연결 자료 구조
(Linked data structure)

남춘성

연결 자료구조(Linked Data Structure)

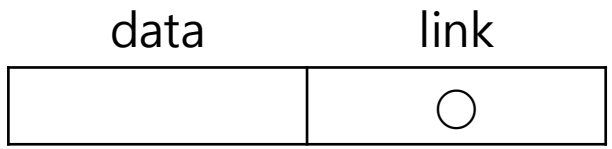
- **순차 자료 구조의 문제점**
  - 삽입연산이나 삭제연사 후에 연속적인 물리 주소를 유지하기 위해서 원소들을 이동시키는 추가적인 작업과 시간 소요
    - 원소들의 이동 작업으로 인한 오버헤드는 원소의 개수가 많고 삽입 및 삭제 연산이 많이 발생하는 겨우 성능상의 문제 발생
  - 순차 자료구조는 배열을 이용하여 구현하기 대문에 배열이 갖고 있는 메모리 사용의 비효율성 문제를 그대로 가짐
  - 순차 자료구조에서의 연산 시간에 대한 문제와 저장 공간에 대한 문제를 개선한 자료 표현 방법 필요

연결 자료구조(Linked Data Structure)

- **연결 자료구조(Linked Data Structure)**
  - 자료의 논리적인 순서와 물리적인 순서가 일치하지 않는 자료구조
    - 각 원소에 저장되어 있는 다음 원소의 주소에 의해 순서가 연결되는 방식
      - **물리적인 순서를 맞추기 위한 오버헤드가 발생하지 않음**
    - 여러 개의 작은 공간을 연결하여 하나의 전체 자료구조를 표현
      - **크기 변경이 유연하고 더 효율적으로 메모리를 사용**
  - 연결 리스트
    - 리스트를 연결 자료구조로 표현한 구조
    - 연결하는 방식에 따라 단순 연결 리스트와 원형 연결 리스트, 이중 연결 리스트, 이중 원형 연결 리스트

연결 자료구조(Linked Data Structure)

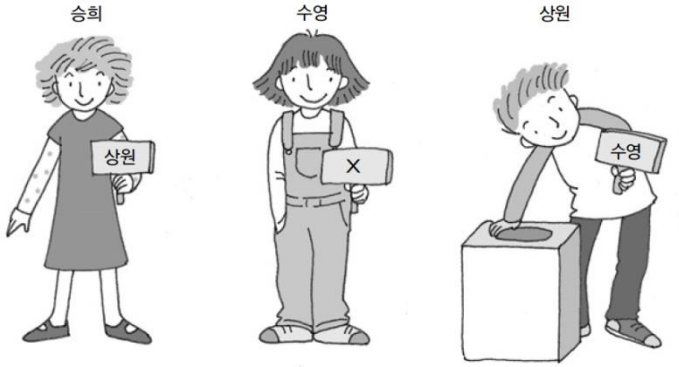
- **연결 리스트의 노드**
  - 연결 자료구조에서 하나의 원소를 표현하기 위한 단위 구조
  - <원소, 주소>의 구조



  - 데이터 필드(data field)
    - 원소의 값을 저장
    - 저장할 원소의 형태에 따라서 하나 이상의 필드로 구성
  - 링크 필드(link field)
    - 다음 노드의 주소를 저장
    - 포인터 변수를 사용하여 주소 값을 저장

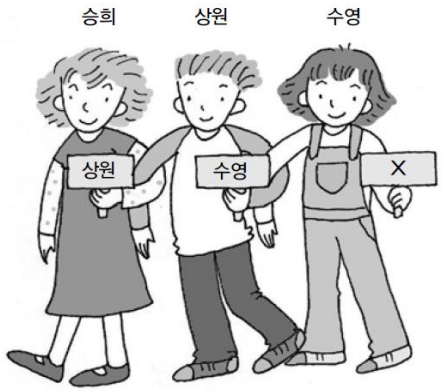
연결 자료구조(Linked Data Structure)

- **노드 연결 방법에 대한 이해 – 기차놀이**
  - 이름표 뽑기



[그림 5-3] 기차놀이 : 이름표 뽑기

  - 자기가 뽑은 이름의 사람을 찾아서 연결하기 : 승희 -> 상원 -> 수원
    - X표를 뽑은 사람은 마지막 기차
    - 기차는 이름표를 들고 있는 방향으로 움직임



[그림 5-4] 기차놀이 : 뽑은 이름표대로 기차 연결하기

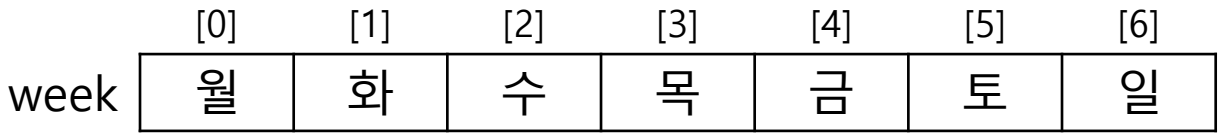
연결 자료구조(Linked Data Structure)

- **노드 연결 방법에 대한 이해 - 기차놀이**
  - 기차놀이와 연결 리스트
    - 기차 놀이 하는 아이들 : 연결 리스트의 노드
    - 이름표 : 노드의 링크 필드

[그림 5-5] 기차놀이에 대한 노드 표현

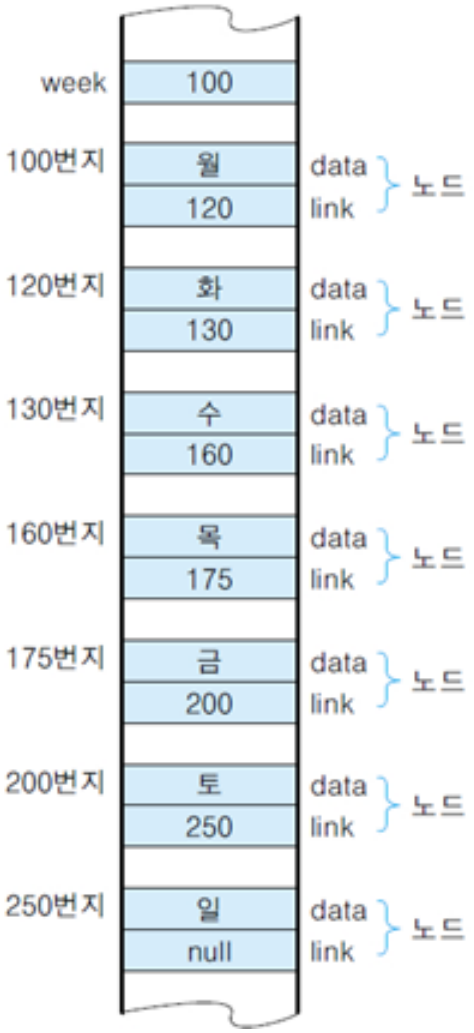
연결 자료구조(Linked Data Structure)

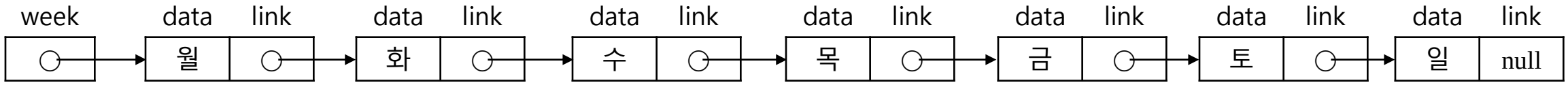
- **선형 리스트와 연결 리스트의 비교**
  - 리스트 week=(월, 화, 수, 목, 금, 토, 일)
  - week에 대한 선형 리스트



연결 자료구조(Linked Data Structure)

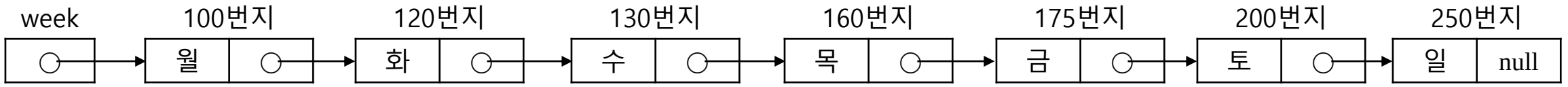
- 리스트 week=(월, 화, 수, 목, 금, 토, 일)
- week에 대한 연결 리스트





연결 자료구조(Linked Data Structure)

- **선형 리스트와 연결 리스트의 비교**
  - **리스트 이름** week - 연결 리스트의 시작을 가리키는 포인터변수
    - 포인터변수 week는 연결 리스트의 첫번째 노드를 가리키는 동시에 연결된 리스트 전체를 의미
  - 연결 리스트의 마지막 노드의 링크필드 - 노드의 끝을 표시하기 위해서 **null**(널) 저장
  - **공백 연결 리스트** - 포인터변수 week에 null을 저장 (널 포인터)
  - 각 노드의 필드에 저장한 값은 포인터의 **점 연산자**를 사용하여 액세스
    - week.data : 포인터 week가 가리키는 노드**의** 데이터 필드 값 "월"
    - week.link : 포인터 week가 가리키는 노드**의** 링크 필드에 저장된 주소값 "120"



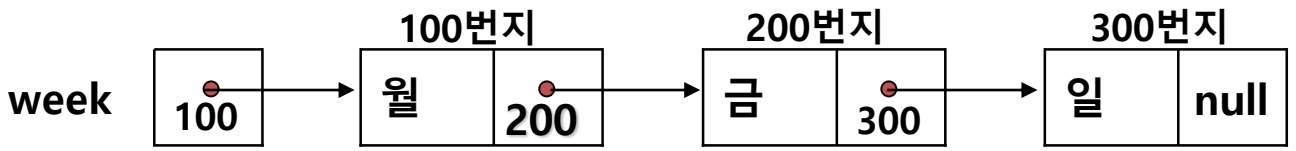
연결 자료구조(Linked Data Structure)

- **선형 리스트와 연결 리스트의 비교**
  - 리스트 week의 노드에 대한 python 프로그램 class 정의

```
class Node:
    def __init__(self, data, next=None):
        self.data = data
        self.next = next
```

연결 자료구조(단순 연결리스트)

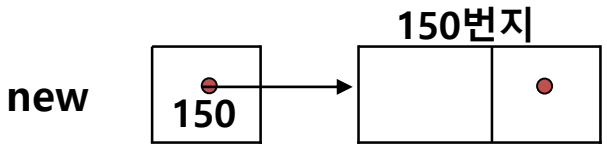
- **단순 연결 리스트(singly linked list)**
  - 노드가 하나의 링크 필드에 의해서 다음 노드와 연결되는 구조를 가진 연결 리스트
  - 연결 리스트, 선형 연결 리스트(linear linked list), 단순 연결 선형 리스트(singly linked linear list)



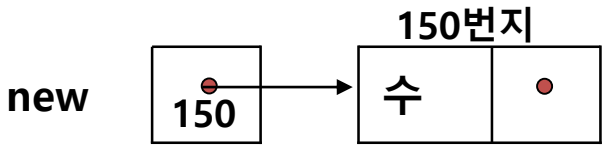
연결 자료구조(단순 연결리스트)

- **단순 연결 리스트의 삽입**
  - 리스트 week2=(월, 금, 일)에서 원소 "월"과 "금"사이에 새 원소"수" 삽입하기

① 삽입할 새 노드를 만들 공백노드를 메모리에서 가져와서 포인터변수 new가 가리키게 한다.



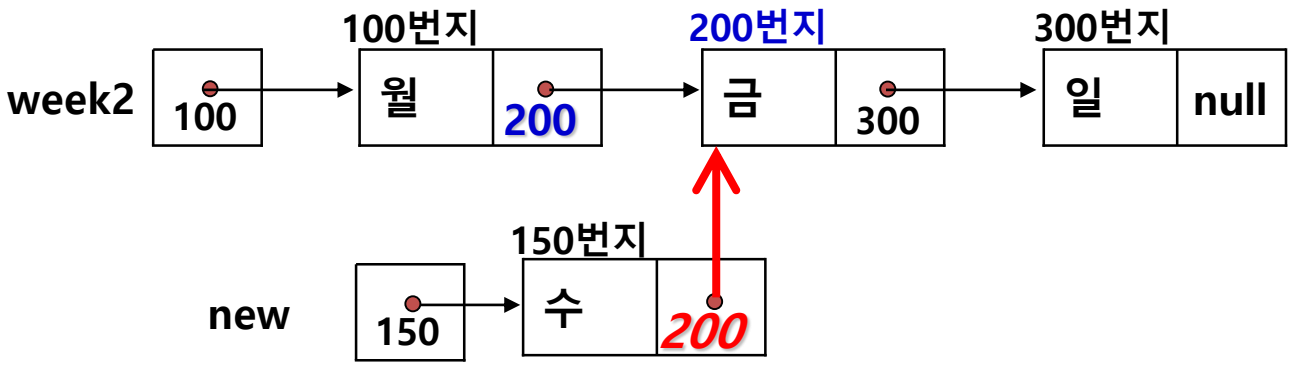
② new의 데이터 필드에 "수"를 저장한다.



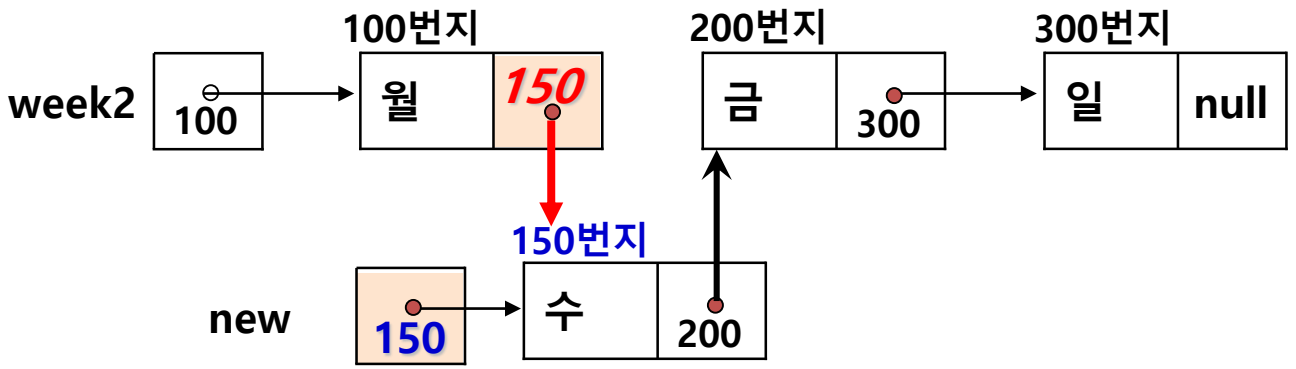
연결 자료구조(단순 연결리스트)

- **단순 연결 리스트의 삽입**

③ new의 앞 노드, 즉 "월"노드의 링크 필드 값을 new의 링크 필드에 저장한다.



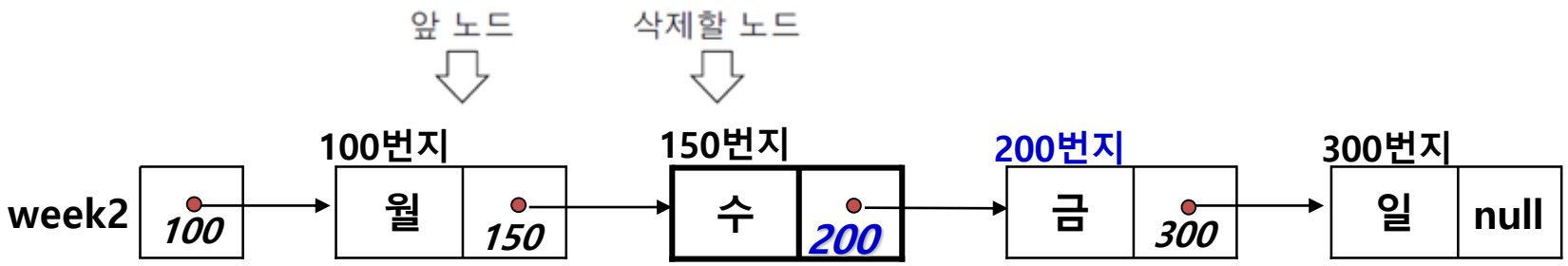
④ new의 값(new가 가리키고 있는 새 노드의 주소)을 "월"노드의 링크 필드에 저장한다



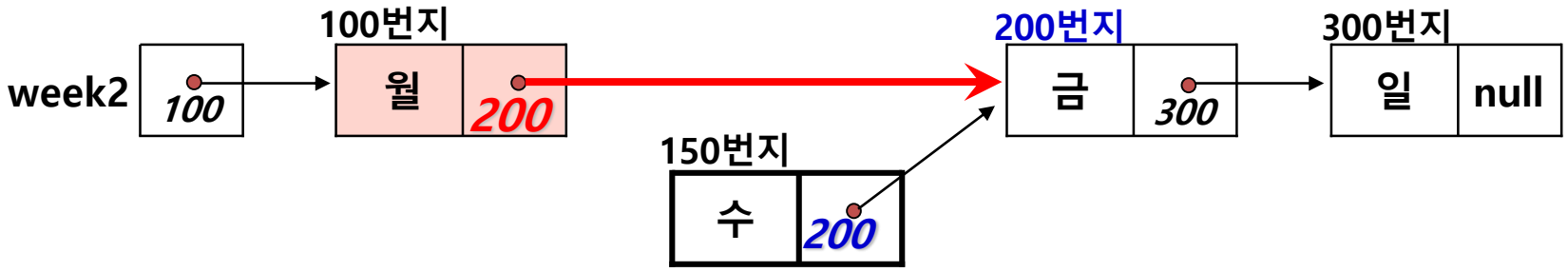
연결 자료구조(단순 연결리스트)

- **단순 연결 리스트의 삭제**
  - 리스트 week2=(월, 수, 금, 일)에서 원소 "수" 삭제하기

① 삭제할 원소의 앞 노드(선행자)를 찾는다.



② 삭제할 원소 "수"의 링크 필드 값을 앞 노드의 링크 필드에 저장한다.

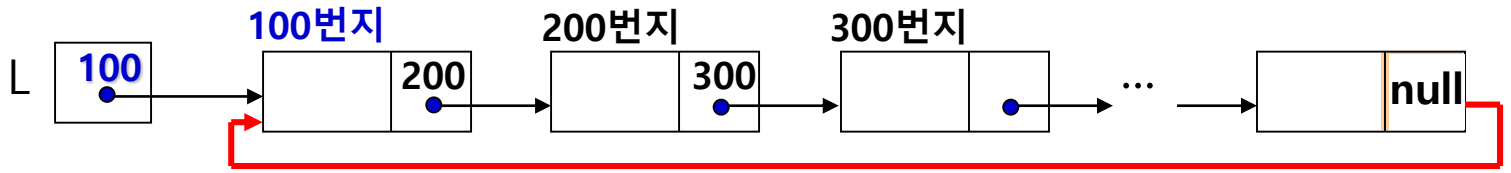


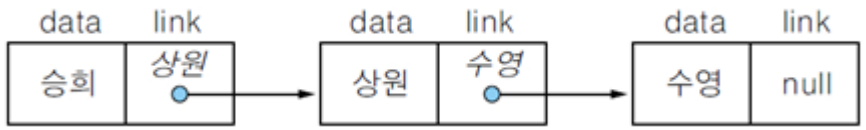
연결 자료구조(단순 연결리스트)

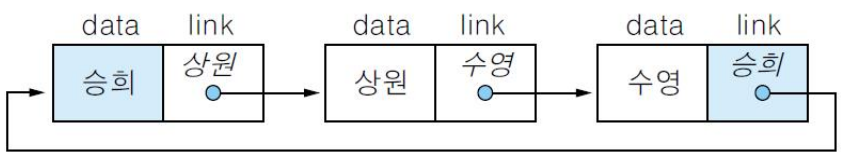
- **실습**
  - Linked list 삽입, 삭제를 구현하시요.
    - 노드 5개 생성 (100, 200, 300, 400, 500)
    - 노드 중 4개 연결리스트로 연결 (100, 200, 400, 500)
    - 노드 300을 연결리스트에 삽입하기
    - 노드 300을 연결리스트에 삭제하기

연결 자료구조(원형 연결리스트)

- **원형 연결 리스트(circular linked list)**
  - 단순 연결 리스트에서 마지막 노드가 리스트의 첫 번째 노드를 가리키게 하여 리스트의 구조를 원형으로 만든 연결 리스트
    - 단순 연결 리스트의 마지막 노드의 링크 필드에 첫 번째 노드의 주소를 저장하여 구성
    - 링크를 따라 계속 순회하면 이전 노드에 접근 가능





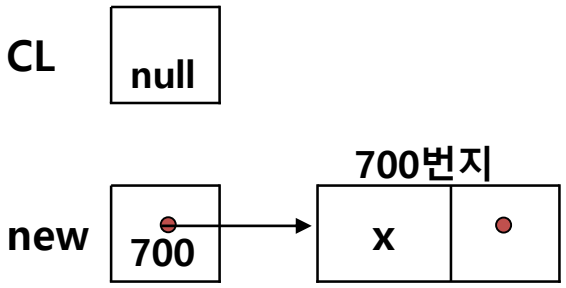


연결 자료구조(원형 연결리스트)

- **원형 연결 리스트의 삽입 연산**
  - 마지막 노드의 링크를 첫 번째 노드로 연결하는 부분만 제외하고는 단순 연결 리스트에서의 삽입 연산과 같은 연산

<< 원형리스트가 공백 리스트인 경우 >>

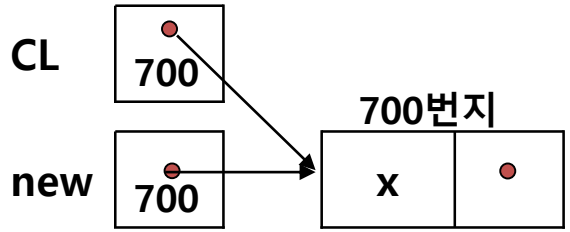
- 삽입하는 노드new는 리스트의 첫 번째 노드이자 마지막 노드가 되어야 함



연결 자료구조(원형 연결리스트)

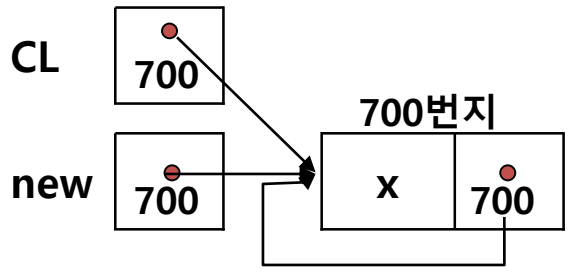
① CL ← new;

- 포인터 CL이 노드 new를 가리키게 한다.



① new.link ← new;

- 노드 new가 자기자신을 가리키게 함으로써 노드 new를 첫 번째 노드이자 마지막 노드가 되도록 지정한다.

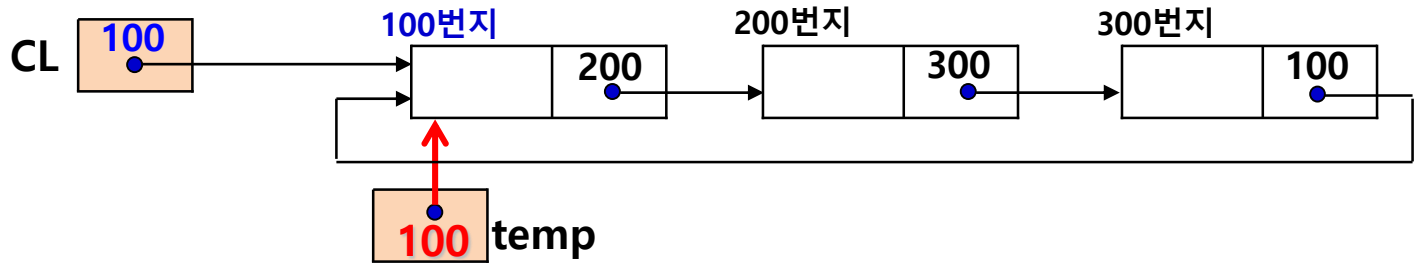


연결 자료구조(원형 연결리스트)

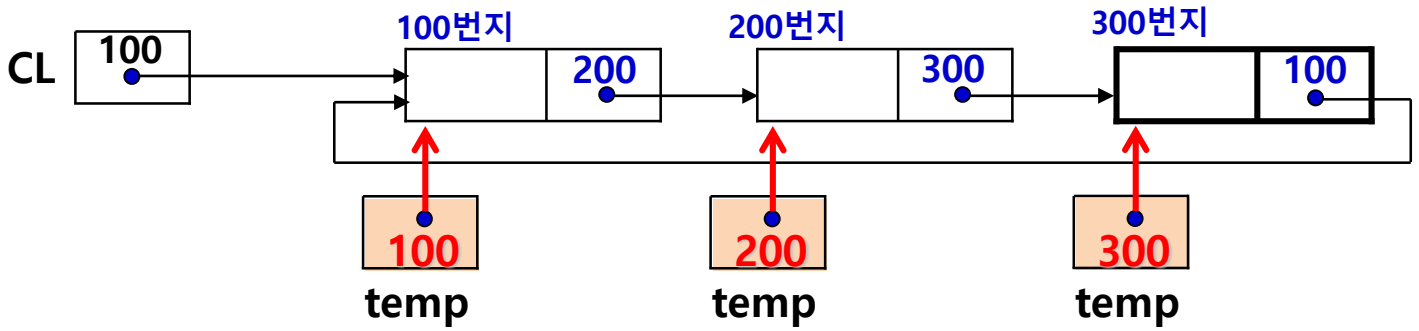
<< 원형리스트가 공백 리스트가 아닌 경우 >>

① temp ← CL;

- 리스트가 공백리스트가 아닌 경우에는 첫 번째 노드의 주소를 임시 순회 포인터 temp에 저장하여 노드 순회의 시작점을 지정한다.

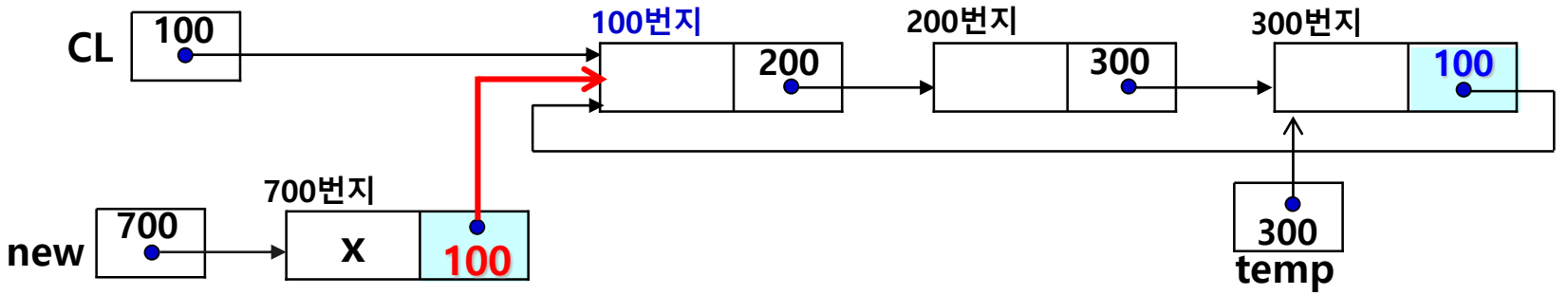


② while 반복문을 수행하여 순회 포인터 temp를 링크를 따라 마지막 노드까지 이동

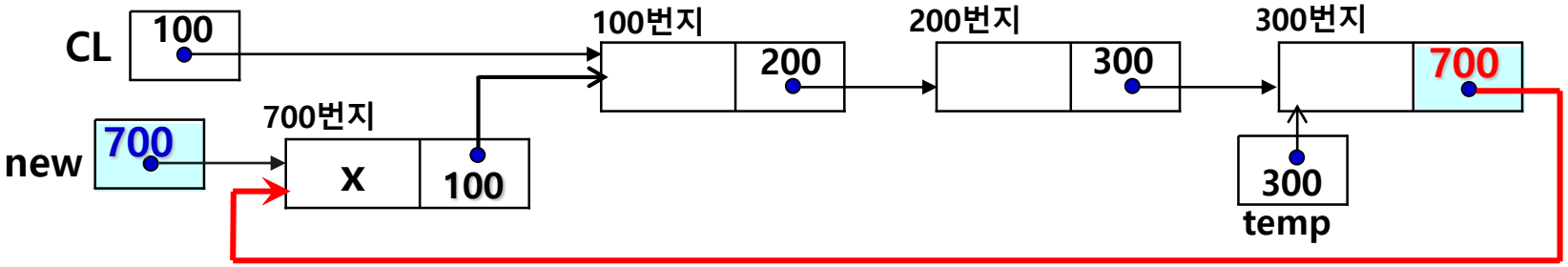


연결 자료구조(원형 연결리스트)

③ **리스트의 마지막 노드의 링크 값을 노드 new의 링크에** 저장하여, 노드 new가 노드 temp의 다음 노드를 가리키게 한다. 리스트 CL은 원형 연결 리스트이므로 마지막 노드의 다음 노드는 리스트의 첫 번째 노드가 된다.

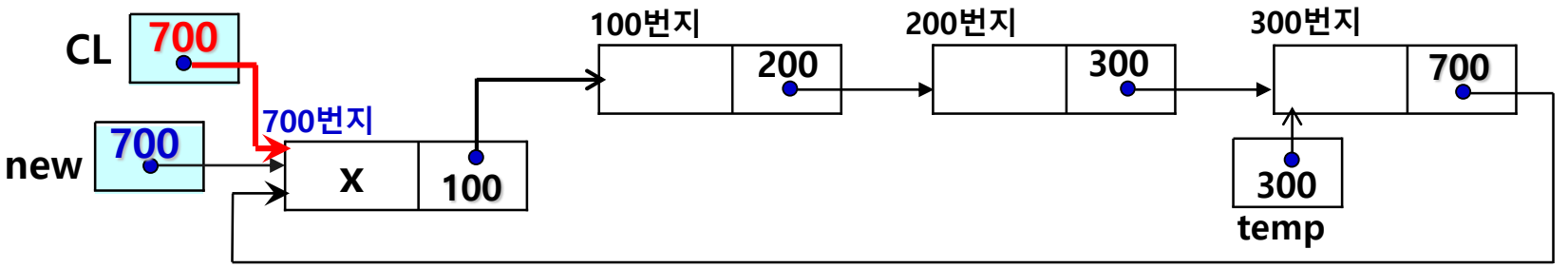


④ **포인터 new의 값을 포인터 temp가 가리키고 있는 마지막 노드의 링크에** 저장하여, 리스트의 마지막 노드가 노드 new를 가리키게 한다.

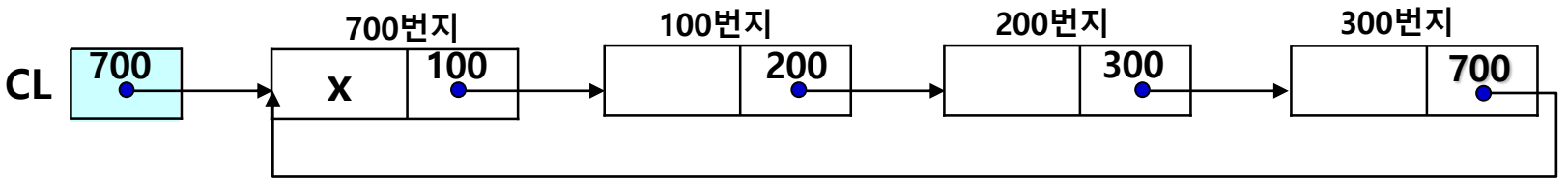


연결 자료구조(원형 연결리스트)

⑤ **노드 new의 값을** 리스트 **포인터 CL에** 저장하여 노드 new가 리스트의 첫 번째 노드가 되도록 지정



- 최종 실행 결과



연결 자료구조(원형 연결리스트)

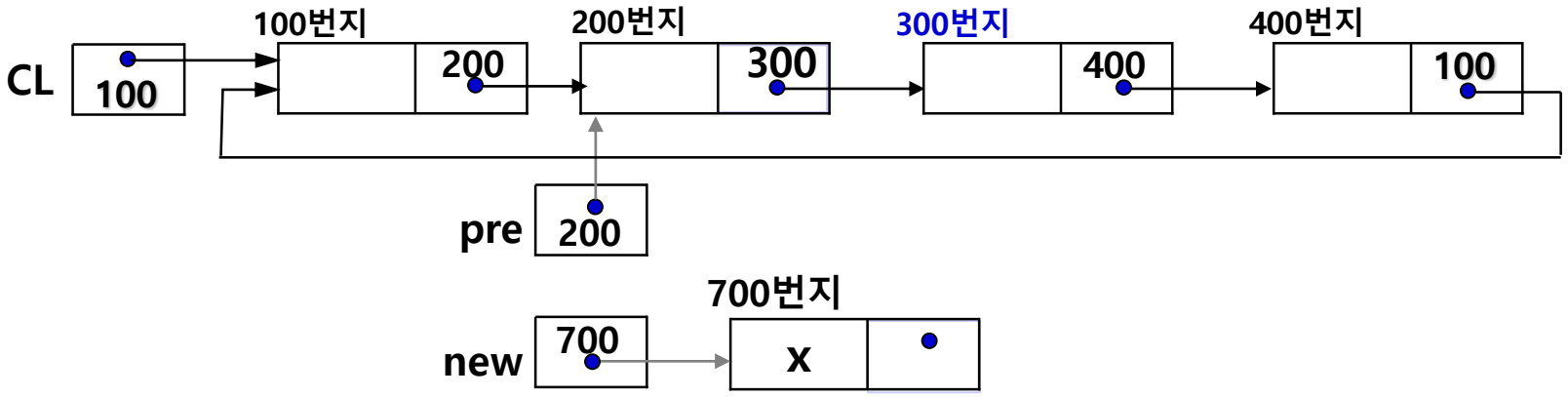
– 중간 노드로 삽입하기

- 원형 연결 리스트 CL에 x 값을 갖는 노드 new를 포인터 pre가 가리키는 노드의 다음 노드로 삽입하는 알고리즘

<< 원형 연결 리스트 CL이 공백 리스트인 경우 >>

- 공백 리스트에 첫번째 노드를 삽입하는 연산과 같음

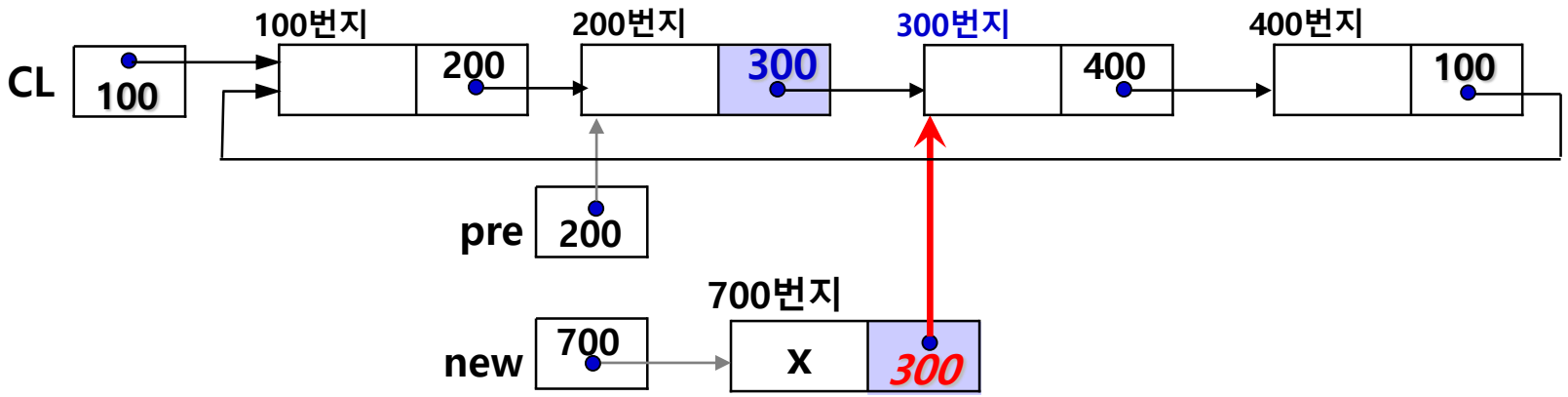
<< 원형 연결 리스트 CL이 공백 리스트가 아닌 경우 >>



연결 자료구조(원형 연결리스트)

① new.link ← pre.link;

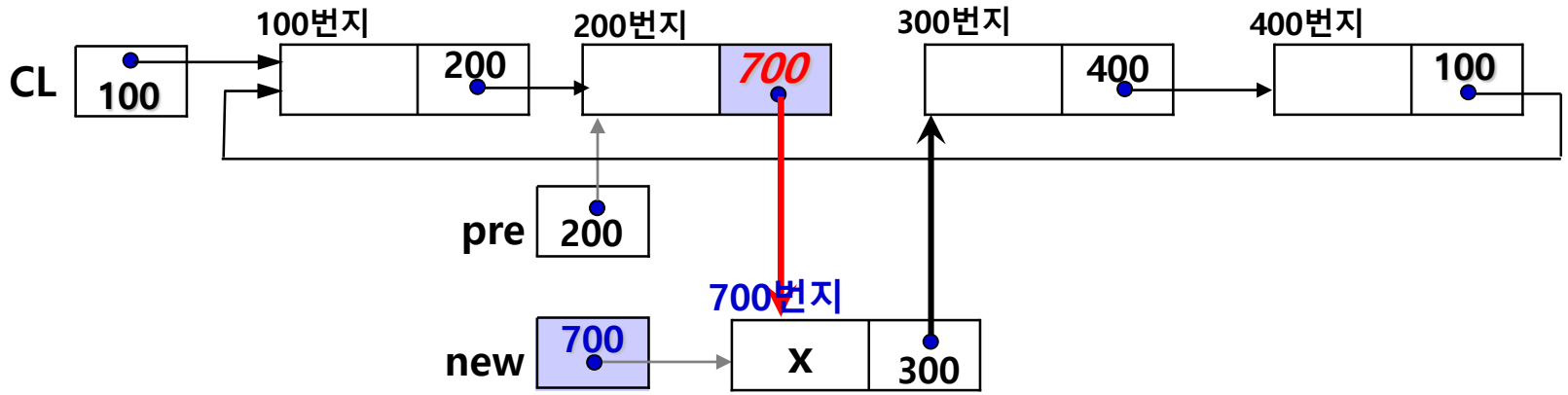
- 노드 pre의 다음 노드로 new를 삽입하기 위해서,
  먼저 **노드 pre의 다음 노드(pre.link)를 new의 다음 노드(new.link)로** 연결



연결 자료구조(원형 연결리스트)

② pre.link ← new;

- **노드 new의 값(삽입할 노드의 주소)을 노드 pre의 링크에** 저장하여, 노드 pre가 노드 new를 가리키도록 한다.



연결 자료구조(원형 연결리스트)

- **실습**
  - Circular linked list 삽입, 삭제를 구현하시요.
    - 노드 5개 생성 (100, 200, 300, 400, 500)
    - 노드 중 4개 원형 연결리스트로 연결 (100, 200, 400, 500)
    - 노드 300을 원형 연결리스트에 삽입하기
    - 노드 300을 원형 연결리스트에 삭제하기

연결 자료구조(이중 연결 연결리스트)

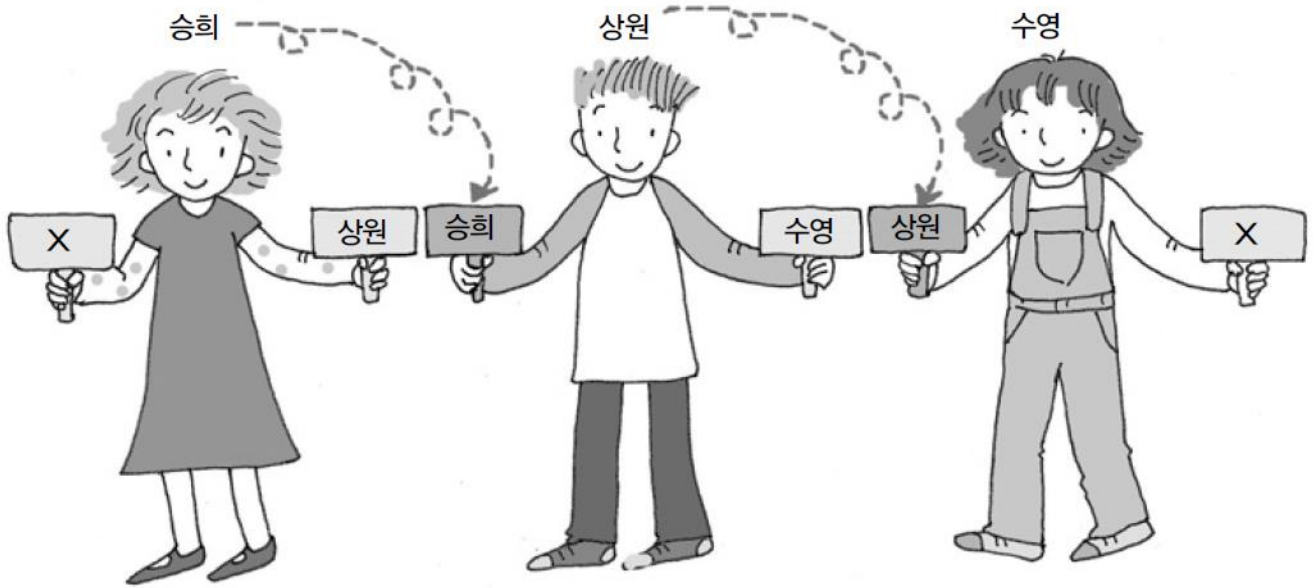
- **이중 연결 리스트(doubly linked list)**
  - 양쪽 방향으로 순회할 수 있도록 노드를 연결한 리스트
  - 이중 연결 리스트의 노드 구조
    - 두 개의 링크 필드와 한 개의 데이터 필드로 구성
    - llink(left link) 필드 : 왼쪽노드와 연결하는 포인터
    - rlink(right link) 필드 : 오른쪽 노드와 연결하는 포인터

- 노드 구조에 대한 class 정의

```
class Node:
    def __init__(self, data, llink=None, rlink=none):
        self.data = data
        self.llink = llink
        self.rlink = rlink
```

연결 자료구조(이중 연결 연결리스트)

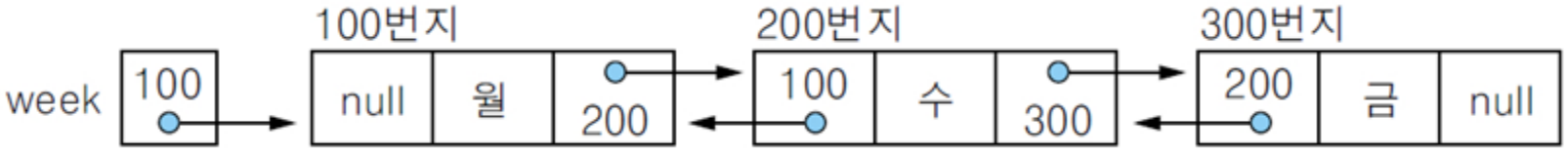
[그림 5-69] 단방향 기차놀이



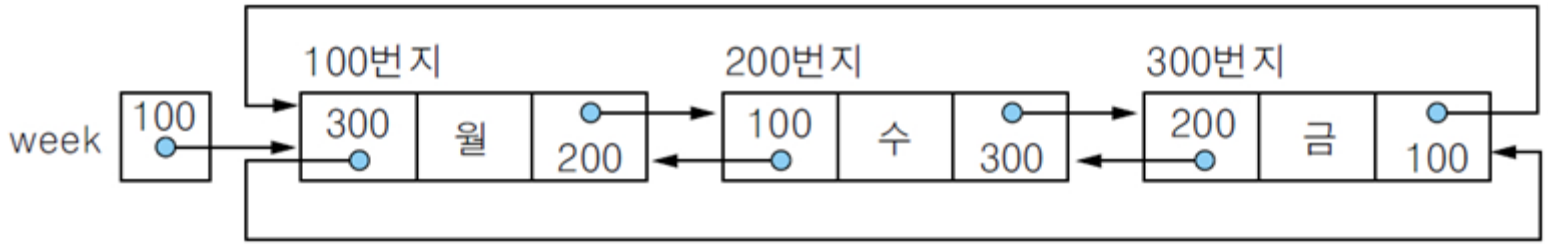
[그림 5-70] 양방향 기차놀이

연결 자료구조(이중 연결 연결리스트)

- 리스트 week=(월, 수, 금)의 이중 연결 리스트 구성



- 원형 이중 연결 리스트
  - 이중 연결 리스트를 원형으로 구성

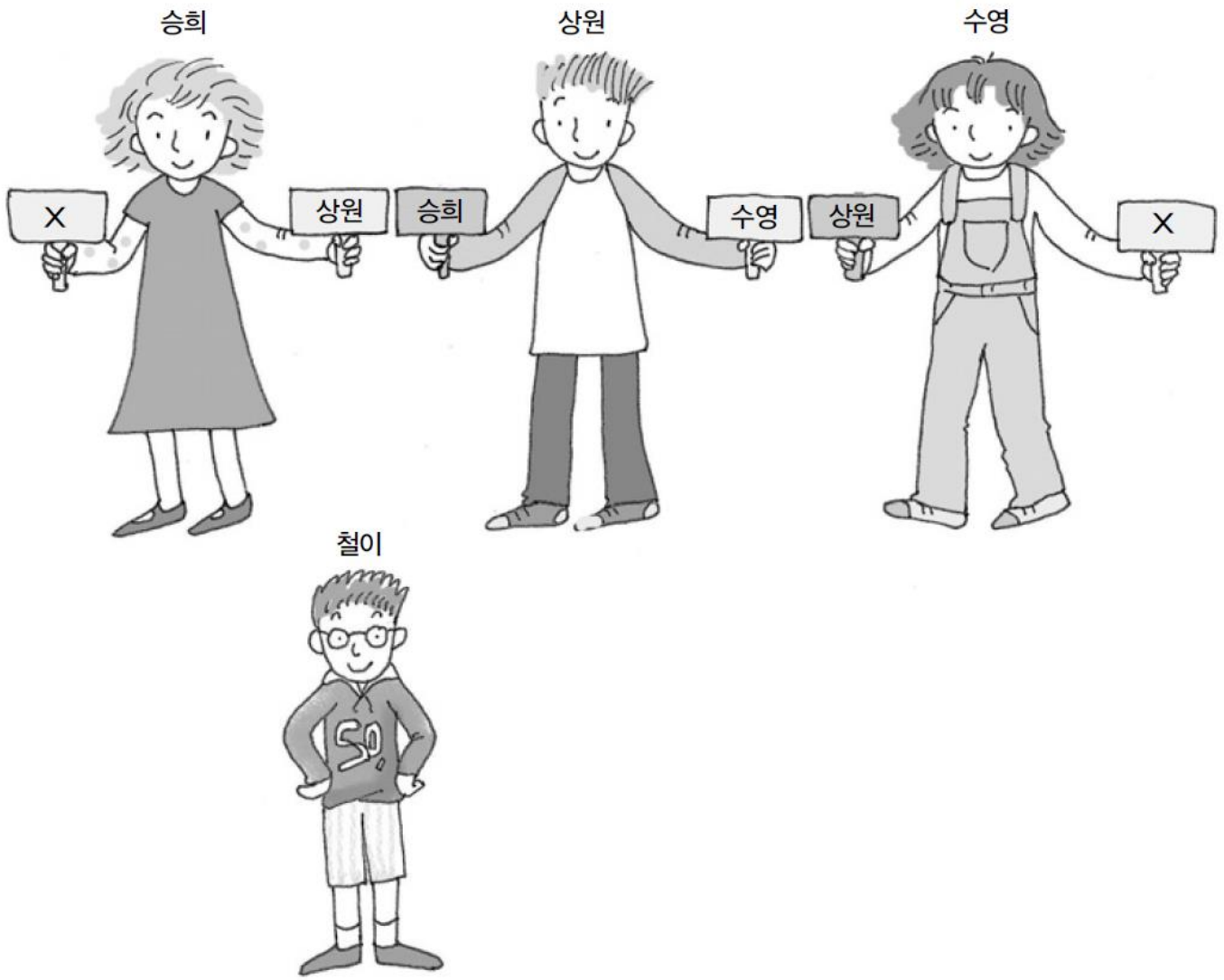


연결 자료구조(이중 연결 연결리스트)

- **이중 연결 리스트에서의 삽입 연산**
  - 이중 연결 리스트에서의 삽입 연산 과정

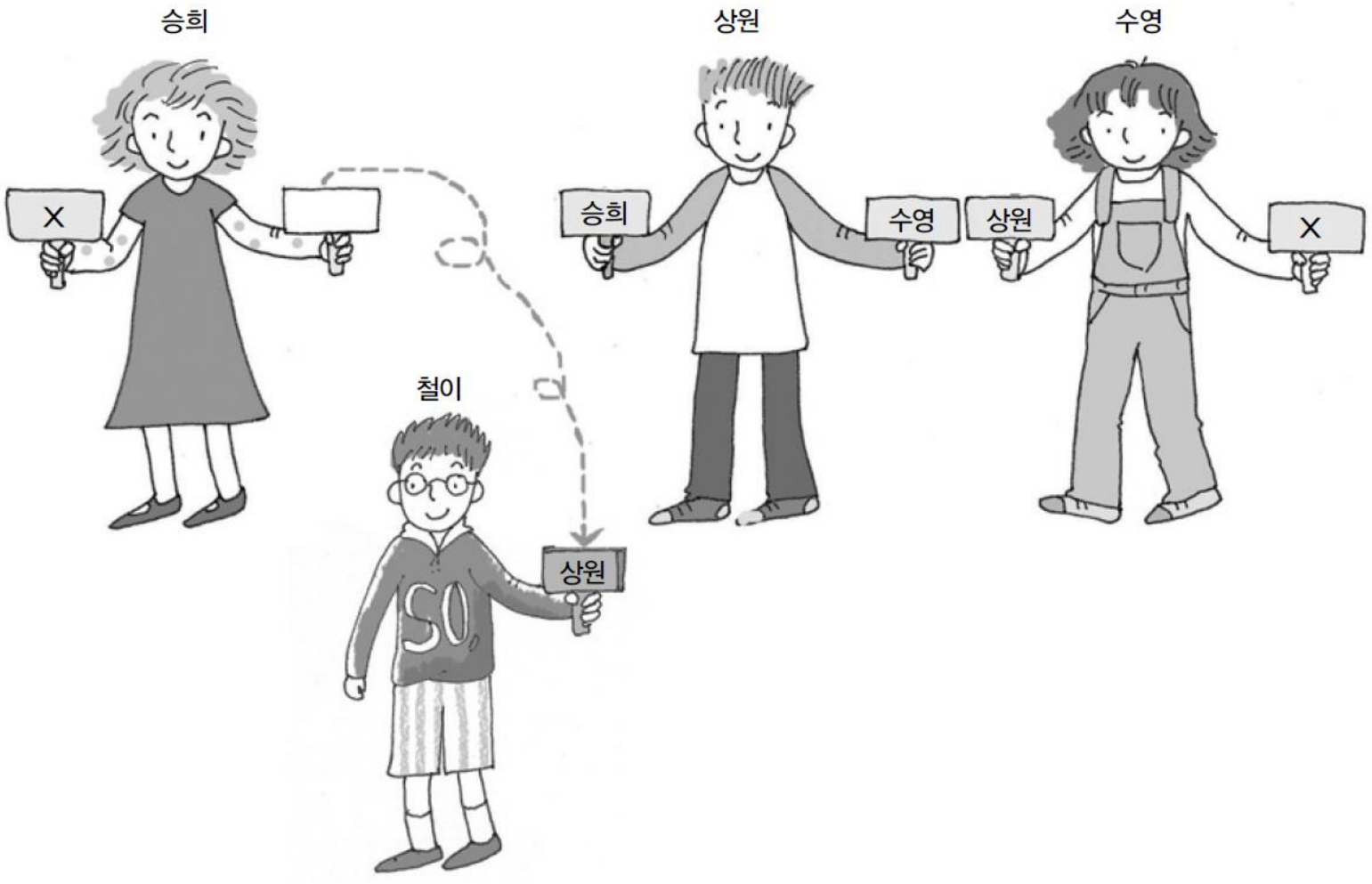
❶ **삽입할 노드를 가져온다.**
❷ 새 노드의 데이터 필드에 값을 저장한다.
❸ **새 노드의 왼쪽 노드(new.llink)의 오른쪽 링크(rlink)를 새 노드의 오른쪽 링크(rlink)에 저장한다.**
❹ 그리고 왼쪽 노드의 오른쪽 링크(rlink)에 새 노드의 주소를 저장한다.
❺ **새 노드의 오른쪽 노드(new.rlink)의 왼쪽 링크(llink)를 새 노드의 왼쪽 링크(llink)에 저장한다.**
❻ 그리고 오른쪽 노드의 왼쪽 링크(llink)에 새 노드의 주소를 저장한다.

연결 자료구조(이중 연결 연결리스트)



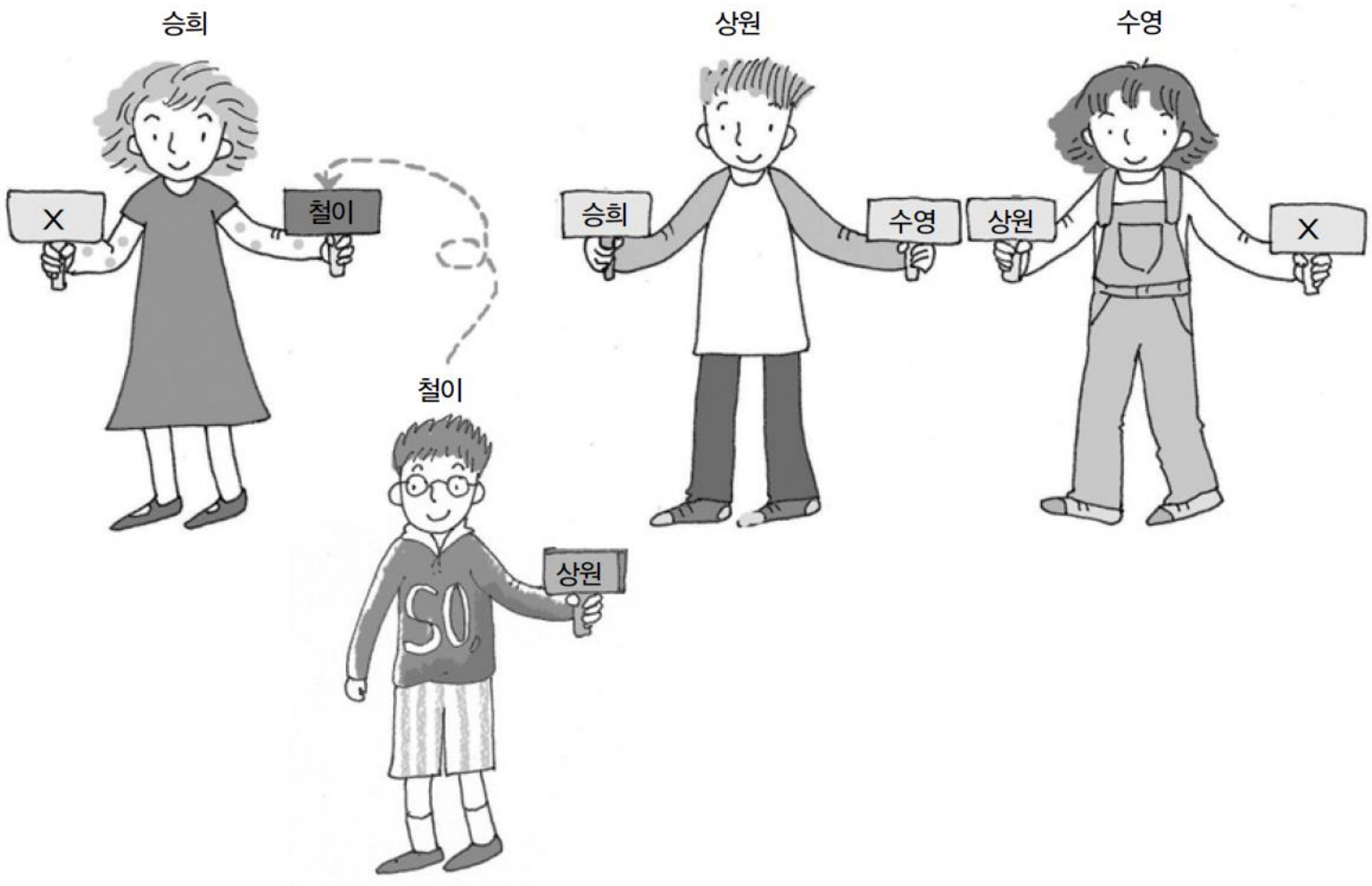
[그림 5-73] 양방향 기차놀이에 사람 추가하기 : 초기 상태

연결 자료구조(이중 연결 연결리스트)



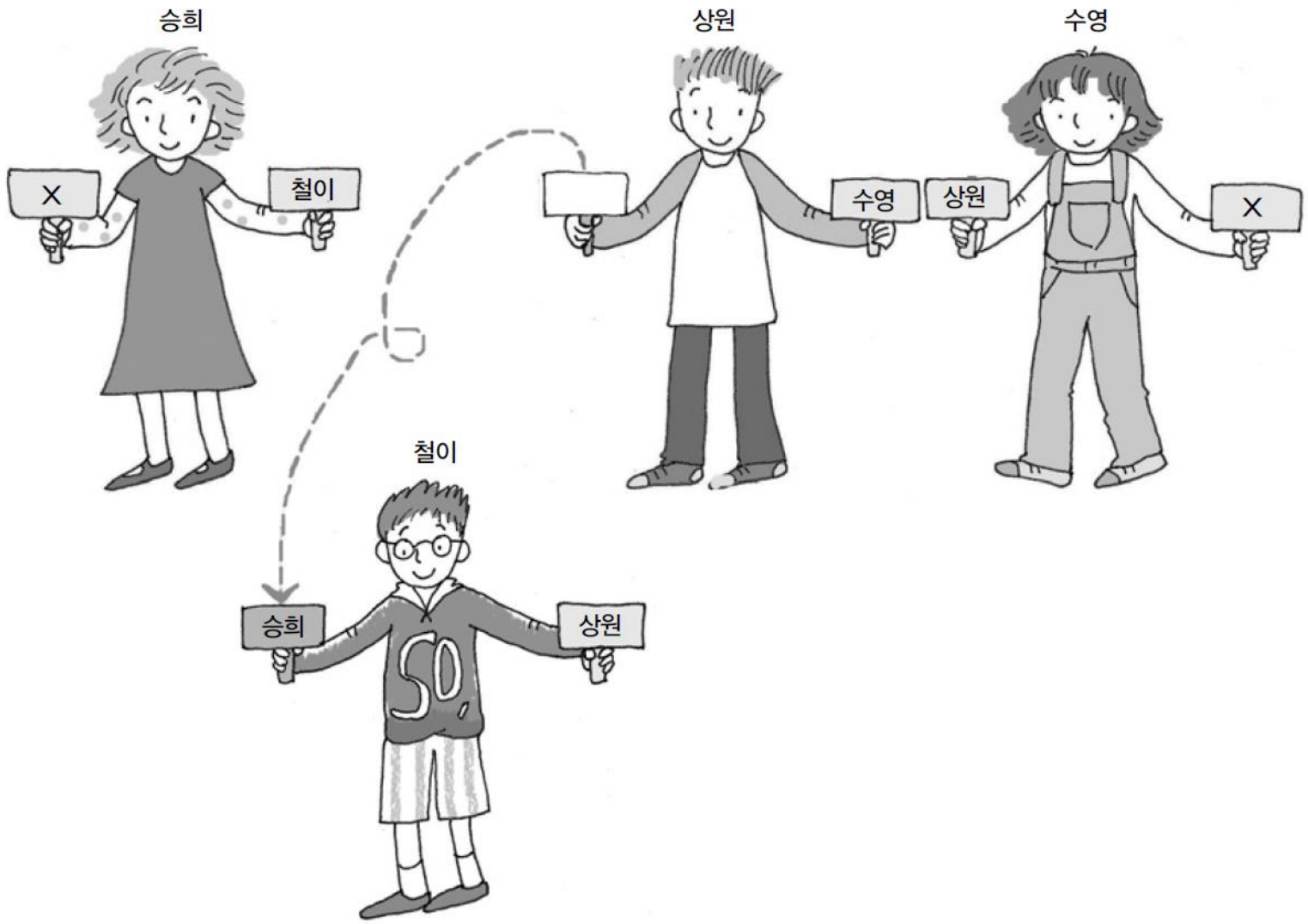
[그림 5-74] 양방향 기차놀이에 사람 추가하기 : 왼쪽 사람의 오른손 이름표 받기

연결 자료구조(이중 연결 연결리스트)



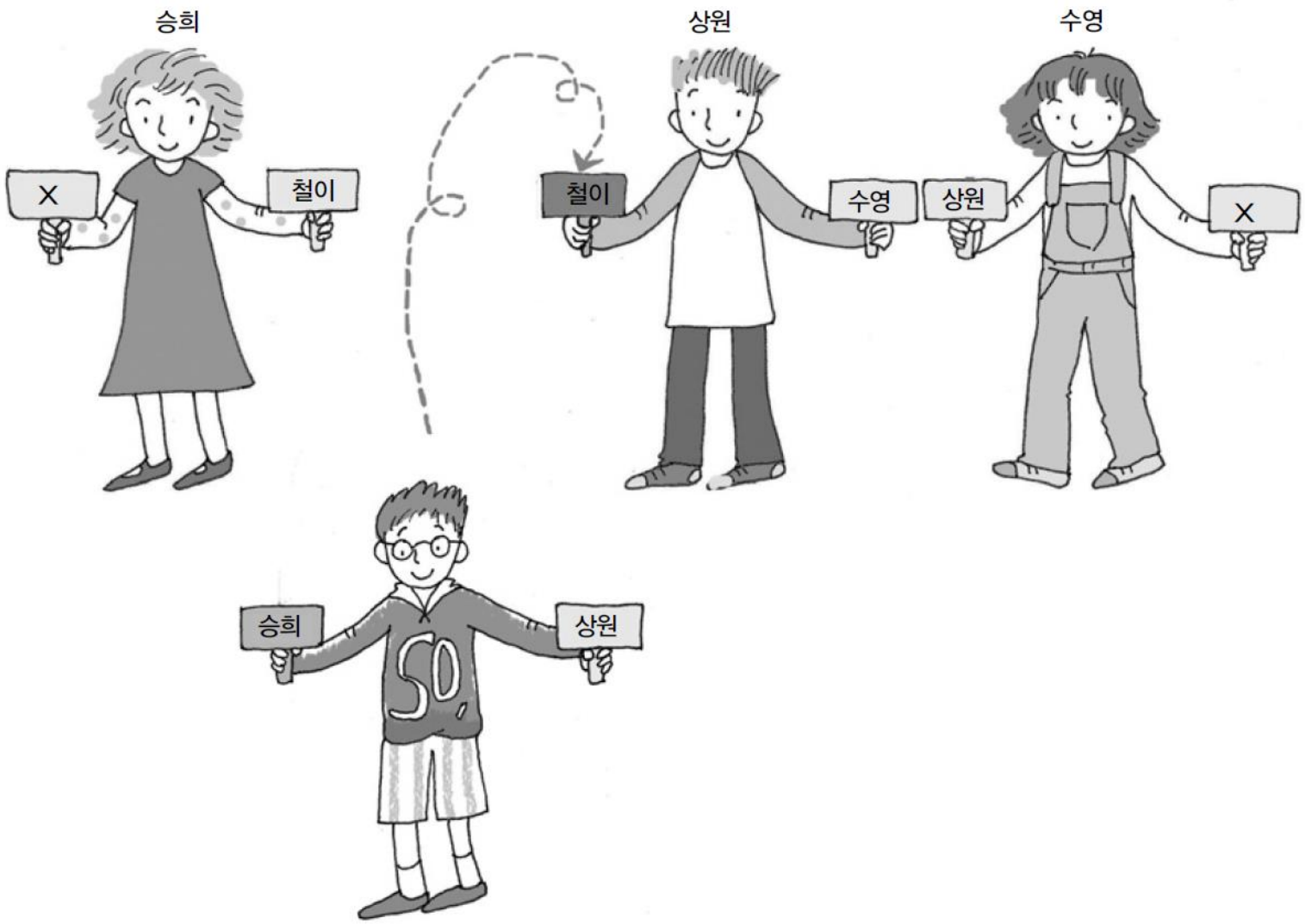
[그림 5-75] 양방향 기차놀이에 사람 추가하기 : 왼쪽 사람에게 자기 이름표 주기

연결 자료구조(이중 연결 연결리스트)



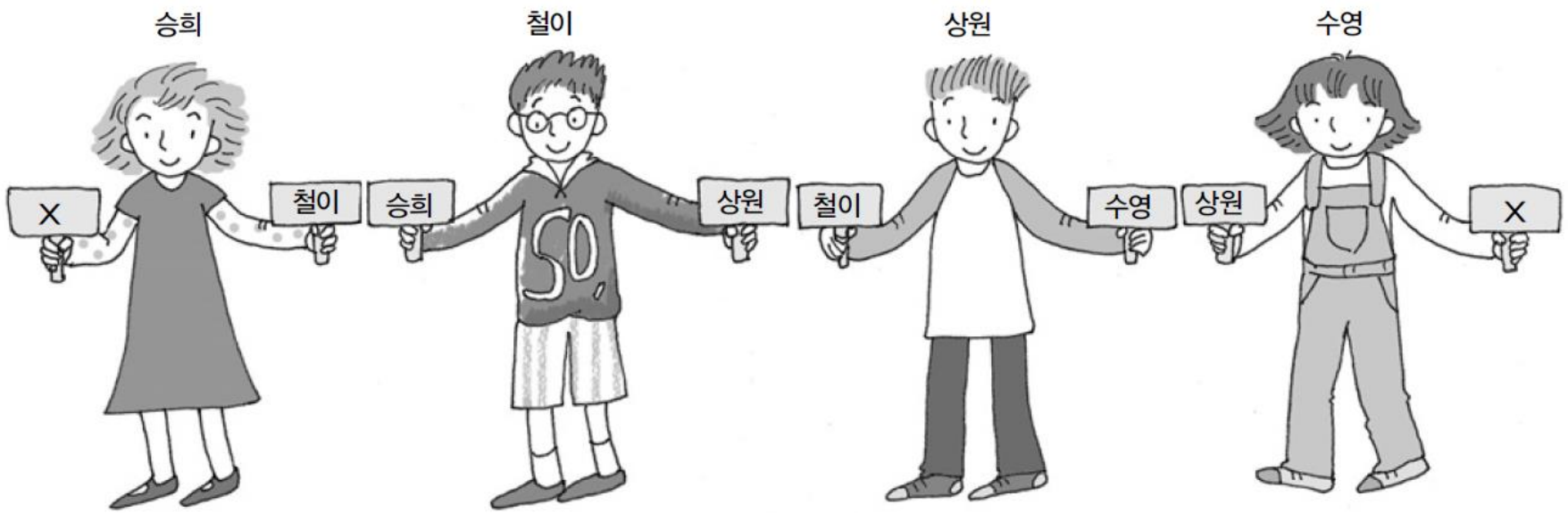
[그림 5-76] 양방향 기차놀이에 사람 추가하기 : 오른쪽 사람의 왼손 이름표 받기

연결 자료구조(이중 연결 연결리스트)



[그림 5-77] 양방향 기차놀이에 사람 추가하기: 오른쪽 사람에게 자기 이름표 주기

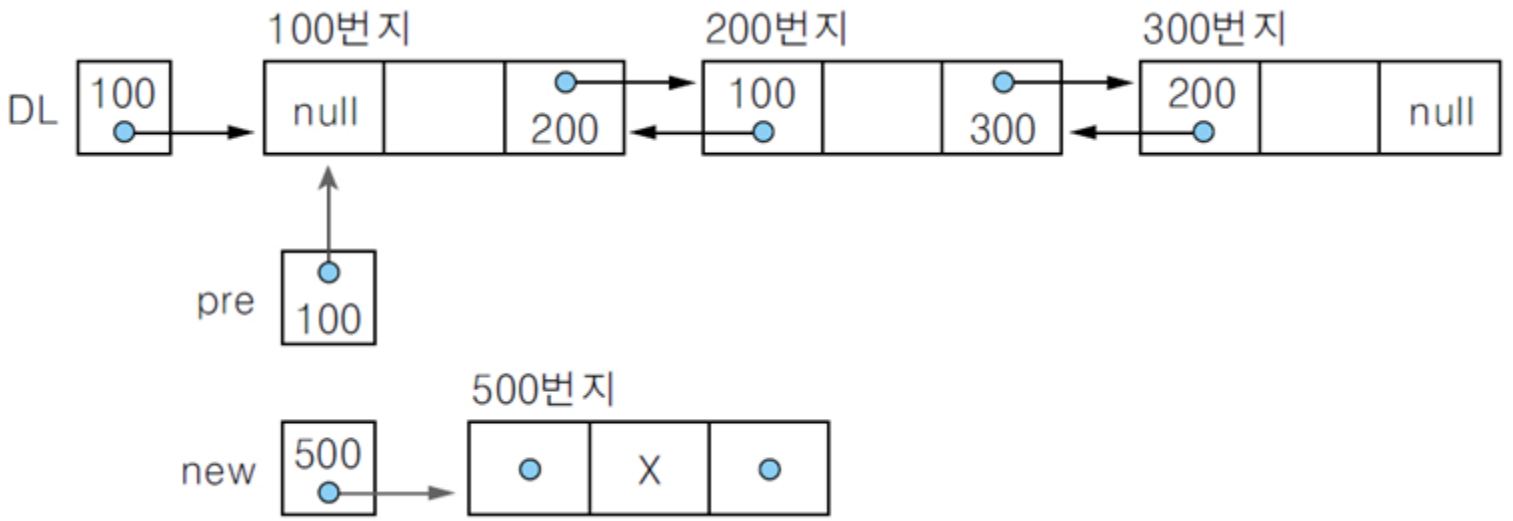
연결 자료구조(이중 연결 연결리스트)



[그림 5-78] 양방향 기차놀이에 사람 추가하기 : 완성

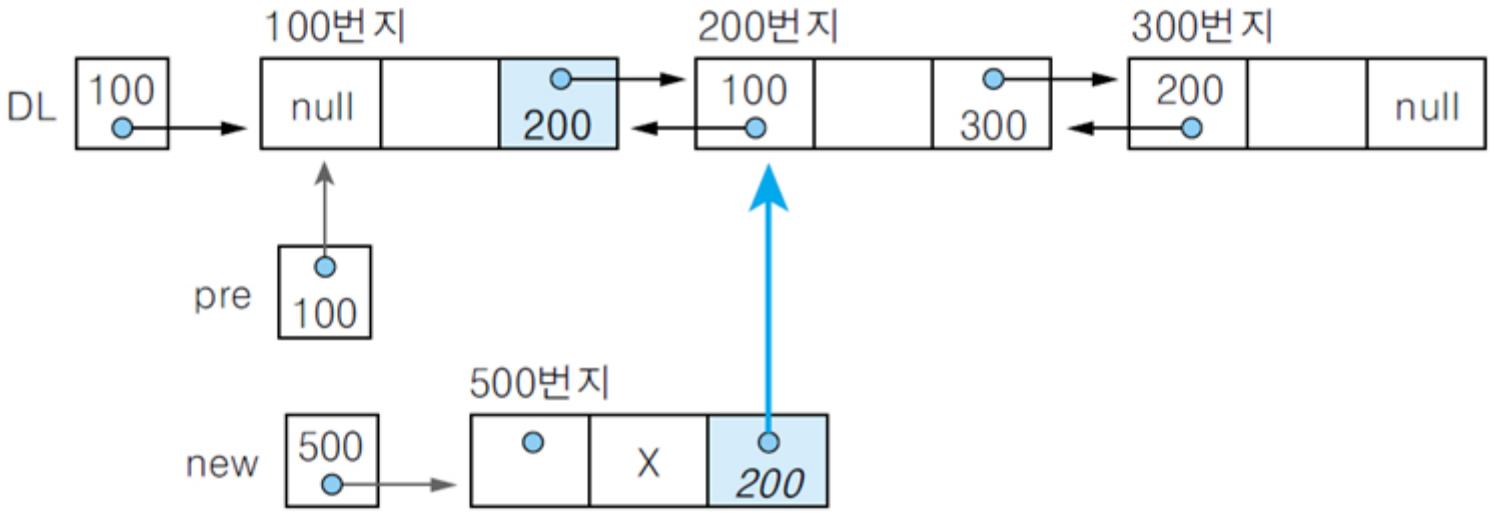
연결 자료구조(이중 연결 연결리스트)

- 이중 연결 리스트 DL에서 포인터 pre가 가리키는 노드의 다음 노드로 노드 new를 삽입하는 과정



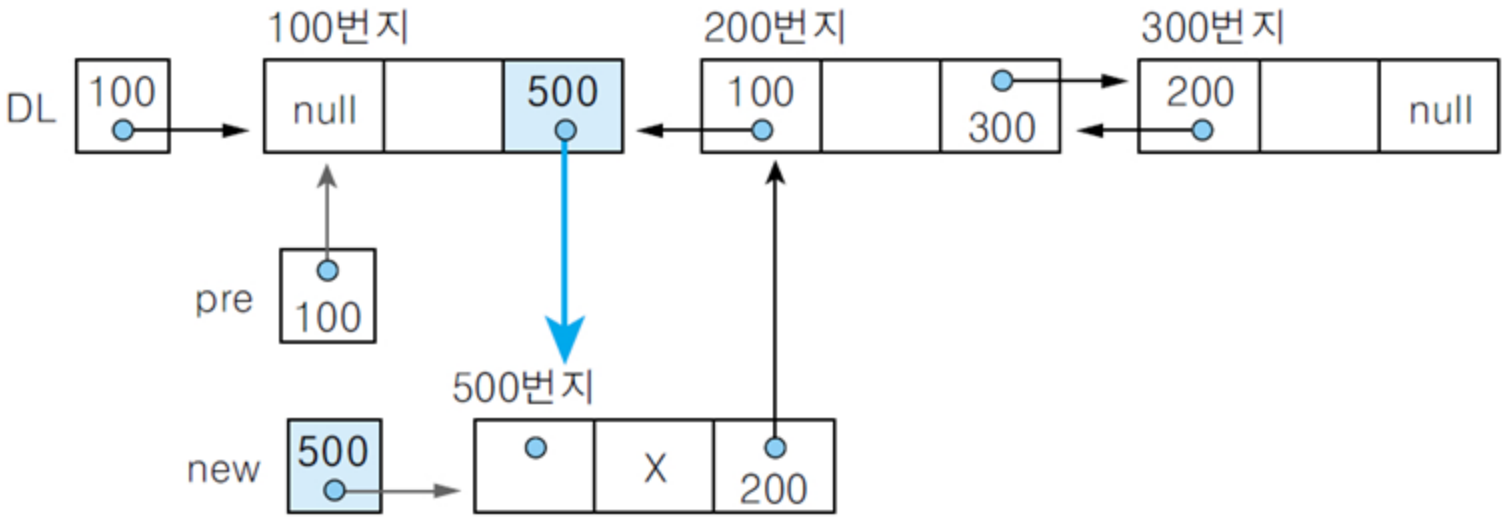
연결 자료구조(이중 연결 연결리스트)

① 노드 pre의 rlink를 노드 new의 rlink에 저장하여, 노드 pre의 오른쪽 노드를 삽입할 노드 new의 오른쪽 노드로 연결



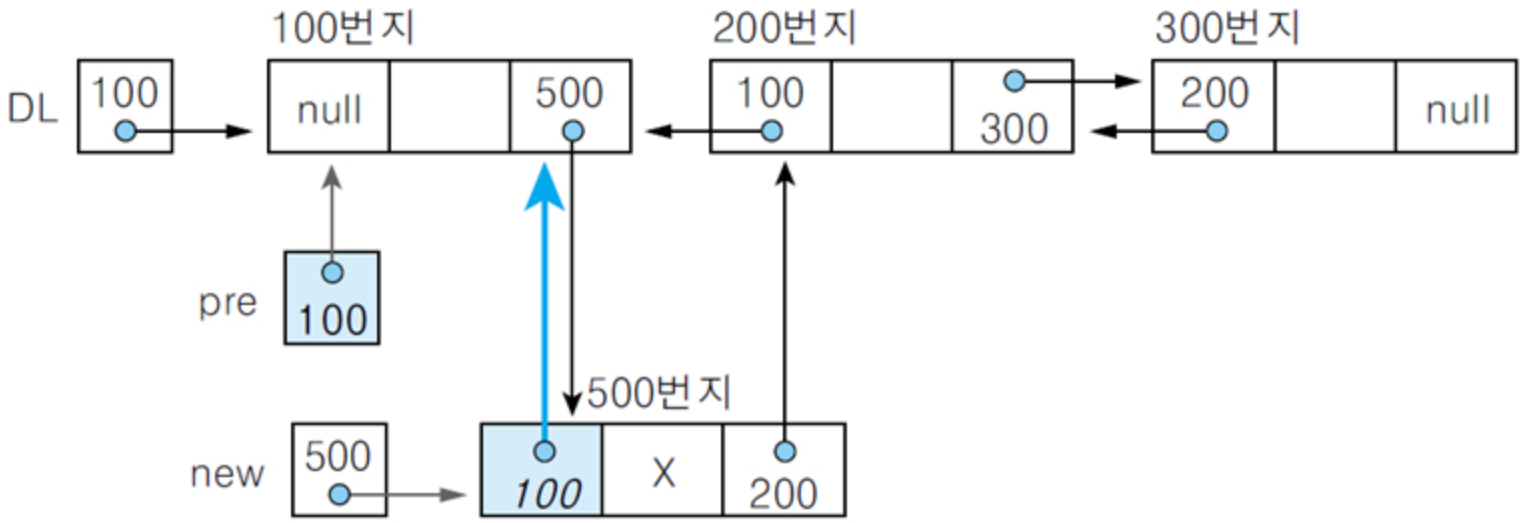
연결 자료구조(이중 연결 연결리스트)

② 새 노드 new의 주소를 노드 pre의 rlink에 저장하여, 노드 new를 노드 pre의 오른쪽 노드로 연결



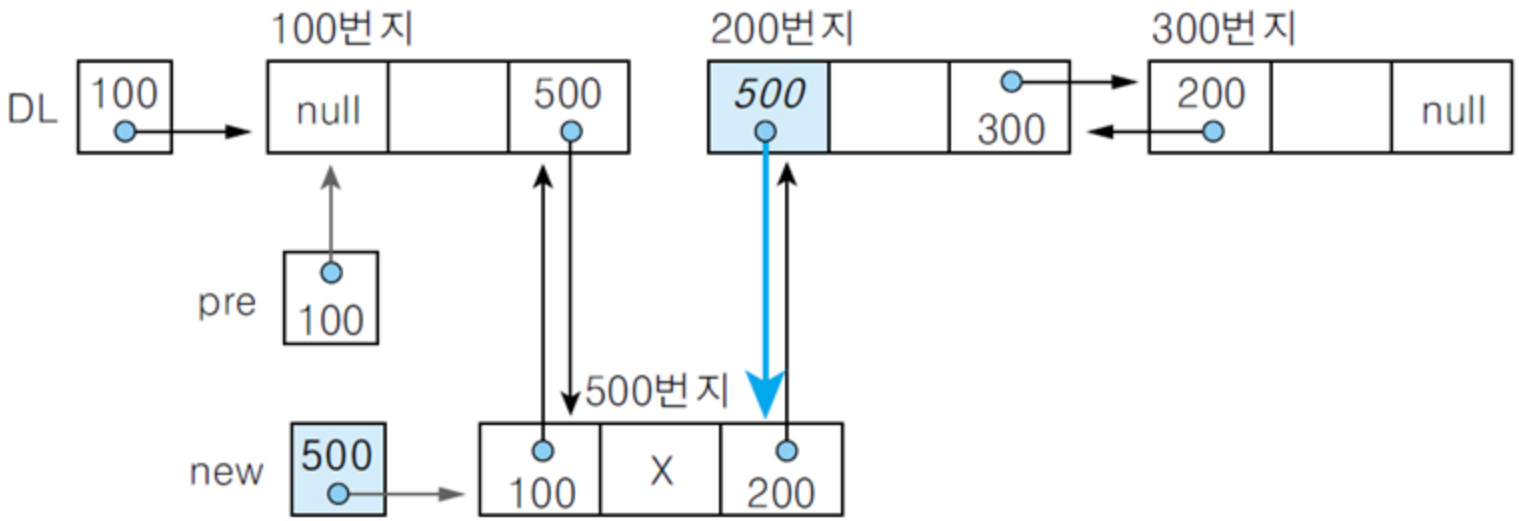
연결 자료구조(이중 연결 연결리스트)

③ 포인터 pre의 값을 삽입할 노드 new의 llink에 저장하여, 노드 pre를 노드 new의 왼쪽노드로 연결



연결 자료구조(이중 연결 연결리스트)

④ 포인터 new의 값을 노드 new의 오른쪽노드(new.rlink)의 llink에 저장하여, 노드 new의 오른쪽노드의 왼쪽노드로 노드 new를 연결



연결 자료구조(이중 연결 연결리스트)

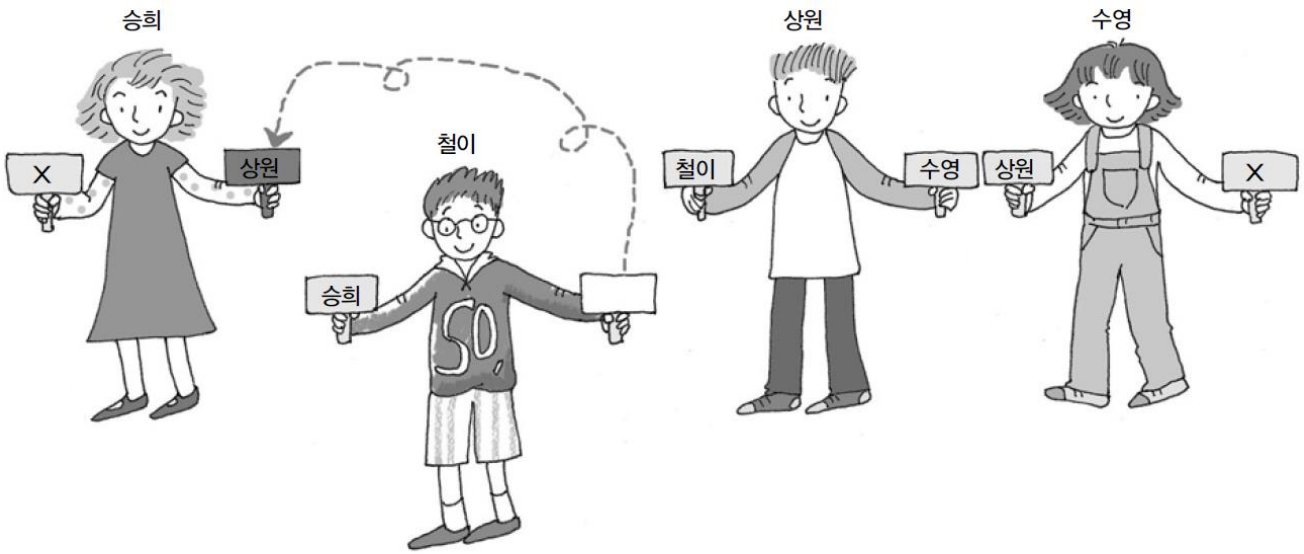
- **이중 연결 리스트에서의 삭제 연산**
  - 이중 연결 리스트에서의 삭제 연산 과정

❶ 삭제할 노드의 오른쪽 노드의 주소(old.rlink)를
삭제할 노드의 왼쪽 노드(old.llink)의 오른쪽 링크(rlink)에 저장한다.

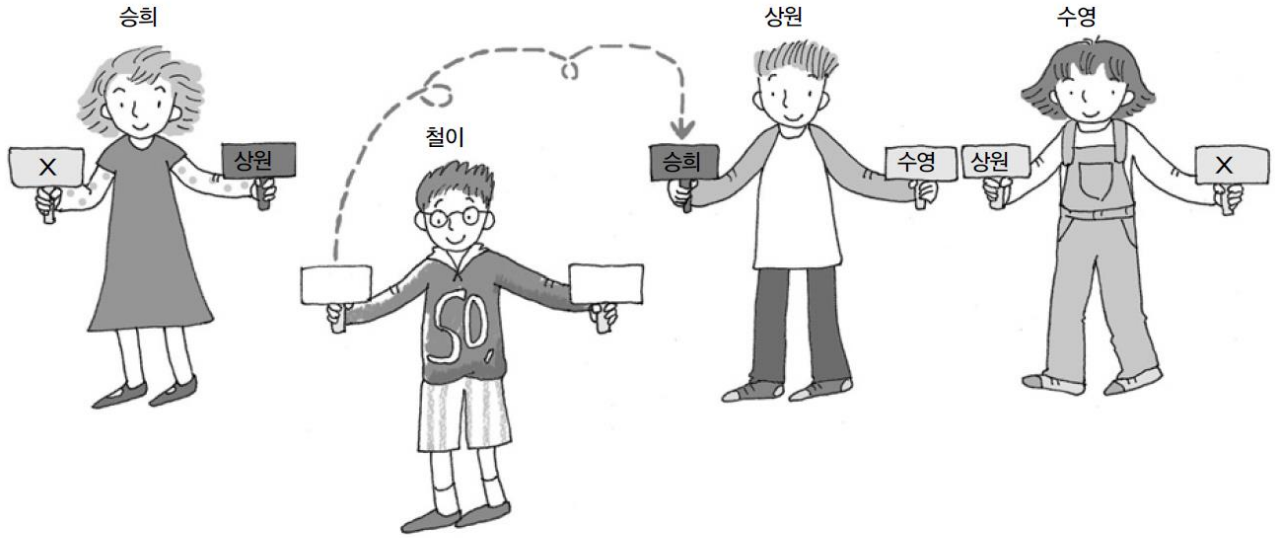
❷ 삭제할 노드의 왼쪽 노드의 주소(old.llink)를
삭제할 노드의 오른쪽 노드(old.rlink)의 왼쪽 링크(llink)에 저장한다.

❸ 삭제한 노드를 자유공간리스트에 반환한다.

연결 자료구조(이중 연결 연결리스트)

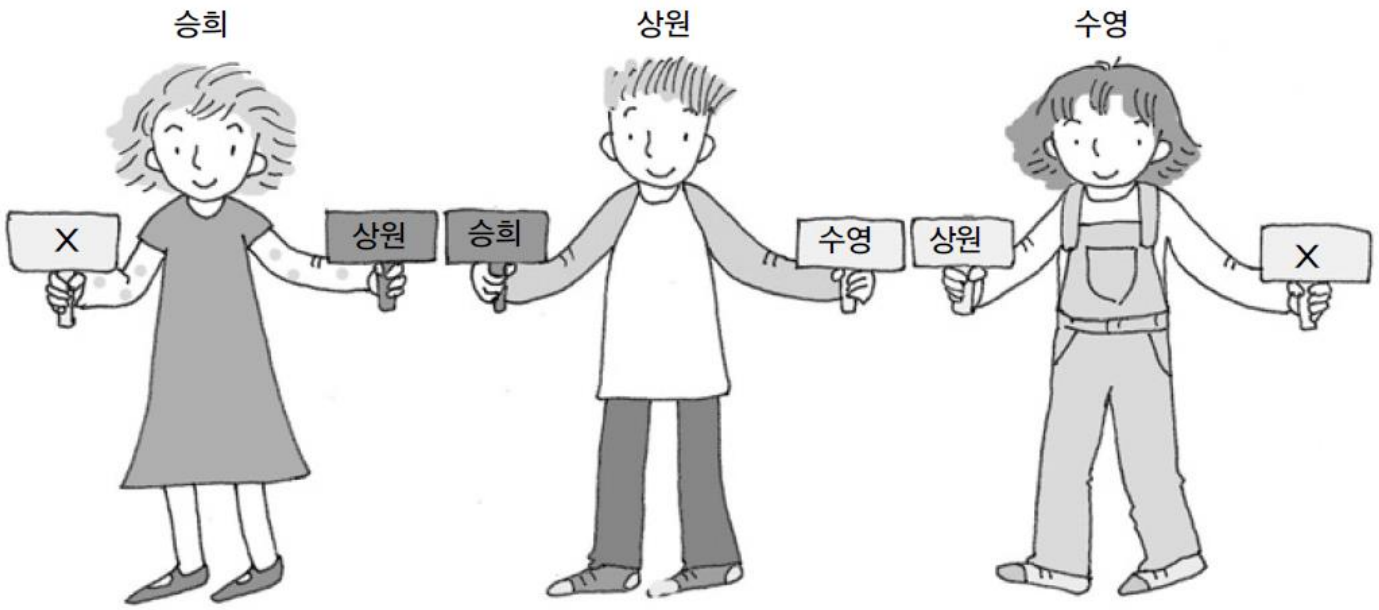


[그림 5-84] 양방향 기차놀이에서 사람 나가기 : 오른손 이름표 넘겨주기



[그림 5-85] 양방향 기차놀이에서 사람 나가기 : 왼손 이름표 넘겨주기

연결 자료구조(이중 연결 연결리스트)

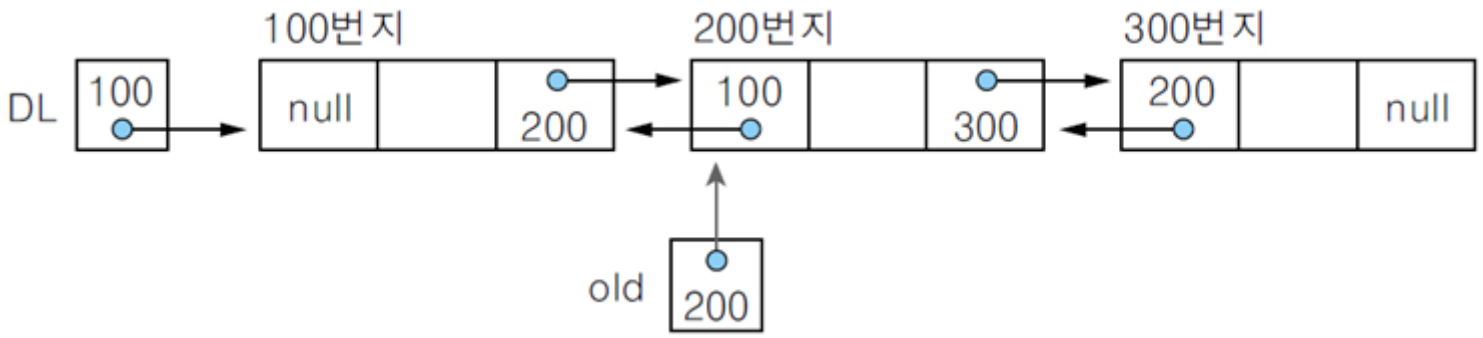


[그림 5-86] 양방향 기차놀이에서 사람 나가기 : 완성

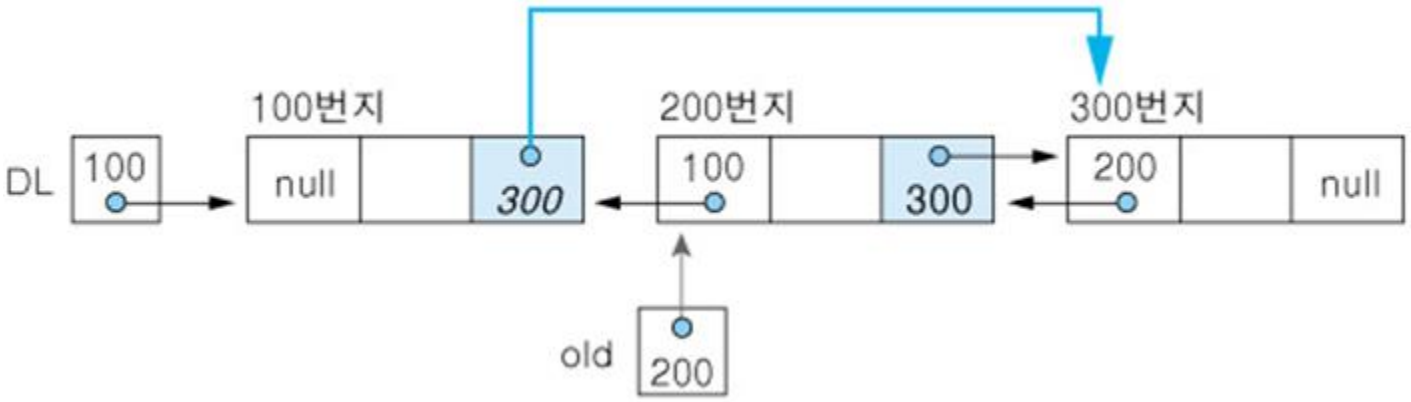
연결 자료구조(이중 연결 연결리스트)

– 이중 연결 리스트 DL에서 포인터 old가 가리키는 노드를 삭제하는 과정

- 초기 상태

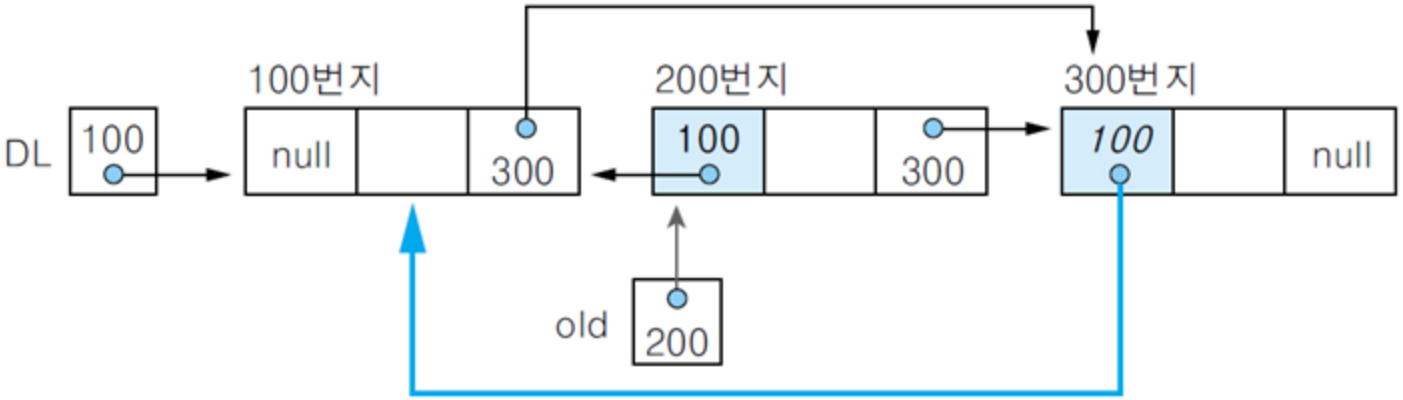


① 삭제할 노드 old의 오른쪽노드의 주소를 노드 old의 왼쪽노드의 rlink에 저장하여, 노드 old의 오른쪽노드를 노드 old의 왼쪽노드의 오른쪽노드로 연결

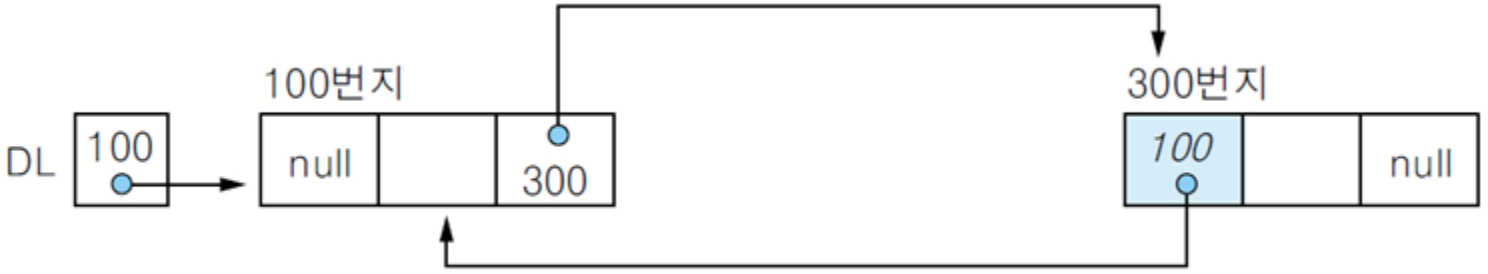


연결 자료구조(이중 연결 연결리스트)

② 삭제할 노드 old의 왼쪽노드의 주소를 노드 old의 오른쪽노드의 llink에 저장하여, 노드 old의 왼쪽노드를 노드 old의 오른쪽노드로 연결



③ 삭제된 노드 old는 자유공간리스트에 반환



연결 자료구조(이중 원형 연결리스트)

- **실습(과제)**
  - Double linked list 삽입, 삭제를 구현하시요.
    - 노드 5개 생성 (100, 200, 300, 400, 500)
    - 노드 중 4개 원형 연결리스트로 연결 (100, 200, 400, 500)
    - 노드 300을 원형 연결리스트에 삽입하기
    - 노드 300을 원형 연결리스트에 삭제하기