Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Метрология, стандартизация и сертификация в информационных технологиях (МСиСвИнфТ)

**ОТЧЕТ**

по работе № 1

Тема работы: Расчет метрик Холстеда

Выполнили

студенты: гр. 951007 Бакыт Мади

Майский В.И.

Проверил: Болтак С.В.

Минск 2020

Содержание

[1 Принцип работы программы 3](#_Toc52485641)

[1.1 Лексер 3](#_Toc52485642)

[1.2 Парсер 4](#_Toc52485643)

[1.3 Метрика 4](#_Toc52485644)

[2 Исходный код программы 5](#_Toc52485645)

[2.1 Lexer.java 5](#_Toc52485646)

[2.2 Parser.java 8](#_Toc52485647)

[2.3 Metric.java 12](#_Toc52485648)

[3 Код анализируемой программы (программа 1) 14](#_Toc52485649)

[4 Расчет метрик по программе 1 18](#_Toc52485650)

[5 Код анализируемой программы (программа 2) 20](#_Toc52485651)

[6 Расчет метрик по программе 2 20](#_Toc52485652)

[6.1 Расчет метрик вручную 20](#_Toc52485653)

[6.2 Расчет метрик 21](#_Toc52485654)

# 1 Принцип работы программы

## 1.1 Лексер

Данная программа получает исходный код, в виде сырого текста, из файла, с расширением .kt. Затем, считывая текст символ за символом, переводит его в набор лексем (токенизирует).

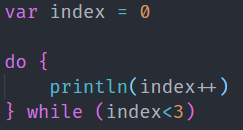


Рисунок 1.1 – Исходный код алгоритма

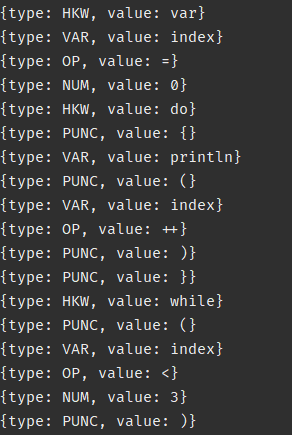


Рисунок 1.2 – Таблица лексем

В рисунках 1 и 2 видно, как преобразуется исходный код в набор лексем. Каждая лексема (токен) представляет собой объект с двумя полями: тип и значение. В типе записывается один из восьми типов лексемы:

1. пунктуация;
2. число;
3. строка;
4. символ;
5. ключевое слово;
6. символ операции;
7. переменная (**не** ключевое слово),

а в значение записывается строковое значение токена.

## 1.2 Парсер

На следующем этапе таблица токенов передается в подобие парсера, которое переводит ее в набор аргументов, путем разделения токенов на различные подгруппы и/или учитывая окружающие ее лексемы.

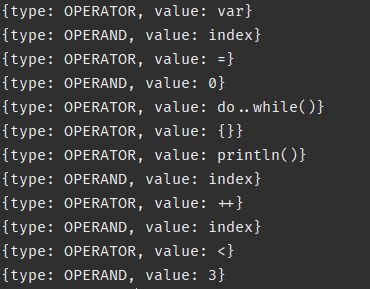


Рисунок 1.3 – Таблица аргументов

В рисунке 1.3 показано, как выглядит код из рисунка 1.1, после преобразования его в таблицу аргументов. Каждый аргумент представляет собой объект с двумя полями: тип и значение. Тип может быть либо **оператор**, либо **операнд**, значение же содержит репрезентацию аргумента в виде строки.

## 1.3 Метрика

На последнем этапе таблица аргументов группируется, разделяется на две подгруппы: операторы и операнды и подсчитываются сами метрики, после чего результаты выводятся на экран. Перед выводом на экран, таблицы операторов и операндов сортируются в невозрастающем (убывающем) порядке.

# 2 Исходный код программы

## 2.1 Lexer.java

**package** by.lexer;

**import** java.util.ArrayList;

**import** by.lexer.Enums.Type;

**public** **class** Lexer {

**private** ArrayList<Token> tokens;

**private** **char**[] inputStream;

**private** **int** currentIndex = 0;

**public** Lexer(String unparsedText) {

        tokens = **new** ArrayList<Token>();

**this**.inputStream = unparsedText.toCharArray();

        parse();

    }

**public** ArrayList<Token> getTokens() {

**return** tokens;

    }

**private** **void** parse() {

**while**(currentIndex < inputStream.length) {

            readNext();

            currentIndex++;

        }

    }

**public** String toString() {

        StringBuilder toReturn = **new** StringBuilder("");

**for**(**int** i = 0; i < tokens.size(); i++)

            toReturn.append(tokens.get(i).toString() + "**\n**");

**return** toReturn.toString();

    }

**private** **void** readNext() {

**char** ch = inputStream[currentIndex];

**if**(ch == ' ' || ch == '**\n**' || ch == '**\t**') {

**return**;

        }

**if**(ch == '/' && (inputStream[currentIndex+1] == '/' || inputStream[currentIndex+1] == '\*')) {

            skipComment();

**return**;

        }

**if**(ch == '"') {

            readString();

**return**;

        }

**if**(ch == '**\'**') {

            readChar();

**return**;

        }

**if**(Character.isDigit(ch)) {

            readNumber();

**return**;

        }

**if**(Character.isLetter(ch) || ch == '\_') {

            readIdent();

**return**;

        }

**if**(";,(){}[]".indexOf(ch) >= 0) {

            Token t = **new** Token(Type.PUNC, ch + "");

            tokens.add(t);

**return**;

        }

**if**(isOp(ch)) {

            readOp();

**return**;

        }

        System.err.printf("Cannot read character '%c' [%d]**\n**",ch,currentIndex);

    }

**private** **void** skipComment() {

**if**(inputStream[currentIndex+1] == '/') {

**while**(inputStream[currentIndex] != '**\n**')

                currentIndex++;

        } **else** **if**(inputStream[currentIndex+1] == '\*') {

            currentIndex += 3;

**while**(inputStream[currentIndex-1] != '\*' || inputStream[currentIndex] != '/') {

                currentIndex++;

            }

        }

    }

**private** **void** readString() {

**if**(inputStream[currentIndex] == '"' &&

            inputStream[currentIndex + 1] == '"' &&

            inputStream[currentIndex + 2] == '"') {

            readMultilineString();

        } **else** {

            StringBuilder lexem = **new** StringBuilder("**\"**");

            currentIndex++;

**while**(inputStream[currentIndex] != '"') {

**if**(inputStream[currentIndex] == '**\\**')

                    lexem.append(inputStream[currentIndex++]);

                lexem.append(inputStream[currentIndex++]);

            }

            lexem.append("**\"**");

            tokens.add(**new** Token(Type.STR, lexem.toString()));

        }

    }

**private** **void** readMultilineString() {

        StringBuilder lexem = **new** StringBuilder("**\"**");

        currentIndex += 3;

**while**(!(inputStream[currentIndex] == '"' &&

                inputStream[currentIndex + 1] == '"' &&

                inputStream[currentIndex + 2] == '"')) {

**if**(inputStream[currentIndex] == '**\n**' ||

                    inputStream[currentIndex] == '**\r**' ||

                    inputStream[currentIndex] == '**\t**') {

                currentIndex++;

**continue**;

            }

            lexem.append(inputStream[currentIndex++]);

        }

        lexem.append("**\"**");

        currentIndex += 2;

        tokens.add(**new** Token(Type.STR, lexem.toString()));

    }

**private** **void** readChar() {

        StringBuilder lexem = **new** StringBuilder("'");

        currentIndex++;

**while**(inputStream[currentIndex] != '**\'**') {

**if**(inputStream[currentIndex] == '**\\**')

                lexem.append(inputStream[currentIndex++]);

            lexem.append(inputStream[currentIndex++]);

        }

        lexem.append("'");

        tokens.add(**new** Token(Type.CHAR, lexem.toString()));

    }

**private** **void** readNumber() {

        StringBuilder lexem = **new** StringBuilder("");

**while**(isDigit(inputStream[currentIndex]) ||

                (inputStream[currentIndex] == '.' && isDigit(inputStream[currentIndex + 1]))) {

**if**(inputStream[currentIndex] == 'e')

                lexem.append(inputStream[currentIndex++]);

            lexem.append(inputStream[currentIndex++]);

        }

        currentIndex--;

        tokens.add(**new** Token(Type.NUM, lexem.toString()));

    }

**private** **boolean** isDigit(**char** ch) {

**return** "0123456789ABCDEFbefLx\_".indexOf(ch) >= 0;

    }

**private** **void** readIdent() {

        StringBuilder lexem = **new** StringBuilder("");

**while**(Character.isDigit(inputStream[currentIndex]) ||

                Character.isLetter(inputStream[currentIndex]) ||

                inputStream[currentIndex] == '\_') {

            lexem.append(inputStream[currentIndex++]);

        }

        currentIndex--;

        Type type = isHardKeyword(lexem.toString()) ? Type.HKW : isSoftKeyword(lexem.toString()) ? Type.SKW : Type.VAR;

        tokens.add(**new** Token(type, lexem.toString()));

    }

**private** **boolean** isHardKeyword(String text) {

        String keywords = " as break class continue do else false for fun if in interface is null object package return super this throw true try typealias typeof val var when while ";

**return** keywords.indexOf(" " + text + " ") >= 0;

    }

**private** **boolean** isSoftKeyword(String text) {

        String keywords = " by catch constructor delegate dynamic field file finally get import init param property receiver set setparam where actual abstract annotation companion const crossinline data enum expect external final inflix inline inner internal lateinit noinline open operator out override private protected public reified sealed suspend tailrec vararg field it ";

**return** keywords.indexOf(" " + text + " ") >= 0;

    }

**private** **boolean** isOp(**char** ch) {

**return** "+-/\*=%&|<>!@:?.".indexOf(ch) >= 0;

    }

**private** **boolean** isCombOp(String op, **char** next) {

**return** " += -= \*= /= %= ++ -- && || == != === !== >= <= !! ?. ?: :: .. -> ".indexOf(op + next) >= 0;

    }

**private** **void** readOp() {

        StringBuilder lexem = **new** StringBuilder("");

**while**(isOp(inputStream[currentIndex]) && isCombOp(lexem.toString(), inputStream[currentIndex])) {

            lexem.append(inputStream[currentIndex++]);

        }

        currentIndex--;

        tokens.add(**new** Token(Type.OP, lexem.toString()));

    }

}

## 2.2 Parser.java

**package** by.lexer;

**import** java.util.ArrayList;

**import** by.lexer.Enums.Op;

**import** by.lexer.Enums.Type;

**public** **class** Parser {

**private** ArrayList<Token> tokens;

**private** ArrayList<ArgumentToken> args;

**private** **int** currentIndex = 0;

**public** Parser(ArrayList<Token> tokens) {

        args = **new** ArrayList<ArgumentToken>();

**this**.tokens = **new** ArrayList<Token>(tokens);

        parse();

    }

**public** Parser(String raw) {

**this**(**new** Lexer(raw).getTokens());

    }

**public** ArrayList<ArgumentToken> getArgTokens() {

**return** args;

    }

**private** **void** parse() {

**while**(currentIndex < tokens.size()) {

            readNext();

            currentIndex++;

        }

    }

**public** String toString() {

        StringBuilder toReturn = **new** StringBuilder("");

**for**(**int** i = 0; i < args.size(); i++)

            toReturn.append(args.get(i).toString() + "**\n**");

**return** toReturn.toString();

    }

**public** **void** readNext() {

        Token t = tokens.get(currentIndex);

**if**(t.type == Type.HKW) {

            readHKW();

**return**;

        }

**if**(t.type == Type.SKW) {

            readSKW();

**return**;

        }

**if**(t.type == Type.VAR) {

            args.add(**new** ArgumentToken(Op.OPERAND, t.value));

**return**;

        }

**if**(t.type == Type.OP) {

            readOP();

**return**;

        }

**if**(t.type == Type.NUM || t.type == Type.CHAR || t.type == Type.STR) {

            args.add(**new** ArgumentToken(Op.OPERAND, t.value));

**return**;

        }

**if**(t.type == Type.PUNC) {

            readPunc();

**return**;

        }

        System.err.printf("Cannot read character '%s' [%s]**\n**",t.type,t.value);

    }

**private** **void** readHKW() {

        String value = tokens.get(currentIndex).value;

**if**(value.equals("true") || value.equals("false") || value.equals("null")) {

            args.add(**new** ArgumentToken(Op.OPERAND, tokens.get(currentIndex).value));

**return**;

        }

**if**(value.equals("do")) {

            readDo();

**return**;

        }

**if**(value.equals("if")) {

**boolean** isReturn = readIf();

**if**(isReturn)

**return**;

        }

**if**(value.equals("else")) {

            readElse();

**return**;

        }

        args.add(**new** ArgumentToken(Op.OPERATOR, tokens.get(currentIndex).value));

    }

**private** **void** readDo() {

**int** amount = 1;

**int** temp = currentIndex;

        temp += 2;

**while**(amount != 0) {

**if**(tokens.get(temp).value.equals("{"))

                amount++;

**else** **if**(tokens.get(temp).value.equals("}"))

                amount--;

            temp++;

        }

        tokens.remove(temp);

        tokens.remove(temp);

        args.add(**new** ArgumentToken(Op.OPERATOR, "do..while()"));

    }

**private** **boolean** readIf() {

**int** temp = currentIndex + 2;

**while**(!tokens.get(temp).value.equals(")"))

            temp++;

        temp++;

**boolean** isReturn = **false**;

**if**(tokens.get(temp).value.equals("{"))

             isReturn = readIfElse(temp);

**return** isReturn;

    }

**private** **boolean** readIfElse(**int** index) {

**int** temp = index+1;

**int** amount = 1;

**while**(amount != 0) {

**if**(tokens.get(temp).value.equals("{"))

                amount++;

**else** **if**(tokens.get(temp).value.equals("}"))

                amount--;

            temp++;

        }

**if**(tokens.get(temp).value.equals("else")) {

            tokens.remove(temp);

            args.add(**new** ArgumentToken(Op.OPERATOR, "if()..else"));

            currentIndex++;

**return** **true**;

        }

**return** **false**;

    }

**private** **void** readElse() {

**if**(tokens.get(currentIndex + 1).value.equals("->")) {

            args.add(**new** ArgumentToken(Op.OPERATOR, "else"));

        } **else** {

**int** temp = args.size() - 1;

**while**(!args.get(temp).value.equals("if()"))

                temp--;

            args.get(temp).value += "..else";

        }

    }

**private** **void** readSKW() {

**if**(tokens.get(currentIndex).value.equals("catch")) {

**int** index = 0;

**for**(index = args.size() - 1; !args.get(index).value.equals("try"); index--);

            args.get(index).value = "try..catch()";

            currentIndex++;

**return**;

        }

**if**(tokens.get(currentIndex).value.equals("finally")) {

**int** index = 0;

**for**(index = args.size() - 1; !args.get(index).value.equals("try..catch()"); index--);

            args.get(index).value = "try..catch()..finally";

**return**;

        }

        args.add(**new** ArgumentToken(Op.OPERATOR, tokens.get(currentIndex).value));

    }

**private** **void** readOP() {

**if**(tokens.get(currentIndex).value.equals(".") && tokens.get(currentIndex + 1).type == Type.NUM) {

            tokens.get(currentIndex + 1).value = '.' + tokens.get(currentIndex + 1).value;

**return**;

        }

**if** (tokens.get(currentIndex).value.equals(".")) {

            args.get(args.size() - 1).value += "." +  tokens.get(currentIndex + 1).value;

            currentIndex++;

**return**;

        }

**if** (tokens.get(currentIndex).value.equals(">")) {

**int** index;

**for**(index = args.size() - 1; args.get(index).op != Op.OPERATOR; index--);

**if**(args.get(index).value.equals("<")) {

                args.get(index).value = "<>";

**return**;

            }

        }

        args.add(**new** ArgumentToken(Op.OPERATOR, tokens.get(currentIndex).value));

    }

**private** **void** readPunc() {

**if**(tokens.get(currentIndex).value.equals("(") && (tokens.get(currentIndex - 1).type == Type.HKW || tokens.get(currentIndex - 1).type == Type.SKW)) {

            args.get(args.size() - 1).value += "()" ;

**return**;

        }

**if**(tokens.get(currentIndex).value.equals("(") && tokens.get(currentIndex - 1).type == Type.VAR) {

            args.get(args.size() - 1).value += "()" ;

            args.get(args.size() - 1).op = Op.OPERATOR ;

**return**;

        }

**if**(tokens.get(currentIndex).value.equals("(")) {

            args.add(**new** ArgumentToken(Op.OPERATOR, "()"));

**return**;

        }

**if**(tokens.get(currentIndex).value.equals("{")) {

            args.add(**new** ArgumentToken(Op.OPERATOR, "{}"));

**return**;

        }

**if**(tokens.get(currentIndex).value.equals("[")) {

            args.add(**new** ArgumentToken(Op.OPERATOR, "[]"));

**return**;

        }

    }

}

## 2.3 Metric.java

**package** by.lexer;

**import** java.util.ArrayList;

**import** by.lexer.Enums.Op;

**public** **class** Metric {

**private** ArrayList<ArgumentToken> argTokens;

**private** ArrayList<Argument> argOperators;

**private** ArrayList<Argument> argOperands;

**private** **int** nu1;

**private** **int** nu2;

**private** **int** n1;

**private** **int** n2;

**private** **int** nu;

**private** **int** n;

**private** **int** v;

**public** Metric(ArrayList<ArgumentToken> argTokens) {

**this**.argTokens = **new** ArrayList<ArgumentToken>(argTokens);

        argOperators = **new** ArrayList<Argument>();

        argOperands = **new** ArrayList<Argument>();

        countMetrics();

    }

**public** Metric(String raw) {

**this**(**new** Parser(**new** Lexer(raw).getTokens()).getArgTokens());

    }

**public** ArrayList<Argument> getOperatorsArgs() {

**return** argOperators;

    }

**public** ArrayList<Argument> getOperandsArgs() {

**return** argOperands;

    }

**public** **int** getNu1() {

**return** nu1;

    }

**public** **int** getNu2() {

**return** nu2;

    }

**public** **int** getN1() {

**return** n1;

    }

**public** **int** getN2() {

**return** n2;

    }

**public** **int** getNu() {

**return** nu;

    }

**public** **int** getN() {

**return** n;

    }

**public** **int** getV() {

**return** v;

    }

**private** **void** countMetrics() {

        argTokens.**forEach**(at -> countArg(at.op == Op.OPERATOR ?

                argOperators : argOperands, at));

        argOperators.sort((t1, t2) -> Integer.compare(t2.amount, t1.amount));

        argOperands.sort((t1, t2) -> Integer.compare(t2.amount, t1.amount));

        nu1 = argOperators.size();

        nu2 = argOperands.size();

        argOperators.**forEach**(a -> n1 += a.amount);

        argOperands.**forEach**(a -> n2 += a.amount);

        nu = nu1 + nu2;

        n = n1 + n2;

        v = (**int**) Math.ceil(n \* (Math.log(nu) / Math.log(2)));

    }

**private** **void** countArg(ArrayList<Argument> args, ArgumentToken at) {

**int** index = getIndexOf(args, at);

**if**(index == -1) {

            args.add(**new** Argument(at.value));

        } **else** {

            args.get(index).incAmount();

        }

    }

**private** **int** getIndexOf(ArrayList<Argument> args, ArgumentToken at) {

**for**(**int** i = 0; i < args.size(); i++) {

**if**(at.value.equals(args.get(i).value))

**return** i;

        }

**return** -1;

    }

}

# 3 Код анализируемой программы (программа 1)

**fun** main() {

    print(factorial(4))

**val** list = bubbleSort(intArrayOf(2,15,1,8,4))

    printArray(list.toTypedArray())

    getAmountOfSymbols(1235)

    printWeek()

**val** resArray : Array<Int> = getFilledArray();

    findAverageValue(resArray)

    println(taylorSin(Math.PI))

    println(compare(12,13))

    println(getSign(-12.3))

**var** x = 20.0

    x = findSin(x) + findCos(x)

    print(x)

}

*// Developed by Vlad Maiski and Bakyt Madi*

**fun** getAbs(a : Double) : Double {

**if** (a < 0)

**return** -a

**else**

**return** a

}

*/\**

*Some comments which can help u to understand the program*

*And also some code here to try confuse the metric's code*

*fun isOp() {*

*if ( a & 1 ) {*

*println("нечетно")*

*} else {*

*println("четно")*

*}*

*\*/*

**fun** getSign(a : Double) : **Int** {

**if** (a > 0)

**return** 1

**else**

**return** -1

}

**fun** resolveFunction(a : Double) : Double {

**val** eps : Double = 0.00001

**val** x : Double

**var** y : Double

**var** det : Double

**var** n : **Int** = 3

    x = a

    y = a

    det = a

**do** {

        det = -det \* x \* x / ((n + 1)\*n)

        n++

        y += det

    } **while** (getAbs(det) > eps)

**return** y

}

**fun** findExp(a : Double) : Double {

**val** eps : Double = 0.00001

**var** x : Double

**var** y : Double

**var** det : Double

**var** n : **Int** = 1

    x = a

    y = 1.0

    det = 1.0

**do** {

        det = det \* x / n

        n++

        y += det

    } **while** (getAbs(det) > eps)

**return** y

}

**fun** printWeek() {

**var** userChoose: **Int**

**do** {

        println("""

           1 - Monday

           2 - Tuesday

           3 - Wednesday

           4 - Thursday

           5 - Friday

           6 - Saturday

           7 - Sunday

           8 - to exit program

       """.trimIndent())

        userChoose = readLine().toString().toInt()

**when**(userChoose) {

            1 -> println("Monday")

            2 -> println("Tuesday")

            3 -> println("Wednesday")

            4 -> println("Thursday")

            5 -> println("Friday")

            6 -> println("Saturday")

            7 -> println("Sunday")

            8 -> **return**;

**else** -> println("try again")

        }

    }**while** (**true**)

}

**fun** calcuate(a : Double) : Double {

**val** eps : Double = 0.00001

**var** x : Double

**var** y : Double

**var** det : Double

**var** n : **Int** = 3

    x = a

    y = a

    det = a

**do** {

        det = -det \* x \* x / ((n + 1)\*n)

        n++

        y += det

    } **while** (getAbs(det) > eps)

**return** y

}

**fun** compare(a : **Int**, b : **Int**) : **Int** {

**if** (a < b) {

**return** b

    } **else**

**return** a

}

**fun** findSin(a : Double) : Double {

**val** eps : Double = 0.00001

**var** x : Double

**var** y : Double

**var** vs : Double

**var** n : **Int** = 2

    x = a

    y = x

    vs = x

**do** {

        vs = - vs \* x \* x / ((2 \* n - 1)\*(2 \* n - 2))

        n++

        y += vs

    } **while** (getAbs(vs) > eps)

**return** y % (2 \* 3.1415926)

}

**fun** findCos(a : Double) : Double {

**val** eps : Double = 0.00001

**var** x : Double

**var** y : Double

**var** vs : Double

**var** n : **Int** = 2

    x = a

    y = 1.0

    vs = 1.0

**do** {

        vs = - vs \* x \* x / ((n - 1)\*(n))

        n += 2

        y += vs

    } **while** (getAbs(vs) > eps)

**return** y % (2 \* 3.1415926)

}

**fun** getFilledArray() : Array<Int> {

    println("Enter amount of elements")

**val** amountOfElmnts : **Int** = readLine().toString().toInt()

**val** resArray = IntArray(amountOfElmnts)

**for** (i **in** 0 until amountOfElmnts) {

        println("Enter element number " + (i + 1))

        resArray[i] = readLine().toString().toInt()

    }

**return** resArray.toTypedArray()

}

**fun** findAverageValue(array : Array<Int>) {

**var** sum : **Int** = 0

**for**(i **in** array.indices) {

        print(array[i].toString() + " ")

        sum += array[i]

    }

    println("**\n**" +*/\*output sum/size!\*/* sum.toDouble() / array.size)

}

**fun** printArray(array : Array<Int>) {

**for** (i **in** array.indices)

        print(" " + array[i])

    println()

}

**fun** getAmountOfSymbols(\_value : **Int**) {

**var** value = \_value

**var** amount = 0

**while** (value != 0) {

        value /= 10

        amount++

    }

    println(amount)

}

*//Bubble Sort*

**fun** bubbleSort(arr:IntArray):IntArray{

**var** swap = **true**

**while**(swap){

        swap = **false**

**for**(i **in** 0 until arr.size-1){

**if**(arr[i] > arr[i+1]){

**val** temp = arr[i]

                arr[i] = arr[i+1]

                arr[i + 1] = temp

                swap = **true**

            }

        }

    }

**return** arr

}

**fun** factorial(n: **Int**): Double {

**var** result = 1.0

**for** (i **in** 1..n) {

        result \*= i

    }

**return** result

}

**fun** taylorSin(x: Double): Double {

**var** sumNeu: Double

**var** sumOld: Double

**var** sum: Double

**var** i = 1

    sumNeu = x

    sum = sumNeu *// This should calculating the first term Value*

**do**  *//the loop do will calculating the Tailor Series*

    {

        sumOld = sumNeu

        i++

        sum = +sum \* x \* x / i

        i++

        sum /= i

        sumNeu = sumOld + sum

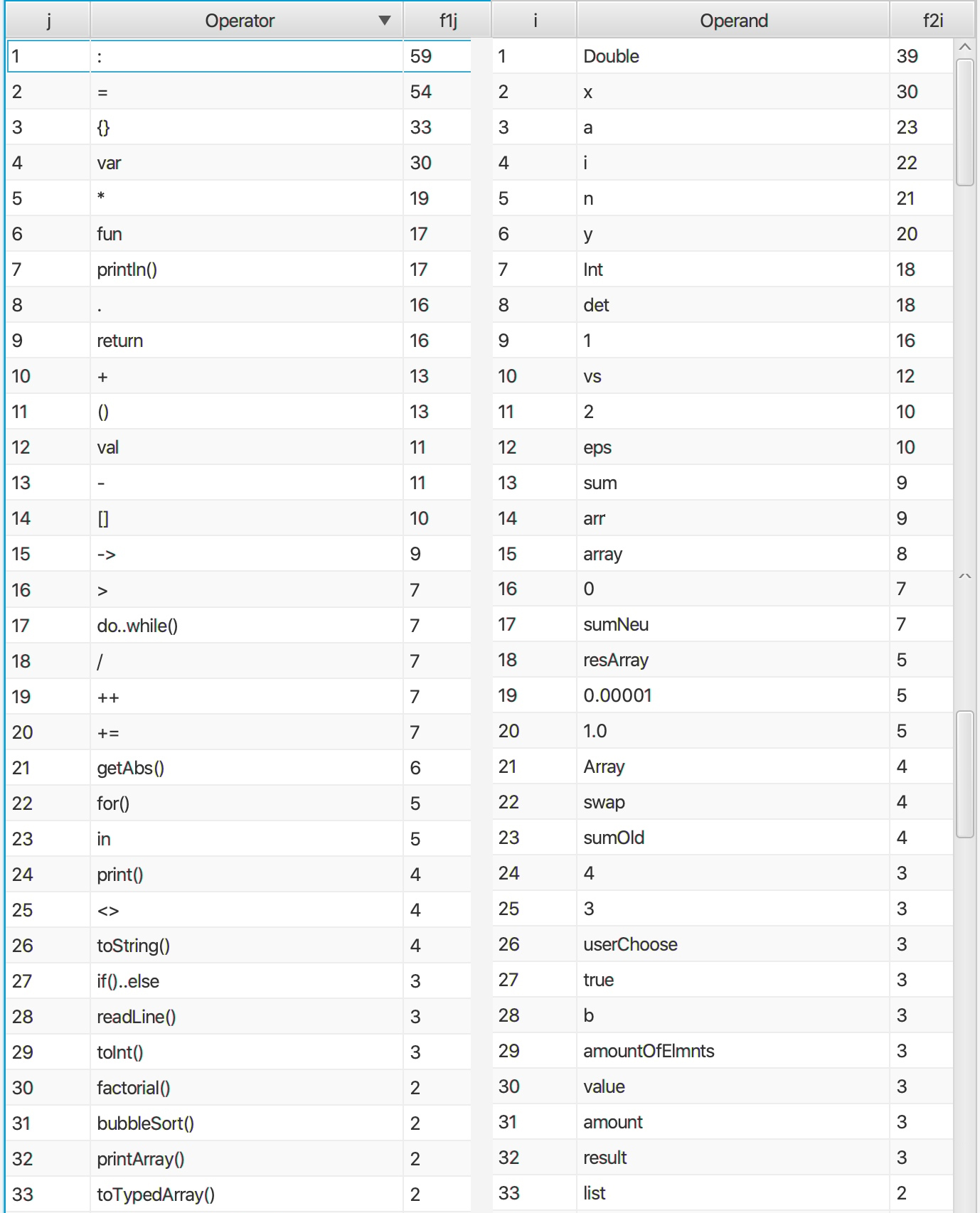
    } **while** (sumNeu != sumOld)

**return** sumNeu

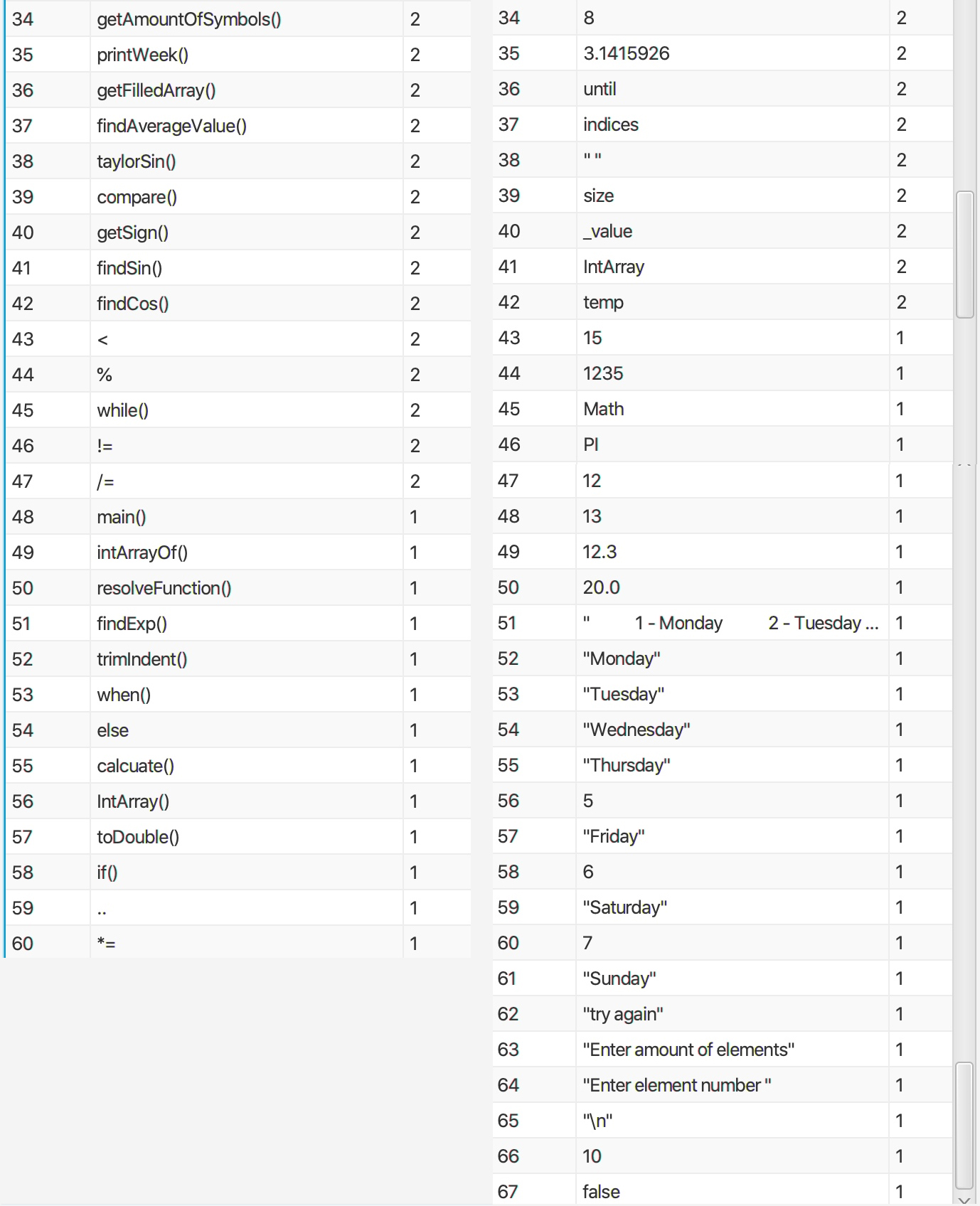
}

# 4 Расчет метрик по программе 1

Таблица 4.1 – метрики программы 1



Продолжение таблицы 4.1



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| μ1 |  | N1 | μ2 |  | N2 |
| 60 |  | 449 | 67 |  | 378 |

Словарь программы μ = 127

Длина программы N = 827

Объем программы V = 5780

# 5 Код анализируемой программы (программа 2)

**fun** main() {

*//This program outputs Fibonacci numbers*

**var** l = 1;

**var** r = 1;

**for**(i **in** 3..50) {

**val** tmp = l

        l += r

        r = tmp

    }

**if**(l > 1000)

        println("It's big!");

**else**

        println(l)

*/\**

*return a+b*

*\*/*

}

# 6 Расчет метрик по программе 2

## 6.1 Расчет метрик вручную

Таблица 6.1 – расчет метрик вручную

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| j | Оператор | f1j | i | Операнд | f2i |
| 1 | fun | 1 | 1 | l | 5 |
| 2 | main() | 1 | 2 | 1 | 2 |
| 3 | {} | 2 | 3 | r | 3 |
| 4 | var | 2 | 4 | i | 1 |
| 5 | = | 4 | 5 | 3 | 1 |
| 6 | for() | 1 | 6 | 50 | 1 |
| 7 | in | 1 | 7 | tmp | 2 |
| 8 | .. | 1 | 8 | 1000 | 1 |
| 9 | val | 1 | 9 | “It’s big!” | 1 |
| 10 | += | 1 |  |  |  |
| 11 | if()..else | 1 |  |  |  |
| 12 | > | 1 |  |  |  |
| 13 | println() | 2 |  |  |  |
| μ1 |  | N1 | μ2 |  | N2 |
| 13 |  | 19 | 9 |  | 17 |

Словарь программы μ = μ1 + μ2 = 13 + 9 = 22

Длина программы N = N1 + N2 = 36

Объем программы V = N

## 6.2 Расчет метрик

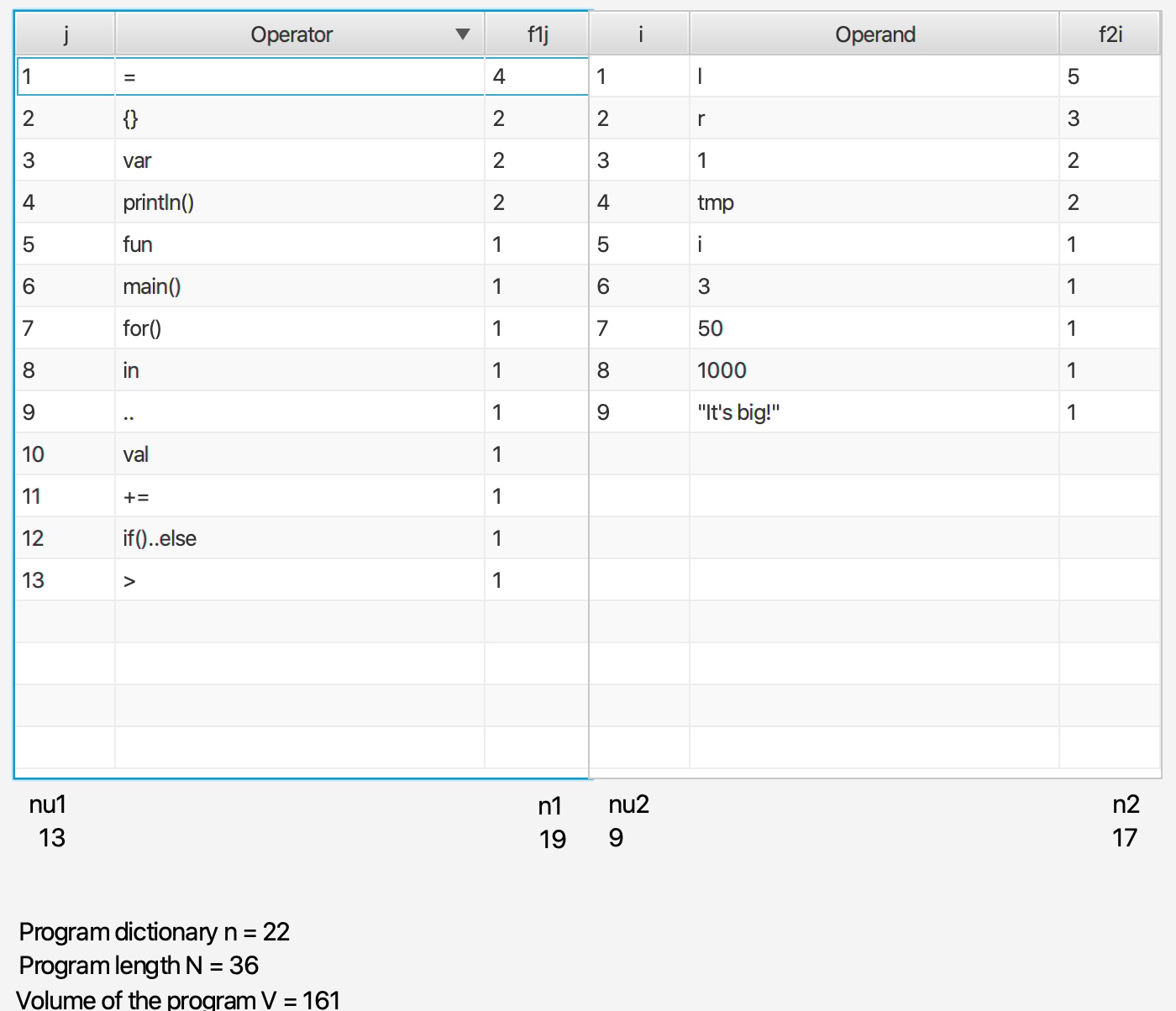


Рисунок 6.1 – Расчет метрик в программе-анализаторе