6주차 스택(Stack) 응용

3.2 스택의 응용

- 컴파일러의 괄호 짝 맞추기
- 회문(Palindrome) 검사하기
- 후위표기법(Postfix Notation) 수식 계산하기
- 중위표기법(Infix Notation) 수식의 후위표기법 변환
- Undo기능
- 미로 찾기
- 트리의 방문
- 그래프의 깊이우선탐색
- 프로그래밍에서 매우 중요한 함수/메소드 호출 및 재귀호출도 스택 자료구조를 바탕으로 구현

회문 검사하기

- 회문(Palindrome): 앞으로부터 읽으나 뒤로부터 읽으나 동일한 스트링

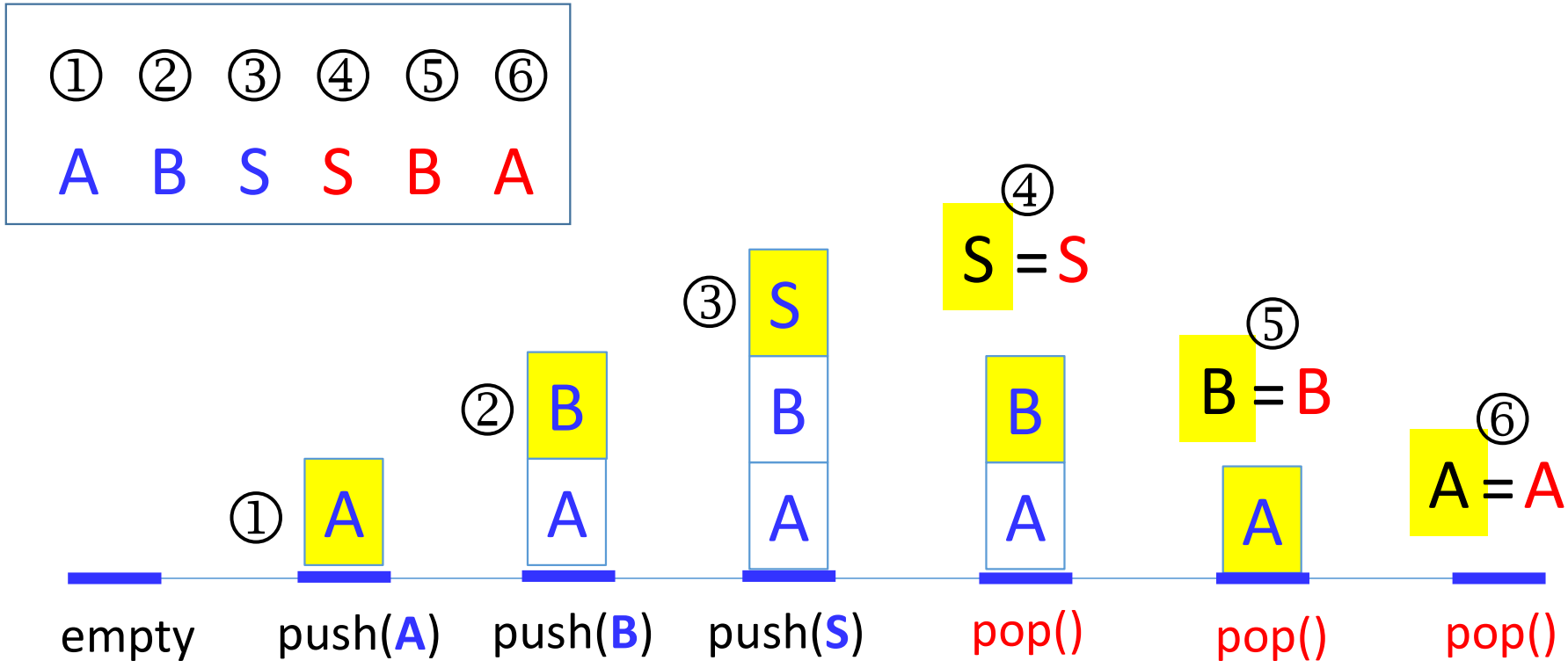
[핵심 아이디어] 전반부의 문자들을 스택에 push한 후, 후반부의 각 문자를 차례로 pop한 문자와 비교

- 회문 검사하기는 주어진 스트링의 앞부분 반을 차례대로 읽어 스택에 push한 후, 문자열의 길이가 짝수이면 뒷부분의 문자 1 개를 읽을 때마다 pop하여 읽어 들인 문자와 pop된 문자를 비교하는 과정을 반복 수행

- 만약 마지막 비교까지 두 문자가 동일하고 스택이 empty가 되면, 입력 문자열은 회문

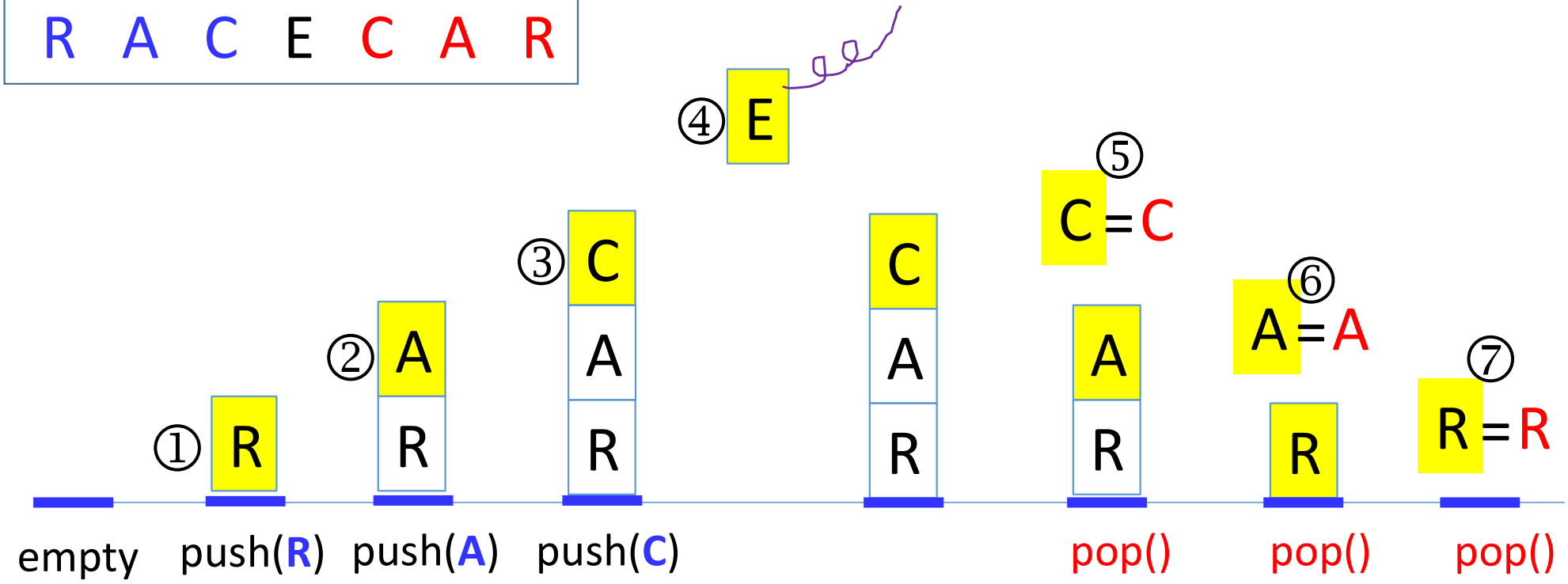
- 문자열의 길이가 홀수인 경우, 주어진 스트링의 앞부분 반을 차례로 읽어 스택에 push한 후, 중간 문자를 읽고 버린다. 이후 짝수 경우와 동일하게 비교 수행

[예제 1]



[예제 2]

①	②	③	④	⑤	⑥	⑦
R	A	C	E	C	A	R



수식의 표기법

- 프로그램을 작성할 때 수식에서 +, –, *, /와 같은 이항연산자는 2개의 피연산자들 사이에 위치
- 이러한 방식의 수식 표현이 중위표기법(Infix Notation)
- 컴파일러는 중위표기법 수식을 후위표기법(Postfix Notation)으로 바꾼다.
  - 그 이유는 후위표기법 수식은 괄호 없이 중위표기법 수식을 표현할 수 있기 때문
- 전위표기법(Prefix Notation): 연산자를 피연산자들 앞에 두는 표기법

중위표기법 수식과 대응되는 후위표기법, 전위표기법 수식

중위표기법	후위표기법	전위표기법
A + B	A B +	+ A B
A + B – C	A B + C –	–+ A B C
A + B * C – D	A B C * + D –	– + A * B C D
(A + B) / (C – D)	A B+ C D – /	/ + A B – C D

후위표기법 수식 계산

- [핵심 아이디어] 피연산자는 스택에 push하고, 연산자는 2회 pop하여 계산한 후 push

후위표기법으로 표현된 수식 계산 알고리즘

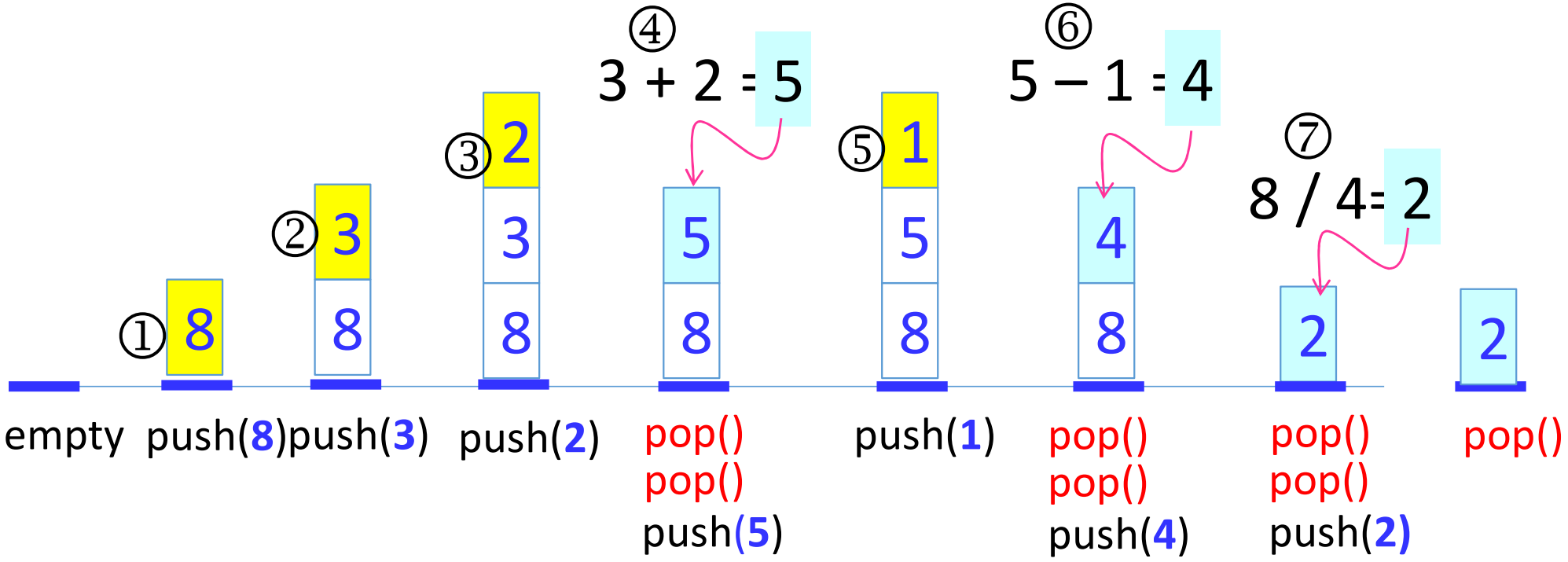
- 입력을 좌에서 우로 문자를 한 개씩 읽는다. 읽은 문자를 C라고하면

  [1] C가 피연산자이면 스택에 push

  [2] C가 연산자(op)이면 pop을 2회 수행한다. 먼저 pop된 피연산자가 B이고, 나중에 pop된 피연산자가 A라면, (A op B)를 수행하여 그 결과 값을 push

[예제]

①	②	③	④	⑤	⑥	⑦
8	3	2	+	1	−	/

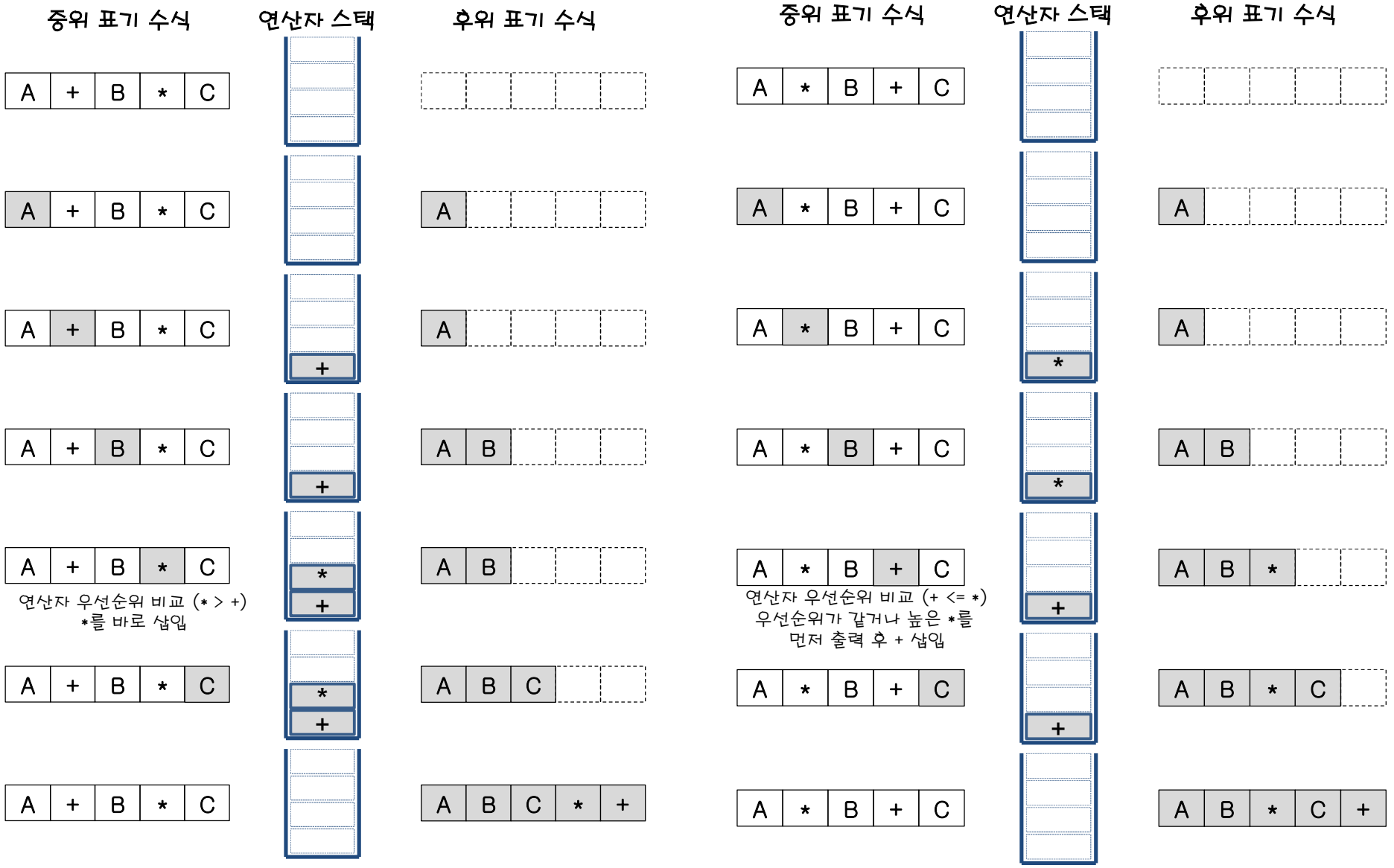


중위표기법 수식을 후위표기법으로 변환

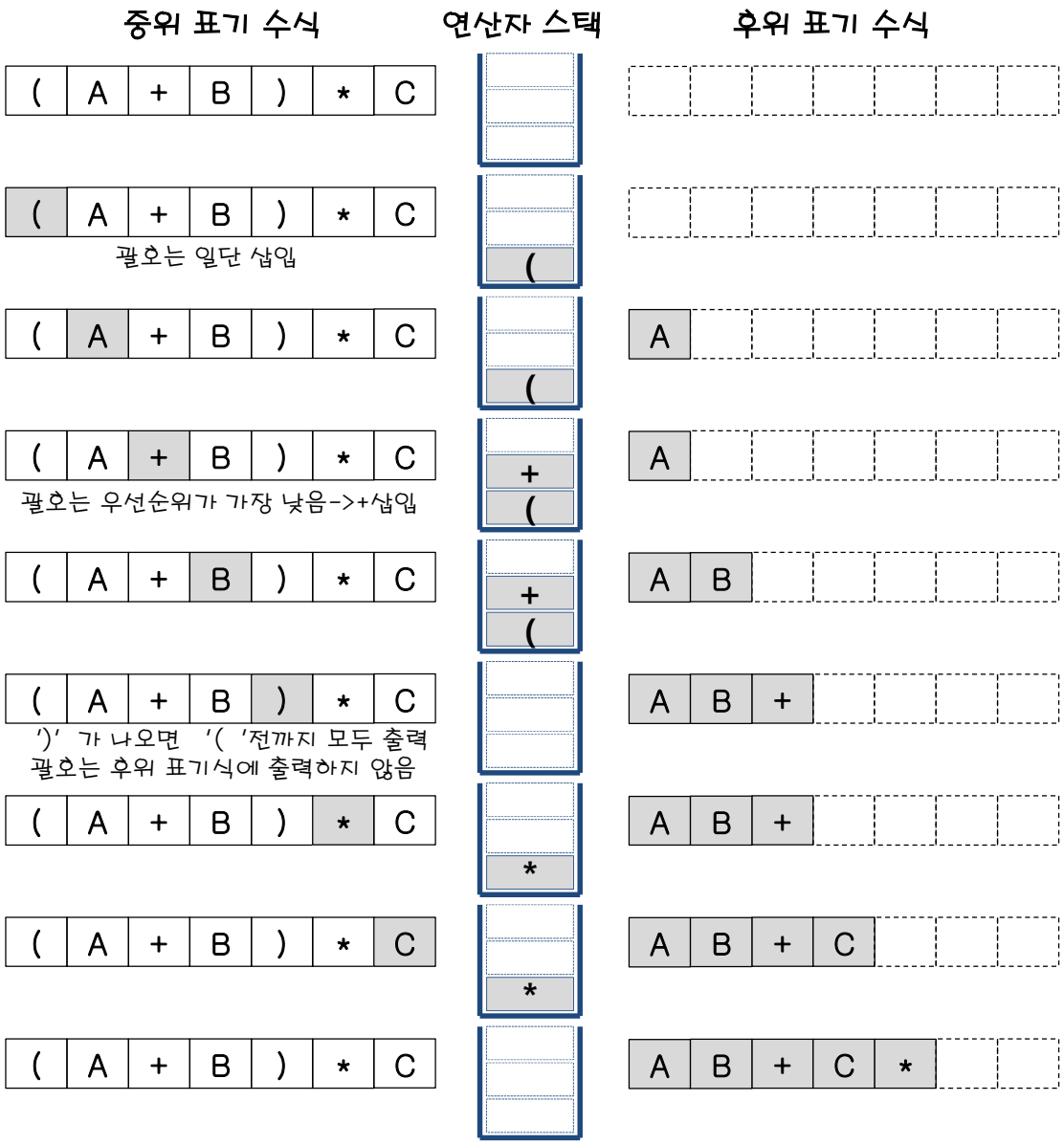
- [핵심 아이디어] 왼쪽 괄호나 연산자는 스택에 push하고, 피연산자는 출력

- 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 괄호위에 쌓여있는 모든 연산자를 출력

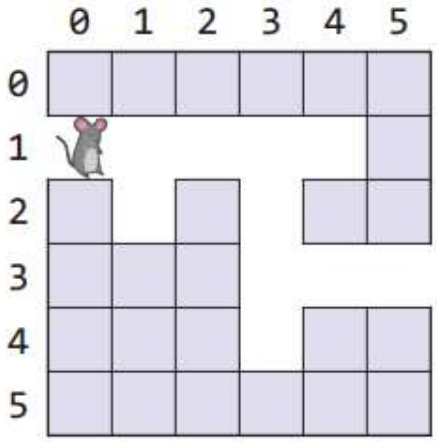
중위 → 후위 표기 변환 예


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

후위 표기 변환 예

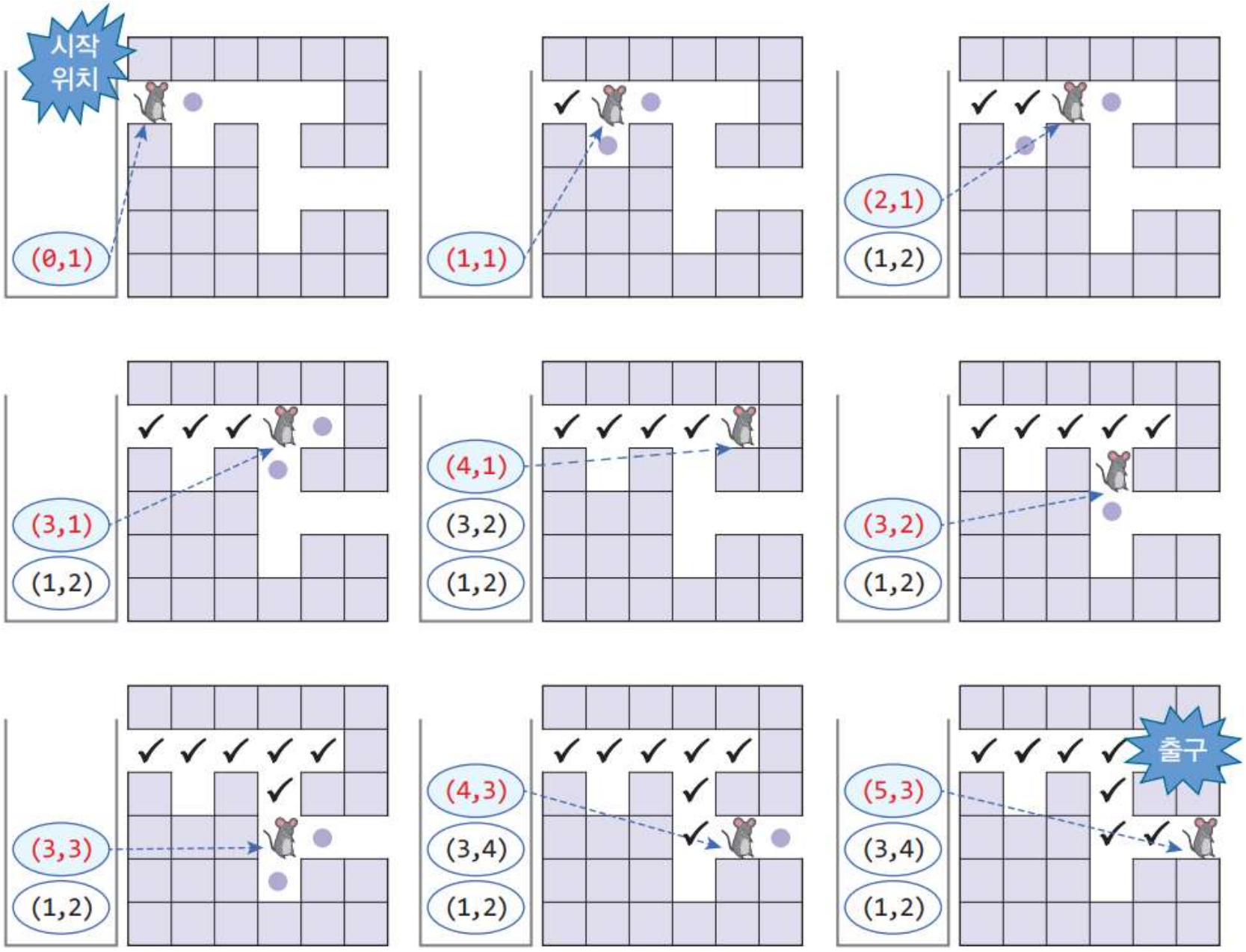

중위 표기 수식
연산자 스택
후위 표기 수식
(A + B) * C
괄호는 일단 삽입
괄호는 우선순위가 가장 낮음->+삽입
')' 가 나오면 '(' 전까지 모두 출력
괄호는 후위 표기식에 출력하지 않음

스택의 응용: 미로 탐색




```
map = [ [ '1', '1', '1', '1', '1', '1' ],
        [ 'e', '0', '0', '0', '0', '1' ],
        [ '1', '0', '1', '0', '1', '1' ],
        [ '1', '1', '1', '0', '0', 'x' ],
        [ '1', '1', '1', '0', '1', '1' ],
        [ '1', '1', '1', '1', '1', '1' ]]
MAZE_SIZE = 6
```

깊이우선탐색: 스택 사용



깊이우선탐색 알고리즘

```
def DFS() :                          # 깊이우선탐색 함수
    stack = Stack()                  # 사용할 스택 객체를 준비
    stack.push( (0,1) )              # 시작위치 삽입. (0,1)은 튜플
    print('DFS: ')

    while not stack.isEmpty():       # 공백이 아닐 동안
        here = stack.pop()           # 항목을 꺼냄(pop)
        print(here, end='->')
        (x, y) = here                # 스택에 저장된 튜플은 (x,y) 순서임.
        if (map[y][x] == 'x') :      # 출구이면 탐색 성공. True 반환
            return True
        else :
            map[y][x] = '.'          # 현재위치를 지나왔다고 '.'표시
            # 4방향의 이웃을 검사해 갈 수 있으면 스택에 삽입
            if isValidPos(x, y - 1): stack.push((x, y - 1)) # 상
            if isValidPos(x, y + 1): stack.push((x, y + 1)) # 하
            if isValidPos(x - 1, y): stack.push((x - 1, y)) # 좌
            if isValidPos(x + 1, y): stack.push((x + 1, y)) # 우
        print(' 현재 스택: ', stack.top() )  # 현재 스택 내용 출력
    return False                     # 탐색 실패. False 반환
```

stack.top

테스트 프로그램

```
result = DFS()
if result : print(' --> 미로탐색 성공')
else : print(' --> 미로탐색 실패')
```