UNIVERSIDADE FEDERAL DE LAVRAS
COMPILADORES
MAURICIO RONNY DE ALMEIDA SOUZA

BRUNO CRESPO FERREIRA
FRANCISCO AFONSO DE OLIVEIRA NETO
HEITOR RODRIGUES SABINO
RANULFO MASCARI NETO

Relatório Final

LAVRAS
2023

1. Visão geral da linguagem

O nome da linguagem é UaiScript.

- Tipos:
    - Inteiro: cado
    - Real: tiquim
    - Texto: trem
    - Booleano: paia (verdadeiro: god, falso: bigode)

- Abertura e fechamento de bloco:
    - Abertura: {
    - Fechamento: }

- Operadores Aritméticos:
    - Adição: botaMais
    - Subtração: tira
    - Multiplicação: vezes
    - Divisão: partidoEm
    - Módulo: restinDe
    - Exponenciação: elevado

- Operadores Relacionais:
    - Igualdade: igualzin
    - Diferença: nadaveh
    - Maior que: maioh
    - Menor que: menoh

- Operadores Lógicos:
    - And: e
    - Or: ou
    - Not: nop

- Operadores de incremento e decremento:
    - Incremento: maisUm

    - Decremento: tiraUm

- Operador de Atribuição: <-|

- Fim de linha: !

- Estruturas:
    - Condicional if: truco
    - Condicional else if: meiPau
    - Condicional else: corri
    - Repetição while: todaVida

- Funções específicas:
    - Ler uma entrada qualquer: ler
    - Imprimir na tela uma variável qualquer: mostrar

- Parênteses:
    - Abre parênteses: (
    - Fecha parênteses:)

- Representações:
    - Variável: [a..z] ([a..zA..Z]+[0..9])*
    - Número Inteiro: [0..9]+
    - Número Real: [0..9]+ . [0..9]+

2. Definição léxica da linguagem

Padrão	Tipo de Lexema	Sigla e Tokens
Os próprios lexemas	Palavras-chave: cado, tiquim, trem, paia, truco, meiPau, corri, todaVida, ler, mostrar.	Cada conjunto de tokens possui uma classe/sigla independente. Exemplos: <cado> <tiquim> <trem> <paia> <truco> <meiPau> <corri> <todaVida> <ler> <mostrar>
Os próprios lexemas	tipo falso ou verdadeiro: god (verdadeiro), bigode (falso)	Bool Exemplos: <BoolValue, god> <BoolValue, bigode>
Os próprios lexemas	Operadores aritméticos: +, -, *, /, %, ^ +: botaMais -: tira *: vezes /: partidoEm %: restinDe ^: elevado	OpArit Exemplos: <OpArit, botaMais> <OpArit, tira> <OpArit, vezes> <OpArit, partidoEm> <OpArit, restinDe> <OpArit, elevado>

Os próprios lexemas	Operadores relacionais: =, ≠, >, < =: igualzin ≠: nadaveh >: maioh <: menoh	OpRel Exemplos: <OpRel, igualzin> <OpRel, nadaveh> <OpRel, maioh> <OpRel, menoh>
Os próprios lexemas	Operadores lógicos: and, or, not and: e or: ou not: nop	OpLog Exemplos: <OpLog, e> <OpLog, ou> <OpLog, nop>
Os próprios lexemas	Operadores de incremento e decremento: ++, -- ++: maisUm --: tiraUm	OpCrem Exemplos: <OpCrem, maisUm> <OpCrem, tiraUm>
O próprio lexema	Operador de atribuição: <-|	OpAtrib Exemplo: <OpAtrib>
Os próprios lexemas	Parênteses: (,)	AP / FP Exemplos: <AP> <FP>
Os próprios lexemas	Chaves de bloco: {, }	AB / FB Exemplos:

		<AB> <FB>
O próprio lexema	Fim de linha: !	FL Exemplos: <FL>
Sequência de letras e números que começam com uma letra [a..zA..Z] ([a..zA..Z]+[0..9])*	Variável	VAR Exemplos: <VAR, variavel1> <VAR, valorFoda>
Sequência de dígitos (sem ponto) [0..9]+	Número Inteiro	NumI Exemplos: <NumI, 43> <NumI, 7>
Sequência de dígitos (com ponto) [0..9]+ . [0..9]+	Número Real	NumR Exemplos: <NumR, 54.432> <NumR, 7.777>
Sequências de caracteres envolta por aspas “Σ*”	Cadeia	Str Exemplos: <Str, “arroz”> <Str, “abroba”>

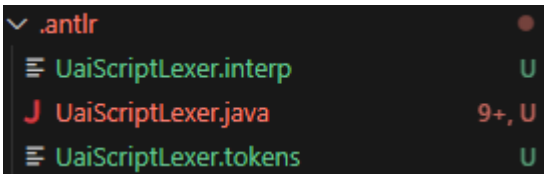
3. Implementação do analisador léxico

O código para a geração do analisador léxico é representado a seguir:

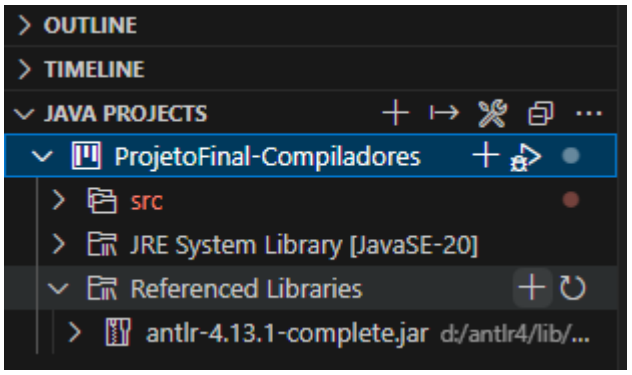
```
grammar UaiScript;

/***** Definição Léxica (tokens) *****/
// Tipos
Cado: 'cado';
Tiquim: 'tiquim';
Trem: 'trem';
Paia: 'paia';
// Estruturas
Truco: 'truco';
MeiPau: 'meiPau';
Corri: 'corri';
TodaVida: 'todaVida';
// Funções Específicas
Ler: 'ler';
Mostrar: 'mostrar';
// Verdadeiro e Falso
BoolValue: 'god' | 'bigode';
// Operadores Aritméticos, respectivamente: + - * / % ^
OpArit: 'botaMais' | 'tira' | 'vezes' | 'partidoEm' | 'restinDe' |
'elevado';
// Operadores Relacionais, respectivamente: = ≠ > <
OpRel: 'igualzin' | 'nadaveh' | 'maioh' | 'menoh';
// Operadores Lógicos, respectivamente: and or not
OpLog: 'e' | 'ou' | 'nop';
// Operadores de Incremento e Decremento, respectivamente ++ --
OpCrem: 'maisUm' | 'tiraUm';
// Operador de Atribuição
OpAtrib: '<-|';
// Parênteses
AP: '(';
FP: ')';
// Chaves de bloco
AB: '{';
FB: '}';
// Fim de linha
FL: '!';
// Identificadores das variáveis
VAR: LETRA(DIGITO|LETRA)*;
// Identificadores dos números inteiros e reais
```

```
NumI: DIGITO+;
NumR: DIGITO+'.'DIGITO+;
// Identificador das cadeias (baguio doido)
Str: '"' ('\\' ["\\] | ~["\\\r\n])* '"' ;
// Fragmentos auxiliares
fragment LETRA: [a-zA-Z];
fragment DIGITO: [0-9];
// Ignorar espaços, quebras de linha, etc
WS: [ \r\t\n]+ ->skip;
```

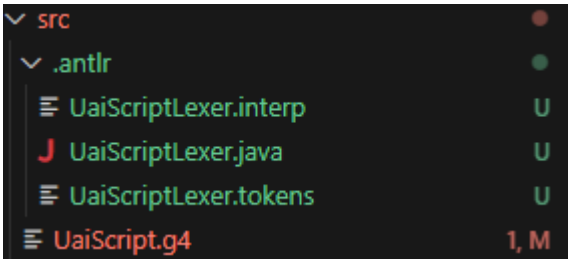
Ao utilizar o comando *antlr UaiScript.g4* ou, simplesmente dar Ctrl-S no Visual Studio (utilizando a extensão do Antlr), geram-se os arquivos:



Como é possível perceber o Lexer possui alguns erros, isso é porque mesmo com o Antlr instalado é preciso definir sua biblioteca externa. Como estamos realizando um projeto Java basta adicionarmos a biblioteca external, como a seguir (imagem já foi realizado o processo):



Com isso temos a seguinte situação:



O arquivo UaiScript.g4 estar “errado” é normal, pois ainda não definimos nenhuma regra para ele.

Ao tentar criar um classe nova para utilizar os comandos do antlr, encontrou-se dificuldades em utilizar a pasta “.antlr”, já que o Java não aceita pacotes que possuam o caractere ‘.’. Portanto, foi decidido inserir a pasta .antlr dentro do .gitignore e utilizar o comando *antlr UaiScript.g4* no terminal.

Foi criado um programa chamado ExemploLexer.java que analisa lexicalmente e retorna os tokens de um arquivo txt que deve estar presente na pasta casosTeste, veja o resultado para o caso teste “exemploSimples.txt” abaixo:

Token: <Cado>
    Linha: 1
    Pos Inicial: 0
    Pos Final: 3
Token: <VAR, abacaxi>
    Linha: 1
    Pos Inicial: 5
    Pos Final: 11
Token: <FL>
    Linha: 1
    Pos Inicial: 12
    Pos Final: 12
Token: <Cado>
    Linha: 2
    Pos Inicial: 15
    Pos Final: 18
Token: <VAR, valor>
    Linha: 2
    Pos Inicial: 20
    Pos Final: 24
Token: <OpAtrib>
    Linha: 2
    Pos Inicial: 26
    Pos Final: 28
Token: <NumI, 2>
    Linha: 2

Pos Inicial: 30
Pos Final: 30
Token: <FL>
Linha: 2
Pos Inicial: 31
Pos Final: 31
Token: <Trem>
Linha: 3
Pos Inicial: 34
Pos Final: 37
Token: <VAR, palavra>
Linha: 3
Pos Inicial: 39
Pos Final: 45
Token: <FL>
Linha: 3
Pos Inicial: 46
Pos Final: 46
Token: <VAR, abacaxi>
Linha: 5
Pos Inicial: 51
Pos Final: 57
Token: <OpAtrib>
Linha: 5
Pos Inicial: 59
Pos Final: 61
Token: <NumI, 50>
Linha: 5
Pos Inicial: 63
Pos Final: 64
Token: <FL>
Linha: 5
Pos Inicial: 65
Pos Final: 65

Token: <VAR, palavra>
Linha: 6
Pos Inicial: 68
Pos Final: 74
Token: <OpAtrib>
Linha: 6
Pos Inicial: 76
Pos Final: 78
Token: <Str, "abroba2">
Linha: 6
Pos Inicial: 80
Pos Final: 88
Token: <FL>
Linha: 6
Pos Inicial: 89
Pos Final: 89
Token: <VAR, abacaxi>
Linha: 8
Pos Inicial: 94
Pos Final: 100
Token: <OpAtrib>
Linha: 8
Pos Inicial: 102
Pos Final: 104
Token: <VAR, valor>
Linha: 8
Pos Inicial: 106
Pos Final: 110
Token: <OpArit, botaMais>
Linha: 8
Pos Inicial: 112
Pos Final: 119
Token: <VAR, abacaxi>
Linha: 8

Pos Inicial: 121
Pos Final: 127
Token: <FL>
Linha: 8
Pos Inicial: 128
Pos Final: 128
Token: <Mostrar>
Linha: 10
Pos Inicial: 133
Pos Final: 139
Token: <VAR, abacaxi>
Linha: 10
Pos Inicial: 141
Pos Final: 147
Token: <FL>
Linha: 10
Pos Inicial: 148
Pos Final: 148
Token: <Mostrar>
Linha: 11
Pos Inicial: 151
Pos Final: 157
Token: <VAR, palavra>
Linha: 11
Pos Inicial: 159
Pos Final: 165
Token: <FL>
Linha: 11
Pos Inicial: 166
Pos Final: 166

4. Definição sintática da linguagem

A gramática livre de contexto é demonstrada a seguir:

$programa \longrightarrow instrucao+\ EOF$

$instrucao \longrightarrow tipo\ VAR\ FL$
$instrucao \longrightarrow tipo\ VAR\ OpAtrib\ expressao\ FL$
$instrucao \longrightarrow VAR\ OpAtrib\ expressao\ FL$
$instrucao \longrightarrow VAR\ OpCrem\ FL$
$instrucao \longrightarrow Truco\ condicao\ bloco\ MeiPau\ bloco$
$instrucao \longrightarrow Truco\ condicao\ bloco\ Corri\ bloco$
$instrucao \longrightarrow Truco\ condicao\ bloco$
$instrucao \longrightarrow TodaVida\ condicao\ bloco$
$instrucao \longrightarrow Ler\ VAR\ FL$
$instrucao \longrightarrow Mostrar\ expressao\ FL$

$expressao \longrightarrow elemento$
$expressao \longrightarrow expressao\ OpArit\ expressao$
$expressao \longrightarrow AP\ expressao\ OpCrem?\ FP$

$condicao \longrightarrow expressao$
$condicao \longrightarrow condicao\ OpRel\ condicao$
$condicao \longrightarrow condicao\ OpLog\ condicao$
$condicao \longrightarrow AP\ condicao\ FP$

Regras auxiliares:

$tipo \longrightarrow Cado \mid Tiquim \mid Trem \mid Paia$
$bloco \longrightarrow AB\ instrucao+\ FB$
$elemento \longrightarrow VAR \mid NumI \mid NumR \mid BoolValue \mid Str$

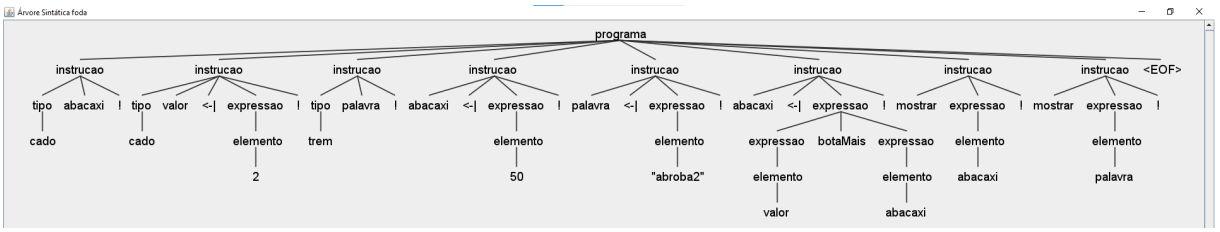
5. Implementação do analisador sintático

O código para a geração do analisador sintático é representado a seguir:

```
grammar UaiScript;

/***** Definição Sintática *****/
programa: instrucao+ EOF;
instrucao: tipo VAR FL   // declaração
| tipo VAR OpAtrib expressao FL   // declaração com atribuição
| VAR OpAtrib expressao FL   // atribuição variável já existente
| VAR OpCrem FL    // instrução de incremento ou decremento
|  Truco  condicao  bloco  MeiPau  bloco    //  if  condição  bloco  else  if
condição bloco
| Truco condicao bloco Corri bloco   // if condição bloco else bloco
| Truco condicao bloco   // if condição bloco
| TodaVida condicao bloco   // while condição bloco
| Ler VAR FL   // ler entrada, atribui uma string a VAR
| Mostrar expressao FL   // imprimir
;

expressao: elemento   // algum elemento
| expressao OpArit expressao   // operações aritméticas
| AP expressao OpCrem? FP   // operações considerando precedência
;

condicao: expressao   // alguma expressao
| condicao OpRel condicao   // operações relacionais
| condicao OpLog condicao   // operações lógicas
| AP condicao FP   // operações considerando precedência
;

tipo: Cado | Tiquim | Trem | Paia;
bloco: AB instrucao+ FB;
elemento: VAR | NumI | NumR | BoolValue | Str;
```

Após o comando *antlr UaiScript.g4* geram-se os arquivos:

Foi criado um novo arquivo java, chamado CompiladorUaiScript.java, o qual tem a função de gerar a árvore sintática de algum código qualquer escrito na linguagem UaiScript. Abaixo, está representado o resultado para o caso teste exemploSimples.txt:



6. Definição semântica da linguagem

A linguagem UaiScript incorpora um conjunto abrangente de verificações semânticas realizadas por meio do framework ANTLR, utilizando a base da linguagem Java. Este processo desafiador culmina na implementação de diversas checagens vitais para a integridade do código:

Checagem de Tipo: Avaliamos a compatibilidade dos tipos durante operações, destacando erros de incompatibilidade. Por exemplo, se tentarmos somar um valor do tipo "cado" (inteiro) com um do tipo "trem" (string), o programa sinalizará um erro de incompatibilidade de tipo.

Checagem de Variáveis não Declaradas: Nesta etapa, verificamos se uma variável não declarada está sendo utilizada, emitindo um erro de "variável não declarada". Por exemplo, tentar usar a variável "numeroQualquer32" sem tê-la declarado anteriormente resultaria nesse tipo de erro. Dado que nossa linguagem é fortemente tipada, essa verificação é crucial para manter essa característica.

Checagem de Declarações Duplicadas de Variáveis: Aqui, asseguramos que não existam duas variáveis declaradas com o mesmo nome. Utilizando uma estrutura de dados, neste caso, o HashMap, que funciona como uma estrutura chave-valor, onde as chaves são únicas, garantimos que, ao tentar inserir uma variável já declarada, o HashMap impeça a duplicação. Isso identifica eficazmente a existência prévia da variável, impedindo sua declaração repetida. Um exemplo claro seria:

Exemplo do erro:

```
cado numeroQualquer!
tiquim numeroQualquer!
```

Nesse caso, a segunda tentativa de declarar "numeroQualquer" seria identificada como uma declaração duplicada, evitando ambiguidades no código. Essas checagens contribuem significativamente para a robustez e confiabilidade da linguagem UaiScript durante a compilação e execução do código-fonte.

7. Implementação do analisador semântico

O código para a geração do analisador semântico é representado a seguir:

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

import org.antlr.v4.runtime.Token;

public class MyListener extends UaiScriptBaseListener {

   /* A lógica para a checagem de erros é feita da seguinte forma:
    * 1. Checagem variaveis duplicadas
    * 2. Checagem variaveis não declaradas
    * 3. Checagem tipos incompativeis
     * Ou seja, por exemplo, mesmo se uma instrução com VAR tenha os 3
erros, somente aparecerá o 1° erro
   */

     private Map<String, String> tabelaSimbolos = new HashMap<String,
String>();
   private String tipoElemento = null;

   @Override
   public void exitInstrucao(UaiScriptParser.InstrucaoContext ctx) {

       // tipo VAR OpAtrib expressao FL (declaração)
         if(ctx.tipo() != null && ctx.VAR() != null && ctx.OpAtrib() !=
null && ctx.expressao() != null && ctx.FL() != null) {
            String id = ctx.VAR().getText();
            String tipo = ctx.tipo().getText().toString();
            if(!verificarVarDuplicacao(id, ctx.start)) {
                        if(!verificarTipoIncompativelExpressao(id, tipo,
ctx.start)) {
                    // declaração sucedida
                    tabelaSimbolos.put(id, tipo);
                }
            }
       }

       // tipo VAR FL (declaração)
         else if(ctx.tipo() != null && ctx.VAR() != null && ctx.FL() !=
null) {
            String id = ctx.VAR().getText();
```

```
            if(!verificarVarDuplicacao(id, ctx.start)) {
                // declaração sucedida
                String tipo = ctx.tipo().getText().toString();
                tabelaSimbolos.put(id, tipo);
            }
        }

        // VAR OpAtrib expressao FL
              else if(ctx.VAR() != null && ctx.OpAtrib() != null &&
ctx.expressao() != null && ctx.FL() != null) {
            String id = ctx.VAR().getText();
            if(!verificarVarNaoDeclarada(id, ctx.start)) {
                verificarTipoIncompativelExpressao(id, null, ctx.start);
            }
        }

        // VAR OpCrem FL
         else if(ctx.VAR() != null && ctx.OpCrem() != null && ctx.FL() !=
null) {
            String id = ctx.VAR().toString();
            if(!verificarVarNaoDeclarada(id, ctx.start)) {
                verificarTipoIncompativelOpCrem(id, ctx.start);
            }
        }

        // Ler VAR FL
          else if(ctx.Ler() != null && ctx.VAR() != null && ctx.FL() !=
null) {
            String id = ctx.VAR().toString();
            if(!verificarVarNaoDeclarada(id, ctx.start)) {
                verificarTipoIncompativelTexto(id, ctx.start);
            }
        }

        tipoElemento = null;
    }

    @Override
    public void enterExpressao(UaiScriptParser.ExpressaoContext ctx) {

        // AP expressao OpCrem? FP (considerando o OpCrem)
        if(ctx.OpCrem() != null) {
```

```
                        UaiScriptParser.ElementoContext elemento =
ctx.expressao(0).elemento();
            String tipo = verificarTipoElemento(elemento);
            if(!tipo.equals("cado") && !tipo.equals("tiquim")) {
                imprimirTipo(elemento.getText().toString(), tipo);
                System.out.println("\tTipo esperado: cado ou tiquim");
                imprimirErro(ctx.expressao(0).start);
            }
            if(tipoElemento != null && !tipo.equals(tipoElemento)) {
                tipoElemento = "indefinido";
            }
        }

        // elemento
        if(ctx.elemento() != null) {
            if(tipoElemento == null) {
                tipoElemento = verificarTipoElemento(ctx.elemento());
            }
            else if(!tipoElemento.equals("indefinido")) {
                tipoElemento = verificarTipoElemento(ctx.elemento());
            }
        }

        // expressao OpArit expressao
        else if(ctx.OpArit() != null) {

                        UaiScriptParser.ElementoContext elemento1 =
ctx.expressao(0).elemento();
            String tipo1 = "indefinido";
            if(elemento1 != null) {
                tipo1 = verificarTipoElemento(elemento1);
            }

                        UaiScriptParser.ElementoContext elemento2 =
ctx.expressao(1).elemento();
            String tipo2 = "indefinido";
            if(elemento2 != null) {
                tipo2 = verificarTipoElemento(elemento2);
            }

                                    if(!tipo1.equals("indefinido") &&
tipo2.equals("indefinido")) {
                tipoElemento = tipo1;
```

```
            }
                                else   if(tipo1.equals("indefinido")   &&
!tipo2.equals("indefinido")) {
                tipoElemento = tipo2;
            }
            else if(!tipo1.equals(tipo2)) {
                tipoElemento = "indefinido";
            }
        }
    }

    private String verificarTipoElemento(UaiScriptParser.ElementoContext
ctx) {
        if(ctx.BoolValue() != null) {
            return ctx.BoolValue().getText().toString();
        }
        else if(ctx.NumI() != null) {
            return "cado";
        }
        else if(ctx.NumR() != null) {
            return "tiquim";
        }
        else if(ctx.Str() != null) {
            return "trem";
        }
        else if(ctx.VAR() != null) {
            String id = ctx.VAR().toString();

            if(verificarVarNaoDeclarada(id, ctx.start)) {
                return "indefinido";
            }
            else {
                return tabelaSimbolos.get(id);
            }
        }
        return "indefinido";
    }

    // Checagem variável duplicada
    private boolean verificarVarDuplicacao(String id, Token start) {
        if(tabelaSimbolos.containsKey(id)) {
             System.out.println("Declaração duplicada! Variável " + id +
" já foi declarada!");
```

```
            imprimirErro(start);
            return true;
        }
        else {
            return false;
        }
    }

    // Checagem variável não declarada
    private boolean verificarVarNaoDeclarada(String id, Token start) {
        if(!tabelaSimbolos.containsKey(id)) {
                    System.out.println("Variável " + id + " não foi
declarada!");
            imprimirErro(start);
            return true;
        }
        else {
            return false;
        }
    }

    // Checagem tipos incompatíveis para expressao
    private boolean verificarTipoIncompativelExpressao(String id, String
tipo, Token start) {
        // se a variável foi declarada
        if(tipo == null) {
            tipo = tabelaSimbolos.get(id);
        }

        if(!tipo.equals(tipoElemento)) {
            imprimirTipo(id, tipo);
            System.out.println("\tTipo da expressao: " + tipoElemento);
            imprimirErro(start);
            return true;
        }
        else {
            return false;
        }
    }

    // Checagem tipos incompatíveis para OpCrem
    private void verificarTipoIncompativelOpCrem(String id, Token start)
{
```

```
        String tipo = tabelaSimbolos.get(id);

        if(!tipo.equals("cado") && !tipo.equals("tiquim")) {
            imprimirTipo(id, tipo);
            System.out.println("\tTipo esperado: cado ou tiquim");
            imprimirErro(start);
        }
    }

    // Checagem tipos incompatíveis para texto
    private void verificarTipoIncompativelTexto(String id, Token start)
{
        String tipo = tabelaSimbolos.get(id);

        if(!tipo.equals("trem")) {
            imprimirTipo(id, tipo);
            System.out.println("\tTipo esperado: trem");
            imprimirErro(start);
        }
    }

      // Imprime no terminal o tipo de id, função auxiliar para
verificarTipoIncompativel*
    private void imprimirTipo(String id, String tipo) {
         System.out.println("Tipo incompatível com a variável/valor: " +
id);
        System.out.println("\tTipo de " + id + ": " + tipo);
    }

    // Imprime no terminal a linha e posições referentes a certo erro
    private void imprimirErro(Token start) {
        System.out.println("\tLinha: " + start.getLine());
        System.out.println("\tPos Inicial: " + start.getStartIndex());
        System.out.println("\tPos Final: " + start.getStopIndex());
    }

    public Map<String, String> getTabelaSimbolos() {
        return tabelaSimbolos;
    }
}
```

Foi criado um novo arquivo java, chamado AnalisadorSemantico.java, o qual tem a função de gerar a árvore semântica de algum código qualquer escrito na linguagem UaiScript que será percorrido pelo outro programa, chamado MyListener.java, o qual estende a classe UaiScriptBaseListener.java, a fim de escrever as sobrescritas dos métodos de entrada e saída das regras da gramática. Abaixo, está representado o resultado para o caso teste exemploSimples.txt:

{palavra=trem, abacaxi=cado, valor=cado}

8. Detalhes da implementação do programa gerado

Nome do arquivo: fibonacci.txt

Análise Léxica:

Token: <Mostrar>
Linha: 1
Pos Inicial: 0
Pos Final: 6
Token: <Str, "Sequencia de Fibonacci">
Linha: 1
Pos Inicial: 8
Pos Final: 31
Token: <FL>
Linha: 1
Pos Inicial: 32
Pos Final: 32
Token: <Cado>
Linha: 3
Pos Inicial: 37
Pos Final: 40
Token: <VAR, n>
Linha: 3
Pos Inicial: 42
Pos Final: 42
Token: <OpAtrib>
Linha: 3
Pos Inicial: 44
Pos Final: 46
Token: <NumI, 10>
Linha: 3
Pos Inicial: 48
Pos Final: 49
Token: <FL>

Linha: 3
Pos Inicial: 50
Pos Final: 50
Token: <Cado>
Linha: 5
Pos Inicial: 55
Pos Final: 58
Token: <VAR, fib0>
Linha: 5
Pos Inicial: 60
Pos Final: 63
Token: <OpAtrib>
Linha: 5
Pos Inicial: 65
Pos Final: 67
Token: <NumI, 0>
Linha: 5
Pos Inicial: 69
Pos Final: 69
Token: <FL>
Linha: 5
Pos Inicial: 70
Pos Final: 70
Token: <Cado>
Linha: 6
Pos Inicial: 73
Pos Final: 76
Token: <VAR, fib1>
Linha: 6
Pos Inicial: 78
Pos Final: 81
Token: <FL>
Linha: 6
Pos Inicial: 82

Pos Final: 82
Token: <VAR, fib1>
Linha: 7
Pos Inicial: 85
Pos Final: 88
Token: <OpAtrib>
Linha: 7
Pos Inicial: 90
Pos Final: 92
Token: <NumI, 1>
Linha: 7
Pos Inicial: 94
Pos Final: 94
Token: <FL>
Linha: 7
Pos Inicial: 95
Pos Final: 95
Token: <Cado>
Linha: 9
Pos Inicial: 100
Pos Final: 103
Token: <VAR, cont>
Linha: 9
Pos Inicial: 105
Pos Final: 108
Token: <OpAtrib>
Linha: 9
Pos Inicial: 110
Pos Final: 112
Token: <NumI, 0>
Linha: 9
Pos Inicial: 114
Pos Final: 114
Token: <FL>

Linha: 9
Pos Inicial: 115
Pos Final: 115
Token: <TodaVida>
Linha: 10
Pos Inicial: 118
Pos Final: 125
Token: <AP>
Linha: 10
Pos Inicial: 127
Pos Final: 127
Token: <VAR, cont>
Linha: 10
Pos Inicial: 128
Pos Final: 131
Token: <OpRel, menoh>
Linha: 10
Pos Inicial: 133
Pos Final: 137
Token: <AP>
Linha: 10
Pos Inicial: 139
Pos Final: 139
Token: <VAR, n>
Linha: 10
Pos Inicial: 140
Pos Final: 140
Token: <OpArit, botaMais>
Linha: 10
Pos Inicial: 142
Pos Final: 149
Token: <NumI, 1>
Linha: 10
Pos Inicial: 151

Pos Final: 151
Token: <FP>
Linha: 10
Pos Inicial: 152
Pos Final: 152
Token: <FP>
Linha: 10
Pos Inicial: 153
Pos Final: 153
Token: <AB>
Linha: 10
Pos Inicial: 155
Pos Final: 155
Token: <Mostrar>
Linha: 11
Pos Inicial: 162
Pos Final: 168
Token: <VAR, fib0>
Linha: 11
Pos Inicial: 170
Pos Final: 173
Token: <FL>
Linha: 11
Pos Inicial: 174
Pos Final: 174
Token: <VAR, fib1>
Linha: 12
Pos Inicial: 181
Pos Final: 184
Token: <OpAtrib>
Linha: 12
Pos Inicial: 186
Pos Final: 188
Token: <VAR, fib0>

Linha: 12
Pos Inicial: 190
Pos Final: 193
Token: <OpArit, botaMais>
Linha: 12
Pos Inicial: 195
Pos Final: 202
Token: <VAR, fib1>
Linha: 12
Pos Inicial: 204
Pos Final: 207
Token: <FL>
Linha: 12
Pos Inicial: 208
Pos Final: 208
Token: <VAR, fib0>
Linha: 13
Pos Inicial: 215
Pos Final: 218
Token: <OpAtrib>
Linha: 13
Pos Inicial: 220
Pos Final: 222
Token: <VAR, fib1>
Linha: 13
Pos Inicial: 224
Pos Final: 227
Token: <OpArit, tira>
Linha: 13
Pos Inicial: 229
Pos Final: 232
Token: <VAR, fib0>
Linha: 13
Pos Inicial: 234

Pos Final: 237

Token: <FL>

Linha: 13

Pos Inicial: 238

Pos Final: 238

Token: <VAR, cont>

Linha: 15

Pos Inicial: 247

Pos Final: 250

Token: <OpCrem, maisUm>

Linha: 15

Pos Inicial: 252

Pos Final: 257

Token: <FL>

Linha: 15

Pos Inicial: 258

Pos Final: 258

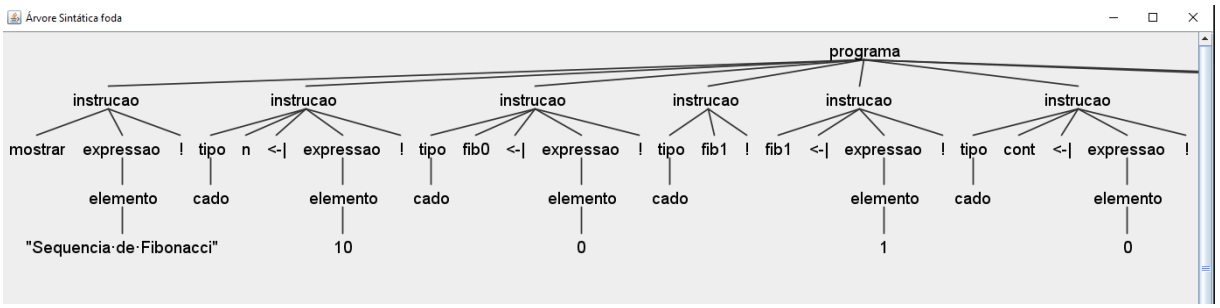
Token: <FB>

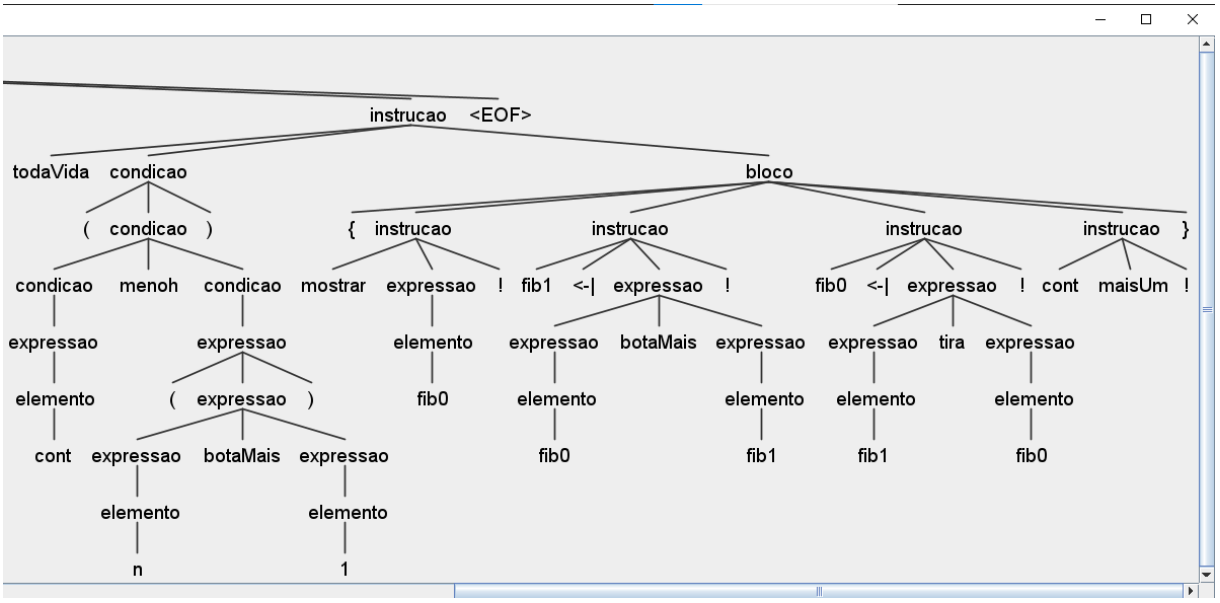
Linha: 16

Pos Inicial: 261

Pos Final: 261

Análise Sintática:





Análise Semântica: {cont=cado, fib1=cado, n=cado, fib0=cado}

9. Casos de teste

Além do caso teste “exemploSimples.txt”, também criamos um simples HelloWorld, soma de dois números, fatorial e um código da sequência de fibonacci (mostrado no tópico 8), além de códigos para demonstrar os erros léxicos, sintáticos e semânticos. Os casos de teste estão disponíveis na pasta casosTeste.

9.1. Hello World

Nome do arquivo: helloWorld.txt

Análise Léxica:

Token: <Mostrar>

Linha: 1

Pos Inicial: 0

Pos Final: 6

Token: <Str, " Ola Mundo ">

Linha: 1

Pos Inicial: 8

Pos Final: 20

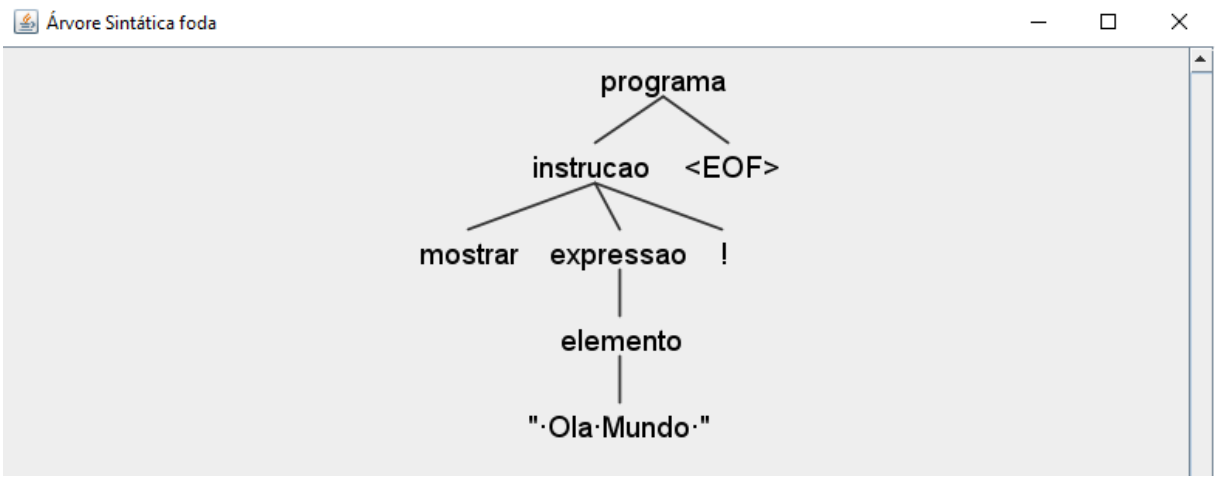
Token: <FL>

Linha: 1

Pos Inicial: 22

Pos Final: 22

Análise Sintática:



Análise Semântica: {}

9.2. soma2numeros

Nome do arquivo: soma2numeros.txt

Análise Léxica:

Token: <Tiquim>
Linha: 1
Pos Inicial: 0
Pos Final: 5
Token: <VAR, num1>
Linha: 1
Pos Inicial: 7
Pos Final: 10
Token: <OpAtrib>
Linha: 1
Pos Inicial: 12
Pos Final: 14
Token: <NumR, 25.3>
Linha: 1
Pos Inicial: 16
Pos Final: 19
Token: <FL>
Linha: 1
Pos Inicial: 20
Pos Final: 20
Token: <Tiquim>
Linha: 3
Pos Inicial: 25
Pos Final: 30
Token: <VAR, num2>
Linha: 3

Pos Inicial: 32
Pos Final: 35
Token: <OpAtrib>
Linha: 3
Pos Inicial: 37
Pos Final: 39
Token: <NumR, 53.1>
Linha: 3
Pos Inicial: 41
Pos Final: 44
Token: <FL>
Linha: 3
Pos Inicial: 45
Pos Final: 45
Token: <Tiquim>
Linha: 5
Pos Inicial: 50
Pos Final: 55
Token: <VAR, soma>
Linha: 5
Pos Inicial: 57
Pos Final: 60
Token: <OpAtrib>
Linha: 5
Pos Inicial: 62
Pos Final: 64
Token: <AP>
Linha: 5
Pos Inicial: 66
Pos Final: 66
Token: <VAR, num1>
Linha: 5
Pos Inicial: 67
Pos Final: 70

Token: <OpArit, botaMais>

Linha: 5

Pos Inicial: 72

Pos Final: 79

Token: <VAR, num2>

Linha: 5

Pos Inicial: 81

Pos Final: 84

Token: <FP>

Linha: 5

Pos Inicial: 85

Pos Final: 85

Token: <FL>

Linha: 5

Pos Inicial: 86

Pos Final: 86

Token: <Mostrar>

Linha: 7

Pos Inicial: 91

Pos Final: 97

Token: <VAR, soma>

Linha: 7

Pos Inicial: 99

Pos Final: 102

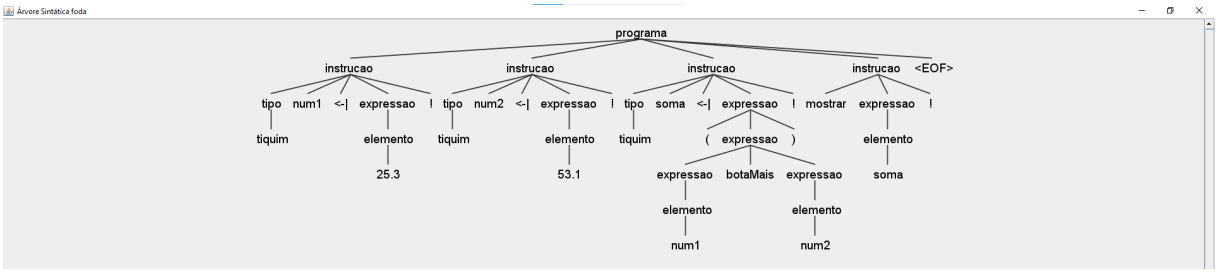
Token: <FL>

Linha: 7

Pos Inicial: 103

Pos Final: 103

Análise Sintática:



Análise Semântica: {soma=tiquim, num1=tiquim, num2=tiquim}

9.3. fatorial

Nome do arquivo: fatorial.txt

Análise Léxica:

Token: <Cado>

Linha: 1

Pos Inicial: 0

Pos Final: 3

Token: <VAR, numero>

Linha: 1

Pos Inicial: 5

Pos Final: 10

Token: <OpAtrib>

Linha: 1

Pos Inicial: 12

Pos Final: 14

Token: <NumI, 7>

Linha: 1

Pos Inicial: 16

Pos Final: 16

Token: <FL>

Linha: 1

Pos Inicial: 17

Pos Final: 17

Token: <Cado>

Linha: 2
Pos Inicial: 20
Pos Final: 23
Token: <VAR, resultado>
Linha: 2
Pos Inicial: 25
Pos Final: 33
Token: <OpAtrib>
Linha: 2
Pos Inicial: 35
Pos Final: 37
Token: <NumI, 0>
Linha: 2
Pos Inicial: 39
Pos Final: 39
Token: <FL>
Linha: 2
Pos Inicial: 40
Pos Final: 40
Token: <TodaVida>
Linha: 4
Pos Inicial: 45
Pos Final: 52
Token: <AP>
Linha: 4
Pos Inicial: 54
Pos Final: 54
Token: <VAR, numero>
Linha: 4
Pos Inicial: 55
Pos Final: 60
Token: <OpRel, maioh>
Linha: 4
Pos Inicial: 62

Pos Final: 66
Token: <NumI, 0>
Linha: 4
Pos Inicial: 68
Pos Final: 68
Token: <FP>
Linha: 4
Pos Inicial: 69
Pos Final: 69
Token: <AB>
Linha: 4
Pos Inicial: 71
Pos Final: 71
Token: <VAR, resultado>
Linha: 5
Pos Inicial: 76
Pos Final: 84
Token: <OpAtrib>
Linha: 5
Pos Inicial: 86
Pos Final: 88
Token: <VAR, resultado>
Linha: 5
Pos Inicial: 90
Pos Final: 98
Token: <OpArit, botaMais>
Linha: 5
Pos Inicial: 100
Pos Final: 107
Token: <VAR, numero>
Linha: 5
Pos Inicial: 109
Pos Final: 114
Token: <FL>

Linha: 5

Pos Inicial: 115

Pos Final: 115

Token: <VAR, numero>

Linha: 6

Pos Inicial: 120

Pos Final: 125

Token: <OpCrem, tiraUm>

Linha: 6

Pos Inicial: 127

Pos Final: 132

Token: <FL>

Linha: 6

Pos Inicial: 133

Pos Final: 133

Token: <FB>

Linha: 7

Pos Inicial: 136

Pos Final: 136

Token: <Mostrar>

Linha: 9

Pos Inicial: 141

Pos Final: 147

Token: <VAR, resultado>

Linha: 9

Pos Inicial: 149

Pos Final: 157

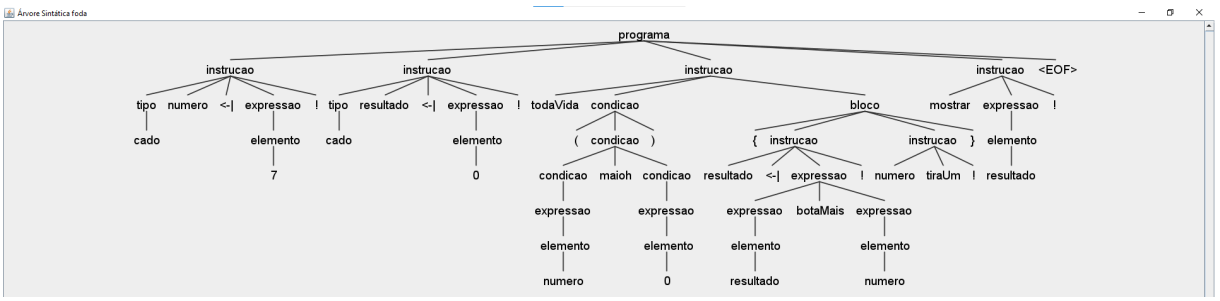
Token: <FL>

Linha: 9

Pos Inicial: 158

Pos Final: 158

Análise Sintática:



Análise Semântica: {numero=cado, resultado=cado}

9.4. variavelDuplicada

Nome do arquivo: variavelDuplicada.txt

Análise Léxica:

Token: <Tiquim>

Linha: 1

Pos Inicial: 0

Pos Final: 5

Token: <VAR, num1>

Linha: 1

Pos Inicial: 7

Pos Final: 10

Token: <FL>

Linha: 1

Pos Inicial: 11

Pos Final: 11

Token: <Cado>

Linha: 2

Pos Inicial: 14

Pos Final: 17

Token: <VAR, num2>

Linha: 2

Pos Inicial: 19

Pos Final: 22

Token: <OpAtrib>
    Linha: 2
    Pos Inicial: 24
    Pos Final: 26
Token: <NumI, 32>
    Linha: 2
    Pos Inicial: 28
    Pos Final: 29
Token: <FL>
    Linha: 2
    Pos Inicial: 30
    Pos Final: 30
Token: <Cado>
    Linha: 3
    Pos Inicial: 33
    Pos Final: 36
Token: <VAR, num2>
    Linha: 3
    Pos Inicial: 38
    Pos Final: 41
Token: <FL>
    Linha: 3
    Pos Inicial: 42
    Pos Final: 42
Token: <VAR, num1>
    Linha: 5
    Pos Inicial: 47
    Pos Final: 50
Token: <OpAtrib>
    Linha: 5
    Pos Inicial: 52
    Pos Final: 54
Token: <NumR, 12.42>
    Linha: 5

Pos Inicial: 56

Pos Final: 60

Token: <FL>

Linha: 5

Pos Inicial: 61

Pos Final: 61

Token: <Mostrar>

Linha: 7

Pos Inicial: 66

Pos Final: 72

Token: <VAR, num1>

Linha: 7

Pos Inicial: 74

Pos Final: 77

Token: <FL>

Linha: 7

Pos Inicial: 78

Pos Final: 78

Token: <Mostrar>

Linha: 8

Pos Inicial: 81

Pos Final: 87

Token: <VAR, num2>

Linha: 8

Pos Inicial: 89

Pos Final: 92

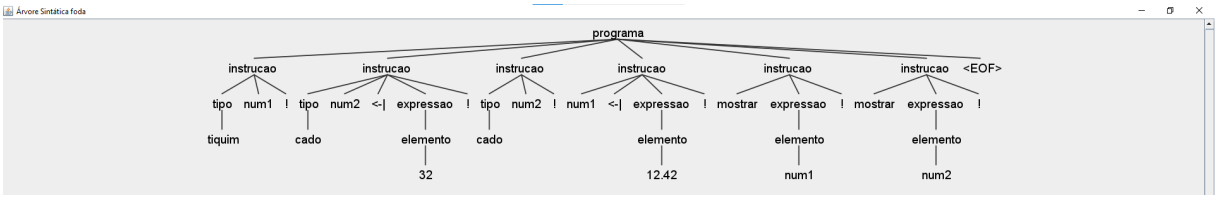
Token: <FL>

Linha: 8

Pos Inicial: 93

Pos Final: 93

Análise Sintática:



Análise Semântica:

Declaração duplicada! Variável num2 já foi declarada!

Linha: 3

Pos Inicial: 33

Pos Final: 36

{num1=tiquim, num2=cado}

9.5. variavelNaoDeclarada

Nome do arquivo: variavelNaoDeclarada.txt

Análise Léxica:

Token: <Mostrar>

Linha: 1

Pos Inicial: 0

Pos Final: 6

Token: <Str, "teste para erro variavel nao declarada!">

Linha: 1

Pos Inicial: 8

Pos Final: 48

Token: <FL>

Linha: 1

Pos Inicial: 49

Pos Final: 49

Token: <Trem>

Linha: 3

Pos Inicial: 54

Pos Final: 57

Token: <VAR, entrada1>

Linha: 3
Pos Inicial: 59
Pos Final: 66
Token: <FL>
Linha: 3
Pos Inicial: 67
Pos Final: 67
Token: <Ler>
Linha: 4
Pos Inicial: 70
Pos Final: 72
Token: <VAR, entrada1>
Linha: 4
Pos Inicial: 74
Pos Final: 81
Token: <FL>
Linha: 4
Pos Inicial: 82
Pos Final: 82
Token: <Ler>
Linha: 6
Pos Inicial: 87
Pos Final: 89
Token: <VAR, entrada2>
Linha: 6
Pos Inicial: 91
Pos Final: 98
Token: <FL>
Linha: 6
Pos Inicial: 99
Pos Final: 99
Token: <Mostrar>
Linha: 8
Pos Inicial: 104

Pos Final: 110
Token: <AP>
Linha: 8
Pos Inicial: 112
Pos Final: 112
Token: <VAR, entrada1>
Linha: 8
Pos Inicial: 113
Pos Final: 120
Token: <OpArit, botaMais>
Linha: 8
Pos Inicial: 122
Pos Final: 129
Token: <VAR, entrada2>
Linha: 8
Pos Inicial: 131
Pos Final: 138
Token: <FP>
Linha: 8
Pos Inicial: 139
Pos Final: 139
Token: <FL>
Linha: 8
Pos Inicial: 140
Pos Final: 140
Token: <VAR, entrada2>
Linha: 10
Pos Inicial: 145
Pos Final: 152
Token: <OpCrem, tiraUm>
Linha: 10
Pos Inicial: 154
Pos Final: 159
Token: <FL>

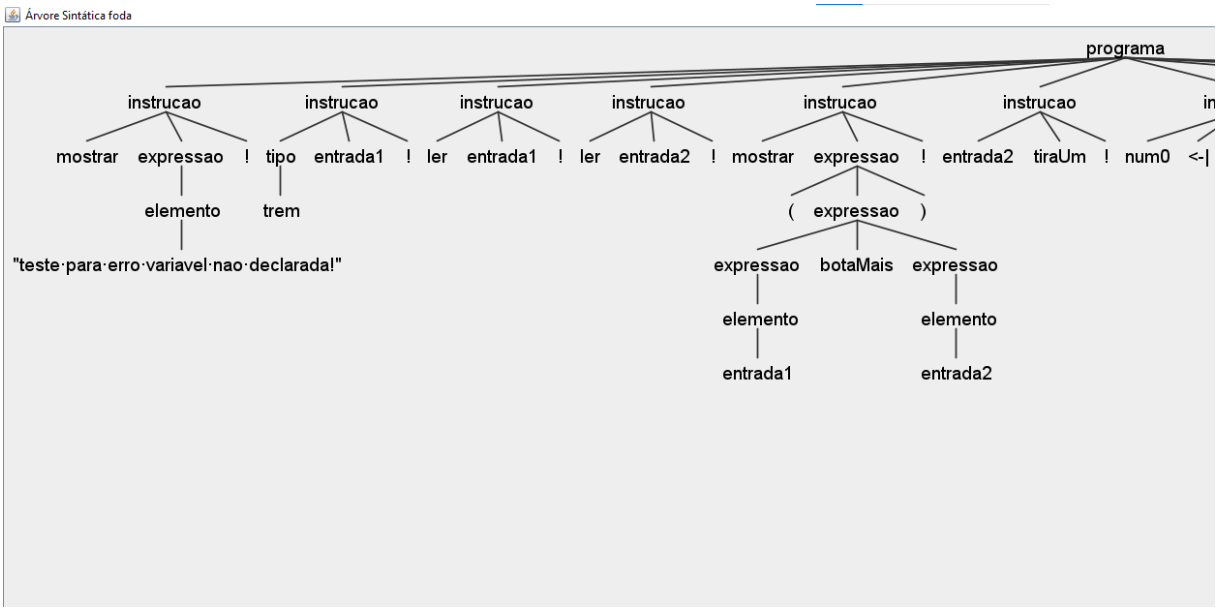
Linha: 10
Pos Inicial: 160
Pos Final: 160
Token: <VAR, num0>
Linha: 12
Pos Inicial: 165
Pos Final: 168
Token: <OpAtrib>
Linha: 12
Pos Inicial: 170
Pos Final: 172
Token: <NumR, 12.42>
Linha: 12
Pos Inicial: 174
Pos Final: 178
Token: <FL>
Linha: 12
Pos Inicial: 179
Pos Final: 179
Token: <Cado>
Linha: 14
Pos Inicial: 184
Pos Final: 187
Token: <VAR, num1>
Linha: 14
Pos Inicial: 189
Pos Final: 192
Token: <OpAtrib>
Linha: 14
Pos Inicial: 194
Pos Final: 196
Token: <NumI, 20>
Linha: 14
Pos Inicial: 198

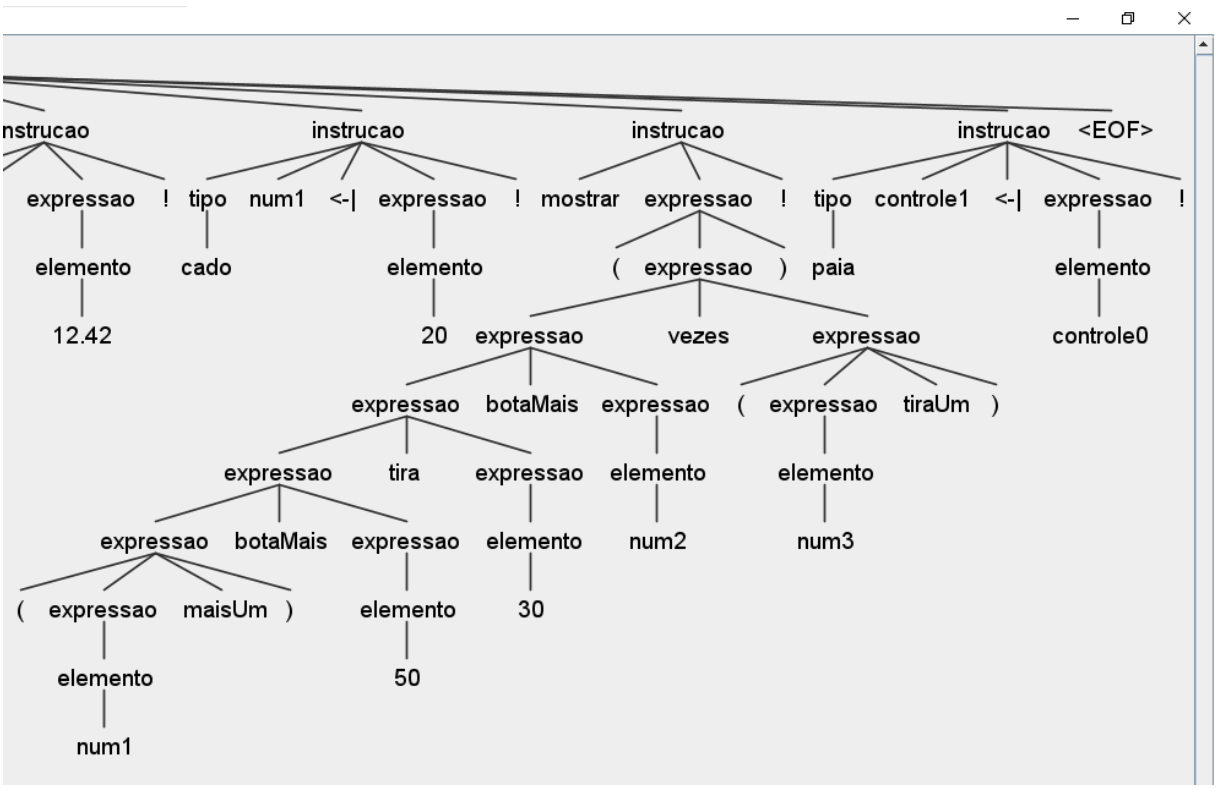
Pos Final: 199
Token: <FL>
Linha: 14
Pos Inicial: 200
Pos Final: 200
Token: <Mostrar>
Linha: 16
Pos Inicial: 205
Pos Final: 211
Token: <AP>
Linha: 16
Pos Inicial: 213
Pos Final: 213
Token: <AP>
Linha: 16
Pos Inicial: 214
Pos Final: 214
Token: <VAR, num1>
Linha: 16
Pos Inicial: 215
Pos Final: 218
Token: <OpCrem, maisUm>
Linha: 16
Pos Inicial: 220
Pos Final: 225
Token: <FP>
Linha: 16
Pos Inicial: 226
Pos Final: 226
Token: <OpArit, botaMais>
Linha: 16
Pos Inicial: 228
Pos Final: 235
Token: <NumI, 50>

Linha: 16
Pos Inicial: 237
Pos Final: 238
Token: <OpArit, tira>
Linha: 16
Pos Inicial: 240
Pos Final: 243
Token: <NumI, 30>
Linha: 16
Pos Inicial: 245
Pos Final: 246
Token: <OpArit, botaMais>
Linha: 16
Pos Inicial: 248
Pos Final: 255
Token: <VAR, num2>
Linha: 16
Pos Inicial: 257
Pos Final: 260
Token: <OpArit, vezes>
Linha: 16
Pos Inicial: 262
Pos Final: 266
Token: <AP>
Linha: 16
Pos Inicial: 268
Pos Final: 268
Token: <VAR, num3>
Linha: 16
Pos Inicial: 269
Pos Final: 272
Token: <OpCrem, tiraUm>
Linha: 16
Pos Inicial: 274

Pos Final: 279
Token: <FP>
Linha: 16
Pos Inicial: 280
Pos Final: 280
Token: <FP>
Linha: 16
Pos Inicial: 281
Pos Final: 281
Token: <FL>
Linha: 16
Pos Inicial: 282
Pos Final: 282
Token: <Paia>
Linha: 18
Pos Inicial: 287
Pos Final: 290
Token: <VAR, controle1>
Linha: 18
Pos Inicial: 292
Pos Final: 300
Token: <OpAtrib>
Linha: 18
Pos Inicial: 302
Pos Final: 304
Token: <VAR, controle0>
Linha: 18
Pos Inicial: 306
Pos Final: 314
Token: <FL>
Linha: 18
Pos Inicial: 316
Pos Final: 316

Análise Sintática:





Análise Semântica:

Variável entrada2 não foi declarada!

Linha: 6

Pos Inicial: 87

Pos Final: 89

Variável entrada2 não foi declarada!

Linha: 8

Pos Inicial: 131

Pos Final: 138

Variável entrada2 não foi declarada!

Linha: 8

Pos Inicial: 131

Pos Final: 138

Variável entrada2 não foi declarada!

Linha: 10

Pos Inicial: 145

Pos Final: 152

Variável num0 não foi declarada!

Linha: 12

Pos Inicial: 165

Pos Final: 168

Variável num2 não foi declarada!

Linha: 16

Pos Inicial: 257

Pos Final: 260

Variável num2 não foi declarada!

Linha: 16

Pos Inicial: 257

Pos Final: 260

Variável num3 não foi declarada!

Linha: 16

Pos Inicial: 269

Pos Final: 272

Tipo incompatível com a variável/valor: num3

Tipo de num3: indefinido

Tipo esperado: cado ou tiquim

Linha: 16

Pos Inicial: 269

Pos Final: 272

Variável controle0 não foi declarada!

Linha: 18

Pos Inicial: 306

Pos Final: 314

Tipo incompatível com a variável/valor: controle1

Tipo de controle1: paia

Tipo da expressao: indefinido

Linha: 18

Pos Inicial: 287

Pos Final: 290

{entrada1=trem, num1=cado}

9.6. tiposIncompativeis

Nome do arquivo:tiposIncompativeis.txt

Análise Léxica:

Token: <Cado>

Linha: 1

Pos Inicial: 0

Pos Final: 3

Token: <VAR, num1>

Linha: 1

Pos Inicial: 5

Pos Final: 8

Token: <OpAtrib>

Linha: 1

Pos Inicial: 10

Pos Final: 12

Token: <AP>

Linha: 1

Pos Inicial: 14

Pos Final: 14

Token: <NumR, 2.4>

Linha: 1

Pos Inicial: 15

Pos Final: 17

Token: <OpArit, botaMais>

Linha: 1

Pos Inicial: 19

Pos Final: 26

Token: <NumR, 5.5>

Linha: 1

Pos Inicial: 28

Pos Final: 30

Token: <FP>

Linha: 1

Pos Inicial: 31

Pos Final: 31

Token: <FL>

Linha: 1

Pos Inicial: 32

Pos Final: 32

Token: <Cado>

Linha: 2

Pos Inicial: 35

Pos Final: 38

Token: <VAR, num1>

Linha: 2

Pos Inicial: 40

Pos Final: 43

Token: <OpAtrib>

Linha: 2

Pos Inicial: 45

Pos Final: 47

Token: <NumI, 25>

Linha: 2

Pos Inicial: 49

Pos Final: 50

Token: <FL>
    Linha: 2
    Pos Inicial: 51
    Pos Final: 51
Token: <Paia>
    Linha: 4
    Pos Inicial: 56
    Pos Final: 59
Token: <VAR, num2>
    Linha: 4
    Pos Inicial: 61
    Pos Final: 64
Token: <FL>
    Linha: 4
    Pos Inicial: 65
    Pos Final: 65
Token: <VAR, num2>
    Linha: 5
    Pos Inicial: 68
    Pos Final: 71
Token: <OpCrem, maisUm>
    Linha: 5
    Pos Inicial: 73
    Pos Final: 78
Token: <FL>
    Linha: 5
    Pos Inicial: 79
    Pos Final: 79
Token: <Tiquim>
    Linha: 7
    Pos Inicial: 84
    Pos Final: 89
Token: <VAR, num3>
    Linha: 7

Pos Inicial: 91
Pos Final: 94
Token: <OpAtrib>
Linha: 7
Pos Inicial: 96
Pos Final: 98
Token: <AP>
Linha: 7
Pos Inicial: 100
Pos Final: 100
Token: <AP>
Linha: 7
Pos Inicial: 101
Pos Final: 101
Token: <NumR, 2.4>
Linha: 7
Pos Inicial: 102
Pos Final: 104
Token: <OpCrem, maisUm>
Linha: 7
Pos Inicial: 106
Pos Final: 111
Token: <FP>
Linha: 7
Pos Inicial: 112
Pos Final: 112
Token: <OpArit, tira>
Linha: 7
Pos Inicial: 114
Pos Final: 117
Token: <Str, "oi">
Linha: 7
Pos Inicial: 119
Pos Final: 122

Token: <FP>

Linha: 7

Pos Inicial: 123

Pos Final: 123

Token: <FL>

Linha: 7

Pos Inicial: 124

Pos Final: 124

Token: <Tiquim>

Linha: 9

Pos Inicial: 129

Pos Final: 134

Token: <VAR, palavra>

Linha: 9

Pos Inicial: 136

Pos Final: 142

Token: <FL>

Linha: 9

Pos Inicial: 143

Pos Final: 143

Token: <Ler>

Linha: 10

Pos Inicial: 146

Pos Final: 148

Token: <VAR, palavra>

Linha: 10

Pos Inicial: 150

Pos Final: 156

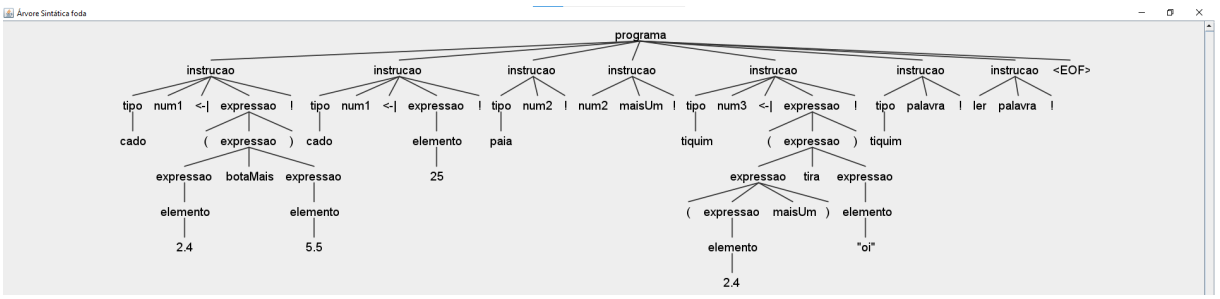
Token: <FL>

Linha: 10

Pos Inicial: 157

Pos Final: 157

Análise Sintática:



Análise Semântica:

Tipo incompatível com a variável/valor: num1

Tipo de num1: cado

Tipo da expressao: tiquim

Linha: 1

Pos Inicial: 0

Pos Final: 3

Tipo incompatível com a variável/valor: num2

Tipo de num2: paia

Tipo esperado: cado ou tiquim

Linha: 5

Pos Inicial: 68

Pos Final: 71

Tipo incompatível com a variável/valor: num3

Tipo de num3: tiquim

Tipo da expressao: indefinido

Linha: 7

Pos Inicial: 84

Pos Final: 89

Tipo incompatível com a variável/valor: palavra

Tipo de palavra: tiquim

Tipo esperado: trem

Linha: 10

Pos Inicial: 146

Pos Final: 148

{palavra=tiquim, num1=cado, num2=paia}

10. Repositório no GitHub

O projeto pode ser acessado pelo link: brunof5/ProjetoFinal-Compiladores