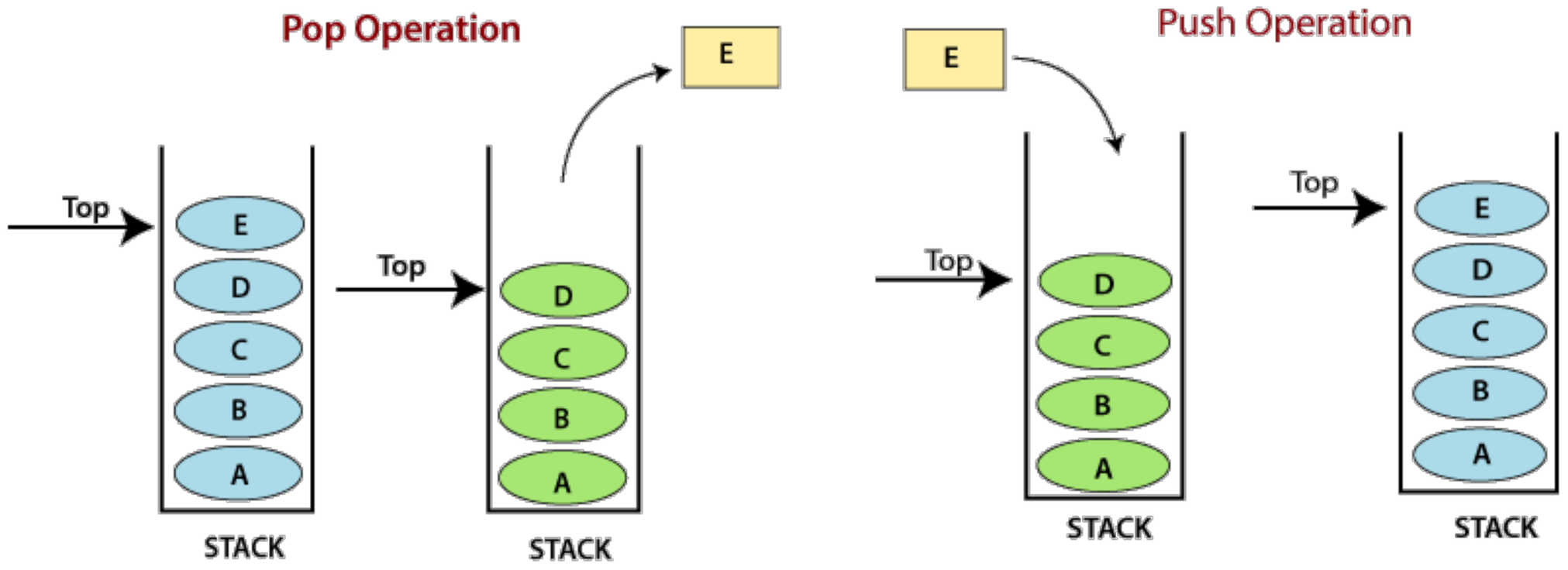


Stack

Jin Hyun Kim

Fall, 2019

References

- https://www.javatpoint.com/python-stack-and-queue

스택 (Stack)

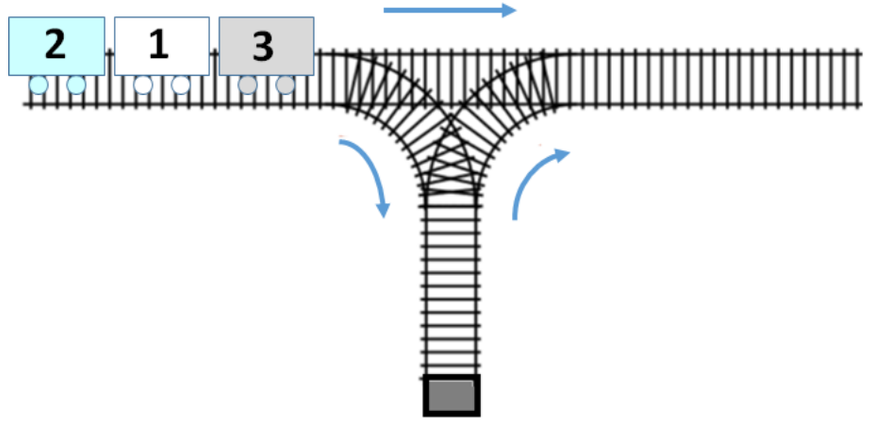
- 한 쪽 끝에서만 item(항목)을 삭제하거나 새로운 item을 저장하는 자료구조

- 새 item을 저장하는 연산: push

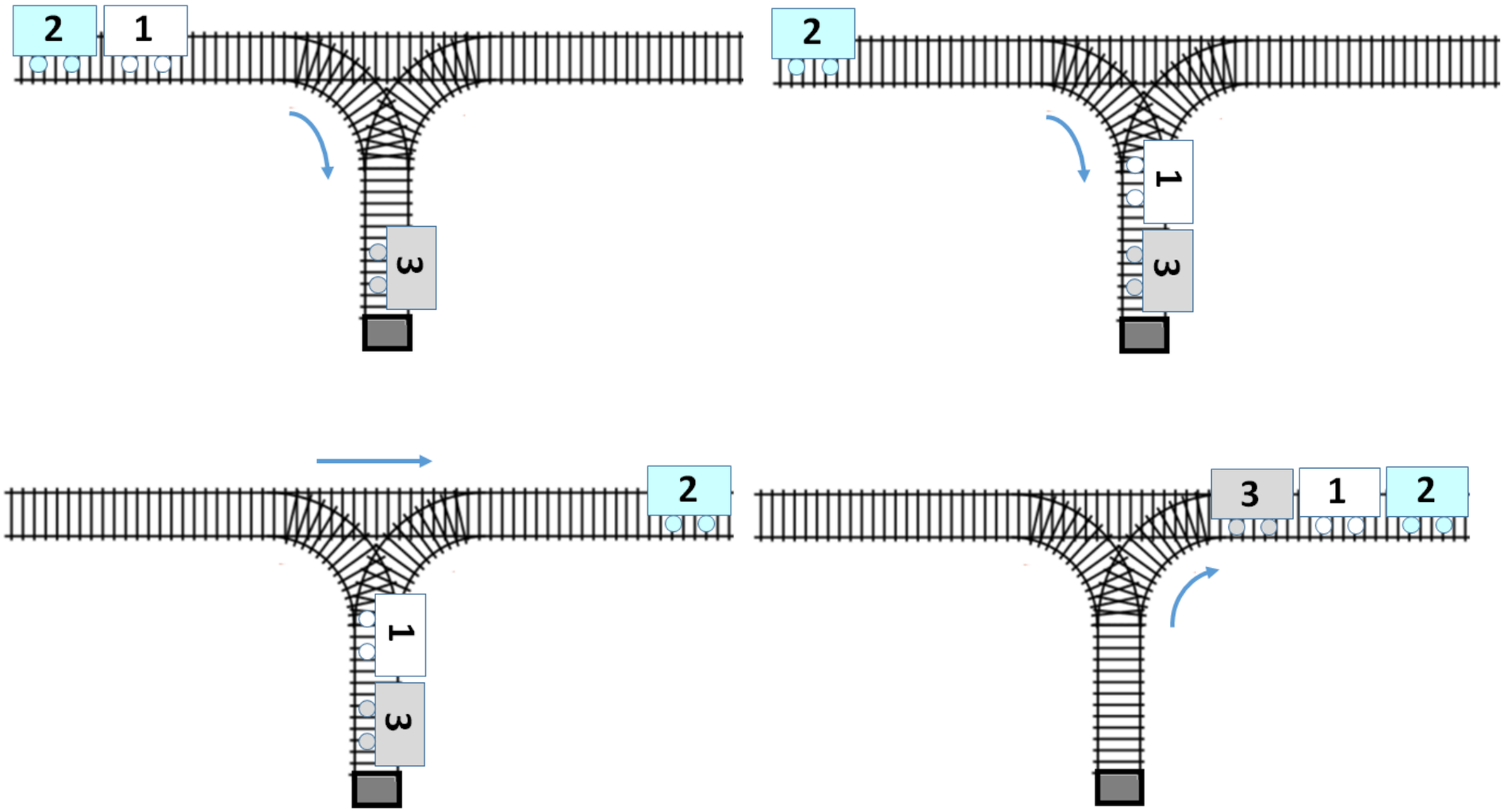
- Top item을 삭제하는 연산: pop

- 후입 선출(Last-In First-Out, LIFO) 원칙 하에 item의 삽입과 삭제 수행

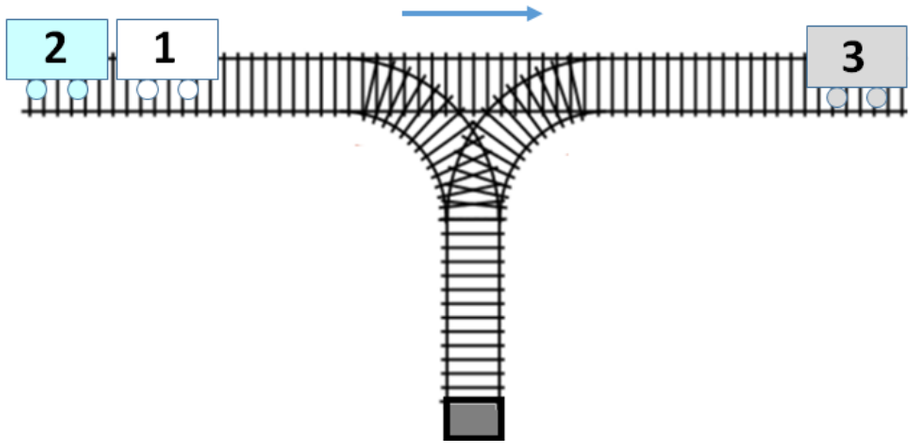
열차순서바꾸기

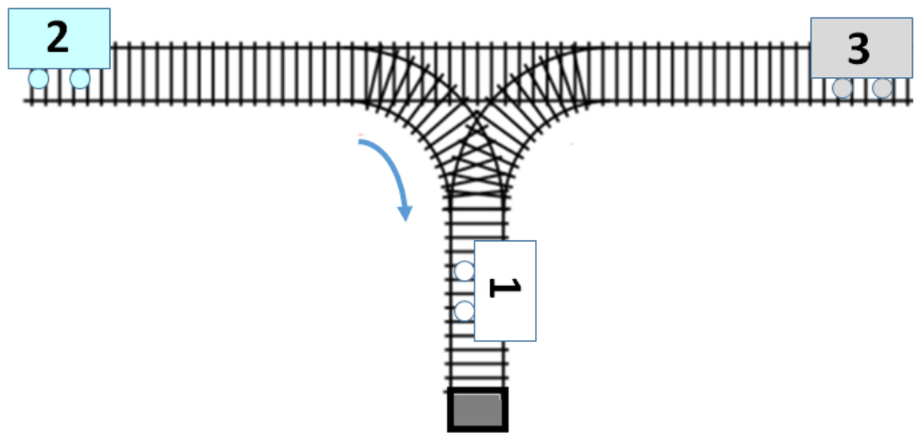


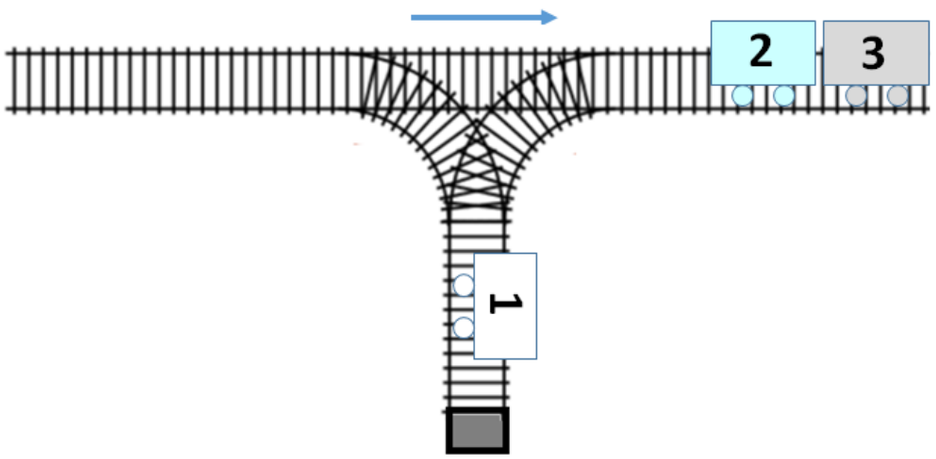
스택

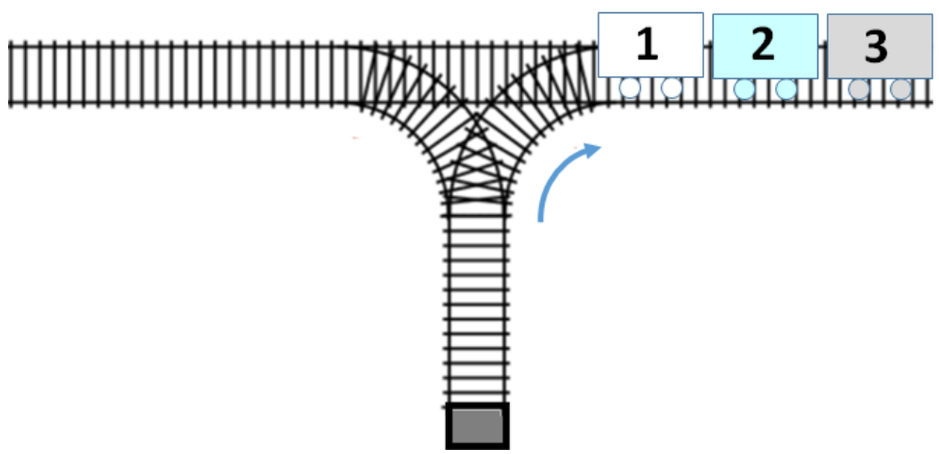


스택

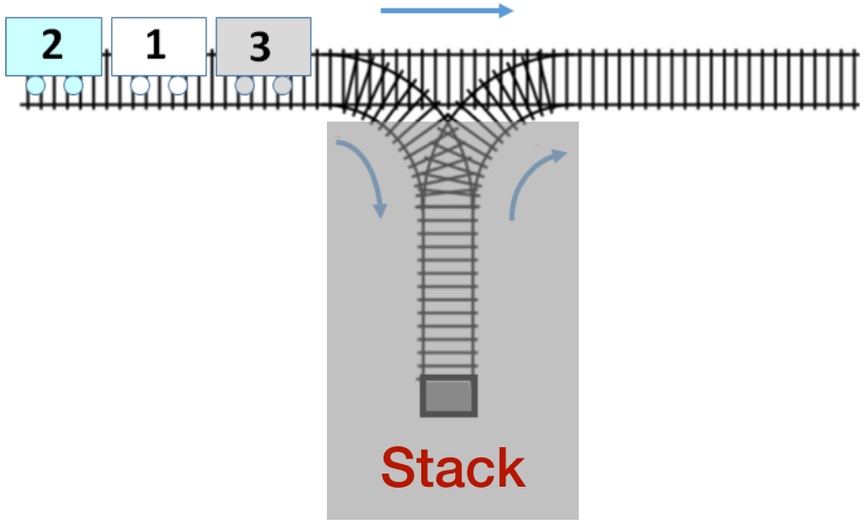


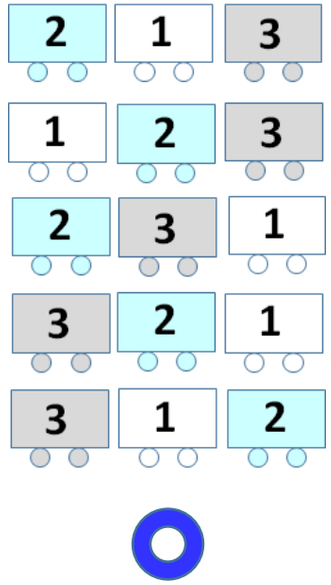






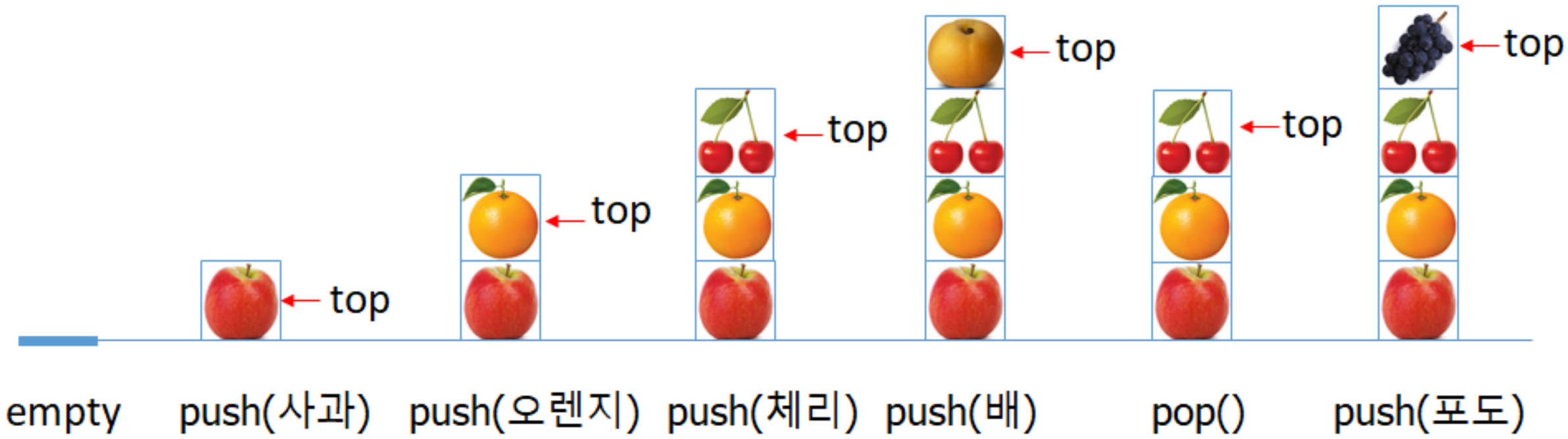
열차순서바꾸기





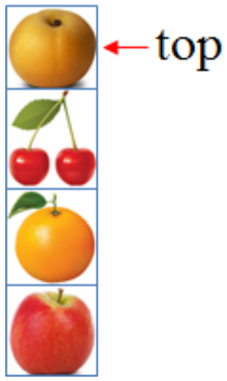
스택의 연산들

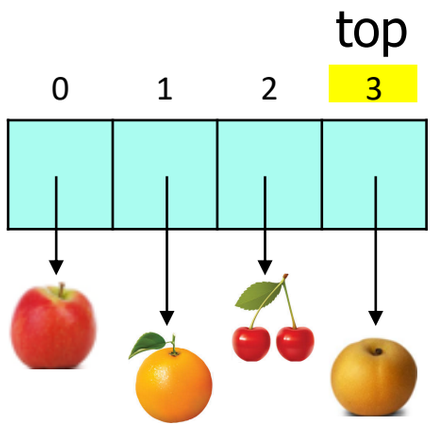
- empty()
- puch(item)
- pop()

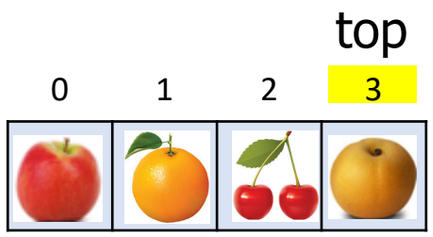


스택의 구현

- 리스트로 구현

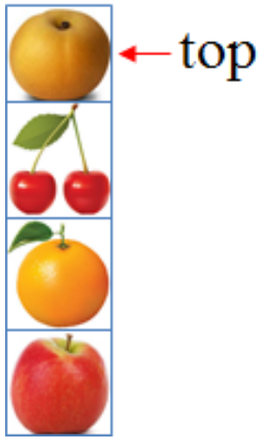


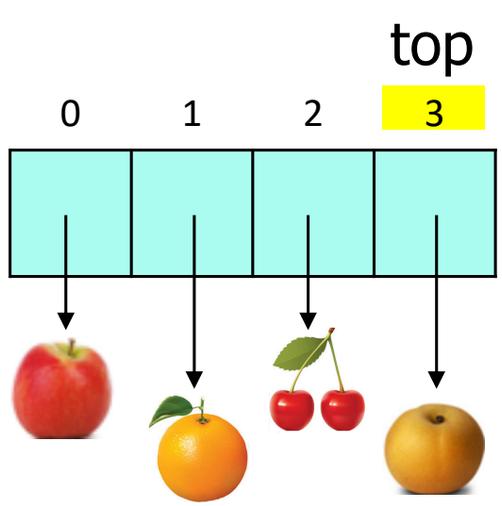


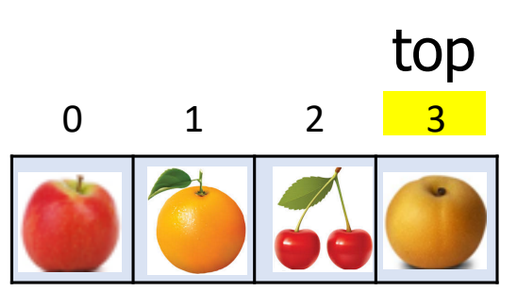


스택의 구현

- 리스트 (배열 혹은 연결리스트) 로 구현







연결리스트로 만든 스택 실험

```
34 top = None
35 size = 0
36 push('apple')
37 push('orange')
38 push('cherry')
39 print('사과, 오렌지, 체리 push 후:\t', end='')
40 print_stack()
41 print('top 항목: ', end='')
42 print(peek())
43 push('pear')
44 print('배 push 후:\t\t', end='')
45 print_stack()
46 pop()
47 push('grape')
48 print('pop(), 포도 push 후:\t', end='')
49 print_stack()
```

초기화

프로그램 3-1과 동일함

배열 (리스트, 파이썬) 이용

```
01 def push(item): # 삽입 연산
02     stack.append(item)
03
04 def peek(): # top 항목 접근
05     if len(stack) != 0:
06         return stack[-1]
07
08 def pop(): # 삭제 연산
09     if len(stack) != 0:
10         item = stack.pop(-1)
11         return item
12 stack = []
```

push() = append()
리스트의 맨 뒤에 item 추가

top 항목
= 리스트의 맨 뒤 항목 리턴

pop()
리스트의 맨 뒤에 있는 항목 제거

리스트 선언

배열로 만든 스택 실험

```
13 push('apple')
14 push('orange')
15 push('cherry')
16 print('사과, 오렌지, 체리 push 후:\t', end='')
17 print(stack, '\t<- top')
18 print('top 항목: ', end='')
19 print(peek())
20 push('pear')
21 print('배 push 후:\t\t', end='')
22 print(stack, '\t<- top')
23 pop()
24 push('grape')
25 print('pop(), 포도 push 후:\t', end='')
26 print(stack, '\t<- top')
```

일련의 스택 연산과 출력

단순연결리트로 구현

```
01 class Node: # Node 클래스
02     def __init__(self, item, link):
03         self.item = item
04         self.next = link
05
06 def push(item): # push 연산
07     global top
08     global size
09     top = Node(item, top)
10     size += 1
11
12 def peek(): # peek 연산
13     if size != 0:
14         return top.item
15
```

노드 생성자
항목과 다음 노드 레퍼런스

전역 변수

새 노드 객체를 생성하여
연결리스트의 첫 노드로 삽입

top 항목만 리턴

단순연결리트로 구현

```
16 def pop(): # pop 연산
17     global top
18     global size
19     if size != 0:
20         top_item = top.item
21         top = top.next
22         size -= 1
23         return top_item

24 def print_stack(): # 스택 출력
25     print('top ->\t', end='')
26     p = top
27     while p:
28         if p.next != None:
29             print(p.item, '-> ', end='')
30         else:
31             print(p.item, end='')
32         p = p.next
33     print()
```

전역 변수

연결리스트에서 top이 참조하던 노드 분리시킴

제거된 top 항목 리턴

수행시간

- 파이썬의 리스트로 구현한 스택의 push와 pop 연산은 각각 O(1) 시간이 소요

- 파이썬의 리스트는 크기가 동적으로 확대 또는 축소되며, 이러한 크기 조절은 사용자도 모르게 수행된다. 이러한 동적 크기 조절은 스택(리스트)의 모든 항목들을 새 리스트로 복사해야 하기 때문에 O(N) 시간이 소요

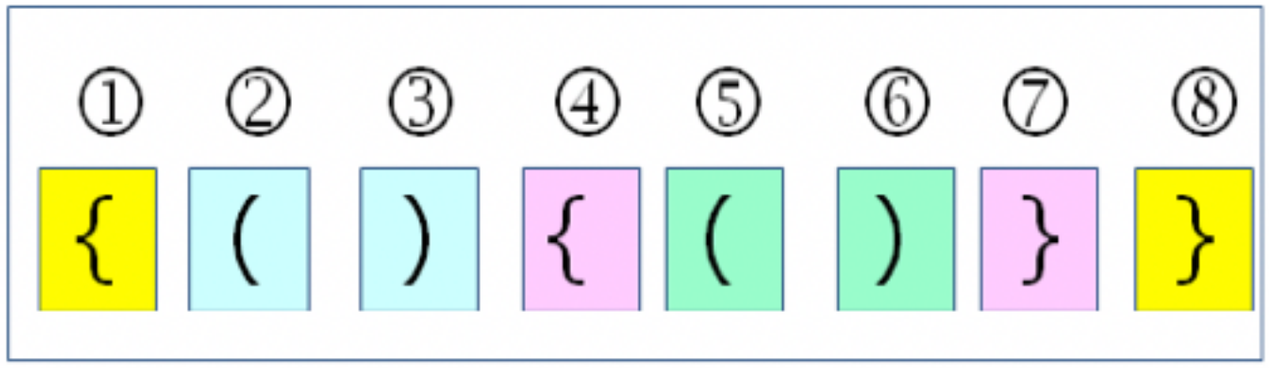
- 단순연결리스트로 구현한 스택의 push와 pop 연산은 각각 O(1) 시간

- 연결리스트의 맨 앞 부분에서 노드를 삽입하거나 삭제하기 때문

스택의 이용 (실습)

- 컴파일러의 괄호 짝 맞추기
- 회문(Palindrome) 검사하기 (**커플 프로그래밍-수업 과제로 제출**)
- 수식의 표기법 (**개인 프로그래밍-개인 과제로 제출**)

컴파일러 괄호 짝 맞추기

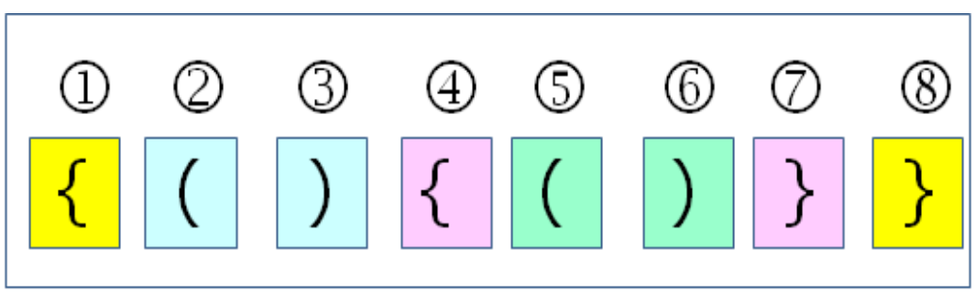


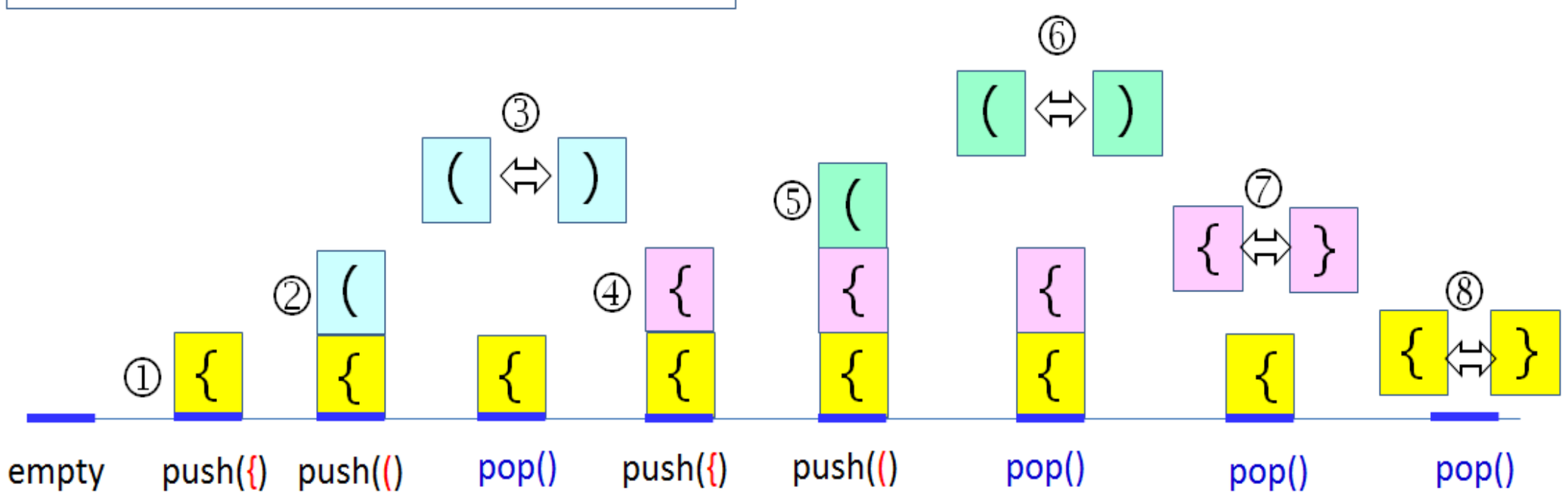
- 연괄호 모양과 닫는 괄호 모양이 일치하고 연것 만큼 닫는지 확인

기본 아이디어

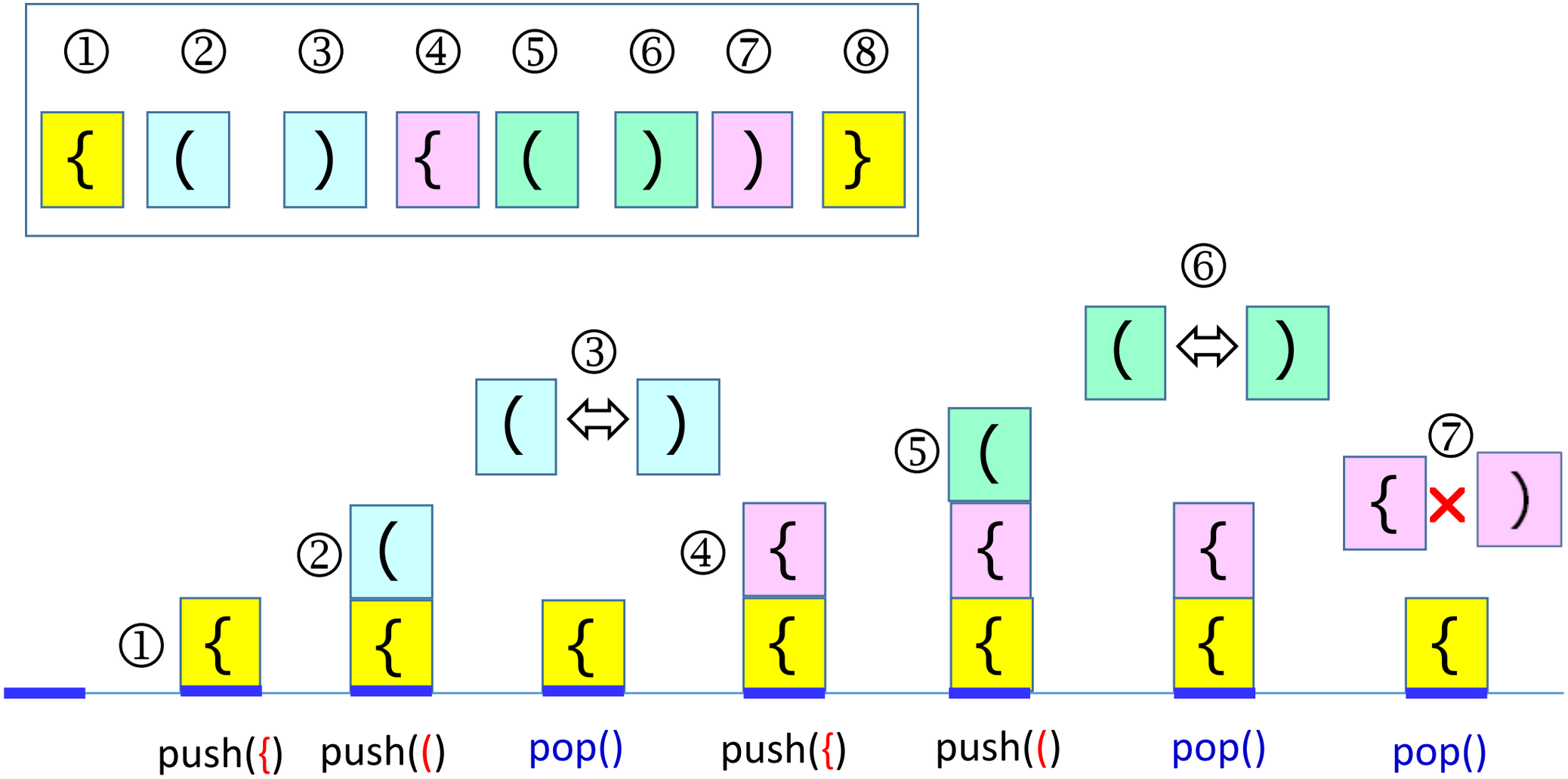
- 왼쪽 괄호 (“(”, “{”, “[”) 는 스택에 push, 오른쪽 괄호 (“)”, “}”, “]”)를 읽으면 pop 수행
  - pop된 왼쪽 괄호와 바로 읽었던 오른쪽 괄호가 다른 종류이면 에러 처리, 같은 종류이면 다음 괄호를 읽음
  - 모든 괄호를 읽은 뒤 에러가 없고 스택이 empty이면, 괄호들이 정상적으로 사용된 것
  - 만일 모든 괄호를 처리한 후 스택이 empty가 아니면 짝이 맞지 않는 괄호가 스택에 남은 것이므로 에러 처리

예 - 옳은 경우



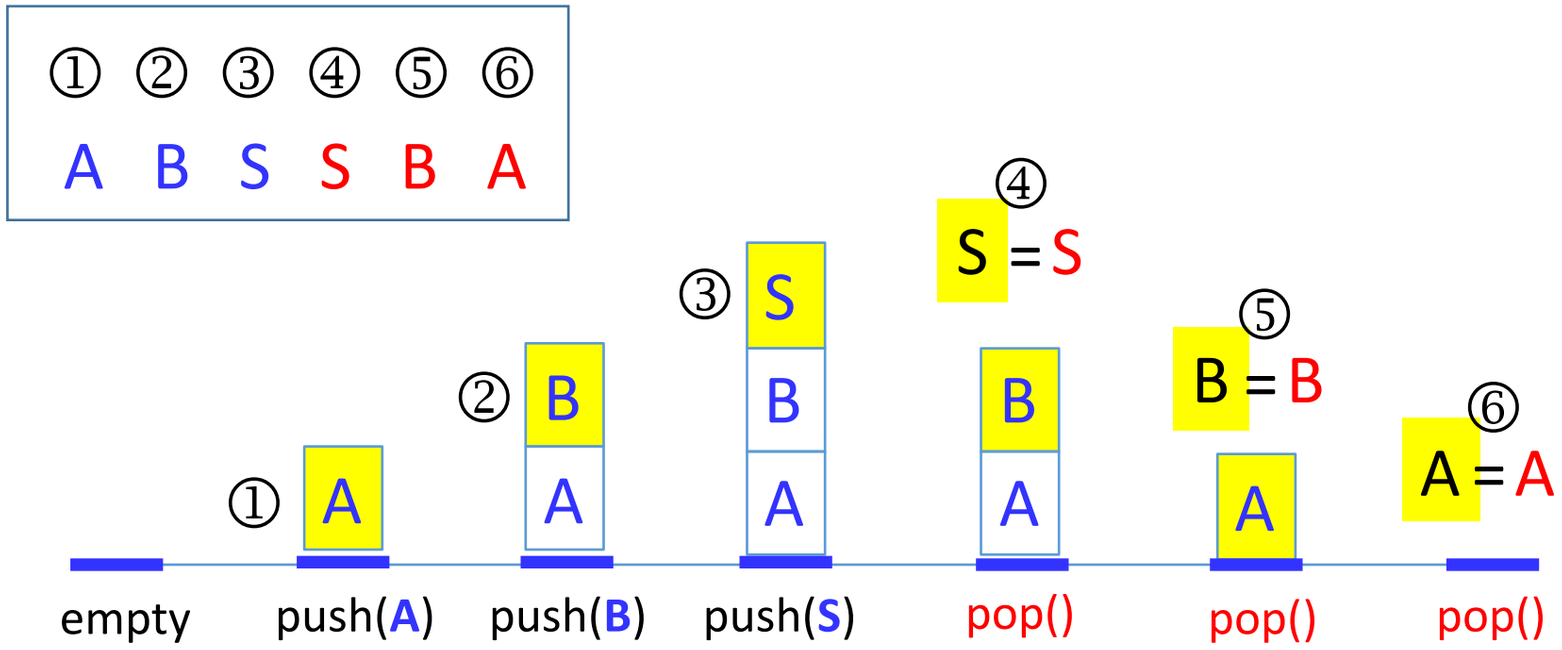


예- 틀린경우



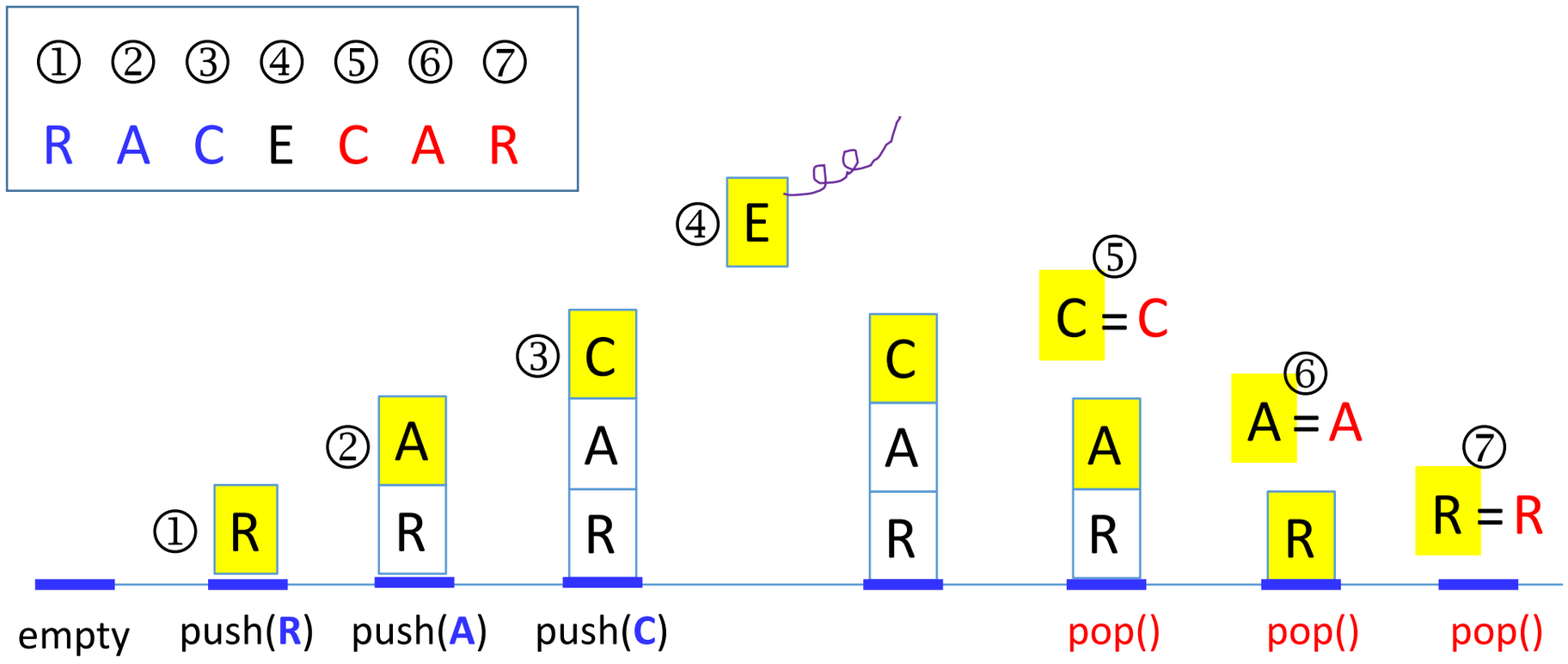
회문 검사하기

- 문자열이 짝수인 경우



회문 검사하기

- 문자열이 홀수인 경우



기타

- 후위표기법(Postfix Notation) 수식 계산하기
- 중위표기법(Infix Notation) 수식의 후위표기법 변환
- 미로 찾기
- 트리의 방문(4장)
- 그래프의 깊이우선탐색(8장)
- 프로그래밍에서 매우 중요한 함수/메소드 호출 및 재귀호출도 스택 자료구조를 바탕으로 구현

수식표기법

- 중위표기법(Infix Notation)
  - 프로그램을 작성할 때 수식에서 +, –, *, /와 같은 이항연산자는 2개의 피연산자들 사이에 위치
- 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

중위, 후위, 전위 표기법

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

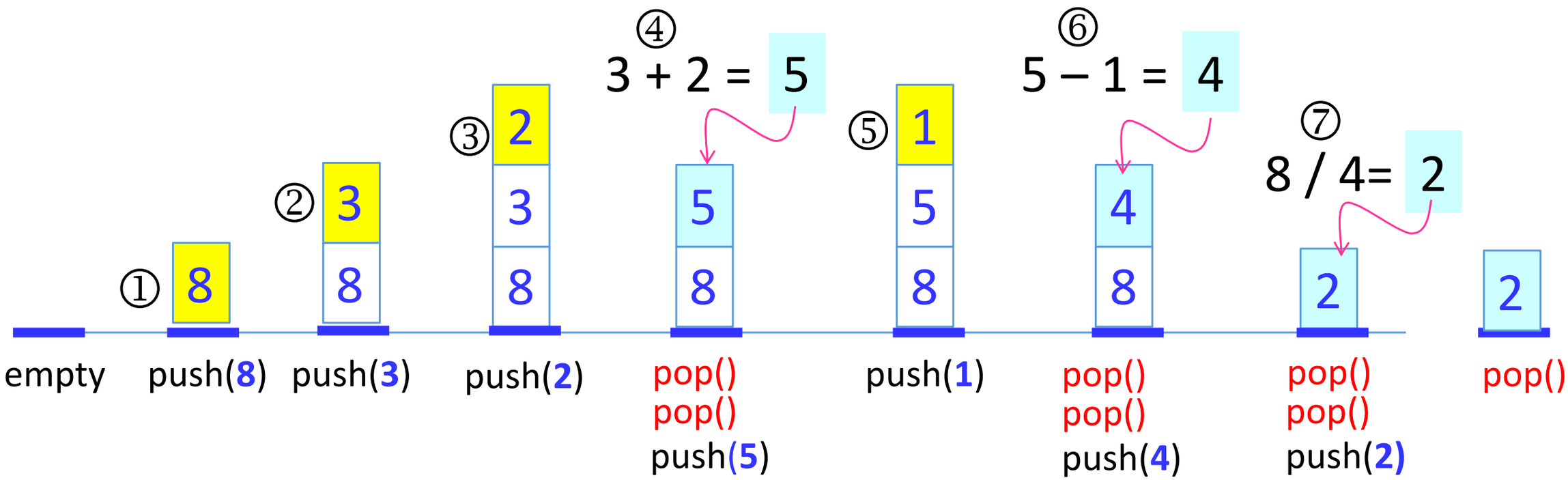
후위표기법 수식계산

중위표기법	후위표기법
A + B * C – D	A B C * + D –

- 피연산자는 스택에 push하고, 연산자는 2회 pop하여 계산한 후 push
- 입력을 좌에서 우로 문자를 한 개씩 읽는다. 읽은 문자를 C라고하면
  - C가 피연산자이면 스택에 push
  - C가 연산자(op)이면 pop을 2회 수행한다. 먼저 pop된 피연산자가 A이고, 나중에 pop된 피연산자가 B라면, (A op B)를 수행하여 그 결과 값을 push

후위표기법 수식 계산

①	②	③	④	⑤	⑥	⑦
8	3	2	+	1	−	/



중위 표기법을
후위 표기법으로 **변환**

중위표기법	후위표기법
A + B * C – D	A B C * + D –

중위 표기법을
후위 표기법으로 **변환**

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

- 왼쪽 괄호나 연산자는 스택에 push하고, 피연산자는 출력
- 입력을 좌에서 우로 문자를 1개씩 읽는다. 읽은 문자가

  1.피연산자이면, 읽은 문자를 출력

  2.왼쪽 괄호이면, push

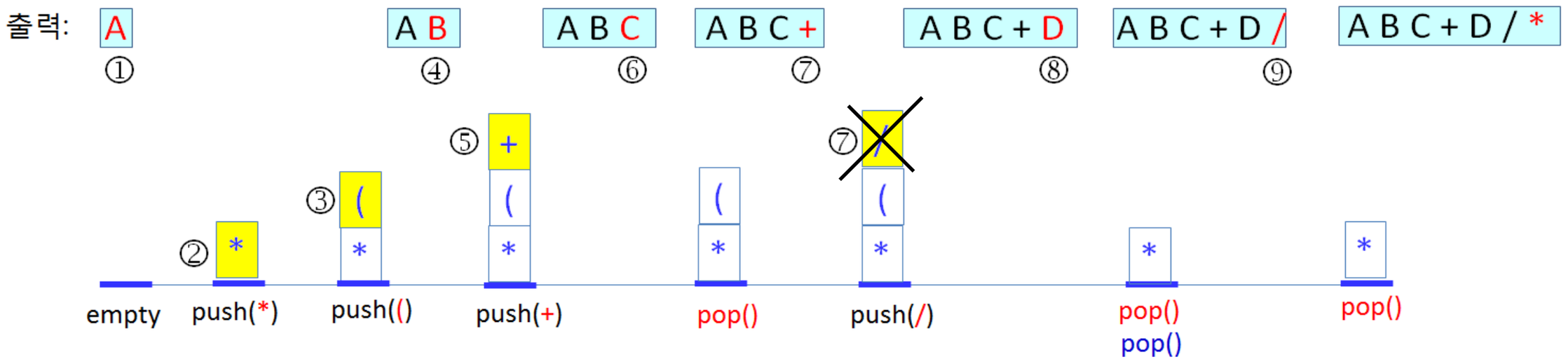
  3.오른쪽 괄호이면, 왼쪽 괄호가 나올 때까지 pop하여 출력. 단, 오른쪽이나 왼쪽 괄호는 출력하지 않음

  4.연산자이면, 입력으로 들어온 연산자의 우선순위보다 **낮은 연산자 (e.g. “+” < “*”)가 스택 top에 올 때까지** pop하여 출력하고 읽은 연산자를 push

- 입력을 모두 읽었으면 스택이 empty될 때까지 pop하여 출력

중위 표기법을
후위 표기법으로 변환

①	②	③	④	⑤	⑥	⑦	⑧	⑨
A	*	(	B	+	C	/	D	)

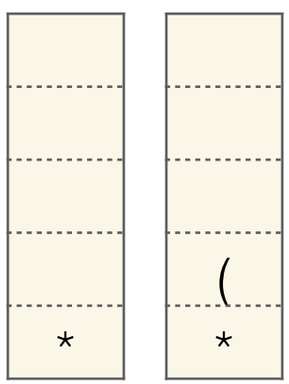


중위 표기법을
후위 표기법으로 변환

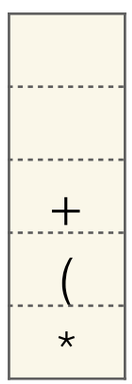
①	②	③	④	⑤	⑥	⑦	⑧	⑨
A	*	(	B	+	C	/	D	)

1 A

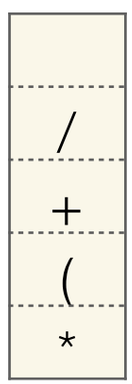
2 3 AB



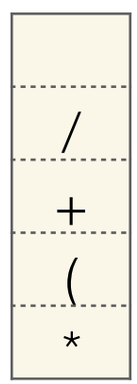
4 5 ABC



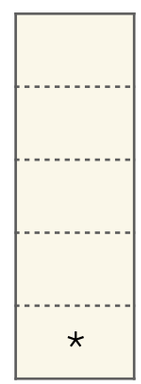
6 7 ABCD



8 ABCD/+



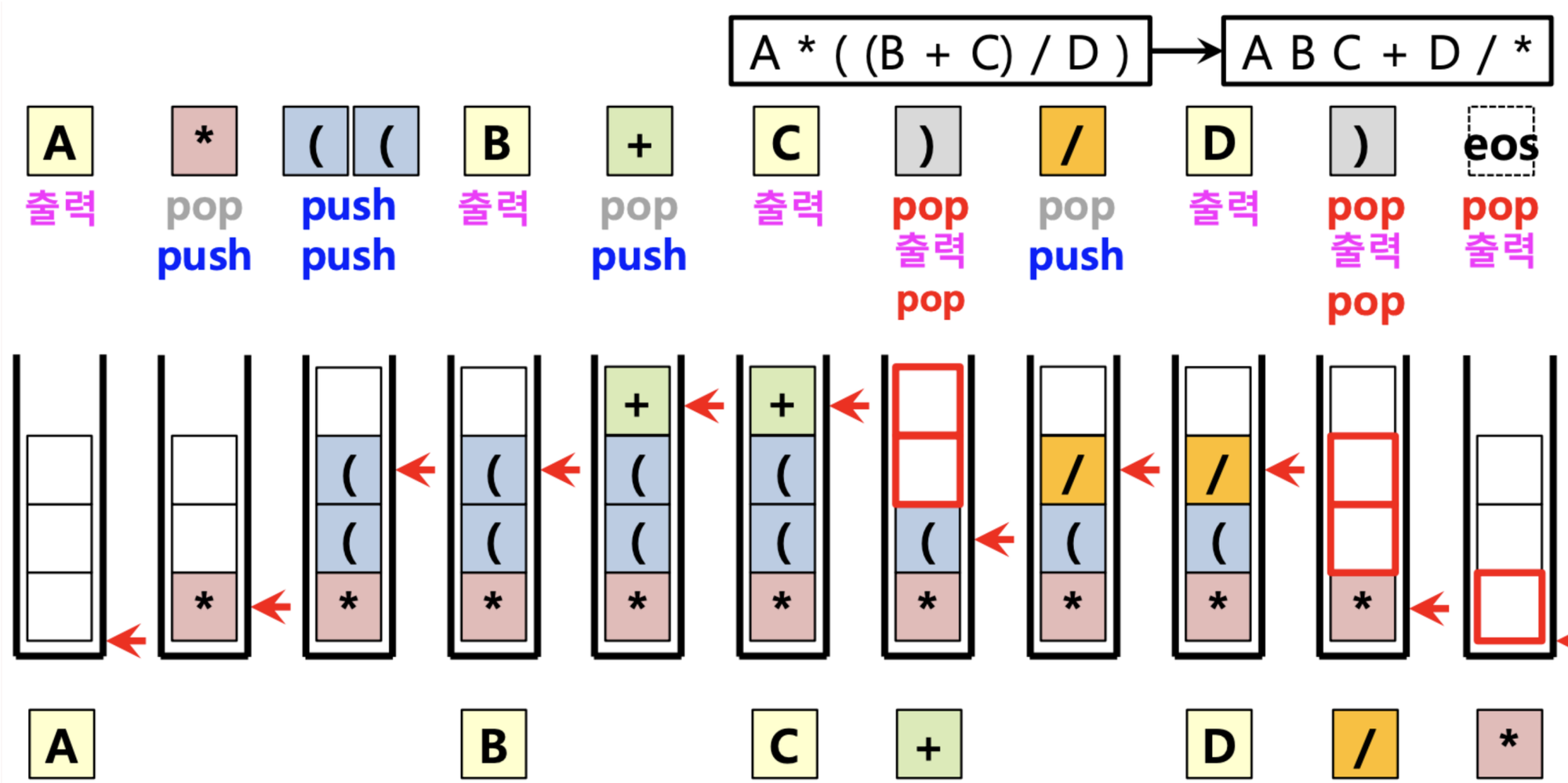
9 ABCD/+*



9

중위 표기법을
후위 표기법으로 변환

A * ((B+C) / D)



https://claude-u.tistory.com/74

예

- A-B+C*D/F
- A*(B+C)*(D-F)
- A+B-C*D*F

요약

- 스택은 한 쪽 끝에서만 item을 삭제하거나 새로운 item을 저장하는 후입선출(LIFO) 자료구조
- 스택은 컴파일러의 괄호 짝 맞추기, 회문 검사하기, 후위표기법수식 계산하기, 중위표기법 수식을 후위표기법으로 변환하기, 미로 찾기, 트리의 노드 방문, 그래프의 깊이우선탐색에 사용. 또한 프로그래밍에서 매우 중요한 메소드 호출 및 재귀호출도 스택 자료구조를 바탕으로 구현