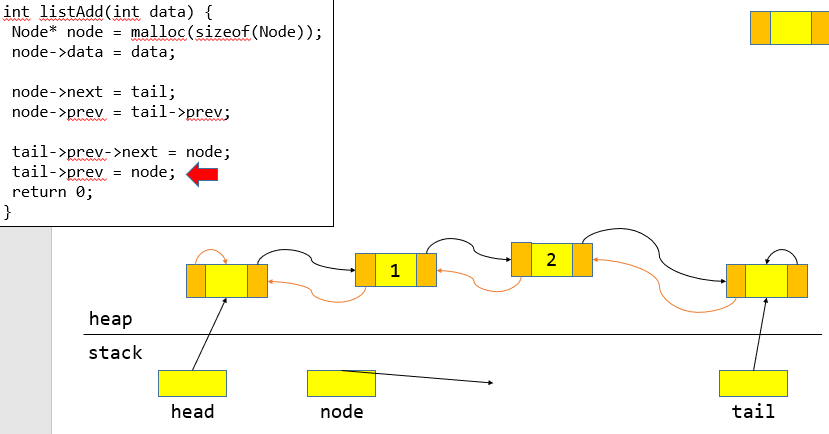
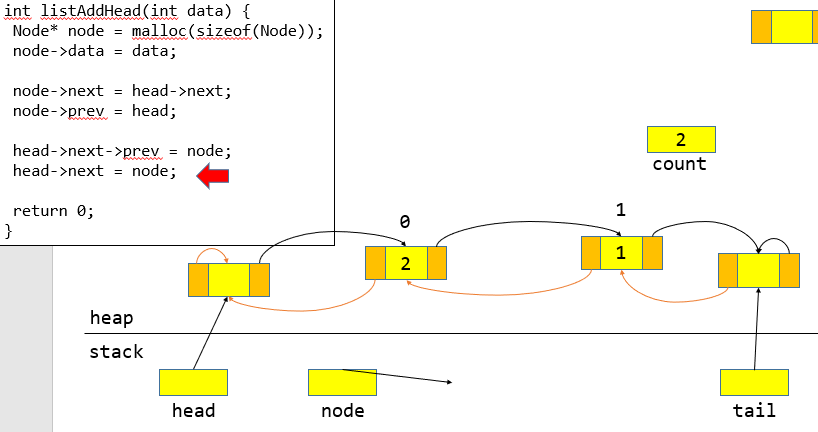


🡺2.c



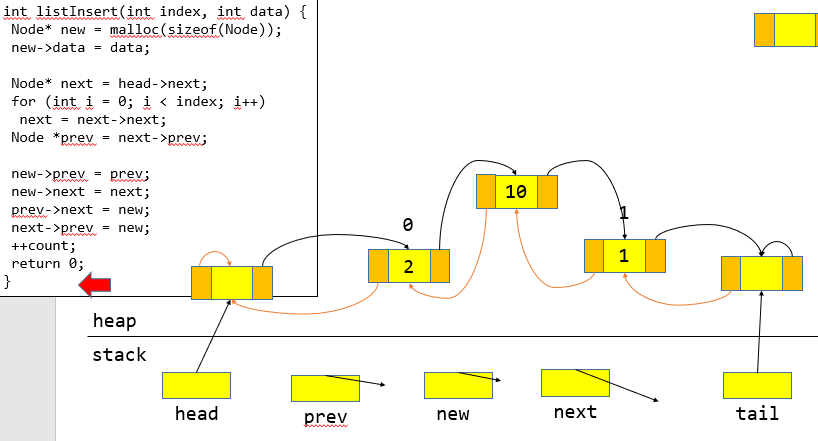
listAdd() 함수 (마지막에 노드 추가하는함수)

🡺3.c



listAddHea 함수(헤드 바로 다음에 노드 추가하는 함수)

🡺4.c



listInsert() 함수 (노드들 중간에 삽입하는 함수)

🡺5.c

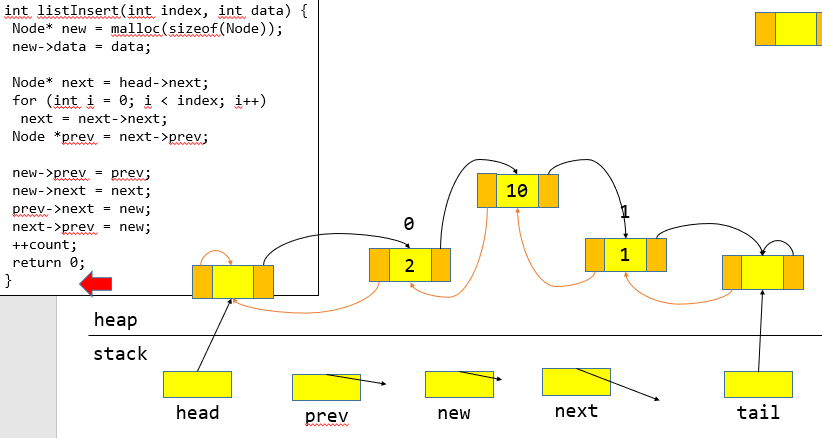
listGet() 함수 (해당 index의 값 찾는 함수)

그러나 뒤쪽에 있는 노드를 찾기 위해서는 비효율적이다. 그래서 /2를 하여 index의 위치가 가운데 기준 앞인지 뒤인지 찾으면 시간이 단축될것이다.

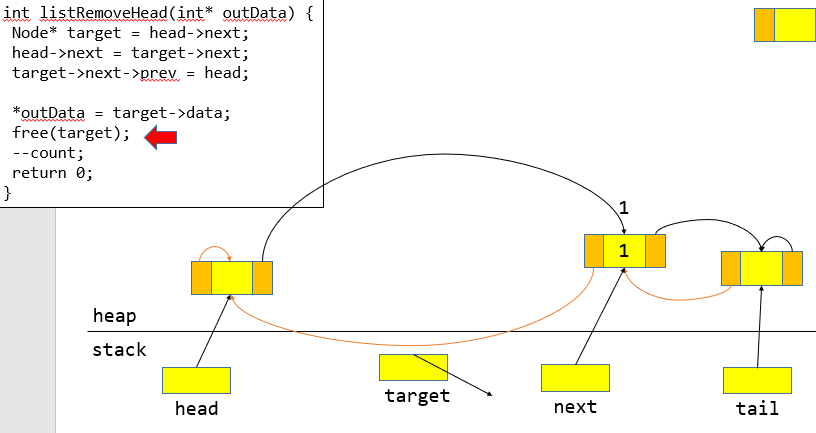
(기본 listGet()함수는 주석처리)

그러나 count/2 는 다른 연산에 비해 비용이 굉장히 많이든다. 그래서 오른쪽으로 한칸 **shift**하면 나누기2 왼쪽으로 **shift**하면 곱하기 2 이므로 이것을 이용하자.

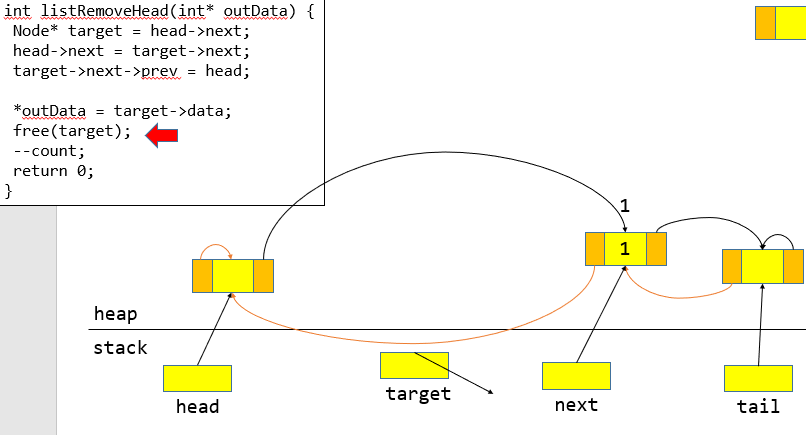
🡺6.c



listRemoveTail()부분



listRemoveHead()부분



listRemove()부분

listRemoveTail() -> 더미테일 이전 노드 제거

listRemoveHead() -> 더미헤드 다음 노드 제거

listRemove(int index) -> 임의의 위치 제거

🡺7.c

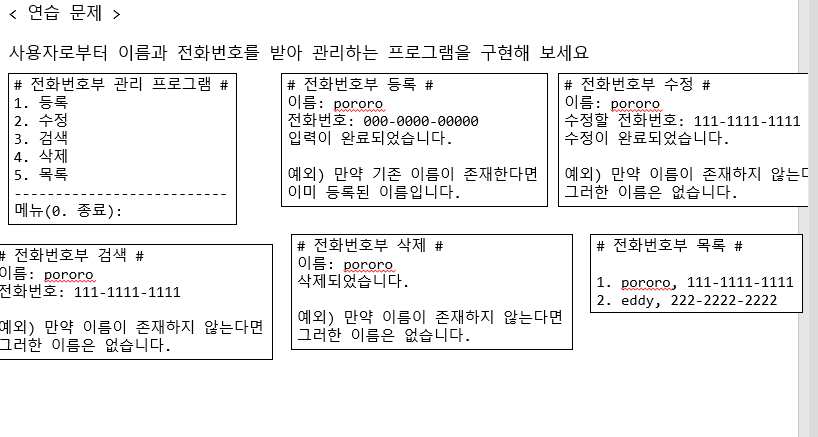
이전 자료구조는 노드 탐색을 위한 코드가 중복되어있고 리스트 전체에 걸쳐 탐색을 수행하므로 성능 상의 이슈가 존재한다. 이를 해결해보자

내부에서만 쓰이는 함수이므로 이럴경우엔 꼭 static을 붙이자! 또한 정의를 쓰지않았으므로 위쪽에 함수를 구현하자.

🡺8.c

이전 코드들은

1. head/tail/count가 전역변수이다. 그래서 이를 추상화해보자.
2. 해제하는 코드가 없다.
3. int 타입만을 저장가능한데 이를 모든 타입이 들어갈수 있도록 바꾸자.



🡺9.c

listAdd에서는 설계자가 동적할당을 강요한다는 단점이있다.(즉, 할당은 사용자에게 권한을 주면됌) 그래서 노드를 스택에 만들고 그 노드의 시작주소만 넘겨주면 된다!(a.c에서 수정할 것)

🡺a.c

->이전 자료구조는 멤리 사용정책을 사용자에게 강요하고 있다는 단점이 있다. 이를 해결하기 위해 메모리 할당 정책을 사용자에게 위임하자