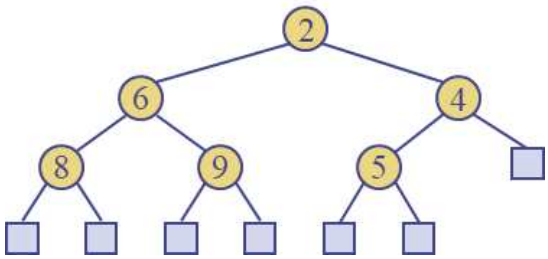
※ T를 n개의 키를 저장하며 루트가 v인 최소 힙이라 하자. 주어진 검색 키 x보다 작은 T의 키를 모두 보고하는 효율적인 알고리즘 findSmallerKeys(v, x)를 의사코드로 작성하라. 여기서 x는 T에 존재하지 않을 수도 있다. 예를 들어 아래 주어진 힙과 검색 키 x = 7에 대해 알고리즘은 2, 4, 5, 6을 보고한다. 이때 키들은 정렬 순서로 보고되지 않아도 좋다. leftChild(), rightChild(), key(), isInternal(), isExternal()등의 Heap ADT 함수들을 이용하여 작성한다.



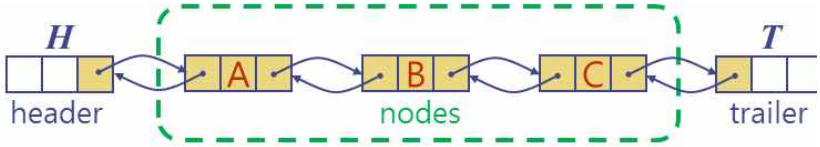
Alg ***findSmallerKeys(v, x)***
input heap with root ***v***, key ***x***
output keys in the heap that are smaller than ***x***

※ 아래의 설명과 코드 템플릿을 이용하여 문제를 해결합니다.

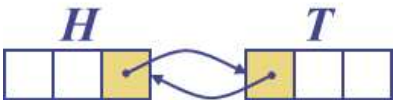
1. 사용자로부터 정수 원소의 개수(10개로 통일합시다)를 입력받아 11 ~ 40까지 임의의 수를 발생하여 리스트를 생성한다. 노드는 addLast() 함수를 이용하여 생성된 순서대로 리스트의 마지막에 삽입되어지도록 한다.

2. 이 때 리스트는 이중연결리스트로 구현한다. 헤더노드와 트레일러노드를 구성하여 리스트의 처음과 마지막 노드를 연결한다. 노드와 리스트의 기본 구조는 제공되는 코드 템플릿을 사용한다.

(a) 노드의 구조



(b) 이중연결리스트



(c) 리스트 초기화

3. 위에서 생성된 리스트를 집합으로 변환한다. 즉, 리스트로부터 원소들의 중복을 삭제해야 한다. 집합의 정의 상 원소들 간의 순서는 없지만 여러 가지 작업의 효율적인 구현을 위해 집합 원소들을 정렬된 리스트로 표현한다. 따라서 생성된 리스트에서 중복을 제거하며 정렬된 리스트로 변환하는 함수를 만든다. 이 때 정렬은 삽입정렬과 선택정렬을 이용하며 제자리 정렬이어야 한다(별도의 리스트를 만들면 안 된다는 의미임). 따라서 두 개의 함수를 만들면 된다.

4. 위의 과정에서 집합을 만들었으므로 두 개의 리스트 집합을 대상으로 합집합 연산과 차집합 연산을 수행하는 함수를 만들어보자.

5. 테스트 예시

```
원소의 개수 입력 : 10

리스트 A : [17] [37] [38] [13] [14] [25] [31] [15] [14] [39] <= 리스트 2개 생성
리스트 B : [17] [13] [15] [17] [32] [15] [32] [15] [20] [23]

선택정렬 - 집합 A : [13] [14] [15] [17] [25] [31] [37] [38] [39] <= 선택정렬로 집합 생성
삽입정렬 - 집합 B : [13] [15] [17] [20] [23] [32] <= 삽입정렬로 집합 생성

A와 B의 합집합 : [13] [14] [15] [17] [20] [23] [25] [31] [32] [37] [38] [39]
A와 B의 차집합 : [14] [25] [31] [37] [38] [39]
```

6. 다음의 코드를 이용하여 문제 해결을 시작한다.

```
typedef struct DListNode
{
        int elem;
        struct DListNode* prev, * next;
}DListNode;

typedef struct
{
        DListNode* H;
        DListNode* T;
}SetType;

void initNode(DListNode* H, DListNode* T)
{
        H->next = T;
        T->prev = H;
}

.....

int main()
{
        SetType* s1 = (SetType*)malloc(sizeof(SetType));
        initSet(s1);
        SetType* s2 = (SetType*)malloc(sizeof(SetType));
        initSet(s2);

.....

}
```