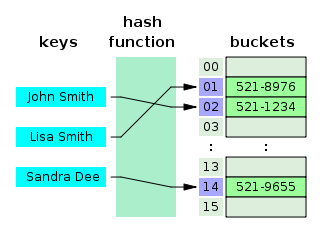
해시 테이블



-key를 input으로 얻은 hash 값을 index로 사용

-O(1)이라는 뛰어난 시간복잡도를 지니나 해시 충돌 문제가 발생

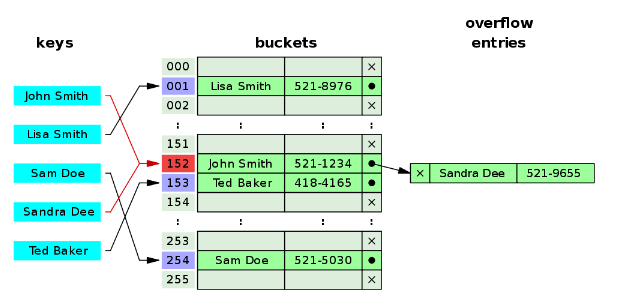
해시 충돌

-2개 이상의 서로 다른 key가 같은 hash값을 지니는 현상

-bucket의 size가 key 전체 size보다 작으므로 필연적으로 발생

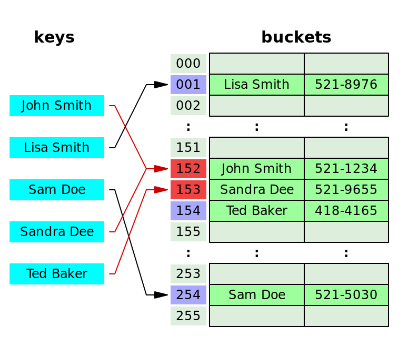
-2가지 해결법, Chaining & Open Addressing

1)Chaining



-한 버켓에 여러 key를 연결리스트로 연결함

2)open addressing



다른 주소에 키를 저장함

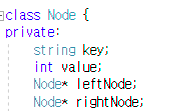


각 방식마다 장단점이 존재하는데, 이번의 경우 key의 개수가 bucket보다 많을 수 있으므로chaining 방식을 사용하여야 한다.

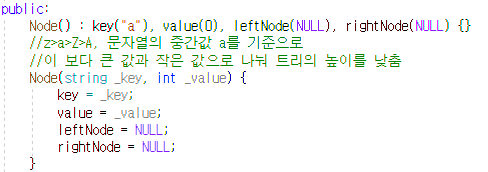
chaining 방식의 문제점

-> 최악의 경우 = 모든 key의 해시 값이 동일한 경우, O(n)이라는 시간 복잡도를 지니게 됨

-> 이를 완화하기 위해, overflow entry를 이진 검색 트리로 구현하였음



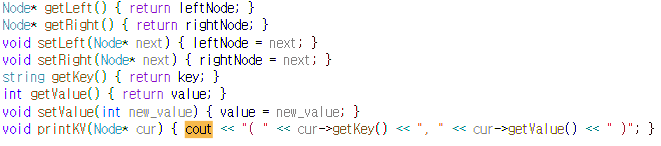
각 node는 key와 value, 그리고 LeftNode, RightNode로 구성됨



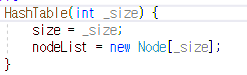
key는 string이며, string 비교 연산시 z>a>Z>A로 계산되므로

전체 문자열의 중간 정도에 위치한 “a” 문자열을 디폴트 값으로 사용,

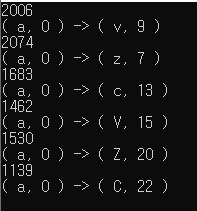
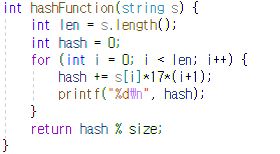
효율적으로 트리의 높이를 낮춤



node의 값을 return하거나 수정, 출력하는 메서드

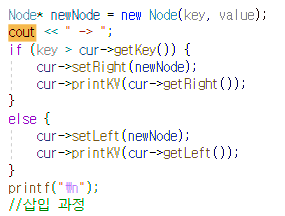
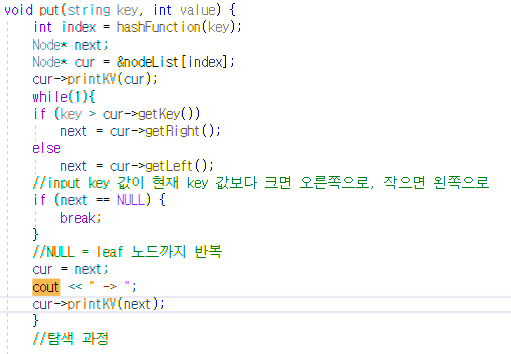


HashTable 생성자, 입력받은 size로 bucket 생성



해시 함수는 digit folding방식을 사용했고,

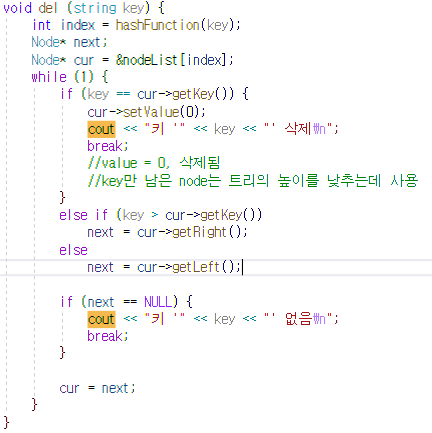
key 문자열이 짧더라도 큰 값이 나올 수 있게 추가적인 값을 곱했다.

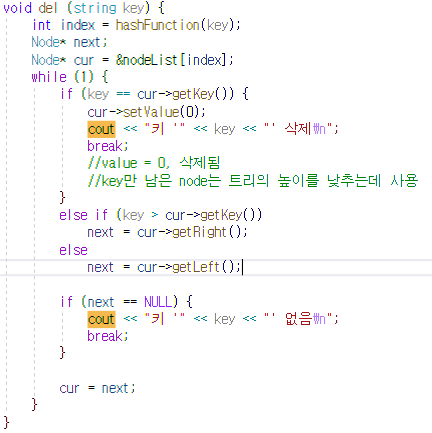


put, 이진 탐색 후 leaf 노드에 도달하면 key를 삽입



get, 이진 탐색 후 input과 동일한 키 값이 있으면 출력





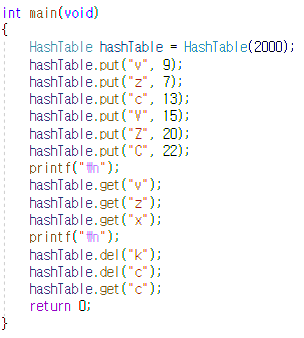
del, 이진 탐색 후 input과 동일한 key 값이 존재하면 value를 0으로 변경

node를 삭제하고 왼쪽 서브트리를 RMIN의 leftnode로 연결하면 되지만,

이 경우 tree의 높이가 비약적으로 높아지므로

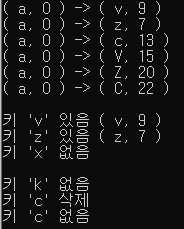
단순히 value만 0으로 만들고 냅두는게 탐색에 용이하다 생각됨

테스트



메인 함수에 다음과 같은 key value 쌍을 삽입,

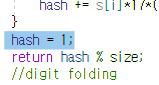
테이블에 존재하는 key와 존재하지 않는 key에 대해 get, del 수행

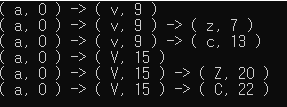


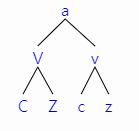
\*(a,0)은 디폴트 root 노드

input 시도가 적은 만큼 해시 충돌이 발생하지 않음.

삽입과 삭제가 정상적으로 수행되는 것을 확인







고의적으로 해시 충돌을 일으키고 key가 이상적으로 삽입 된 경우.

단순 연결 리스트일때보다 평균 시간 복잡도를 낮출 수 있음

이진 탐색 트리는 편향될 수 있으므로 B-tree를 사용하면 더 좋았겠으나,

이번의 메인 과제는 트리가 아닌 해시 테이블이므로

간단히 이진 탐색트리로 구현함