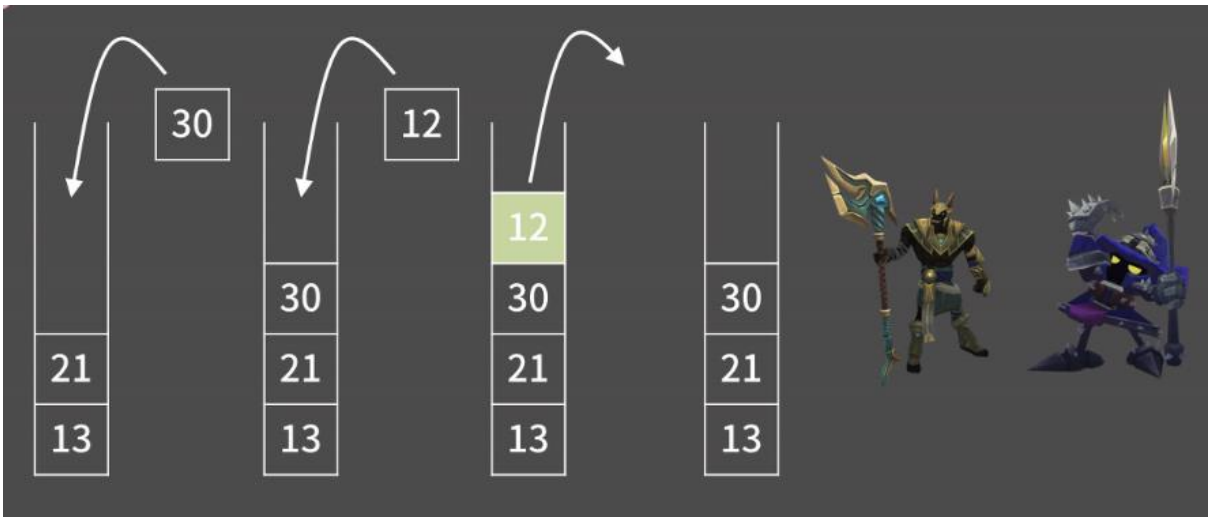
스택

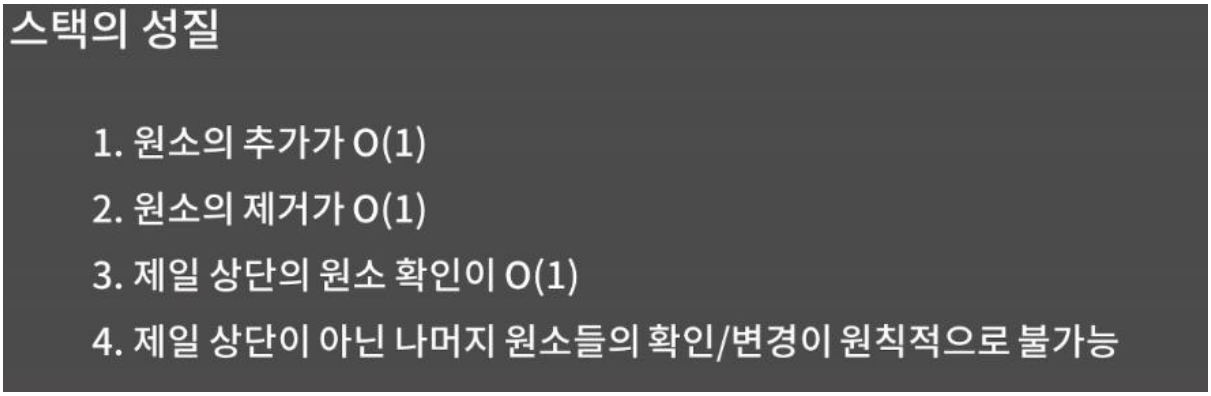
강의 출처: 바킹독 실전 알고리즘 (https://blog.encrypted.gg/922?category=773649)

1. 정의와 성질



스택은 바로 한쪽 끝에서만 원소를 넣거나 뺄 수 있는 자료구조입니다. 대충 프링글스 통을 생각하면 이해가 쉬울 것입니다. 아니면 엘리베이터를 생각해도 되겠습니다.

스택은 구조적으로 먼저 들어간 원소가 제일 나중에 나오게 되는데, 이런 의미로 FILO(First In Last Out) 자료구조라고 부르기도 합니다. 참고로 큐나 덱도 스택처럼 특정 위치에서만 원소를 넣거나 뺄 수 있는 제한이 걸려있습니다. 그래서 스택, 큐, 덱을 묶어서 Restricted Structure라고 부르기도 합니다.



스택은 특정 위치에서만 원소를 넣거나 뺄 수 있게 제한을 둔 대신에 원소의 추가와 제거가 모두 O(1)입니다. 나중에 구현을 같이 해보겠지만 우리가 배열의 끝에서 원소를 추가/제거할 때 시간복잡도가 O(1)이었던 것과 완전 상황이 똑같습니다. 그리고 제일 상단의 원소 확인 또한 O(1)입니다.

대신 스택에서는 제일 상단이 아닌 나머지 원소들의 확인/변경이 원칙적으로 불가능합니다. 원칙적으로 불가능하다는 뜻을 잘 이해할 필요가 있는데, 원래 스택이라는 자료구조는 원소의 추가/제거/제일 상단의 원소 확인이라는 기능만 제공하는 자료구조입니다. 그래서 제일 상단이 아닌 나머지 원소들의 확인/변경은 스택에서 제공하는 기능이 아닙니다.

2. 기능과 구현

스택을 배열로 구현할 때에는 단지 원소를 담은 큰 배열 한 개와 인덱스를 저장할 변수 한 개만 필요합니다. 이들이 실제 스택에 어떻게 대응되는 방식을 확인해보면 오른쪽 그림과 같습니다.

{13, 21, 30}이 담겨있는 스택을 나타내기 위해 dat[0], dat[1], dat[2] 각각에 13, 21, 30을 썼고 pos는 3이라는 값을 가집니다. 이와 같이 스택의 값들은 dat의 0번지부터 저장되고 pos는 다음에 원소가 추가될 때 삽입해야하는 곳을 가리키고 있습니다. 그리고 잘 생각해보면 pos의 값이 곧 스택의 길이, 즉 스택 내의 원소의 수를 의미한다는 것을 알 수 있습니다.

reference : http://www.cplusplus.com/reference/stack/stack/

https://github.com/blisstoner/basic-algo-lecture-metarial/blob/master/0x05/stack_example.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(void) {
  stack<int> S;
  S.push(10); // 10
  S.push(20); // 10 20
  S.push(30); // 10 20 30
  cout << S.size() << '\n'; // 3
  if(S.empty()) cout << "S is empty\n";
  else cout << "S is not empty\n"; // S is not empty
  S.pop(); // 10 20
  cout << S.top() << '\n'; // 20
  S.pop(); // 10
  cout << S.top() << '\n'; // 10
  S.pop(); // empty
  if(S.empty()) cout << "S is empty\n"; // S is empty
  cout << S.top() << '\n'; // runtime error 발생
}
```

스택이 비어있는데 top을 호출하면 런타임 에러가 발생합니다. 스택이 비어있는데 pop을 호출해도 마찬가지입니다. 그렇기 때문에 스택이 비어있을 때에는 top이나 pop을 호출하지 않도록 해야합니다. 또 제출한 코드의 결과로 런타임 에러를 받았을 경우에는 다양한 원인이 있을 수 있겠지만 스택이 비어있을 때 top이나 pop을 하지는 않았을지 의심해볼 수 있습니다.