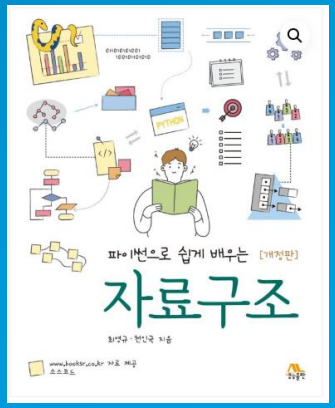


04 CHAPTER

스택

4장. 스택

4.1 스택이란?

4.2 스택의 구현

4.3 스택의 응용: 괄호 검사

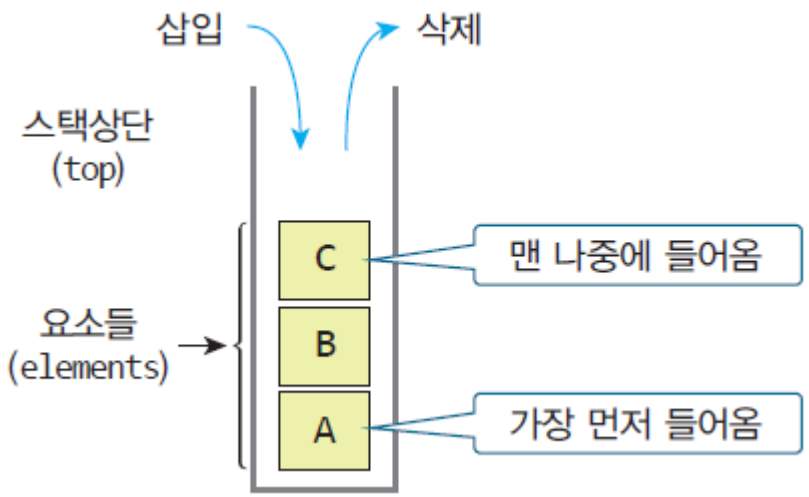
4.4 스택의 응용: 수식의 계산

4.5 스택의 응용: 미로 탐색

4.1 스택이란?

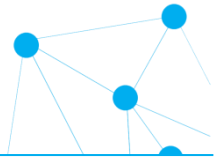
- 스택(stack)
  - "stack": 쌓아놓은 더미
  - 후입선출(LIFO:Last-In First-Out)의 자료구조
  - 가장 최근에 들어온 데이터가 가장 먼저 나감

- 스택의 구조



  - 자료의 입출력이 상단으로만 가능

스택의 추상 자료형



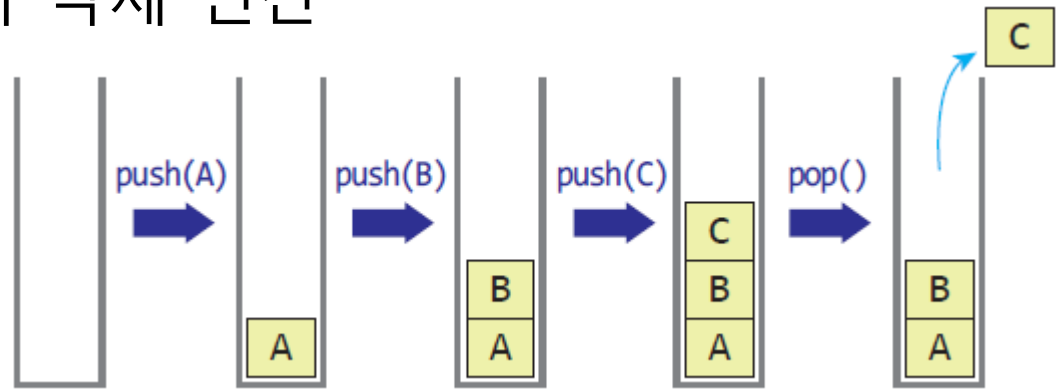
데이터: 후입선출(LIFO)의 접근 방법을 유지하는 항목들의 모음

연산

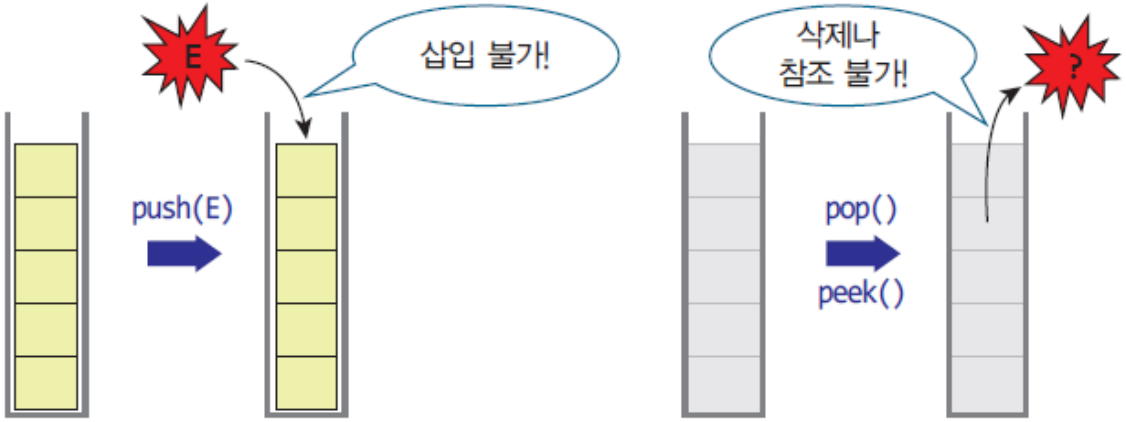
- push(e): 요소 e를 스택의 맨 위에 추가한다.
- pop(): 스택의 맨 위에 있는 요소를 꺼내 반환한다.
- isEmpty(): 스택이 비어있으면 True를 아니면 False를 반환한다.
- isFull(): 스택이 가득 차 있으면 True를 아니면 False를 반환한다.
- peek(): 스택의 맨 위에 있는 항목을 삭제하지 않고 반환한다.

스택의 연산들

- 삽입과 삭제 연산



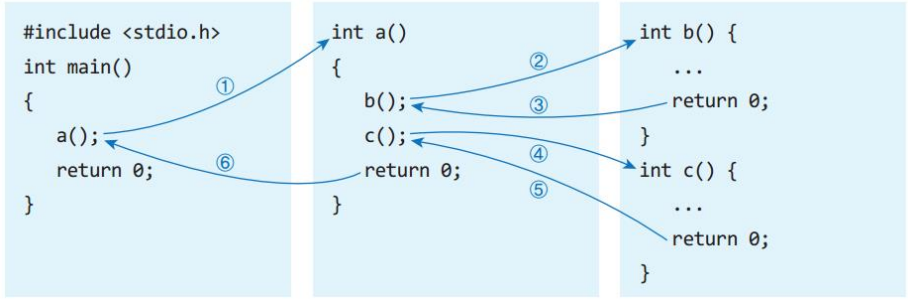
- 두 가지 오류 상황



스택의 용도

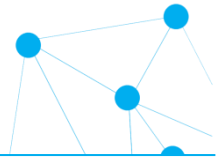
- 편집기의 되돌리기
- 이전페이지로 이동

- 함수 호출: 시스템 스택

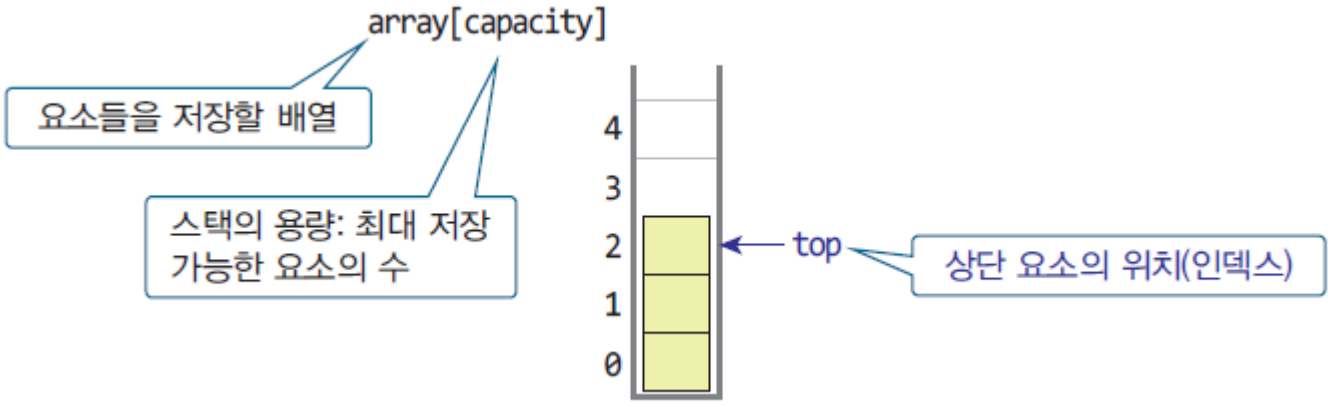


- 괄호 검사
- 계산기
  - 후위 표기식 계산, 중위 표기식의 후위 표기식 변환
- 미로 탐색 등

4.2 스택의 구현

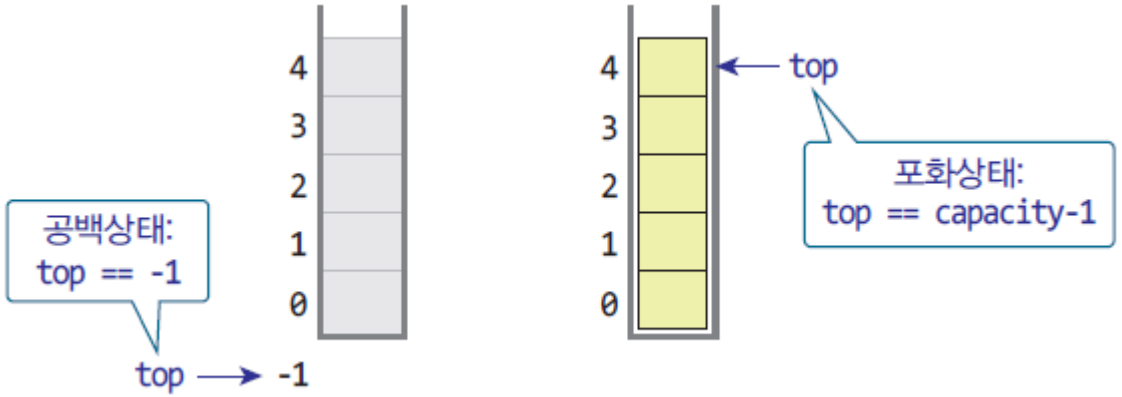


- 배열을 이용한 스택의 구조
  - 용량이 고정된 스택



스택의 연산

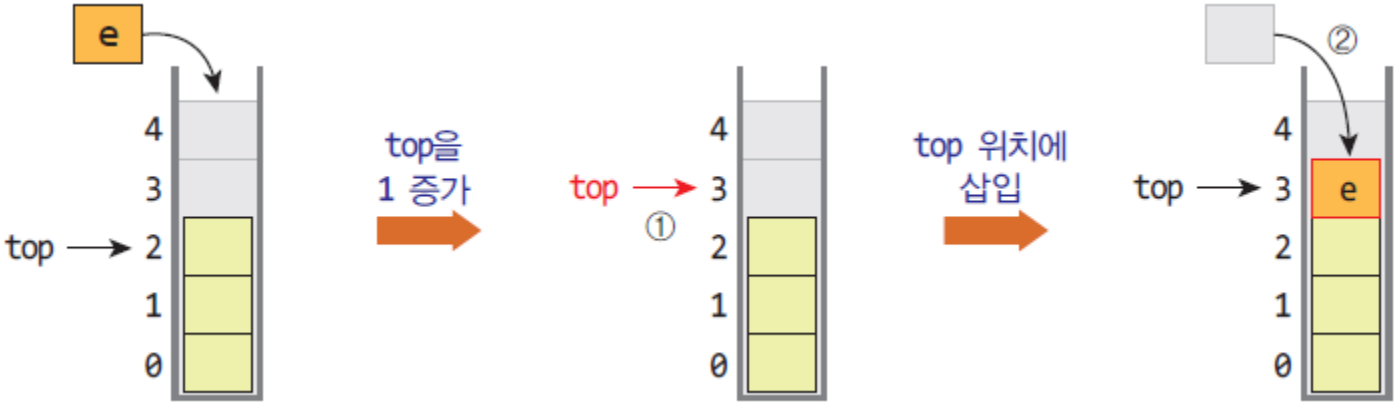
- 공백상태와 포화상태 검사




```
def isEmpty( ) :
    if top == -1 : return True
    else : return False
```

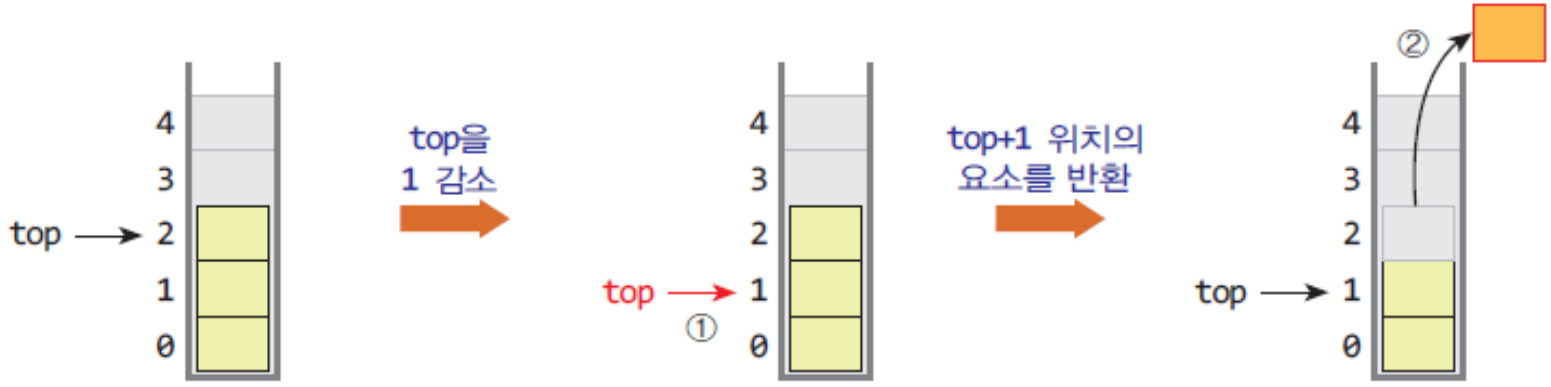
```
def isFull( ) :
    return top == capacity-1
```

- 새로운 요소 e를 삽입하는 push(e)



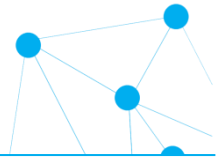

```
def push( e ) :
    global top          # top은 전역변수
    if not isFull() :
        top += 1
        array[top] = e
    else :
        print("stack overflow")
        exit()
```

- 상단 요소를 삭제하는 pop()




```
def pop( ) :
    global top          # top은 전역변수
    if not isEmpty():
        top -= 1
        return array[top+1]
    else:
        print("stack underflow")
        exit()
```

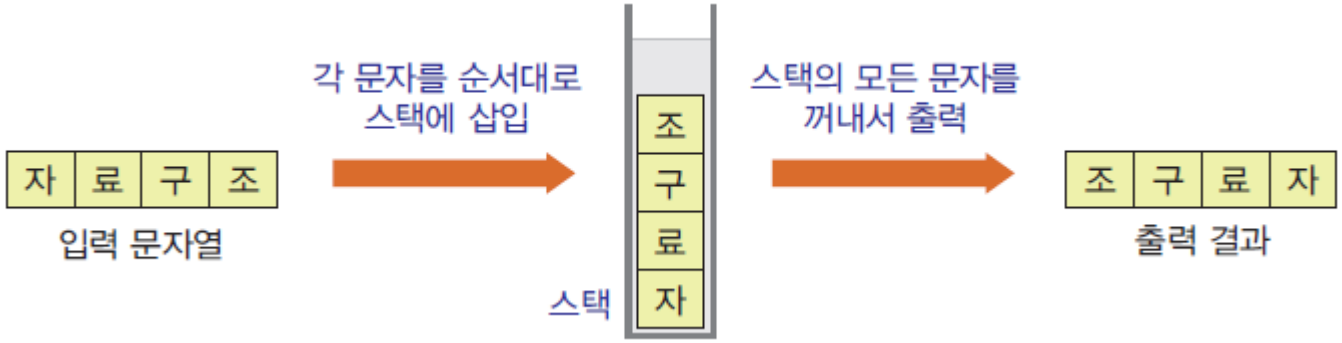
스택의 구현(클래스 버전)



```
class ArrayStack :
    def __init__( self, capacity ):
        self.capacity = capacity
        self.array = [None]*self.capacity
        self.top = -1

    # 스택의 연산들을 멤버 함수로 구현
    def isEmpty( self ) :
        return self.top == -1

    def isFull( self ) :
        return self.top == self.capacity-1
```

```
    def push( self, e ):
        if not self.isFull() :
            self.top += 1
            self.arry[self.top] = e
        else: pass

    def pop( self ):
        if not self.isEmpty():
            self.top -= 1
            return self.array[self.top+1]
        else: pass

    def peek( self ):
        if not self.isEmpty():
            return self.array[self.top]
        else: pass
```

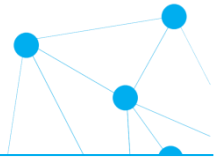
문자열 역순 출력 프로그램




```
from ArrayStack import ArrayStack
s = ArrayStack(100)

msg = input("문자열 입력: ")
for c in msg :
    s.push(c)

print("문자열 출력: ", end='')
while not s.isEmpty():
    print(s.pop(), end='')
print()
```

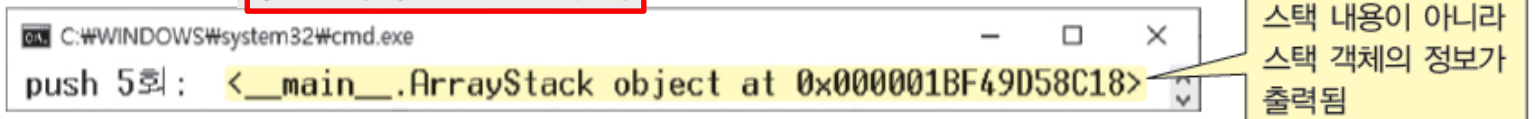
```
C:₩WINDOWS₩system32₩cmd.exe
문자열 입력: 안녕하세요. 반갑습니다.
문자열 출력: .다니습갑반 .요세하녕안
```

연산들의 추가: 화면 출력



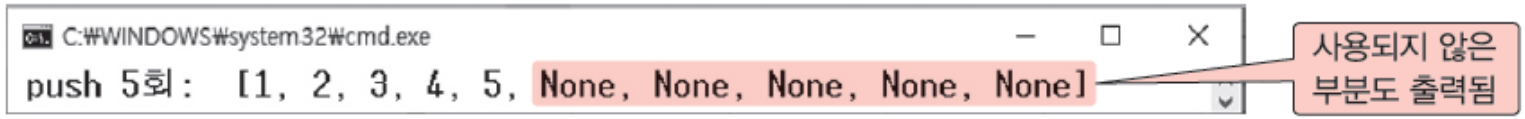
- 방법 1

```
s = ArrayStack(10)
for i in range(1,6):
    s.push(i)
print(' push 5회: ', s)
```



- 방법 2

```
print(' push 5회: ', s.array)
```

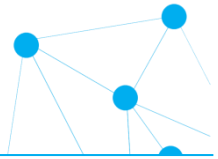


- 연산자 중복 + 슬라이싱

```
def __str__(self ) :
    return str(self.array[0:self.top+1])
```

```
print(' push 5회: ', s)
```

4.3 스택의 응용: 괄호 검사

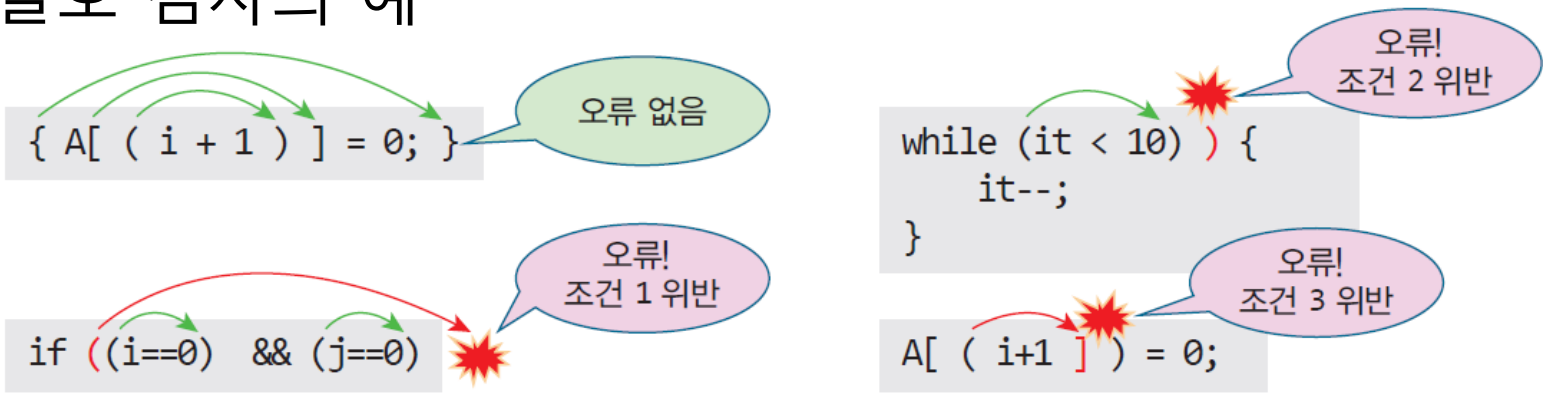


- 괄호들이 문법에 맞게 사용되었는지를 검사
  - 프로그래밍 언어, HTML 문서, 수식 표기 등
- 예: C언어 소스코드

```
int find_array_max(int score[], int n)
{
    int i, tmp=score[0];
    for( i=1 ; i<n ; i++ ) {
        if( score[i] > tmp ) {
            tmp = score[i];
        }
    }
    return tmp;
}
```

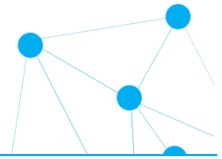
괄호 사용의 조건

- 조건들
  - ① 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 한다.
  - ② 같은 타입의 괄호에서 왼쪽 괄호가 오른쪽 괄호보다 먼저 나와야 한다.
  - ③ 서로 다른 타입의 괄호 쌍이 서로를 교차하면 안 된다.
- 괄호 검사의 예



- 괄호 검사를 위해 스택을 사용

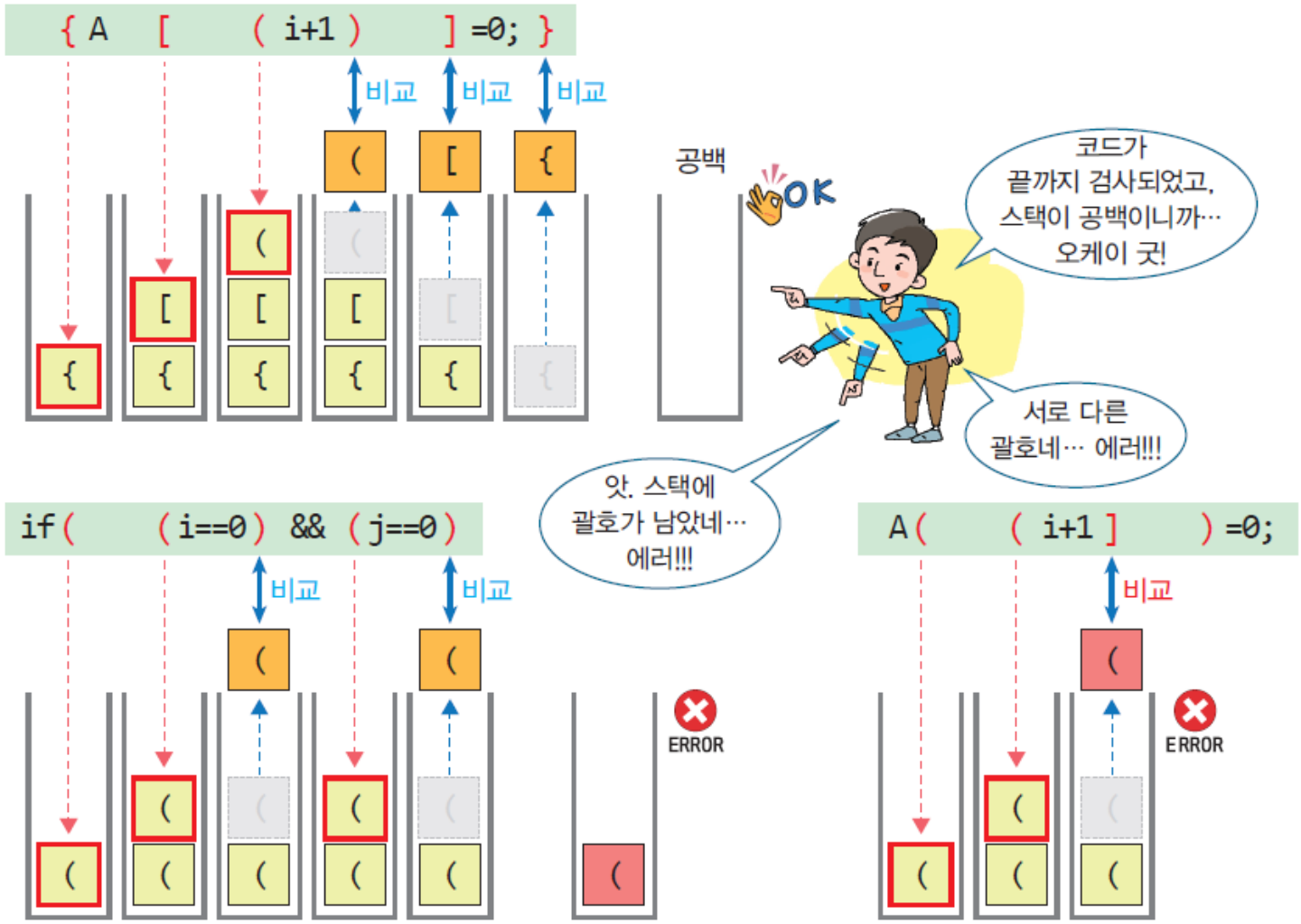
괄호 검사 방법



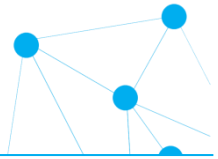
- 가장 가까운 거리에 있는 괄호끼리 쌍을 이루어야 됨

- 검사 방법
  - 문자를 저장하는 스택을 준비한다. 처음에는 공백상태가 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


{ A [(i+1)] =0; }
비교
비교
비교
공백
OK
코드가
끝까지 검사되었고,
스택이 공백이니까…
오케이 굿!
서로 다른
괄호네… 에러!!!
앗. 스택에
괄호가 남았네…
에러!!!
if((i==0) && (j==0)
비교
비교
ERROR
A((i+1]) =0;
비교
ERROR

괄호 검사 알고리즘

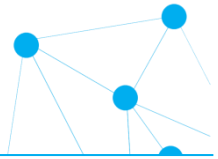


```
def checkBrackets(statement):
    stack = ArrayStack(100)
    for ch in statement:
        if ch=='{' or ch=='[' or ch=='(' :
            stack.push(ch)

        elif ch=='}' or ch==']' or ch==')' :
            if stack.isEmpty() :
                return False
            else :
                left = stack.pop()
                if (ch == "}" and left != "{") or \
                   (ch == "]" and left != "[") or \
                   (ch == ")" and left != "(") :
                    return False

    return stack.isEmpty()
```

```
s1 = "{ A[ (i+1) ] = 0; } "
s2 = "if( (i==0) && (j==0)"
s3 = "A[ ( i+1 ] ) = 0;   "
print(s1, " ---> ", checkBrackets(s1))
print(s2, " ---> ", checkBrackets(s2))
print(s3, " ---> ", checkBrackets(s3))
```

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

소스 파일에서 괄호검사



- 내장함수 open() 사용

```
filename = "ArrayStack.h"
infile = open(filename , "r")
str = infile.read()
infile.close()

print("소스파일", filename, " ---> ", checkBrackets(str))
```

파일을 읽기 모드로 열어 파일객체 infile에 저장

파일객체의 read() 메소드를 이용해 모든 데이터를 문자열로 읽어 str에 저장

파일 핸들러를 반드시 닫아주어야 함

문자열 str에 대해 괄호 검사

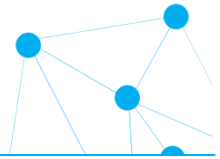
```
C:₩WINDOWS₩system32₩cmd.exe
소스파일 ArrayStack.h ---> True
```

- with ~ as 사용

```
filename = "ArrayStack.h"
with open(filename , "r") as infile :
    str = infile.read()
    print("소스파일", filename, " ---> ", checkBrackets(str))
```

with open(파일경로, 모드) as 파일객체:를 이용해 파일을 여는데, 이 구문이 끝나면 자동으로 파일을 닫아줌

4.4 스택의 응용: 수식의 계산



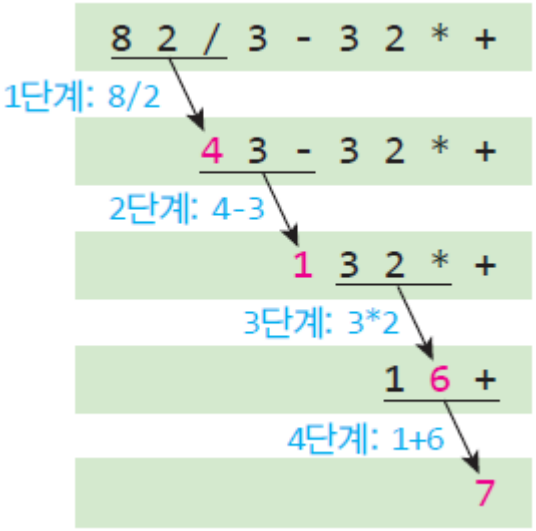
- 수식의 표기 방법 3가지

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 * +

- 후위 표기식의 장점
  - 괄호를 사용하지 않아도 계산 순서를 알 수 있음
  - 연산자의 우선순위를 생각할 필요가 없음
  - 수식을 읽으면서 바로 계산할 수 있음
- 계산기 프로그램
  - 1단계: 중위표기를 후위표기 수식으로 변환
  - 2단계: 후위표기 수식의 계산

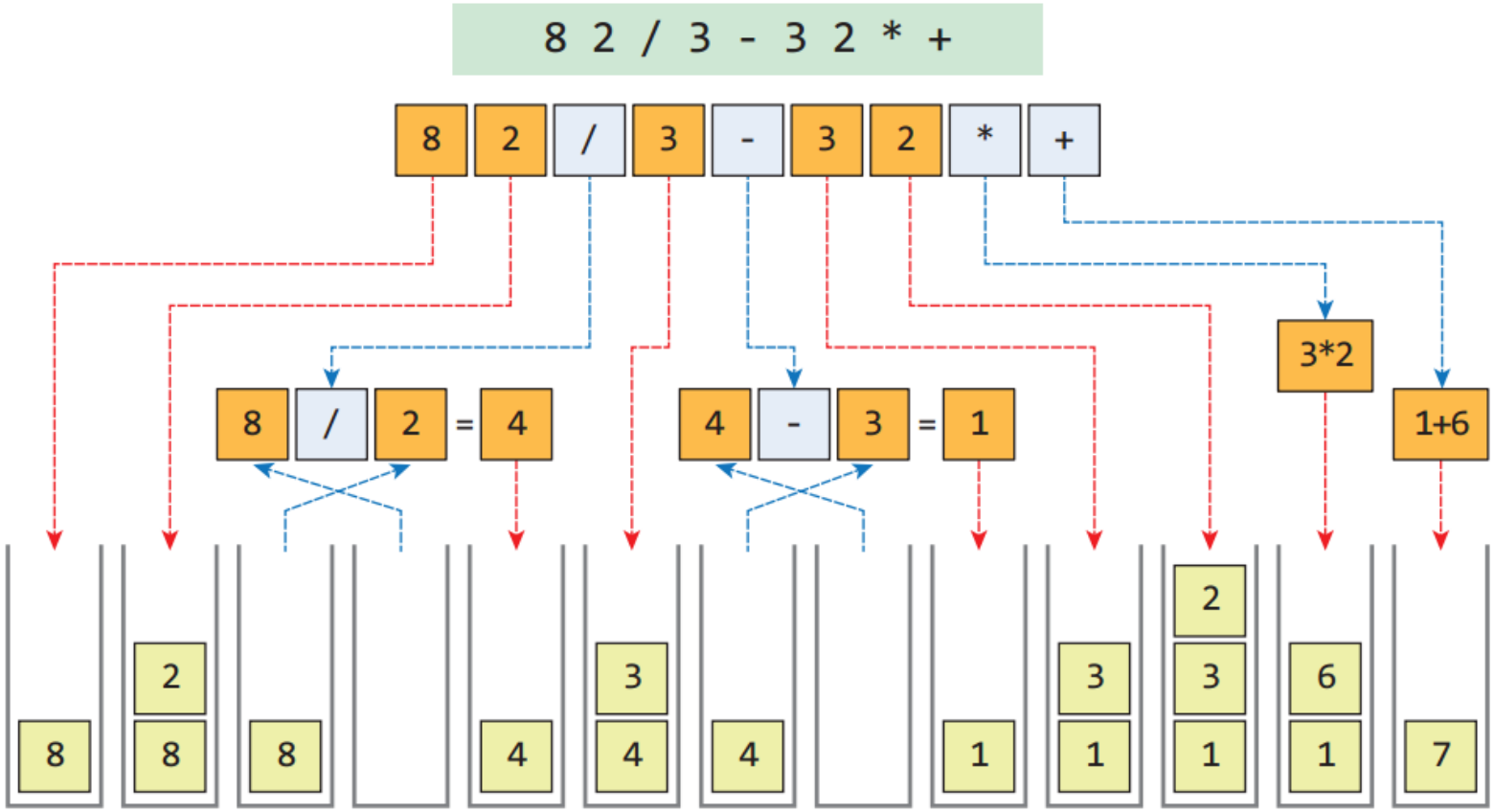
후위표기 수식 과정

- 후위표기 수식 계산 예

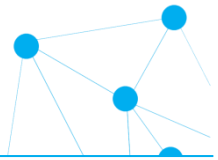


- 알고리즘: 스택을 사용
  - 수식을 스캔하다가 피연산자가 나오면 스택에 저장
  - 연산자가 나오면 스택에서 피연산자 두 개를 꺼내 연산을 실행하고 그 결과를 다시 스택에 저장
  - 이 과정을 수식이 모두 처리될 때 까지 반복
  - 마지막으로 스택에는 최종 계산 결과가 남음

후위표기 계산 과정 예



후위 표기 수식 계산 알고리즘

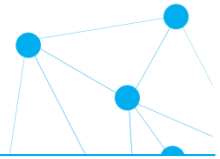


```
def evalPostfix( expr ):
    s = ArrayStack(100)

    for token in expr :
        if token in "+-*/" :
            val2 = s.pop()     # 피연산자2
            val1 = s.pop()     # 피연산자1
            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()
```

```
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
```

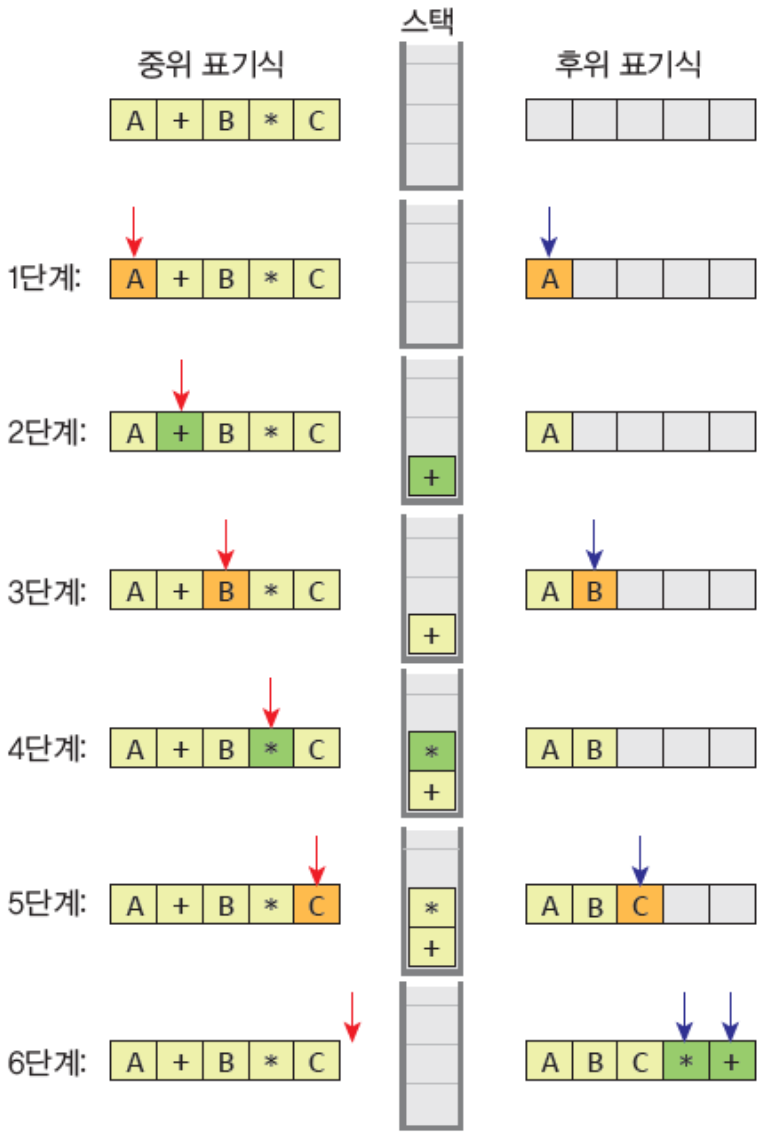
중위 표기 수식의 후위 표기 변환



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

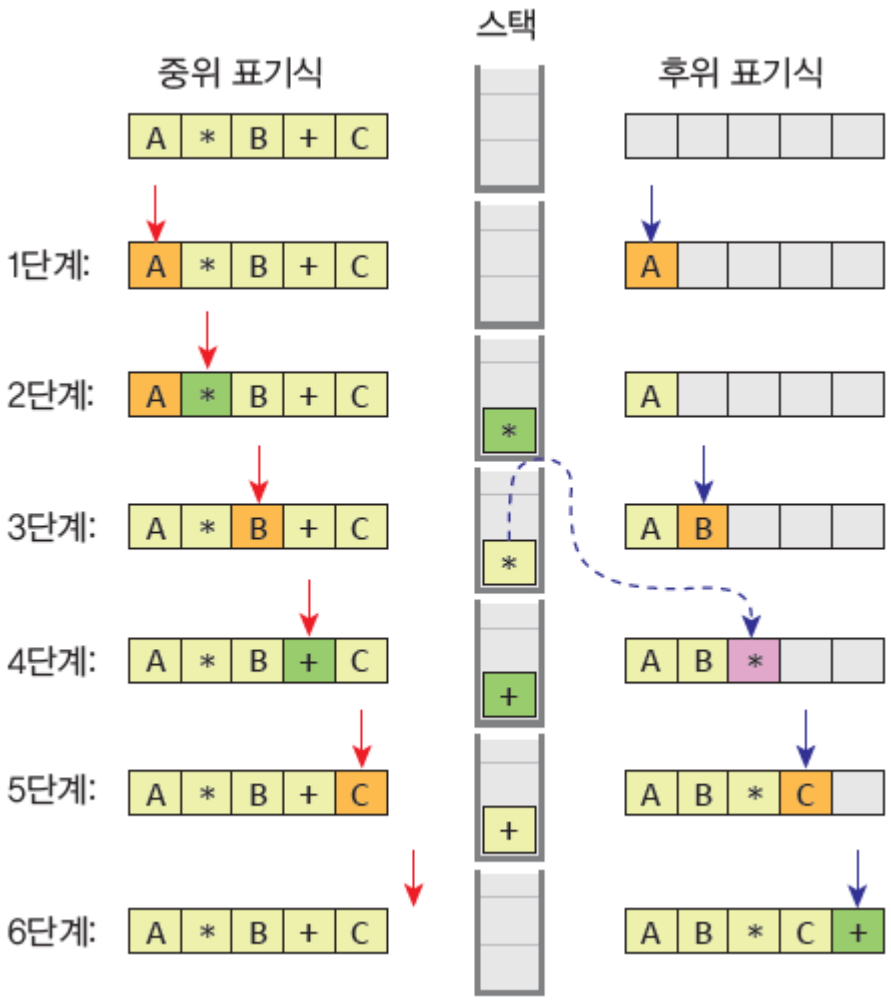
예제 1) 중위 → 후위 변환

- A+B*C



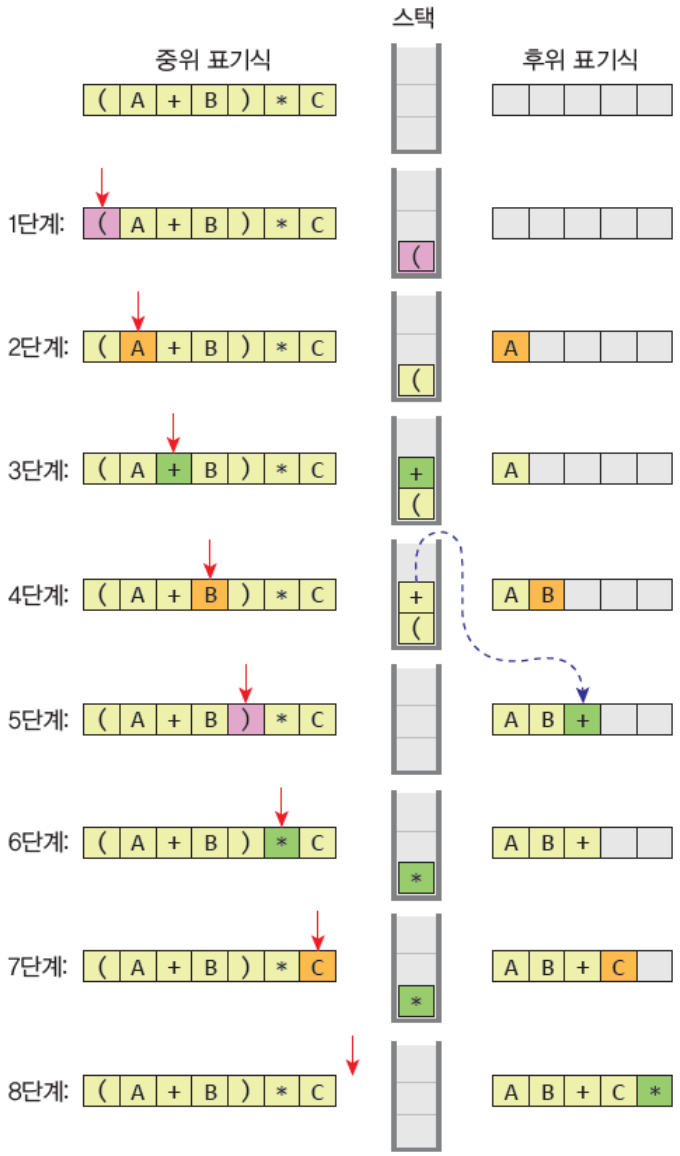
예제 2) 중위 → 후위 변환

- A*B+C

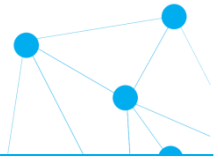


예제 3) 중위 → 후위 변환

- (A+B)*C
  - 괄호가 있는 경우



중위 → 후위 변환 알고리즘

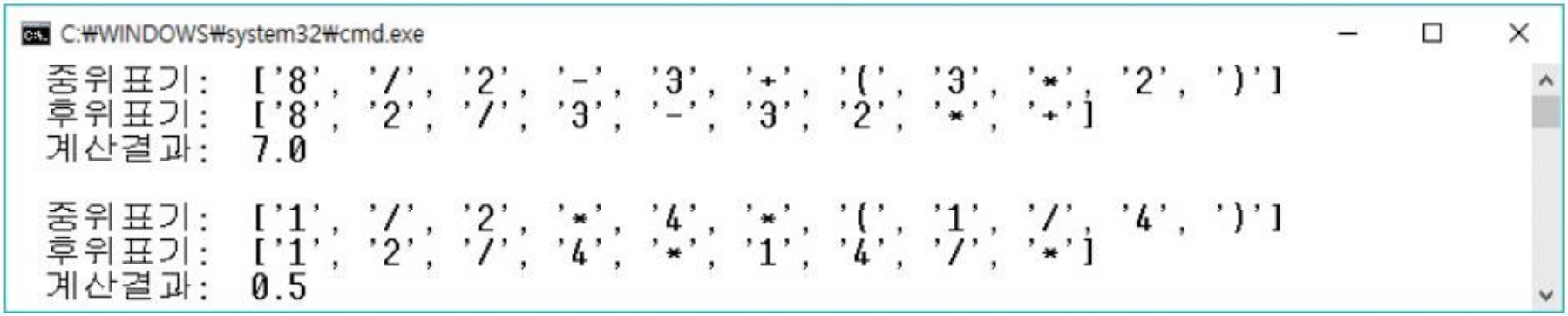


```
def Infix2Postfix( expr ):
    s = ArrayStack(100)
    output = []

    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
```

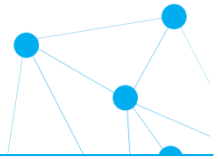
```
def precedence (op):
    if   op=='(' or op==')' : return 0
    elif op=='+' or op=='-' : return 1
    elif op=='*' or op=='/' : return 2
    else : return -1
```

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)
```

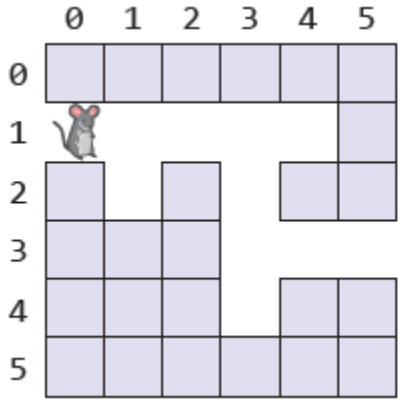


4.5 스택의 응용: 미로 탐색



- 미로 탐색
  - 미로에서 출구를 찾는 것
  - 시행 착오: 일단 하나의 경로를 선택하여 시도해 보고, 막히면 다시 다른 경로를 시도
  - 다른 경로들을 어딘가에 저장해야 함

- 미로의 표현




```
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
```

갈 수 있는 칸

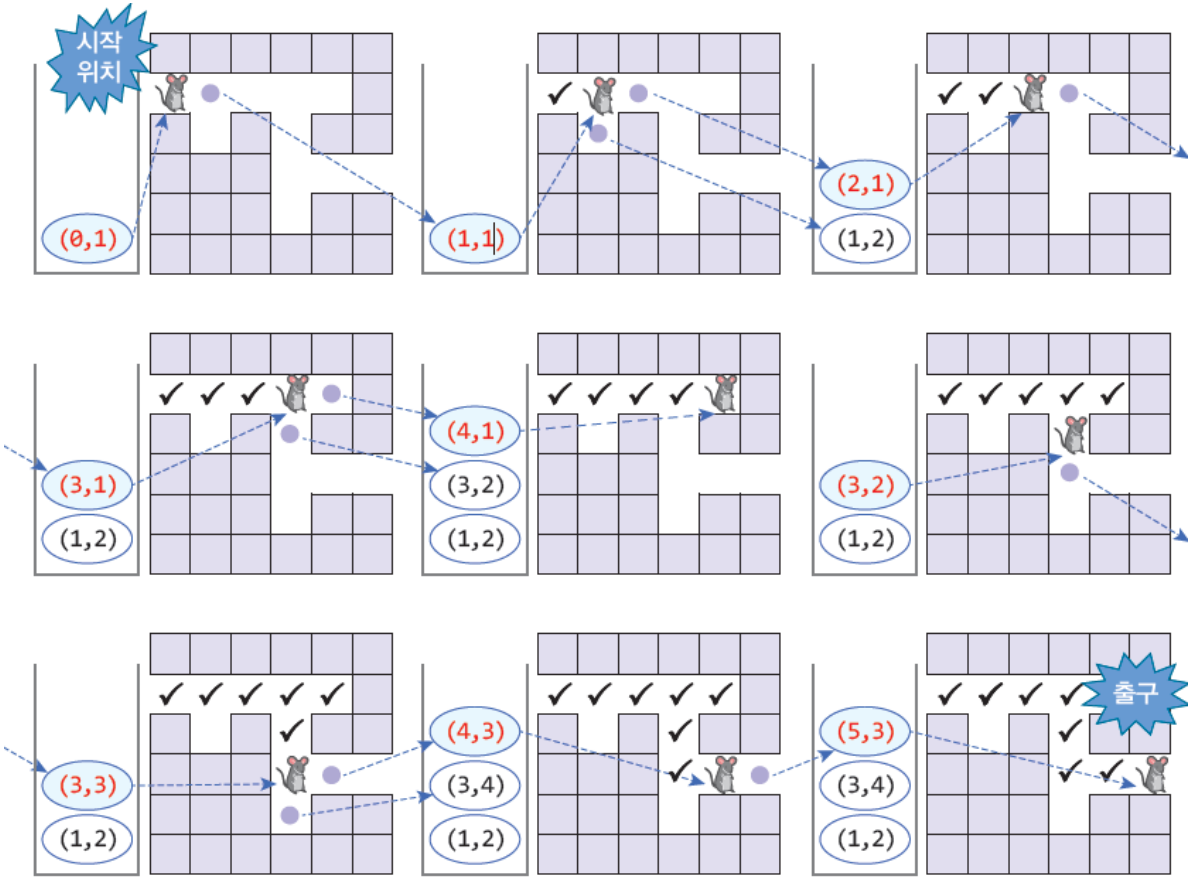
벽. 막힌 칸

출구

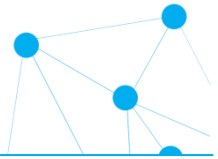
미로의 가로와 세로 크기

깊이우선탐색

- DFS, Depth First Search
  - 가던 길이 막히면 가장 최근에 있었던 갈림길로 되돌아가서 다른 길을 찾는 방법 → 스택 사용



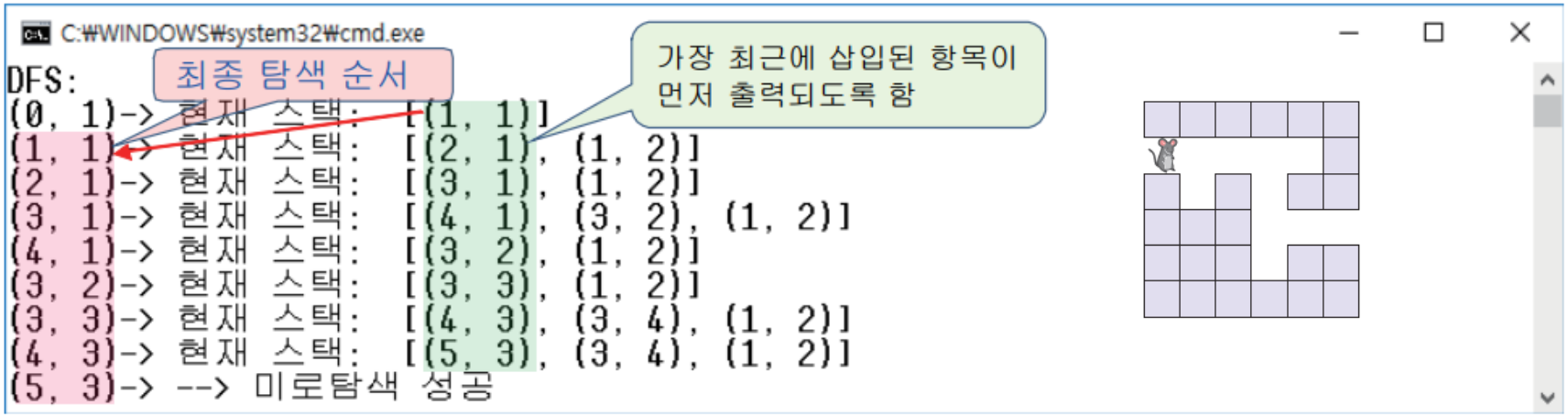
DFS 알고리즘



```
def DFS() :
    print('DFS: ')
    stack = ArrayStack(100)
    stack.push((0,1))

    while not stack.isEmpty():
        here = stack.pop()
        print(here, end='->')
        (x,y) = here

        if (map[y][x] == 'x') :
            return True
        else :
            map[y][x] = '.'          # 현재 위치는 방문함. '.'표시
            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)

    return False                     # 탐색 실패
```

테스트 프로그램

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



감사합니다!