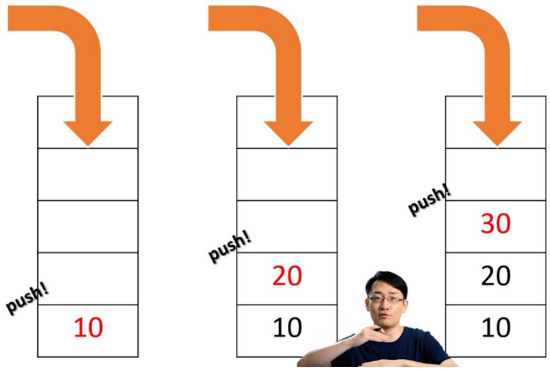
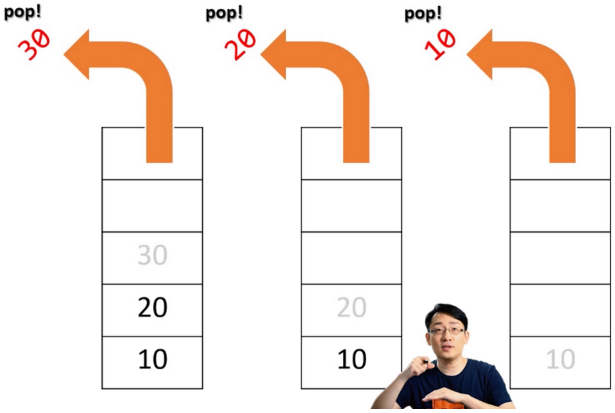
스택

2021년 2월 18일 목요일 오후 6:37

- 스택(stack)
  - 자료의 삽입과 삭제에 대한 규칙이 있는 자료구조 중 하나
  - 가장 먼저 자료구조에 삽입(push)된 데이터가 제일 마지막에 삭제(pop)됨
- 스택에 데이터를 삽입하는 그림



- 스택의 데이터를 제거하는 그림



- 스택을 선입후출 혹은 후입선출이라고 함
  - 가장 '먼저' 넣은 값이 가장 '나중에' 삭제됨
  - 이를 선입후출(First In Last Out)
  - 가장 '나중에' 넣은 값이 가장 '먼저' 삭제되는 건
  - 후입선출(Last In First Out)
- 스택을 현실에서 찾아본다면?
  - 수건을 쌓아두면 맨 위의 수건부터 쓰게 됨
  - 엘리베이터
    - 먼저 탄 사람이 안 쪽에 서고
    - 나중에 탄 사람이 바깥쪽에 서게 됨
    - 내릴 때도 나중에 탄 사람이 먼저 내림
- 스택의 중간 요소 삭제
- 스택은 중간에 있는 요소를 제거할 수 없나?
  - remove_at(3) 이런거
- 안됨
  - 언제나 젤 위에 있는 자료만 제거 가능
    즉, 중간 자료에 임의 접근 안 됨
- 스택 메모리와 개념이 같다
  - 스택 메모리도 같은 개념
  - 데이터가 쌓여감에 따라 주소가 감소하기 때문에 반대로 그렸을 뿐

- 스택의 구현
  - 배열로 쉽게 가능
  - 그냥 배열의 앞에서부터 데이터를 추가
  - 마지막 요소의 위치를 스택의 젤 위(top)라 생각하면 됨
  - 스택의 삽입 삭제가 왜 O(1)일까?
    - 배열에서 삽입 삭제가 O(1)인 경우를 생각하면 됨
- 스택의 삽입 예

```
enum { MAX_NUMS = 8 };

int s_nums[MAX_NUMS];
size_t s_num_count = 0;
```

```
void push(int n)
{
    assert(s_num_count < MAX_NUMS);
    s_nums[s_num_count++] = n;
}
```

```
int main(void)
{
    push(88);
    push(44);
    push(22);

    return 0;
}
```

  - s_num_count는 다음 요소가 들어갈 위치
  - main함수를 실행하고나서의 스택의 모습

    - 다음 요소가 들어갈 위치는 3이므로 s_num_count는 3임
- 스택의 삽입
  - 보통 푸시(push)라고 표현
    - 젤 위에 밀어 넣어 쌓는다고 해서
  - 시간 복잡도: O(1)
- 스택의 제거 예

- 스택이 비어있을때 pop을 하면 안되니까 is_empty()로 비어있나 검사
- 첫번째 pop()을 하고난후 상태

- 두번째 pop()을 하고난후 상태

- 스택의 제거
  - 스택에서 뽑아낸다고 해서 팝(pop) 이라고 함
  - 시간 복잡도: O(1)
- 스택의 검색
  - 시간 복잡도: O(n)
  - 젤 위부터 찾을 때까지 뒤져야 함
  - 보통 push()와 pop()만 허용하므로 임의의 요소에 접근할 방법이 없음
  - 그럼 검색을 어떻게 할까?
- 전부 다 뽑으며 확인 후 다시 넣어줌
  - 모든 요소를 다 제거했다가 다시 원상복구해야 함

    - 빼낸 스택 요소를 다른 스택에 넣으면, 원상복구할 때 원래 순서대로 넣을 수 있음
- 스택 검색의 시간 복잡도
    - 모든 요소를 다 제거했다가 다시 원상복구해야 함
    - 그래서 제거에 O(n) + 복구에 O(n)이 필요
    - 따라서, O(2n)이지만 이건 그냥 O(n)라고 말함
        - O 표기법은 Order of ** 의 약자
        - 어떤 것의 비례인지만 표기함
            - 뭐의 정비례관계, 뭐의 제곱의 정비례관계, 2의 뭐승의 정비례관계
        - 그래서 2n, 3n, 등등 모두 n 으로 봄
- 스택의 용도
    - 일련의 자료들의 순서를 뒤집는데 유용
    - 왜 뒤집지?
        - 그냥 뒤집고 싶어서
        - 현재 데이터 순서대로 연산하는 것이 컴퓨터에 적합하지 않은 경우
            - 데이터 순서를 뒤집어야 연산하기 알맞은 때가 있다
    - 생각보다 스택이 유용한 곳이 정말 많음
        - 알고리듬 구현시
- 스택으로 자료순서 뒤집기
    - 스택에 배열의 요소들을 넣음

- 스택의 요소들을 하나씩 뽑아서 배열에 저장하면
  배열의 요소가 뒤집어짐!

- ## 컴퓨터 연산 순서에 맞게 자료 재정리
  - 다음과 같은 식(문자열로 들어 옴)을 평가하는 계산기를 만든다면?

    
  - 이걸 중위(infix) 표기법이라고 함
  - 사람들에게 매우 익숙
  - 그러나 순서대로 한 글자씩 읽으면서 평가가 불가능
  - 이걸 후위(postfix) 표기법으로 바꾸면 컴퓨터로 연산이 쉬움

    
    - 중위 -> 후위로 바꾸는 것도 스택으로 가능하나 그 알고리즘은 찾아보도록 하자
  - 후위 표기법으로 적은 식은 간단하게 스택을 이용해서 계산 가능
- ## 후위 표기법 계산 로직
  1. 한 글자를 읽는다
  2. 글자를 읽는데 성공한 경우
     i. 피연산자면, 스택에 넣는다
     ii. 연산자면, 피연산자 둘을 스택에서 꺼내 연산자로 계산하고 그 결과를 다시 스택에 넣는다
     iii. 1번으로 돌아감
  3. 글자를 읽는데 실패한 경우(마지막)
     i. 스택에서 꺼내면 그게 결과
- ## 스택은 재귀함수를 제거하는데도 유용하다
  - 재귀 함수는 함수 호출 트리를 이용
  - 함수는 각 호출마다 새로 스택 프레임을 만들어서 중간 결과를 저장
    - 스택에 중간 변수값들을 계속 올리고 있는것임
      이걸 우리가 스택으로 구현하면 재귀를 안 써도 된다
  - 따라서, 스택 자료구조를 이용하면 꽤 많은 재귀 함수를 재귀 없이 반복문으로 구현할 수 있음
  - 직접 해보자