

3 연결 자료구조와 연결 리스트

1. 연결 자료구조와 연결 리스트의 이해

❖ 순차 자료구조의 문제점

- 삽입 연산이나 삭제 연산 후에 연속적인 물리 주소를 유지하기 위해서 원소들을 이동시키는 추가 작업과 시간 소요
 - 원소들의 이동 작업으로 인한 오버헤드로 원소의 개수가 많고 삽입·삭제 연산이 많이 발생하는 경우에 성능상의 문제 발생
- 순차 자료구조는 배열을 이용해 구현하기 때문에 배열이 갖고 있는 메모리 사용의 비효율성 문제를 그대로 가짐
- 순차 자료구조에서의 연산 시간에 대한 문제와 저장 공간에 대한 문제를 개선한 자료 표현 방법 필요



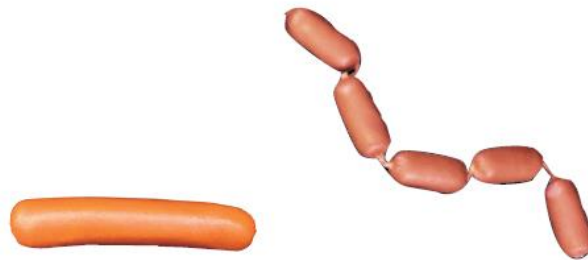
1. 연결 자료구조와 연결 리스트의 이해

❖ 연결 자료구조 Linked Data Structure

- 자료의 논리적인 순서와 물리적인 순서가 불일치
 - 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
 - 리스트의 연결 자료구조로 표현
 - 연결하는 방식에 따라 단순 연결 리스트와 원형 연결 리스트, 이중 연결 리스트, 이중 원형 연결 리스트



(a) 이불과 퀼트 이불



(b) 소시지와 줄줄이 소시지

그림 4-1 순차 자료구조와 연결 자료구조 예



1. 연결 자료구조와 연결 리스트의 이해

❖ 연결 리스트의 노드

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



그림 4-2 노드의 논리적 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 주소값을 저장



1. 연결 자료구조와 연결 리스트의 이해

❖ 순차 자료구조와 연결 자료구조의 비교

표 4-1 순차 자료구조와 연결 자료구조의 비교

구분	순차 자료구조	연결 자료구조
메모리 저장 방식	필요한 전체 메모리 크기를 계산하여 할당하고, 할당된 메모리의 시작 위치부터 빈자리 없이 자료를 순서대로 연속하여 저장한다.	노드 단위로 메모리가 할당되며, 저장 위치의 순서와 상관없이 노드의 링크 필드에 다음 자료의 주소를 저장한다.
연산 특징	삽입·삭제 연산 후에도 빈자리 없이 자료가 순서대로 연속 저장되어, 변경된 논리적인 순서와 저장된 물리적인 순서가 일치한다.	삽입·삭제 연산 후 논리적인 순서가 변경되어도 링크 정보만 변경되고 물리적 위치는 변경되지 않는다.
프로그램 기법	배열을 이용한 구현	포인터를 이용한 구현

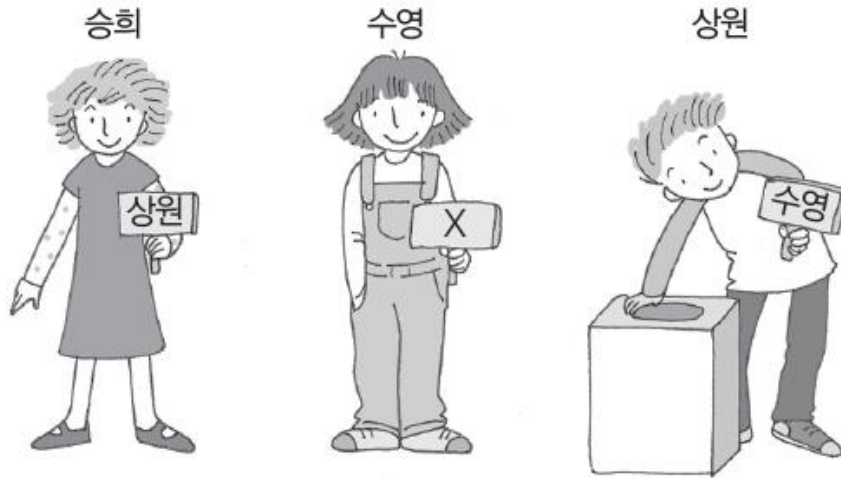


1. 연결 자료구조와 연결 리스트의 이해

❖ 연결 리스트의 이해

■ 기차놀이

① 이름표 뽑기



② 뽑은 사람을 찾아 연결 : 승희 → 상원 → 수영



그림 4-3 기차놀이 : 이름표를 뽑아 이름표대로 기차 연결하기



1. 연결 자료구조와 연결 리스트의 이해

❖ 연결 리스트의 이해

- 기차놀이와 연결 리스트
 - 기차놀이 하는 아이들 : 연결 리스트의 노드
 - 이름표 : 노드의 link 필드

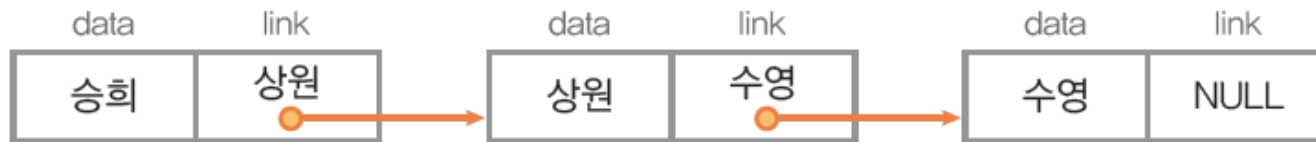


그림 4-4 기차놀이의 노드 표현



1. 연결 자료구조와 연결 리스트의 이해

- 선형 리스트 week의 순차 리스트 표현
 - 리스트 week=(월, 화, 수, 목, 금, 토, 일)

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
week	월	화	수	목	금	토	일

week [0]	월
[1]	화
[2]	수
[3]	목
[4]	금
[5]	토
[6]	일

(a) 논리적 구조

(b) 물리적 구조

그림 4-5 선형 리스트 week의 순차 리스트 표현

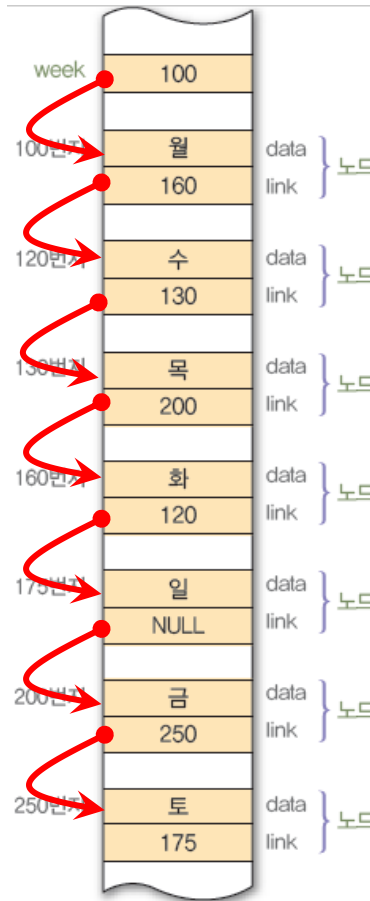


1. 연결 자료구조와 연결 리스트의 이해

- 선형 리스트 week의 연결 리스트 표현



(a) 논리적 구조



(b) 물리적 구조

그림 4-6 선형 리스트 week의 연결 리스트 표현



1. 연결 자료구조와 연결 리스트의 이해

■ 선형 리스트와 연결 리스트의 비교

- **리스트 이름** week : 연결 리스트의 시작을 가리키는 포인터 변수
 - week는 연결 리스트의 첫 번째 노드를 가리킴과 동시에 연결된 리스트 전체 의미
- 연결 리스트의 마지막 노드의 링크 필드 : 노드 끝을 표시하기 위해 **NULL**(널) 저장
- **공백 연결 리스트** : 포인터 변수 week에 NULL 저장(널 포인터) week NULL
- 각 노드의 필드에 저장한 값은 **점 연산자**를 사용해 액세스
 - week.data : 포인터 week가 가리키는 노드 데이터 필드 값 "월"
 - week.link : 포인터 week가 가리키는 노드 링크 필드에 저장된 주소값 "120"

그림 4-7 공백 연결 리스트



그림 4-8 선형 리스트 week의 연결 리스트 구조



1. 연결 자료구조와 연결 리스트의 이해

- 리스트 week의 노드에 대한 구조체 정의

```
typedef struct Node {  
    char data[4];  
    struct Node* link;  
};
```

그림 4-9 리스트 week 노드의 구조체



1. 연결 자료구조와 연결 리스트의 이해

- 연결 리스트의 노드와 점 연산자 표현
 - (a)의 대화에 담긴 진호와 상원의 관계를
(b)의 연결 리스트의 노드로 표현하고,
(c)와 같이 점 연산자(.)를 사용하여 액세스



(c) 진호.link.link.link.link.link.link.link.data

그림 4-10 연결 리스트의 노드와 점 연산자 표현



2. 단순 연결 리스트

❖ 단순 연결 리스트 `singly linked list`의 개념

- 노드가 하나의 링크 필드에 의해서 다음 노드와 연결되는 구조를 가짐
- 연결 리스트, 선형 연결 리스트 `linear linked list`,
단순 연결 선형 리스트 `singly linked linear list`



2. 단순 연결 리스트 : 삽입 연산

❖ 단순 연결 리스트에서의 삽입 연산

0 기차놀이에 철이를 끼우기 전 상태



1 앞사람에게 이름표 받아 오기



2 앞사람에게 자기 이름표 주기



3 이름표대로 연결하기



2. 단순 연결 리스트 : 삽입 연산

■ 단순 연결 리스트에 삽입하는 방법

- ① 삽입할 노드를 준비한다.
- ② 새 노드의 데이터 필드에 값을 저장한다.
- ③ 새 노드의 링크값을 지정한다.
- ④ 리스트의 앞 노드에 새 노드를 연결한다.



2. 단순 연결 리스트 : 삽입 연산

- 단순 연결 리스트 week2=(월, 금, 일), '월'과 '금' 사이에 '수' 삽입 과정
 - 초기상태

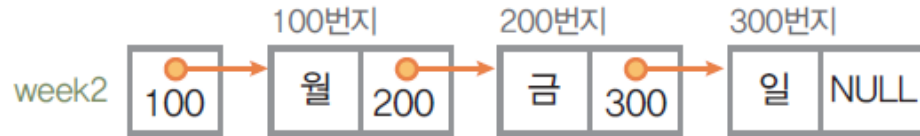
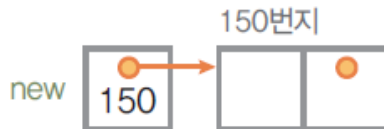
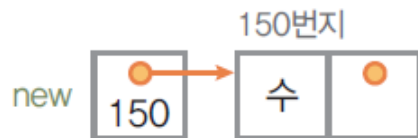


그림 4-12 단순 연결 리스트 week2에 노드를 삽입하기 전인 초기 상태

- ① 삽입할 노드 준비 : 공백 노드를 가져와 포인터 변수 new가 가리키게 함

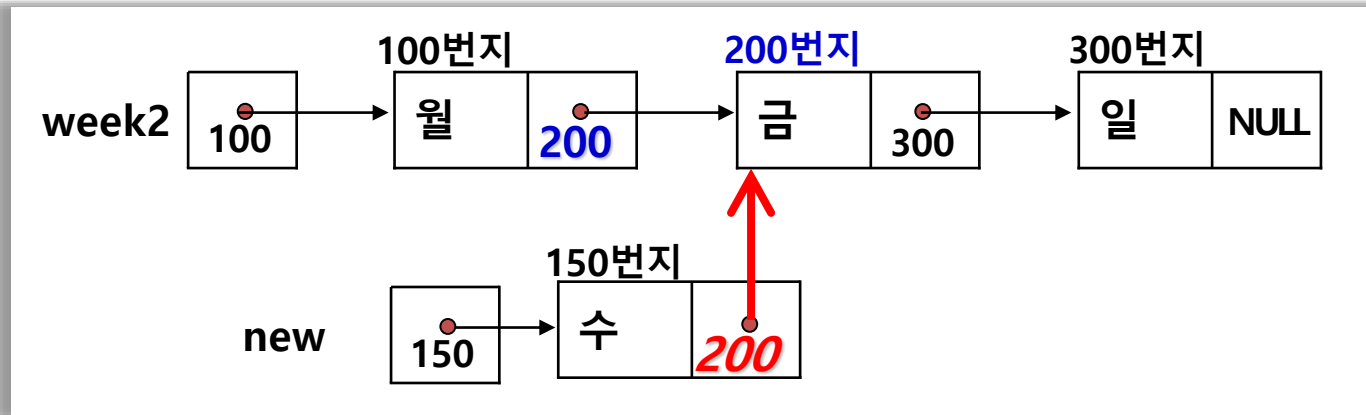


- ② 새 노드의 데이터 필드값 저장 : new의 데이터 필드에 "수"를 저장

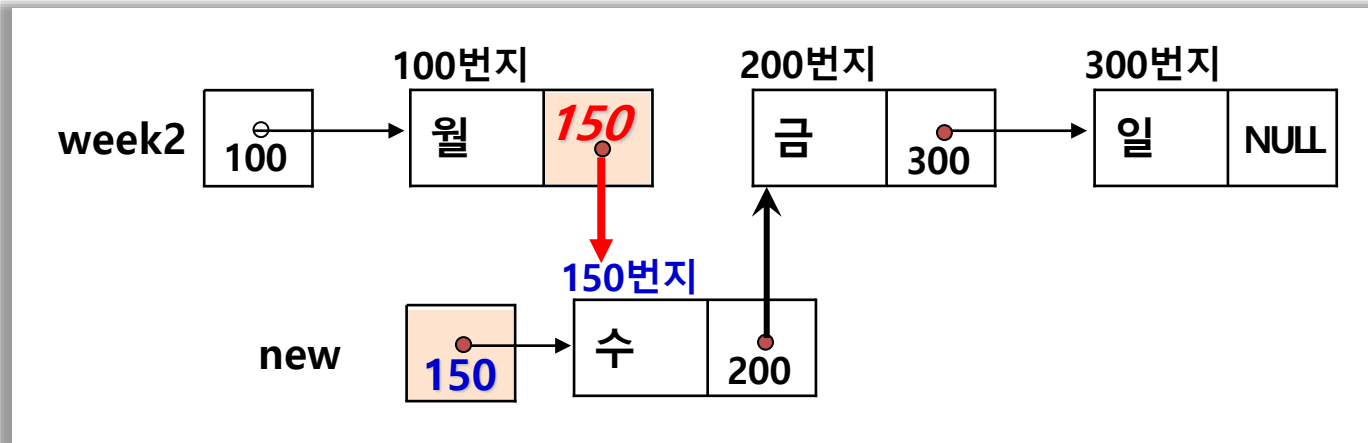


2. 단순 연결 리스트 : 삽입 연산

- ③ 새 노드의 링크 필드값 지정 : new의 앞 노드, 즉 "월"노드의 링크 필드 값을 new의 링크 필드에 저장



- ④ 리스트의 앞 노드에 새 노드 연결 : new의 값을 "월"노드의 링크 필드에 저장



2. 단순 연결 리스트 : 삭제 연산

❖ 단순 연결 리스트에서의 삭제 연산

0 기차놀이에서 철이가 빠지기 전 상태



1 앞사람에게 이름표 넘겨주기



2 이름표대로 연결하기



2. 단순 연결 리스트 : 삭제 연산

■ 단순 연결 리스트에서 노드를 삭제하는 방법

- ❶ 삭제할 노드의 앞 노드를 찾는다.
- ❷ 앞 노드에 삭제할 노드의 링크 필드값을 저장한다.
- ❸ 삭제한 노드의 앞 노드와 삭제한 노드의 다음 노드를 연결한다.



2. 단순 연결 리스트 : 삭제 연산

- 단순 연결 리스트 week2=(월, 수, 금, 일)에서 원소 '수' 삭제 과정
 - 초기 상태



그림 4-14 단순 연결 리스트 week2에서 '수' 노드를 삭제하기 전의 초기 상태

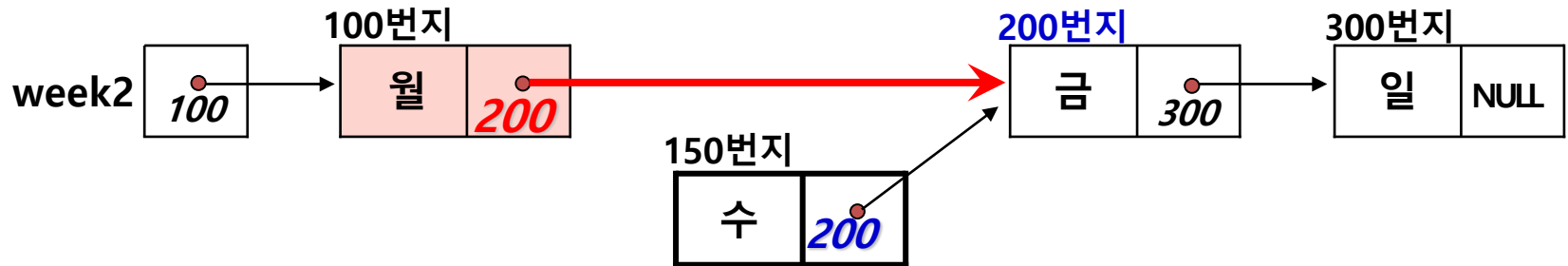


2. 단순 연결 리스트 : 삭제 연산

- ① 앞 노드를 찾음 : 삭제할 원소의 앞 노드(선행자)를 찾음



- ② 앞 노드에 삭제할 노드의 링크 필드값 저장 : 삭제할 원소 "수"의 링크 필드 값을 앞 노드의 링크 필드에 저장
- ③ 삭제한 노드의 앞뒤 노드 연결 : 삭제한 노드의 앞 노드인 '월' 노드를 삭제한 노드의 다음 노드인 '금' 노드에 연결



2. 단순 연결 리스트

❖ 단순 연결 리스트의 알고리즘

- 첫 번째 노드로 삽입하기

알고리즘 4-1 단순 연결 리스트의 첫 번째 노드 삽입

```
insertFirstNode(L, x)
  ① new ← getNode();
  ② new.data ← x;
  ③ new.link ← L;
  ④ L ← new;
end insertFirstNode()
```

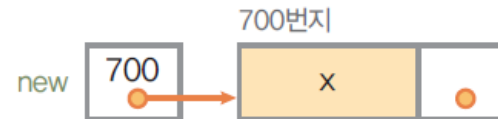


2. 단순 연결 리스트

① `new ← getNode();`

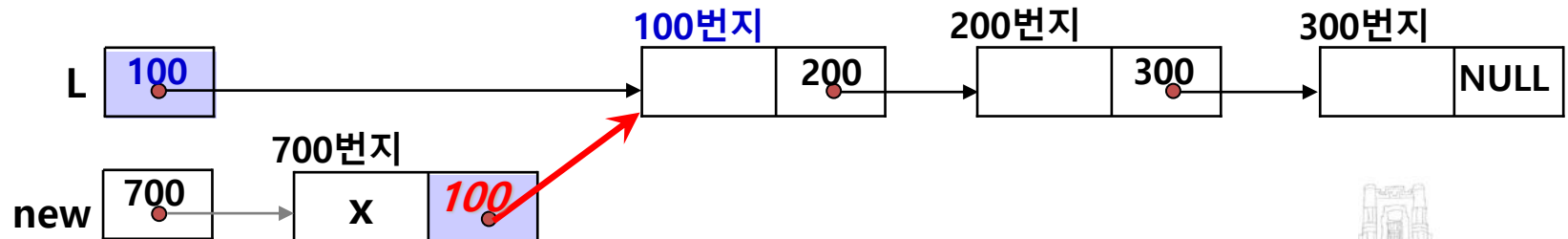


② `new.data ← x;`



③ `new.link ← L;`

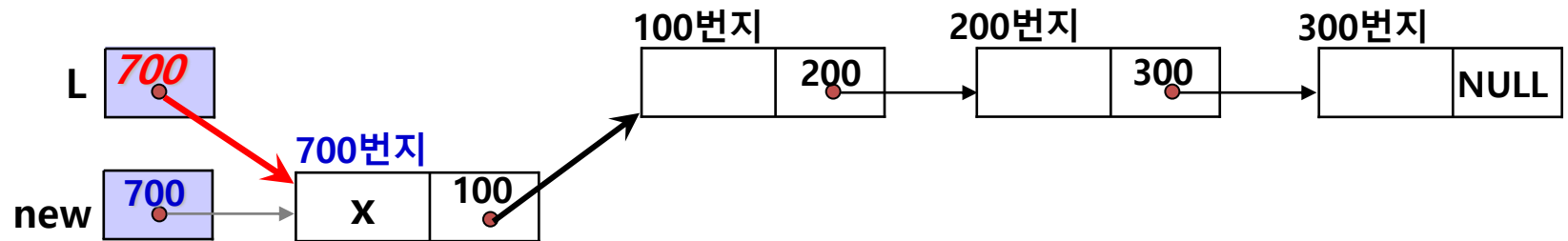
삽입할 노드를 연결하기 위해서 **리스트의 첫 번째 노드 주소(L)**를 삽입할 새 노드 **new의 링크 필드(new.link)**에 저장하여, 새 노드 new가 리스트의 첫 번째 노드를 가리키게 한다.



2. 단순 연결 리스트

④ $L \leftarrow \text{new};$

리스트의 첫 번째 노드 주소를 저장하고 있는 **포인터 L**에, **새 노드의 주소 new**를 저장하여, 포인터 L이 새 노드를 첫 번째 노드로 가리키도록 지정



2. 단순 연결 리스트

■ 중간 노드로 삽입하기

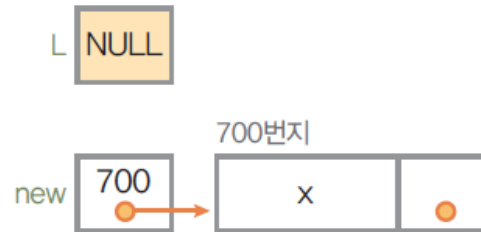
알고리즘 4-2 단순 연결 리스트의 중간 노드 삽입

```
insertMiddleNode(L, pre, x)
  ① new ← getNode();
  ② new.data ← x;
  {
    if (L = NULL) then {
      ③-a L ← new;
      ③-b new.link ← NULL;
    }
    else {
      ④-a new.link ← pre.link;
      ④-b pre.link ← new;
    }
  }
end insertMiddleNode()
```



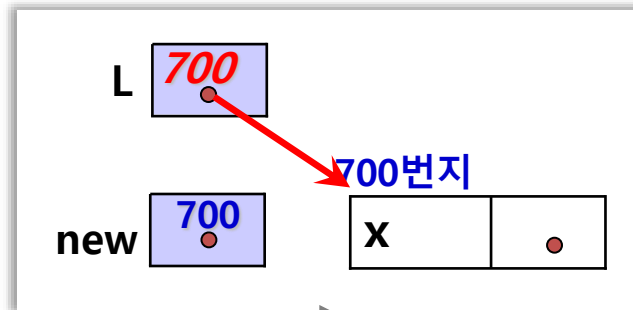
2. 단순 연결 리스트

- 1 new \leftarrow getNode();
- 2 new.data \leftarrow x;
- 3 공백 리스트인 경우



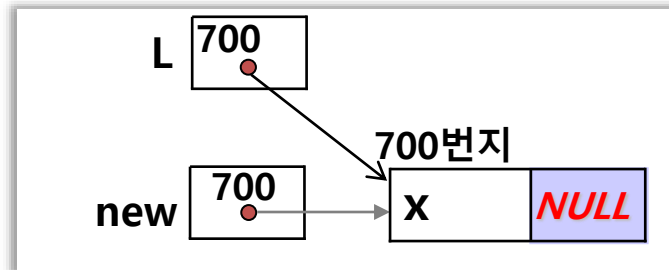
3- a L \leftarrow new;

리스트 포인터 **L**에 새 노드 **new**의 주소를 저장하여, 새 노드 **new**가 리스트의 첫 번째 노드가 되도록 함



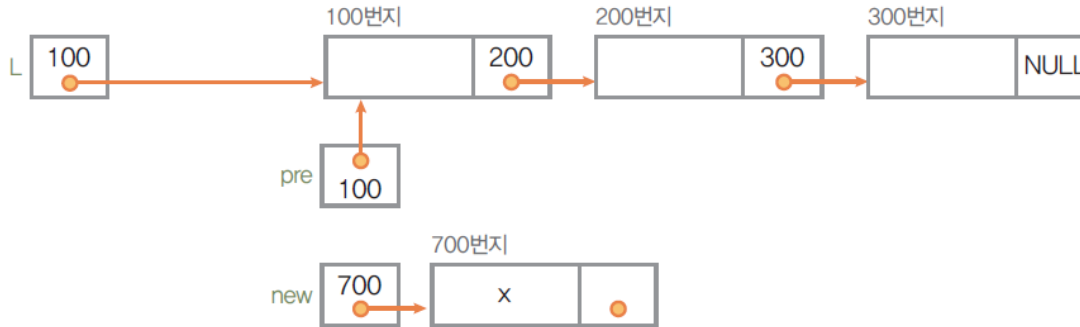
3- b new.link \leftarrow NULL;

리스트의 마지막 노드인 **new**의 링크 필드에 **NULL**을 저장해 마지막 노드임을 표시



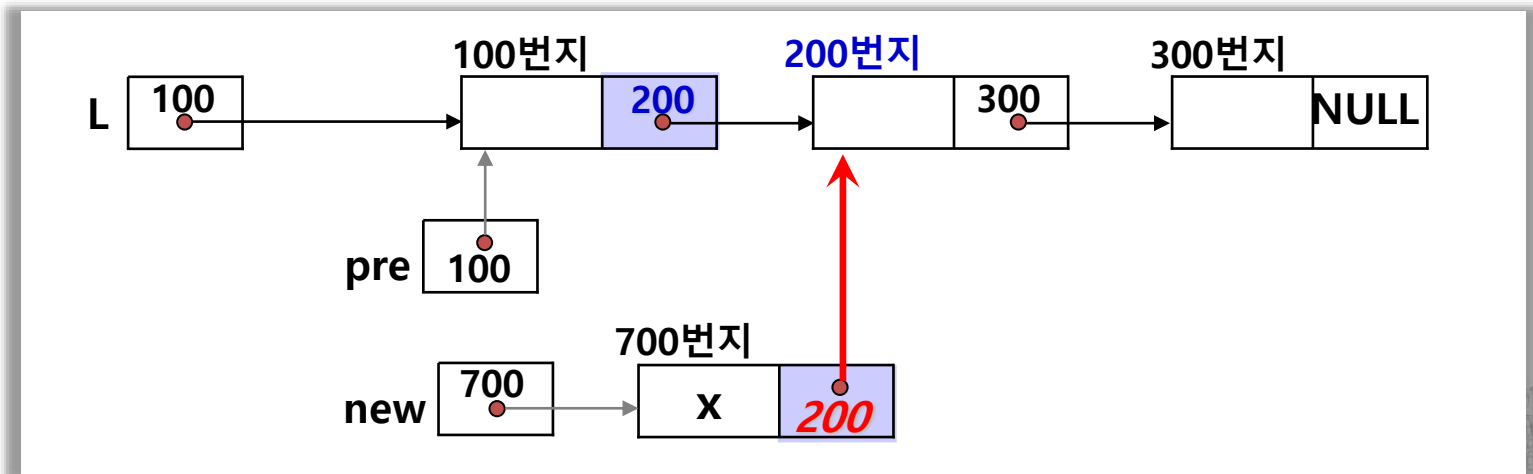
2. 단순 연결 리스트

4 공백 리스트가 아닌 경우



4- a new.link ← pre.link;

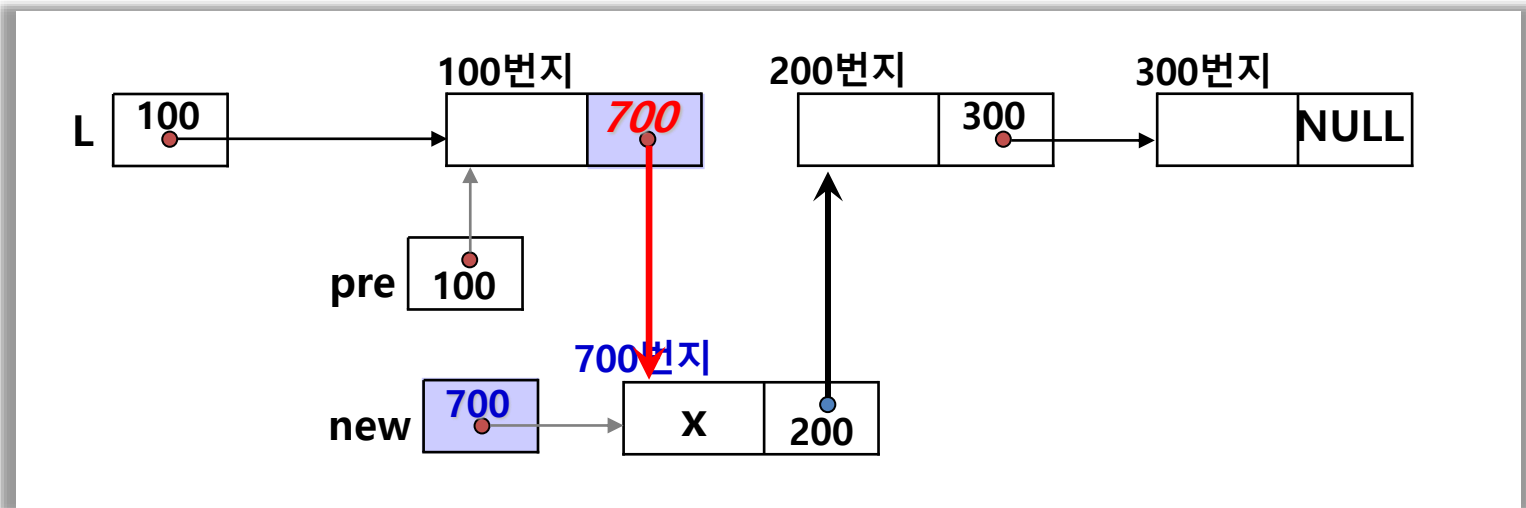
노드 pre의 링크 필드 값을 노드 new의 링크 필드에 저장하여, 새 노드 new가 노드 pre의 다음 노드를 가리키도록 함



2. 단순 연결 리스트

4- b $\text{pre.link} \leftarrow \text{new};$

포인터 **new**의 값을 **노드 pre의 링크 필드에** 저장하여, 노드 pre가 새 노드 new를 다음 노드로 가리키도록 함



2. 단순 연결 리스트

■ 마지막 노드로 삽입하기

알고리즘 4-3 단순 연결 리스트의 마지막 노드 삽입

```
insertLastNode(L, x)
    ① new ← getNode();
    ② new.data ← x;
    ③ new.link ← NULL;
    ④ { if (L = NULL) then {
        ④-a L ← new;
        return;
    }
    ⑤ { ⑤-a temp ← L;
        while (temp.link ≠ NULL) do
            ⑤-b temp ← temp.link;
        ⑤-c temp.link ← new;
    }
end insertLastNode()
```



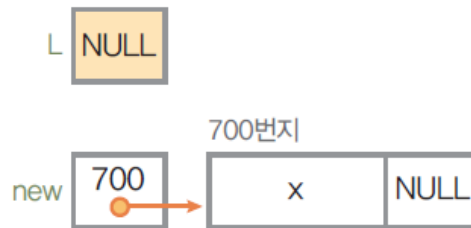
2. 단순 연결 리스트

① $\text{new} \leftarrow \text{getNode}();$

② $\text{new.data} \leftarrow x;$

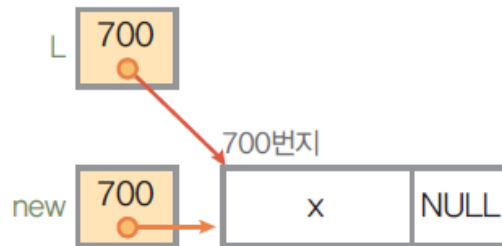
③ $\text{new.link} \leftarrow \text{NULL};$

④ 공백 리스트인 경우



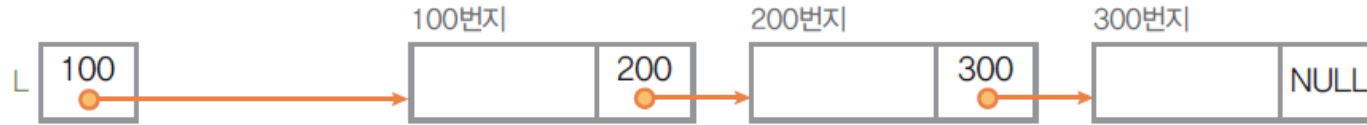
④- a $L \leftarrow \text{new};$

리스트 포인터 L에 새 노드 new의 주소(700) 저장. new는 리스트 L의 첫 번째 노드이자 마지막 노드



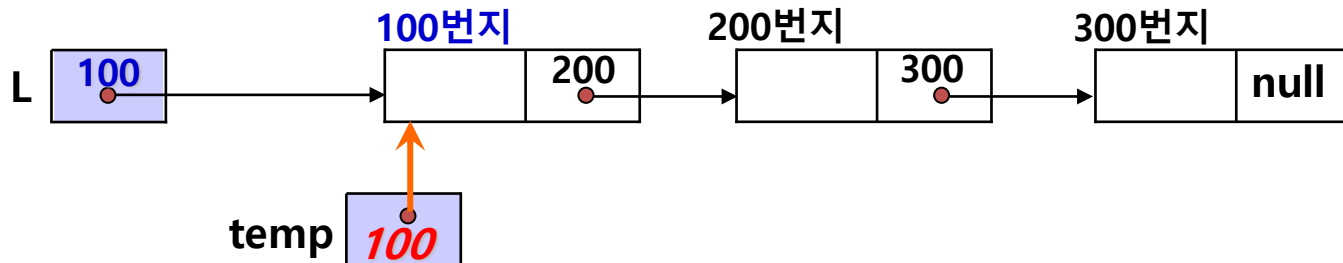
2. 단순 연결 리스트

⑤ 공백 리스트가 아닌 경우



⑤- a temp ← L;

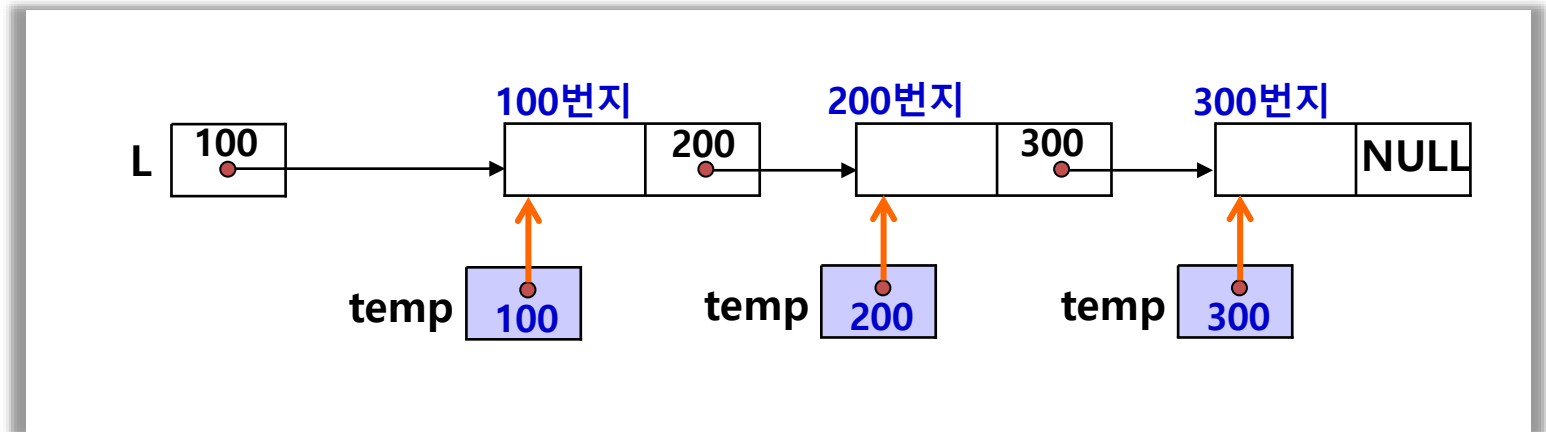
현재 리스트 L의 마지막 노드를 찾기 위해서 노드를 순회할 임시포인터 temp에 리스트의 첫 번째 노드의 주소(L)를 지정



2. 단순 연결 리스트

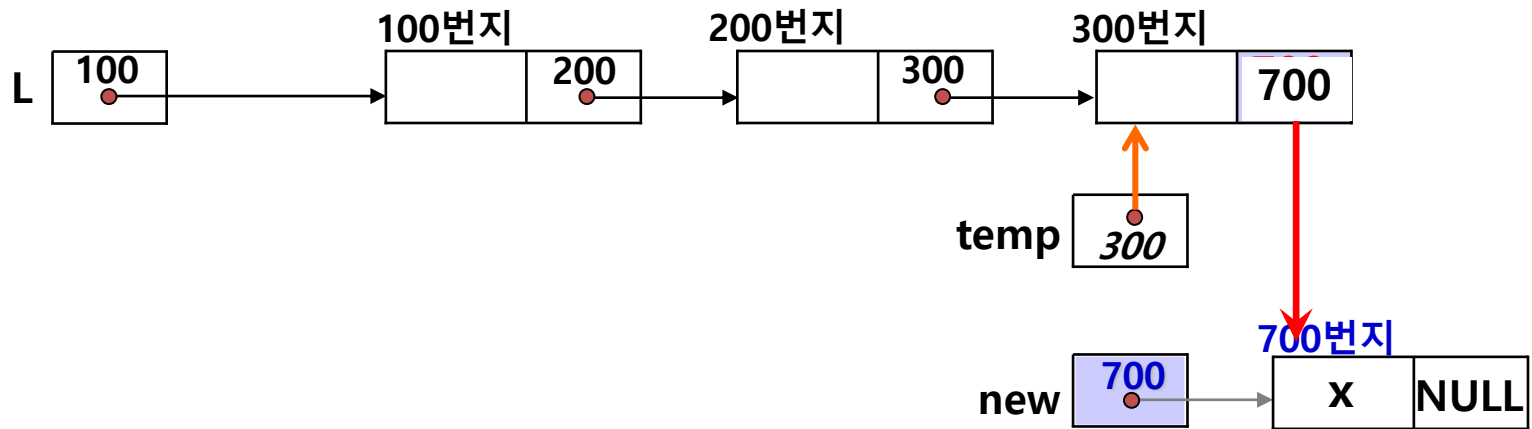
⑤- b $\text{temp} \leftarrow \text{temp.link};$

순회 포인터 temp가 가리키는 노드의 링크 필드가 NULL이 아닌 동안(while ($\text{temp.link} \neq \text{NULL}$)) 링크 필드를 따라 이동. 링크 필드가 NULL인 노드 즉, 마지막 노드를 찾으면 while 문을 끝내고 ⑤- c를 수행



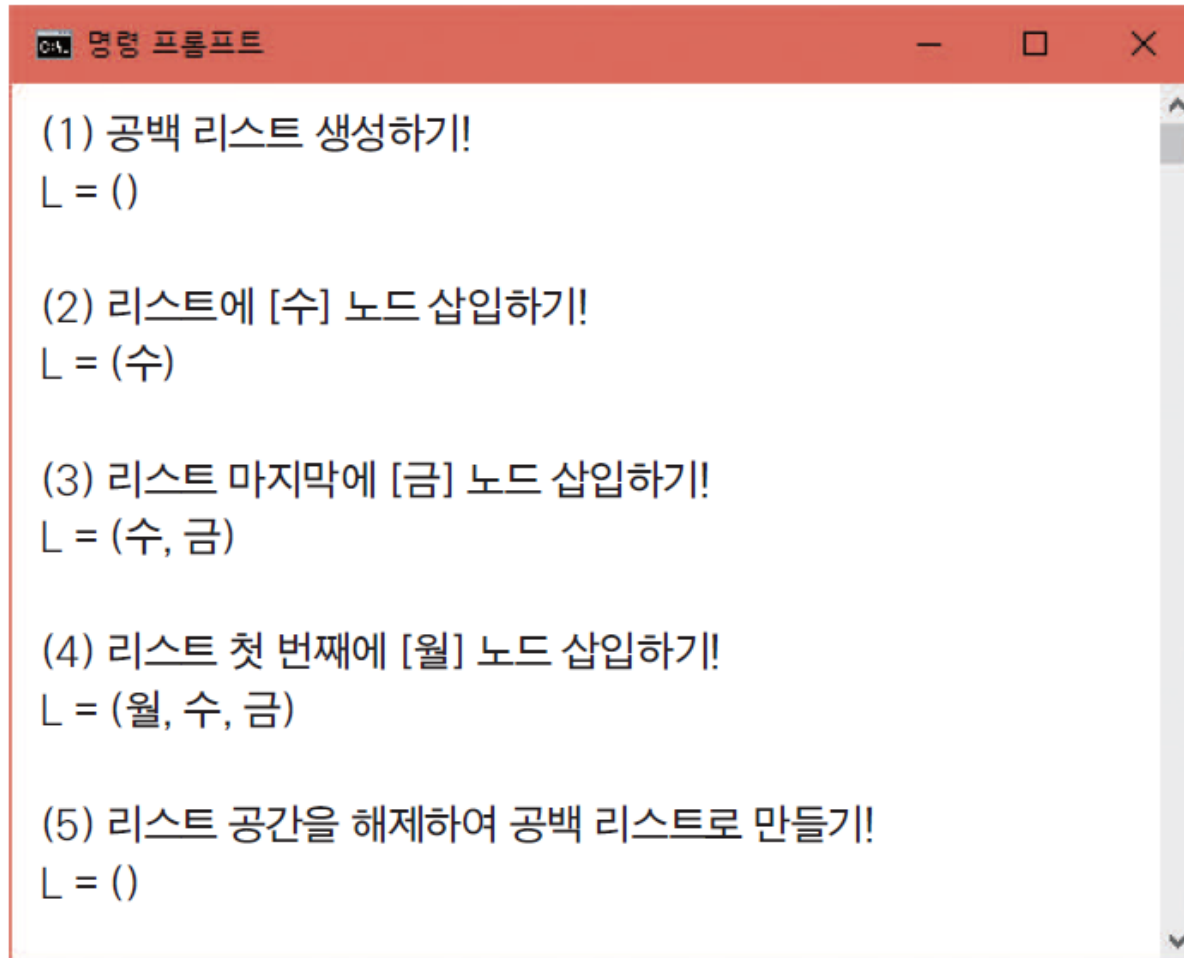
2. 단순 연결 리스트

⑤- c temp.link ← new;



2. 단순 연결 리스트

- 단순 연결 리스트 프로그램 1 : [교재 169p](#)
- 실행 결과



```
명령 프롬프트

(1) 공백 리스트 생성하기!
L = ()

(2) 리스트에 [수] 노드 삽입하기!
L = (수)

(3) 리스트 마지막에 [금] 노드 삽입하기!
L = (수, 금)

(4) 리스트 첫 번째에 [월] 노드 삽입하기!
L = (월, 수, 금)

(5) 리스트 공간을 해제하여 공백 리스트로 만들기!
L = ()
```



2. 단순 연결 리스트

- 리스트 L에서 포인터 pre가 가리키는 노드의 다음 노드 삭제 알고리즘 : 포인터 old(삭제할 노드)

알고리즘 4-4 단순 연결 리스트의 노드 삭제

```
deleteNode(L, pre)
  ① if (L = NULL) then error;
  2 {
    ②-a old ← pre.link;
    ②-b if (old = NULL) then return;
    ②-c pre.link ← old.link;
    ②-d returnNode(old);
  }
end deleteNode()
```



2. 단순 연결 리스트

- 삭제 연산 수행 과정

- ① 공백 리스트인 경우

- 공백 연결 리스트 L이 공백이면 오류 처리한다.



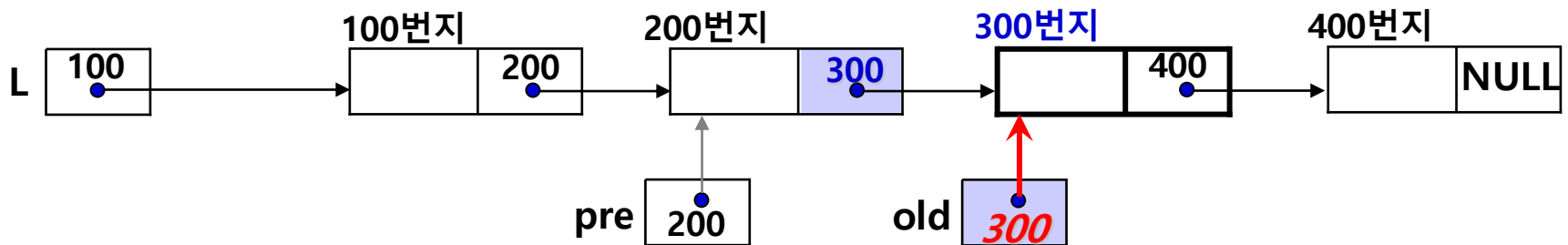
2. 단순 연결 리스트

■ 삭제 연산 수행 과정

② 공백 리스트가 아닌 경우

②- a $old \leftarrow pre.link;$

노드 **pre**의 다음노드의 주소(**pre.link**)를 포인터 **old**에 저장하여, 포인터 **old**가 다음 노드를 가리키게 한다.

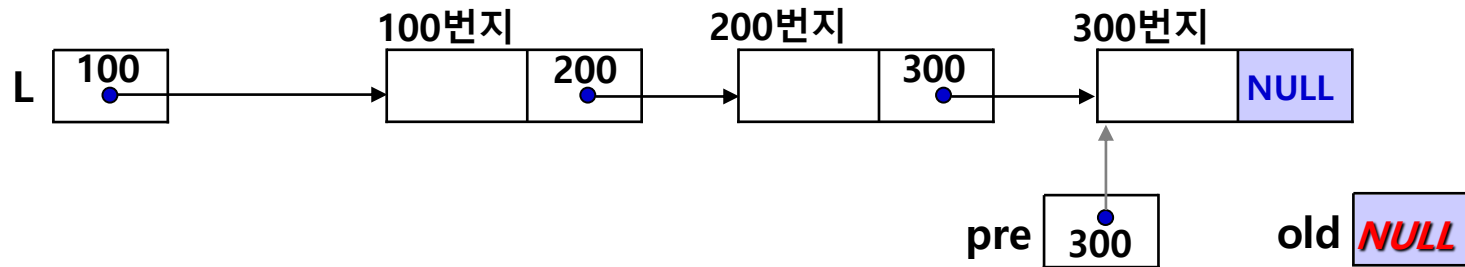


2. 단순 연결 리스트

②- b if (old = NULL) then return;

만약 노드 pre가 리스트의 마지막 노드였다면 :

포인터 pre가 가리키는 노드가 마지막 노드인 경우에,
②-① old \leftarrow pre.link; 를 수행한 상태 :

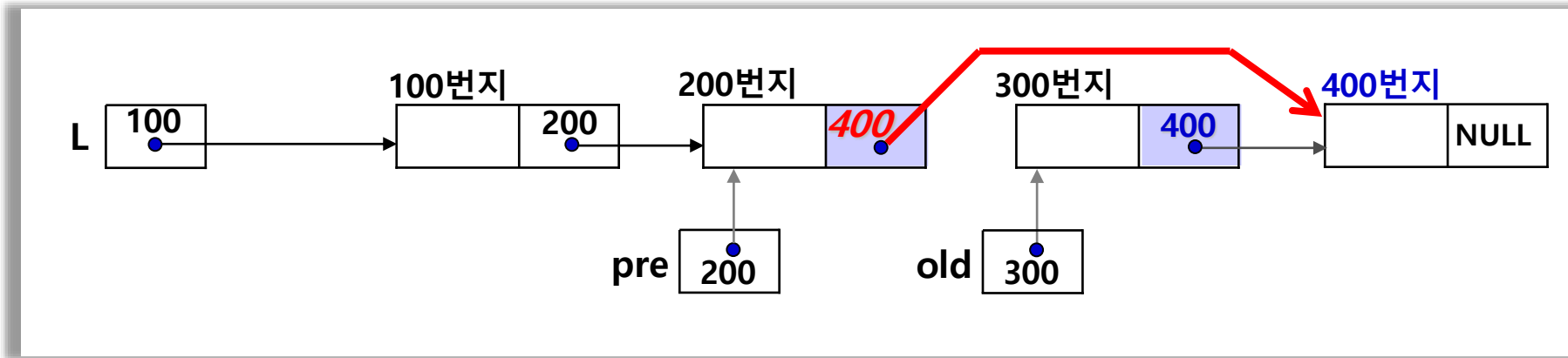


\Rightarrow pre.link의 값은 null이므로 포인터 old의 값은 NULL이 된다. 결국 노드 pre 다음에 삭제할 노드가 없다는 의미이므로 삭제연산을 수행하지 못하고 반환 (return).



2. 단순 연결 리스트

②- c `pre.link ← old.link;`



②- d `returnNode(old);`

삭제한 노드 old의 메모리를 반환



2. 단순 연결 리스트

❖ 노드를 탐색하는 알고리즘과 프로그램

■ 노드를 탐색하는 알고리즘

알고리즘 4-5 단순 연결 리스트의 노드 탐색

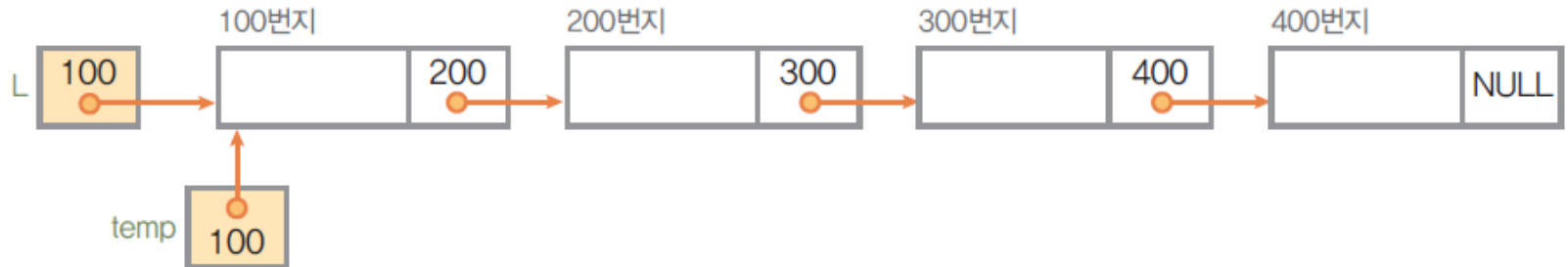
```
searchNode(L, x)
  ① temp ← L;
  { while (temp ≠ NULL) do {
    ② { ②-a if (temp.data = x) then return temp;
        ②-b else temp ← temp.link;
      }
    ③ return temp;
  }
end searchNode()
```



2. 단순 연결 리스트

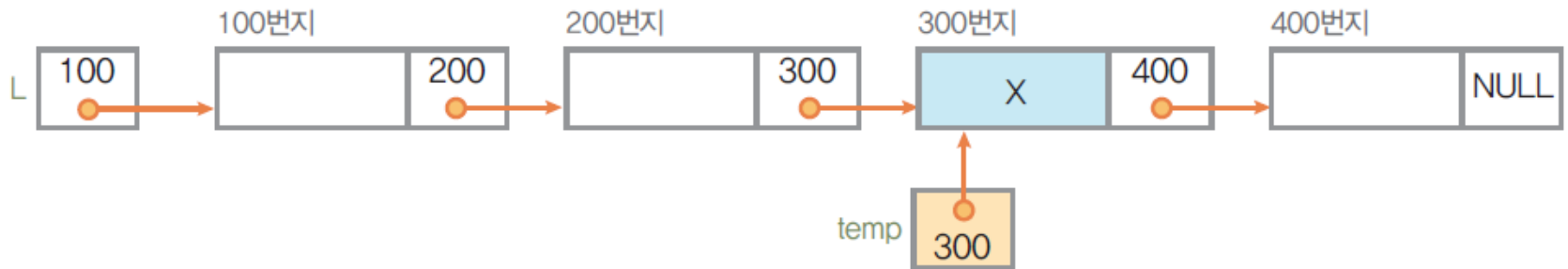
- 단순 연결 리스트에서 x 노드 탐색 과정

① $\text{temp} \leftarrow L;$



② 순회 포인터가 NULL이 아닌 경우

②- a if ($\text{temp.data} = x$) then return temp;

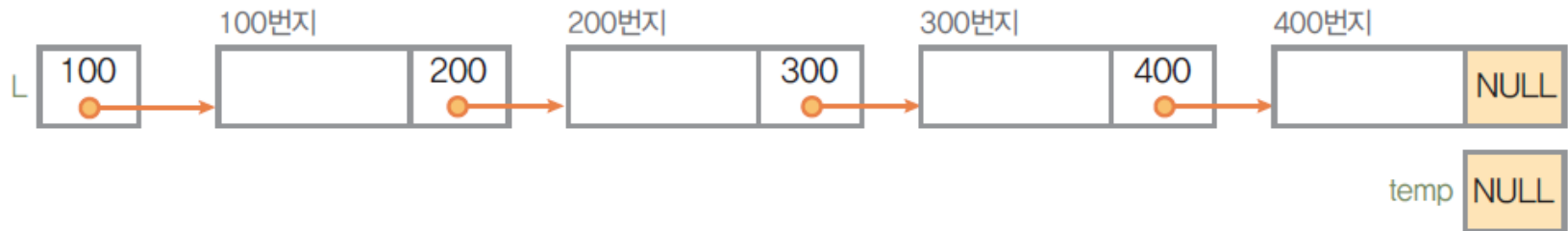


2. 단순 연결 리스트

- 단순 연결 리스트에서 x 노드 탐색 과정

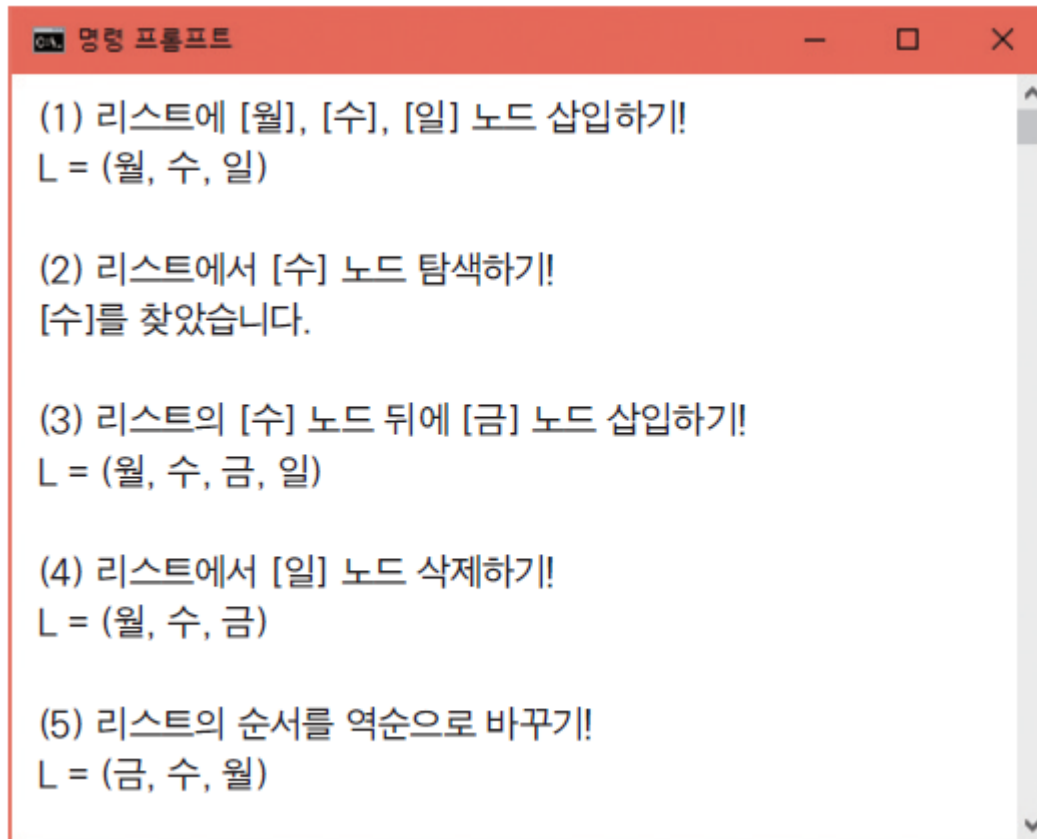
②- b else temp \leftarrow temp.link;

③ return temp;



2. 단순 연결 리스트

- 단순 연결 리스트 프로그램 2 : [교재 174p](#)
- 실행 결과



```
명령 프롬프트

(1) 리스트에 [월], [수], [일] 노드 삽입하기!
L = (월, 수, 일)

(2) 리스트에서 [수] 노드 탐색하기!
[수]를 찾았습니다.

(3) 리스트의 [수] 노드 뒤에 [금] 노드 삽입하기!
L = (월, 수, 금, 일)

(4) 리스트에서 [일] 노드 삭제하기!
L = (월, 수, 금)

(5) 리스트의 순서를 역순으로 바꾸기!
L = (금, 수, 월)
```



2. 단순 연결 리스트

- 129행 ~ 131행 시작 상태

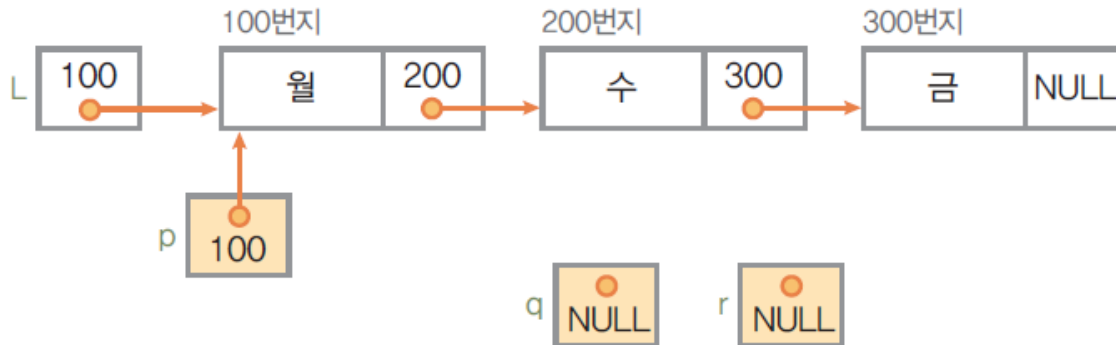
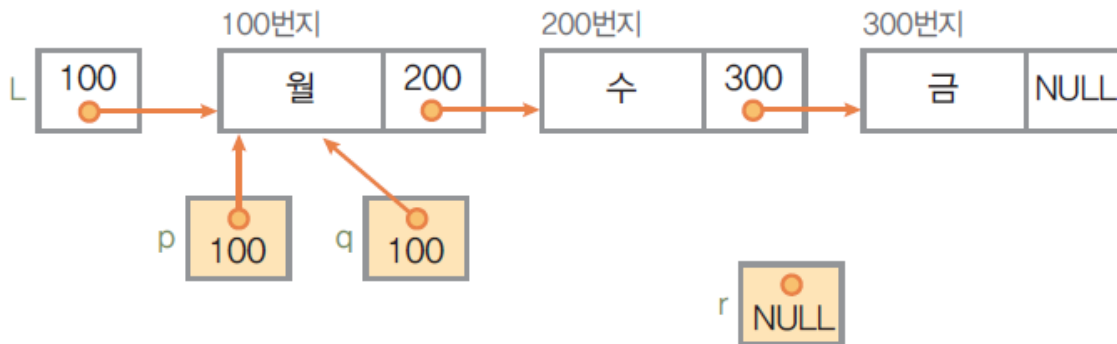


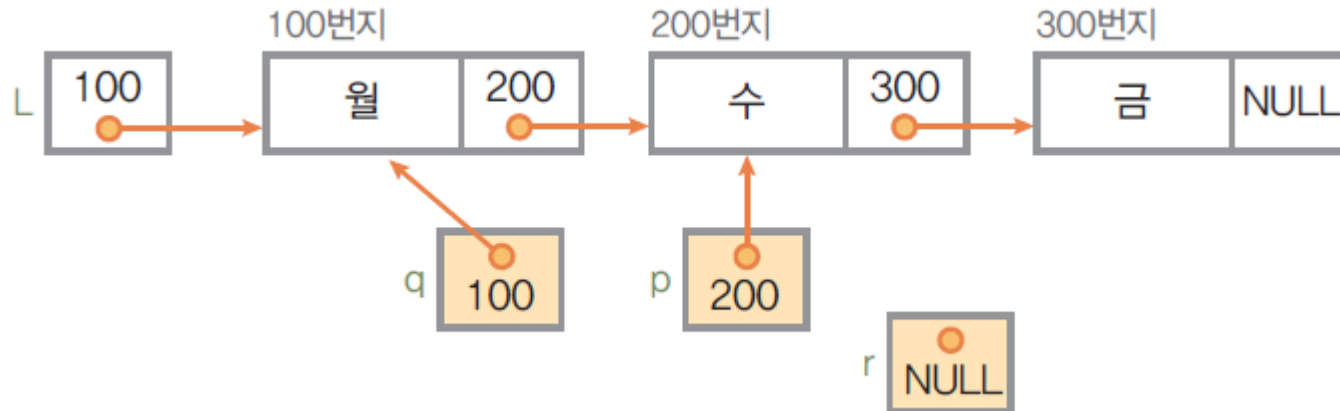
그림 4-15 초기 상태

- 137행 : 포인터 q를 포인터 p가 가리키는 현재 노드에 연결

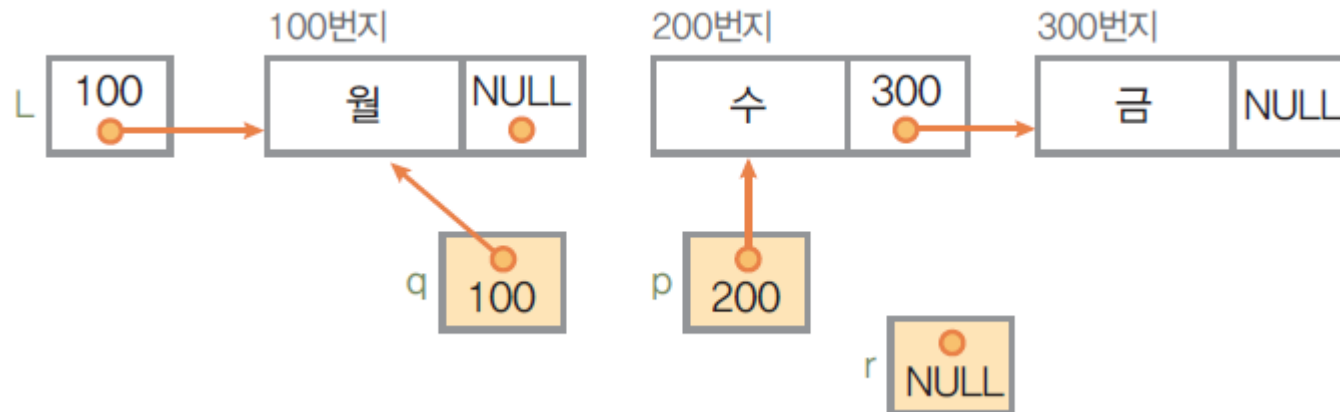


2. 단순 연결 리스트

- 138행 : 포인터 p는 링크를 따라 다음 노드로 이동

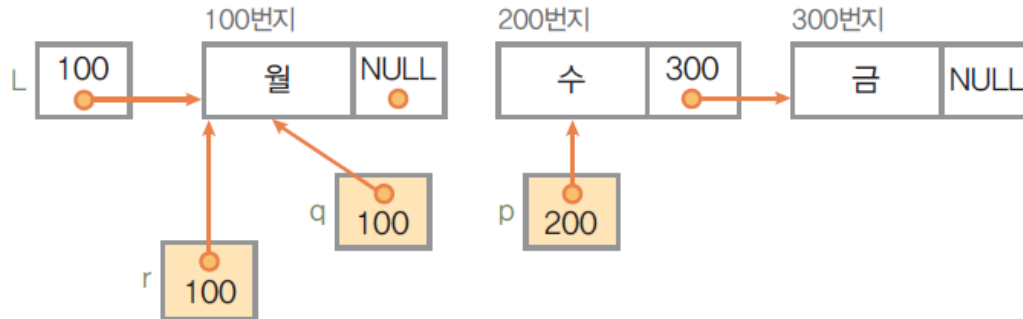


- while 문 한 번 수행

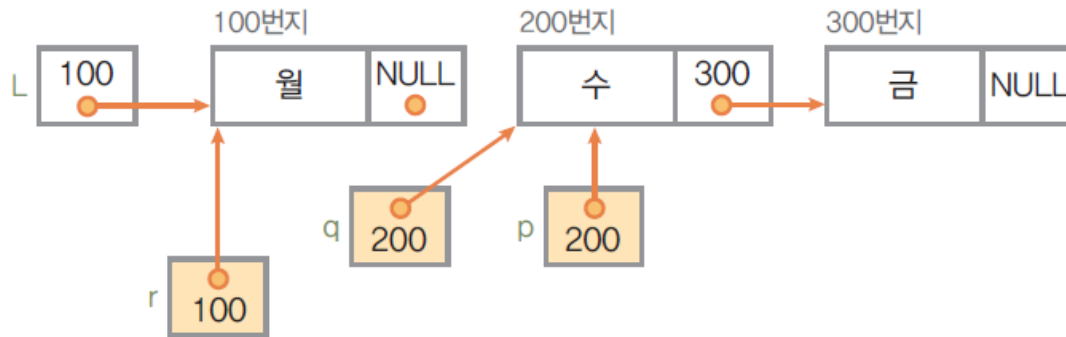


2. 단순 연결 리스트

- 136행 : 포인터 r을 포인터 q가 가리키는 노드에 연결

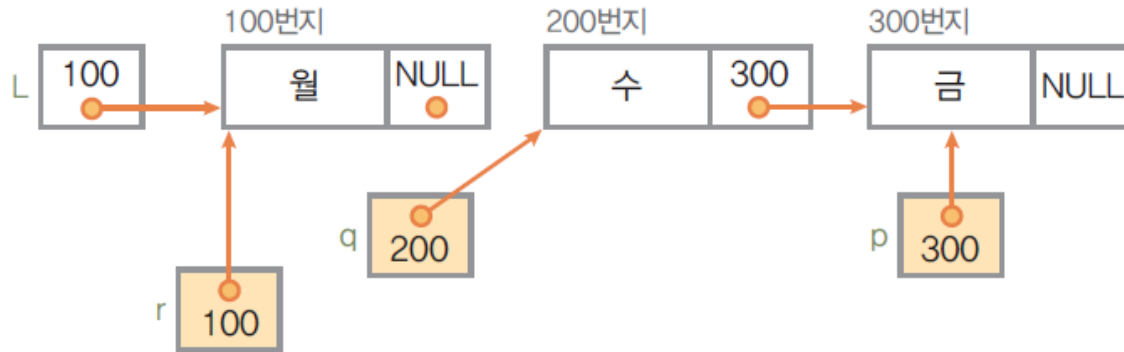


- 137행 : 포인터 q를 포인터 p가 가리키는 현재 노드에 연결

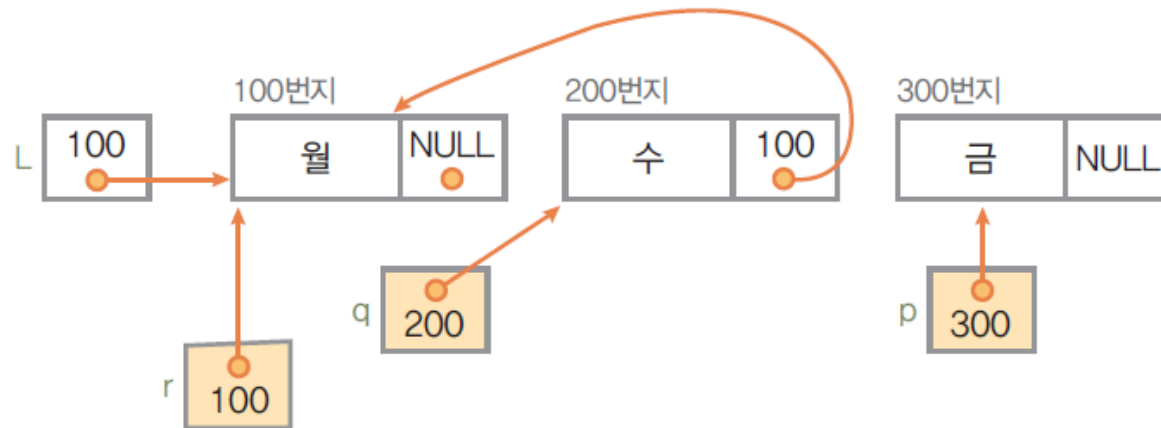


2. 단순 연결 리스트

- 138행 : 포인터 p는 링크를 따라 다음 노드로 이동



- 139행 : 포인터 r의 값(100)을 q 노드의 링크 필드에 설정



2. 단순 연결 리스트

- while 문을 두 번 수행

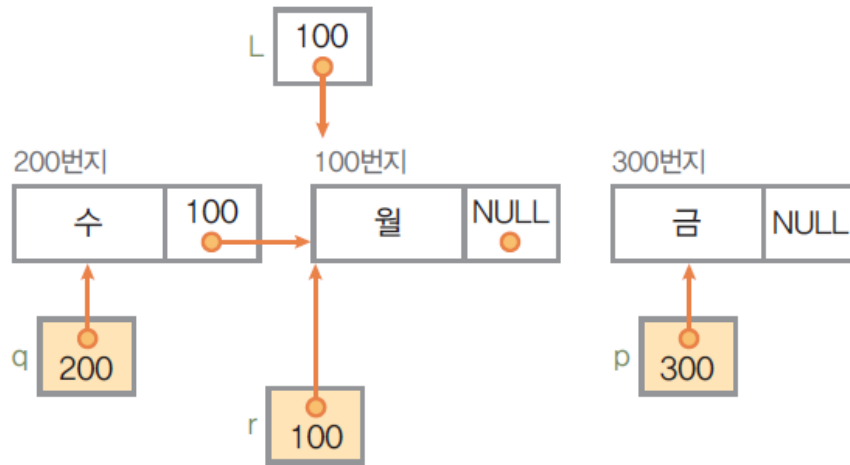
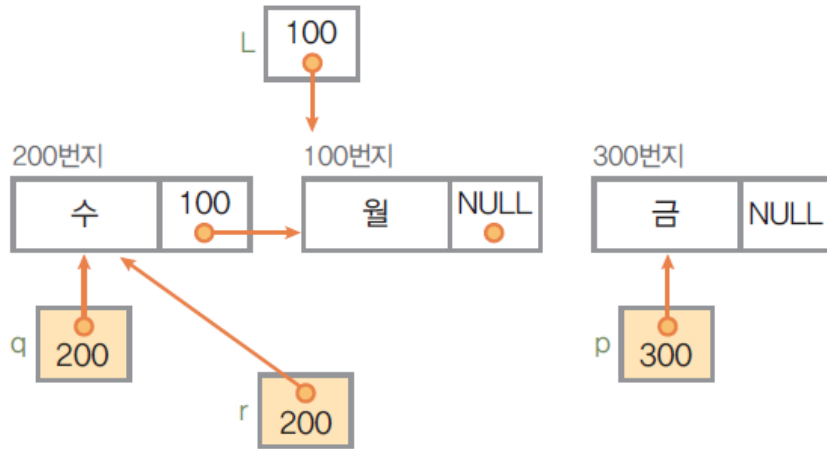


그림 4-17 while 문을 두 번 수행한 결과

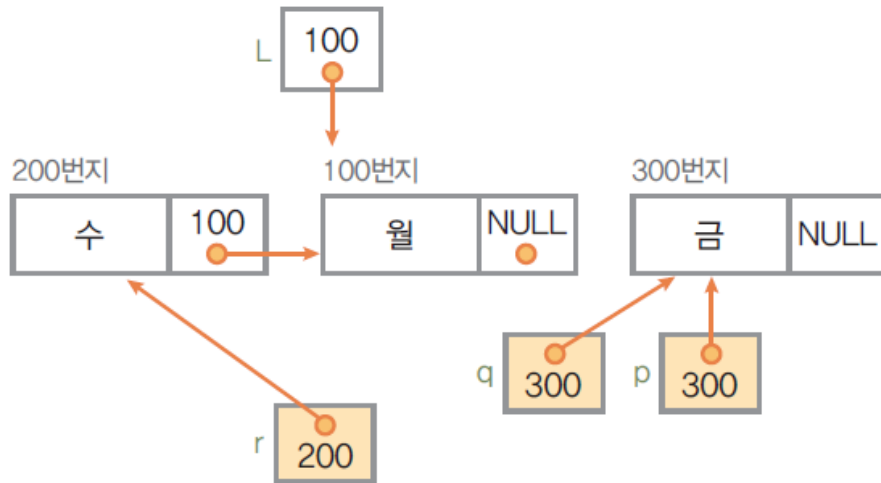


2. 단순 연결 리스트

- 136행 : 포인터 r을 포인터 q가 가리키는 노드에 연결

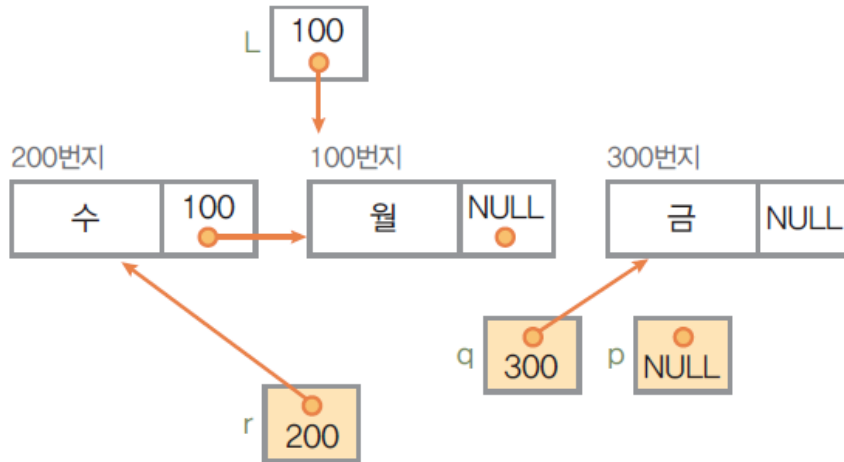


- 137행 : 포인터 q를 포인터 p가 가리키는 노드에 연결

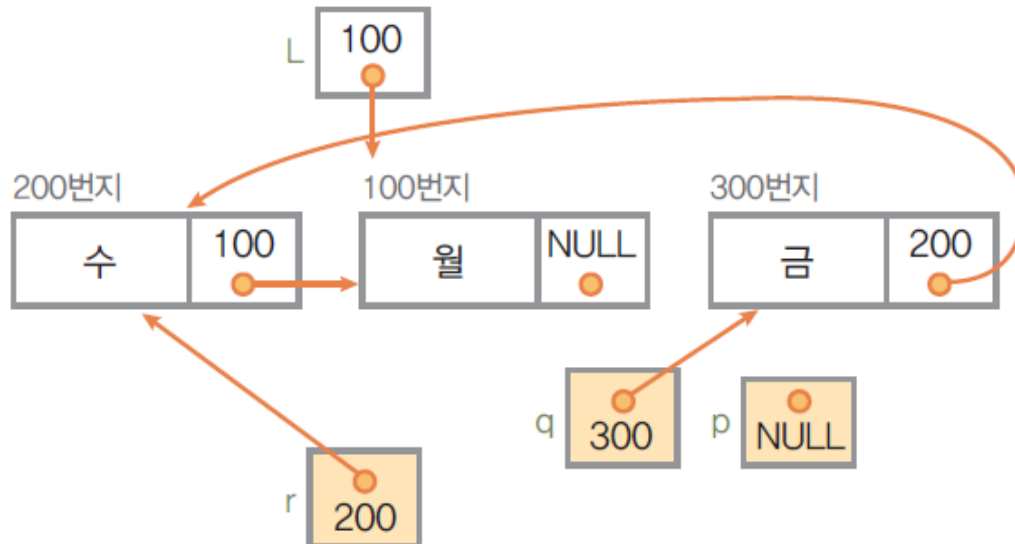


2. 단순 연결 리스트

- 138행 : 포인터 p는 링크를 따라 다음 노드로 이동



- 139행 : 포인터 r의 값(200)을 q 노드의 링크 필드에 설정



2. 단순 연결 리스트

- while 문을 세 번 수행

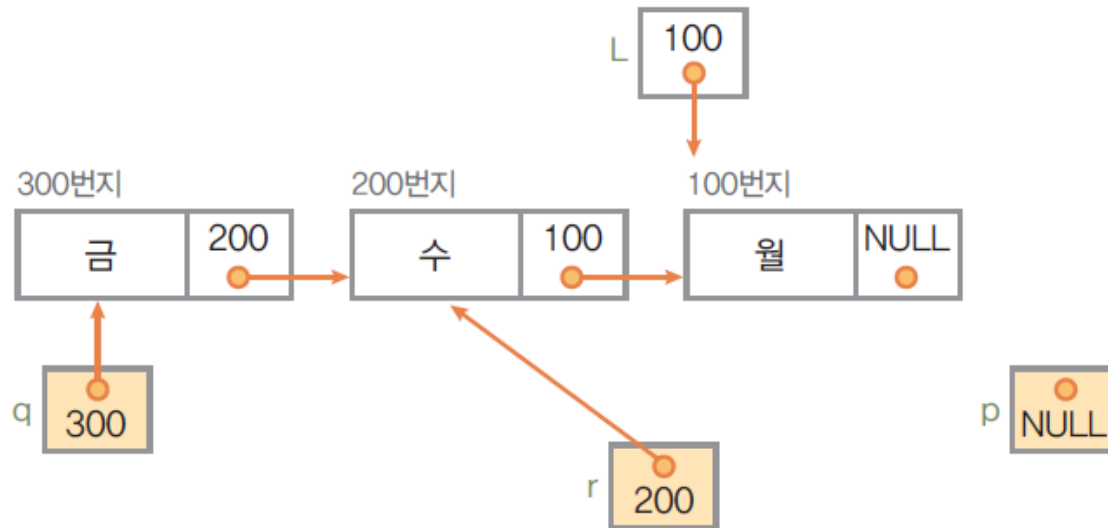


그림 4-18 while 문을 세 번 수행한 결과



2. 단순 연결 리스트

■ 완성된 연결 리스트

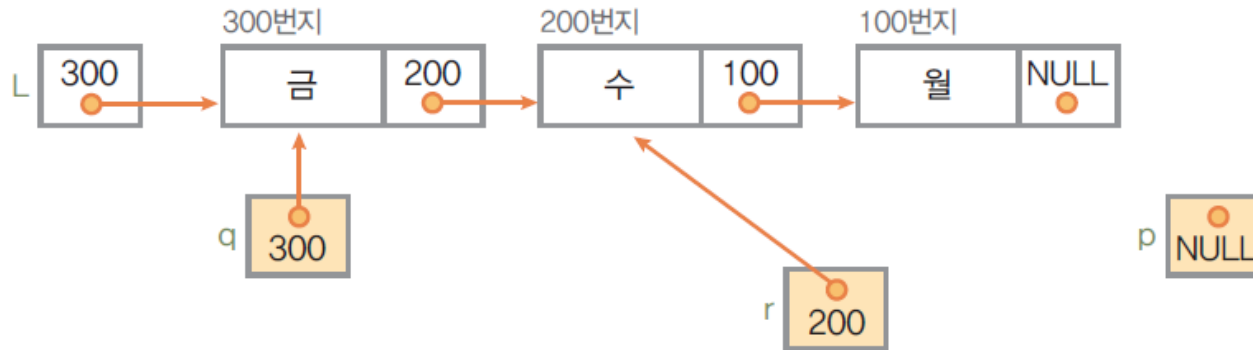


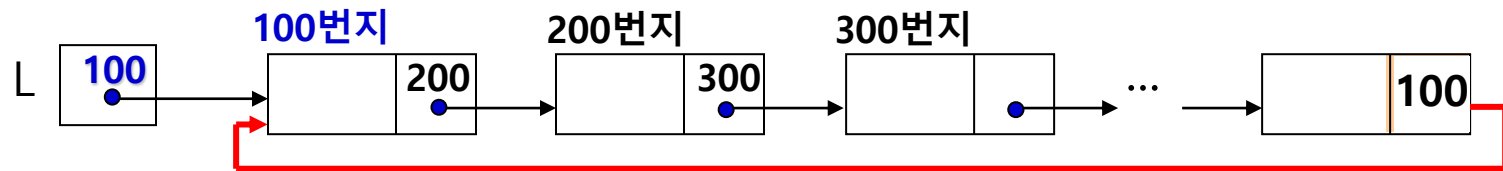
그림 4-19 완성된 연결 리스트



3. 원형 연결 리스트

❖ 원형 연결 리스트의 개념

- 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
 - 예



3. 원형 연결 리스트

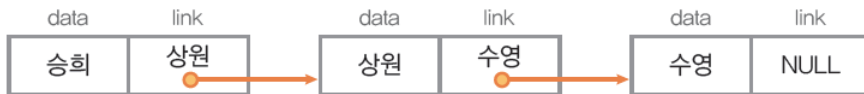
- 선형 기차 놀이와 원형 기차놀이 비교



선형 기차 놀이



원형 기차 놀이



기차 놀이에 대한 노드 표현



3. 원형 연결 리스트

❖ 원형 연결 리스트의 알고리즘

- 마지막 노드의 링크를 첫 번째 노드로 연결하는 부분만 제외하고는 단순 연결 리스트에서의 삽입 연산과 같은 연산
- 리스트에 첫 번째 노드로 삽입하는 알고리즘

알고리즘 4-6 원형 연결 리스트의 첫 번째 노드 삽입

```
insertFirstNode(CL, x)
    new ← getNode();
    new.data ← x;
    {
        1 { if (CL = NULL) then {
            1-a CL ← new;
            1-b new.link ← new;
        }

        else {
            2-a temp ← CL;
            while (temp.link ≠ CL) do
            2 { 2-b temp ← temp.link;
                2-c new.link ← temp.link;
                2-d temp.link ← new;
                2-e CL ← new;
            }
        }
    }
end insertFirstNode()
```



3. 원형 연결 리스트

- 원형 연결 리스트 마지막에 노드를 삽입하는 과정

❶ 공백 리스트인 경우

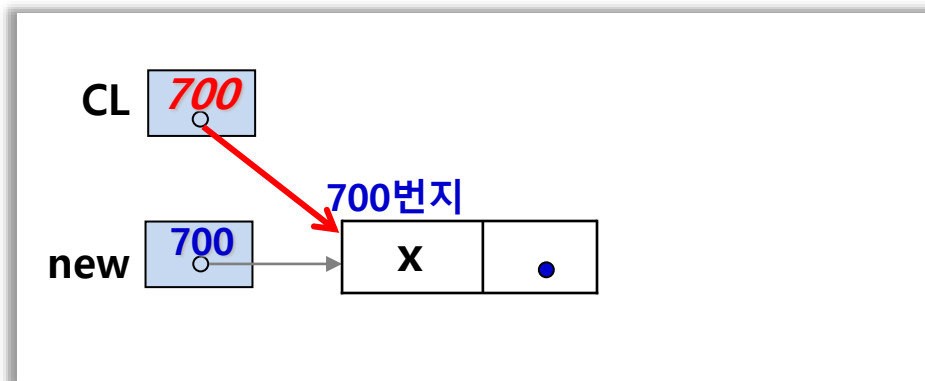
삽입하는 노드 new는 리스트의 첫 번째 노드이자 마지막 노드가 되어야 한다.

CL NULL



❶- a $CL \leftarrow new;$

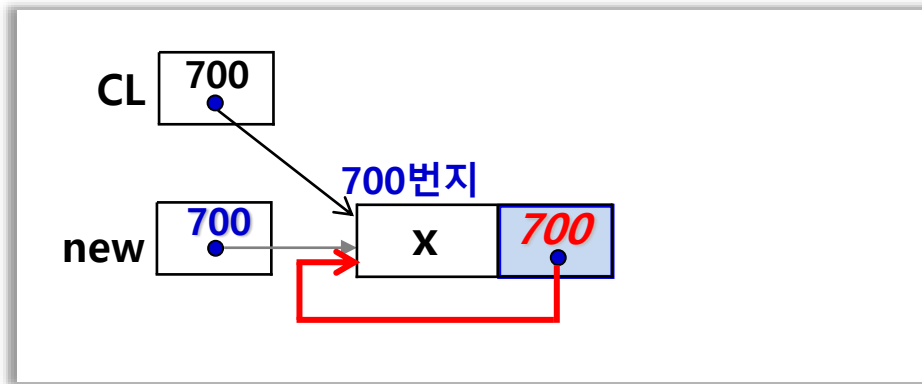
포인터 CL이 노드 new를 가리키게 한다.



3. 원형 연결 리스트

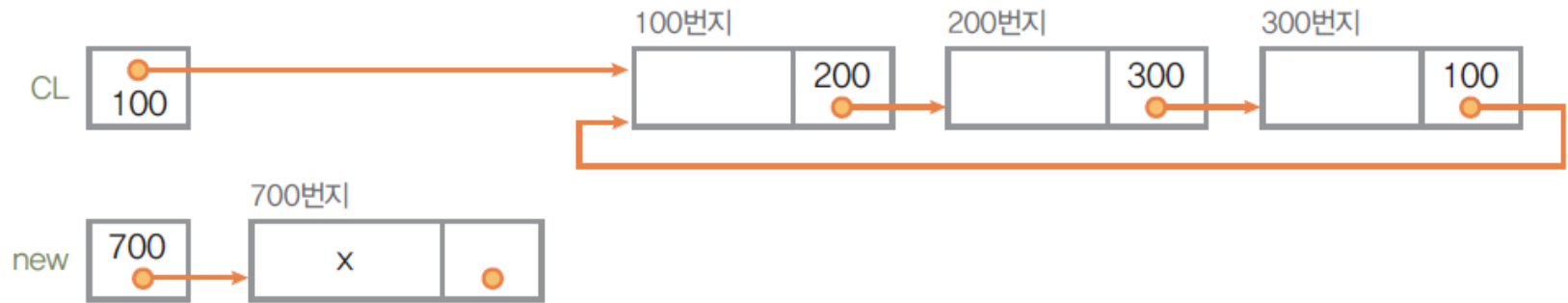
①- b `new.link ← new;`

- 노드 `new`가 자기자신을 가리키게 함으로써 노드 `new`를 첫 번째 노드이자 마지막 노드가 되도록 지정한다.



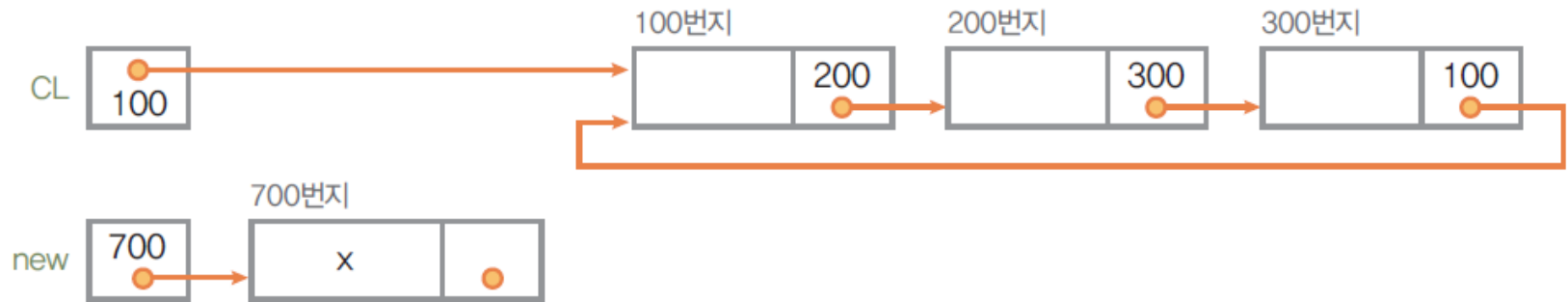
3. 원형 연결 리스트

② 공백 리스트가 아닌 경우



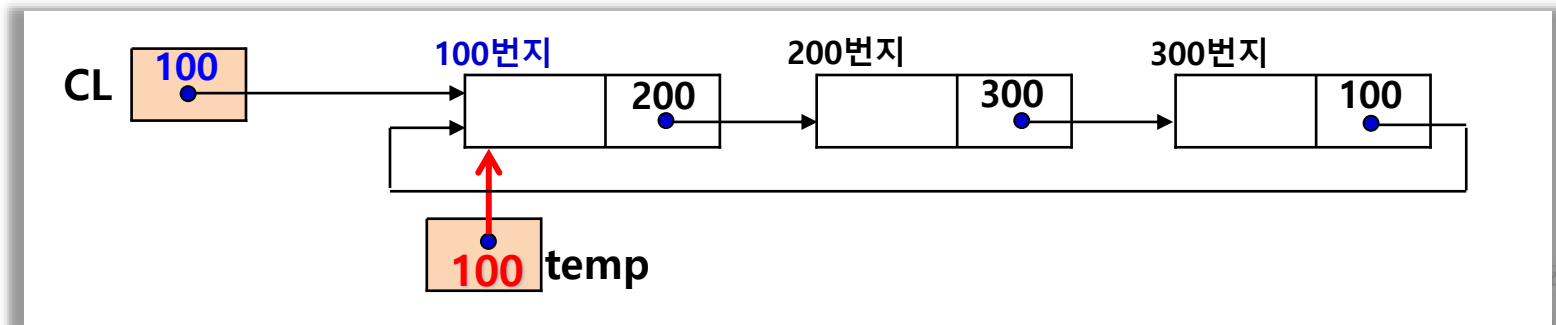
3. 원형 연결 리스트

② 공백 리스트가 아닌 경우



②- a temp ← CL;

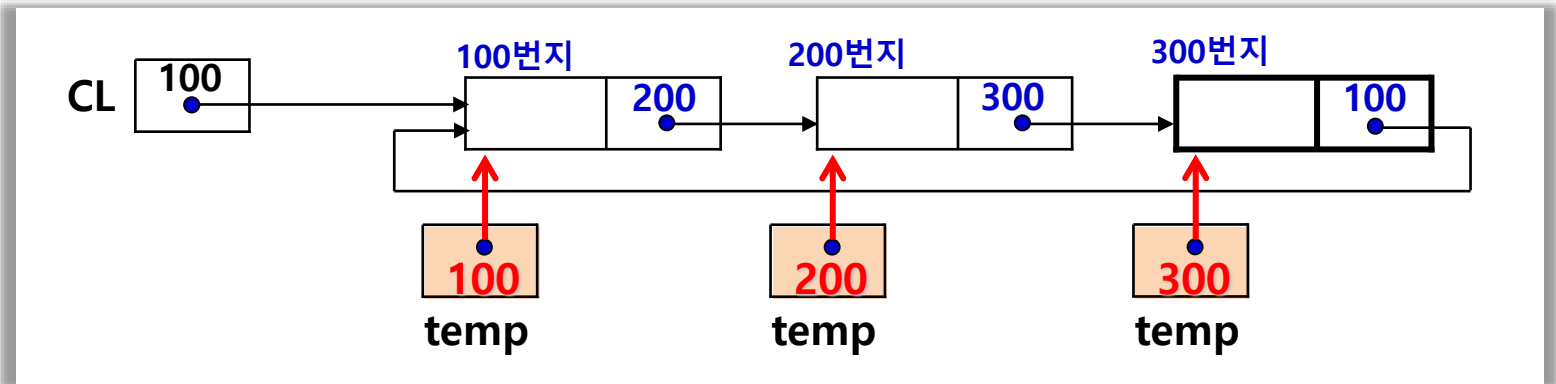
리스트가 공백리스트가 아닌 경우에는 첫 번째 노드의 주소를 임시 순회 포인터 temp에 저장하여 노드 순회의 시작점을 지정한다.



3. 원형 연결 리스트

②- b $\text{temp} \leftarrow \text{temp.link};$

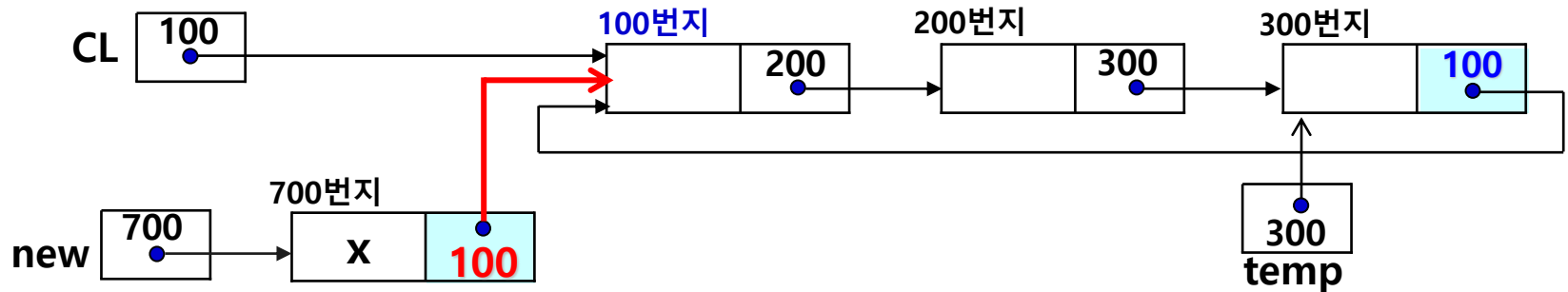
while 문을 수행해 순회 포인터 temp를 링크를 따라 마지막 노드까지 이동



3. 원형 연결 리스트

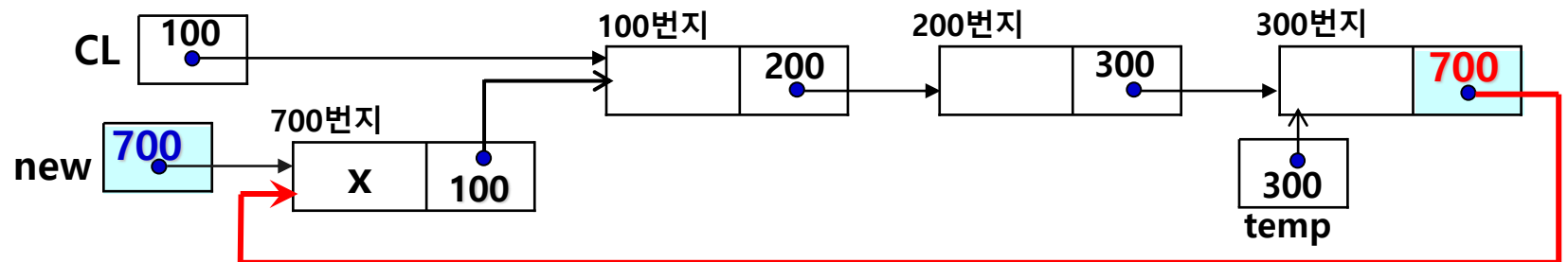
②- c $\text{new.link} \leftarrow \text{temp.link};$

리스트의 마지막 노드의 링크 값을 **노드 new의 링크**에 저장한다. 노드 new는 노드 temp의 다음 노드인 첫 번째 노드와 연결된다.



②- d $\text{temp.link} \leftarrow \text{new};$

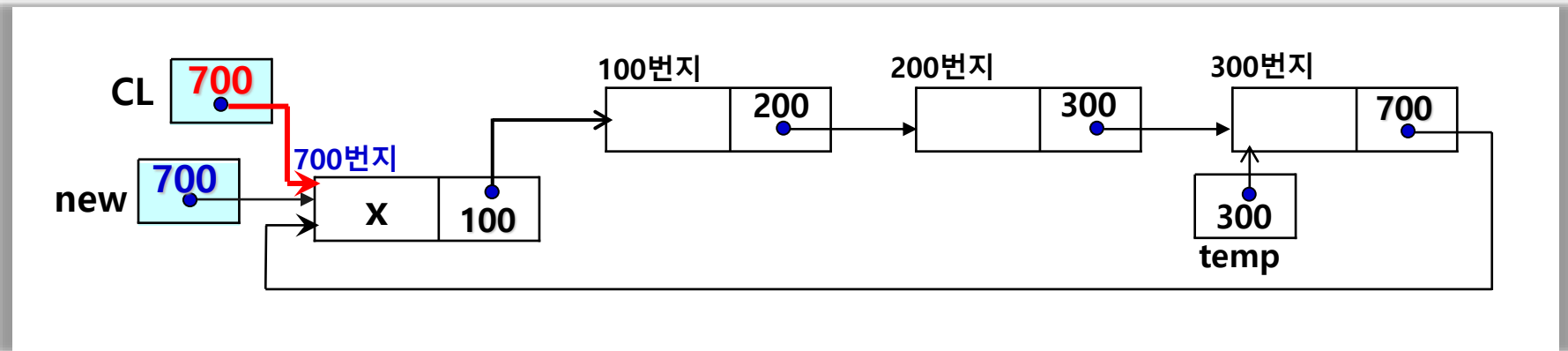
포인터 **new의 값**을 포인터 **temp**가 가리키고 있는 마지막 노드의 링크에 저장하여, 리스트의 마지막 노드가 노드 new를 가리키게 한다.



3. 원형 연결 리스트

②- e $CL \leftarrow new;$

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



3. 원형 연결 리스트

- 최종 결과



그림 4-22 원형 연결 리스트 CL이 공백이 아닌 경우에 첫 번째 노드로 삽입한 최종 결과



3. 원형 연결 리스트

- 리스트 중간에 노드를 삽입하는 알고리즘

알고리즘 4-7 원형 연결 리스트의 중간 노드 삽입

```
insertMiddleNode(CL, pre, x)
    new ← getNode();
    new.data ← x;
    {
        ① if (CL = NULL) then {
            CL ← new;
            new.link ← new;
        }
        {
            ② else {
                ②-a new.link ← pre.link;
                ②-b pre.link ← new;
            }
        }
    }
end insertMiddleNode()
```



3. 원형 연결 리스트

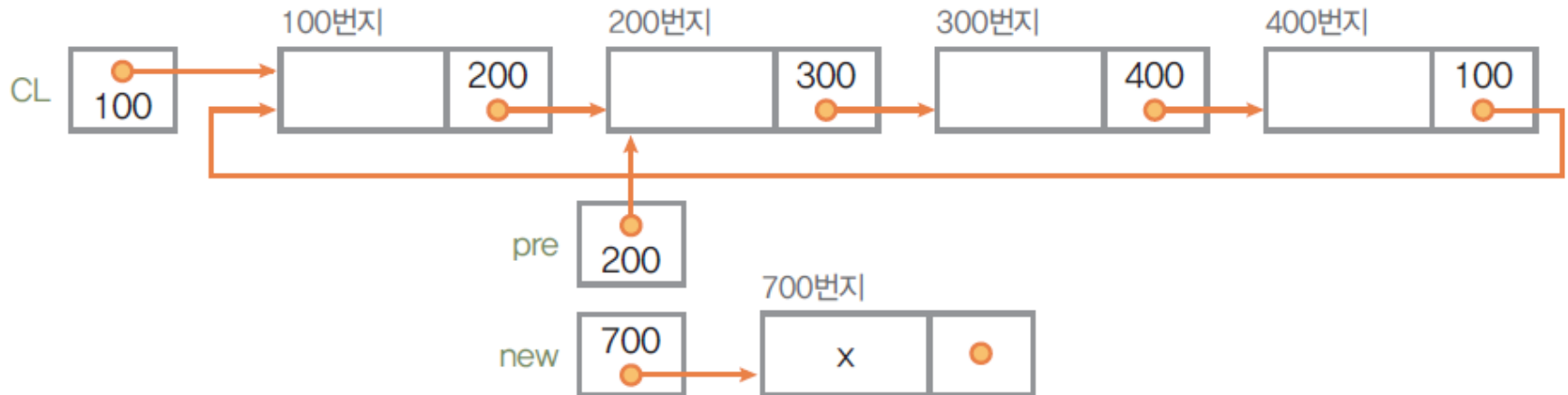
- 원형 연결 리스트 중간에 노드를 삽입하는 과정

① 공백 리스트인 경우

CL NULL



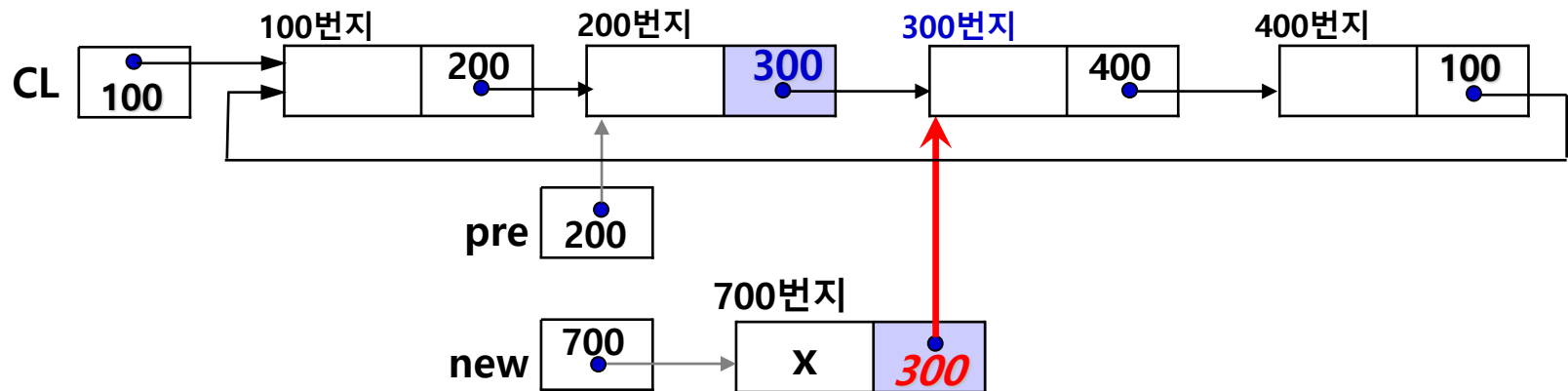
② 공백 리스트가 아닌 경우



3. 원형 연결 리스트

②- a new.link ← pre.link;

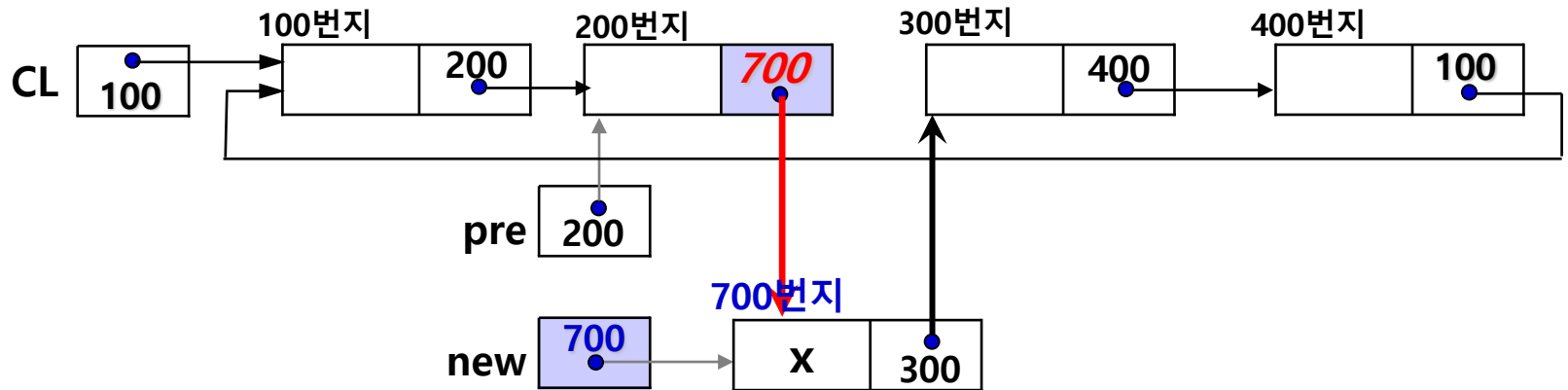
노드 pre의 다음 노드로 new를 삽입하기 위해서, 먼저 **노드 pre의 다음 노드 (pre.link)를 new의 다음 노드(new.link)로 연결**



3. 원형 연결 리스트

②- b $\text{pre.link} \leftarrow \text{new};$

노드 **new**의 값(삽입할 노드의 주소)을 노드 **pre**의 링크에 저장하여, 노드 **pre**가 노드 **new**를 가리키도록 한다.



3. 원형 연결 리스트

- 최종 결과



그림 4-23 원형 연결 리스트 CL이 공백인 아닌 경우에 중간 노드로 삽입한 최종 결과



3. 원형 연결 리스트

■ 노드 삭제 알고리즘

알고리즘 4-8 원형 연결 리스트의 노드 삭제

```
deleteNode(CL, pre)
  if (CL = NULL) then error;
  else {
    ❶ old ← pre.link;
    ❷ pre.link ← old.link;
    ❸ { if (old = CL) then
        ❸-a CL ← old.link;
    ❹ returnNode(old);
  }
end deleteNode()
```



3. 원형 연결 리스트

- 원형 연결 리스트에서 노드를 삭제하는 과정
 - 초기상태

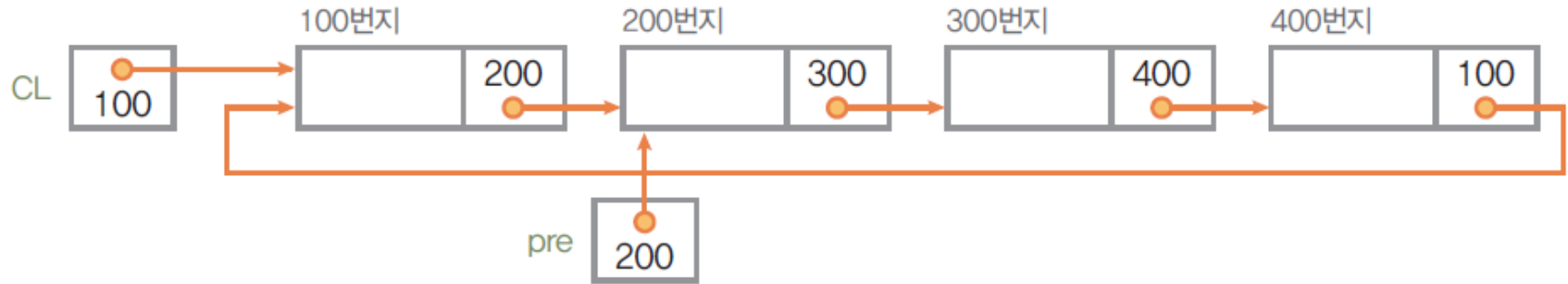


그림 4-24 초기 상태

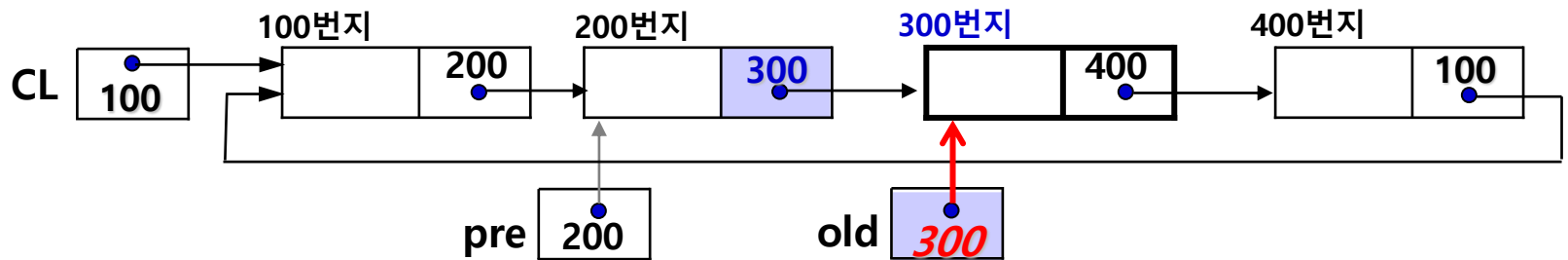


3. 원형 연결 리스트

- 원형 연결 리스트에서 노드를 삭제하는 과정

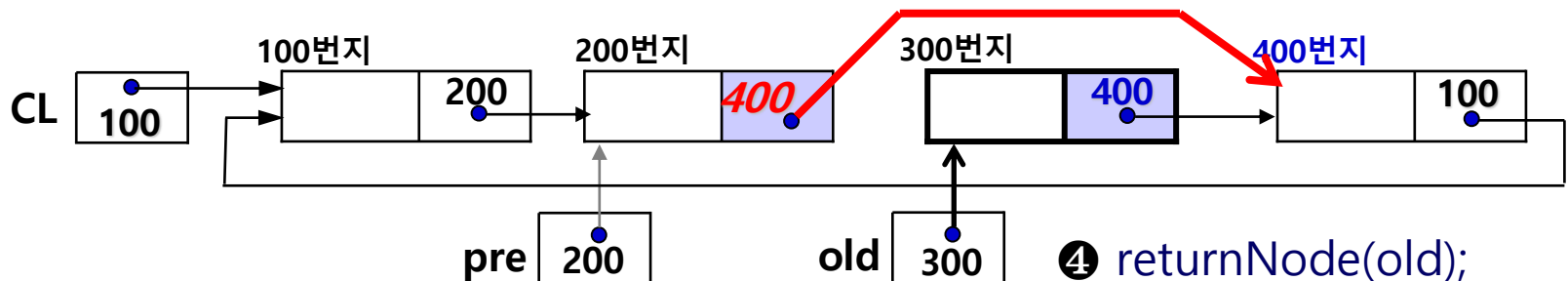
❶ $\text{old} \leftarrow \text{pre.link};$

노드 **pre**의 다음노드(**pre.link**)를 삭제할 노드 **old**로 지정



❷ $\text{pre.link} \leftarrow \text{old.link};$

old의 다음노드(**old.link**)를 노드 **old**의 이전노드 **pre** 노드와 서로 연결

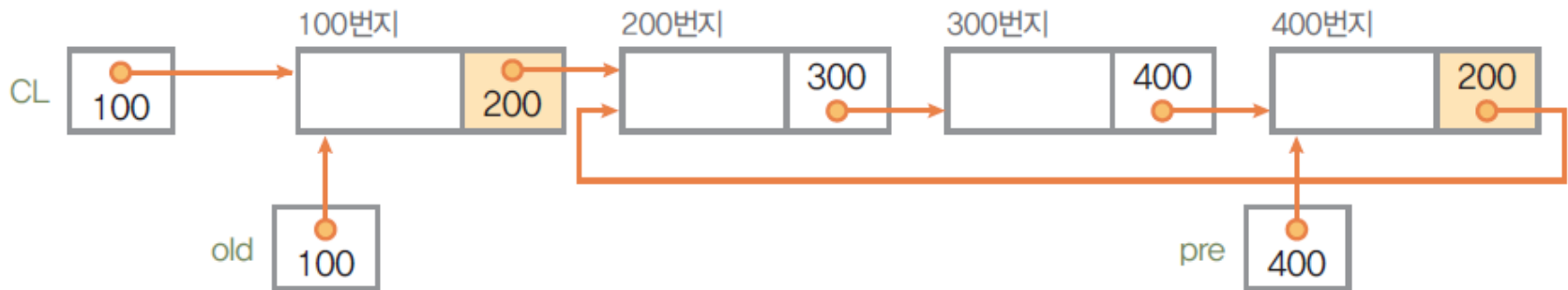


❸ $\text{returnNode}(\text{old});$

3. 원형 연결 리스트

③ if (old = CL) then

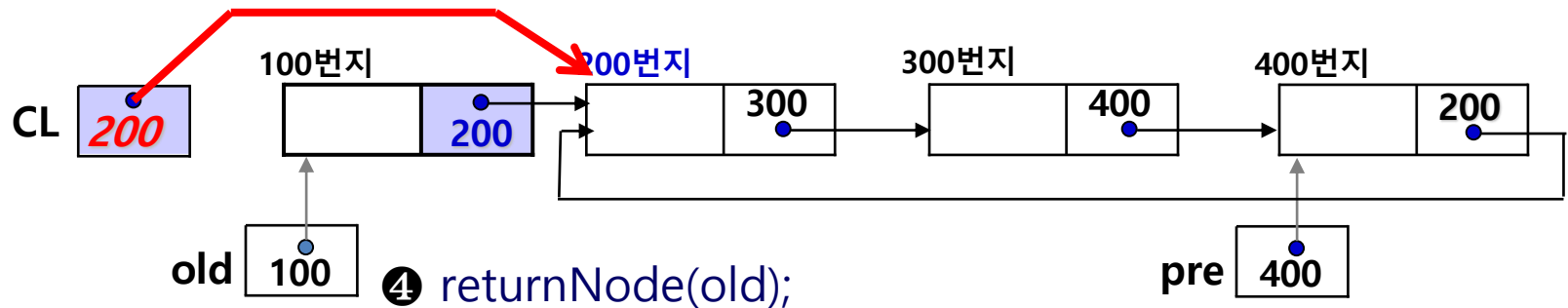
삭제할 노드 old가 원형 연결 리스트의 첫번째 노드인 경우 >>



3. 원형 연결 리스트

③ - a CL \leftarrow old.link;

첫 번째 노드를 삭제하는 경우에는 **노드 old의 링크 값을 리스트 포인터 CL에** 저장하여 두 번째 노드가 리스트의 첫 번째 노드가 되도록 조정



④ returnNode(old);

삭제한 노드 old의 메모리를 반환한다.



3. 원형 연결 리스트

- 최종 결과

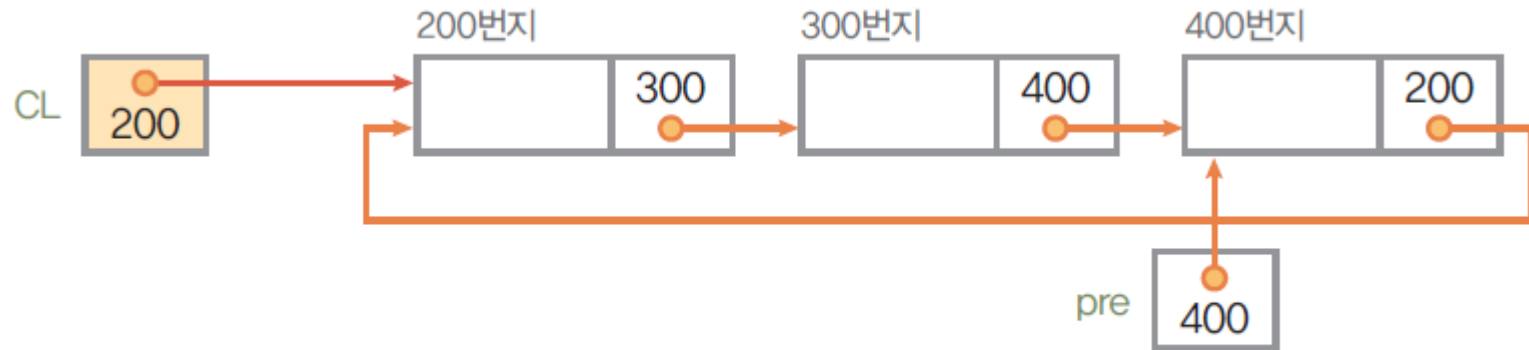
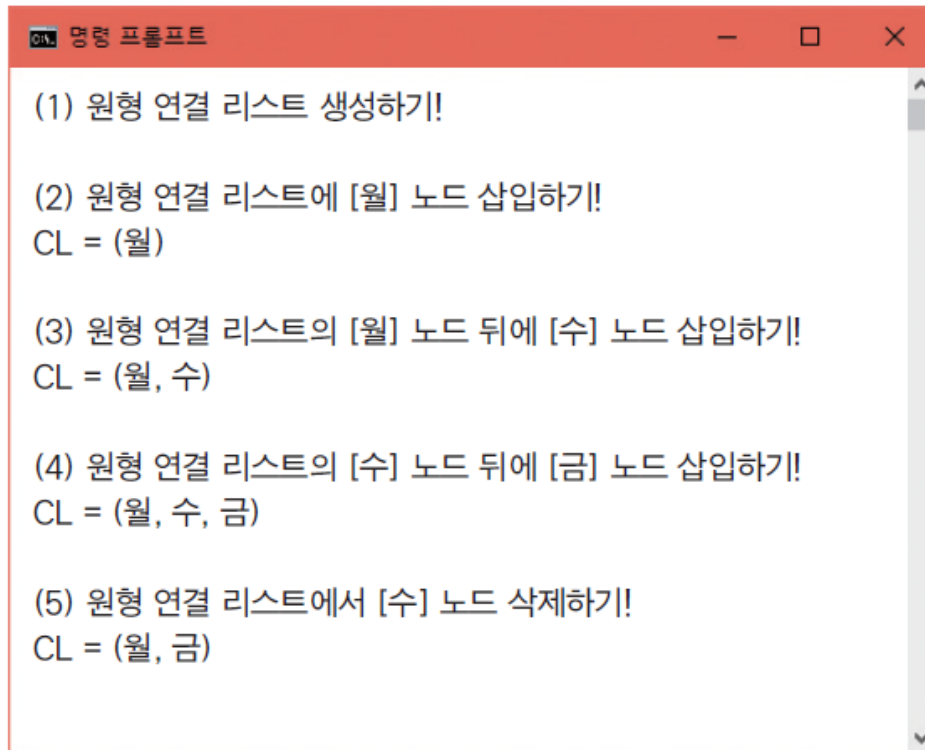


그림 4-25 원형 연결 리스트에서 노드를 삭제한 최종 결과



3. 원형 연결 리스트

- 원형 연결 리스트 프로그램 : [교재 190p](#)
- 실행 결과



```
명령 프롬프트

(1) 원형 연결 리스트 생성하기!

(2) 원형 연결 리스트에 [월] 노드 삽입하기!
CL = (월)

(3) 원형 연결 리스트의 [월] 노드 뒤에 [수] 노드 삽입하기!
CL = (월, 수)

(4) 원형 연결 리스트의 [수] 노드 뒤에 [금] 노드 삽입하기!
CL = (월, 수, 금)

(5) 원형 연결 리스트에서 [수] 노드 삭제하기!
CL = (월, 금)
```



4. 이중 연결 리스트

❖ 이중 연결 리스트의 개념

- 양쪽 방향으로 순회할 수 있도록 노드를 연결한 리스트
- 이중 연결 리스트의 노드 구조와 구조체 정의
 - llink(left link) 필드 : 왼쪽노드와 연결하는 포인터
 - rlink(right link) 필드 : 오른쪽 노드와 연결하는 포인터



(a) 노드 구조

```
typedef struct Dnode {  
    struct Dnode *llink;  
    char data[5];  
    struct Dnode *rlink;  
}
```

(b) 구조체 정의

그림 4-26 이중 연결 리스트의 노드 구조와 구조체 정의

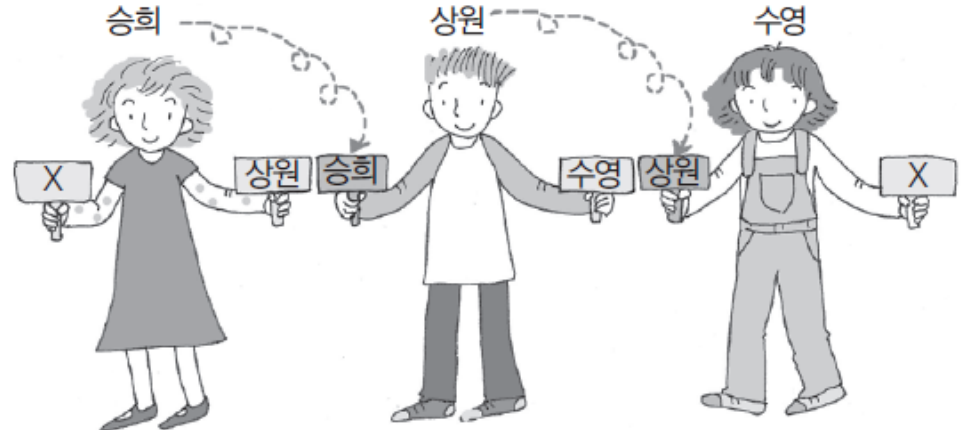


4. 이중 연결 리스트

■ 단방향과 양방향 기차놀이



(a) 단방향 기차놀이



(b) 양방향 기차놀이

그림 4-27 단방향과 양방향 기차놀이

- (b)의 양방향 기차를 이중 연결 리스트라고 생각하면 아이들은 노드
- 왼손의 이름표는 llink, 오른손의 이름표는 rlink



4. 이중 연결 리스트

- 리스트 week=(월, 수, 금)의 이중 연결 리스트 구성



그림 4-28 리스트 week의 이중 연결 리스트

- 원형 이중 연결 리스트
 - 이중 연결 리스트를 원형으로 구성



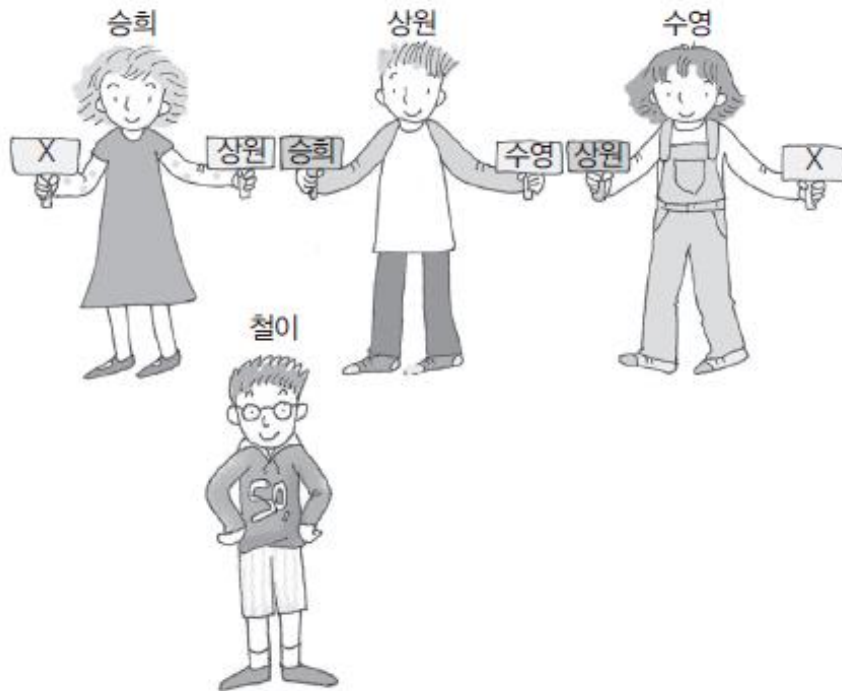
그림 4-29 리스트 week의 이중 원형 연결 리스트



4. 이중 연결 리스트

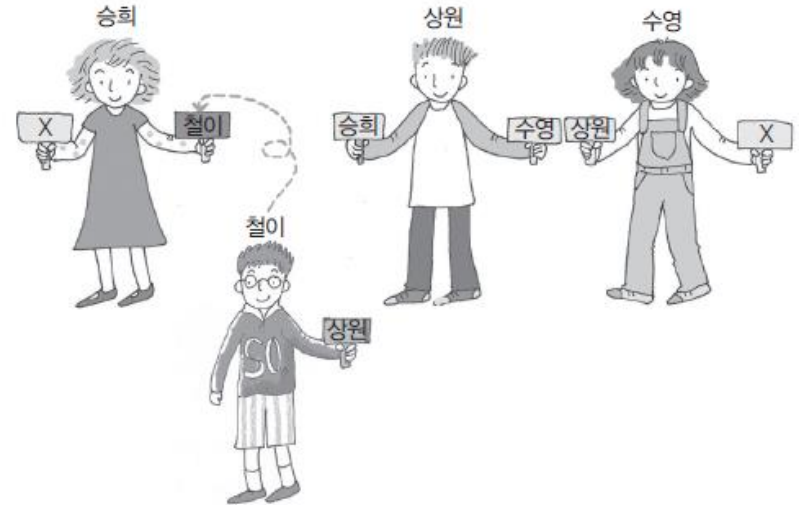
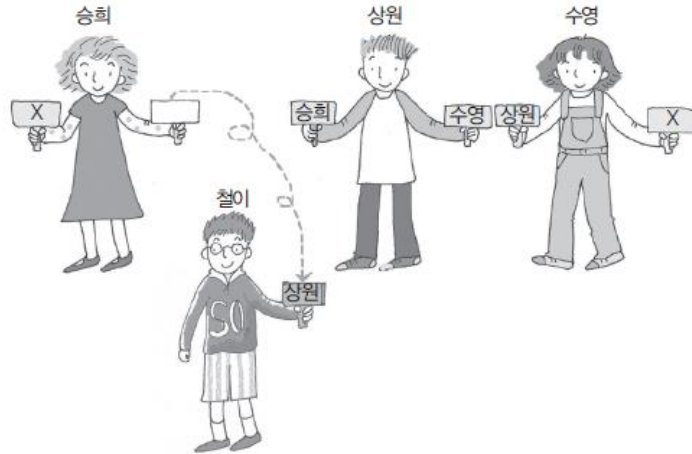
- 이중 연결 리스트에 노드를 삽입하는 과정과 알고리즘

1 철이를 불러내 승희와 상원이 사이에 끼우기 전의 상태이다.

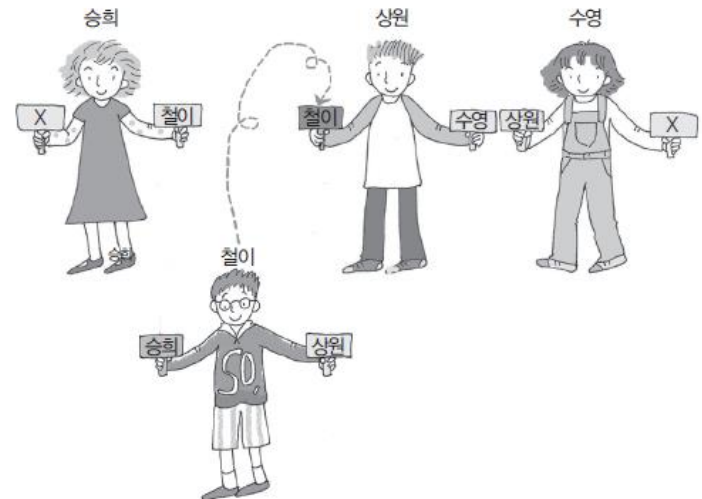
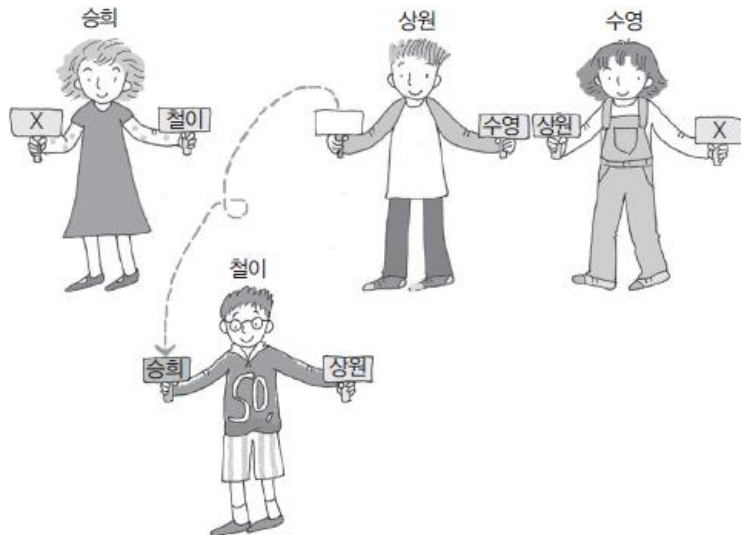


4. 이중 연결 리스트

2 철이 왼쪽에 설 승희는 오른손에 쥐고 있던 '상원' 이름표를 철이 오른손에 준다. 3 대신 철이는 '철이' 이름표를 승희에게 준다.

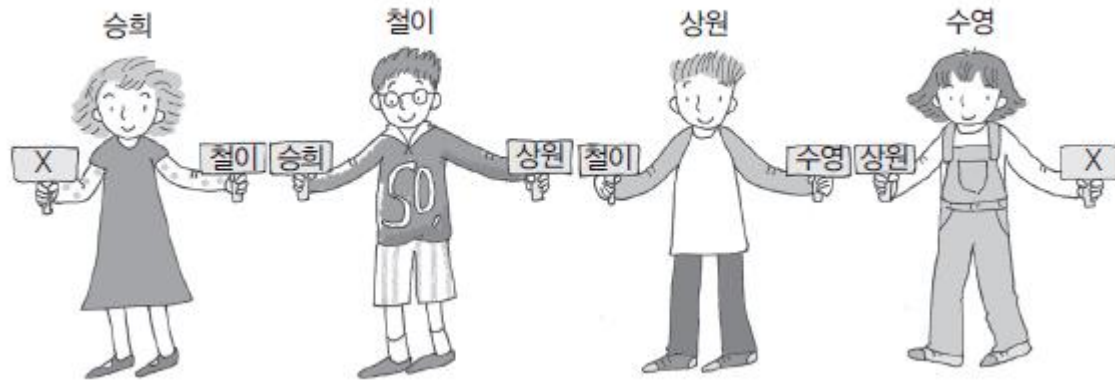


4 철이 오른쪽에 설 상원이는 왼손에 쥐고 있던 '승희' 이름표를 철이 왼손에 준다. 5 대신 철이는 '철이' 이름표를 상원에게 준다.



4. 이중 연결 리스트

6 이름표대로 연결하면 다음과 같은 양방향 기차가 완성된다.



4. 이중 연결 리스트

■ 이중 연결 리스트에서 노드를 삽입하는 방법

- 1 삽입할 노드를 준비한다.
- 2 새 노드의 데이터 필드에 값을 저장한다.
- 3 새 노드 왼쪽 노드의 오른쪽 링크 필드(rlink)에 있던 값을 새 노드의 오른쪽 링크 필드(rlink)에 저장한다.
- 4 왼쪽 노드의 오른쪽 링크 필드(rlink)에 새 노드의 주소를 저장한다.
- 5 새 노드 오른쪽 노드의 왼쪽 링크 필드(llink)에 있던 값을 새 노드의 왼쪽 링크 필드(llink)에 저장한다.
- 6 오른쪽 노드의 왼쪽 링크 필드(llink)에 새 노드의 주소를 저장한다.
- 7 노드를 순서대로 연결한다.



4. 이중 연결 리스트

- 이중 연결 리스트에 원소를 삽입하는 알고리즘

알고리즘 4-9 이중 연결 리스트의 노드 삽입

```
insertNode(DL, pre, x)
    new ← getNode();
    new.data ← x;
    ① new.rlink ← pre.rlink;
    ② pre.rlink ← new;
    ③ new.llink ← pre;
    ④ new.rlink.llink ← new;
end insertNode()
```



4. 이중 연결 리스트

- 이중 연결 리스트에 노드를 삽입하는 과정
 - 데이터 필드값이 x인 노드 new를 준비, 포인터 pre가 가리키는 노드의 다음 노드로 삽입하려고 한다고 가정

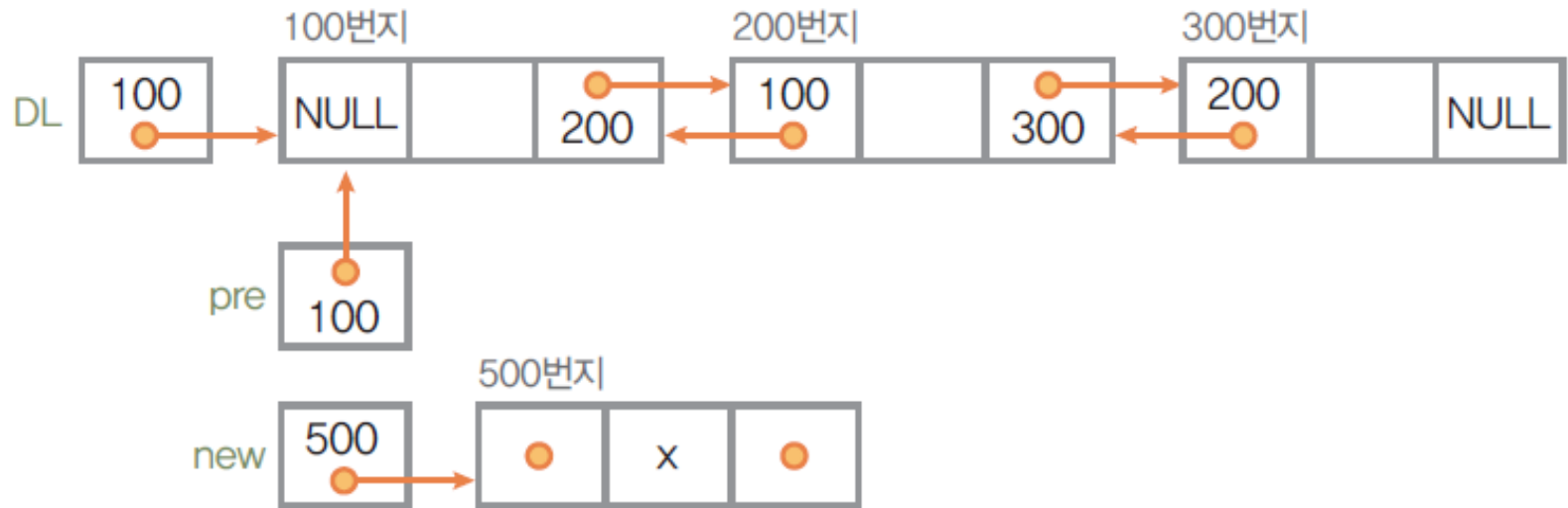
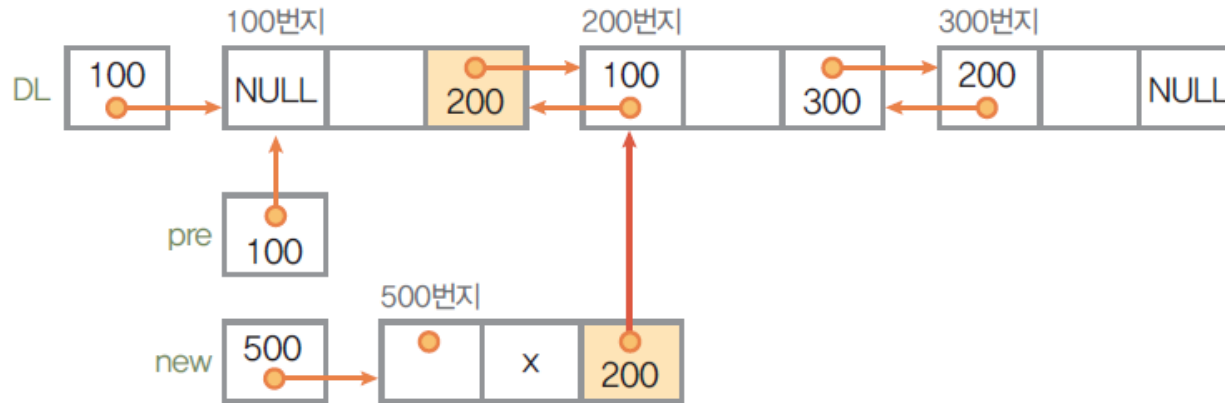


그림 4-31 초기 상태

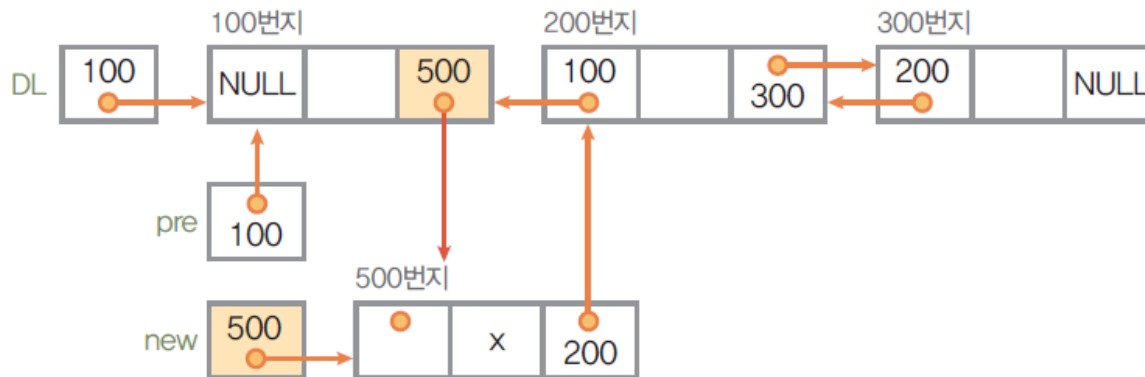


4. 이중 연결 리스트

① $\text{new.rlink} \leftarrow \text{pre.rlink};$

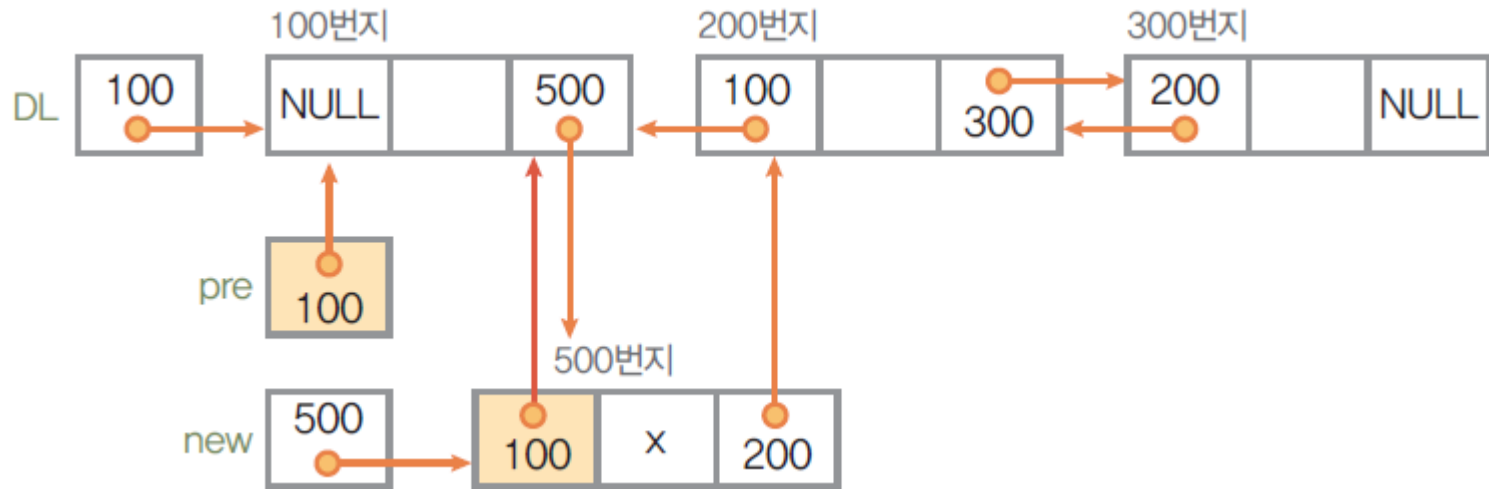


② $\text{pre.rlink} \leftarrow \text{new};$

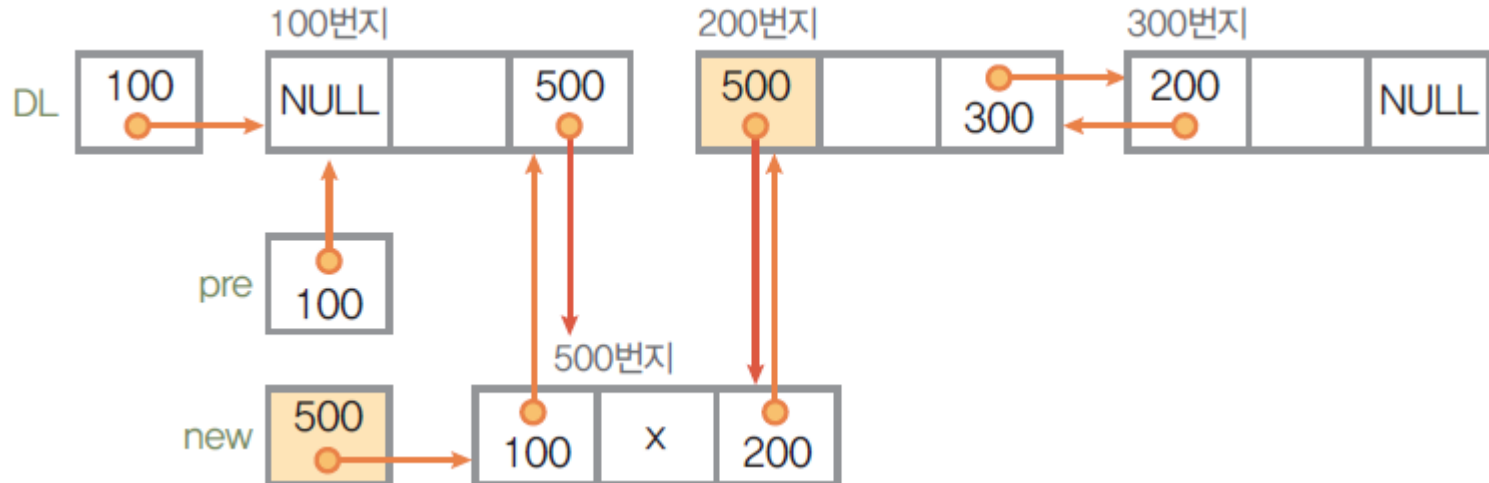


4. 이중 연결 리스트

③ `new.llink ← pre;`



④ `new.rlink.llink ← new;`



4. 이중 연결 리스트

- 최종 결과

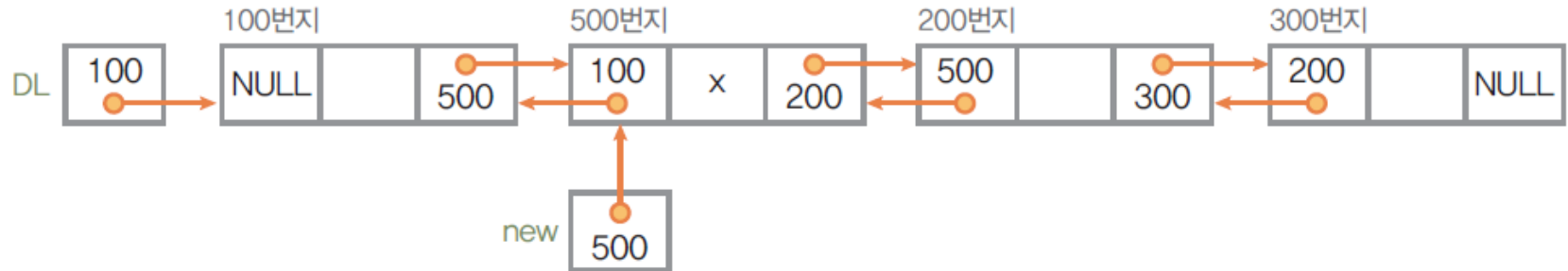


그림 4-32 이중 연결 리스트 DL에 new 노드가 삽입된 최종 결과



4. 이중 연결 리스트

- 이중 연결 리스트에 노드를 삭제하는 과정과 알고리즘
 - 이중 연결 리스트에서 노드를 삭제하는 방법

- 1 삭제할 노드의 오른쪽 노드와 왼쪽 노드를 찾는다.
- 2 삭제할 노드의 오른쪽 노드의 주소(old.rlink)를 삭제할 노드의 왼쪽 노드(old.llink)의 오른쪽 링크 필드(rlink)에 저장한다.
- 3 삭제할 노드의 왼쪽 노드(old.llink)의 주소를 삭제할 노드의 오른쪽 노드(old.rlink)의 왼쪽 링크 필드에 저장한다.
- 4 노드를 순서대로 연결한다.

그림 4-33 이중 연결 리스트에서 노드를 삭제하는 과정

- 이중 연결 리스트에서 노드를 삭제하는 방법을 알고리즘으로 정의

알고리즘 4-10 이중 연결 리스트의 노드 삭제

```
deleteNode(DL, old)
    ① old.llink.rlink ← old.rlink;
    ② old.rlink.llink ← old.llink;
    ③ returnNode(old);
end deleteNode()
```



4. 이중 연결 리스트

- 이중 연결 리스트에서 노드를 삭제하는 과정
 - 초기 상태

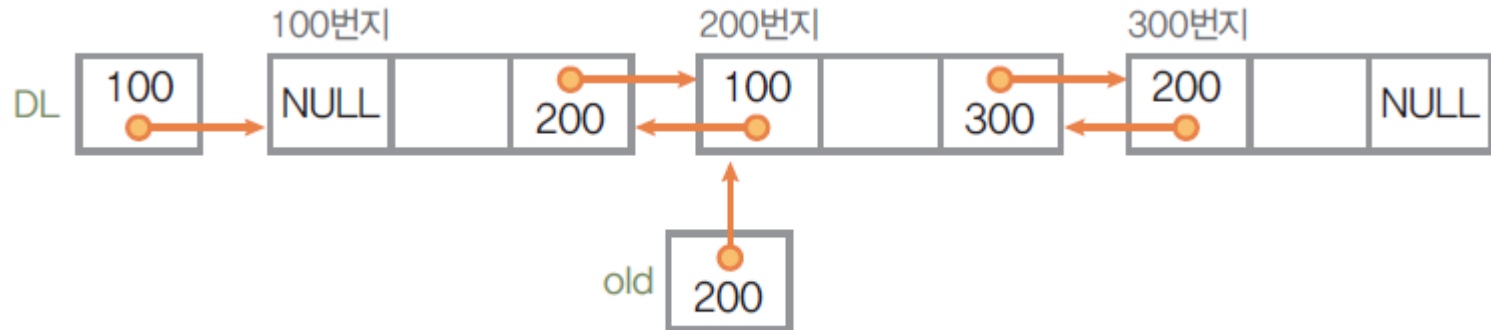
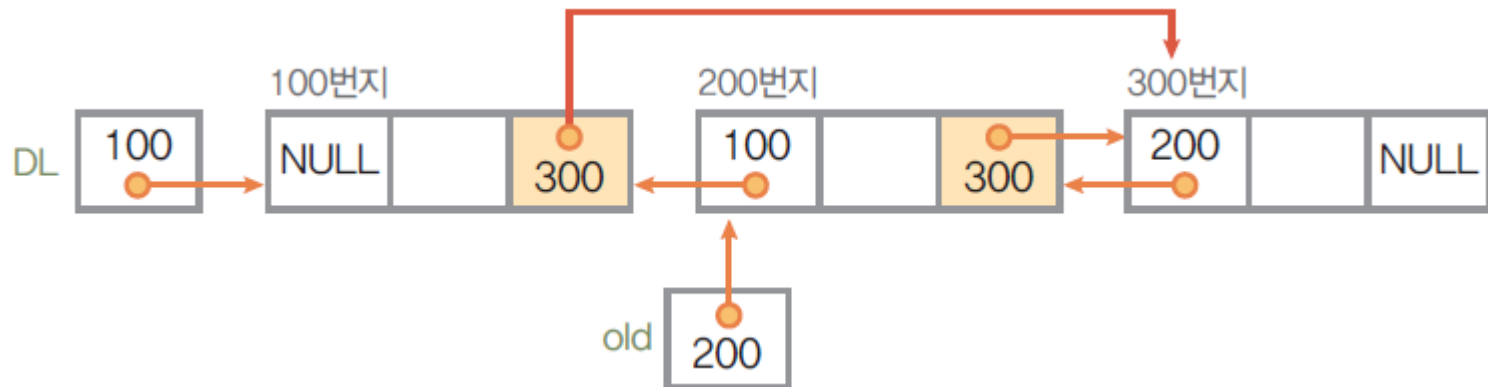


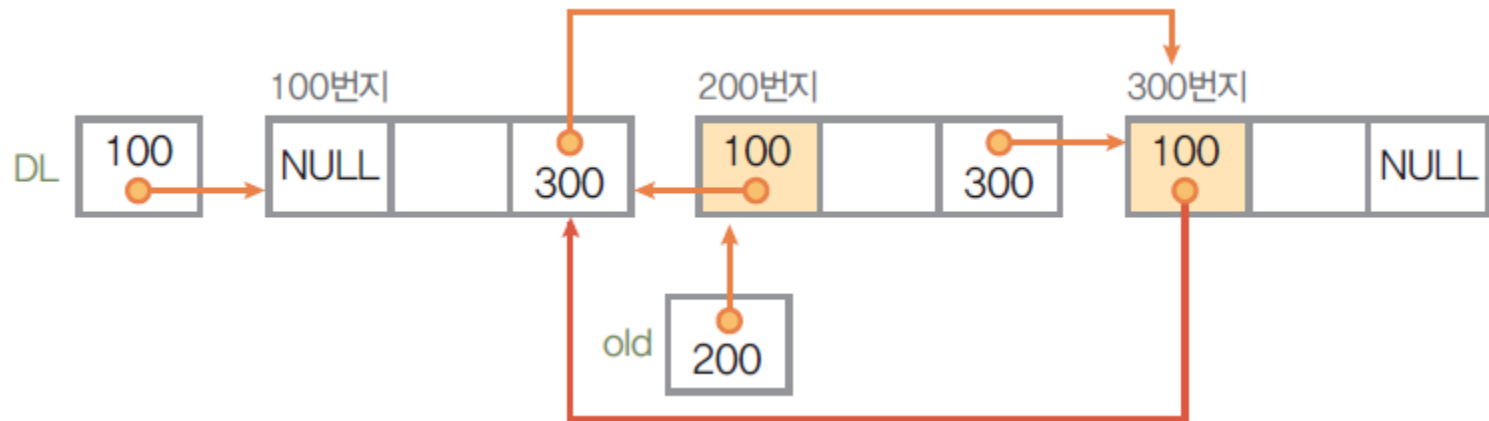
그림 4-34 초기 상태

❶ `old.llink.rlink ← old.rlink;`



4. 이중 연결 리스트

② `old.rlink.llink ← old.llink;`



③ `returnNode(old);`



4. 이중 연결 리스트

- 최종 결과



그림 4-35 이중 연결 리스트에서 old 노드를 삭제한 최종 결과



4. 이중 연결 리스트

- 이중 연결 리스트 프로그램 : [교재 203p](#)
- 실행 결과

```
C:\> 명령 프롬프트

(1) 이중 연결 리스트 생성하기!
DL = ()

(2) 이중 연결 리스트에 [월] 노드 삽입하기!
DL = (월)

(3) 이중 연결 리스트의 [월] 노드 뒤에 [수] 노드 삽입하기!
DL = (월, 수)

(4) 이중 연결 리스트의 [수] 노드 뒤에 [금] 노드 삽입하기!
DL = (월, 수, 금)

(5) 이중 연결 리스트에서 [수] 노드 삭제하기!
DL = (월, 금)
```





Thank You

