PL Assignment #7: Cute18 Built-in Function 구현

과제물 부과일 : 2018-05-10 (목)
Program Upload 마감일 : 2018-05-22 (화) 23:59:59

문제

Cute18 문법에 따라 작성된 program이 as08.txt에 저장되어 있다. 이 프로그램은 Cute18의 built-in function을 사용하여 리스트를 조작하며 그 결과를 구하고 있다. 이러한 프로그램을 input file로 하여, 프로그램의 syntax tree를 출력하는 프로그램을 작성하시오.

Cute18의 built-in function

이번 과제에서 구현 해야하는 built-in function들은 다음과 같다.

- 리스트 연산

car | cdr | cons

1. car

List의 맨 처음 원소를 리턴한다.

```
> ( car ‘ ( 2 3 4 ) )
2
> ( car ‘ ( ( 2 3 ) ( 4 5 ) 6 ) )
(2 3)
```

주의: car는 list가 아닌 데이터나 ()이 인자로 주어지는 경우 error 를 내게 된다. 그러나, 본 과제에서는 이러한 error 발생 경우에 대해서는 고려하지 않고, 모든 입력이 올바르게 되어 주어진다고 가정한다. (이것은 car 외의 다른 함수에도 동일하게 적용한다.)

2. cdr

list의 맨 처음 원소를 제외한 나머지 list를 리턴한다. list가 아닌 데이터에 대해서는 error 를 낸다.

```
> ( cdr ‘ ( 2 3 4 ) )
‘(3 4)
> ( cdr ‘ ( ( 2 3 ) ( 4 5 ) 6 ) )
‘((4 5) 6 )
> ( cdr ‘( 2 ) )
‘( )
```

3. cons

한 개의 원소(head)와 한 개의 리스트(tail)를 붙여서 새로운 리스트를 만들어 리턴한다.

```
> ( cons 1 '( 2 3 4 ) )
' ( 1 2 3 4 )
> ( cons ' (2 3) ' ( 4 5 6 ) )
' ( ( 2 3 ) 4 5 6 )
> ( cons 2 ' ( ) )
' ( 2 )
```

4. null?

리스트가 NULL 인지 검사한다. 즉, () 인지 검사한다.

```
> ( null? ' ( ) )
#T
> ( null? ' ( 1 2 ) )
#F
> ( null? '( ( ) ) )
#F
```

5. atom?

list가 아니면 모두 atom 이다. 따라서 list인 경우는 false, list 가 아닌 경우는 true를 리턴한다.

```
> ( atom? ' a )
#T
> ( atom? ' ( 1 2 ) )
#F
```

6. eq?

두 노드의 값이 같은지 비교한 값을 반환한다.

```
> ( eq? 'a 'a )
#T
> ( eq? 'a 'b )
#F
> ( eq? ' ( a b ) '( a b ) )
#T
```

7. 기타 연산

- 산술 연산

"+" | "-" | "*" | "/"

예) (+ 1 2)

3

예) (+ (+ 1 2) 3)

6

- 관계연산

"<" | "=" | ">"

예) (> 1 5)

#F

예) (> (+ 9 3) 5)

#T

- 논리 연산

"not"

예) (not #F)

#T

예) (not (< 1 2))

#F

- 조건문

"cond"

예) (cond ((> 1 2) 0) (#T 1))

1

예) (cond (#F 3) (#T 2))

2

예) (cond (#T 2) (#T 1))

2

Programming

car, cdr, cons등의 built-in 함수와 숫자 및 다른 연산을 수행하는 함수를 작성한다.

1. Built-in 함수 구현

```
package interpreter;
import parser.ast.*;
import parser.parse.CuteParser;
import java.io.File;

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

    public Node runExpr(Node rootExpr) {
        if (rootExpr == null)
            return null;

        if (rootExpr instanceof IdNode)
            return rootExpr;
        else if (rootExpr instanceof IntNode)
            return rootExpr;
        else if (rootExpr instanceof BooleanNode)
            return rootExpr;
        else if (rootExpr instanceof ListNode)
            return runList((ListNode) rootExpr);
        else
            errorLog("run Expr error");

        return null;

    }
    private Node runList(ListNode list) {
        if(list.equals(ListNode.EMPTYLIST))
            return list;
        if(list.car() instanceof FunctionNode){
```

```
            return runFunction((FunctionNode)list.car(), list.cdr());
        }
        if(list.car() instanceof BinaryOpNode){
            return runBinary(list);
        }
        return list;
    }

    private Node runFunction(FunctionNode operator, ListNode operand) {
        switch (operator.value){
            // CAR, CDR, CONS등에 대한 동작 구현
            case CAR:
            case CDR:
            case CONS:
            case COND:
            case NOT:
            default:
                break;
        }
        return null;
    }

private Node runBinary(ListNode list) {
    BinaryNode operator = list.car();
    // 구현과정에서 필요한 변수 및 함수 작업 가능

    switch (operator.value){
            // +,-,/ 등에 대한 바이너리 연산 동작 구현
            case PLUS:
            case MINUS:
            case DIV:
            case TIMES:
            case LT:
            case GT:
            case EQ:
            default:
                break;
```

```
        }
        return null;
    }

    private Node runQuote(ListNode node) {
        return ((QuoteNode)node.car()).nodeInside();
    }

    public static void main(String[] args) {
        ClassLoader cloader = CuteInterpreter.class.getClassLoader();
        File file = new
File(cloader.getResource("interpreter/as07.txt").getFile());
        CuteParser cuteParser = new CuteParser(file);
        Node parseTree = cuteParser.parseExpr();
        CuteInterpreter i = new CuteInterpreter();
        Node resultNode = i.runExpr(parseTree);
        NodePrinter.getPrinter(System.out).prettyPrint(resultNode);}
```

2. Print 함수 수정

기존의 Type과 Value를 출력해주는 형태에서 Value값만 출력해 주는 형태로 변경. 저번 과제와 동일한 클래스 파일을 이번 과제에 맞게 출력 함수만 다시 작성

```
package parser.parse;
import java.io.PrintStream;
import parser.ast.*;
public class NodePrinter {
    private PrintStream ps;

    public static NodePrinter getPrinter(PrintStream ps) {
        return new NodePrinter(ps);
    }
    private NodePrinter(PrintStream ps) {
        this.ps = ps;
    }
    private void printNode(ListNode listNode) {
    if (listNode.equals(ListNode.EMPTYLIST)) {
            ps.print("( )");
            return;
        }
        if (listNode.equals(ListNode.ENDLIST)) {
            return;
        }
        printNode(listNode.car());
        if(listNode.cdr().equals(ListNode.EMPTYLIST)) {
            ps.print(" ");
        }
        printNode(listNode.cdr())
    }
    private void printNode(QuoteNode quoteNode) {
        if (quoteNode.nodeInside() == null)
            return;
        ps.print("'");
        printNode(quoteNode.nodeInside());
        }
    }
    private void printNode(Node node) {
```

```
    if (node == null)
                return;

        if (node instanceof ListNode) {
            ps.print("(");
            printNode((ListNode)node);
            ps.print(" )");
        } else if (node instanceof QuoteNode) {
            printNode((QuoteNode)node);
        } else {
            String temp = node.toString();
            StringTokenizer st = new StringTokenizer(temp, " ");
            st.nextToken();
            ps.print(" " + st.nextToken());
        }

    public void prettyPrint(Node node){
        printNode(node);
    }
}
```

유의사항

- 입력 명령의 실행 결과를 알고 싶을 경우 이전 과제에서 소개했던 Racket을 이용하여 확인한다.
- 출력 및 결과는 Racket과 완전히 똑같지 않다. 해당 PDF의 예제로 확인한다.
- (car (cdr (cdr (cdr '(a b c d))))과 같은 각각의 연산과정에서 4개이상의 연산이 들어가는 경우는 고려하지 않는다.
- 모든 연산 명령은 띄어쓰기가 포함되어 있다.
- Package 및 출력 띄어쓰기를 반드시 지켜야 하는 것은 아니다.
- 각각의 Node 내부에 있는 필드에 접근하기 위한 작업은 각자 진행한다.