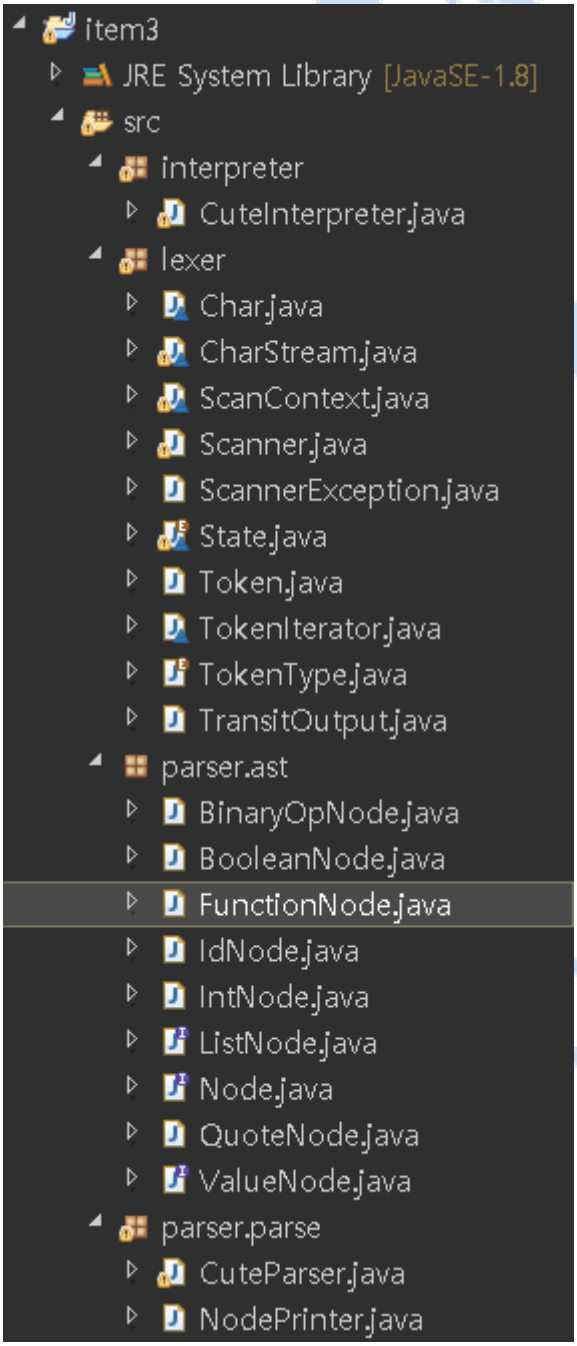
프로그래밍언어개론 보고서

컴퓨터공학과 201604140 박경수

Project Item 3. 함수의 바인딩 처리 및 코드 발전

전체 구조



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 if (node instanceof BinaryOpNode) {
        ps.print(node.toString());
    }
```

BinaryOpNode가 왔을 경우에 대한 print 값이 지정되어있지 않았었다.

FunctionNode와 같은 경우로 처리해준다.

runFunction case문 추가 - LAMBDA

```
case LAMBDA:
    if(operand.car() == null || operand.cdr() == null) { // 바인딩 할 변수 또는 함수가 없으면 에러
        errorLog("runFunction(LAMBDA) error");
        return null;
    }
    else {
        if(((ListNode)operand.cdr().car()).car() instanceof FunctionNode) { // 함수가 function일 경우
            return runFunction((FunctionNode)((ListNode)operand.cdr().car()).car(), ((ListNode)operand.cdr().car()).cdr());
        } else if(((ListNode)operand.cdr().car()).car() instanceof BinaryOpNode) { // 함수가 binary일 경우
            return runBinary((ListNode)operand.cdr().car());
        } else {
            errorLog("runFunction(LAMBDA) error");
            return null;
        }
    }
```

lambda로 선언하게 되면 먼저 바인딩 할 변수 또는 함수가 있는지 검사한다.

바인딩 할 변수나 함수가 있으면 함수가 FunctionNode일 경우와 BinaryOpNode일 경우에 대해 각각 처리를 해주게 된다.

insertTable 수정

```
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) {
            if (((FunctionNode)((ListNode)value.car()).car()).value.equals(FunctionNode.FunctionType.LAMBDA)) { // 만약에 람다일 경우(전역함수 구현)
                hm.put(id.toString(), value);
            }
            else hm.put(id.toString(), runFunction((FunctionNode)((ListNode)value.car()).car(), ((ListNode)value.car()).cdr()));
            System.out.println("insertTable success");
        }
    }
    else {
        errorLog("insertTable error");
    }
}
```

만약에 functionNode중에 LAMBDA일 경우를 추가해주었다.

람다일 경우 전역함수이며, 전역함수를 hashmap에 저장해준다.

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);
    }
    else if(list.car() instanceof IdNode){ // 전역 함수 일 경우는 IdNdode이다.
        if(hm.get(list.car().toString()) != null) { // 변수인 경우 바꿔준다.
            ListNode stemp = (ListNode)hm.get(list.car().toString());

            hm.put(((ListNode)((ListNode)stemp.car()).cdr().car()).car().toString(), list.cdr().car());
            Node temp_node = runFunction((FunctionNode)(((ListNode)stemp.car()).car()), ((ListNode)stemp.car()).cdr()); // 람다의 경우 runFunction 진행
            hm.remove(((ListNode)((ListNode)stemp.car()).cdr().car()).car().toString());
            return temp_node;

        }
    }
    else if(((ListNode)list.car()).car() instanceof FunctionNode) { // 람다일 경우
        hm.put(((ListNode)((ListNode)list.car()).cdr().car()).car().toString(), list.cdr().car());
        Node temp_node = runFunction((FunctionNode)(((ListNode)list.car()).car()), ((ListNode)list.car()).cdr()); // 람다의 경우 runFunction 진행
        hm.remove(((ListNode)((ListNode)list.car()).cdr().car()).car().toString());
        return temp_node;
    }
    return list;
}
```

전역함수일 경우는 ‘plus1’과 같이 IdNode인데, 이 경우에 hashmap에 저장 되어있는지 검사하여 아래 람다와 같은 경우와 동일하게 처리해준다.

람다일 경우는 먼저 바인딩 될 변수를 임시로 변수 테이블에 넣어준다. 임시로 runFunction을 수행한 값을 temp_node에 저장하고, 변수 테이블에 넣어줬던 값을 삭제 후 temp_node를 리턴 해준다.

실행결과

```
> ( ( lambda ( x ) ( + x 1 ) ) 2 )
...  3
> ( define plus1 ( lambda ( x ) ( + x 1 ) ) )
insertTable success
...
> ( plus1 3 )
...  4
>
```

전역함수까지 구현하였습니다.