

8월 19일 (목)

1. 백준 알고리즘 문제풀이 - 스택수열 (1874번)

문제

스택 (stack)은 기본적인 자료구조 중 하나로, 컴퓨터 프로그램을 작성할 때 자주 이용되는 개념이다. 스택은 자료를 넣는 (push) 입구와 자료를 뽑는 (pop) 입구가 같아 제일 나중에 들어간 자료가 제일 먼저 나오는 (LIFO, Last in First out) 특성을 가지고 있다.

1부터 n 까지의 수를 스택에 넣었다가 뽑아 늘어놓음으로써, 하나의 수열을 만들 수 있다. 이때, 스택에 push하는 순서는 반드시 오름차순을 지키도록 한다고 하자. 임의의 수열이 주어졌을 때 스택을 이용해 그 수열을 만들 수 있는지 없는지, 있다면 어떤 순서로 push와 pop 연산을 수행해야 하는지를 알아낼 수 있다. 이를 계산하는 프로그램을 작성하라.

입력

첫 줄에 n ($1 \leq n \leq 100,000$)이 주어진다. 둘째 줄부터 n 개의 줄에는 수열을 이루는 1이상 n 이하의 정수가 하나씩 순서대로 주어진다. 물론 같은 정수가 두 번 나오는 일은 없다.

출력

입력된 수열을 만들기 위해 필요한 연산을 한 줄에 한 개씩 출력한다. push연산은 +로, pop 연산은 -로 표현하도록 한다. 불가능한 경우 NO를 출력한다.

예제 입력 1 복사

```
8
4
3
6
8
7
5
2
1
```

예제 출력 1 복사

```
+
```

```
+
```

```
+
```

```
+
```

```
-
```

```
-
```

```
+
```

```
+
```

```
+
```

```
+
```

```
-
```

```
-
```

```
-
```

```
-
```

```
-
```

```

# 1874번 스택수열 문제
# 처음엔 이해 안갔지만, 하나씩 확인 해본 후 이해

n = int(input())
count = 0 # 오름차순
stack = [] # 숫자가 들어갈 스택 (1,2,3,4,5,6,7,8,9 순서로)
result = [] # 부호가 들어갈 스택
no_message=True

for i in range(0,n):
    x = int(input())

    while count < x:
        count += 1
        stack.append(count)
        result.append("+")

    if stack[-1]==x:
        stack.pop()
        result.append("-")
    else:
        no_message = False # 스택 구성이 안되면 False 로 변경 뒤 NO 출력
        break

if no_message==False:
    print("NO")
else:
    print("\n".join(result))

```

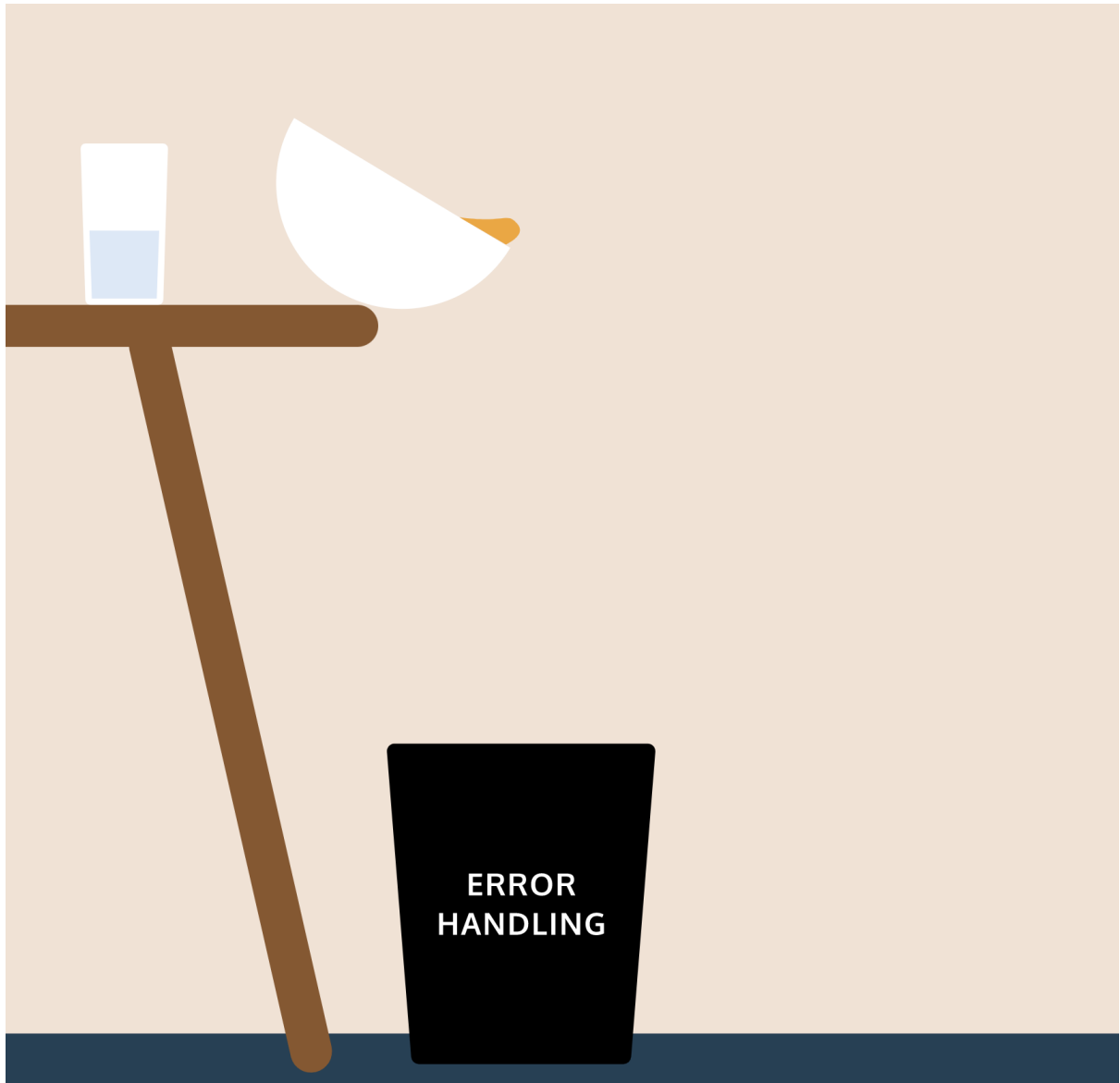
2. Codecademy - JavaScript Syntax, Part 3 - Learn JavaScript Syntax: Error Handling

Lesson 1. Learn Error Handling

- Introduction to Error Handling
- Runtime Errors

- Constructing an Error
- The throw Keyword
- The try...catch Statement
- Handling with try...catch
- Error Handling Review

- **Introduction to Error Handling**

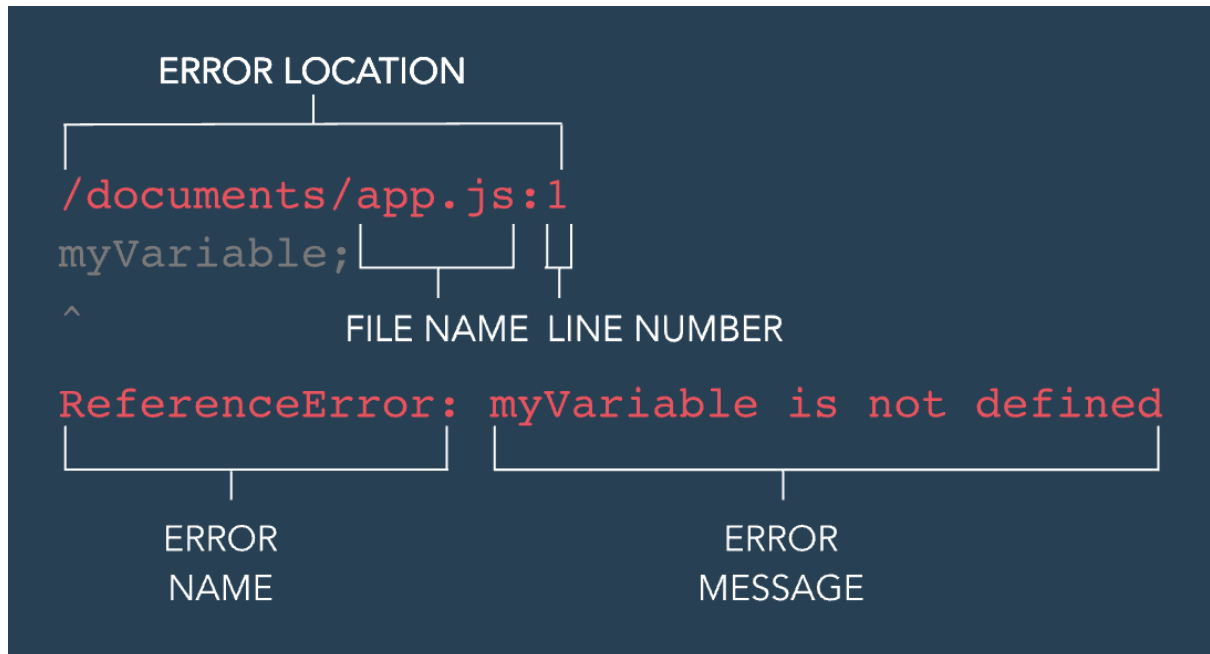


오르

우리는 항상 에러가 일어나기 전에 멈출 수 없다. 그러나 우리는 프로그래밍이 계속 돌아갈 수 있도록, 에러에 반응하고 잘 넘어갈 수 있는 백업 플랜을 설정할 수 있다. 바로 `try...catch` 을 사용하여

- Runtime Errors

에러들은 왜 우리의 프로그램이 작동하지 않는지, 또 왜 그러한 에러가 던져졌는지를 우리에게 말해주는 유용한 메시지를 담고 있다. 에러가 던져졌을 때, 우리의 프로그램은 실행을 멈추고 콘솔이 빨간색 텍스트와 함께 등장하게 된다.



이렇게 코드가 실행 중일 때 던져지는 에러를 바로 "런타임 에러" 라고 한다. 자바스크립트에는 무엇이 틀렸는 지 알려주는 이름과 메시지 프로퍼티들을 가진 내장된 error 객체가 있다. 빌트인 런타임 에러는 다음 두 가지를 가지고 있다.

⇒ `ReferenceError` : 호출하려고 하는 변수 나 함수를 발견할 수 없을 때

⇒ `TypeError` : 벨류가 유효한 타입이 아닐 때

```
const reminder = 'Reduce, Reuse, Recycle';
reminder = 'Save the world';
// TypeError: Assignment to constant variable.
console.log('This will never be printed!');
```

저렇게 런타임 도중 에러가 나게 되면 에러가 난 코드의 그 이후 코드들은 실행조차 할 수 없다. 즉, 위 `console.log()` 코드는 출력되지 않는다. 이러한 것을 방지하기 위해 `try ... catch` 를 사용하는 것이다.

- **Constructing an Error (구조적인 에러)**

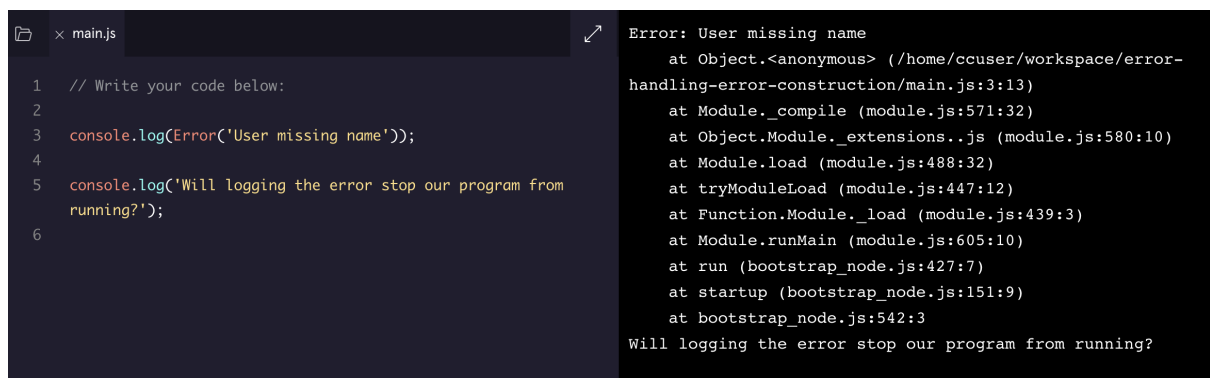
자바스크립트 에러 객체에 없는 에러는 어떻게 처리할까? 이럴 경우 Error 함수를 사용해서 에러문을 만들면 된다.

```
console.log(Error('Your password is too weak.'));  
// Prints: Error: Your password is too weak.
```

```
console.log(new Error('Your password is too weak.'));  
// Prints: Error: Your password is too weak.
```

위는 에러를 생성하는 것이지 에러를 유저에게 던지는 것은 아니다. 에러를 생성하는 것은 에러를 던지는 것과 같지않다는 것을 명심해라. 던져지는 에러는 프로그램이 멈추도록 할 것이다. 다음 챕터에는 어떻게 생성된 에러를 던질 수 있는지에 대해 배울 것이다.

실습예시



The screenshot shows a code editor with a file named 'main.js'. The code contains two console.log statements. The first one logs an Error object created with the message 'User missing name'. The second one logs a string message. The output in the console shows the error message and a detailed stack trace starting from the anonymous function at line 3:13, through module compilation and loading, to the bootstrap node startup. The string message is also printed at the bottom of the console output.

```
1 // Write your code below:  
2  
3 console.log(Error('User missing name'));  
4  
5 console.log('Will logging the error stop our program from  
6 running?');
```

```
Error: User missing name  
    at Object.<anonymous> (/home/ccuser/workspace/error-  
handling-error-construction/main.js:3:13)  
    at Module._compile (module.js:571:32)  
    at Object.Module._extensions..js (module.js:580:10)  
    at Module.load (module.js:488:32)  
    at tryModuleLoad (module.js:447:12)  
    at Function.Module._load (module.js:439:3)  
    at Module.runMain (module.js:605:10)  
    at run (bootstrap_node.js:427:7)  
    at startup (bootstrap_node.js:151:9)  
    at bootstrap_node.js:542:3  
Will logging the error stop our program from running?
```

프로그램이 멈추지않고 계속해서 코드를 출력하는 것을 볼 수 있다.

- The throw Keyword

에러를 생성하는 것은 프로그램을 멈출 수 없다. 에러는 프로그램이 멈추기 위해 던져져야 된다는 것을 기억해라.

에러를 던져주기 위해 throw 키워드를 사용할 수 있다.

```
throw Error('Something wrong happened');  
// Error: Something wrong happened
```

When we use the `throw` keyword, the error is thrown and code after `throw` statement will not execute. Take for example:

```
throw Error('Something wrong happened');  
// Error: Something wrong happened  
  
console.log('This will never run');
```

빌트인 에러 객체인 `TypeError`, `ReferenceError` 와 같이 코드의 실행을 멈추고 에러의 내용을 던져 줄 것이다.

실습 예시

```
main.js
1 throw Error('Username or password do not match');
```

```
/home/ccuser/workspace/error-handling-throw/main.js:1
(function (exports, require, module, __filename, __dirname)
{ throw Error('Username or password do not match');

^
Error: Username or password do not match
    at Object.<anonymous> (/home/ccuser/workspace/error-handling-throw/main.js:1:69)
    at Module._compile (module.js:571:32)
    at Object.Module._extensions..js (module.js:580:10)
    at Module.load (module.js:488:32)
    at tryModuleLoad (module.js:447:12)
    at Function.Module._load (module.js:439:3)
    at Module.runMain (module.js:605:10)
    at run (bootstrap_node.js:427:7)
    at startup (bootstrap_node.js:151:9)
    at bootstrap_node.js:542:3
```

이렇게 빨간 글씨로 에러를 던져준다.

• The try...catch Statement

try...catch 문은 에러를 다룬다. 이는 에러를 다루는 동시에 계속해서 코드가 실행될 수 있도록 한다. 아래 예시를 보자.

```
try {
  throw Error('This error will get caught');
} catch (e) {
  console.log(e);
}
// Prints: This error will get caught

console.log('The thrown error that was caught in the
try...catch statement!');
// Prints: 'The thrown error that was caught in the
try...catch statement!'
```

→ try {} 를 try block 이라고 하며, try 내에 에러를 다루는 코드가 적혀 있다

- catch block 은 try block 에서부터 던져진 에러를 받는다. 'e' 는 던져진 에러를 나타낸다.
- catch(e) 문은 에러를 다루기 위한 코드를 실행시킨다.
- try 문에서 에러가 잡혔기 때문에 try catch 문 이후에 있는 맨 아래 console.log 문은 프린트 될 것이다.

- **Handling with try...catch**

생성 에러뿐 만 아니라 , 빌트인 에러객체 또한 try..catch 문으로 핸들링 하며 다음 코드를 실행시킬 수 있다.

```
const someVar = 'Cannot be reassigned';
try {
  someVar = 'Still going to try';
} catch(e) {
  console.log(e);
}
// Prints: TypeError: Assignment to constant variable.
```

위 예시에서는 throw 키워드를 사용하지 않고 const 변수를 변경하려고 할 때 나타나는 TypeError 를 일으켰다. catch 문은 해당 에러문을 콘솔로그로 출력할 것이다.

빌트인 에러 객체에서 try...catch 문을 사용하는 것은 우리 것에 직접 쓰여지지 않은 외부 소스의 데이터를 사용해야할 때 정말로 유익하다.

예를 들어, 우리가 데이터베이스로 부터 array 를 받아와야 할 때, 잘못된 요청으로 string 이 반환되었다면, 코드는 멈춰버릴 것이다. 하지만 try...catch 문을 사용하면 그러한 에러를 핸들링하면서 무엇이 잘못됐는지 알려주면서 우리의 프로그래밍이 계속해서 돌아가도록 해줄 것이다.

실습 예시

```
1 function capAllElements(arr){
2   try {
3     arr.forEach((el, index, array) => {
4       array[index] = el.toUpperCase();
5     });
6   } catch(e) {
7     console.log(e);
8   }
9 }
10
11 capAllElements('Incorrect argument');
```

```
TypeError: arr.forEach is not a function
    at capAllElements (/home/ccuser/workspace/error-handling-try-catch-ii/main.js:3:9)
    at Object.<anonymous> (/home/ccuser/workspace/error-handling-try-catch-ii/main.js:11:1)
    at Module._compile (module.js:571:32)
    at Object.Module._extensions..js (module.js:580:10)
    at Module.load (module.js:488:32)
    at tryModuleLoad (module.js:447:12)
    at Function.Module._load (module.js:439:3)
    at Module.runMain (module.js:605:10)
    at run (bootstrap_node.js:427:7)
    at startup (bootstrap_node.js:151:9)
```

오늘의 단어

- Let's say : 예를 들어 (= let us say)