

# 자료구조



## 스택의 활용



한국기술교육대학교  
온라인평생교육원

## 학습내용

- 후위 표기 수식의 계산
- 중위 표기 수식의 후위 표기 변환
- 미로 탐색

## 학습목표

- 수식의 표기 방법들을 설명할 수 있다.
- 수식의 계산 과정을 구현할 수 있다.
- 미로 탐색과 깊이 우선 탐색을 설명할 수 있다.

# 후위 표기 수식의 계산



## 스택의 활용

### 후위 표기 수식의 계산

#### 1 수식 표기 방법

##### 계산기 프로그램

- 수식을 적은 문자열을 입력 받아 이를 계산하고 결과를 출력하는 프로그램

예  $2 + 3 - 4 \rightarrow 1$  (출력)  
 $2 + 3 * 4 \rightarrow 14$  (출력)  
 $(2 + 3) * 4 \rightarrow 20$  (출력)

스택 사용해 구현!

▶ 수식을 표기하는 방법은 위의 방법 외에도 다양

## 스택의 활용

### 후위 표기 수식의 계산

#### 1 수식 표기 방법

##### ● 수식의 표기 방법

▶ 연산자와 피연산자 위치를 기준으로 3가지로 분류

우리가 일반적으로 사용하는 방법

전위(prefix) 표기

중위(infix) 표기

후위(postfix) 표기

| 연산자 피연산자1 피연산자2 | 피연산자1 연산자 피연산자2 | 피연산자1 피연산자2 연산자 |
|-----------------|-----------------|-----------------|
| $+ AB$          | $A + B$         | $AB +$          |
| $+ 5 * AB$      | $5 + A * B$     | $5 AB * +$      |

# 후위 표기 수식의 계산



## 스택의 활용

### 후위 표기 수식의 계산

#### 1 수식 표기 방법

- 사람들이 선호하는 표기 방법은?

▶ 중위 표기

연산자가 피연산자 사이에 들어감

괄호를 사용 가능

예  $5 + 3 * (4 - 7)$

연산자 우선순위를 반영해 계산

▶ 컴퓨터는 중위 표기를 좋아하지 않음

## 스택의 활용

### 후위 표기 수식의 계산

#### 1 수식 표기 방법

- 컴퓨터의 수식 계산 방법

▶ 컴퓨터에서 수식을 계산하는 과정

1. 중위 표기 수식을 입력 받음(사용자 입력)

2. 중위 표기 수식을 후위 표기로 변환

3. 후위 표기 수식을 계산

▶ 과정 2, 3에서 모두 스택이 사용



# 후위 표기 수식의 계산

## 스택의 활용

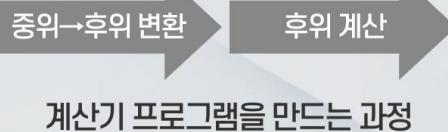
### 후위 표기 수식의 계산

#### 1 수식 표기 방법

##### ◉ 컴퓨터의 수식 계산 방법

▶ 계산기 프로그램의 전체 구조

난 중위 표기법만  
사용 할거야.  
그게 제일 쉽거든.



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



## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### 후위 표기 수식 계산

- **후위 표기 수식의 예  $82/3-$**
- 중위 표기:  $8/2-3$  또는  $(8/2)-3$
- 계산 결과: 1
- 후위 표기 수식에는 **괄호 X**  
(이미 수식에 반영)

# 후위 표기 수식의 계산



## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 아이디어

후위 수식을 왼쪽에서 오른쪽으로 스캔

피연산자를 만나면 스택에 저장

연산자를 만나면

- ① 필요한 수 만큼의 피연산자를 스택에서 꺼냄
- ② 꺼낸 피연산자와 연산자로 연산을 실행
- ③ 연산의 결과를 다시 스택에 저장

## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $8 \ 2 \ / \ 3 \ -$



피연산자 → 스택에 삽입



# 후위 표기 수식의 계산

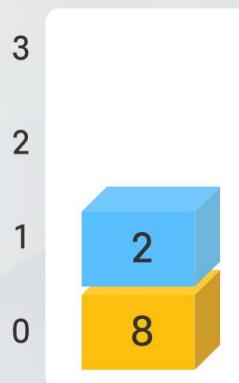
## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예

 $8 \ 2 \ / \ 3 \ -$ 

피연산자 → 스택에 삽입

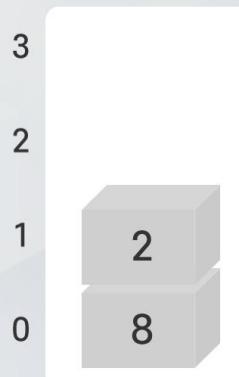
## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예

 $8 \ 2 \ / \ 3 \ -$ 

피연산자 → 삽입



# 후위 표기 수식의 계산

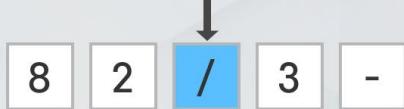
## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $8 \ 2 \ / \ 3 \ -$



연산결과를 스택에 다시 저장

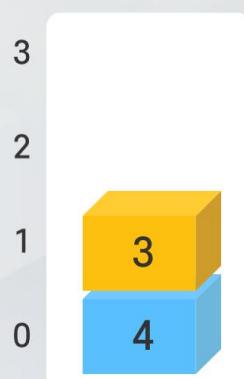
## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $8 \ 2 \ / \ 3 \ -$



연산결과를 스택에 다시 저장



# 후위 표기 수식의 계산

## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $82/3-$



연산결과를 스택에 다시 저장

## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $82/3-$



연산 종료: 전체 연산 결과=1



# 후위 표기 수식의 계산

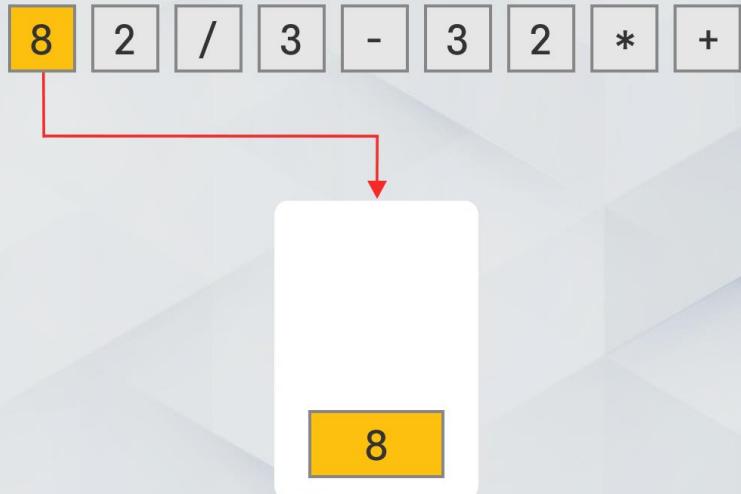
## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $8 \ 2 \ / \ 3 \ - \ 3 \ 2 \ * \ +$



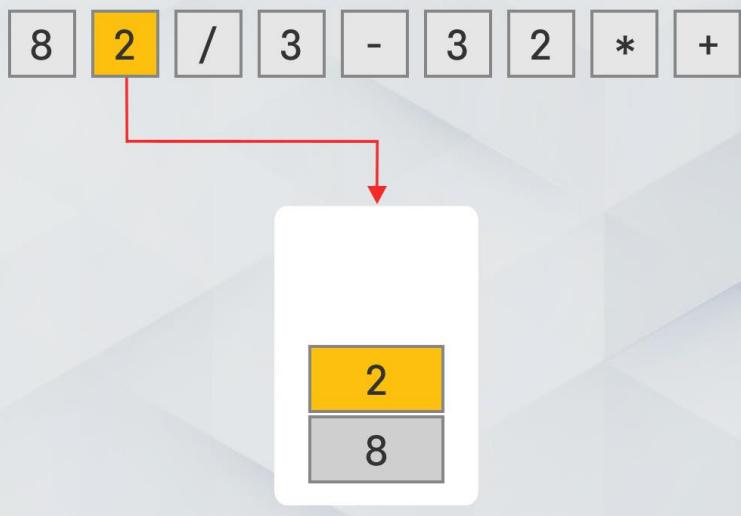
## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $8 \ 2 \ / \ 3 \ - \ 3 \ 2 \ * \ +$



피연산자 → 스택에 삽입



# 후위 표기 수식의 계산

## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $82/3-32*+$

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 8 | 2 | / | 3 | - | 3 | 2 | * | + |
|---|---|---|---|---|---|---|---|---|

$$\boxed{8} \quad \boxed{/} \quad \boxed{2} = \boxed{4}$$

연산자 → 스택에서 피연산자 2개를 꺼내 계산



피연산자 → 스택에 삽입

## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $82/3-32*+$

|   |   |   |          |   |   |   |   |   |
|---|---|---|----------|---|---|---|---|---|
| 8 | 2 | / | <b>3</b> | - | 3 | 2 | * | + |
|---|---|---|----------|---|---|---|---|---|



피연산자 → 스택에 삽입



# 후위 표기 수식의 계산

## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $82/3-32*+$

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 8 | 2 | / | 3 | - | 3 | 2 | * | + |
|---|---|---|---|---|---|---|---|---|

$$4 \quad - \quad 3 = 1$$

연산자 → 스택에서 피연산자 2개를 꺼내 계산

1

계산 결과를 스택에 삽입

## 스택의 활용

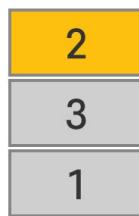
### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $82/3-32*+$

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 8 | 2 | / | 3 | - | 3 | 2 | * | + |
|---|---|---|---|---|---|---|---|---|



피연산자 → 스택에 삽입



# 후위 표기 수식의 계산

## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $82/3-32*+$

8 2 / 3 - 3 2 \* +

$$3 * 2 = 6$$

연산자 → 스택에서 피연산자 2개를 꺼내 계산



계산 결과를 스택에 삽입

## 스택의 활용

### 후위 표기 수식의 계산

#### 2 후위 표기 수식의 계산 방법

##### ◎ 후위 표기식 계산 예

예  $82/3-32*+$

8 2 / 3 - 3 2 \* +

$$1 + 6 = 7$$

연산자 → 스택에서 피연산자 2개를 꺼내 계산



계산 결과를 스택에 삽입



# 후위 표기 수식의 계산

스택의 활용

후위 표기 수식의 계산

## 2 후위 표기 수식의 계산 방법

```
from stackClass import Stack

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()
```

스택의 활용

후위 표기 수식의 계산

## 2 후위 표기 수식의 계산 방법

### ◎ 테스트 프로그램

```
expr1 = [ '8', '2', '/', '3', '-', '3', '2', '*', '+' ]
expr2 = [ '1', '2', '/', '4', '*', '1', '4', '/', '*' ]
str3 = ' 8 2 / 3 - 3 2 * +'
expr3 = str3.split()

print(expr1, ' -> ', evalPostfix(expr1))
print(expr2, ' -> ', evalPostfix(expr2))
print(expr3, ' -> ', evalPostfix(expr3))
```

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

# 후위 표기 수식의 계산



파일(F) 편집(E) 보기(V) 프로젝트(P) 디버그(O) 테스트(S) 분석(R) 도구(T) 확장(X) 찾(W) 도움말(H) 검색(Ctrl + Q) 고체 파이썬

구성 앱

EvalPostfixExpr.py - Python 3.6 (64-bit)

```

1  from stackClass import Stack
2
3  def evalPostfix( expr ):
4      s = Stack()
5      for token in expr :
6          if token in "+-*/" :
7              val2 = s.pop()
8              val1 = s.pop()
9              if (token == '+'): s.push(val1 + val2)
10             elif (token == '-'): s.push(val1 - val2)
11             elif (token == '*'): s.push(val1 * val2)
12             elif (token == '/'): s.push(val1 / val2)
13         else :
14             s.push( float(token) )
15
16     return s.pop()
17
18 #-----
19 if __name__ == "__main__":
20     print('스택의 응용2: 후위표기식 계산\n')
21
22     expr1 = [ '8', '2', '/', '3', '-', '3', '2', '*', '+' ]

```

출처

출처 보기 선택(O)

오류 목록 | 경고 표 | 출처

저장되었습니다.

## 실습 단계

### 후위 표기식 계산 알고리즘 테스트

#### 테스트 코드

#### 소스 코드 실행 결과

#### 테스트 코드 수정

#### 소스 코드 실행 결과

# 중위 표기 수식의 후위 표기 변환



## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

| 공통점   | 차이점  |
|---|--|
| <p>피연산자의 순서는 동일<br/>(<math>2 + 3 * 4 \rightarrow 2 3 4 * +</math>)</p> <p>→ 피연산자는 순서대로 출력</p> | <p>연산자의 우선순위를 반영해<br/>연산자들의 순서가 다름<br/>(<math>2 + 3 * 4 \rightarrow 2 3 4 * +</math>)</p> <p>→ 연산자들은 스택에 저장했다가<br/>우선순위를 고려해서 출력</p> |

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

##### ● 중위 표기에서 후위 표기 변환을 위한 아이디어

▶ 변환 아이디어

중위 표기식에서 피연산자를 만나면 그대로 출력

연산자를 만나면 잠시 스택에 저장

- 연산자의 우선 순위를 처리
- 스택에는 우선 순위가 높은 연산자가 위에 있어야 함
- 우선 순위가 낮은 연산자는 높은 연산자들을 모두 꺼낸 후 스택에 삽입



# 중위 표기 수식의 후위 표기 변환

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

- 중위 표기에서 후위 표기 변환을 위한 아이디어

왼쪽괄호, 오른쪽괄호에 대한 처리 필요

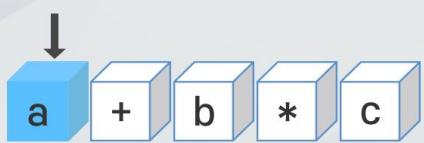
- 오른쪽괄호가 나오면 스택에서 왼쪽괄호 위에 쌓여 있는 모든 연산자 출력

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

- Case 1:  $a + b * c$



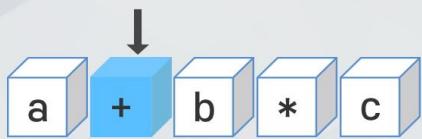


# 중위 표기 수식의 후위 표기 변환

스택의 활용

중위 표기 수식의 후위 표기 변환

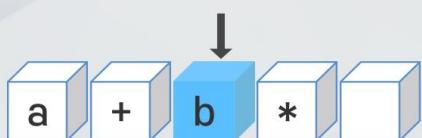
## 1 중위와 후위 표기 수식의 비교

◎ Case 1:  $a + b * c$ 

스택의 활용

중위 표기 수식의 후위 표기 변환

## 1 중위와 후위 표기 수식의 비교

◎ Case 1:  $a + b * c$ 



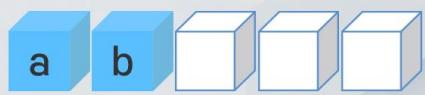
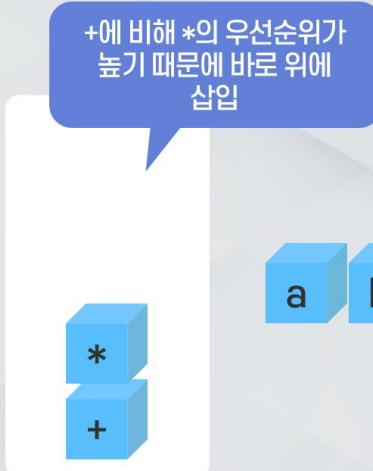
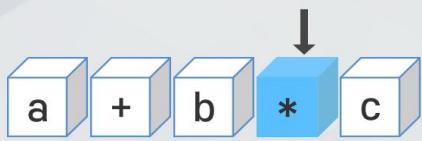
# 중위 표기 수식의 후위 표기 변환

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

##### Case 1: $a + b * c$

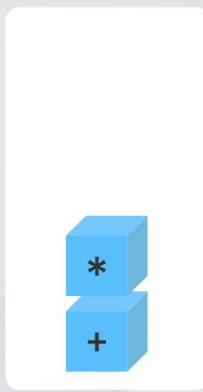
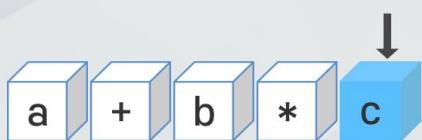


## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

##### Case 1: $a + b * c$



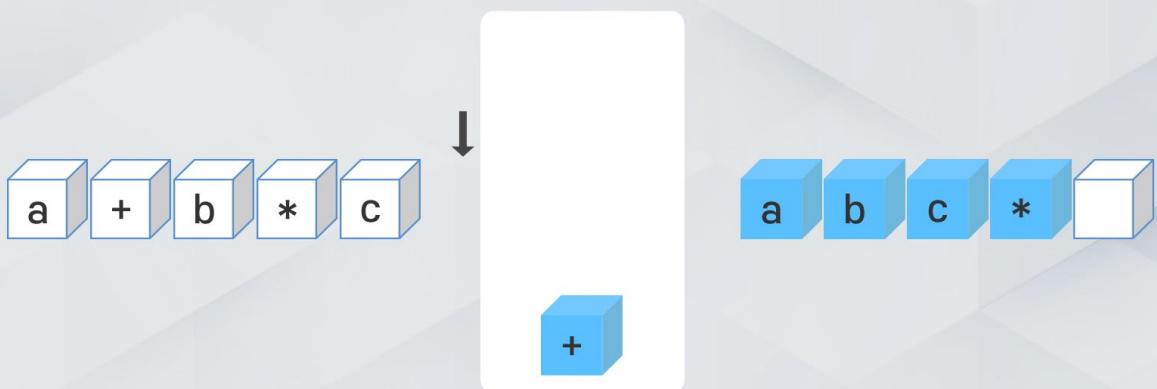
# 중위 표기 수식의 후위 표기 변환



스택의 활용

중위 표기 수식의 후위 표기 변환

## 1 중위와 후위 표기 수식의 비교

◎ Case 1:  $a + b * c$ 

스택의 활용

중위 표기 수식의 후위 표기 변환

## 1 중위와 후위 표기 수식의 비교

◎ Case 1:  $a + b * c$ 

# 중위 표기 수식의 후위 표기 변환

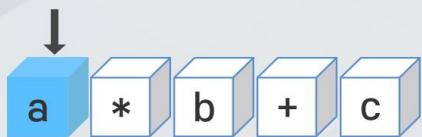


## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 2:  $a * b + c$

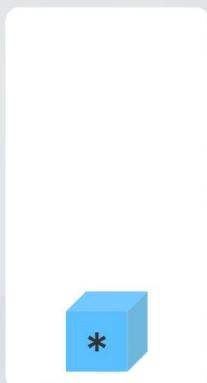


## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 2:  $a * b + c$



# 중위 표기 수식의 후위 표기 변환

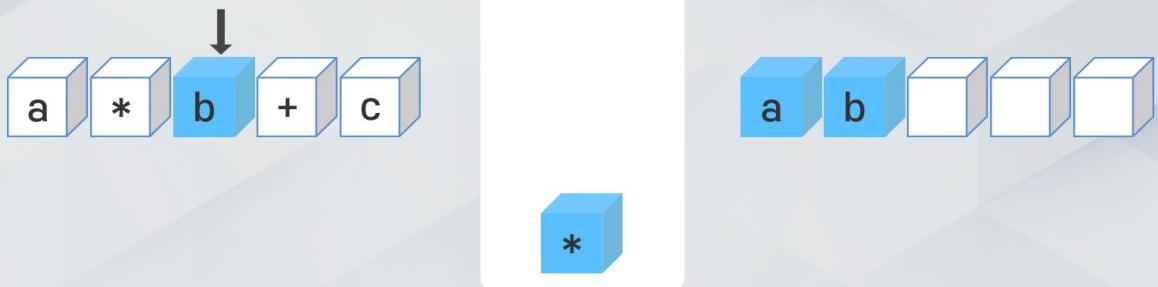


## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 2:  $a * b + c$



## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 2:  $a * b + c$



# 중위 표기 수식의 후위 표기 변환



## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 2:  $a * b + c$



## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 2:  $a * b + c$





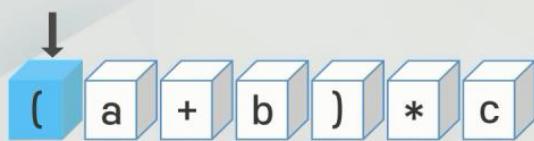
# 중위 표기 수식의 후위 표기 변환

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 3:  $(a+b)*c$



왼쪽 괄호 '('가 나오면  
무조건 삽입



## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 3:  $(a+b)*c$





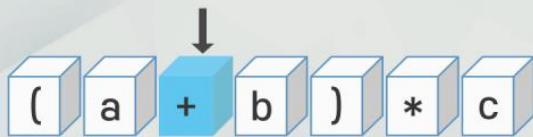
# 중위 표기 수식의 후위 표기 변환

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

##### Case 3: $(a+b)*c$



스택에서 '('는 가장  
우선순위가 낮은 연산자로 취급함.  
따라서 +를 위에 삽입

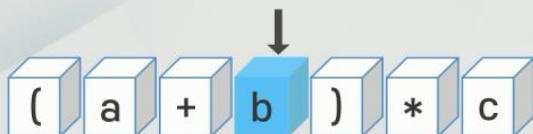


## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

##### Case 3: $(a+b)*c$





# 중위 표기 수식의 후위 표기 변환

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 3:  $(a+b)*c$



')' 가 나오면 스택에서  
 '(' 가 나올 때까지 모든 연산자를  
 꺼내서 출력



## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 3:  $(a+b)*c$



\*





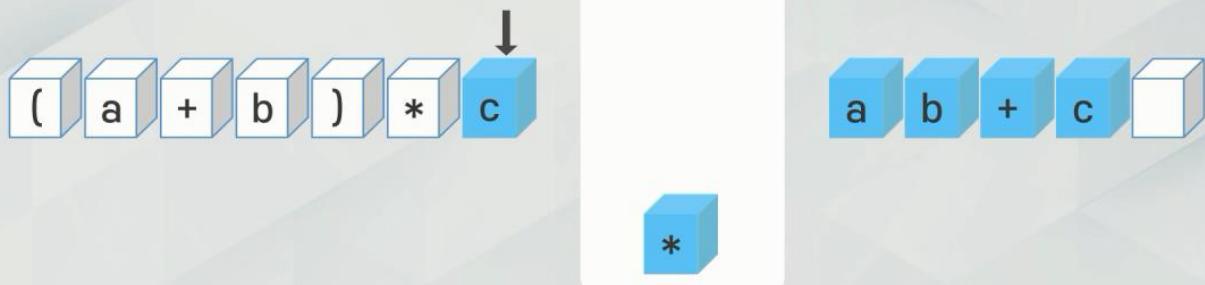
# 중위 표기 수식의 후위 표기 변환

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 3:  $(a+b)*c$

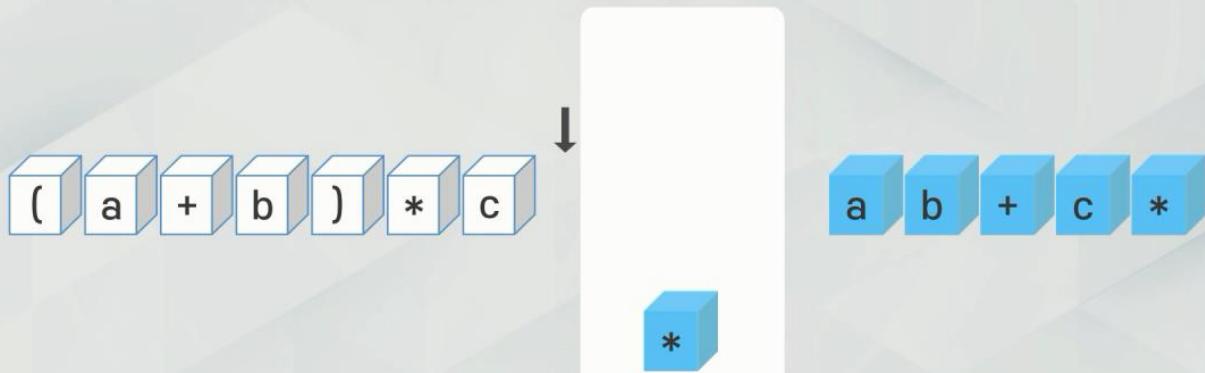


## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

◎ Case 3:  $(a+b)*c$



모든 피연산자 출력 후, Stack에 있는 연산자도 모두 꺼내 출력





# 중위 표기 수식의 후위 표기 변환

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

##### ● 중위 표기에서 후위 표기 변환 알고리즘

▶ 변환 알고리즘 정리

중위 표기식에서 피연산자를 만나면 그대로 출력

연산자(+-/\*)를 만나면 잠시 스택에 저장

- 스택에 저장하기 전에, 현재 연산자보다 우선 순위가 같거나 높은 연산자들은 모두 스택에서 꺼내 출력

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 1 중위와 후위 표기 수식의 비교

##### ● 중위 표기에서 후위 표기 변환 알고리즘

왼쪽 괄호 '('가 나오면

- ▶ 무조건 스택에 저장
- ▶ 이후 우선순위가 가장 낮은 연산자로 취급

오른쪽 괄호 ')'가 나오면

- ▶ 스택에서 왼쪽 괄호 위에 쌓여있는 모든 연산자를 출력

# 중위 표기 수식의 후위 표기 변환



## 스택의 활용

### ■ 중위 표기 수식의 후위 표기 변환

#### 2 중위 표기의 후위 표기 변환 알고리즘

- ▶ Stack, evalPostfix() 포함

```
from stackClass import Stack
from EvalPostfixExpr import evalPostfix
```

- ▶ 연산자의 우선순위 계산 함수

```
from stackClass import Stack
from EvalPostfixExpr import evalPostfix

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

## 스택의 활용

### ■ 중위 표기 수식의 후위 표기 변환

#### 2 중위 표기의 후위 표기 변환 알고리즘

- 중위 표기에서 후위 표기 변환 알고리즘

```
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)
```

# 중위 표기 수식의 후위 표기 변환



## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 2 중위 표기의 후위 표기 변환 알고리즘

##### ● 중위 표기에서 후위 표기 변환 알고리즘

```

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 :                      # 피연산자
    output.append(term)

while not s.isEmpty() :
    output.append(s.pop())

return output

```

## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

#### 2 중위 표기의 후위 표기 변환 알고리즘

##### ● 테스트 프로그램

```

str1 = '8 / 2 - 3 + ( 3 * 2 )'
infix1 = str1.split()
#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)

```

# 중위 표기 수식의 후위 표기 변환



## 스택의 활용

### 중위 표기 수식의 후위 표기 변환

## 2 중위 표기의 후위 표기 변환 알고리즘

### ● 테스트 프로그램

```
C:\WINDOWS\system32\cmd.exe
중위표기: ['8', '/', '2', '+', '3', '(', '3', '*', '2', ')']
후위표기: ['8', '2', '/', '3', '+', '3', '2', '*', '+']
계산결과: 7.0

중위표기: ['1', '/', '2', '*', '4', '(', '1', '4', '/', '*']
후위표기: ['1', '2', '/', '4', '*', '1', '4', '/', '*']
계산결과: 0.5
```

# 중위 표기 수식의 후위 표기 변환



파일(F) 편집(E) 보기(V) 프로젝트(P) 디버그(D) 테스트(S) 브레이크(B) 도구(T) 확장(X) 찾(W) 도움말(H) 검색(Ctrl+Q) 교재 파이썬

설명서 보기 - 줄다 보기

EvalPostfixExpr.py

```

1     from stackClass import Stack
2     from EvalPostfixExpr import evalPostfix
3
4     def precedence (op):
5         if (op=='(' or op==')') : return 0;
6         elif (op=='+' or op=='-') : return 1;
7         elif (op=='*' or op=='/') : return 2;
8         else : return -1
9
10    def Infix2Postfix( expr ):
11        s = Stack()
12        output = []
13        for term in expr :
14            if term in '(' :
15                s.push('(')
16
17            elif term in ')' :
18                while not s.isEmpty() :
19                    op = s.pop()
20                    if op=='(' :
21                        break;
22                    else :

```

줄다  
설명서 보기 선택(S)

오류 목록: 경고장 출처

0% 207% 5% 표제가 결제되지 않을

10% 20% 30% 40% 50% 60% 70% 80%

100% 110% 120% 130% 140% 150% 160% 170% 180% 190% 200%

210% 220% 230% 240% 250% 260%

## 실습 단계

중위 표기 수식을 후위 표기 식으로 변환, 계산 과정

후위 표기 수식: evalPostfix

우선 순위

Infix2Postfix 알고리즘

테스트 코드

소스 코드 실행 결과



# 미로 탐색

스택의 활용



## 1 미로 탐색 문제

### 미로 탐색 문제

미로에 갇힌 생쥐가 출구를 찾는 문제

이곳을 어떻게  
빠져나가지?



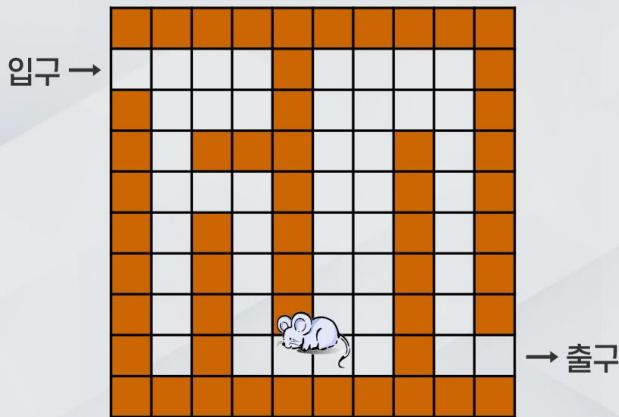
지도는 있는데 너무 복잡하네.  
자료구조를 좀 더 열심히  
공부할 걸 그랬어….

스택의 활용



## 1 미로 탐색 문제

▶ 미로의 예



|   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| → | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |   |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |   |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |   |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |   |
| 1 | 0 | 1 | 0 | 1 | m | 0 | 1 | 0 | 1 |   |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | x | → |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |



# 미로 탐색

## 스택의 활용



### 미로 탐색

#### 1 미로 탐색 문제

##### ◉ 미로 탐색 문제란?

###### 미로 탐색 전략

현재의 위치에서 가능한 방향을 어딘가에 저장해 놓았다가  
막다른 길을 만나면 저장된 다음 위치를 꺼냄

###### 어디에 저장할 것인가?

가장 최근에 만났던 분기점으로 되돌아가  
저장된 위치를 꺼내 탐색하는 방법

## 스택의 활용



### 미로 탐색

#### 1 미로 탐색 문제

##### ◉ 미로 탐색 문제란?

스택 사용

깊이 우선 탐색  
(Depth-First Search, DFS)



# 미로 탐색

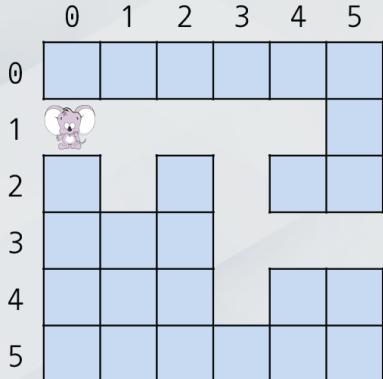
스택의 활용

미로 탐색

## 1 미로 탐색 문제

### ● 깊이 우선 탐색의 예

▶ 미로의 예



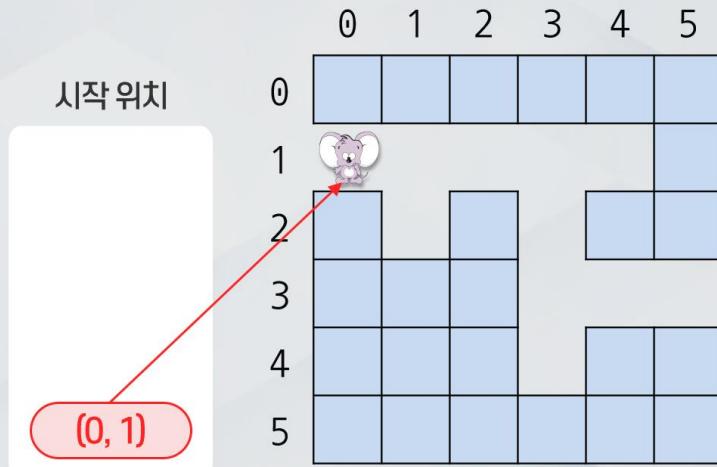
```
map = [
    ['1', '1', '1', '1', '1', '1'],
    ['e', '0', '0', '0', '0', '1'],
    ['1', '0', '1', '0', '1', '1'],
    ['1', '1', '1', '0', '0', 'x'],
    ['1', '1', '1', '0', '1', '1'],
    ['1', '1', '1', '1', '1', '1']
]
```

스택의 활용

미로 탐색

## 1 미로 탐색 문제

### ● 깊이 우선 탐색의 예





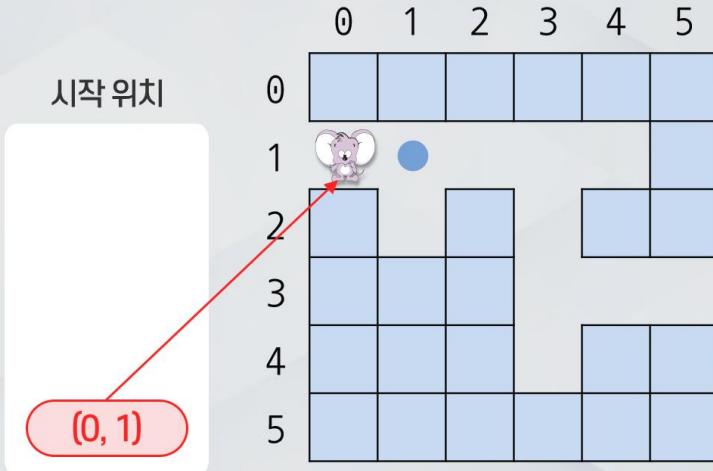
# 미로 탐색

스택의 활용

미로 탐색

## 1 미로 탐색 문제

### ◎ 깊이 우선 탐색의 예

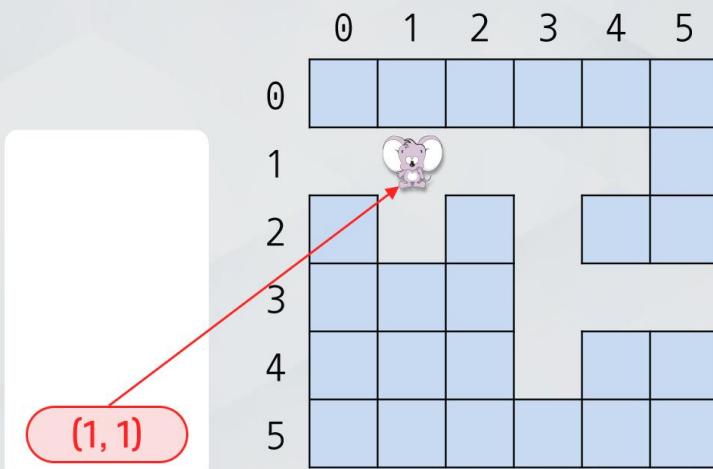


스택의 활용

미로 탐색

## 1 미로 탐색 문제

### ◎ 깊이 우선 탐색의 예





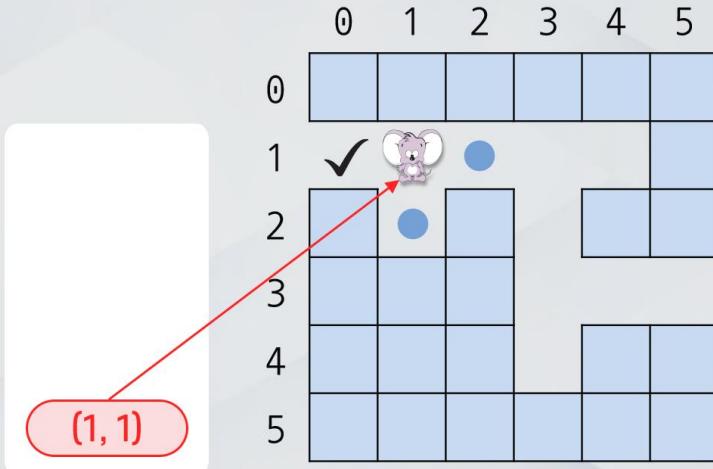
# 미로 탐색

스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예

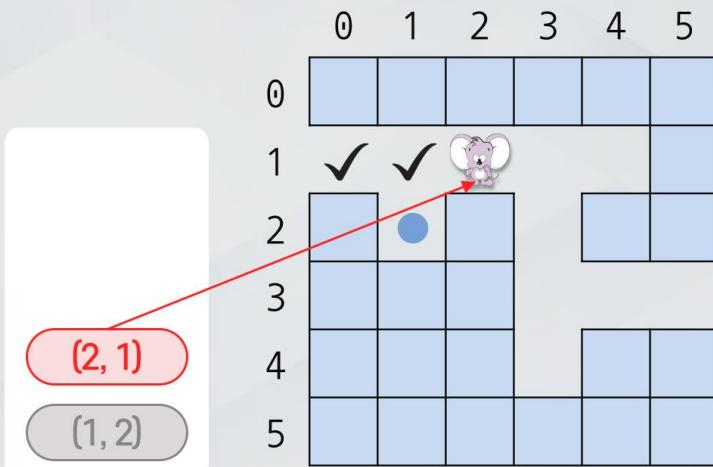


스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예





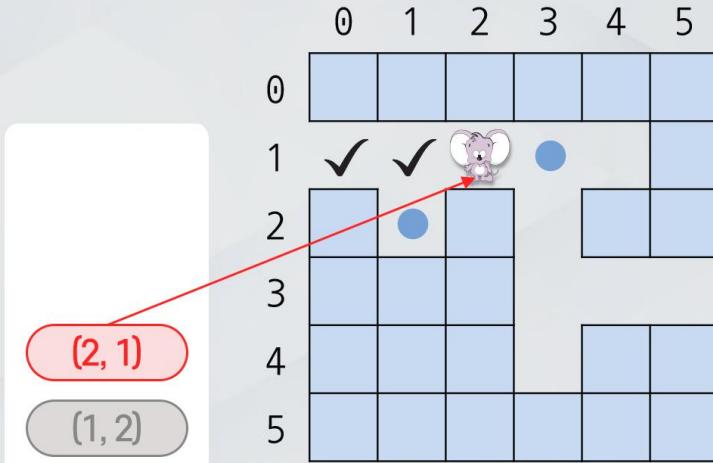
# 미로 탐색

스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예

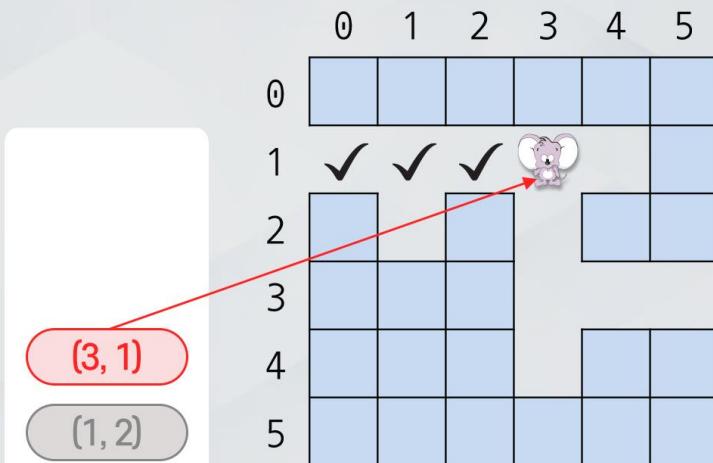


스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예





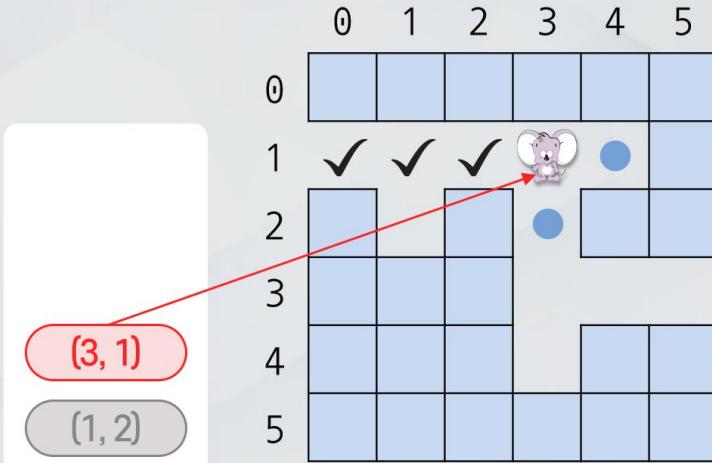
# 미로 탐색

스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예

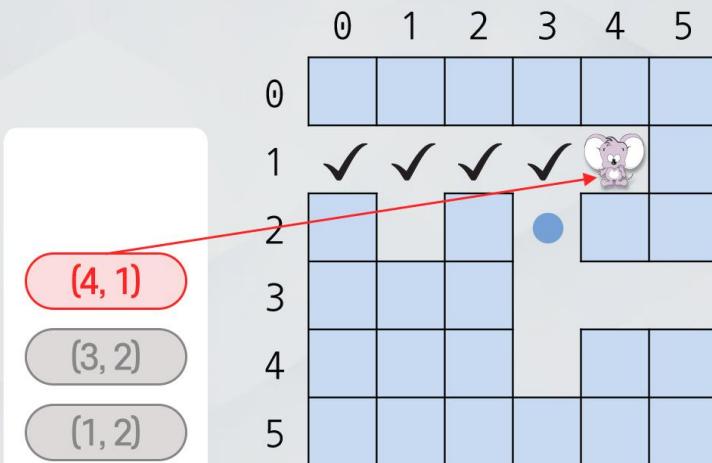


스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예





# 미로 탐색

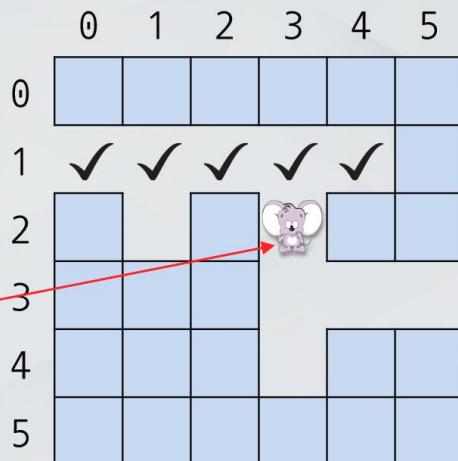
스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예

(3, 2)  
(1, 2)



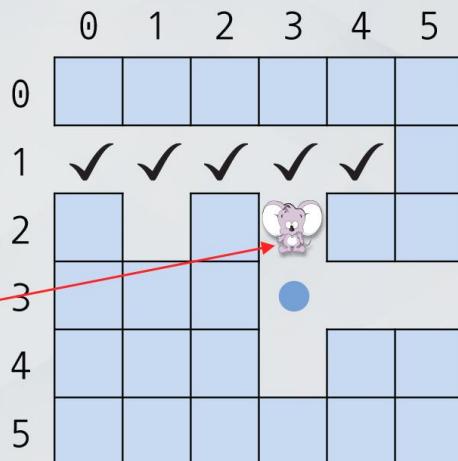
스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예

(3, 2)  
(1, 2)





# 미로 탐색

스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예

(3, 3)  
(1, 2)

|   | 0 | 1 | 2 | 3 | 4     | 5 |
|---|---|---|---|---|-------|---|
| 0 |   |   |   |   |       |   |
| 1 | ✓ | ✓ | ✓ | ✓ | ✓     |   |
| 2 |   |   |   | ✓ |       |   |
| 3 |   |   |   |   | Mouse |   |
| 4 |   |   |   |   |       |   |
| 5 |   |   |   |   |       |   |

스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예

(3, 3)  
(1, 2)

|   | 0 | 1 | 2 | 3 | 4     | 5 |
|---|---|---|---|---|-------|---|
| 0 |   |   |   |   |       |   |
| 1 | ✓ | ✓ | ✓ | ✓ | ✓     |   |
| 2 |   |   |   | ✓ |       |   |
| 3 |   |   |   |   | Mouse |   |
| 4 |   |   |   |   | ●     |   |
| 5 |   |   |   |   | ●     |   |



# 미로 탐색

스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예

- (4, 3)
- (3, 4)
- (1, 2)

|   | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 |   |   |   |   |   |   |
| 1 | ✓ | ✓ | ✓ | ✓ | ✓ |   |
| 2 |   |   |   | ✓ |   |   |
| 3 |   |   |   |   | ✓ |   |
| 4 |   |   |   |   |   |   |
| 5 |   |   |   |   |   |   |

스택의 활용

미로 탐색

## 1 미로 탐색 문제

◎ 깊이 우선 탐색의 예

- (4, 3)
- (3, 4)
- (1, 2)

|   | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 |   |   |   |   |   |   |
| 1 | ✓ | ✓ | ✓ | ✓ | ✓ |   |
| 2 |   |   |   | ✓ |   |   |
| 3 |   |   |   |   | ✓ |   |
| 4 |   |   |   |   |   |   |
| 5 |   |   |   |   |   |   |

# 미로 탐색



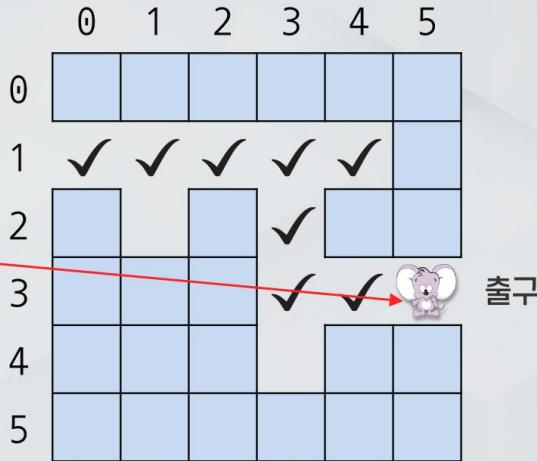
## 스택의 활용

### 미로 탐색

#### 1 미로 탐색 문제

##### ◎ 깊이 우선 탐색의 예

- (5, 3)
- (3, 4)
- (1, 2)



## 스택의 활용

### 미로 탐색

#### 2 깊이 우선 탐색 알고리즘

$(x, y)$ 가 갈 수 있는 칸인지 검사

- ▶ 미로 내부
- ▶ 출구이거나 아직 방문하지 않은 경우

```
def isValidPos(x, y) :
    if x < 0 or y < 0 or x >= MAZE_SIZE or y >= MAZE_SIZE :
        return False
    else :
        return map[y][x] == '0' or map[y][x] == 'x'
    # (x,y)가 미로 밖이면 → 갈 수 없음
    # 출구( 'x' )이거나 방( '0' )이면 갈 수 있음
```



# 미로 탐색

## 스택의 활용

### 미로 탐색

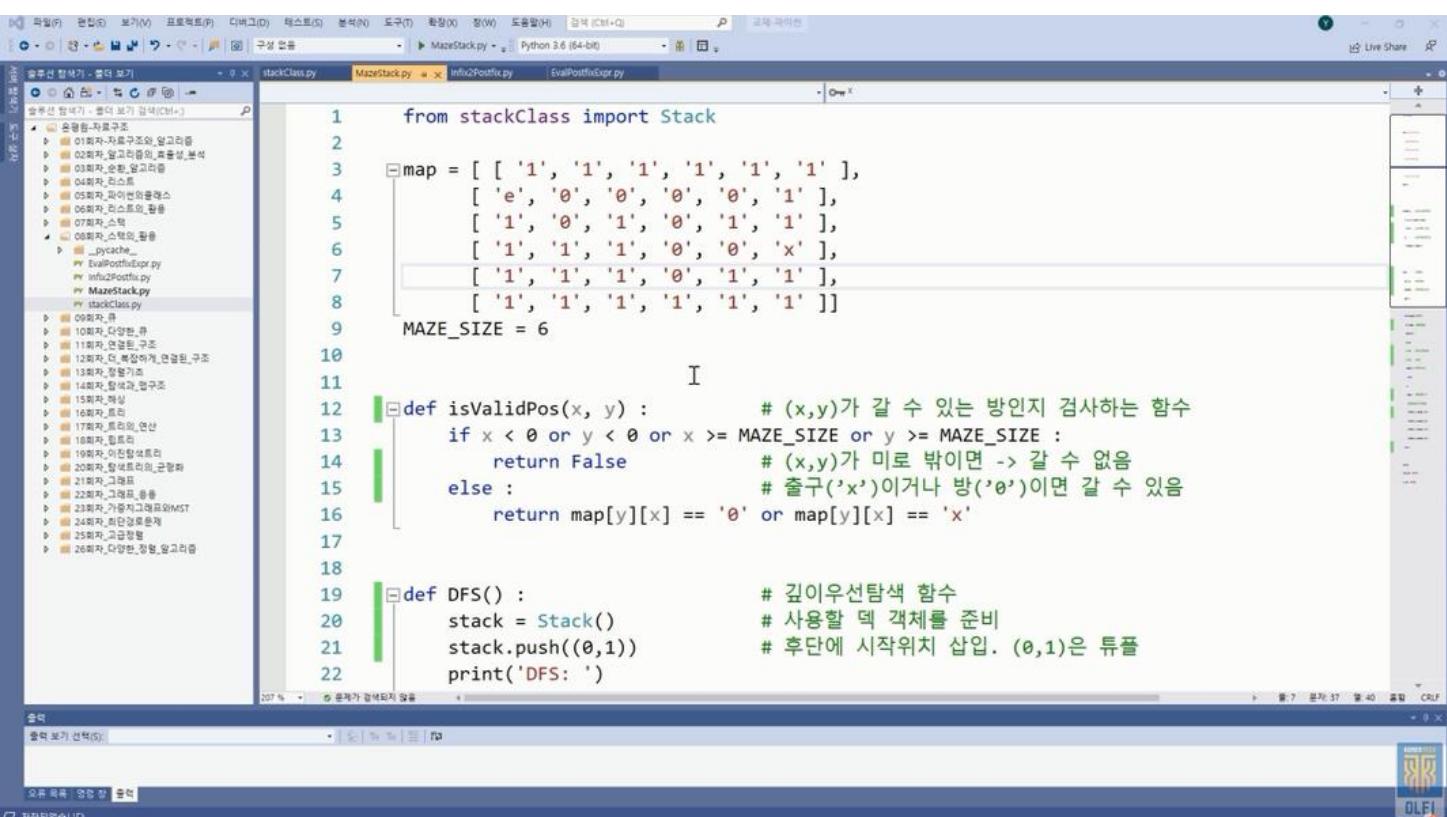
#### 2 깊이 우선 탐색 알고리즘

```

def DFS() :
    stack = Stack()
    stack.push((0,1))
    print('DFS: ')
    while not stack.isEmpty():
        here = stack.pop()
        print(here, end='→')
        (x,y) = here
        if (map[y][x] == 'x') :
            return True
        else :
            map[y][x] = '.'
            if isValidPos(x - 1, y): stack.push((x - 1, y)) # 좌
            if isValidPos(x + 1, y): stack.push((x + 1, y)) # 우
            if isValidPos(x, y - 1): stack.push((x, y - 1)) # 상
            if isValidPos(x, y + 1): stack.push((x, y + 1)) # 하

```

# 미로 탐색



```

1  from stackClass import Stack
2
3  map = [ [ '1', '1', '1', '1', '1', '1' ],
4         [ 'e', '0', '0', '0', '0', '1' ],
5         [ '1', '0', '1', '0', '1', '1' ],
6         [ '1', '1', '1', '0', '0', 'x' ],
7         [ '1', '1', '1', '0', '1', '1' ],
8         [ '1', '1', '1', '1', '1', '1' ] ]
9  MAZE_SIZE = 6
10
11
12  def isValidPos(x, y) :          # (x,y)가 갈 수 있는 방인지 검사하는 함수
13      if x < 0 or y < 0 or x >= MAZE_SIZE or y >= MAZE_SIZE :
14          return False             # (x,y)가 미로 밖이면 -> 갈 수 없음
15      else :                     # 출구('x')이거나 벽('0')이면 갈 수 있음
16          return map[y][x] == '0' or map[y][x] == 'x'
17
18
19  def DFS() :                    # 깊이우선탐색 함수
20      stack = Stack()           # 사용할 데 객체를 준비
21      stack.push((0,1))         # 후단에 시작위치 삽입. (0,1)은 튜플
22      print('DFS: ')

```

## 실습 단계

map은 2차원 배열(list의 list)로 만듭니다.

입력은 e, 출력은 x, 갈 수 있는 방은 0, 벽은 1로 표현합니다.

MAZE\_SIZE = 6: 한 방향의 크기입니다.

출구 찾기 성공: True반환/ 출구 찾기 실패: False 반환

실행 결과 확인