프로그래밍 언어 개론

Item3

00분반 201602004 박태현

함수 및 SSR 구현 내용
람다 구현
함수 define 구현
전역 함수 호출
함수 내에서 전역 함수 호출
변수 scope 구현
중첩 함수 구현
재귀 함수 구현
함수를 인자로 사용 구현

람다를 받으면 처리해주는 함수를 만들었습니다. 람다에 대한 평가는 실제 람다 실행에서 이루어집니다

```
private Node evalLambda(ListNode operand) {
    return new Closure(listCar(operand), cdr(operand) );
}
```

repl.it에서 함수를 검색하면 #<Closure>라는 것을 출력해주길래 클래스 명을 클로저로 해주었습니다. 클로저 객체를 통해 람다 구현, 함수 정의, 함수 호출, 전역함수 호출, 일급 객체 함수를 구현하는 기반을 만들었습니다.

```
public class Closure implements Node {
    private ListNode arguments;
    private ListNode functionBody;
    private ActivationRecord parentARI;

public Closure(ActivationRecord ARI, ListNode arguments, ListNode functionBody){
    this.parentARI = ARI;
    this.arguments = arguments;
    this.functionBody = functionBody;
}
```

람다를 인터프리터에서 만나면 클로저 객체로 만들어서 인자와 함수 바디를 받아서 일단 노드 형태로 남겨둡니다. 스킴은 SSR을 따릅니다. 따라서 정의되는 시점에 ARI를 자신의 부모 ARI로 잡아두어야 하므로, 정의시에 부모 ARI를 받도록 했습니다.
ARI를 통해 SSR 구현에 따라서 중첩 함수, 지역 변수, 재귀 함수를 구현했습니다

```
private Node runList(ListNode list) {
    list = (ListNode)stripList(list);

    if (list.equals(ListNode.EMPTYLIST) || list.equals(ListNode.ENDLIST)) {
        return ListNode.EMPTYLIST;
    }
    else if (car(list) instanceof FunctionNode && !(list.isQuoted())) {
        return runFunction((FunctionNode) list.car(), list.cdr());
    }
    else if (car(list) instanceof BinaryOpNode && !(list.isQuoted())) {
        return runBinary(list);
    }
    /* Quoted List Process */
    Node nextNode = car(cdr(list));
    if(nextNode != null && (nextNode.equals(ListNode.ENDLIST) || nextNode.equals(ListNode.EMPTYLIST))) {
        return ListNode.EMPTYLIST;
    }
    else if (car(list) instanceof QuoteNode){
        return car(cdr(list));
    }
    else { // lambda
        return runLambda(list);
    }
}
```

리스트가 비거나 마지막이면 빈 리스트를 돌려줍니다.
리스트의 첫번째 인자가 함수나 연산자면 함수나 연산자 처리를 합니다.
만약 리스트의 다음 노드가 마지막이거나 없다면 빈 리스트를 돌려줍니다
첫번째가 quote라면 그냥 내부의 리스트를 꺼내서 돌려줍니다
이외의 경우는 모두 람다로 처리합니다.
(expr) 인 리스트는 함수, 연산자, 람다 세가지 뿐이므로 함수도 아니고 연산자도 아니고 quote도 아닌 리스트는 람다로 처리했습니다

```
private Node runLambda(ListNode list) {
    Closure lambdaExpression = null;

    if ( car(list) instanceof ListNode )
        lambdaExpression = (Closure)runList(listCar(list));
    else if ( car(list) instanceof IdNode ) { // is Closure
        lambdaExpression = (Closure)ARI.lookUptable(car(list));
    }
    else {
        throw new IllegalArgumentException(car(list).toString());
    }
```

람다를 받으면 첫번째 인자를 확인합니다. 첫번째 인자가 리스트라면
((lambda ...)) 형태를 가졌으므로 람다함수입니다. 따라서 우선 클로저를 생성합니다.
IdNode라면 그것은 함수 이름입니다 (x ...) 따라서 ARI로 가서 해당하는 Id에 바인딩 되어 있는 클로저를 가져옵니다. 만약 끝까지 못 찾았으면 예외를 던집니다.

```
public class ActivationRecord{

    private Map<String, Node> varTable;
    private ActivationRecord parentARI;

    public ActivationRecord(ActivationRecord parentARI) {
        varTable = new HashMap<String,Node>();
        this.parentARI = parentARI;
    }

public Node lookUptable(Node keyNode) throws IllegalArgumentException {
    Node node = varTable.get(keyNode.toString());
    if(node == null) {
        if(parentARI != null)
            return parentARI.lookUptable(keyNode);
        else
            throw new IllegalArgumentException(keyNode.toString());
    }
    return node;
}
```

활성 레코드는 부모 ARI를 기억하고 있습니다. 최상위 ARI는 null을 부모로 가리키고 있습니다. lookUpTable에서 현재 ARI에서 변수를 찾습니다. 만약 못 찾았으면 부모가 존재하는지 확인하고, 존재하면 부모로 가서 찾습니다. 끝까지 못 찾았으면 예외를 반환하고 끝냅니다

```
    ListNode formalParams = lambdaExpression.getArgs();
    ListNode actualParams = getArgs(cdr(list));
    ListNode functionBody = lambdaExpression.getFuncBody();
```

클로저를 받아왔다면 클로저는 인자와 함수 바디를 가지고 있습니다. 인자는 formalParam으로 가져옵니다.
(func args) 형태로 함수를 실행하므로, arg는 현재 받은 리스트의 cdr에 들어가있습니다. 따라서 cdr(list)를 getArgs로 인자를 평가한 뒤 가져옵니다. 이 평가하는 과정에서 함수나 연산자가 있다면 그것을 수행하고, 함수가 있다면 클로저를 받아서 가져오게 됩니다

```
private ListNode getArgs(ListNode list) {
    if(list.equals(ListNode.ENDLIST))
        return list;
    Node node = runExpr(car(list));

    return ListNode.cons(node, getArgs(cdr(list)));
}
```

각 인자를 평가하여 새로운 리스트를 만듭니다
함수 바디도 가져옵니다

```
if(length(formalParams) > length(actualParams))
    throw new InputMismatchException(new NodePrinter(list).printedData());

ActivationRecord newARI = new ActivationRecord(lambdaExpression.getParentARI());
ActivationRecord dsrParentARI = ARI;
ARI = newARI;
```

만약 formal보다 actual이 적다면, 예외를 던집니다

```
public class CuteInterpreter {

    private ActivationRecord ARI;
```

인터프리터에서 현재 시점에서 사용하는 ARI는 필드로 가지고 있습니다.

```
public CuteInterpreter() {
    ParserMain.class.getClassLoader();
    view = new ConsoleView();
    input = new ConsoleInput();
    ARI = new ActivationRecord(null);
}
```

인터프리터 생성시 ARI를 만들고 부모를 NULL로 둡니다

```
ActivationRecord newARI = new ActivationRecord(lambdaExpression.getParentARI());
ActivationRecord dsrParentARI = ARI;
ARI = newARI;

updateARI(formalParams, actualParams);
```

새로운 함수를 호출했고, 그 안에 새로운 ARI를 정의해야합니다. 람다 정의할 때 저장해둔 SSR 부모를 가져와서 ARI를 생성하고 그 ARI의 부모로 지정합니다. 현재 ARI는 새로운 람다의 DSR 부모가 됩니다. 임시로 저장을 해두고, 새로운 ARI를 인터프리터의 ARI로 지정합니다. 그리고 formalParam에 actualParam을 바인딩시킵니다.

```
private void updateARI(ListNode formalParams, ListNode actualParams) {
    while(car(formalParams) != null) {
        ARI.insertTable(car(formalParams), car(actualParams));
        formalParams = cdr(formalParams);
        actualParams = cdr(actualParams);
    }
}
```

updateARI에서 바인딩을 해줍니다

```
ListNode expr = listCar(functionBody);
functionBody = cdr(functionBody);
Node res = null;
try {
    while(expr != ListNode.ENDLIST) {
        if((res = runList(expr)) != null)
            break;
        expr = listCar(functionBody);
        functionBody = cdr(functionBody);
    }
} catch(Exception e) {
    ARI = dsrParentARI;
    throw new IllegalArgumentException(new NodePrinter(((ListNode)expr)).printedData());
}

ARI = dsrParentARI;
return res;
```

그리고 함수 바디를 하나씩 가져옵니다

함수는 (func_name (formal_param) (define) (define) ... (expr)) 형태를 가집니다. 함수 바디는 ((define) (define) ... (expr)) 이 부분이므로, 하나씩 가져와서 리스

트를 실행시킵니다. 실행한 결과를 계속 res에 저장하면서 null이라면 정의이므로 계속 진행합니다. 리턴 값이 있다면, 더이상 반복하지 않고 나옵니다. 그리고 나서 ARI를 현재 ARI의 부모 ARI로 바꾸어주면 ARI 복원이 끝나고, 함수 수행 결과를 돌려주면 됩니다.

실행 결과
람다 구현

```
… Welcome to Scheme Console!
… Project From 2020 Spring Programming Language 101 Lecture.
… by 201602004 ParkTaehyun.
…
> ( ( lambda ( x ) ( + x 1 ) ) 1 )
… 2
```

함수 define 구현 및 전역 함수 호출

```
> ( define func ( lambda ( x ) ( * x 2 ) ) )
> func
… #<Closure>
> ( func 20 )
… 40
```

함수 내에서 전역 함수 호출

```
( define func2 ( lambda ( y ) ( + ( func 1 ) y ) ) )
> ( func2 2 )
… 4
>
```

변수 scope 구현

```
…
… Welcome to Scheme Console!
… Project From 2020 Spring Programming Language 101 Lecture.
… by 201602004 ParkTaehyun.
…
> ( define func ( lambda ( x ) ( define y 3 ) ( + x y ) ) )
> ( func 3 )
… 6
> x
… Illegal Token : x
> y
… Illegal Token : y
>
```

함수 수행 후 내부 변수를 인식하지 못하는 것을 확인했습니다

중첩 함수 구현

```
> ( define func ( lambda ( y ) ( define func2 ( lambda ( y ) ( * y 3 ) ) ) ( + 3 ( func2 y ) ) ) )
> ( func 3 )
… 12
> y
… Illegal Token : y
> func2
… Illegal Token : func2
>
```

중첩된 함수를 외부에서 인식하지 못하는 것을 확인했습니다

재귀 함수 구현

```
> ( define lastitem ( lambda ( ls ) ( cond ( ( null? ( cdr ls ) ) ( car ls ) ) ( #T ( lastitem ( cdr ls ) ) ) ) ) )
> ( lastitem ' ( 1 2 3 4 ) )
… 4
> ( lastitem ' ( 1 2 3 asdf ) )
… asdf
> ( lastitem ' ( 1 2 ( a ( 4 ) ) ) )
… ( a ( 4 ) )
>
```

예제로 주어진 재귀함수를 이용해보았습니다

함수를 인자로 사용 구현

```
> ( define compose ( lambda ( func1 func2 x ) ( func1 ( func2 x ) ) ) )
> ( define f ( lambda ( x ) ( + x 1 ) ) )
> ( define g ( lambda ( x ) ( * x 2 ) ) )
> ( compose f g 1 )
… 3
```

함수를 함수 인자로 넘겨서 처리할 수 있는 것을 확인했습니다

```java
/* Item 3 */
@Test
public void lambdaTest() {
    assertEquals("2", computeString("( ( lambda ( x ) ( + x 1 ) ) 1 )"));
    assertEquals("10", computeString("( ( lambda ( x ) ( + ( * x 3 ) 1 ) ) 3 )"));
    assertEquals("13", computeString("( ( lambda ( x z ) ( + ( * x z ) 1 ) ) 3 4 )"));
}

@Test
public void funcDefineAndCallTest() {
    assertEquals("", computeString("( define func ( lambda ( x ) ( + x 1 ) ) )"));
    assertEquals("", computeString("( define func2 ( lambda ( x y ) ( * 2 x y ) ) )"));
    assertEquals("", computeString("( define func3 ( lambda ( x y ) ( * 2 y ( func x ) ) ) )"));
    assertEquals("16", computeString("( func2 4 2 )"));
    assertEquals("2", computeString("( func 1 )"));
    assertEquals("30", computeString("( func3 4 3 )"));
}

@Test
public void localVarTest() {
    assertEquals("", computeString("( define func ( lambda ( x ) ( define y 3 ) ( + x y ) ) )"));
    assertEquals("6", computeString("( func 3 )"));
    assertEquals("#<Closure>", computeString("func"));
    assertEquals(IllegalArgException, computeString("y"));
    assertEquals(IllegalArgException, computeString("x"));
}

@Test
public void nestedFuncTest() {
    assertEquals("", computeString("( define func ( lambda ( y ) ( define func2 ( lambda ( y ) ( * y 3 ) ) ) ( + 3 ( func2 y ) ) ) )"));
    assertEquals("12", computeString("( func 3 )"));
    assertEquals(IllegalArgException, computeString("y"));
    assertEquals(IllegalArgException, computeString("func2"));
}

@Test
public void recursionFuncTest() {
    assertEquals("", computeString("( define lastitem ( lambda ( ls ) ( cond ( ( null? ( cdr ls ) ) ( car ls ) ) ( #T ( lastitem ( cdr ls ) ) ) ) ) )"));
    assertEquals("4", computeString("( lastitem ' ( 1 2 3 4 ) )"));
    assertEquals("asdf", computeString("( lastitem ' ( 1 2 3 asdf ) )"));
    assertEquals("( df )", computeString("( lastitem ' ( 1 2 as ( df ) ) )"));
    assertEquals("( a ( 4 ) )", computeString("( lastitem ' ( 1 2 ( a ( 4 ) ) ) )"));

    assertEquals(IllegalArgException, computeString("ls"));
}


@Test
public void firstObjectFunctionTest() {
    assertEquals("", computeString("( define compose ( lambda ( func1 func2 x ) ( func1 ( func2 x ) ) ) )"));
    assertEquals("", computeString("( define f ( lambda ( x ) ( + x 1 ) ) )"));
    assertEquals("", computeString("( define g ( lambda ( x ) ( * x 2 ) ) )"));
    assertEquals("3", computeString("( compose f g 1 )"));

    assertEquals(IllegalArgException, computeString("x"));
}
```

아이템에 대한 대표적인 테스트 케이스를 돌려보았습니다