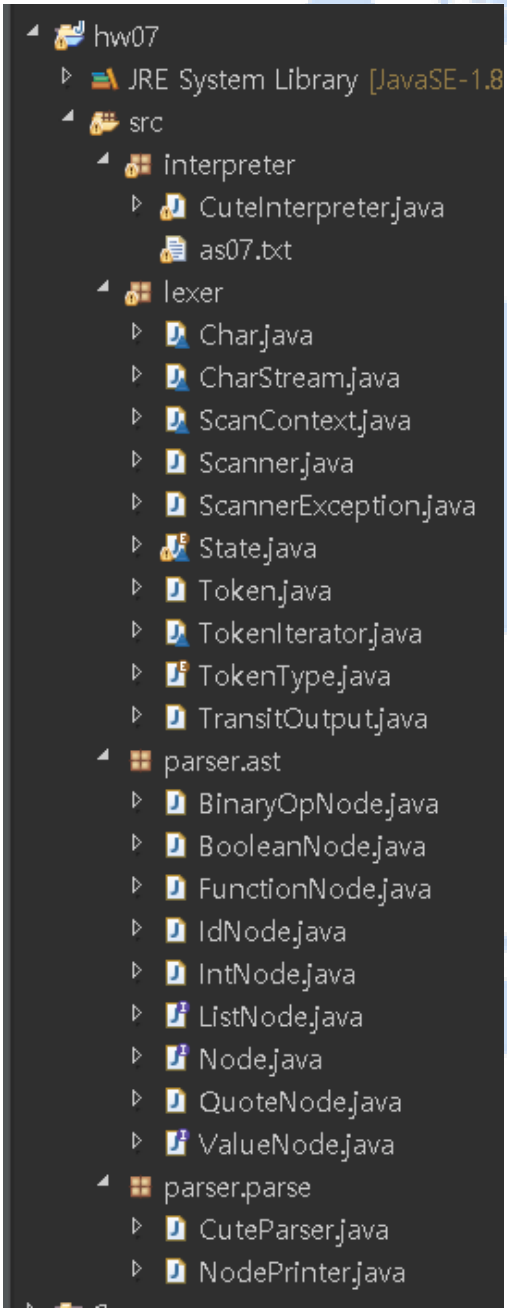
프로그래밍언어개론 보고서

컴퓨터공학과 201604140 박경수

PL Assignment #7: Cute18 Built-in Function

전체 구조



BinaryOpNode 수정

BinaryOpNode.java

```java
package parser.ast;

import java.util.HashMap;

public class BinaryOpNode implements Node{
    public enum BinType {
        MINUS { TokenType tokenType() {return TokenType.MINUS;} },
        PLUS { TokenType tokenType() {return TokenType.PLUS;} },
        TIMES { TokenType tokenType() {return TokenType.TIMES;} },
        DIV { TokenType tokenType() {return TokenType.DIV;} },
        LT { TokenType tokenType() {return TokenType.LT;} },
        GT { TokenType tokenType() {return TokenType.GT;} },
        EQ { TokenType tokenType() {return TokenType.EQ;} };
```

1. enum BinType을 public형으로 변경해준다.

FunctionNode 수정

FunctionNode.java

```java
package parser.ast;

import java.util.HashMap;

public class FunctionNode implements Node{ //binaryOpNode클래스를 보고 참고해서 작성
    public enum FunctionType {
        ATOM_Q { TokenType tokenType() {return TokenType.ATOM_Q;} },
        CAR { TokenType tokenType() {return TokenType.CAR;} },
        CDR { TokenType tokenType() {return TokenType.CDR;} },
        COND { TokenType tokenType() {return TokenType.COND;} },
        CONS { TokenType tokenType() {return TokenType.CONS;} },
        DEFINE { TokenType tokenType() {return TokenType.DEFINE;} },
        EQ_Q { TokenType tokenType() {return TokenType.EQ_Q;} },
        LAMBDA { TokenType tokenType() {return TokenType.LAMBDA;} },
        NOT { TokenType tokenType() {return TokenType.NOT;} },
        NULL_Q { TokenType tokenType() {return TokenType.NULL_Q;} },
        APOSTROPHE { TokenType tokenType() {return TokenType.APOSTROPHE;} };
```

1. enum FunctionType을 public형으로 변경해준다.

runFunction - CAR

CuteParser.java | CuteInterpreter.java

```
40
41    private Node runFunction(FunctionNode operator, ListNode operand) {
42        switch (operator.value){
43            // CAR, CDR, CONS등에 대한 동작 구현
44            case CAR:
45                if (operand.car() instanceof QuoteNode) { // car다음에 오는 노드가 ` 인지 검사
46                    ListNode result = (ListNode)runQuote(operand); // runQuote를 통해 nodeInside로 리스트 안쪽 리스트노드를 가져옴
47                    return result.car();
48                }else {
49                    errorLog("run Expr error");
50                    return null;
51                }
```

1. operator는 CAR이고 operand는 그 다음에 오는 리스트노드이다.
2. operand의 car이 QuoteNode일 경우 runQuote를 통해 리스트 안쪽 리스트노드를 가져와
3. 해당 리스트노드의 첫 번째 원소를 리턴한다.
4. QuoteNode가 아닐 경우 error를 발생시킨다.

runFunction - CDR

```
52            case CDR:
53                if (operand.car() instanceof QuoteNode) { // car다음에 오는 노드가 ` 인지 검사
54                ListNode result = (ListNode)runQuote(operand); // runQuote를 통해 nodeInside로 리스트 안쪽 리스트노드를 가져옴
55                return result.cdr();
56                }else {
57                    errorLog("run Expr error");
58                    return null;
59                }
```

1. operator는 CDR이고 operand는 그 다음에 오는 리스트노드이다.
2. operand의 car이 QuoteNode일 경우 runQuote를 통해 리스트 안쪽 리스트노드를 가져와
3. 해당 리스트노드의 첫 번째 원소를 제외한 나머지를 리턴한다.
4. QuoteNode가 아닐 경우 error를 발생시킨다.

runFunction - CONS

```
60              case CONS:
61                  if(operand.car() instanceof QuoteNode) {
62                      ListNode result = ListNode.cons((ListNode)runQuote(operand), (ListNode)runQuote(operand.cdr()))
63                      return result;
64                  }
65                  else {
66                      ListNode result = ListNode.cons(operand.car(), (ListNode)runQuote(operand.cdr()));
67                      return result;
68                  }
```

1. operator는 CONS이고 operand는 그 다음에 오는 리스트노드이다.

2. operand의 car이 QuoteNode일 경우 ListNode의 cons를 통해 head와 tail을 지정해 준 후

3. 해당 리스트노드를 리턴한다.

4. QuoteNode가 아닐 경우는 IntNode가 먼저 나온 경우 이므로, 앞 원소와 나머지의 runQuote 결과를 cons 해준다.

runFunction - null?

```
case NULL_Q:
     if(((ListNode)runQuote(operand)).car() == null) { // 리스트가 null일 경우
        BooleanNode result =  BooleanNode.TRUE_NODE;
        return result;
     }else{
        BooleanNode result =  BooleanNode.FALSE_NODE;
        return result;
     }
```

1. operator는 NULL_Q고 operand는 그 다음에 오는 리스트노드이다.

2. operand의 runQuote 결과 리스트의 car이 null일 경우 BooleanNode의 TRUE_NODE를 리턴해준다.

3. null이 아닐 경우 BooleanNode의 FALSE_NODE를 리턴해준다.

runFunction - COND

```
case COND:
    if(operand.cdr() == null) {
        errorLog("run Expr error");
        return null;
    }
    if(((ListNode)operand.car()).car() instanceof BooleanNode) { // 리스트 안에 노드가 Boolean 일경우
        if(((ListNode)operand.car()).car() == BooleanNode.TRUE_NODE) return ((ListNode)operand.car()).cdr().car();
        else return runFunction((FunctionNode)operator, operand.cdr());
    }
    else if(((ListNode)operand.car()).car() instanceof ListNode) { // 리스트 안에 리스트가 있을때
        Node temp = null;
        if(((ListNode)((ListNode)operand.car()).car()).car() instanceof FunctionNode) {
            temp = runFunction((FunctionNode)((ListNode)((ListNode)operand.car()).car()).car(), ((ListNode)((ListNode)operand.car()).car()).cdr());
        }
        else if(((ListNode)((ListNode)operand.car()).car()).car() instanceof BinaryOpNode) {
            temp = runBinary((ListNode)((ListNode)operand.car()).car());
        }
        if(temp instanceof IntNode) return temp;
        else if(temp instanceof BooleanNode) {
            if(temp == BooleanNode.TRUE_NODE) return ((ListNode)operand.car()).cdr().car();
            else return runFunction((FunctionNode)operator, operand.cdr());
        }
    }
    else if(((ListNode)operand.car()).car() instanceof FunctionNode) { // 리스트 안에 노드가 funchtion 일 경우
        if(runFunction((FunctionNode)((ListNode)operand.car()).car(), ((ListNode)operand.car()).cdr()) == BooleanNode.TRUE_NODE) return ((ListNode)operand.car()).cdr().car();
        else return runFunction((FunctionNode)operator, operand.cdr());
    }
    else if(((ListNode)operand.car()).car() instanceof BinaryOpNode) { // 리스트 안에 노드가 BinaryOpNode 일 경우
        if(runBinary((ListNode)operand.car()) == BooleanNode.TRUE_NODE) return ((ListNode)operand.car()).cdr().car();
        else return runFunction((FunctionNode)operator, operand.cdr());
    }
    else {
        errorLog("run Expr error");
        return null;
    }
```

1. operator는 COND고 operand는 그 다음에 오는 리스트노드이다.

2. operand의 car의 car, 즉 두 개의 리스트 중 첫 번째 리스트의 타입이 BooleanNode일 경우 TRUE, FALSE를 검사해서 TRUE일 경우 리스트 오른쪽 값을 리턴해 준다. FALSE일 경우 옆에 있는 리스트에 대한 재귀를 한다.

3. operand의 car의 car, 즉 두 개의 리스트 중 첫 번째 리스트의 타입이 ListNode일 경우 FunctionNode 나 BinaryOpNode 와 같은 연산이다. 해당 연산에 대한 리턴값을 temp에 저장해서 타입을 검사하여 IntNode일 경우에는 값을 리턴해주고, BooleanNode의 TRUE일 경우 해당 리스트의 오른쪽 값을 리턴해 준다. FALSE일 경우 옆에 있는 리스트에 대해 재귀를 한다.

4. functionNode일 경우와 BinaryOpNode에 대해 TRUE일 경우 오른쪽 값 리턴, FALSE일 경우 옆의 리스트에 대해 재귀를 해준다. (ListNode일 경우에 재귀를 하면 4번에 대해 처리)

runFunction - ATOM_Q

```
case ATOM_Q:
    if (runQuote(operand) instanceof IdNode || runQuote(operand) instanceof IntNode) { // list가 아닌경우는 IdNo
        return BooleanNode.TRUE_NODE;
    } else if ((ListNode)runQuote(operand) instanceof ListNode) // list일경우
        return BooleanNode.FALSE_NODE;
    else {
        errorLog("run Expr error");
        return null;
    }
```

1. operator는 ATOM_Q고 operand는 그 다음에 오는 리스트노드이다.

2. operand의 runQuote 결과 리스트가 IdNode나 IntNode 일경우 BooleanNode의 TRUE_NODE를 리턴해준다.

3. 리스트일 경우에는 BooleanNode의 FALSE_NODE를 리턴해준다.

runFunction - EQ_Q

```
case EQ_Q:
    if(runQuote(operand) instanceof IdNode || runQuote(operand) instanceof IntNode) { // IdNode 이거나 IntNode 일 경우
        if(runQuote(operand).toString().equals(runQuote(operand.cdr()).toString())) return BooleanNode.TRUE_NODE;
        else return BooleanNode.FALSE_NODE;
    }
    else { // ListNode 일 경우
        ListNode temp = (ListNode)runQuote((ListNode)operand);
        ListNode temp2 = (ListNode)runQuote((ListNode)operand.cdr());

        while(temp.car() != null && temp.cdr() != null && temp2.car() != null && temp2.cdr() != null) {
            if(temp.car().toString().equals(temp2.car().toString())) {
                temp = temp.cdr();
                temp2 = temp2.cdr();
                continue;
            }
            else return BooleanNode.FALSE_NODE;
        }
        if(temp.car() != null || temp.cdr() != null || temp2.car() != null || temp2.cdr() != null) return BooleanNode.FALSE_NODE;
        else return BooleanNode.TRUE_NODE;
    }
```

1. operator는 EQ_Q고 operand는 그 다음에 오는 리스트노드이다.

2. operand의 runQuote 결과 리스트가 IdNode나 IntNode 일경우 toString과 equals를 이용하여 값을 비교하고 같으면 BooleanNode의 TRUE_NODE를 리턴해준다.

3. 리스트일 경우에는 리스트를 equals를 통해 비교하면 true가 나오지 않기 때문에 직접 리스트 안의 노드를 하나하나 비교해준다. 다르면 BooleanNode의 FALSE_NODE를 리턴해준다.

4. 비교를 다 하고나서 노드의 개수가 다를경우에도 FALSE_NODE를 리턴해주고 다 맞으면 TRUE_NODE를 리턴해준다.

runFunction - NOT

```
case NOT:
    if(operand.car() instanceof BooleanNode) { // BooleanNode 일경우
        if(operand == BooleanNode.TRUE_NODE) return BooleanNode.FALSE_NODE;
        else return BooleanNode.TRUE_NODE;
    }
    else if(((ListNode)operand.car()).car() instanceof FunctionNode) { // FunctionNode 일경우
        if(runFunction((FunctionNode)((ListNode)operand.car()).car(), ((ListNode)operand.car()).cdr()) == BooleanNode.TRUE_NODE) return BooleanNode.FALSE_NODE;
        else return BooleanNode.TRUE_NODE;
    }
    else if (((ListNode)operand.car()).car() instanceof BinaryOpNode) { // BinaryOpNode 일경우
        if(runBinary((ListNode)operand.car()) == BooleanNode.TRUE_NODE) return BooleanNode.FALSE_NODE;
        else return BooleanNode.TRUE_NODE;
    }
    else {
        errorLog("run Expr error");
        return null;
    }
```

1. operator는 NOT이고 operand는 그 다음에 오는 리스트노드이다.

2. operand의 첫 노드가 BooleanNode일 경우 반대의 BooleanNode를 리턴해준다.

3. operand의 리스트의 첫노드가 FunctionNode일 경우 runFunction 재귀를 통해 TRUE 인지 FALSE 인지를 받아온 후 반대의 BooleanNode를 리턴해준다.

4. operand의 리스트의 첫노드가 BinaryOpNode일 경우 runBinary 재귀를 통해 TRUE 인지 FALSE 인지를 받아온 후 반대의 BooleanNode를 리턴해준다.

runBinary - PLUS

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

      Node node1 = list.cdr().car(); // 앞 노드
      Node node2 = list.cdr().cdr().car(); // 뒤 노드
      IntNode num1 = (IntNode)runExpr(node1);  // runExpr했을때 리스트 노드이고 리스트의 노드가 연산자 일 경우 재귀
      IntNode num2 = (IntNode)runExpr(node2);

      switch (operator.value) {
      case PLUS:
         try {
            IntNode result = new IntNode(num1.value + num2.value + "");
            return result;
         } catch (Exception e) {
            errorLog("run Expr error");
         }
```

1. 리스트의 앞 노드와 리스트의 뒤 노드를 먼저 저장한다.

2. 노드 두개가 IntNode가 나올 때 까지 runExpr를 통해서 재귀를 해준다.

3. 값을 더해 새로운 IntNode를 생성하여 리턴해준다.

runBinary - MINUS

```
case MINUS:
    try {

        IntNode result = new IntNode(num1.value - num2.value + "");
        return result;
    } catch (Exception e) {
        errorLog("run Expr error");
    }
```

1. 값을 빼서 새로운 IntNode를 생성하여 리턴해준다.

runBinary - DIV

```
case DIV:
    try {
        IntNode result = new IntNode(num1.value / num2.value + "");
        return result;
    } catch (Exception e) {
        errorLog("run Expr error");
    }
```

1. 값을 나누어서 새로운 IntNode를 생성하여 리턴해준다.

runBinary - TIMES

```
case TIMES:
   try {
      IntNode result = new IntNode(num1.value * num2.value + "");
      return result;
   } catch (Exception e) {
      errorLog("run Expr error");
   }
```

1. 값을 곱해서 새로운 IntNode를 생성하여 리턴해준다.

runBinary - LT

```
case LT:
      if(num1.value < num2.value) {
         BooleanNode result = BooleanNode.TRUE_NODE;
         return result;
      }else if(num1.value > num2.value) {
         BooleanNode result = BooleanNode.FALSE_NODE;
         return result;
      }else {
         errorLog("run Expr error");
      }
```

1. 값을 비교해서 '<'에 대해 TRUE이면 TRUE_NODE를 리턴하고, 아닐경우 FALSE_NODE를 리턴한다.

runBinary - GT

```
case GT:
    if(num1.value > num2.value) {
        BooleanNode result = BooleanNode.TRUE_NODE;
        return result;
    }else if(num1.value < num2.value) {
        BooleanNode result = BooleanNode.FALSE_NODE;
        return result;
    }else {
        errorLog("run Expr error");
    }
```

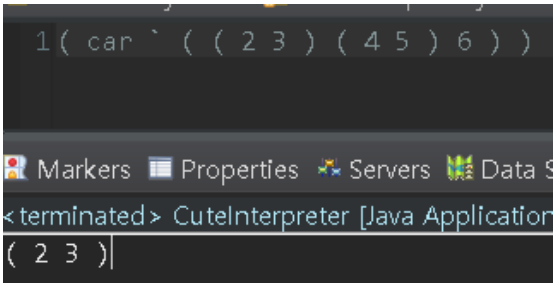
1. 값을 비교해서 '>'에 대해 TRUE이면 TRUE_NODE를 리턴하고, 아닐경우 FALSE_NODE를 리턴한다.

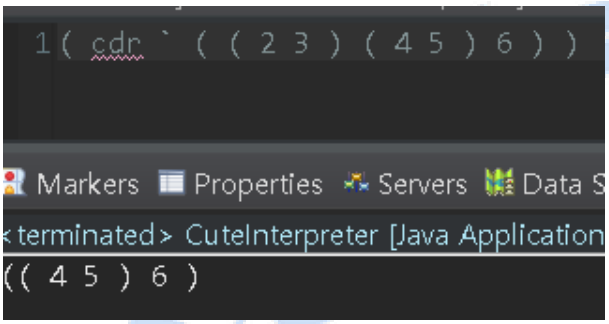
runBinary - EQ

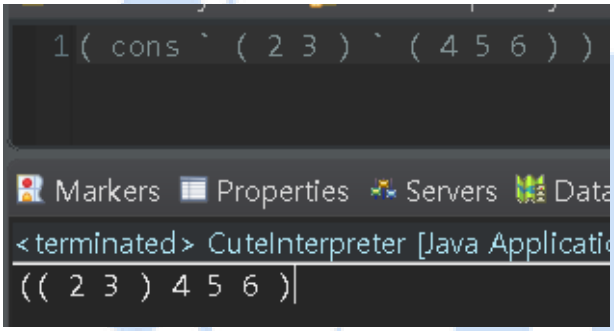
```
case EQ:
    try {
    if(num1.value == num2.value) {
        BooleanNode result = BooleanNode.TRUE_NODE;
        return result;
    }else {
        BooleanNode result = BooleanNode.FALSE_NODE;
        return result;
    }
    }catch(Exception e) {
        errorLog("run Expr error");
    }
```

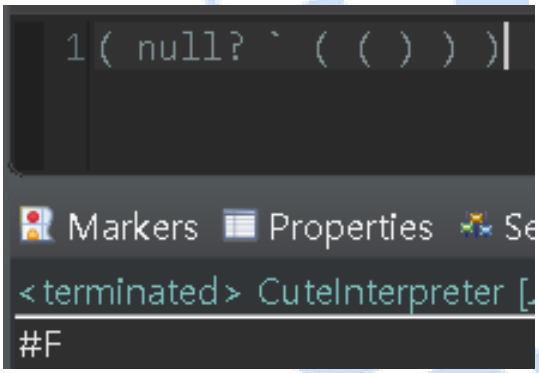
1. 값을 비교해서 '='에 대해 TRUE이면 TRUE_NODE를 리턴하고, 아닐경우 FALSE_NODE를 리턴한다.

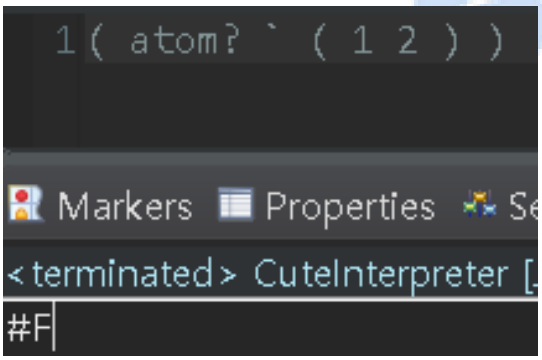
실행결과

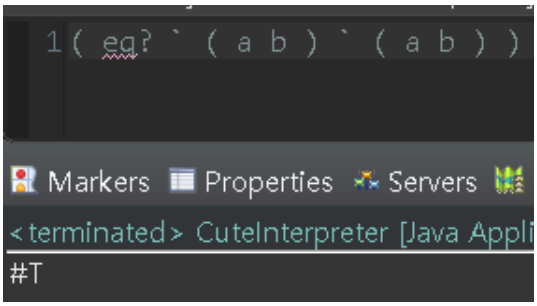

1 (car ` ((2 3) (4 5) 6))
Markers Properties Servers Data S
<terminated> CuteInterpreter [Java Application
(2 3)


1 (cdr ` ((2 3) (4 5) 6))
Markers Properties Servers Data S
<terminated> CuteInterpreter [Java Application
((4 5) 6)

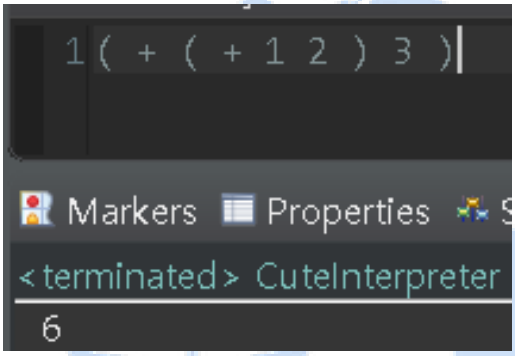

1 (cons ` ( 2 3 ) ` (4 5 6))
Markers Properties Servers Data
<terminated> CuteInterpreter [Java Applicati
((2 3) 4 5 6)

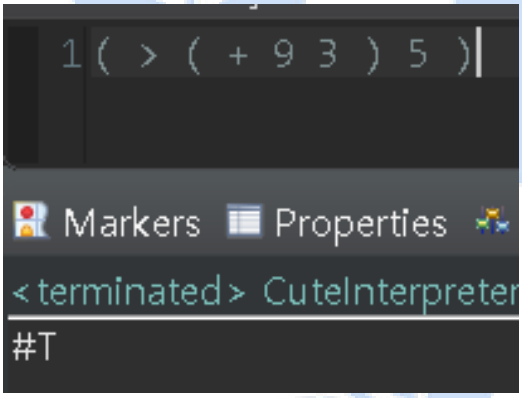

1 (null? ` (( )))
Markers Properties Se
<terminated> CuteInterpreter [
#F

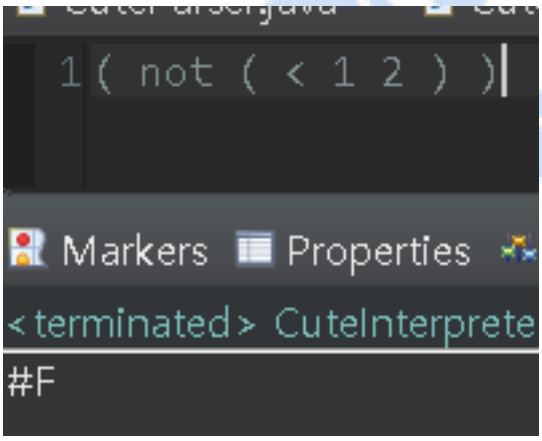

1 (atom? ` (1 2))
Markers Properties Se
<terminated> CuteInterpreter [
#F

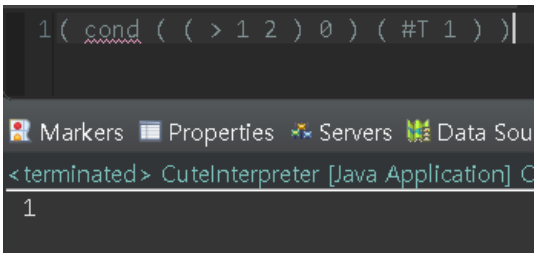

1 (eq? ` ( a b ) ` (a b))
Markers Properties Servers
<terminated> CuteInterpreter [Java Appli
#T

기타연산


1 (+ (+ 1 2) 3)
Markers Properties
<terminated> CuteInterpreter
6


1 (> (+ 9 3) 5)
Markers Properties
<terminated> CuteInterpreter
#T


1 (not (< 1 2))
Markers Properties
<terminated> CuteInterprete
#F


1 (cond ((> 1 2) 0) (#T 1))
Markers
Properties
Servers
Data Sou
<terminated> CuteInterpreter [Java Application] C
1