<자료구조>

6. 연결 리스트 I

한국외국어대학교
컴퓨터.전자시스템공학전공
2016년 1학기
고 석 훈

학습 목표

- 연결 자료구조를 이해한다.
- 순차 자료구조와 연결 자료구조의 차이점을 알아본다.
- 연결 리스트의 종류와 특징을 알아본다.

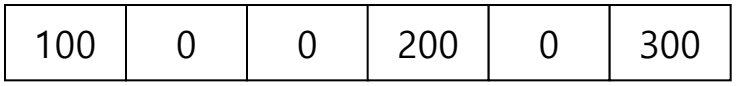
순차 자료구조의 문제점

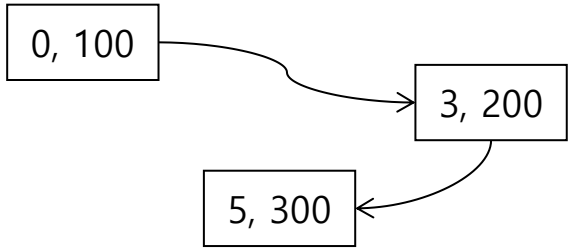
- 순차 자료구조(sequential data structure)
  - 원소들의 논리적 순서와 원소들이 저장된 물리적 순서가 동일
  - 삽입, 삭제 연산 후에 연속적인 물리 주소를 유지하기 위해 원소들을 이동시키는 오버헤드 발생
  - 순차 자료구조가 사용하는 배열의 메모리 비효율성 문제 발생

➔ 순차 자료구조에서의 연산 시간에 대한 문제와
저장 공간에 대한 문제를 개선한 자료 표현 방법 필요

연결 자료구조

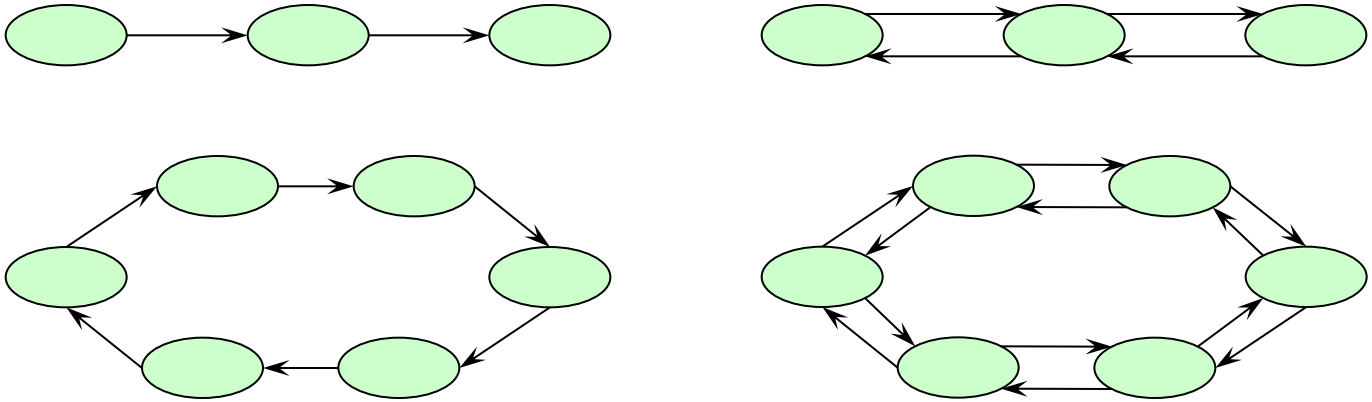
- 자료의 논리적 순서와 물리적 순서가 일치하지 않는 자료구조
  - 물리적으로 입체적인 다양한 형태를 갖는다.
- 각 원소에 다음 원소의 주소를 저장하여 연결되는 방식
  - 물리적인 순서를 맞추기 위한 오버헤드가 발생하지 않는다.
- 여러 개의 작은 공간을 연결하여 하나의 전체 자료구조를 표현
  - 크기 변경이 유연하고 더 효율적으로 메모리를 사용한다.





연결 리스트

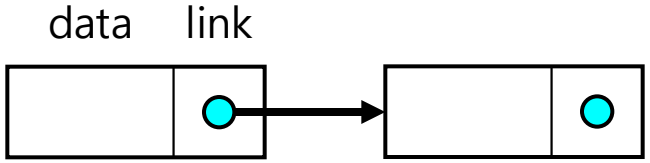
- 연결 리스트(linked list)
  - 리스트를 연결 자료구조로 표현한 구조
  - 연결하는 방식에 따라 구분
    - 단순 연결 리스트, 원형 연결 리스트
    - 이중 연결 리스트, 이중 원형 연결 리스트



연결 자료구조의 단위

- 노드(node)

- 노드의 구조
  -
    - 원소의 값을 저장
    - 저장할 원소의 형태에 따라 하나 이상의 필드로 구성

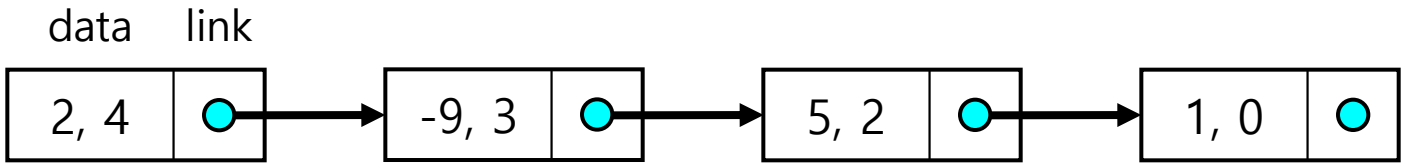


  -
    - 포인터 변수를 사용하여 다음 노드의 주소를 저장
    - 필요에 따라 2개 이상의 링크 필드를 가질 수 있음
    - 포인터(pointer), 참조(reference)라고도 함

단순 연결 리스트

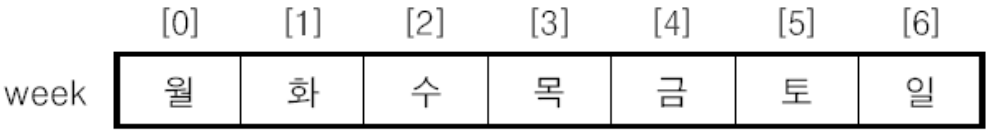
- 단순 연결 리스트(singly linked list)

  - 선형 연결 리스트(linear linked list),
    단순 연결 선형 리스트(singly linked linear list)라고도 한다.

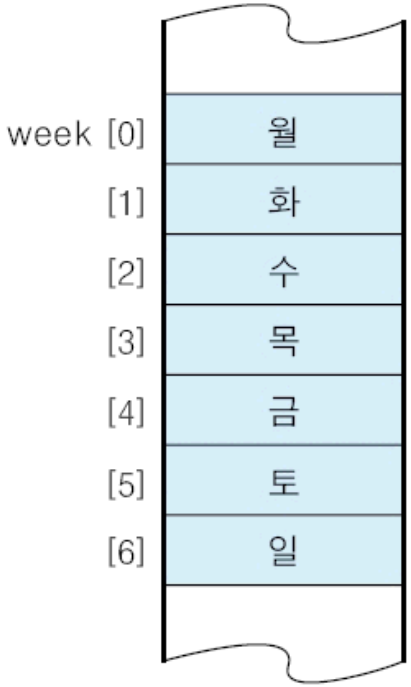


선형 리스트와 연결 리스트의 비교 [1/2]

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



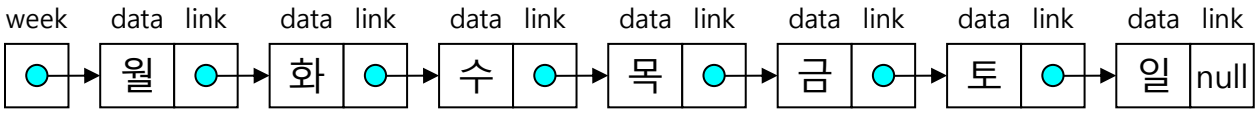
(a) 논리구조



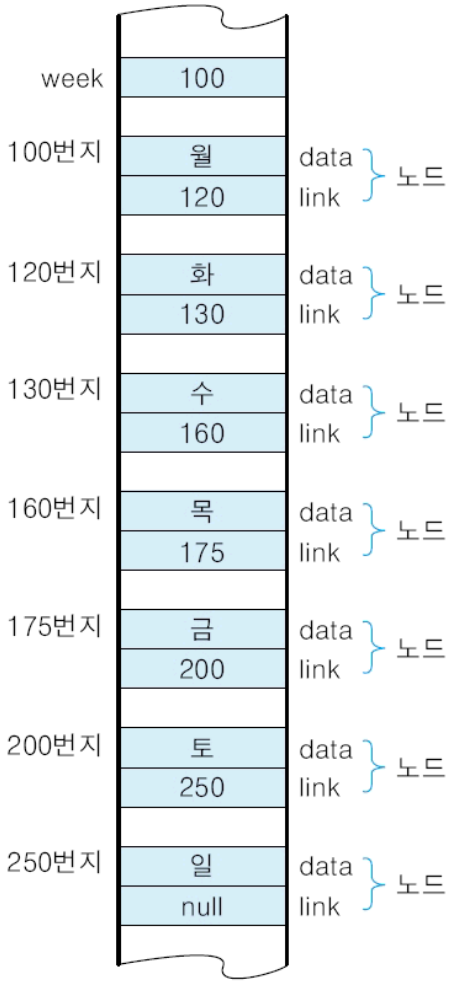
(b) 물리구조

선형 리스트와 연결 리스트의 비교 [2/2]

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



(a) 논리구조

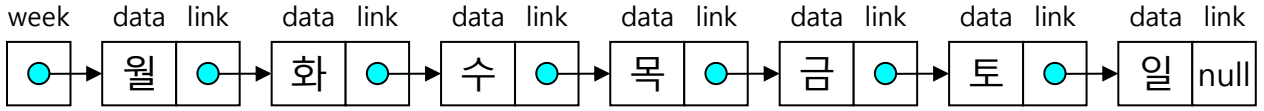


(b) 물리구조

  - 단순 연결 리스트의 구성
    - 리스트 이름 : 연결 리스트의 시작을 가리키는 포인터 변수, 연결 리스트 전체를 의미함
    - 마지막 노드의 링크 필드는 리스트의 끝을 의미하는 null 저장

단순 연결 리스트의 구성

- 리스트 week의 노드에 대한 C 프로그램 구조체 정의

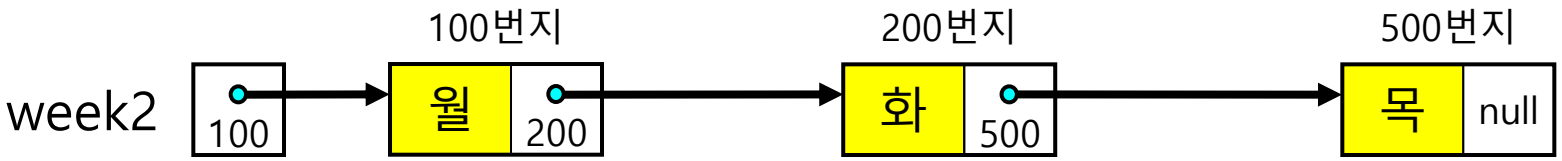


- week : 리스트의 첫번째 노드의 주소, 100
- week->data :
  week가 가리키는 노드의
  데이터 필드 값, "월"
- week->link :
  week가 가리키는 노드의
  링크 필드에 저장된 주소, 120
- week->link->data :
  week가 가리키는 노드의 링크 필드가
  가리키는 노드의 데이터 필드 값, "화"

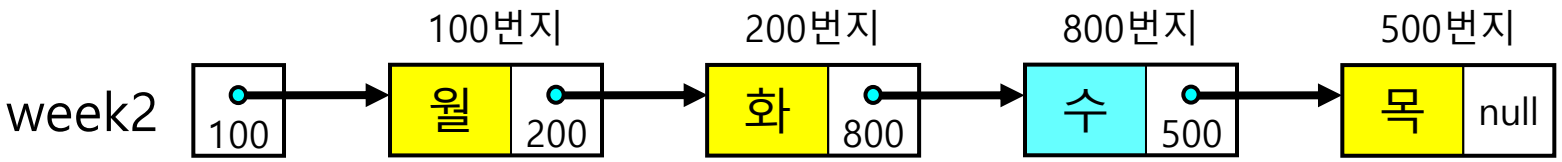
```
typedef struct Node {
    char   data[4];
    struct Node* link;
} node;

node *week;
```

단순 연결 리스트에 노드 삽입 [1/5]

- week2 = (월, 화, 목)에 "화"과 "목"사이에 원소 "수" 삽입
  - 초기 상태

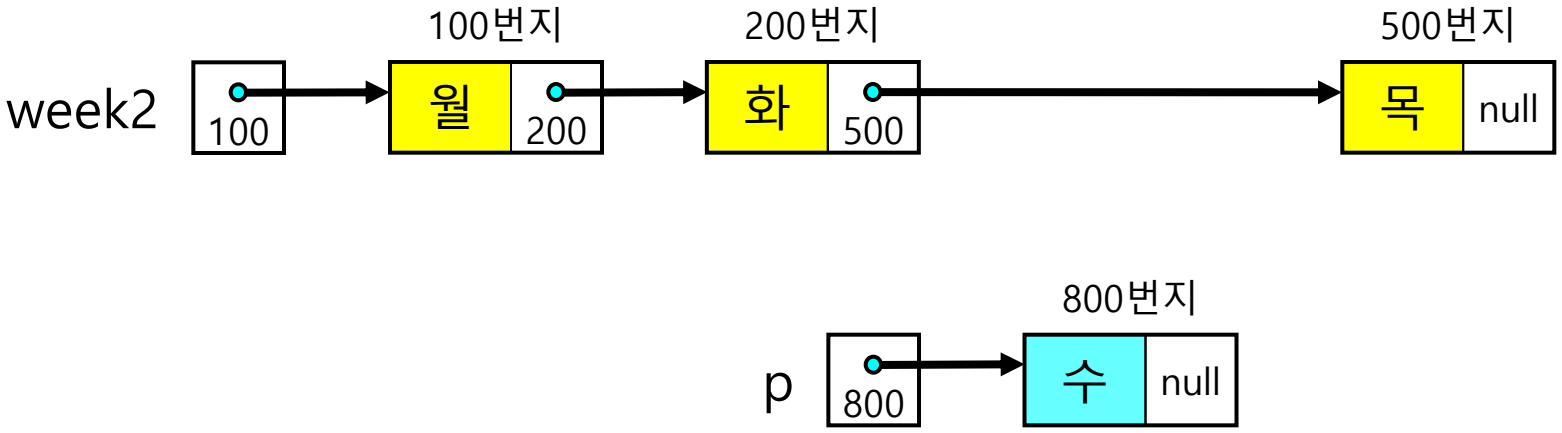


  - 목표 상태



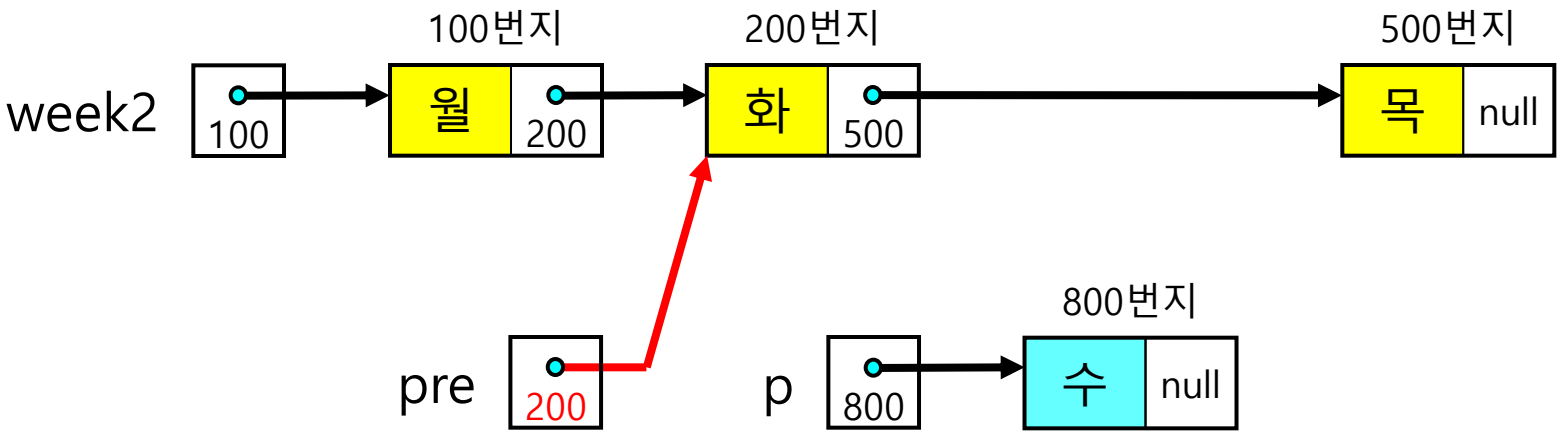
단순 연결 리스트에 노드 삽입 [2/5]

① make new node: 메모리에서 삽입할 새 노드를 생성해서, 포인터 변수 p가 가리키게 하고 데이터 필드에 "수"를 저장한다.



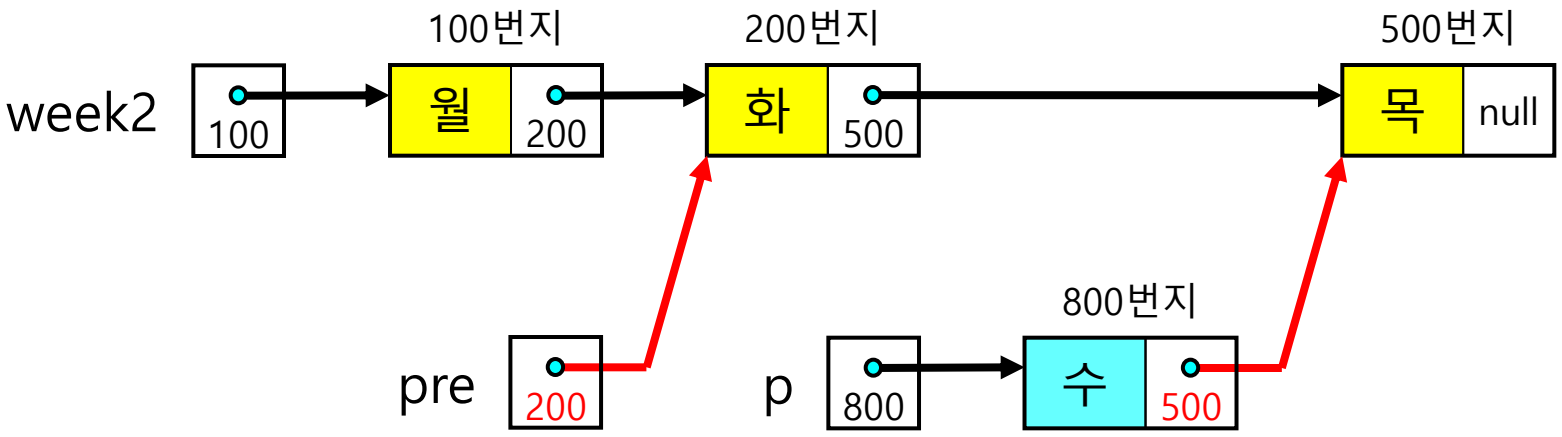
단순 연결 리스트에 노드 삽입 [3/5]

② find insert position: 새 노드가 삽입될 위치의 앞 노드를 찾아 포인터 변수 pre가 가리키도록 한다.



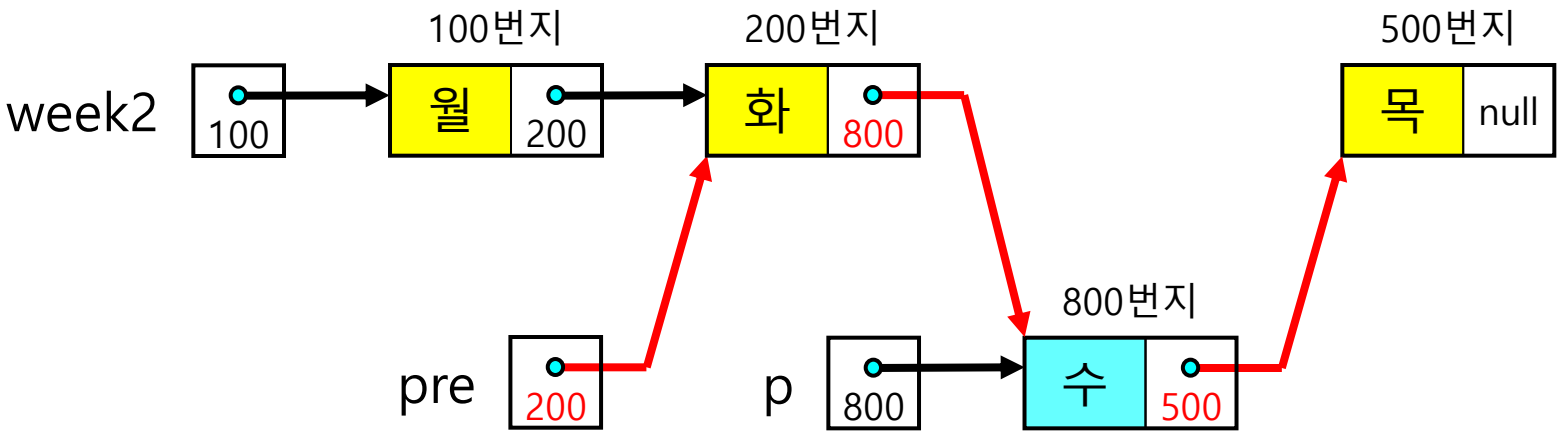
단순 연결 리스트에 노드 삽입 [4/5]

③ 새 노드가 삽입될 위치의 다음 노드를 가리키도록 한다.
즉, p→link가 pre→link 값을 갖도록 한다.



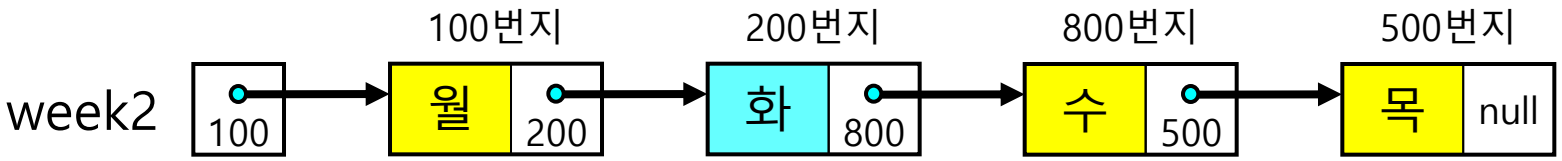
단순 연결 리스트에 노드 삽입 [5/5]

④ 새 노드가 삽입될 위치의 앞 노드가 새 노드를 가리키도록 한다. 즉 pre→link가 p값을 갖도록 한다.

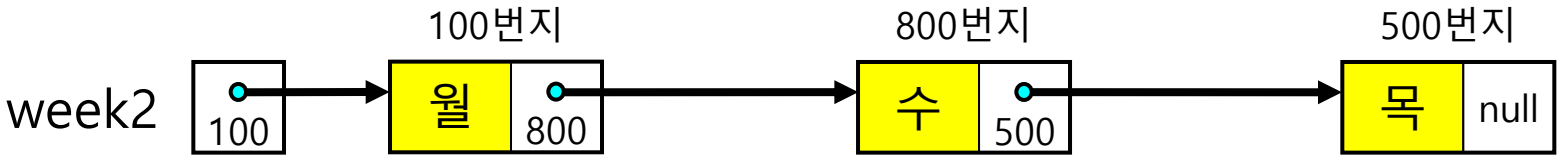


단순 연결 리스트에 노드 삭제 [1/4]

- week2 = (월, 화, 수, 목)에서 원소 “화” 삭제
  - 초기상태

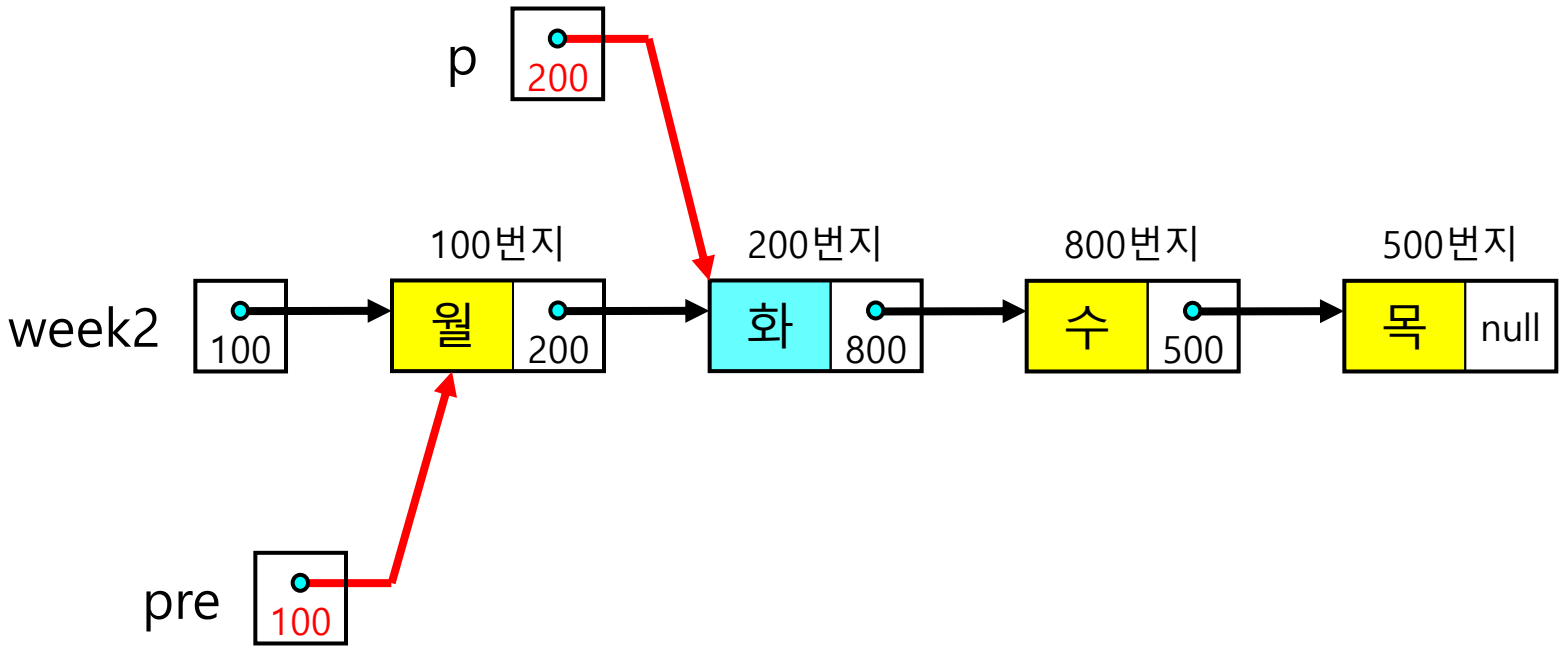


  - 목표상태



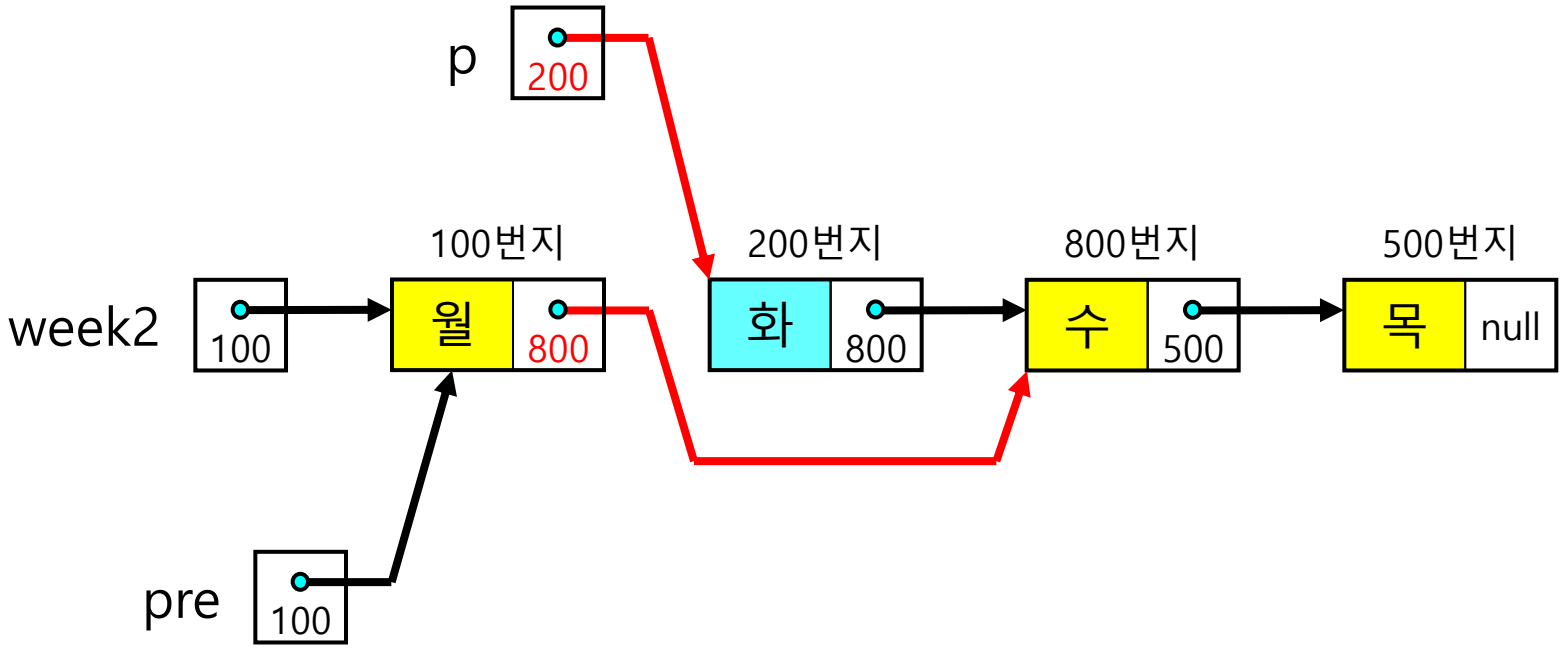
단순 연결 리스트에 노드 삭제 [2/4]

① 삭제할 노드p의 앞 노드(선행자)를 찾아 pre로 지정한다.



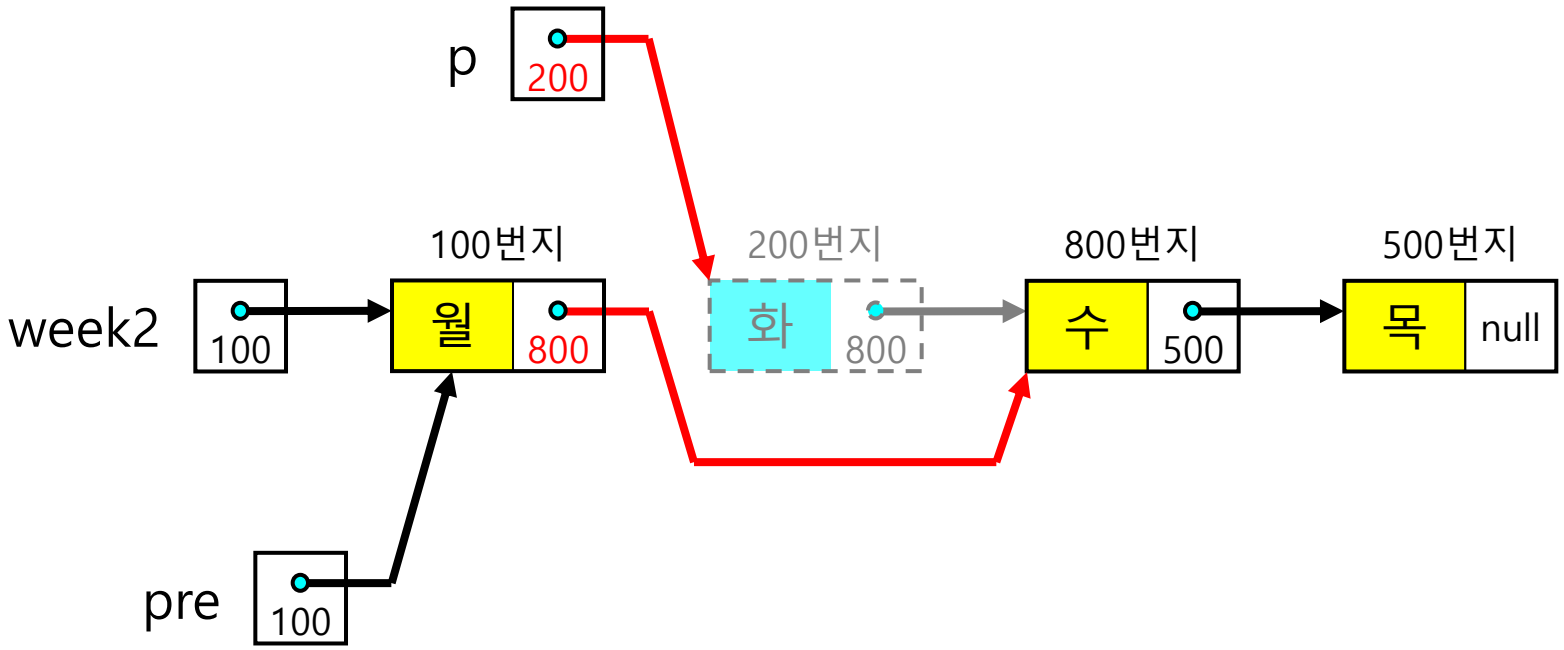
단순 연결 리스트에 노드 삭제 [3/4]

② 선행자의 링크 필드가 삭제할 노드의 다음 노드를 가리키도록 한다. 즉, pre→link에 p→link를 지정한다.



단순 연결 리스트에 노드 삭제 [4/4]

② p가 가리키는 노드를 메모리에서 제거한다.



동적 메모리 할당과 반환

- 노드 생성과 제거: malloc(), free() 함수 사용

```
node *getNode( )
{
    node *p;

    p = (node*)malloc(sizeof(node));
    // if (p == NULL) ERROR();

    return p;
}

void returnNode(node *old)
{
    free(old);
}
```

단순 연결 리스트 ADT

이 름 : SinglyLinkedList

데이터 : integer (데이터 필드)

연 산 : L ∈ SinglyLinkedList; x ∈ value; pre ∈ pointer of node;

// L은 단순 연결 리스트, x는 데이터 필드 값, pre는 노드 포인터를 의미

getNode( );	**// 새 노드 할당**
returnNode(pre);	**// pre 노드 반환**
insertFirstNode(L, x);	**// 리스트 L의 맨 앞에 x값을 갖는 노드 추가**
insertMiddleNode(L, pre, x);	**// pre가 가리키는 노드 뒤에 x값을 갖는 노드 추가**
insertLastNode(L, x);	**// 리스트 L의 맨 뒤에 x값을 갖는 노드 추가**
deleteNode(L, pre);	**// pre가 가리키는 노드의 다음 노드를 삭제**
searchNode(L, x);	**// 리스트 L에서 x값을 갖는 노드의 주소를 반환**
reverseList(L);	**// 리스트 L의 순서를 역순으로 바꾼다.**

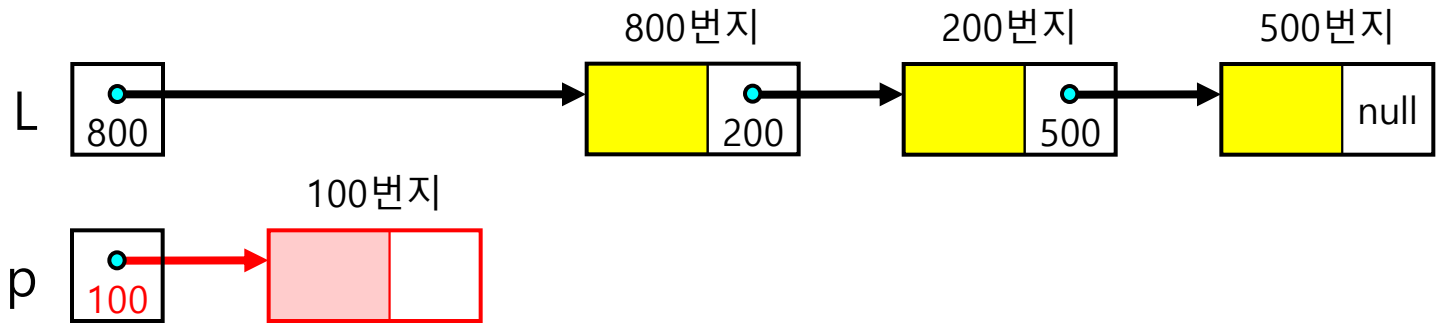
단순 연결 삽입: 첫번째 노드 삽입 [1/6]

- 리스트 L의 맨 앞에 데이터 필드 값이 x인 새 노드를 삽입하는 알고리즘

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

단순 연결 삽입: 첫번째 노드 삽입 [2/6]

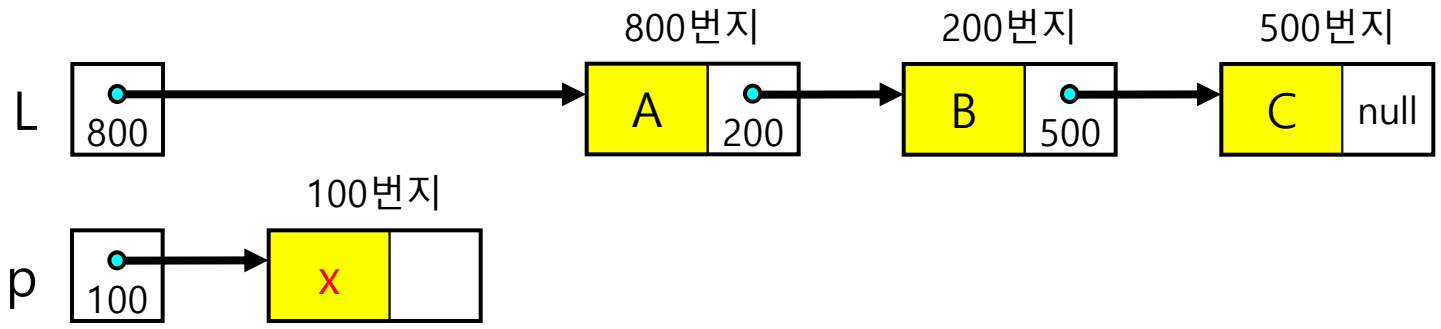
① 삽입할 노드를 자유 공간리스트에서 할당 받는다.




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

단순 연결 삽입: 첫번째 노드 삽입 [3/6]

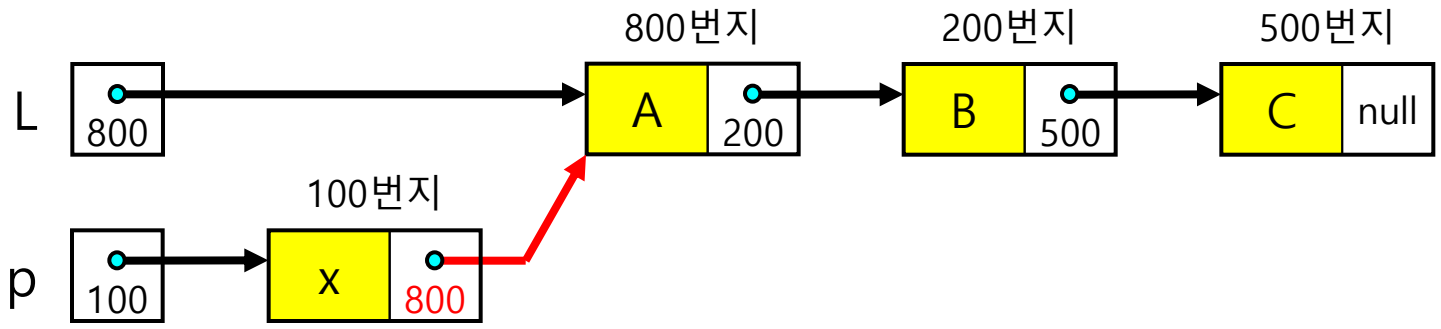
② 새 노드의 데이터 필드에 x를 저장




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

단순 연결 삽입: 첫번째 노드 삽입 [4/6]

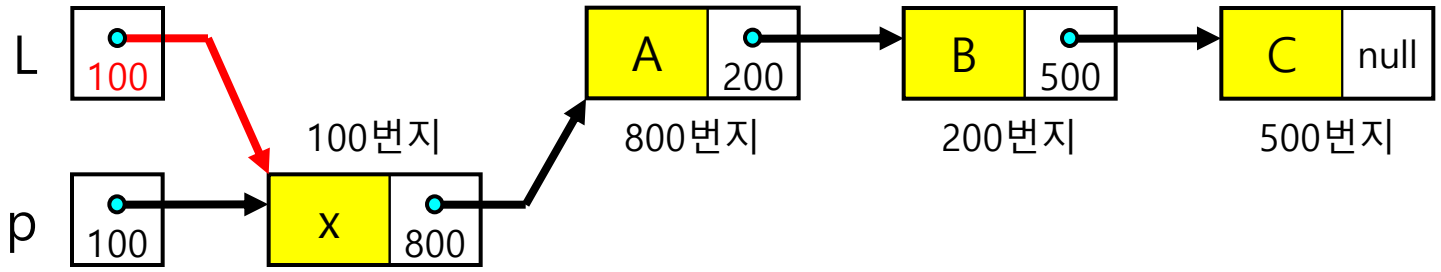
③ 새 노드의 링크 필드가 리스트 L을 가리키도록 하여, 리스트에 삽입한다.




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

단순 연결 삽입: 첫번째 노드 삽입 [5/6]

④ 리스트 포인터 L이 p를 가리키도록 하여, 새로운 노드 p가 리스트 L의 첫 번째 노드가 되도록 한다.




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

단순 연결 삽입: 첫번째 노드 삽입 [6/6]

- C코드

```
node *insertFirstNode(node *L, data x)
{
    node *p;

    p = getNode();      // ①
    p->data = x;        // ②
    p->link = L;        // ③
    L = p;              // ④

    return L;
}
```

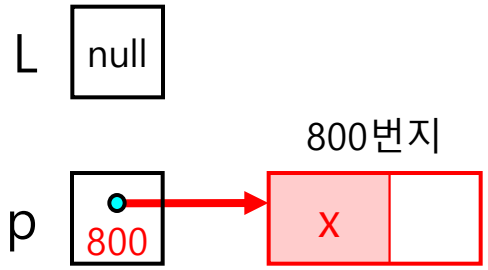
단순 연결 삽입: 중간 노드 삽입 [1/8]

- 리스트 L에서 포인터 변수 pre가 가리키는 노드의 다음에 데이터 필드 값이 x인 새 노드를 삽입하는 알고리즘

```
insertMiddleNode(L, pre, x)
        p ← getNode();
        p.data ← x;
        if (L = null) then                  // ①
                L ← p;                      // ②
                p.link ← null;              // ③
        else                                // ④
                p.link ← pre->link;         // ⑤
                pre.link ← p;               // ⑥
        end if
end insertMiddleNode()
```

단순 연결 삽입: 중간 노드 삽입 [2/8]

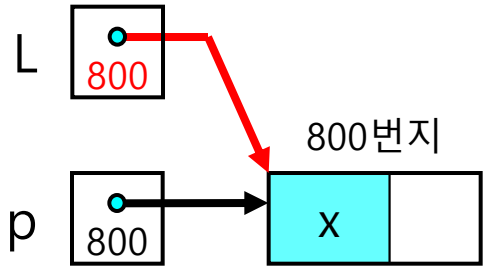
① 리스트 L이 공백 리스트인 경우




```
insertMiddleNode(L, pre, x)
        p ← getNode();
        p.data ← x;
        if (L = null) then                  // ①
                L ← p;                      // ②
                p.link ← null;              // ③
        else                                // ④
                p.link ← pre.link;          // ⑤
                pre.link ← p;               // ⑥
        end if
end insertMiddleNode()
```

단순 연결 삽입: 중간 노드 삽입 [3/8]

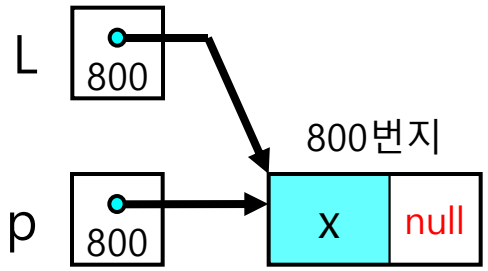
② 리스트 포인터 L에 새 노드 p의 주소를 저장한다.




```
insertMiddleNode(L, pre, x)
        p ← getNode();
        p.data ← x;
        if (L = null) then                  // ①
                L ← p;                      // ②
                p.link ← null;              // ③
        else                                // ④
                p.link ← pre.link;          // ⑤
                pre.link ← p;               // ⑥
        end if
end insertMiddleNode()
```

단순 연결 삽입: 중간 노드 삽입 [4/8]

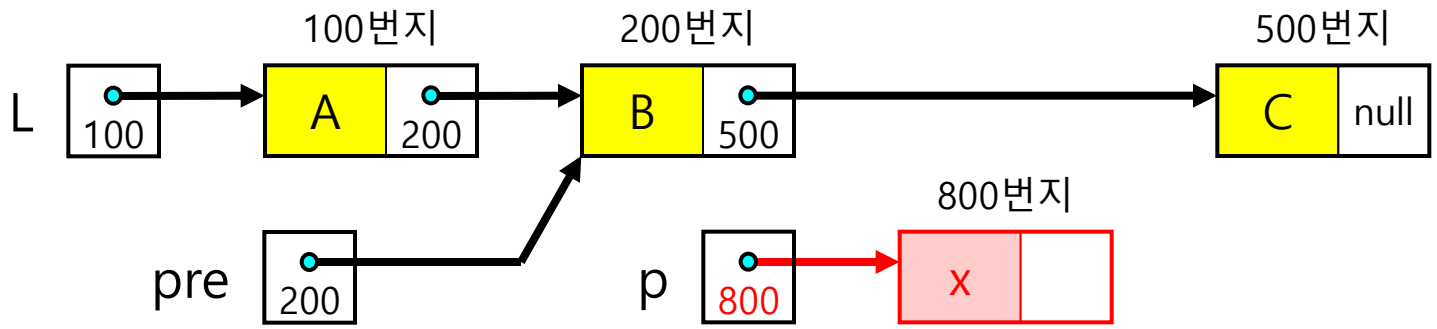
③ p의 링크 필드에 null을 저장하여 마지막 노드를 표시




```
insertMiddleNode(L, pre, x)
        p ← getNode();
        p.data ← x;
        if (L = null) then                  // ①
                L ← p;                      // ②
                p.link ← null;              // ③
        else                                // ④
                p.link ← pre.link;          // ⑤
                pre.link ← p;               // ⑥
        end if
end insertMiddleNode()
```

단순 연결 삽입: 중간 노드 삽입 [5/8]

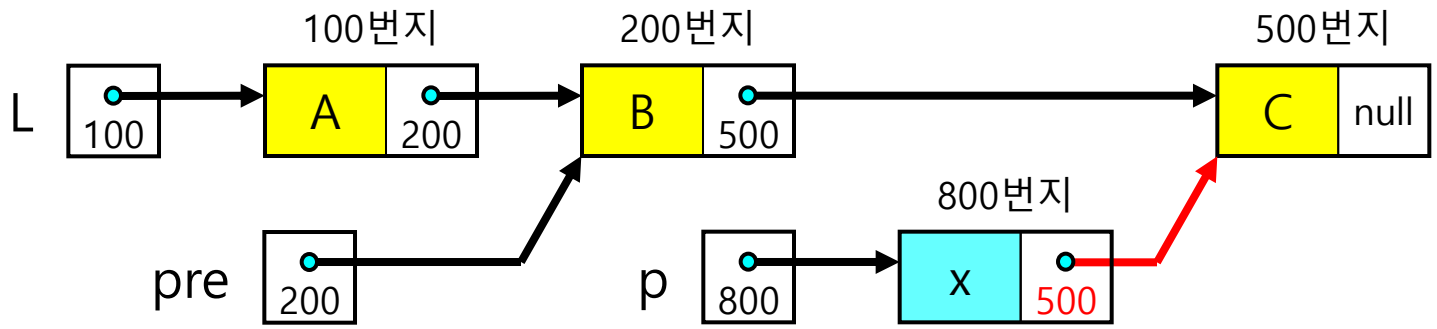
④ 리스트 L이 공백 리스트가 아닌 경우




```
insertMiddleNode(L, pre, x)
        p ← getNode();
        p.data ← x;
        if (L = null) then                  // ①
                L ← p;                      // ②
                p.link ← null;              // ③
        else                                // ④
                p.link ← pre.link;          // ⑤
                pre.link ← p;               // ⑥
        end if
end insertMiddleNode()
```

단순 연결 삽입: 중간 노드 삽입 [6/8]

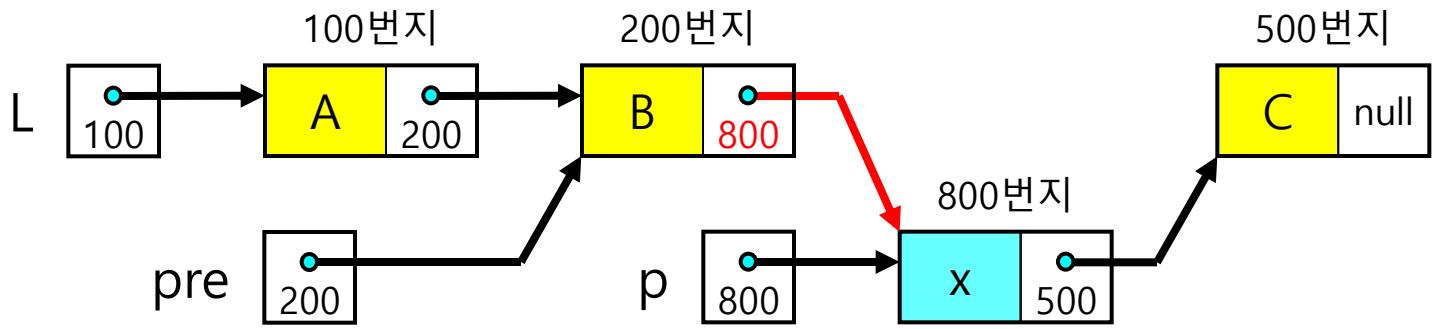
⑤ 노드 pre의 링크 필드 값을 노드 p의 링크 필드에 저장




```
insertMiddleNode(L, pre, x)
        p ← getNode();
        p.data ← x;
        if (L = null) then                  // ①
                L ← p;                      // ②
                p.link ← null;              // ③
        else                                // ④
                p.link ← pre.link;          // ⑤
                pre.link ← p;               // ⑥
        end if
end insertMiddleNode()
```

단순 연결 삽입: 중간 노드 삽입 [7/8]

⑥ 포인터 p의 값을 노드 pre의 링크 필드에 저장한다.




```
insertMiddleNode(L, pre, x)
        p ← getNode();
        p.data ← x;
        if (L = null) then                  // ①
                L ← p;                      // ②
                p.link ← null;              // ③
        else                                // ④
                p.link ← pre.link;          // ⑤
                pre.link ← p;               // ⑥
        end if
end insertMiddleNode()
```

단순 연결 삽입: 중간 노드 삽입 [8/8]

```
node *insertMiddleNode(node *L, node *pre, data x)
{
    node *p;

    p = getNode();
    p->data = x;
    if (L == NULL) {                    // ①
        L = p;                          // ②
        p->link = NULL;                 // ③
    } else {                            // ④
        p->link = pre->link;            // ⑤
        pre->link = p;                  // ⑥
    }
    return L;
}
```

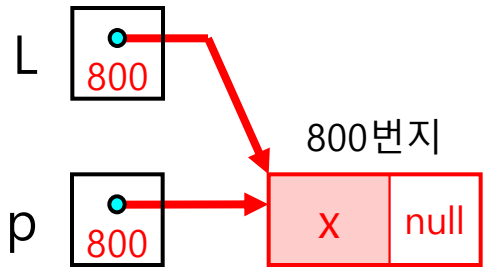
단순 연결 삽입: 마지막 노드 삽입 [1/6]

- 리스트 L의 마지막에 새로운 노드를 삽입하는 알고리즘

```
insertLastNode(L, x)
        p ← getNode();
        p.data ← x;
        p.link ← null;
        if (L = null) then                          // ①
                L ← p;
        else
                q ← L;                              // ②
                while (q.link ≠ null) do
                        q ← q.link;                 // ③
                q.link ← p;                         // ④
        end if
end insertLastNode()
```

단순 연결 삽입: 마지막 노드 삽입 [2/6]

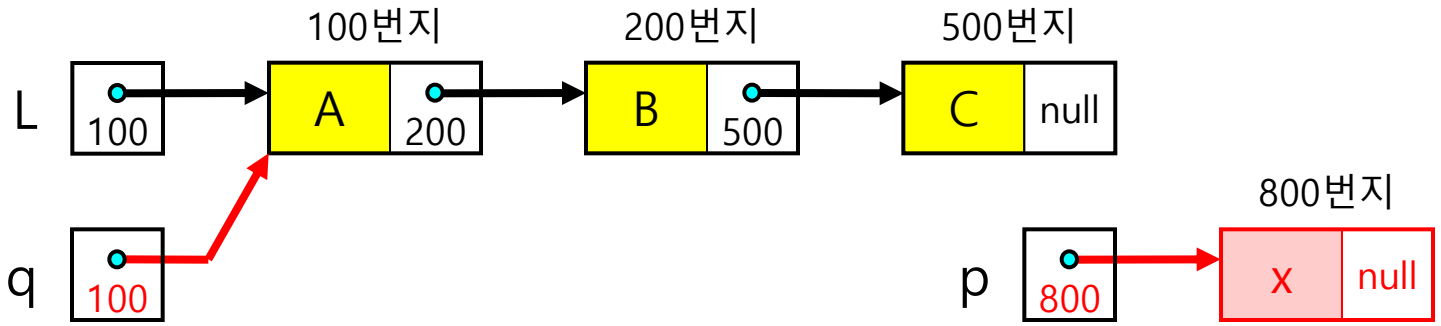
① 리스트 L이 공백 리스트인 경우, 리스트 포인터 L에 새 노드 p의 주소를 저장한다.




```
insertLastNode(L, x)
        p ← getNode();
        p.data ← x;
        p.link ← null;
        if (L = null) then                    // ①
                L ← p;
        else
                q ← L;                        // ②
                while (q.link ≠ null) do
                        q ← q.link;           // ③
                q.link ← p;                   // ④
        end if
end insertLastNode()
```

단순 연결 삽입: 마지막 노드 삽입 [3/6]

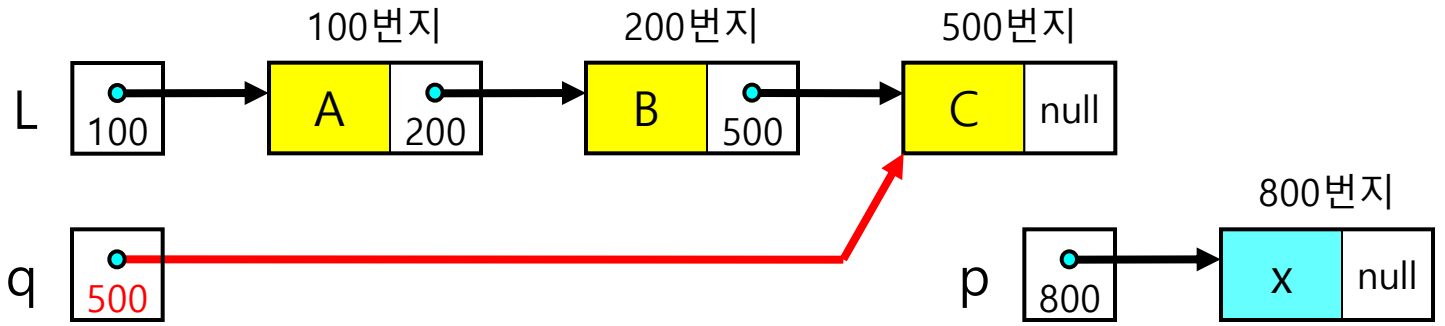
② 리스트 L의 마지막 노드를 찾기 위해 노드를 순회할
임시 포인터 q에 리스트의 첫 번째 노드의 주소를 지정




```
insertLastNode(L, x)
        p ← getNode();
        p.data ← x;
        p.link ← null;
        if (L = null) then                    // ①
                L ← p;
        else
                q ← L;                        // ②
                while (q.link ≠ null) do
                        q ← q.link;           // ③
                q.link ← p;                   // ④
        end if
end insertLastNode()
```

단순 연결 삽입: 마지막 노드 삽입 [4/6]

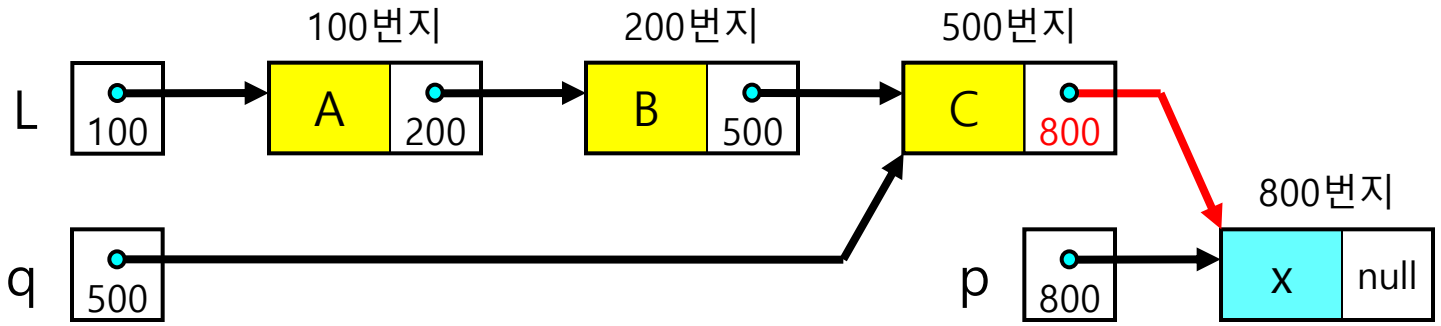
③ while문에 의해 순회 포인터 q가 노드의 링크 필드를 따라 이동하면서 링크 필드가 null인 마지막 노드를 찾는다.




```
insertLastNode(L, x)
        p ← getNode();
        p.data ← x;
        p.link ← null;
        if (L = null) then                    // ①
                L ← p;
        else
                q ← L;                        // ②
                while (q.link ≠ null) do
                        q ← q.link;           // ③
                q.link ← p;                   // ④
        end if
end insertLastNode()
```

단순 연결 삽입: 마지막 노드 삽입 [5/6]

④ 순회 포인터 q가 가리키는 리스트의 마지막 노드의 링크 필드에 새 노드 p의 주소를 저장하여, 리스트의 마지막에 새 노드를 연결




```
insertLastNode(L, x)
        p ← getNode();
        p.data ← x;
        p.link ← null;
        if (L = null) then                  // ①
                L ← p;
        else
                q ← L;                      // ②
                while (q.link ≠ null) do
                        q ← q.link;         // ③
                q.link ← p;                 // ④
        end if
end insertLastNode()
```

단순 연결 삽입: 마지막 노드 삽입 [6/6]

```
node *insertLastNode(node *L, data x)
{
    node *p, *q;

    p = getNode();
    p->data = x;
    P->link = NULL;

    if (L == NULL)                  // ①
        L = p;
    else {
        q = L;                      // ②
        while (q->link != NULL)
            q = q->link;            // ③
        q->link = p;                // ④
    }
    return L;
}
```

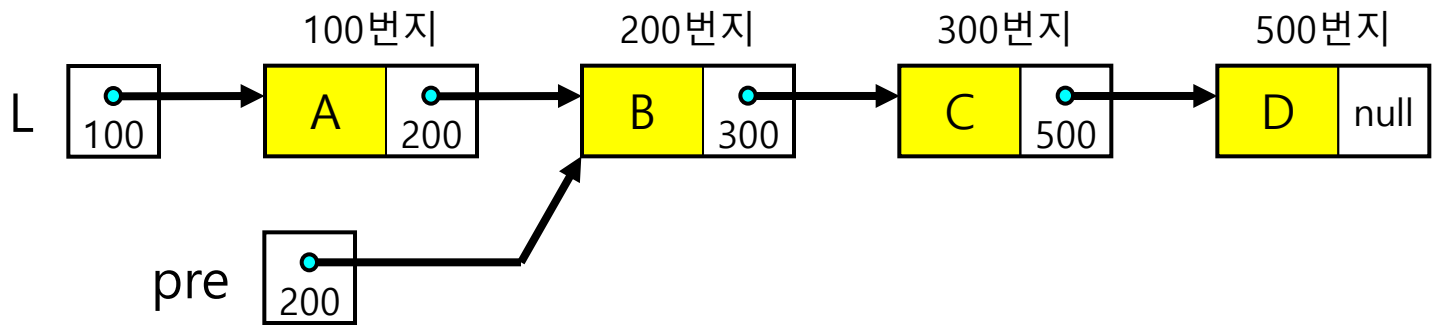
단순 연결 리스트의 노드 삭제 [1/6]

- 리스트 L에서 포인터 pre가 가리키는 노드의 다음 노드를 삭제하는 알고리즘

```
deleteNode(L, pre)
        if (L = null) then error;
                                                    // ①
        old ← pre.link;                             // ②
        if (old ≠ null) then
                pre.link ← old.link;                // ③
                returnNode(old);                    // ④
        end if
end deleteNode()
```

단순 연결 리스트의 노드 삭제 [2/6]

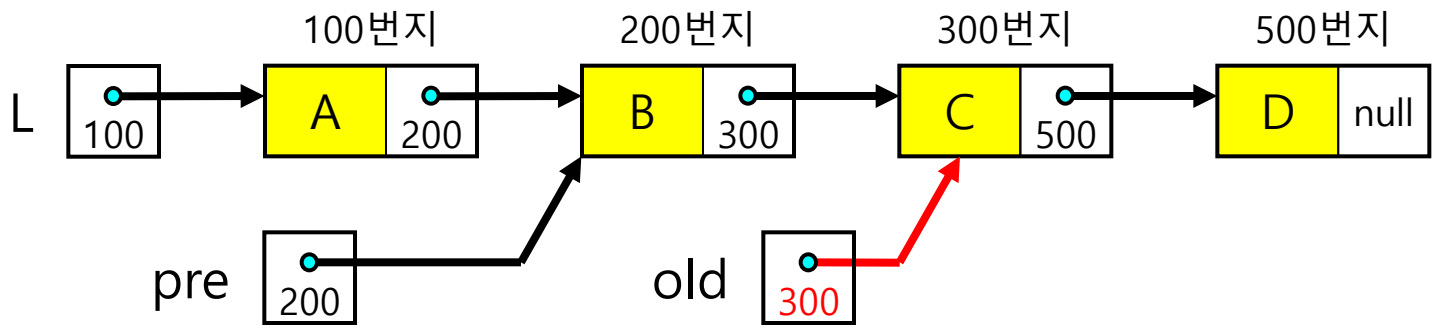
① 리스트 L이 null이 아닌 경우




```
deleteNode(L, pre)
    if (L = null) then error;
                                        // ①
    old ← pre.link;                     // ②
    if (old ≠ null) then
        pre.link ← old.link;            // ③
        returnNode(old);                // ④
    end if
end deleteNode()
```

단순 연결 리스트의 노드 삭제 [3/6]

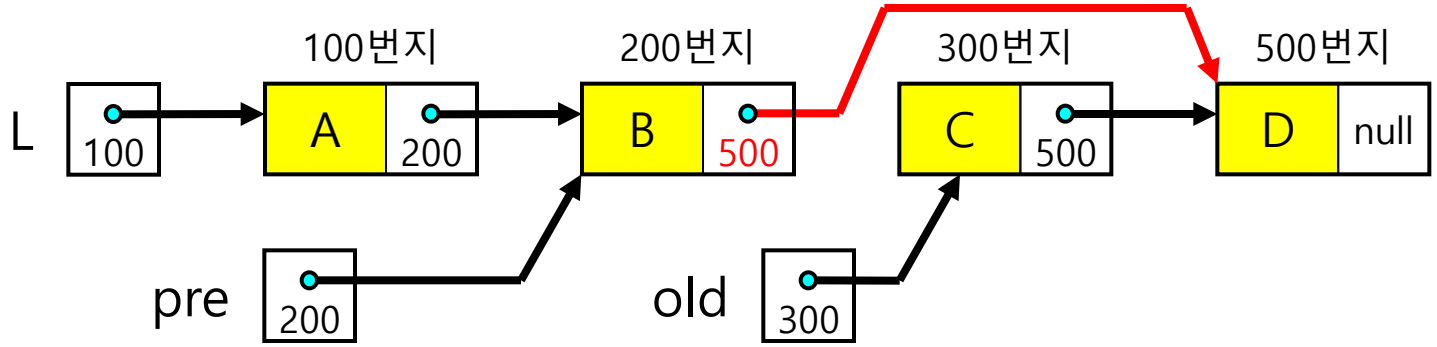
② 노드 pre의 다음노드의 주소를 포인터 old에 저장 한다.




```
deleteNode(L, pre)
    if (L = null) then error;
                                    // ①
    old ← pre.link;                 // ②
    if (old ≠ null) then
        pre.link ← old.link;        // ③
        returnNode(old);            // ④
    end if
end deleteNode()
```

단순 연결 리스트의 노드 삭제 [4/6]

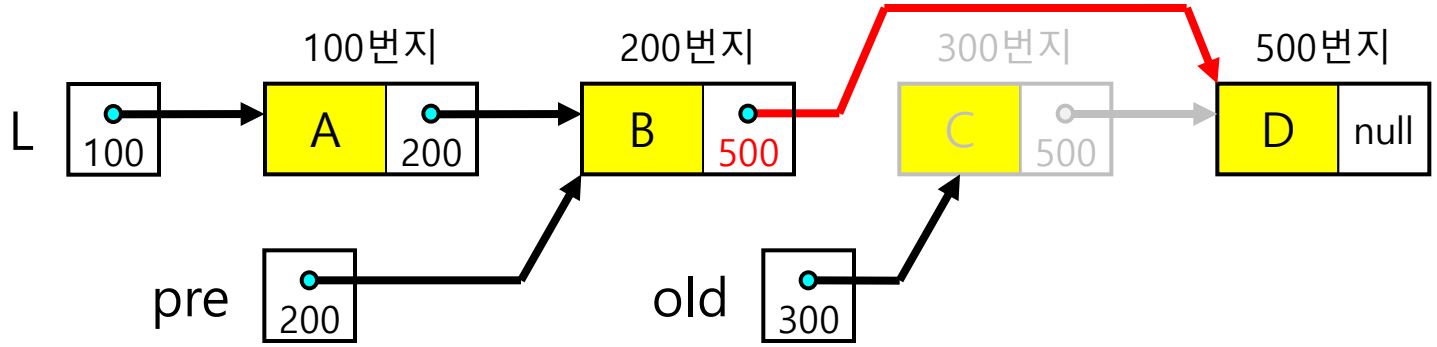
③ 삭제할 노드 old의 링크 필드를 pre의 링크 필드에 지정




```
deleteNode(L, pre)
    if (L = null) then error;
                                        // ①
    old ← pre.link;                     // ②
    if (old ≠ null) then
        pre.link ← old.link;            // ③
        returnNode(old);                // ④
    end if
end deleteNode()
```

단순 연결 리스트의 노드 삭제 [5/6]

④ 삭제한 노드 old를 자유 공간리스트에 반환




```
deleteNode(L, pre)
    if (L = null) then error;
                                        // ①
    old ← pre.link;                     // ②
    if (old ≠ null) then
        pre.link ← old.link;            // ③
        returnNode(old);                // ④
    end if
end deleteNode()
```

단순 연결 리스트의 노드 삭제 [6/6]

```
node *deleteNode(node *L, node *pre)
{
    node *old;

    if (L == NULL) return L;
                                        // ①
    old = pre->link;                    // ②
    if (old != NULL) {
        pre->link = old->link;          // ③
        returnNode(old);                // ④
    }
    return L;
}
```

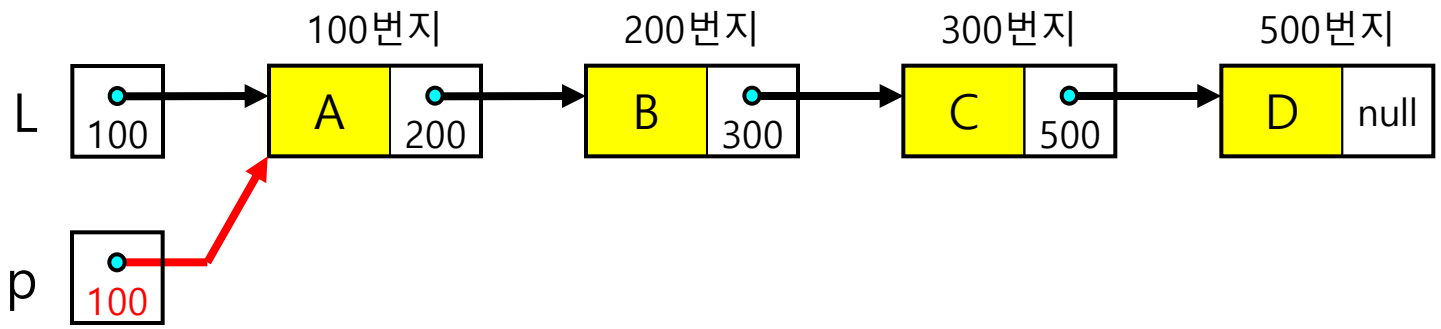
단순 연결 리스트의 노드 탐색 [1/6]

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

```
searchNode(L, x)
    p ← L;                          // ①
    while (p ≠ null) do
        if (p.data = x ) then
            return p;               // ③
        p ← p.link;                 // ②
    end while
    return p;                       // ④
end searchNode()
```

단순 연결 리스트의 노드 탐색 [2/6]

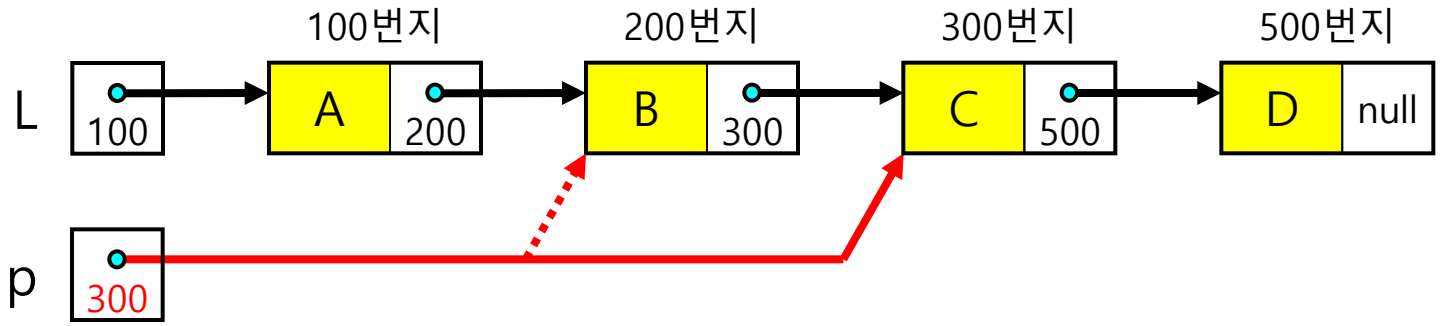
① 순회 포인터 p에 리스트 L의 첫번째 노드를 지정한다.




```
searchNode(L, x)
    p ← L;                          // ①
    while (p ≠ null) do
        if (p.data = x ) then
            return p;               // ③
        p ← p.link;                 // ②
    end while
    return p;                       // ④
end searchNode()
```

단순 연결 리스트의 노드 탐색 [3/6]

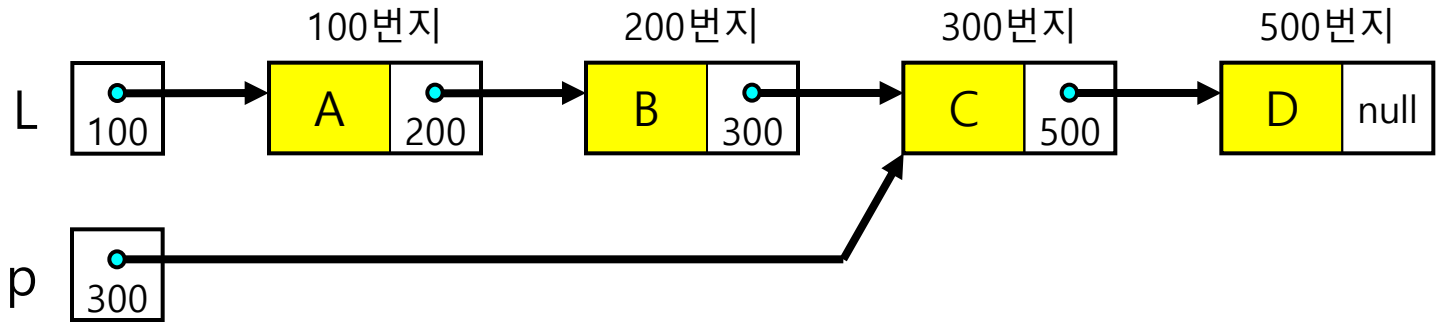
② 순회 포인터 p로 리스트 L의 노드를 순회한다.




```
searchNode(L, x)
    p ← L;                          // ①
    while (p ≠ null) do
        if (p.data = x ) then
            return p;               // ③
        p ← p.link;                 // ②
    end while
    return p;                       // ④
end searchNode()
```

단순 연결 리스트의 노드 탐색 [4/6]

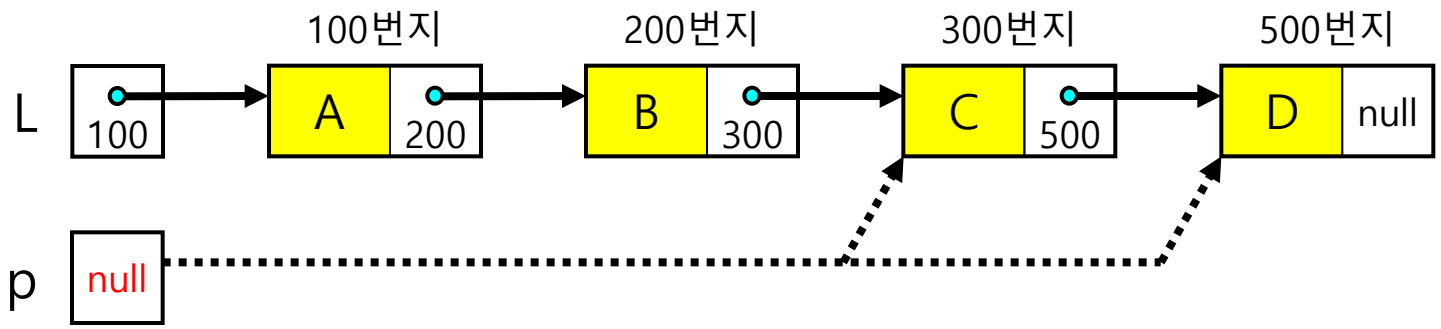
③ 원하는 데이터 x를 찾은 경우 p를 돌려준다.




```
searchNode(L, x)
    p ← L;                          // ①
    while (p ≠ null) do
        if (p.data = x ) then
            return p;               // ③
        p ← p.link;                 // ②
    end while
    return p;                       // ④
end searchNode()
```

단순 연결 리스트의 노드 탐색 [5/6]

④ 리스트 끝까지 데이터가 없으면 null을 돌려준다.




```
searchNode(L, x)
    p ← L;                          // ①
    while (p ≠ null) do
        if (p.data = x ) then
            return p;               // ③
        p ← p.link;                 // ②
    end while
    return p;                       // ④
end searchNode()
```

단순 연결 리스트의 노드 탐색 [6/6]

```
node *searchNode(node *L, data x)
{
    node *p;

    p = L;                          // ①
    while (p != NULL) {
        if (p->data == x)
            return p;               // ③
        p = p->link;                // ②
    }
    return p;                       // ④
}
```

역순 리스트 만들기 [1/7]

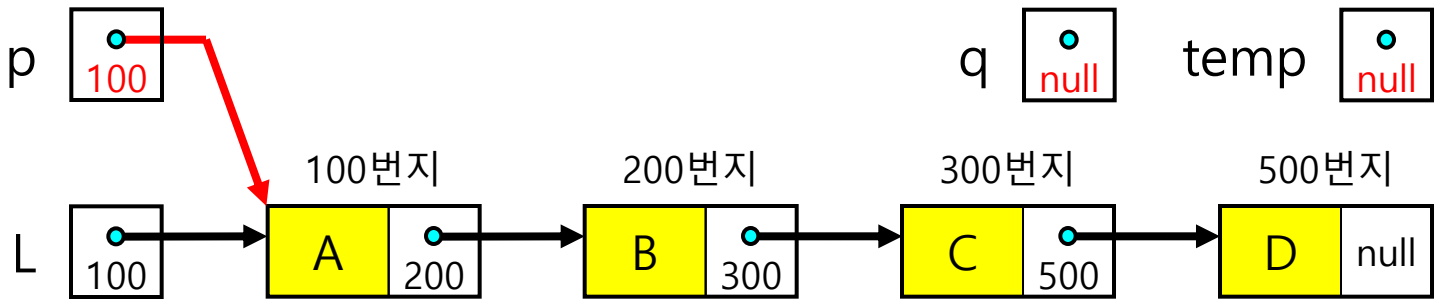
- 리스트 L의 노드를 역순으로 바꾸는 알고리즘

```
reverseList(L)
        p ← L;
        q ← null;                          // ①
        while (p ≠ null) do
                temp ← q;
                q ← p;
                p ← p.link;
                q.link ← temp;             // ②
        end while
        L ← q;                             // ③
end reverseList()
```

역순 리스트 만들기 [2/7]

① p: 원래 리스트,
q: 역순 리스트,
temp: 역순 리스트의 헤더를 임시로 지정하는 포인터

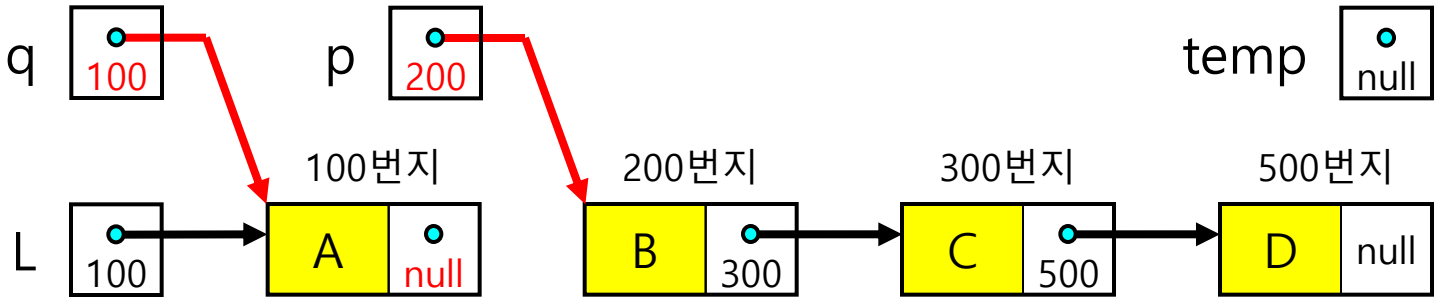
```
reverseList(L)
    p ← L, q ← null;                    // ①
    while (p ≠ null) do
        temp ← q;
        q ← p, p ← p.link;
        q.link ← temp;                  // ②
    end while
    L ← q;                              // ③
end reverseList()
```



역순 리스트 만들기 [3/7]

② while 루프를 반복하며 리스트 헤더 p가 가리키는 노드를 역순 리스트 q의 첫번째 노드로 이동한다.

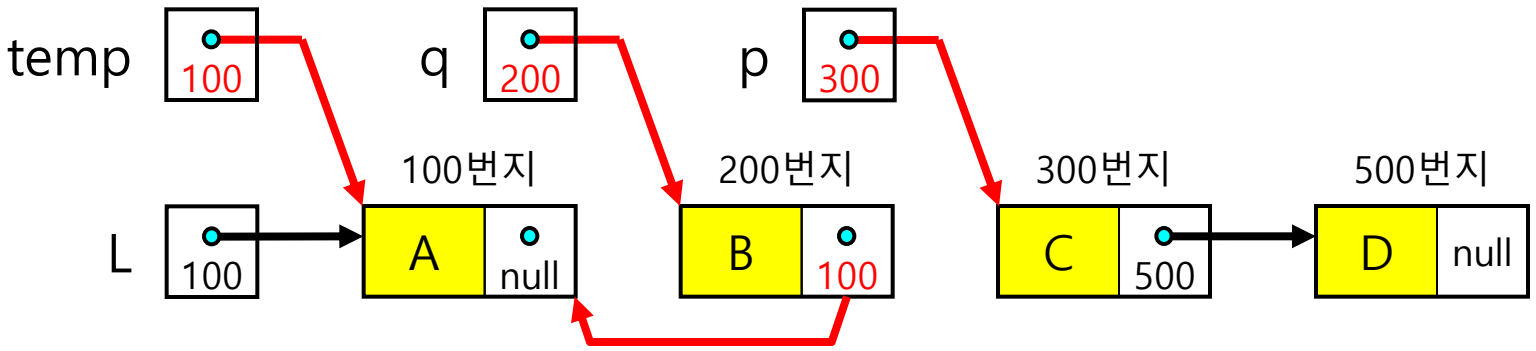
```
reverseList(L)
    p ← L, q ← null;                    // ①
    while (p ≠ null) do
        temp ← q;
        q ← p, p ← p.link;
        q.link ← temp;                  // ②
    end while
    L ← q;                              // ③
end reverseList()
```



역순 리스트 만들기 [4/7]

② while 루프를 반복하며 리스트 헤더 p가 가리키는 노드를 역순 리스트 q의 첫번째 노드로 이동한다.

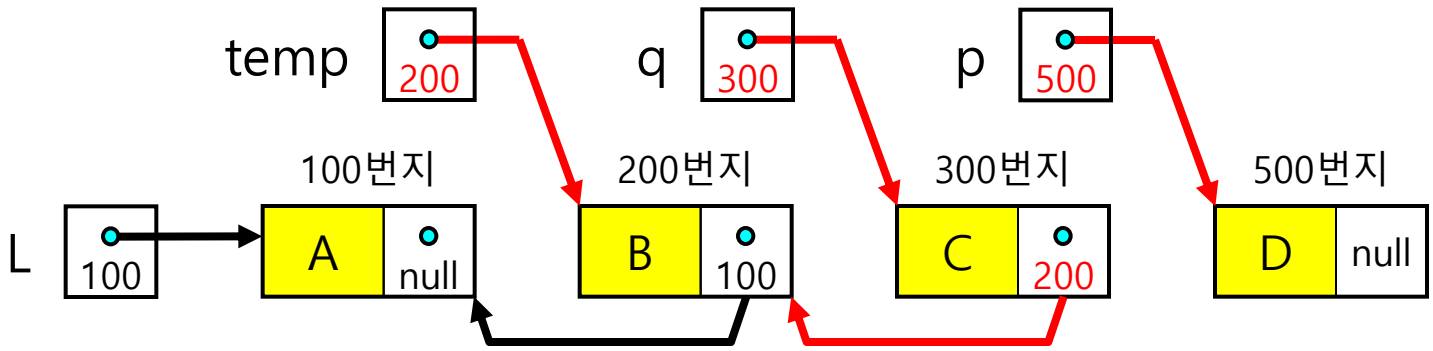
```
reverseList(L)
    p ← L, q ← null;                    // ①
    while (p ≠ null) do
        temp ← q;
        q ← p, p ← p.link;
        q.link ← temp;                  // ②
    end while
    L ← q;                              // ③
end reverseList()
```



역순 리스트 만들기 [5/7]

② while 루프를 반복하며 리스트 헤더 p가 가리키는 노드를 역순 리스트 q의 첫번째 노드로 이동한다.

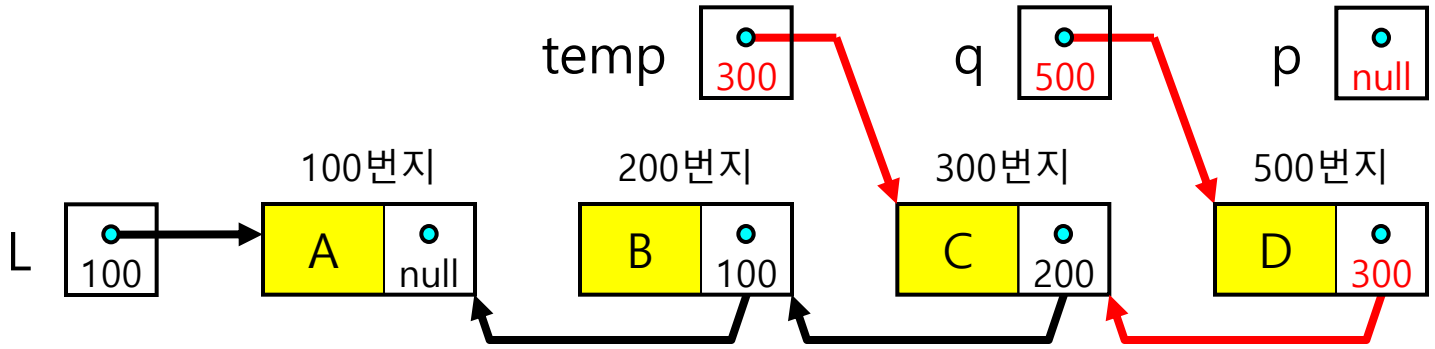
```
reverseList(L)
    p ← L, q ← null;                    // ①
    while (p ≠ null) do
        temp ← q;
        q ← p, p ← p.link;
        q.link ← temp;                  // ②
    end while
    L ← q;                              // ③
end reverseList()
```



역순 리스트 만들기 [6/7]

② while 루프를 반복하며 리스트 헤더 p가 가리키는 노드를 역순 리스트 q의 첫번째 노드로 이동한다.

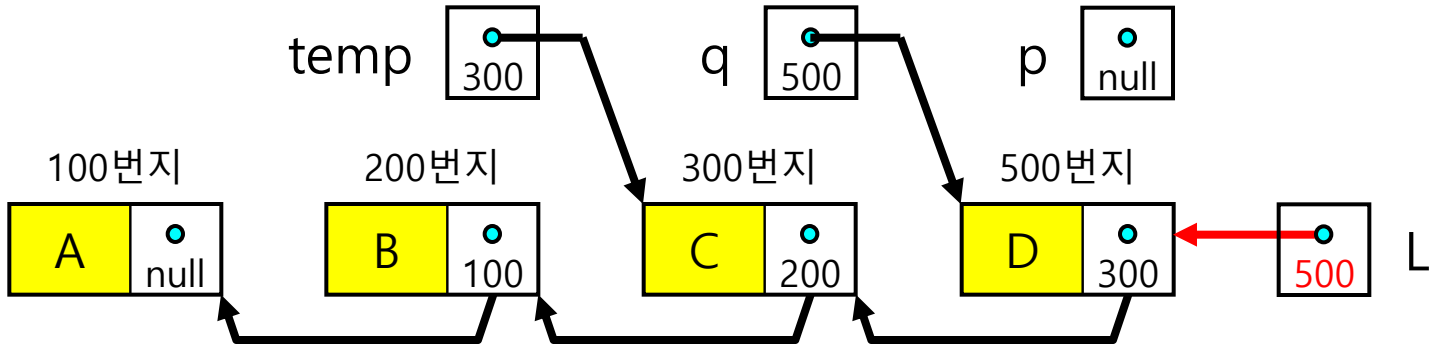
```
reverseList(L)
    p ← L, q ← null;                    // ①
    while (p ≠ null) do
        temp ← q;
        q ← p, p ← p.link;
        q.link ← temp;                  // ②
    end while
    L ← q;                              // ③
end reverseList()
```



역순 리스트 만들기 [7/7]

③ 리스트 L의 시작 주소를 역순 리스트 q의 첫번째 노드로 지정한다.

```
reverseList(L)
    p ← L, q ← null;                    // ①
    while (p ≠ null) do
        temp ← q;
        q ← p, p ← p.link;
        q.link ← temp;                  // ②
    end while
    L ← q;                              // ③
end reverseList()
```



요약

- 연결 자료구조(linked data structure)
  - 자료의 논리적 순서와 물리적 순서가 일치하지 않는 자료구조
  - 각 원소에 다음 원소의 주소를 저장하여 연결되는 방식
  - 여러 개의 작은 공간을 연결하여 전체 자료구조를 표현
- 노드(node)
  - 연결 자료구조에서 하나의 원소를 표현하기 위한 단위 구조
  - 구성:
- 단순 연결 리스트(singly linked list)
  - 노드가 하나의 링크 필드에 의해 다음 노드와 연결되는 리스트 자료구조
  - 주요 연산
    - insertFirstNode(L, x), insertMiddleNode(L, pre, x), insertLastNode(L, x)
    - searchNode(L, x), deleteNode(L, pre)
    - reverseList(L)