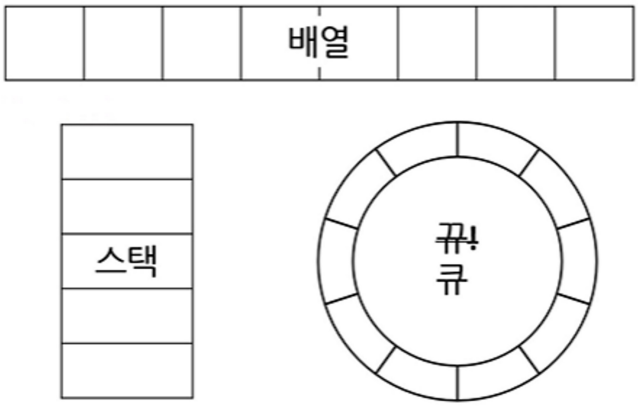
연결 리스트

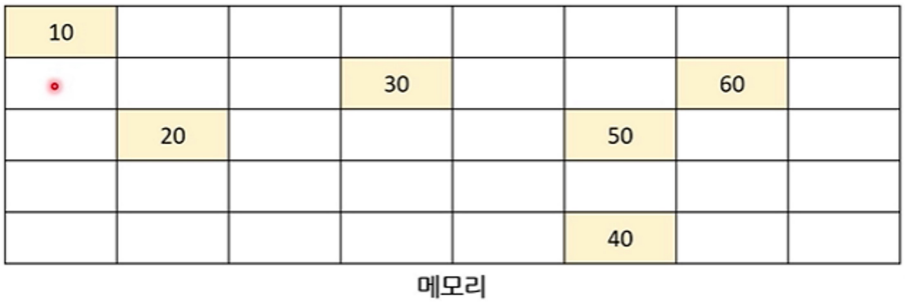
2021년 2월 21일 일요일 오후 12:38

256. 연결 리스트, 연결 리스트의 삽입/제거/검색

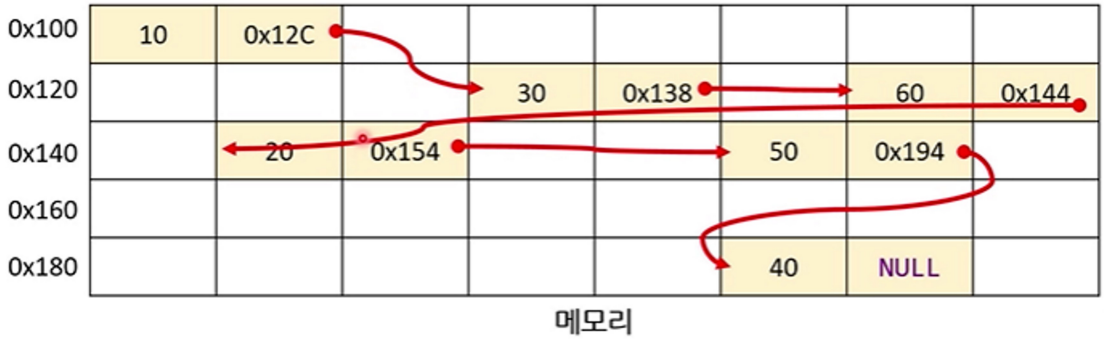
- 지금까지 우리가 본 자료구조들
  - 여태까지 본 자료구조들은 메모리에서 연속된 저장 방법에 기초



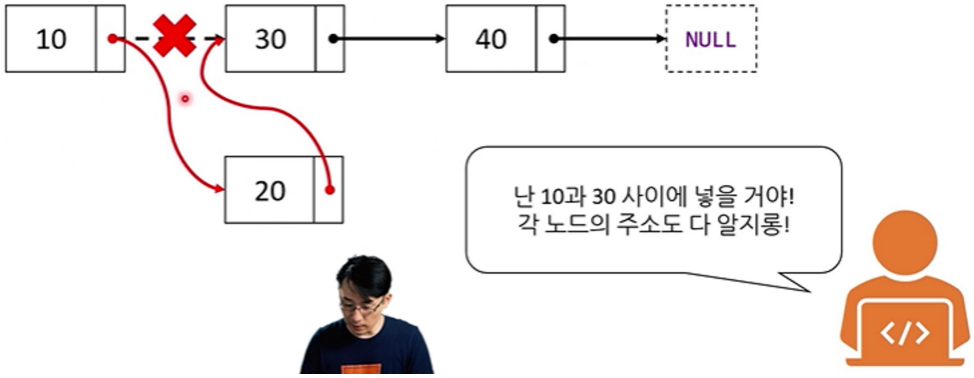
- 그러나 연결 리스트는 다르다
  - 여태까지 본 자료구조들은 메모리에서 연속된 저장 방법에 기초
  - 연결 리스트는 그런 제약을 깸
  - 연결 리스트는 중요하므로 언제든지 작성할 수 있어야 됨
- 연결 리스트(linked list)
  - 자료들이 메모리에 이렇게 산재해 있음
    - 연결 리스트의 각 자료를 노드(node)라고 부름
  - 자료형이 어떻게 산재해 있을 수 있는거지?
    - 동적 메모리 할당으로 필요에 따라 각 노드를 할당!



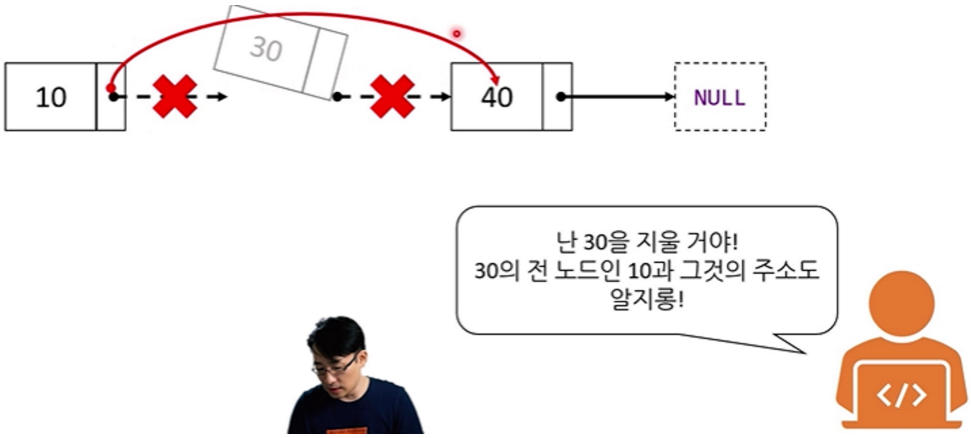
    - 노드 하나하나가 각각 동적 할당된 거임(같이 할당된 경우도 있고)
- 근데 어떻게 서로 연결하나요?
  - 그 둘 사이의 선후 관계를 별도로 지정
    - 어떻게? 다음에 오는 노드의 메모리 주소를 기억
    - 어디에? 노드에 있는 포인터 변수에
    - 제일 마지막 노드는 다음에 올 노드가 없으니 널 포인터



- 연결 리스트는 어려울 수 있는 자료구조
  - 연결 리스트는 매우 훌륭한 면접 문제
    - 메모리 관리 능력
    - 이중 포인터 사용 능력
  - 여태까지 설레설레 자료구조를 봐 왔다면 이건 집중해서 봐야 함
  - 면접 볼 때 빈 종이에다가 연결 리스트 코드 작성할 수 있을 정도가 되어야 함
    - 경력자도 면접 전에 한 번 보고가면 좋은 것 중에 하나
      - 경력자도 까먹은 경우가 있고 그만큼 중요해서
- 연결 리스트의 삽입
  - 이미 삽입할 위치를 알면 O(1)



  - 각 요소를 밀고 당길 필요 없이 삽입할 주소만 알고 있다면, 포인터가 가리키는 주소만 바꾸면 됨
    - 그래서 O(1)
- 연결 리스트의 제거
  - 이미 삭제할 위치를 알면 O(1)



- 연결 리스트의 검색
  - O(n)
  - 젤 첫 노드부터 찾을 때까지 뒤져야 함
    - 보통, 이 노드를 헤드(head)라고 부름
  - 색인으로 접근 불가능
  - 이렇게 찾은 뒤에 삽입을 하면 O(1)이 되는 것

    - 30과 40사이에 넣으면 된다
    - 처음 노드인 10부터 차례대로 검색

257. 연결 리스트: 전체 출력 예 & 노드 메모리 해제 예, 헤드 노드

- 연결 리스트 전체를 출력하는 코드 예

  - 헤드가 NULL일때까지 반복
- 헤드 노드
  - 연결 리스트의 첫 번째 노드를 가리키는 포인터
  - 처음 시작할 때 값은 NULL
    - 아직 노드가 하나도 없으니까
    - 여기다 새로운 노드를 동적 할당해서 대입해줄 것임

- 연결 리스트 해제 코드

  - 각 노드들의 동적 할당 메모리를 free()해줌
  - 해제한 다음, head는 어떤 메모리를 가리키고 있었고 그 메모리는 해제 됐으므로 head가 NULL을 가리키게 한다

258. 연결 리스트: 삽입하기 예

- 삽입하기 코드 예

- 새로 추가하는 값이 언제나 제일 앞에(head) 추가가 됨
- phead가 가리키는 값을 new_node가 가리키게 되고,
  phead는 new_node를 가리키게 된다
  - 처음에는 NULL을 new_node가 가리키게되고
    그 new_node를 phead가 가리킨다

- 왜 함수가 이중 포인터를 받는가?
  - 그냥 포인터일 때는 메인 함수의 head와 새로운 노드를 연결할 수가 없음
    - head의 사본이 전달되므로

  - 이중 포인터일 때는 매개변수 phead를 통해 메인 함수의 head와 연결 가능

259. 연결 리스트, 오름차 순으로 삽입하기 예

260. 연결 리스트: 노드 삭제

- 노드를 삭제하는 코드

  - 삭제하는 코드 역시 이전 노드의 주소를 저장하는 이중포인터 pp가 있음
    - 여기서는 이전 노드가 아닌 지울 노드가 가리키는 주소를 저장한다
      그래서 지우고 난 후 그 주소로 노드를 연결해줌

261. 연결 리스트의 용도

- 연결 리스트의 용도
  - 스택/큐과 같은 특성(삽입/삭제 방향) 때문에 쓰는 자료구조는 아님
    - 이 특성을 알고리즘에 이용하곤 함
  - 오히려 길이를 자유롭게 늘리거나 줄일 수 있기에 **배열의 한계를 넘으려고** 사용하던 자료구조
  - 즉, 최대 길이를 미리 특정할 수 없고 삽입/삭제가 빈번할 경우 사용
    - 연결 리스트는 삽입 삭제가 O(1)이니깐
  - 오늘날 어플리케이션 프로그램에서 사용 빈도는 많이 줄음
  - 기본적으로 동적 할당 배열을 더 흔히 사용
    - C#에서 List도 연결 리스트가 아니라 동적 할당 배열임
    - 최신 하드웨어의 특징 상 배열이 보장하는 훌륭한 메모리 지역성(인접한 메모리를 사용)이 성능에 유리한 경우가 많기 때문
  - 하지만 커널 모드 프로그래밍(예: 드라이버)에서는 여전히 많이 사용
    - 메모리 지역성을 해치지 않으면서도 충분히 큰 메모리(예: 4KB)를 미리 할당
    - 필요에 따라 그 메모리를 쪼개 연결 리스트의 노드로 사용 (예: 메모리 풀)

- 참고로 배열 다음으로 많이 사용하는 자료형은 '해시 맵'

참고하면 좋은 것들

- 단일 연결 리스트(singly-linked list)
  - 우리가 살펴 본 연결 리스트의 형태
  - 다음 노드를 가리키는 포인터만 저장
- 그 전의 노드를 가리키는 포인터도 저장하는 이중 연결 리스트(doubly-linked list)도 있음
- 이중 연결 리스트는 보통 head 외에 tail 포인터 변수도 가지고 있음