자료구조론

6장 연결 자료구조 표현 방식

❑ 이 장에서 다룰 내용

- 연결 자료구조 방식
- 단순 연결 리스트
- 원형 연결 리스트
- 이중 연결 리스트
- 다항식의 연결 자료구조 표현

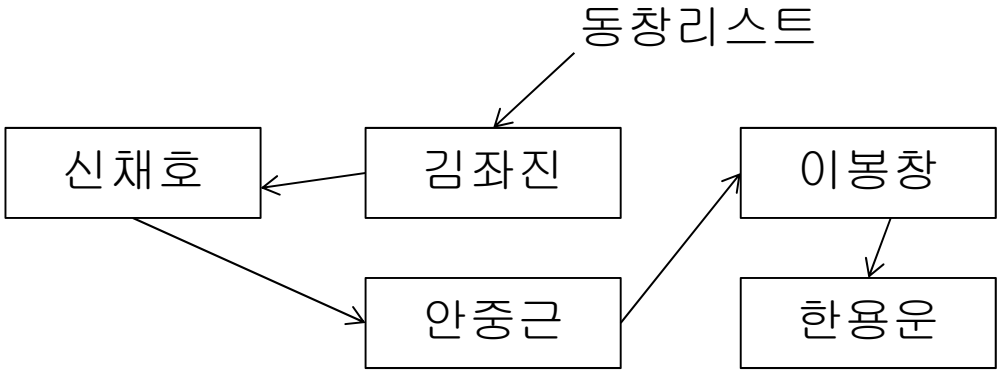
❑ 연결 자료구조 방식

- 선형 리스트를 구현하는 2가지 방법
  - 순차(sequential) 자료구조 (5장)
  - 연결(linked) 자료구조 (6장)
- 순차 자료구조 구현의 문제점
  - 연산 시간 문제: 삽입/삭제 연산 후에 연속적인 물리 주소를 유지하기 위해서 원소들을 이동시키는 작업이 필요
    - 원소의 개수가 많으면서 삽입/삭제 연산이 빈번하게 일어나는 경우, 원소 이동 오버헤드로 성능상의 문제 발생
  - 저장 공간 문제: 배열을 이용하여 구현하기 때문에 배열이 갖고 있는 메모리 사용의 비효율성 문제를 지님
    - 리스트 크기 변경에 유연하게 대처하지 못함
    - 원소수가 동적으로 변하는 경우 배열 크기를 충분히 크게 잡아야 함 ➔ 실제 원소수가 배열 크기보다 작은 경우 메모리 낭비
- 순차 자료구조의 문제점을 해결하는 자료 표현 방법 필요
  - 연결 자료구조

❑ 연결 자료구조 방식

❖ 연결 자료구조(linked data structure)

- 자료의 논리적인 순서와 물리적인 순서가 일치하지 않음
- 각 원소를 저장할 때 다음 원소의 주소도 함께 저장해 두고, 이 주소에 의해 원소들이 연결됨
  - 순차 자료구조에서 물리적인 순서를 맞추기 위해 원소를 이동하는 오버헤드가 연결 자료구조에서는 없음

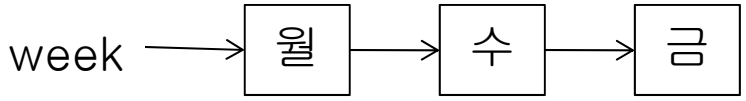


- 여러 개의 작은 공간을 연결하여 하나의 전체 자료구조를 표현
  - 크기 변경이 유연
  - 불필요하게 큰 메모리 공간을 미리 확보할 필요 없으므로 효율적
- Q: 순차 자료구조에 비해 연결 자료구조는 항상 메모리를 절약하는가?

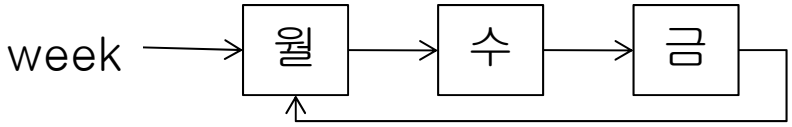
❑ 연결 자료구조 방식

❖ 연결 리스트(linked list)

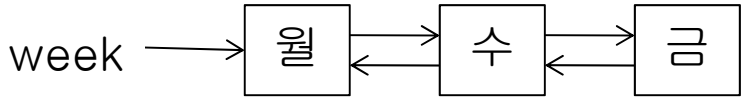
- 리스트를 연결 자료구조로 표현한 구조
- 연결 방식에 따른 구분
  - 단순 연결 리스트(singly linked list)



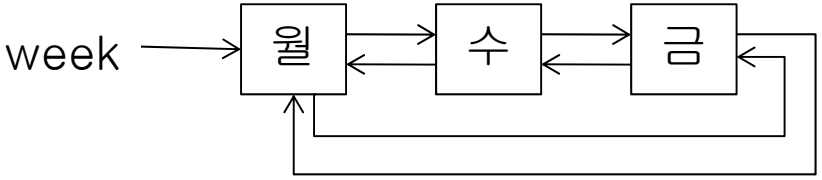
  - 원형 연결 리스트(circular linked list) : 환형 연결 리스트



  - 이중 연결 리스트(doubly linked list)



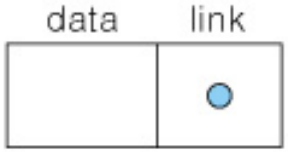
  - 원형 이중 연결 리스트(circular doubly linked list)



❑ 연결 자료구조 방식

❖ 노드

- 연결 자료구조에서 하나의 원소를 표현하기 위한 단위 구조
- <원소, 다음노드주소> 로 구성



  - 데이터 필드(data field)
    - ➢원소의 값을 저장
    - ➢저장할 원소의 형태에 따라서 하나 이상의 필드로 구성
  - 링크 필드(link field)
    - ➢다음(next) 노드의 주소를 저장
    - ➢자바에서는 노드 객체 참조값(reference)을 사용
    - ➢리스트의 마지막 노드는 링크 필드가 null
- 저장할 원소가 문자열일 때, 노드를 자바 클래스 Node로 정의하면:

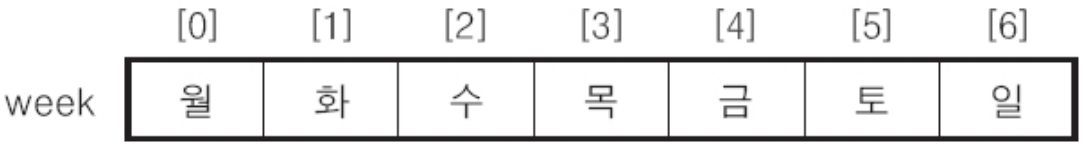
```
class Node {
    String data;
    Node link;
}
```

❑ 연결 자료구조 방식

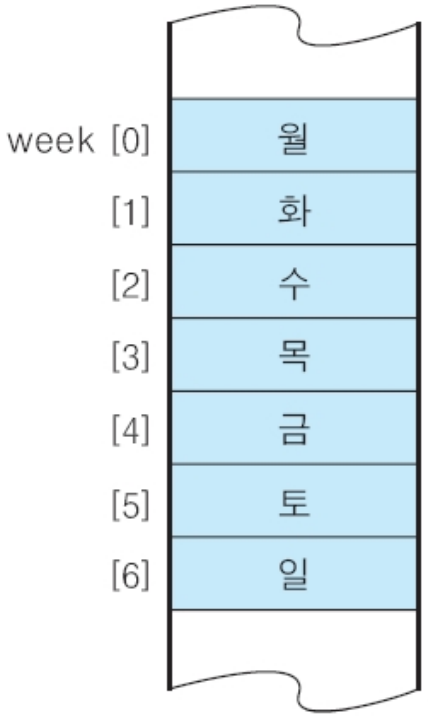
❖ 순차 리스트와 연결 리스트

- 표현하고자 하는 리스트가 다음과 같다면:
  week=(월, 화, 수, 목, 금, 토, 일)

- 순차 리스트 표현



(a) 논리구조



(b) 물리구조

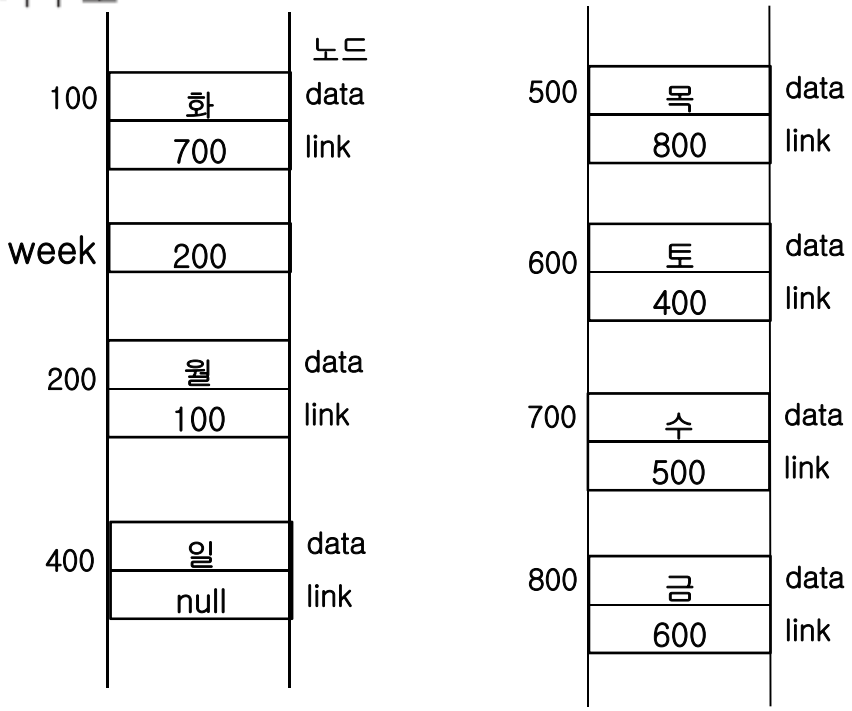
❑ 연결 자료구조 방식

- 연결 리스트 표현

(a) 논리구조

✓ 리스트를 사용하려면 첫번째 노드의 주소를 알고 있으면 된다.
(링크를 따라가며 리스트 모든 원소에 접근할 수 있다)

(b) 물리구조



❑ 연결 자료구조 방식

- 연결 리스트 표현
  - 노드 구조가 다음과 같다고 보자.

```
class Node {
        String data;
        Node link;
}
```

  - 리스트 이름 week는 첫번째 노드를 가리키는 레퍼런스 변수

```
Node week;
```

  - 공백 연결 리스트를 표현하려면 변수 week에 null을 저장

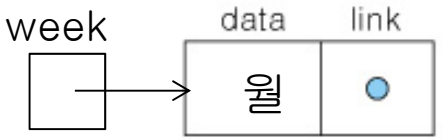
```
week = null;
```

  - 노드 생성

```
week = new Node();
```

  - 노드의 필드는 . 연산자로 액세스

```
week.data = "월";
week.link = null;
```

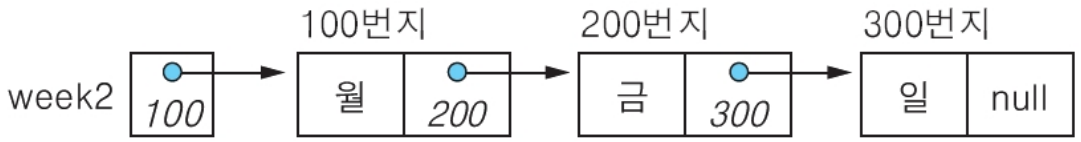


❑ 단순 연결 리스트

❖ 단순 연결 리스트(singly linked list)

- 노드에 링크 필드가 하나이며, 이 링크 필드에 의해 다음 노드와 연결되는 구조를 가짐

- 예)

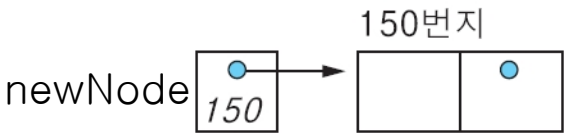


  - 첫번째 노드의 주소를 기억해 두어야 리스트에 접근할 수 있다.

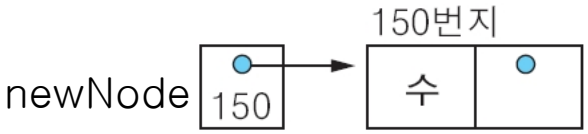
❑ 단순 연결 리스트

- 노드 삽입
  - 리스트 week2=(월, 금, 일) 에서 원소 “월”과 “금”사이에 “수” 삽입하기

① 삽입할 새 공백노드 newNode를 생성한다.

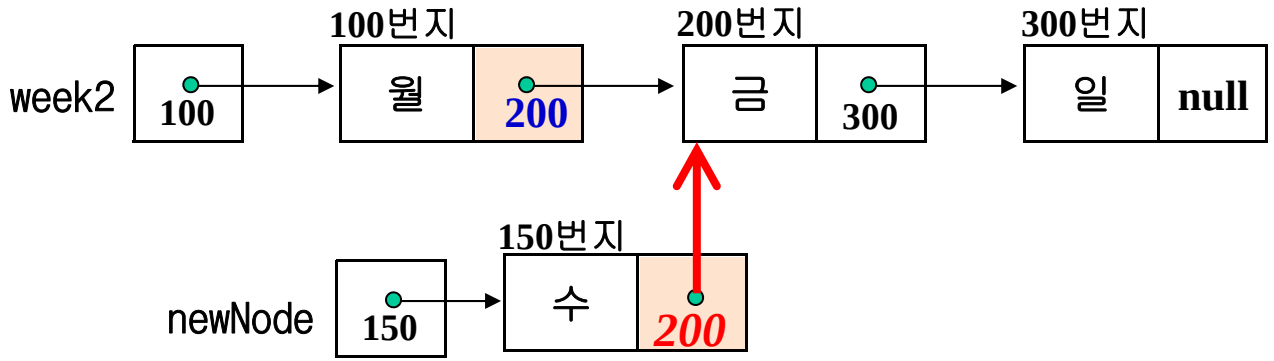


② newNode의 데이터 필드에 “수”를 저장한다.

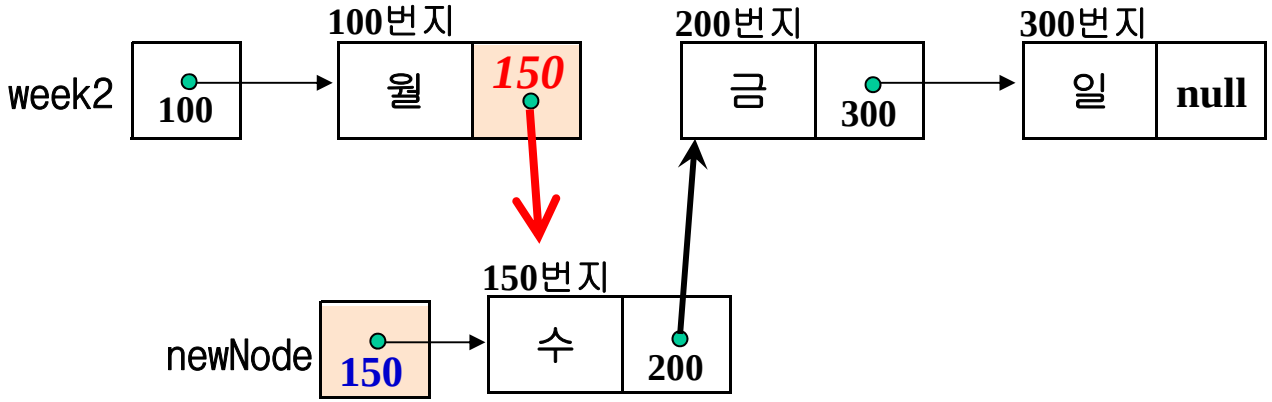


❑ 단순 연결 리스트

③ newNode의 앞노드가 될 노드 , 즉 “월”노드의 링크 필드 값을 newNode의 링크 필드에 저장한다.



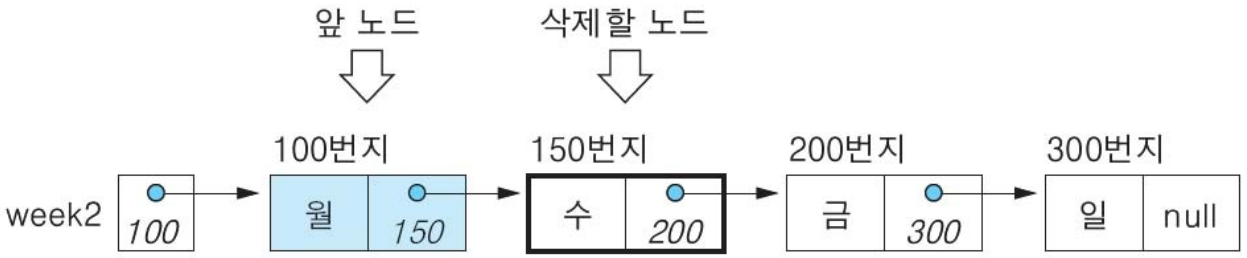
④ “월”노드의 링크 필드에 newNode의 값(newNode가 가리키는 노드의 주소)을 저장한다.



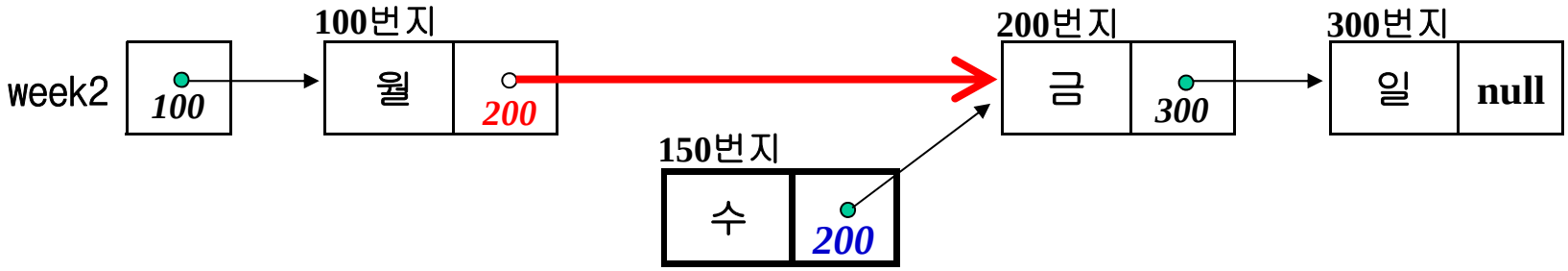
❑ 단순 연결 리스트

- 노드 삭제
  - 리스트 week2=(월, 수, 금, 일)에서 원소 “수” 삭제하기

① 삭제할 원소의 앞 노드(선행자)를 찾는다. ➔ “월” 노드



② “월” 노드의 링크 필드에, 삭제할 원소 “수”의 링크 필드 값을 저장한다.



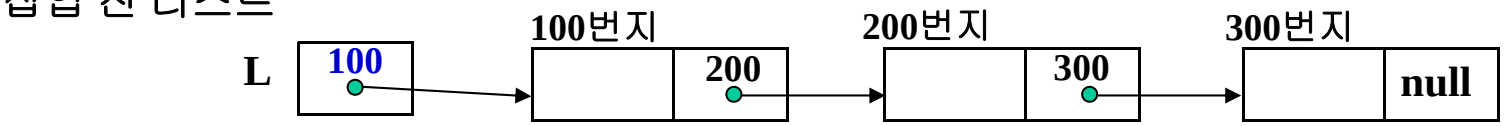
❑ 단순 연결 리스트

❖ 단순 연결리스트의 원소 삽입 알고리즘

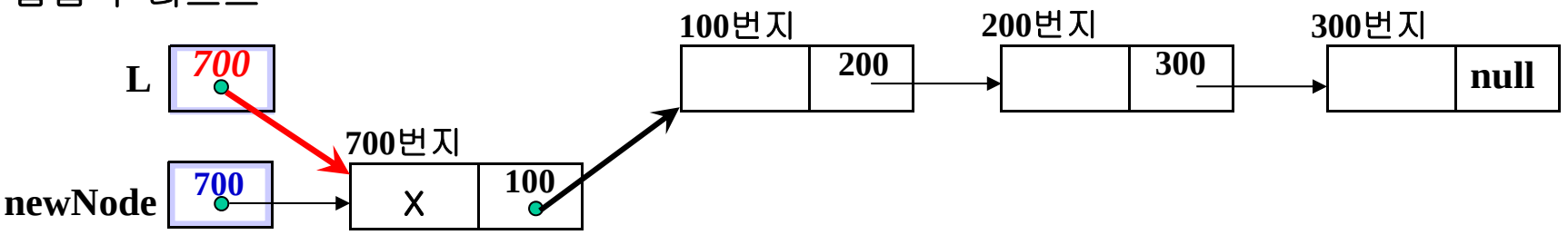
```
insertFirstNode(L, x)
    newNode ← getNode();
    newNode.data ← x;
    newNode.link ← L;
    L ← newNode;
end insertFirstNode()
```

리스트 L의 맨 앞에 원소 x를 삽입하는 알고리즘

삽입 전 리스트



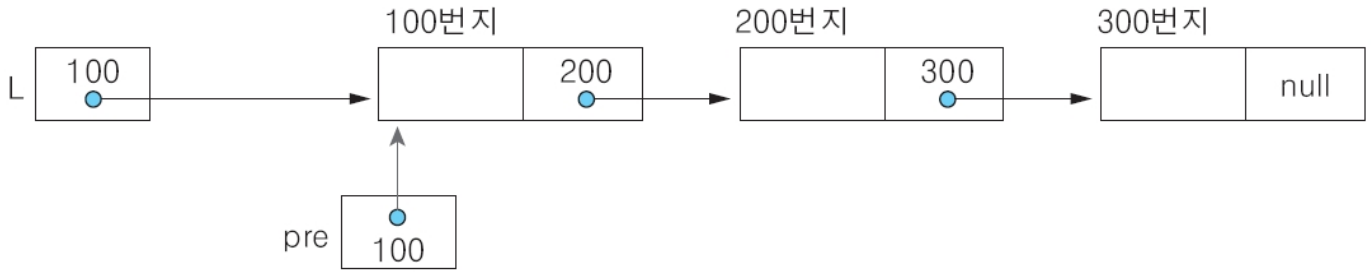
삽입 후 리스트



❑ 단순 연결 리스트

```
insertMiddleNode(L, pre, x)
    newNode ← getNode( );
    newNode.data ← x;
    if (L=null) then {          // L이 공백리스트인 경우
        L ← newNode;
        newNode.link ← null;
    }
    else {                      // L이 공백리스트가 아닌 경우
        newNode.link ← pre.link;
        pre.link ← newNode;
    }
end insertMiddleNode( )
```

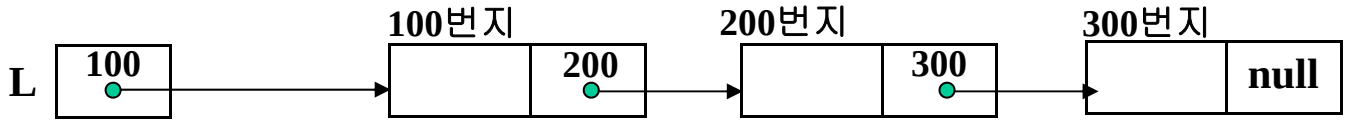
리스트 중간에 삽입 : 리스트 L에서 pre 노드 다음에 x 를 삽입하는 알고리즘



❑ 단순 연결 리스트

리스트의 마지막에 삽입 : 리스트 L의 가장 뒤에 x 값을 삽입하는 알고리즘

```
insertLastNode(L, x)
    newNode ← getNode();
    newNode.data ← x;
    newNode.link ← null;
    if (L = null) then {          // L이 공백리스트인 경우
        L ← newNode;
    }
     else {                       // L이 공백리스트가 아닌 경우
        temp ← L;
        while (temp.link ≠ null) do   // 마지막 노드 temp를 찾음
            temp ← temp.link;
        temp.link ← newNode;
     }
end insertLastNode()
```

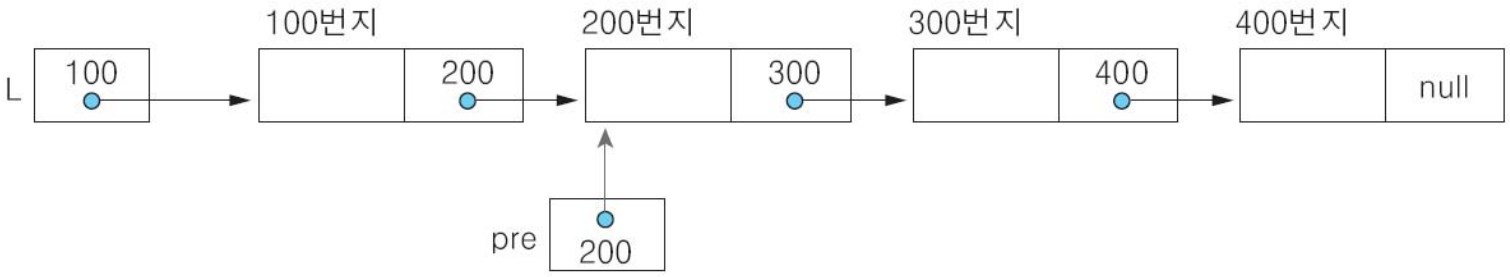


❑ 단순 연결 리스트

❖ 단순 연결 리스트의 노드 삭제 알고리즘

```
deleteNode(L, pre)

    if (L = null) then error;      // L이 공백리스트인 경우
    else {                         // L이 공백리스트가 아닌 경우
        old ← pre.link;            // old : 삭제 대상인 노드
        if (old = null)  then return;
        pre.link ← old.link;
    }
end deleteNode( )
```

리스트 L에서 pre 노드의 다음 노드를 삭제하는 알고리즘



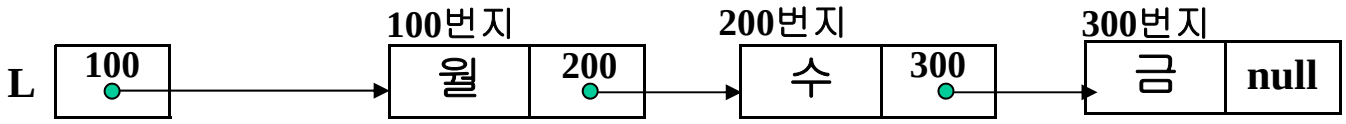
❑ 단순 연결 리스트

❖ 단순 연결 리스트의 노드 탐색 알고리즘

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

리스트 L에서 x 값을 갖는 노드를 찾아 리턴하는 알고리즘

```
searchNode(L, x)
    temp ← L;
    while (temp ≠ null) do {
        if (temp.data = x ) then      // 탐색 성공
            return temp;
        temp ← temp.link;
    }
    return temp;                      // 탐색 실패시 null 리턴
end searchNode()
```



❑ 단순 연결 리스트

❖ 단순 연결 리스트 프로그램

```
public class Ex6_1 {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();

        System.out.println("공백 리스트에 노드 3개 삽입하기");
        list.insertLastNode("월");
        list.insertLastNode("수");
        list.insertLastNode("일");

        System.out.println("월 다음에 금 삽입하기");
        list.insertMiddleNode("월", "금");

        System.out.println("리스트의 노드를 역순으로 바꾸기");
        list.reverseList();

        System.out.println("리스트의 마지막 노드 삭제하기");
        list.deleteLastNode();
        list.printList();
    }
}
```

❑ 단순 연결 리스트

```
public class LinkedList {
    private ListNode head;

    public LinkedList() {
        head = null;
    }

    public void insertMiddleNode(String preData, String data) {
        ListNode pre = searchNode(preData);

        if(pre  == null) {
            System.out.println(preData + "를 찾을 수 없습니다. ");
        }
        else {
            ListNode newNode = new ListNode(data);
            newNode.link = pre.link;
            pre.link = newNode;
        }
    }
   // 다음 슬라이드에 계속
```

❑ 단순 연결 리스트

```
public void insertLastNode(String data) {
    ListNode newNode = new ListNode(data);

    if(head == null) {
        head = newNode;
    }
    else {
        ListNode temp = head;
        while(temp.link != null) temp = temp.link;
        temp.link = newNode;
    }
}

public void insertFirstNode(String data) {
    // ...
}

// 다음 슬라이드에 계속
```

❑ 단순 연결 리스트

```
public void deleteLastNode() {
    ListNode pre, temp;
    if(head == null) return;
    if(head.link == null) {
        head = null;
    }
    else {
        pre = head;
        temp = head.link;
        while(temp.link != null) {
            pre = temp;
            temp = temp.link;
        }
        pre.link = null;
    }
}
// 다음 슬라이드에 계속
```

❑ 단순 연결 리스트

```
public void reverseList() {
    ListNode next = head;
    ListNode current = null;
    ListNode pre = null;

    while(next != null) {
        pre = current;
        current = next;
        next = next.link;
        current.link = pre;
    }

    head = current;
}
// 다음 슬라이드에 계속
```

❑ 단순 연결 리스트

```
public void printList() {
    ListNode temp = head;
    System.out.print("L = (");
    while(temp != null) {
        System.out.print(temp.data);
        temp = temp.link;
        if(temp != null)
            System.out.print(", ");
    }
    System.out.println(")");
}
private ListNode searchNode(String data) {
    ListNode temp = head;
    while(temp != null) {
        if(data == temp.data)  return temp;
        else temp = temp.link;
    }
    return temp;
}
// 다음 슬라이드에 계속
```

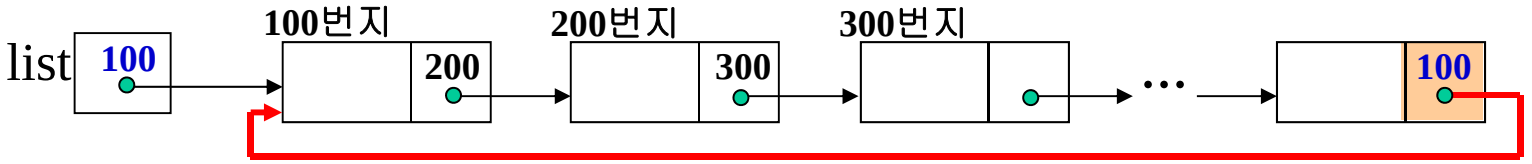
❑ 단순 연결 리스트

```
    private class ListNode {
        String data;
        ListNode link;

        ListNode() {
            data = null;
            link = null;
        }

        ListNode(String data) {
            this.data = data;
            this.link = null;
        }

        ListNode(String data, ListNode link) {
            this.data = data;
            this.link = link;
        }
    }
}
```

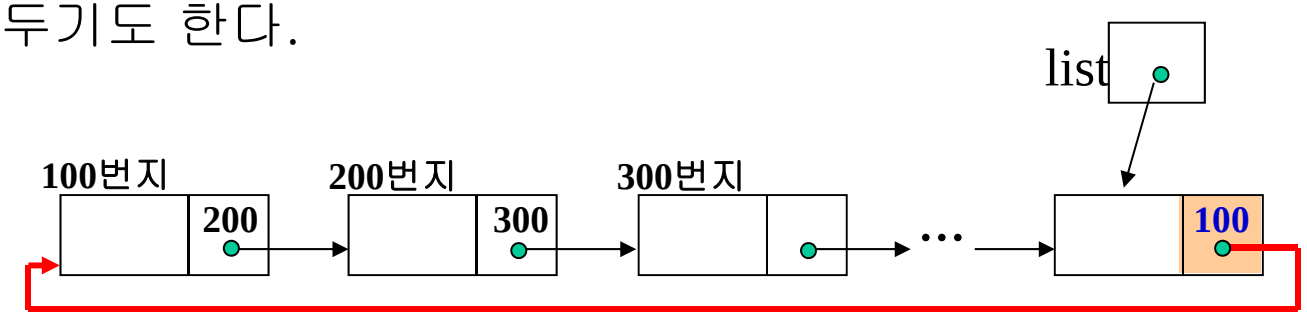
❑ 원형 연결 리스트

❖ 원형 연결 리스트(circular linked list)

- 단순 연결 리스트에서 마지막 노드가 리스트의 첫 번째 노드를 가리키게 하여 리스트의 구조를 원형으로 만든 연결 리스트
  - 마지막 노드의 링크 필드에 첫 번째 노드의 주소를 저장



- 첫번째 노드의 주소를 기억해 두는 대신, 마지막 노드의 주소를 기억해두기도 한다.



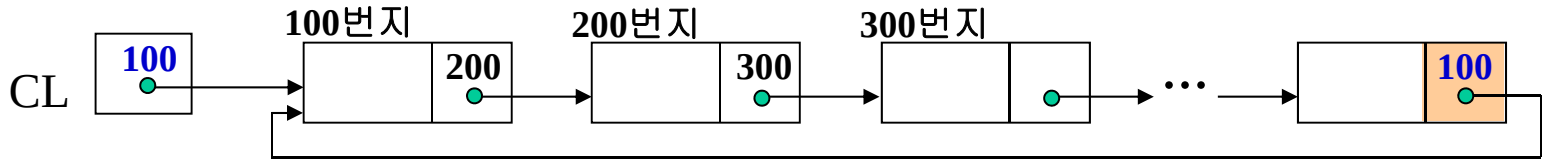
  - 첫번째 노드와 마지막 노드를 둘 다 빨리 접근할 수 있음

❑ 원형 연결 리스트

❖ 원형 연결 리스트의 원소 삽입 알고리즘

첫 번째 노드로 삽입 : 원형 연결 리스트 CL의 맨 앞에 x를 삽입하는 알고리즘

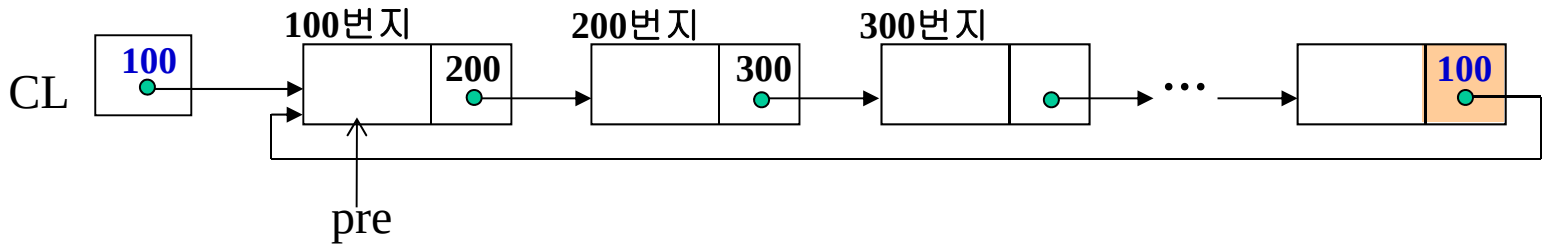
```
insertFirstNode(CL, x)
    newNode ← getNode();
    newNode.data ← x;
    if (CL = null) then {          // CL이 공백 리스트인 경우
        CL ← newNode;
        newNode.link ← newNode;
    }
    else{                          // CL이 공백 리스트가 아닌 경우
        temp ← CL;
        while (temp.link ≠ CL) do temp ← temp.link; // 마지막 노드 탐색
        newNode.link ← temp.link;
        temp.link ← newNode;
        CL ← newNode;
    }
end insertFirstNode()
```



❑ 원형 연결 리스트

```
insertMiddleNode(CL, pre, x)
    newNode ← getNode();
    newNode.data ← x;
    if (CL=null) then {          // CL이 공백 리스트인 경우
        CL ← newNode;
         newNode.link ← newNode;
    }
    else {                       // CL이 공백 리스트가 아닌 경우
        newNode.link ← pre.link;    // pre 다음에 newNode를 삽입
        pre.link ← newNode;
    }
end insertMiddleNode()
```

중간 노드로 삽입 : 원형 연결 리스트
CL에서 pre 노드 바로 다음에 x를 삽입

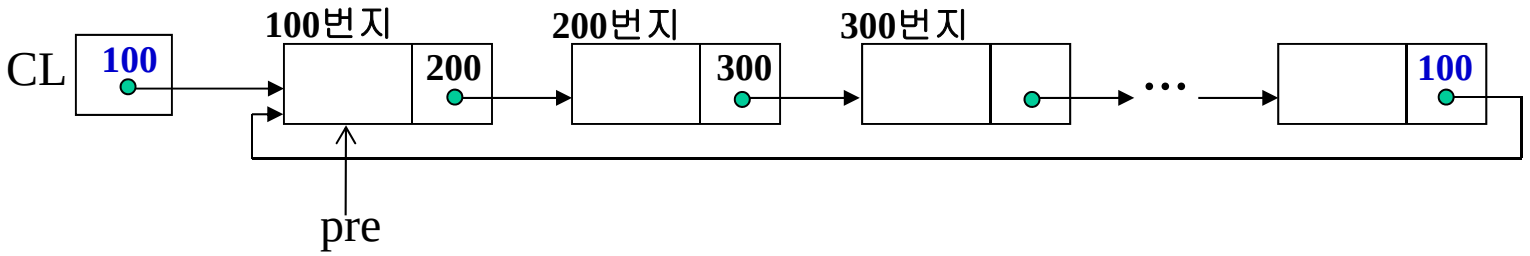


❑ 원형 연결 리스트

❖ 원형 연결 리스트의 노드 삭제 알고리즘

```
deleteNode(CL, pre)
    if (CL = null) then error;
    else {
        old ← pre.link;
        pre.link ← old.link;
        if (old = pre) then          // 리스트 노드가 하나인 경우
            CL ← null;
        else if (old = CL) then      // pre가 마지막 노드인 경우
            CL ← old.link;
    }
end deleteNode()
```

원형 연결리스트 CL에서 pre 노드의 다음 노드를 삭제하는 알고리즘



❑ 이중 연결 리스트

❖ 이중 연결 리스트(doubly linked list)

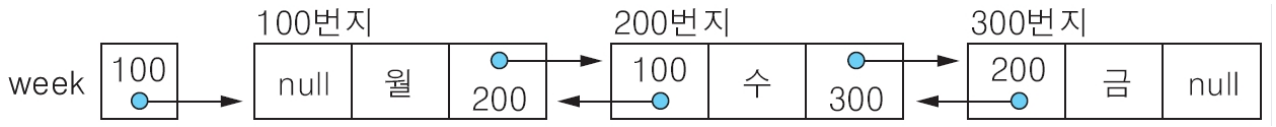
- 양쪽 방향으로 순회할 수 있도록 노드들을 연결한 리스트
- 이중 연결 리스트의 노드 구조
  - 데이터 필드와 두 개의 링크 필드로 구성
    - ➢ llink (left link) 필드 : 왼쪽 노드와 연결
    - ➢ rlink (right link) 필드 : 오른쪽 노드와 연결

  - 노드 구조에 대한 클래스 정의

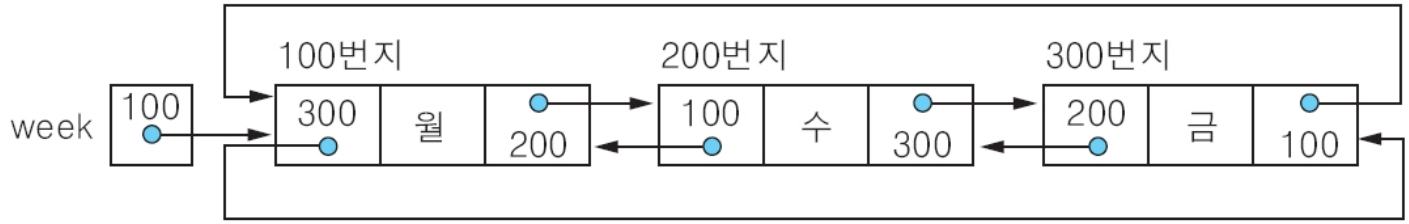
```
class DblNode{
        String data;
        DblNode llink;
        DblNode rlink;
 }
```

❑ 이중 연결 리스트

- 리스트 week=(월, 수, 금)의 이중 연결 리스트 구성



- 원형 이중 연결 리스트
  - 이중 연결 리스트를 원형으로 구성

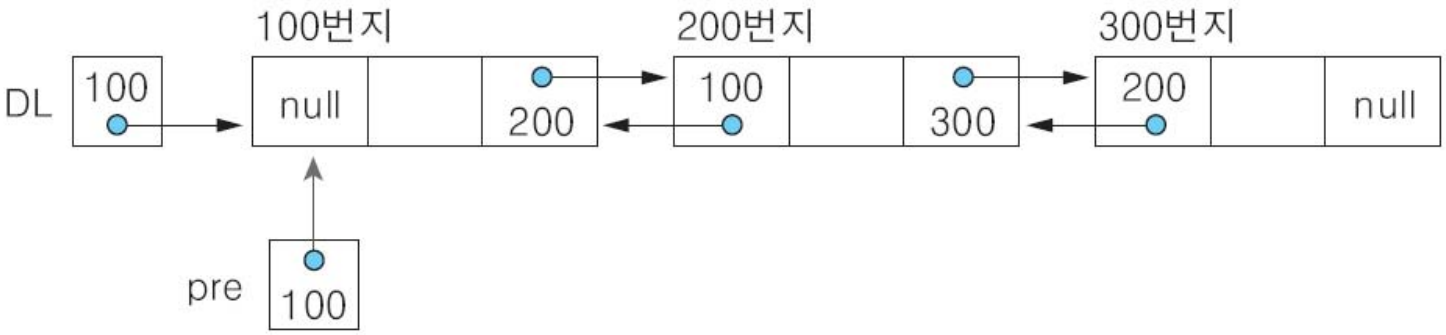


❑ 이중 연결 리스트

❖ 이중 연결리스트의 원소 삽입 알고리즘

```
insertNode(DL, pre, x)
    newNode ← getNode();
    newNode.data ← x;
    newNode.rlink ← pre.rlink ;
    pre.rlink ← newNode;
    newNode.llink ← pre;
    if (newNode.rlink ≠ null) then   // 마지막 노드가 아니면
        newNode.rlink.llink ← newNode;
end insertNode()
```

이중 연결리스트 DL에서 pre 노드 다음에 x 값을 삽입하는 알고리즘

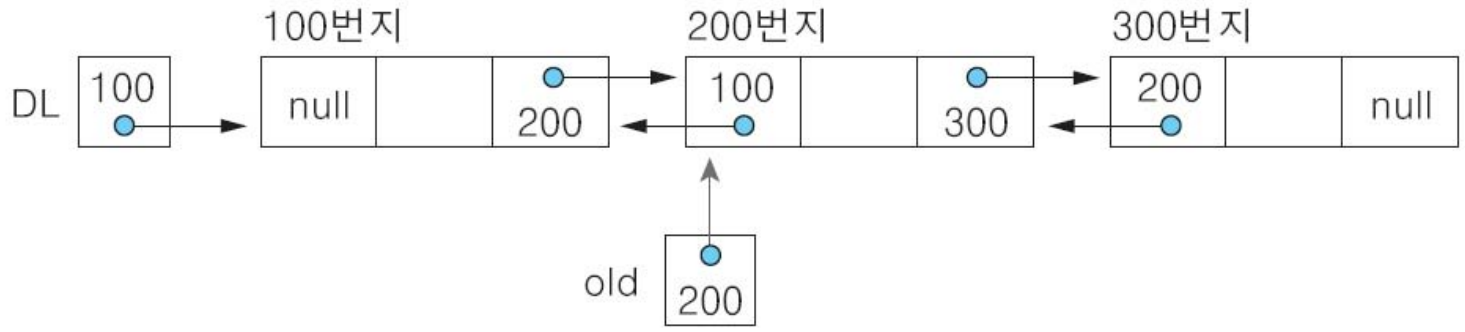


❑ 이중 연결 리스트

❖ 이중 연결리스트의 노드 삭제 알고리즘

이중 연결 리스트 DL에서 old 노드를 삭제하는 알고리즘

```
deleteNode(DL, old)
    if (old.llink ≠ null) then   // old가 첫번째 노드가 아니면
        old.llink.rlink ← old.rlink;
    else   // old가 첫번째 노드이면
        DL = old.rlink;
    if (old.rlink ≠ null) then // old가 마지막 노드가 아니면
        old.rlink.llink ← old.llink;
end deleteNode( )
```



❑ 다항식의 연결 자료구조 표현

❖ 단순 연결 리스트를 이용한 다항식 표현

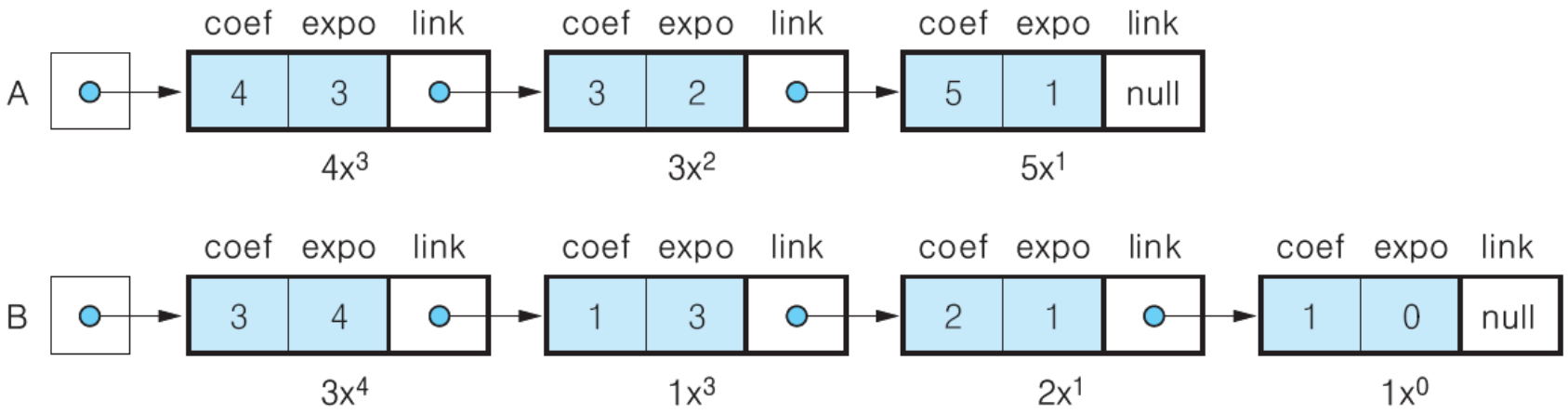
- 다항식의 각 항을 하나의 노드로 표현
  - 각 항의 계수와 지수를 저장
- 노드 구조
  - 두 개의 데이터 필드
    - ➢ 계수를 저장하는 coef
    - ➢ 지수를 저장하는 expo
  - 링크 필드 link

- 노드에 대한 클래스 정의

```
class Node {
    float coef;
    int expo;
    Node link;
}
```

❑ 다항식의 연결 자료구조 표현

- 다항식의 단순 연결 리스트 표현 예
  - 다항식 $A(x)=4x^3+3x^2+5x$
  - 다항식 $B(x)=3x^4+x^3+2x+1$

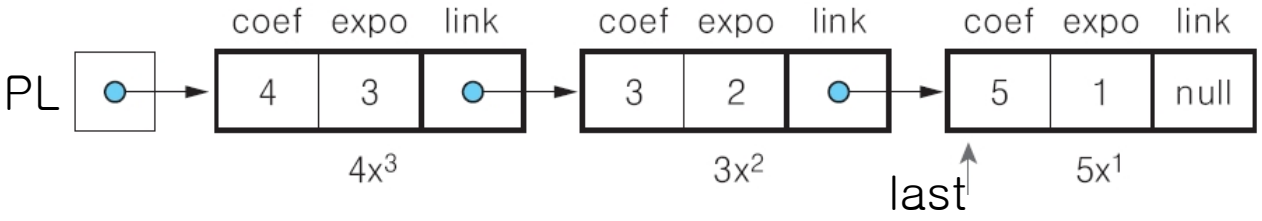


❑ 다항식의 연결 자료구조 표현

❖ 다항식 리스트에 항 추가(append)

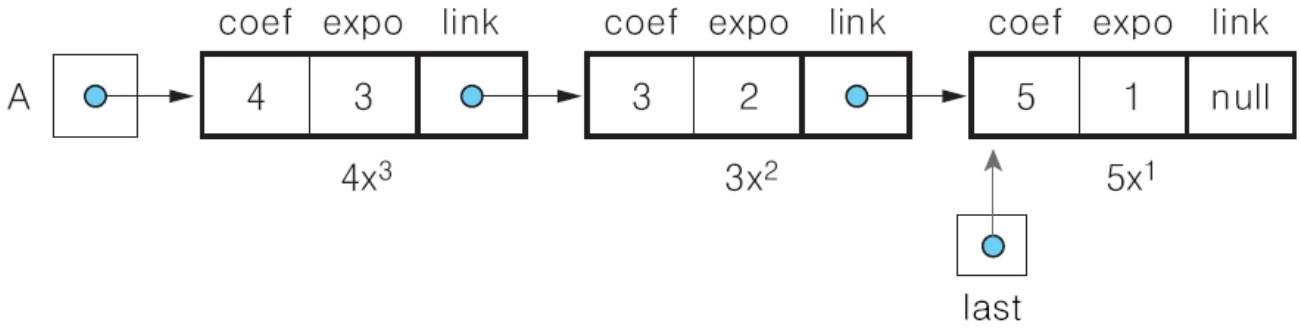
```
appendTerm(PL, coef, expo, last)
    newNode ← getNode( );
    newNode.expo ← expo;
    newNode.coef ← coef;
    newNode.link ← null;
    if (PL = null) then {          // PL이 공백 다항식일 때
        PL ← newNode;
        last ← newNode;
    }
    else {                         // PL이 공백 다항식이 아닐 때
        last.link ← newNode;
        last ← newNode;
    }
end appendTerm( )
```

다항식 리스트 PL의 마지막에 계수가 coef, 지수가 expo인 항을 추가하는 알고리즘 (last는 리스트의 마지막 노드를 가리킴)

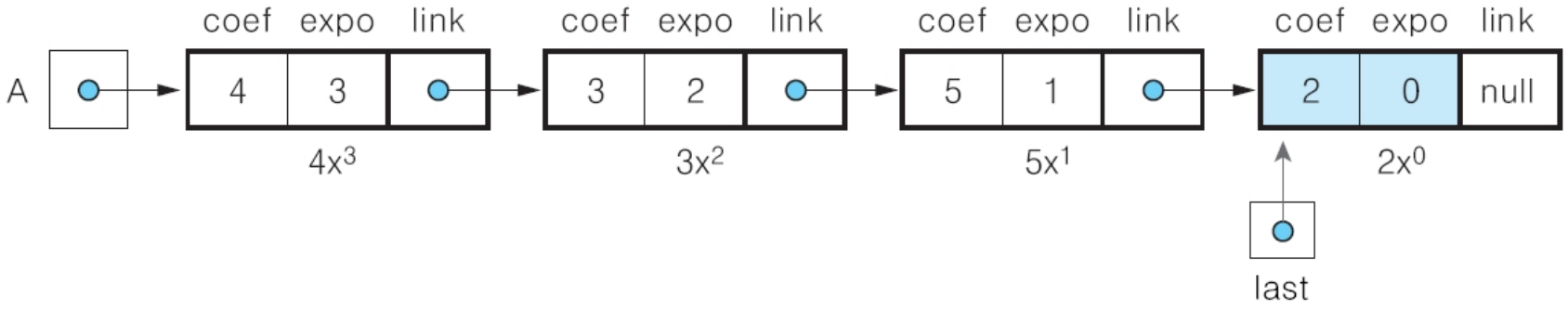


❑ 다항식의 연결 자료구조 표현

- 예) 다항식 리스트 A에 appendTerm() 알고리즘을 사용하여 $2x^0$항 (즉, 상수항 2)을 추가



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

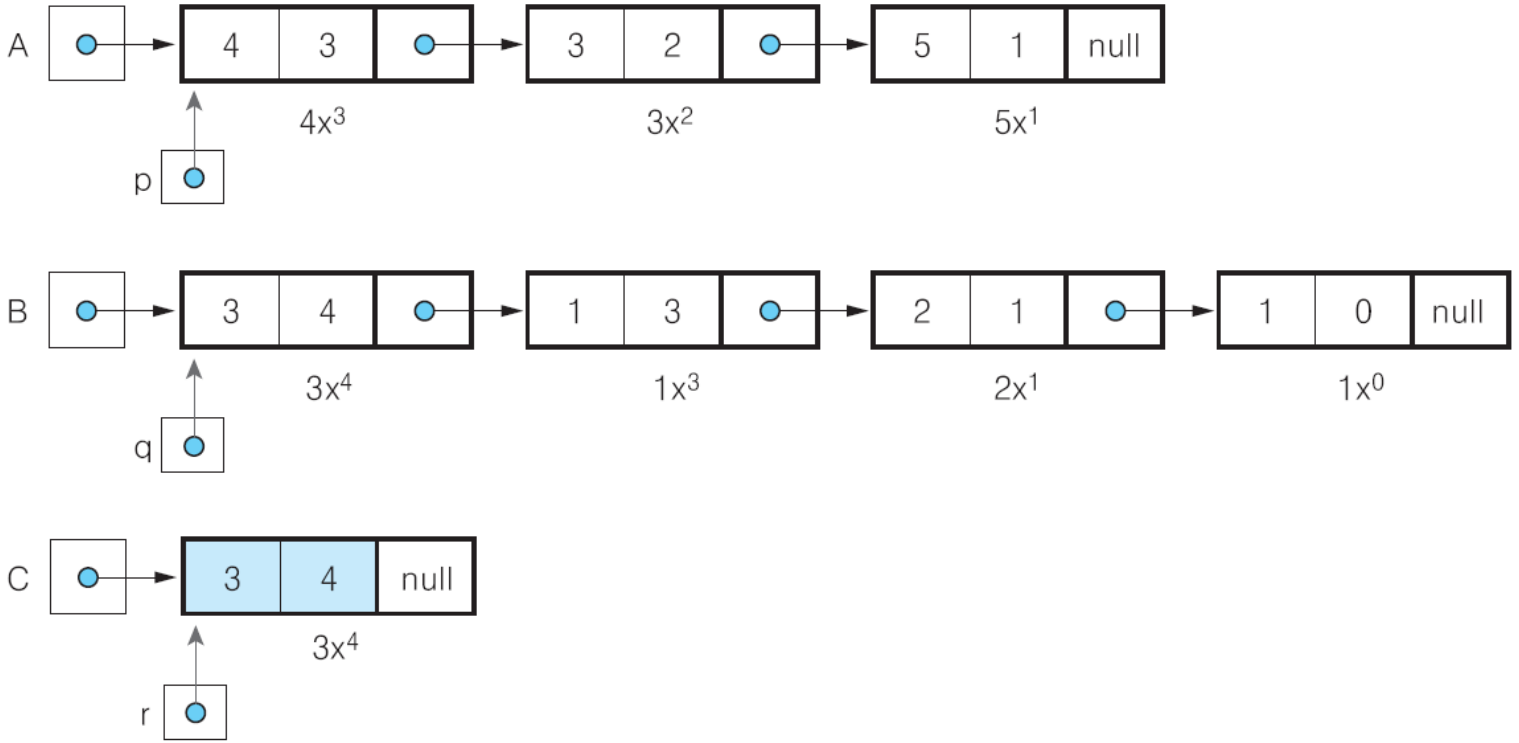


(b) appendTerm(A,2,0,last) 함수 실행 후의 다항식 리스트 A

❑ 다항식의 연결 자료구조 표현

❖ 다항식의 덧셈 연산

- 덧셈 A(x)+B(x)=C(x)를 단순 연결 리스트 자료구조로 연산
- 세 개의 참조 변수를 사용
  - p : 다항식 A(x)에서 비교할 항을 지시
  - q : 다항식 B(x)에서 비교할 항을 지시
  - r : 덧셈연산 결과 만들어지는 다항식 C(x)의 항을 지시



❑ 다항식의 연결 자료구조 표현

① p.expo < q.expo : 다항식 A(x) 항의 지수가 작은 경우

- q가 가리키는 다항식 B(x)의 항을 C(x)의 항으로 복사
- q를 다음 노드로 이동

② p.expo = q.expo : 두 항의 지수가 같은 경우

- p.coef와 q.coef를 더하여 C(x)의 항, 즉 r.coef에 저장하고 지수가 같아야 하므로 p.expo(또는 q.expo)를 r.expo에 저장
- 다음 항을 비교하기 위해 p와 q를 각각 다음 노드로 이동

③ p.expo > q.expo : 다항식 A(x) 항의 지수가 큰 경우

- p가 가리키는 다항식 A(x)의 항을 C(x)의 항으로 복사
- p를 다음 노드로 이동

❑ 다항식의 연결 자료구조 표현

다항식 덧셈: 단순 연결 리스트로 표현된 다항식 A와 B를 더하여 새로운 다항식 C를 리턴

```
addPoly(A, B)
    p ← A;
    q ← B;
    C ← null;     // 결과 다항식
    r ← null;     // 결과 다항식의 마지막 노드를 가리킴
    while (p ≠ null and q ≠ null) do {  // p, q는 순회용 참조변수
        case {
            p.expo = q.expo :
                sum ← p.coef + q.coef
                if (sum ≠ 0) then appendTerm(C, sum, p.expo, r);
                p ← p.link;
                q ← q.link;
            p.expo < q.expo :
                appendTerm(C, q.coef, q.expo, r);
                q ← q.link;
            else : // p.expo > q.expo인 경우
                appendTerm(C, p.coef, p.expo, r);
                p ← p.link;
        } // end case
    } // end while                                  // 다음 슬라이드에 계속
```

❑ 다항식의 연결 자료구조 표현

```
    while (p ≠ null) do {   // A의 나머지 항들을 C에 복사
        appendTerm(C, p.coef, p.expo, r);
        p ← p.link;
    }
    while (q ≠ null) do {   // B의 나머지 항들을 C에 복사
        appendTerm(C, q.coef, q.expo, r);
        q ← q.link;
    }
    return C;
end addPoly()
```

❑ 다항식의 연결 자료구조 표현

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

  - 초기 상태