2022-2 프로그래밍언어

PA1 보고서

2019312979 김나영

1) Grammar file

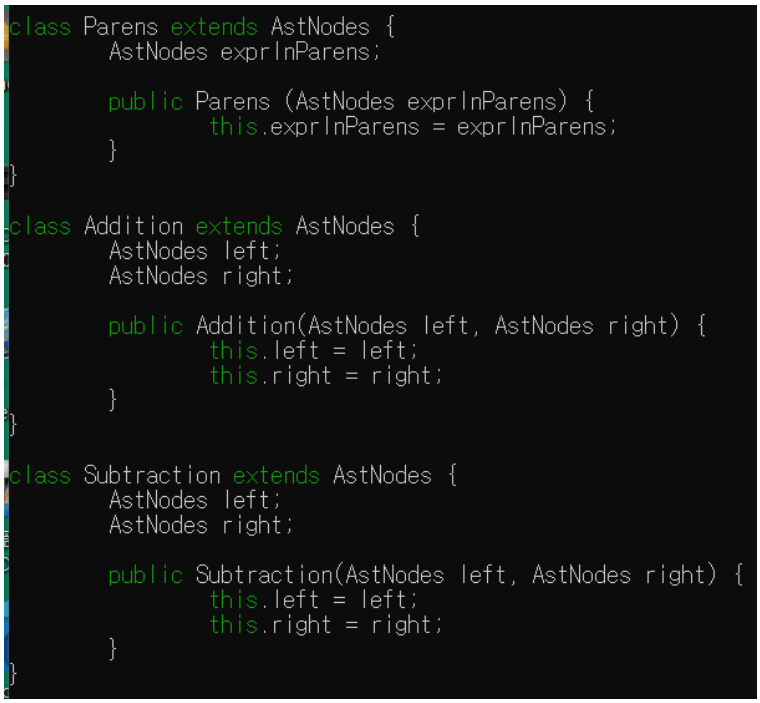
```
grammar Expr;

// parser rules
prog : (expr ';' NEWLINE?)*;
expr :('min' | 'max' | 'pow' | 'sqrt') '(' num ','? num? ')'  # functionCall
     | '(' expr ')'          # parensExpr
     | expr ('*'|'/') expr   # infixExpr
     | expr ('+'|'-') expr   # infixExpr
     | ID '=' num            # varDec
     | num                   # numberExpr
     | ID                      # idExpr
     ;

num  : INT
     | REAL
     ;

// lexer rules
NEWLINE: [\r\n]+ ;
INT: '0' | '-'?[1-9][0-9]* ;         // should handle negatives
REAL: '-'?[0-9]+'.'[0-9]* ; // should handle signs(+/-)
ID: [a-zA-Z]+ ;
FUNCT: 'min' | 'max' | 'pow' | 'sqrt' ;  // java.lang.Math functions
WS: [ \t\r\n]+ -> skip ;
```

Grammar를 위와 같이 함수 호출, 변수 선언 및 음수 handling이 가능하도록 수정.

2) AstNodes.java

```
import java.util.ArrayList;

/*

define Ast Nodes

- Define AST nodes to print
- The nodes have to be defined as class

*/

public class AstNodes{

}

class Prog extends AstNodes{
        public ArrayList<AstNodes> expressions;

        public Prog() {
                this.expressions = new ArrayList<>();
        }

        public void addExpression(AstNodes e) {
                expressions.add(e);
        }

}


class Number extends AstNodes{
        double num;

        public Number(double num) {
                this.num = num;
        }
}
```

```
class VariableDeclaration extends AstNodes {
        public String id;
        public double value;

        public VariableDeclaration(String id, double value) {
                this.id = id;
                this.value = value;
        }
}

class minFunction extends AstNodes {
        double left;
        double right;

        public minFunction(double left, double right) {
                this.left = left;
                this.right = right;
        }
}

class maxFunction extends AstNodes {
        double left;
        double right;

        public maxFunction(double left, double right) {
                this.left = left;
                this.right = right;
        }
}
```



```
class Parens extends AstNodes {
        AstNodes exprInParens;

        public Parens (AstNodes exprInParens) {
                this.exprInParens = exprInParens;
        }
}

class Addition extends AstNodes {
        AstNodes left;
        AstNodes right;

        public Addition(AstNodes left, AstNodes right) {
                this.left = left;
                this.right = right;
        }
}

class Subtraction extends AstNodes {
        AstNodes left;
        AstNodes right;

        public Subtraction(AstNodes left, AstNodes right) {
                this.left = left;
                this.right = right;
        }
}
```

위와 같이 Node들을 클래스로 정의함. 모든 노드들은 AstNodes 클래스를 상속.

최상위 노드인 Prog는 expressions라는 리스트 멤버를 가지는데, 이 리스트에 실제 AstNodes들(Addition, VariableDeclaration, Parens 등)을 저장함. addExpression이라는 메서드로 expressions에 노드를 추가할 수 있도록 구현함. Prog 외의 하위 노드에는 Number, Variable, VariableDeclaration, minFunction, maxFunction, powFunction, sqrtFunction, Parens, Addition, Subtraction, Multiplication, Division이 있음. (위의 캡쳐본에는 이 중 일부만 포함)

3) BuildAstVisitor.java

```java
import java.util.ArrayList;
import java.util.List;

/*

Build Ast using the method in ExprBaseVisitor.java
you should override the methods.

*/


public class BuildAstVisitor extends ExprBaseVisitor<AstNodes>{

	private List<String> vars; // to store variables declared

	public BuildAstVisitor() {
		vars = new ArrayList<>();
	}

	@Override
	public AstNodes visitProg(ExprParser.ProgContext ctx) {
		Prog prog = new Prog();

		// helper visitor for transforming each subtree
		// into an AstNodes object
		BuildAstVisitor exprVisitor = new BuildAstVisitor();

		for (int i = 0;i < ctx.getChildCount(); i++) {
			if (i == ctx.getChildCount() - 1) {
			// do not visit the last child of start symbol prog
			// and attempt to convert it to an AstNodes object
			}
			else {
				prog.addExpression(exprVisitor.visit(ctx.getChild(i)));
			}
		}
		return prog;
	}
```

antlr4 -visitor Expr.g4 command 실행 후 생성된 ExprBaseVisitor.java를 활용(상속).

각각의 visit 메서드 -> ExprParser.java에 정의된 클래스에 상응하는 object를 argument로 넘겨줌(ProgContext, InfixExprContext, NumberContext 등)

BuildAstVisitor 클래스 안에 vars 리스트 멤버를 생성함. 이는 선언되는 변수의 이름을 저장하기 위한 것.

visitProg: Prog 노드의 마지막 child인 '₩n'를 제외한 노드를 expressions 리스트에 추가한 후 prog object를 return.

```
	@Override
	public AstNodes visitInfixExpr(ExprParser.InfixExprContext ctx) {
		AstNodes astNode = new AstNodes();
		AstNodes left = visit(ctx.getChild(0)); // recursively visit the left subtree of the current
InfixExpr node
		AstNodes right = visit(ctx.getChild(2));
		String operator = ctx.getChild(1).getText();

		if (operator.equals("+")) {
			return new Addition(left, right);
		}
		else if (operator.equals("-")) {
			return new Subtraction(left, right);
		}
		else if (operator.equals("*")) {
			return new Multiplication(left, right);
		}
		else if (operator.equals("/")) {
			return new Division(left, right);
		}

		return astNode;
	}
```

visitInfixExpr: InfixExpr는 left, right에 또다른 expression이 올 수 있기 때문에 recursivr하게 노드를 visit -> operator(사칙연산) 종류에 따라 각각 다른 클래스의 노드를 return.

```
	@Override
	public AstNodes visitFunctionCall(ExprParser.FunctionCallContext ctx) {
		AstNodes astNode = new AstNodes();
		String leftText = ctx.getChild(2).getText();
		double left = Double.parseDouble(leftText);
		String rightText;
		double right;

		if (ctx.getChildCount() > 4) {
			rightText = ctx.getChild(4).getText();
			right = Double.parseDouble(rightText);
		}
		else {
			rightText = "";
			right = 0;
		}

		String functionName = ctx.getChild(0).getText();

		if (functionName.equals("min")) {
			return new minFunction(left, right);
		} else if (functionName.equals("max")) {
			return new maxFunction(left, right);
		} else if (functionName.equals("pow")) {
			return new powFunction(left, right);
		} else if (functionName.equals("sqrt")) {
			return new sqrtFunction(left);
		}

		return astNode;
	}
```

visitFunctionCall: 함수 argument 숫자들을 double형으로 바꾼 후 함수 이름에 따라 각각 다른 클래스의 노드를 return.

```java
@Override
public AstNodes visitParensExpr(ExprParser.ParensExprContext ctx) {
        AstNodes exprInParens = visit(ctx.getChild(1));

        return new Parens(exprInParens);
}

@Override
public AstNodes visitNumberExpr(ExprParser.NumberExprContext ctx) {
        String numText = ctx.getChild(0).getText(); // pass index to getChild
        double num = Double.parseDouble(numText);
        return new Number(num);
}

@Override
public AstNodes visitVarDec(ExprParser.VarDecContext ctx) {
        String id = ctx.getChild(0).getText();
        if (!vars.contains(id)) {
                vars.add(id);
        }
        double value = Double.parseDouble(ctx.getChild(2).getText());
        return new VariableDeclaration(id, value);
}

@Override
public AstNodes visitIdExpr(ExprParser.IdExprContext ctx) {
        String id = ctx.getChild(0).getText();
        return new Variable(id);
}
```

나머지 Parens, Number, VarDec(VariableDeclaration), Id expression 노드의 visitor 메서드들은 위와 같이 구현.

4) AstCall.java

```
public class AstCall {

	List<AstNodes> list;
	int nodeCnt = 0;
	String base = String.format("%8s", " ");

	public AstCall(List<AstNodes> list) {
		this.list = list;
	}

	public void PrintNode() {


		for (AstNodes e: list) {
			String result = "";
			if (e instanceof VariableDeclaration) { // e is an instance of VariableDeclaration
				result += CallVarDecNode((VariableDeclaration) e);
			}
			else if (e instanceof Number) {
				result += CallNumNode((Number) e);
			}
			else if (e instanceof Parens) {
				result += CallParensNode((Parens) e);
			}
			else if (e instanceof minFunction) {
				result += CallMinFunct((minFunction) e);
			}
			else if (e instanceof maxFunction) {
				result += CallMaxFunct((maxFunction) e);
			}
			else if (e instanceof powFunction) {
				result += CallPowFunct((powFunction) e);
			}
			else if (e instanceof sqrtFunction) {
				result += CallSqrtFunct((sqrtFunction) e);
			}
			else if (e instanceof Addition) {
				result += CallAddNode((Addition) e);
			}
			else if (e instanceof Subtraction) {
				result += CallSubNode((Subtraction) e);
			}
			else if (e instanceof Multiplication) {
				result += CallMulNode((Multiplication) e);
			}
			else if (e instanceof Division) {
				result += CallDivNode((Division) e);
			}
			else {
				continue;
			}


			System.out.println(result);
			nodeCnt = 0;
		}

	}
```

AstCall Constructor: AstNodes들이 포함된 리스트를 넘겨받음.

PrintNode: 리스트 안의 각각의 node object별 타입을 파악하여 적절한 Call 함수를 호출함. Call 함수는 트리를 순회하며 출력해야 할 문자열을 만들어 return -> result에 이를 저장한 후 출력. 각각의 expression을 구분하는 세미콜론(';')을 만나면 마지막의 else문이 실행되어 아무것도 출력되지 않고 다음 노드로 넘어감.

nodeCnt, base는 트리 레벨별 들여쓰기 레벨을 조절하기 위한 것.

```
private String CallVarDecNode(VariableDeclaration e) {
        VariableDeclaration decl = (VariableDeclaration) e;
        String result = String.format("%-8s₩n%-8s%-8s₩n%-8s%-8s", "ASSIGN", " ", decl.id, " ",
                        decl.value + "");

        return result;
}

private String CallVarNode(Variable e) {
        Variable variable = (Variable) e;
        return variable.id;
}

private String CallNumNode(Number e) {
        Number num = (Number) e;
        return num.num + "";
}

private String CallParensNode(Parens e) {
        Parens parens = (Parens) e;
        AstNodes exprInParens = parens.exprInParens;

        return GetLeftStr(exprInParens);
}
```

VariableDeclaration, Variable, Number는 그 안에 다른 노드를 포함할 수 없음 -> recursive한 구현 필요 X.

Parens 노드는 괄호 안에 다른 expression이 올 수 있으므로, 괄호 안의 expression을 argument로 넘겨 GetLeftStr 함수 호출(recursive)

```
private String CallMinFunct(minFunction e) {
        minFunction minFunct = (minFunction) e;
        String spaces = "";
        nodeCnt++;
        for (int i = 0;i < nodeCnt;i++)
                spaces += base;

        return String.format("%-8s₩n%-8s%-8s₩n%-8s%-8s", "min", spaces, minFunct.left, spaces, minFunct.right);
}

private String CallMaxFunct(maxFunction e) {
        maxFunction maxFunct = (maxFunction) e;
        String spaces = "";
        nodeCnt++;
        for (int i = 0;i < nodeCnt;i++)
                spaces += base;

        return String.format("%-8s₩n%-8s%-8s₩n%-8s%-8s", "max", spaces, maxFunct.left, spaces, maxFunct.right);
}

private String CallPowFunct(powFunction e) {
        powFunction powFunct = (powFunction) e;
        String spaces = "";
        nodeCnt++;
        for (int i = 0;i < nodeCnt;i++)
                spaces += base;

        return String.format("%-8s₩n%-8s%-8s₩n%-8s%-8s", "pow", spaces, powFunct.left, spaces, powFunct.right);
}

private String CallSqrtFunct(sqrtFunction e) {
        sqrtFunction sqrtFunct = (sqrtFunction) e;
        String spaces = "";
        nodeCnt++;
        for (int i = 0;i < nodeCnt;i++)
                spaces += base;

        return String.format("%-8s₩n%-8s%-8s", "sqrt", spaces, sqrtFunct.num);
}
```

Min, max, pow, sqrt는 괄호 안에 argument로 다른 expression이 올 수 없고, 오직 숫자만 올 수 있기 때문에 recursive call 없이 바로 결과 문자열을 return

```
private String CallAddNode(Addition e) {
        Addition add = (Addition) e;
        AstNodes leftNode = add.left;
        AstNodes rightNode = add.right;
        String leftStr = "";
        String rightStr = "";
        String spaces = "";

        nodeCnt++;
        for (int i = 0;i < nodeCnt;i++)
                spaces += base;
        int temp = nodeCnt;
        leftStr = GetLeftStr(leftNode);
        nodeCnt = temp;
        rightStr = GetRightStr(rightNode);

        return String.format("%-8s₩n%-8s%-8s₩n%-8s%-8s", "ADD", spaces, leftStr, spaces, rightStr);
}

private String CallSubNode(Subtraction e) {
        Subtraction sub = (Subtraction) e;
        AstNodes leftNode = sub.left;
        AstNodes rightNode = sub.right;
        String leftStr = "";
        String rightStr = "";
        String spaces = "";

        nodeCnt++;
        for (int i = 0;i < nodeCnt;i++)
                spaces += base;
        int temp = nodeCnt;
        leftStr = GetLeftStr(leftNode);
        nodeCnt = temp;
        rightStr = GetRightStr(rightNode);

        return String.format("%-8s₩n%-8s%-8s₩n%-8s%-8s", "SUB", spaces, leftStr, spaces, rightStr);
}
```

Addition, Subtraction, Multiplication, Division과 같은 사칙연산 expression은 그 안에 또다른 expression을 포함할 수 있기 때문에 leftNode와 rightNode를 argument로 넘겨 GetLeftStr, GetRightStr 함수 호출(recursive). Multiplication, Division은 캡쳐에서 생략. Addition, Subtraction과 동일한 구조.

```
private String GetLeftStr(AstNodes e) {
        AstNodes leftNode = e;
        String leftStr = "";

        if (leftNode instanceof Number) leftStr += CallNumNode((Number) leftNode);
        else if (leftNode instanceof Variable) leftStr += CallVarNode((Variable) leftNode);
        else if (leftNode instanceof Parens) leftStr += CallParensNode((Parens) leftNode);
        else if (leftNode instanceof minFunction) leftStr += CallMinFunct((minFunction) leftNode);
        else if (leftNode instanceof maxFunction) leftStr += CallMaxFunct((maxFunction) leftNode);
        else if (leftNode instanceof powFunction) leftStr += CallPowFunct((powFunction) leftNode);
        else if (leftNode instanceof sqrtFunction) leftStr += CallSqrtFunct((sqrtFunction) leftNode);
        else if (leftNode instanceof Addition) leftStr += CallAddNode((Addition) leftNode);
        else if (leftNode instanceof Subtraction) leftStr += CallSubNode((Subtraction) leftNode);
        else if (leftNode instanceof Multiplication) leftStr += CallMulNode((Multiplication) leftNode);
        else if (leftNode instanceof Division) leftStr += CallDivNode((Division) leftNode);

        return leftStr;
}

private String GetRightStr(AstNodes e) {
        AstNodes rightNode = e;
        String rightStr = "";

        if (rightNode instanceof Number) rightStr += CallNumNode((Number) rightNode);
        else if (rightNode instanceof Variable) rightStr += CallVarNode((Variable) rightNode);
        else if (rightNode instanceof Parens) rightStr += CallParensNode((Parens) rightNode);
        else if (rightNode instanceof minFunction) rightStr += CallMinFunct((minFunction) rightNode);
        else if (rightNode instanceof maxFunction) rightStr += CallMaxFunct((maxFunction) rightNode);
        else if (rightNode instanceof powFunction) rightStr += CallPowFunct((powFunction) rightNode);
        else if (rightNode instanceof sqrtFunction) rightStr += CallSqrtFunct((sqrtFunction) rightNode);
        else if (rightNode instanceof Addition) rightStr += CallAddNode((Addition) rightNode);
        else if (rightNode instanceof Subtraction) rightStr += CallSubNode((Subtraction) rightNode);
        else if (rightNode instanceof Multiplication) rightStr += CallMulNode((Multiplication) rightNode);
        else if (rightNode instanceof Division) rightStr += CallDivNode((Division) rightNode);

        return rightStr;
}
```

사칙연산 및 Parens node의 recursive call(expression 안에 또다른 expression)에서 사용된 GetLeftStr과 GetRightStr 메서드는 위와 같이 구현함.

5) Evaluate.java

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


/*

Calculate The Input String
And Return the result

- Define methods to calculate the expression we get as input
- The name of the method should be "evaluate"

*/
```

노드 calculation 결과를 출력하기 위해서는 변수(variable)에 저장된 값을 알아야 함. 이

를 위해 HashMap 구조를 활용(라이브러리 import).

```
class Evaluate {


	List<AstNodes> list;
	/* Symbol table for storing values of variables */
	public Map<String, Double> values;

	public Evaluate(List<AstNodes> list) {
		this.list = list;
		values = new HashMap<>();
	}

	public List<Double> evaluate() {
		List<Double> evaluations = new ArrayList<>();

		for (AstNodes e: list) {
			if (e instanceof VariableDeclaration) { // e is an instance of VariableDeclaration
				VariableDeclaration decl = (VariableDeclaration) e;
				values.put(decl.id, decl.value);
				evaluations.add(0.0);
			}
			else if (e instanceof AstNodes){ // e is not an instance of VariableDeclaration
				double result = getEvalResult(e);
				evaluations.add(result);
			}
		}

		return evaluations;
	}
```

위와 같이 Evaluate 클래스의 멤버 및 Constuctor를 구현하여 values라는 HashMap 구조에 변수 이름 & 값의 pair를 저장할 수 있도록 하였음.

evaluate 메서드는 list 안의 AstNode마다 getEvalResult 메서드를 호출하여 evaluations 리스트에 계산 결과를 저장하고, 이 evaluations 리스트를 return. 만약 VariableDeclaration 노드인 경우 values에 변수 이름 & 값 pair를 저장하고, evaluations 리스트에는 0.0을 add.

```
private double getEvalResult(AstNodes e) {
        double result = 0;

        if (e instanceof Number) {
                Number num = (Number) e;
                result = num.num;
        }
        else if (e instanceof Variable){
                Variable variable = (Variable) e;
                result = values.get(variable.id);
        }
        else if (e instanceof Parens) {
                Parens parens = (Parens) e;
                result = getEvalResult(parens.exprInParens);
        }
        else if (e instanceof maxFunction) {
                maxFunction maxFunct = (maxFunction) e;
                double left = maxFunct.left;
                double right = maxFunct.right;
                result = Math.max(left, right);
        }
        else if (e instanceof minFunction) {
                minFunction minFunct = (minFunction) e;
                double left = minFunct.left;
                double right = minFunct.right;
                result = Math.min(left, right);
        }
        else if (e instanceof powFunction) {
                powFunction powFunct = (powFunction) e;
                double left = powFunct.left;
                double right = powFunct.right;
                result = Math.pow(left, right);
        }
        else if (e instanceof sqrtFunction) {
                sqrtFunction sqrtFunct = (sqrtFunction) e;
                double num = sqrtFunct.num;
                result = Math.sqrt(num);
        }
        else if (e instanceof Addition) {
                Addition add = (Addition) e;
                double left = getEvalResult(add.left);
                double right = getEvalResult(add.right);
                result = left + right;
        }
        else if (e instanceof Subtraction){
                Subtraction sub = (Subtraction) e;
                double left = getEvalResult(sub.left);
                double right = getEvalResult(sub.right);
                result = left - right;
        }
        else if (e instanceof Multiplication) {
                Multiplication mul = (Multiplication) e;
                double left = getEvalResult(mul.left);
                double right = getEvalResult(mul.right);
                result = left * right;
        }
        else if (e instanceof Division) {
                Division div = (Division) e;
                double left = getEvalResult(div.left);
                double right = getEvalResult(div.right);
                result = left / right;
        }

        return result;

}
```

getEvalResult 메서드는 위와 같이 구현함.

6) program.java

```
import java.io.IOException;

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


/*

- Define the main method in the file

- In the main method,
  1) Build parse tree
  2) Accept input as command line
  3) Call the method as defined (call and evaluate)
  4) Print out the resulting value
  ** Calculation should be in double (5/2 = 2.5, not 2)
  ** ctrl + d after your enter input

*/
```

```
public class program {

    public static void main(String[] args) throws IOException {

        // Get Lexer
        ExprLexer lexer = new ExprLexer(CharStreams.fromStream(System.in));

        // Get a list of matched tokens
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        // Pass tokens to parser
        ExprParser parser = new ExprParser(tokens);

        // write your code here
        // tell ANTLR to build a parse tree
        // parse from the start symbol 'prog'
        ParseTree antlrAST = parser.prog();

        // create a visitor for converting the parse tree
        // into AstNodes objects
        BuildAstVisitor progVisitor = new BuildAstVisitor();
        Prog prog = (Prog) progVisitor.visit(antlrAST);


        // Print out the AST nodes
        AstCall caller = new AstCall(prog.expressions);
        caller.PrintNode();


        // Print out resulting values using evaluate method
        Evaluate eval = new Evaluate(prog.expressions);
        for (Double evaluation: eval.evaluate()) {
                System.out.println(evaluation);
        }
    }
}
```

program의 main 메서드는 위와 같이 구현함. AstNodes.java에서 Prog.expressions 리스트에 각각의 하위 AstNodes를 저장하도록 구현했기 때문에, AstCall과 Evaluate object를 생성할 때는 Prog.expressions를 argument로 넘겨줌.

7) 실행 결과

```
yzl1015@DESKTOP-BTRCQ6V:~/pa_visit9$ vim program.java
yzl1015@DESKTOP-BTRCQ6V:~/pa_visit9$ javac *.java
yzl1015@DESKTOP-BTRCQ6V:~/pa_visit9$ java program
a = 3; a + pow(3, 2); a / 2 + (2 * 3 + 5);
ASSIGN
        a
        3.0
ADD
        a
        pow
                3.0
                2.0
ADD
        DIV
                a
                2.0
        ADD
                MUL
                        2.0
                        3.0
                5.0
0.0
12.0
12.5
yzl1015@DESKTOP-BTRCQ6V:~/pa_visit9$ java program
a = 10; b = 5; sqrt(16) / 2 + ((3 + 4) * 2) - -1;
ASSIGN
        a
        10.0
ASSIGN
        b
        5.0
SUB
        ADD
                DIV
                        sqrt
                                16.0
                        2.0
                MUL
                        ADD
                                3.0
                                4.0
                        2.0
        -1.0
0.0
0.0
17.0
```

```
yzl1015@DESKTOP-BTRCQ6V:~/pa_visit9$ java program
0.02 + 0.03; 0.02 - -0.03;
ADD
        0.02
        0.03
SUB
        0.02
        -0.03
0.05
0.05
```