2025 겨울방학 알고리즘 스터디

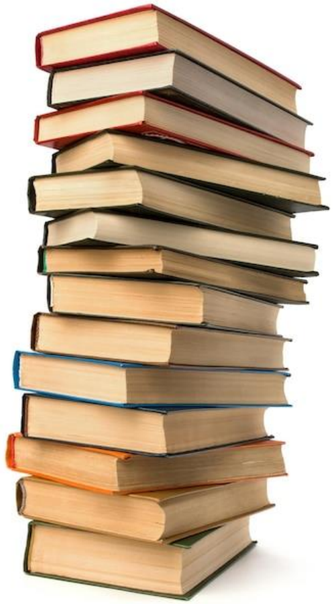
스택

컴퓨터 공학과 20230546 서보경

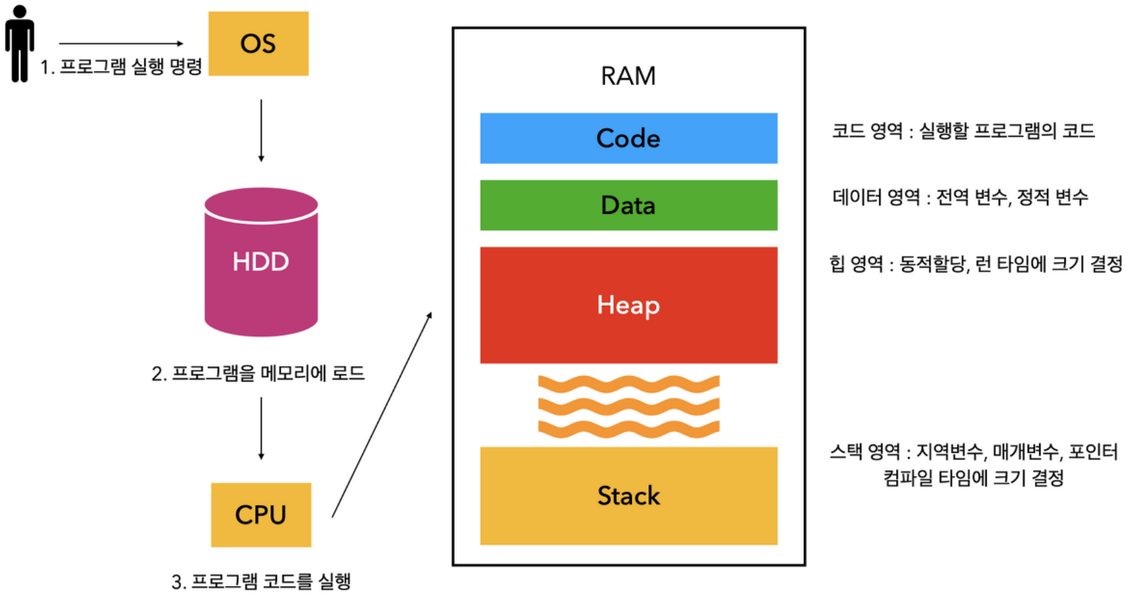
목차

스택이란?

스택(Stack) 은 후입선출(Last In, First Out, LIFO) 원칙을 따르는 선형 자료구조이다.
후입 선출이란 가장 나중에 추가된 데이터가 가장 먼저 제거되는 구조를 의미한다.
스택은 한쪽 끝에서만 데이터를 삽입하거나 제거할 수 있다.

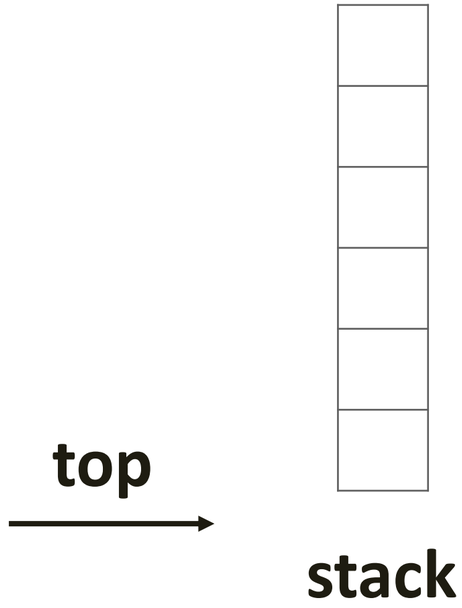


책을 아래부터 쌓는다고 생각하자.



메모리 구조에서의 스택의 쓰임.
지역변수 관리를 위해 주로 쓰인다.

스택의 사용법

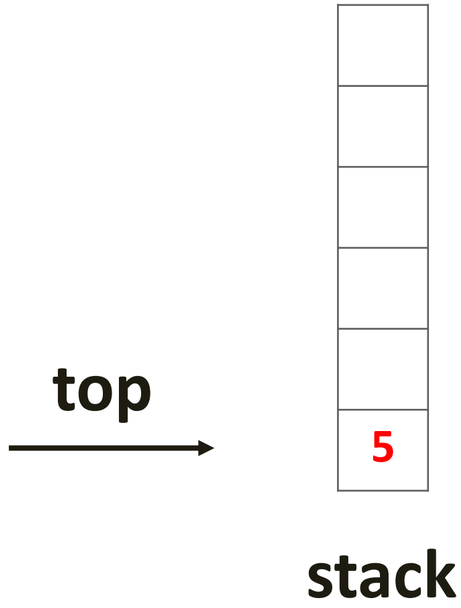


스택은 주로 배열을 통해 구현하게 된다.

이때, 최상단 원소를 가리키는 포인터도 따로 두는데 이 변수의 이름을 top이라고 하겠다.

배열은 처음은 비어 있는 상태이다. top은 -1로 초기화 한다.
최상단에 원소가 없는 것을 의미하기 위해서이다.

스택의 사용법



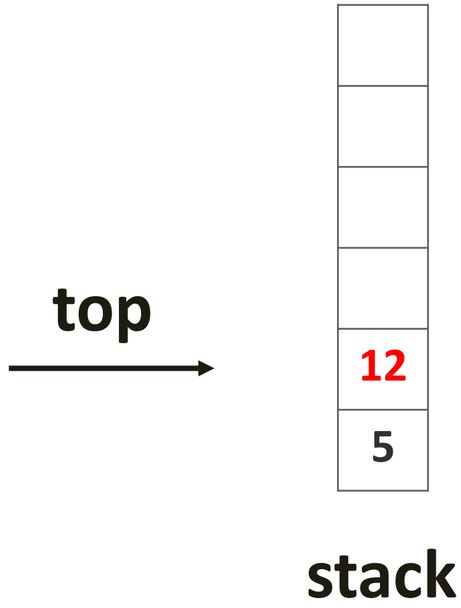
query : push 5

스택에 5를 넣는 연산이다.

먼저, 현재 포인터는 -1이다. 이는 스택의 최대 용량의 위치인 5 보다 아래이므로 가용공간이 존재한다는 뜻이다.

포인터를 한 칸 올려주고 그 위치에 원소를 집어 넣는다.

스택의 사용법



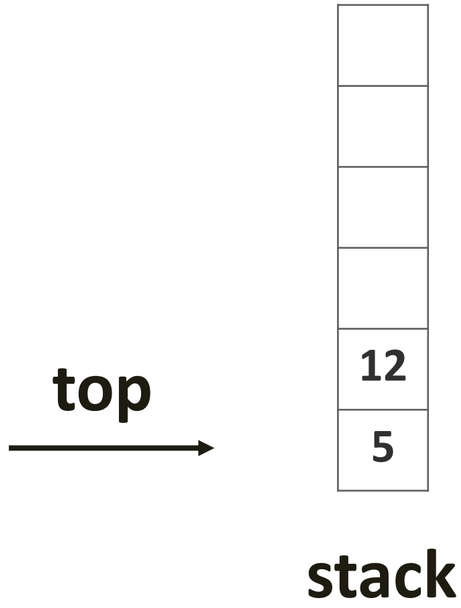
query : push 12

스택에 12을 넣는 연산이다.

먼저, 현재 포인터는 0이다. 이는 스택의 최대 용량의 위치인 5 보다 아래이므로 가용공간이 존재한다는 뜻이다.

포인터를 한 칸 올려주고 그 위치에 원소를 집어 넣는다.

스택의 사용법



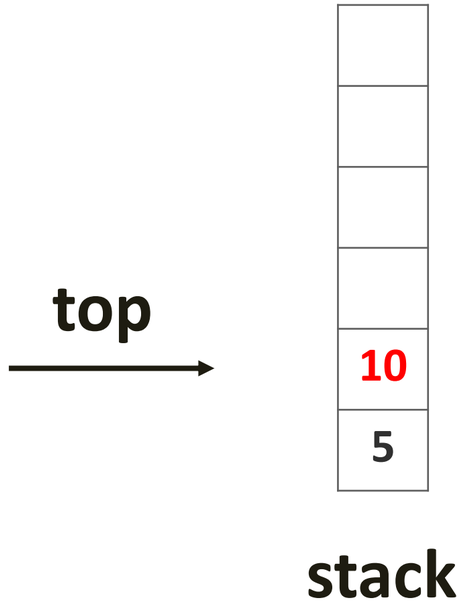
query : pop

스택에서 최상단 원소를 지우는 연산이다.

먼저, 현재 포인터는 1이다. 포인터가 -1보다 크므로 스택에 원소가 있는 것을 알 수 있다.

포인터를 한 칸 내려준다. (삭제 처리)

스택의 사용법



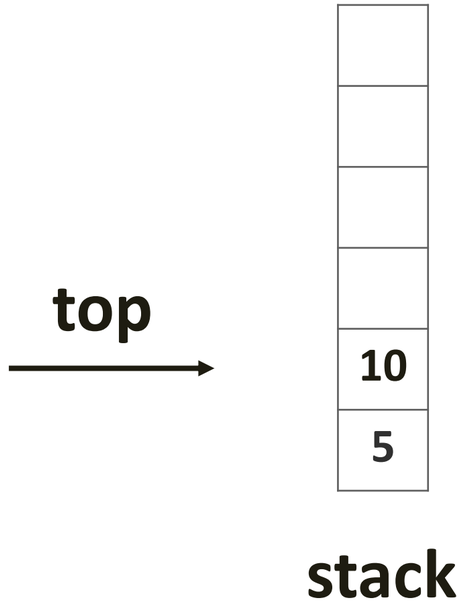
query : push 10

스택에 10을 넣는 연산이다.

먼저, 현재 포인터는 1이다. 이는 스택의 최대 용량의 위치인 5 보다 아래이므로 가용공간이 존재한다는 뜻이다.

포인터를 한 칸 올려주고 그 위치에 원소를 집어 넣는다. (덮어 쓰기)

스택의 사용법



스택 사용할 때 주의할 점

항상 최상단 원소에만 관심이 있기 때문에 포인터(top)를 적절히 옮겨야 한다.

포인터가 -1을 가리키면 스택이 비었다는 걸 의미하고, 사용자 지정 가용 용량크기 -1을 넘어가면 스택이 가득 찼다는 걸 의미한다. (C언어에선 배열 넉넉히 잡는게 좋음)

삽입, 삭제 연산은 실제로 값을 지우는 것이 아니라 포인터만 이동시키는 것이다!

스택의 활용 – 후위 표기식

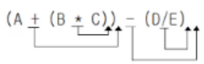
수식은 일반적으로 3가지 표기법으로 표현할 수 있다. 연산자가 피연산자 가운데 위치하는 중위 표기법(일반적으로 우리가 쓰는 방법이다), 연산자가 피연산자 앞에 위치하는 전위 표기법(prefix notation), 연산자가 피연산자 뒤에 위치하는 후위 표기법(postfix notation)이 그것이다. 예를 들어 중위 표기법으로 표현된 `a+b` 는 전위 표기법으로는 `+ab` 이고, 후위 표기법으로는 `ab+` 가 된다.

이 문제에서 우리가 다룰 표기법은 후위 표기법이다. 후위 표기법은 위에서 말한 법과 같이 연산자가 피연산자 뒤에 위치하는 방법이다. 이 방법의 장점은 다음과 같다. 우리가 흔히 쓰는 중위 표기식 같은 경우에는 덧셈과 곱셈의 우선순위에 차이가 있어 왼쪽부터 차례로 계산할 수 없지만 후위 표기식을 사용하면 순서를 적절히 조절하여 순서를 정해줄 수 있다. 또한 같은 방법으로 괄호 등도 필요 없게 된다. 예를 들어 `a+b*c` 를 후위 표기식으로 바꾸면 `abc*+` 가 된다.

중위 표기식을 후위 표기식으로 바꾸는 방법을 간단히 설명하면 이렇다. 우선 주어진 중위 표기식을 연산자의 우선순위에 따라 괄호로 묶어준다. 그런 다음에 괄호 안의 연산자를 괄호의 오른쪽으로 옮겨주면 된다.

예를 들어 `a+b*c` 는 `(a+(b*c))` 의 식과 같게 된다. 그 다음에 안에 있는 괄호의 연산자 `*` 를 괄호 밖으로 꺼내게 되면 `(a+bc*)` 가 된다. 마지막으로 또 `+` 를 괄호의 오른쪽으로 고치면 `abc*+` 가 되게 된다.

다른 예를 들어 그림으로 표현하면 `A+B*C-D/E` 를 완전하게 괄호로 묶고 연산자를 이동시킬 장소를 표시하면 다음과 같이 된다.



결과: `ABC*+DE/-`

이러한 사실을 알고 중위 표기식이 주어졌을 때 후위 표기식으로 고치는 프로그램을 작성하시오

입력

첫째 줄에 중위 표기식이 주어진다. 단 이 수식의 피연산자는 알파벳 대문자로 이루어지며 수식에서 한 번씩만 등장한다. 그리고 `-A+B` 와 같이 `-` 가 가장 앞에 오거나 `AB` 와 같이 `*` 가 생략되는 등의 수식은 주어지지 않는다. 표기식은 알파벳 대문자와 `+`, `-`, `*`, `/`, `(`, `)` 로만 이루어져 있으며, 길이는 100을 넘지 않는다.

출력

첫째 줄에 후위 표기식으로 바뀐 식을 출력하시오

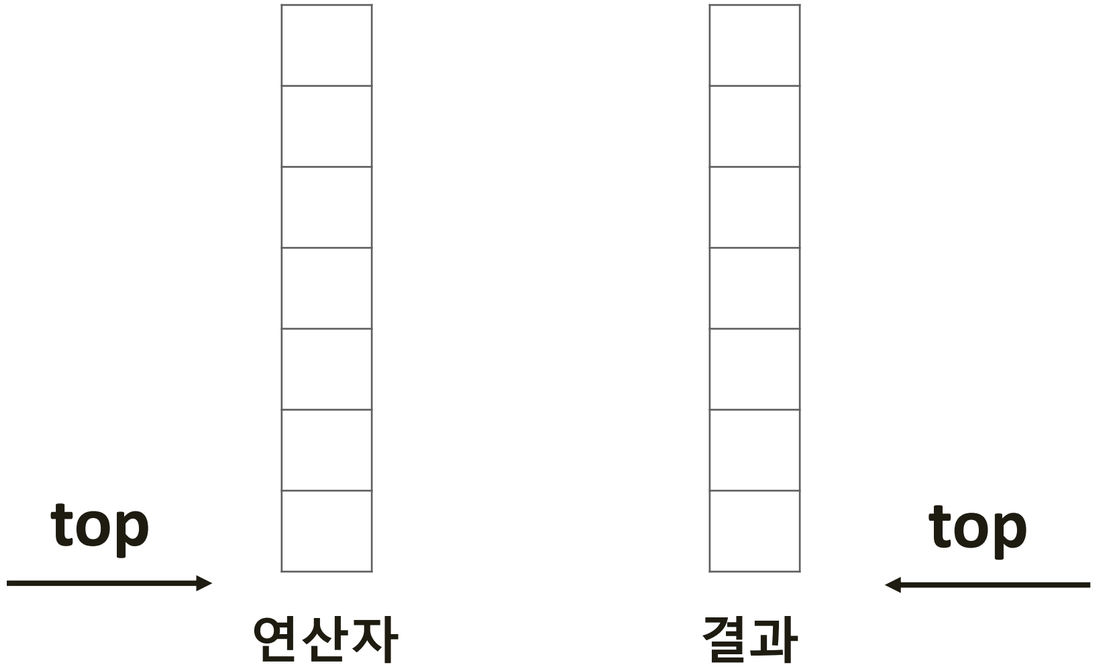
문자열을 한번에 입력 받고 바로 후위 표기식으로 만드는건 쉽지 않아 보이는데.. 우선순위도 있고... 앞에서 부터 하나 하나씩 읽어나가면서 만들어 보는건 어떨까?

그런데, 연산자랑 피연산자를 하나의 배열(스택)에 저장하면 관리하기가 어려울거 같은데 어떻게 해야하지..?

➔ 연산자 스택과 결과 스택을 따로 만들어서 관리를 하면 되겠구나! 또, 스택의 구조를 이용하여 왼쪽부터 오른쪽으로 가면서 순차적으로 검사도 할 수 있고!

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



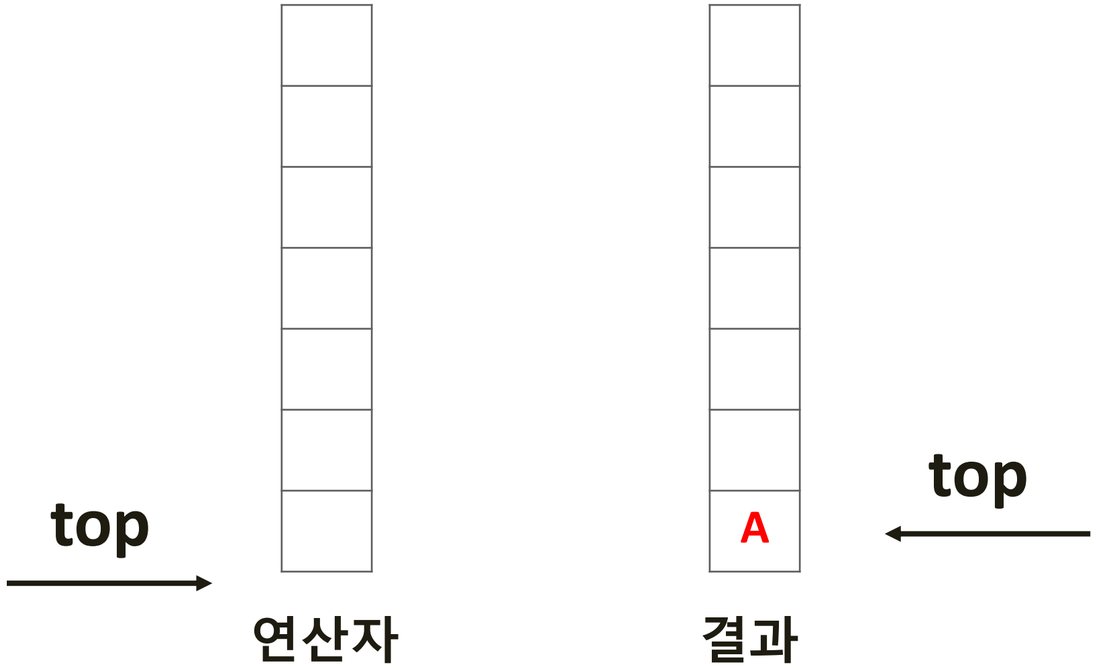
후위 표기식 구현에 앞서, 먼저 연산자와 피연산자를 담아줄 스택 두개를 선언한다.

또한, 각 스택을 조종할 포인터 변수도 각각 한개씩 선언한다.

후위 표기법은 연산자가 피연산자 뒤에 위치하는 방법이고, 연산자 우선순위를 따른다.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 : A

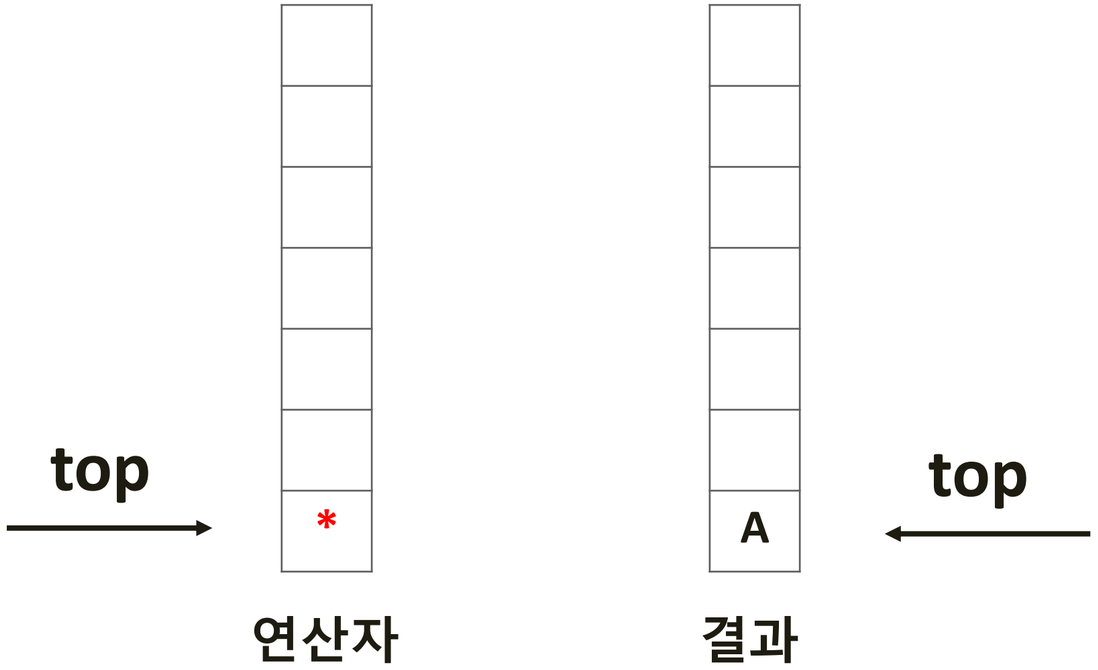
현재 문자를 확인한다.

현재 문자는 A이므로 피연산자 이다.

고로, 결과 스택에 push. (연산자가 피연산자 뒤에 위치하여, 피연산자를 우선 처리)

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 : *

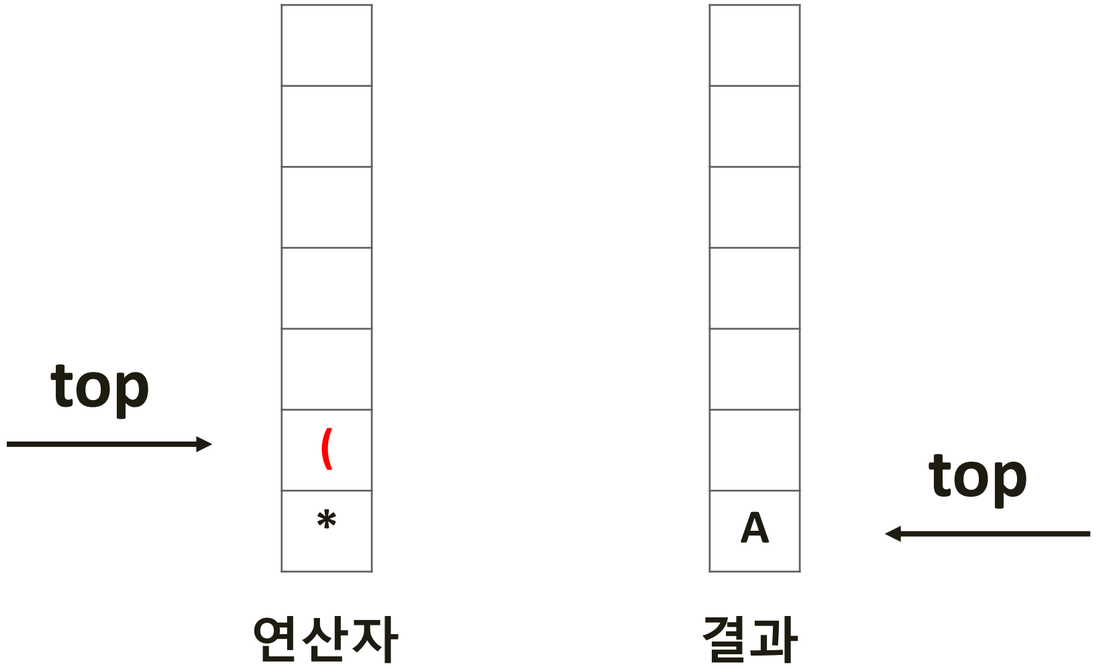
현재 문자를 확인한다.

현재 문자는 *이므로 연산자 이다.

연산자 스택이 비었으므로 아무것도 하지말고 연산자 스택에 push.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 : (

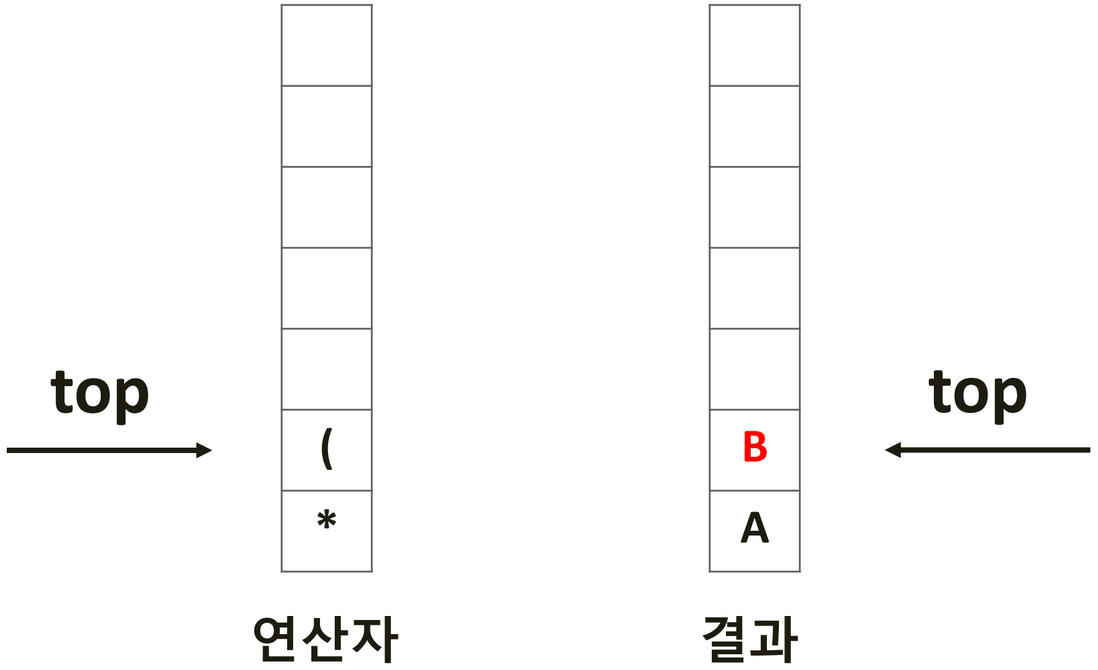
현재 문자를 확인한다.

현재 문자는 (이므로 연산자 이다.

연산자 스택이 비지 않았지만, 여는 괄호 연산자는 무조건 연산자 스택 push.
(새로운 우선순위 분기)

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 : B

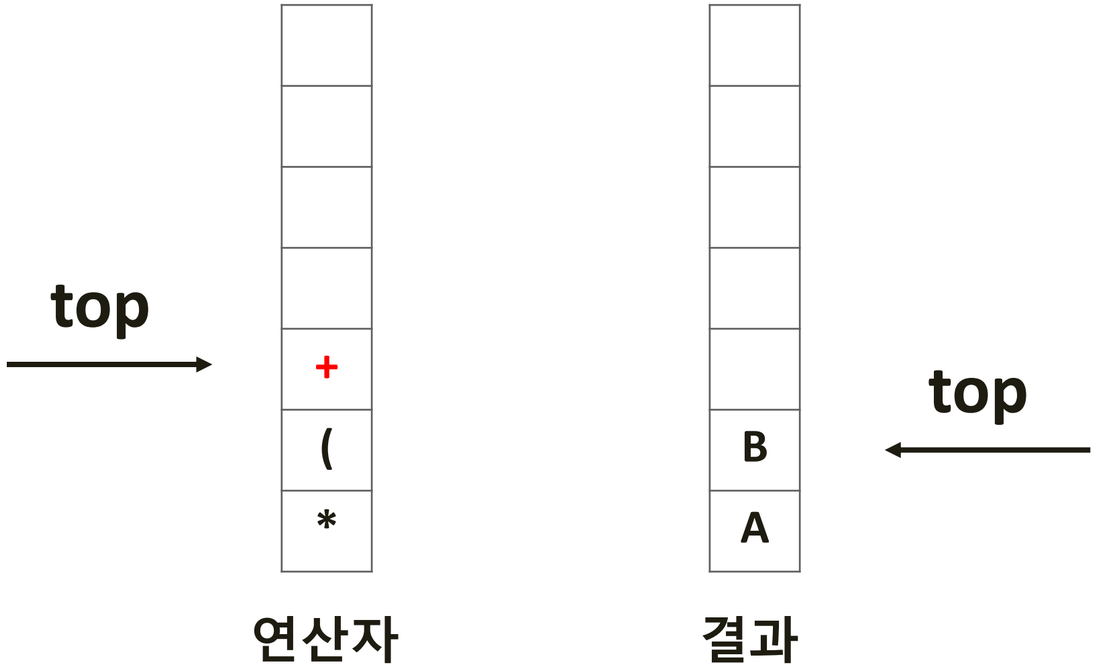
현재 문자를 확인한다.

현재 문자는 B이므로 피연산자 이다.

고로, 결과 스택에 push.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 : +

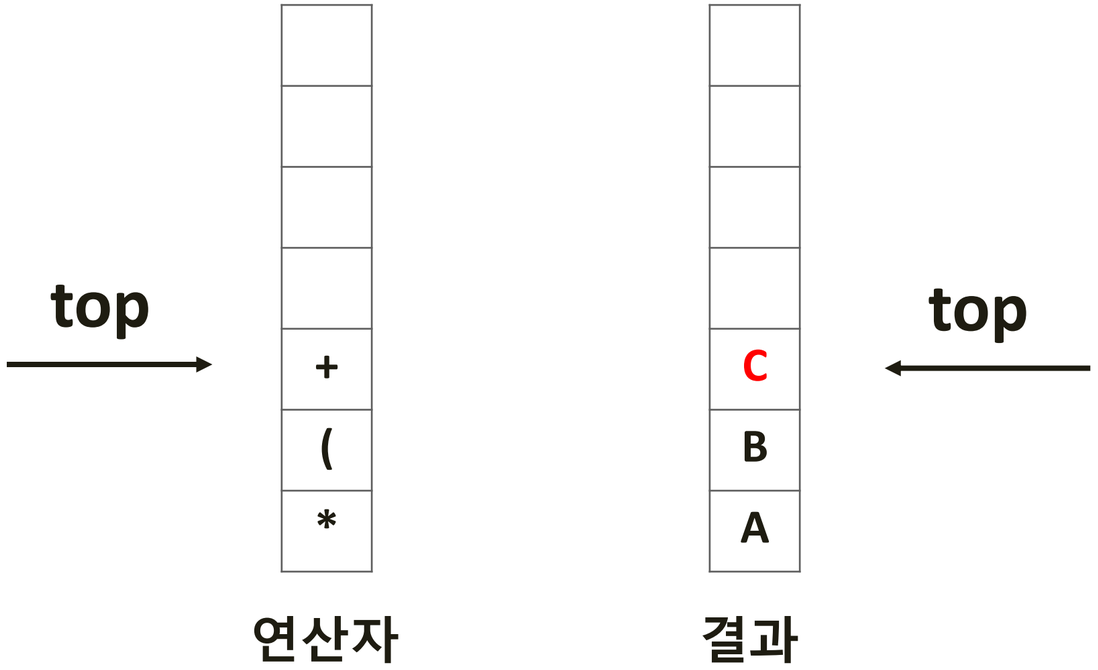
현재 문자를 확인한다.

현재 문자는 +이므로 연산자 이다.

먼저, stack의 top을 확인한다. 만약, * , / , + , - 중 하나라도 있었으면 먼저 방출해야 했겠지만, 여는 괄호 쌍이기 때문에 그대로 연산자 스택에 push (우선순위 + 먼저 온 순 고려)

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 : C

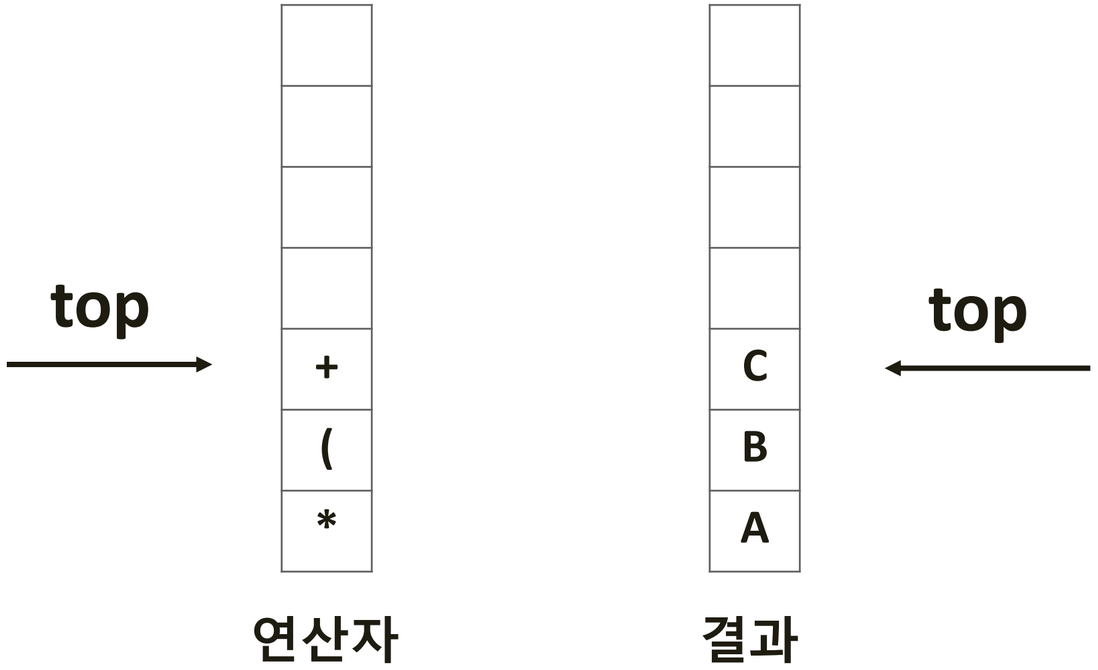
현재 문자를 확인한다.

현재 문자는 c이므로 피연산자 이다.

고로, 결과 스택에 push.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 :)

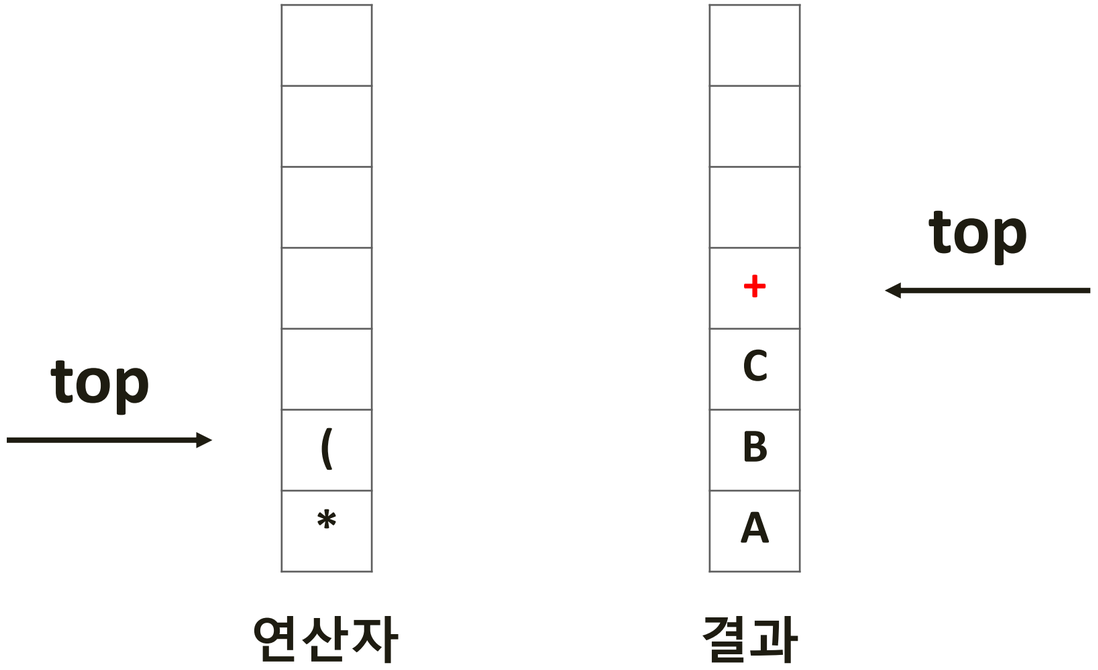
현재 문자를 확인한다.

현재 문자는)이므로 연산자 이다.

닫는 괄호가 나왔으므로 연산자 스택이 비거나, 여는 괄호 쌍이 나올 때 까지 계속 pop 작업을 해주어야 한다.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 :) | stack[top] = +

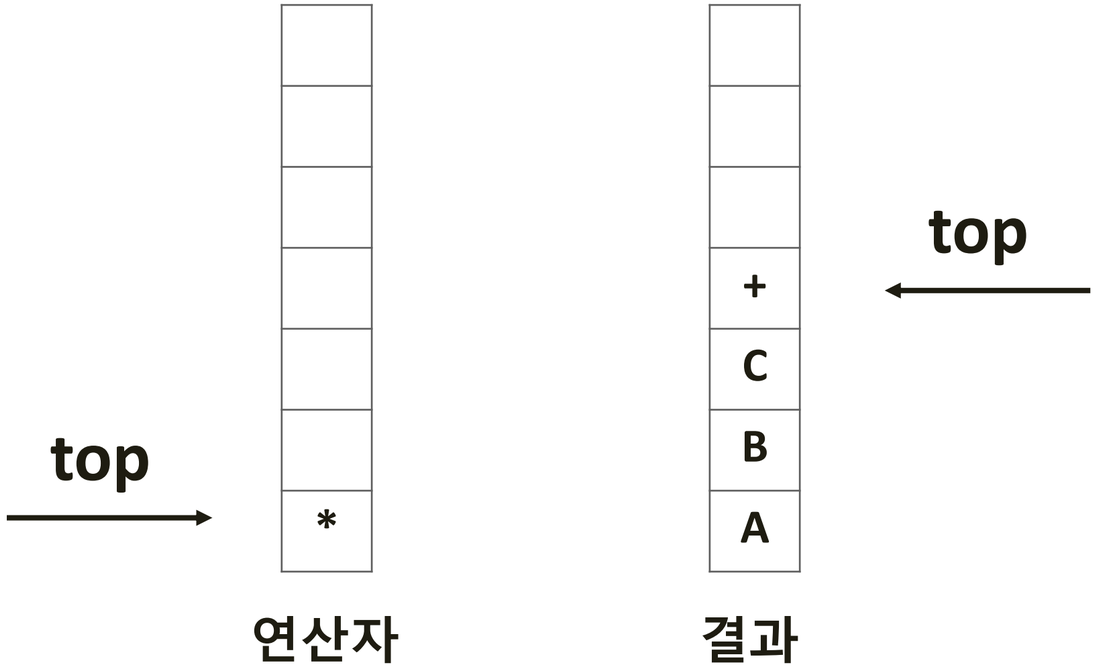
연산자 stack의 top을 확인한다.

top이 -1도 아니고 닫는 괄호 쌍도 아니다.

고로 그 연산자(+)을 결과 스택에 push. 그 후, top을 감소시킨다.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 :) | stack[top] = (

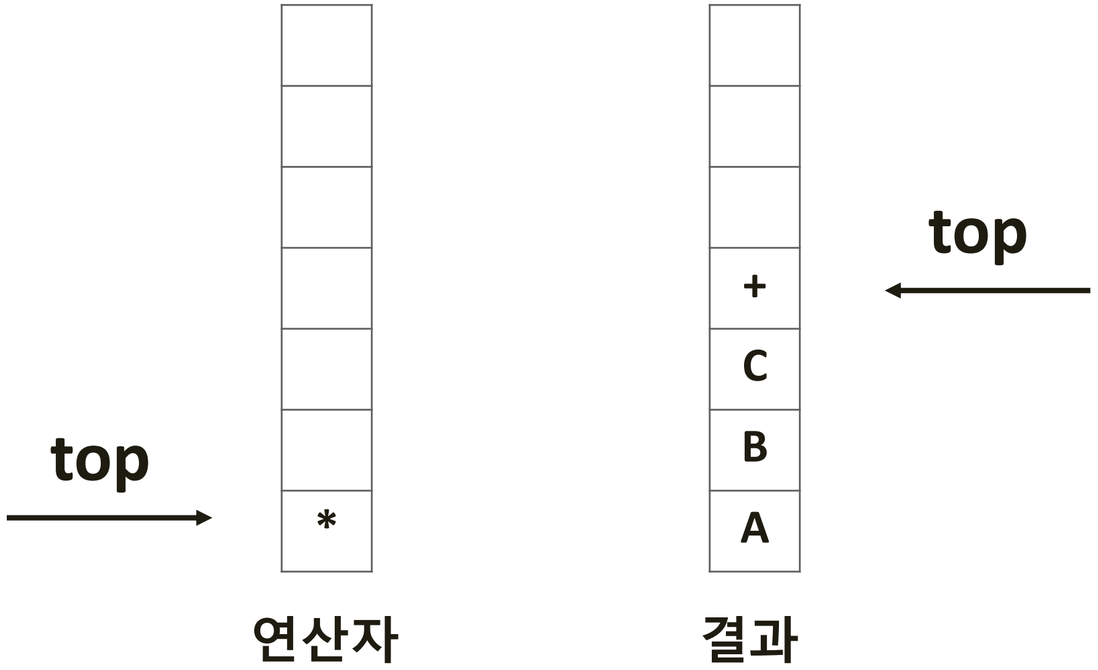
연산자 stack의 top을 확인한다.

top이 -1이 아니지만, 닫는 괄호 쌍이다.

후위 표기식은 괄호가 필요 없으므로 연산자 스택에서 pop 작업을 해주고 아무것도 하지 않는다.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 : /

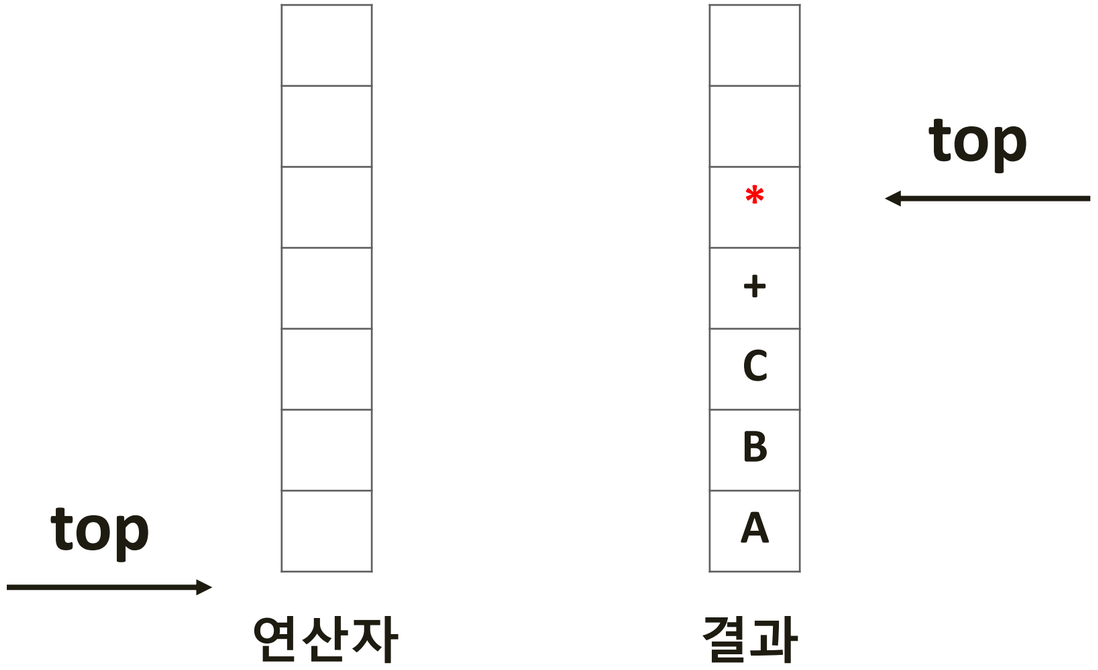
현재 문자를 확인한다.

현재 문자는 / 이므로 연산자 이다.

나누기 연산자는 우선순위가 제일 높으므로 (괄호 연산자 제외) 스택이 비거나, 여는 괄호를 만나거나, 혹은 우선순위가 낮은(+,-)연산자를 만나기 전까지 pop을 해주어야 한다.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 : / | stack[top] = *

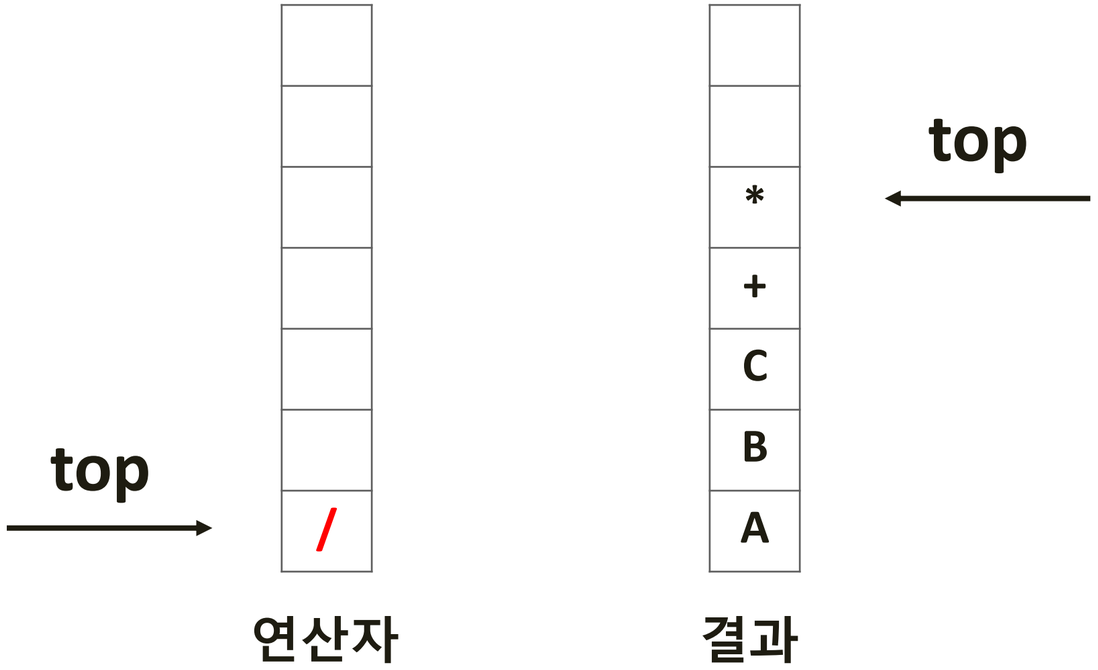
연산자 stack의 top을 확인한다.

top이 -1이 아니지만, * 연산자가 있다.

곱하기 연산자는 나누기 연산자와 우선순위가 같지만, 스택에 이미 들어있었다는 건 먼저 제시되었다는 뜻이므로 pop 하여 결과 스택에 push.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 : / | top = -1

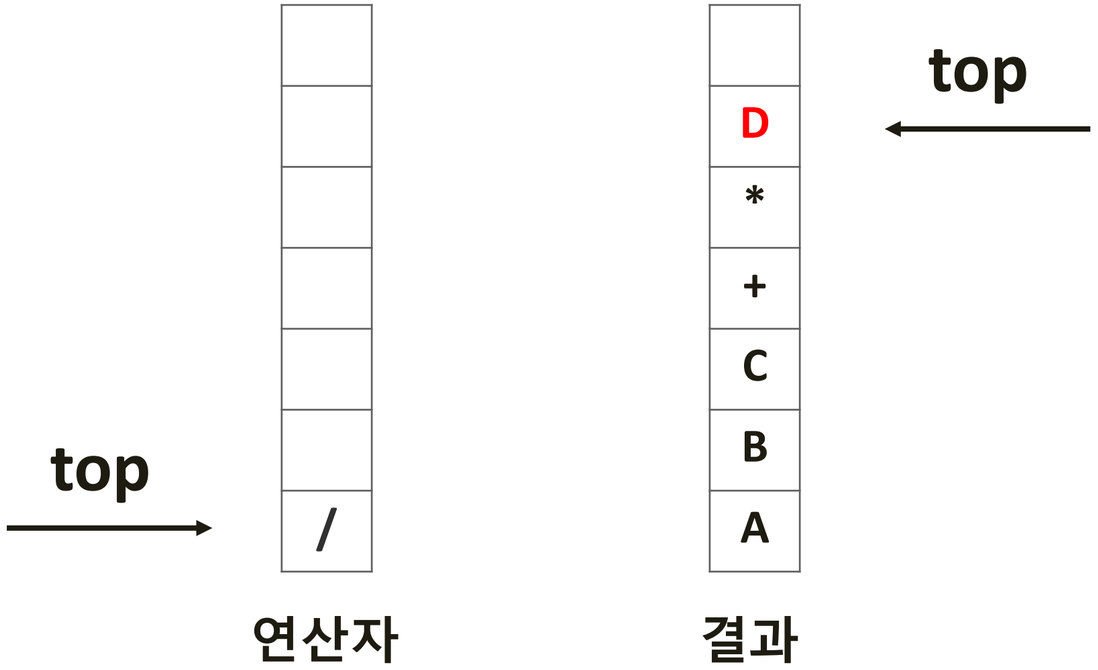
연산자 stack의 top을 확인한다.

top이 -1이므로 pop 작업을 종료한다.

그 후, 나누기 연산자를 연산자 스택에 push.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



현재 문자 : D

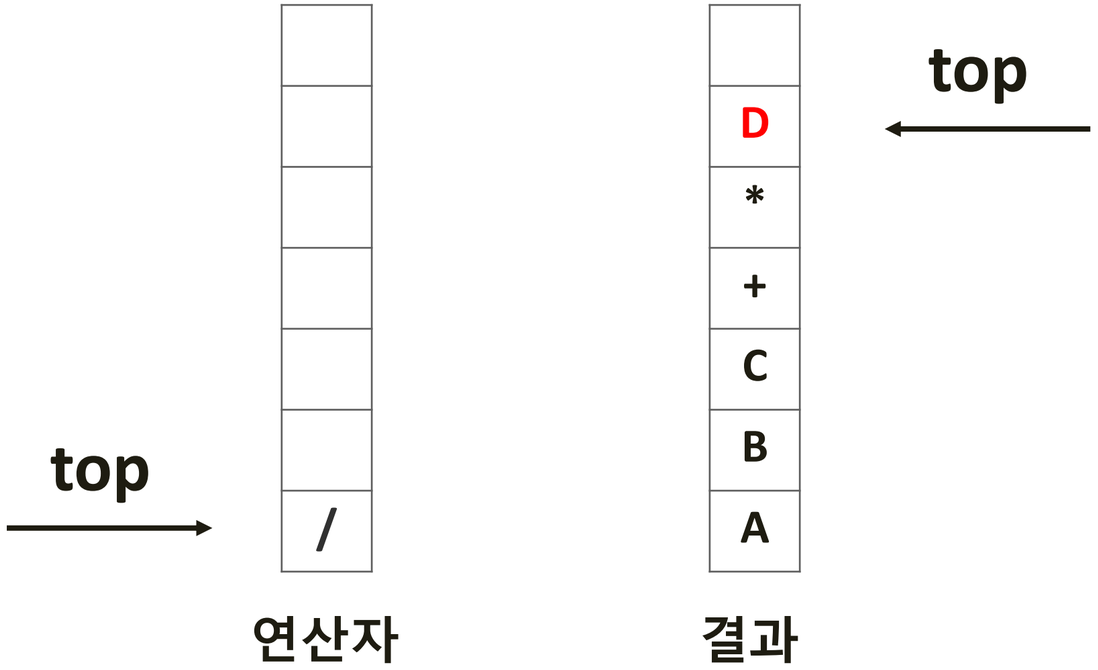
현재 문자를 확인한다.

현재 문자는 D이므로 피연산자 이다.

고로, 결과 스택에 push. 중위 표기식을 전부 다 순회하였다.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D

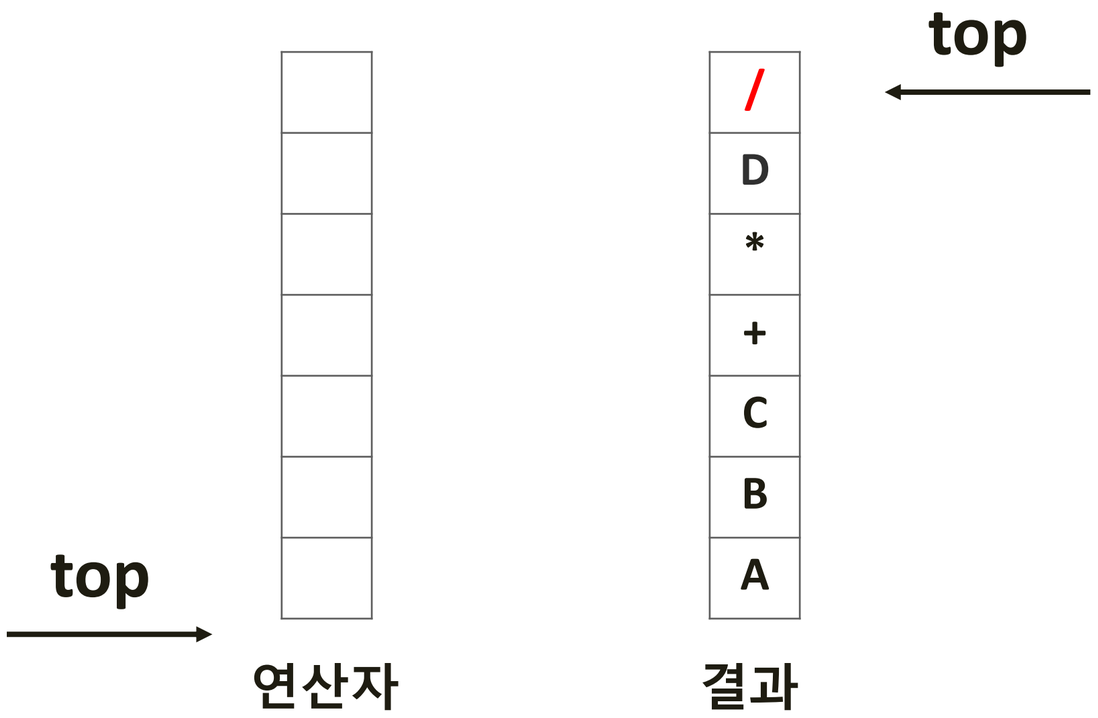


이후, 연산자 스택에 남은 연산자들을 순차적으로 pop하여 결과 스택에 넣어주면 된다.

이때, 우선순위는 신경 쓰지 않는데, 이미 연산자 스택 push 이전에 적절한 pop 작업을 하여 균형을 맞춰 주었기 때문이다.

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



stack[top] = /

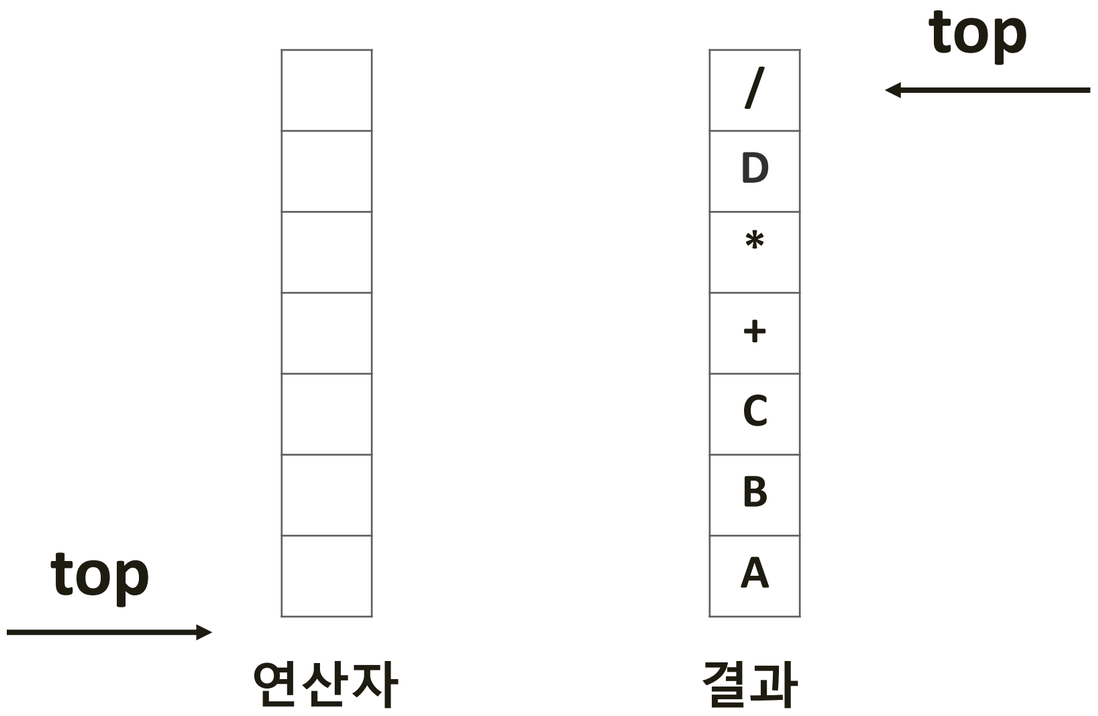
연산자 stack의 top을 확인한다.

top이 -1이 아니지만, / 연산자가 있다.

결과 스택에 push

스택의 활용 – 후위 표기식

중위 표기식 : A * (B+C) / D



top = -1

연산자 stack의 top을 확인한다.

top이 -1이므로 연산자 스택이 비었다.

고로 후위 표기식 결과는 “ABC+*D/” 이다.

스택의 활용 – 단조성

KOI 통신연구소는 레이저를 이용한 새로운 비밀 통신 시스템 개발을 위한 실험을 하고 있다. 실험을 위하여 일직선 위에 N개의 높이가 서로 다른 탑을 수평 직선의 왼쪽부터 오른쪽 방향으로 차례로 세우고, 각 탑의 꼭대기에 레이저 송신기를 설치하였다. 모든 탑의 레이저 송신기는 레이저 신호를 지표면과 평행하게 수평 직선의 왼쪽 방향으로 발사하고, 탑의 기둥 모두에는 레이저 신호를 수신하는 장치가 설치되어 있다. 하나의 탑에서 발사된 레이저 신호는 가장 먼저 만나는 단 하나의 탑에서만 수신이 가능하다.

예를 들어 높이가 6, 9, 5, 7, 4인 다섯 개의 탑이 수평 직선에 일렬로 서 있고, 모든 탑에서는 주어진 탑 순서의 반대 방향(왼쪽 방향)으로 동시에 레이저 신호를 발사한다고 하자. 그러면, 높이가 4인 다섯 번째 탑에서 발사한 레이저 신호는 높이가 7인 네 번째 탑이 수신을 하고, 높이가 7인 네 번째 탑의 신호는 높이가 9인 두 번째 탑이, 높이가 5인 세 번째 탑의 신호도 높이가 9인 두 번째 탑이 수신을 한다. 높이가 9인 두 번째 탑과 높이가 6인 첫 번째 탑이 보낸 레이저 신호는 어떤 탑에서도 수신을 하지 못한다.

탑들의 개수 N과 탑들의 높이가 주어질 때, 각각의 탑에서 발사한 레이저 신호를 어느 탑에서 수신하는지를 알아내는 프로그램을 작성하라.

아까랑은 조금 다른 패턴 이지만...
왼쪽에서 오른쪽으로 훑으면서 판단하면 될거같은데?

-> 스택을 사용하자!

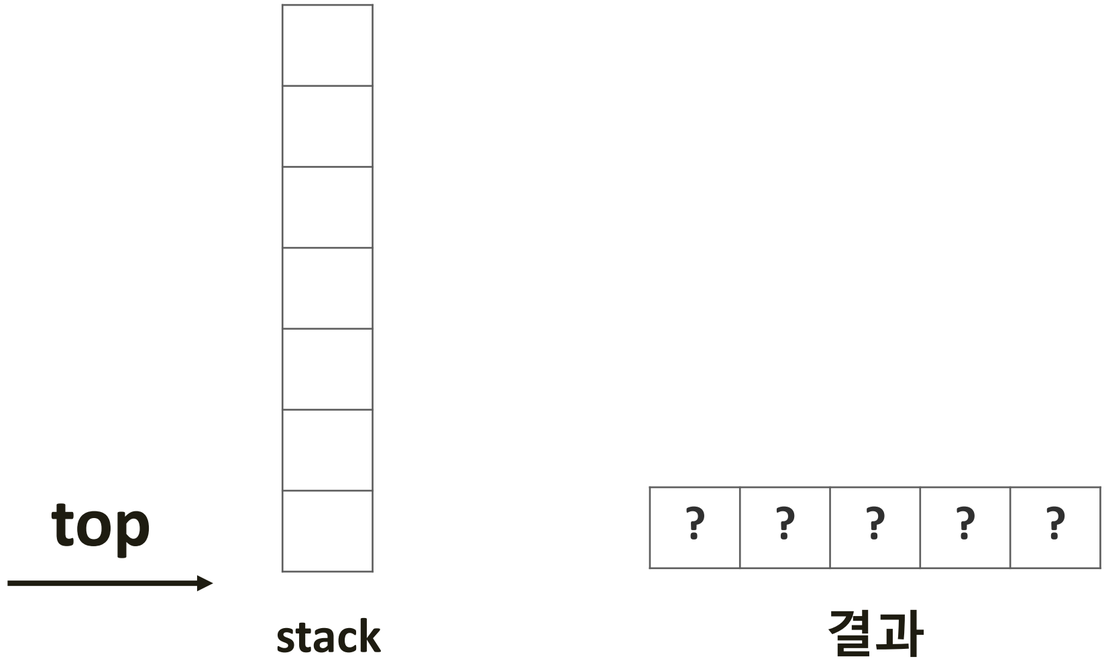
단조성이란?

함수나 수열등에서 값이 어떤 방향으로 변하는 경향성이 일정하게 유지되는 성질을 말한다.

단조 증가와 단조 감소 두 가지로 나뉘며 보통 비엄격한 단조성을 많이 사용한다. (경향성이 일치하기만 하면 값이 같아도 무관함)

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



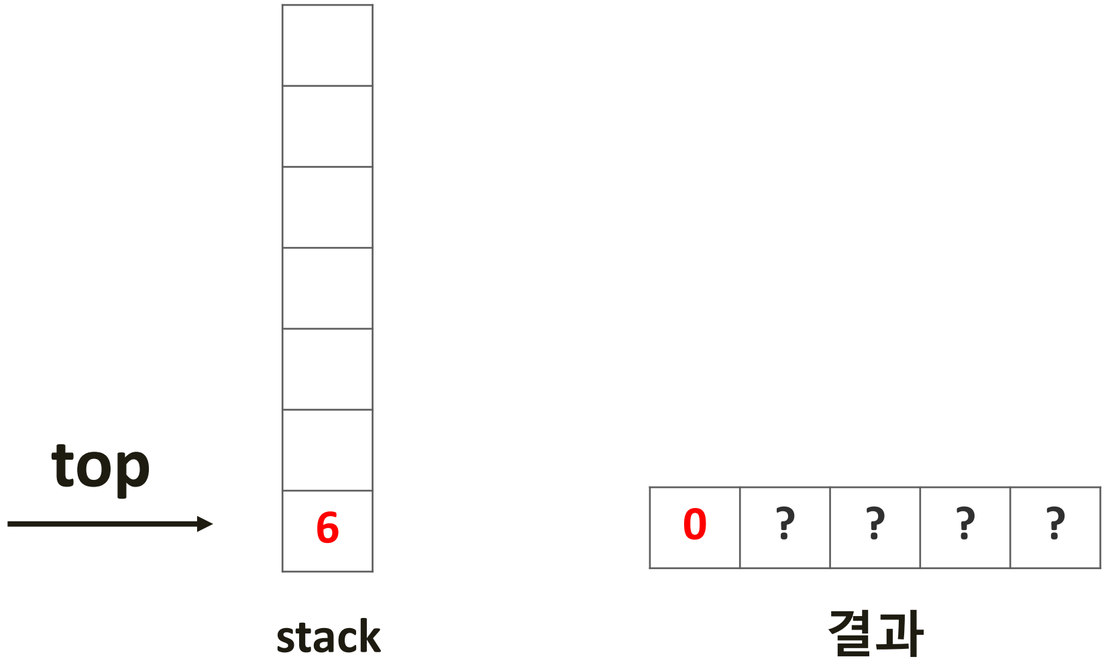
자신의 위치에서 왼쪽으로 레이저를 쏠 수 있다.

이때 레이저 수신 조건은 두가지가 있다.

1) 왼쪽에 탑이 있어야 한다. 2) 탑이 있다면, 적어도 하나 이상의 탑이 현재 탑보다 높아야 한다.

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



현재 탑의 높이 : 6

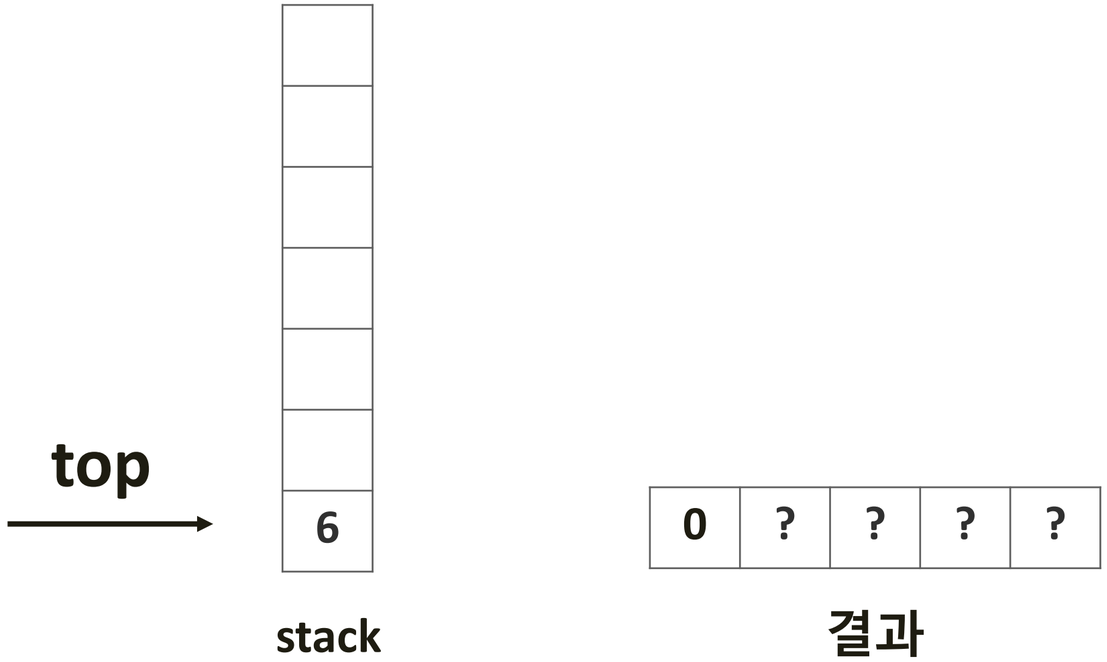
현재 탑의 높이는 6이다.

스택이 비었으므로 이 탑의 신호를 받을 수 있는 탑은 존재 하지 않는다.

고로 결과를 0으로 설정하고, 스택에 push

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



현재 탑의 높이 : 9

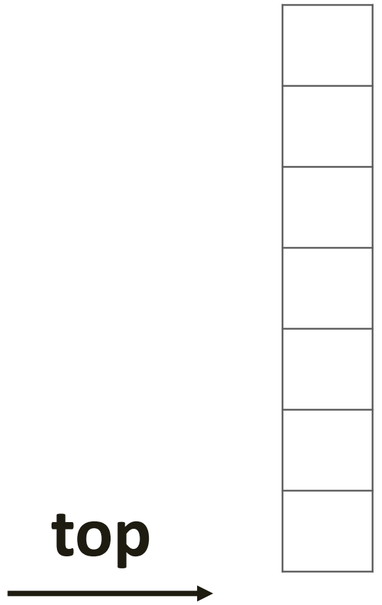
현재 탑의 높이는 9이다.

스택이 비지 않았기 때문에, 스택 내에 있는 높이 중에서 수신 가능한 놈을 찾아야한다.

스택 내에서 현재 높이 보다 작은 놈은 수신 할 수 없으므로 pop.
(만약 스택 내에서 현재 높이 보다 작으면, 뒤에 탑이 송신해도 현재 탑이 다 수신해버림)

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



0	?	?	?	?

결과

현재 탑의 높이 : 9 | stack[top] = 6

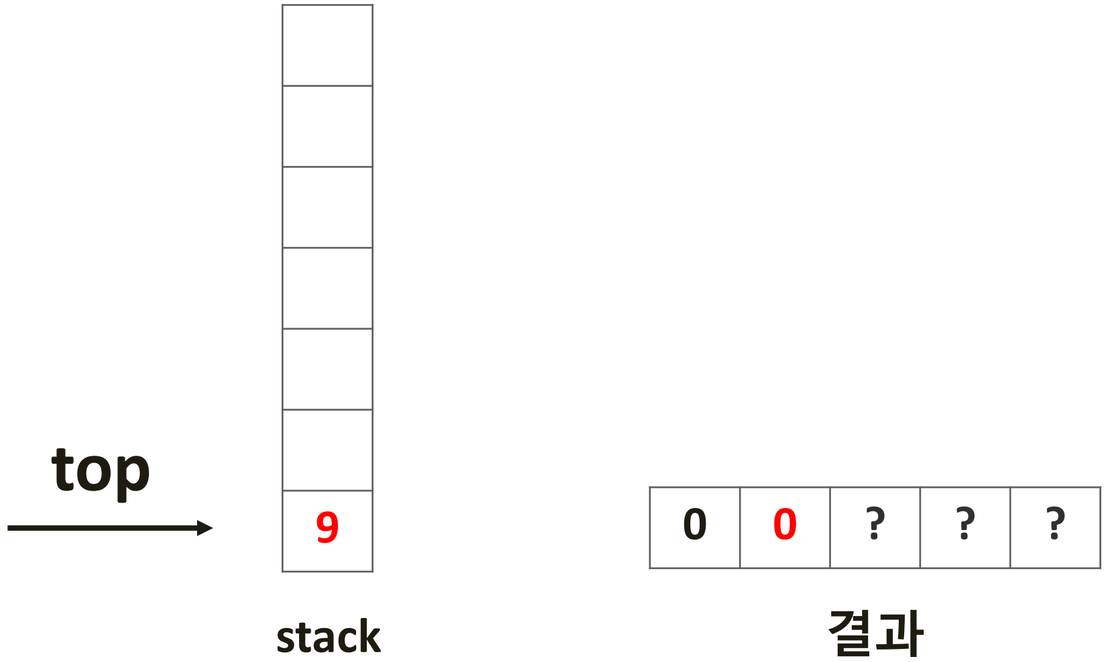
stack의 top을 확인한다.

top이 -1이 아니지만 현재 탑의 높이 보다 작다.

고로 6을 pop.

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



현재 탑의 높이 : 9 | top = -1

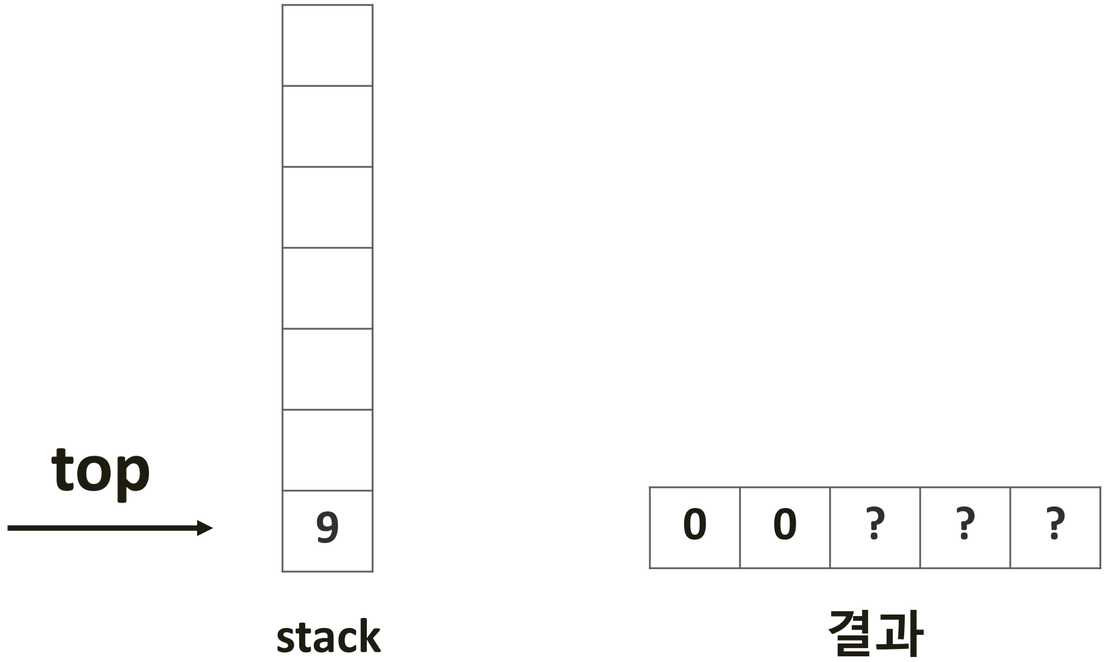
stack의 top을 확인한다.

top이 -1이므로 스택이 비었다.

스택이 빌 때 동안 송신 가능한 탑을 찾지 못했으므로 결과에 0을 등록한다.
또한 스택에 9를 push.

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



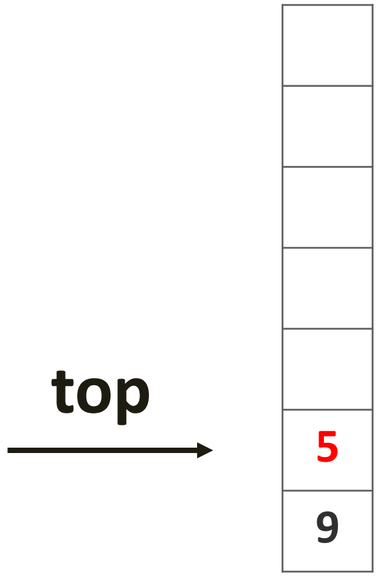
현재 탑의 높이 : 5

현재 탑의 높이는 5이다.

스택이 비지 않았기 때문에, 스택 내에 있는 높이 중에서 수신 가능한 놈을 찾아야한다.

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



stack

0	0	2	?	?

결과

현재 탑의 높이 : 5 | stack[top] = 9

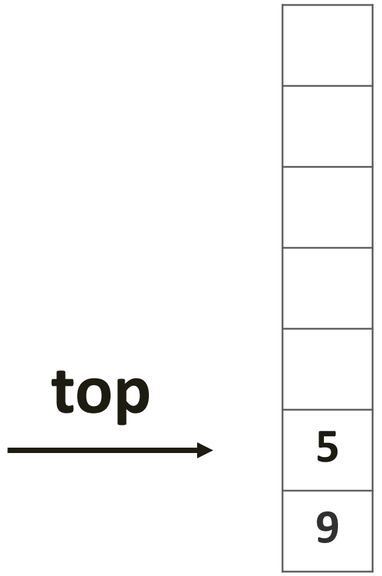
stack의 top을 확인한다.

top이 -1이 아니고, 현재 탑의 높이보다 높으므로 송신 가능하다.

고로 결과에 2를 등록하고 5를 push. (높이 9의 탑은 2번째 위치에 있기 때문에)

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



결과

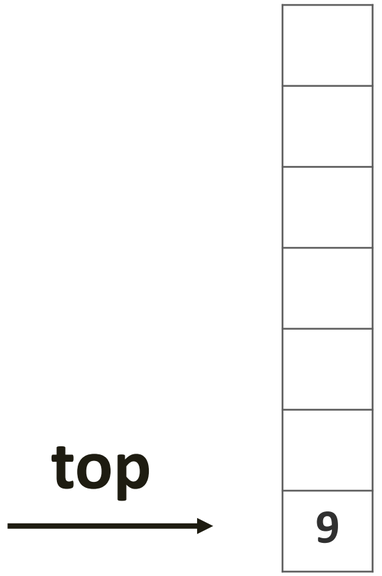
현재 탑의 높이 : 7

현재 탑의 높이는 7이다.

스택이 비지 않았기 때문에, 스택 내에 있는 높이 중에서 수신 가능한 놈을 찾아야한다.

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



stack

0	0	2	?	?

결과

현재 탑의 높이 : 7 | stack[top] = 5

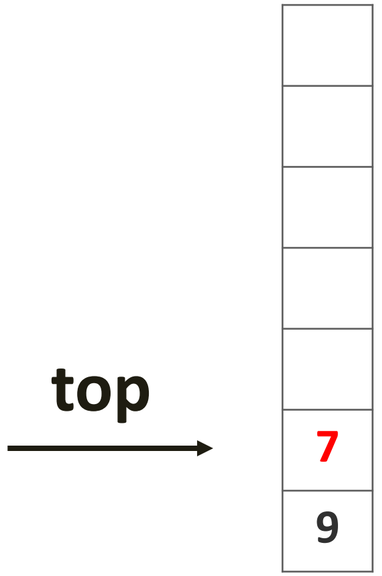
stack의 top을 확인한다.

top이 -1이 아니지만, 현재 탑의 높이 보다 작다.

고로 5을 pop.

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



stack

0	0	2	2	?

결과

현재 탑의 높이 : 7 | stack[top] = 9

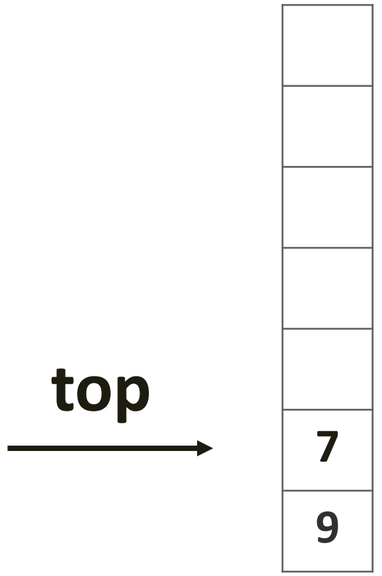
stack의 top을 확인한다.

top이 -1이 아니고, 현재 탑의 높이보다 높으므로 송신 가능하다.

고로 결과에 2를 등록하고 7를 push.

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



stack

0	0	2	2	?

결과

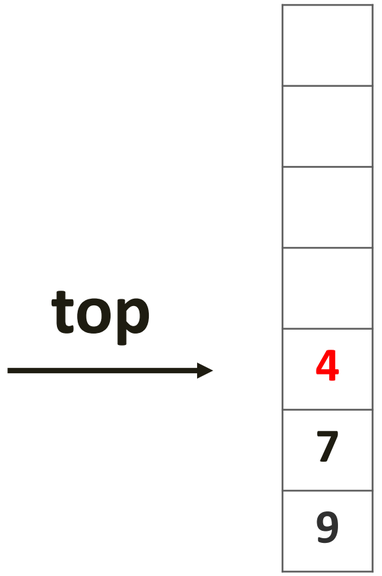
현재 탑의 높이 : 4

현재 탑의 높이는 4이다.

스택이 비지 않았기 때문에, 스택 내에 있는 높이 중에서 수신 가능한 놈을 찾아야한다.

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



stack

0	0	2	2	4

결과

현재 탑의 높이 : 4 | stack[top] = 7

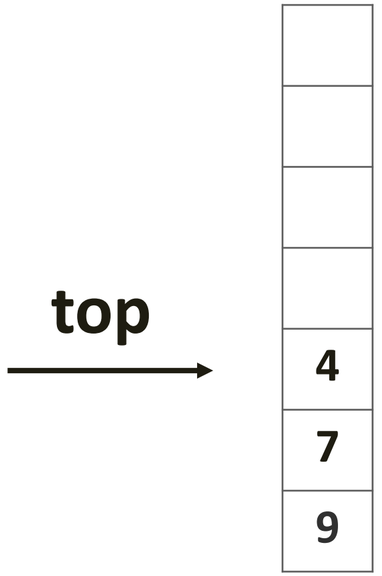
stack의 top을 확인한다.

top이 -1이 아니고, 현재 탑의 높이보다 높으므로 송신 가능하다.

고로 결과에 4를 등록하고 4를 push. (높이 7의 탑은 4번째 위치에 있기 때문에)

스택의 활용 – 단조성

```
5
6 9 5 7 4
```



0	0	2	2	4

결과

단조 감소를 통한 스택을 이용하여, 탑 문제를 풀어 보았다.

감소 뿐만 아니라, 단조 증가에 대해서도 여러 방면으로 활용할 수 있다.

스택은 왼쪽부터 오른쪽으로 훑으면서 조건을 판단하는 문제를 풀 때 빛을 발한다.