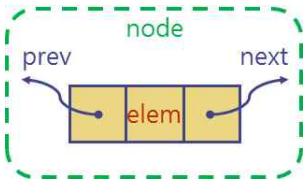
<알고리즘 실습> - 자료구조 복습

※ 입출력에 대한 안내

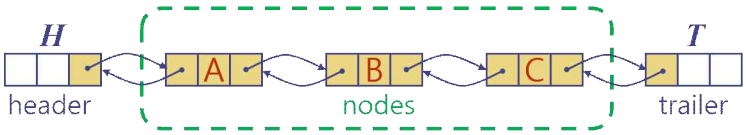
- 특별한 언급이 없으면 문제의 조건에 맞지 않는 입력은 입력되지 않는다고 가정하라.
- 특별한 언급이 없으면, 각 줄의 맨 앞과 맨 뒤에는 공백을 출력하지 않는다.
- 출력 예시에서 □는 각 줄의 맨 앞과 맨 뒤에 출력되는 공백을 의미한다.
- 입출력 예시에서 ↦ 이 후는 각 입력과 출력에 대한 설명이다.

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 순위 **r**에 원소 **e**를 삽입
   - **O(n)** 시간 소요

5. 이중연결리스트에서 삭제
   - 이중연결리스트로부터 지정된 순위 **r**의 노드를 삭제하고 원소를 반환
   - **O(n)** 시간 소요

※ 참고: 초기화, 순회, 삽입, 삭제에 관한 상세 알고리즘은 교재를 참고

[문제 1] 위에서 설명한 **이중연결리스트**를 이용하여 영문자 **리스트 ADT**를 구현하시오.

○ 다음 네 가지 연산을 지원해야 함 (순위는 **1**부터 시작한다고 가정)

- **add**(**list**, **r**, **e**) : **list**의 순위 **r**에 원소 **e**를 추가한다.
- **delete**(**list**, **r**) : **list**의 순위 **r**에 위치한 원소를 삭제한다. (주교재의 **remove**와 동일)
- **get**(**list**, **r**) : **list**의 순위 **r**에 위치한 원소를 반환한다.
- **print**(**list**) : **list**의 모든 원소를 저장 순위대로 공백없이 출력한다.

※ 순위 정보가 유효하지 않으면 화면에 에러 메시지 "invalid position"을 출력하고, 해당 연산을 무시한다.

○ 입력에 대한 설명 (아래 입출력 예시 참조)

- 각 연산의 내용이 한 줄에 한 개씩 입력되고, 한 개의 줄에는 연산의 종류, 순위, 원소 순서로 입력된다.
- **연산의 종류:** 연산 이름의 맨 앞 영문자가 대문자 **A**, **D**, **G**, **P**로 주어진다.
- **순위:** 양의 정수
- **원소:** 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       ↦ 연산 P에 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
invalid position

Say
y
```

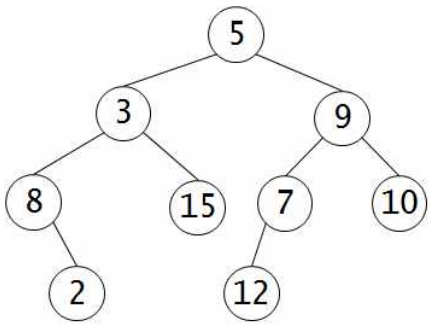
이진트리 만들기 및 탐색 (문제 2 참고 내용)

1. 트리 만들기 (구현)
   - 트리는 **연결이진트리**로 구현하고, 각 노드에 저장되는 정보는 아래와 같다.

왼쪽 자식 링크	노드 번호	오른쪽 자식 링크

   - 선위순회 순서로 각 노드에 대한 정보가 주어지면, 트리를 루트부터 확장해 가는 방식으로 트리를 구성할 수 있다.
     - 노드 번호는 유일한 양의 정수며, 노드 번호에 특별한 순서는 없다.
     - 각 노드에 대한 정보는 괄호에 싸인 세 개의 정수, 즉 (**x y z**)로 표현된다 - 여기서 **x**는 해당 노드의 번호, **y**는 **x**의 왼쪽 자식 노드의 번호, **z**는 **x**의 오른쪽 자식 노드의 번호를 나타낸다. 해당 자식이 없는 경우에는 번호 **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

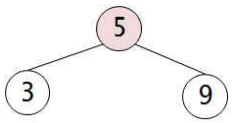


위 노드 정보에서, **x**에 해당하는 노드 번호를 차례로 쓰면, 선위순회 결과가 된다.

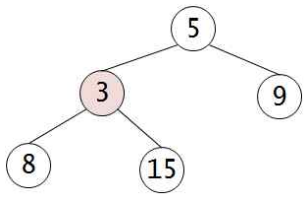
5 3 8 2 15 9 7 12 10

○ 위 예에서 트리가 만들어 지는 과정

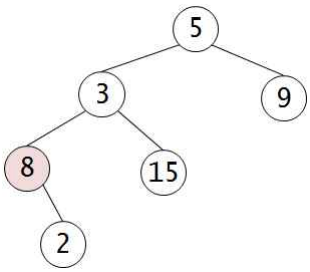
1) 첫 번째 노드 정보 (5 3 9)를 처리한 후의 트리 모양



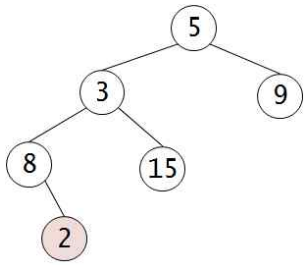
2) 두 번째 노드 정보 (3 8 15)까지 처리한 후의 트리 모양



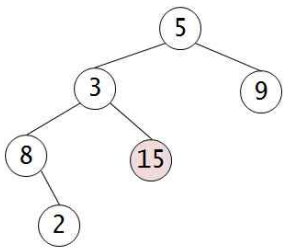
3) 세 번째 노드 정보 (8 0 2)까지 처리한 후의 트리 모양



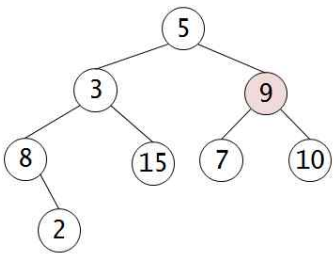
4) 네 번째 노드 정보 (2 0 0)까지 처리한 후의 트리 모양



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 루트에서 시작하여, 자식 링크를 따라 내려가면서 진행된다.

- 탐색 도중 만나는 노드에서 어느 자식을 따라 내려가는지 정보가 주어지면, 탐색 중 방문되는 노드 번호들이 유일하게 결정된다.

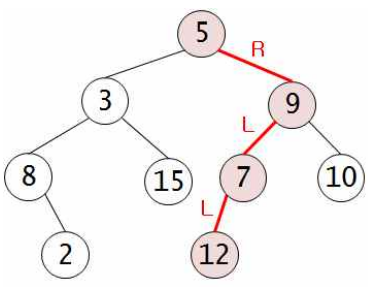
예) 탐색 정보가 아래와 같이 주어질 경우 (단, **L**은 왼쪽 자식, **R**은 오른쪽 자식을 의미),

RLL

탐색 중 방문하는 노드의 번호를 순서대로 적으면,

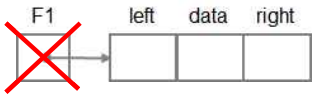
5 9 7 12

가 된다(오른쪽 그림 참조).



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

- <자료구조 및 실습>의 트리 1주차 실습에서처럼 모든 노드마다 자신의 위치를 가리키는 포인터 변수를 만들어 사용하면 안 됨.



  - 오직 루트에 대해서만 허용. 즉, 트리는 루트를 통해서만 접근 가능하다.

입력 상세:

- 트리 정보
  - 첫 째 줄에 노드의 개수 **n**이 주어진다.
  - 다음 **n**개의 줄에, 선위순회 순서로 노드의 정보가 주어진다(위 설명 참조).
- 탐색 정보 (트리 정보가 모두 주어진 후)
  - 탐색 횟수 **s**가 주어진다.
  - 다음 **s**개의 줄에 탐색 정보가 주어진다(각 탐색은 루트에서 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 **10**인데, 노드 **10**의 오른쪽 자식은 정의되지 않았다.

출력 상세:

- 탐색 시 방문하는 노드의 번호를 순서대로 출력한다(한 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
```