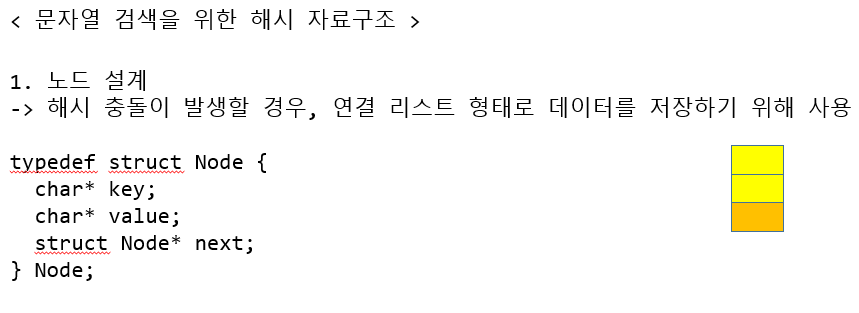
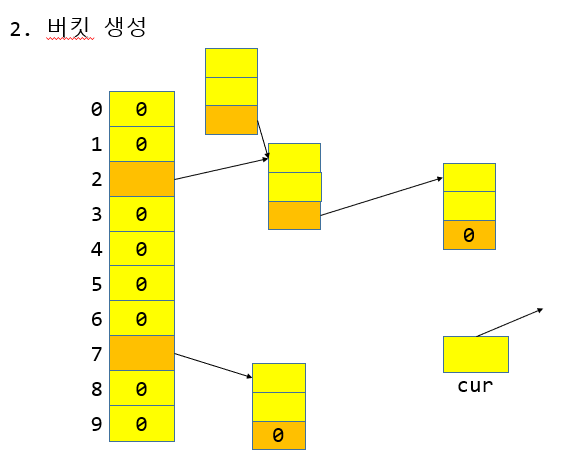
해시맵

목적: 데이터 검색을 위한 목적(데이터의 중복이 발생하면 안됌)



🡺b.c



버킷만들기

node\*\* buckets -> 원래는 node\*였는데 동적배열로 만들기위해 \* 하나 더 붙여줌

🡺C.C

PUT()함수

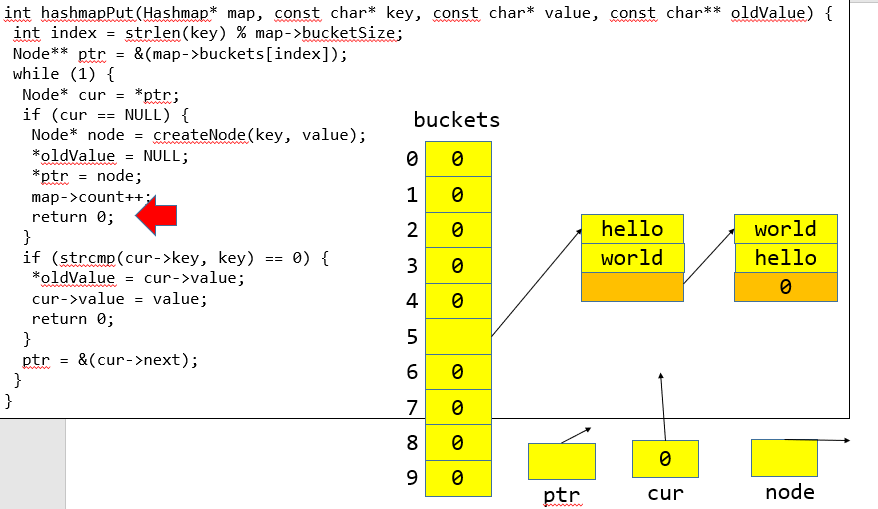
해당하는 키가 존재하는지 아닌지의 여부(충돌이 발생한 경우)

🡺D.C

출력하는 함수

Day0603

🡺0.c

Hashmap()함수에는 조건별로 이원화 되어있으므로 즉 이원화 되어있으므로 따로 겹치는부분을createNode()라는 함수로 빼자

이중포인터를 사용해서 전체를 가르키는 것이 아닌 next를 가르키도록 해보자

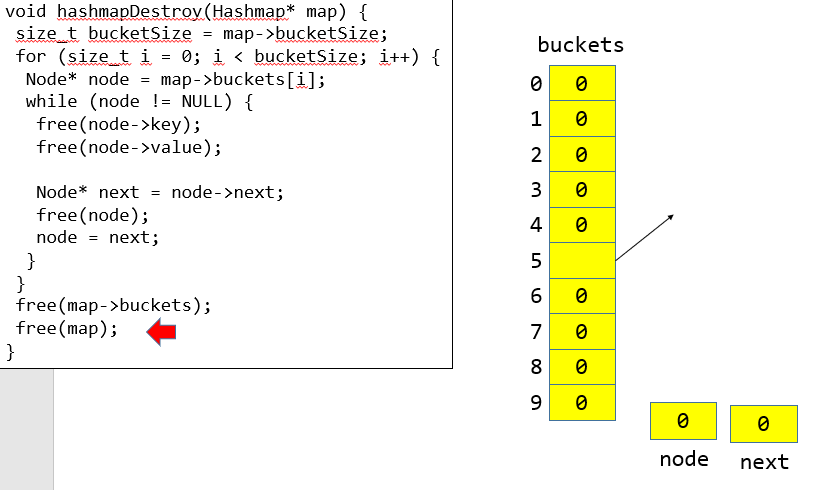
🡺1.c

만약 key를 지역변수에 대한 값을 참조하게되면 (얕은복사가 되므로) 댕글링 포인터가 되게 된다. foo()함수

->즉, 함수 호출이 끝나면 해당 포인터는 댕글링 포인터가 되므로 이를 해결하기 위해 깊은복사를 수행해야한다.

key, value를 얕은복사하는 부분을 전부 깊은복사로 바꾸자

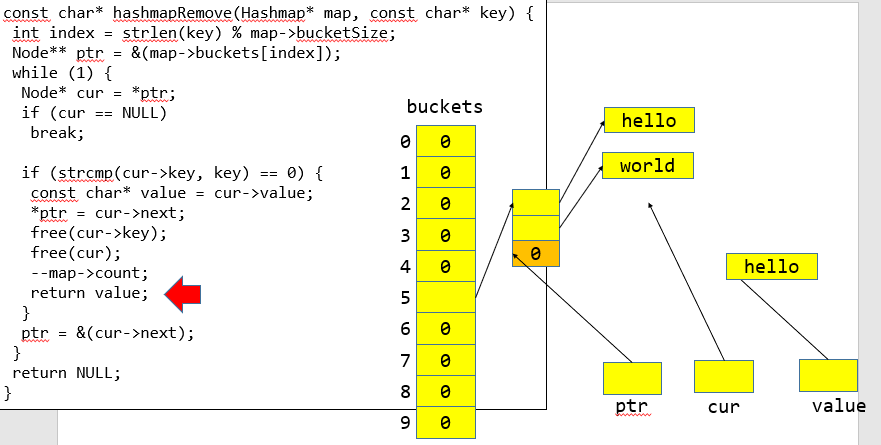
🡺2.c

destory는 얕은복사용이었으므로 이도 깊은복사 해제로 바꿔준다.

🡺3.c

hashmapGet()함수

🡺4.c

hashmapRemove()함수

ptr을 한번 역참조하게되면 bucket의 그 다음 노드가 나오게 된다.

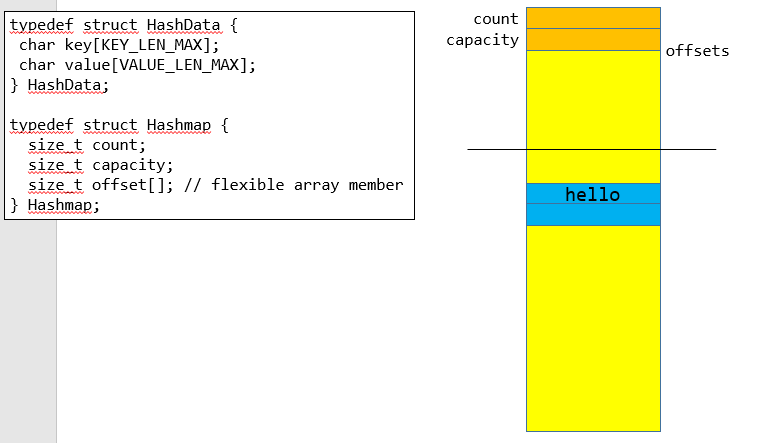
즉 cur은 바로 첫번째 노드를 가르킨다.

🡺5.c

1. createNode()시에 동적할당 -> memory pool 사용하면 해결됌.

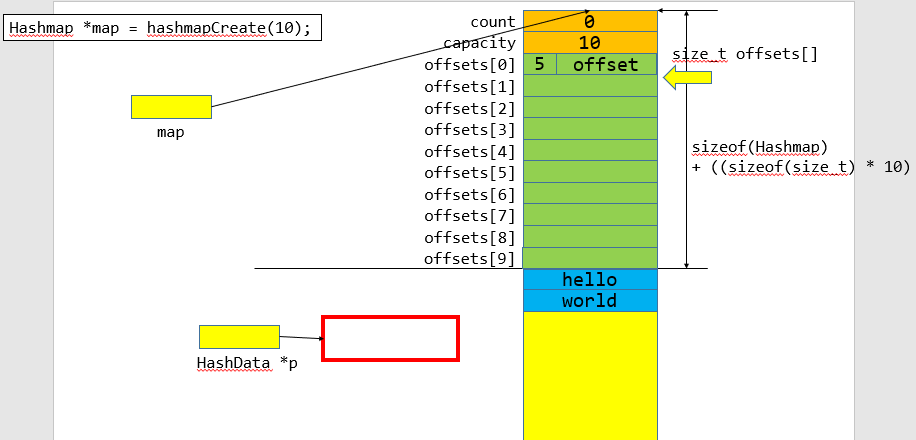
->즉 필요할때마다 할당하지말고, 풀에 필요한만큼 쌓아두고 필요시에 가져다쓰자.

2. 페이지에 대한 메모리낭비

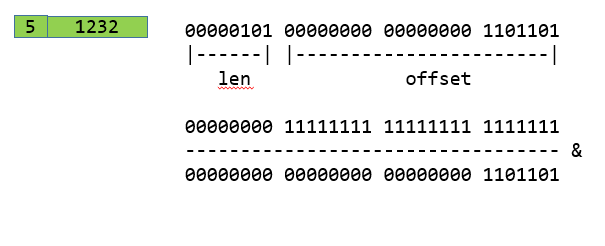
offset 은 얼마나 떨어지있느지에 대한 위치를 알기위해.

🡪생성과 제거에 대한 코드

🡺6.c

string compare는 굉장히 비싸므로 애초에 compare하기 전에 문자열의 길이를 먼저 비교한 다음에 같으면 compare하면 더 효울적일것이다. 그러므로 offset에 문자열의 길이를 미리 넣어놓자

cf.



🡺7.c

hashmapFind함수 구현

🡺8.c

hashmapGet()함수 구현

🡺9.c

삭제 함수 구현

->지우려고 하는 offset의 index를 우선적으로 구하자-> getIndex 함수부터 작성

🡺ppt12

메모리풀은 동적할당을 수시로 할필요는 없지만 활용성이 좀 떨어져서 0604부터는 다시 리스트 형태로 hashmap 만듦.

Day0604

🡺1.c

‘hashmapCreate()’와 ‘destroy()일부’

create에서 count를 굳이 설정안해줘도 되는 이유는 어차피 동적할당할떄 calloc으로 해줬기 때문에

🡺2.c