СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Создание языка программирования очень сложная задача, но выполнимая. Желание создать что-то своё и оставить след в истории посещает всех людей, в том числе программистов. Создание собственного языка программирования – это подходящая возможность.[1]

Для того, чтобы код было сравнительно легко поддерживать, программу надо разбить на части и выделить промежуточные структуры данных.

У всякого интерпретатора языка программирования можно выделить три изолированных друг от друга части, каждая из которых, как отдельный участок общего конвейера, выполняет свою работу.

Разбивка строки входящего текста на отдельные элементы — так называемые «токены». Токены являются своеобразными «атомами» разрабатываемого языка — минимальными неделимыми элементами.

Перевод получившегося массива токенов в некую удобную для последующих вычислений форму.

Первую часть называют «лексером» (lexer), вторую «парсером» (parser), третью «интерпретатором» (interpreter).

При передаче данных от второй части к третьей удобно было бы к токенам добавлять сопутствующую информацию о том, к какому типу относится токен (число это, оператор или одна из двух скобок).

Данное требование мы реализуем путём представления каждого токена в виде хеша из двух элементов: тип и, собственно, значение токена (если требуется).

1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1.1 Введение

Составленное техническое задание по дисциплине «Разработка предметно-ориентированных языков программирования» является документом к курсовой работе, который отражает все этапы разработки языка программирования, а также процесс проектирования и выявление требований, предъявляемых конечному продукту.

1.2 Основание для разработки

Основанием для разработки является курсовая работа по дисциплине «Разработка предметно-ориентированных языков программирования», предусмотренная учебным планом направления подготовки 09.03.01 «Информатика и вычислительная техника» профиля «Цифровые комплексы, системы и сети».

1.3 Требования, предъявляемые к программе

1.3.1 Требования к функциональным характеристикам программы

В приложении должны быть реализованы следующие операции:

- Работа с вещественными числами.
- Выполнение операций сложения, вычитания, умножения и деления.
- Выполнение операций сравнения.
- Оператор скобки
- Оператор присвоения, переменные.
- Строки, оператор вывода.
- Логические выражения.
- Разделение на блоки с помощью фигурных скобок.
- Текстовый редактор с графическим интерфейсом
- Запуск выполнения программы с помощью кнопки
- Вывод результата выполнения в отдельном окне
- Функции сохранения, открытия, создания текстового файла
- Функции копирования, вырезания, вставки текста

1.3.2 Требования к техническим средствам, используемым при работе программы

Персональный компьютер пользователя должен быть оснащён графическим адаптером, операционной системой Windows, Linux или Mac OS, должна быть установлена Java JRE, версия не ниже Java 14.

1.3.3 Требования к языкам программы и среде разработки программы

Для разработки используется язык программирования Java, в качестве среды разработки выступает IntelliJ IDEA, сборка проекта осуществляется при помощи Maven.

1.4 Требования к программной документации

1. Пояснительная записка оформляется в соответствии с ЛНА РТУ МИРЭА.
2. Проектная документация, составленная в соответствии с ГОСТ.

В процессе создания приложения вся проделанная работа документируется, должны быть сохранены все детали разработки, а также трудности, с которыми пришлось столкнуться. Всё вышеперечисленное должно быть отраженов пояснительной записке, которая прилагается к работе.

2 ОПИСАНИЕ ПРОГРАММЫ

2.1 Общие сведения

В ходе выполнения курсовой работы был разработан язык программирования, кроссплатформенный, благодаря тому что написан на Java, может работать на любой из популярных операционных систем. А также было разработано десктоп приложение графический редактор кода для созданного языка. В курсовой работе выполняются все условия, обозначенные в техническом задании, и содержатся все необходимые компоненты, инструменты для корректной работы.

2.1.1 Программное обеспечение, необходимое для функционирования программы

Для корректного функционирования данного программного продукта необходимо, чтобы на персональном компьютере или ноутбуке пользователя была любая популярная операционная система, например Windows, Linux или Mac OS. Также требуется наличие графического адаптера, чтобы взаимодействовать с программой через графический интерфейс. Другие требования к устройству пользователя не предусмотрены.

2.1.2 Язык программирования, на котором написана программа

Для написания программы был выбран язык программирования Java, так как он отлично подходит для реализации больших проектов, позволяет реализовать программу, которая будет работать на разных операционных системах, имеет графическую библиотеку, с помощью которой был создан редактор кода.

2.2 Описание логической структуры языка

2.2.1 Класс Token

Класс Token (Листинг А.8, Приложение А) используется лексером (lexer) для создания списка токенов из входного текста программы, который будет использоваться в последующем анализе и выполнении программы. Таким образом, Token является важным элементом, необходимым для выполнения программы на языке программирования.

Класс Token представляет токен,[2] который является базовым элементом в синтаксисе программирования. Токены представляют собой лексемы, разделенные в исходном коде пробелами, символами табуляции и переводами строк.

Класс Token имеет два поля: тип токена (type) и текстовое значение токена (text). Тип токена указывает на лексему, которую он представляет, а текстовое значение указывает на саму лексему.

2.2.2 Енам TokenType

TokenType (Листинг А.9, Приложение А) используется лексером (lexer) для определения типа каждой лексемы в исходном коде программы. Таким образом, TokenType позволяет детектировать различные типы лексем, что необходимо для выполнения правильного анализа и последующего выполнения программы на языке программирования.

Перечисление TokenType имеет следующие значения:

- NUMBER: Числа
- WORD: Переменная
- TEXT: Строка
- PRINT: Вывод

- IF: Если
- ELSE: Иначе
- PLUS: «+» Сложение
- MINUS: «-» Вычитание
- STAR: «*» Умножение
- SLASH: «/» Деление
- EQ: «=» Присвоение
- EQEQ: «==» Равно
- EXCL: «!» Не
- EXCLEQ: «!=» Не равно
- LT: «<» Меньше
- LTEQ: «<=» Меньше или равно
- GT: «>» Больше
- GTEQ: «>=» Больше или равно
- BAR: «|»
- BARBAR: «||» Или
- AMP: «&»
- AMPAMP: «&&» И
- LPAREN: «(»
- RPAREN: «)»
- LBRACE: «{»
- RBRACE: «}»
- EOF: Конец файла

2.2.3 Класс Parser

В классе Parser (Листинг А.16, Приложение А) есть приватные поля tokens, size и pos, конструктор, который инициализирует tokens, size и pos.

Также класс содержит методы для работы с операторами присваивания, операторами ветвления и другими языковыми конструкциями.

Метод logicalAnd составляет логическое выражение, содержащее оператор && (логический оператор "и"), метод equality - выражение сравнения, содержащее операторы == (равно) и != (не равно), метод conditional - выражение сравнения, содержащее операторы < (меньше), <= (меньше или равно), > (больше) и >= (больше или равно), метод additive - арифметическое выражение, содержащее операторы + (сложение) и - (вычитание), а метод multiplicative - арифметическое выражение, содержащее операторы * (умножение) и / (деление).

Также в коде присутствуют методы unary и primary, которые выполняют различные операции с элементами выражения, такие как изменение знака числа и получение значения переменной или числа из лексем.

В данном коде определен метод get, который используется для получения лексемы, находящейся на относительной позиции relativePosition от текущей позиции pos в массиве лексем tokens. Если указанная позиция находится за пределами массива, метод возвращает специальную лексему EOF, которая обозначает конец файла.

Также есть метод block и statementOrBlock для парсинга выражений по блокам, которые находятся между фигурных скобок.

В целом, данный код выполняет функцию разбора входного кода в соответствии с синтаксисом языка и формирует дерево синтаксического анализа для последующей интерпретации.

2.2.4 Класс Lexer

Данный код (Листинг А.10, Приложение А) представляет собой лексический анализатор, который принимает на вход строку с исходным кодом

на языке программирования и разбивает его на лексемы - минимальные единицы языка, которые имеют смысл и являются частью синтаксиса.

2.2.5 Класс Interpreter

Этот код (Листинг А.22, Приложение А) представляет собой класс Interpreter, который содержит метод execute, выполняющий интерпретацию списка выражений.

выражения прошло без ошибок, то результат выводится на экран с помощью метода println. Если же вычисление вызвало ошибку, то программа выведет сообщение об ошибке на экран.

Архитектура проекта включает в себя отдельные сущности, такие как statement, expression, value, variables. Создан интерфейс Statement, который реализуют классы AssignmentStatement, BlockStatement, IfStatement, PrintStatement. Операторы присвоения, деления на блоки, ветвления и вывода соответственно. У интерфейса Statement есть метод execute, который реализуют эти классы – метод, который исполняет код и вызывается из интерпретатора.

Также создан интерфейс Expression и реализующие его классы BinaryExpression, UnaryExpression, ConditionalExpression, ValueExpression, VariableExpression. Интерфейс имеет метод eval, который возвращает значение. Эти классы используются для парсинга программы.

Ещё для динамической типизации создан интерфейс Value и реализующие его классы NumberValue, StringValue. Value имеет два метода asNumber и asString, которые позволяют представить значение как число или как строку.

Создан класс Variables, который позволяет создавать и работать с константами, такими как: число ПИ, число Е и золотое сечение.

2.3 Вызов программы

При запуске программного продукта появляется окно текстового редактора, где можно писать код, либо открыть уже существующий файл. Также есть функции создания нового файла, сохранение файла. Все эти фунции находятся во вкладке File.

Реализована возможность копирования, вырезания и вставка текста, все они расположены во вкладке Edit.

Для запуска программы есть кнопка Run, нажав на которую появится окно с результатом выполнения. После чего это окно можно закрыть и продолжить дальше писать код.

Далее на Рисунке 2.1 будет показана работа языка программирования на примере программы по поиску максимального числа из трех чисел.

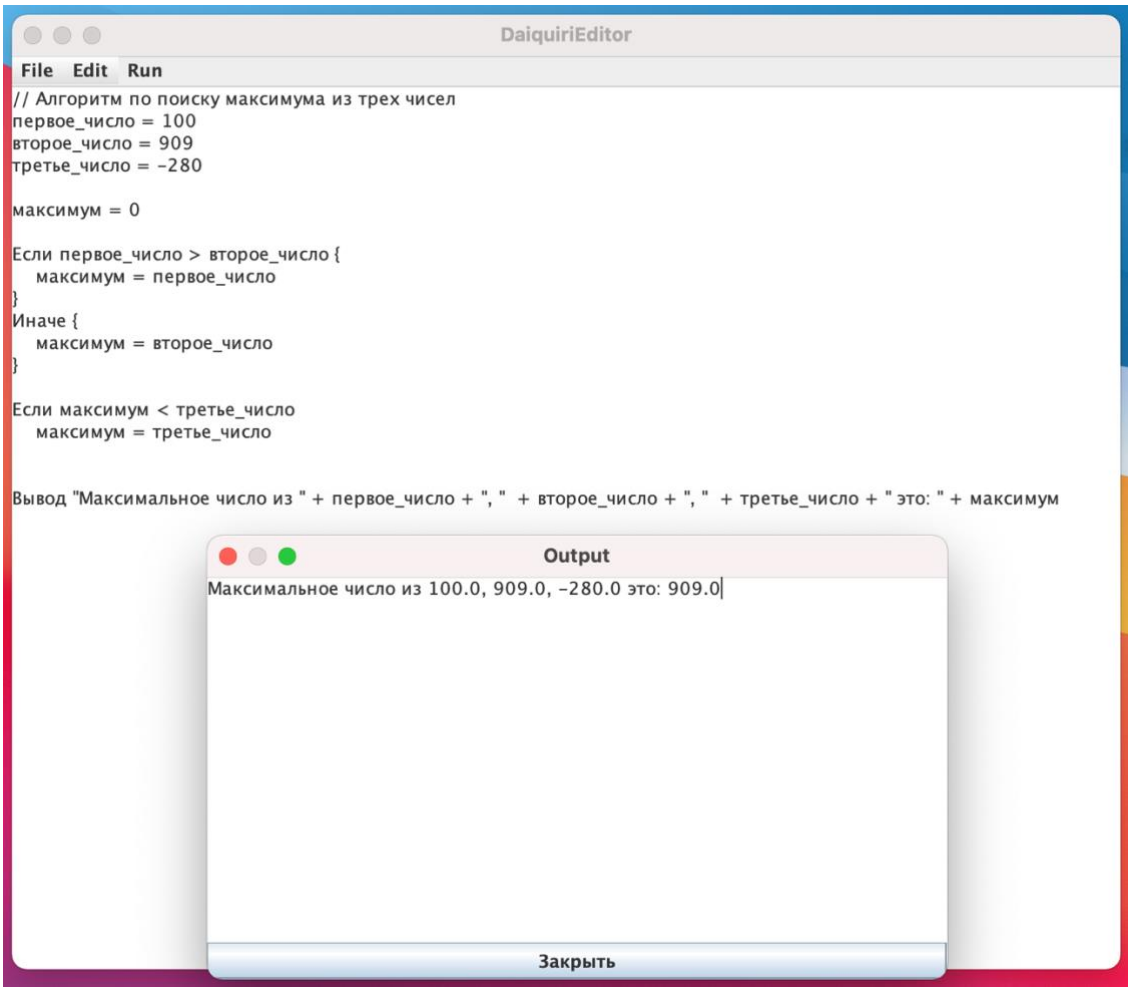

Рисунок 2.1 – Пример работы программы

ЗАКЛЮЧЕНИЕ

На протяжении всего процесса проектирования и создания программного продукта были получены практические навыки в области структур данных, которые используются для хранения и обработки различных типов данных.

Успешно выполнены поставленные задачи: создание программы, позволяющей осуществлять операции над числами, строками и работе с условиями. В результате выполнения данной курсовой работы был получен программный продукт, названный «Объектно-ориентированный язык программирования» на языке Java (программа находится в приложении).

При написании курсовой работы были пополнены теоретические сведения о современном техническом и программном обеспечении офисной деятельности; произведено знакомство с реферативными журналами и другими информационными источниками по объектно-ориентированному и системному программированию с целью анализа состояния решаемой задачи; усовершенствованы практические навыки работы с операционными системами, программными оболочками, разнообразными служебными и сервисными средствами, были углублены и закреплены знания по алгоритмизации, программированию и решению в интегрированной визуальной среде программирования. Получен опыт в сфере разработки десктоп приложений с графическим интерфейсом.

Язык имеет конструкции на русском и поддерживает основные логические и математические операции, форматированный вывод и конкатенацию. Также имеет удобный редактор кода. Такой инструмент мог бы стать полезным, для начало обучения программированию, так как прост в использовании и изучении.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Васильев А. А. Объектно-ориентированное программирование на Python. Издательство: Наука и Техника. Санкт-Петербург, 2019г. 543 стр.

2. Блог для программистов [Электронный ресурс] – URL: https://proglib.io/p/your-own-programming-language (дата обращения 21.03.2023).

3. Журнал Яндекс практикума [Электронный ресурс]. URL: https://thecode.media/parsing-2/ (дата обращения 23.03.2023).

4. Русскоязычный веб-сайт в формате системы тематических коллективных блогов (именуемых хабами) с элементами новостного сайта, [Электронный ресурс]. URL: https://habr.com/ru (дата обращения 02.04.2023).

ПРИЛОЖЕНИЯ

Приложение А – Листинг программы.

Приложение А.

Листинг А.1 – Класс Main, содержащий функцию main - функцию запуска программы.

```
package org.example;

import javax.swing.text.BadLocationException;
import java.io.*;

public class Main {
public static void main(String[] args) throws BadLocationException {
Editor e = new Editor();
    }
}
```

Листинг А.2 – Класс Editor, реализующий графический интерфейс и логику работы редактора кода.

```
package org.example; // Java Program to create a text editor using java
import org.example.ast.Statement;
import org.example.parser.Interpreter;
import org.example.parser.Lexer;
import org.example.parser.Parser;
import org.example.parser.Token;

import java.awt.*;
import javax.swing.*;
import java.io.*;
import java.awt.event.*;
import java.util.List;
import javax.swing.plaf.metal.*;
import javax.swing.text.*;

class Editor extends JFrame implements ActionListener {
    // Text component
     JTextArea t;
    private JTextField textField;
    private JTextArea textArea;

    // Frame
    JFrame f;
```

Листинг А.3 – Продолжение класса Editor, конструктор.

```
Editor() throws BadLocationException {
    // Create a frame
    f = new JFrame("DaiquiriEditor");

    try {
        // Set metal look and feel
        UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

        // Set theme to ocean
        MetalLookAndFeel.setCurrentTheme(new OceanTheme());
    } catch (Exception e) {
    }

    // Text component
    t = new JTextArea();


    // Create a menubar
    JMenuBar mb = new JMenuBar();

    // Create amenu for menu
    JMenu m1 = new JMenu("File");

    // Create menu items
    JMenuItem mi1 = new JMenuItem("New");
    JMenuItem mi2 = new JMenuItem("Open");
    JMenuItem mi3 = new JMenuItem("Save");
    JMenuItem mi9 = new JMenuItem("Print");

    // Add action listener
    mi1.addActionListener(this);
    mi2.addActionListener(this);
    mi3.addActionListener(this);
    mi9.addActionListener(this);

    m1.add(mi1);
    m1.add(mi2);
    m1.add(mi3);
    m1.add(mi9);

    // Create amenu for menu
    JMenu m2 = new JMenu("Edit");

    // Create menu items
    JMenuItem mi4 = new JMenuItem("cut");
    JMenuItem mi5 = new JMenuItem("copy");
    JMenuItem mi6 = new JMenuItem("paste");
```

Листинг А.4 – Продолжение класса Editor, продолжение конструктора.

```
    // Add action listener
    mi4.addActionListener(this);
    mi5.addActionListener(this);
    mi6.addActionListener(this);

    m2.add(mi4);
    m2.add(mi5);
    m2.add(mi6);

    JMenuItem mr = new JMenuItem("Run");



    mr.addActionListener(this);

    mb.add(m1);
    mb.add(m2);
    mb.add(mr);

    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setJMenuBar(mb);
    f.add(t);
    f.setSize(500, 500);
    f.show();

}
```

Листинг А.5 – Продолжение класса Editor, метод реагирования на нажатие кнопок.

```
// If a button is pressed
public void actionPerformed(ActionEvent e) {
    String s = e.getActionCommand();

    if (s.equals("cut")) {
        t.cut();
    } else if (s.equals("copy")) {
        t.copy();
    } else if (s.equals("paste")) {
        t.paste();
    } else if (s.equals("Save")) {
        // Create an object of JFileChooser class
        JFileChooser j = new JFileChooser("f:");

        // Invoke the showsSaveDialog function to show the save dialog
        int r = j.showSaveDialog(null);

        if (r == JFileChooser.APPROVE_OPTION) {

            // Set the label to the path of the selected directory
            File fi = new File(j.getSelectedFile().getAbsolutePath());

            try {
                // Create a file writer
                FileWriter wr = new FileWriter(fi, false);

                // Create buffered writer to write
                BufferedWriter w = new BufferedWriter(wr);

                // Write
                w.write(t.getText());

                w.flush();
                w.close();
            } catch (Exception evt) {
                JOptionPane.showMessageDialog(f, evt.getMessage());
            }
        }
            // If the user cancelled the operation
    else
        JOptionPane.showMessageDialog(f, "the user cancelled the operation");
}
else if (s.equals("Print")) {
    try {
        // print the file
        t.print();
    } catch (Exception evt) {
        JOptionPane.showMessageDialog(f, evt.getMessage());
    }
```

Листинг А.6 – Продолжение класса Editor, продолжение метода реагирования на нажатие кнопок.

```
} else if (s.equals("Open")) {
    // Create an object of JFileChooser class
    JFileChooser j = new JFileChooser("f:");

    // Invoke the showsOpenDialog function to show the save dialog
    int r = j.showOpenDialog(null);

    // If the user selects a file
    if (r == JFileChooser.APPROVE_OPTION) {
        // Set the label to the path of the selected directory
        File fi = new File(j.getSelectedFile().getAbsolutePath());

        try {
            // String
            String s1 = "", sl = "";

            // File reader
            FileReader fr = new FileReader(fi);

            // Buffered reader
            BufferedReader br = new BufferedReader(fr);

            // Initialize sl
            sl = br.readLine();

            // Take the input from the file
            while ((s1 = br.readLine()) != null) {
                sl = sl + "\n" + s1;
            }

                        // Set the text
             t.setText(sl);
             } catch (Exception evt) {
        JOptionPane.showMessageDialog(f, evt.getMessage());
             }
}
        // If the user cancelled the operation
        else
            JOptionPane.showMessageDialog(f, "the user cancelled the operation");
    } else if (s.equals("New")) {
        t.setText("");

    } else if (s.equals("Run")) {
        String programOuput = executeProgram();
        createOutputPanel(programOuput);
    }
}
```

Листинг А.7 – Продолжение класса Editor, методы необходимые для исполнения программы и вывода результата.

```
public String executeProgram() {
        String output;
        try {
            String input = t.getText();
            // Run program
            final List<Token> tokens = new Lexer(input).tokenize();
//              for (Token token : tokens)
//                  System.out.println(token.toString());
            final Statement program = new Parser(tokens).parse();
//              System.out.println(program.toString());
            Interpreter interpreter = new Interpreter(program);
            output = interpreter.execute();
        } catch (Exception exception) {
            output = exception.toString();
        }
        return output;
}

public void addTextToTextArea(JTextArea textArea, String text) {
        textArea.setText(text);
}

public void createOutputPanel(String text) {
        JDialog dialog = new JDialog();
        JTextArea textArea = new JTextArea();
        JButton closeButton = new JButton("Закрыть");
        closeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                dialog.dispose();
            }
        });
        JPanel panel = new JPanel(new BorderLayout());
        panel.add(textArea, BorderLayout.CENTER);
        panel.add(closeButton, BorderLayout.SOUTH);
        dialog.add(panel);
        dialog.pack();
        dialog.setSize(500, 300);
        dialog.setTitle("Output");
        dialog.setVisible(true);

        addTextToTextArea(textArea, text);
    }

}
```

Листинг А.8 – Класс Token.

```
package org.example.parser;

public final class Token {
    private TokenType type;
    private String text;

    public Token(TokenType type, String text) {
        this.text = text;
        this.type = type;
    }
    public Token() {
    }

    public TokenType getType() {
        return type;
    }

    @Override
    public String toString() {
        return type + " " + text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public String getText() {
        return text;
    }
}
```

Листинг А.9 – Енам TokenType.

```
package org.example.parser;

public enum TokenType {
    NUMBER,
    WORD,
    TEXT,

    // Keywords
    PRINT,
    IF,
    ELSE,
    PLUS,
    MINUS,
    STAR,
    SLASH,
    EQ,
    EQEQ,
    EXCL,
    EXCLEQ,
    LT,
    LTEQ,
    GT,
    GTEQ,

    BAR,
    BARBAR,
    AMP,
    AMPAMP,

    LPAREN,
    RPAREN,
    LBRACE,
    RBRACE,

    EOF
}
```

Листинг А.10 – Класс Lexer.

```
package org.example.parser;

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

public final class Lexer {
    private final String input;
    private final int length;
    private final List<Token> tokens;
    private int pos;
    private static final String OPERATOR_CHARS = "+-*/=<>(){}!&|";
    private static final Map<String, TokenType> OPERATORS;
    static {
        OPERATORS = new HashMap<>();
        OPERATORS.put("+", TokenType.PLUS);
        OPERATORS.put("-", TokenType.MINUS);
        OPERATORS.put("*", TokenType.STAR);
        OPERATORS.put("/", TokenType.SLASH);

        OPERATORS.put("(", TokenType.LPAREN);
        OPERATORS.put(")", TokenType.RPAREN);
        OPERATORS.put("{", TokenType.LBRACE);
        OPERATORS.put("}", TokenType.RBRACE);
        OPERATORS.put("=", TokenType.EQ);
        OPERATORS.put("<", TokenType.LT);
        OPERATORS.put(">", TokenType.GT);

        OPERATORS.put("!", TokenType.EXCL);
        OPERATORS.put("&", TokenType.AMP);
        OPERATORS.put("|", TokenType.BAR);

        OPERATORS.put("==", TokenType.EQEQ);
        OPERATORS.put("!=", TokenType.EXCLEQ);
        OPERATORS.put("<=", TokenType.LTEQ);
        OPERATORS.put(">=", TokenType.GTEQ);

        OPERATORS.put("&&", TokenType.AMPAMP);
        OPERATORS.put("||", TokenType.BARBAR);

    }
```

Листинг А.11 – Продолжение класса Lexer, конструктор, основной метод токенизации tokenize и метод токенизации числа tokenizeNumber.

```
public Lexer(String input) {
    this.input = input;
    length = input.length();
    tokens = new ArrayList<>();
}
public List<Token> tokenize() {
    while (pos < length) {
        final char current = peek(0);
        if (Character.isDigit(current)) tokenizeNumber();
        else if (current == '"') tokenizeText();
        else if (Character.isLetter(current)) tokenizeWord();
        else if (OPERATOR_CHARS.indexOf(current) != -1) {
            tokenizeOperator();
        } else {
            // whitespaces
            next();
        }

    }
    return tokens;
}
private void tokenizeNumber() {
    final StringBuilder buffer = new StringBuilder();
    char current = peek(0);
    while (true) {
        if (current == '.') {
            if (buffer.indexOf(".") != -1)
                throw new RuntimeException("invalid float number");
        } else if (!Character.isDigit(current)) {
            break;
        }
        buffer.append(current);
        current = next();
    }
    addToken(TokenType.NUMBER, buffer.toString());
}
```

Листинг А.12 – Продолжение класса Lexer, метод токенизации операторов tokenizeOperator, в котором реализована обработка комментариев в коде.

```
private void tokenizeOperator() {
    char current = peek(0);
    if (current == '/') {
        if (peek(1) == '/') {
            next();
            next();
            tokenizeComment();
            return;
        } else if (peek(1) == '*') {
            next();
            next();
            tokenizeMultilineComment();
            return;
        }
    }
    final StringBuilder buffer = new StringBuilder();
    while (true) {
        final String text = buffer.toString();
        if (!OPERATORS.containsKey(text + current) && !text.isEmpty()) {
            addToken(OPERATORS.get(text));
            return;
        }
        buffer.append(current);
        current = next();
    }
}
```

Листинг А.13 – Продолжение класса Lexer, метод токенизации имен и ключевых слов языка tokenizeWord.

```
private void tokenizeWord() {
    final StringBuilder buffer = new StringBuilder();
    char current = peek(0);
    while (true) {
        if (!Character.isLetterOrDigit(current) && (current != '_') && (current != '$')) {
            break;
        }
        buffer.append(current);
        current = next();
    }
    String word = buffer.toString();
    switch (word) {
        case "Вывод" : {
            addToken(TokenType.PRINT);
            break;
        }
        case "Если" : {
            addToken(TokenType.IF);
            break;
        }
        case "Иначе" : {
            addToken(TokenType.ELSE);
            break;
        }
        default: {
            addToken(TokenType.WORD, word);
            break;
        }
    }
}
```

Листинг А.14 – Продолжение класса Lexer, метод токенизации текста tokenizeText, а также метод токенизации однострочных комментариев tokenizeComment.

```
private void tokenizeText() {
    next(); // skip "
    final StringBuilder buffer = new StringBuilder();
    char current = peek(0);
    while (true) {
        if (current == '\\') {
            current = next();
            switch (current) {
                case '"': {
                    current = next();
                    buffer.append('"');
                    continue;
                }
                case 'n': {
                    current = next();
                    buffer.append('\n');
                    continue;
                }
                case 't': {
                    current = next();
                    buffer.append('\t');
                    continue;
                }
            }
            buffer.append('\\');
            continue;
        }
        if (current == '"') break;
        buffer.append(current);
        current = next();
    }
    next(); // skip "

    addToken(TokenType.TEXT, buffer.toString());
}

private void tokenizeComment() {
    char current = peek(0);
    while ("\r\n\0".indexOf(current) == -1) {
        current = next();
    }
}
```

Листинг А.15 – Продолжение класса Lexer, метод токенизации многострочных комментариев tokenizeMultilineComment, а также методы для итерации по тексту программы и взаимодействия со списком токенов.

```
private void tokenizeMultilineComment() {
        char current = peek(0);
        while (true) {
            if (current == '\0') throw new RuntimeException("Missing close tag");
            if (current == '*' && peek(1) == '/')
                break;
            current = next();
        }
        next(); // *
        next(); // /
    }
    private char next() {
        pos++;
        return peek(0);
    }
    private char peek(int relativePosition) { // peek(0) - текущий символ
        final int position = pos + relativePosition;
        if (position >= length) return '\0';
        return input.charAt(position);
    }
    private void addToken(TokenType type) {
        addToken(type, "");
    }
    private void addToken(TokenType type, String text) {
        tokens.add(new Token(type, text));
    }

}
```

Листинг А.16 – Класс Parser.

```
package org.example.parser;

import org.example.ast.*;

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

public final class Parser {

    private static final Token EOF = new Token(TokenType.EOF, "");

    private final List<Token> tokens;
    private final int size;

    private int pos;

    public Parser(List<Token> tokens) {
        this.tokens = tokens;
        size = tokens.size();
    }

    public Statement parse() {
        final BlockStatement result = new BlockStatement();
        while (!match(TokenType.EOF)) {
            result.add(statement());
        }
        return result;
    }
    private Statement block() {
        final BlockStatement block = new BlockStatement();
        consume(TokenType.LBRACE);
        while (!match(TokenType.RBRACE)) {
            block.add(statement());
        }
        return block;
    }
    private Statement statementOrBlock() {
    if (get(0).getType() == TokenType.LBRACE) return block();
    return statement();
    }
```

Листинг А.17 – Продолжение класса Parser.

```
private Statement statement() {
    if (match(TokenType.PRINT)) {
        return new PrintStatement(expression());
    }
    if (match(TokenType.IF)) {
        return ifElse();
    }
    return assignmentStatement();
}

private Statement assignmentStatement() {
    // WORD EQ
    final Token current = get(0);
    if (match(TokenType.WORD) && get(0).getType() == TokenType.EQ) {
        final String variable = current.getText();
        consume(TokenType.EQ);
        return new AssignmentStatement(variable, expression());
    }
    throw new RuntimeException("Unknown statement");
}

private Statement ifElse() {
    final Expression condition = expression();
    final Statement ifStatement = statementOrBlock();
    final Statement elseStatement;
    if (match(TokenType.ELSE)) {
        elseStatement = statementOrBlock();
    } else {
        elseStatement = null;
    }
    return new ifStatement(condition, ifStatement, elseStatement);
}

private Expression expression() {
    return logicalOr();
}

private Expression logicalOr() {
    Expression result = logicalAnd();
    while (true) {
        if (match(TokenType.BARBAR)) {
            result = new ConditionalExpression(ConditionalExpression.Operator.OR, result,
logicalAnd());
            continue;
        }
        break;
    }
    return result;
}
```

Листинг А.18 – Продолжение класса Parser.

```
private Expression logicalAnd() {
    Expression result = equality();

    while (true) {
        if (match(TokenType.AMPAMP)) {
            result = new ConditionalExpression(ConditionalExpression.Operator.AND, result,
equality());
            continue;
        }
        break;
    }

    return result;
}

private Expression equality() {
    Expression result = conditional();

    if (match(TokenType.EQEQ)) {
        return new ConditionalExpression(ConditionalExpression.Operator.EQUALS, result,
conditional());
    }
    if (match(TokenType.EXCLEQ)) {
        return new ConditionalExpression(ConditionalExpression.Operator.NOT_EQUALS, result,
conditional());
    }

    return result;
}
```

Листинг А.19 – Продолжение класса Parser.

```
private Expression conditional() {
    Expression result = additive();
    while (true) {
        if (match(TokenType.LT)) {
            result = new ConditionalExpression(ConditionalExpression.Operator.LT, result,
additive());
            continue;
        }
        if (match(TokenType.LTEQ)) {
            result = new ConditionalExpression(ConditionalExpression.Operator.LTEQ, result,
additive());
            continue;
        }
        if (match(TokenType.GT)) {
            result = new ConditionalExpression(ConditionalExpression.Operator.GT, result,
additive());
            continue;
        }
        if (match(TokenType.GTEQ)) {
            result = new ConditionalExpression(ConditionalExpression.Operator.GTEQ, result,
additive());
            continue;
        }
        break;
    }
    return result;
}

private Expression additive() {
    Expression result = multiplicative();
    while (true) {
        if (match(TokenType.PLUS)) {
            result = new BinaryExpression('+', result, multiplicative());
            continue;
        }
        if (match(TokenType.MINUS)) {
            result = new BinaryExpression('-', result, multiplicative());
            continue;
        }
        break;
    }
    return result;
}
```

Листинг А.20 – Продолжение класса Parser.

```
private Expression multiplicative() {
    Expression result = unary();

    while (true) {
        // 2 * 6 / 3
        if (match(TokenType.STAR)) {
            result = new BinaryExpression('*', result, unary());
            continue;
        }
        if (match(TokenType.SLASH)) {
            result = new BinaryExpression('/', result, unary());
            continue;
        }
        break;
    }

    return result;
}

private Expression unary() {
    if (match(TokenType.MINUS)) {
        return new UnaryExpression('-', primary());
    }
    if (match(TokenType.PLUS)) {
        return primary();
    }
    return primary();
}
```

Листинг А.21 – Продолжение класса Parser.

```
private Expression primary() {
        final Token current = get(0);
        if (match(TokenType.NUMBER)) {
            return new ValueExpression(Double.parseDouble(current.getText()));
        }
        if (match(TokenType.WORD)) {
            return new VariableExpression(current.getText());
        }
        if (match(TokenType.TEXT)) {
            return new ValueExpression(current.getText());
        }
        if (match(TokenType.LPAREN)) {
            Expression result = expression();
            match(TokenType.RPAREN);
            return result;
        }
        throw new RuntimeException("Unknown expression");
    }

    private Token consume(TokenType type) {
        final Token current = get(0);
        if (type != current.getType()) throw new RuntimeException("Token " + current + " doesn't
match " + type);
        pos++;
        return current;
    }

    private boolean match(TokenType type) {
        final Token current = get(0);
        if (type != current.getType()) return false;
        pos++;
        return true;
    }

    private Token get(int relativePosition) {
        final int position = pos + relativePosition;
        if (position >= size) return EOF;
        return tokens.get(position);
    }
}
```

Листинг А.22 – Класс Interpreter.

```
package org.example.parser;

import org.example.ast.Statement;

public final class Interpreter {
    private final Statement program;

    public Interpreter(Statement program) {
        this.program = program;
    }
    public String execute() {
        String result;
        try {
            result = program.execute();
        } catch (Exception e) {
            throw new RuntimeException("The program was aborted due to an error");
        }
        return result;
    }
}
```

Листинг А.23 – Интерфейс Value.

```
package org.example.lib;

public interface Value {
    double asNumber();
    String asString();

}
```

Листинг А.24 – Класс Variables.

```
package org.example.lib;

import java.util.HashMap;
import java.util.Map;

public final class Variables {
    public static final NumberValue ZERO = new NumberValue(0);
    private static Map<String, Value> variables;
    static {
        variables = new HashMap<>();
        variables.put("PI", new NumberValue(Math.PI));
        variables.put("ПИ", new NumberValue(Math.PI));
        variables.put("E", new NumberValue(Math.E));
        variables.put("E", new NumberValue(Math.E));
        variables.put("GOLDEN_RATIO", new NumberValue(1.618));
    }
    public static boolean isExists(String key) {
        return variables.containsKey(key);
    }
    public static Value get(String key) {
        if (!isExists(key)) return ZERO;
        return variables.get(key);

    }
    public static void set(String key, Value value) {
        variables.put(key, value);
    }
}
```

Листинг А.25 – Класс StringValue, реализующий интерфейс Value.

```
package org.example.lib;

public class StringValue implements Value{
    private final String value;

    public StringValue(String value) {
        this.value = value;
    }

    @Override
    public double asNumber() {
        try {
            return Double.parseDouble(value);
        } catch (NumberFormatException e) {
            return 0;
        }
    }

    @Override
    public String asString() {
        return value;
    }

    @Override
    public String toString() {
        return asString();
    }
}
```

Листинг А.26 – Класс NumberValue, реализующий интерфейс Value.

```
package org.example.lib;

public class NumberValue implements Value{
    private final double value;

    public NumberValue(double value) {
        this.value = value;
    }
    public NumberValue(boolean value) {
            this.value = value ? 1 : 0;
        }

    @Override
    public double asNumber() {
        return value;
    }

    @Override
    public String toString() {
        return asString();
    }

    @Override
    public String asString() {
        return Double.toString(value);
    }
}
```

Листинг А.27 – Интерфейсы Expression и Statement.

```
package org.example.ast;

import org.example.lib.Value;

public interface Expression {
    Value eval();
}
```

```
package org.example.ast;

public interface Statement {
    String execute();
}
```

Листинг А.28 – Класс VariableExpression, реализующий интерфейс Expression.

```
package org.example.ast;

import org.example.lib.Value;
import org.example.lib.Variables;

public class VariableExpression implements Expression {
    private final String name;

    public VariableExpression(String name) {
        this.name = name;
    }

    @Override
    public Value eval() {
        if (!Variables.isExists(name))
            throw new RuntimeException("Constant doesn't exist");
        return Variables.get(name);
    }

    @Override
    public String toString() {
        return String.format("%s %f", name, Variables.get(name).asNumber());
    }
}
```

Листинг А.29 – Класс ValueExpression, реализующий интерфейс Expression.

```
package org.example.ast;

import org.example.lib.NumberValue;
import org.example.lib.StringValue;
import org.example.lib.Value;

public class ValueExpression implements Expression {
    private final Value value;

    public ValueExpression(double value) {
        this.value = new NumberValue(value);
    }
    public ValueExpression(String value) {
        this.value = new StringValue(value);
    }

    @Override
    public Value eval() {
        return value;
    }

    @Override
    public String toString() {
        return value.asString();
    }
}
```

Листинг А.30 – Класс UnaryExpression, реализующий интерфейс Expression.

```
package org.example.ast;

import org.example.lib.NumberValue;
import org.example.lib.Value;

public final class UnaryExpression implements Expression {
   private final Expression expr1;
   private final char operation;
   @Override
   public Value eval() {
      switch (operation) {
         case '-': return new NumberValue(-expr1.eval().asNumber());
         case '+':
         default:
            return new NumberValue(expr1.eval().asNumber());
      }

   }

   public UnaryExpression( char operation, Expression expr1) {
      this.operation = operation;
      this.expr1 = expr1;
   }
}
```

Листинг А.31 – Класс BinaryExpression, реализующий интерфейс Expression.

```
package org.example.ast;
import org.example.lib.NumberValue;
import org.example.lib.StringValue;
import org.example.lib.Value;

public class BinaryExpression implements Expression {
   private final Expression exp1, exp2;
   private final char operation;
   public BinaryExpression(char operation, Expression exp1, Expression exp2) {
      this.operation = operation;
      this.exp1 = exp1;
      this.exp2 = exp2;
   }
   @Override
   public Value eval() {
      final Value value1 = exp1.eval();
      final Value value2 = exp2.eval();
      if (value1 instanceof StringValue) {
         final String string1 = value1.asString();
         switch (operation) {
            case '*': {
               final int iterations = (int) value2.asNumber();
               final StringBuilder buffer = new StringBuilder();
               for (int i = 0; i < 10; i++) {
                  buffer.append(string1);
               }
               return new StringValue(buffer.toString());
            }
            case '+':
            default:
               return new StringValue(string1 + value2.asString());
         }
      }
      final double number1 = value1.asNumber();
      final double number2 = value2.asNumber();
      switch (operation) {
         case '-': return new NumberValue(number1 - number2);
         case '*': return new NumberValue(number1 * number2);
         case '/': return new NumberValue(number1 / number2);
         case '+':
         default:
            return new NumberValue(number1 + number2);
      }
   }
   @Override
   public String toString() {
      return String.format("[%s %c %s]", exp1, operation, exp2);
   }
}
```

Листинг А.32 – Класс ConditionalExpression, реализующий интерфейс Expression.

```
package org.example.ast;

import org.example.lib.NumberValue;
import org.example.lib.StringValue;
import org.example.lib.Value;

public class ConditionalExpression implements Expression {
  public static enum Operator {
      PLUS("+"),
      MINUS("-"),
      MULTIPLY("*"),
      DIVIDE("/"),

      EQUALS("=="),
      NOT_EQUALS("!="),

      LT("<"),
      LTEQ("<="),
      GT(">"),
      GTEQ(">="),

      AND("&&"),
      OR("||");

      private final String name;
      private Operator(String name) {
          this.name = name;
      }

     public String getName() {
         return name;
     }
  }
   private final Expression exp1, exp2;
   private final Operator operation;

   public ConditionalExpression(Operator operation, Expression exp1, Expression exp2) {
      this.operation = operation;
      this.exp1 = exp1;
      this.exp2 = exp2;
   }
```

Листинг А.33 – Продолжение класса ConditionalExpression, реализующего интерфейс Expression.

```
    @Override
    public Value eval() {
        final Value value1 = exp1.eval();
        final Value value2 = exp2.eval();

        double number1, number2;
        if (value1 instanceof StringValue) {
            number1 = value1.asString().compareTo(value2.asString());
            number2 = 0;
        } else {
            number1 = value1.asNumber();
            number2 = value2.asNumber();
        }
        boolean result;
        switch (operation) {
            case LT: result = number1 < number2; break;
            case LTEQ: result = number1 <= number2; break;
            case GT: result =  number1 > number2; break;
            case GTEQ: result =  number1 >= number2; break;
            case NOT_EQUALS: result =  number1 != number2; break;

            case AND: result = (number1 != 0) && (number2 != 0);
            case OR: result = (number1 != 0) || (number2 != 0);
            case EQUALS:
            default:
                result = number1 == number2; break;
        }
        return new NumberValue(result);
    }

    @Override
    public String toString() {
        return String.format("[%s %s %s]", exp1, operation.getName(), exp2);
    }
}
```

Листинг А.34 –Класс BlockStatement, реализующий интерфейс Statement.

```
package org.example.ast;

import java.nio.Buffer;
import java.util.ArrayList;
import java.util.List;

public class BlockStatement implements Statement{
    private final List<Statement> statements;
    private StringBuilder result;
    public BlockStatement() {
        statements = new ArrayList<>();
        result = new StringBuilder();
    }


    public void add(Statement statement) {
        statements.add(statement);
    }
    @Override
    public String execute() {
        for (Statement statement : statements) {
            result.append(statement.execute());
        }
        return result.toString();
    }

    @Override
    public String toString() {
        final StringBuilder result = new StringBuilder();
        for (Statement statement : statements) {
            result.append(statement.toString()).append(System.lineSeparator());
        }
        return result.toString();
    }
}
```

Листинг А.35 –Класс AssignmentStatement, реализующий интерфейс Statement.

```
package org.example.ast;

import org.example.lib.Value;
import org.example.lib.Variables;

public final class AssignmentStatement implements Statement {

    private final String variable;
    private final Expression expression;

    public AssignmentStatement(String variable, Expression expression) {
        this.variable = variable;
        this.expression = expression;
    }

    @Override
    public String execute() {
        final Value result = expression.eval();
        Variables.set(variable, result);
        return "";
    }

    @Override
    public String toString() {
        return String.format("%s = %s", variable, expression);
    }
}
```

Листинг А.36 –Класс ifStatement, реализующий интерфейс Statement.

```
package org.example.ast;

public class ifStatement implements Statement{
    private final Expression expression;
    private final Statement ifStatement;
    private final Statement elseStatement;

    public ifStatement(Expression expression, Statement ifStatement, Statement elseStatement) {
        this.expression = expression;
        this.ifStatement = ifStatement;
        this.elseStatement = elseStatement;
    }

    @Override
    public String execute() {
        final double result = expression.eval().asNumber();
        if (result != 0) {
            ifStatement.execute();
        } else if (elseStatement != null) {
            elseStatement.execute();
        }
        return "";
    }

    @Override
    public String toString() {
        final StringBuilder result = new StringBuilder();
        result.append("if ").append(expression).append(' ').append(ifStatement);
        if (elseStatement != null) {
            result.append("\nelse ").append(elseStatement);
        }
        return result.toString();
    }
}
```

Листинг А.37 –Класс PrintStatement, реализующий интерфейс Statement.

```
package org.example.ast;

public class PrintStatement implements Statement {
    private final Expression expression;
    private String result;
    public PrintStatement(Expression expression) {
        this.expression = expression;
    }
    public String getResult() {
        return result;
    }
    @Override
    public String execute() {
        return expression.eval().asString();
//        System.out.println(result);
    }

    @Override
    public String toString() {
        return "print " + expression;
    }
}
```

Листинг А.38 – Файл pom.xml для сборки проекта при помощи Maven

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>DaiquiriLanguage</artifactId>
    <version>1.0-SNAPSHOT</version>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.3.0</version>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>org.example.Main</mainClass>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <source>20</source>
                    <target>20</target>
                    <encoding>utf-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```