8월 20일 (금)

1. 백준 알고리즘 문제 풀이 - 에디터 (1406번)

문제

한 줄로 된 간단한 에디터를 구현하려고 한다. 이 편집기는 영어 소문자만을 기록할 수 있는 편집기로, 최대 600,000글자까지 입력할 수 있다.

이 편집기에는 '커서'라는 것이 있는데, 커서는 문장의 맨 앞(첫 번째 문자의 왼쪽), 문장의 맨 뒤(마지막 문자의 오른쪽), 또는 문장 중간 임의의 곳(모든 연속된 두 문자 사이)에 위치할 수 있다. 즉 길이가 L인 문자열이 현재 편집기에 입력되어 있으면, 커서가 위치할 수 있는 곳은 L+1가지 경우가 있다.

이 편집기가 지원하는 명령어는 다음과 같다.

L	커서를 왼쪽으로 한 칸 옮김 (커서가 문장의 맨 앞이면 무시됨)
D	커서를 오른쪽으로 한 칸 옮김 (커서가 문장의 맨 뒤이면 무시됨)
B	커서 왼쪽에 있는 문자를 삭제함 (커서가 문장의 맨 앞이면 무시됨) 삭제로 인해 커서는 한 칸 왼쪽으로 이동한 것처럼 나타나지만, 실제로 커서의 오른쪽에 있던 문자는 그대로임
P $	$라는 문자를 커서 왼쪽에 추가함

초기에 편집기에 입력되어 있는 문자열이 주어지고, 그 이후 입력한 명령어가 차례로 주어졌을 때, 모든 명령어를 수행하고 난 후 편집기에 입력되어 있는 문자열을 구하는 프로그램을 작성하시오. 단, 명령어가 수행되기 전에 커서는 문장의 맨 뒤에 위치하고 있다고 한다.

입력

첫째 줄에는 초기에 편집기에 입력되어 있는 문자열이 주어진다. 이 문자열은 길이가 N이고, 영어 소문자로만 이루어져 있으며, 길이는 100,000을 넘지 않는다. 둘째 줄에는 입력할 명령어의 개수를 나타내는 정수 M(1 ≤ M ≤ 500,000)이 주어진다. 셋째 줄부터 M개의 줄에 걸쳐 입력할 명령어가 순서대로 주어진다. 명령어는 위의 네 가지 중 하나의 형태로만 주어진다.

출력

첫째 줄에 모든 명령어를 수행하고 난 후 편집기에 입력되어 있는 문자열을 출력한다.

예제 입력 1 복사

```
abcd
3
P x
L
P y
```

예제 출력 1 복사

```
abcdyx
```

예제 입력 2 복사

```
abc
9
L
L
L
L
L
P x
L
B
P y
```

예제 출력 2 복사

```
yxabc
```

예제 입력 3 복사

```
dmih
11
B
B
P x
L
B
B
B
P y
D
D
P z
```

예제 출력 3 복사

```
yxz
```

출처

Olympiad > Croatian Highschool Competitions in Informatics > 2004 > National Competition #1 - Juniors 2번

```python
# 백준 1406번 에디터 문제 - 두 개의 stack 이용하여 문제해결

from sys import stdin

stack_l = list(stdin.readline().strip()) # 여러개의 숫자를 입력받을 때는 sys모듈의 stdin readline()을 사용하자.
stack_r = []
n = int(input()) # 한 개만 입력받기에 input() 써도 된다.

for _ in range(n):
    temp = stdin.readline() # 실행 명령어를 받는다.
    if temp[0] == 'L':
        if len(stack_l) == 0:
            continue
        stack_r.append(stack_l.pop())
    elif temp[0] == 'D':
        if len(stack_r) == 0:
            continue
        stack_l.append(stack_r.pop())
    elif temp[0] == 'B':
        if len(stack_l) == 0:
            continue
        stack_l.pop()
    elif temp[0] == 'P':
        stack_l.append(temp[2])

# 1번째 출력 방법
stack_l.extend(stack_r[::-1])

# 2번째 출력 방법
stack_r.reverse()
stack_l.extend(stack_r)
# 출력
print("".join(stack_l))
```

- 입력 받을 때 input( ) 대신 sys.stdin.readline() 을 사용하는 이유

한 두줄 입력받는 문제들과 다르게, 반복문으로 여러줄을 입력 받아야 할 때는 input( ) 을 사용하게 되면 시간초과가 발생할 수 있다.

```python
import sys

T = int(input()) #Test case
for i in range(T):
        a,b = map(int, sys.stdin.readline().split())
        print(a+b)
```

- **sys.stdin.readline( ) 사용법**

⇒ 한 개의 정수를 입력받을 때

```
import sys
a = int(sys.stdin.readline())
```

⇒ 임의의 개수의 정수를 한줄에 입력받아 리스트에 저장할 때

```
import sys
data = list(map(int,sys.stdin.readline().split()))
```

⇒ 정해진 개수의 정수를 한줄에 입력받을 때

```
import sys
a,b,c = map(int,sys.stdin.readline().split())
```

⇒ 임의의 개수의 정수를 n줄 입력받아 2차원 리스트에 저장할 때

```
import sys
data = []
n = int(sys.stdin.readline())
for i in range(n):
  data.append(list(map(int,sys.stdin.readline().split())))
```

⇒ 문자열 n줄을 입력받아 리스트에 저장할 때

```
import sys
n = int(sys.stdin.readline())
data = [sys.stdin.readline().strip() for i in range(n)]
```

- **파이썬 자료형 별 주요 연산자의 시간 복잡도(Big-O)**

파이썬 자료형 별 주요 연산자의 시간 복잡도 (Big-O)

14 Jun 2017 2 Comments |

들어가기

알고리즘 문제를 풀다 보면 시간복잡도를 생각해야 하는 경우가 종종 생긴다. 특히 codility는 문제마다 시간복잡도 기준이 있어서, 기준을 넘기지 못하면 문제를 풀어도 score가 50 이하로 나오는 경우가 많다.

찾아보니 파이썬 주요 함수, 메소드의 시간복잡도를 정리한 페이지가 있었다. (Complexity of Python Operations) 자주 사용하는 것들을 이곳에 정리하고 종종 참고하려고 한다.

list

Operation	Example	Big-O	Notes
Index	l[i]	O(1)	
Store	l[i] = 0	O(1)	
Length	len(l)	O(1)	
Append	l.append(5)	O(1)	
Pop	l.pop()	O(1)	l.pop(-1) 과 동일
Clear	l.clear()	O(1)	l = [] 과 유사
Slice	l[a:b]	O(b-a)	l[:] : O(len(l)-0) = O(N)
Extend	l.extend(…)	O(len(…))	확장 길이에 따라
Construction	list(…)	O(len(…))	요소 길이에 따라
check ==, !=	l1 == l2	O(N)	비교
Insert	l .insert(i, v)	O(N)	i 위치에 v를 추가
Delete	del l[i]	O(N)	
Remove	l.remove(…)	O(N)	
Containment	x in/not in l	O(N)	검색
Copy	l.copy()	O(N)	l[:] 과 동일 - O(N)
Pop	l.pop(i)	O(N)	l.pop(0):O(N)
Extreme value	min(l)/max(l)	O(N)	검색
Reverse	l.reverse()	O(N)	그대로 반대로
Iteration	for v in l:	O(N)	
Sort	l.sort()	O(N Log N)	
Multiply	k*l	O(k N)	[1,2,3] * 3 » O(N**2)

Dict

Operation	Example	Big-O	Notes
Index	d[k]	O(1)	
Store	d[k] = v	O(1)	
Length	len(d)	O(1)	
Delete	del d[k]	O(1)	
get/setdefault	d.method	O(1)	
Pop	d.pop(k)	O(1)	
Pop item	d.popitem()	O(1)	
Clear	d.clear()	O(1)	s = {} or = dict() 유사
View	d.keys()	O(1)	d.values() 동일
Construction	dict(…)	O(len(…))	
Iteration	for k in d:	O(N)	

- **파이썬 extend( ) vs append( )**

파이썬 리스트에 새로운 원소를 추가하는 방법인 extend 와 append 의 차이점에 대해 알아보자.

(insert(i, x) 함수도 리스트에 새로운 원소를 추가하는 방법이다. 위치 i에 x 를 추가한다.)

list.append(x) 는 리스트 끝에 x 1개를 그대로 넣는다.

list.extend(iterable)은 리스트 끝에 가장 바깥쪽 iterable 의 모든 항목을 넣는다.

이해를 돕기 위해 실습을 통해 알아보겠습니다.

y가 리스트형일 때입니다.

```
x = ['Tick', 'Tock', 'Song']
y = ['Ping', 'Pong']
x.append(y)
print('x:', x)
```

```
x: ['Tick', 'Tock', 'Song', ['Ping', 'Pong']]
```

```
x = ['Tick', 'Tock', 'Song']
y = ['Ping', 'Pong']
x.extend(y)
print('x:', x)
```

```
x: ['Tick', 'Tock', 'Song', 'Ping', 'Pong']
```

append는 x 그 자체를 원소로 넣고 extend는 iterable의 각 항목들을 넣습니다

그럼 리스트안에 리스트는 어떻게 처리될까요? y가 리스트형안에 리스트형이 있을 때입니다.

```
x = ['Tick', 'Tock', 'Song']
y = [['Ping', 'Pong']]
x.append(y)
print('x:', x)
```

```
x: ['Tick', 'Tock', 'Song', [['Ping', 'Pong']]]
```

```
x = ['Tick', 'Tock', 'Song']
y = [['Ping', 'Pong']]
x.extend(y)
print('x:', x)
```

```
x: ['Tick', 'Tock', 'Song', ['Ping', 'Pong']]
```

append는 x 그 자체를 원소로 넣고 extend는 가장 바깥쪽 iterable을 넣습니다

y가 문자열일 때입니다.

```
x = ['Tick', 'Tock', 'Song']
y = 'Ping'
x.append(y)
print('x:', x)
```

```
x: ['Tick', 'Tock', 'Song', 'Ping']
```

```
x = ['Tick', 'Tock', 'Song']
y = 'Ping'
x.extend(y)
print('x:', x)
```

```
x: ['Tick', 'Tock', 'Song', 'P', 'i', 'n', 'g']
```

append는 x 그 자체를 원소로 넣고 extend는 문자열의 각 알파벳을 넣습니다

2. Codecademy - TDD Fundamentals - Why Test?

Lesson 1. Why Test?

- **Introduction**
- **Manual Testing**

- **Automated Testing**
- **The Test Suite**
- **Tests As Documentation**
- **Regression**
- **Review**

- **Introduction**

소프트웨어 에러는 많은 유저나 개발자들이 많은 비용을 낭비하게 할 수 있다. 하지만 이러한 에러는 불가피한 것으로 제품을 만들면서, 또 배포한 후에도 계속해서 테스트를 진행해야 한다. 자동화된 테스트 수행으로 제품이 에러로부터 안전하다는 확신을 심어줄 수 있을 것이다. 이번 레슨에서는 automated test suite 에 대한 정의, 이러한 테스트 수행이 소프트웨어 개발에 있어서 어떻게 사용되는 지, 자동화된 테스팅의 이점에 대해서 알아볼 것이다.

- **Manual Testing (수동적인 테스팅)**

소프트웨어 테스팅은 컴퓨터 소프트웨어의 완벽성과 퀄리티를 평가하는 과정이다. 이는 소프트웨어 시스템의 기대행동양식과 실제행동양식을 비교하는 것이다.

소프트웨어 테스팅을 수행하기 위한 한가지 방법은 manual testing(수동적 테스팅)이다. 이는 사람이 직접 웹사이트에서 클릭, 드래그, 타이핑하는 것으로 만약 기대행동과 다른 결과를 산출한다면 그 애플리케이션은 에러를 가지고 있는 것이다.

에러는 버그라고도 한다. 버그는 error, fault, flaw 라고 하며 소프트웨어 시스템에서 기대되지 않는 행동양식이다.

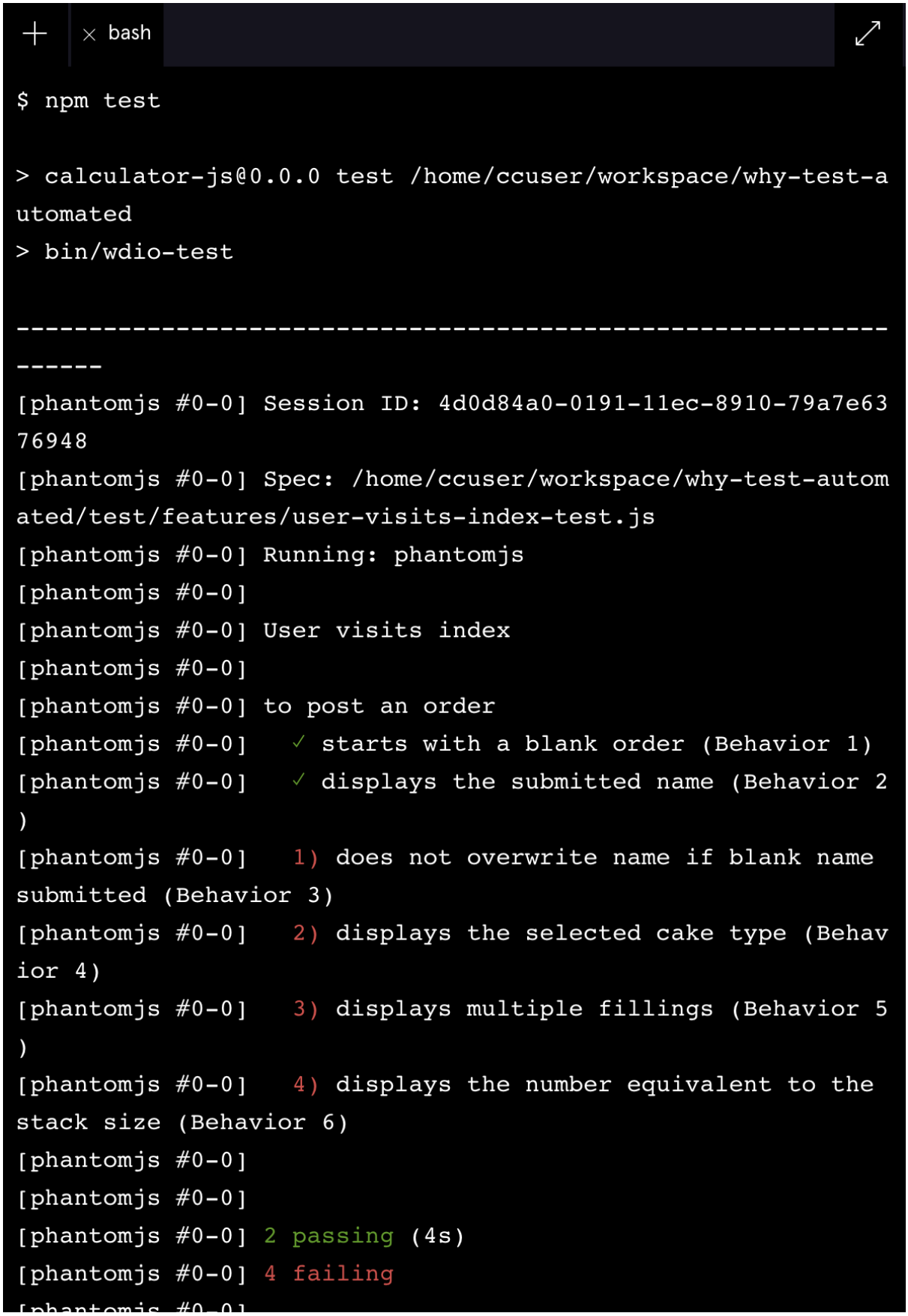
- **Automated Testing (자동화 테스트)**

수동적인 테스팅은 너무 오래걸리며, 실수할 가능성이 있다. 자동화 테스팅을 통해 시간과 비용을 아끼며, 프로그램의 퀄리티 또한 개선 시킬 수 있다. 자동화 테스팅은 수동 테스팅보다 빠르며, 더 믿음직하며, 유지보수성이 뛰어나다.(review, edit, extend a collection of tests 가 가능)

자동화 테스팅의 과정은 다음과 같다.

1. 코드와 대응하는 테스트를 작성한다.
2. 터미널에 테스트를 실행하는 커맨드를 입력한다.
3. 만약 앱이 의도한대로 실행된다면, 모든 테스트는 pass 가 되어야하고, 개발은 완성된다.
4. 만약 앱이 의도하지않은 행동을 한다면, 적어도 하나의 테스트는 fail을 한 것이고, 코드를 고치고 step2로 다시 돌아간다.

터미널 테스팅 예시


bash
$ npm test
> calculator-js@0.0.0 test /home/ccuser/workspace/why-test-automated
> bin/wdio-test
[phantomjs #0-0] Session ID: 4d0d84a0-0191-11ec-8910-79a7e6376948
[phantomjs #0-0] Spec: /home/ccuser/workspace/why-test-automated/test/features/user-visits-index-test.js
[phantomjs #0-0] Running: phantomjs
[phantomjs #0-0]
[phantomjs #0-0] User visits index
[phantomjs #0-0]
[phantomjs #0-0] to post an order
[phantomjs #0-0] ✓ starts with a blank order (Behavior 1)
[phantomjs #0-0] ✓ displays the submitted name (Behavior 2)
[phantomjs #0-0] 1) does not overwrite name if blank name submitted (Behavior 3)
[phantomjs #0-0] 2) displays the selected cake type (Behavior 4)
[phantomjs #0-0] 3) displays multiple fillings (Behavior 5)
[phantomjs #0-0] 4) displays the number equivalent to the stack size (Behavior 6)
[phantomjs #0-0]
[phantomjs #0-0]
[phantomjs #0-0] 2 passing (4s)
[phantomjs #0-0] 4 failing

```
[phantomjs #0-0] 1) to post an order does not overwrite name
 if blank name submitted (Behavior 3):
[phantomjs #0-0] expected 'DELIVER TO:' to include 'Hungry P
erson'
[phantomjs #0-0] AssertionError: expected 'DELIVER TO:' to i
nclude 'Hungry Person'
[phantomjs #0-0]     at Context.it (/home/ccuser/workspace/w
hy-test-automated/test/features/user-visits-index-test.js:41
:14)
[phantomjs #0-0]
[phantomjs #0-0] 2) to post an order displays the selected c
ake type (Behavior 4):
[phantomjs #0-0] expected 'CAKE:' to include 'Whole Wheat'
[phantomjs #0-0] AssertionError: expected 'CAKE:' to include
 'Whole Wheat'
[phantomjs #0-0]     at Context.it (/home/ccuser/workspace/w
hy-test-automated/test/features/user-visits-index-test.js:55
:14)
[phantomjs #0-0]
[phantomjs #0-0] 3) to post an order displays multiple filli
ngs (Behavior 5):
[phantomjs #0-0] expected 'FILLINGS: Strawberries' to includ
e 'Banana'
[phantomjs #0-0] AssertionError: expected 'FILLINGS: Strawbe
rries' to include 'Banana'
[phantomjs #0-0]     at Context.it (/home/ccuser/workspace/w
hy-test-automated/test/features/user-visits-index-test.js:72
:14)
[phantomjs #0-0]
[phantomjs #0-0] 4) to post an order displays the number equ
ivalent to the stack size (Behavior 6):
[phantomjs #0-0] expected 'PANCAKE COUNT: 0' to include '7'
[phantomjs #0-0] AssertionError: expected 'PANCAKE COUNT: 0'
```

```
[phantomjs #0-0] expected 'PANCAKE COUNT: 0' to include '7'
[phantomjs #0-0] AssertionError: expected 'PANCAKE COUNT: 0'
 to include '7'
[phantomjs #0-0]     at Context.it (/home/ccuser/workspace/w
hy-test-automated/test/features/user-visits-index-test.js:87
:14)
[phantomjs #0-0]

npm ERR! Test failed.  See above for more details.
npm WARN Local package.json exists, but node_modules missing
, did you mean to install?
$
```

수동적인 테스팅보다 훨씬 빠르고 간편하다.

- **The Test Suite**

너의 웹앱과 마찬가지로 테스트 또한 코드로 작성되었다. 너의 앱의 코드는 implementation code(구현 코드), 테스트를 위한 코드를 test code 라고 한다. 웹 애플리케이션의 테스트들의 일련의 모음을 test suite(테스터 슈트) 라고 한다. 지난 챕터에서 npm test 커맨드를 통해 test suite 를 실행했다. 테스트 슈트는 앱 테스트를 위한 모든 테스트들을 포함하고 있다.

테스트 코드는 구현 코드와 비슷하게 포함되고, 구성된다. 종종 테스트 코드의 변경은 구현 코드의 변경과 관련되거나 그 반대의 경우도 있습니다. 두 상황 모두 그들이 같은 장소에 저장될 때 유지하기가 더 쉽다.

예를 들어 만약 구현 코드가 index.js 안에 쓰여져 있다면, 그에 상응하는 테스트코드는 index-test.js 에 쓰여질 것이다.

- **Tests As Documentation**

우리가 해당 앱을 누군가에게 설명한다면 어떻게 해야할까? 모든 파일의 구현코드를 읽을 수 있고, 또는 documentation 을 읽을 수 있다.

문서는 어떻게 작동하고 사용하는 지를 설명하는 구현코드로 부터 분리된 내용이다. 그것은 구현코드보다 훨씬 더 간결한 요약과 설명을 제공한다.

문서에는 많은 양식이 있을 수 있다. 일반적인 텍스트, 다이어그램, 테스트로.. 문서로서의 테스트는 많은 다른 형식이 할 수 없는 것을 제공한다. 예를 들어, 사람이 읽을 수 있는 텍스트와 기계가 실행시킬 수 있는 코드를 동시에 제공한다. 아래 name 기능에 대한 테스트를 설명하는 문서화된 테스트 예시가 있다.

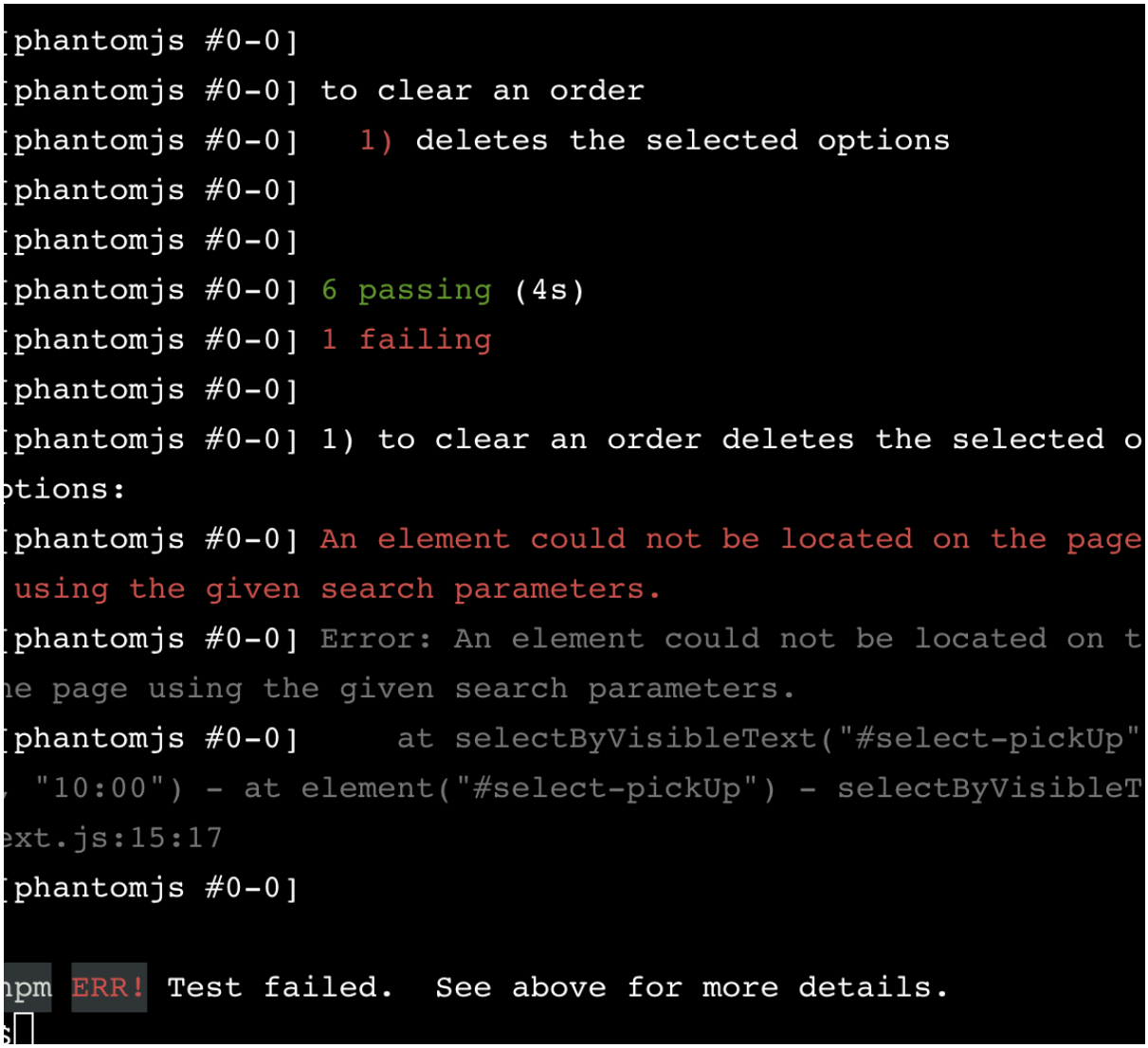
```
it('accepts the customer name', () => {
  const name = 'Hungry Person';

  browser.url('/');
  browser.setValue('#name', name);
  browser.click('#submit-order');
  browser.url('/');

  assert.include(browser.getText('#deliver-to'), name);
});
```

우리는 일반적인 텍스트인 'accepts the customer name' 과 그 아래 컴퓨터가 읽을 수 있는 코드를 동시에 볼 수 있다.

실습 예시

```
90
91      // Add "clear order" test here
92      describe('to clear an order', () => {
93      it('deletes the selected options', () => {
94          const name = 'Indecisive Person';
95          const time = '10:00';
96
97          browser.url('/');
98          browser.setValue('#name', name);
99          browser.selectByVisibleText('#select-pickUp', time)
100         browser.click('#submit-order');
101         browser.click('#clear-order');
102         browser.url('/');
103
104         assert.equal(browser.getText('#deliver-to'), '');
105         assert.equal(browser.getText('#cake-type'), '');
106         assert.equal(browser.getText('#fillings'), '');
107         assert.equal(browser.getText('#size'), '');
108         assert.equal(browser.getText('#pickUp'), '');
109     });
110 });
111
112 });
113
```

테스트 js 에 새로운 테스트 항목을 넣어줬다.



npm test 결과 fail 이 떴다.

어떤 문제인지 수동적인 테스팅을 해보자. npm start 을 입력후 브라우저를 리로딩해보자.

clear button 에 대한 테스트인데 아직 페이지에 clear button 이 구현되지 않았기 때문에 해당 테스트가 fail 이 뜬것이다.

- **Regression (회귀)**

새로운 테스트를 만족하기 위해 위에서 오류가 난 새로운 "clear order"에 대한 버튼을 구현했다.

새로운 기능을 제품에 추가할 때, 프로그램이 깨질 확률이 있다. 만약 그 깨짐(에러)가 먼저 개발된 기능 즉, 코드에서 일어난다면 그것을 regression 이라고 부른다. 이전에 개발되고 테스트된 기능이 멈출때, 우리는 이것을 functionality regressed(회귀된 기능, 퇴보된 기능)라고 부른다.

자동화 테스트 수행을 실행하는 것은 빠르고 반복성이 있다. 이것은 너가 프로그램을 변경할 때마다 전에 만든 기능이 여전히 돌아가는지 확인하기 위해 테스트를 실행할 수 있다는 것이다. 만약 전의 기능들이 퇴행했다면, 그 테스트 결과는 너에게 퇴행에 대한 정보를 알려줄 것이다. 너는 에러가 난 장소로 찾아가서 코드를 올바르게 고치면 될 것이다.

- **Review**

배운 것들

⇒ 자동화 테스팅은 제품이 기대했던대로 실행한다는 확신을 심어줄 수 있다.

⇒ 문서화까지 개선시킨다.

⇒ 회귀의 가능성을 줄인다.

⇒ 어디에 그리고 왜 테스트 코드가 구현코드와 나란히 저장되는 지 배웠다.

⇒ 테스팅의 이점에 대해 소통하기 위한 용어들에 대해 배웠다. (메뉴얼 테스팅, 자동화 테스팅, test suite, bug, documentation, regression)

⇒ npm start, npm test 에 대해서 배웠다.

Article 1. Test-Driven Development (TDD)

테스트 주도 개발은 매우 짧은 개발 사이클을 반복하는 소프트웨어 개발 프로세스 중 하나이다. 개발자는 먼저 요구사항을 검증하는 자동화된 테스트 케이스를 작성한다. 그런 후에, 그 테스트 케이스를 통과힉 위한 최소한의 코드를 생성하낟. 마지막으로 작성한 코드를 표준에

맞도록 리팩토링한다. 이 기법을 개발했거나 재발견한 것으로 인정되는 Kent Beck 은 2003년에 TDD가 단순한 설계를 장려하고 자신감을 불어넣어준다고 말했다.

TDD 관련 블로그 글

1. https://wooaoe.tistory.com/33
2. https://gmlwjd9405.github.io/2018/06/03/agile-tdd.html

Article 2. Unit Testing (단위 테스트)

모듈을 테스트하는 단계이다.

시나공 교재 참고

Article 3. Integration Testing (통합 테스트)

단위 테스트가 끝난 소프트웨어를 결합해 가며 테스트하는 방법이다. 단위 테스트가 끝난 ㅗ듈들을 좀 더 큰 단위의 집합으로 통합 구성한 후, 통합 시험 계획에 따라 테스트를 수행한다. 통합 테스트를 통과한 모듈집합은 시스템 검사 단계의 테스트 대상으로서 넘어가게 된다.

시나공 교재 참고

오늘의 단어

- vice versa : 거꾸로 반대로
- Regression : 퇴행, 퇴보 ; 회귀