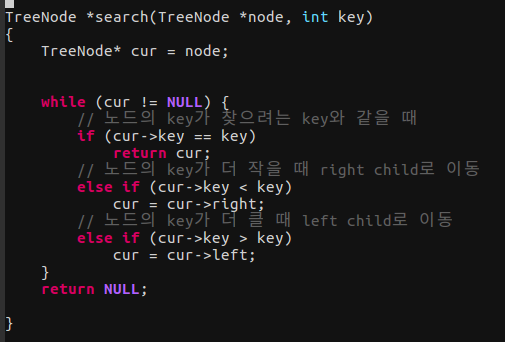
HW5: Binary Search Tree

2012003567

이상화

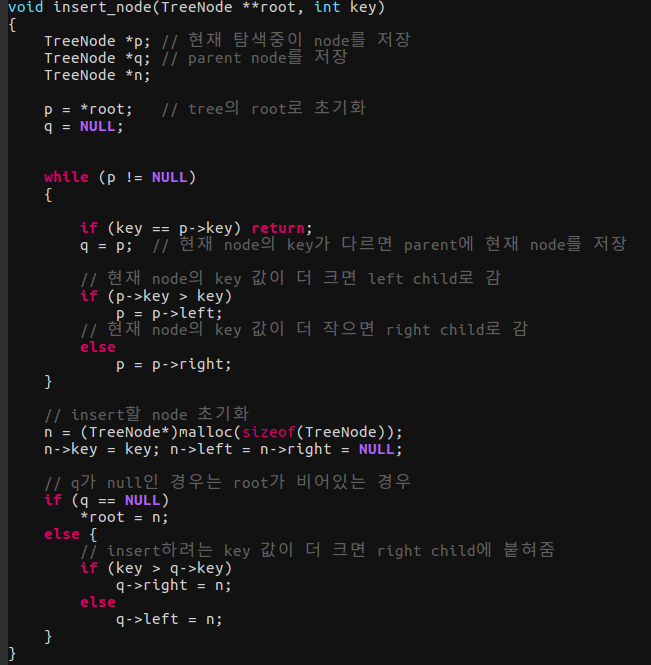
**1. 문제 해결의 주요 코드**

(1) Search()



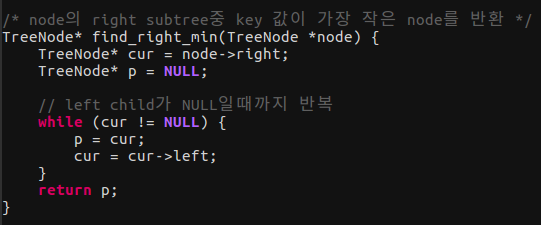
Key 값을 가진 node를 찾아서 반환하는 함수이다. Node cur을 root로 초기화 한 뒤 찾으려는 key가 node의 key보다 크면 right child, 작으면 left child로 이동하며 일치하는 key값이 있는 node를 찾으면 그 노드의 주소값을 반환한다.

(2) insert\_node()



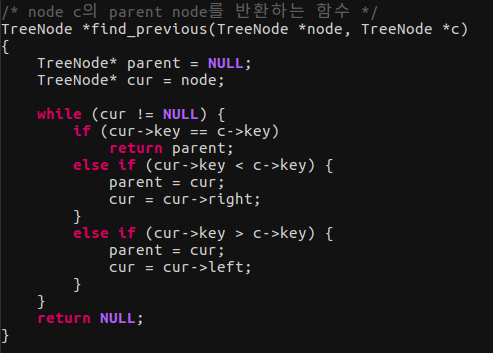
Insert할 위치를 찾는 데에 TreeNode \*p,\*q를 이용한다. p는 현재 방문한 node이고 q는 다음 node로 넘어가기전에 현재 node를 저장해주는 포인터이다. search와 같은 방식으로 insert할 위치를 찾는다. 탐색을 진행하다 p가 null이 되면 그 위치에 삽입해야 하므로 p의 parent q의 key값과 삽입하려는 key 값을 비교해서 q의 left,right중 어느곳에 삽입할지를 찾는다.

(3) find\_right\_min()



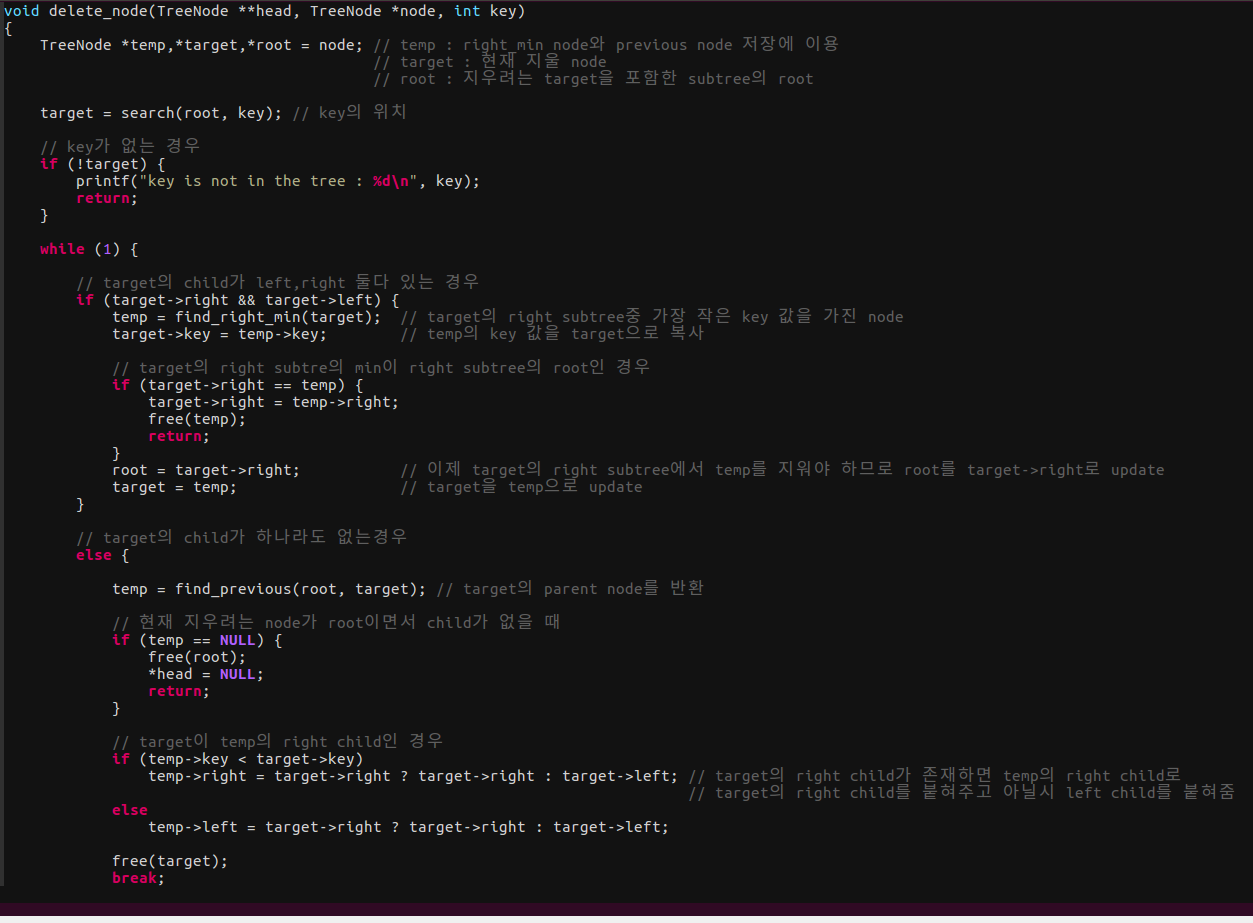
delete 함수에서 이용하기 위해 node의 right subtree중 key값이 가장 작은 node를 반환하는 함수를 만들었다. right child로 이동한 뒤에 left가 null일때까지 반복한다.

(4) find\_previous()



어떤 node c의 parent node를 찾아서 반환해주는 함수이다. 역시 delete 함수에서 이용하기 위해 만들었다. search()와 탐색법이 같지만 parent 포인터를 만들어 이것을 반환한다.

(5) delete\_node()



delete를 하기 위해서는 먼저 key를 가지고 있는 node를 찾아야 한다. 그 node를 target에 저장하였다. target이 null인 경우는 key를 가진 node가 없는 것이므로 메시지를 출력하고 return 한다. 삭제시 target의 child들의 개수에 따라 case가 갈라진다 target의 child가 둘다 있을 경우에는 target의 right subtree의 min 값을 target->key에 복사 해준다. 그 뒤 root를 target->right로 update 해주고 그 다음 target을 temp로 해줌으로써 다음 지울 대상을 update해준다. 중간에 if(target->right == temp)문은 target의 right subtree의 min node가 right subtree의 root인 경우를 따로 처리해주는 if문이다. 다른 케이스는 target의 child가 둘중 하나라도 없는 경우이다. 이때 첫번째 if문으로 tree에 node가 한 개밖에 없는 경우를 처리해준다. 전체 tree의 root를 가리키는 head를 인자로 받아서 root를 null로 만들어준 뒤에 return 한다.parent node인 temp의 key가 target의 key보다 큰 경우에 target의 right child가 존재하면 temp의 right child로 target의 right child를 붙혀주고 아닐시에 left child를 붙혀준다. temp의 key가 target의 key보다 큰 경우에는 temp의 left child로 붙혀주면 된다.

**1. 실행결과**

