데이터구조 2장

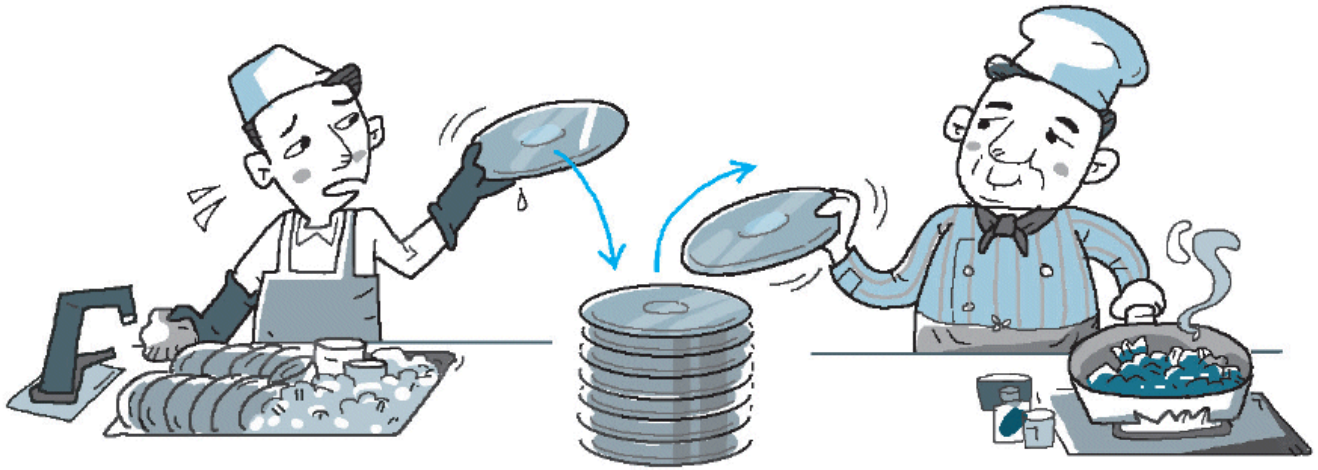
02-1. 스택개념과 연산

스택(STACK)이란?

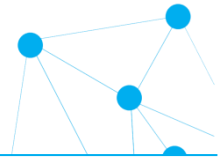
- 스택 : 목록의 한쪽 끝으로만 자료의 삽입, 삭제 작업이 이루어지는 자료구조

중간에서 작업불가능

- **후입선출(LIFO:Last-In First-Out)방식**
  - 가장 최근에(나중에) 들어온 데이터가 가장 먼저 나감



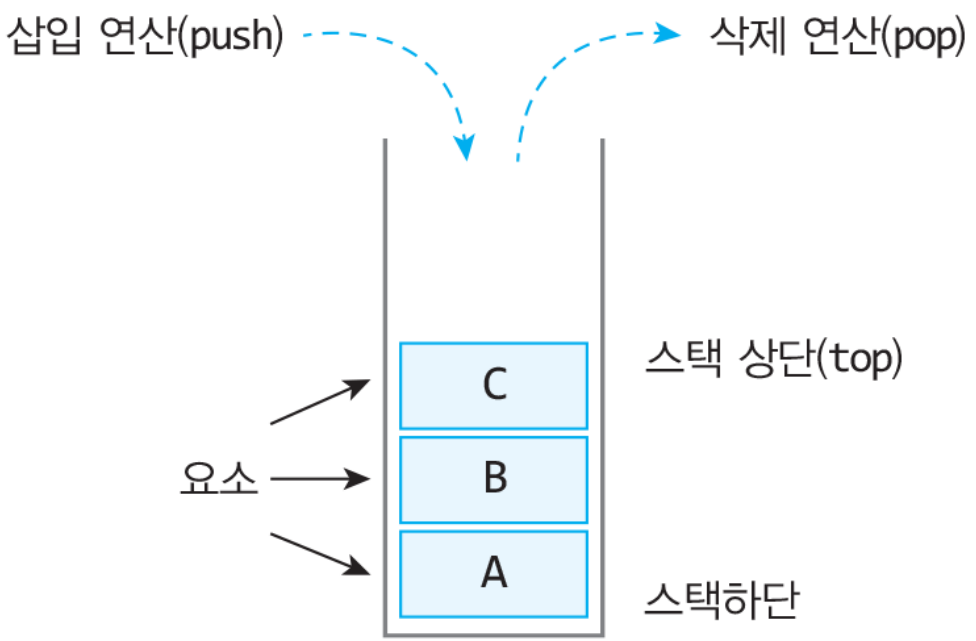
스택의 용도



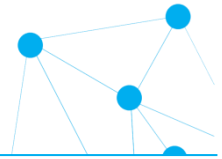
- 함수호출의 순서제어
- 인터럽트 발생시 복귀주소 저장
- 텍스트편집기의 Undo기능
- 컴파일러의 괄호검사
- 후위표기의 수식계산
- 미로탐색 등

스택의 구조

- 이해가 필요한 용어
  - 스택 상단: top
  - 스택 하단: bottom (실제론 불필요)
  - 요소(=항목)
  - 공백상태, 포화상태
  - 삽입 push
  - 삭제 pop
  - 보기 peek



스택의 추상자료형



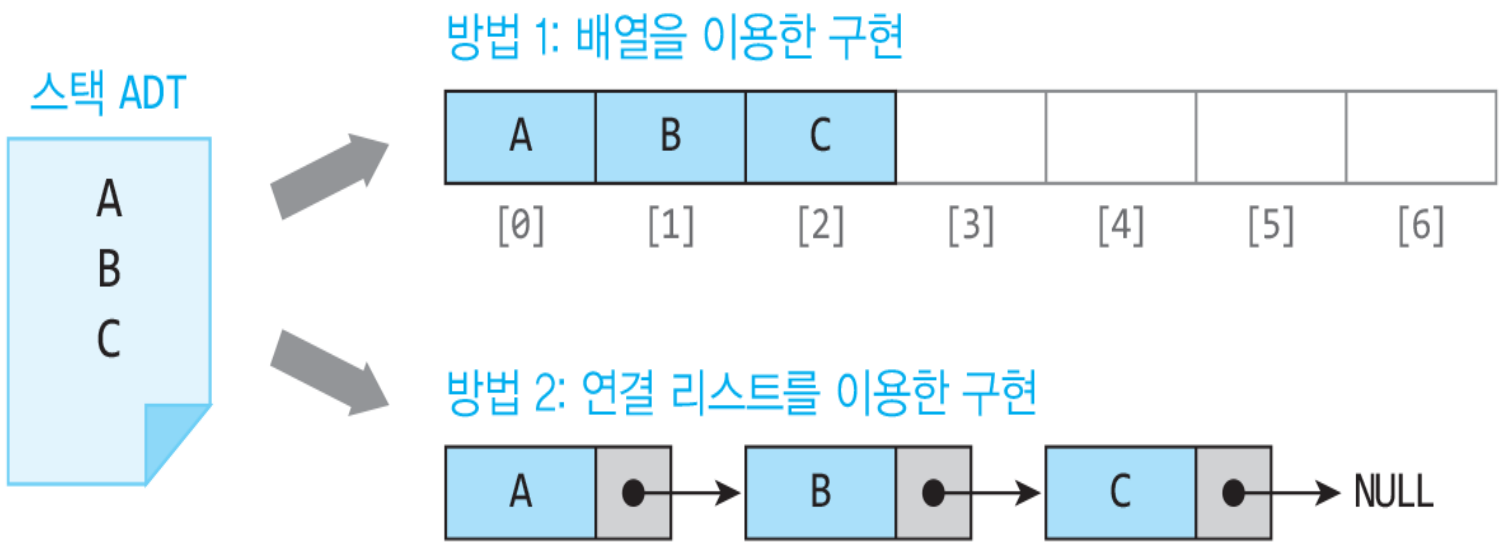
- **Stack ADT**

데이터: 후입선출(LIFO)의 접근 방법을 유지하는 요소들의 모음

연산:

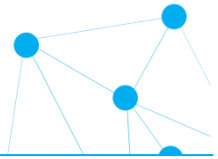
- is_empty(s): 스택(s)이 비어있으면 TRUE를 아니면 FALSE를 반환한다.
- is_full(s): 스택이 가득 차 있으면 TRUE를 아니면 FALSE을 반환한다.
- push(s, x): 주어진 요소 x를 스택의 맨 위에 추가한다.
- pop(s): 스택 맨 위에 있는 요소를 제거하고 반환한다.
- peek(s): 스택 맨 위에 있는 요소를 제거하지 않고 반환한다.

스택 구현 방법

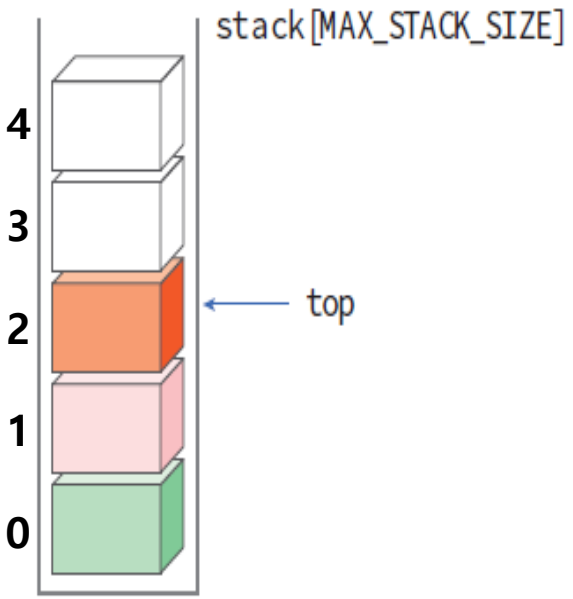


저장된 모양이 아니라 저장한 뒤 운영하는 방식에 집중

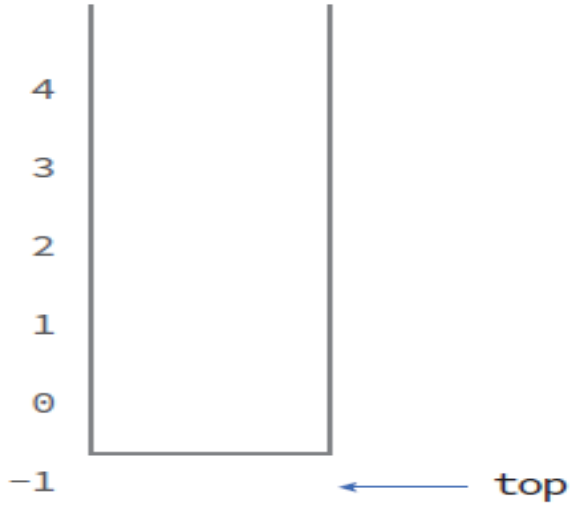
배열을 이용한 스택의 구현



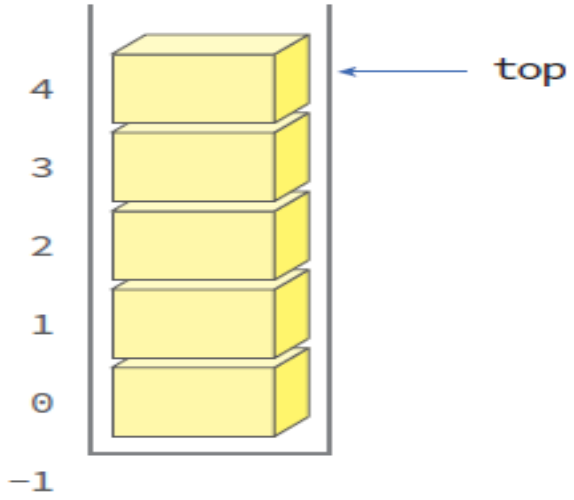
- 1차원 배열 : stack[MAX_STACK_SIZE]
- 정적 Array
  - 정해진 크기(정적)만 사용
- 스택에서 가장 최근에 입력되었던 자료를 가리키는 top 변수
  - 가장 먼저 들어온 요소는 stack[0]
  - 가장 최근에 들어온 요소는 stack[top]
  - 스택이 공백상태이면 top은 -1



스택의 연산(1)



(a) 공백상태

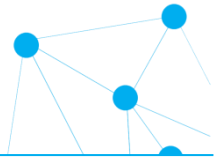


(b) 포화상태

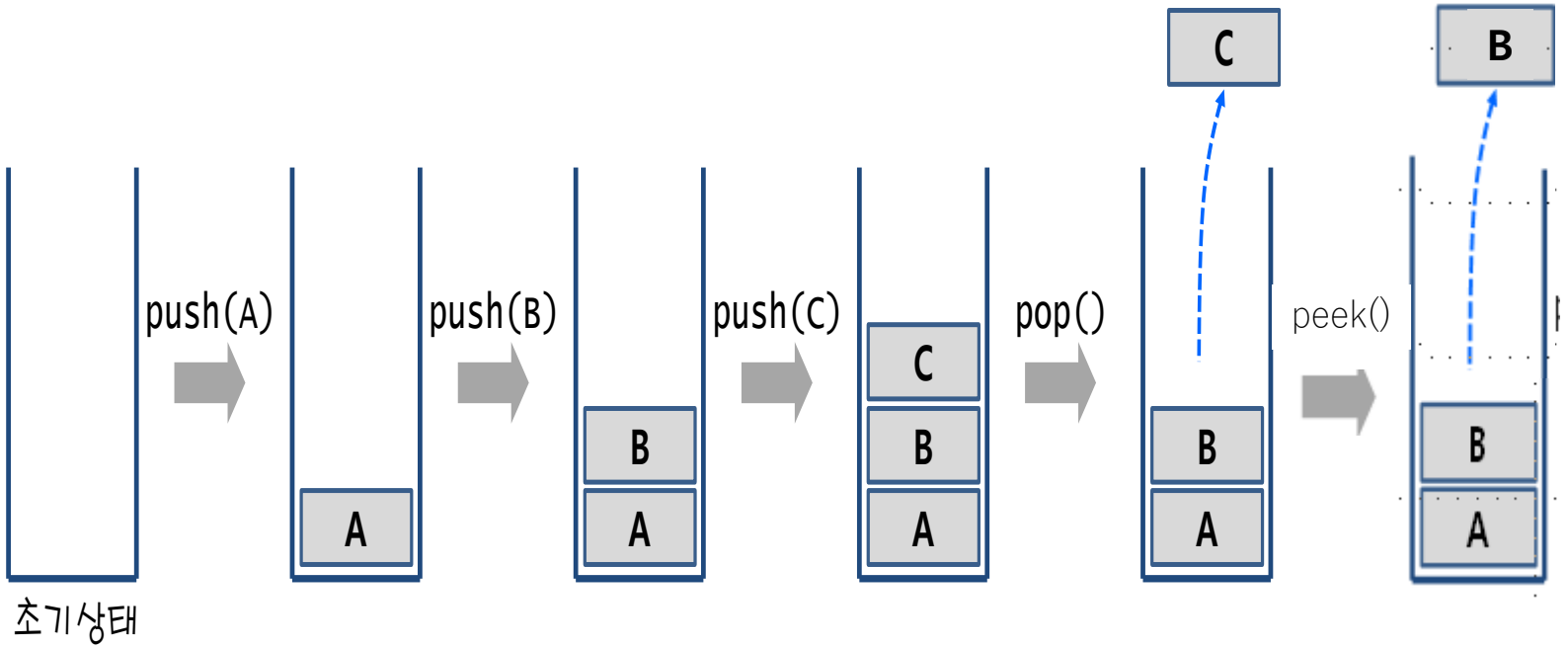
```
is_empty(S):

if top == -1 then
    return TRUE
else
    return FALSE
```

```
is_full(S):

if top >= (MAX_STACK_SIZE-1) then
    return TRUE
else
    return FALSE
```

스택의 연산(2)- 미리보기



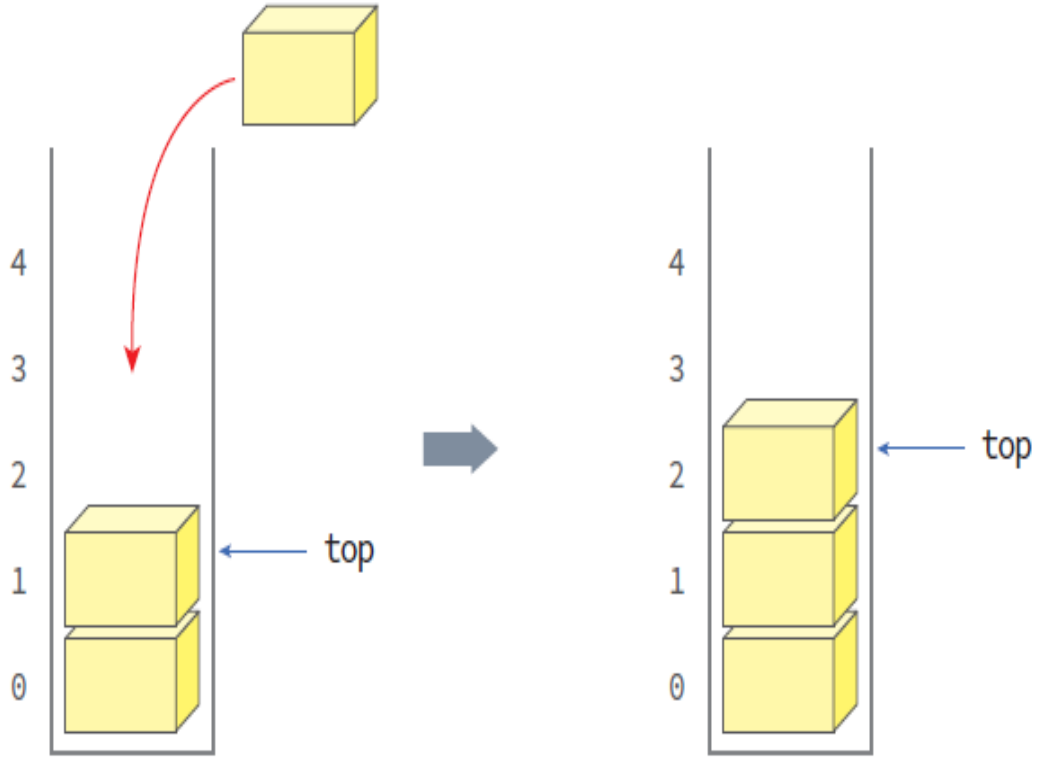
– **삽입(push), 삭제(pop),** 보기(peek)



스택의 push 연산

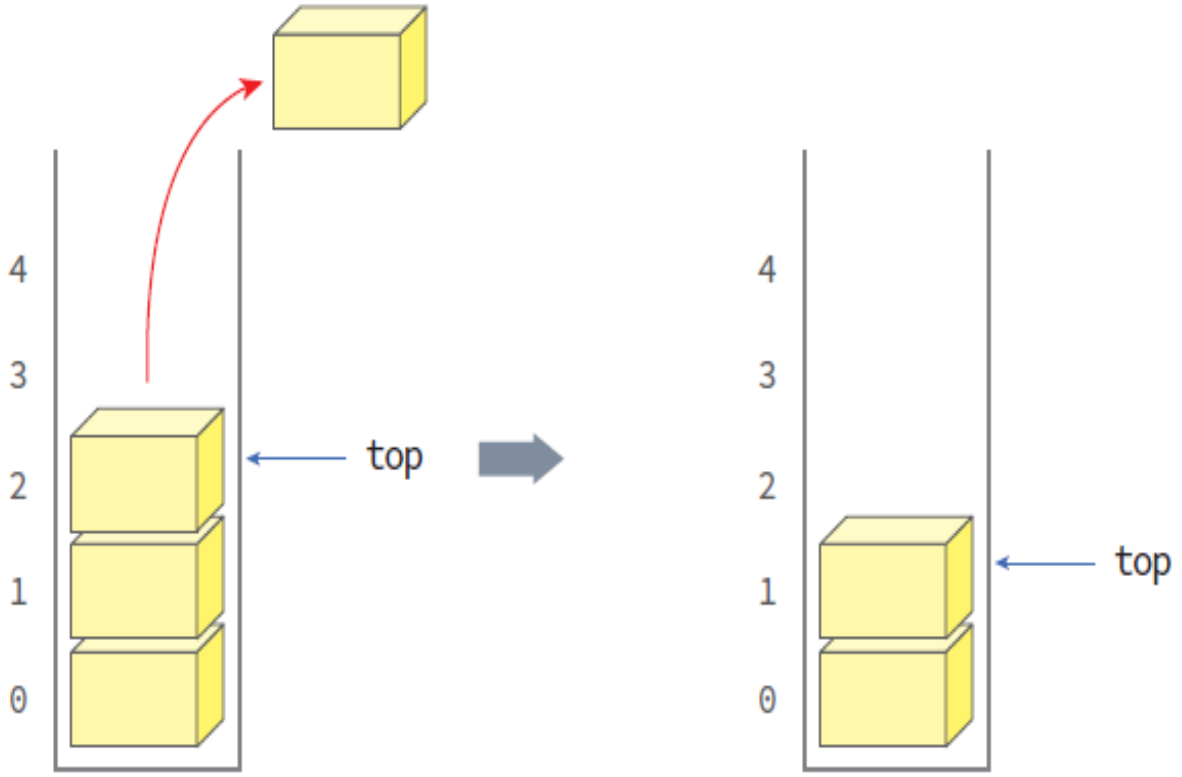
```
push(s, x):

if is_full(s) then
    error "overflow"
else
    top ← top+1
    s[top] ← x
```

top초기값을 0에서 출발
할수도 있음
s[top] ← x
top ← top+1



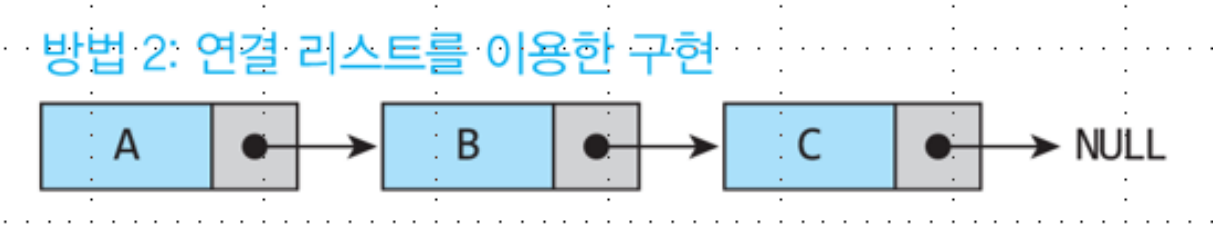
스택의 pop 연산

```
pop(S):

if is_empty(S) then
    error "underflow"
else
    e ← s[top]
    top ← top-1
    return e
```



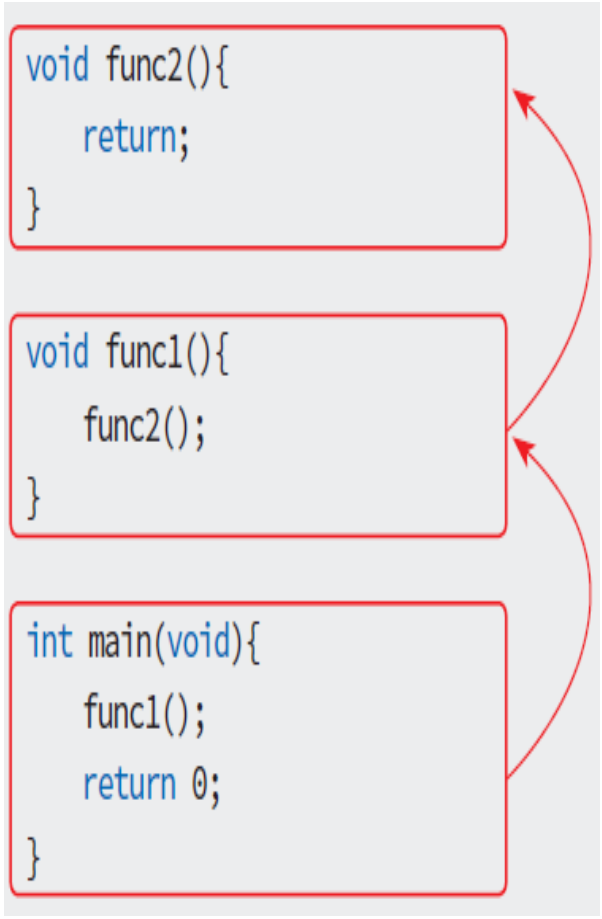
연결리스트를 이용한 스택의 구현

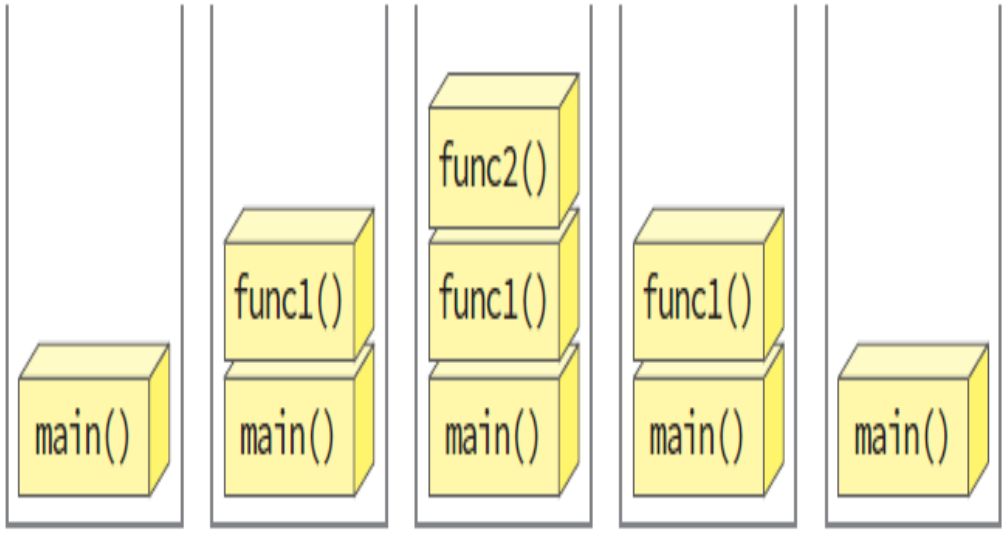
- 단순연결리스트, 원형연결리스트, 이중연결리스트 등의 형태로 구현 가능
- 동적 Array
  - 동적할당 방식을 이용해 구현
  - 스택에 데이터를 추가할 때마다 공간 동적메모리 할당, 삭제할 때마다 메모리 해제



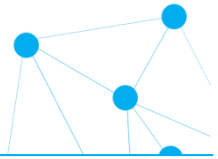
02-2. 스택응용

스택의 응용 : 함수호출 순서제어





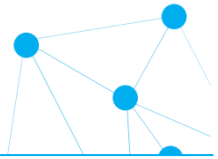
스택 응용 : 괄호검사



- 괄호의 종류: 대중소 (‘[’, ‘]’), (‘{’, ‘}’), (‘(’, ‘)’)
- 컴파일러가 프로그램에서 사용되는 괄호가 올바르게 사용되었는지 스택을 사용하여 검사

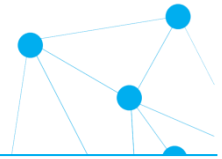
- 조건
  ① 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 한다.
  ② 같은 종류의 괄호에서 왼쪽 괄호는 오른쪽 괄호보다 먼저 나와야 한다.
  ③ 서로 다른 종류의 왼쪽 괄호와 오른쪽 괄호 쌍은 서로를 교차하면 안 된다.
- 잘못된 괄호 사용의 예
  - (a(b)
  - a(b)c)
  - a{b(c[d]e}f)

스택의 괄호검사 알고리즘



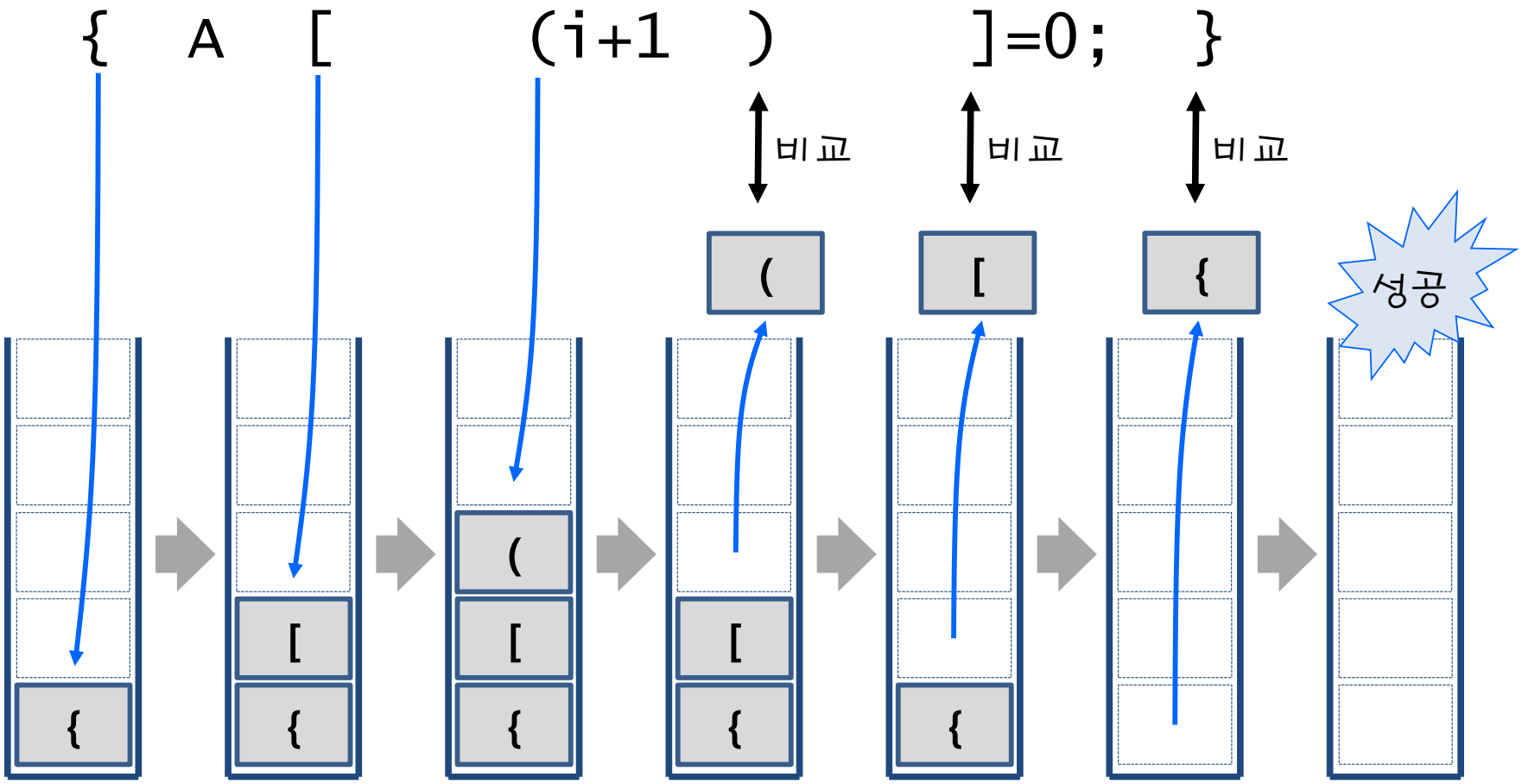
- 알고리즘의 개요
  - 문자열에 있는 괄호를 차례대로 조사하면서 왼쪽 괄호를 만나면 스택에 삽입하고, 오른쪽 괄호를 만나면 스택에서 top 괄호를 삭제한 후 오른쪽 괄호와 짝이 맞는지를 검사한다.
  - 이 때, 스택이 비어 있으면 조건 1 또는 조건 2 등을 위배하게 되고 괄호의 짝이 맞지 않으면 조건 3 등에 위배된다.
  - 마지막 괄호까지를 조사한 후에도 스택에 괄호가 남아 있으면 조건 1에 위배되므로 0(거짓)을 반환하고, 그렇지 않으면 1(참)을 반환한다.

스택의 괄호검사 알고리즘

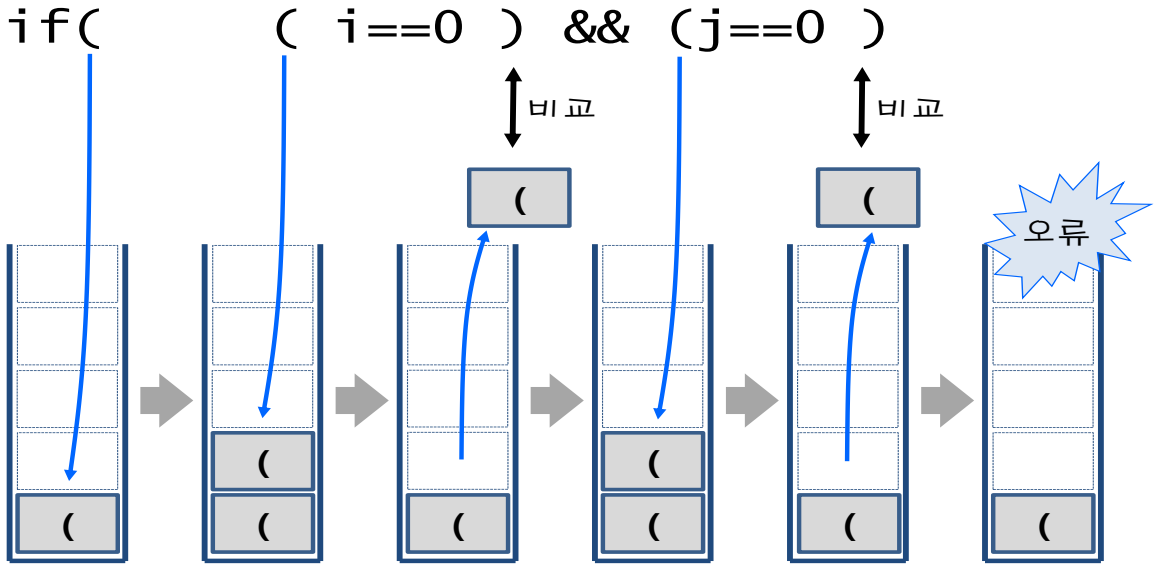


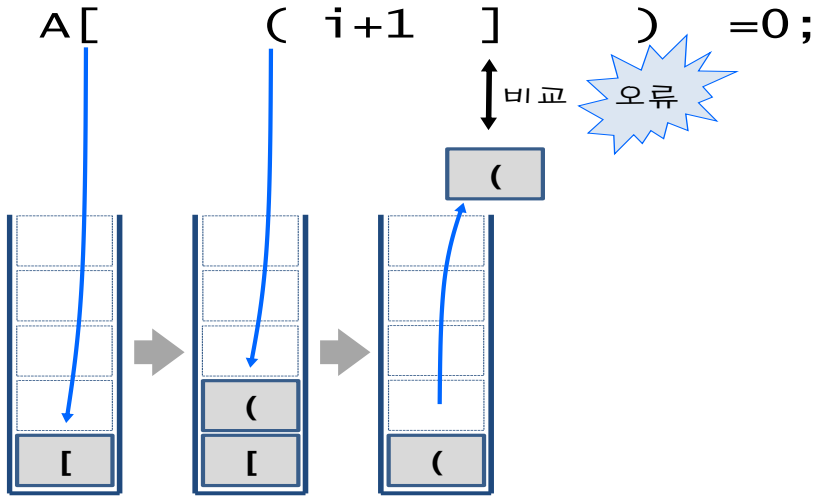
```
check_matching(expr)


while (입력 expr의 끝이 아니면)
  ch ← expr의 다음 글자
  switch(ch)
    case '(': case '[': case '{':
       ch를 스택에 삽입
       break
    case ')': case ']': case ']':
       if ( 스택이 비어 있으면 ) then
          오류
       else
          스택에서 open_ch를 꺼낸다
          if (ch 와 open_ch가 같은 짝이 아니면) then
              오류 보고
       break

if( 스택이 비어 있지 않으면 ) then
    오류
```

괄호검사 예

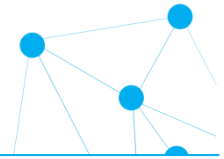


괄호검사 예





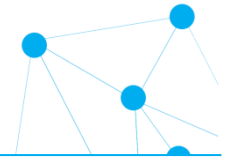
스택 응용 : 후위표기 수식의 계산



- 수식의 표기방법:
  - 중위(infix), 전위(prefix), 후위(postfix)

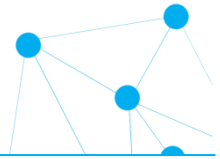
중위 표기법	전위 표기법	후위 표기법
2+3*4	+2*34	234*+
a*b+5	+*ab5	ab*5+
(1+2)+7	++127	12+7+

스택 응용 : 후위표기 수식의 계산



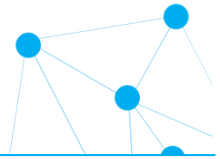
- 스택을 이용한 컴퓨터에서의 수식 계산순서
  - 중위표기식 → 후위표기식 → 계산
  - 2+3*4 → 234*+ → 14
  - 모두 스택을 사용

후위표기식의 계산알고리즘



```
Calc_postfix (expr)
스택 생성후 초기화
for item in expr
     if (항목이 피연산자이면) then
        push(s, item)
     else if (항목이 연산자 op이면) then
        second ← pop(s)
        first ← pop(s)
        temp ← first op second // op 는 +-*/중의 하나
        push(s, temp)
result ← pop(s)
```

후위표기수식의 계산

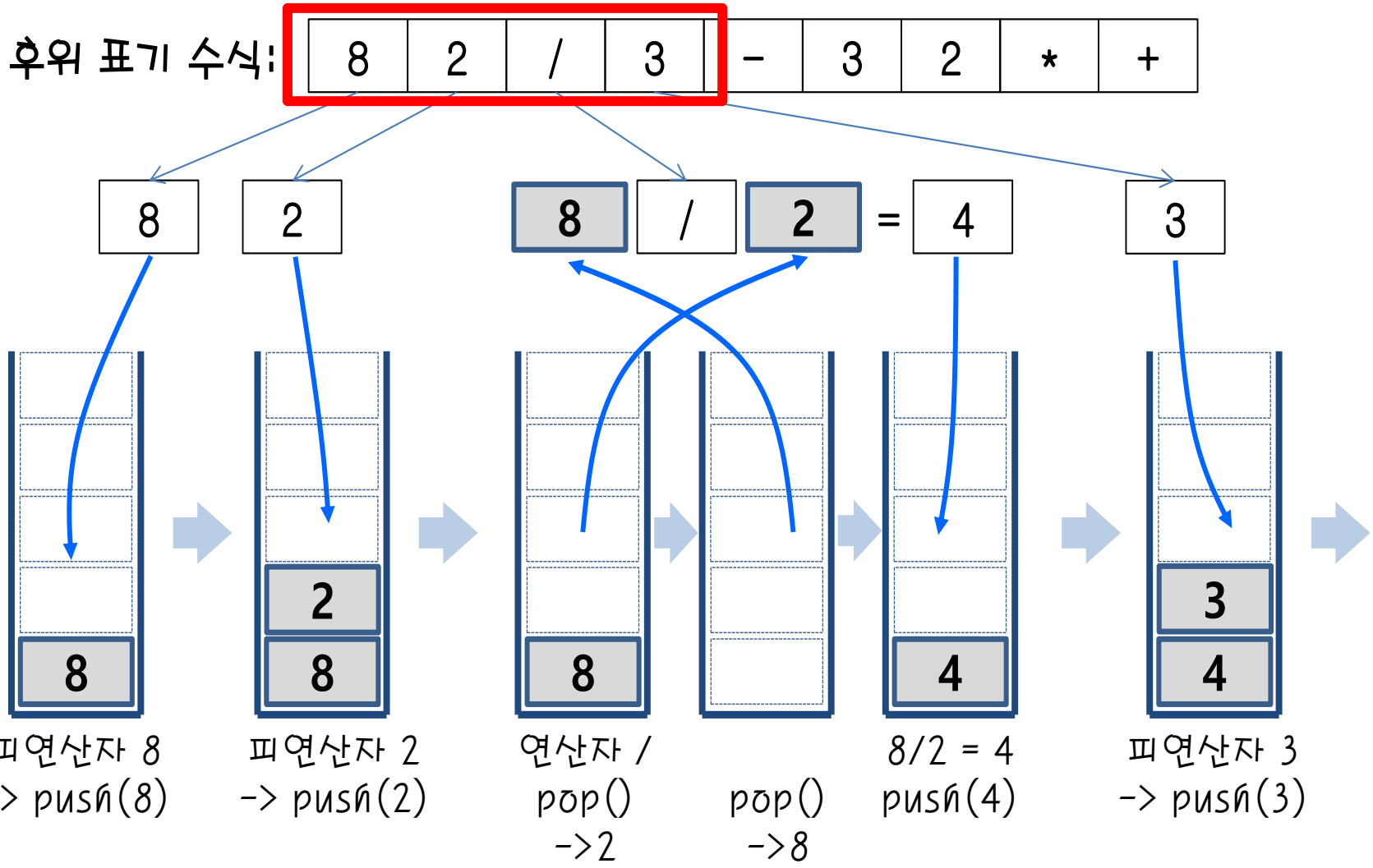


- 수식을 왼쪽에서 오른쪽으로 스캔하여
  - 피연산자이면 스택에 저장하고
  - 연산자이면 필요한 수만큼의 피연산자를 스택에서 꺼내 연산을 실행하고
  - 연산의 결과를 다시 스택에 저장

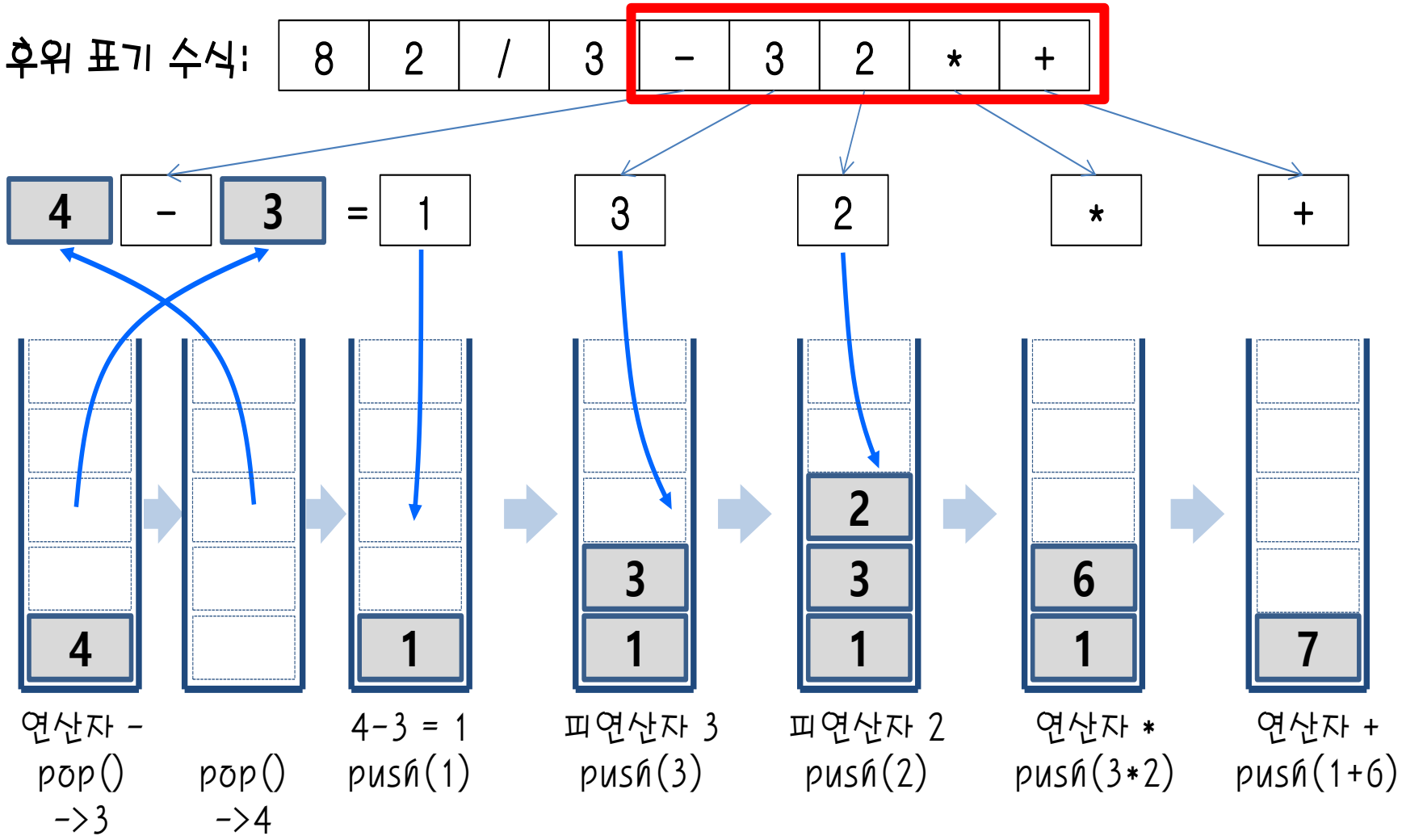
(연습) **82/3-32*+**

	스택						
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
8	8						
2	8	2					
/	4						
3	4	3					
-	1						
3	1	3					
2	1	3	2				
*	1	6					
+	7						

후위표기수식 계산 예


후위 표기 수식:
8 2 / 3 - 3 2 * +
8
2
8 / 2 = 4
3
피연산자 8
-> push(8)
피연산자 2
-> push(2)
연산자 /
pop()
->2
pop()
->8
8/2 = 4
push(4)
피연산자 3
-> push(3)

후위표기수식 계산 예

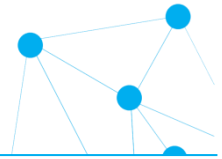

후위 표기 수식:
8
2
/
3
-
3
2
*
+
4
-
3
=
1
3
2
*
+
4
1
3
1
2
3
1
6
1
7
연산자 -
pop()
->3
pop()
->4
4-3 = 1
push(1)
피연산자 3
push(3)
피연산자 2
push(2)
연산자 *
push(3*2)
연산자 +
push(1+6)

스택응용 : 중위표기 수식의 후위표기 변환

- 중위표기와 후위표기
  - 중위와 후위 표기법의 공통점 : 피연산자의 순서가 동일
  - 연산자들의 순서만 다름(우선순위순서)
    - 연산자만 스택에 저장했다가 출력
    - 2+3*4 -> 234*+

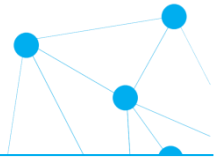
- 알고리즘
  - 피연산자를 만나면 그대로 출력
  - 연산자를 만나면 스택에 저장했다가 스택보다 우선 순위가 낮은 연산자가 나오면 그때 출력
  - 왼쪽 괄호는 우선순위가 가장 낮은 연산자로 취급
  - 오른쪽 괄호가 나오면 스택에서 왼쪽 괄호 위에 쌓여있는 모든 연산자를 출력

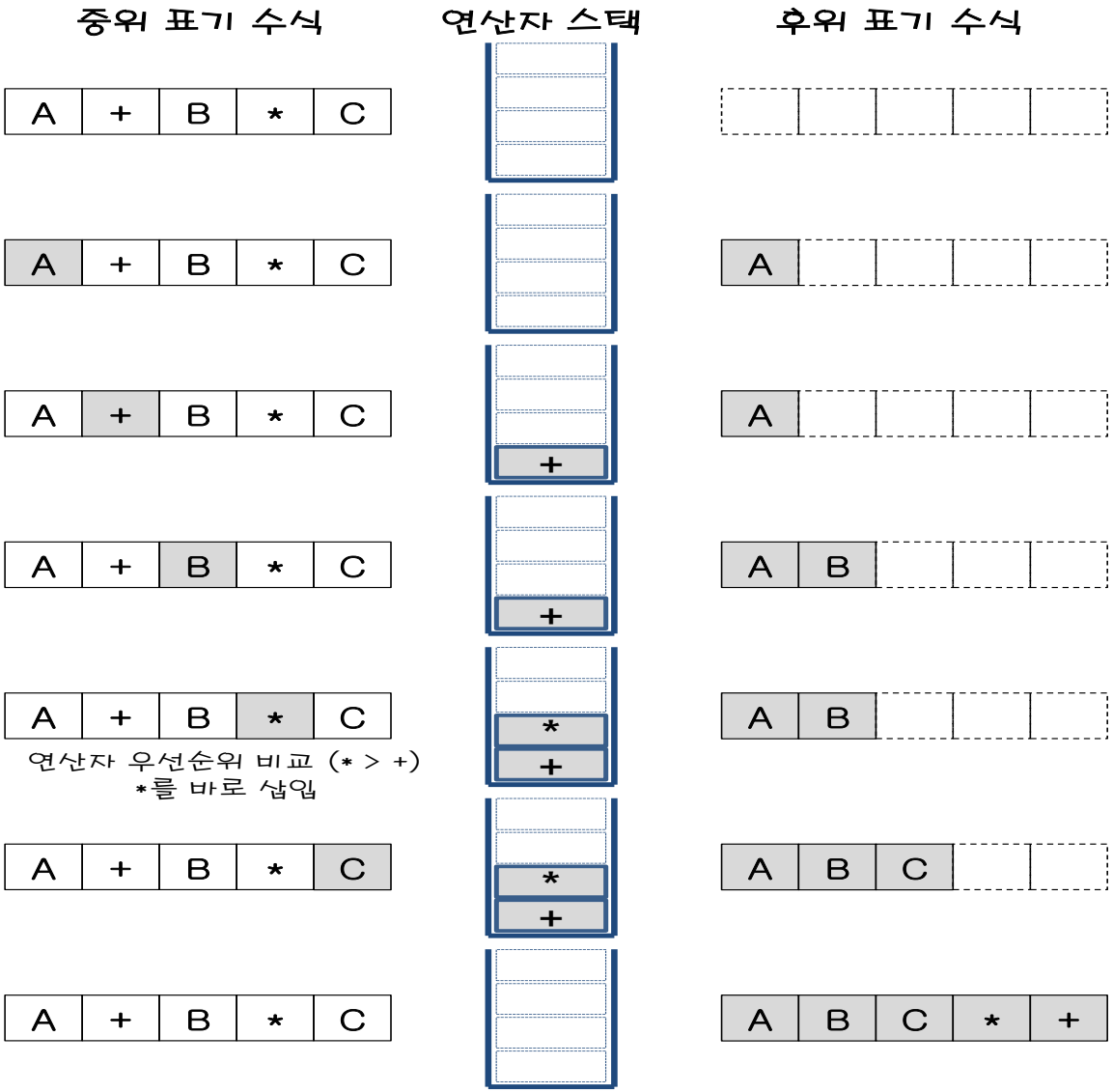
후위표기 변환 알고리즘



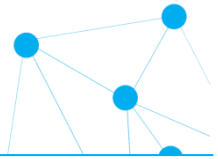
```
Infix_to_postfix(expr)

스택 초기화.
while (expr에 처리할 항이 남아 있으면)
     term ← 다음에 처리할 항;
     switch (term)
       case 피연산자:
         term을 출력;
         break;
       case 왼쪽 괄호:
         push(s, term);
         break;
       case 오른쪽 괄호:
         e ← pop(s);
         while( e ≠ 왼쪽 괄호 )
              e를 출력;
              e ← pop(s);
         break;
       case 연산자:
         while ( peek()의 우선순위 ≥ term의 우선순위 )
              e ← pop(s);
              e를 출력;
         push(s, term);
         break;
while( not is_empty() )
     e ← pop(s);
     e를 출력;
```

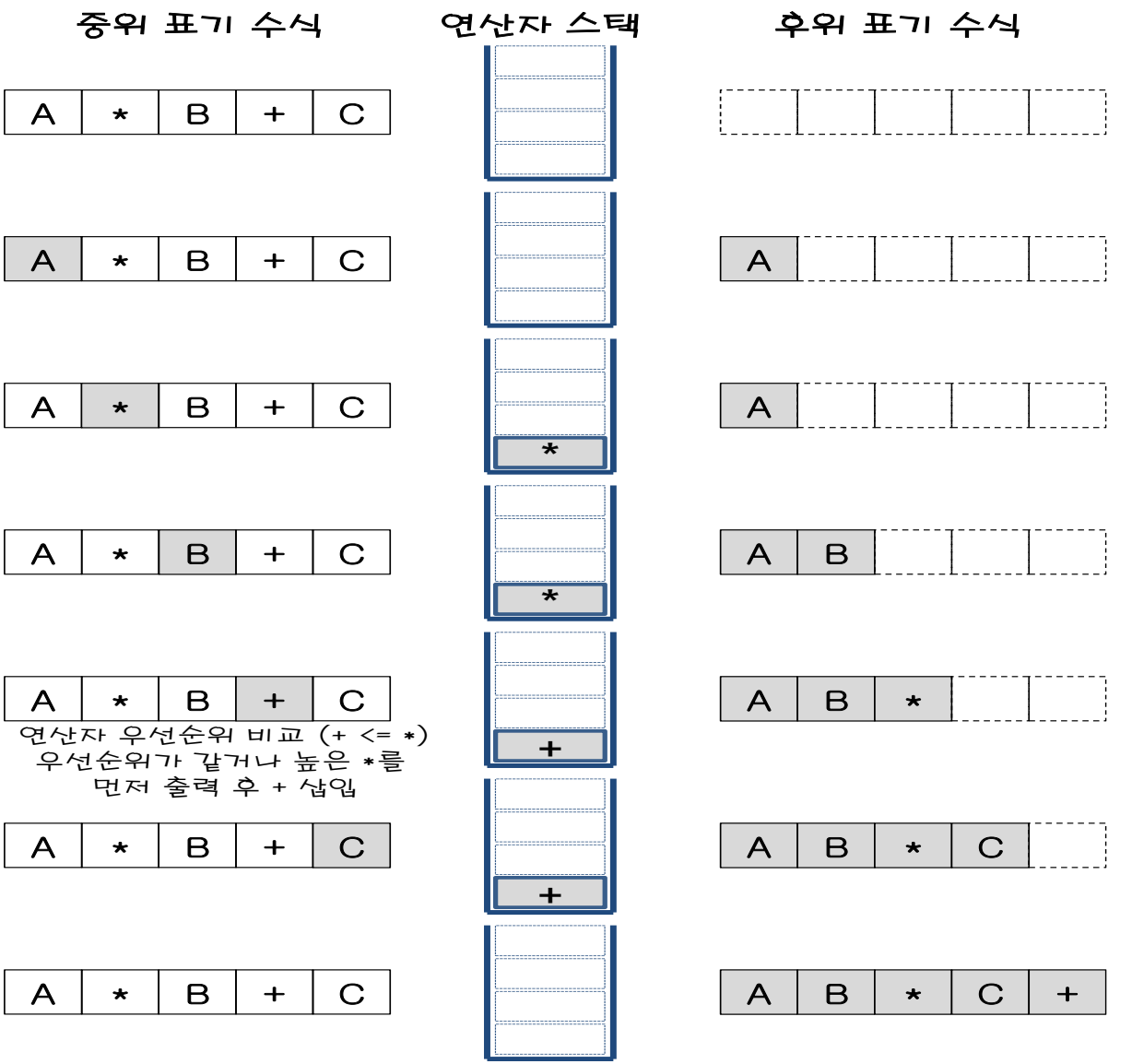
중위 → 후위 표기 변환 예



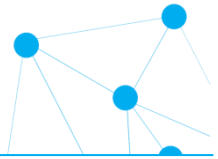

중위 표기 수식
연산자 스택
후위 표기 수식
A + B * C
A + B * C
A
A + B * C
+
A
A + B * C
+
A B
A + B * C
*
+
A B
연산자 우선순위 비교 (* > +)
*를 바로 삽입
A + B * C
*
+
A B C
A + B * C
A B C * +

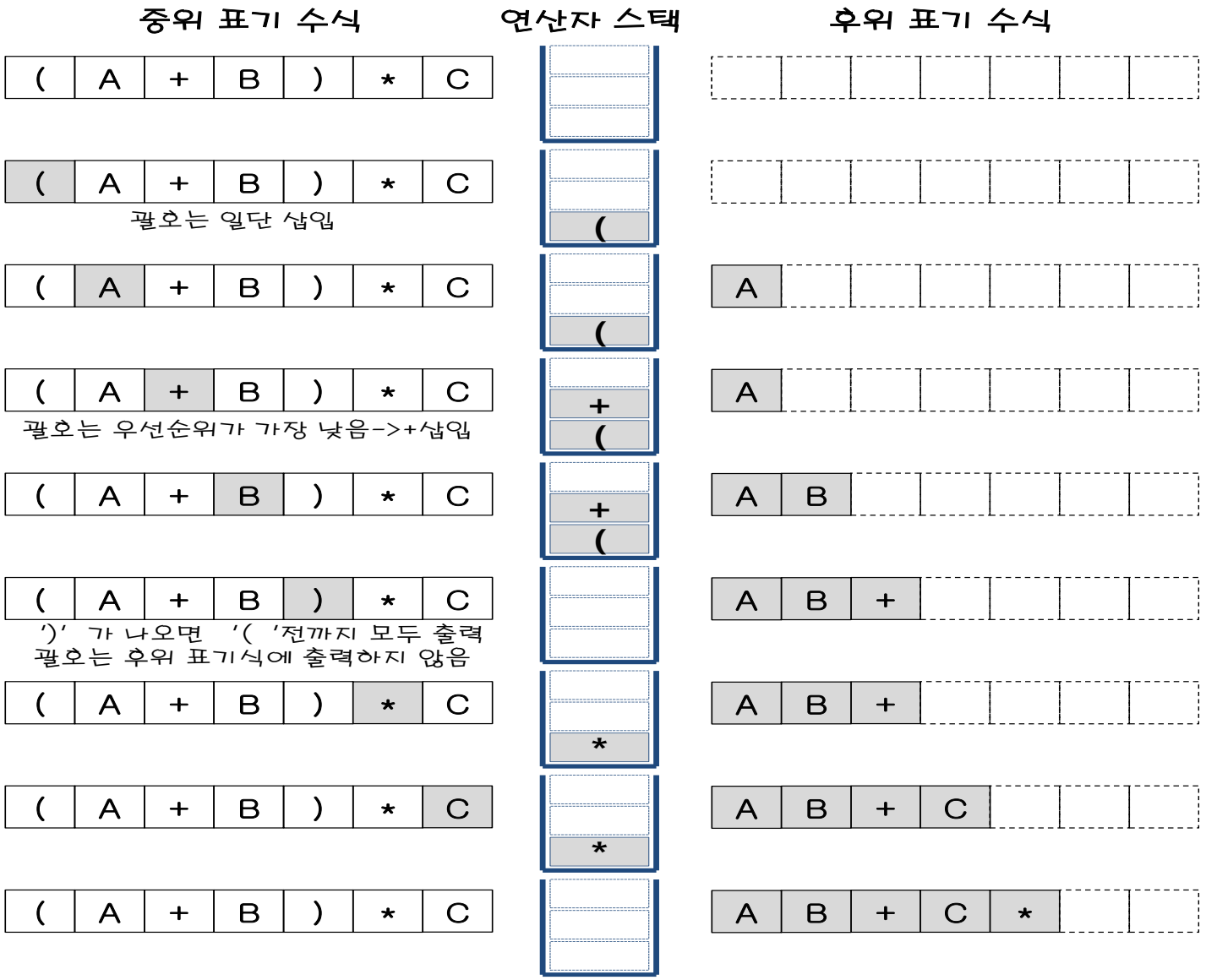
중위 → 후위 표기 변환 예




중위 표기 수식
연산자 스택
후위 표기 수식
A * B + C
A * B + C
A
A * B + C
*
A
A * B + C
*
A B
A * B + C
+
A B *
연산자 우선순위 비교 (+ <= *)
우선순위가 같거나 높은 *를
먼저 출력 후 + 삽입
A * B + C
+
A B * C
A * B + C
A B * C +

중위 → 후위 표기 변환 예




중위 표기 수식
연산자 스택
후위 표기 수식
(A + B) * C
괄호는 일단 삽입
괄호는 우선순위가 가장 낮음->+삽입
')' 가 나오면 '(' 전까지 모두 출력
괄호는 후위 표기식에 출력하지 않음