03

재귀 함수와 Class의 활용

03. 재귀함수와 Class의 활용

I. 재귀함수로 2진트리 구성

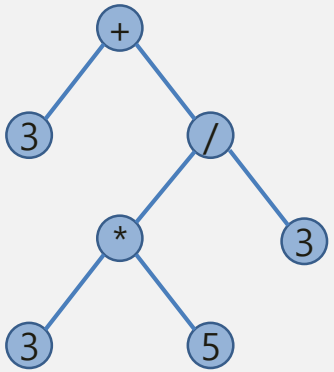
- **[실습] 재귀함수로 2진트리 구성 (0)**

Ch10-ClassTreeConstruct.py

◆ 다항 연산식을 입력받아 2진트리로 구성하는 construct_binary_tree() 함수이다.
- find_priority_operator()는 ()에 의한 우선순위가 높은 연산자를 식별

```
def post_order_traverse(node):
    if node is not None:
        post_order_traverse(node.left)  # 왼쪽 서브트리 중위 순회
        post_order_traverse(node.right)  # 오른쪽 서브트리 중위 순회
        print(node.value, end=' ')  # 현재 노드 방문

## Main 부분
expression = "3+((3*5)/3)"
expression = expression.replace(' ','')
root = construct_binary_tree(expression)
print("₩n<Post-Order>", end=' ')
post_order_traverse(root) #left > right > root 순회
```




```
[Post-Order traverse] 3 3 5 * 3 / +
```

03. 재귀함수와 Class의 활용

II. 2진트리 다항식을 스택을 이용한 2항 연산 반복 처리

- **[실습] 다항식 연산을 2항 연산의 반복으로 해결 (1)** Ch10-ClassTreeOperation.py

◆ 다항 연산식을 2진트리로 구성한 후에 sub-tree에 대한 2항 연산을 반복하여 해결
- sub-tree에 대한 2항 연산은 피연산자 2개를 Push 한 후에 Pop 하여 연산하고 그 결과를 Push하여 다음 연산에 활용 과정을 반복적으로 재귀함수 처리

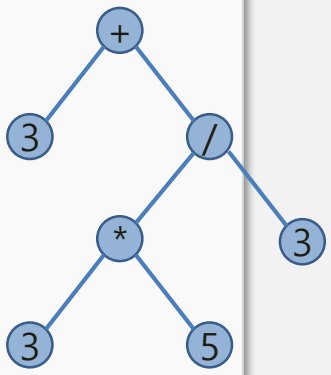
```
## Main 부분
expression = "3+((3*5)/3)"
expression = expression.replace(' ','')
root = construct_binary_tree(expression)
print("₩n[Post-Order traverse]", end=' ')
post_order_traverse(root) #left > right > root 순회
print('₩n')

result_stack = evaluate_binary_tree(root)
result = result_stack[0]
print("₩n[Last result]", result)
```

```
[Post-Order traverse] 3 3 5 * 3 / +

<<Push: 3
<<Push: 3
<<Push: 5
>>Pop : 3
>>Pop : 5
[Operation]) *
<<Push(result): 15
<<Push: 3
>>Pop : 15
>>Pop : 3
[Operation]) /
<<Push(result): 5.0
>>Pop : 3
>>Pop : 5.0
[Operation]) +
<<Push(result): 8.0

[Last result] 8.0
```



03. 재귀함수와 Class의 활용

I. 재귀함수로 2진트리 구성

- **[실습] 재귀함수로 2진트리 구성 (2)**

Ch10-ClassTreeConstruct.py

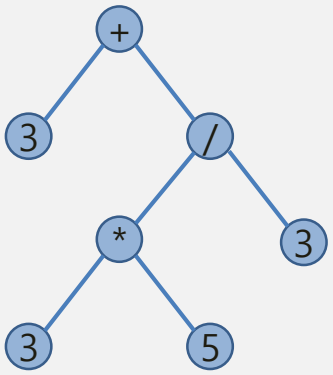
◆ 다항 연산식을 입력받아 2진트리로 구성하는 construct_binary_tree() 함수이다.
- find_priority_operator()는 ()에 의한 우선순위가 높은 연산자를 식별

```
def construct_binary_tree(expression):
    global operators
    if expression[0] == '(':  #끝의 (나 )를 제거
        expression = expression[1:-1]
    if len(expression) == 0:
        return None

    #우선순위가 가장 빠른 연산자의 index를 찾음
    operator_index = find_priority_operator(expression)
    if operator_index == -1: # 연산자가 없으면 피연산자로 간주
        node = Node(expression)
    else: # 연산자를 기준으로 왼쪽과 오른쪽의 표현식을 나눕니다.
        operator = expression[operator_index]
        left_expression = expression[:operator_index]
        right_expression = expression[operator_index+1:]

        # 노드를 생성하고 재귀적으로 왼쪽과 오른쪽 서브트리를 구성
        node = Node(operator)
        node.left = construct_binary_tree(left_expression)
        node.right = construct_binary_tree(right_expression)

    return node
```



expression = "3+((3*5)/3)"

03. 재귀함수와 Class의 활용

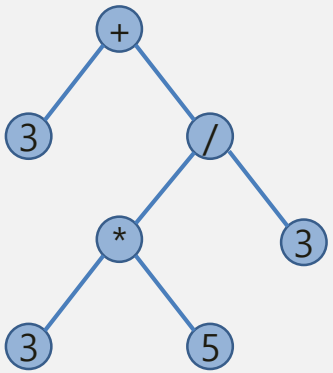
II. 2진트리 다항식을 스택을 이용한 2항 연산 반복 처리

- **[실습] 다항식 연산을 2항 연산의 반복으로 해결 (3)** Ch10-ClassTreeOperation.py

◆ 다항 연산식을 2진트리로 구성한 후에 sub-tree에 대한 2항 연산을 반복하여 해결

- sub-tree에 대한 2항 연산은 피연산자 2개를 Push 한 후에 Pop 하여 연산하고 그 결과를 Push하여 다음 연산에 활용 과정을 반복적으로 재귀함수 처리

```
def evaluate_binary_tree(root):
    stack = []
    if root is not None:  #Post Order Traverse로 운행
        stack.extend(evaluate_binary_tree(root.left))     #left traverse
        stack.extend(evaluate_binary_tree(root.right))  #right traverse
        if root.value.isdigit(): #숫자이면 Push              #root traverse
            stack.append(int(root.value))
            #print("<<Push:", root.value)
        else:   #연산자 만나면, Pop, Pop 하여 연산 후 결과를 Push
            operand2 = stack.pop()
            operand1 = stack.pop()
            #print(">>Pop :", operand1)
            #print(">>Pop :", operand2)
            result = perform_operation(root.value, operand1, operand2)
            print("[Operation])", root.value)
            stack.append(result)  #연산 결과 Push
            #print("<<Push(result):", result)
    return stack
```



03. 재귀함수와 Class의 활용

I. 재귀함수로 2진트리 구성

- **[실습] 재귀함수로 2진트리 구성 (4)**

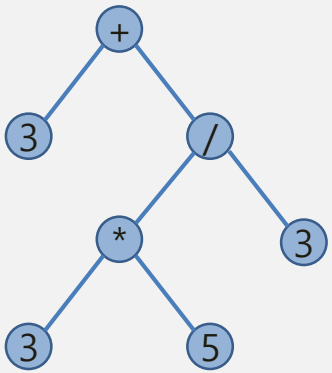
Ch10-ClassTreeConstruct.py

◆ 다항 연산식을 입력받아 2진트리로 구성하는 construct_binary_tree() 함수이다.
- find_priority_operator()는 ()에 의한 우선순위가 높은 연산자를 식별

```
operators = {'+': 1, '-': 1, '*': 2, '/': 2}  #연산자 우선순위 고려

def find_priority_operator(expression):
    global operators
    #우선순위가 가장 빠른 연산자의 index를 찾음
    operator_index = -1    #선택된 연산자 index
    max_precedence = -1    #가장 빠른 연산 우선순위 index
    parentheses_count = 0  #()의 깊이

    for i in range(len(expression) - 1, -1, -1):
        char = expression[i]
        if char == ')':
            parentheses_count += 1
        elif char == '(':
            parentheses_count -= 1
        elif char in operators and parentheses_count == 0:  #operater이고 ()로 완성되면
            if operators[char] >= max_precedence:
                operator_index = i
                max_precedence = operators[char]
                return i
    return -1
```



expression = "3+((3*5)/3)"

03. 재귀함수와 Class의 활용

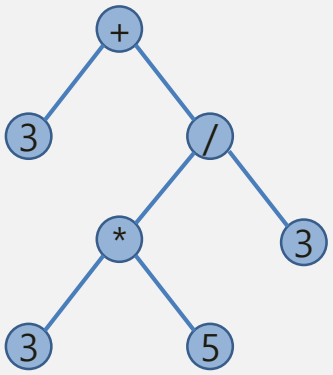
II. 2진트리 다항식을 스택을 이용한 2항 연산 반복 처리

- **[실습] 다항식 연산을 2항 연산의 반복으로 해결 (5)** Ch10-ClassTreeOperation.py

◆ 다항 연산식을 2진트리로 구성한 후에 sub-tree에 대한 2항 연산을 반복하여 해결

- sub-tree에 대한 2항 연산은 피연산자 2개를 Push 한 후에 Pop 하여 연산하고 그 결과를 Push하여 다음 연산에 활용 과정을 반복적으로 재귀함수 처리

```
def perform_operation(operator, operand1, operand2):  #2항 연산
    if operator == '+':
        return operand1 + operand2
    elif operator == '-':
        return operand1 - operand2
    elif operator == '*':
        return operand1 * operand2
    elif operator == '/':
        return operand1 / operand2
```



Thank You !