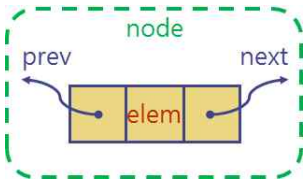
<알고리즘 실습> - 자료구조 복습

※ 입출력에 대한 안내

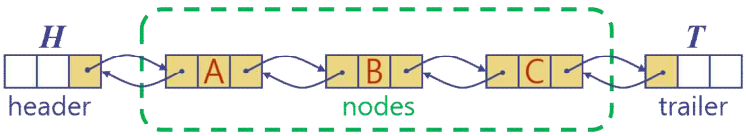
- 특별한 언급이 없으면 문제의 조건에 맞지 않는 입력은 입력되지 않는다고 가정하라.
- 특별한 언급이 없으면, 각 줄의 맨 앞과 맨 뒤에는 공백을 출력하지 않는다.
- 출력 예시에서 □는 각 줄의 맨 앞과 맨 뒤에 출력되는 공백을 의미한다.
- 입출력 예시에서 ↦ 이 후는 각 입력과 출력에 대한 설명이다.

이중연결리스트 + 헤더 및 트레일러 노드 (문제 1 참고 내용)

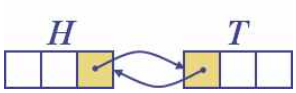
1. 연결리스트 구조
   - 각 노드에 저장되는 정보
     - elem: 원소
     - prev: 이전 노드를 가리키는 링크
     - next: 다음 노드를 가리키는 링크



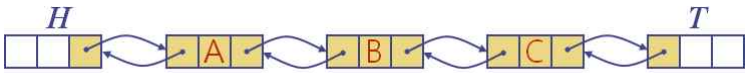
   - 헤더 및 트레일러 노드
     - 데이터를 가지지 않는 특별 노드



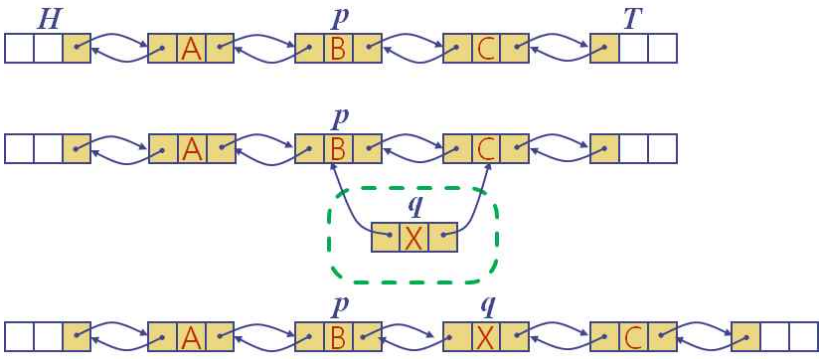
2. 이중연결리스트의 초기화
   - 초기에는 헤더 및 트레일러 노드만 존재
   - O(1) 시간 소요



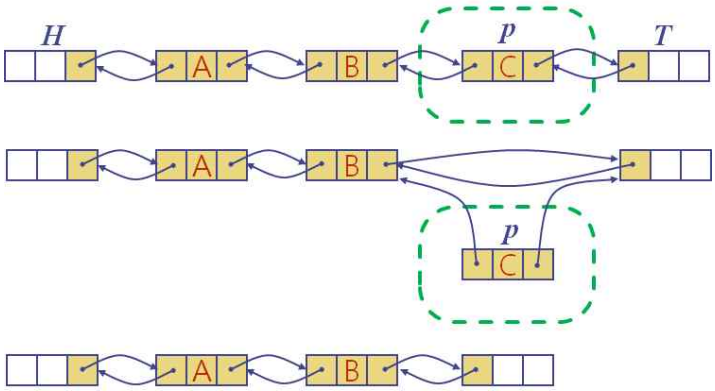
3. 이중연결리스트의 순회
   - 연결리스트의 모든 원소들을 방문 (순회하면서 필요한 작업 수행. 예를 들면 출력)
   - O(n) 시간 소요



4. 이중연결리스트에서 삽입
   - 이중연결리스트의 지정된 노드(p) 뒤에 노드를 삽입하고, 노드 q의 주소를 반환
   - O(1) 시간 소요



5. 이중연결리스트에서 삭제
   - 이중연결리스트로부터 지정된 위치의 노드(p)를 삭제하고, 원소 값 반환
   - O(1) 시간 소요



[문제 1] 위에서 설명한 이중연결리스트를 구현하고, 영문자 리스트 ADT를 구현하시오.

   - 다음 네 가지 연산을 지원해야 함 (위치는 1번째부터 시작한다고 가정)
     - add(list, position, item) : list의 position번째에 item을 추가한다.
     - delete(list, position) : list의 position번째 위치한 item을 삭제한다.
     - get_entry(list, position) : list의 position번째 위치한 item 값을 리턴한다.
     - print(list) : list의 모든 item을 list에 저장된 순서대로 공백없이 출력한다.

     ※ position 정보가 유효하지 않으면 화면에 에러 메시지 "invalid position" 출력하고, 해당 연산은 무시한다.

   - 입력에 대한 설명 (아래 입출력 예시 참조)
     - 각 연산의 내용이 한 줄에 하나씩 입력되고, 하나의 줄에는 연산의 종류, 위치, 아이템이 차례로 입력된다.
     - 연산의 종류: 연산 이름의 맨 앞 영문자가 대문자 (A, D, G, P) 로 주어진다.
     - 위치: 양의 정수
     - 아이템: 영문자 (대문자, 소문자 모두 가능)

입력 예시 1

```
5          ↦ 연산의 개수: 5
A 1 S      ↦ add(list, 1, 'S')
A 2 t      ↦ add(list, 2, 't')
A 3 r      ↦ add(list, 3, 'r')
A 3 a      ↦ add(list, 3, 'a')
P          ↦ print(list)
```

출력 예시 1

```
Star       ↦ 5번째 연산(P)에 의한 출력
```

입력 예시 2

```
9          ↦ 연산의 개수: 9
A 1 D      ↦ add(list, 1, 'D')
A 2 a      ↦ add(list, 2, 'a')
A 3 y      ↦ add(list, 3, 'y')
D 1        ↦ delete(list, 1)
P          ↦ print(list)
G 3        ↦ get_entry(list, 3)
A 1 S      ↦ add(list, 1, 'S')
P          ↦ print(list)
G 3        ↦ get_entry(list, 3)
```

출력 예시 2

```
ay                ↦  5번째 연산(P)에 의한 출력
invalid position  ↦ 6번째 연산 (G 3)에 의한 출력
Say               ↦ 8번째 연산 (P)에 의한 출력
y                 ↦ 9번째 연산 (G 3)에 의한 출력
```

이진트리 만들기 및 탐색 (문제 2 참고 내용)

1. 트리 만들기 (구현)

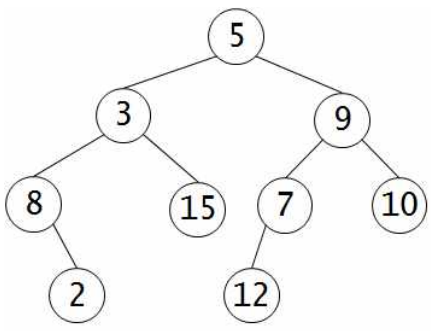
◦ 트리는 연결이진트리로 구현 (또는 링크 표현법 사용)하고, 각 노드에 저장되는 정보는 아래와 같다.

왼쪽 자식 링크	노드 번호	오른쪽 자식 링크

◦ 전위(선위) 순회 순서로 각 노드에 대한 정보가 주어지면, 트리를 루트부터 확장해 가는 방식으로 트리를 구성할 수 있다.

- 노드 번호는 양의 정수로 모두 다르고, 노드 번호에 특별한 순서는 없다.
- 각 노드의 정보는 3개의 정수, (x, y, z)로 표현되는 데, x는 해당 노드의 번호, y는 x의 왼쪽 자식 노드의 번호, z는 x의 오른쪽 자식 노드의 번호를 나타낸다.
  해당되는 자식이 없는 경우에는 0 이 주어진다.

예) 5 3 9 ➜ 5의 왼쪽 자식은 3, 오른쪽 자식은 9
3 8 15 ➜ 3의 왼쪽 자식은 8, 오른쪽 자식은 15
8 0 2 ➜ 8의 왼쪽 자식은 없고, 오른쪽 자식은 2
2 0 0 ➜ (이하 생략)
15 0 0
9 7 10
7 12 0
12 0 0
10 0 0

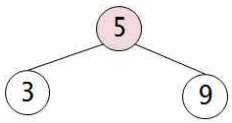


위 노드 정보에서, x에 해당하는 노드 번호를 차례로 쓰면, 전위(선위) 순회 결과가 된다.

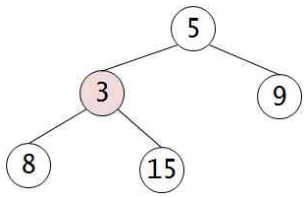
5 3 8 2 15 9 7 12 10

◦ 위 예에서 트리가 만들어 지는 과정

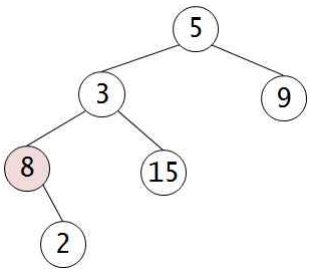
1) 첫 번째 노드 정보 (5 3 9)를 처리한 후의 트리 모양



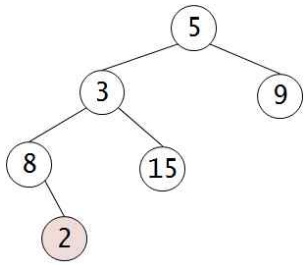
2) 두 번째 노드 정보 (3 8 15)까지 처리한 후의 트리 모양



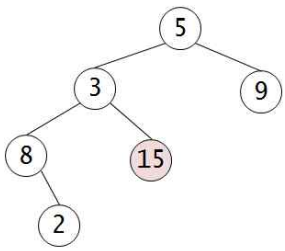
3) 세 번째 노드 정보 (8 0 2)까지 처리한 후의 트리 모양



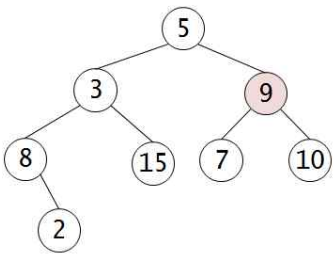
4) 네 번째 노드 정보 (2 0 0)까지 처리한 후의 트리 모양



5) 다섯 번째 노드 정보 (15 0 0)까지 처리한 후의 트리 모양



6) 여섯 번째 노드 정보 (9 7 10)까지 처리한 후의 트리 모양



(이 후 과정 생략)

2. 트리 탐색

◦ 트리 탐색은 루트(root) 노드에서 시작하여, 자식 링크를 따라 내려가면서 진행됨

- 탐색 도중 만나는 노드에서 어느 자식을 따라 내려가는 지 정보가 주어지면, 탐색 중 방문하는 노드의 번호들은 유일하게 결정됨.

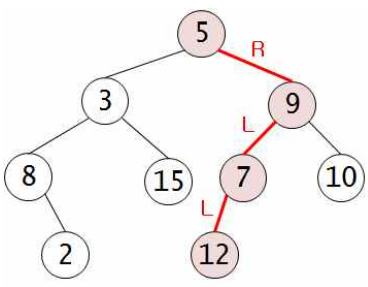
예) 탐색 정보가 아래와 같이 주어지면 (L은 왼쪽 자식, R은 오른쪽 자식을 의미),

RLL

탐색 중 방문하는 노드의 번호를 순서대로 적으면,

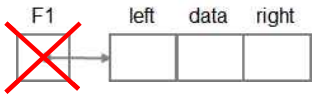
5 9 7 12

가 된다. (오른쪽 그림 참조)



[문제 2] 위에서 설명한 방식대로 트리 정보와 탐색 정보가 주어졌을 때, 트리를 생성하고 탐색 도중 방문하는 노드의 번호를 차례로 출력하는 프로그램을 작성하시오.

- 트리 1주차 실습에서처럼 모든 노드마다 자신의 위치를 가리키는 포인터변수를 만들어 사용하면 안 됨.



  - 오직 루트(root) 노드에 대해서만 허용. 즉, 트리는 루트 노드를 통해서만 접근 가능

입력 상세:

- 트리 정보
  - 첫 째 줄에 노드의 개수 n이 주어진다.
  - 다음 n개의 줄에, 전위(선위) 순회 순서로 노드의 정보가 주어진다. (위 설명 참조)
- 탐색 정보 (트리 정보가 모두 주어진 후)
  - 탐색 횟수 s가 주어진다.
  - 다음 s개의 줄에, 탐색 정보가 주어진다. (각 탐색은 루트 노드에서 새로 시작)
  - 하나의 탐색 정보는 공백없이, 'L'과 'R'로 구성된 문자열**(최대 길이 100)**로 주어진다.
  - 유효하지 않은 탐색 정보는 주어지지 않는다. 예를 들어, 위 트리에서 "RRR" 과 같은 탐색 정보는 유효하지 않다. 두 번 오른쪽 자식을 따라 내려가면 노드 10인데, 노드 10의 오른쪽 자식은 정의되지 않았다.

출력 상세:

- 탐색 시 방문하는 노드의 번호를 순서대로 출력한다. (하나의 줄에 한 번의 탐색 결과 출력)

입력 예시 1

```
9          ↦ 노드 개수
5 3 9
3 8 15
8 0 2
2 0 0
15 0 0
9 7 10
7 12 0
12 0 0
10 0 0
3          ↦ 탐색 횟수
RLL
LL
LR
```

출력 예시 1

```
□5 9 7 12      ↦ 첫 번째 탐색 결과
□5 3 8         ↦ 두 번째 탐색 결과
□5 3 15        ↦ 두 번째 탐색 결과
```