PL Assignment #8: Cute19 Built-in Function 구현

과제물 부과일 : 2019-05-15 (수)
Program Upload 마감일 : 2019-05-29 (수) 23:59:59

문제

Cute19 문법에 따라 작성된 program이 as08.txt에 저장되어 있다. 이 프로그램은 Cute19의 built-in function을 사용하여 리스트를 조작하며 그 결과를 구하고 있다. 이러한 프로그램을 input file로 하여, 프로그램의 syntax tree를 출력하는 프로그램을 작성하시오.

Cute19의 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. **주의 : null list은 atom 으로 취급된다.**

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

6. eq?

두 노드를 비교하여 값이 같은 객체를 참조하는 지 반환한다.

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

```
> ( eq? ‘ ( a b ) ‘ ( a b ) )
#F
```

6. 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

Programming

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

1. 수정된 클래스

```
// 새로 수정된 IdNode
public class IdNode implements ValueNode {
    private String idString;
```

```
    public IdNode(String text) {
        idString = text;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof IdNode)) return false;
        IdNode idNode = (IdNode) o;
        return Objects.equals(idString, idNode.idString);
    }
    @Override
    public String toString() {
        return idString;
    }
}

// 새로 수정된 IntNode
public class IntNode implements ValueNode {
    private Integer value;
    public IntNode(String text) {
        this.value = new Integer(text);
    }
    public Integer getValue() {
        return value;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof IntNode)) return false;
        IntNode intNode = (IntNode) o;
        return Objects.equals(value, intNode.value);
    }
    @Override
    public String toString() {
        return value.toString();
    }
}

// 새로 수정된 ListNode (기존에 ENDLIST 가 하던 기능을 EMPTYLIST 가 대체)
public interface ListNode extends Node {
    ListNode EMPTYLIST = new ListNode() {

        @Override
        public Node car() {
            return null;
        }

        @Override
        public ListNode cdr() {
            return null;
        }
    };

    static ListNode cons(Node head, ListNode tail) {
```

```
        return new ListNode() {

            @Override
            public Node car() {
                return head;
            }

            @Override
            public ListNode cdr() {
                return tail;
            }

        };
    }

    Node car();
    ListNode cdr();
}
```

```
// 새로 수정된 접근자
public FunctionType funcType; -- ( class FunctionNode )
public BinType binType;  --  ( class BinaryOpNode )
```

2. Built-in 함수 구현

```
public class CuteInterpreter {
    public static void main(String[] args) {
        ClassLoader cloader = ParserMain.class.getClassLoader();
        File file = new
File(cloader.getResource("interpreter/as08.txt").getFile());
        CuteParser cuteParser = new CuteParser(file);
        CuteInterpreter interpreter = new CuteInterpreter();
        Node parseTree = cuteParser.parseExpr();
        Node resultNode = interpreter.runExpr(parseTree);
        NodePrinter nodePrinter = new NodePrinter(resultNode);
        nodePrinter.prettyPrint();
    }

    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(), (ListNode)
stripList(list.cdr()));
        }
        if (list.car() instanceof BinaryOpNode) {
            return runBinary(list);
        }
        return list;
    }

    private Node runFunction(FunctionNode operator, ListNode operand) {
        switch (operator.funcType) {
            // CAR, CDR, CONS등에 대한 동작 구현
            case CAR:
            case CDR:
            case CONS:
            …
            default:
                break;
        }
        return null;
    }

    private Node stripList(ListNode node) {
        if (node.car() instanceof ListNode && node.cdr() == ListNode.EMPTYLIST) {
            Node listNode = node.car();
            return listNode;
        } else {
            return node;
        }
    }

    private Node runBinary(ListNode list) {
        BinaryOpNode operator = (BinaryOpNode) list.car();

// 구현과정에서 필요한 변수 및 함수 작업 가능
        switch (operator.binType) {
            // +,-,/ 등에 대한 바이너리 연산 동작 구현
            case PLUS:
            case MINUS:
            case TIMES:
            case DIV:
            …
            default:
                break;
        }
        return null;
```

```
    }

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

유의사항

- 입력 명령의 실행 결과를 알고 싶을 경우 이전 과제에서 소개했던 Racket을 이용하여 확인한다.
- 출력은 Racket과 완전히 똑같지 않다. 해당 PDF의 예제로 확인한다.
- 반복문(for, while)을 사용하지 않는다.
- 잘못된 입력 및 연산은 고려하지 않는다.
- 모든 연산 명령은 띄어쓰기가 포함되어 있다.
- Package 및 출력 띄어쓰기를 반드시 지켜야 하는 것은 아니다.
- atom?을 Racket 에서 테스트하려면 (define (atom? x) (not (pair? x))) 정의 후 테스트