

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

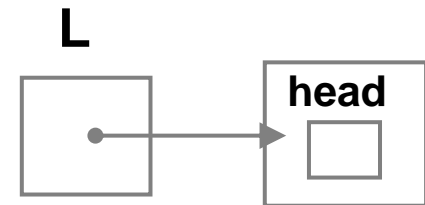
```
typedef struct ListNode{  
    char data[10];  
    struct ListNode* link;  
} listNode;
```

```
typedef struct{  
    listNode* head;  
} linkedList_h;
```

```
linkedList_h* createLinkedList_h(void);  
void freeLinkedList_h(linkedList_h*);  
void addLastNode(linkedList_h*, char*);  
void deleteLastNode(linkedList_h*);  
void reverse(linkedList_h*);  
void printList(linkedList_h*);
```

Q: 반환값이 있는 함수는?

```
linkedList_h* createLinkedList_h(void){  
    linkedList_h* L;  
    L = (linkedList_h*)malloc(sizeof(linkedList_h));  
    L -> head = NULL;  
    return L;  
}
```



## ▪ malloc 함수

함수밖에서도 사용하기 위해 메모리를 할당.  
할당된 메모리의 주소값을 반환.

```
void addLastNode(linkedList_h* L, char* x){
    listNode* newNode;
    listNode* p;
    newNode = (listNode*)malloc(sizeof(listNode));
    strcpy(newNode->data, x);
    newNode->link= NULL;
    if (L->head == NULL){
        L->head = newNode;
        return;
    }
    p = L->head;
    while (p->link != NULL) {
        p = p->link;
    }
    p ->link = newNode;
}
```

Q: 신규노드의 링크필드가 수정되는 명령은?

Q: 신규노드의 앞노드의 링크필드가 수정되는 명령은?

```
void deleteLastNode(linkedList_h * L){
    listNode* previous;
    listNode* current;
    if (L->head == NULL) return;
    if (L->head->link == NULL) {
        free(L->head);
        L->head = NULL;
        return;
    }
    else {
        previous = L->head;
        current = L->head->link;
        while(current ->link != NULL)
        {
            previous = current;
            current = current->link;
        }
        free(current);
        previous->link = NULL;
    }
}
```

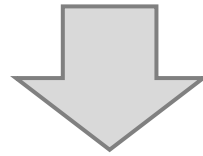
Q: 모두 몇가지의 상황을 고려하고 있는가?

Q: 삭제될 노드의 앞노드의 링크필드가 수정되는 명령은?

## Exercise

Q: 아래와 같은 연결리스트의 순서를 역방향으로 변환하는 알고리즘을 자연어로 표현 하시오.

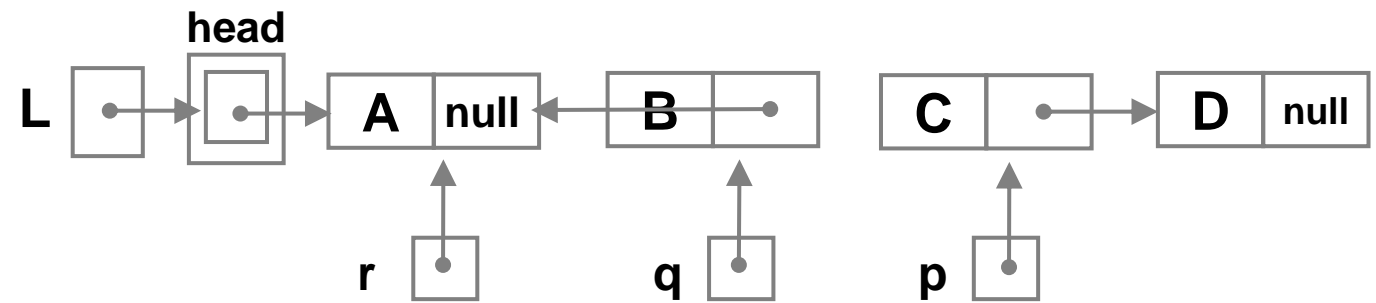
Q: 몇 개의 포인터가 필요할까?



# reverse ( )

```
void reverse(linkedList_h * L){
```

```
    listNode* p;  
    listNode* q;  
    listNode* r;
```



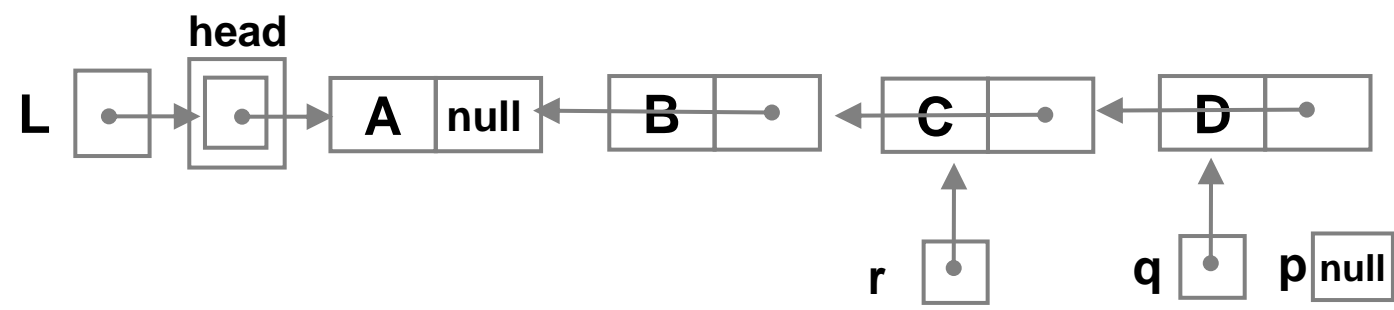
```
    p = L->head;  
    q=NULL;  
    r=NULL;
```

Q: while문이 종료되었을때 p,q,r의 위치는?

```
while (p!= NULL){
```

```
    r = q;  
    q = p;  
    p = p->link;  
    q->link = r;
```

```
}  
L->head = q;  
}
```



## ◆ 단순 연결 리스트의 노드 탐색 알고리즘

리스트의 노드를 처음부터 하나씩 순회하면서 노드의 데이터 필드의 값과 x를 비교하여 일치하는 노드를 찾는 연산

```
searchNode(L, x)
    temp ← L;                      // ①
    while (temp ≠ null) do {
        if (temp.data = x ) then return temp;    // ②
        temp ← temp.link;
    }
    return temp;                    // ③
end searchNode()
```

Q1: 위의 코드에서 다음 노드로 이동시키는 명령은?

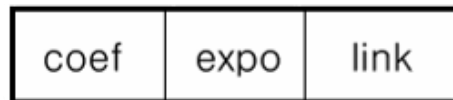
Q2: temp=null 인 경우는 어떤 상황에서 발생하는가?

## ◆ 다항식의 연결 자료구조 표현

- 단순 연결 리스트를 이용하여 다항식 표현
- 다항식의 항 : 단순 연결 리스트의 노드

## ◆ 노드 구조

- 각 항에 대해서 계수와 지수를 저장
- 계수를 저장하는 coef와 지수를 저장하는 expo의 두 개의 필드로 구성
- 링크 필드 : 다음 항을 연결하는 포인터로 구성



## ◆ 노드에 대한 구조체 정의

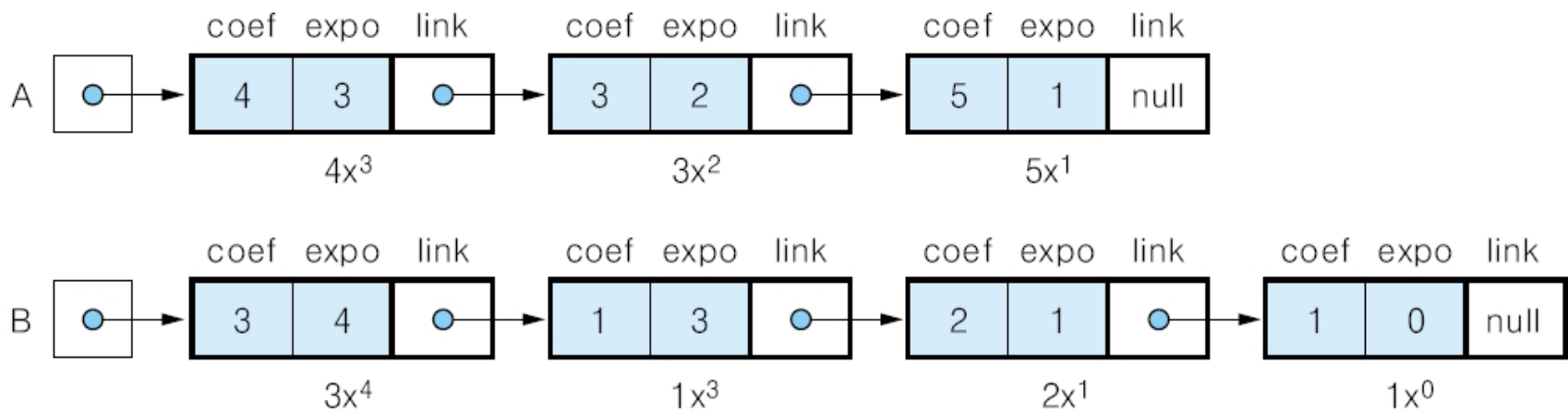
```
struct Node {  
    float coef;  
    int expo;  
    struct Node *link;  
};
```



◆ 다항식의 단순 연결 리스트 표현 예




다항식  $A(x)=4x^3+3x^2+5x$

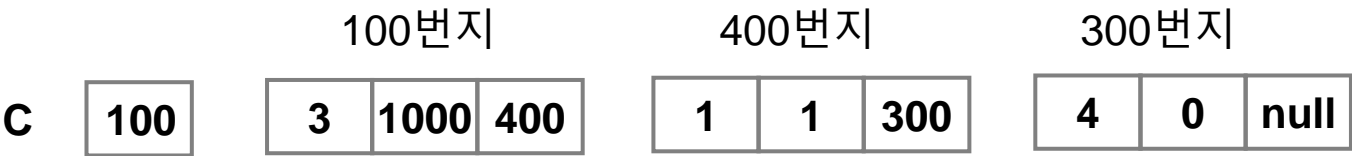
다항식  $B(x)=3x^4+x^3+2x+1$



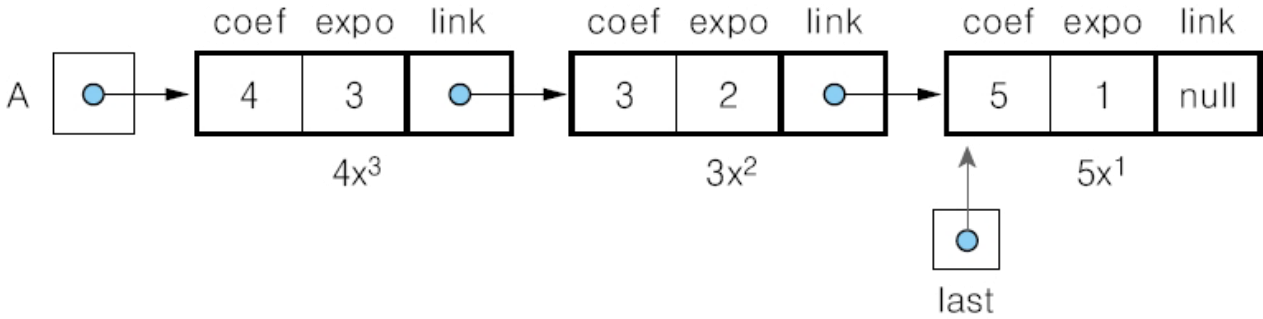
◆다항식을 순차자료로 모델링하는 것과 비교하여 연결리스트를 이용할 때의 장점은?

$C(x)=3x^{1000} + x + 4$  의 표현방법

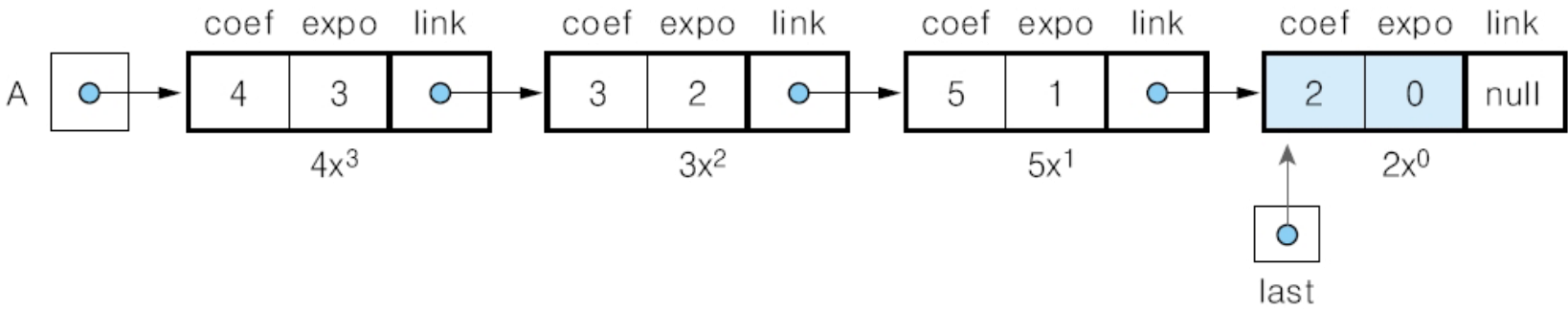
	[0]	[1]	
[0]	1000	3	 $3x^{1000}$
[1]	1	1	 $x$
[2]	0	4	 $4$



## ◆ 다항식 리스트 A에 상수항 추가



(a) appendTerm(A,2,0,last) 함수 실행 전의 다항식 리스트 A



(b) appendTerm(A,2,0,last) 함수 실행 후의 다항식 리스트 A

**Q:** 위의 그림에서  $6x^4$  를 추가하려면?

◆ 단순 연결 리스트로 표현된 2개의 다항식을 더하는 알고리즘은?

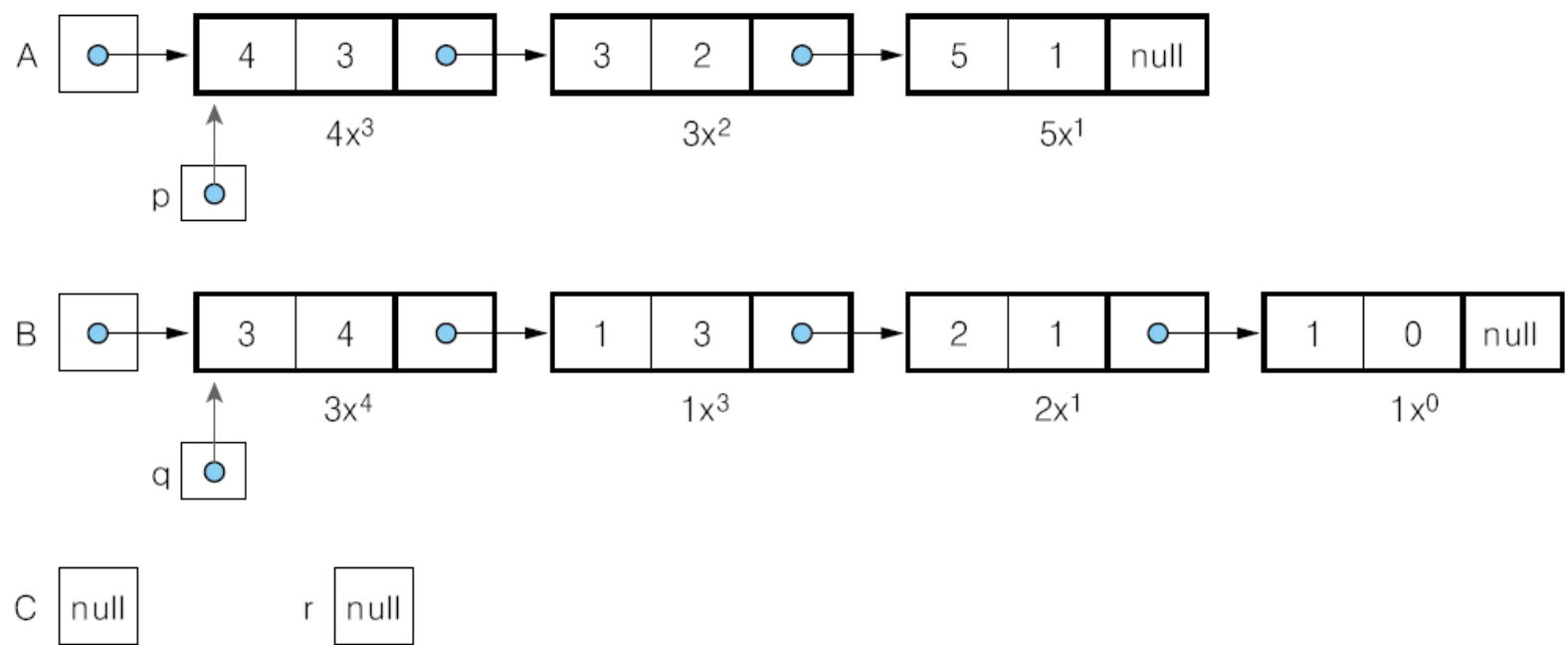
- $A(x)=4x^3+3x^2+5x$
- $B(x)=3x^4+x^3+2x+1$

◆ 덧셈  $A(x)+B(x)=C(x)$ 를 단순 연결 리스트 자료구조를 사용하여 연산

- 다항식  $A(x)$ 와  $B(x)$ ,  $C(x)$ 의 항을 지시하기 위해서 세 개의 포인터를 사용
- 포인터  $p$  : 다항식  $A(x)$ 에서 비교할 항을 지시
- 포인터  $q$  : 다항식  $B(x)$ 에서 비교할 항을 지시
- 포인터  $r$  : 덧셈연산 결과 만들어지는 다항식  $C(x)$ 의 항을 지시

◆ 다항식의 덧셈 예

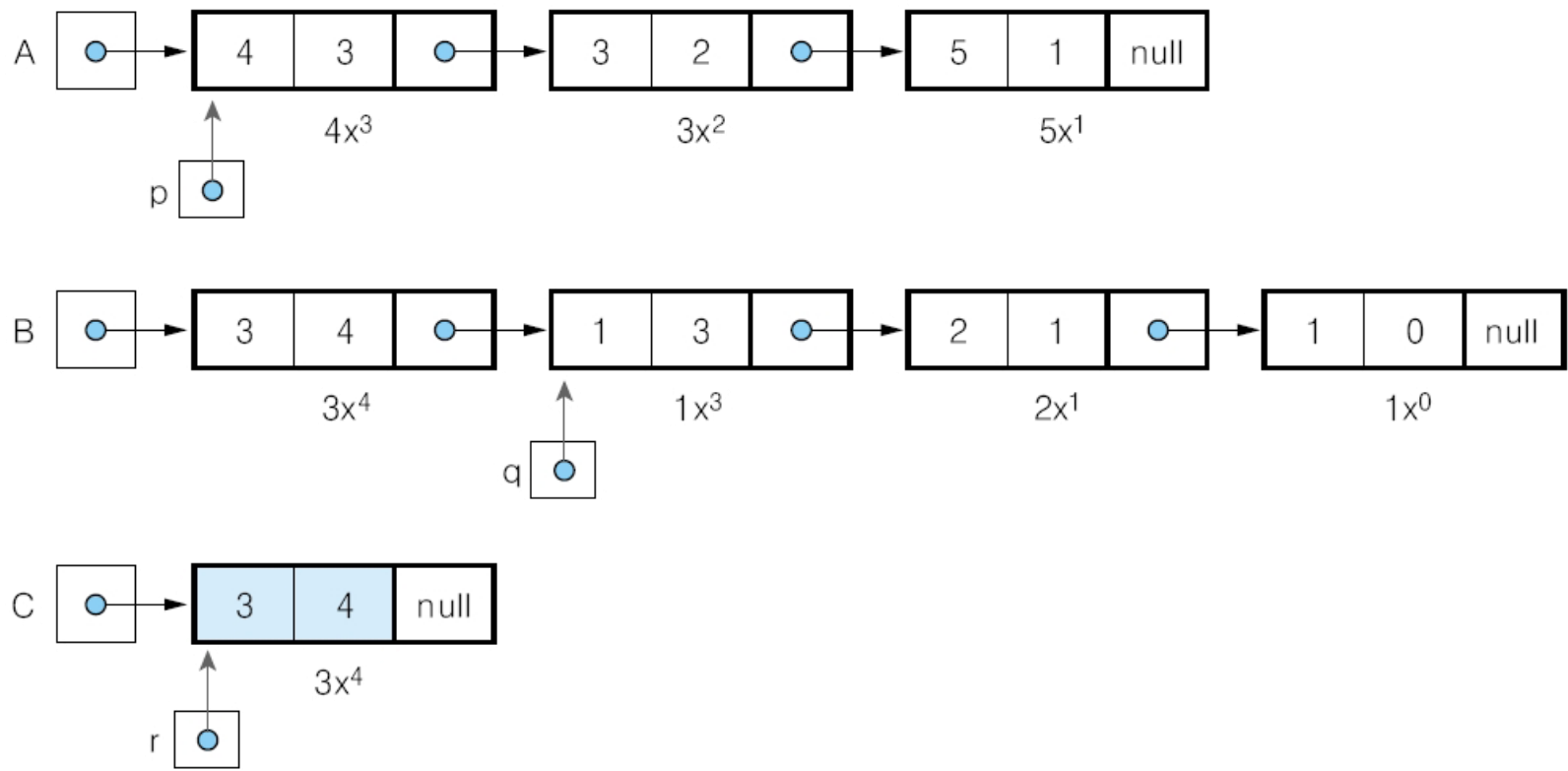
- $A(x)=4x^3+3x^2+5x$
- $B(x)=3x^4+x^3+2x+1$
- 초기 상태



Q: 위의 그림에서 포인터를 이동시키면서 다항식을 더하는 알고리즘은?

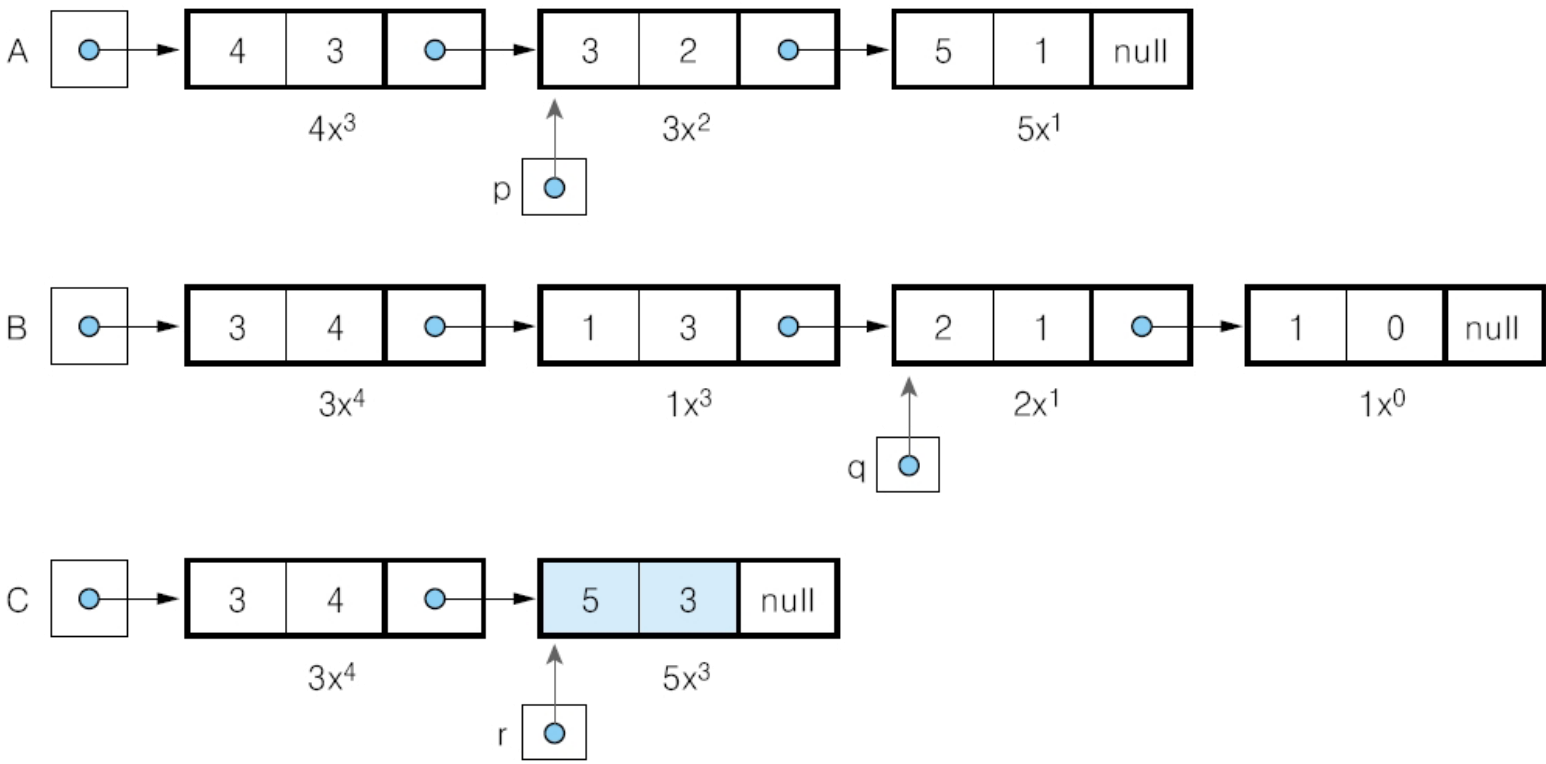
## ① $4x^3$ 항과 $3x^4$ 항에 대한 처리

- $p.\text{expo} < q.\text{expo}$ 이므로 지수가 큰  $q$  노드를 리스트 C에 복사
- 포인터  $q$ 는 다음 노드로 이동



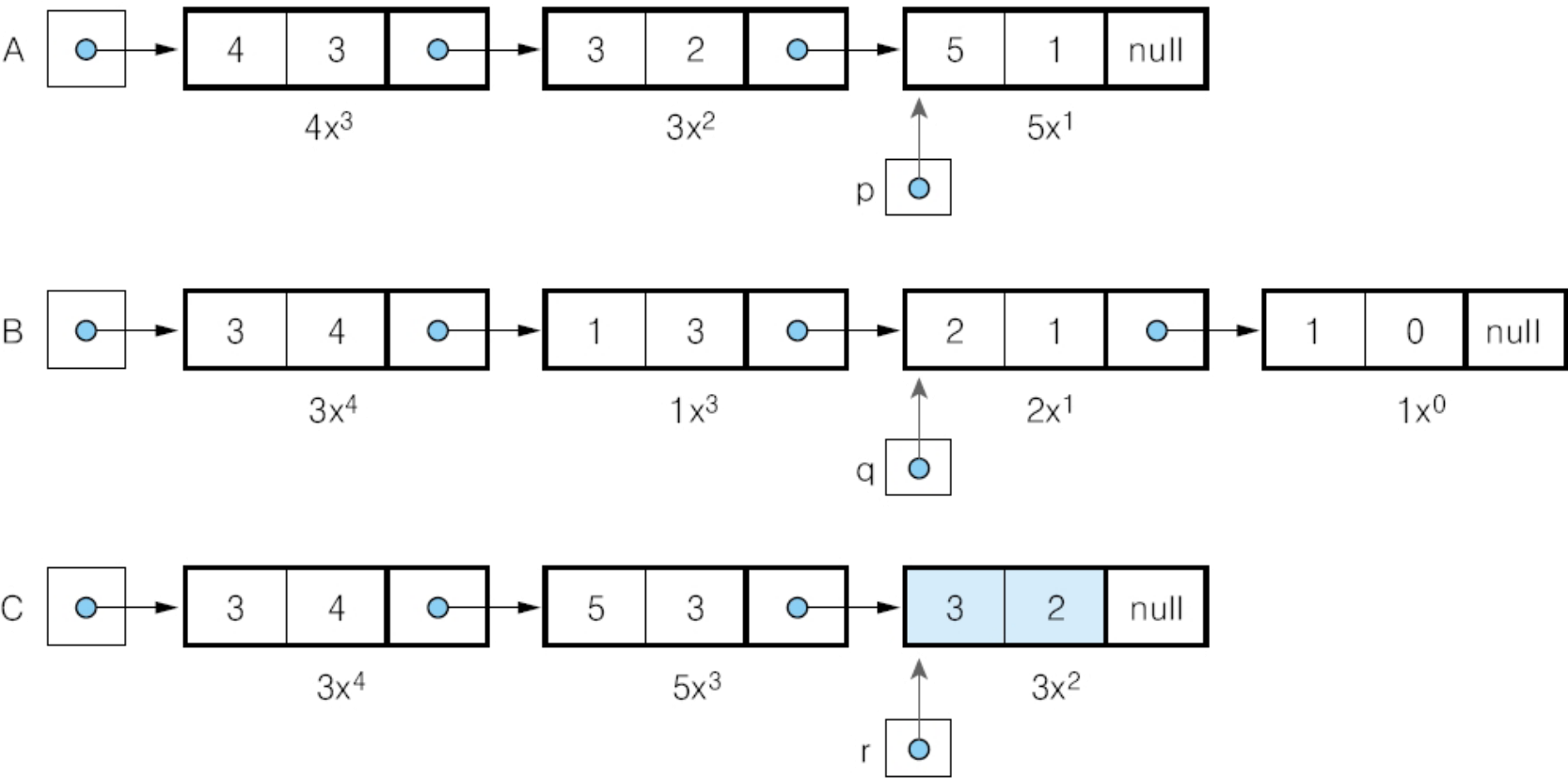
②  $4x^3$ 항과  $1x^3$  항에 대한 처리

- $p.expo = q.expo$  이므로 두 노드의 coef 필드의 값을 더하고, expo 필드의 값을 복사한 노드를 리스트 C에 추가
- 포인터 p와 q는 다음 노드로



③  $3x^2$ 항과  $2x$  항에 대한 처리

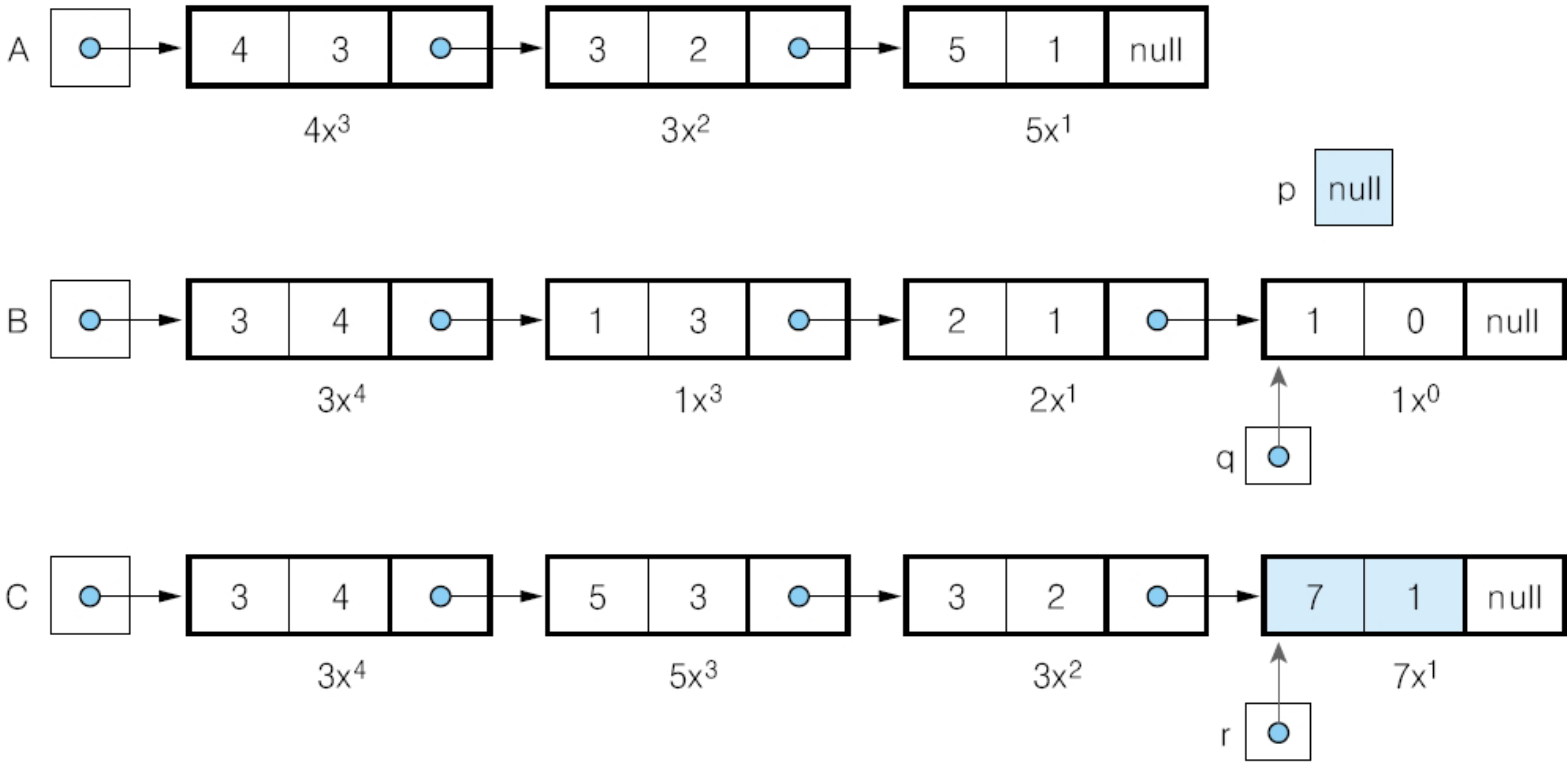
- $p.expo > q.expo$  이므로 지수가 큰  $p$  노드를 리스트 C에 복사
- 포인터  $p$ 는 다음 노드로 이동





④  $5x$  항과  $2x$  항에 대한 처리

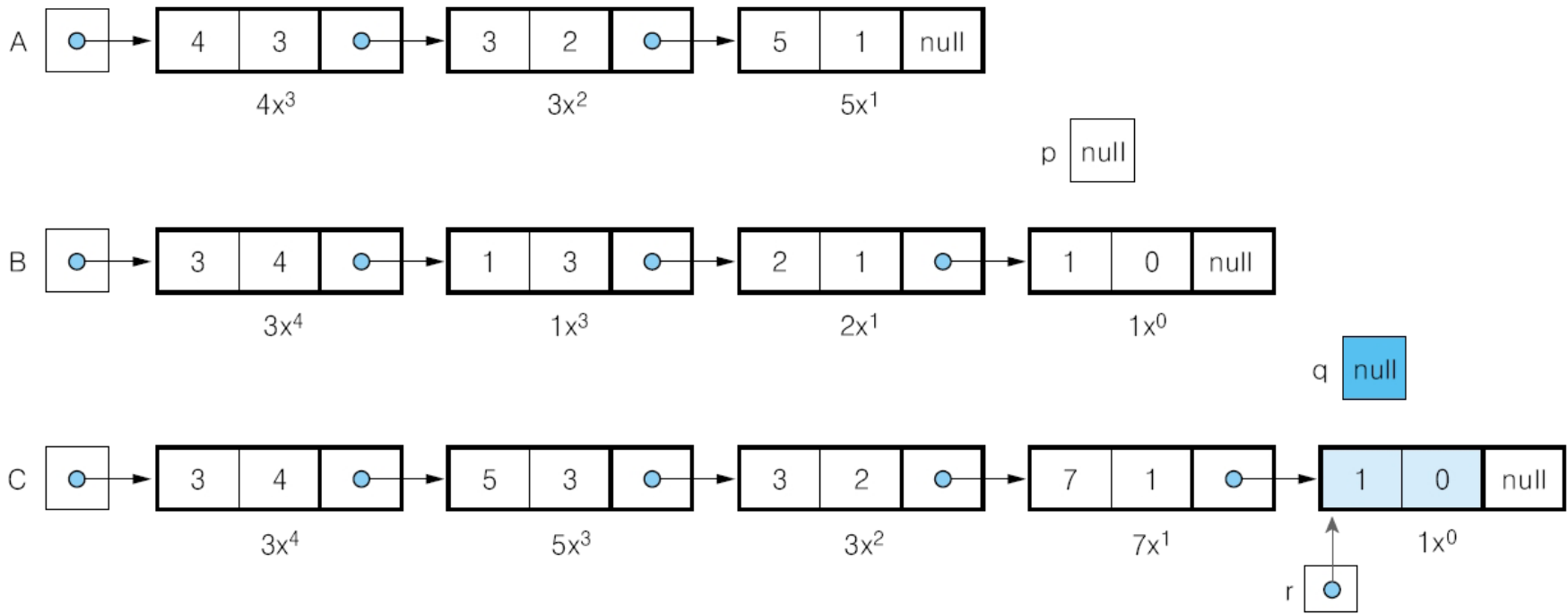
- $p.expo = q.expo$  이므로 두 노드의 coef 필드의 값을 더하고, expo 필드의 값을 복사한 노드를 리스트 C에 추가
- 포인터 p와 q는 다음 노드로 이동



⑤ B(x)의 남은 항에 대한 처리

포인터 p가 null이므로 다항식 A(x)의 항에 대한 처리 끝

처리할 항이 남아있는 다항식 B(x)의 포인터 q는 null이 될 때까지 모든 노드를 복사하여 리스트 C에 추가



## Exercise

Q1: 다항식을 아래와 같은 연결리스트로 표현하는 것이 가능한가?

Q2: 아래와 같은 다항식 A와 B를 더하는 알고리즘을 설명하시오.



# 이중 연결 리스트

## ◆ 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



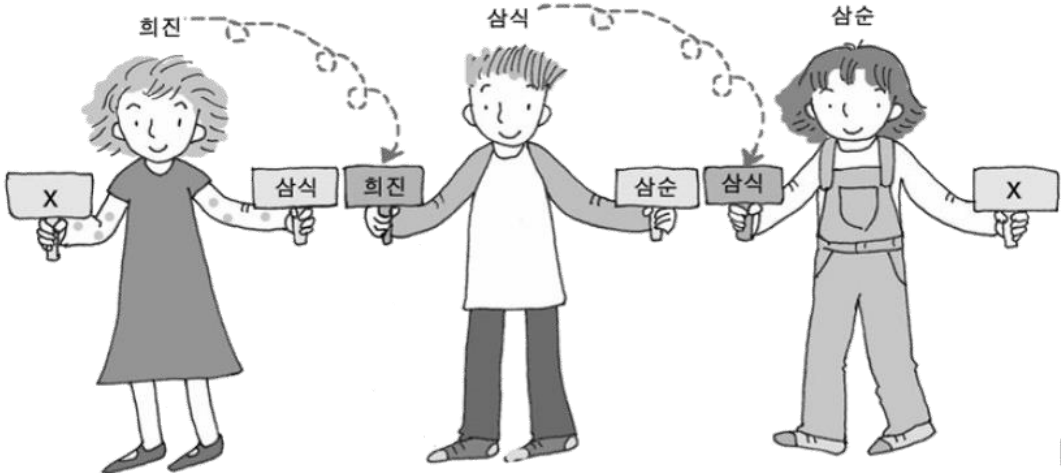
## ◆노드 구조에 대한 구조체 정의

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

◆ 단순연결기차



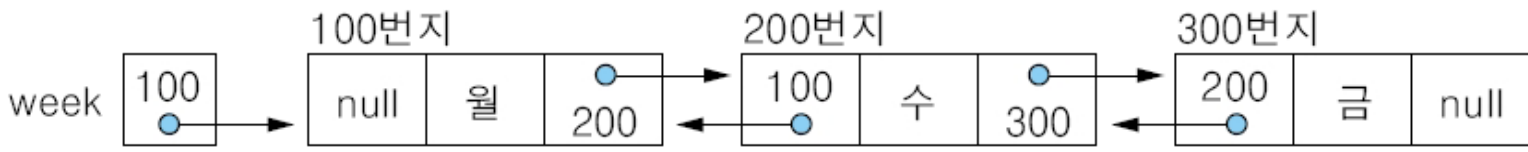
◆ 이중연결기차



# 이중 연결 리스트

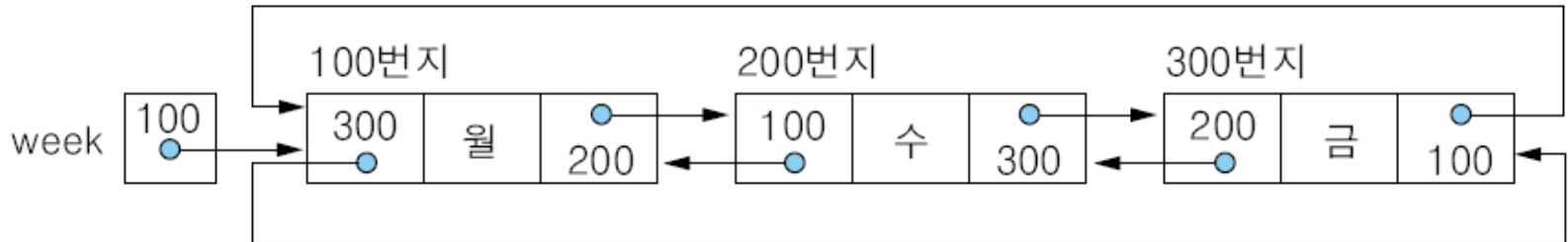
Q: 이중 연결 리스트 구성하면 'null' 이 몇번 등장하는가?

◆리스트 week=(월, 수, 금)의 이중 연결 리스트

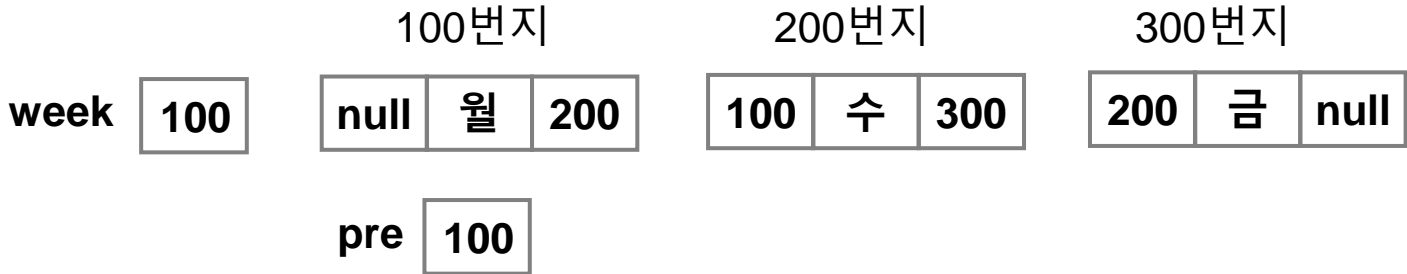


Q2: 원형 이중 연결리스트를 구성하면 'null'이 몇번 등장하는가?

◆ 리스트 week=(월, 수, 금)의 원형 이중 연결 리스트



# 이중 연결 리스트 - 점연산자



Q1: 월요일 왼쪽 링크필드값을 pre를 이용해 표현하면?

pre.llink

Q2: 월요일 오른쪽 링크필드값을 pre를 이용해 표현하면?

pre.rlink

Q3: 수요일 왼쪽 링크필드값을 pre를 이용해 표현하면?

pre.rlink.llink

Q4: 금요일 오른쪽 링크필드값을 pre를 이용해 표현하면?

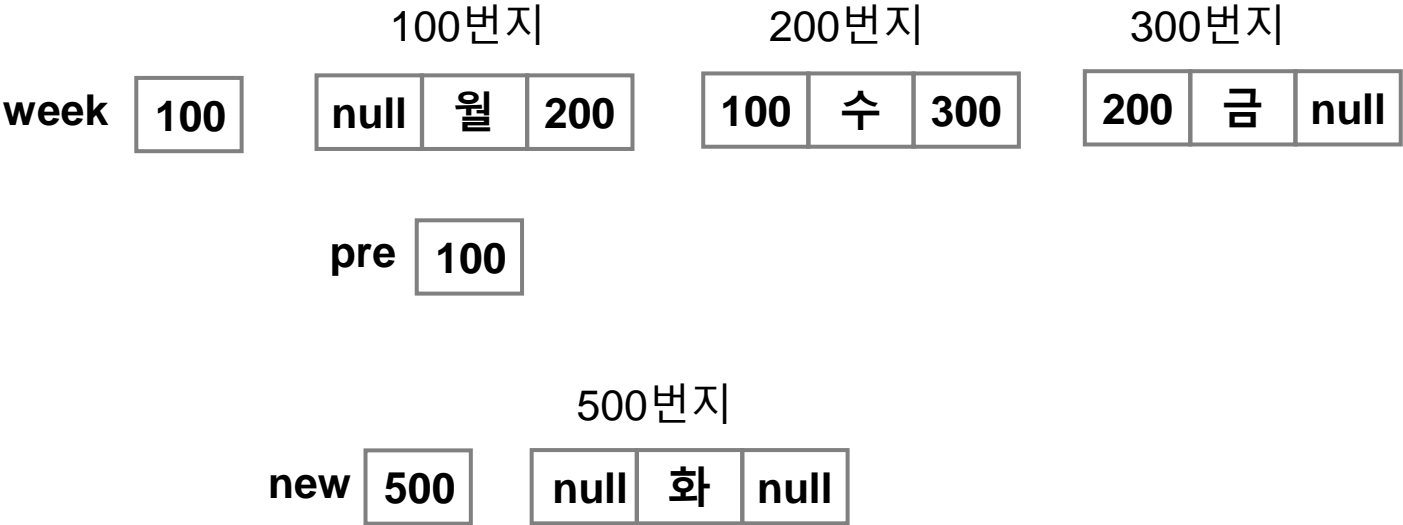
pre.rlink.rlink.rlink



# 이중 연결 리스트 - 삽입연산

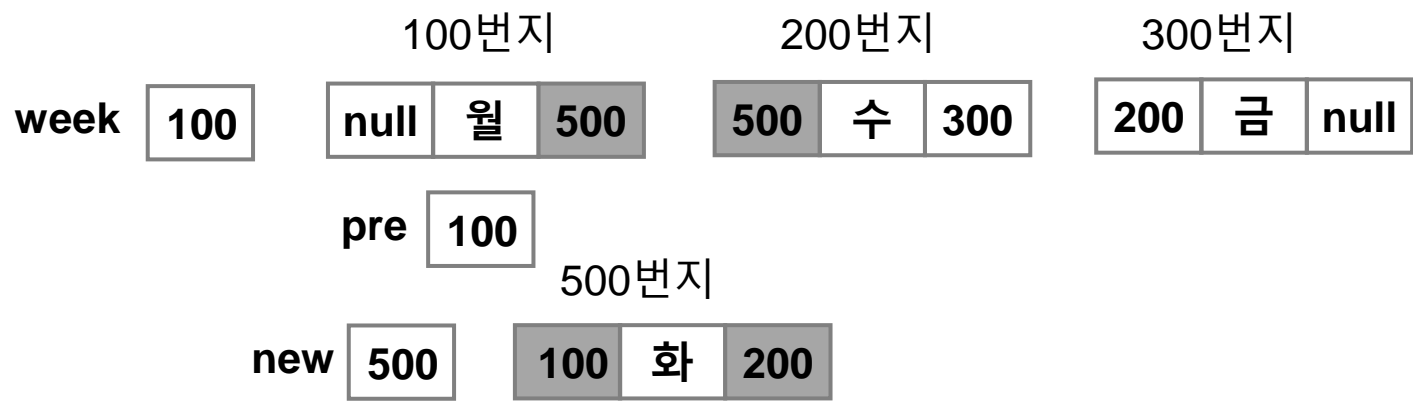
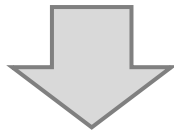
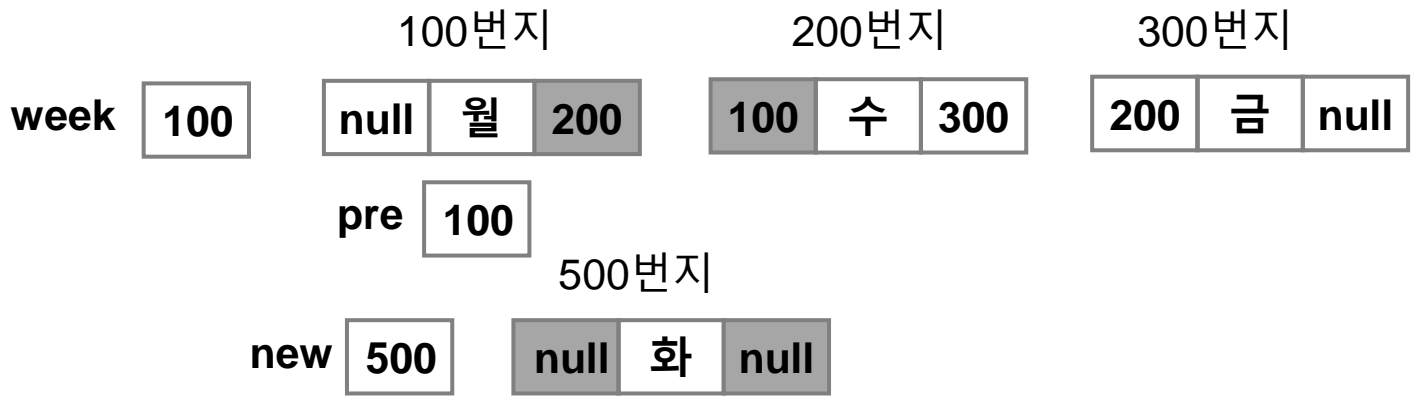
Q1:월요일과 수요일 사이에 화요일 노드를 삽입하려고 한다(pre 다음 노드에 신규 노드를 삽입) 수정되는 필드는 몇 개인가?

Q2: 링크필드를 어떤 순서로 수정해야 하는가?





# 이중 연결 리스트 - 삽입연산

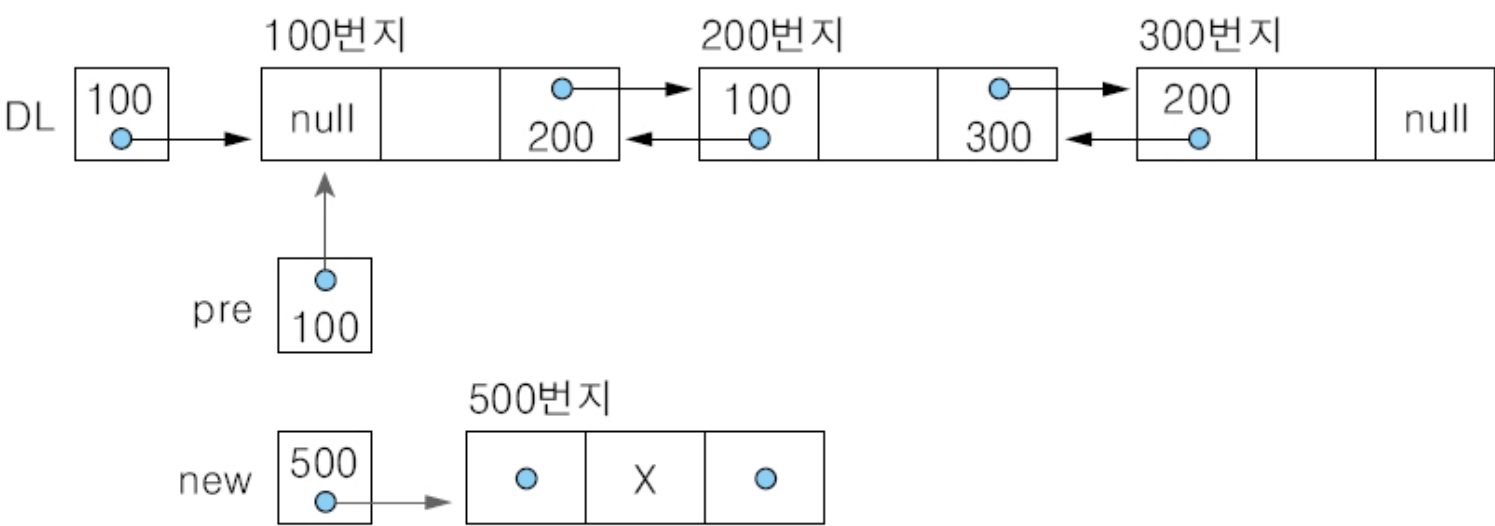


### ◆ 이중 연결 리스트에서의 삽입 연산 과정

- ① 삽입할 노드를 가져온다.
- ② 새 노드의 데이터 필드에 값을 저장한다.
- ③ 새 노드의 왼쪽 노드의 오른쪽 링크(rlink)를 새 노드의 오른쪽 링크(rlink)에 저장한다.
- ④ 그리고 왼쪽 노드의 오른쪽 링크(rlink)에 새 노드의 주소를 저장한다.
- ⑤ 새 노드의 오른쪽 노드의 왼쪽 링크(llink)를 새 노드의 왼쪽 링크(llink)에 저장한다.
- ⑥ 그리고 오른쪽 노드의 왼쪽 링크(llink)에 새 노드의 주소를 저장한다.

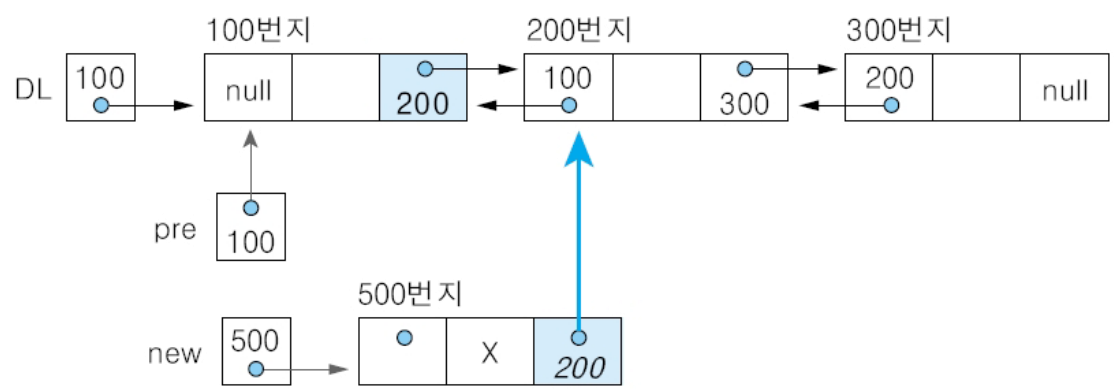
◆ 이중 연결 리스트에서의 삽입 연산

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



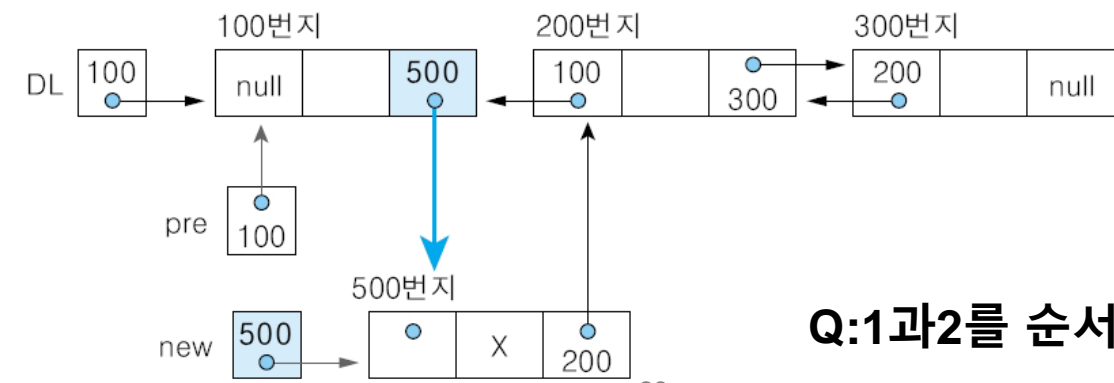
①  $\text{new.rlink} \leftarrow \text{pre.rlink};$

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



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

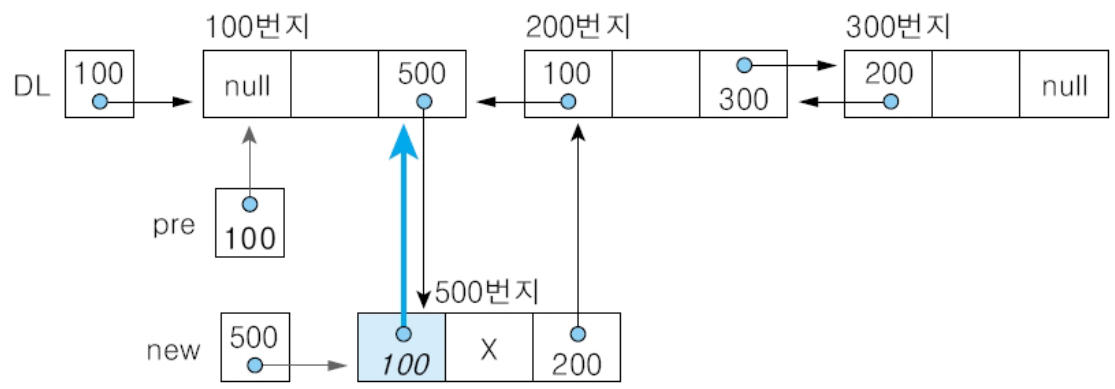
새 노드 new의 주소를 노드 pre의 rlink에 저장하여, 노드 new를 노드 pre의 오른쪽 노드로 연결



Q:1과2를 순서를 바꿔서 실행하면?

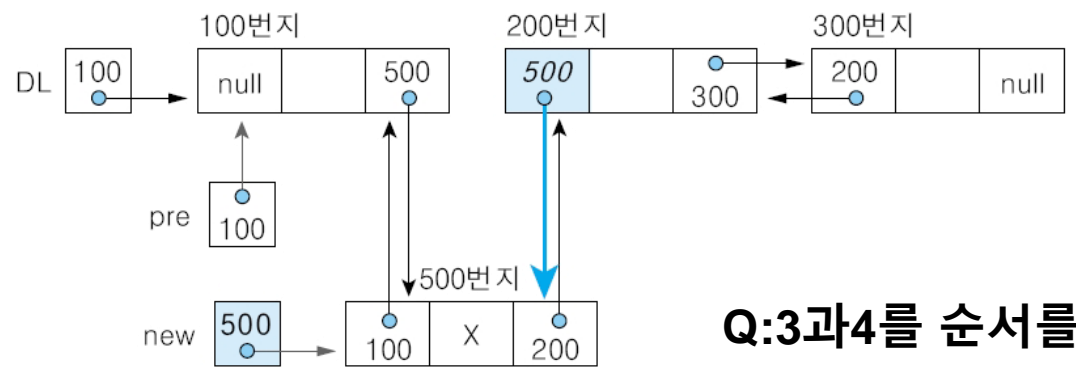
③ new.llink ← pre;

포인터 pre의 값을 삽입할 노드 new의 llink에 저장하여, 노드 pre를 노드 new의 왼쪽 노드로 연결



④ new.rlink.llink ← new;

포인터 new의 값을 노드 new의 오른쪽 노드(new.rlink)의 llink에 저장하여, 노드 new의 오른쪽 노드의 왼쪽 노드로 노드 new를 연결



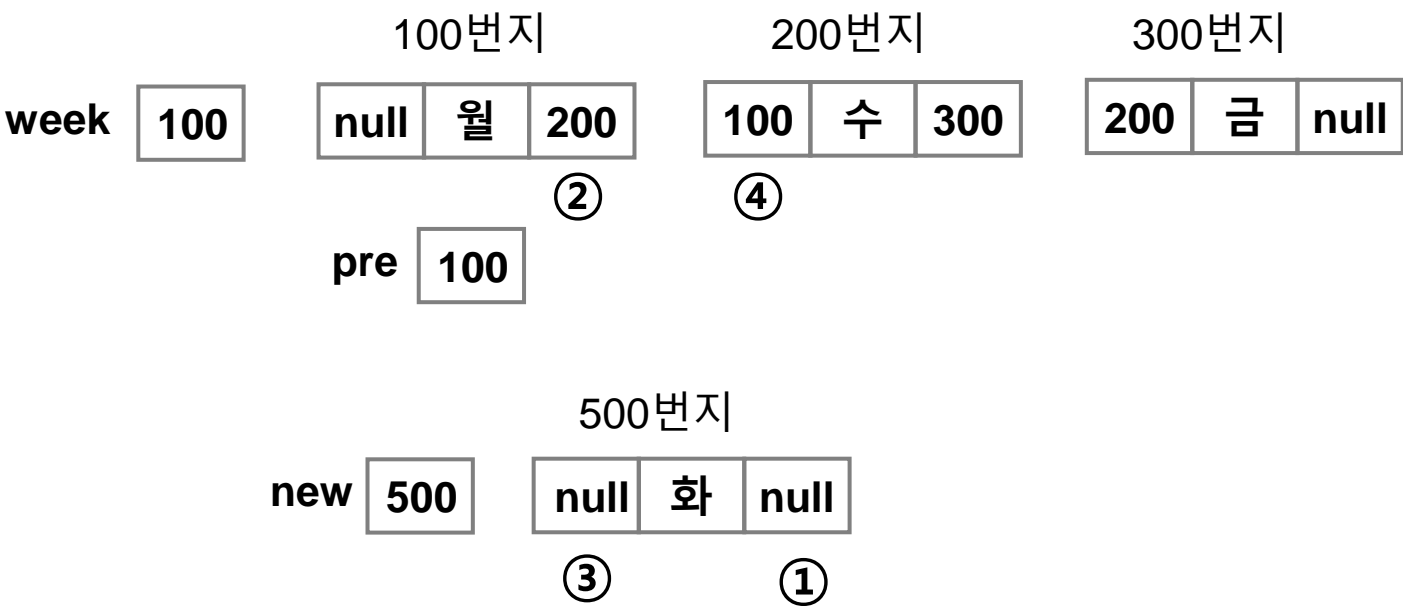
Q:3과4를 순서를 바꿔서 실행하면?

## ◆ 이중 연결 리스트에서의 삽입 알고리즘

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

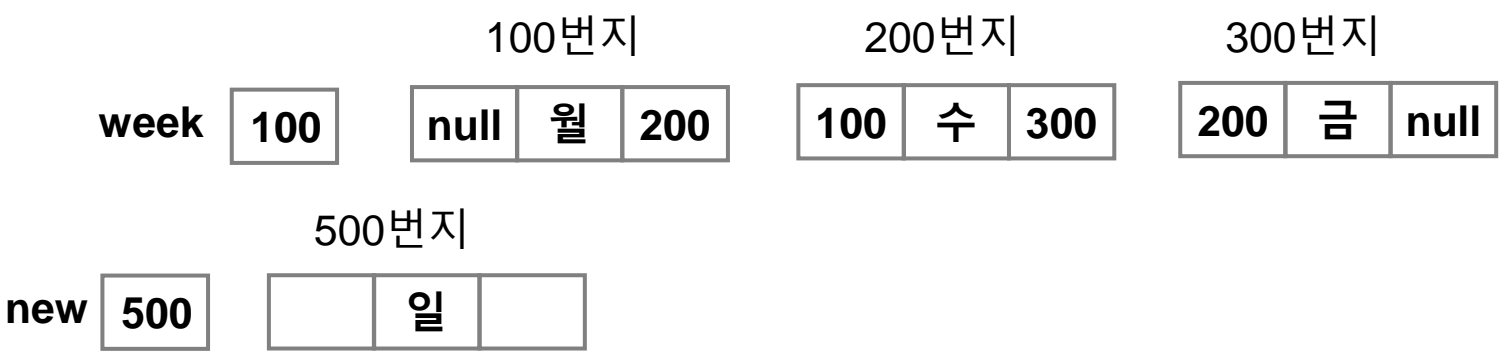
# Exercise

Q:교재에 나오는 순서 이외에 가능한 다른 순서들을 모두 찾으시오.

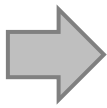


# 첫번째 노드로 삽입

Q:아래와 같이 첫번째 노드로 삽입하려고 할때, insertNode함수를 어떻게 수정해야 하는가?



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

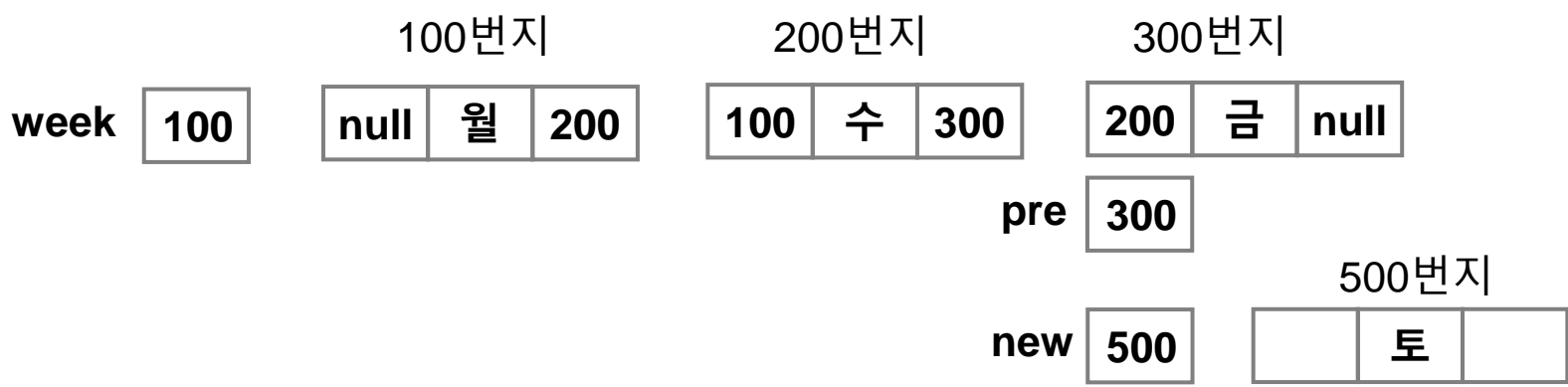


```
insertNode(DL, x)
    new ← getNode();
    new.data ← x;
    new.rlink ← week ;      ①
    week ← new;             ②
    new.llink ← null;       ③
    new.rlink.llink ← new;  ④
end insertNode()
```



# 마지막 노드로 삽입

Q:아래와 같이 마지막 노드로 삽입하려고 할때, insertNode함수를 어떻게 수정해야 하는가?



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

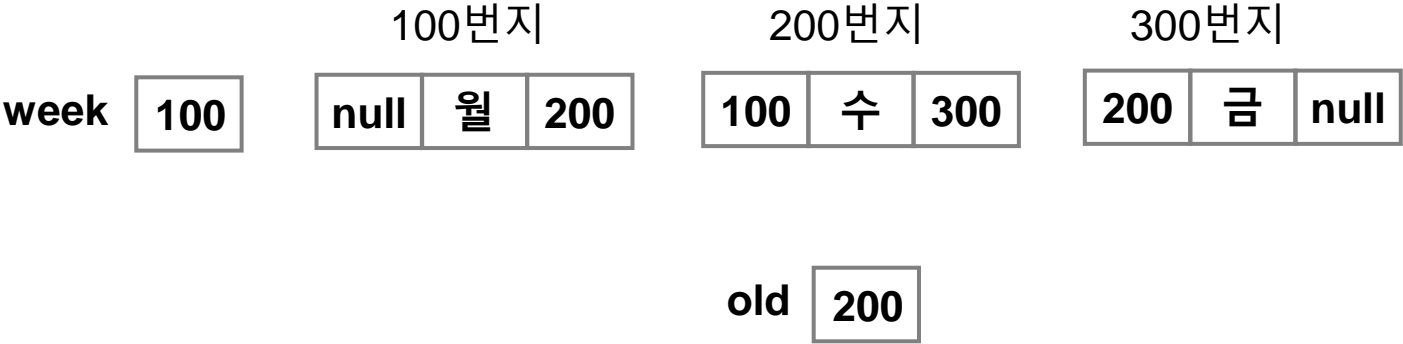


```
insertNode(DL, pre, x)
  new ← getNode();
  new.data ← x;
  new.rlink ← pre.rlink ; ①
  pre.rlink ← new;        ②
  new.llink ← pre;         ③
end insertNode()
```

① `new.rlink ← pre.rlink` 은 어떻게 변경해도 되는가?

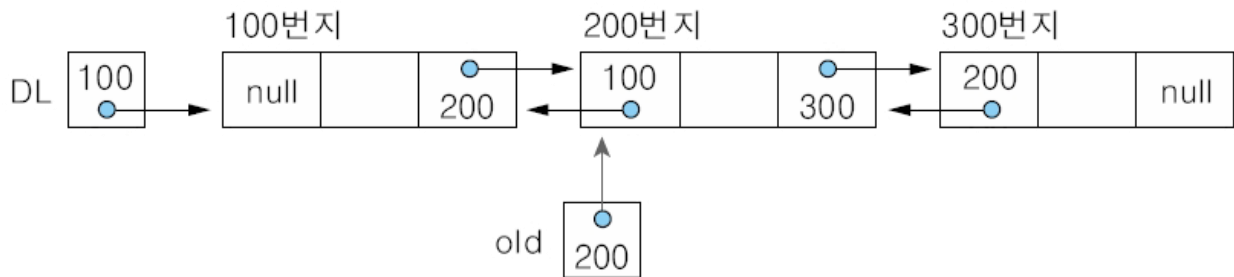
# 이중 연결 리스트 - 삭제연산

- Q1:아래의 이중연결리스트에서 old가 가리키는 노드를 삭제하려고 할 때, 수정되는 링크필드는 몇 개인가?
- Q2: 삭제할 노드의 직전노드를 가리키는 포인터변수가 필요한가?
- Q3: 링크필드는 어떤 순서로 수정되어야 하는가?



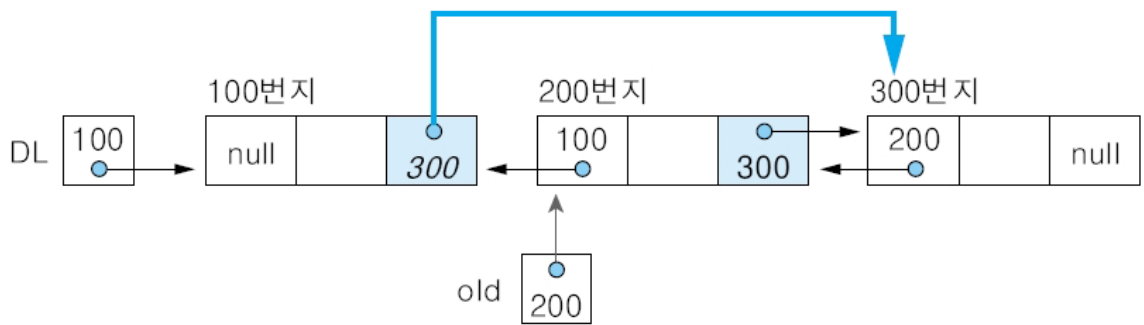
◆ 이중 연결 리스트에서의 삭제 연산

이중 연결 리스트 DL에서 포인터 old가 가리키는 노드를 삭제하는 과정



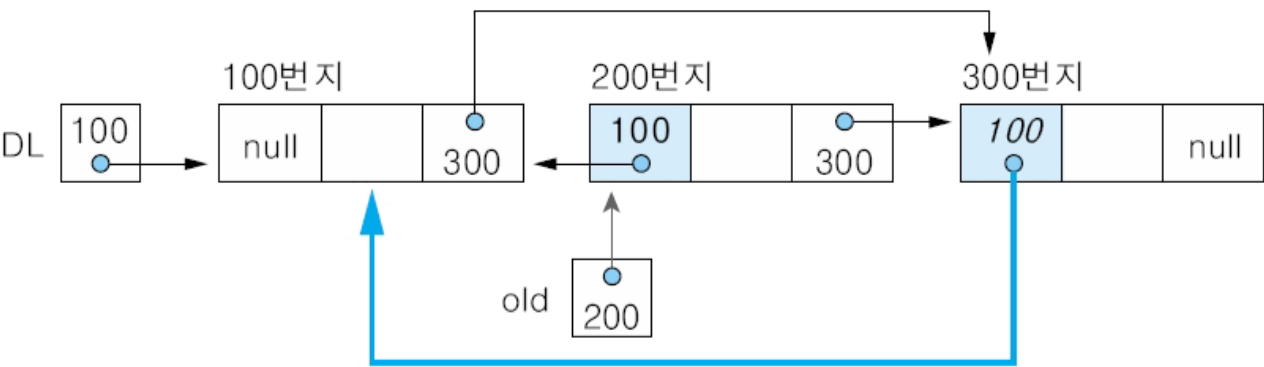
① `old.llink.rlink ← old.rlink;`

삭제할 노드 old의 오른쪽 노드의 주소를 노드 old의 왼쪽 노드의 rlink에 저장하여, 노드 old의 오른쪽 노드를 노드 old의 왼쪽 노드의 오른쪽 노드로 연결



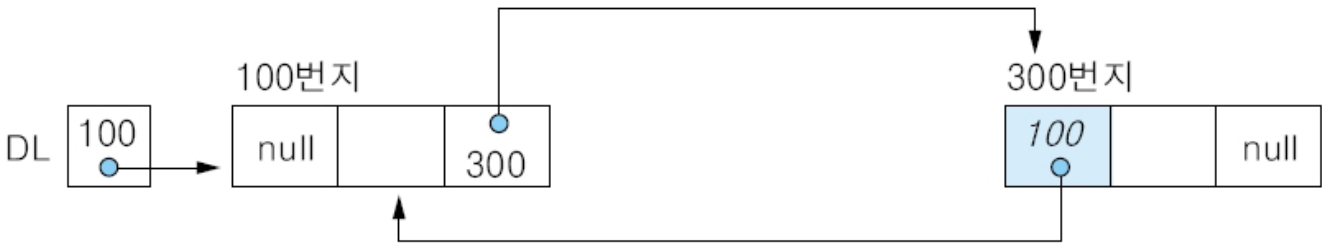
② `old.rlink.llink ← old.llink;`

삭제할 노드 `old`의 왼쪽 노드의 주소를 노드 `old`의 오른쪽노드의 `llink`에 저장하여, 노드 `old`의 왼쪽 노드를 노드 `old`의 오른쪽 노드의 왼쪽 노드로 연결



③ `returnNode(old);`

삭제된 노드 `old`는 자유공간리스트에 반환



### ◆ 이중 연결 리스트에서의 삭제 연산 과정

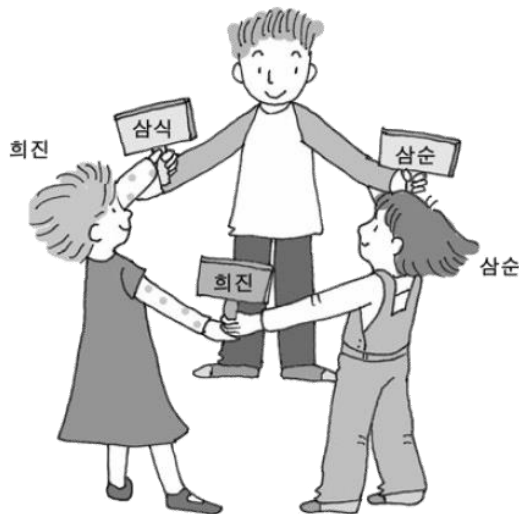
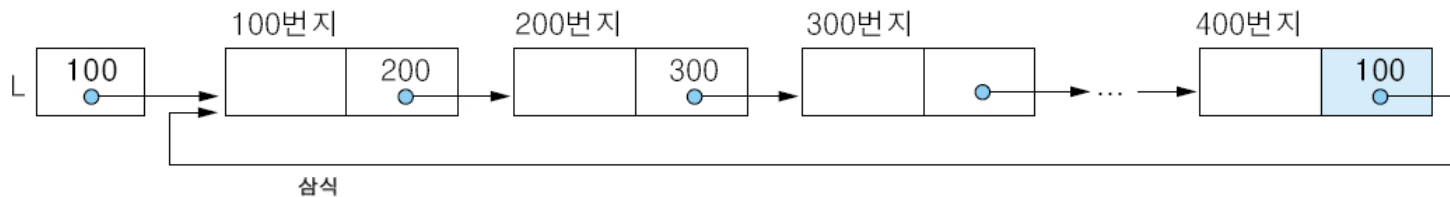
- (1) 삭제할 노드의 오른쪽 노드의 주소(old.rlink)를 삭제할 노드의 왼쪽 노드 (old.llink)의 오른쪽 링크(rlink)에 저장한다.
- (2) 삭제할 노드의 왼쪽 노드의 주소(old.llink)를 삭제할 노드의 오른쪽 노드 (old.rlink)의 왼쪽 링크(llink)에 저장한다.
- (3) 삭제한 노드를 자유 공간 리스트에 반환한다.

### ◆ 이중 연결 리스트에서의 삭제 알고리즘

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

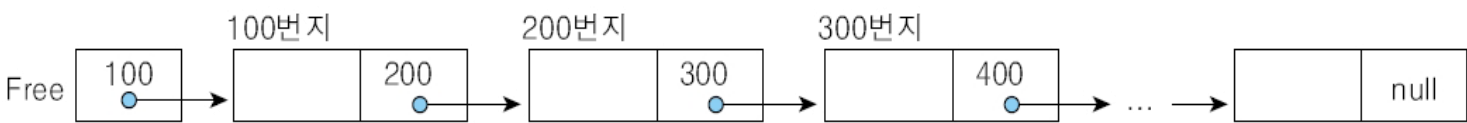
## ◆ 원형 연결 리스트(circular linked list)

- 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



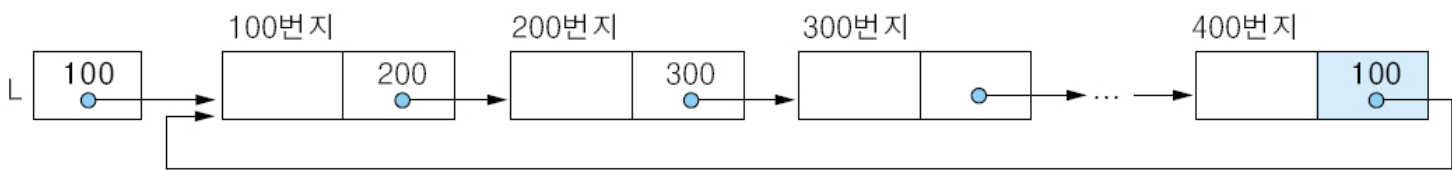
## ◆단순 연결 리스트

- 노드를 탐색하기 위해서는 항상 첫번째 노드로 이동해서 탐색



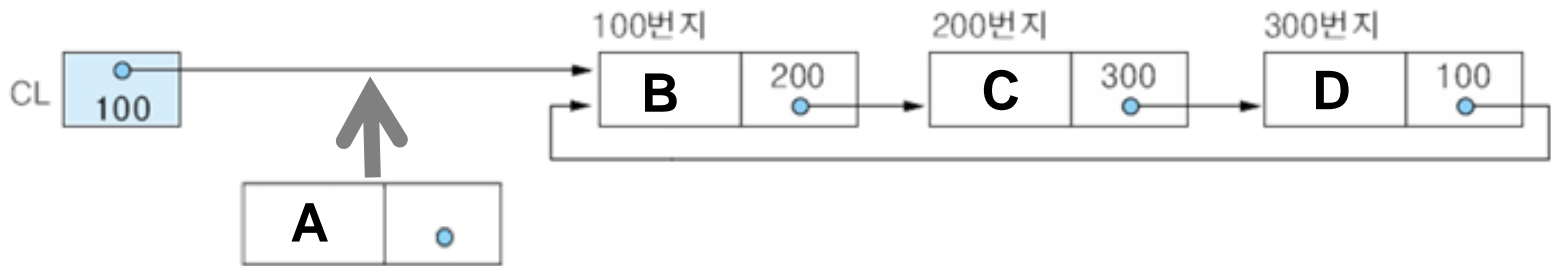
## ◆원형 연결 리스트

- 노드를 탐색하기 위해서 어느노드부터 출발해도 상관없음.



## 원형 연결 리스트 - 삽입연산

Q: 아래의 원형 연결리스트의 첫번째 자리에 'A' 값을 갖고 있는 신규노드를 삽입하는 방법은?



Q1: 총 몇 개의 필드가 값이 바뀌는가?

Q2: 삽입할 노드의 앞노드는 누구인가?

Q3: 삽입할 노드의 앞노드는 어떻게 찾아야 하는가?

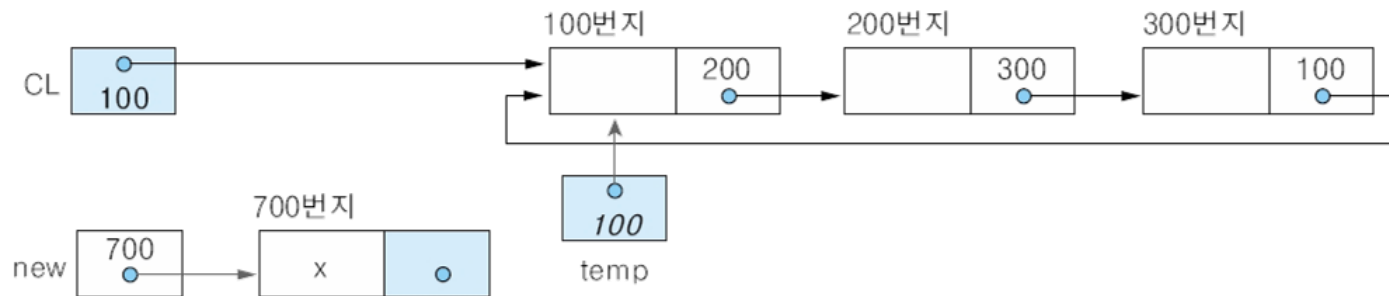
➔ link필드 값이 CL의 값과 같은 노드를 찾는다!



## ◆첫번째 노드로 삽입하기

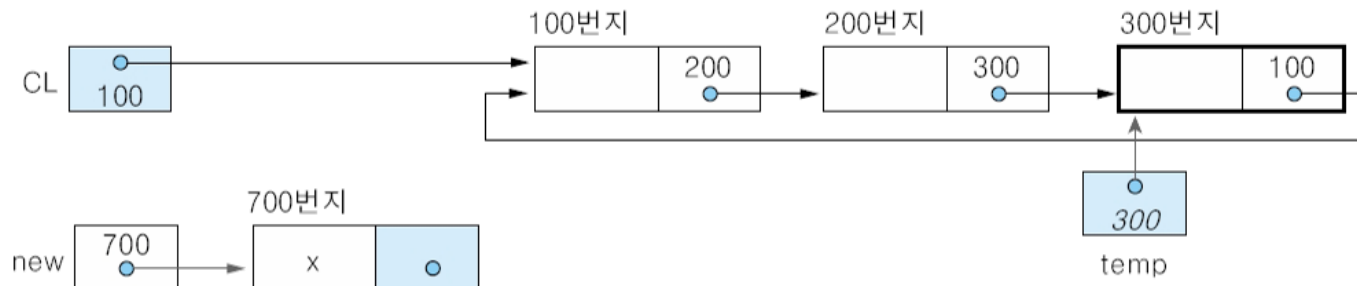
### ① $temp \leftarrow CL;$

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



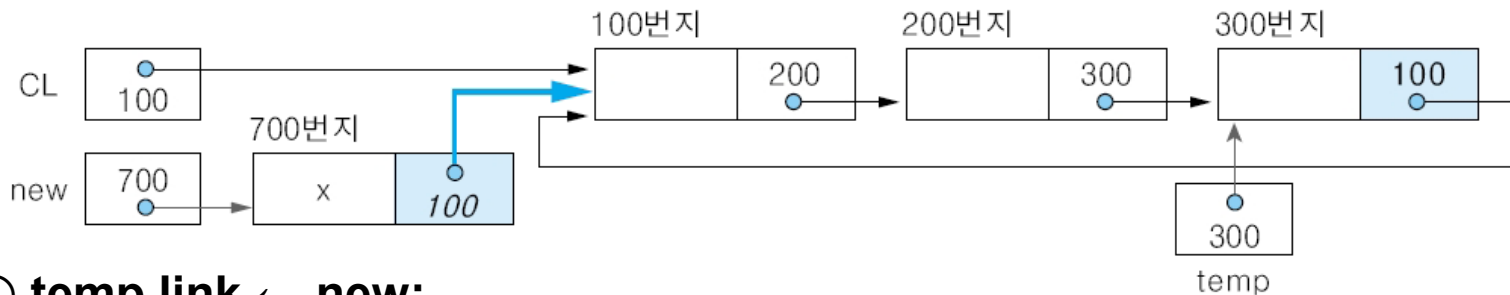
### ② $while (temp.link \neq CL) \text{ do } temp \leftarrow temp.link;$

$while$  반복문을 수행하여 순회 포인터  $temp$ 를 링크를 따라 마지막 노드까지 이동



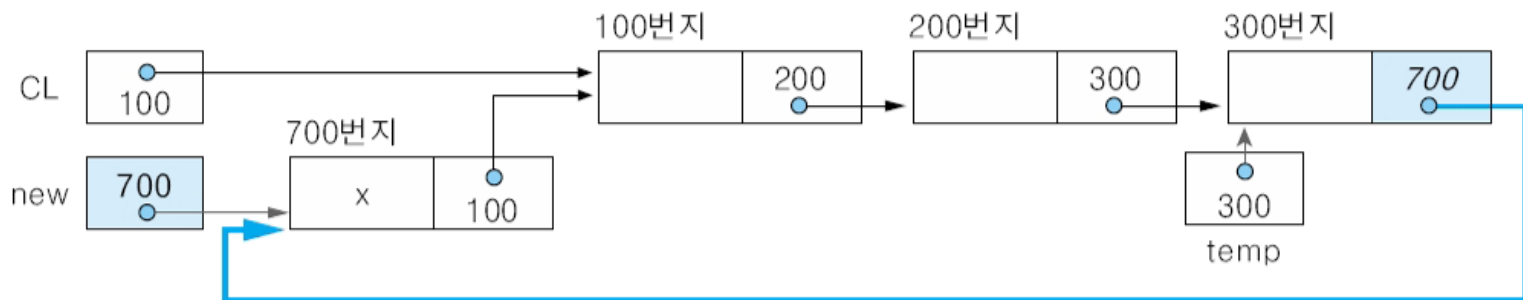
## ③ new.link ← temp.link;

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.



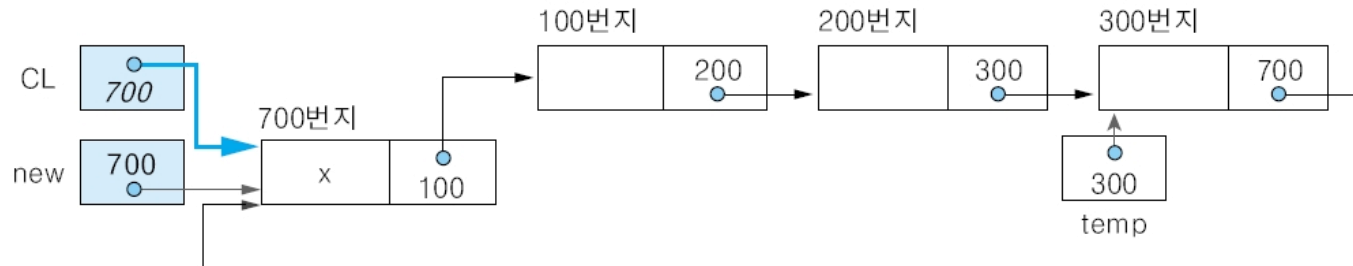
## ④ temp.link ← new;

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



⑤  $CL \leftarrow new$ ;

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



❖ 원형 연결 리스트의 첫 번째 노드 삽입 연산 결과



## ◆첫번째 노드로 삽입하기: 공백리스트인 경우



Q: 삽입이 완료된 후의 모습은?

① `new.link ← new;`

원형연결리스트에서 노드가 하나이면, 링크필드가 자기 자신을 가리켜야 함.

② `CL ← new;`

Q: 위의 순서는 바뀌어도 되는가?

### ◆ 원형 연결 리스트의 첫번째 노드로의 삽입 연산

- 마지막 노드의 링크를 첫 번째 노드로 연결하는 부분만 제외하고는 단순 연결 리스트에서의 삽입 연산과 같은 연산
- 원형 연결 리스트 CL에 x 값을 갖는 노드 new를 첫번째에 삽입하는 알고리즘

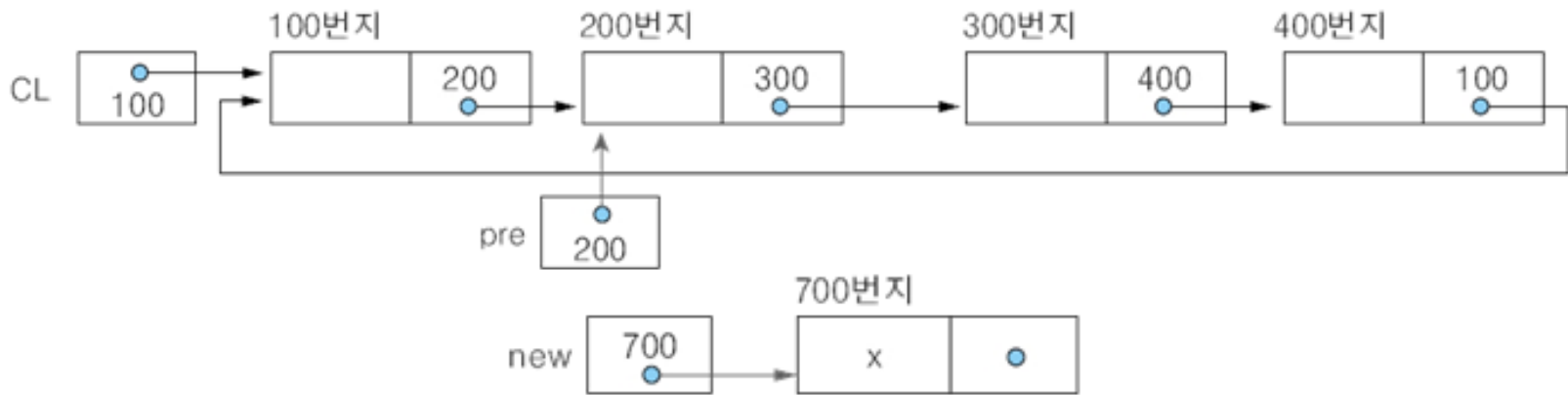
```
insertFirstNode(CL, x)
    new ← getNode();
    new.data ← x;
    if (CL = null) then {           // ① 공백 원형연결리스트인 경우
        CL ← new;                  // ①-a
        new.link ← new;            // ①-b
    }
    temp ← CL;                      // ②
    while (temp.link ≠ CL) do       // ③ 마지막 노드까지 temp포인터 이동
        temp ← temp.link;
    new.link ← temp.link;           // ④
    temp.link ← new;                // ⑤
    CL ← new;                       // ⑥
end insertFirstNode()
```

Q: ①~ ⑥ 중에서 순서가 다른곳으로 바뀌어도 되는 라인은?

```
insertFirstNode(CL, x)
  new ← getNode();
  new.data ← x;
  if (CL = null) then {           // ① 공백 원형연결리스트인 경우
    CL ← new;                     // ①-a
    new.link ← new;              // ①-b
  }
  temp ← CL;                     // ②
  while (temp.link ≠ CL) do      // ③ 마지막 노드까지 temp포인터 이동
    temp ← temp.link;
  new.link ← temp.link;         // ④
  temp.link ← new;              // ⑤
  CL ← new;                     // ⑥
end insertFirstNode()
```

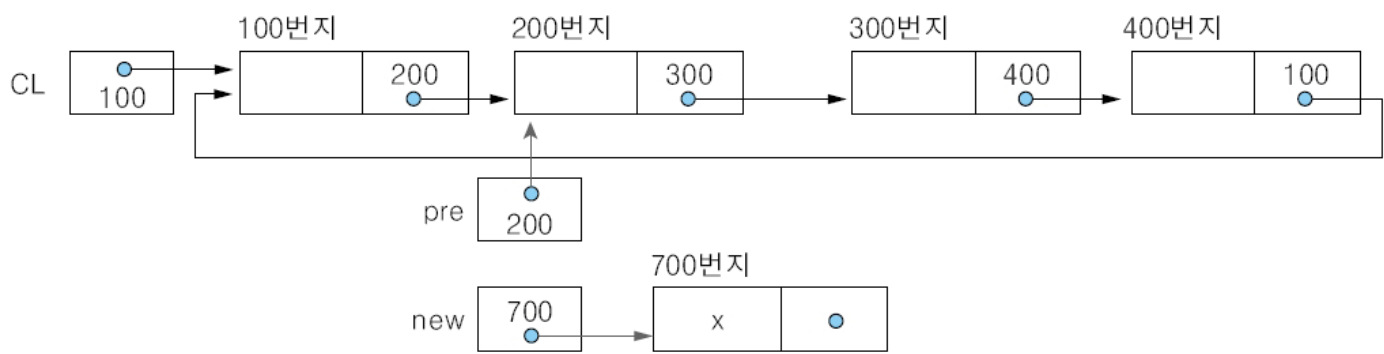
## ◆ 중간 노드로 삽입하기

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

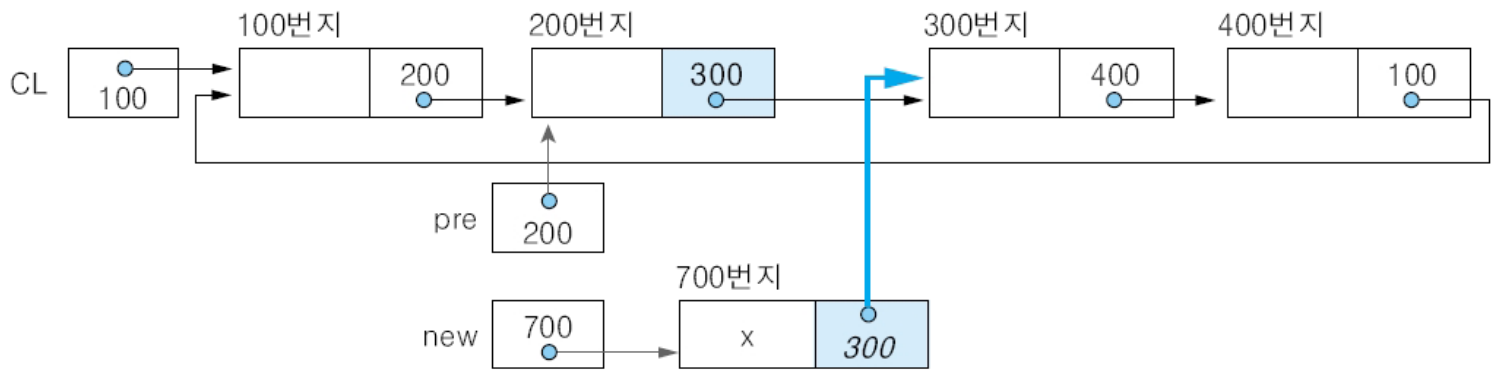


Q: 단순연결리스트에서의 중간 노드로 삽입과 원형연결리스트에서의 중간 노드로 삽입은 어떤 차이점을 가지고 있는가?

# 원형 연결 리스트 - 삽입연산



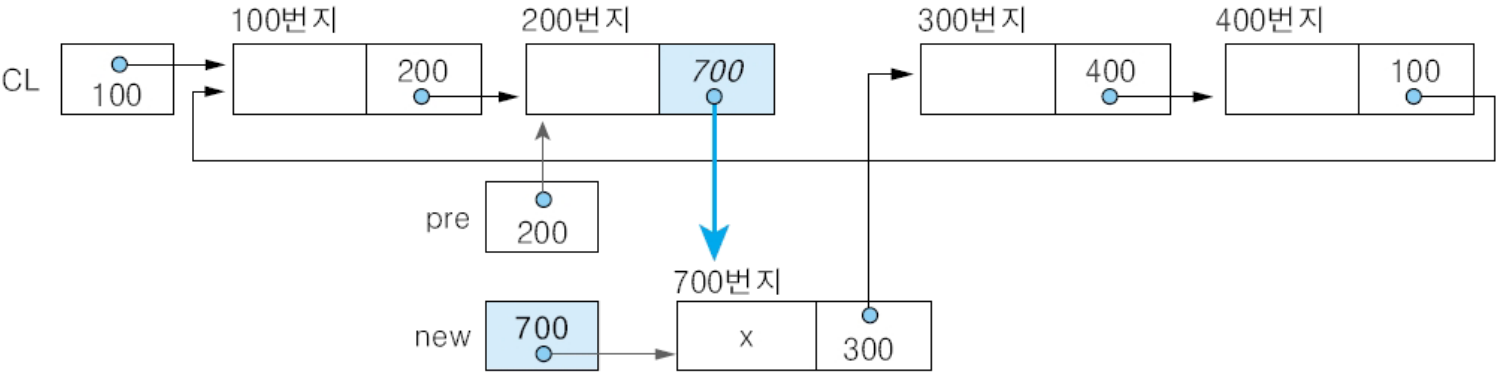
① **new.link ← pre.link;**  
노드 pre의 다음 노드로 new를 삽입하기 위해서, 먼저 노드 pre의 다음 노드(pre.link)를 new의 다음 노드(new.link)로 연결





② `pre.link ← new;`

노드 new의 주소를 노드 pre의 링크에 저장하여, 노드 pre가 노드 new를 가리키게 한다.



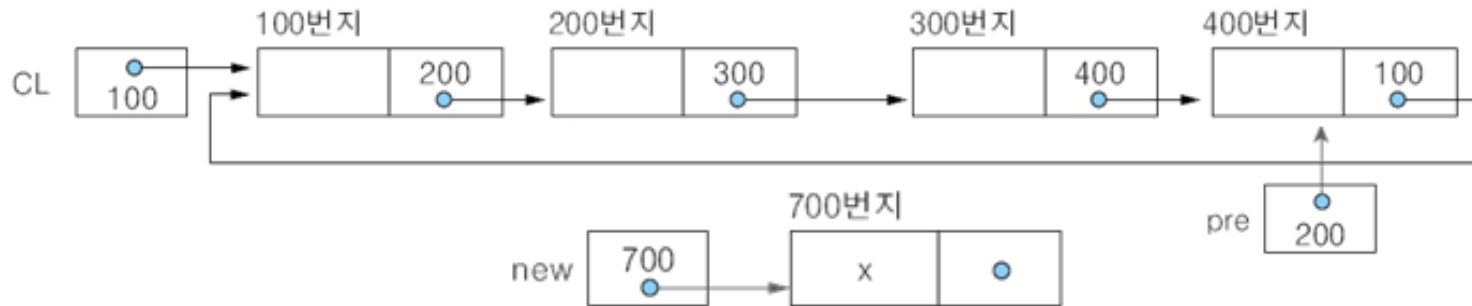
Q: 아래 코드에서 ②-㉠와 ②-㉡에 들어갈 명령은?

```
insertMiddleNode(CL, pre, x)
    new ← getNode();
    new.data ← x;
    if (CL=null) then {           // ①
        CL ← new;
        new.link ← new;
    }
    else {                       // ②
        new.link ← pre.link;    // ㉠
        pre.link ← new;        // ㉡
    }
end insertMiddleNode()
```

# Discussion

Q: 아래 코드를 사용하여 마지막 위치에 신규노드를 삽입을 수행할 수 있는가?

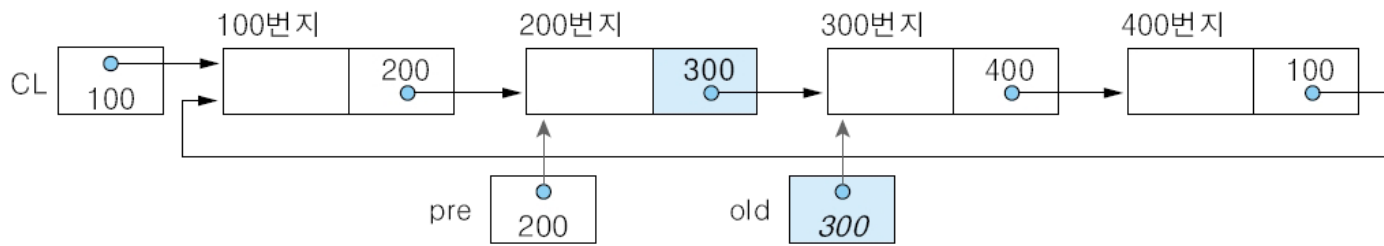
Q: 원형연결리스트에서 중간노드로의 삽입과 마지막노드로의 삽입은 어떤 차이가 있는가?



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

## ◆ 원형 연결 리스트의 삭제 연산

- 원형 연결 리스트 CL에서 포인터 pre가 가리키는 노드의 다음 노드를 삭제하고 삭제한 노드는 자유공간 리스트에 반환하는 연산

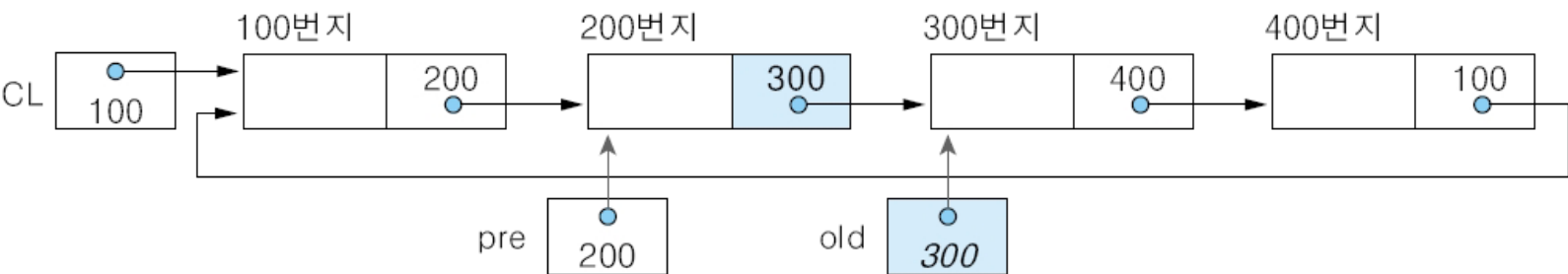


- Q: 중간노드를 삭제하는 경우와 마지막노드를 삭제하는 경우 어떤 차이점이 있는가?
- Q: 중간노드를 삭제하는 경우와 첫번째 노드를 삭제하는 경우 어떤 차이점이 있는가?

## ◆ 삭제할 노드가 리스트의 중간 또는 마지막인 경우

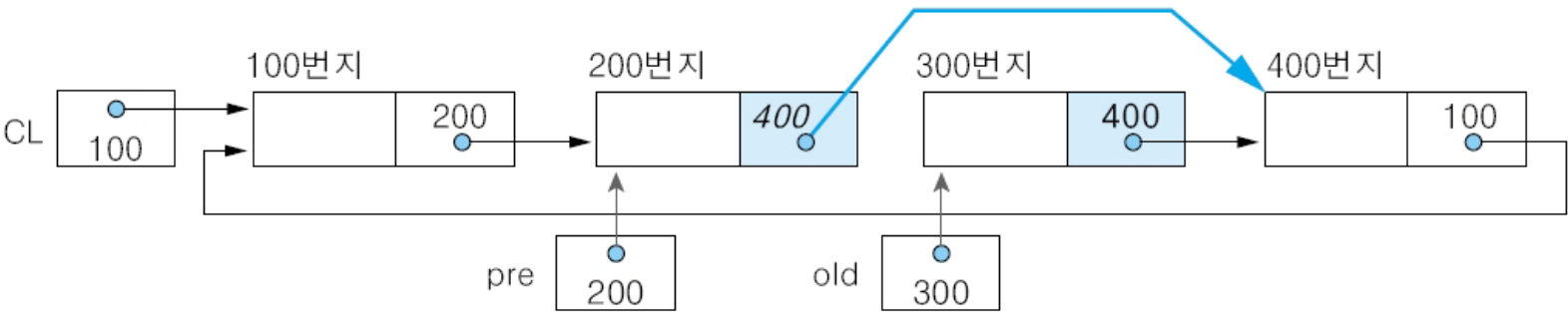
①  $old \leftarrow pre.link;$

노드 pre의 다음 노드를 삭제할 노드 old로 지정

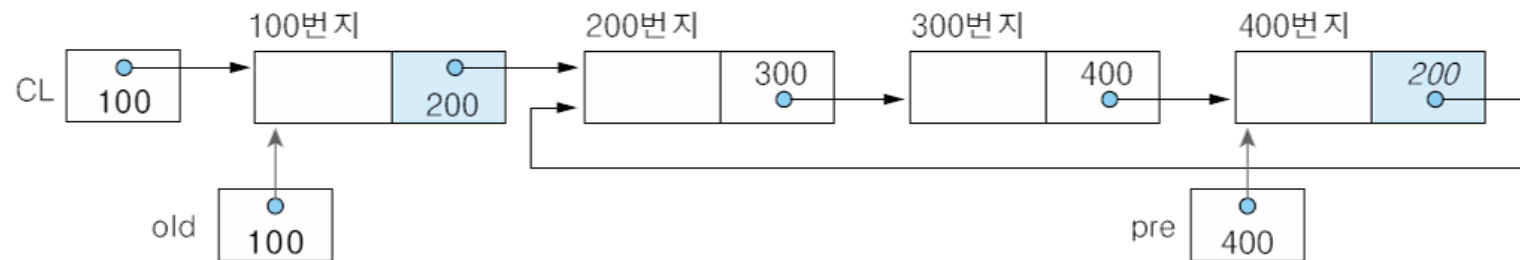


②  $pre.link \leftarrow old.link;$

노드 old의 이전 노드와 다음 노드를 서로 연결

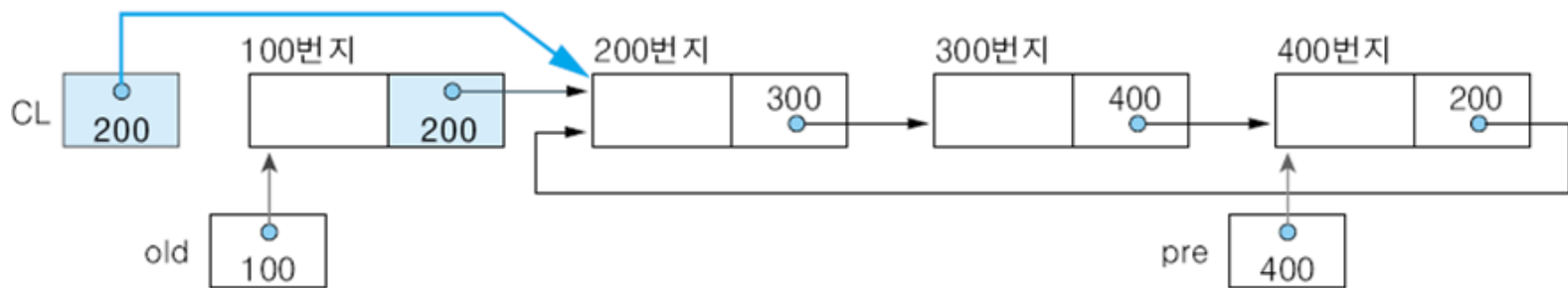


◆ 삭제할 노드가 리스트의 첫번째 노드인 경우 (CL=old)



③ 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



Q: 아래의 코드는 모두 몇 가지의 상황을 고려하고 있는가?

- ① 공백리스트인 경우
- ② 중간 또는 마지막 노드를 삭제하는 경우
- ③ 첫번째 노드를 삭제하는 경우

```
deleteNode(CL, pre)
  if (CL = null) then error;      // ①
  else {
    old ← pre.link;              // ②
    pre.link ← old.link;
    if (old = CL) then           // ③
      CL ← old.link;
    returnNode(old);
  }
end deleteNode()
```