SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

APS 기본

SAMSUNG SW ACADEMY FOR YOUTH

APS(Algorithm Problem Solving)

스택 (Stack) & 큐 (Queue)

- 계산기
- 재귀호출

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

계산기

계산기

- **문자열로 된 계산식이 주어질 때, 스택을 이용하여 이 계산식의 값을 계산할 수 있다.**
- **문자열 수식 계산의 일반적 방법**
  - step1. 중위 표기법의 수식을 후위 표기법으로 변경한다. (스택 이용)
  - step2. 후위 표기법의 수식을 스택을 이용하여 계산한다.

중위표기법(infix notation)
- 연산자를 피연산자의 가운데 표기하는 방법
  예) A+B

후위표기법(postfix notation)
- 연산자를 피연산자 뒤에 표기하는 방법
  예) AB+

계산기

step1. 중위표기식의 후위표기식 변환 방법1

- 수식의 각 연산자에 대해서 우선순위에 따라 괄호를 사용하여 다시 표현한다.
- 각 연산자를 그에 대응하는 오른쪽괄호의 뒤로 이동시킨다.
- 괄호를 제거한다.

예) A*B-C/D

1단계 : ((A*B) - (C/D))

2단계 : ((A B)* (C D)/)-

3단계 : AB*CD/-

step1. 중위 표기법에서 후위 표기법으로의 변환 알고리즘(스택 이용)2

① 입력 받은 중위 표기식에서 토큰을 읽는다.

② 토큰이 피연산자이면 토큰을 출력한다.

③ 토큰이 연산자(괄호포함)일 때, 이 토큰이 스택의 top에 저장되어 있는 연산자보다 우선순위가 높으면 스택에 push하고, 그렇지 않다면 스택 top의 연산자의 우선순위가 토큰의 우선순위보다 작을 때까지 스택에서 pop 한 후 토큰의 연산자를 push한다. 만약 top에 연산자가 없으면 push한다.

④ 토큰이 오른쪽 괄호 ')'이면 스택 top에 왼쪽 괄호 '('가 올 때까지 스택에 pop 연산을 수행하고 pop 한 연산자를 출력한다. 왼쪽 괄호를 만나면 pop만 하고 출력하지는 않는다.

⑤ 중위 표기식에 더 읽을 것이 없다면 중지하고, 더 읽을 것이 있다면 1부터 다시 반복한다.

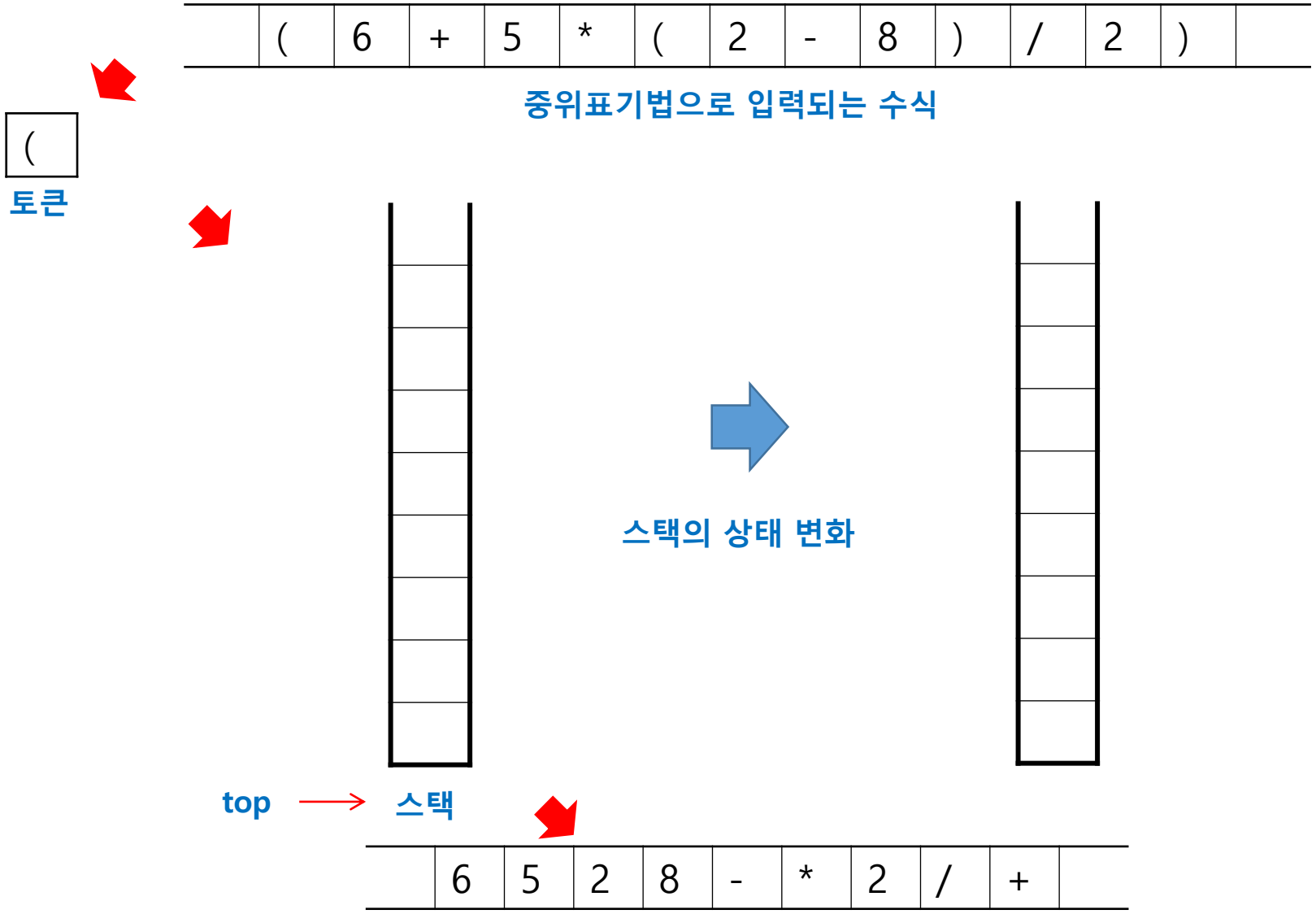
⑥ 스택에 남아 있는 연산자를 모두 pop하여 출력한다.

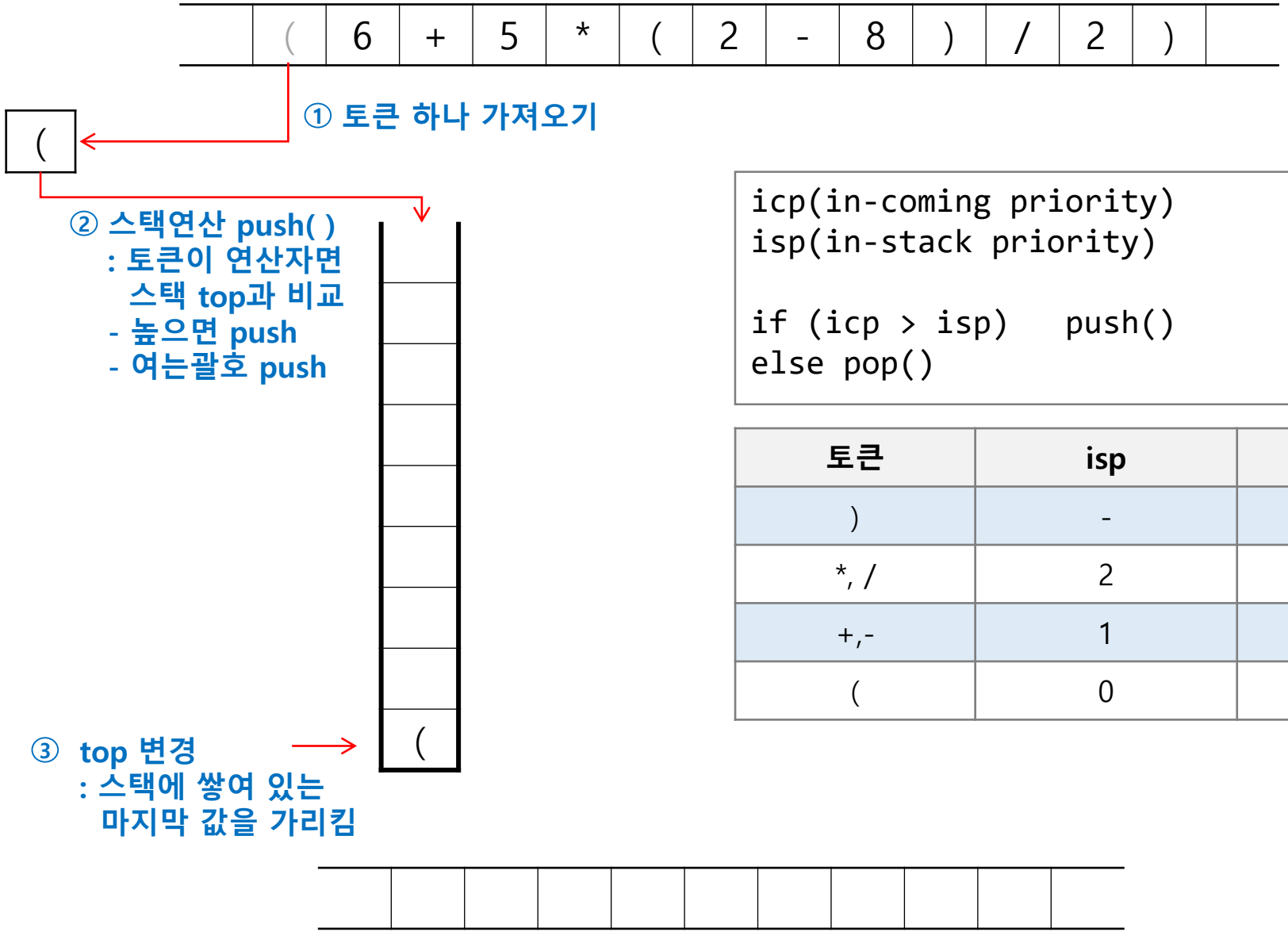
- **스택 밖의 왼쪽 괄호는 우선 순위가 가장 높으며, 스택 안의 왼쪽 괄호는 우선 순위가 가장 낮다.**

계산기

- 우선 중위 표기법에서 후위 표기법으로의 변환한다.

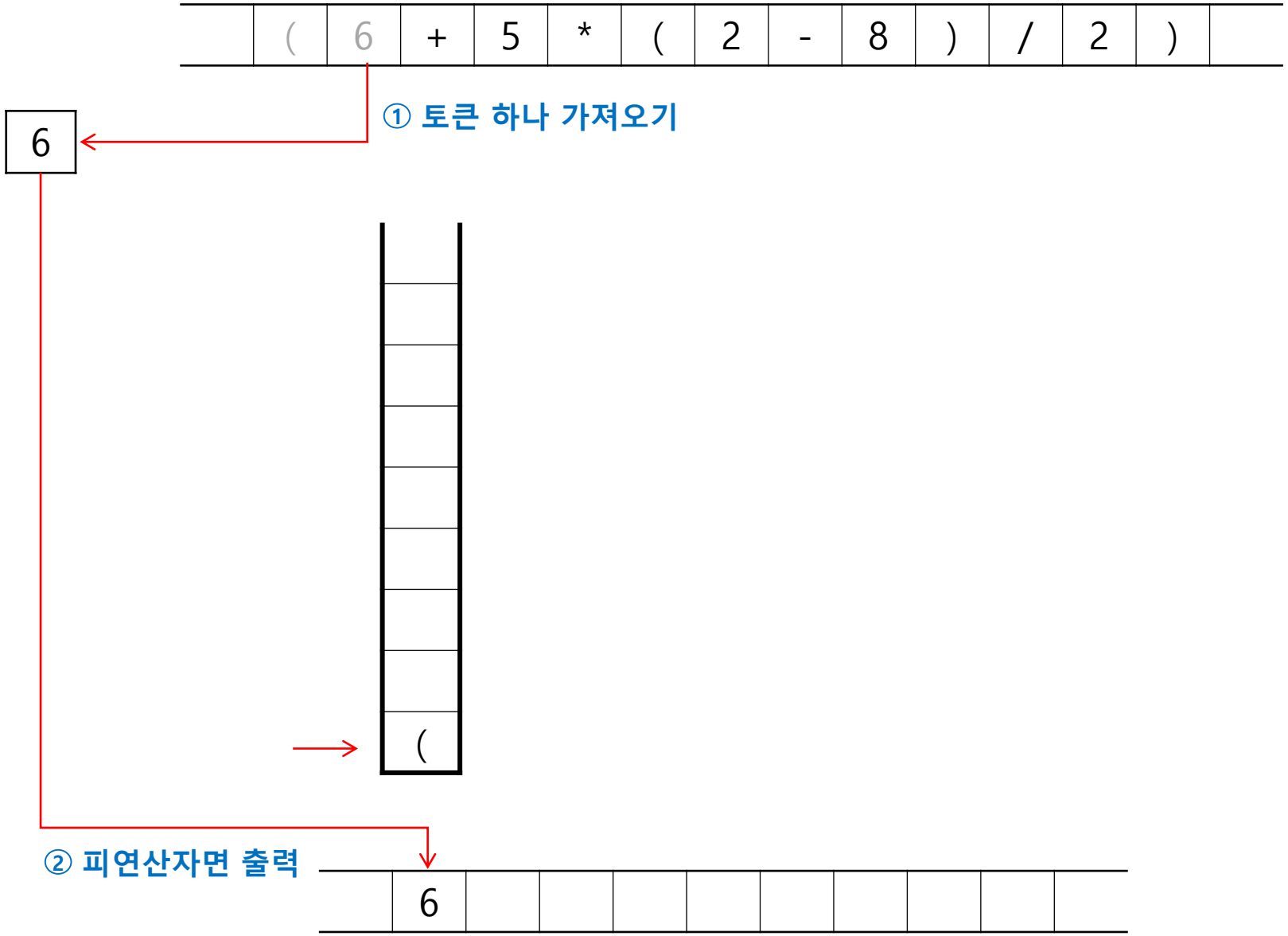
- 중위 표기법으로 표현된 수식 예
  - (6 + 5 * (2 – 8) / 2)

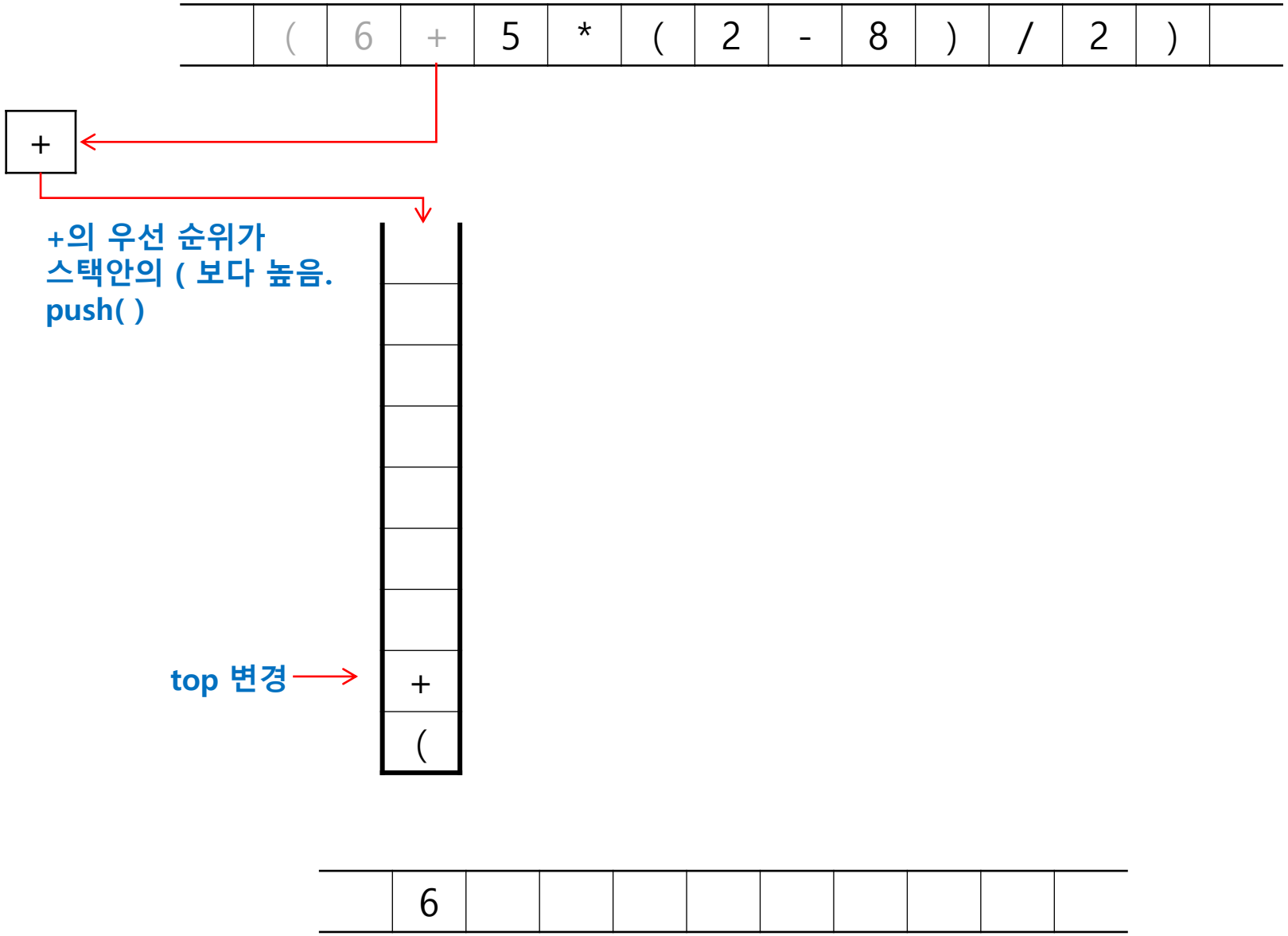

(6 + 5 * (2 - 8) / 2)
중위표기법으로 입력되는 수식
(
토큰
스택의 상태 변화
top
스택
6 5 2 8 - * 2 / +
후위표기법으로 출력될 수식

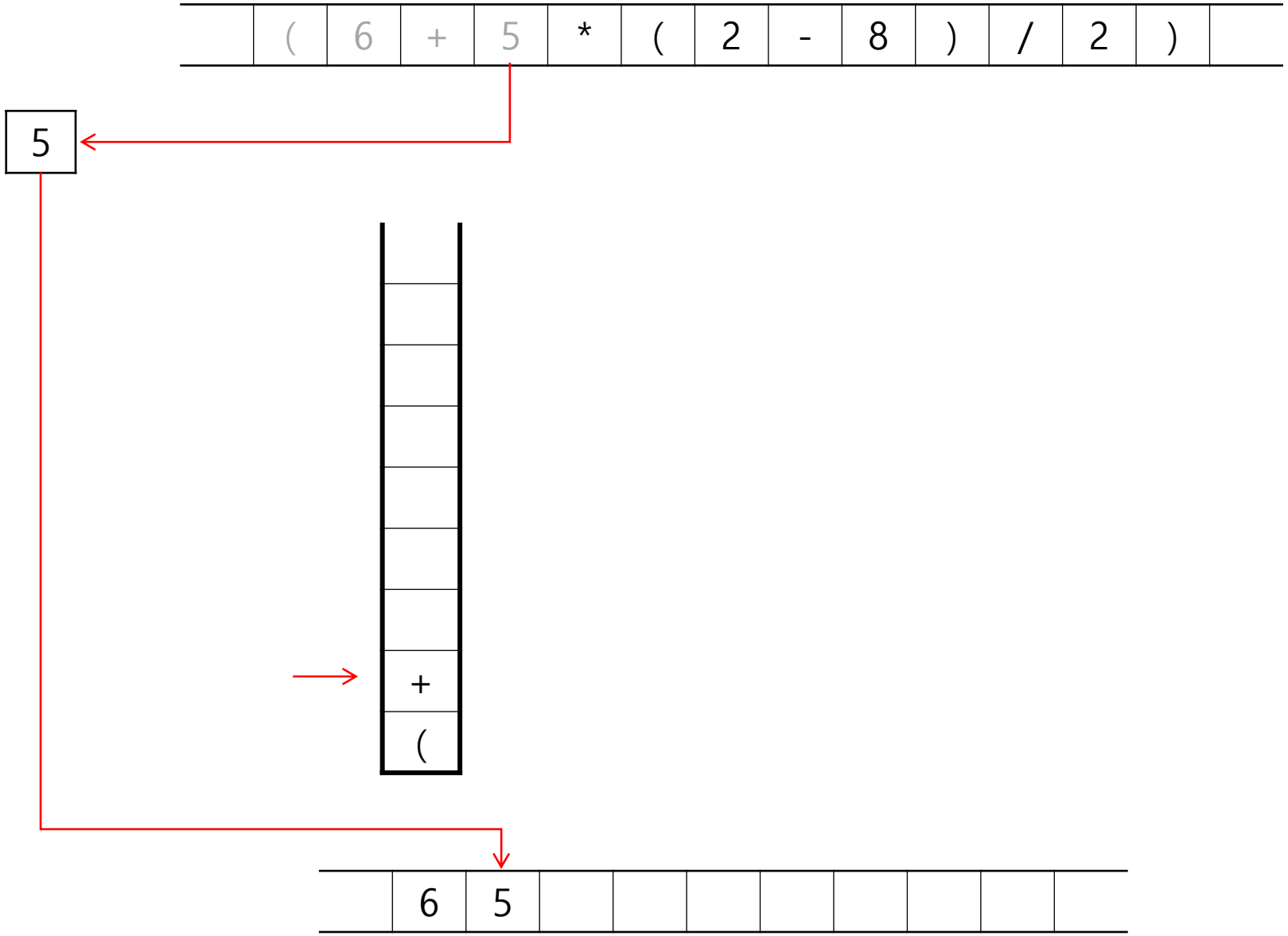


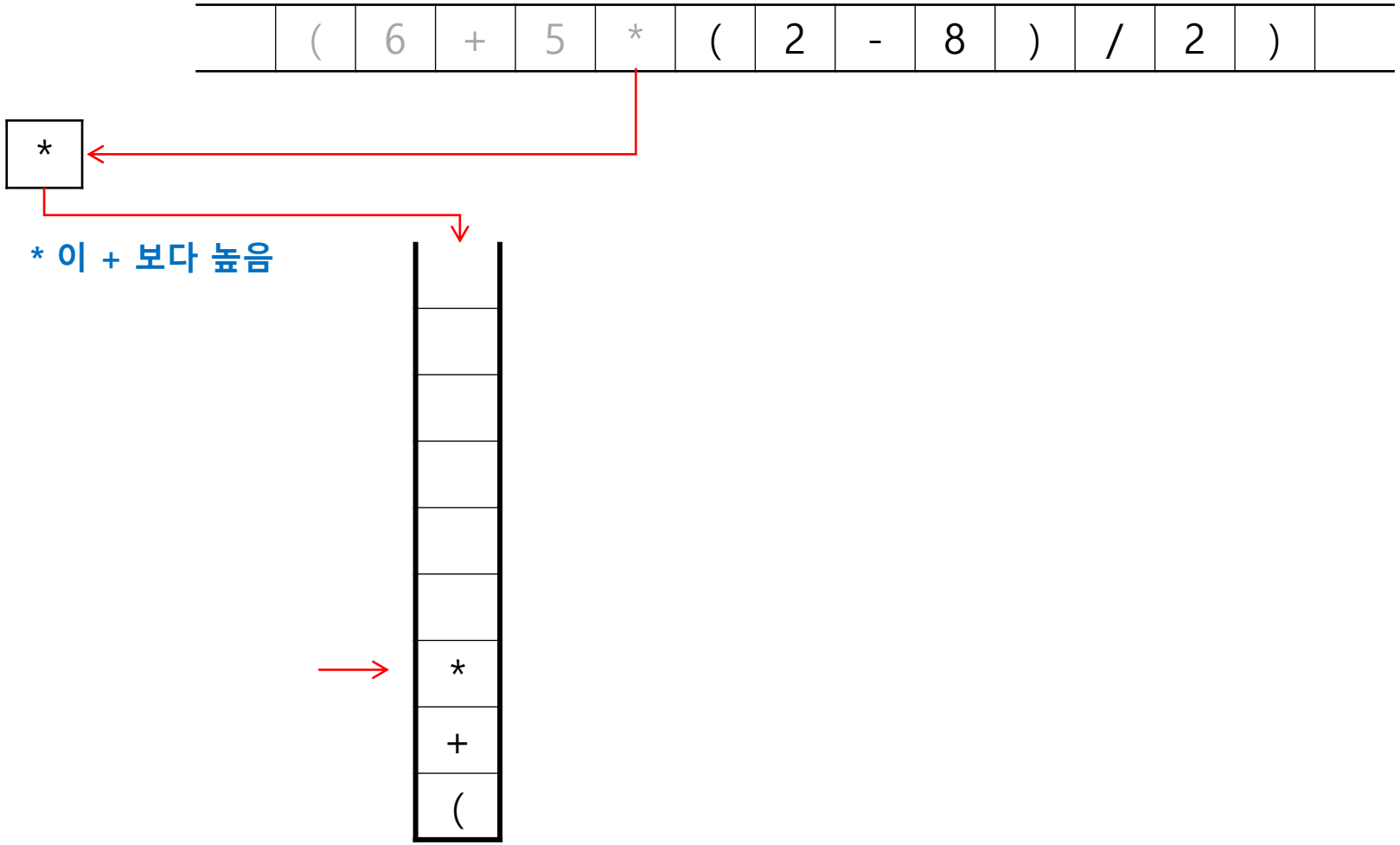

```
icp(in-coming priority)
isp(in-stack priority)

if (icp > isp)   push()
else pop()
```

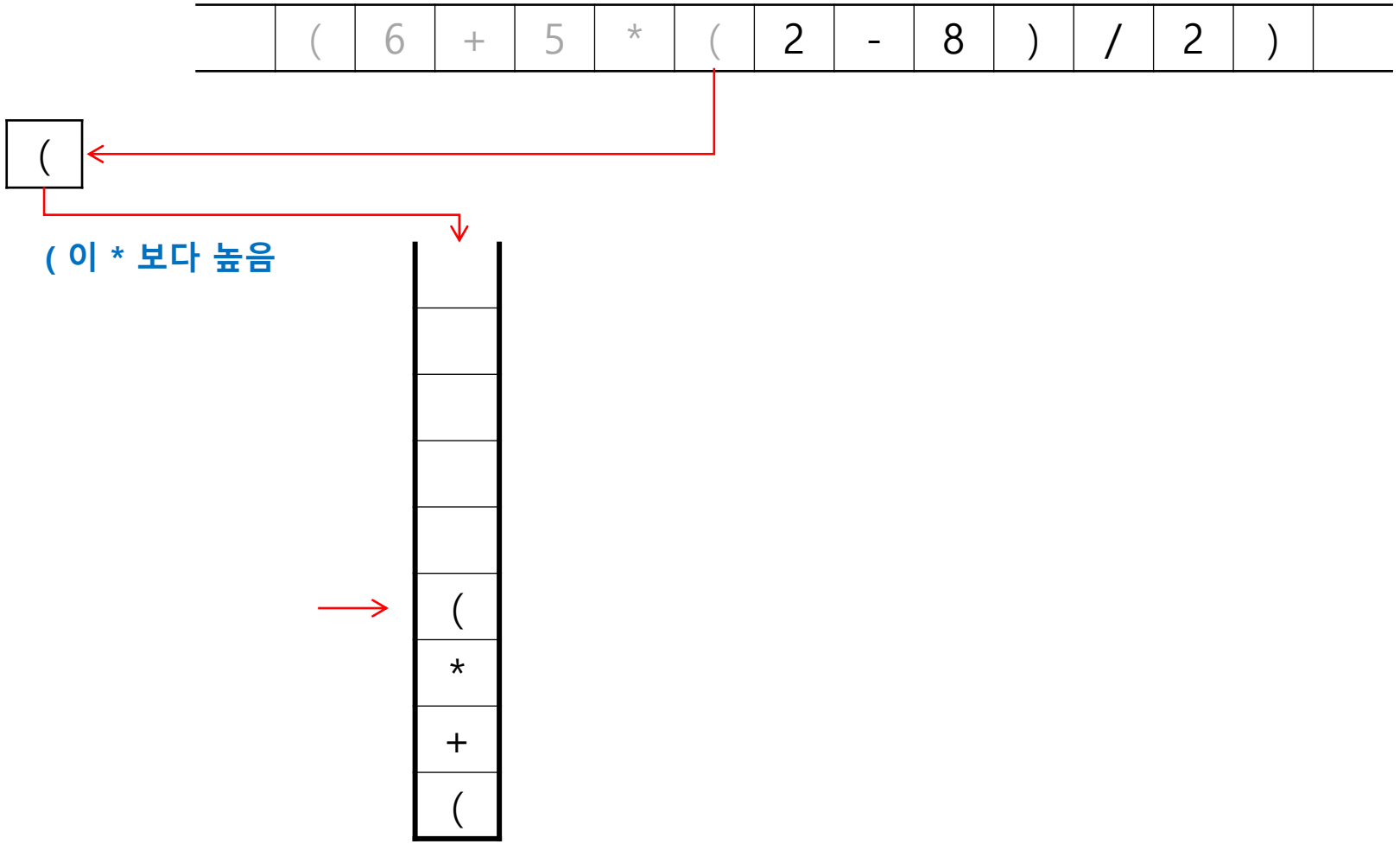
토큰	isp	icp
)	-	-
*, /	2	2
+,-	1	1
(	0	3

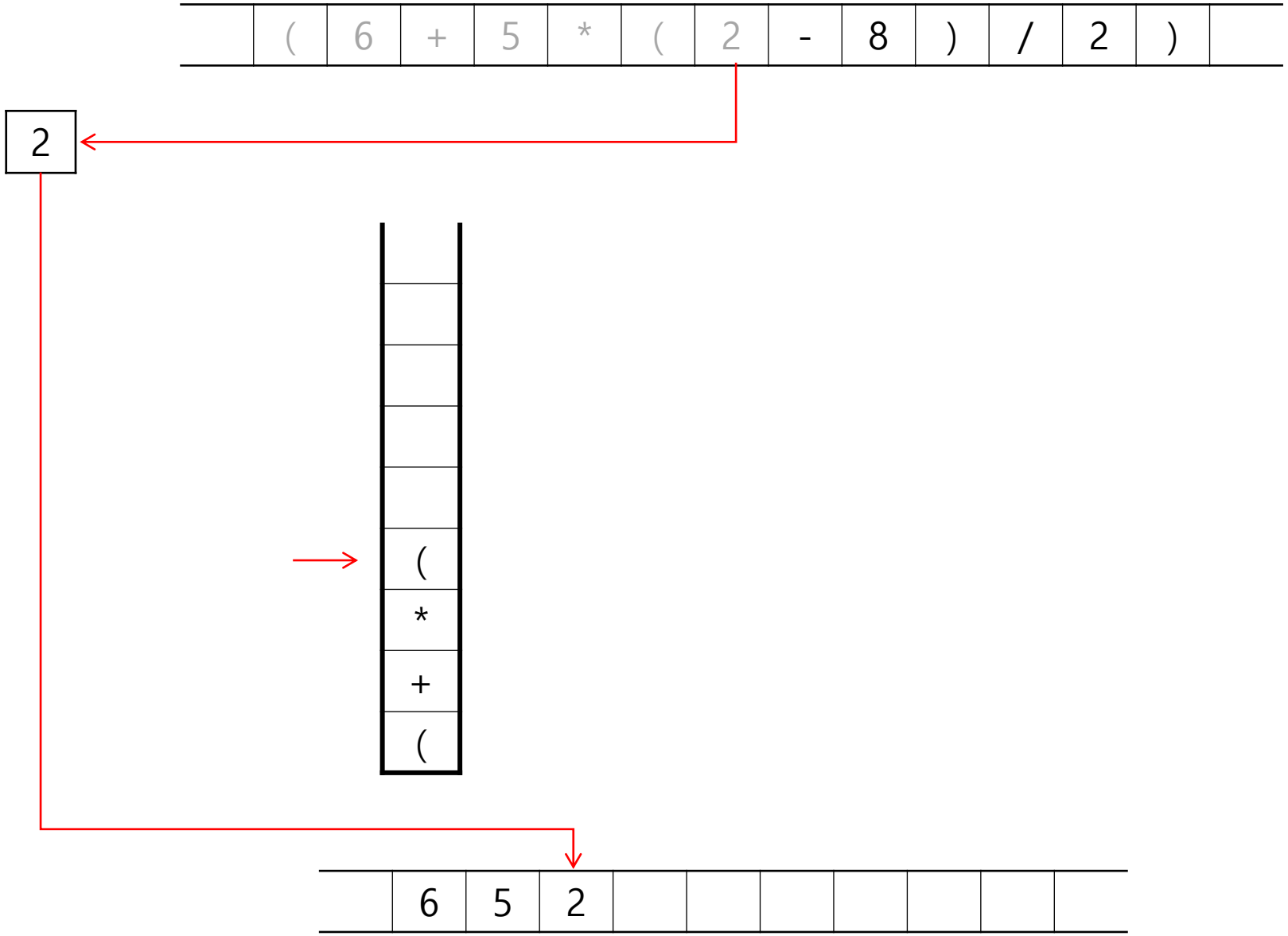

(6 + 5 * (2 - 8) / 2)
① 토큰 하나 가져오기
6
(
② 피연산자면 출력
6

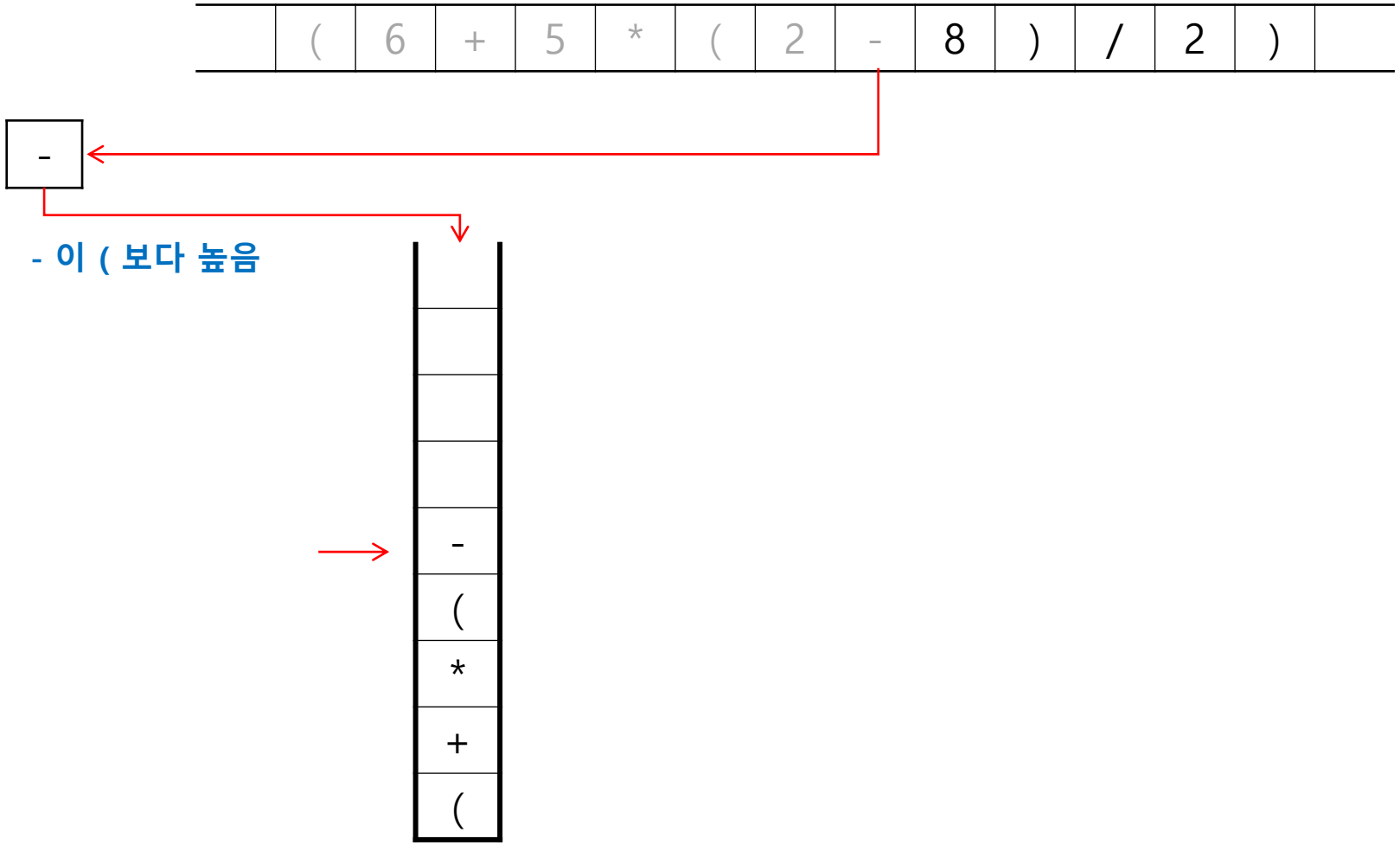

(6 + 5 * (2 - 8) / 2)
+
+의 우선 순위가
스택안의 (보다 높음.
push( )
top 변경
+
(
6

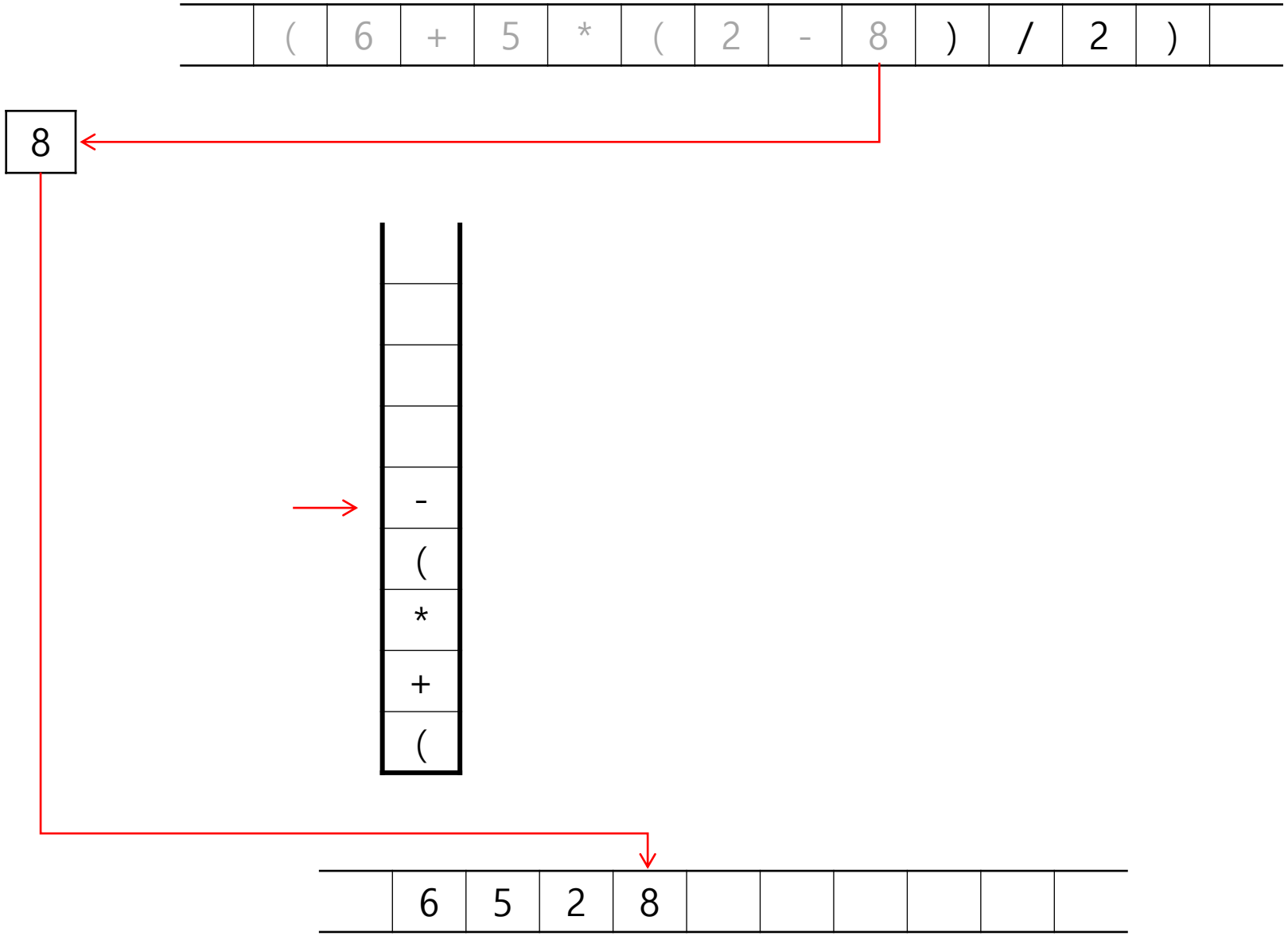

(
6
+
5
*
(
2
-
8
)
/
2
)
5
+
(
6
5

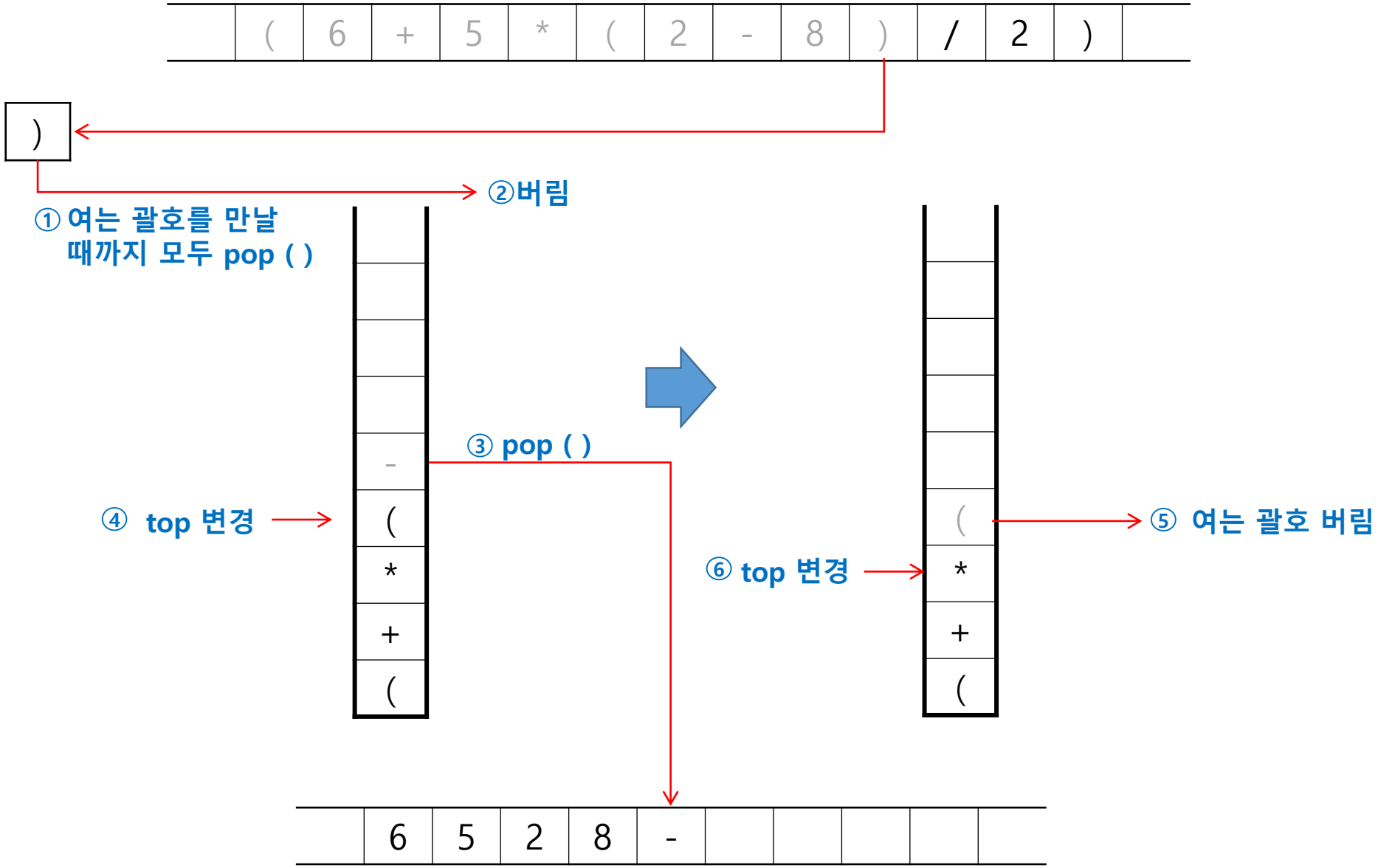

(6 + 5 * (2 - 8) / 2)
*
* 이 + 보다 높음
*
+
(
6 5

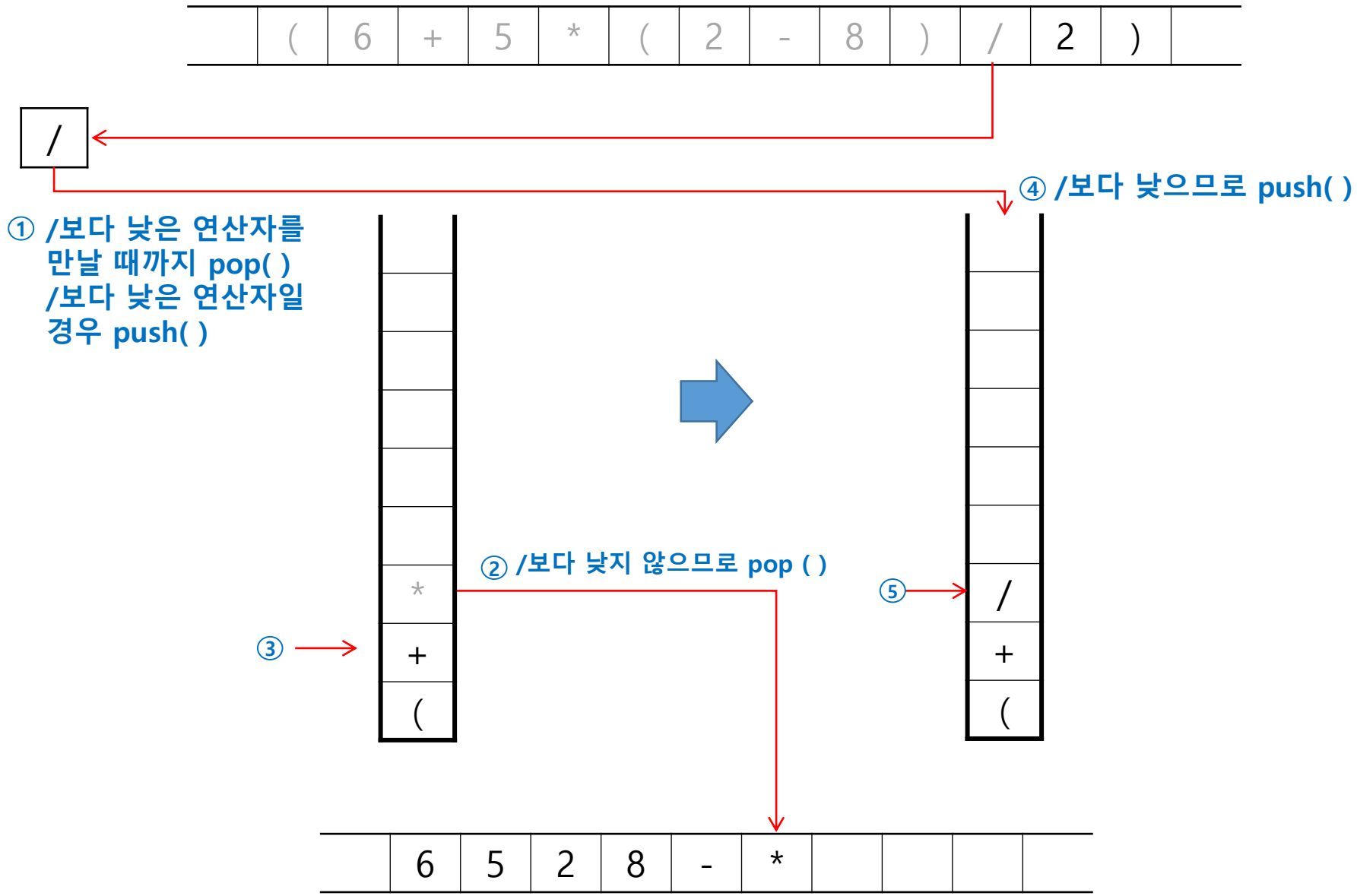

(6 + 5 * (2 - 8) / 2)
(
(이 * 보다 높음
(
*
+
(
6 5

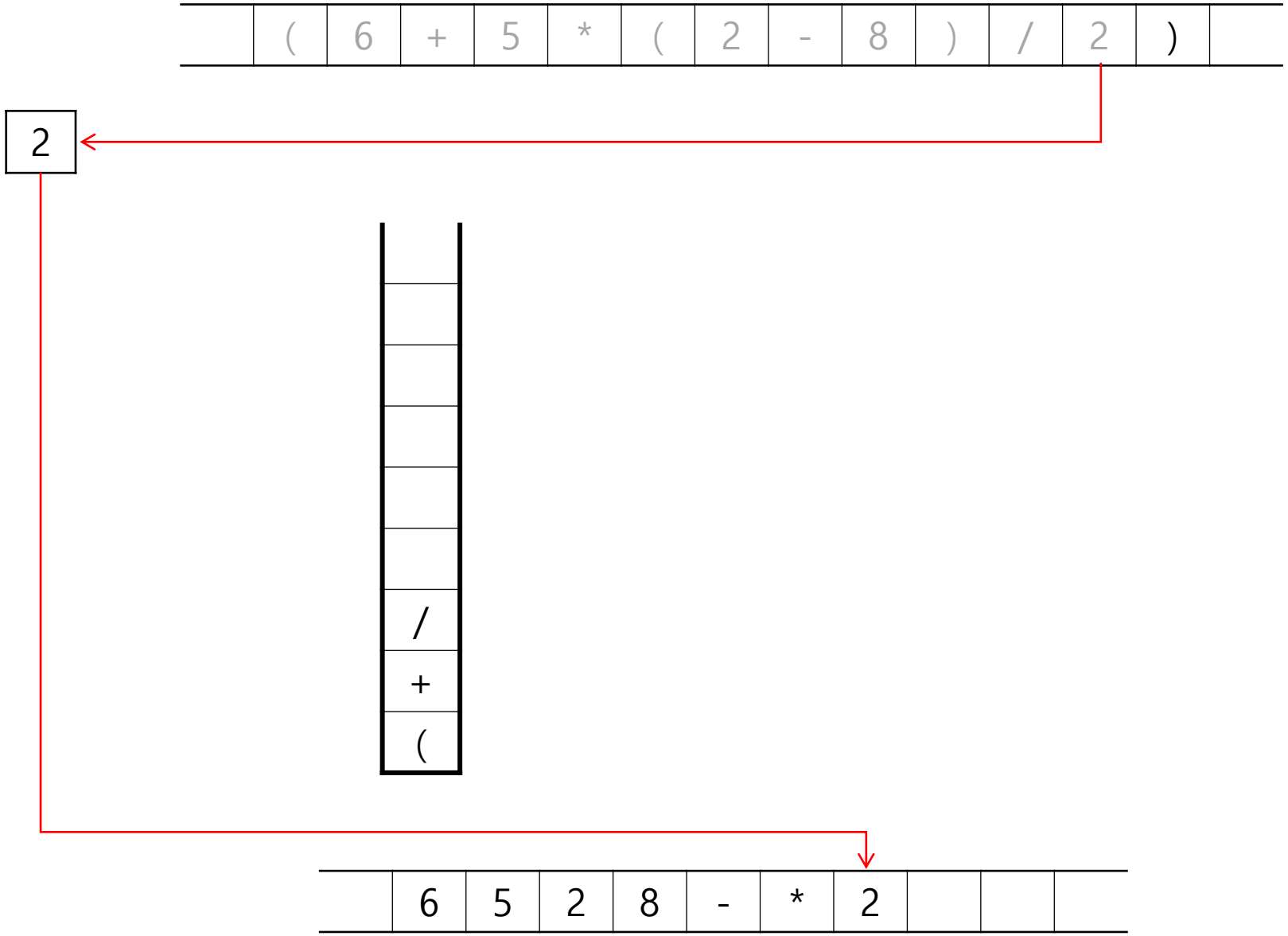

(6 + 5 * (2 - 8) / 2)
2
(
*
+
(
6 5 2

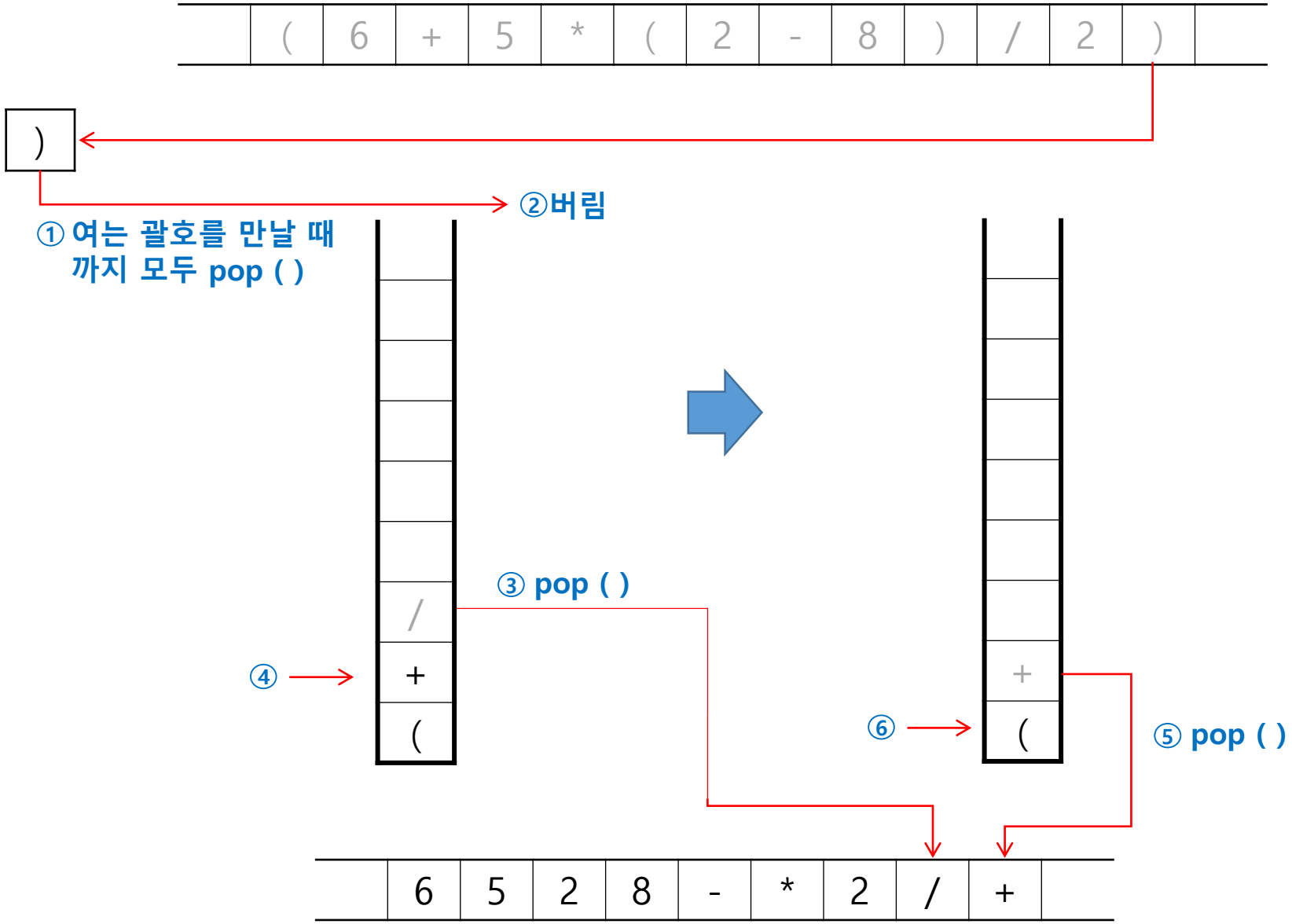

(6 + 5 * (2 - 8) / 2)
-
- 이 (보다 높음
-
(
*
+
(
6 5 2

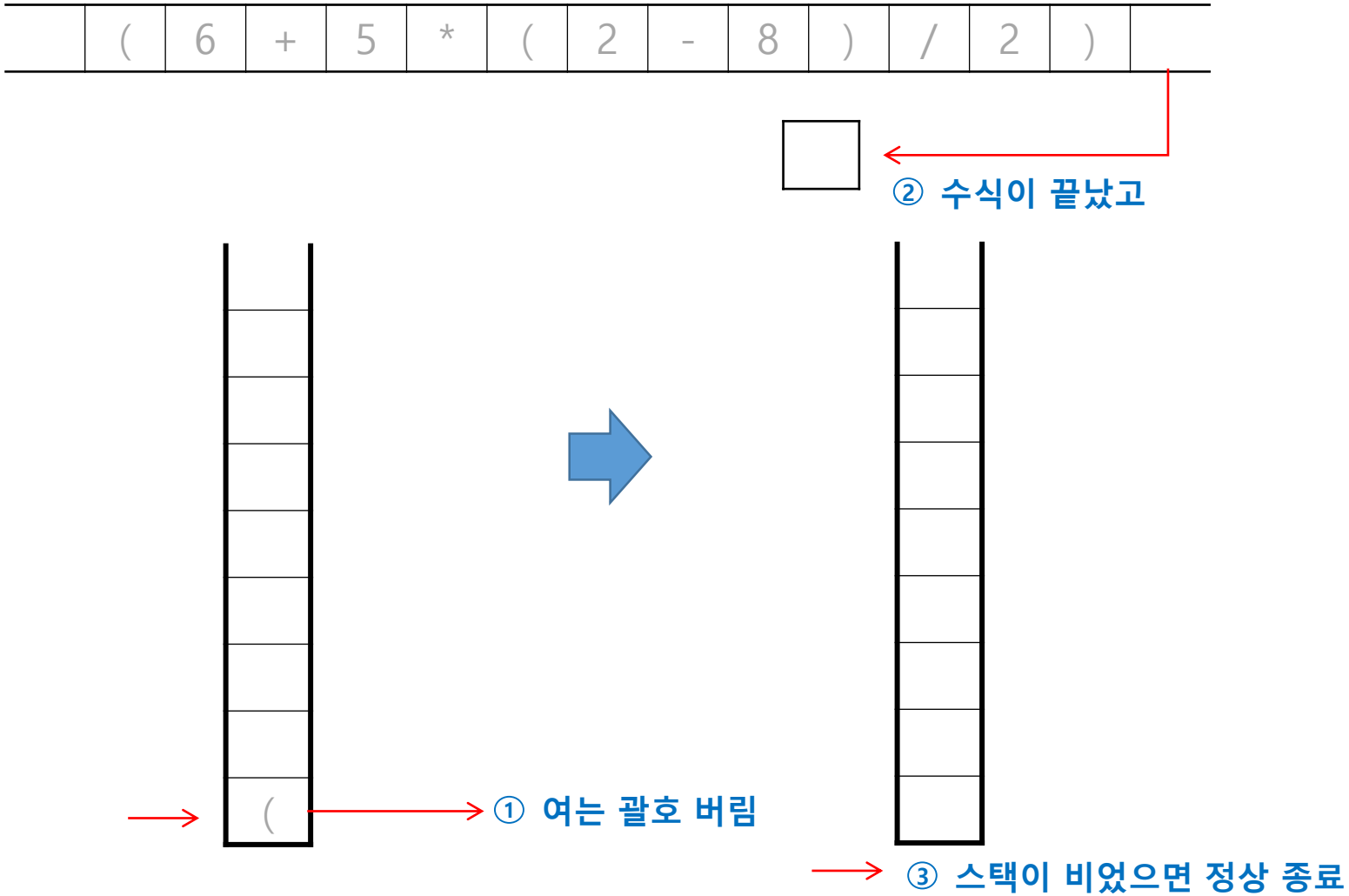

(6 + 5 * (2 - 8) / 2)
8
-
(
*
+
(
6 5 2 8


(6 + 5 * (2 - 8) / 2)
)
② 버림
① 여는 괄호를 만날 때까지 모두 pop ( )
③ pop ( )
④ top 변경
⑤ 여는 괄호 버림
⑥ top 변경
6 5 2 8 -


(6 + 5 * (2 - 8) / 2)
/
① /보다 낮은 연산자를 만날 때까지 pop( ) /보다 낮은 연산자일 경우 push( )
④ /보다 낮으므로 push( )
② /보다 낮지 않으므로 pop ( )
③
⑤
* + (
/ + (
6 5 2 8 - *


(6 + 5 * (2 - 8) / 2)
2
/
+
(
6 5 2 8 - * 2


(6 + 5 * (2 - 8) / 2)
)
① 여는 괄호를 만날 때 까지 모두 pop ( )
②버림
③ pop ( )
④
/
+
(
⑤ pop ( )
⑥
+
(
6 5 2 8 - * 2 / +

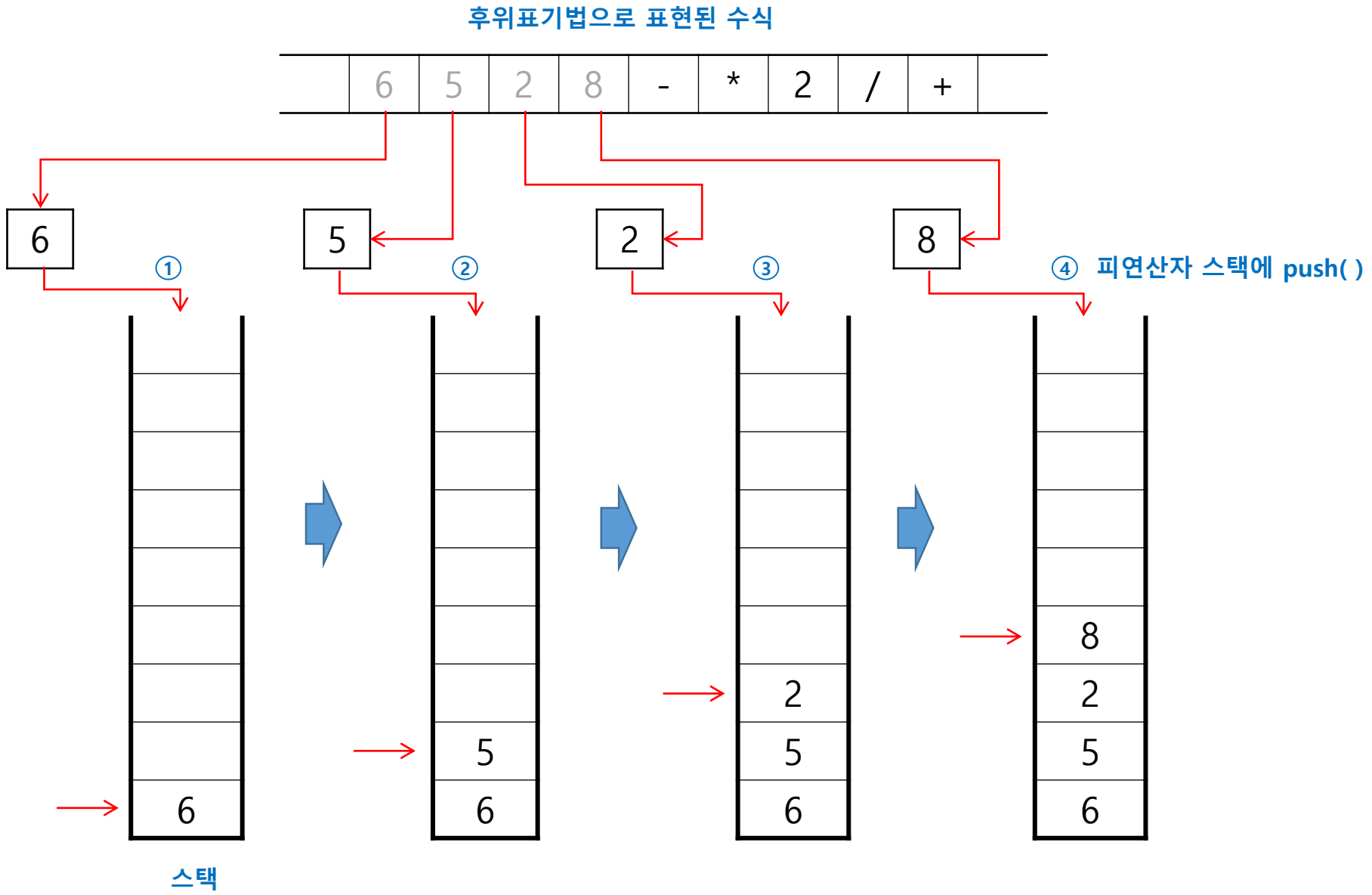

(6 + 5 * (2 - 8) / 2)
② 수식이 끝났고
(
① 여는 괄호 버림
③ 스택이 비었으면 정상 종료
6 5 2 8 - * 2 / +
④ 결과 값(후위표기법 수식)

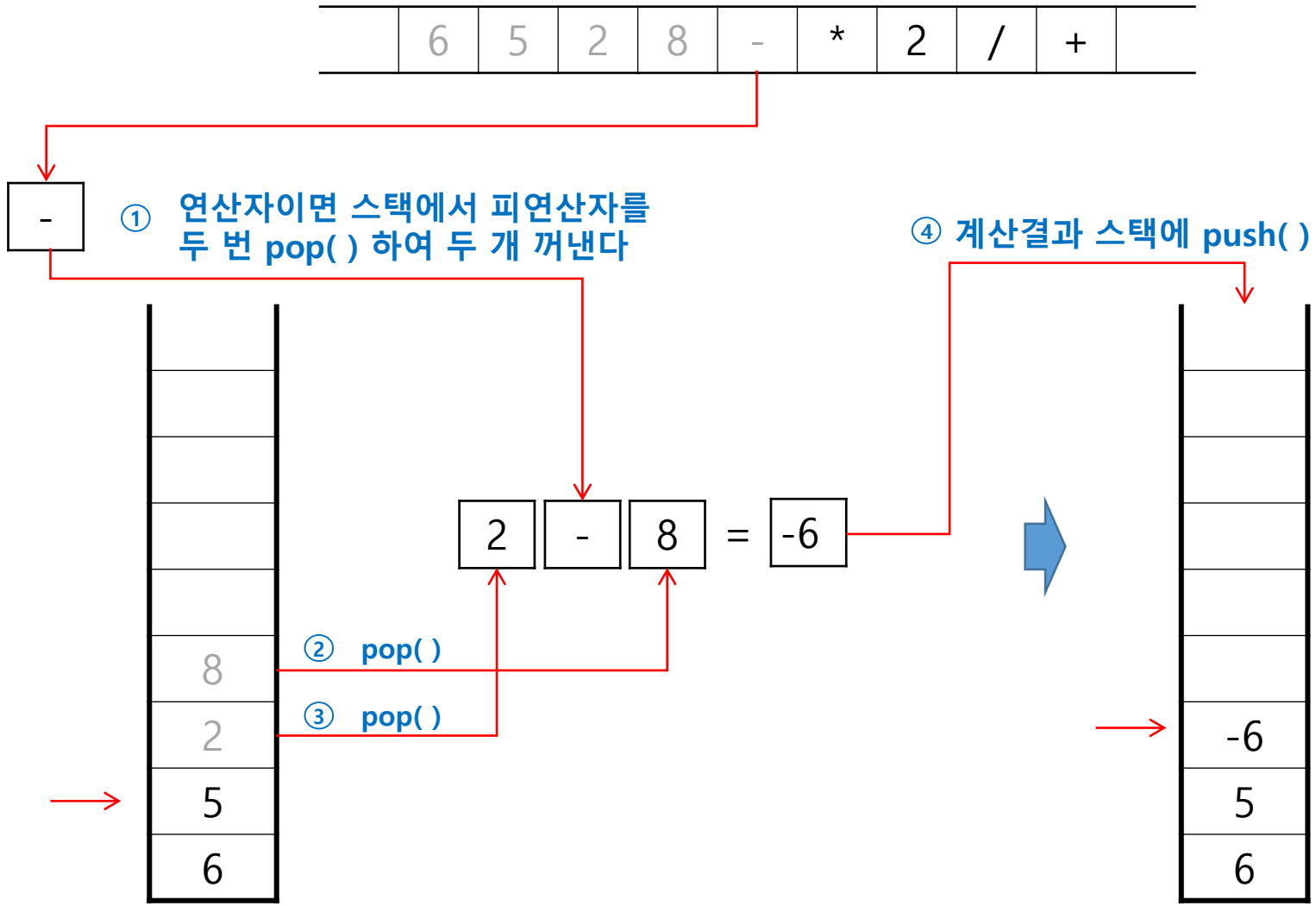
step2. 후위 표기법의 수식을 스택을 이용하여 계산

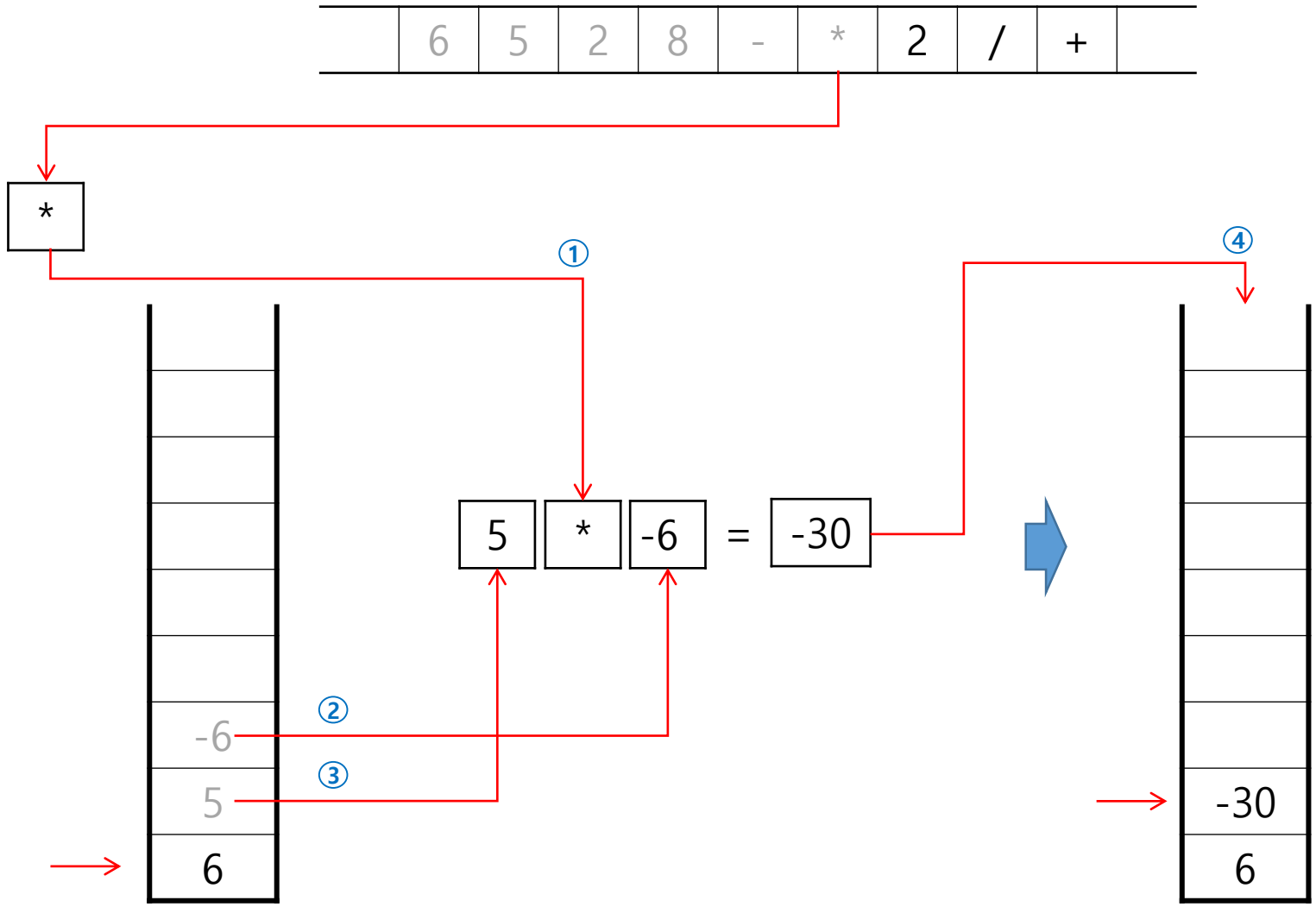
① 피연산자를 만나면 스택에 push 한다.

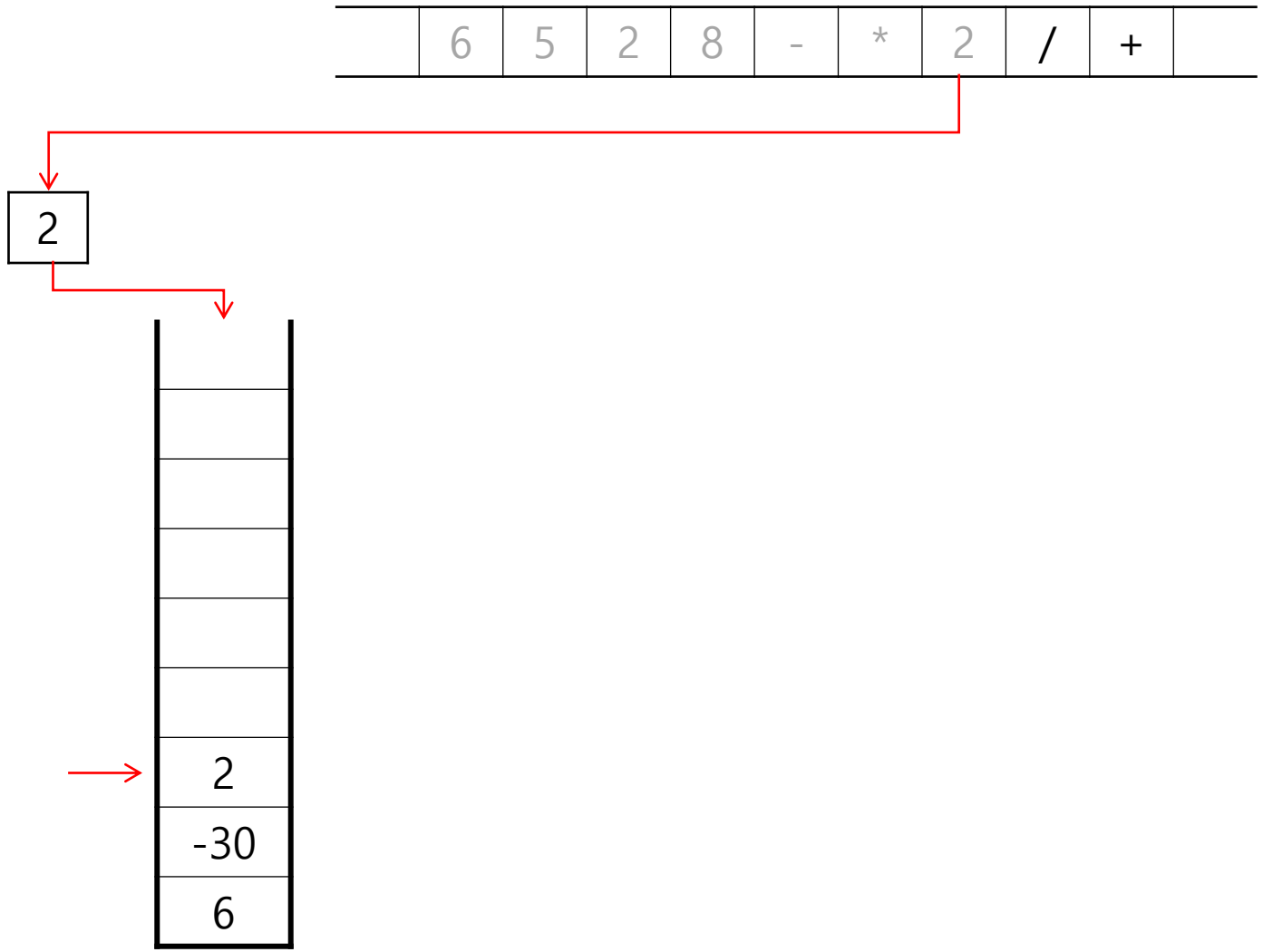
② 연산자를 만나면 필요한 만큼의 피연산자를 스택에서 pop하여 연산하고, 연산결과를 다시 스택에 push 한다.

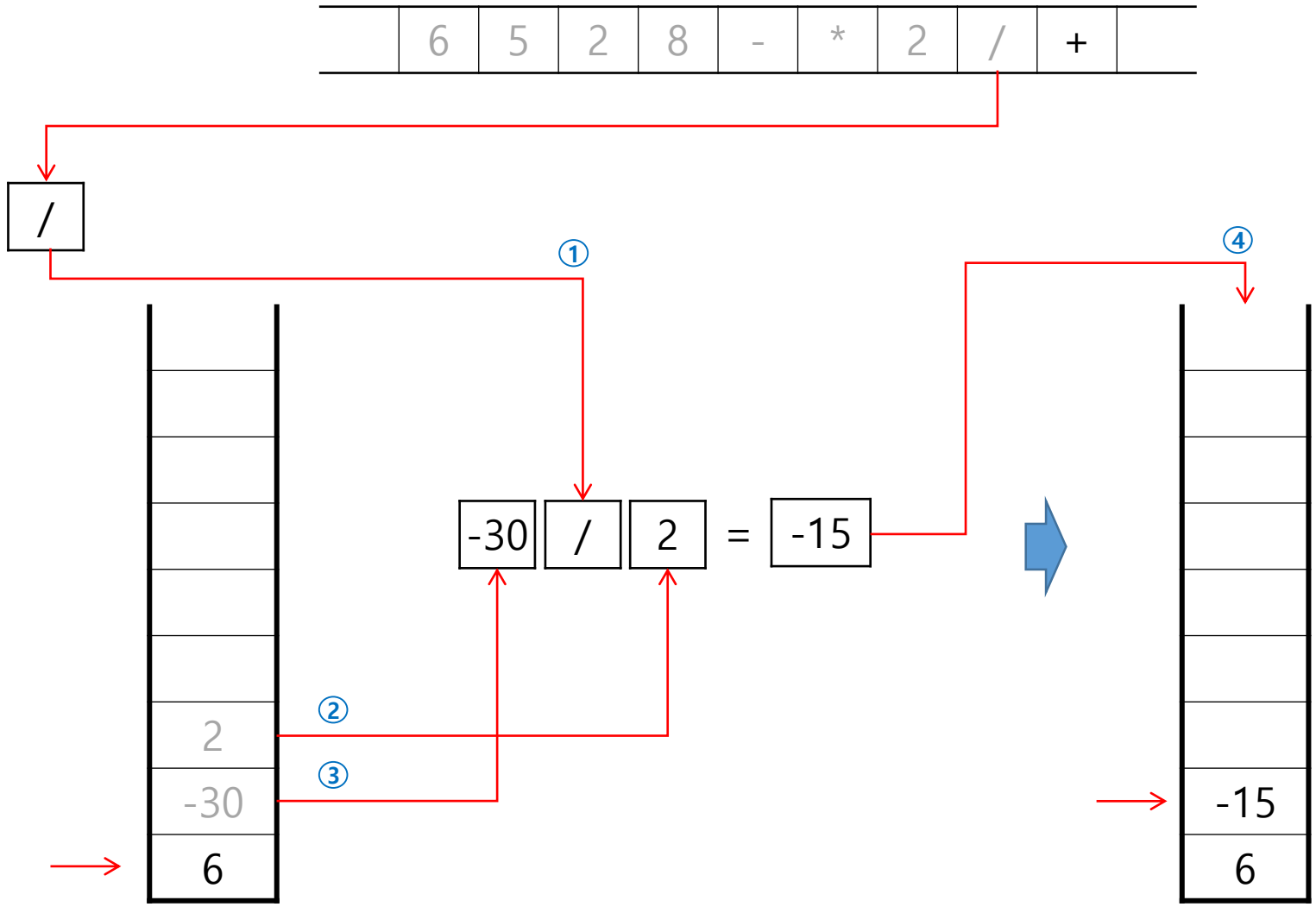
③ 수식이 끝나면, 마지막으로 스택을 pop하여 출력한다.

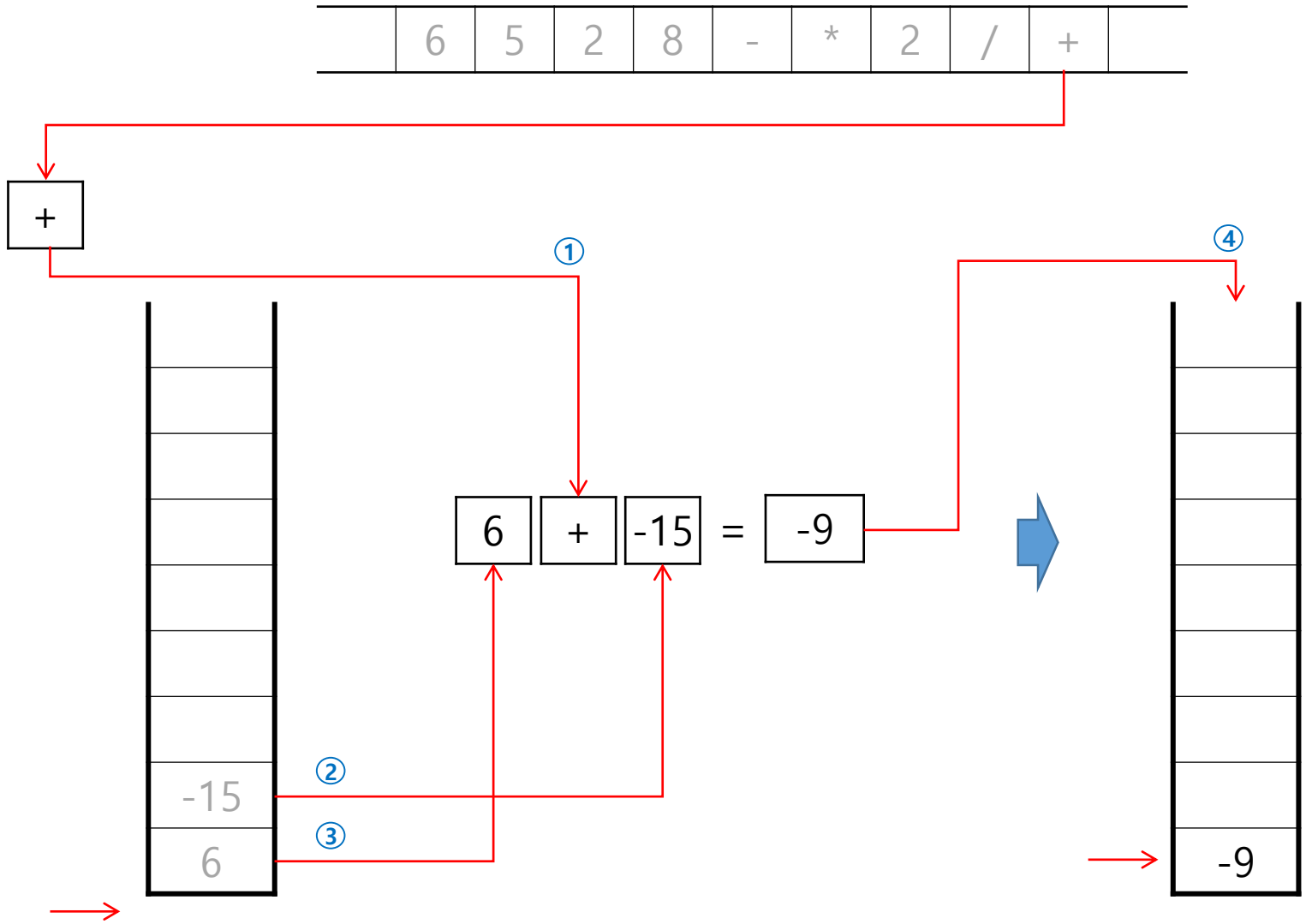

후위표기법으로 표현된 수식
6 5 2 8 - * 2 / +
6
5
2
8
① 피연산자 스택에 push( )
스택

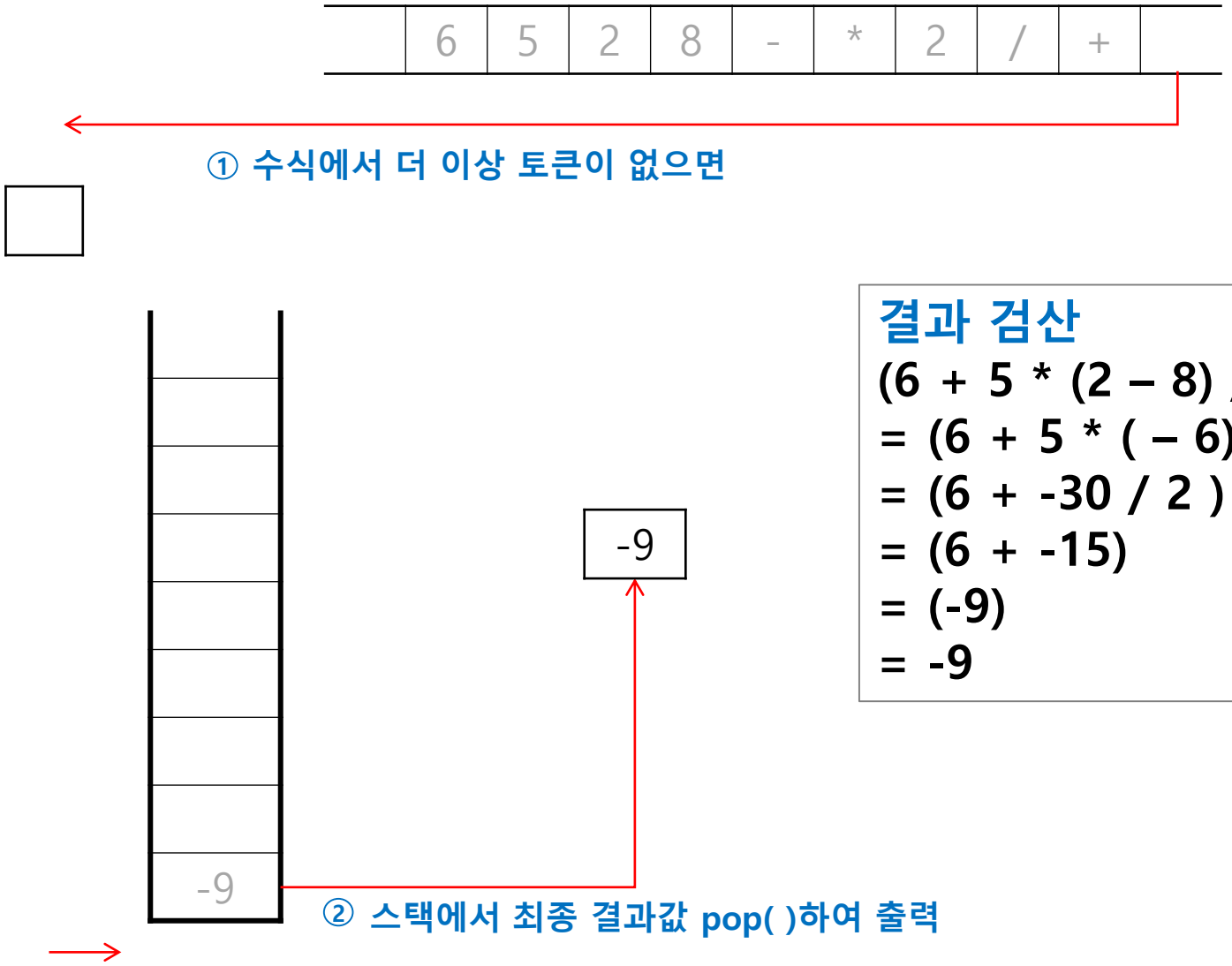

6 5 2 8 - * 2 / +
-
① 연산자이면 스택에서 피연산자를
두 번 pop( ) 하여 두 개 꺼낸다
④ 계산결과 스택에 push( )
2 - 8 = -6
② pop( )
③ pop( )
8
2
5
6
-6
5
6


6
5
2
8
-
*
2
/
+
*
①
②
③
④
5
*
-6
=
-30
-6
5
6
-30
6


6
5
2
8
-
*
2
/
+
2
2
-30
6


6
5
2
8
-
*
2
/
+
/
①
②
③
④
-30
/
2
=
-15
2
-30
6
-15
6


6
5
2
8
-
*
2
/
+
+
①
②
③
④
6
+
-15
=
-9
-15
6
-9



결과 검산

(6 + 5 * (2 – 8) / 2)
= (6 + 5 * (– 6) / 2)
= (6 + -30 / 2)
= (6 + -15)
= (-9)
= -9

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

재귀 호출

재귀호출

- 再 (다시 재) 歸(돌아갈 귀)
- 자기 자신을 호출하여 순환 수행되는 것
- 함수 호출은 메모리 구조에서 스택을 사용한다. (이름만 같은 다른 메서드)
  - 간단한 문제에 대해서는 반복문에 비해 메모리 및 속도에서 성능저하가 발생한다.
- 일반적으로 기본 부분(Base case), 재귀 부분(Recursive case)로 구성된다.
  - Base case : 재귀 호출에서 빠져 나가기 위한 조건
  - Recursive case : 자신을 호출하는 부분 (Base case로 유도한다.)
- 재귀적 프로그램을 작성하는 것은 반복 구조에 비해 간결하고 이해하기 쉽다.

재귀호출

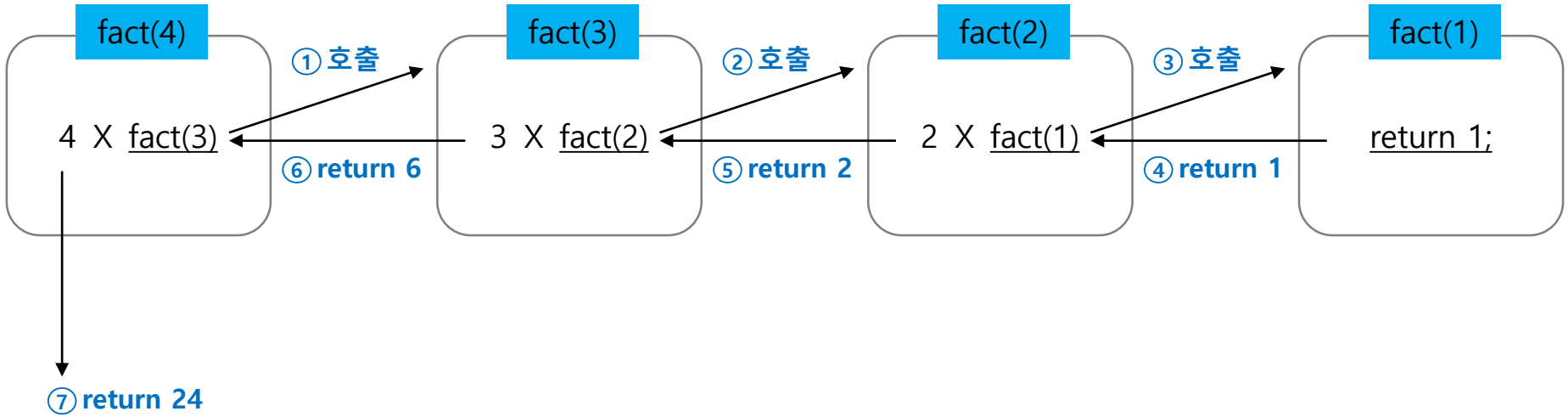
- **함수에서 실행해야 하는 작업의 특성에 따라 일반적인 호출방식보다 재귀호출방식을 사용하여 함수를 만들면 프로그램의 크기를 줄이고 간단하게 작성**
  - 재귀 호출의 예) factorial
    - n에 대한 factorial : 1부터 n까지의 모든 자연수를 곱하여 구하는 연산

```
 n! = n x (n-1)!
   (n-1)! = (n-1) x (n-2)!
   (n-2)! = (n-2) x (n-3)!
…
   2! = 2 x 1!
   1! = 1
```

    - 마지막에 구한 하위 값을 이용하여 상위 값을 구하는 작업을 반복

재귀호출

factorial 함수에서 n=4 인 경우의 실행



재귀호출

- 0과 1로 시작하고 이전의 두 수 합을 다음 항으로 하는 수열을 피보나치라 한다
  - 0, 1, 1, 2, 3, 5, 8, 13, ...
- 피보나치 수열의 i번 째 값을 계산하는 함수 F를 정의 하면 다음과 같다.
  - $F_0 = 0, F_1 = 1$
  - $F_i = F_{i-1} + F_{i-2}$ for $i \geq 2$
- 위의 정의로부터 피보나치 수열의 i번째 항을 반환하는 함수를 재귀함수로 구현할 수 있다.

재귀호출

- 피보나치 수를 구하는 재귀함수

```
public static int fibo(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fibo(n - 1) + fibo(n - 2);
    }
}
```

Memoization

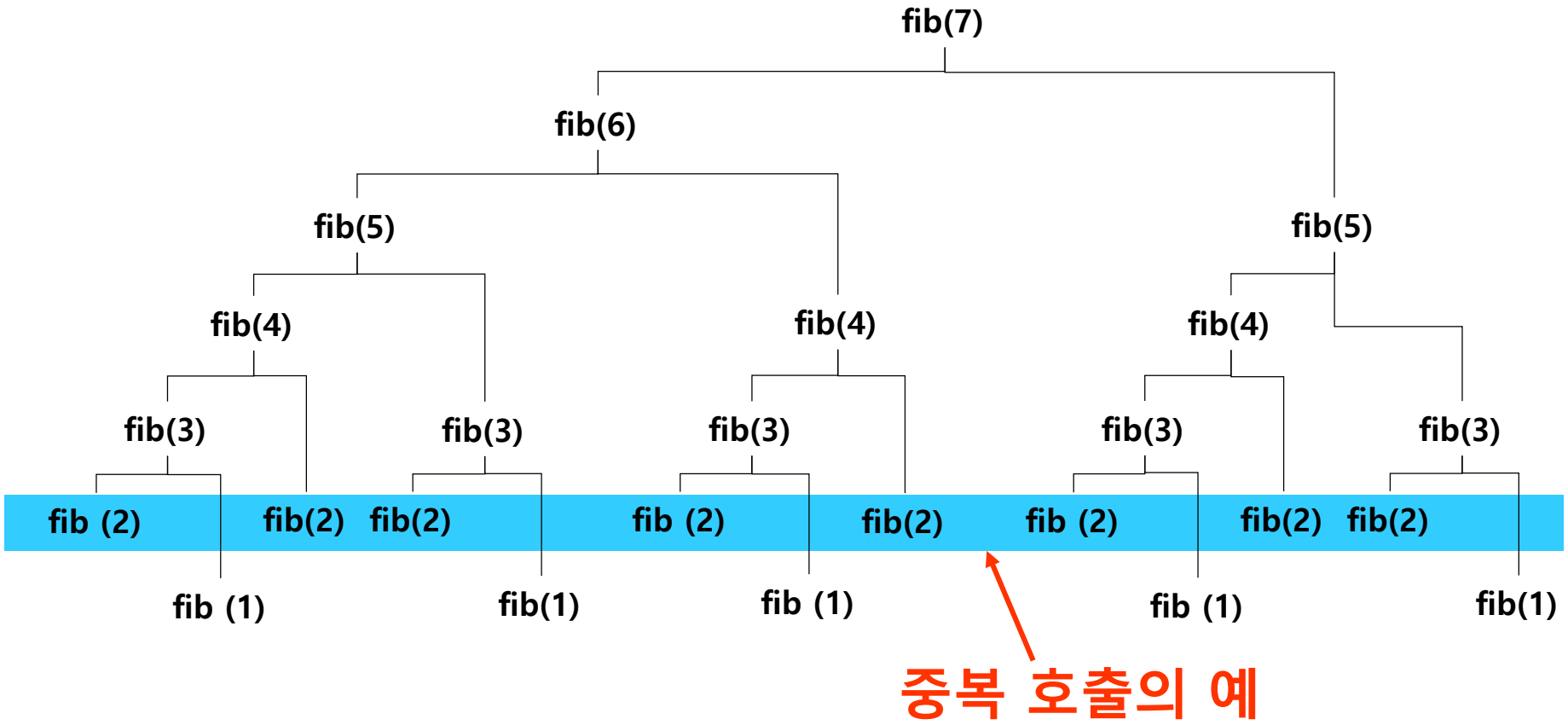
- 앞의 예에서 피보나치 수를 구하는 함수를 재귀함수로 구현한 알고리즘은 문제점이 있다.

- “엄청난 중복 호출이 존재한다”는 것이다.

Memoization

- 피보나치 수열의 Call Tree

$\Theta(2^n)$



Memoization

- 앞의 예에서 피보나치 수를 구하는 알고리즘에서 fibo(n)의 값을 계산하자마자 저장하면(memoize), 실행시간을 Θ(n)으로 줄일 수 있다.

- Memoization 방법을 적용한 알고리즘은 다음과 같다.

```
memo를 위한 배열을 할당하고, 모두 0으로 초기화 한다;
memo[0]을 0으로 memo[1]는 1로 초기화 한다;

public static int mFibo(int n) {
    if(n >=2 && memo[n] ==0) {
        memo[n] = mFibo(n-1)+mFibo(n-2);
    }
    return memo[n];
}
```

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미