연습문제 14장

1

In [1]:
```
import tokenize as tk
import StringIO

f = StringIO.StringIO('3 + 5 + (6 + 7)')

for ele in tk.generate_tokens(f.readline):
    print ele
```

```
(2, '3', (1, 0), (1, 1), '3 + 5 + (6 + 7)')
(51, '+', (1, 2), (1, 3), '3 + 5 + (6 + 7)')
(2, '5', (1, 4), (1, 5), '3 + 5 + (6 + 7)')
(51, '+', (1, 6), (1, 7), '3 + 5 + (6 + 7)')
(51, '(', (1, 8), (1, 9), '3 + 5 + (6 + 7)')
(2, '6', (1, 9), (1, 10), '3 + 5 + (6 + 7)')
(51, '+', (1, 11), (1, 12), '3 + 5 + (6 + 7)')
(2, '7', (1, 13), (1, 14), '3 + 5 + (6 + 7)')
(51, ')', (1, 14), (1, 15), '3 + 5 + (6 + 7)')
(0, '', (2, 0), (2, 0), '')
```

각 라인의 출력 형식은 다음과 같다.

- 토큰 타입 (2, 51, 0등)
- 토큰 문자열
- 토큰 문자열의 시작 위치 (행, 열)
- 토큰 문자열의 끝 위치 (행, 열)
- 토큰이 사용된 라인

각 토큰 타입의 의미는 다음과 같다.

In [2]:
```
tk.NUMBER
```

Out[2]: 2

In [3]:
```
tk.OP
```

Out[3]: 51

In [4]:
```
tk.ENDMARKER
```

Out[4]: 0

위의 처리된 결과중 괄호 연산자를 살펴보면 일반 연산자(52, OP)로 분류되어 있다. 이 것을 좀 더 세분화 하기 위하여 다음과 같은 우리의 tokenizer() 생성자를 정의하고 테스트 해본다. '('는 LPAR, ')'는 RPAR로 구분한다. 또한 연산자의 우선 순위 정보를 입력한다.

In [5]:
```
from collections import namedtuple

Token = namedtuple('Token', ('type', 'string', 'precedence'))

def tokenizer(f):
    for ele in tk.generate_tokens(f):
        ttype = ele[0]
        string = ele[1]
        if (ttype, string) == (tk.OP, '('):
            ttype = tk.LPAR
            prec = 10
        elif (type, string) == (tk.OP, ')'):
            ttype = tk.RPAR
            prec = 10
        elif string in ('*', '/'):
            prec = 2
        elif string in ('+', '-'):
            prec = 3
        else:
            prec = 0
        yield Token(ttype, string, prec)
```

tokenizer() 발생자를 간단히 테스트 해보자. 토큰이 잘 구해졌는지 살펴보자.

In [6]:
```
# tokenizer test

f = StringIO.StringIO('3 + 5 + (6 + 7)')
list(tokenizer(f.readline))
```

Out[6]:
```
[Token(type=2, string='3', precedence=0),
 Token(type=51, string='+', precedence=3),
 Token(type=2, string='5', precedence=0),
 Token(type=51, string='+', precedence=3),
 Token(type=7, string='(', precedence=10),
 Token(type=2, string='6', precedence=0),
 Token(type=51, string='+', precedence=3),
 Token(type=2, string='7', precedence=0),
 Token(type=51, string=')', precedence=0),
 Token(type=0, string='', precedence=0)]
```

이렇게 구해진 토큰을 이용하여 인픽스를 포스트픽스로 변환하는 발생자를 다음과 같이 작성한다. 알고리즘은 다음과 같다.

1. stack을 이용한다
2. 피연산자(operand)는 그냥 출력한다
3. 연산자(operator)는
   A. 스택의 연산자가 더 낮은 우선순위일 때까지 연산자를 꺼내어 출력한다.
   B. 그리고 나서 현재의 연산자를 스택에 넣는다
4.)를 만나면 (를 만날 때까지 출력한다.
5. (는 스택에서 가장 낮은 우선 순위를 갖는다.
6. 입력의 마지막을 만나면 스택에서 모든 것을 꺼내온다.

In [7]:
```
def infix2postfix(tokens):
    stack = []
    for tkn in tokens:
        if tkn.type in (tk.NUMBER, tk.NAME):
            yield tkn
        elif tkn.type == tk.OP:
            while stack and stack[-1].precedence <= tkn.precedenc
e:
                yield stack.pop()
            stack.append(tkn)
        elif tkn.type == tk.LPAR:
            stack.append(tkn)
        elif tkn.type == tk.RPAR:
            while stack[-1].type != tk.LPAR:
                yield stack.pop()
            stack.pop()
        elif tkn.type in (tk.NL, tk.ENDMARKER):
            while stack:
                yield stack.pop()
```

In [8]:
```
f = StringIO.StringIO('4 + 5')
post = infix2postfix( tokenizer(f.readline) )
' '.join((e.string for e in post))
```

Out[8]: '4 5 +'

In [9]:
```
f = StringIO.StringIO('3+4*5/6')
post = infix2postfix( tokenizer(f.readline) )
' '.join((e.string for e in post))
```

Out[9]: '3 4 5 * 6 / +'

종합

In [10]:
```
import tokenize as tk
import StringIO
from collections import namedtuple

Token = namedtuple('Token', ('type', 'string', 'precedence'))

def tokenizer(f):
    for ele in tk.generate_tokens(f):
        ttype = ele[0]
        string = ele[1]
        if (ttype, string) == (tk.OP, '('):
            ttype = tk.LPAR
            prec = 10
        elif (type, string) == (tk.OP, ')'):
            ttype = tk.RPAR
            prec = 10
        elif string in ('*', '/'):
            prec = 2
        elif string in ('+', '-'):
            prec = 3
        else:
            prec = 0
        yield Token(ttype, string, prec)

def infix2postfix(tokens):
    stack = []
    for tkn in tokens:
        if tkn.type in (tk.NUMBER, tk.NAME):
            yield tkn
        elif tkn.type == tk.OP:
            while stack and stack[-1].precedence <= tkn.precedenc
e:
                yield stack.pop()
            stack.append(tkn)
        elif tkn.type == tk.LPAR:
            stack.append(tkn)
        elif tkn.type == tk.RPAR:
            while stack[-1].type != tk.LPAR:
                yield stack.pop()
            stack.pop()
        elif tkn.type in (tk.NL, tk.ENDMARKER):
            while stack:
                yield stack.pop()
```

In [11]:
```
f = StringIO.StringIO('3+4*5/6')
post = infix2postfix( tokenizer(f.readline) )
' '.join((e.string for e in post))
```

Out[11]: '3 4 5 * 6 / +'

In [12]:
```
```