

## PL Assignment #5: Cute17 Parser

과제물 부과일 : 2017-04-06(목)

Program Upload 마감일 : 2017-04-12(수) 23:59:59

### 문제

Cute17 문법에 따라 작성된 program이 문자열에 저장되어 있다. 이를 input으로 하여, 프로그램의 syntax tree를 구성 하시오. (이 과정을 parsing이라고 한다.) 그리고 syntax tree를 root로부터 pre-order traverse하여 원래 입력된 프로그램과 구조가 동일한 프로그램을 출력해야 한다. (이 과정을 unparsing이라고 한다.)

예를 들어, Cute17으로 작성된 program이 아래와 같을 경우

(+ (- 3 2) -378)

이와 같은 프로그램의 출력 결과는 모두 다음과 같다.

([PLUS] ([MINUS] [INT:3] [INT:2] ) [INT:-378] )

이번 과제에서는 ‘기호나 QUOTE function이 없다고 가정한다.

### Cute17의 문법

List → '(' ItemList ')'

ItemList → Item ItemList | ε

Item → id // id에는 define, cond, lambda, ...등의 키워드는 제외됨  
| int //integer const  
| "#T" | "#F"  
| '+' | '-' | '\*' | '/' | '<' | '>' | '='  
| "define" | "cond" | "not" | "lambda" | "car" | "cdr" | "cons"  
| "eq?" | "atom?" | "null?"  
| List

추가 설명

- 문법에서 대문자로 시작하는 이름은 non-terminal이고, 소문자로 시작하는 이름인 id와 int는 terminal이다.
- Terminal 중에서 id는 모든 identifier를 총칭하고, int는 모든 정수형 상수를 총칭한다. 따라서 id와 int는 token이라고 볼 수 있으며, token으로 처리하기 위해서 token 이름을 아래와 같이 명명할 수 있다.

id                      TokenType.ID  
int                     TokenType.INT

- ```
// id와 int에는 여러 가지가 있을 수 있으므로,
// lexeme으로 구별해 주어야 한다.
```
- 특별한 의미를 가지는 keyword와 해당 token 이름은 다음과 같다. (keyword가 아니면 ID로 간주)
 

|          |                  |
|----------|------------------|
| "define" | TokenType.DEFINE |
| "lambda" | TokenType.LAMBDA |
| "cond"   | TokenType.COND   |
| "quote"  | TokenType.QUOTE  |
| "not"    | TokenType.NOT    |
| "cdr"    | TokenType.CDR    |
| "car"    | TokenType.CAR    |
| "cons"   | TokenType.CONS   |
| "eq?"    | TokenType.EQ_Q   |
| "null?"  | TokenType.NULL_Q |
| "atom?"  | TokenType.ATOM_Q |
  - Boolean 상수는 terminal이며, 해당 token은 다음과 같다.
 

|    |                 |
|----|-----------------|
| #T | TokenType.TRUE  |
| #F | TokenType.FALSE |
  - 특수 문자들은 terminal이며, 다음과 같이 token 이름을 부여할 수 있다.
 

|   |                      |
|---|----------------------|
| ( | TokenType.L_PAREN    |
| ) | TokenType.R_PAREN    |
| + | TokenType.PLUS       |
| - | TokenType.MINUS      |
| * | TokenType.TIMES      |
| / | TokenType.DIV        |
| < | TokenType.LT         |
| = | TokenType.EQ         |
| > | TokenType.GT         |
| ' | TokenType.APOSTROPHE |

## Cute17의 특징

괄호를 써서 프로그램이 표현되는 Cute17는 list가 기본 표현이다. 또한 아래와 같이 각 list의 맨 첫번째 원소를 연산자나 함수 호출로 보고 리스트의 나머지를 피연산자로 간주한 후 evaluate 하게 된다. (다음 예는 > 를 prompt 로 사용하고 있는 인터프리터를 보여준다.)

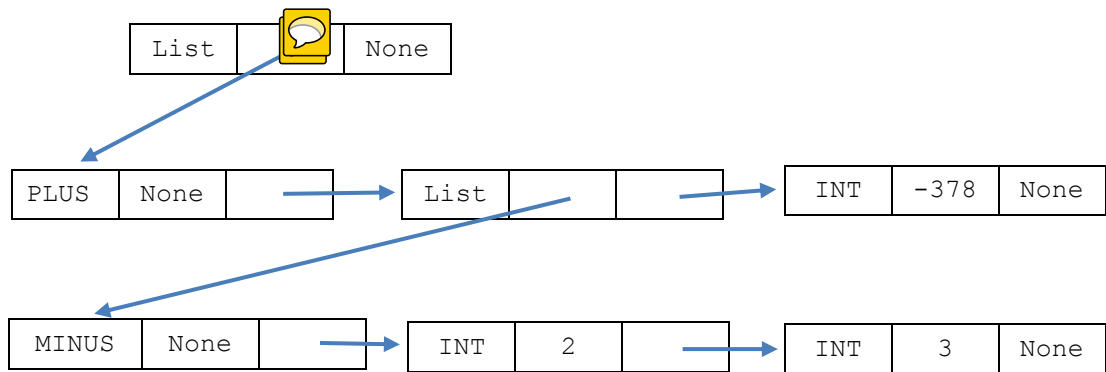
```
> (+ 2 3)
5
> (* (+ 3 3) 2)
12
```

먼저 원소가 되는 node의 구조는 다음과 같이 type, value, next로 구성된다. Type은 원소의 type을 나타내고, value는 원소가 가지는 값을 의미한다. 원소의 type에 따라서 value는 가지지 않을 수도 있다. Next는 원소를 구성하는 node의 다음에 연결되어 있는 node이다.

|      |       |      |
|------|-------|------|
| Type | Value | Next |
|------|-------|------|

다음과 같은 프로그램이 있다고 가정하면, parse tree 는 다음과 같이 된다.

(+ (- 2 3) -378)



그리고 프로그램의 출력결과는 다음과 같다.

(+ (- 3 2) -378)

([PLUS] ( [MINUS] [INT:3] [INT:2] ) [INT:-378]))

---

## Programming

### 1. 노드의 자료구조

노드는 Python 의 특징을 이용한다. Python 은 타입이 실행중에 정의된다. 노드의 value 의 타입은 type 변수가 가지는 값에 따라 타입이 정해진다. 주의할 점은 노드가 List 타입 일 경우에는 노드를 value 로 가진다.

```
class Node (object):

    def __init__(self, type, value=None):
        self.next = None
        self.value = value
        self.type = type

    def set_last_next(self, next_node):
        if self.next is not None:
            self.next.set_last_next(next_node)

        else:
            self.next = next_node

    def __str__(self):
        result = ''

        if self.type is TokenType.ID:
            result = '['+self.value+']'
        elif self.type is TokenType.INT:
            result = self.value
        elif self.type is TokenType.LIST:
            result = '('+str(self.value)+')'
        else:
            result = '['+NODETYPE_NAMES[self.type]+']'

        # fill out
        # next 노드에 대해서도 출력하도록 작성
        # recursion 이용
```

## 2. 프로그램을 수행하는 프로그램 예시

```
class BasicPaser(object):
```

```
    def __init__(self, token_list):
```

```
        """
```

```
        :type token_list:list
```

```
        :param token_list:
```

```
        :return:
```

```
        """
```

```
        self.token_iter = iter(token_list)
```

```
    def _get_next_token(self):
```

```
        """
```

```
        :rtype: Token
```

```
        :return:
```

```
        """
```

```
        next_token = next(self.token_iter, None)
```

```
        if next_token is None:
```

```
            return None
```

```
        return next_token
```

```
    def parse_expr(self):
```

```
        """
```

```
        :rtype : Node
```

```
        :return:
```

```
        """
```

```
        token = self._get_next_token()
```

```
        # 하나의 Token을 create_node()를 이용하여 Node로 만들어 반환함
```

```
        '""':type :Token"""
```

```
        if token is None:
```

```
            return None
```

```
        result = self._create_node(token)
```

```
        return result
```

```
    def _create_node(self, token):
```

```
        # 토큰을 Node로 만듦
```

```
        if token is None:
```

```
            return None
```

```
        elif token.type is CuteType.INT:
```

```
            return Node(TokenType.INT, token.lexeme)
```

```
        elif token.type is CuteType.ID:
```

```
            return Node(TokenType.ID, token.lexeme)
```

```
        elif token.type is CuteType.L_PAREN:
```

```
            return Node(TokenType.LIST, self._parse_expr_list())
```

```
        elif token.type is CuteType.R_PAREN:
```

```
            return None
```

```
        elif # 조건 작성, INT, ID, L_PAREN, R_PAREN을 제외한 나머지 경우:
```

```
            # "#T", "#F", '+', '-', '*', '/', '<', '>', '=', "define", "cond", "not", "car",
```

```
            # "cdr", "cons", "eq?", "atom?", "null?", 에 대해서 작성
```

```
            # 조건에 맞는 노드를 반환하도록 작성
```

```
            # 기호 '와 quote에 대해서도 작성
```

```
        else:
```

```
            return None
```

```
    def _parse_expr_list(self):
```

```

# Token이 '('일 경우, list 형태로 만들어서 반환
head = self.parse_expr()
''' :type :Node'''
if head is not None:
    head.next = self._parse_expr_list()
return head

```

### 3. 테스트

```

def Test_BasicPaser():
    test_cute = CuteScanner('( + ( - 3 String 2 ) ( ) -378 )')
    test_tokens = test_cute.tokenize()
    print test_tokens
    test_basic_paser = BasicPaser(test_tokens)
    node = test_basic_paser.parse_expr()
    print node

```

Test\_BasicPaser()

## 유의사항

- 주어진 코드를 수정해서는 안됨
- 변수와 메소드를 추가해서는 안됨

## 실행결과

```

Run test
C:\Python27\python.exe D:/PyCharm/workstation/test.py
=== tokenize ===
( + ( - 3 String 2 ) ( ) -378 )
scanning...
[L_PAREN: (]
[PLUS: +]
[L_PAREN: (]
[MINUS: -]
[INT: 3]
[ID: String]
[INT: 2]
[R_PAREN: )]
[L_PAREN: (]
[R_PAREN: )]
[INT: -378]
[R_PAREN: )]
[[L_PAREN: (], [PLUS: +], [L_PAREN: (], [MINUS: -], [INT: 3], [ID: String], [INT: 2], [R_PAREN: )], [L_PAREN: (], [R_PAREN: )], [INT: -378], [R_PAREN: )]]
([PLUS] ([MINUS] [INT:3] [ID:String] [INT:2]) (None) [INT:-378])

Process finished with exit code 0

```