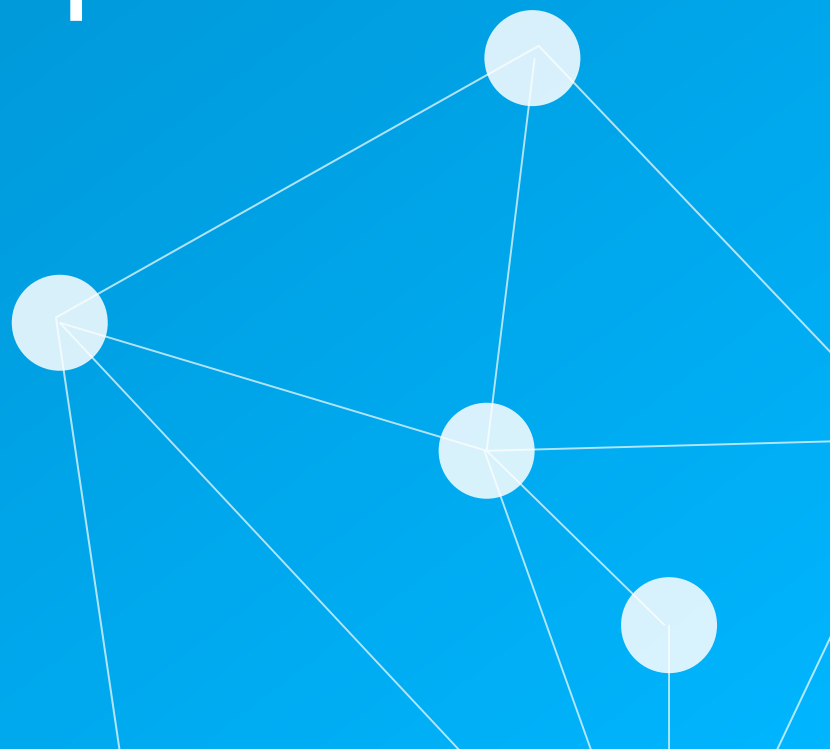

파이썬
자료구조

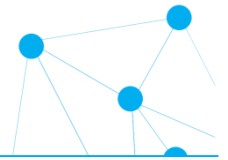
04

CHAPTER

스택

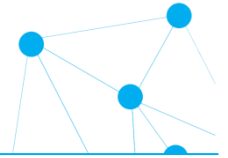


4장. 학습 목표



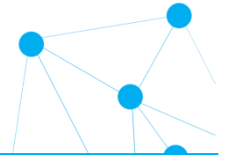
- 스택의 개념과 동작 원리를 이해한다.
- 배열 구조(파이썬 리스트)를 이용한 스택의 구현 방법을 이해한다.
- 스택의 함수 구현과 클래스 구현의 차이를 이해한다.
- 괄호 검사, 수식의 계산, 미로 탐색 등에 스택을 활용하여 문제를 해결할 수 있는 능력을 배양한다.

4.1 스택이란?



- 스택은 후입선출(Last-In First Out)의 자료구조이다.
 - 스택의 구조와 연산들
 - 스택 ADT
- 스택 용도

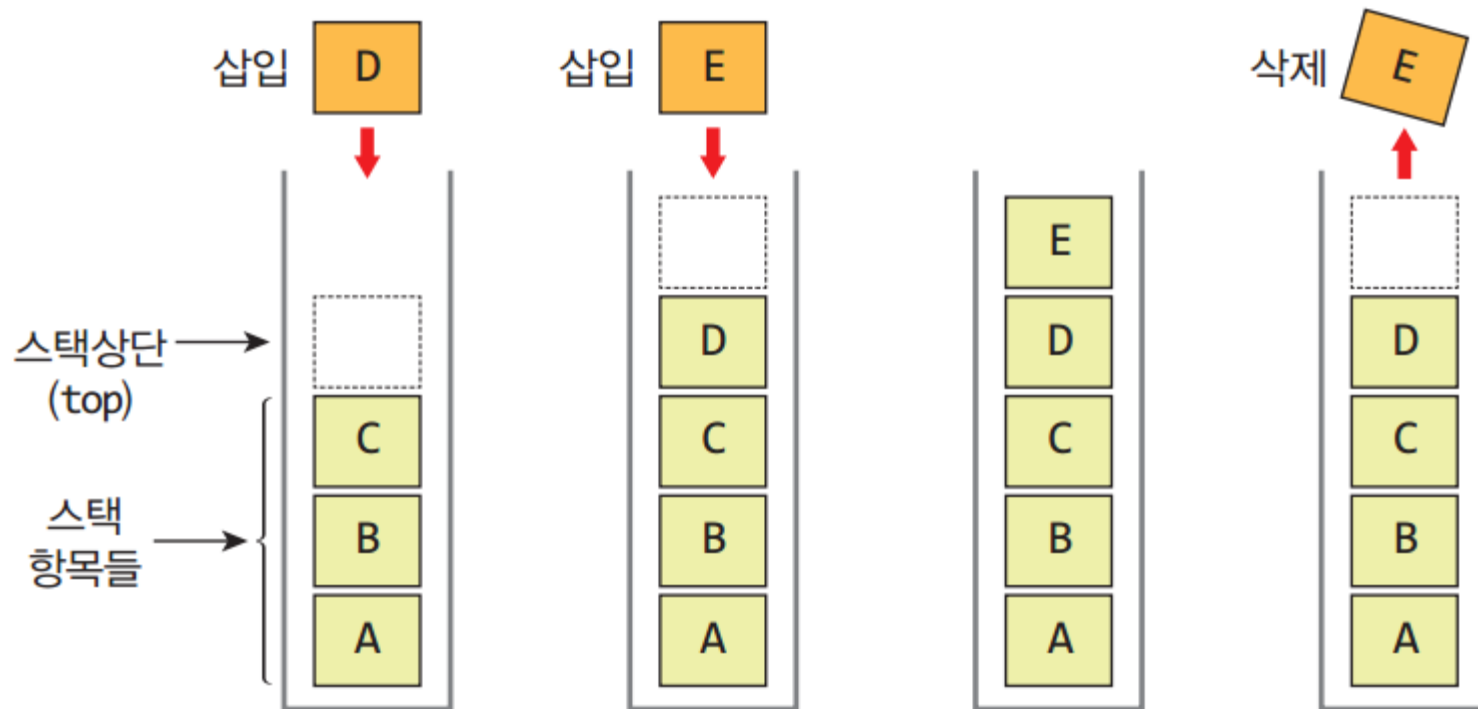
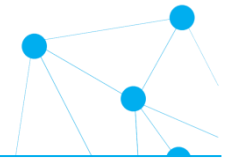
스택이란?



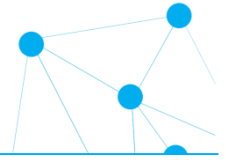
- 스택(stack): 쌓아놓은 더미
- 후입선출(LIFO: Last-In First-Out)
 - 가장 최근에 들어온 데이터가 가장 먼저 나감



스택의 구조와 일련의 연산들



스택 ADT



정의 4.1 Stack ADT

데이터: 후입선출(LIFO)의 접근 방법을 유지하는 항목들의 모임
연산

- `Stack()`: 비어 있는 새로운 스택을 만든다.
- `isEmpty()`: 스택이 비어있으면 `True`를 아니면 `False`를 반환한다.
- `push(e)`: 항목 `e`를 스택의 맨 위에 추가한다.
- `pop()`: 스택의 맨 위에 있는 항목을 꺼내 반환한다.
- `peek()`: 스택의 맨 위에 있는 항목을 삭제하지 않고 반환한다.
- `size()`: 스택내의 모든 항목들의 개수를 반환한다.
- `clear()`: 스택을 공백상태로 만든다.

맨위 접시종류?
접시 개수?
접시보관함 비우기

부가적

스택의 용도

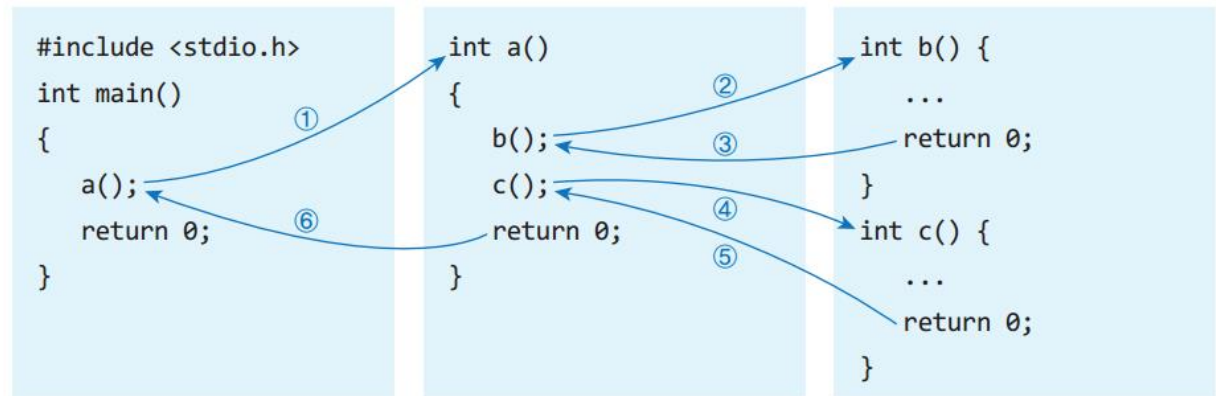


- 되돌리기

이전 페이지로 이동



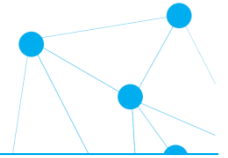
- 함수호출



- 괄호 검사

- 계산기: 후위 표기식 계산, 중위 표기식의 후위 표기식 변환
- 미로 탐색 등

4.2 스택의 구현

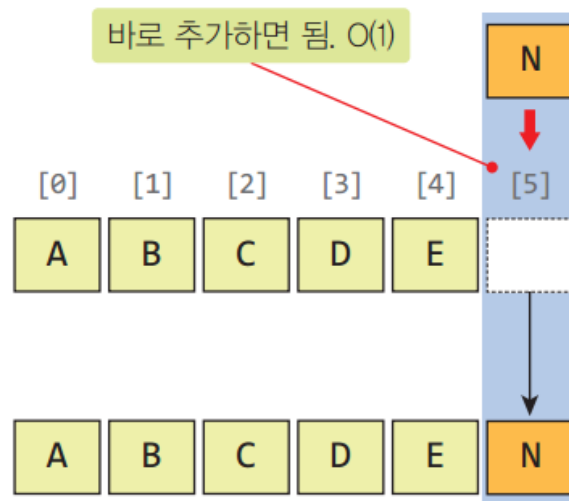


- 배열 구조를 이용한 스택
- 스택의 함수 구현
- 스택의 클래스 구현
- 스택의 활용
- 출력 방법 수정(파이썬 슬라이싱 기능)

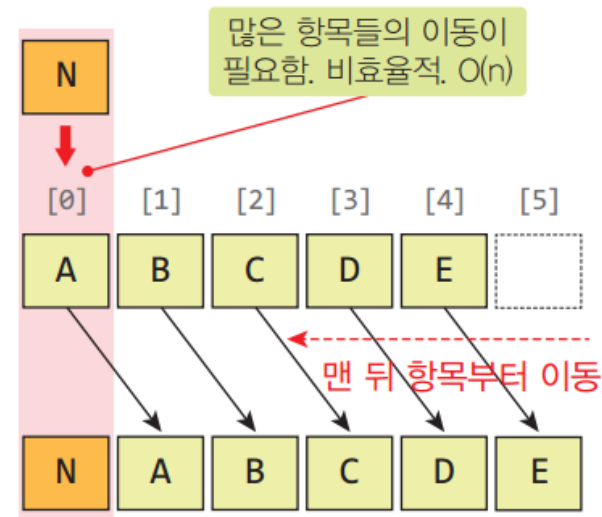
스택의 구현(배열 구조)



- 데이터
 - top: 스택 항목을 저장하는 파이썬 리스트
 - 항목의 개수는 `len(top)`으로 구할 수 있음
- 연산: `isEmpty()`, `push()`, `pop()`, `peek()`, `display()`
- 항목 삽입/삭제 위치: 리스트의 맨 뒤가 유리함. Why?



파이썬 리스트의 **후단**을 사용하는 경우



파이썬 리스트의 **전단**을 사용하는 경우

스택의 구현(함수 버전)



```
top = [ ]      # 스택의 데이터: 항목을 위한 공백 리스트
```

```
def isEmpty():  
    return len(top) == 0      # len(top) == 0 의 계산 결과가 True/False
```

```
def push(item):  
    top.append(item)      # 리스트의 맨 뒤에 item을 추가함
```

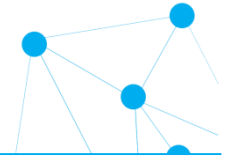
```
def pop():  
    if not isEmpty():      # 공백상태가 아니면  
        return top.pop(-1)  # 리스트의 맨 뒤에서 항목을 하나 꺼내고 반환
```

```
def peek():      # 맨 위의 항목을 삭제하지 않고 반환  
    if not isEmpty():      # 공백상태가 아니면  
        return top[-1]     # 맨 뒤 항목을 반환(삭제하지 않음)
```

```
def size(): return len(top)      # 스택의 크기
```

```
def clear():  
    global top      # top은 전역변수임을 지정함  
    top = []        # 스택의 초기화
```

스택의 활용(함수 버전)



```
for i in range(1,6):           # i = 1, 2, 3, 4, 5
    push(i)                    # push 연산 5회
print(' push 5회: ', top)      # 스택 내용 출력
print(' pop() --> ', pop())    # pop연산 및 반환 항목 출력
print(' pop() --> ', pop())    # pop연산 및 반환 항목 출력
print(' pop 2회: ', top)       # 스택 내용 출력 (str(top)도 동일함)
```

```
C:\ 선택 C:\WINDOWS\system32\cmd.exe
push 5회: [1, 2, 3, 4, 5]
pop() --> 5
pop() --> 4
pop 2회: [1, 2, 3]
```

가장 최근에 삽입된 항목

가장 먼저 삽입된 항목

```
C:\ 선택 C:\WINDOWS\system32\cmd.exe
pop 2회: [1, 2, 3]
push+2회: [1, 2, 3, '홍길동', '이순신']
pop() --> 이순신
pop 1회: [1, 2, 3, '홍길동']
```

정수

문자열

스택이 2개일때는 전역변수 top[] 1개이기때문에 사용 못함 → 클래스

스택의 구현(클래스 버전)



```
class Stack :  
    def __init__( self ):  
        self.top = []  
        # 생성자  
        # top이 이제 클래스의 멤버 변수가 됨  
  
    def isEmpty( self ): return len(self.top) == 0  
    def size( self ): return len(self.top)  
    def clear( self ): self.top = []  
        # 주의:이제 전역변수 선언이 필요없다.  
  
    def push( self, item ):  
        self.top.append(item)  
  
    def pop( self ):  
        if not self.isEmpty():  
            return self.top.pop(-1)  
  
    def peek( self ):  
        if not self.isEmpty():  
            return self.top[-1]
```

스택의 활용(클래스 버전)



```
odd = Stack()
even = Stack()
for i in range(10):
    if i%2 == 0 : even.push(i)
    else : odd.push(i)
print(' 스택 even push 5회: ', even)
print(' 스택 odd push 5회: ', odd)
print(' 스택 even      peek: ', even.peek())
print(' 스택 odd      peek: ', odd.peek())
for _ in range(2) : even.pop()
for _ in range(3) : odd.pop()
print(' 스택 even pop 2회: ', even)
print(' 스택 odd pop 3회: ', odd)
```

홀수 저장을 위한 스택
짝수 저장을 위한 스택
i = 0, 1, 2, ..., 9
짝수는 even 에 push
홀수는 even 에 push
even 스택 출력
odd 스택 출력
even 스택 peek()
odd 스택 peek()
even스택에서 두 번 pop()
odd스택에서 세 번 pop()
even 스택 출력
odd 스택 출력

even, odd객체 출력

```
C:\WINDOWS\system32\cmd.exe
스택 even push 5회: <__main__.Stack object at 0x0000024A32CD3320>
스택 odd push 5회: <__main__.Stack object at 0x0000024A32CD3208>
스택 even      peek: 8
스택 odd      peek: 9
스택 even pop 2회: <__main__.Stack object at 0x0000024A32CD3320>
스택 odd pop 3회: <__main__.Stack object at 0x0000024A32CD3208>
```

???
스택 내용이 아니라
스택 객체의 정보가
출력 됨

스택의 출력 방법 수정



- 방법 1

```
print(' 스택 even push 5회: ', even.top)    # even 스택 출력
print(' 스택 odd  push 5회: ', odd.top)     # odd 스택 출력
```

```
C:\WINDOWS\system32\cmd.exe
스택 even push 5회: [0, 2, 4, 6, 8]
스택 odd  push 5회: [1, 3, 5, 7, 9]
스택 even      peek: 8
스택 odd      peek: 9
스택 even pop 2회: [0, 2, 4]
스택 odd  pop 3회: [1, 3]
```

스택의 내용이 정상적으로 출력됨

- 방법 2: 연산자 중복 + 슬라이싱 기법

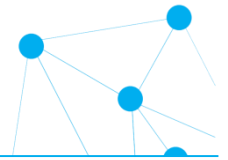
```
def __str__( self ):
    return str(self.top[::-1])
```

객체를 문자열 변환하는 연산자 중복(2장)
역순으로 출력. 최근의 항목을 먼저.

```
선택 C:\WINDOWS\system32\cmd.exe
스택 even push 5회: [8, 6, 4, 2, 0]
스택 odd  push 5회: [9, 7, 5, 3, 1]
스택 even      peek: 8
스택 odd      peek: 9
스택 even pop 2회: [4, 2, 0]
스택 odd  pop 3회: [3, 1]
```

내용이 역순으로 출력됨.
최근에 삽입된 항목이 먼저 출력됨.

4.3 스택의 응용: 괄호 검사



- 괄호 검사란?
- 괄호 검사 방법
- 구현 및 테스트
- 소스파일의 괄호검사

괄호 검사란?



- 괄호의 종류: 대중소 ('[', ']'), ('{', '}'), ('(', ')')

- 조건 1: 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 한다.
- 조건 2: 같은 타입의 괄호에서 왼쪽 괄호가 오른쪽 괄호보다 먼저 나와야 한다.
- 조건 3: 서로 다른 타입의 괄호 쌍이 서로를 교차하면 안 된다. **비교 오류**

- 괄호 사용 예

```
{ A[(i+1)]=0; }
```

→ 오류 없음

```
if ((i==0) && (j==0))
```

→ 오류: 조건 1 위반

```
while (it < 10)) { it--; }
```

→ 오류: 조건 2 위반

```
A[(i+1)]=0;
```

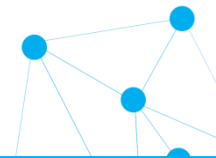
→ 오류: 조건 3 위반

괄호 검사 방법

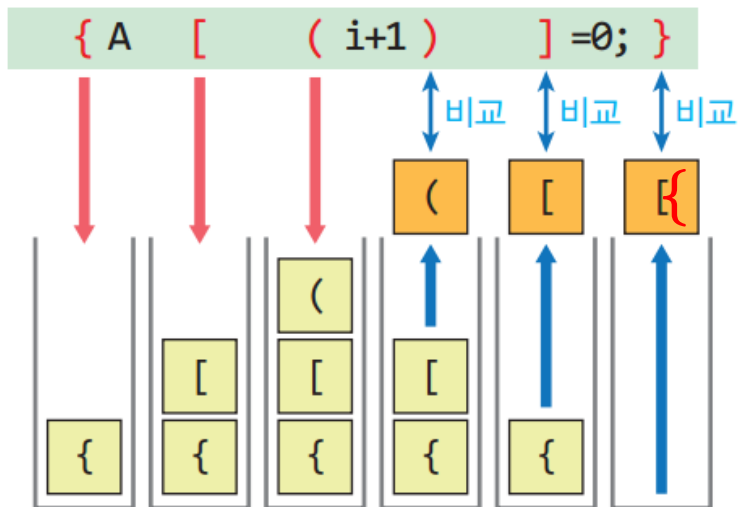


- 문자를 저장하는 스택을 준비한다. 처음에는 공백 상태가 되어야 한다.
- 입력 문자열의 문자를 하나씩 읽어 왼쪽 괄호를 만나면 스택에 삽입한다.
- 오른쪽 괄호를 만나면 `pop()` 연산으로 가장 최근에 삽입된 괄호를 꺼낸다. 이때 스택이 비었으면 조건 2에 위배된다.
- 꺼낸 괄호가 오른쪽 괄호와 짝이 맞지 않으면 조건 3에 위배된다.
- 끝까지 처리했는데 스택에 괄호가 남아 있으면 조건 1에 위배된다.

괄호 검사 예



1. 열린 괄호나오면 스택에 삽입
2. 닫힌 괄호나오면 스택에서 빼내어 비교



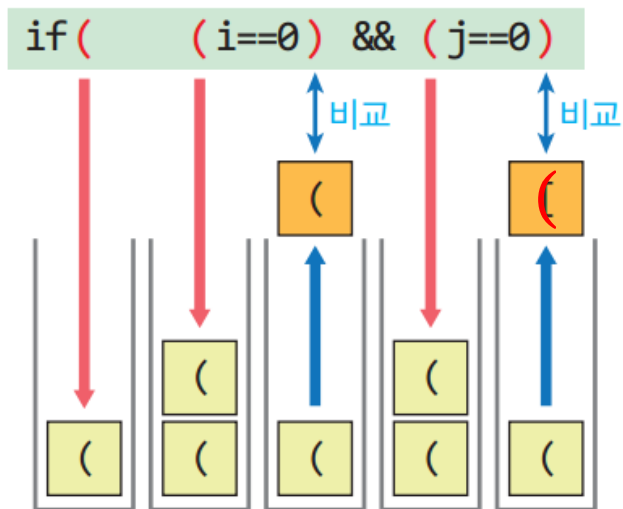
공백



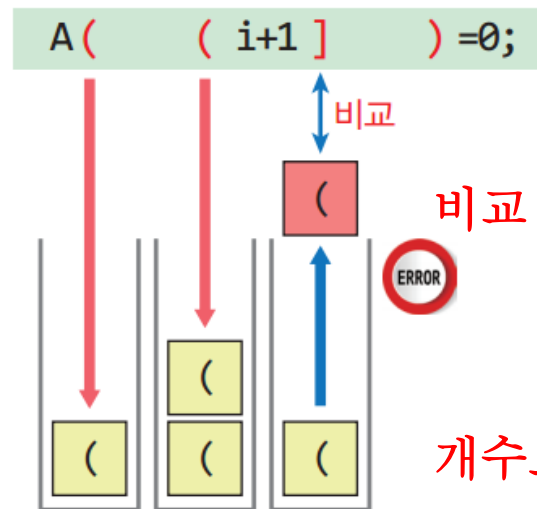
코드가
끝까지 검사되었고,
스택이 공백이니까...
오케이 굿!

서로 다른
괄호네... 에러!!!

앗. 스택에
괄호가 남았네...
에러!!!



ERROR



비교 오류

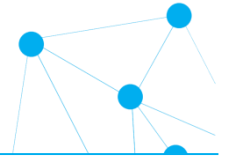
개수오류

괄호 검사 알고리즘




```
def checkBrackets(statement):
    stack = Stack()
    for ch in statement:
        # 문자열의 각 문자에 대해
        if ch in ('{', '[', '('):
            # in '[(('도 동일하게 동작함
            stack.push(ch)
        elif ch in ('}', ']', ')'):
            # in ')])'도 동일하게 동작함
            if stack.isEmpty() :
                return False
                # 조건 2 위반          스택공백 오류
            else :
                left = stack.pop()
                if (ch == "}" and left != "{") or \
                    (ch == "]" and left != "[") or \
                    (ch == ")" and left != "(") :
                    return False
                    # 조건 3 위반      비교 오류
    return stack.isEmpty()
    # False이면 조건 1 위반
```

테스트 프로그램



```
str = ( "{ A[(i+1)] = 0; }", "if( (i==0) && (j==0 )", "A[ (i+1) ] = 0;" )  
for s in str:  
    m = checkBrackets(s)  
    print(s, " ---> ", m)
```



```
C:\WINDOWS\system32\cmd.exe  
{ A[(i+1)] = 0; } ---> True  
if( (i==0) && (j==0 ) ---> False  
A[ (i+1) ] = 0; ---> False
```

소스파일의 괄호검사



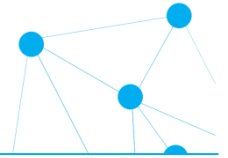
```
filename = "ArrayStack.h"           # 괄호를 검사할 소스 파일
infile = open(filename, "r")         # 파일 열기
lines = infile.readlines()           # 파일 전체를 라인별로 읽은 리스트
infile.close()                       # 파일 닫기

result = checkBracketsV2(lines)       # 검사 함수 호출 (약간 수정 필요)
print(filename, " ---> ", result)    # 결과 출력
```

```
def checkBracketsV2(lines):
    stack = Stack()
    for line in lines:                # 리스트의 모든 문자열에 대해
        for ch in statement:         # 문자열의 각 문자에 대해
            ...
```

```
C:\WINDOWS\system32\cmd.exe
ArrayStack.h ---> True
계속하려면 아무 키나 누르십시오 . . .
```

4.4 스택의 응용: 수식의 계산



- 계산기 프로그램은 어떻게 만들까?
- 스택을 이용한 후위표기 수식의 계산
- 스택을 이용한 중위표기 수식의 후위표기 변환

계산기 프로그램은 어떻게 만들까?

- 수식의 표기 방법 3가지

전위(prefix)	중위(infix)	후위(postfix)
연산자 피연산자1 피연산자2	피연산자1 연산자 피연산자2	피연산자1 피연산자2 연산자
+ A B	A + B	A B +
+ 5 * A B	5 + A * B	5 A B * +

- 수식 계산 과정

중위 $(A + B) * C$
후위 $A B + C *$

전 후위 표기법이
좋아요. 괄호도 없고,
우선순위 몰라도 되고, 특히 수식을
읽으면서 바로 계산할
수 있으니까요.

난 중위 표기법만
사용 할거야.
그게 제일 쉽거든.



중위→후위 변환

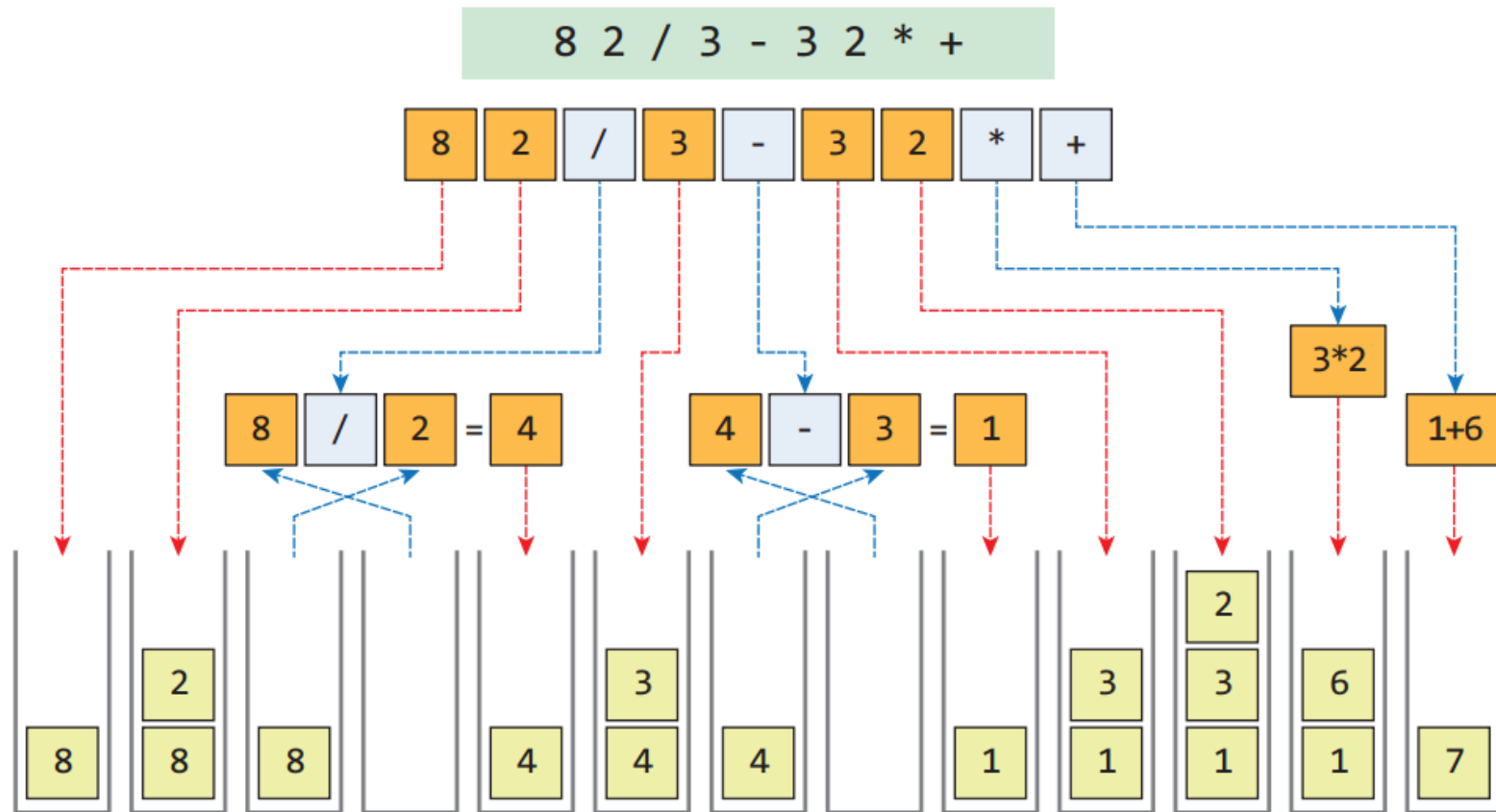
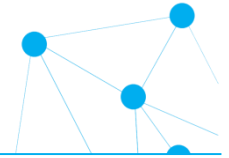
후위 계산

계산기 프로그램을 만드는 과정



괄호 X
우선순위 X
읽으면서 바로 계산

후위표기 수식의 계산 방법



후위 표기 수식 계산 알고리즘



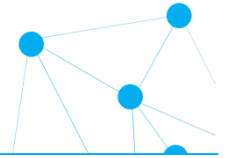
```
def evalPostfix( expr ):  
    s = Stack()  
    for token in expr :  
        if token in "+-*/" :  
            val2 = s.pop()  
            val1 = s.pop()  
            if (token == '+'): s.push(val1 + val2)  
            elif (token == '-'): s.push(val1 - val2)  
            elif (token == '*'): s.push(val1 * val2)  
            elif (token == '/'): s.push(val1 / val2)  
        else :  
            s.push( float(token) )  
  
    return s.pop()
```

스택 객체 생성
리스트의 모든 항목에 대해
항목이 연산자이면
피연산자2
피연산자1
각 연산 수행
결과는 스택에
다시 저장

항목이 피연산자이면
실수로 변경해서 스택에 저장

최종 결과를 반환

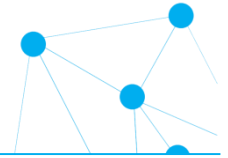
테스트 프로그램



```
expr1 = [ '8', '2', '/', '3', '-', '3', '2', '*', '+' ]  
expr2 = [ '1', '2', '/', '4', '*', '1', '4', '/', '*' ]  
print(expr1, ' --> ', evalPostfix(expr1))  
print(expr2, ' --> ', evalPostfix(expr2))
```

```
C:\WINDOWS\system32\cmd.exe  
['8', '2', '/', '3', '-', '3', '2', '*', '+'] --> 7.0  
['1', '2', '/', '4', '*', '1', '4', '/', '*'] --> 0.5
```

중위 표기 수식의 후위 표기 변환



- 중위표기와 후위표기

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

후위표기: 숫자(피연산자) 스택에 저장
중위표기: 연산자 스택에 저장

- 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 괄호위에 쌓여있는 모든 연산자를 출력

중위 → 후위 변환: $A+B*C$



단계	중위표기 수식	스택(우측이 상단)	후위표기 수식
0	A + B * C	[]	
1	A + B * C	[]	A
2	A + B * C	['+']	A
3	A + B * C	['+']	A B
4	A + B * C	['+', '*']	A B
5	A + B * C	['+']	A B C
6	A + B * C	[]	A B C +

연산자 우선순위 *부터 꺼내기

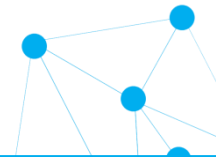
중위 → 후위 변환: $A*B+C$



단계	중위표기 수식	스택(우측이 상단)	후위표기 수식
0	A * B + C	[]	
1	A * B + C	[]	A
2	A * B + C	['*']	A
3	A * B + C	['*']	A B
4	A * B + C	['+']	A B *
5	A * B + C	['+']	A B * C
6	A * B + C	[]	A B * C +

연산자 우선순위 *부터 꺼내기

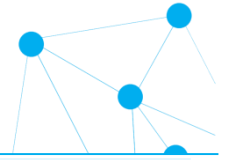
중위 → 후위 변환: $(A+B)*C$



단계	중위표기 수식	스택	후위표기 수식
0	(A + B) * C	[]	
1	(A + B) * C	['(']	
2	(A + B) * C	['(']	
3	(A + B) * C	['(', '+']	A
4	(A + B) * C	['(', '+']	A B
5	(A + B) * C	[]	A B +
6	(A + B) * C	['*']	A B +
7	(A + B) * C	['*']	A B + C
8	(A + B) * C	[]	A B + C *

왼쪽 괄호는 무조건 스택 저장
우선순위 가장 낮다
오른쪽 괄호는 왼쪽 괄호 나올때
까지 꺼내기

중위 → 후위 변환 알고리즘



```
def precedence (op):
```

```
    if op=='(' or op==')' : return 0
    elif op=='+' or op=='-' : return 1
    elif op=='*' or op=='/' : return 2
    else : return -1
```

```
def Infix2Postfix( expr ):
```

```
    s = Stack()
```

```
    output = []
```

```
    for term in expr :
```

```
        if term in '(' :
```

```
            s.push('(')
```

```
        elif term in ')' :
```

```
            while not s.isEmpty() :
```

```
                op = s.pop()
```

```
                if op=='(' : break;
```

```
            else :
```

```
                output.append(op)
```

```
        elif term in "+-*/" :
```

```
            while not s.isEmpty() :
```

```
                op = s.peek()
```

```
                if( precedence(term) <= precedence(op)):
```

```
                    output.append(op)
```

```
                    s.pop()
```

```
            else: break
```

```
            s.push(term)
```

```
        else :
```

```
            output.append(term)
```

```
    while not s.isEmpty() :
```

```
        output.append(s.pop())
```

```
    return output
```

```
# expr: 입력 리스트(중위 표기식)
```

```
# output: 출력 리스트(후위 표기식)
```

```
# 왼쪽 괄호이면
```

```
# 스택에 삽입
```

```
# 오른쪽 괄호이면
```

```
# 왼쪽 괄호가 나올 때 까지
```

```
# 스택에서 연산자를 꺼내 출력
```

```
# 연산자이면
```

```
# 우선순위가 높거나 같은 연산자를
```

```
# 스택에서 모두 꺼내 출력
```

```
# 마지막으로 현재 연산자 삽입
```

```
# 피연산자이면
```

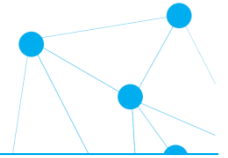
```
# 바로 출력
```

```
# 처리가 끝났으면 스택에 남은 항목을
```

```
# 모두 출력
```

```
# 결과(후위표기식 리스트)를 반환
```

테스트 프로그램



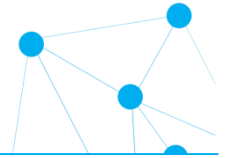
```
infix1 = [ '8', '/', '2', '-', '3', '+', '(', '3', '*', '2', ')']
infix2 = [ '1', '/', '2', '*', '4', '*', '(', '1', '/', '4', ')']
postfix1 = Infix2Postfix(infix1)
postfix2 = Infix2Postfix(infix2)
result1 = evalPostfix(postfix1)
result2 = evalPostfix(postfix2)
print(' 중위표기: ', infix1)
print(' 후위표기: ', postfix1)
print(' 계산결과: ', result1, end='\n\n')
print(' 중위표기: ', infix2)
print(' 후위표기: ', postfix2)
print(' 계산결과: ', result2)
```

C:\WINDOWS\system32\cmd.exe

```
중위 표기:  ['8', '/', '2', '-', '3', '+', '(', '3', '*', '2', ')']
후위 표기:  ['8', '2', '/', '3', '-', '3', '2', '*', '+']
계산결과:  7.0
```

```
중위 표기:  ['1', '/', '2', '*', '4', '*', '(', '1', '/', '4', ')']
후위 표기:  ['1', '2', '/', '4', '*', '1', '4', '/', '*']
계산결과:  0.5
```


4.5 스택의 응용: 미로 탐색



- 미로 탐색이란?
- 깊이우선탐색
 - 깊이우선탐색 알고리즘
 - 스택을 이용한 깊이우선탐색의 구현

미로 탐색이란?



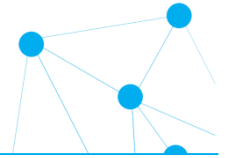
	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

```
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' ] ]
```

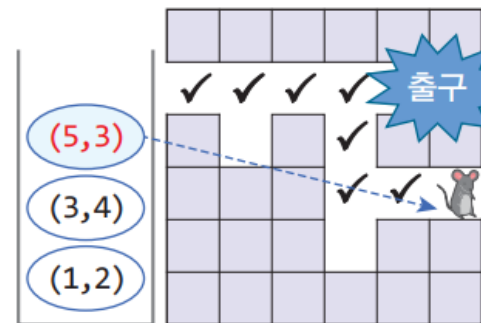
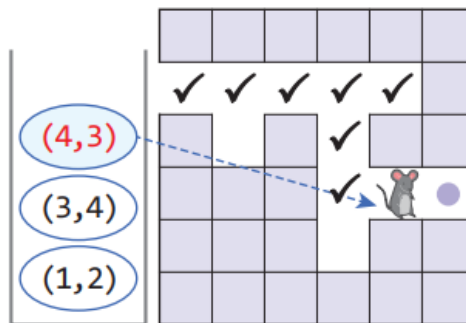
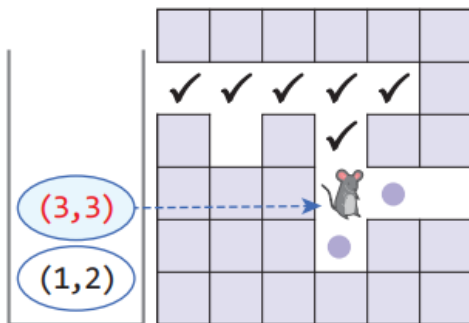
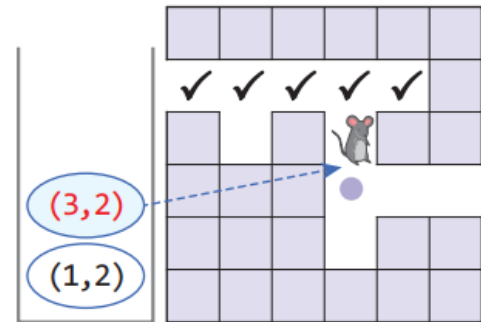
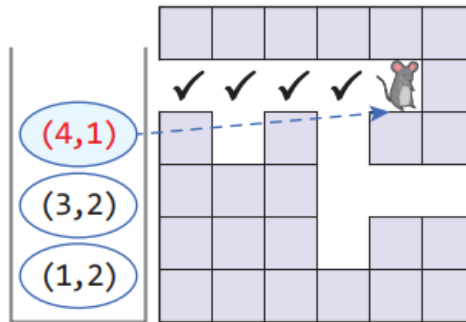
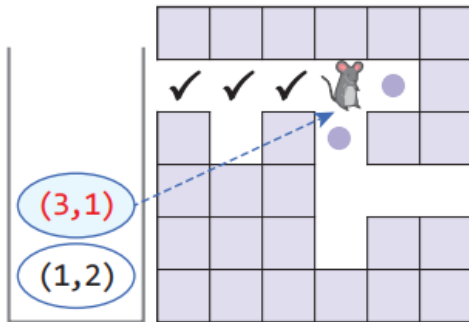
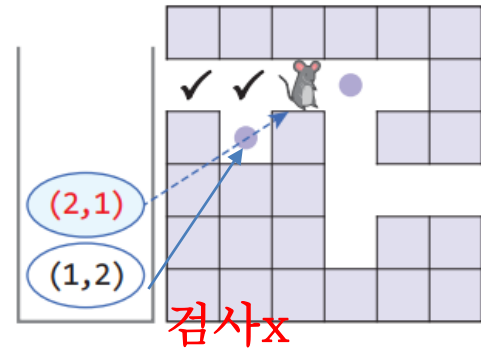
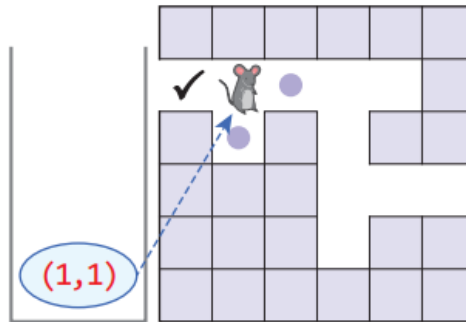
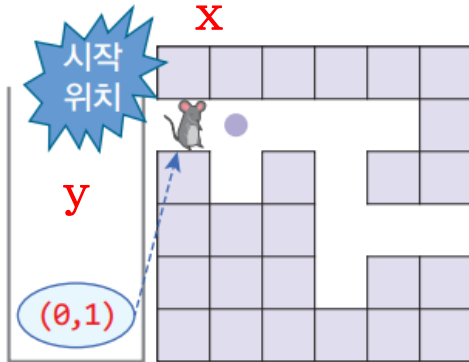
MAZE_SIZE = 6

2차원배열, 리스트에
리스트

깊이우선탐색: 스택 사용



(x,y)



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) # 현재 스택 내용 출력
    return False            # 탐색 실패. False 반환
```

테스트 프로그램



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

C:\WINDOWS\system32\cmd.exe

DFS:

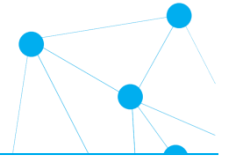
최종 탐색 순서

가장 최근에 삽입된 항목이 먼저 출력되도록 함

```
(0, 1) -> 현재 스택: [(1, 1)]
(1, 1) -> 현재 스택: [(2, 1), (1, 2)]
(2, 1) -> 현재 스택: [(3, 1), (1, 2)]
(3, 1) -> 현재 스택: [(4, 1), (3, 2), (1, 2)]
(4, 1) -> 현재 스택: [(3, 2), (1, 2)]
(3, 2) -> 현재 스택: [(3, 3), (1, 2)]
(3, 3) -> 현재 스택: [(4, 3), (3, 4), (1, 2)]
(4, 3) -> 현재 스택: [(5, 3), (3, 4), (1, 2)]
(5, 3) -> --> 미로탐색 성공
```

4장 연습문제, 실습문제





감사합니다!