5주차 3차시 단순 연결 리스트2

【학습목표】

1. 단순 연결 리스트에서 노드를 삽입할 때와 삭제할 때, 탐색할 때 사용하는 알고리즘에 대하여 설명할 수 있다.
2. 첫 번째 노드와 중간 노드, 마지막 노드 삽입을 구분할 수 있다.

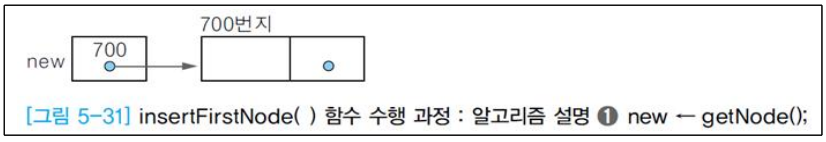
학습내용1 : 단순 연결 리스트의 삽입 알고리즘

1. 첫 번째 노드로 삽입하기

알고리즘 5-3 리스트의 첫 번째 노드 삽입 알고리즘

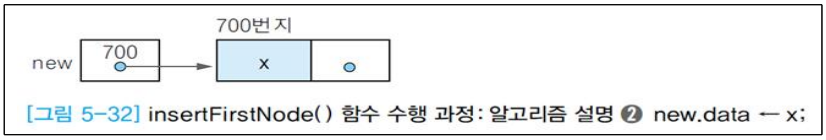
```
insertFirstNode(L, x)
      new ← getNode();      // ❶
      new.data ← x;         // ❷
      new.link ← L;         // ❸
      L ← new;              // ❹
end insertFirstNode()
```

❶ 삽입할 노드를 자유 공간 리스트에서 할당받는다



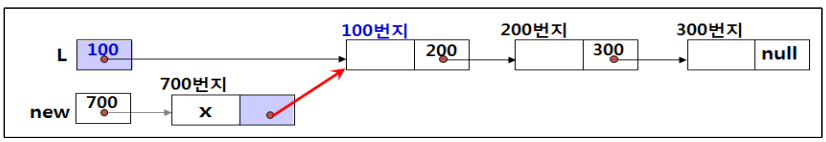
[그림 5-31] insertFirstNode( ) 함수 수행 과정 : 알고리즘 설명 ❶ new ← getNode();

❷ 새 노드의 데이터 필드에 x를 저장한다

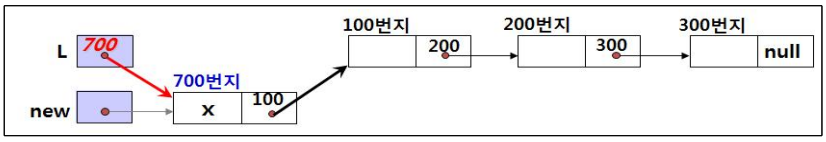


[그림 5-32] insertFirstNode( ) 함수 수행 과정: 알고리즘 설명 ❷ new.data ← x;

❸ 삽입할 노드를 연결하기 위해서 리스트의 첫 번째 노드 주소(L, 100)를 삽입할 새 노드 new의 링크 필드(new.link)에 저장하여 새 노드 new가 리스트의 첫 번째 노드를 가리키게 한다.



❹ 리스트의 첫 번째 노드 주소를 저장하고 있는 포인터 L에 새 노드의 주소(new, 700)을 저장하여 포인터 L이 새 노드를 리스트의 첫 번째 노드로 가리키도록 연결한다



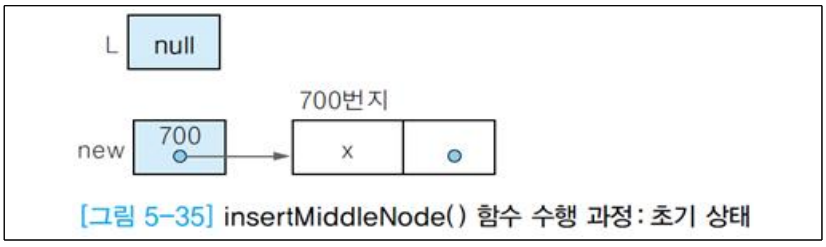
2. 중간 노드로 삽입하기

* 리스트 L에서 포인터변수 pre가 가리키고 있는 노드의 다음에 데이터 필드값이 x인 새 노드를 삽입하는 알고리즘

알고리즘 5-4 리스트의 중간 노드 삽입 알고리즘

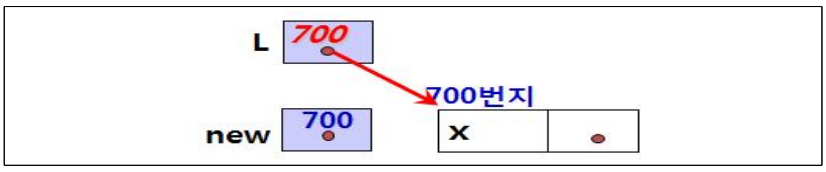
```
insertMiddleNode(L, pre, x)
        new ← getNode();
        new.data ← x;
        if (L=null) then {                  // ❶ 공백 리스트일 경우
                L ← new;                    // ❶-ⓐ
                new.link ← null;            // ❶-ⓑ
        }
        else {                              // ❷ 공백 리스트가 아닐 경우
                new.link ← pre.link;        // ❷-ⓐ
                pre.link ← new;             // ❷-ⓑ
         }
end insertMiddleNode()
```

❶ [리스트 L이 공백인 경우]

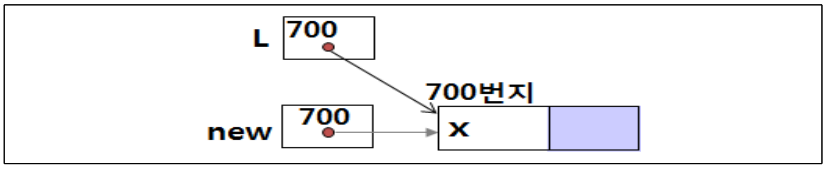


[그림 5-35] insertMiddleNode() 함수 수행 과정 : 초기 상태

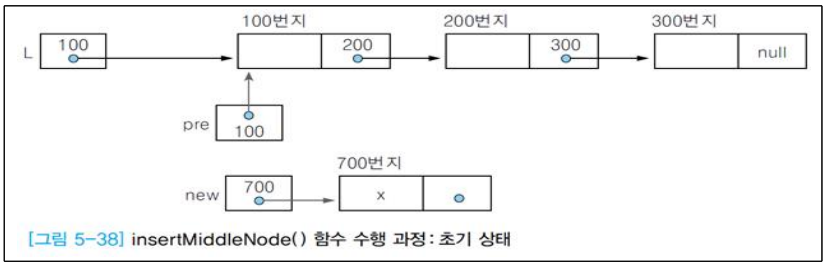
❶-ⓐ 리스트 포인터 L에 새 노드 new의 주소(700)를 저장하여 new가 리스트의 첫 번째 노드가 되도록 한다



❶-ⓑ 리스트의 마지막 노드인 new의 링크 필드에 null을 저장하여 마지막 노드임을 표시한다

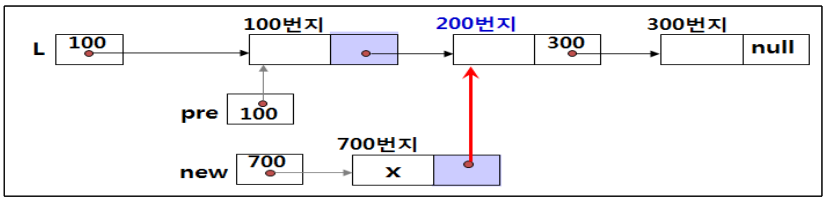


❷ [리스트 L이 공백이 아닌 경우]

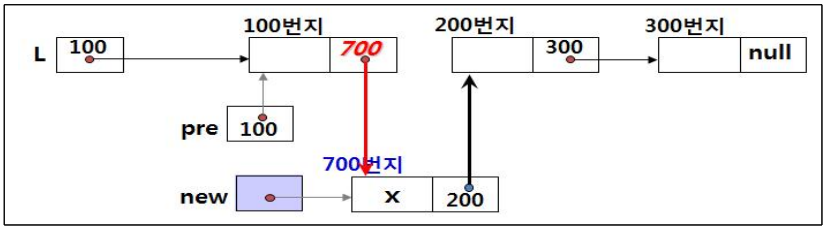


[그림 5-38] insertMiddleNode() 함수 수행 과정 : 초기 상태

❷-ⓐ 포인터 pre가 가리키는 노드의 다음 노드로 새 노드 new를 연결해야 하므로 pre의 링크 필드값(200)을 노드 new의 링크 필드에 저장하여 새 노드 new가 노드 pre의 다음 노드를 가리키도록 한다



❷-ⓑ 포인터 new의 값(700)을 노드 pre의 링크 필드에 저장하여 노드 pre가 새 노드 new로 연결되도록 한다



3. 마지막 노드로 삽입하기

알고리즘 5-5 리스트의 마지막 노드 삽입 알고리즘

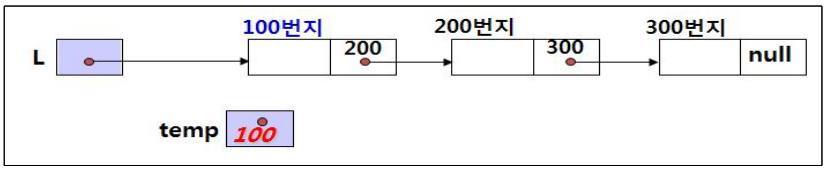
```
insertLastNode(L, x)
        new ← getNode();
        new.data ← x;
        new.link ← null;
        if (L = null) then {                    // ❶ 공백 리스트인 경우
             L ← new;
             return;
        }                                       // ❷ 공백 리스트가 아닌 경우
        temp ← L;                               // ❷-ⓐ
        while (temp.link ≠ null) do
             temp ← temp.link;                  // ❷-ⓑ
        temp.link ← new;                        // ❷-ⓒ
end insertLastNode()
```

- 링크 필드가 null인 노드가 마지막 노드
- 리스트이 노드를 순회하는 임시 포인터 temp가 필요

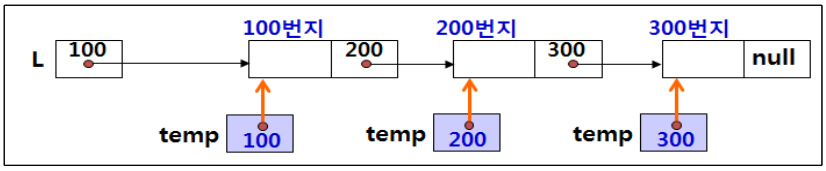
❶ [리스트 L이 공백인 경우]
- 중간 노드 삽입 알고리즘의 공백 리스트에 노드 삽입하는 연산과 동일
- 삽인하는 새 노드 new는 리스트 L의 첫 번째 노드이자 마지막 노드

[그림 5-41] insertLastNode( ) 함수 수행 과정: 초기 상태

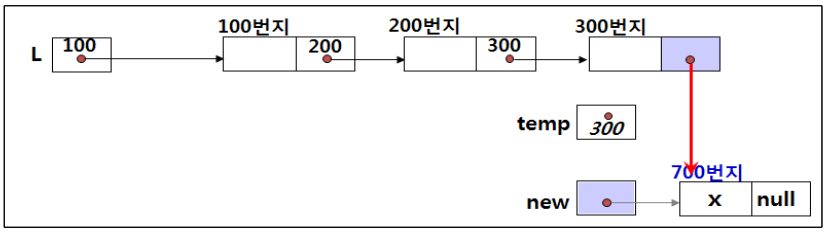
❷ [리스트 L이 공백이 아닌 경우]
❷-ⓐ 현재 리스트 L의 마지막 노드를 찾기 위해서 노드를 순회할 임시포인터 temp에 리스트이 첫 번째 노드의 주소(L)을 지정



❷-ⓑ while 반복문을 수행하여 순회 포인터 temp가 노드의 링크 필드를 따라 이동하면서 링크필드가 null인 마지막 노드를 찾도록 한다



❷-ⓒ 순회 포인터 temp가 가리키는 노드 즉, 리스트의 마지막 노드의 링크 필드(temp.lijk)에 삽입할 새 노드 new의 주소를 저장하여 리스트이 마지막 노드가 새 노드 new를 연결



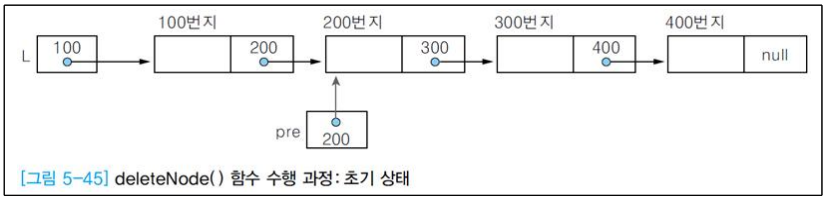
학습내용2 : 단순 연결 리스트의 삭제 알고리즘

알고리즘 5-6 리스트의 노드 삭제 알고리즘

```
deleteNode(L, pre)
        if (L = null) then error;              // ❶ 공백 리스트인 경우
        else {                                 // ❷ 공백 리스트가 아닌 경우
                old ← pre.link;                // ❷-ⓐ
                if (old = null) then return;   // ❷-ⓑ
                pre.link ← old.link;           // ❷-ⓒ
        }
        returnNode(old);                       // ❷-ⓓ
end deleteNode()
```

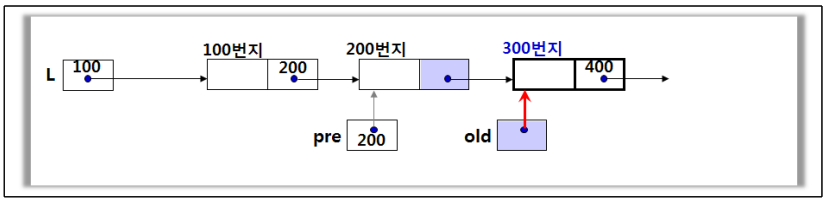
- 포인터 pre가 가리키는 노드의 다음 노드를 삭제하는 알고리즘
- 포인터 old가 삭제할 노드

❶ 삭제 연산을 수행하기 위해서는 반드시 하나 이상의 노드가 존재해야 함
- 삭제 연산을 수행하기 전에 리스트 L이 공백 리스트인지를 검사하여 공백 리스트인 경우 에러 처리(Underflow 처리)를 한다

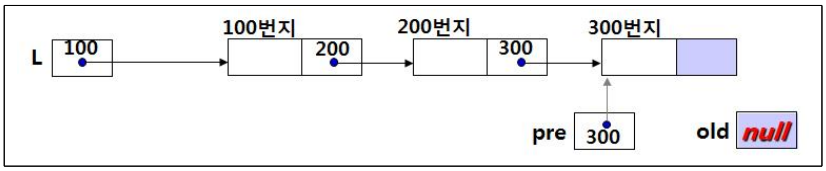


[그림 5-45] deleteNode( ) 함수 수행 과정: 초기 상태

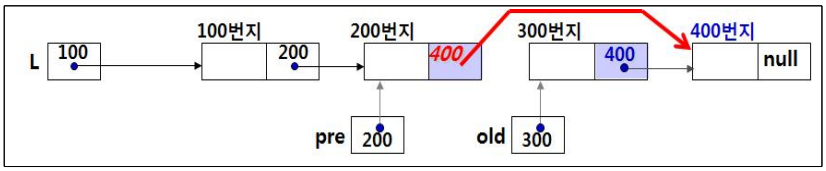
❷ [리스트 L이 공백이 아닌 경우]
❷-ⓐ 노드 old가 삭제할 노드를 가리키게 하기 위해서 노드 pre의 링크 필드값(300)을 노드 old에 저장하여 포인터 old가 pre 다음 노드를 가리키도록 한다



❷-ⓑ if (old = null) then return;
- 만약 노드 pre가 리스트의 마지막 노드였다면;
- 노드 pre 다음에 삭제할 노드가 없다는 의미로 삭제 연산을 수행하지 못하고 반환함



❷-ⓒ 삭제할 노드 old의 다음 노드(old.link, 400)를 노드 pre의 다음 노드(pre.link)가 되도록 연결한다



❷-ⓓ 삭제한 노드 old를 자유 공간 리스트에 반환

학습내용3 : 단순 연결 리스트의 노드 탐색 알고리즘

1. 노드 탐색 알고리즘

- 연결 리스트에서 원소값이 x인 노드를 탐색하려면 리스트의 노드를 처음부터 하나씩 순회하면서 노드의 데이터 필드값과 x를 비교하여 일치하는 노드를 찾음

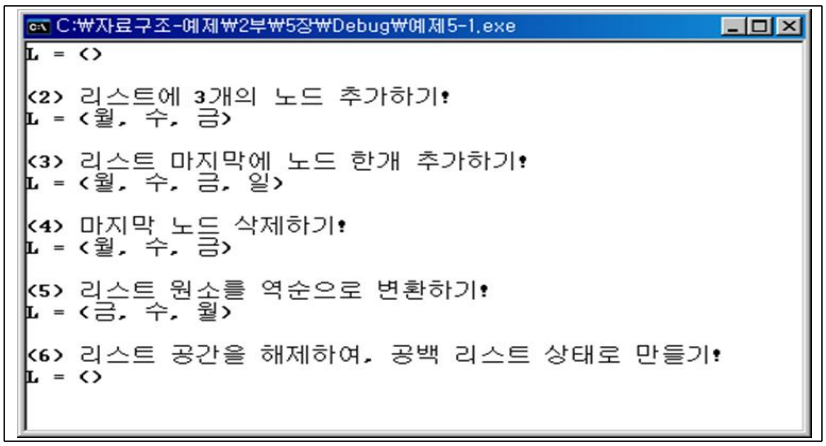
알고리즘 5-7 리스트 L에서 x 노드 탐색 알고리즘

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

❶ 리스트 L으 시작 주소를 노드 탐색을 위해 사용할 순회 포인터 temp에 저장하여 포인터 temp의 시작 위치를 지정
❷ 순회 포인터 temp가 null이 아닌 동안 즉, temp가 가리키는 노드가 마지막 노드가 될 때까지 노드 temp의 데이터 필드값이 x인지를 검사하여 x이면 그 노드의 주소를 반환하고 x가 아니면 링크를 따라 다음 노드로 순회 포인터 temp를 이동시킨다
❸ while 반복문에서 x 노드를 찾지 못하고 결국 리스트의 마지막 노드에서 while문 수행이 끝난 상태이므로 포인터 temp를 반환한다

2. 단순 연결 리스트 프로그램

- [예제 5-1]은 앞에서 다룬 삽입, 삭제 및 노드 탐색 알고리즘을 활용한 단순 연결 리스트 프로그램 리스트
- 실행결과



【학습정리】

1. 리스트의 처음부터 마지막까지 노드를 하나씩 순회하면서 노드의 필드값과 일치하는 노드를 찾는 것을 노드 탐색이라고 한다.