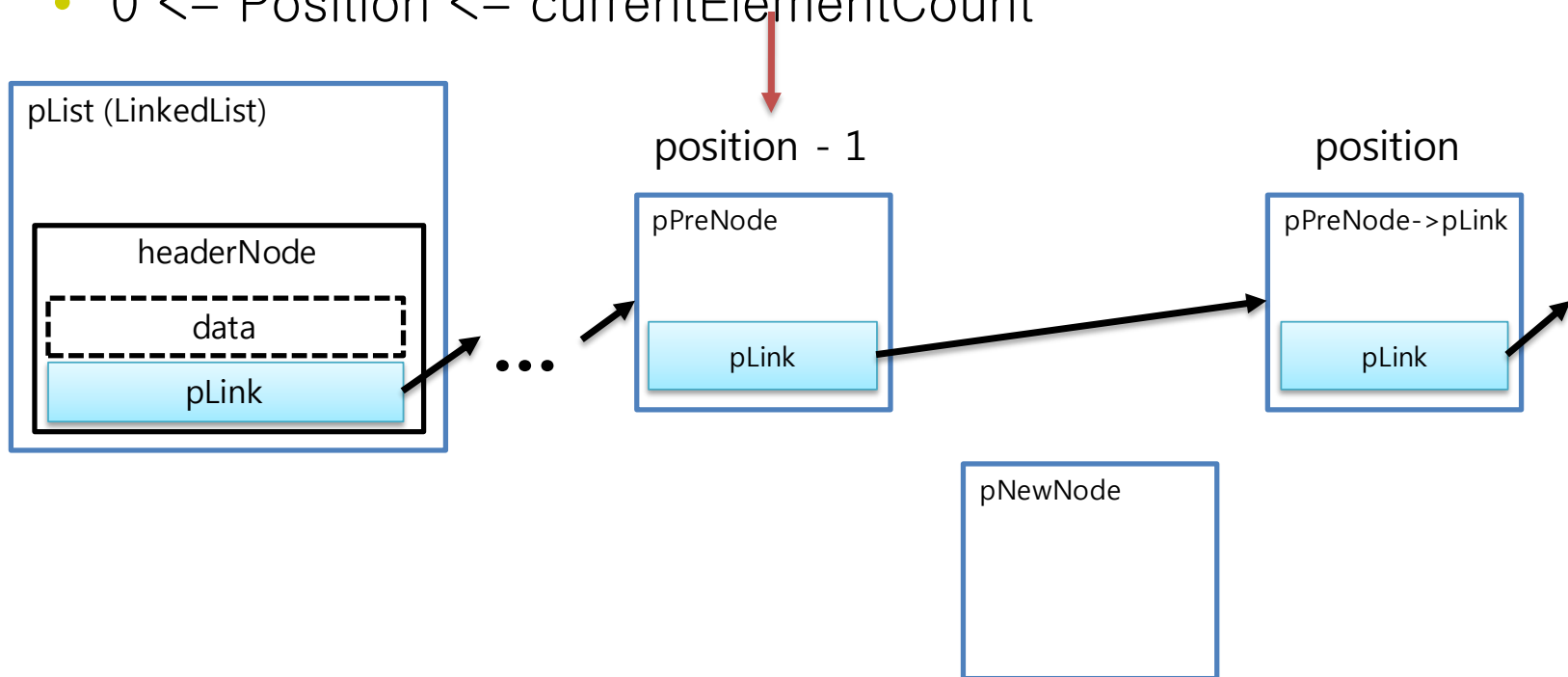
- 헤더 노드(Header Node)
  - 사용 목적
    - 예) 교재-단순 연결 리스트, 이중 연결 리스트
  - Cf) 헤드 포인터(Head Pointer)
    - 원형 연결 리스트의 구현

## 5. 단순 연결 리스트 (3/6)

- 연결 리스트의 생성
- 노드 추가
- 노드 제거
- 리스트 원소 반환과 리스트 순회
- 기타
- 예제 프로그램

## 5. 단순 연결 리스트 (4/6)

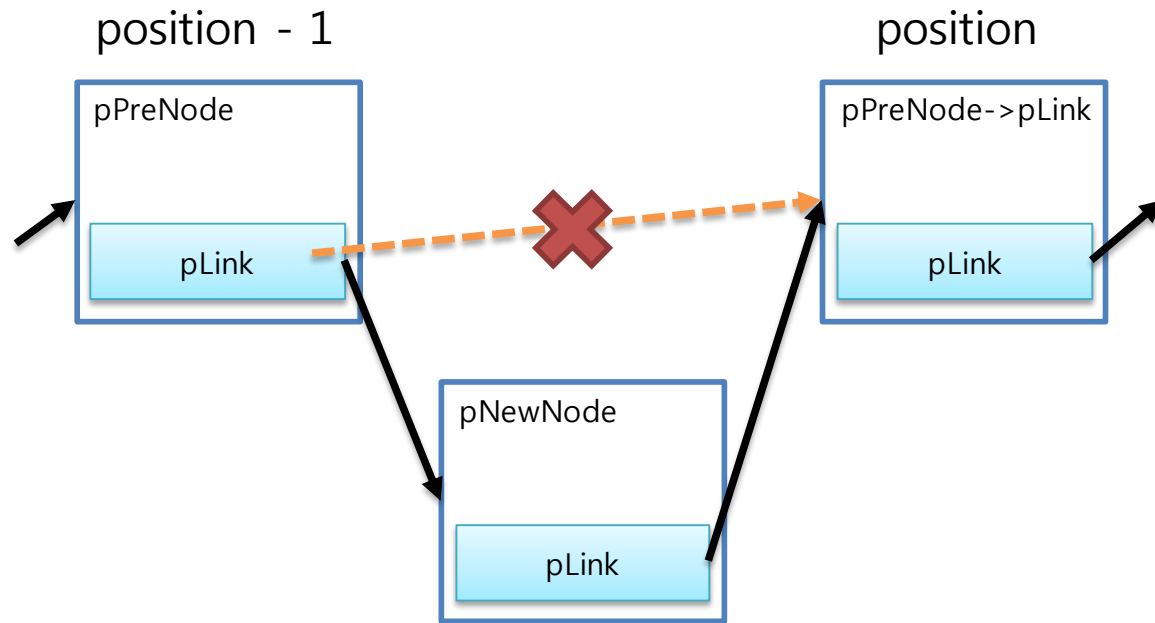
- 노드 추가
  - linkedlist.c / addLLElement()
  - Position의 범위?
    - $0 \leq \text{Position} \leq \text{currentElementCount}$





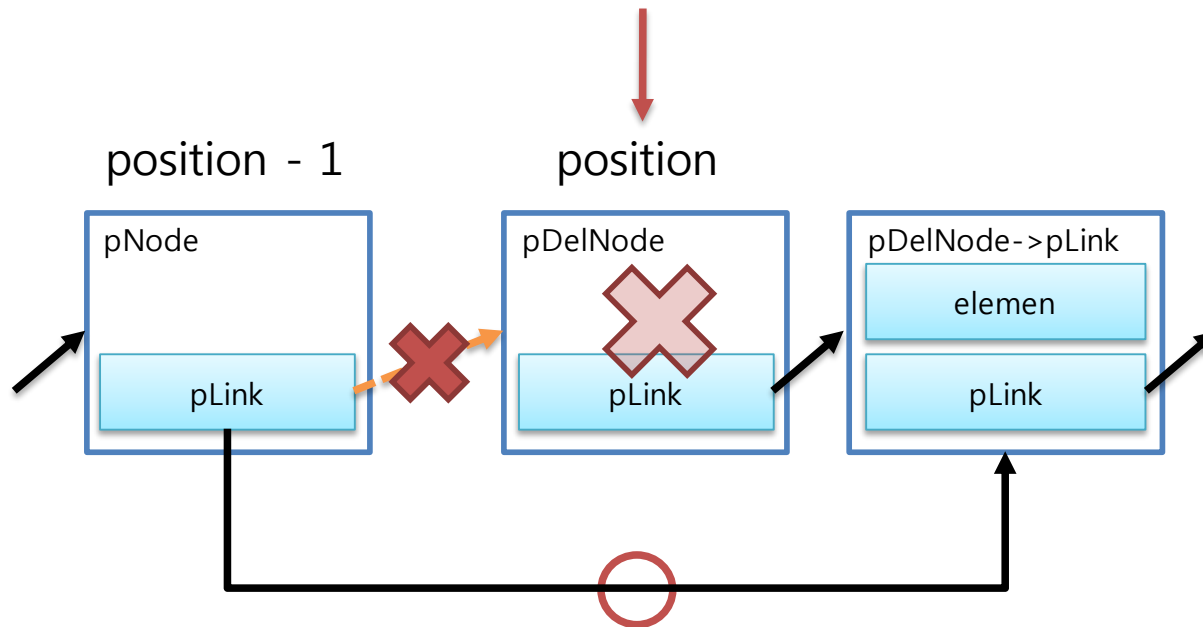
## 5. 단순 연결 리스트 (5/6)

- 노드 추가



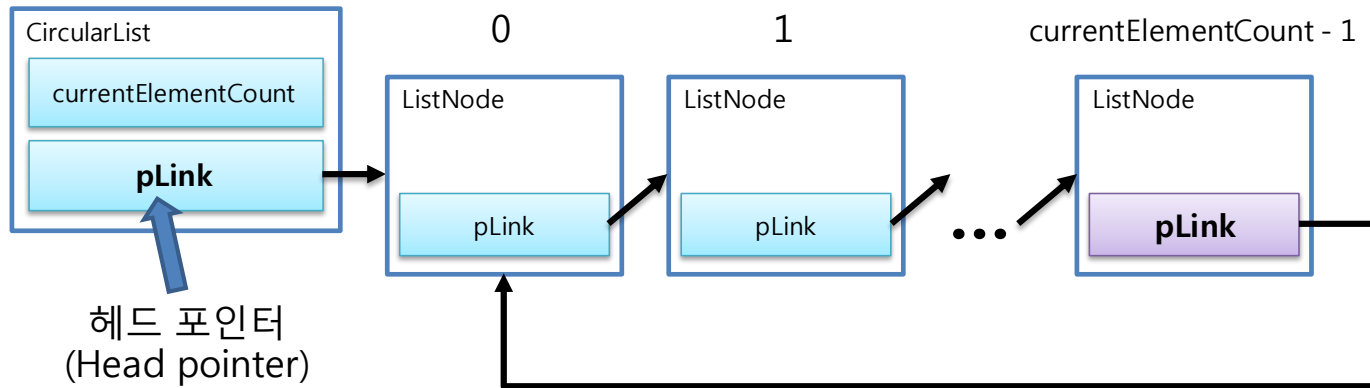
## 5. 단순 연결 리스트 (6/6)

- 노드 제거
  - removeLLElement()



## 6. 원형 연결 리스트 (1/11)

- 원형 연결 리스트(Circular Linked List)
  - 순환(Circular) 구조



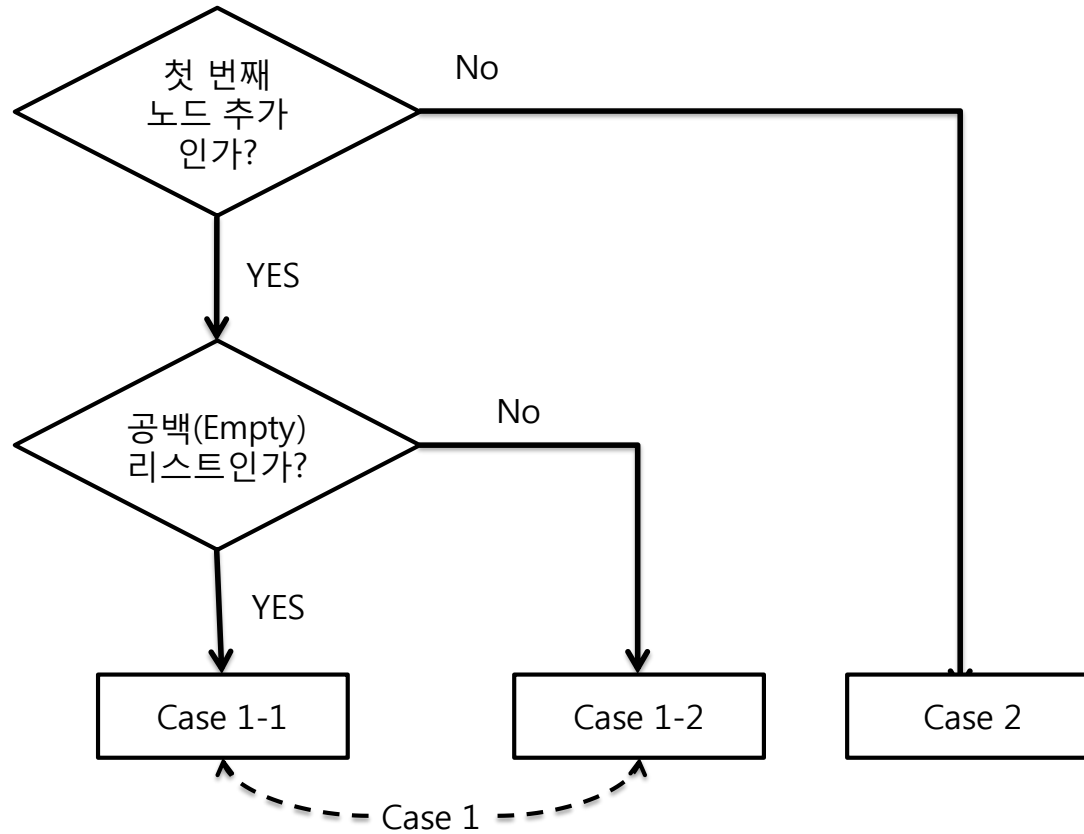
- 헤드 포인터 (Head pointer)

## 6. 원형 연결 리스트 (3/11)

- 연결 리스트의 생성
- 노드 추가
- 노드 제거
- 리스트 원소 반환과 리스트 순회
- 기타
- 예제 프로그램

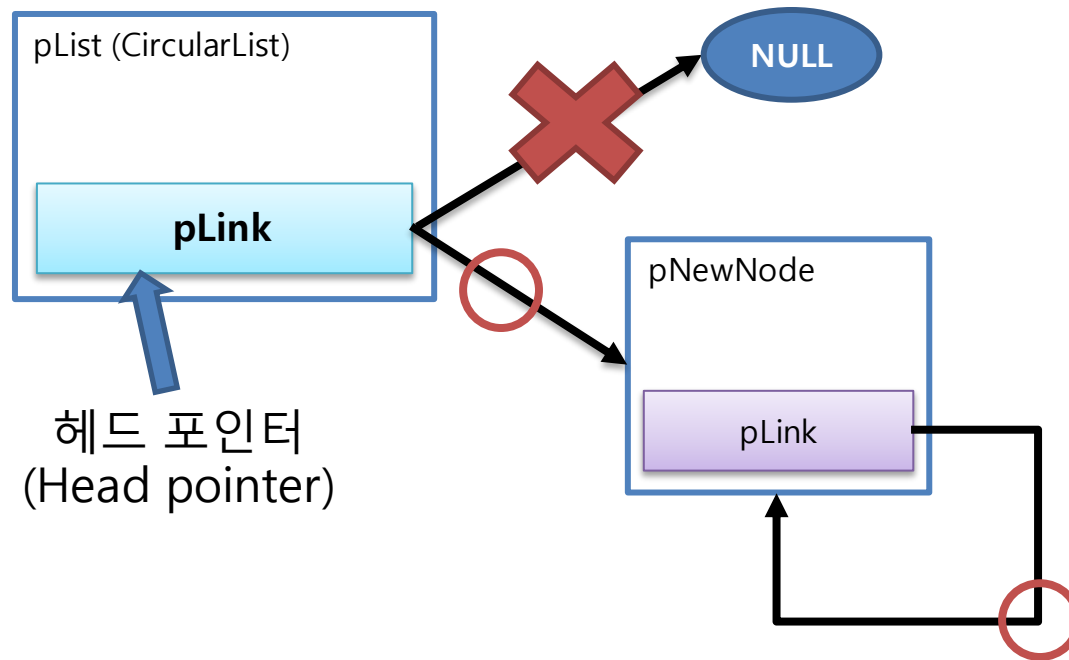
## 6. 원형 연결 리스트 (4/11)

- 노드 추가
  - 헤드 포인터 사용



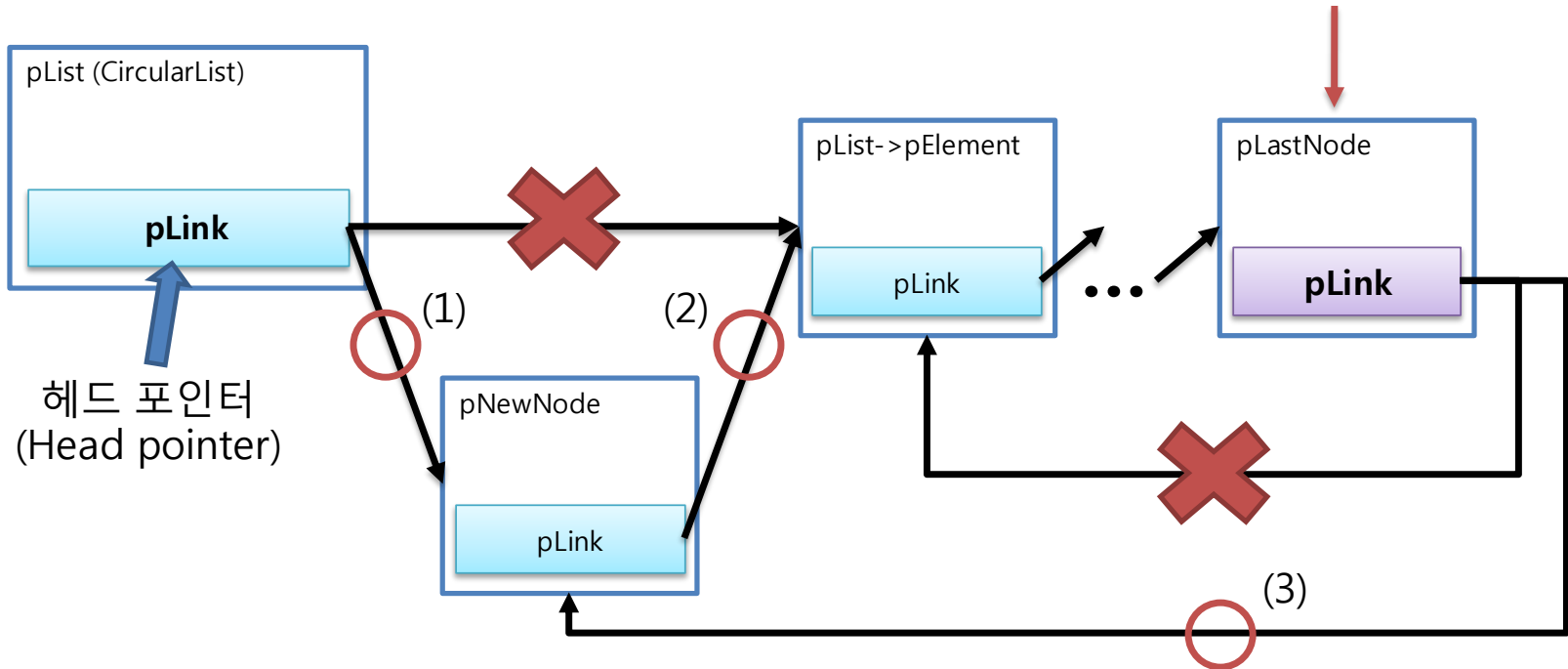
## 6. 원형 연결 리스트 (5/11)

- 노드 추가: Case 1-1



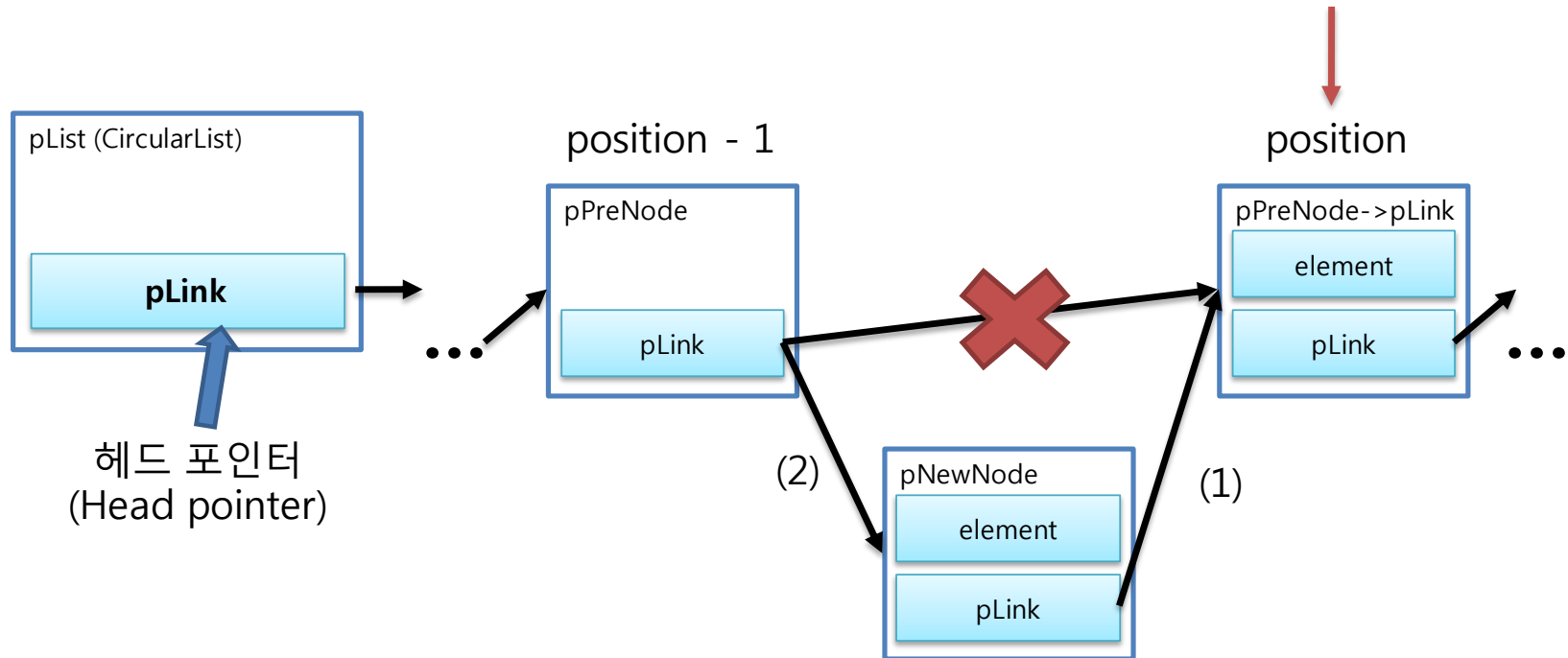
## 6. 원형 연결 리스트 (6/11)

- 노드 추가: Case 1-2



## 6. 원형 연결 리스트 (7/11)

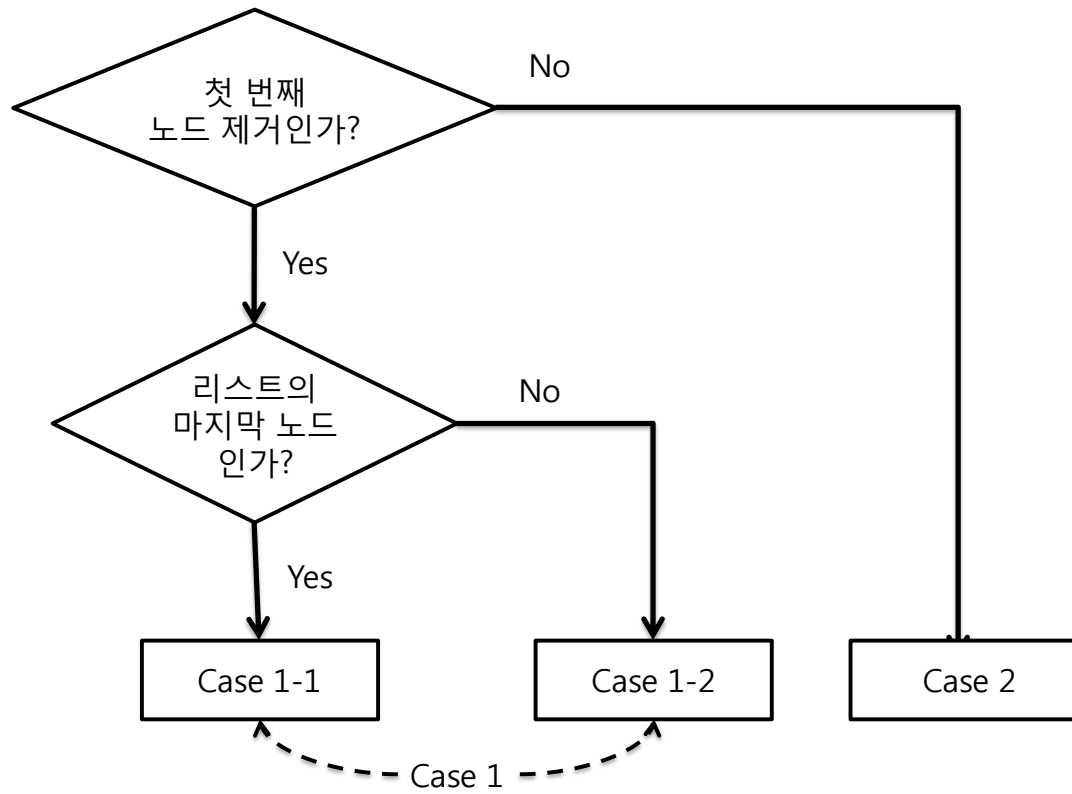
- 노드 추가: Case 2





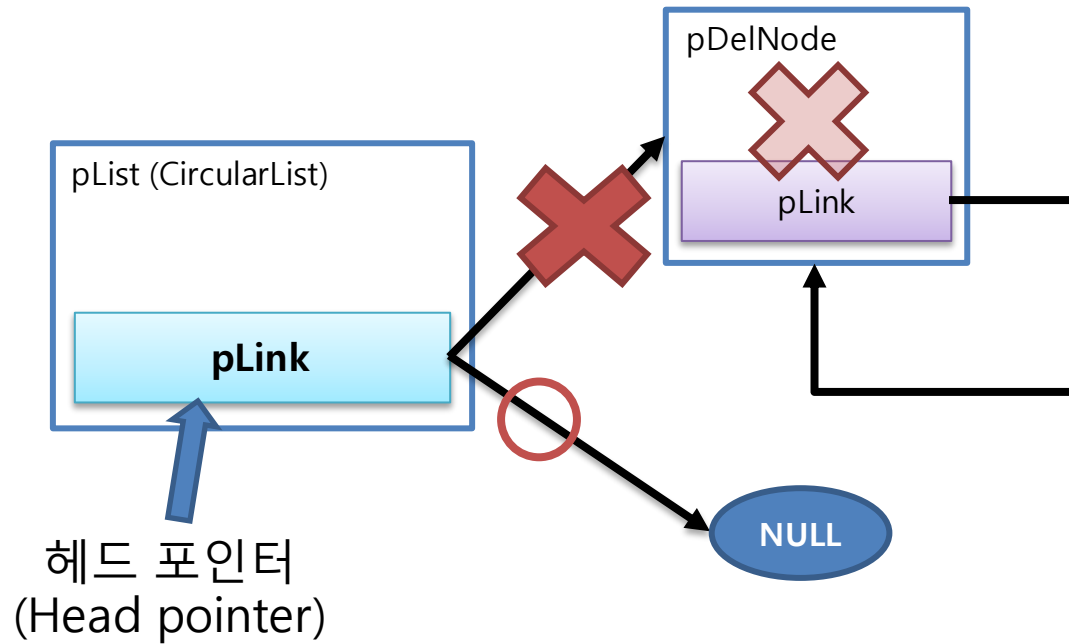
## 6. 원형 연결 리스트 (8/11)

- 노드 제거
  - 헤드 포인터



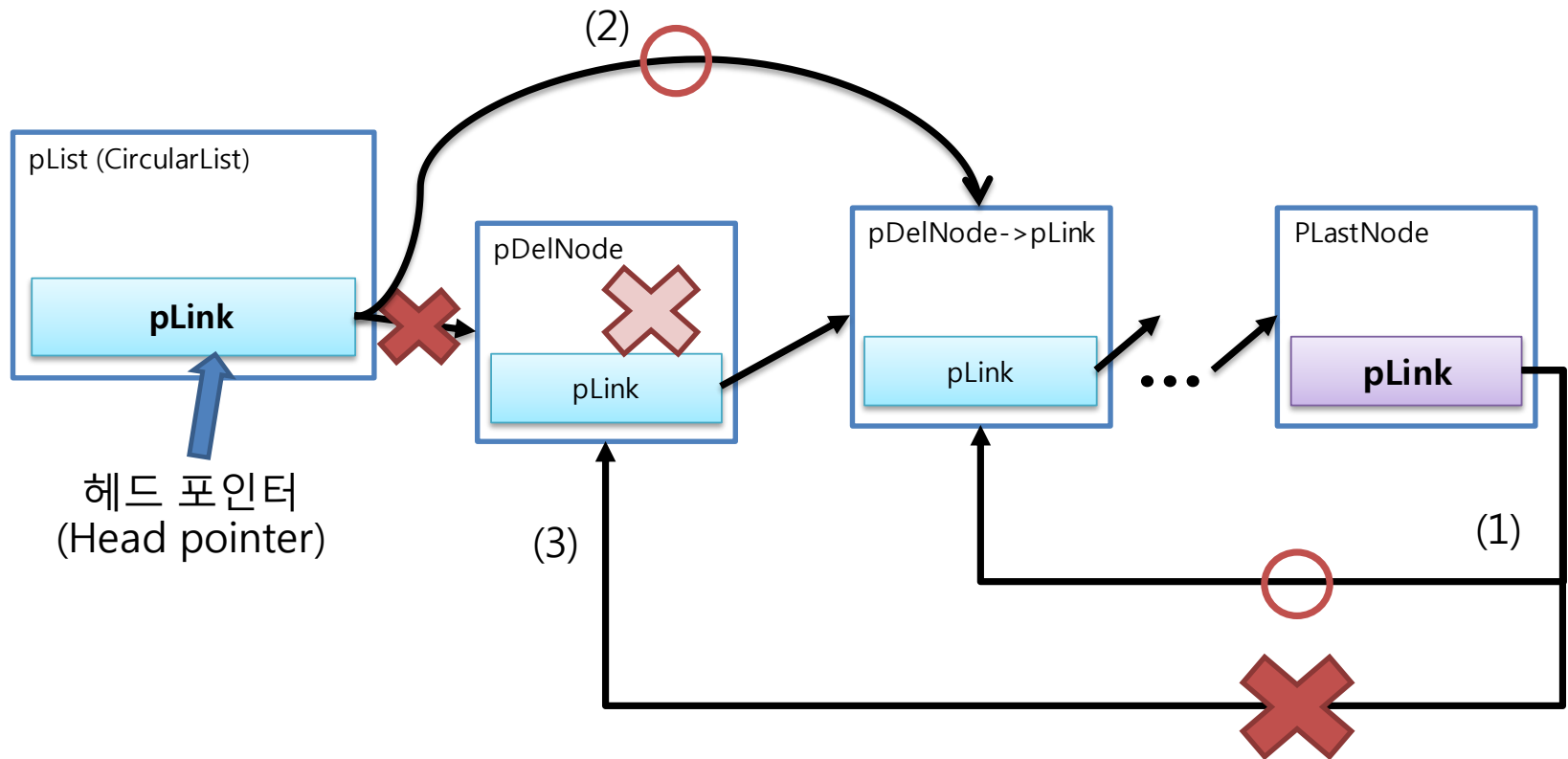
## 6. 원형 연결 리스트 (9/11)

- 노드 제거: Case 1



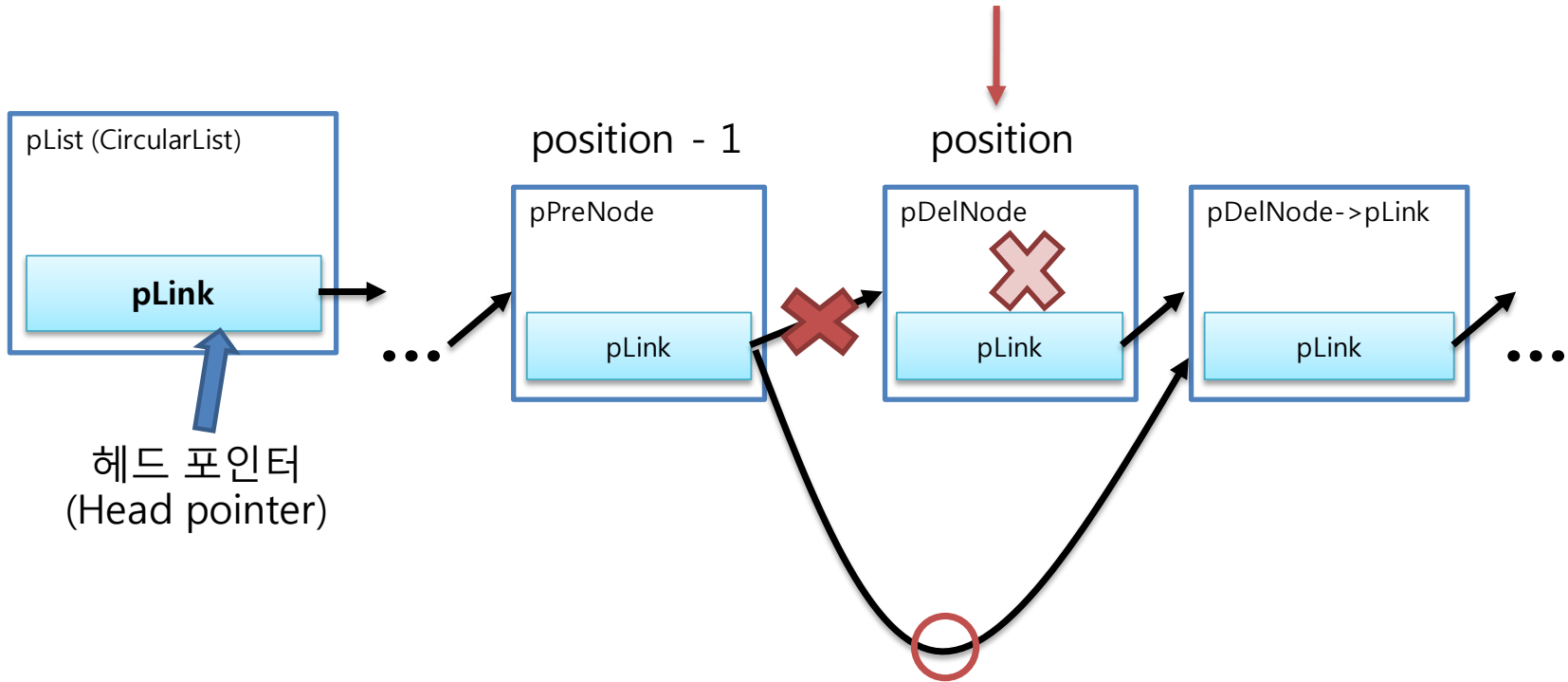
## 6. 원형 연결 리스트 (10/11)

- 노드 제거: Case 2-1



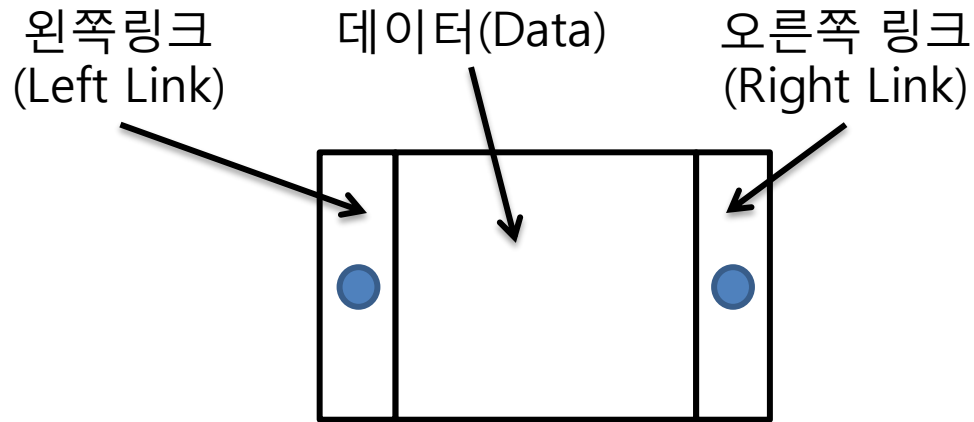
## 6. 원형 연결 리스트 (11/11)

- 노드 제거: Case 2-2



## 7. 이중 연결 리스트

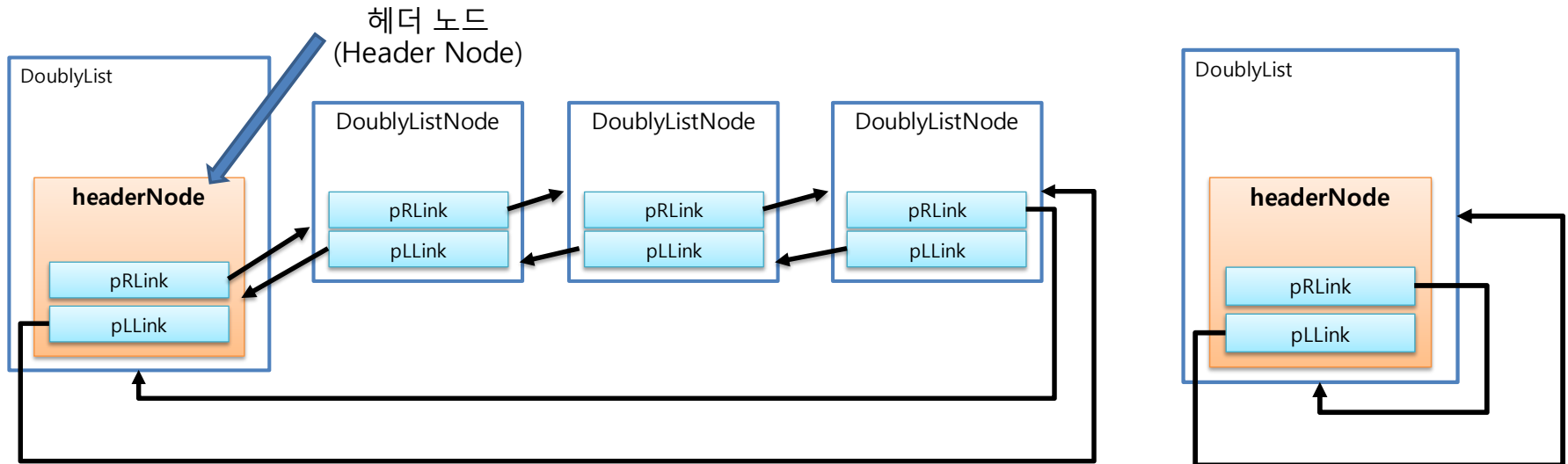
- 이중 연결 리스트(Doubly Linked List)



- 장점
  - 접근 편의성
- 단점
  - 추가적 메모리 공간 사용

## 7. 이중 연결 리스트

- 헤더 노드(Header Node)의 사용



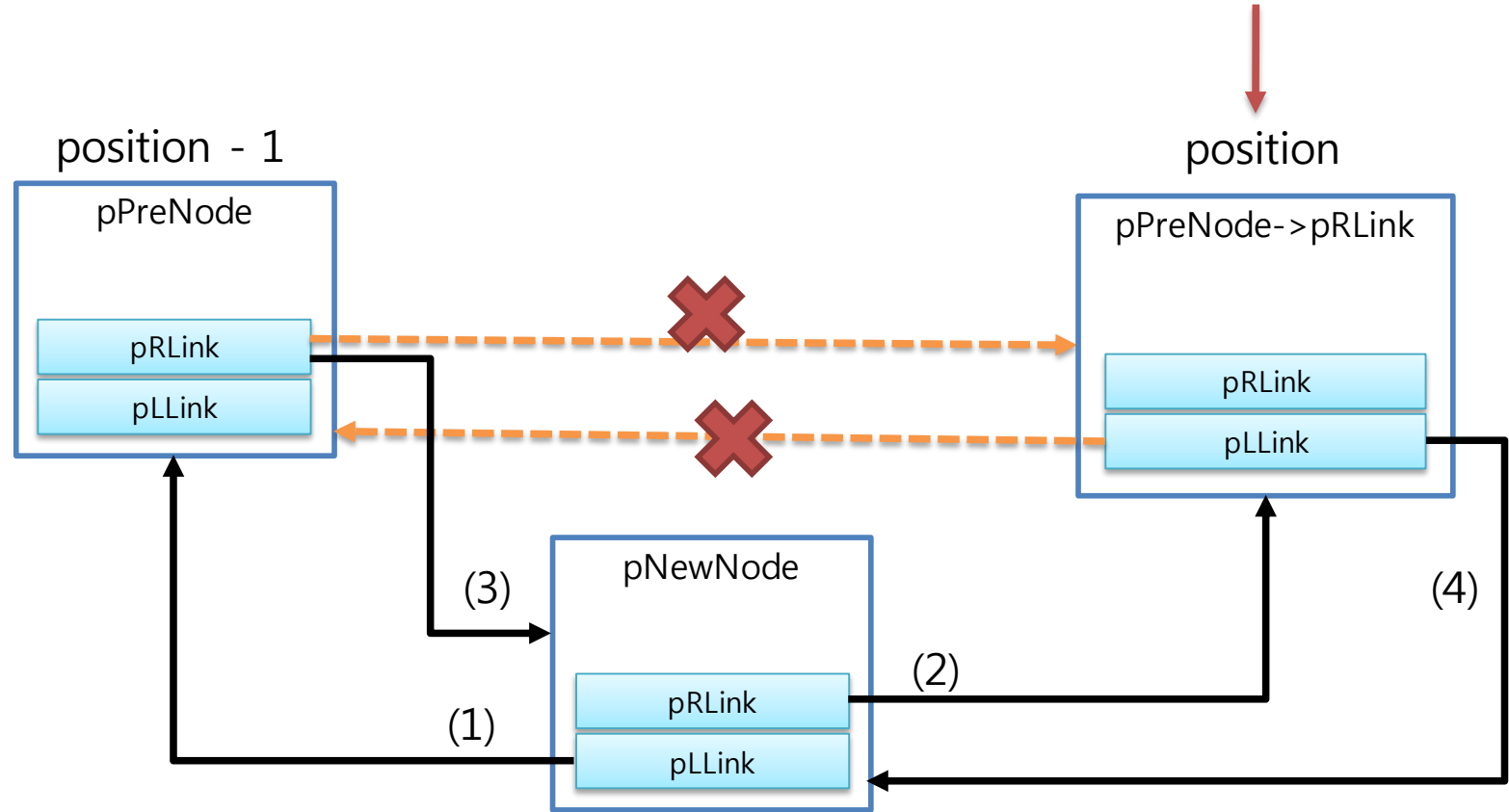
$pNode == pNode \rightarrow pLLink \rightarrow pRLink == pNode \rightarrow pRLink \rightarrow pLLink$

## 7. 이중 연결 리스트

- 연결 리스트의 생성
- 노드 추가
- 노드 제거
- 기타
- 예제 프로그램
  - example03\_04.c

## 7. 이중 연결 리스트

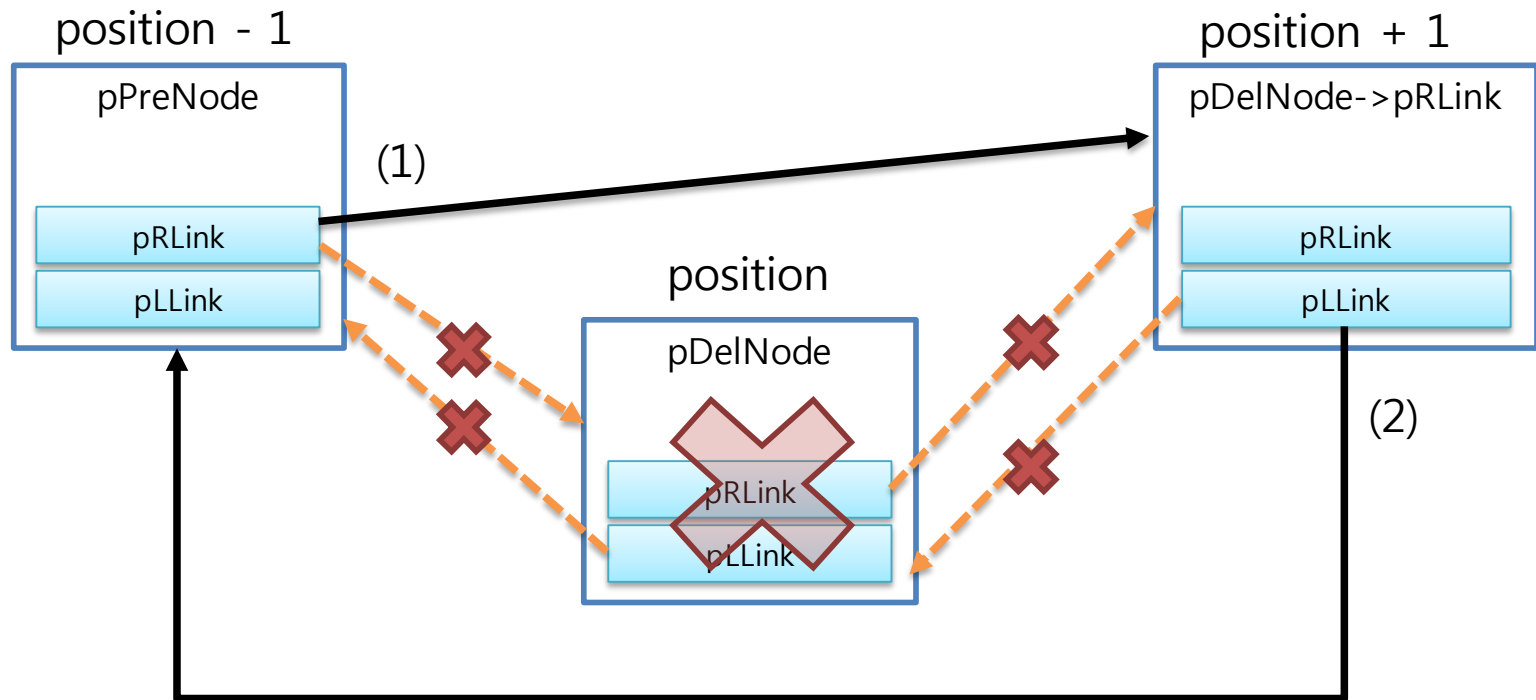
- 노드 추가





## 7. 이중 연결 리스트

- 노드 제거

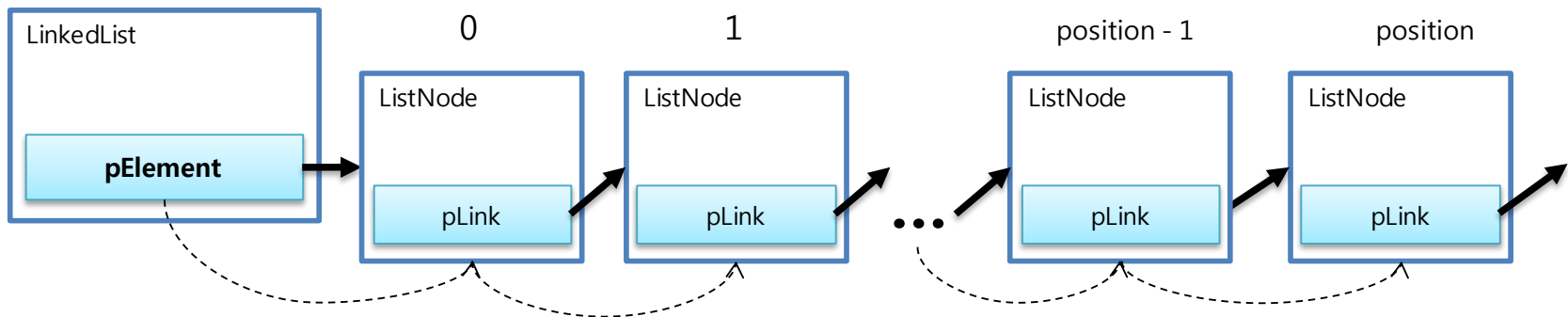


## 8.1.1. 연결 리스트 순회

- 기존의 구현된 리스트 순회 함수
  - 함수 `displayLinkedList()`

## 8. 연결 리스트의 응용

```
for(i = 0; i < pList->currentElementCount; i++) {  
    printf("[%d],%d\n", i, getLLElement(pList, i)->element);  
}
```



## 8.1.1. 연결 리스트 순회

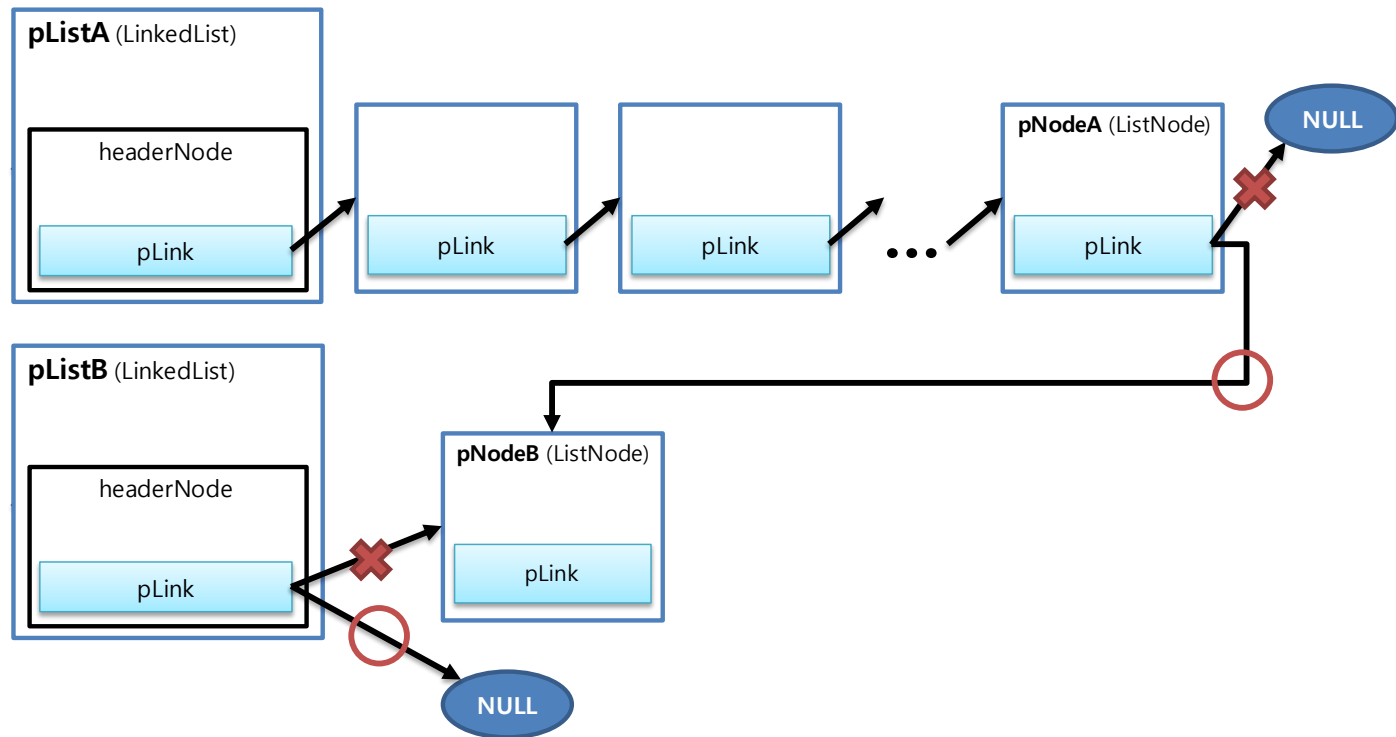
- 순회 함수 iterateLinkedList()

## 8. 연결 리스트의 응용

```
06: void iterateLinkedList(LinkedList* pList)
07: {
08:     ListNode* pNode = NULL;
09:     int count = 0;
10:     if (pList != NULL) {
11:         pNode = pList->headerNode.pLink;
12:         while(pNode != NULL) {
13:             printf("[%d],%d\n", count, pNode->data);
14:             count++;
15:
16:             pNode = pNode->pLink;
17:         }
18:         printf("노드 개수: %d\n", count);
19:     }
20:     else {
21:         printf("공백 리스트입니다");
22:     }
23: }
```

## 8.1.2. 연결 리스트 연결

### 8. 연결 리스트의 응용



- 주의할 사항
  - 링크 제거(메모리 해제)
  - 노드 개수 변경

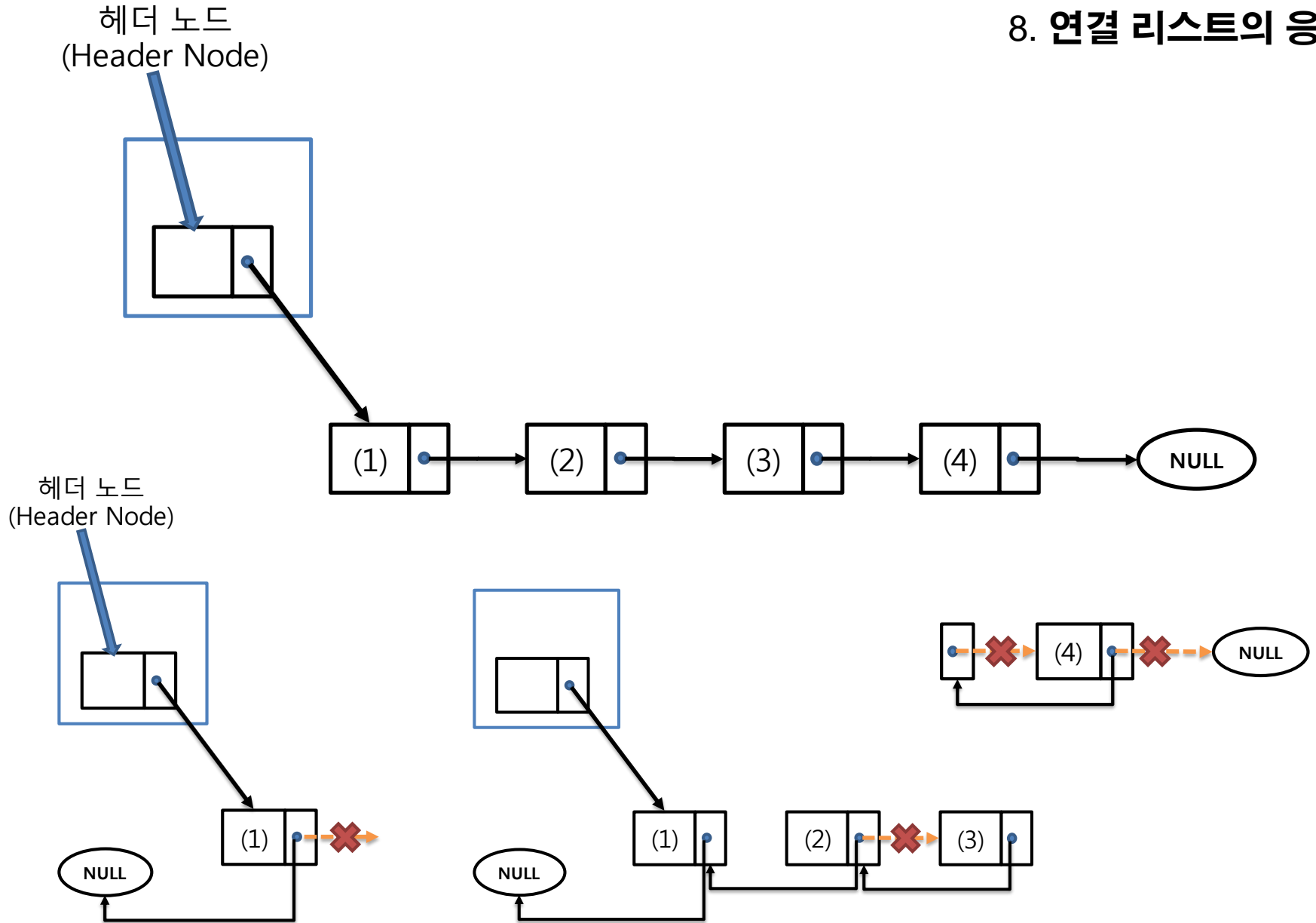
## 8.1.2. 연결 리스트 연결

### 8. 연결 리스트의 응용

```
25: void concatLinkedList(LinkedList* pListA, LinkedList* pListB)
26: {
27:     ListNode *pNodeA = NULL, *pNodeB = NULL;
28:
29:     if (pListA != NULL && pListB != NULL) {
30:         pNodeA = pListA->headerNode.pLink;
31:         while(pNodeA->pLink != NULL) {
32:             pNodeA = pNodeA->pLink;
33:         }
34:         pNodeA->pLink = pListB->headerNode.pLink;
35:         pListA->currentElementCount += pListB->currentElementCount;
36:
37:         pListB->headerNode.pLink = NULL;
38:         pListB->currentElementCount = 0;
39:     }
40: }
```

## 8.1.3. 연결 리스트 역순 만들기

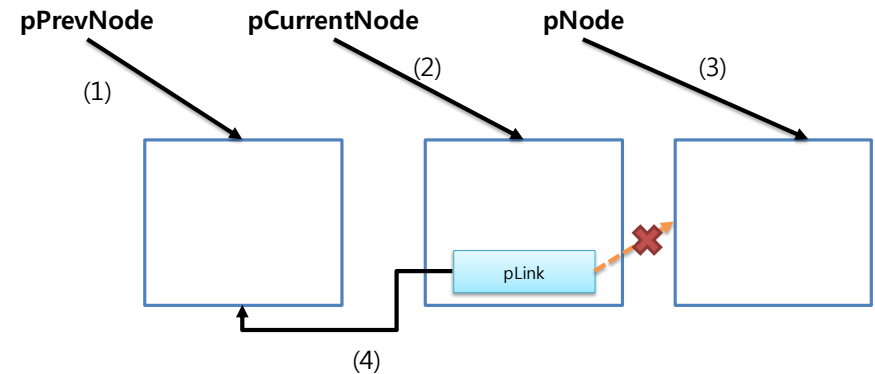
### 8. 연결 리스트의 응용



## 8.1.3. 연결 리스트 역순 만들기

### 8. 연결 리스트의 응용

```
42: void reverseLinkedList(LinkedList* pList)
43: {
44:     ListNode *pNode = NULL, *pCurrentNode = NULL, *pPrevNode = NULL;
45:
46:     if (pList != NULL) {
47:         pNode = pList->headerNode.pLink;
48:         while(pNode != NULL) {
49:             pPrevNode = pCurrentNode;           // (1)
50:             pCurrentNode = pNode;                // (2)
51:             pNode = pNode->pLink;                // (3)
52:             pCurrentNode->pLink = pPrevNode;     // (4)
53:         }
54:
55:         pList->headerNode.pLink = pCurrentNode;
56:     }
57: }
```



## 8.2. 다항식

### 8. 연결 리스트의 응용

- 다항식

- 항: 계수 + 차수
- 일반적인 예

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x^1 + a_0x^0$$

- 항
- 계수
- 차수
- 구현 방법
  - 단순 연결 리스트 활용
  - E.g. 덧셈 연산



## 8.2. 다항식

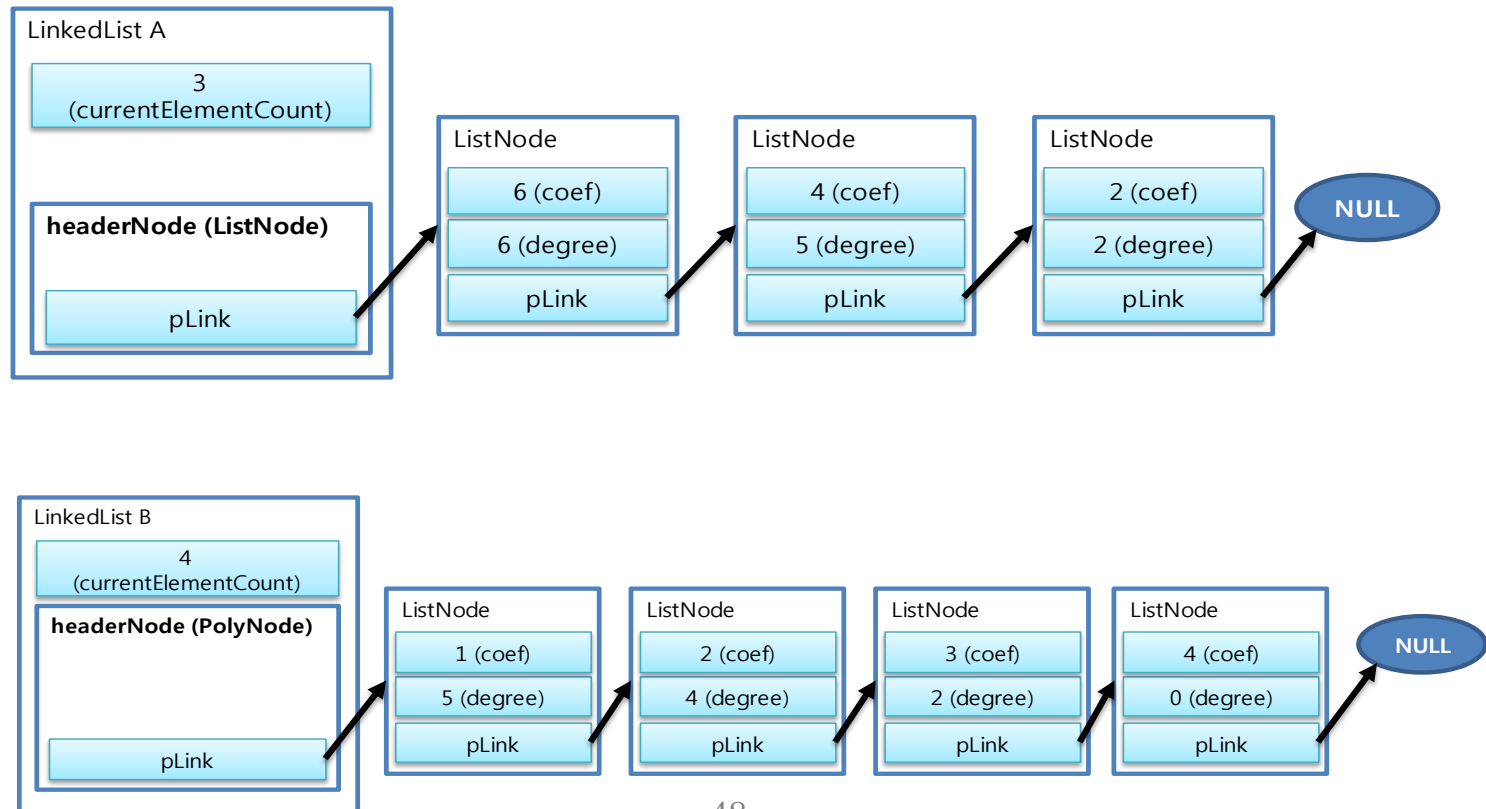
### 8. 연결 리스트의 응용

- 다항식의 덧셈

- 예

다항식 A(x):  $6x^6 + 4x^5 + 2x^2$

다항식 B(x):  $1x^5 + 2x^4 + 3x^2 + 4$



## 8.2. 다항식

- 다항식의 생성: 다항식의 항 추가 연산

## 8. 연결 리스트의 응용

```
006:    int addPolyNodeLast(LinkedList* pList, float coef, int degree)
007:    {
008:        int ret = FALSE, i = 0;
009:
010:        ListNode node = {0,};
011:        node.coef = coef;
012:        node.degree = degree;
013:
014:        if (pList != NULL) {
015:            int length = getLinkedListLength(pList);
016:            ret = addLLElement(pList, length, node);
017:        }
018:
019:        return ret;
020:    }
```

## 8.2. 다항식

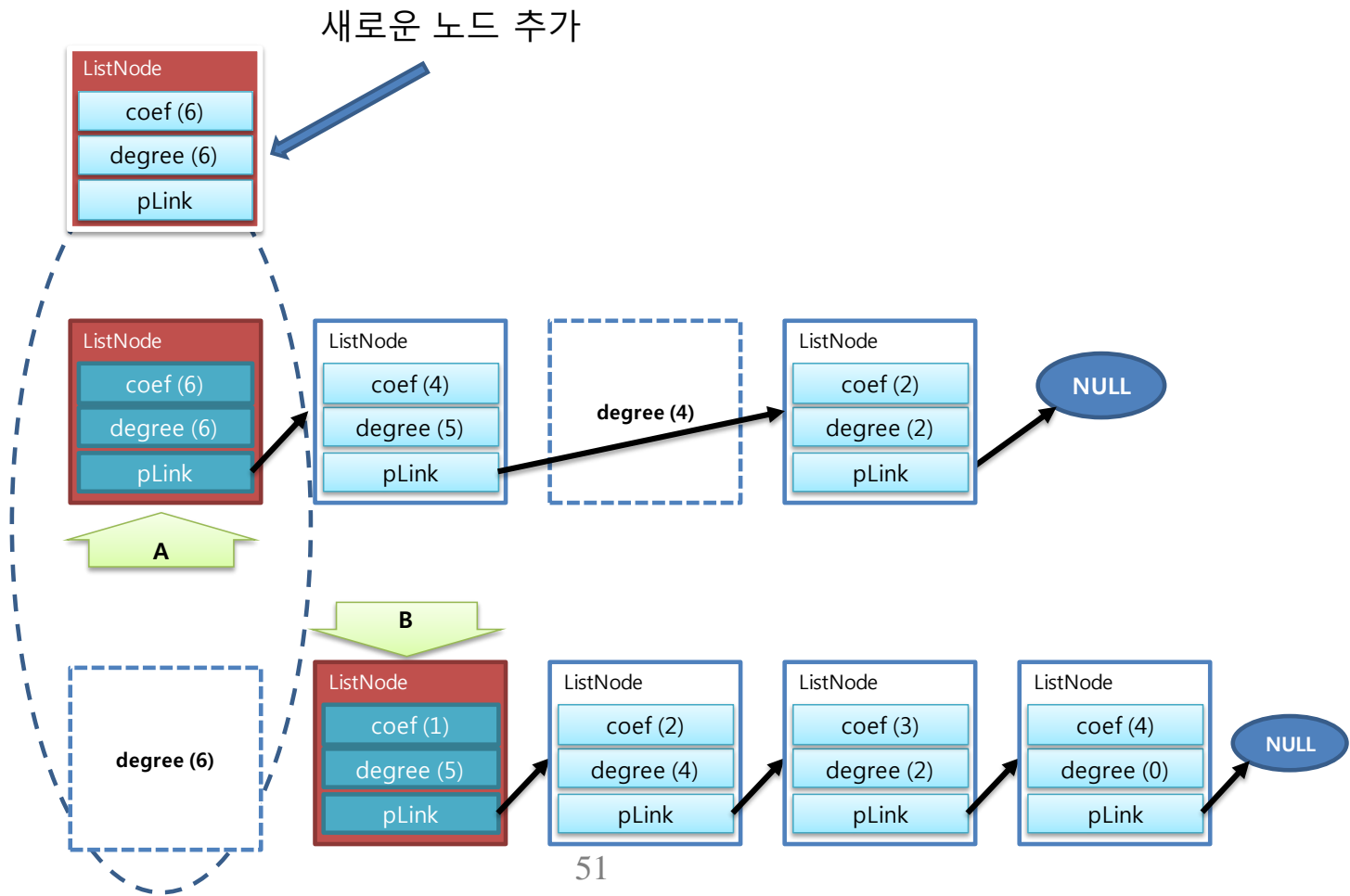
- 다항식 덧셈 연산
  - 다항식 A의 항(Term) 차수가 높은 경우
  - 다항식 B의 항(Term) 차수가 높은 경우
  - 다항식 A와 다항식 B의 차수가 같은 경우
  - 남은 노드 처리

## 8. 연결 리스트의 응용

## 8.2. 다항식

- 다항식 A의 항(Term) 차수가 높은 경우

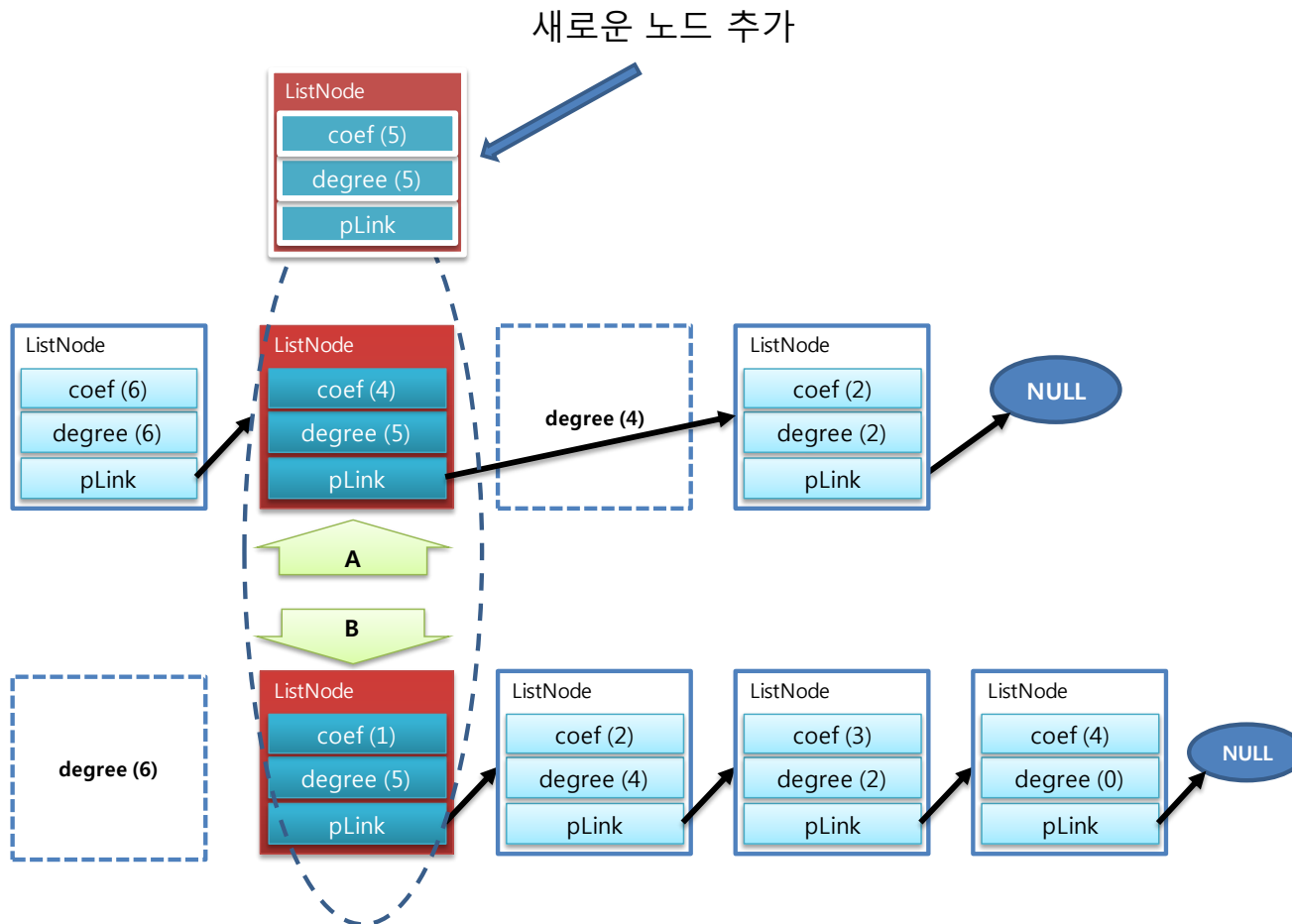
## 8. 연결 리스트의 응용



## 8.2. 다항식

- 다항식 A와 다항식 B의 항 차수가 같은 경우

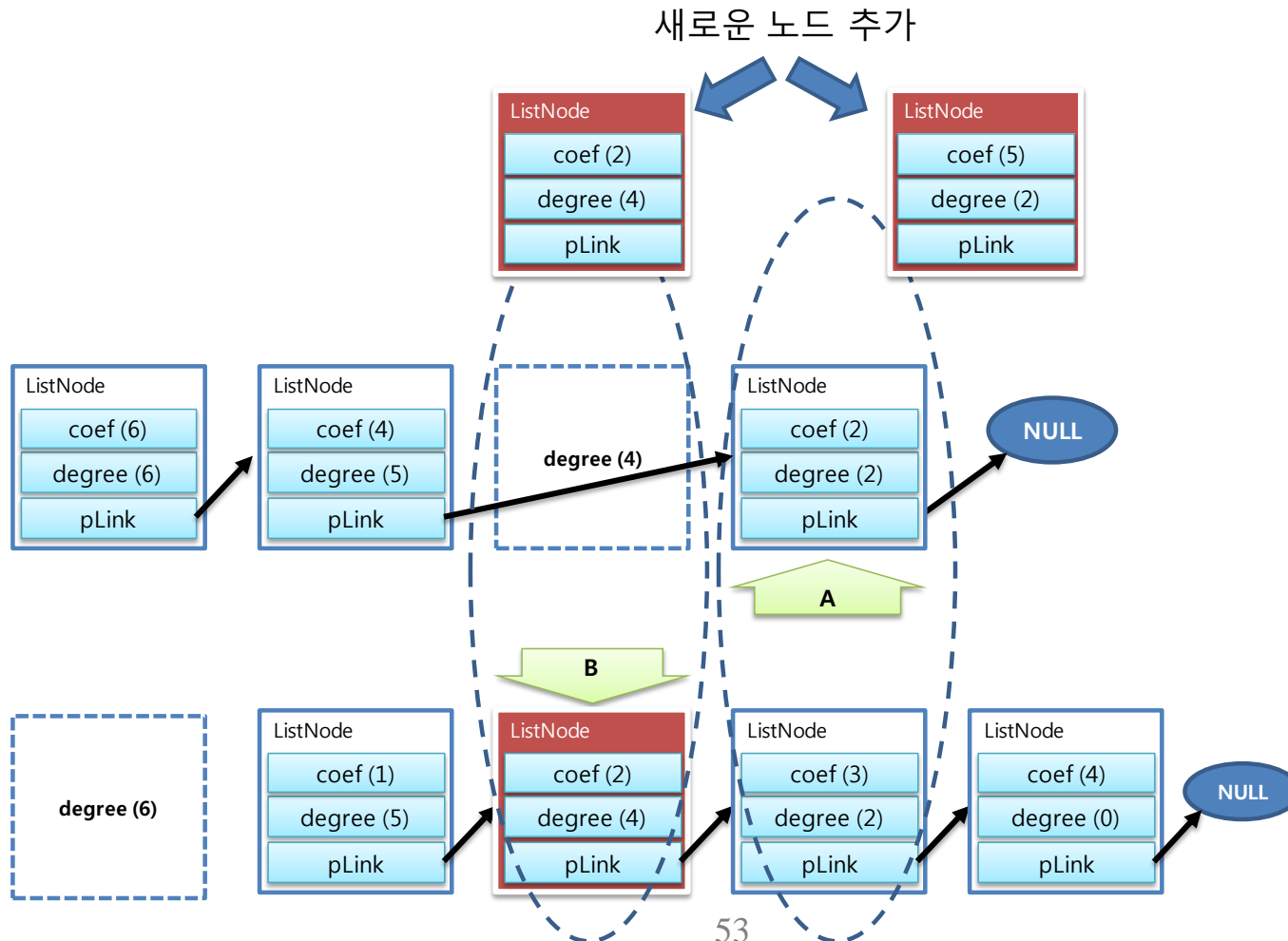
## 8. 연결 리스트의 응용



## 8.2. 다항식

- 다항식 B의 항(Term) 차수가 높은 경우

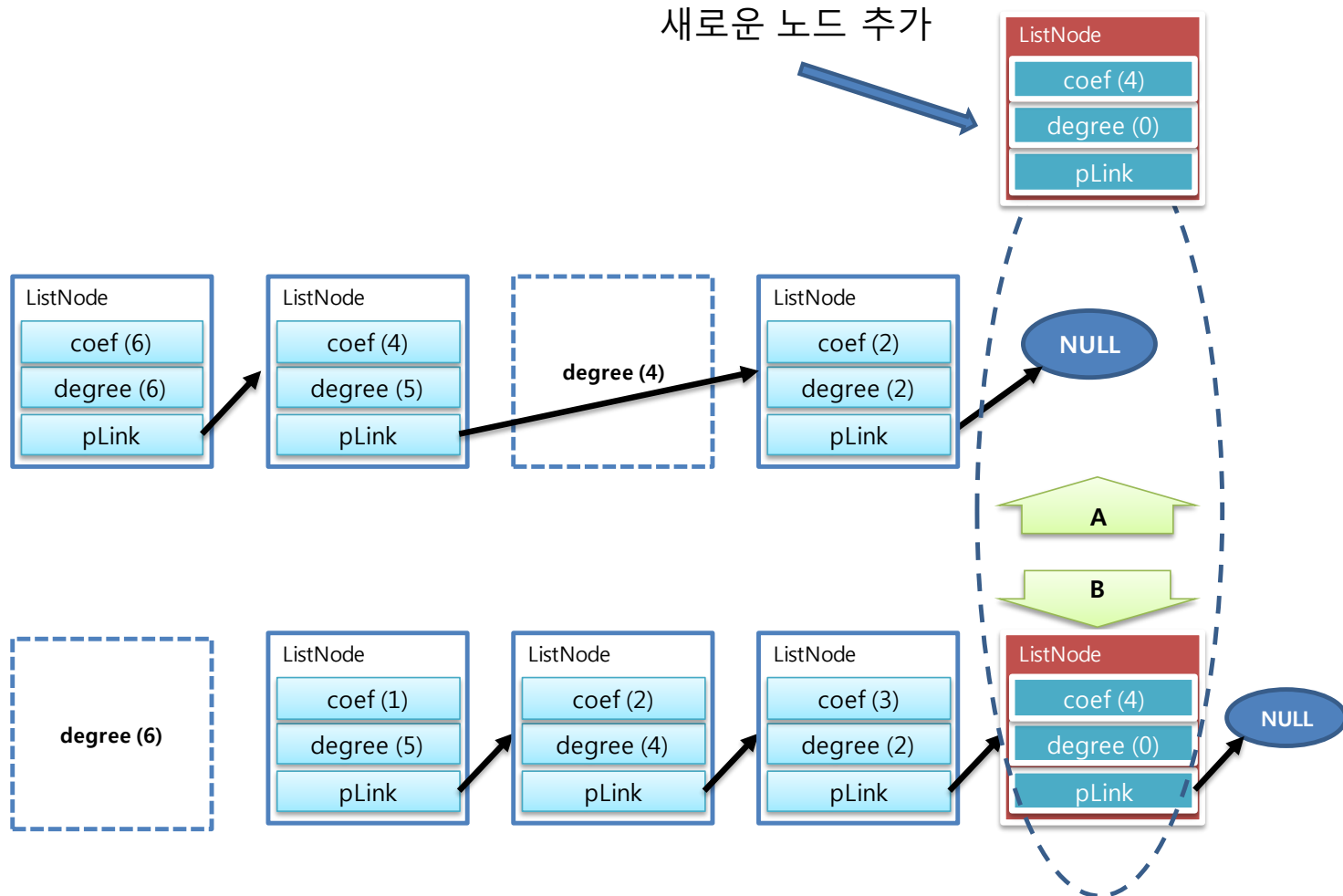
## 8. 연결 리스트의 응용



## 8.2. 다항식

- 남은 노드 처리

## 8. 연결 리스트의 응용



## 이번 장에서는

- 리스트의 개념
- 리스트 추상 자료형
- 배열 리스트
- 연결 리스트의 개념
- 단순 연결 리스트
- 원형 연결 리스트
- 이중 연결 리스트
- 연결 리스트의 응용