実践的プログラミング課題２

C0115349
山本聖樹

概要

今回の課題では、前回の課題で実装した字句解析を基に、構文解析木を内部的に保持するプログラムを実装する。

ソースコード

表 1 に main クラスのソースコードを示す。
ここでは、構文解析木の中でもっとも上層である ProgramNode の isMatch と Parse を行なっている。

表 1　newLang3.java

```
001|//
002|// Masaki Yamamoto (GitHub name: MasakiYamamoto38)
003|// Made in Taiwan
004|//
005|// Di2 ke ti de Source code
006|//
007|
008|package newlang3;
009|
010|import newlang3.Node.Node;
011|import newlang3.Node.ProgramNode;
012|
013|class main
014|{
015|    public static void main(String args[])
016|    {
017|        System.out.println("----- START -----");
018|        // 指定したファイルからテキストを読み込む
019|        try{
020|            String srcText = FileManager.readAll("/Users/mayagalaxy/Desktop/ext1.txt");
021|            char[] srcChars = FileManager.getCharctors(srcText);
022|
023|            LexicalAnalyzerImpl lai = new LexicalAnalyzerImpl(srcChars);
024|            LexicalUnit first = lai.get();
025|
026|            Environment env = new Environment(lai);
027|
028|            ProgramNode ex =  ProgramNode.isMatch(env,first);
029|            if (ex == null)return;
030|            if (ex.Parse())
031|                System.out.println(ex.toString());
032|
033|        }
034|        catch(Exception e)
035|        {
036|            System.out.println(e.getMessage());
037|            return;
038|        }
039|
040|        System.out.println("----- END -----");
041|
042|    }
043|}
```

表 2 に StmtListNode クラスのソースコードを示す。

表 2 StmtListNode.java

```
001|package newlang3.Node;
002|
003|import newlang3.*;
004|import java.util.ArrayList;
005|
006|
007|/* 従属関係 */
008|// ProgramNode
009|//   StmtListNode  <- Ning de uei zhi
010|//     StmtList.StmtNode
011|//     StmtList.BlockNode
012|
013|public class StmtListNode extends Node
014|{
015|    private ArrayList<Node> NodeList = new ArrayList<>();
016|    private Environment env;
017|    private LexicalUnit lu;
018|
019|    public StmtListNode(Environment targetEnv, LexicalUnit targetLu)
020|    {
021|        env = targetEnv;
022|        lu = targetLu;
023|    }
024|
025|    //StmtBlock もしくは BlockNode
026|    public static Node isMatch(Environment env, LexicalUnit lu) {
027|
028|        Node node = BlockNode.isMatch(env, lu);
029|        if (node != null)
030|            return new StmtListNode(env, lu);
031|
032|        node = StmtNode.isMatch(env, lu);
033|        if (node != null)
034|            return new StmtListNode(env, lu);
035|
036|        return null;
037|    }
038|
039|    @Override
040|    public boolean Parse() throws Exception
041|    {
042|        LexicalAnalyzerImpl lai = env.getInput();
043|        NodeList.clear();
044|
046|        while (true)
047|        {
048|            Node sNode;
049|
050|            lu = lai.get();
051|            LexicalType lut = lu.getType();
052|
053|            if (lut == LexicalType.EOF || lu == null ||
054|                    lut == LexicalType.ELSE || lut == LexicalType.ENDIF || lut == LexicalType.LOOP)
055|            {
056|                if(lut == LexicalType.ELSE||lut == LexicalType.ENDIF||lut == LexicalType.LOOP)
057|                {
058|                    lai.unget(lu);
059|                    lai.unget(new LexicalUnit(LexicalType.NL));
060|                }
061|                break;
062|            }
063|            if (lut == LexicalType.NL)
064|                continue;
```

```
065|
066|            sNode = StmtNode.isMatch(env, lu);
067|            if (sNode == null)
068|                sNode = BlockNode.isMatch(env, lu);
069|
070|            if (sNode == null)
071|            {
072|                lai.unget(lu);
073|                return false;
074|            }
075|
076|            lai.unget(lu);
077|            if (!sNode.Parse()) return false;
078|
079|            count += 1;
080|            NodeList.add(sNode);
081|        }
082|
083|        return true;
084|    }
085|
086|    public Value getValue() throws Exception
087|    {
088|        Value val = null;
091|
092|        int count = 0;
093|        while (true)
094|        {
095|            if (NodeList.size() == count)
096|                break;
097|
098|            Node popNode = NodeList.get(count);
099|            count++;
100|            System.out.print(count + "|");
101|            val = popNode.getValue();
102|        }
103|        return val;
104|    }
105|
106|    @Override
107|    public String toString()
108|    {
109|        return "";
110|    }
111|}
```

表 3 に StmtNode クラスのソースコードを示す。

表 3 StmtNode.java

```
001|package newlang3.Node;
002|
003|import newlang3.Environment;
004|import newlang3.LexicalType;
005|import newlang3.LexicalUnit;
006|
007|import java.util.HashSet;
008|import java.util.Set;
009|
010|public class StmtNode extends Node
011|{
012|    Node body;
013|    LexicalUnit lu;
014|
015|    public StmtNode(Environment env,LexicalUnit first)
016|    {
017|        super.env = env;
018|        super.type = NodeType.STMT;
019|        lu = first;
020|    }
021|
022|    private static Set<LexicalType> firstSet = new HashSet<LexicalType>();
023|    static
024|    {
025|        firstSet.add(LexicalType.NAME);
026|        firstSet.add(LexicalType.FOR);
027|        firstSet.add(LexicalType.END);
028|    }
029|
030|    public static Node isMatch(Environment env, LexicalUnit first)
031|    {
032|        if(!firstSet.contains(first.getType()))
033|        {
034|            return null;
035|        }
036|
037|        return new StmtNode(env,first);
038|
039|    }
040|
041|    @Override
042|    public boolean Parse() throws Exception
043|    {
044|        LexicalUnit lu = env.getInput().get();
045|        env.getInput().unget(lu);
046|
047|        body = SubstNode.isMatch(env, lu);
048|        if(body != null) return body.Parse();
049|
050|
051|        if (lu.getType() == LexicalType.END)
052|        {
053|            super.type = NodeType.END;
054|            return true;
055|        }
056|
057|        return false;
058|    }
059|
060|    @Override
061|    public String toString()
062|    {
063|        return "";
064|    }
065|}
```

表 4 に Node クラスのソースコードを示す。

このクラスは、全ての構文解析木のクラスが継承するクラスになっている。

表 4 Node.java

```
001|package newlang3.Node;
002|
003|
004|// 全てのノードクラスが継承するクラス
005|
006|import newlang3.Environment;
007|import newlang3.Value;
008|
009|public class Node {
010|    NodeType type;
011|    Environment env;
012|
013|    /** Creates a new instance of Node */
014|    public Node() {
015|    }
016|    public Node(NodeType my_type) {
017|        type = my_type;
018|    }
019|    public Node(Environment my_env) {
020|        env = my_env;
021|    } //env = 環境
022|
023|    public NodeType getType() {
024|        return type;
025|    }
026|
027|    public boolean Parse() throws Exception
028|    {// 構文の解析に失敗したら boolean で返す
029|        return true;
030|        // 構文解析
031|    }
032|
033|    public Value getValue() throws Exception
034|    {
035|        return null;
036|    }
037|
038|    public String toString()
039|    {
040|        if (type == NodeType.END) return "END";
041|        else return "Node";
042|    }
043|
044|
045|}
```

表 4 に PRINT クラスのソースコードを示す。

getValue メソッドにてコンソールに指定された LexicalUnit データを表示する処理を記述している。

表 4 PrintFunction.java

```
001|package newlang3.Node;
002|
003|import newlang3.LexicalUnit;
004|import newlang3.ValueImpl;
005|
006|//PRINT 関数
007|public class PrintFunction
008|{
009|    public ValueImpl getValue(LexicalUnit lu)
010|    {
011|        System.out.println(lu.toString());
012|        return new ValueImpl(lu.toString());
013|    }
014|}
```