

## PL Assignment #5: Cute15 Parser

과제물 부과일 : 2015-04-06(월)

Program Upload 마감일 : 2015-04-12(일) 23:59:59

### 문제

Cute15 문법에 따라 작성된 program이 as05.txt에 저장되어 있다. 이를 input file로 하여, 프로그램의 syntax tree를 구성하시오.(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, Cute15으로 작성된 program이 아래와 같을 경우

```
(+ (- 3 2) -378)
```

이와 같은 프로그램의 출력결과는 다음과 같다.

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

이번 과제에서는 ‘나 QUOTE 기호가 없다고 가정한다.

### Cute15의 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 |

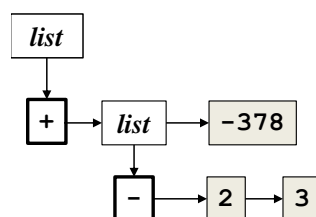
## Cute15의 특징

괄호를 써서 프로그램이 표현되는 Cute15는 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
```

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

```
(+ (- 2 3) -378)
```



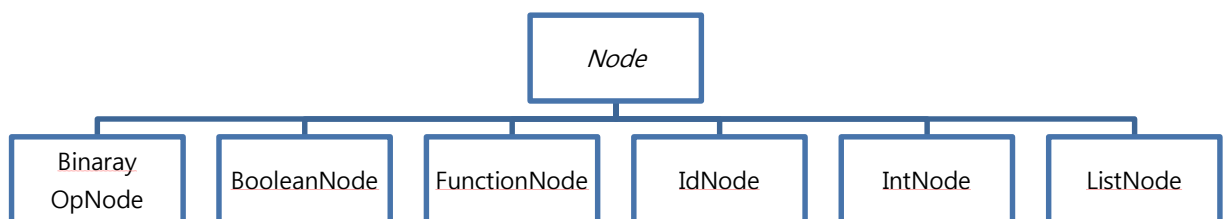
그리고 프로그램의 출력결과는 다음과 같다.

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

## Programming

앞서 과제에서 정의한 getNextToken() 함수를 이용하며, 그 외에도 아래와 같은 자료구조를 사용한다.

### 1. 노드의 자료구조



```
public abstract class Node {
    Node next;
    public Node() { this.next = null; }
    public void setNext(Node next){ this.next = next;}
    public void setLastNext(Node next){
        if(this.next != null) this.next.setLastNext(next);
        else this.next = next;
    }
    public Node getNext(){ return next; }
}
public class ListNode extends Node{
    public Node value;
}
public class IntNode extends Node {
    public int value;

    @Override
    public String toString(){
        return "INT: " + Integer.toString(value);
    }
}
public class IdNode extends Node{
    public String value;
    @Override
    public String toString(){
        return "ID: " + value;
    }
}
public class FunctionNode extends Node{
    public enum FunctionType { DEFINE, LAMBDA, COND, NOT, CDR, CAR, CONS, EQ_Q,
    NULL_Q, ATOM_Q }
    public FunctionType value;
```

```

@Override
public String toString(){
    return value.name();
}
}
public class BooleanNode extends Node{
    public boolean value;
    @Override
    public String toString(){
        return value ? "#T" : "#F";
    }
}
public class BinaryOpNode extends Node{
    public enum BinType { MINUS, PLUS, TIMES, DIV, LT, GT, EQ }
    public BinType value;
    @Override
    public String toString(){
        return value.name();
    }
}
}

```

## 2. 프로그램을 수행하는 프로그램 예시

```

public class BasicParser {
    private ListIterator<Token> iter;

    public BasicParser(List<Token> tokenList) {
        iter = tokenList.listIterator();
    }
    private void errorLog(String err) {
        System.out.println(err);
    }
    private Token getNextToken() {
        if (!iter.hasNext()) return null;
        return iter.next();
    }
    public Node parseExpr() {
        Token t = getNextToken();
        if (t == null) {
            System.out.println("No more token");
            return null;
        }
        switch (t.type) {
            case ID:
                IdNode idNode = new IdNode();
                idNode.value = t.lexeme;
                return idNode;
            case INT:
                IntNode intNode = new IntNode();
                if(t.lexeme==null) System.out.println("???");
                intNode.value = new Integer(t.lexeme);
                return intNode;
                // BinaryOpNode에 대하여 작성
                // +, -, /, *가 해당
            case ATOM_Q:
                FunctionNode atom = new FunctionNode();
                atom.value = FunctionType.ATOM_Q;
                return atom;
                // FunctionNode에 대하여 작성
                // 키워드가 FunctionNode에 해당
        }
    }
}

```

```

    case FALSE:
        BooleanNode falseNode = new BooleanNode();
        falseNode.value = false;
        return falseNode;
        // BooleanNode에 대하여 작성
        // L_PAREN일 경우 parseExprList()를 호출하여 처리
    case L_PAREN:
        ListNode listNode = new ListNode();
        listNode.value = parseExprList();
        return listNode;
    case R_PAREN:
        return null;
    }
    System.out.println("Parsing Error!");
    return null;
}

// List의 value를 생성하는 메소드
private Node parseExprList() {
    Node head = parseExpr();
    //head의 next 노드를 set하십시오.
    return head;
}

}

```

### 3. Per-order traverse print class

```
public class Printer {  
  
    PrintStream ps;  
  
    public Printer(PrintStream ps) {  
        this.ps = ps;  
    }  
  
    //Make your print class  
    //ps.print(...)  
    //node.getNext()  
    //node.toString()  
  
    //recursive call or iterate  
    //void printNode  
    //void printList  
    //... etc  
}
```

### 4. 테스트

```
public static void main(String[] args){  
    read file...  
  
    List<Token> tokens = ...  
    BasicParser p = new BasicParser(tokens);  
    Node node = p.parseExpr();  
  
    Printer pt = new Printer(System.out);  
    ...  
}
```

## 유의사항

- 주어진 코드를 수정해서는 안됨
- 변수와 메소드를 추가해서는 안됨