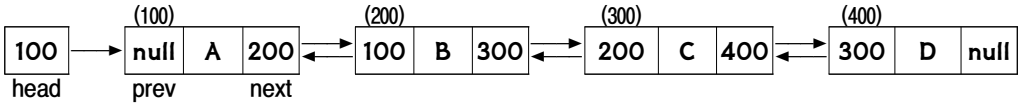
13. 이중연결리스트(doubly linked list)

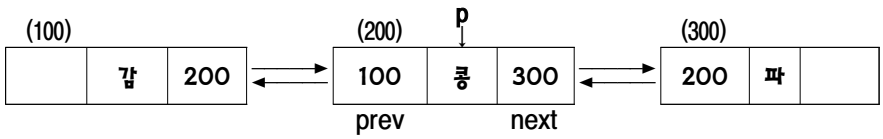
이중연결리스트는 각 노드에 2개의 포인터가 있다. 각 포인터는 좌우측 노드를 가리킨다.

```
typedef struct node node;
struct node
{
    node *prev;      //앞의 노드를 가리키는 포인터
    char data[5];
    node *next;      //다음 노드를 가리키는 포인터
};
```



- 이중연결리스트는 현재 노드에서 순방향과 역방향으로 검색이 가능하다.
- 이중연결리스트는 양방향으로 자료를 처리해야 하는 문제에 적용하기 편리한 구조이다.
- 이중연결리스트는 노드 삽입, 삭제가 쉽다.
- 이중연결리스트는 임의 노드의 포인터가 손상되어도 복구할 수 있다.

◈ 이중연결리스트에서 포인터 값



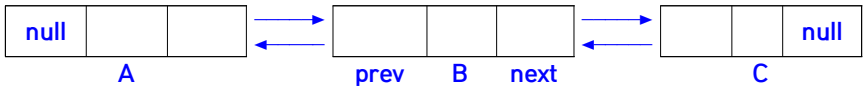
```
p = 200;
p->prev->next = 200;
p->next->prev = 200;
```

위의 구조에서 이 3가지 표현의 포인터 값은 같다.

- 해서, 어떤 포인터가 파괴되어도 복구가 가능하고, 삽입/삭제도 편리하게 된다.

기출문제 분석

1. 노드 A, B, C를 가지는 이중연결리스트에서 노드 B를 삭제하기 위한 의사코드(pseudo code)로 옳지 않은 것은? (단, 노드 B의 메모리는 해제하지 않는다) [2017년 국가 9급]

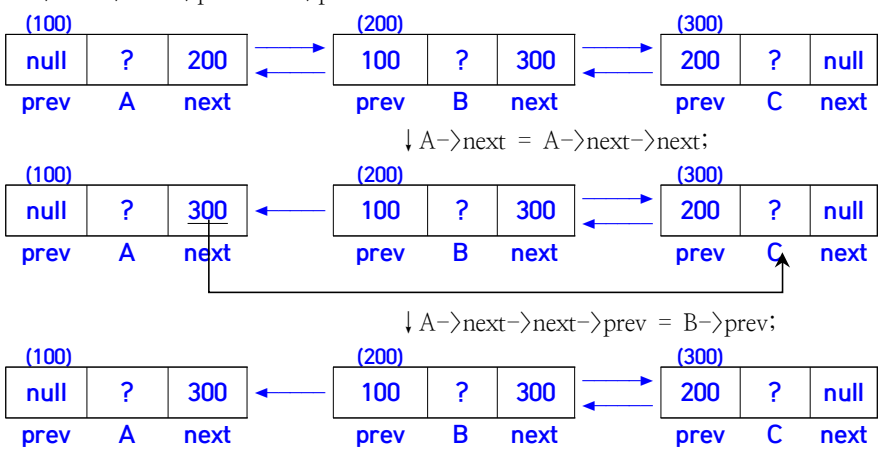


① A->next = C;
C->prev = A;
② A->next = B->next;
C->prev = B->prev;
③ B->prev->next = B->next;
B->next->prev = B->prev;
④ A->next = A->next->next;
A->next->next->prev = B->prev;

♤ **이중연결리스트에서 노드 B를 삭제**

• 먼저, 주어진 문제에서 A, B, C는 노드 A, B, C를 가리키는 주소로 사용되었다.

④ A->next = A->next->next;
A->next->next->prev = B->prev;



• A->next = 300으로 변경되어 있으므로, A->next->next = null이 된다.
• 해서, A->next->next->prev는 성립하지 않는다. 노드 B 삭제 불가

정답 : ④

2. 단순연결리스트 대신 이중연결리스트를 사용하는 이유로 적합한 것은? [2003년 국가 7급]

① 기억장소를 절약하기 위해서
② 포인터가 파괴되었을 때 복구하기 위해서
③ 알고리즘 구현을 간단히 하기 위해서
④ 스택이나 큐를 쉽게 구현하기 위해서

♤ **이중연결리스트**

- 이중연결리스트는 단순연결리스트에 비해 알고리즘 구현이 단순하다.
- 이중연결리스트는 2개의 포인터를 사용한다고 알고리즘 구현이 더 복잡한 것은 아니다.
- 이중연결리스트는 **양방향**으로 추적이 **가능**하다.
- 단순연결리스트는 **양방향**으로 추적이 **불가능**하다.
- 이중연결리스트는 어떤 포인터가 파괴되어도 복구가 가능하고, 삽입/삭제도 편리하게 된다.
- 이중연결리스트는 어떤 포인터 파괴는 특별한 경우이다.

정답 : ③

3. 연결리스트에 대한 설명으로 옳지 않은 것은? [2001년 국가 기술고시]

① 단순연결리스트(singly linked list)에서는 한쪽 방향으로만 이동이 가능하다.
② 연결리스트(linked list)는 원형(circular)이 아닐 수도 있다.
③ 일반적으로 이중연결리스트의 노드는 적어도 3개의 필드를 가진다.
④ 이중연결리스트는 단순연결리스트보다 모든 면에서 우수하다.
⑤ 연결리스트는 헤드노드(head node)를 갖도록 구성할 수 있다.

♤ **연결리스트**

- 이중연결리스트가 모든 면에서 우수한 것은 아니다.
- 예를 들면, 2개의 포인터 필드가 있으므로 기억공간을 그 만큼 더 필요로 한다.

정답 : ④