4 연결 자료구조와 연결 리스트

IT CookBook, C로 배우는 쉬운 자료구조(개정 3판)

한빛아카데미 Hanbit Academy, Inc.

1. 연결 자료구조와 연결 리스트의 이해

❖ 연결 리스트의 노드

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

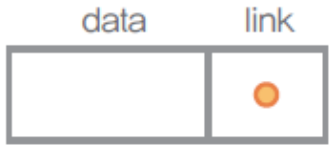


그림 4-2 노드의 논리적 구조

- 데이터 필드data field
  - 원소의 값을 저장
  - 저장할 원소의 형태에 따라서 하나 이상의 필드로 구성
- 링크 필드link 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

- 리스트 week의 노드에 대한 구조체 정의

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

그림 4-9 리스트 week 노드의 구조체

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

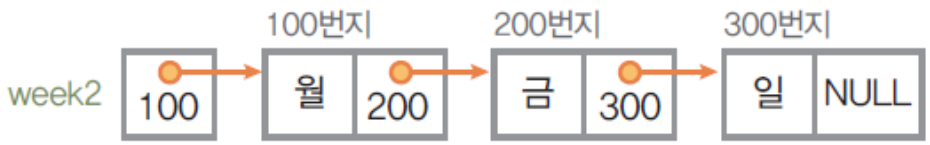
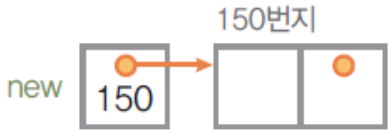
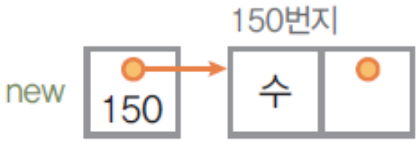


그림 4-12 단순 연결 리스트 week2에 노드를 삽입하기 전인 초기 상태

❶ **삽입할 노드 준비** : 공백 노드를 가져와 포인터 변수 new가 가리키게 함

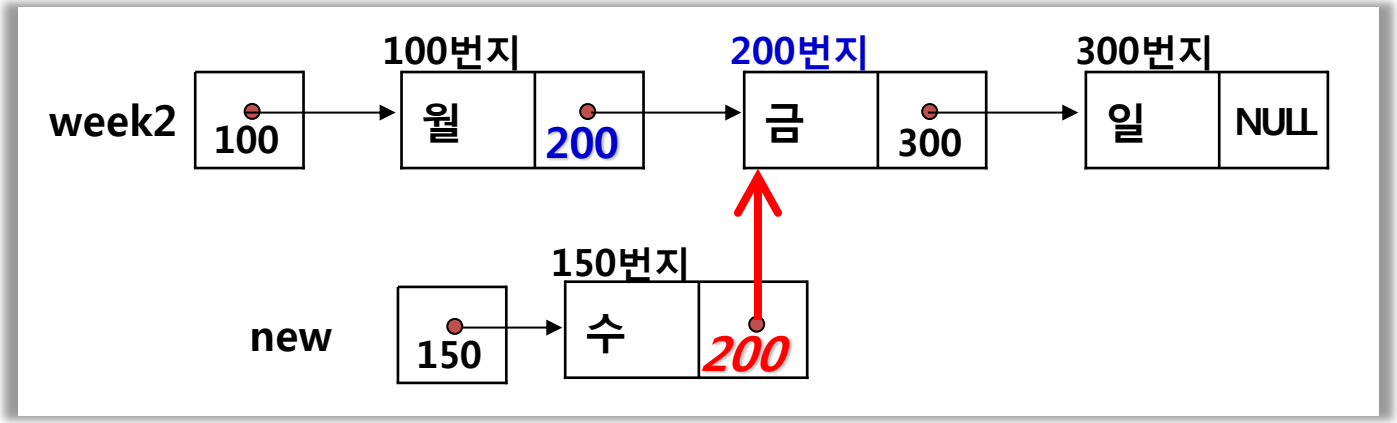


❷ **새 노드의 데이터 필드값 저장** : new의 데이터 필드에 "수"를 저장

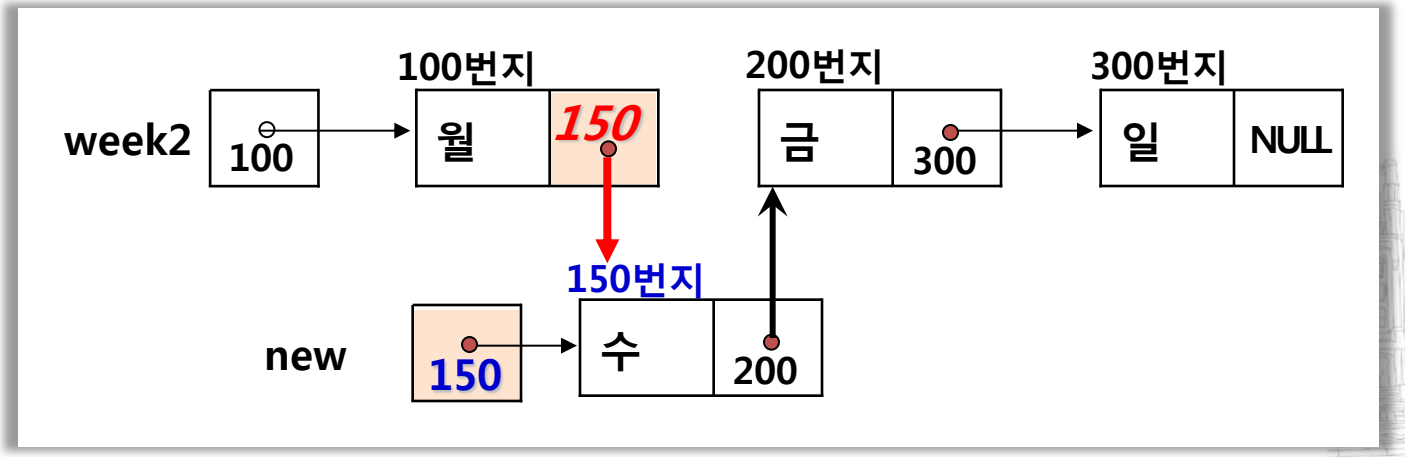


2. 단순 연결 리스트 : 삽입 연산

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



❹ **리스트의 앞 노드에 새 노드 연결** : new의 값을 "월"노드의 링크 필드에 저장



2. 단순 연결 리스트 : 삭제 연산

- 단순 연결 리스트에서 노드를 삭제하는 방법

1. 삭제할 노드의 앞 노드를 찾는다.
2. 앞 노드에 삭제할 노드의 링크 필드값을 저장한다.
3. 삭제한 노드의 앞 노드와 삭제한 노드의 다음 노드를 연결한다.

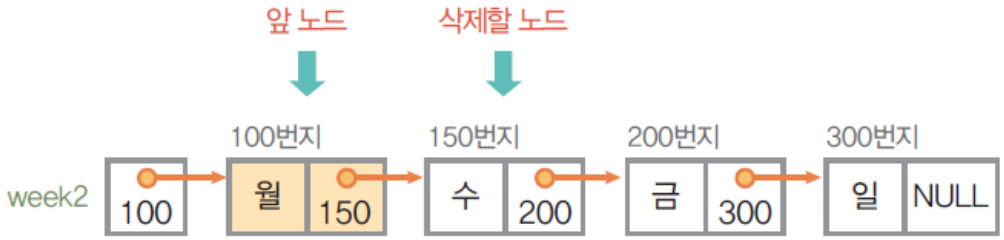
2. 단순 연결 리스트 : 삭제 연산

- 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

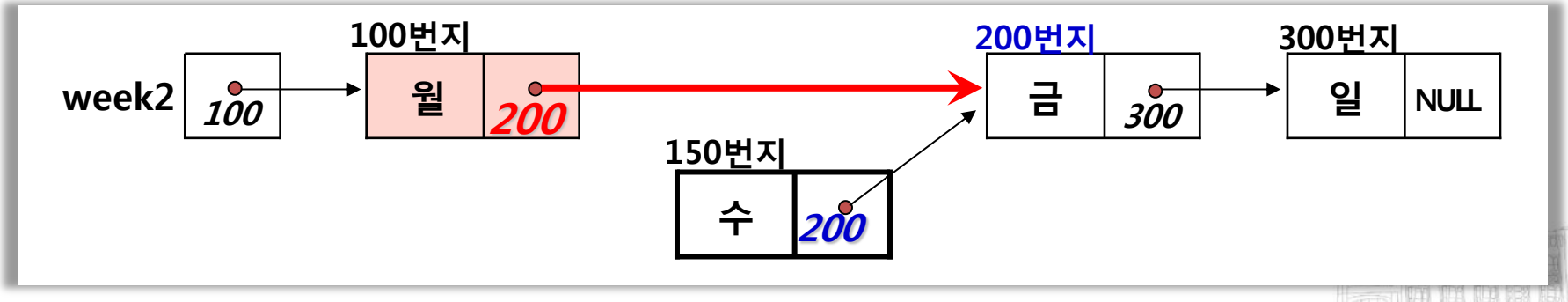
2. 단순 연결 리스트 : 삭제 연산

❶ 앞 노드를 찾음 : 삭제할 원소의 앞 노드(선행자)를 찾음



❷ 앞 노드에 삭제할 노드의 링크 필드값 저장 : 삭제할 원소 "수"의 링크 필드 값을 앞 노드의 링크 필드에 저장

❸ 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)
    ❶ new ← getNode();
    ❷ new.data ← x;
    ❸ new.link ← L;
    ❹ L ← new;
end insertFirstNode()
```

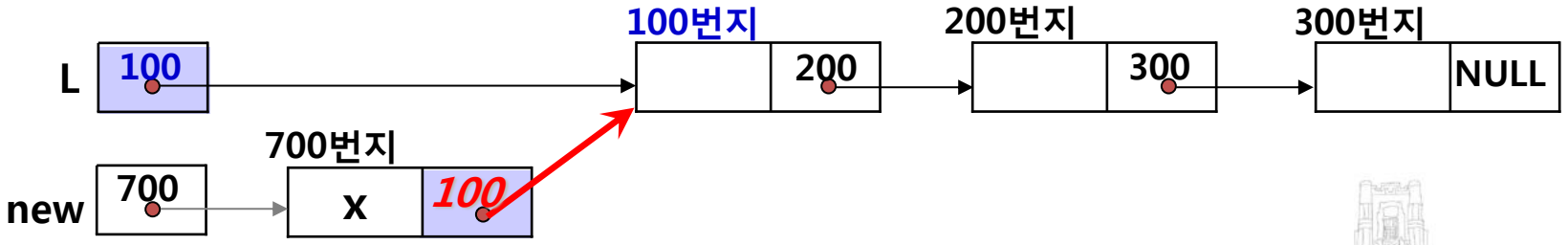
2. 단순 연결 리스트

❶ new ← getNode();

❷ new.data ← x;

❸ new.link ← L;

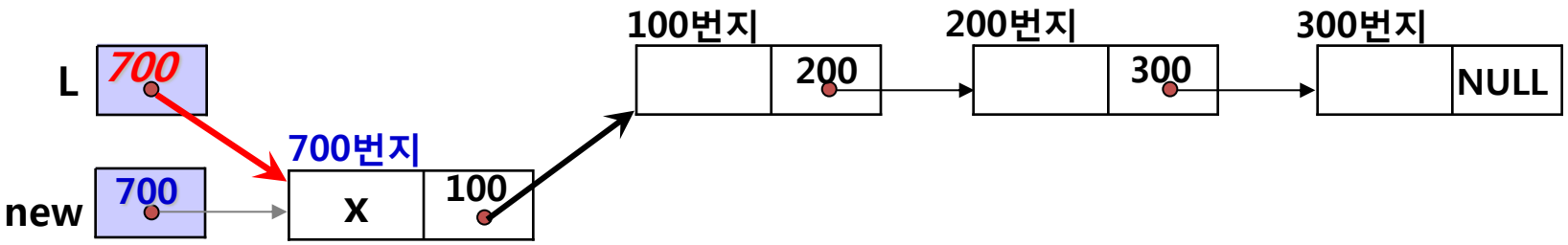
삽입할 노드를 연결하기 위해서 **리스트의 첫 번째 노드 주소(L)를** 삽입할 새 노드 **new의 링크 필드(new.link)에** 저장하여, 새 노드 new가 리스트의 첫 번째 노드를 가리키게 한다.



2. 단순 연결 리스트

❹ L ← new;

리스트의 첫 번째 노드 주소를 저장하고 있는 **포인터 L에**, **새 노드의 주소 new를** 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

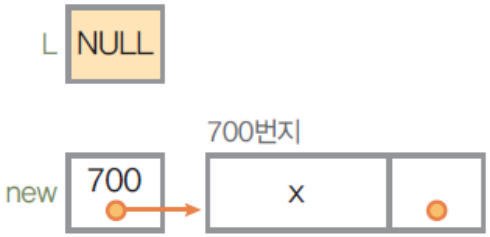
```
insertMiddleNode(L, pre, x)
    ❶ new ← getNode();
    ❷ new.data ← x;
    ❸ if (L = NULL) then {
          ❸-ⓐ L ← new;
          ❸-ⓑ new.link ← NULL;
      }
    ❹ else {
          ❹-ⓐ new.link ← pre.link;
          ❹-ⓑ pre.link ← new;
}
end insertMiddleNode()
```

2. 단순 연결 리스트

❶ new ← getNode();

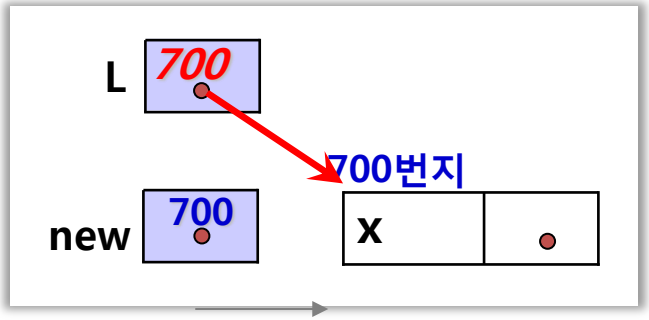
❷ new.data ← x;

❸ **공백 리스트인 경우**



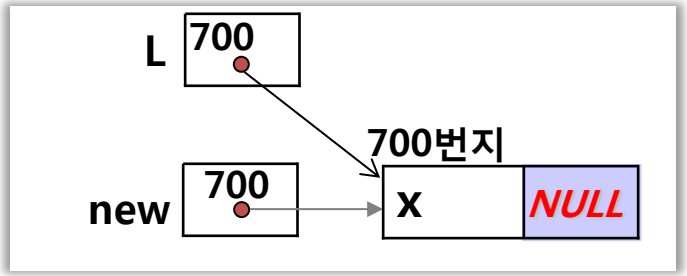
❸- a L ← new;

리스트 포인터 L에 새 노드 new의 주소를 저장하여, 새 노드 new가 리스트의 첫 번째 노드가 되도록 함



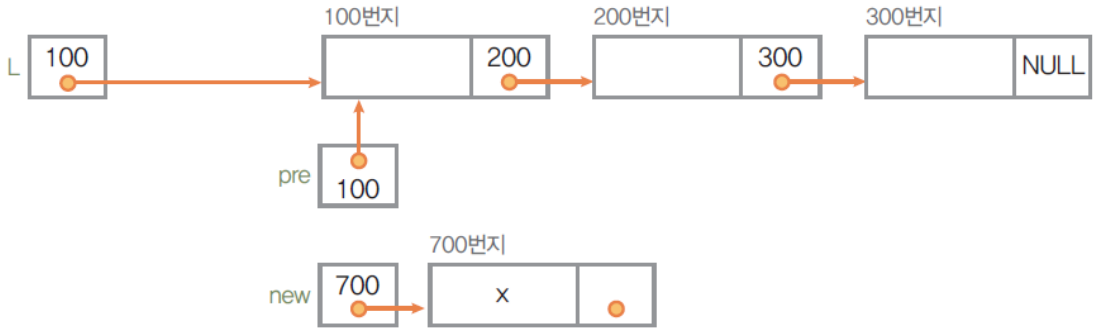
❸- b new.link ← NULL;

리스트의 마지막 노드인 new의 링크 필드에 NULL을 저장해 마지막 노드임을 표시



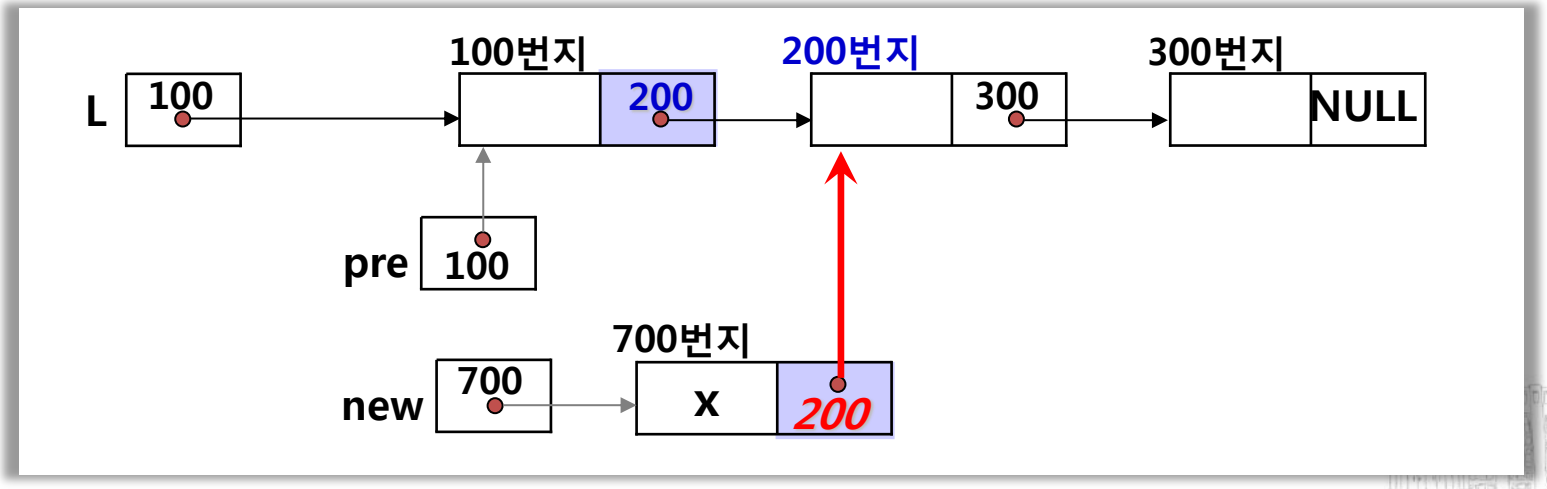
2. 단순 연결 리스트

❹ 공백 리스트가 아닌 경우



❹- a new.link ← pre.link;

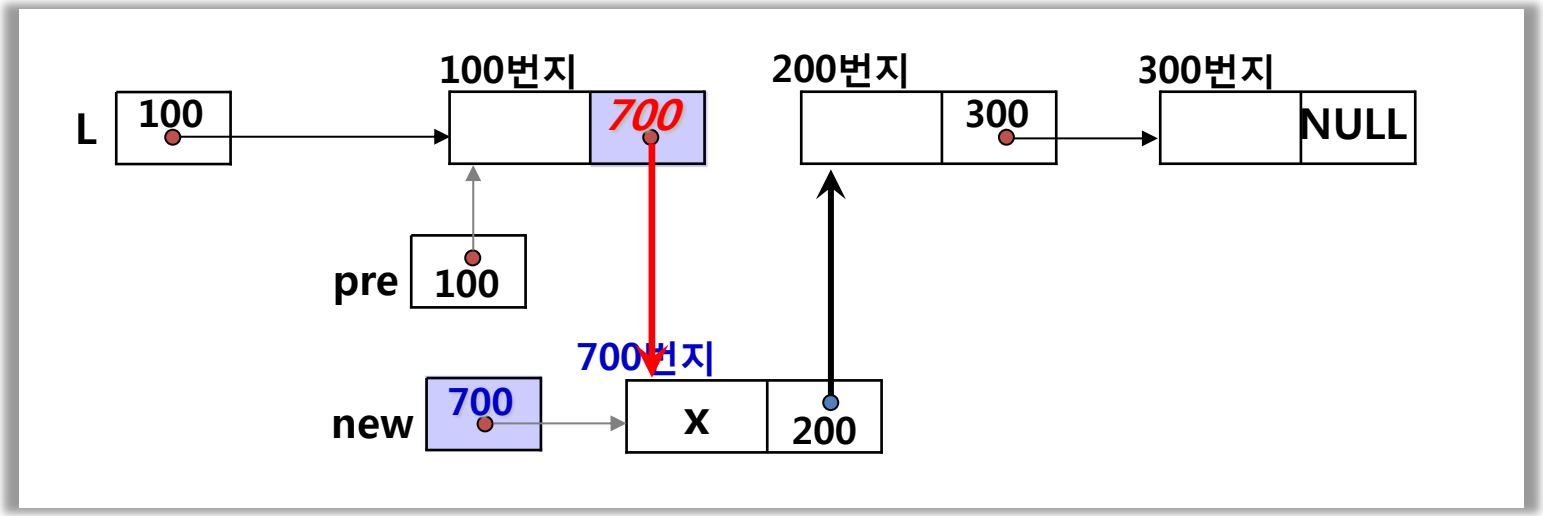
노드 pre의 링크 필드 값을 노드 new의 링크 필드에 저장하여, 새 노드 new가 노드 pre의 다음 노드를 가리키도록 함



2. 단순 연결 리스트

❹- b pre.link ← new;

포인터 new의 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)
    ❶ new ← getNode();
    ❷ new.data ← x;
    ❸ new.link ← NULL;
    ❹ if (L = NULL) then {
          ❹-ⓐ L ← new;
              return;
      }
    ❺ ❺-ⓐ temp ← L;
          while (temp.link ≠ NULL) do
              ❺-ⓑ temp ← temp.link;
      ❺-ⓒ temp.link ← new;
end insertLastNode()
```

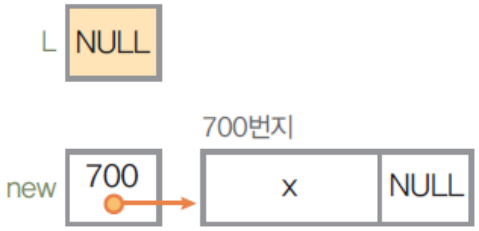
2. 단순 연결 리스트

❶ new ← getNode();

❷ new.data ← x;

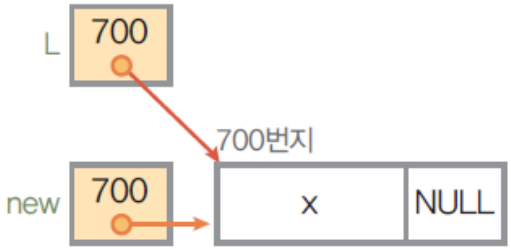
❸ new.link ← NULL;

❹ **공백 리스트인 경우**



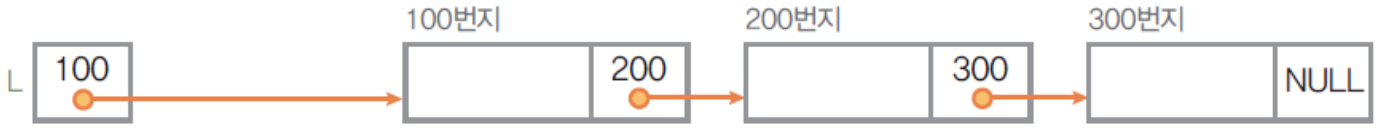
❹- a L ← new;

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



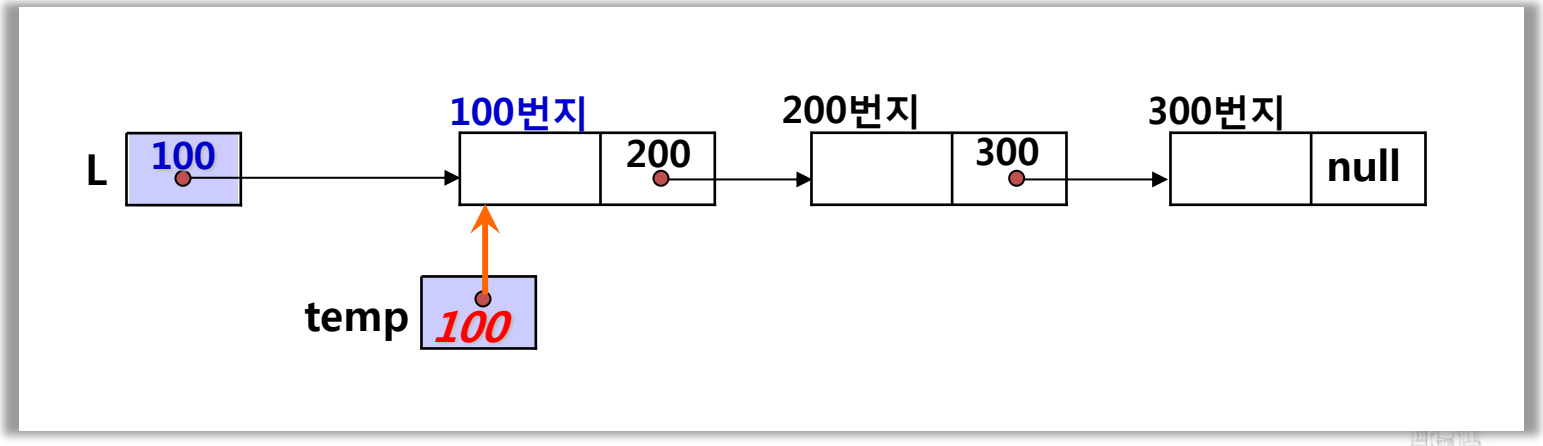
2. 단순 연결 리스트

❺ 공백 리스트가 아닌 경우



❺- a temp ← L;

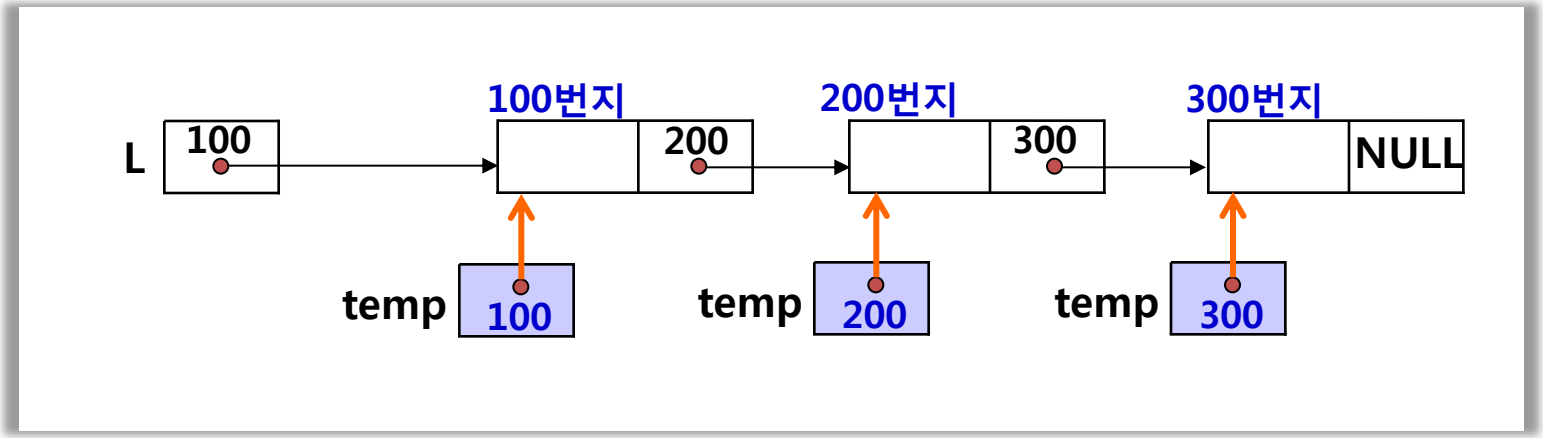
현재 리스트 L의 마지막 노드를 찾기 위해서 노드를 **순회할 임시포인터 temp에** 리스트의 **첫 번째 노드의 주소(L)를** 지정



2. 단순 연결 리스트

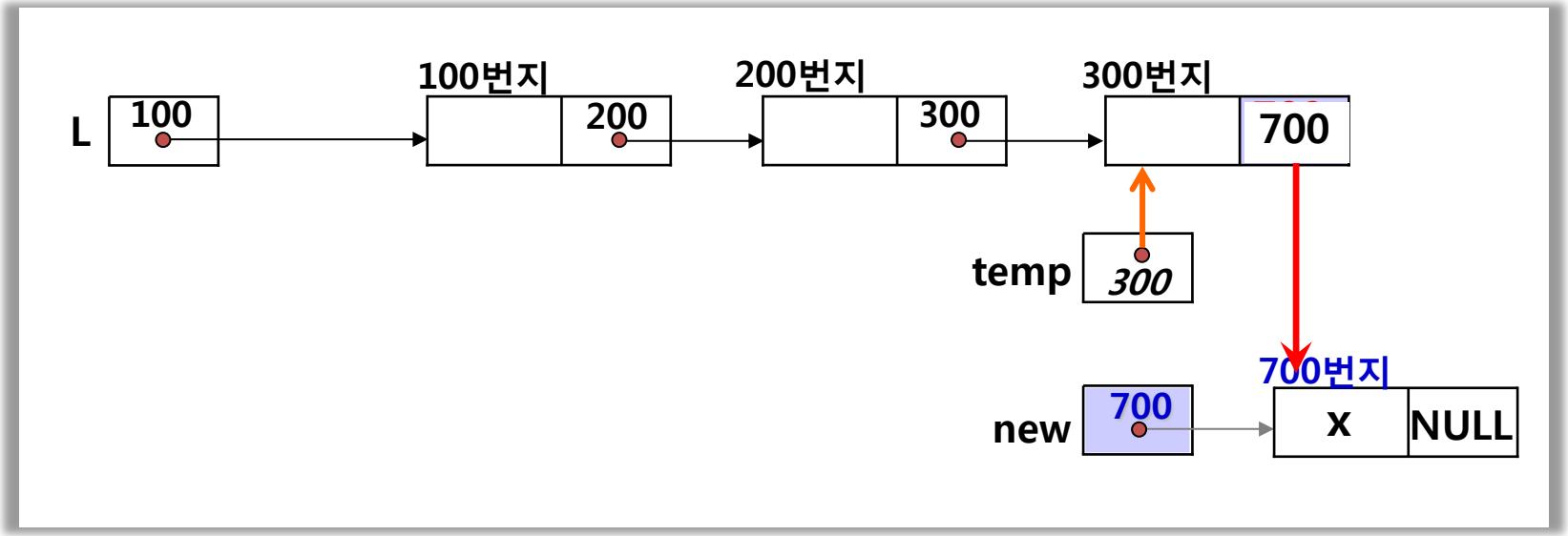
❺- b temp ← temp.link;

순회 포인터 temp가 가리키는 노드의 링크 필드가 NULL이 아닌 동안(while (temp.link ≠ NULL)) 링크 필드를 따라 이동. 링크 필드가 NULL인 노드 즉, 마지막 노드를 찾으면 while 문을 끝내고 ❺- c를 수행



2. 단순 연결 리스트

❺- c temp.link ← new;



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

2. 단순 연결 리스트

- 삭제 연산 수행 과정

  ❶ 공백 리스트인 경우

  공백 연결 리스트 L이 공백이면 오류 처리한다.

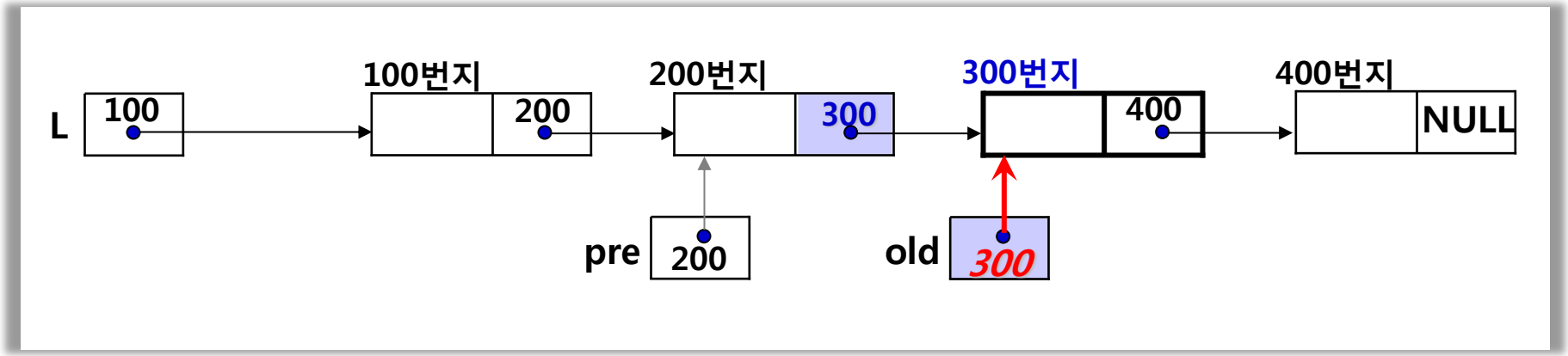
2. 단순 연결 리스트

- 삭제 연산 수행 과정

❷ 공백 리스트가 아닌 경우

❷- a old ← pre.link;

노드 pre의 다음노드의 주소(pre.link)를 포인터 old에 저장하여, 포인터 old가 다음 노드를 가리키게 한다.

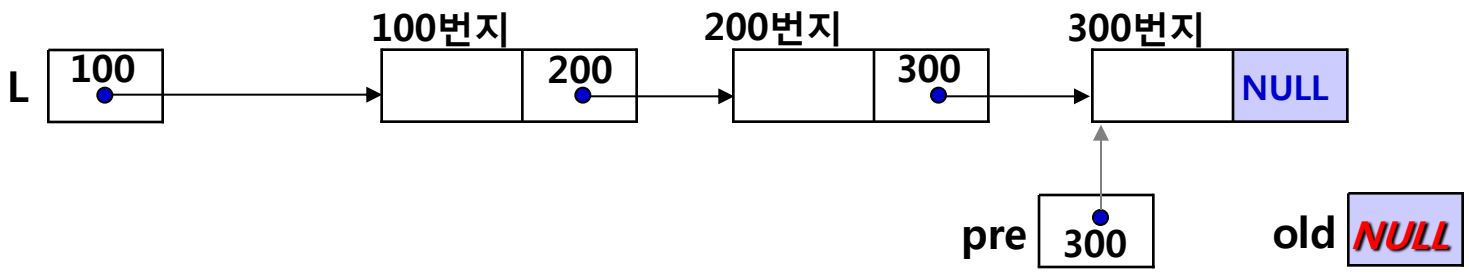


2. 단순 연결 리스트

❷- b if (old = NULL) then return;

만약 노드 pre가 리스트의 마지막 노드였다면 :

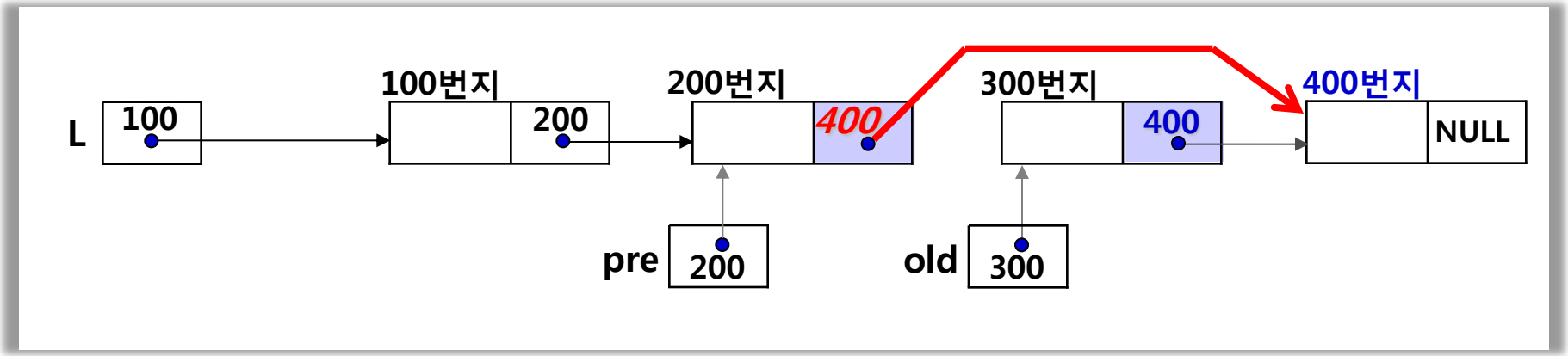
포인터 pre가 가리키는 노드가 마지막 노드인 경우에,
②-ⓐ old ← pre.link; 를 수행한 상태 :



⇒ 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

❷- c pre.link ← old.link;



❷- d returnNode(old);

삭제한 노드 old의 메모리를 반환

2. 단순 연결 리스트

❖ 노드를 탐색하는 알고리즘과 프로그램

- 노드를 탐색하는 알고리즘

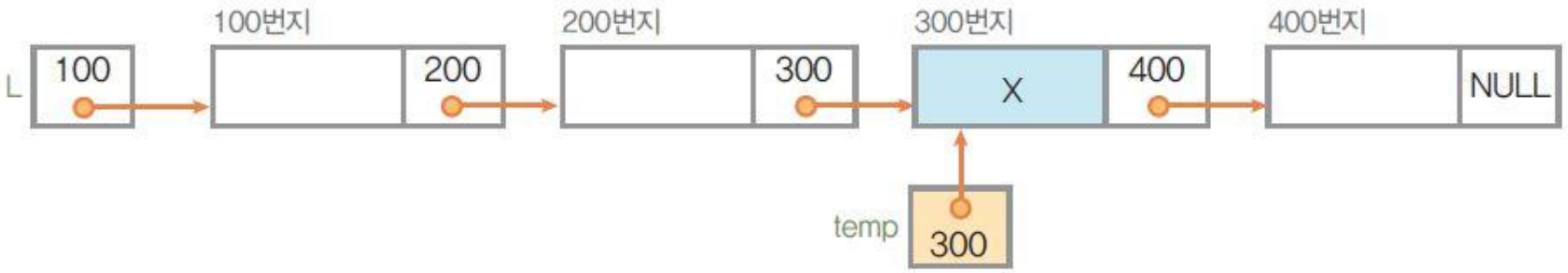
알고리즘 4-5 단순 연결 리스트의 노드 탐색

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

2. 단순 연결 리스트

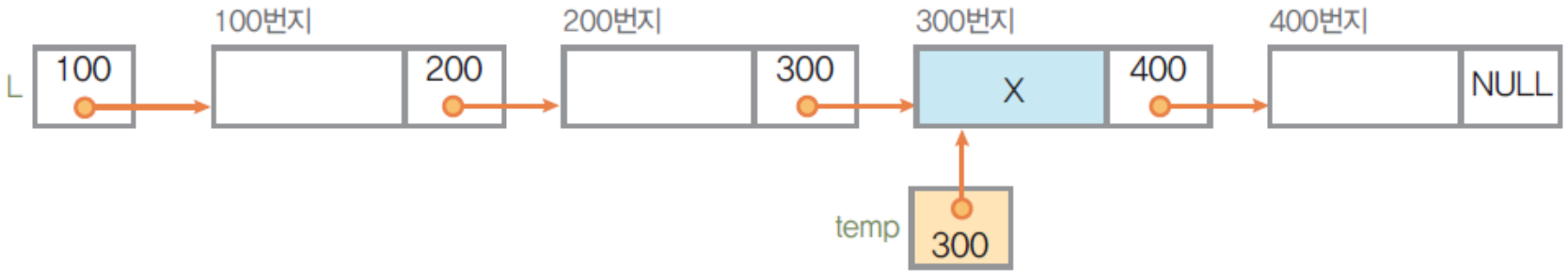
- 단순 연결 리스트에서 x 노드 탐색 과정

❶ temp ← L;



❷ 순회 포인터가 NULL이 아닌 경우

❷- a if (temp.data = x) then return temp;

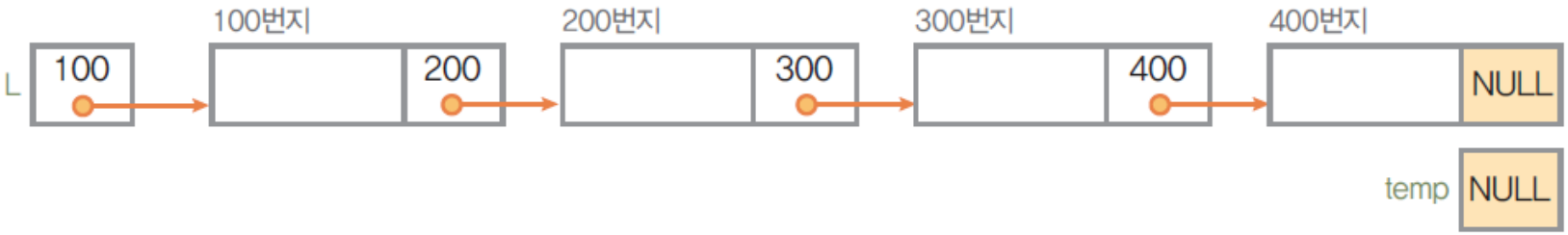


2. 단순 연결 리스트

- 단순 연결 리스트에서 x 노드 탐색 과정

❷- b else temp ← temp.link;

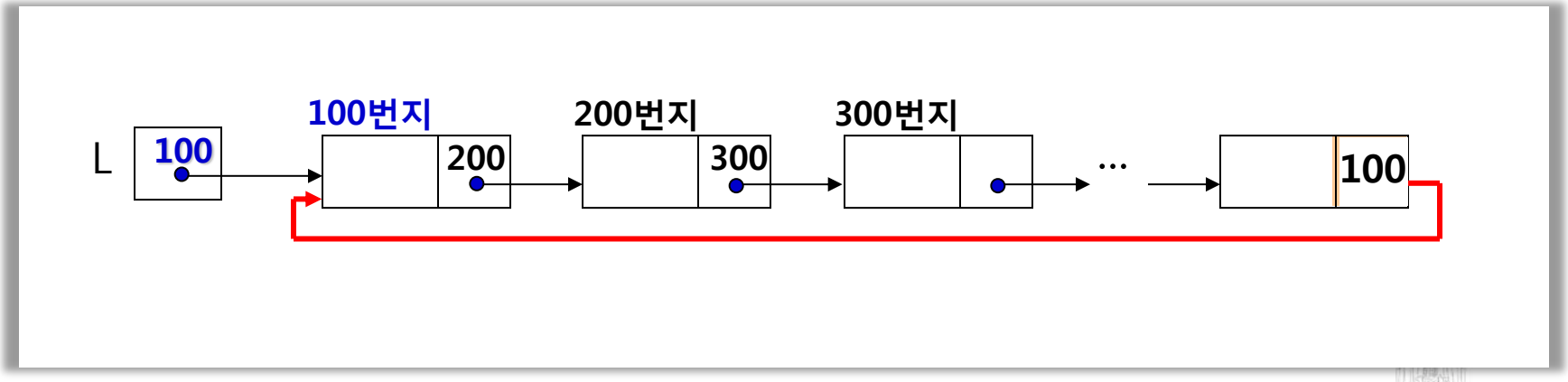
❸ return temp;



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

❖ 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

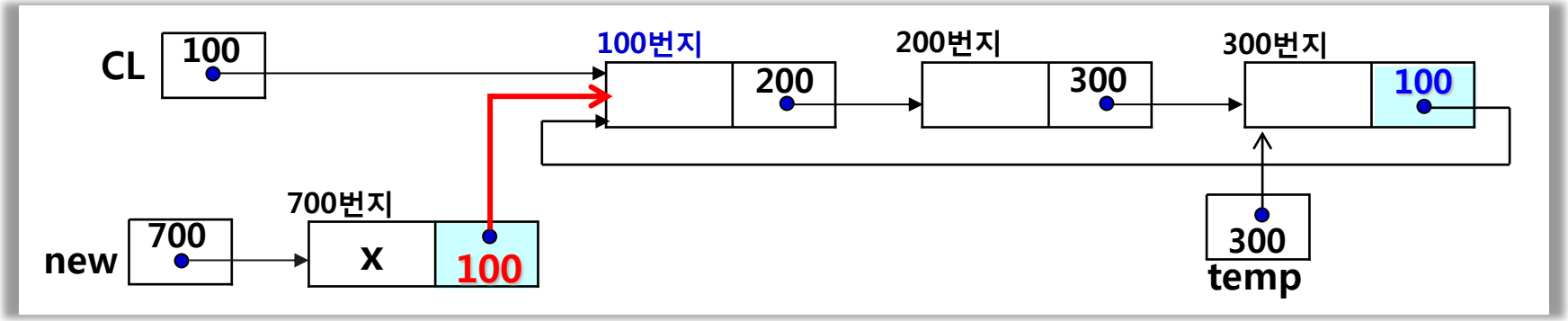
```
insertFirstNode(CL, x)
    new ← getNode();
    new.data ← x;
    if (CL = NULL) then {
①       ❶-ⓐ CL ← new;
        ❶-ⓑ new.link ← new;
    }

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

3. 원형 연결 리스트

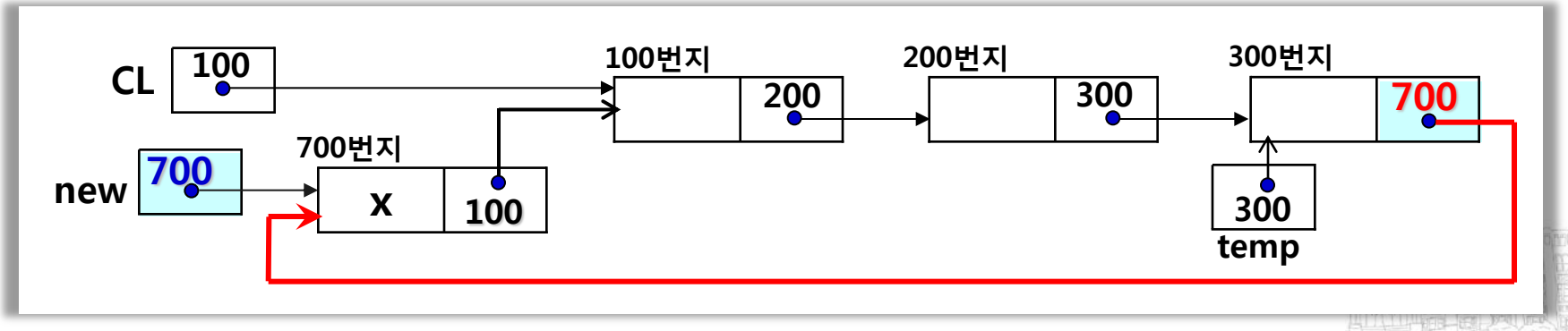
❷- c new.link ← temp.link;

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



❷- d temp.link ← new;

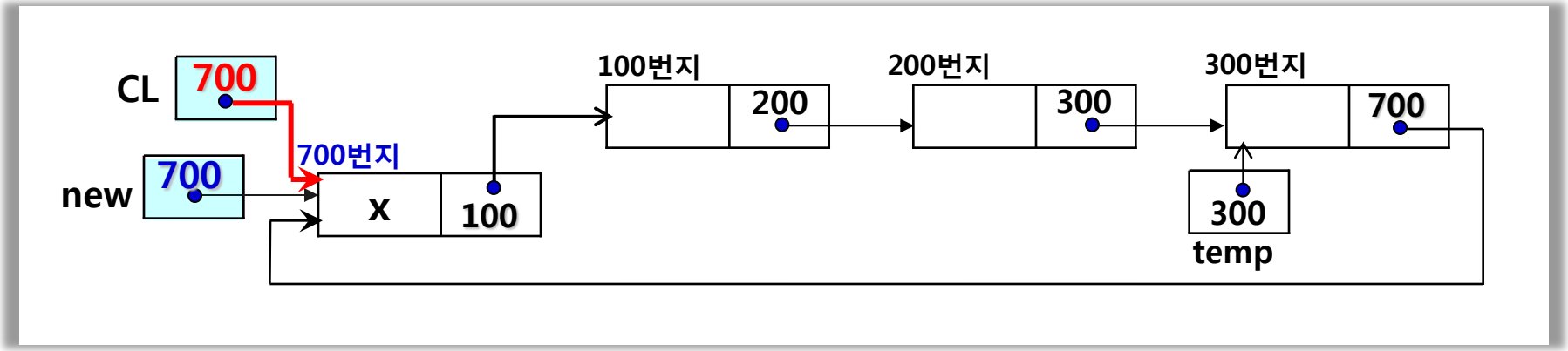
포인터 new의 값을 포인터 temp가 가리키고 있는 마지막 노드의 링크에 저장하여, 리스트의 마지막 노드가 노드 new를 가리키게 한다.



3. 원형 연결 리스트

❷- e CL ← new;

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



3. 원형 연결 리스트

- 최종결과

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

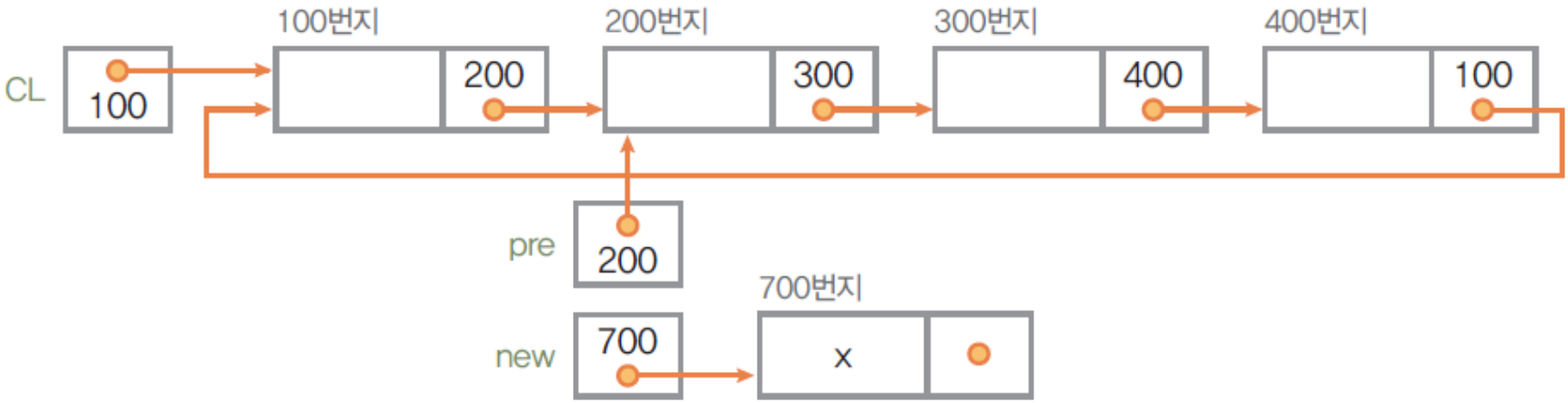
```
insertMiddleNode(CL, pre, x)
    new ← getNode();
    new.data ← x;
    if (CL = NULL) then {                    ❶
        CL ← new;
        new.link ← new;
    }
    else {                                   ❷
        ❷-ⓐ new.link ← pre.link;
        ❷-ⓑ pre.link ← new;
    }
end insertMiddleNode()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

3. 원형 연결 리스트

- 원형 연결 리스트 중간에 노드를 삽입하는 과정

❶ 공백 리스트인 경우

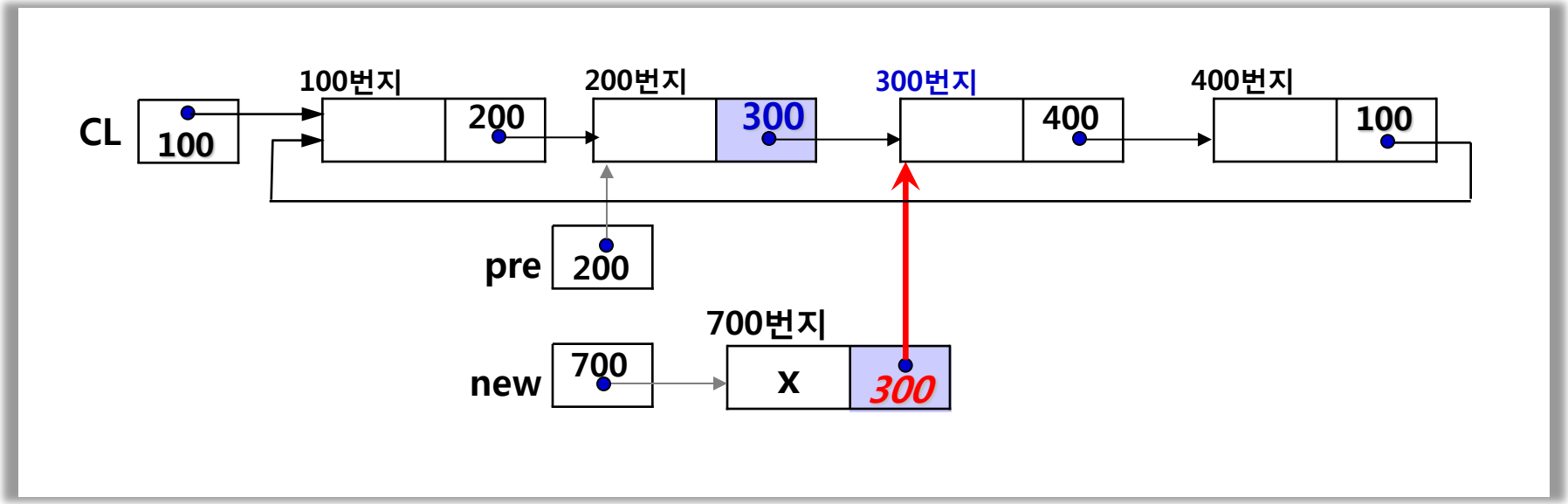
❷ 공백 리스트가 아닌 경우



3. 원형 연결 리스트

❷- a new.link ← pre.link;

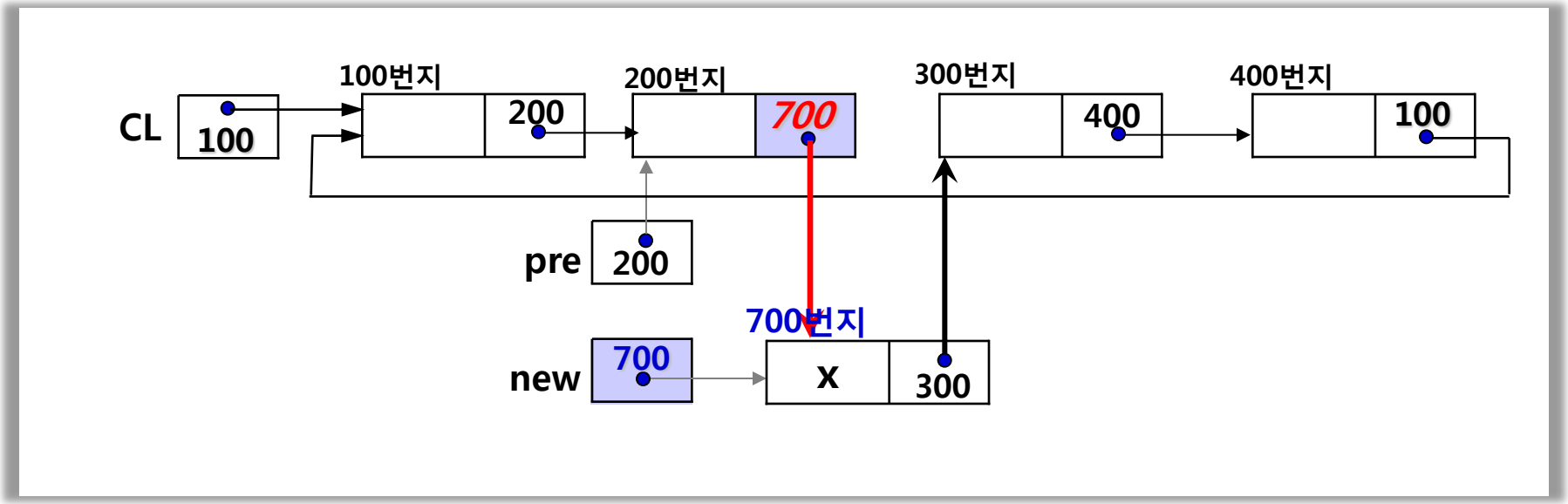
노드 pre의 다음 노드로 new를 삽입하기 위해서, 먼저 **노드 pre의 다음 노드(pre.link)를 new의 다음 노드(new.link)로** 연결



3. 원형 연결 리스트

❷- b pre.link ← new;

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



3. 원형 연결 리스트

- 최종결과

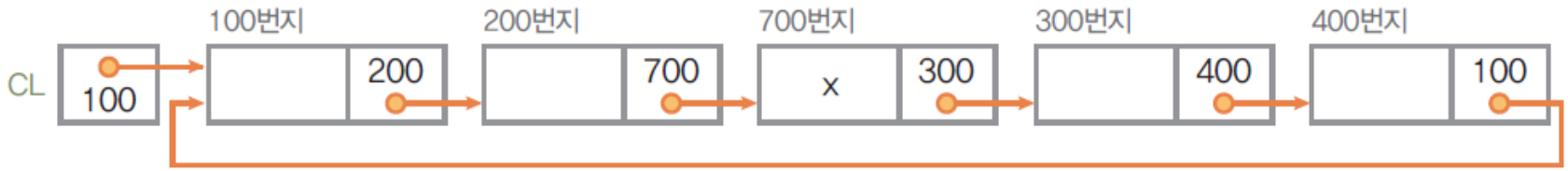


그림 4-23 원형 연결 리스트 CL이 공백인 아닌 경우에 중간 노드로 삽입한 최종 결과

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
              ❸-❷ CL ← old.link;
        ❹ returnNode(old);
    }
end deleteNode()
```

3. 원형 연결 리스트

- 원형 연결 리스트에서 노드를 삭제하는 과정
  - 초기상태

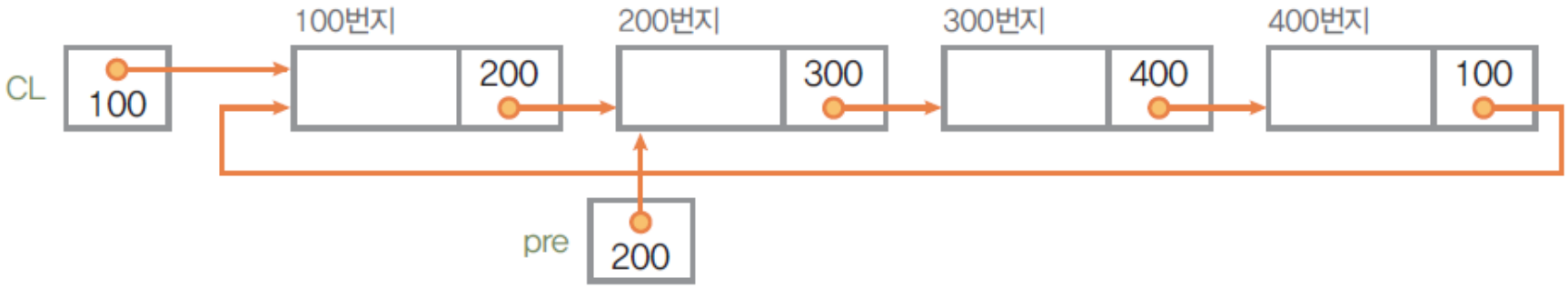


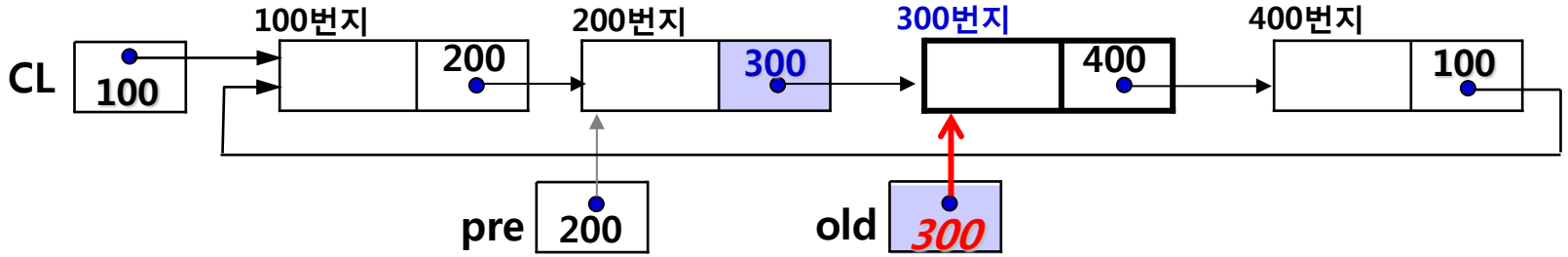
그림 4-24 초기 상태

3. 원형 연결 리스트

- 원형 연결 리스트에서 노드를 삭제하는 과정

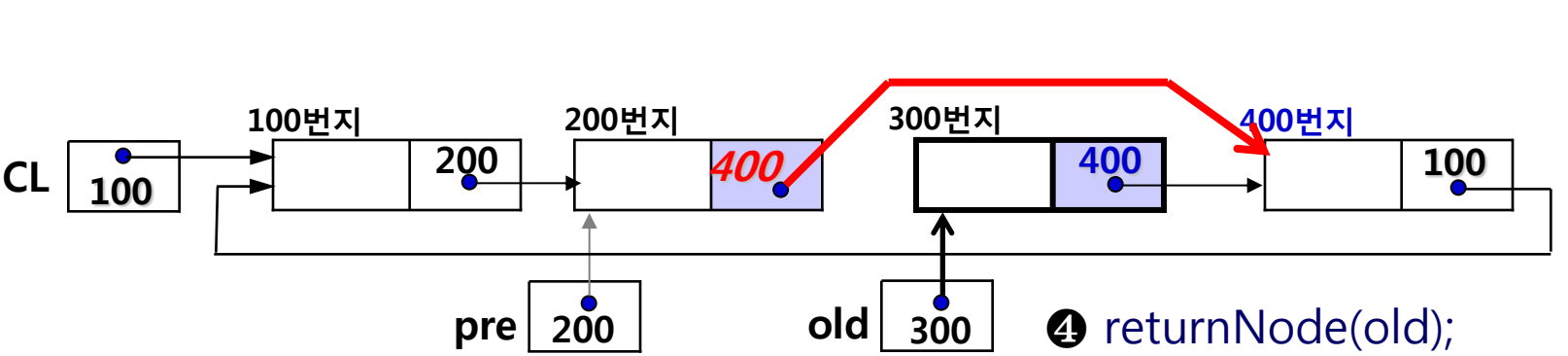
❶ old ← pre.link;

노드 pre의 다음노드(pre.link)를 삭제할 노드 old로 지정



❷ pre.link ← old.link;

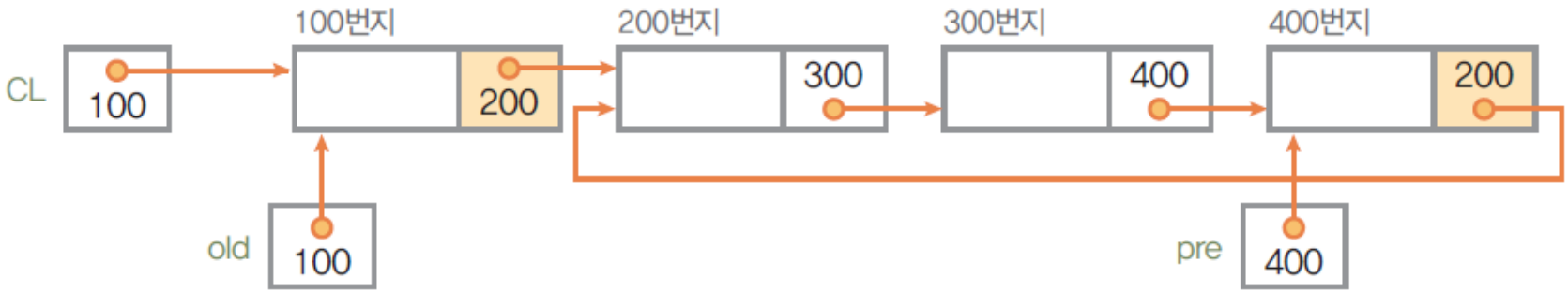
old의 다음노드(old.link)를 노드 old의 이전노드 pre 노드와 서로 연결



3. 원형 연결 리스트

❸ if (old = CL) then

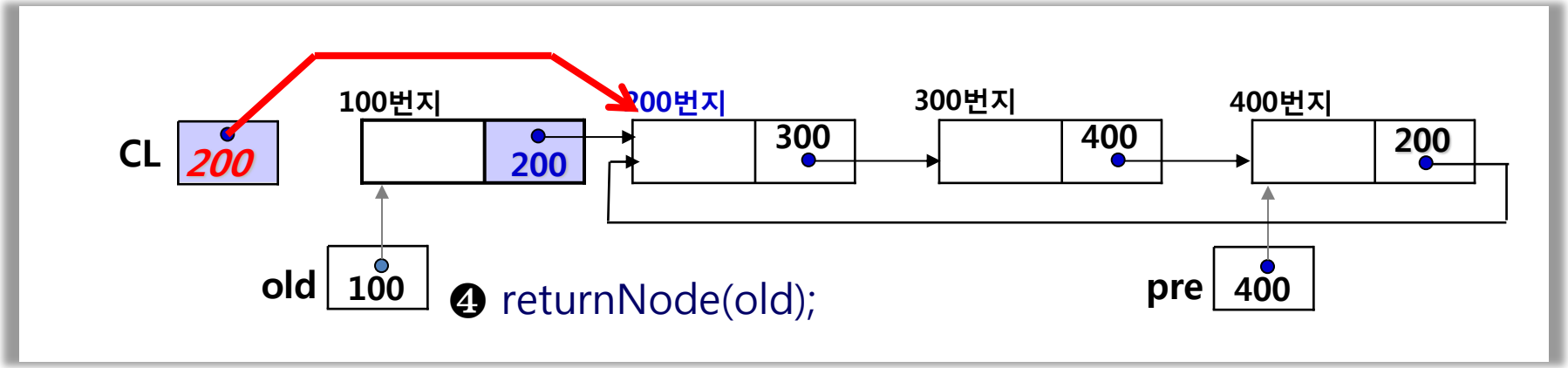
삭제할 노드 old가 원형 연결 리스트의 첫번째 노드인 경우 >>



3. 원형 연결 리스트

❸- a CL ← old.link;

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



❹ returnNode(old);

삭제한 노드 old의 메모리를 반환한다.

3. 원형 연결 리스트

- 최종결과

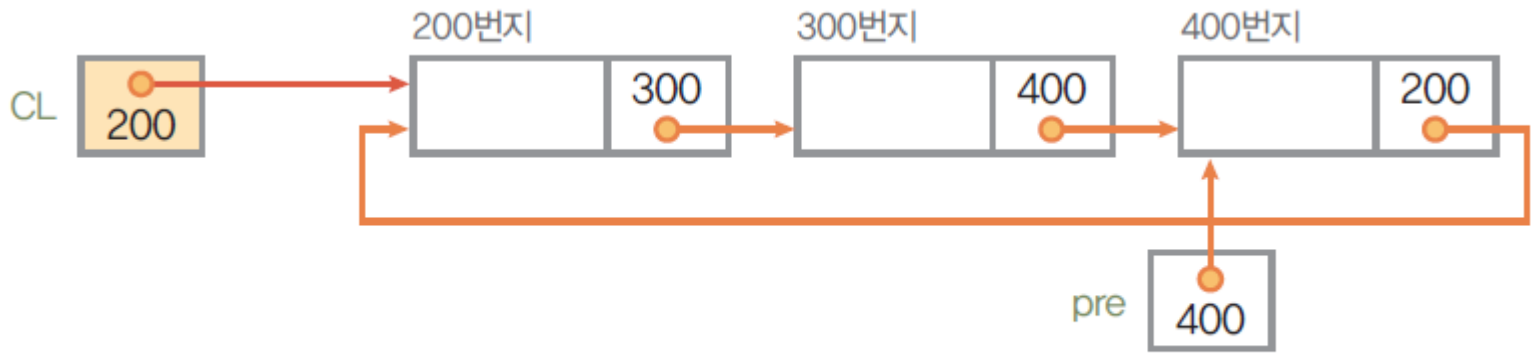


그림 4-25 원형 연결 리스트에서 노드를 삭제한 최종 결과

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

- 이중 연결 리스트에서 노드를 삽입하는 방법

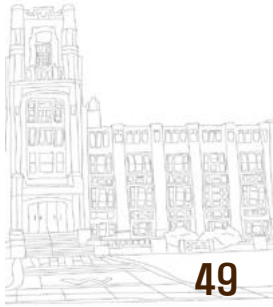
1. 삽입할 노드를 준비한다.
2. 새 노드의 데이터 필드에 값을 저장한다.
3. 새 노드 왼쪽 노드의 오른쪽 링크 필드(rlink)에 있던 값을 새 노드의 오른쪽 링크 필드(rlink)에 저장한다.
4. 왼쪽 노드의 오른쪽 링크 필드(rlink)에 새 노드의 주소를 저장한다.
5. 새 노드 오른쪽 노드의 왼쪽 링크 필드(llink)에 있던 값을 새 노드의 왼쪽 링크 필드(llink)에 저장한다.
6. 오른쪽 노드의 왼쪽 링크 필드(llink)에 새 노드의 주소를 저장한다.
7. 노드를 순서대로 연결한다.

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;
    ❶ new.rlink ← pre.rlink;
    ❷ pre.rlink ← new;
    ❸ new.llink ← pre;
    ❹ 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

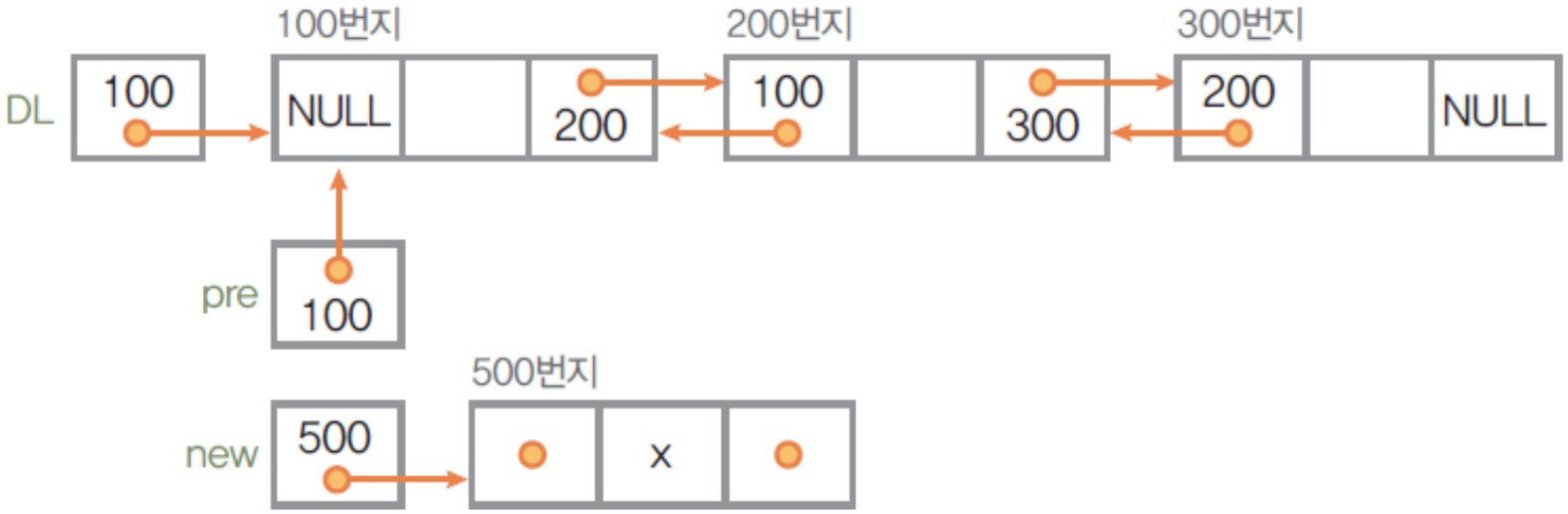
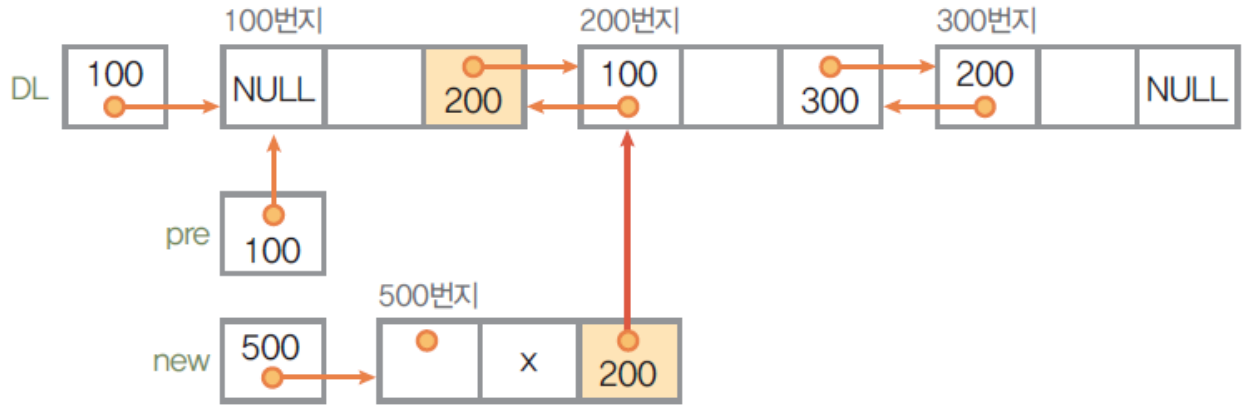


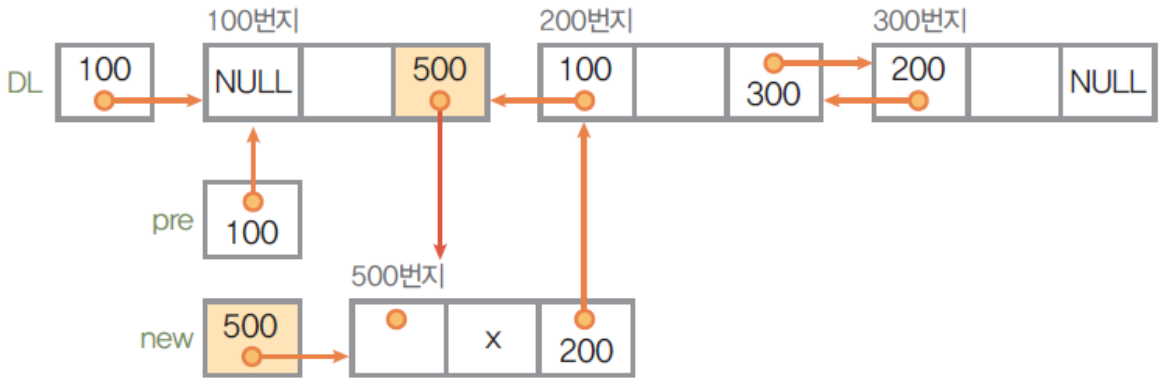
그림 4-31 초기 상태

4. 이중 연결 리스트

❶ new.rlink ← pre.rlink;

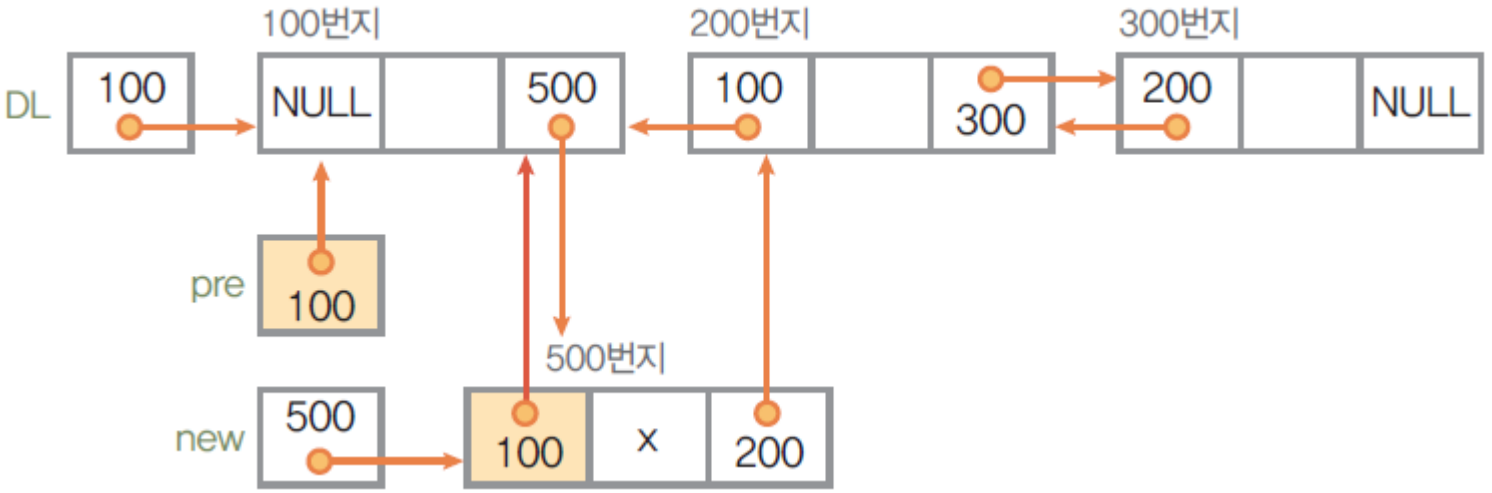


❷ pre.rlink ← new;

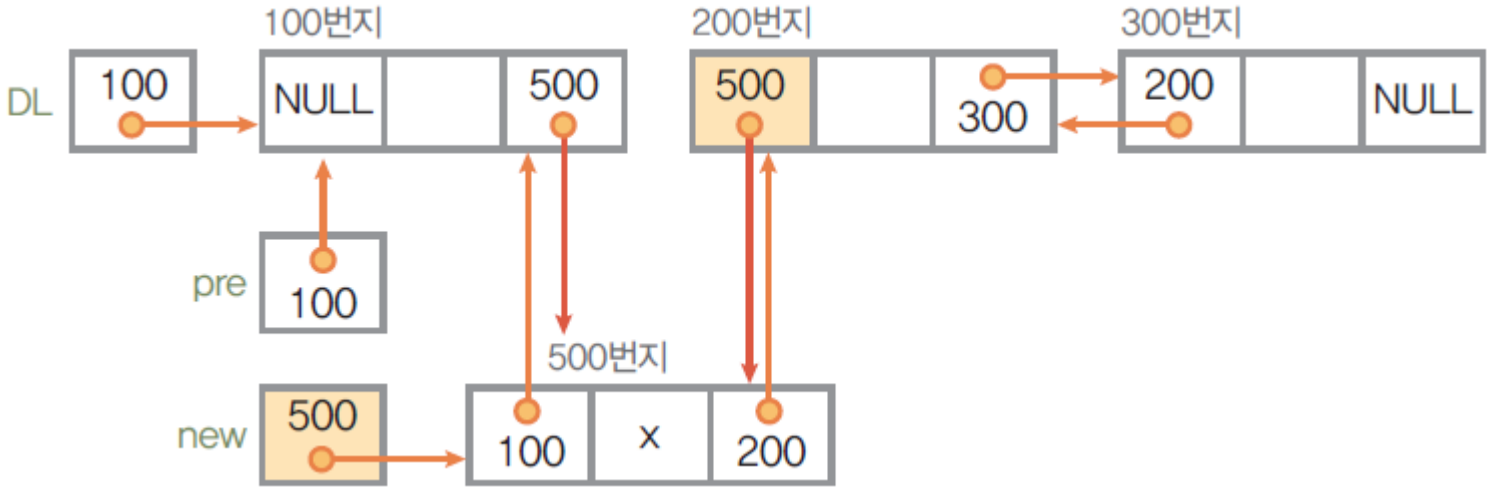


4. 이중 연결 리스트

❸ new.llink ← pre;



❹ new.rlink.llink ← new;



4. 이중 연결 리스트

- 최종 결과

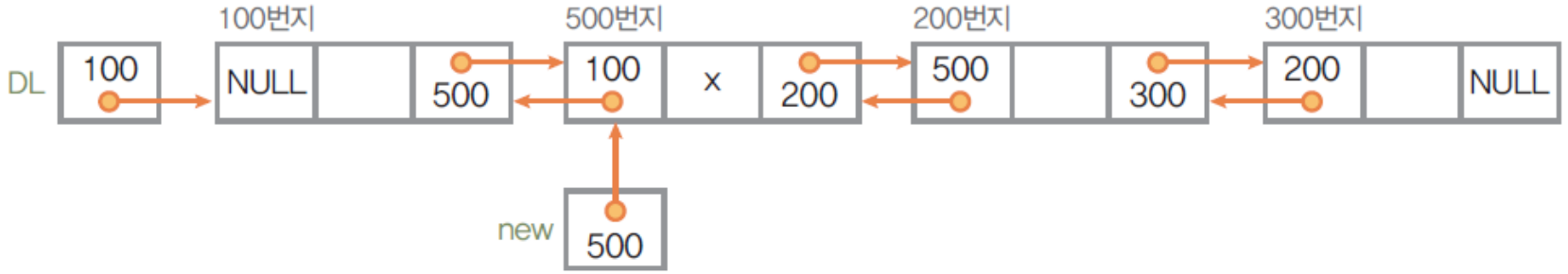


그림 4-32 이중 연결 리스트 DL에 new 노드가 삽입된 최종 결과

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)
    ❶ old.llink.rlink ← old.rlink;
    ❷ old.rlink.llink ← old.llink;
    ❸ returnNode(old);
end deleteNode()
```

4. 이중 연결 리스트

- 이중 연결 리스트에서 노드를 삭제하는 과정
  - 초기 상태

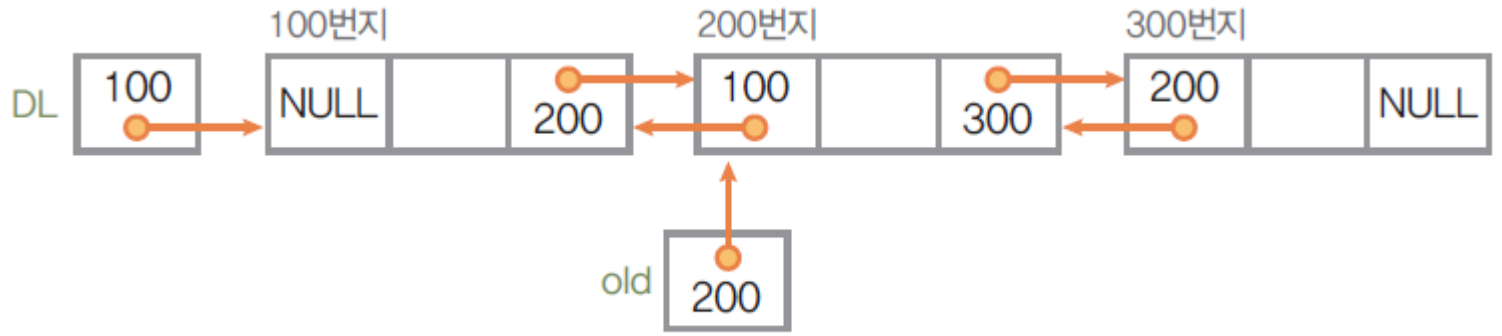
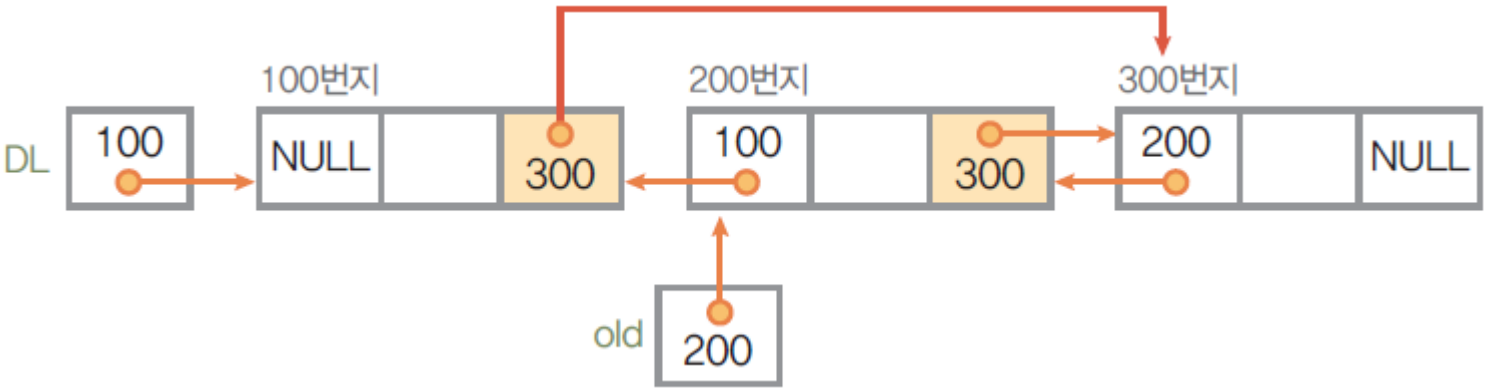


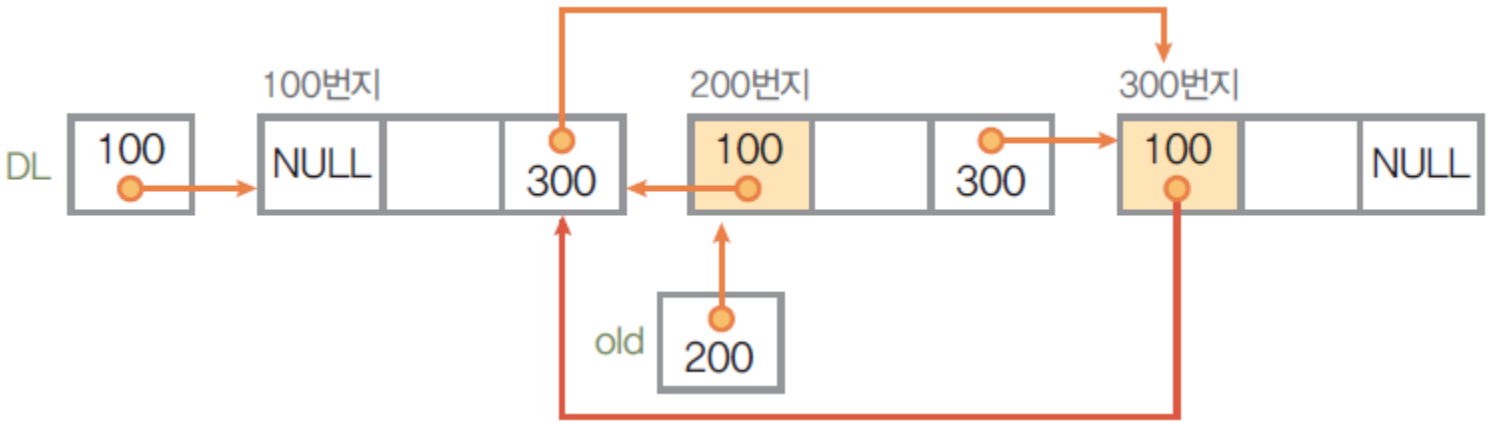
그림 4-34 초기 상태

❶ old.llink.rlink ← old.rlink;



4. 이중 연결 리스트

❷ old.rlink.llink ← old.llink;



❸ returnNode(old);

4. 이중 연결 리스트

- 최종 결과

그림 4-35 이중 연결 리스트에서 old 노드를 삭제한 최종 결과