PL Assignment #4: Cute14 Parser

과제물 부과일 : 2014-04-08(화)
Program Upload 마감일 : 2014-04-14(월) 23:59:59

문제

Cute14 문법에 따라 작성된 program이 as04.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, Cute14으로 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.

Cute14의 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

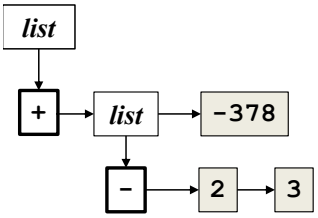
Cute14의 특징

괄호를 써서 프로그램이 표현되는 Cute14은 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)
```



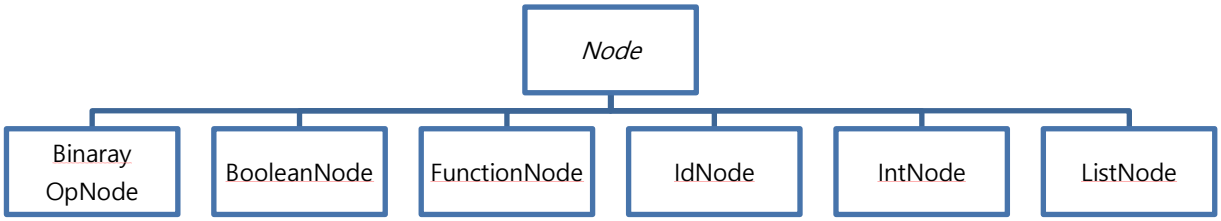
그리고 프로그램의 출력결과는 다음과 같다.

```
(+ (- 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 {
	public enum Type {QUOTED, NOT_QUOTED}
	public final Type type;
	Node next;

	public Node(Type type) {
		this.type = type;
		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 BinarayOpNode extends Node{
	public enum BinType { MINUS, PLUS, TIMES, DIV, LT, GT, EQ }
	public final BinType value;

	public BinarayOpNode(Type type, BinType value) {
```

```
                super(type);
                this.value = value;
        }

        @Override
        public String toString(){
                return value.name();
        }
}

public class BooleanNode extends Node{
        public final boolean value;
        public BooleanNode(Type type, boolean value) {
                super(type);
                this.value = value;
        }

        @Override
        public String toString(){
                return Boolean.toString(value);
        }
}

public class FunctionNode extends Node{
        public enum FunctionType { DEFINE, LAMBDA, COND, NOT, CDR, CAR, CONS, EQ_Q,
NULL_Q, ATOM_Q }
        public final FunctionType value;
        public FunctionNode(Type type, FunctionType value) {
                super(type);
                this.value = value;
        }

        @Override
        public String toString(){
                return value.name();
        }
}

public class IdNode extends Node{
        public final String value;
        public IdNode(Type type, String value){
                super(type);
                this.value = value;
        }

        @Override
        public String toString(){
                return "ID: " + value;
        }
}

public class IntNode extends Node {
        public final int value;
        public IntNode(Type type,int value) {
                super(type);
```

```
                this.value = value;
        }

        @Override
        public String toString(){
                return "INT: " + Integer.toString(value);
        }
}

public class ListNode extends Node{
        public final Node value;

        public ListNode(Type type, Node value) {
                super(type);
                this.value = value;
        }
}
```

2. 프로그램을 parsing하는 클래스 구현 예시

```
public class BasicParser {

	boolean alreadyParse;
	int pos;
	List<Token> tokenList;

	public BasicParser(List<Token> list) {
		this.alreadyParse = false;
		this.pos = 0;
		this.tokenList = list;
	}

	private Token getNextToken() {
		if (pos < tokenList.size())
			return tokenList.get(pos++);
		else
			return null;
	}

	private void ungetToken() {
		if (pos > 0)
			pos--;
	}

	private void errorLog(String err) {
		System.out.println(err);
	}

	// list의 한 원소만을 파싱함
	private Node parseItem() {
		Node item = null;
		Token t = getNextToken();

		if (t != null) {
			switch (t.type) {
			case ID:
				item = new IdNode(Type.NOT_QUOTED, t.lexme);
				break;
			case INT:
				item = new IntNode(Type.NOT_QUOTED, new
Integer(t.lexme));
				break;
			case MINUS:
				item = new BinarayOpNode(Type.NOT_QUOTED,
BinType.MINUS);
				break;
			//이번 과제에서는 '나 QUOTE 기호가 없다고 가정하기 때문에 모든
노드가 Type.NOT_QUOTED이다
			//...
			//...
			//...

			case L_PAREN: // 중첩 list임
				ungetToken();
```

```
                        item = parseList();
                        break;


                default:
                        errorLog("parseItem Error");
                        break;
                }
        }
        return item;
}

// list 의 괄호를 뺀 내부를 parsing 함
private Node parseItemList() {
        Node list = null;// 여기에 list를 구성해서 결과를 리턴 예정
        Token t = getNextToken();

        if (t != null) {
                switch (t.type) {
                case ID:
                case INT:
                //...
                //...

                case L_PAREN:
                        // 즉, 오른쪽 괄호가 아닌 모든 token에 대해
                        ungetToken();
                        list = parseItem(); // 한 원소를 parsing하여 head 를
만듬

                        break;
                default: // 오른쪽 괄호거나 오류. List 끝. 다시 넣어두고 리턴함.
                        ungetToken();
                        break;
                }
        }

        if (list != null)
                list.setNext(parseItemList()); // recursion을 써서 tail을
          만들어 head의 next에 이어 붙임

        return list;
}

public Node parseList() {
        Node list = null; // 이곳에 리스트를 구성하여 리턴 예정
        Token t = getNextToken();

        if (t != null && t.type == TokenType.L_PAREN) {
                Node itemList = parseItemList();

                t = getNextToken();
                if (t == null || t.type != TokenType.R_PAREN) // 오른쪽
괄호가 아니면 에러
                        errorLog("parseList Error rparen");
                else
```

```
                        list = new ListNode(Type.NOT_QUOTED, itemList);

                } else {
                        errorLog("parseList Error lparen");// 왼쪽 괄호가 아니면 에러
                }

                return list;
        }
 }
```

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.parseList();

                Printer pt = new Printer(System.out);
                …

}
```