Adrien Tremblay
40108982
March 5, 2023
Winter 2023

SOEN442 - Assignment 3

Section 1 - Attribute Grammar

The following is the attribute grammar

Adrien Tremblay
40108982
March 5, 2023
Winter 2023

```
     <START> -> <PROG> 'endoffile'

   <ADDOP> -> 'plus' !makeAdditionOperation
   <ADDOP> -> 'minus' !makeSubtractionOperation
   <ADDOP> -> 'or'

   <APARAMS> -> <EXPR> <REPTAPARAMS1>
   <APARAMS> -> EPSILON

   <APARAMSTAIL> -> 'comma' <EXPR>

   <ARITHEXPR> -> <TERM> <RIGHTRECARITHEXPR>

   <ARRAYSIZE> -> 'opensqbr' <ARRAYSIZE2>

   <ARRAYSIZE2> -> 'intnum' !addIntToTop 'closesqbr'
   <ARRAYSIZE2> -> 'closesqbr'

   <EXPR> -> <ARITHEXPR> <EXPR2>

   <EXPR2> -> <RELOP> <ARITHEXPR>
   <EXPR2> -> EPSILON

   <FACTOR> -> 'id' !makeIdentifier <FACTOR2>
      <REPTVARIABLEORFUNCTIONCALL>
   <FACTOR> -> 'intnum' !makeInt
   <FACTOR> -> 'floatnum'
   <FACTOR> -> 'openpar' <ARITHEXPR> 'closepar'
   <FACTOR> -> 'not' <FACTOR>
   <FACTOR> -> <SIGN> <FACTOR>

   <FACTOR2> -> 'openpar' <APARAMS> 'closepar'
   <FACTOR2> -> <REPTIDNEST1>

   <FPARAMS> -> 'id' !makeIdentifier 'colon' <TYPE> !makeType
      !makeVariableDeclaration <REPTFPARAMS3> !closeDims
      <REPTFPARAMS4>
   <FPARAMS> -> EPSILON

   <FPARAMSTAIL> -> 'comma' 'id' !makeIdentifier 'colon' <TYPE>
      !makeType !makeVariableDeclaration <REPTFPARAMSTAIL4>
      !closeDims

   <FUNCBODY> -> !makeStatementBlock 'opencurbr'
      <REPTLOCALVARORSTAT> 'closecurbr'
```

Adrien Tremblay
40108982
March 5, 2023
Winter 2023

```
<MEMBERFUNCDECL> -> <MEMBERFUNCHEAD> !makeFunctionDeclaration
   'semi'

<MEMBERFUNCHEAD> -> 'function' 'id' !makeIdentifier
   !makeParameterList 'colon' 'openpar' <FPARAMS> 'closepar'
   'arrow' <RETURNTYPE> !makeType
<MEMBERFUNCHEAD> -> 'constructorkeyword' 'colon' 'openpar'
   <FPARAMS> 'closepar'

<FUNCDEF> -> <FUNCHEAD> <FUNCBODY> !makeFunctionDefinition

<FUNCHEAD> -> 'function' 'id' !makeIdentifier <FUNCHEADTAIL>
<FUNCHEADTAIL> -> 'sr' <FUNCHEADMEMBERTAIL>
<FUNCHEADTAIL> -> 'openpar' !makeEmptyScopeSpecification
   !makeParameterList <FPARAMS> 'closepar' 'arrow' <RETURNTYPE>
   !makeType
<FUNCHEADMEMBERTAIL> -> 'id' !makeScopeSpecification 'openpar'
   !makeParameterList <FPARAMS> 'closepar' 'arrow' <RETURNTYPE>
   !makeType
<FUNCHEADMEMBERTAIL> -> 'constructorkeyword'
   !makeScopeSpecification 'openpar' !makeParameterList
   <FPARAMS> 'closepar'

<STATEMENT> -> 'id' !makeIdentifier <STATEMENTIDNEST> 'semi'
<STATEMENT> -> 'if' 'openpar' <RELEXPR> 'closepar' 'then'
   <STATBLOCK> 'else' <STATBLOCK> !makeIfStatement 'semi'
<STATEMENT> -> 'while' 'openpar' <RELEXPR> 'closepar'
   <STATBLOCK> 'semi'
<STATEMENT> -> 'read' 'openpar' <VARIABLE> 'closepar' 'semi'
<STATEMENT> -> 'write' 'openpar' <EXPR> 'closepar' 'semi'
<STATEMENT> -> 'return' 'openpar' <EXPR> 'closepar' 'semi'

<STATEMENTIDNEST> -> 'dot' 'id' <STATEMENTIDNEST>
<STATEMENTIDNEST> -> 'openpar' <APARAMS> 'closepar'
   <STATEMENTIDNEST2>
<STATEMENTIDNEST> -> <INDICE> <REPTIDNEST1> <STATEMENTIDNEST3>
<STATEMENTIDNEST> -> 'assign' <EXPR> !makeAssExpr

<STATEMENTIDNEST2> -> EPSILON
<STATEMENTIDNEST2> -> 'dot' 'id' <STATEMENTIDNEST>

<STATEMENTIDNEST3> -> 'assign' <EXPR>
<STATEMENTIDNEST3> -> 'dot' 'id' <STATEMENTIDNEST>

<REPTIDNEST1> -> <INDICE> <REPTIDNEST1>
<REPTIDNEST1> -> EPSILON
```

Adrien Tremblay
40108982
March 5, 2023
Winter 2023

```
<REPTVARIABLEORFUNCTIONCALL> -> <IDNEST>
   <REPTVARIABLEORFUNCTIONCALL>
<REPTVARIABLEORFUNCTIONCALL> -> EPSILON

<IDNEST> -> 'dot' 'id' <IDNEST2>
<IDNEST2> -> 'openpar' <APARAMS> 'closepar'
<IDNEST2> -> <REPTIDNEST1>

<INDICE> -> 'opensqbr' <ARITHEXPR> 'closesqbr'

<MEMBERDECL> -> <MEMBERFUNCDECL>
<MEMBERDECL> -> <MEMBERVARDECL>

<MEMBERVARDECL> -> 'attribute' 'id' !makeIdentifier 'colon'
   <TYPE> !makeType !makeVariableDeclaration <REPTARRAYSIZE>
   !closeDims 'semi'

<MULTOP> -> 'mult' !makeMultiplyOperation
<MULTOP> -> 'div' !makeDivisionOperation
<MULTOP> -> 'and'

<OPTINHERITS> -> 'isa' 'id' !makeIdentifier !consume
   <REPTINHERITSLIST>
<OPTINHERITS> -> EPSILON

<REPTINHERITSLIST> -> 'comma' 'id' !makeIdentifier !consume
   <REPTINHERITSLIST>
<REPTINHERITSLIST> -> EPSILON

<PROG> -> <REPTPROG0> !makeProgram

<RELEXPR> -> <ARITHEXPR> <RELOP> !makeRelativeOperator
   <ARITHEXPR> !makeRelativeExpression

<RELOP> -> 'eq'
<RELOP> -> 'neq'
<RELOP> -> 'lt'
<RELOP> -> 'gt'
<RELOP> -> 'leq'
<RELOP> -> 'geq'

<REPTAPARAMS1> -> <APARAMSTAIL> <REPTAPARAMS1>
<REPTAPARAMS1> -> EPSILON

<REPTFPARAMS3> -> <ARRAYSIZE> <REPTFPARAMS3>
<REPTFPARAMS3> -> EPSILON
```

Adrien Tremblay
40108982
March 5, 2023
Winter 2023

```
<REPTFPARAMS4> -> <FPARAMSTAIL> <REPTFPARAMS4>
<REPTFPARAMS4> -> EPSILON

<REPTFPARAMSTAIL4> -> <ARRAYSIZE> <REPTFPARAMSTAIL4>
<REPTFPARAMSTAIL4> -> EPSILON

<REPTPROG0> -> <CLASSDECLORFUNCDEF> <REPTPROG0>
<REPTPROG0> -> EPSILON

<CLASSDECLORFUNCDEF> -> <CLASSDECL>
<CLASSDECLORFUNCDEF> -> <FUNCDEF>

<REPTSTATBLOCK1> -> <STATEMENT> <REPTSTATBLOCK1>
<REPTSTATBLOCK1> -> EPSILON

<REPTMEMBERDECL> -> <VISIBILITY> <MEMBERDECL> <REPTMEMBERDECL>
<REPTMEMBERDECL> -> EPSILON

<REPTARRAYSIZE> -> <ARRAYSIZE> <REPTARRAYSIZE>
<REPTARRAYSIZE> -> EPSILON

<RETURNTYPE> -> <TYPE>
<RETURNTYPE> -> 'void'

<RIGHTRECARITHEXPR> -> <ADDOP> <TERM> !addFactorToOp
   <RIGHTRECARITHEXPR>
<RIGHTRECARITHEXPR> -> EPSILON

<RIGHTRECTERM> -> <MULTOP> <FACTOR> !addFactorToOp
   <RIGHTRECTERM>
<RIGHTRECTERM> -> EPSILON

<SIGN> -> 'plus'
<SIGN> -> 'minus'

<STATBLOCK> -> 'opencurbr' !makeStatementBlock <REPTSTATBLOCK1>
   'closecurbr'
<STATBLOCK> -> <STATEMENT>
<STATBLOCK> -> EPSILON

<CLASSDECL> -> 'class' 'id' !makeIdentifier !makeInheritanceList
   <OPTINHERITS> 'opencurbr' !makeMemberList <REPTMEMBERDECL>
   'closecurbr' !makeClassDeclaration 'semi'

<TERM> -> <FACTOR> <RIGHTRECTERM>

<TYPE> -> 'integer'
```

Adrien Tremblay
40108982
March 5, 2023
Winter 2023

```
<TYPE> -> 'float'
<TYPE> -> 'id'

<LOCALVARDECL> -> 'localvar' 'id' !makeIdentifier 'colon' <TYPE>
   !makeType !makeVariableDeclaration <ARRAYOROBJECT> !closeDims
   'semi'
<ARRAYOROBJECT> -> <REPTARRAYSIZE>
<ARRAYOROBJECT> -> 'openpar' <APARAMS> 'closepar'

<REPTLOCALVARORSTAT> -> <LOCALVARORSTAT> <REPTLOCALVARORSTAT>
<REPTLOCALVARORSTAT> -> EPSILON

<LOCALVARORSTAT> -> <LOCALVARDECL>
<LOCALVARORSTAT> -> <STATEMENT>

<VARIABLE> -> 'id' <VARIABLE2>

<VARIABLE2> -> <REPTIDNEST1> <REPTVARIABLE>
<VARIABLE2> -> 'openpar' <APARAMS> 'closepar' <VARIDNEST>

<REPTVARIABLE> -> <VARIDNEST> <REPTVARIABLE>
<REPTVARIABLE> -> EPSILON

<VARIDNEST> -> 'dot' 'id' <VARIDNEST2>
<VARIDNEST2> -> 'openpar' <APARAMS> 'closepar' <VARIDNEST>
<VARIDNEST2> -> <REPTIDNEST1>

<VISIBILITY> -> 'public'
<VISIBILITY> -> 'private'
```

Note that the non-terminals are surrounded by pointy brackets (< >). Terminals are surrounded by single quotation marks (‘ ‘). The semantic actions are prefaced with the exclamation mark (!) symbol.

As mentioned in the design document for Assignment 2, my parser works by generating the first and follow sets in code by reading the grammar file (located in grammar/grammar.grm). So I added code to ignore these words in the grammar only for the first and follow set generation steps. But they are present In the cells of the final grammar table.

A detailed description in code of what each semantic action does can be found from line 80 onwards in the Parser.java file. The below image is a screenshot of this file. The switch statement continues down for a lot more cases than is included in this screenshot.

Adrien Tremblay
40108982
March 5, 2023
Winter 2023

```java
String semanticAction = top.substring(1, top.length());
switch (semanticAction) {
    case "makeProgram": // todo: temp implementation
       while (!semanticStack.isEmpty())
            program.addChild(semanticStack.pop());
       break;
    case "makeIdentifier":
        SemanticConcept identifier = new Identifier(lastToken);
        semanticStack.add(identifier);
        break;
    case "makeAssExpr":
        AssignmentStatement assignmentStatement = new AssignmentStatement(semanticStack.pop(), semanticStack.pop());
        semanticStack.peek().addChild(assignmentStatement);
        break;
    case "makeInt":
        Num number = new Num(lastToken);
        semanticStack.push(number);
        break;
    case "makeMultiplyOperation":
        semanticStack.push(new MultiplyOperation(semanticStack.pop()));
        break;
    case "addFactorToOp":
        SemanticConcept factor = semanticStack.pop();
        semanticStack.peek().addChild(factor);
        break;
    case "makeAdditionOperation":
        semanticStack.push(new AdditionOperation(semanticStack.pop()));
        break;
    case "makeSubtractionOperation":
        semanticStack.push(new SubtractionOperation(semanticStack.pop()));
        break;
    case "makeDivisionOperation":
        semanticStack.push(new DivisionOperation(semanticStack.pop()));
        break;
    case "makeType":
```

The vast majority of theses semantic actions are intuitively described by their names. They simply *make* a node of a particular semantic concept and push it to the semantic stack. Often, the this semantic concept node will be passed other semantic concepts in it's constructor by repeatedly calling pop() on the semantic stack. This is a means of propagating information up the tree. See "makeDivisionOperation" for an example of this.

Other times the last processed token is given in the constructor and stored in the semantic concept node. See "makeInt" for an example of this.

Another *hacky* technique I used is that in some semantic actions, instead of adding the newly created semantic concept node to the semantic stack, I add it to the top node of the stack's children. This allows me to avoid complicated logic involved special tokens added to the stack as seen in the slide sets. See "makeFunctionDeclaration" for an example of this.

Section 2 - Design

The following attributes were added to the Parser class.

Adrien Tremblay
40108982
March 5, 2023
Winter 2023

```
7 usages
private FoundToken lastToken;
5 usages
private Program program;
3 usages
private FunctionDefinitionList functionDefinitionList;
3 usages
private ClassDeclarationList classDeclarationList;
54 usages
private Stack<SemanticConcept> semanticStack;
```

lastToken is simply used to store the last seen token and is updated whenever Lexer.nextToken() is called. *Program, FunctionDefenitionList, and ClassDeclarationList* are all semantic concept objects. Program is maintained to keep root of the tree. The reason why variables for the functionDefintion and classDeclarationList are maintained is because in the grammar any sequence of these can be added in any order to the program. So when one of these is encountered, it is manually added to these semantic concept nodes. This was the simplest way I found to implement this behavior. See the below as an example:

```
case "makeFunctionDefinition":
    functionDefinitionList.addChild(new FunctionDefinition(semanticStack.pop(), semanticStack.pop(), semanticStack.pop(), semanticStack.pop(), semanticStack.pop()));
    break;
```

And then finally *semanticStack* represents the semantic stack for the the ast generation.

Below is a screenshot of the top of the parse() method of the Parse class. Program, functionDefinitionList, and program, and the semanticStack. The major addition to this method is starting on line 78 there is an if block that checks if the item in the parseStack is a semantic action. If so, then it is processed and then popped. The processing of the semantic actions consists of one giant switch statement as previously explained.

Adrien Tremblay
40108982
March 5, 2023
Winter 2023

```java
public boolean parse() {
    System.out.println("Starting Parse.");

    program = new Program();
    functionDefinitionList = new FunctionDefinitionList();
    classDeclarationList = new ClassDeclarationList();
    program.addChild(functionDefinitionList);
    program.addChild(classDeclarationList);

    parseStack = new Stack<String>();

    parseStack.push( item: "$");
    parseStack.push( item: "<START>");

    foundToken = lexer.nextToken();

    semanticStack = new Stack<SemanticConcept>();

    String top;
    while ((top = parseStack.peek()) != "$") {
        // skips
        if (foundToken.getToken() == Token.IN_LINE_COMMENT || foundToken.getToken() == Token.BLOCK_COMMENT) {
            foundToken = lexer.nextToken();
            continue;
        }

        if (top.equals("EPSILON")) {
            parseStack.pop();
            continue;
        }

        // checking for semantic actions
        if (top.length() >= 1 && top.charAt(0) == '!') {
            String semanticAction = top.substring(1, top.length());
            switch (semanticAction) {
                case "makeProgram": // todo: temp implementation
                    while (!semanticStack.isEmpty())
                        program.addChild(semanticStack.pop());
```

Below is a picture of the abstract SemanticConcept class that represents a node in the AST. All of semantic concept classes extend it. It contains a set of children and neighbors. As well as an option member attribute that holds a found Token.

Adrien Tremblay
40108982
March 5, 2023
Winter 2023

```
24 inheritors   Adrien Tremblay
public abstract class SemanticConcept {
    3 usages
    private Set<SemanticConcept> children;
    2 usages
    private Set<SemanticConcept> neighbors;
    6 usages
    protected FoundToken member = null;

    Adrien Tremblay
    public SemanticConcept() {
        children = new HashSet<SemanticConcept>();
        neighbors = new HashSet<SemanticConcept>();
    }

    24 implementations   Adrien Tremblay
    public abstract String getName();

    3 usages   Adrien Tremblay
    public FoundToken getMember() {
        return member;
    }

    1 usage   Adrien Tremblay
    public Set<SemanticConcept> getChildren() {
        return children;
    }

    Adrien Tremblay
    public Set<SemanticConcept> getNeighbors() {
        return neighbors;
    }

    Adrien Tremblay
    public void addChild(SemanticConcept child) { children.add(child); }
}
```

Adrien Tremblay
40108982
March 5, 2023
Winter 2023

Another new class I added in the AstPrinter class that takes the root of the completed AST and prints it to a .dot.outast file.

```
1 usage   Adrien Tremblay
public void writeTree(SemanticConcept root) {
    writeNode(root);
    cleanup();
}

2 usages   Adrien Tremblay
private int writeNode(SemanticConcept node) {
    int myId = nextNodeId;
    try {
        if (node.getMember() == null)
            dotFileWriter.write( str: (nextNodeId++) + "[label=\"" + node.getName() +"\"];\n");
        else
            dotFileWriter.write( str: (nextNodeId++) + "[label=\"" + node.getName() + " | " + ((!(node instanceof RelativeOperator)) ? node.getMember().getLexeme() : node.getMember()

        for (SemanticConcept child : node.getChildren())
            dotFileWriter.write( str: myId + "->" + writeNode(child) + "\n");
    } catch (IOException e) {
        e.printStackTrace();
    }
    return myId;
}
```

There are minor changes of note in the GrammarTableGenerator class to ignore the semantic actions which are present in the grammar for the generation of the first and follow sets.

Section 3 - Use of Tools

Only one tool was used for this assignment, albeit indirectly. The xdot program was used to visualize the .dot.outast Abstract Syntax Tree files.