<알고리즘 실습> - 탐색트리

※ 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.

*** 이 문제에서 사용되는 알고리즘은 교재를 중심으로 기술되었음.**

[문제 1] 이진탐색트리 구현

주어진 조건을 만족하는 이진탐색트리를 구현하는 프로그램을 작성하라.

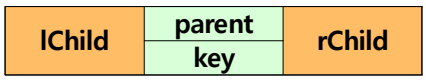
1) 삽입(**i**), 탐색(**s**), 삭제(**d**), 인쇄(**p**), 종료(**q**)의 순서대로 명령어를 반복되게 입력받는다.
   - **i** <키> : 키를 입력받아 노드 생성 및 트리에 삽입
   - **d** <키> : 입력받은 키 값이 트리에 존재하면 해당 노드를 삭제 후 해당 키 값을 출력하고 없을 경우 'X'를 출력
   - **s** <키> : 입력받은 키가 트리 내에 존재하는지 탐색하고, 해당 값이 존재할 경우 해당 키 값을 출력하고 없을 경우 'X'를 출력
   - **p** : 만들어진 트리를 전위순회로 인쇄
   - **q** : 프로그램 종료

주의:

1. 입력받는 키에는 중복이 없는 것으로 전제한다.
2. 문제를 단순화하기 위해, 키 값만 존재하고 원소 값(element)은 없는 것으로 구현한다.

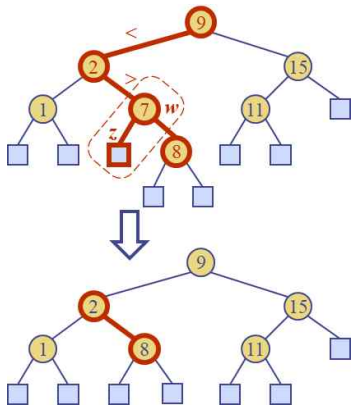
힌트:

1. 각 트리의 **노드**는 아래의 **구조체**를 이용하여 구현한다.



node

2. **전위순회**는 루트 노드를 방문하고, 왼쪽 부트리(subtree), 오른쪽 부트리 순서로 순회한다.
3. 트리 노드 **삭제** 시, 삭제할 노드(w)의 자식 중 하나라도 잎인 경우는 아래 그림처럼 간단히 그 노드를 삭제하고, 자식 노드가 그 노드를 계승한다(reduceExternal(z) 함수 사용).



삭제할 노드의 자식이 둘 다 내부 노드의 경우는 **중위순회 후계자**가 삭제한 노드 위치에 오도록 한다. 중위순회 후계자를 찾는 방법은 오른쪽 자식으로 이동한 후, 거기서부터 왼쪽 자식들만을 끝까지 따라 내려가서 도달하게 되는 외부노드를 찾는 것이다(아래 입력 예시2 참조).

입출력 형식:

1) main 함수는 아래 형식의 표준입력으로 트리를 입력받는다.

입력 : 문자(**i**, **d**, **s**, **p** ,**q**)와 정수형 키 값(**p**, **q** 제외)

2) main 함수는 아래 형식의 표준출력으로 각 명령에 응답한다.

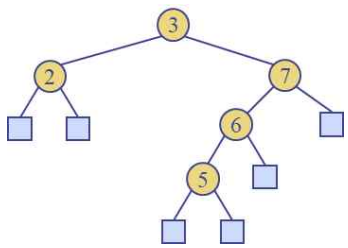
출력 : 키 값 또는 X (**d**, **s** 명령인 경우)
트리의 전위순회 인쇄 (**p** 명령인 경우)

입력 예시1

i 3	↦ 3 삽입
i 2	↦ 2 삽입
i 7	↦ 7 삽입
s 4	↦ 4 탐색
i 6	↦ 6 삽입
p	↦ 전위순회 인쇄
i 5	↦ 5 삽입
s 6	↦ 6 탐색
q	↦ 종료

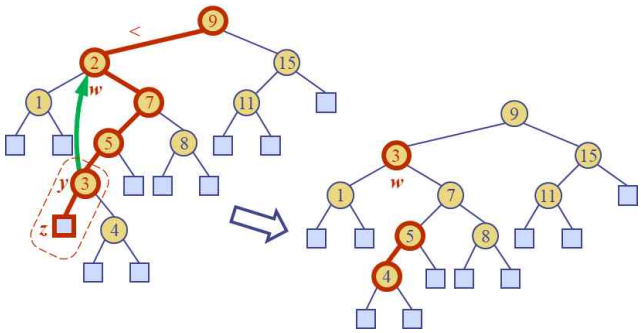
출력 예시1

X	↦ 4 탐색 결과
□3 2 7 6	↦ 전위순회 인쇄
6	↦ 6 탐색 결과



<입력 예시1>

입력 예시2		출력 예시2	
i 9	↦ 9 삽입	□9 2 1 7 5 3 4 8 15 11	↦ 전위순회 인쇄
i 2	↦ 2 삽입	2	↦ 2 삭제 결과
i 15	↦ 15 삽입	X	↦ 13 삭제 결과
i 1	↦ 1 삽입	□9 3 1 7 5 4 8 15 11	↦ 전위순회 인쇄
i 7	↦ 7 삽입		
i 11	↦ 11 삽입		
i 5	↦ 5 삽입		
i 8	↦ 8 삽입		
i 3	↦ 3 삽입		
i 4	↦ 4 삽입		
p	↦ 전위순회 인쇄		
d 2	↦ 2 삭제		
d 13	↦ 13 삭제		
p	↦ 전위순회 인쇄		
q	↦ 종료		



<입력 예시2>

주요 필요 함수:

- **main**() 함수
  - 인자: 없음
  - 반환값: 없음
  - 내용: 반복문, 조건문을 이용하여 계속되는 입력을 받음
- **findElement**() 함수
  - 인자: 탐색할 키 값
  - 반환값: 원소(이 실습문제에서는 키)
  - 내용: 키 값을 찾기 위해 트리 탐색
- **insertItem**() 함수
  - 인자: 삽입할 키 값
  - 반환값: 없음
  - 내용: 기존의 트리에 새 노드를 삽입
- **treeSearch**() 함수
  - 인자: 탐색할 키 값
  - 반환값: 탐색할 키 값을 갖는 내부 노드 반환 혹은 트리에 키가 없을 경우 키가 있어야 할 외부 노드를 반환
  - 내용: 트리에서 원하는 키 값을 찾았는지를 출력

- **removeElement**() 함수
  - 인자: 삭제할 키 값
  - 반환값: 삭제된 원소(이 실습문제에서는 키)
  - 내용: 기존의 트리에서 노드를 삭제한 후 원소를 반환
- **isExternal**() 함수
  - 인자: 노드
  - 반환값: 참 또는 거짓값
  - 내용: 노드가 외부노드인지 여부를 확인
- **inOrderSucc**() 함수
  - 인자: 내부노드
  - 반환값: 내부노드
  - 내용: 중위순회 후계자를 찾아냄

알고리즘 설계를 위한 의사코드 가이드:

```
Alg isExternal(w)
        input node w
        output boolean

1. if (w.left = ∅ and w.right = ∅)
        return true
   else {w.left ≠ ∅ or w.right ≠ ∅}
        return false
```

```
Alg sibling(w)
        input node w
        output sibling of w

1. if (w.parent.left = w)
        return w.parent.right
   else {w.parent.right = w}
        return w.parent.left
```

```
Alg inOrderSucc(w)
        input internal node w
        output inorder successor of w

1. w ← w.right

2. while (!isExternal(w.left))
        w ← w.left

3. return w
```

```
Alg reduceExternal(z)
        input external node z
        output the node replacing the parent node of the removed node z

1. w ← z.parent
2. zs ← sibling(z)

3. if (isRoot(w))
        root ← zs                                   {renew root}
        zs.parent ← ∅
   else
        g ← w.parent
        zs.parent ← g
        if (w = g.left)
                g.left ← zs
        else {w = g.right}
                g.right ← zs

4. putnode(z)                                       {deallocate node z}
5. putnode(w)                                       {deallocate node w}
6. return zs
```

* 수록되지 않은 함수의 의사코드는 교재의 코드를 참고할 수 있다. 하지만 가능하면 교재를 참조하지 않고 스스로 작성해본다.

[문제 2] AVL 트리 생성

주어진 조건을 만족하는 AVL 트리를 구현하는 프로그램을 작성하라.

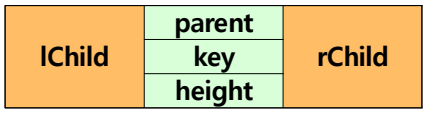
1) 기본적인 입출력구조는 문제 1과 동일하나 **삭제를 제외**한 삽입, 탐색, 출력을 구현한다.
2) main을 통해 명령어와 키를 입력받는다.
3) 입력받은 명령에 따라 AVL 트리를 생성한다.

주의:

1. 문제 1과 동일

힌트:

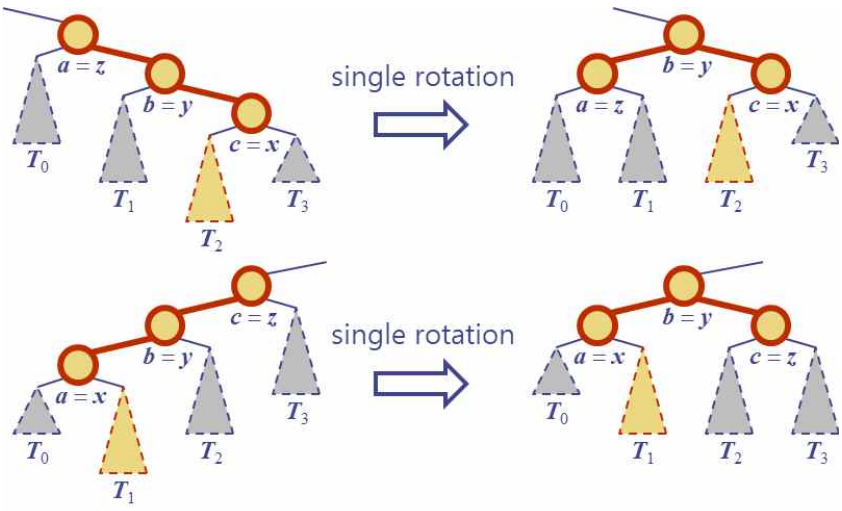
1. 노드 구조체에 **높이**(height)를 추가한다.



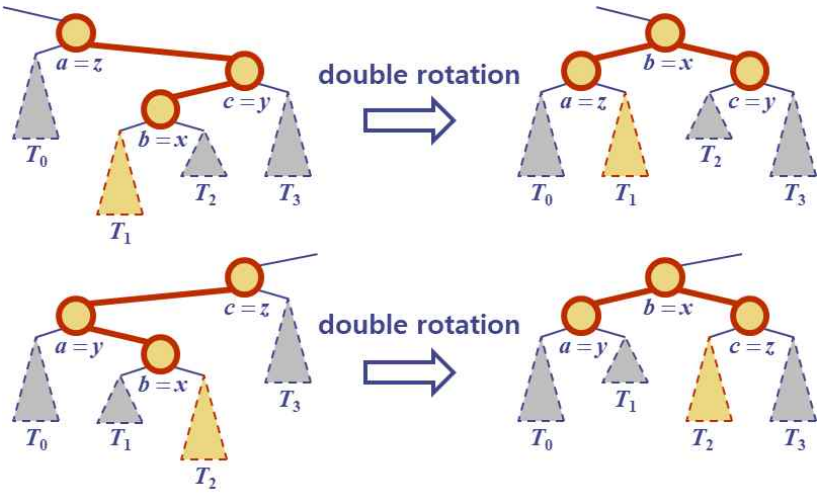
node

2. 문제 1의 이진탐색트리에서 구현한 함수를 그대로 또는 수정해서 사용하고 더 필요한 함수를 추가한다.
3. 노드 삽입 후 트리에 불균형이 발생했을 경우, **개조**(restructure)를 수행한다. 개조는 종종 **회전**(rotation)이라고도 불리며, 좌우대칭을 포함하여 모두 4종류의 유형이 존재한다.

- **단일회전**(single rotation)



- **이중회전**(double rotation)



입출력 형식:

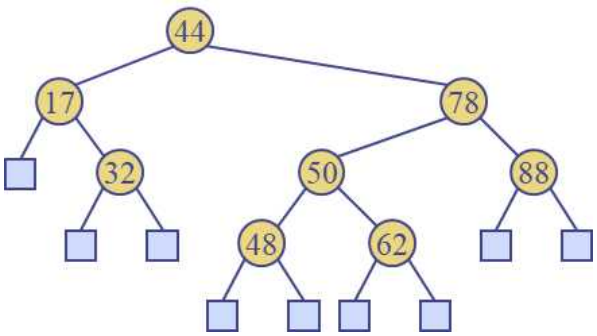
1) 문제 1과 동일(삭제만 제외)

입력 예시1

```
i 44          ↦ 44 삽입
i 17          ↦ 17 삽입
i 78          ↦ 78 삽입
i 32          ↦ 32 삽입
i 50          ↦ 50 삽입
i 88          ↦ 88 탐색
i 48          ↦ 48 삽입
i 62          ↦ 62 삽입
s 88          ↦ 88 탐색
p             ↦ 전위순회 인쇄
q             ↦ 프로그램 종료
```

출력 예시1

```
88                          ↦ 88 탐색
□44 17 32 78 50 48 62 88    ↦ 전위순회 인쇄
```



<입력 예시1>

입력 예시2

```
i 9           ↦ 9 삽입
i 31          ↦ 31 삽입
i 66          ↦ 66 삽입
i 30          ↦ 30 삽입
```

출력 예시2

```
30                          ↦ 30 탐색 결과
□30 9 1 24 31 66            ↦ 전위순회 인쇄
X                           ↦ 47 탐색 결과
```

i 1	↦ 1 삽입	
s 30	↦ 30 탐색	
i 24	↦ 24 삽입	
p	↦ 전위순회 인쇄	
s 47	↦ 47 탐색	
i 61	↦ 61 삽입	
i 13	↦ 13 삽입	
q	↦ 프로그램 종료	

필요 함수:

- **insertItem**() 함수
  - 인자: 삽입할 키 값
  - 반환값: 없음
  - 내용: 기존의 트리에 새 노드를 삽입하고, **searchAndFixAfterInsertion**() 함수를 이용하여 균형검사 및 수리를 수행
- **searchAndFixAfterInsertion**() 함수
  - 인자: 내부노드
  - 반환값: 없음
  - 내용: 균형검사를 수행하고 불균형이 있으면 개조를 통해 높이균형 속성을 회복
- **updateHeight**() 함수
  - 인자: 내부노드
  - 반환값: 참 또는 거짓 값
  - 내용: 노드의 높이를 갱신한 후 갱신 여부를 반환
- **isBalanced**() 함수
  - 인자: 내부노드
  - 반환값: 참 또는 거짓 값
  - 내용: 노드의 좌우자식 높이균형 여부 반환
- **restructure**() 함수
  - 인자: 내부노드 x, y, x
  - 반환값: 참 또는 거짓 값
  - 내용: 3-노드 개조를 수행한 후 (갱신된) 3-노드의 루트를 반환

알고리즘 설계를 위한 의사코드 가이드:

```
Alg expandExternal(w)
        input external node w
        output none

1. l ← getnode()
2. r ← getnode()

3. l.left = ∅
4. l.right = ∅
5. l.parent = w
6. l.height = 0

7. r.left = ∅
8. r.right = ∅
9. r.parent = w
10. r.height = 0

11. w.left = l
12. w.right = r
13. w.height = 1
14. return
```

```
Alg insertItem(k, e)
        input AVL tree T, key k, element e
        output none

1. w ← treeSearch(root(), k)                    {삽입 키 찾기}

2. if (isInternal(w))                           {이미 존재하면 반환}
        return
   else
        Set node w to k
        expandExternal(w)
        searchAndRepairAfterInsertion(w)
        return
```

다음 알고리즘은 교재의 연습문제 11-5의 답을 참고할 수 있다.

Alg searchAndFixAfterInsertion

Alg restructure

Alg updateHeight

Alg isBalanced

[문제 3] AVL 트리 구현(삭제 포함)

문제 2에서 만들어진 AVL 트리에서 **삭제**를 구현하라.

1) 기본적인 입출력 구조는 문제 2와 동일하며 삭제를 추가하여 구현한다.
2) main을 통해 명령어와 키를 입력받는다.
3) 입력받은 명령에 따라 AVL 트리를 생성한다.

주의:

1. 문제 1과 동일

힌트:

1. 문제 2에서 만들어진 함수들을 그대로 이용하고, **삭제 관련 함수**를 추가로 구현한다.
2. AVL 트리에서 삭제는 이진탐색트리에서와 동일하게 수행되지만, 마지막 단계에서 reduceExternal 작업에 의해 삭제된 노드의 부모 노드(그리고 조상 노드들)가 불균형이 될 수 있음에 유의한다.
3. 트리의 불균형을 해소하기 위한 개조(restructure)는 문제 2에서 작성한 함수를 그대로 사용할 수 있다.

입출력 형식:

1) 문제 1과 동일

입력 예시1

```
i 9                 ↦ 9 삽입
i 31                ↦ 31 삽입
i 66                ↦ 66 삽입
i 30                ↦ 30 삽입
i 1                 ↦ 1 삽입
s 30                ↦ 30 탐색
i 24                ↦ 24 삽입
p                   ↦ 전위순회 인쇄
s 47                ↦ 47 탐색
i 61                ↦ 61 삽입
d 30                ↦ 30 삭제
i 13                ↦ 13 삽입
q                   ↦ 프로그램 종료
```

출력 예시1

```
30                  ↦ 30 탐색 결과
□30 9 1 24 31 66    ↦ 전위순회 인쇄
X                   ↦ 47 탐색 결과
30                  ↦ 30 삭제
```

필요 함수:

- **removeElement**() 함수
  - 인자: 삭제할 키 값
  - 반환값: 삭제된 원소(이 문제에서는 키)
  - 내용: 기존의 트리에서 노드를 삭제한 후 **searchAndFixAfterRemoval**() 함수를 이용하여 균형검사 및 수리를 수행

알고리즘 설계를 위한 의사코드 가이드:

```
Alg removeElement(k)
        input AVL tree T, key k
        output key

1. w ← treeSearch(root(), k)                    {삭제 노드 찾기}

2. if (isExternal(w))                           {삭제 노드 없으면 반환}
        return NoSuchKey

3. z ← leftChild(w)

4. if (!isExternal(z))
        z ← rightChild(w)

5. if (isExternal(z))                           {case 1}
        zs ← reduceExternal(z)
   else                                         {case 2}
        y ← inOrderSucc(w)
        z ← leftChild(y)
        Set node w to key(y)
        zs ← reduceExternal(z)

6. searchAndRepairAfterRemoval(parent(zs))
7. return k
```

다음 알고리즘은 교재의 연습문제 11-5의 답을 참고할 것.

Alg searchAndFixAfterRemoval