Abstract Syntax Tree Generation

Compiler Design (COMP 442/6421)
Assignment 3

Name: Garvit Kataria
Student Id: 40155647

Attribute Grammar

START -> PROG <prog> .

PROG -> <epsilon>REPTPROG0<STRUCTORIMPLORFUNCLIST> .

REPTPROG0 -> STRUCTORIMPLORFUNC REPTPROG0 .
REPTPROG0 -> .

STRUCTORIMPLORFUNC -> STRUCTDECL <structDecl, 3> .
STRUCTORIMPLORFUNC -> IMPLDEF <implDecl> .
STRUCTORIMPLORFUNC -> FUNCDEF <funcDecl, 3> .

STRUCTDECL -> struct id <<id>> <<epsilon>>OPTSTRUCTDECL2<inherList> lcurbr <<epsilon>>REPTSTRUCTDECL4<membList> rcurbr semi .

OPTSTRUCTDECL2 -> inherits id <<id>> REPTOPTSTRUCTDECL22 .
OPTSTRUCTDECL2 -> .

REPTOPTSTRUCTDECL22 -> comma id <<id>> REPTOPTSTRUCTDECL22 .
REPTOPTSTRUCTDECL22 -> .

REPTSTRUCTDECL4 -> VISIBILITY <<visibility>> MEMBERDECL <membDecl, 2> REPTSTRUCTDECL4 .
REPTSTRUCTDECL4 -> .

MEMBERDECL -> FUNCDECL<funcDecl, 3> .
MEMBERDECL -> VARDECL<varDecl, 3> .

FUNCDECL -> FUNCHEAD semi .

FUNCHEAD -> func id<<id>> lpar <<epsilon>>FPARAMS<fparamList> rpar arrow RETURNTYPE .

RETURNTYPE -> TYPE .
RETURNTYPE -> void <type> .

TYPE -> integer <type> .
TYPE -> float <type> .
TYPE -> id <type> .

FPARAMS -> id<<id>> colon TYPE <<epsilon>>REPTFPARAMS3<dimList> <fparam, 3> REPTFPARAMS4 .
FPARAMS -> .

REPTFPARAMS3 -> ARRAYSIZE REPTFPARAMS3 .
REPTFPARAMS3 -> .

ARRAYSIZE -> lsqbr ARRAYDASH .

ARRAYDASH -> intnum<<num>> rsqbr | rsqbr .

REPTFPARAMS4 -> FPARAMSTAIL<fparam, 3> REPTFPARAMS4 .
REPTFPARAMS4 -> .

Attribute Grammar

FPARAMSTAIL -> comma id<<id>> colon TYPE <<epsilon>>REPTFPARAMSTAIL4<dimList> .

REPTFPARAMSTAIL4 -> ARRAYSIZE REPTFPARAMSTAIL4 .
REPTFPARAMSTAIL4 -> .

VARDECL -> let id <<id>> colon TYPE <type> <<epsilon>>REPTVARDECL4<dimList> semi .

REPTVARDECL4 -> ARRAYSIZE REPTVARDECL4 .
REPTVARDECL4 -> .

IMPLDEF -> impl id <<id>> lcurbr <epsilon>REPTIMPLDEF3<funcDefList> rcurbr .

FUNCDEF -> FUNCHEAD FUNCBODY .

FUNCBODY -> lcurbr <<epsilon>>REPTFUNCBODY1<statementBlock> rcurbr .

REPTFUNCBODY1 -> VARDECLORSTAT REPTFUNCBODY1 .
REPTFUNCBODY1 -> .

VARDECLORSTAT -> VARDECL .
VARDECLORSTAT -> STATEMENT .

STATEMENT -> if lpar RELEXPR rpar then <<epsilon>>STATBLOCK<statementBlock> else <<epsilon>>STATBLOCK<statementBlock> semi <ifStatement, 3> .
STATEMENT -> while lpar RELEXPR rpar <<epsilon>>STATBLOCK<statementBlock> semi <whileStatement, 2> .
STATEMENT -> read lpar VARIABLE rpar semi <readStatement, 1> .
STATEMENT -> write lpar EXPR rpar semi <writeStatement, 1> .
STATEMENT -> return lpar EXPR rpar semi <returnStatement, 1> .
STATEMENT -> <<epsilon>>REPTVARIABLE0 id<<id>><<var0>> STATEMENTDASH .

STATEMENTDASH -> lpar <<epsilon>>APARAMS<aParams> rpar semi <fcallStatement, 2> .
STATEMENTDASH -> <epsilon>REPTVARIABLE2<indiceList> <var, 2> ASSIGNOP EXPR semi <assignStatement, 2> .

STATBLOCK -> lcurbr REPTSTATBLOCK1 rcurbr .
STATBLOCK -> STATEMENT .
STATBLOCK -> .

REPTSTATBLOCK1 -> STATEMENT REPTSTATBLOCK1 .
REPTSTATBLOCK1 -> .

RELEXPR -> ARITHEXPR RELOP ARITHEXPR <relExpr, 3> .

RELOP -> eq <relOp> .
RELOP -> neq <relOp> .
RELOP -> lt <relOp> .
RELOP -> gt <relOp> .
RELOP -> leq <relOp> .
RELOP -> geq <relOp> .

Attribute Grammar

ARITHEXPR -> TERM RIGHTRECARITHEXPR .

RIGHTRECARITHEXPR -> .
RIGHTRECARITHEXPR -> ADDOP TERM <addOp, 2> RIGHTRECARITHEXPR .

TERM -> FACTOR RIGHTRECTERM .

RIGHTRECTERM -> .
RIGHTRECTERM -> MULTOP FACTOR <multOp, 2> RIGHTRECTERM .

ADDOP -> plus .
ADDOP -> minus .
ADDOP -> or .

==========This is just alot of expr
APARAMS -> EXPR REPTAPARAMS1 .
APARAMS -> .

REPTAPARAMS1 -> APARAMSTAIL REPTAPARAMS1 .
REPTAPARAMS1 -> .

APARAMSTAIL -> comma EXPR .
==========

ASSIGNOP -> equal .

EXPR -> ARITHEXPR EXPRDASH .
EXPRDASH -> RELOP ARITHEXPR <relExp, 3> | .

FACTOR -> <<epsilon>><<epsilon>>REPTVARIABLE0 id<<id>><<var0>> FACTORDASH .
FACTOR -> intlit <intlit> .
FACTOR -> floatlit <floatlit>.
FACTOR -> lpar ARITHEXPR rpar .
FACTOR -> not FACTOR <not, 1> .
FACTOR -> SIGN FACTOR <sign, 1> .

FACTORDASH -> <epsilon>REPTVARIABLE2<indiceList> <dataMember, 2> .
FACTORDASH -> lpar <<epsilon>>APARAMS<aParams> <functionCall, 2> .

Attribute Grammar

REPTVARIABLE2 -> INDICE REPTVARIABLE2 .
REPTVARIABLE2 -> .

INDICE -> lsqbr ARITHEXPR rsqbr .

MULTOP -> mult .
MULTOP -> div .
MULTOP -> and .

REPTIMPLDEF3 -> FUNCDEF <funcDef, 4> REPTIMPLDEF3 .
REPTIMPLDEF3 -> .

SIGN -> plus .
SIGN -> minus .

VARIABLE -> <<epsilon>> <<epsilon>>REPTVARIABLE0 id<<id>><<var0>> <epsilon>REPTVARIABLE2<indiceList><variable> .

REPTVARIABLE0 -> IDNEST REPTVARIABLE0 .
REPTVARIABLE0 -> .

IDNEST -> id<<id>> IDNESTDASH .

IDNESTDASH -> <<epsilon>>REPTIDNEST1<indiceList> dot <<dot>, 2> .
IDNESTDASH -> lpar <<epsilon>>APARAMS<aParams> rpar dot <<dot>, 2> .

REPTIDNEST1 -> INDICE REPTIDNEST1 .
REPTIDNEST1 -> .

VISIBILITY -> public .
VISIBILITY -> private .

Design

```
Node root = null;
static Stack<Node> st = new Stack<>();

public boolean pushInStack(String label) {
    st.push(new Node(label));
    return true;
}
```

The Abstract Syntax Tree is generated during the parsing of the src file with the help of stack

pushInStack pushes the node into the stake for further processing

popFromStack takes noOfChildren argument into account while posing the children of the parent node in the AST

```
public boolean popFromStack(String label, int noOfChildren) {
    Node parent = new Node(label);
    System.out.print ("popFromStack: "+ label+": ");
    while(noOfChildren>0 && st.size ()>0) {
        parent.Children.add(st.peek());
        System.out.print (st.peek().label+" ");
        st.pop();
        noOfChildren--;
    }
    System.out.println ();
    st.push(parent);
    return true;
}
```

Design

```java
public boolean popFromStackUntilEpsilon(String label) {
    Node parent = new Node(label);
    System.out.print ("popFromStackUntilEpsilon: "+ label+": ");
    while(!st.peek().label.equals("epsilon")) {
        parent.Children.add(st.peek());
        System.out.print (st.peek().label+" ");
        st.pop();
    }
    st.pop();
    System.out.println ();
    st.push(parent);
    return true;
}
```

popFromStackUntilEpsilon removes the nodes from the stack until encounters an epsilon node.

Design

```
private boolean FUNCBODY()
{
    HashSet<String> firstSet = new HashSet<> (Arrays.asList(new String[]{"opencubr"}));
    HashSet<String> followSet = new HashSet<> (Arrays.asList(new String[]{}));
    if(!skipErrors(firstSet, followSet, epsilon: false)) return false;
    boolean error = false;
    if (lookahead.equals("opencubr"))
    {
        if (Match( v: "opencubr") && pushInStack( label: "epsilon") && REPTFUNCBODY1()
                && popFromStackUntilEpsilon( label: "statementBlock") && Match( v: "closecubr"))
            writeOutDerivation( str: "FUNCBODY -> lcurbr REPTFUNCBODY1 rcurbr ");
        else
            error = true;
    }
    else
        error = true;

    return !error;
}
```

Example of FUNCBODY attributes grammar shown here.

Use Of Tools

For correctness analysis, print all the Composite Semantic Concepts and Atomic Semantic Concepts. Whenever there is a parent node generated in the Abstract Syntax Tree, all the children and printed to check the correctness of the tree.

```
popFromStack: funcDecl: type fparamList id
popFromStack: membDecl: funcDecl visibility
popFromStackUntilEpsilon: dimList:
popFromStack: fparam: dimList type id
popFromStackUntilEpsilon: fparamList: fparam
popFromStack: funcDecl: type fparamList id
popFromStack: membDecl: funcDecl visibility
popFromStackUntilEpsilon: memberList: membDecl membDecl membDecl membDecl membDecl
popFromStack: structDecl: memberList inheritList id
popFromStackUntilEpsilon: dimList:
popFromStack: fparam: dimList type id
popFromStackUntilEpsilon: fparamList: fparam
popFromStack: returnStatement: intlit
popFromStackUntilEpsilon: statementBlock: returnStatement
popFromStack: funcDef: statementBlock type fparamList id
popFromStackUntilEpsilon: funcDefList: funcDef
```

This method was effective to check small blocks of attribute grammar implementation during the development process.

Use Of Tools

No use of 3rd party tools for this assignment.