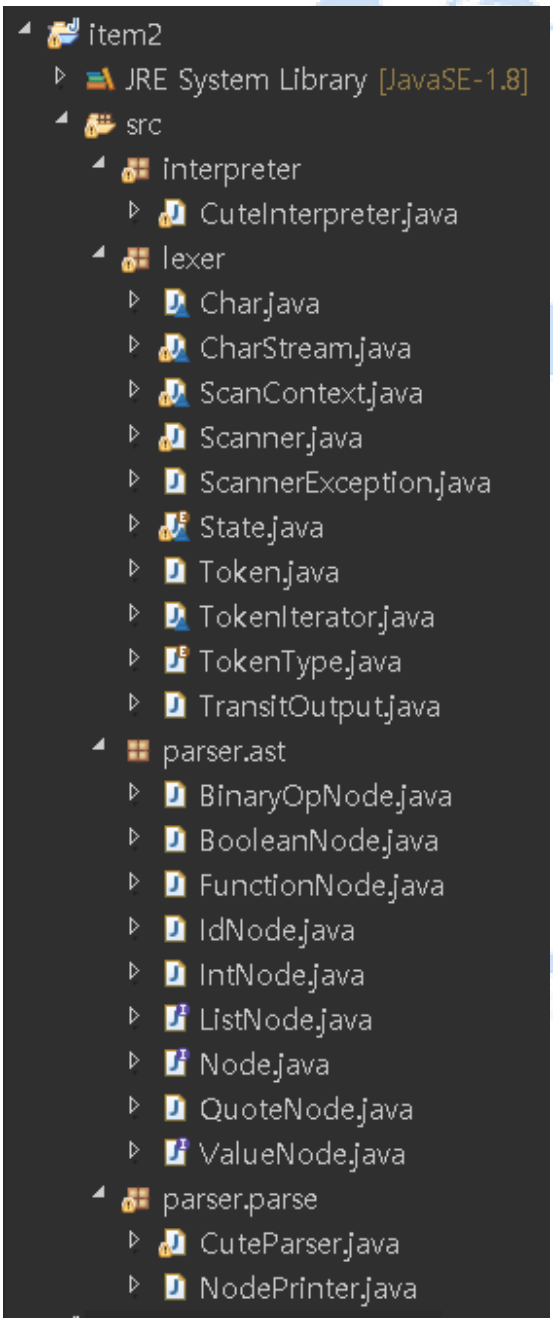
프로그래밍언어개론 보고서

컴퓨터공학과 201604140 박경수

Project Item 2. 변수의 바인딩 처리

전체 구조



HashMap 선언

```
public class CuteInterpreter {
    static HashMap<String, Node> hm = new HashMap<String, Node>();
```

변수에 대해 심벌테이블을 만들기 위해서 Key는 String형, Value값은 Node를 저장할 수 있는 HashMap을 정적으로 선언 해준다.

HashMap은 'hm'이라고 정의한다.

runFunction case문 추가 - DEFINE

```
case DEFINE:
    if(operand.car() instanceof IdNode) {
        insertTable((IdNode)operand.car(), operand.cdr());
    }
    else {
        errorLog("runFunction(DEFINE) error");
        return null;
    }
```

Define으로 변수를 선언하게 되면 먼저 IdNode인지 검사한다. IdNode가 아닐 경우에는 변수를 선언 할 수 없으므로(ex, 1이나 리스트) 에러를 출력해준다.

IdNode일 경우에는 insertTable 메소드를 실행시킨다.

insertTable 추가

```
private void insertTable(IdNode id, ListNode value) {
    if(value.car() instanceof IntNode) {
        hm.put(id.toString(), value.car());
        System.out.println("insertTable success");
    }
    else if (value.car() instanceof QuoteNode || value.car() instanceof BooleanNode) {
        hm.put(id.toString(), value);
        System.out.println("insertTable success");
    }
    else if (value.car() instanceof ListNode) { // ListNode일 경우
        if (((ListNode)value.car()).car() instanceof BinaryOpNode) {
            hm.put(id.toString(), runBinary((ListNode)value.car()));
            System.out.println("insertTable success");
        }
        else if (((ListNode)value.car()).car() instanceof FunctionNode) {
            hm.put(id.toString(), runFunction((FunctionNode)((ListNode)value.car()).car(), ((ListNode)value.car()).cdr()));
            System.out.println("insertTable success");
        }
    }
    else {
        errorLog("insertTable error");
    }
}
```

insertTable 메소드는 IdNode(변수가 될 노드)와 Value값(변수의 내용)을 매개변수로 받아서 실행된다. Value는 리스트로 받기 때문에 리스트에 대한 car()을 통해 타입에 따른 HashMap 저장을 다르게 해준다.

Key값은 모두 IdNode의 toString 값에 대해 저장한다.

Value값은 먼저, IntNode일 때는 Value값의 car()에 대해 저장해준다.

QuoteNode와 BooleanNode일 때는 Value값 그대로, 즉 리스트형태로 저장해준다.

이외의 경우는 ListNode일 경우인데, ListNode일 경우에는 FunctionNode와 BinaryOpNode로 나뉘어 진다. 이 경우에는 runBinary와 runFunction을 처리한 후 저장해준다.

모든 경우가 아닌 경우에 insertTable error를 출력한다.

LookupTable 추가

```
private Node lookupTable(IdNode id) {
    if(hm.get(id.toString()) instanceof ListNode) { // 변수를 가져왔는데 리스트일 경우
        // 리스트의 car이 QuoteNode일 경우
        if(((ListNode)(hm.get(id.toString()))).car() instanceof QuoteNode) {
            // 그 리스트의 notdInside의 car이 BinaryOpNode일 경우에는 그냥 리턴해주지 않고 결과값에 대해 리턴해준다.
            if(((ListNode)((QuoteNode)((ListNode)(hm.get(id.toString()))).car()).nodeInside()).car() instanceof BinaryOpNode) {
                return runList(((ListNode)((QuoteNode)((ListNode)(hm.get(id.toString()))).car()).nodeInside()));
            }
        }
        return ((ListNode)(hm.get(id.toString()))).car(); // BinaryOpNode가 아닐 경우 리스트의 car에 대해 리턴
    }
    return hm.get(id.toString()); // 리스트가 아니면 그냥 리턴해준다.
}
```

LookupTable은 변수에 대한 값을 리턴 해주는 메소드이다.

먼저 변수를 가져온 값이 리스트인지 아닌지에 대해 판별 후, 리스트가 아니면 해당 값을 리턴해준다.

가져온 값이 리스트일 경우는 insertTable에서 QuoteNode와 BooleanNode로 설정했었는데 QuoteNode일 때는 BinaryOpNode이면 runList를 통한 결과값을 리턴해주고 그 외의 경우는 리스트의 car()에 대해 리턴해준다.

runExpr 수정

```
public Node runExpr(Node rootExpr) {
    if (rootExpr == null)
        return null;
    if (rootExpr instanceof IdNode)
        return lookupTable((IdNode)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;
}
```

IdNode일 경우에는 LookupTable을 통해 변수값을 가져온다.

runList 수정

```
private Node runList(ListNode list) {
    Node temp = list.cdr();
    if(list.equals(ListNode.EMPTYLIST))
        return list;
    if(list.car() instanceof FunctionNode){ // functionNode일 경우
        if(((ListNode)temp).cdr().car() != null) { // ` a 같은 경우는 변수가 아니기때문에 원래 처럼 처리
            return runFunction((FunctionNode)list.car(), list.cdr());
        }
        if(hm.get(list.cdr().car().toString()) != null) { // 변수인 경우 바꿔준다.
            temp = hm.get(list.cdr().car().toString());
        }
        return runFunction((FunctionNode)list.car(), (ListNode)temp); // runFunction할때 list.cdr()은 바꿔준것으로 넘김
    }
    else if(list.car() instanceof BinaryOpNode){ // BinaryOpNode일 경우
        return runBinary(list);
    }
    return list;
}
```

기존에 runList는 FunctionNode이면 runFunction(), BinaryOpNode이면 runBinary()를 호출 해주었는데, 변경된 runList는 FunctionNode일 때 일반적인 경우에는 기존처럼 runFunction 처리를 해주고, 변수인 경우에는 원래 값으로 바꾸어 runFunction을 처리한다.

runFunction - CAR, CDR 수정

```
case CAR:
    if (operand.car() instanceof QuoteNode) { // car다음에 오는 노드가 ` 인지 검사
       ListNode result = (ListNode)runQuote(operand); // runQuote를 통해 nodeInside로 리스트 안쪽 리스트노드를 가져옴
       return result.car();
    }else {
        errorLog("runFunction(CAR) error");
        return null;
    }
case CDR:
    if (operand.car() instanceof QuoteNode) { // car다음에 오는 노드가 ` 인지 검사
    ListNode result = (ListNode)runQuote(operand); // runQuote를 통해 nodeInside로 리스트 안쪽 리스트노드를 가져옴
    return result.cdr();
    }else {
        errorLog("runFunction(CDR) error");
        return null;
    }
```

기존과 동일하고, 에러에 대한 출력만 변경해줌.

runFunction - CONS 수정

```
case CONS:
    if(operand.cdr().car() instanceof QuoteNode) { // 뒤에 노드가 QuoteNode 일 경우
        if(operand.car() instanceof QuoteNode) {
            ListNode result = ListNode.cons((ListNode)runQuote(operand), (ListNode)runQuote(operand.cdr()));
            return result;
        }
        else if(operand.car() instanceof IdNode) {
            if(hm.get((IdNode)operand.car()) != null) {
                ListNode result = ListNode.cons(lookupTable((IdNode)operand.car()), (ListNode)runQuote(operand.cdr()));
                return result;
            }
            else {
                ListNode result = ListNode.cons((IdNode)operand.car(), (ListNode)runQuote(operand.cdr()));
                return result;
            }
        }
        else {
            ListNode result = ListNode.cons(operand.car(), (ListNode)runQuote(operand.cdr()));
            return result;
        }
    }
```

CONS는 뒤에 있는 노드가 QuoteNode일 경우와 아닐 경우로 나뉘어 진다.

먼저 QuoteNode일 경우에 앞에 노드의 타입을 검사 해준다.

앞 노드가 QuoteNode일 경우는 앞 뒤 노드에 대해 runQuote 후 cons를 처리한다.

IdNode일 경우에는 해당 IdNode가 변수인지 검사해서 변수가 아닐 경우 IdNode와 뒤 노드를 cons 하고, 변수일 경우에는 변수의 원래 값을 얻어와 cons를 처리한다.

그 외의 경우(ex, IntNode)는 해당 값에 대해 바로 cons를 처리해준다.

```
else {  // 뒤에 노드가 QuoteNode 가 아닐경우
    if(operand.cdr().car() instanceof IdNode) { // operand.cdr()과 리스트를 병합해야하는데 operand.cdr()이 변수 일 경우
        if(hm.get(((ListNode)operand.cdr()).car().toString()) != null) {
            ListNode temp_cdr = (ListNode)runQuote((ListNode)hm.get(((ListNode)operand.cdr()).car().toString()));
            if(operand.car() instanceof QuoteNode) {
                ListNode result = ListNode.cons((ListNode)runQuote(operand), temp_cdr);
                return result;
            }
            else if(operand.car() instanceof IdNode) {
                ListNode result = ListNode.cons(lookupTable((IdNode)operand.car()), temp_cdr);
                return result;

            }
            else {
                ListNode result = ListNode.cons(operand.car(), temp_cdr);
                return result;
            }
        }
        else { // 뒤의 노드가 변수가 아닌데 IdNode일 경우 에러
            errorLog("runFunction(CONS) error");
            return null;
        }
    }
    else { // 뒤의 노드가 결합할 수 없는 node일 경우 에러
        errorLog("runFunction(CONS) error");
        return null;
    }
}
```

뒤에 노드가 QuoteNode가 아닐 경우에는 두 가지 경우인데, 변수로 정의 되어있는 IdNode일 경우를 제외하면 error를 출력한다. (변수가 아닌 IdNode와는 결합할 수 없으므로)

IdNode가 변수로 정의가 되어있으면 이전에 처리했던 앞에 노드의 타입을 검사하는 것을 똑같이 처리 해준다. 다른 점은 결합하려는 노드가 변수에서 가져온 값이다.

runFunction - COND 수정

```
case COND:
    if(operand.cdr() == null) {
        errorLog("runFunction(COND) error");
        return null;
    }
```

COND는 크게 5가지 경우의 수가 존재한다.

먼저 예상치 못한, 값이 없을 경우에는 에러를 출력해준다.

```
if(((ListNode)operand.car()).car() instanceof BooleanNode) { // 리스트 안에 앞 노드가 Boolean 일경우
    if(((ListNode)operand.car()).car() == BooleanNode.TRUE_NODE) { // TRUE면 옆에 노드를 리턴해준다.
        if(hm.get((((ListNode)operand.car()).cdr().car()).toString()) != null) { // 옆에 노드를 리턴해주기 전에 변수일 경우
            if (hm.get((((ListNode)operand.car()).cdr().car()).toString()) instanceof ListNode) { // 그 변수가 리스트일 경우
                return ((ListNode)(hm.get((((ListNode)operand.car()).cdr().car()).toString()))).car(); // 변수의 car에 대해 리턴해준다.
            }
            return hm.get((((ListNode)operand.car()).cdr().car()).toString()); // 변수에 대한 변환을 해준 후 리턴한다.
        }
        else return ((ListNode)operand.car()).cdr().car(); // 변수가 아닐 경우 그냥 리턴해준다.
    }
    else return runFunction((FunctionNode)operator, operand.cdr()); // FALSE면 뒤에 노드에 대해 재귀
}
```

두 번째로 리스트 안에 앞 노드가 Boolean일 경우에 대해 처리한다. (ex, cond (#T …)

True이면 옆에 있는 노드를 리턴 해주면 되는데, 해당 노드가 변수일 경우와 그 변수가 리스트일 경우에 대해서도 처리해준다. FALSE일 경우 뒤에 노드에 대해 재귀를 한다.

```
else if(((ListNode)operand.car()).car() instanceof ListNode) { // 리스트 안에 앞 노드가 ListNode일 때
    Node temp = null;
    if(((ListNode)(((ListNode)operand.car()).car()).car() instanceof FunctionNode) { // 해당 리스트가 FunctionNode일 때
        temp = runFunction((FunctionNode)((ListNode)((ListNode)operand.car()).car()).car(), ((ListNode)((ListNode)operand.car()).car()).cdr()); // F
    }
    else if(((ListNode)((ListNode)operand.car()).car()).car() instanceof BinaryOpNode) { // 해당 리스트가 BinaryOpNode일 때
        temp = runBinary((ListNode)((ListNode)operand.car()).car()); // BinaryOpNode의 결과를 가져온다.
    }
    if(temp instanceof IntNode) {
        errorLog("runFunction(COND) error"); // 가져온 결과가 IntNode일 경우에는 에러를 출력한다.
        return null;
    }
    else if(temp instanceof BooleanNode) { // 가져온 결과가 BooleanNode일 경우에
        if(temp == BooleanNode.TRUE_NODE) {
            if(hm.get((((ListNode)operand.car()).cdr().car()).toString()) != null) { // 옆에 노드를 리턴해주기 전에 변수일 경우
                if (hm.get((((ListNode)operand.car()).cdr().car()).toString()) instanceof ListNode) { // 그 변수가 리스트일 경우
                    return ((ListNode)(hm.get((((ListNode)operand.car()).cdr().car()).toString()))).car(); // 변수의 car에 대해 리턴해준다.
                }
                return hm.get((((ListNode)operand.car()).cdr().car()).toString()); // 변수에 대한 변환을 해준 후 리턴한다.
            }
            else return ((ListNode)operand.car()).cdr().car(); // 변수가 아닐 경우 그냥 리턴해준다.
        }
        else return runFunction((FunctionNode)operator, operand.cdr()); // FALSE이면 뒤에 노드에 대한 재귀
    }
}
```

세 번째로 리스트 안에 앞 노드가 ListNode일 경우에 대해 처리한다. (ex, cond ((> 2 1) …)

ListNode인 경우는 두 가지인 FunctionNode와 BinaryOpNode인 경우 이므로 참, 거짓을 판별 해주고 만약에 결과값이 true, false가 아니라 IntNode가 나왔다면 에러를 출력한다.

True이면 옆에 있는 노드를 리턴 해주면 되는데, 해당 노드가 변수일 경우와 그 변수가 리스트일 경우에 대해서도 처리해준다. FALSE일 경우 뒤에 노드에 대해 재귀를 한다.

```
else if(((ListNode)operand.car()).car() instanceof IdNode) { // 리스트 안에 앞 노드가 IdNode일 경우 (변수)
    if(hm.get((((ListNode)operand.car()).car()).toString()) != null) {
        if(((ListNode)hm.get((((ListNode)operand.car()).car()).toString())).car() == BooleanNode.TRUE_NODE) { // TRUE면 옆에 노드를 리턴해준다.
            if(hm.get((((ListNode)operand.car()).cdr().car()).toString()) != null) { // 옆에 노드를 리턴해주기 전에 변수일 경우
                if (hm.get((((ListNode)operand.car()).cdr().car()).toString()) instanceof ListNode) { // 그 변수가 리스트일 경우
                    return ((ListNode)(hm.get((((ListNode)operand.car()).cdr().car()).toString()))).car(); // 변수의 car에 대해 리턴해준다.
                }
                return hm.get((((ListNode)operand.car()).cdr().car()).toString()); // 변수에 대한 변화를 해준 후 리턴한다.
            }
            else return ((ListNode)operand.car()).cdr().car(); // 변수가 아닐 경우 그냥 리턴해준다.
        }
        else return runFunction((FunctionNode)operator, operand.cdr()); // FALSE이면 뒤에 노드에 대한 재귀
    }
    else { // IdNode인데 변수로 정의되어 있지 않으면 에러
        errorLog("runFunction(COND) error");
        return null;
    }
}
```

네 번째로 리스트 안에 앞 노드가 IdNode일 경우에 대해 처리한다. 해당 경우는 IdNode가 변수가 아닐 경우에는 에러를 출력하고, 변수일 경우 True이면 옆에 있는 노드를 리턴 해주면 되는데, 해당 노드가 변수일 경우와 그 변수가 리스트일 경우에 대해서도 처리해준다. FALSE일 경우 뒤에 노드에 대해 재귀를 한다.

```
else {
    errorLog("runFunction(COND) error");
    return null;
}
```

마지막으로 모든 경우가 아닌 경우에 에러를 출력한다.

runFunction - NULL_Q 수정

```
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;
    }
```

기존과 동일하고, 에러에 대한 출력만 변경해줌.

runFunction - ATOM_Q 수정

```
case ATOM_Q:
    if (operand.car() instanceof QuoteNode) { // runQuote를 할 수 있는 QuoteNode일 경우
        if (runQuote(operand) instanceof IdNode || runQuote(operand) instanceof IntNode) { // list가 아닌경우는 IdNode 이거나 IntNode
            return BooleanNode.TRUE_NODE;
        } else if ((ListNode)runQuote(operand) instanceof ListNode) // list일경우 FALSE
            return BooleanNode.FALSE_NODE;
        else {
            errorLog("runFunction(ATOM_Q) error");
            return null;
        }
    }
    else { // QuoteNode가 아닌경우는 모두 참
        return BooleanNode.TRUE_NODE;
    }
```

기존에는 QuoteNode에 대한 검사를 하지 않았는데, (atom? B) 같은 경우에 대한 처리를 못하기 때문에 QuoteNode가 아닌 경우는 모두 참을 리턴 해주도록 변경

runFunction - EQ_Q 수정

```
if(operand.car() instanceof IdNode) { // 앞의 노드가 IdNode 일 경우
    if(hm.get(operand.car().toString()) != null) { // 앞의 노드가 변수일 경우
        if(((ListNode)operand.cdr()).car() instanceof IdNode) { // 뒤에 노드가 IdNode 일 경우
            if(hm.get((((ListNode)operand.cdr()).car()).toString()) != null) { // 뒤에 노드가 변수일 경우
                if(hm.get(operand.car().toString()) instanceof ListNode) { // 앞에 변수가 리스트인 경우
                    if(hm.get((((ListNode)operand.cdr()).car()).toString()) instanceof ListNode) { // 뒤에 변수도 리스트인 경우
                        ListNode temp = (ListNode)runQuote((ListNode)hm.get(operand.car().toString()));
                        ListNode temp2 = (ListNode)runQuote((ListNode)hm.get((((ListNode)operand.cdr()).car()).toString()));

                        while(temp.car() != null && temp.cdr() != null && temp2.car() != null && temp2.cdr() != null) { // 리스트 하나하나에 대한 비교
                            if(temp.car().toString().equals(temp2.car().toString())) {
                                temp = temp.cdr();
                                temp2 = temp2.cdr();
                                continue;
                            }
                            else return BooleanNode.FALSE_NODE; // 하나라도 다르면 FALSE
                        }
                        if(temp.car() != null || temp.cdr() != null || temp2.car() != null || temp2.cdr() != null) return BooleanNode.FALSE_NODE; // 비교 후 노드의 개수가 다르면 FALSE
                        else return BooleanNode.TRUE_NODE;
                    }
                    else { // 앞에 변수는 리스트인데 뒤에 변수는 리스트가 아닌경우
                        return BooleanNode.FALSE_NODE;
                    }
                }
                else { // 앞에 변수가 리스트가 아닌 경우
                    if(((ListNode)operand.cdr()).car() instanceof ListNode ) { // 뒤에 변수는 리스트인 경우
                        return BooleanNode.FALSE_NODE;
                    }
                    else { // 앞에 변수와 뒤에 변수가 둘다 리스트가아닌 경우
                        if(hm.get(operand.car().toString()).toString().equals(hm.get((((ListNode)operand.cdr()).car()).toString()).toString())) return BooleanNode.TRUE_NODE;
                        else return BooleanNode.FALSE_NODE;
                    }
                }
            }
            else { // 뒤에 노드가 변수가 아닌 IdNode일 경우
                if((hm.get(operand.car().toString())).toString().equals((operand.cdr().car()).toString())) return BooleanNode.TRUE_NODE;
                else return BooleanNode.FALSE_NODE;
            }
        }
        else { // 뒤에 노드가 IntNode 일 경우
            if((hm.get(operand.car().toString())).toString().equals((operand.cdr().car()).toString())) return BooleanNode.TRUE_NODE;
            else return BooleanNode.FALSE_NODE;
        }
    }
    else { // IdNode인데 변수가 정의되어있지 않은 경우
        if(operand.car().toString().equals((operand.cdr().car()).toString())) return BooleanNode.TRUE_NODE;
        else return BooleanNode.FALSE_NODE;
    }
}
```

EQ_Q는 크게 4가지 경우로 나뉘어 진다.

첫 번째로 비교하려는 앞 노드가 IdNode일 경우에는 변수인 경우와 아닌 경우로 처리한다. 변수가 아닌 경우에는 바로 뒤에 노드와 비교해준다.

앞 노드가 변수인 경우에는 뒤에 노드가 변수인 경우와 아닌 경우로 다시 처리한다. 변수가 아닌 경우에는 바로 비교해준다.

두 노드가 변수인 경우에는 이제 변수의 원래 값이 리스트인지를 체크해서 변수의 원래 값들 끼리 비교하여 true false를 출력해준다.

```
else if(operand.car() instanceof IntNode) { // 앞의 노드가 IntNode 일 경우
    if(((ListNode)operand.cdr()).car() instanceof IdNode) { // 뒤에 노드가 IdNode 일 경우
        if(hm.get((((ListNode)operand.cdr()).car()).toString()) != null) {
            if(hm.get((((ListNode)operand.cdr()).car()).toString()) instanceof ListNode) { // 변수가 리스트인 경우
                return BooleanNode.FALSE_NODE;
            }
            else if(operand.car().toString().equals(hm.get((((ListNode)operand.cdr()).car()).toString()).toString())) return BooleanNode.TRUE_NODE;
            else return BooleanNode.FALSE_NODE;
        }
        else { // 뒤에 노드가 정의가 안되었을경우 FALSE
            return BooleanNode.FALSE_NODE;
        }
    }
    else { // 뒤에 노드가 IntNode 일 경우
        if(operand.car().toString().equals((operand.cdr().car()).toString())) return BooleanNode.TRUE_NODE;
        else return BooleanNode.FALSE_NODE;
    }
}
```

두 번째 경우는 앞의 노드가 IntNode일 경우에 대해 처리해준다.

뒤에 노드 또한 IntNode일 경우에는 바로 비교를 해주고, 뒤에 노드가 IdNode일 경우에는 변수가 아니라면 FALSE를 출력해준다.

뒤에 노드가 변수라면 변수에 대한 원래 값과 비교 해준다.

```
else if(runQuote(operand) instanceof IntNode || runQuote(operand) instanceof IdNode) { // runQuote시 IntNode,IdNode 일 경우
    if(runQuote(operand).toString().equals(runQuote(operand.cdr()).toString())) return BooleanNode.TRUE_NODE;
    else return BooleanNode.FALSE_NODE;
}
```

세 번째 경우부터는 이전에 정의했던 그대로이다. 변수가 아닌 RunQuote시에 IntNode, IdNode에 대해서 비교처리를 해준다.

```
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) { // 리스트 하나하나에 대한 비교
        if(temp.car().toString().equals(temp2.car().toString())) {
            temp = temp.cdr();
            temp2 = temp2.cdr();
            continue;
        }
        else return BooleanNode.FALSE_NODE; // 하나라도 다르면 FALSE
    }
    if(temp.car() != null || temp.cdr() != null || temp2.car() != null || temp2.cdr() != null) return BooleanNode.FALSE_NODE; // 비교 후 노드의 개수가 다르면 FALSE
    else return BooleanNode.TRUE_NODE;
}
```

네 번째 경우는 리스트 일 경우에 대해서 처리해준다.

리스트의 노드 하나 하나에 대해 비교해주고 마지막에 개수가 다르면 FALSE를 출력한다.

runFunction - NOT 수정

```
case NOT:
    if(operand.car() instanceof BooleanNode) { // BooleanNode 일경우
        if(operand.car() == 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("runFunction(NOT) error");
        return null;
    }
```

기존과 동일하고, 에러에 대한 출력만 변경해줌.

runBinary - 수정

기존과 동일하고, 에러에 대한 출력만 변경해줌.

NodePrinter - printNode 수정

```
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 if (node instanceof BooleanNode) {
        if(node == BooleanNode.FALSE_NODE) ps.print("#F");
        else ps.print("#T");
    } else if (node instanceof FunctionNode) {
        ps.print(node.toString());
    }
    else {
        String temp = node.toString();
        StringTokenizer st = new StringTokenizer(temp, " ");
        st.nextToken();
        ps.print(" " + st.nextToken());
    }
}
```

functionNode일 경우에 node.toString을 출력시키는 if문을 추가시킨다. (ex, (car ` ( cdr ` (1 2 3 4))))

실행결과

```
CuteInterpreter (4) [Java Application] C:₩Pr
> ( define a 1 )
insertTable success
...
> ( car ` ( a 2 3 ) )
...  a
> ( define b ` ( 1 2 3 ) )
insertTable success
...
> ( car b )
...  1
> ( define a 2 )
insertTable success
...
> ( define b 2 )
insertTable success
...
> ( eq? a b )
... #T
> ( eq? ` a ` b )
... #F
> ( define a ` ( 1 2 3 ) )
insertTable success
...
> ( define b ` ( 1 2 3 ) )
insertTable success
...
> ( eq? a b )
... #T
> ( define a ` ( 2 3 ) )
insertTable success
...
```

```
> ( cons 1 a )
... ( 1 2 3 )
> ( define a 2 )
insertTable success
...
> ( cond ( #T a ) ( #F 6 ) )
...  2
> ( define a ` ( 2 3 ) )
insertTable success
...
> ( cond ( #T a ) ( #F 6 ) )
... ` ( 2 3 )
> ( define a #T )
insertTable success
...
> ( cond ( a ` ( 1 2 ) ) ( #F 6 ) )
... ` ( 1 2 )
> ( define a ( < 1 2 ) )
insertTable success
...
> a
... #T
> ( define a ` ( ) )
insertTable success
...
> a
... ` ( )
> ( null? a )
... #T
```

```
> ( car ` ( cdr ` ( 1 2 3 4 ) ) )
... CDR
> ( car ` ( cdr ` ( 1 2 3 ) ) )
... CDR
>
```