**Faculty of Computing**

**SE-314: Software Construction**

**Class: BESE 13AB**

# Lab 10: Representing Expression

**CLO-03:** Design and develop solutions based on Software Construction principles.  
**CLO-04:** Use modern tools such as Eclipse, NetBeans etc. for software construction.

**Mati ur Rehman**

**419956**

**Instructor: Dr. Mehvish Rashid  
Lab Engineer: Mr. Aftab Farooq**

# Lab 10

**Lab Tasks**

Code:

/\* Copyright (c) 2015-2016 MIT 6.005 course staff, all rights reserved.

\* Redistribution of original or derived work requires permission of course staff.

\*/

package expressivo;

import org.antlr.v4.runtime.CommonTokenStream;

import org.antlr.v4.runtime.ParserRuleContext;

import org.antlr.v4.runtime.TokenStream;

import org.antlr.v4.runtime.misc.ParseCancellationException;

import org.antlr.v4.runtime.tree.ErrorNode;

import org.antlr.v4.runtime.tree.ParseTree;

import org.antlr.v4.runtime.tree.ParseTreeWalker;

import org.antlr.v4.runtime.tree.TerminalNode;

import org.antlr.v4.runtime.ANTLRInputStream;

import org.antlr.v4.runtime.CharStream;

import java.util.Collection;

import java.util.Collections;

import java.util.Map;

import java.util.Stack;

import expressivo.parser.ExpressionLexer;

import expressivo.parser.ExpressionListener;

import expressivo.parser.ExpressionParser;

import expressivo.parser.ExpressionParser.ExpressionContext;

import expressivo.parser.ExpressionParser.MultiplicationContext;

import expressivo.parser.ExpressionParser.ParenthesesContext;

import expressivo.parser.ExpressionParser.PrimitiveContext;

import expressivo.parser.ExpressionParser.RootContext;

import expressivo.parser.ExpressionParser.SumContext;

/\*\*

\* An immutable data type representing a polynomial expression of:

\* + and \*

\* nonnegative integers and floating-point numbers

\* variables (case-sensitive nonempty strings of letters)

\*

\* <p>PS3 instructions: this is a required ADT interface.

\* You MUST NOT change its name or package or the names or type signatures of existing methods.

\* You may, however, add additional methods, or strengthen the specs of existing methods.

\* Declare concrete variants of Expression in their own Java source files.

\*/

public interface Expression {

// Datatype definition

// Expression = Add(left:Expression, right:Expression) +

// Multiplication(left:Expression, right:Expression) +

// Number(number:String) +

// Variables(variable:String)

/\*\*

\* Parse an expression.

\* @param input expression to parse, as defined in the PS3 handout.

\* @return expression AST for the input

\* @throws IllegalArgumentException if the expression is invalid

\*/

public static Expression parse(String input) {

try {

ParseTree tree = makeParser(input).root();

ParseTreeWalker walker = new ParseTreeWalker();

MakeExpression listener = new MakeExpression();

walker.walk(listener, tree);

return listener.getExpression();

} catch (ParseCancellationException e) {

throw new IllegalArgumentException(input);

}

}

/\*\*

\* @return a parsable representation of this expression, such that

\* for all e:Expression, e.equals(Expression.parse(e.toString())).

\* For a Expression is not a primitive, subExpression will be surround by

\* parentheses for grouping, and whitespace will be place before and after operator.

\* Subexpression will be group from right to left.

\* If Expression is a number, output an equivalent number, accurate to at least 4 decimal places,

\*/

@Override

public String toString();

/\*\*

\* @param thatObject any object

\* @return true if and only if this and thatObject are structurally-equal

\* Expressions, as defined in the PS3 handout.

\*/

@Override

public boolean equals(Object thatObject);

/\*\*

\* @return hash code value consistent with the equals() definition of structural

\* equality, such that for all e1,e2:Expression,

\* e1.equals(e2) implies e1.hashCode() == e2.hashCode()

\*/

@Override

public int hashCode();

// TODO more instance methods

/\*\*

\* @return true is this Expression is a primitive element

\*/

public boolean isPrimitive();

/\*\*

\* Differentiate this expression with respect to variable.

\*

\* @param variable to differentiate by.

\* @return expression's derivative with respect to variable.

\* @throws IllegalArgumentException if the expression or variable is invalid

\*/

public Expression differentiate(String variable);

/\*\*

\* Simplify substitutes the values for those variables into the expression,

\* and attempts to simplfy the substituted polynomial as much as it can.

\* If substituted polynomial is a constant expression, with no variables remaining,

\* then simplification must reduce it to a single number.

\* If substituted polynomials that still contain variables is underdetermined,

\* reduce child polynomials to number if no variables remaining.

\* @param environment mapping of variables to values

\* @return a new simplified Expression left original Expression unmodified.

\*/

public Expression simplify(Map<String, Double> environment);

/\*\*

\* @return true expression is constant otherwise false

\*/

public default boolean constant() {

return false;

}

/\*\*

\* @param that Expression to be add

\* @return a new Expression represent add this Expression with other Expression

\*/

public static Expression add(Expression exp1, Expression exp2) {

return new AdditionExpression(exp1, exp2);

}

/\*\*

\* @param that Expression to be multiplication

\* @return a new Expression represent multiplication this Expression with other Expression

\*/

public static Expression multiplication(Expression exp1, Expression exp2) {

return new MultiplicationExpression(exp1, exp2);

}

/\*\*

\* @param input expression

\* @return ExpressionParser generate from input

\*/

private static ExpressionParser makeParser(String input) {

CharStream stream = new ANTLRInputStream(input);

ExpressionLexer lexer = new ExpressionLexer(stream);

lexer.reportErrorsAsExceptions();

TokenStream tokens = new CommonTokenStream(lexer);

ExpressionParser parser = new ExpressionParser(tokens);

parser.reportErrorsAsExceptions();

return parser;

}

}

class MakeExpression implements ExpressionListener {

private final Stack<Expression> stack = new Stack<>();

/\*\*

\* Returns the expression constructed by this listener object.

\* Requires that this listener has completely walked over an IntegerExpression

\* parse tree using ParseTreeWalker.

\* @return IntegerExpression for the parse tree that was walked

\*/

public Expression getExpression() {

return stack.peek();

}

@Override

public void exitRoot(RootContext ctx) {

assert stack.size() == 1;

}

@Override public void exitSum(SumContext ctx) {

// matched the primitive ('+' primitive)+ rule

assert stack.size() >= 2;

Expression sum = stack.pop();

sum = new AdditionExpression(stack.pop(), sum);

stack.push(sum);

}

@Override public void exitMultiplication(MultiplicationContext ctx) {

// matched the primitive ('+' primitive)+ rule

assert stack.size() >= 2;

Expression mul = stack.pop();

mul = new MultiplicationExpression(stack.pop(), mul);

stack.push(mul);

}

@Override public void exitPrimitive(PrimitiveContext ctx) {

if (ctx.NUMBER() != null) {

stack.push(new NumberExpression(ctx.NUMBER().getText()));

} else if (ctx.VARIABLE() != null) {

stack.push(new VariablesExpression(ctx.VARIABLE().getText()));

} else {

// match parentheses

// parentheses expression is already on the stack

}

}

// don't need these here, so just make them empty implementations

@Override public void enterEveryRule(ParserRuleContext arg0) {}

@Override public void exitEveryRule(ParserRuleContext arg0) {}

@Override public void visitErrorNode(ErrorNode arg0) {}

@Override public void visitTerminal(TerminalNode arg0) {}

@Override public void enterRoot(RootContext ctx) {}

@Override public void enterExpression(ExpressionContext ctx) {}

@Override public void exitExpression(ExpressionContext ctx) {}

@Override public void enterSum(SumContext ctx) {}

@Override public void enterMultiplication(MultiplicationContext ctx) {}

@Override public void enterPrimitive(PrimitiveContext ctx) {}

@Override public void enterParentheses(ParenthesesContext ctx) {}

@Override public void exitParentheses(ParenthesesContext ctx) {}

}

Screenshots:

