UNIVERSIDADE SÃO JUDAS TADEU

TEORIA DE COMPILAÇÃO E COMPILADORES

Caio Bonato
Gabriel Castro
Laysla Rodrigues
Leonardo Freitas
Lucas Quireza

UNIVERSIDADE SÃO JUDAS TADEU

Compilador de Linguagem

Professor Anacé Nunes da Silva

São Paulo, 17 de Junho de 2024

Sumário

1 Resumo

Este documento explora a implementação de um compilador utilizando a biblioteca PLY (Python Lex-Yacc) em Python. O objetivo é criar um compilador utilizando o analisador léxico, sintático e semântico para uma linguagem específica, utilizando expressões regulares e regras de parsing para processar código-fonte em um formato personalizado.

1.1 Abstract

This document explores the implementation of a compiler using the PLY (Python Lex-Yacc) library in Python. The goal is to create a compiler utilizing lexical, syntactic, and semantic analysis for a specific language. It employs regular expressions and parsing rules to process code in a customized format.

2 Conceito de um compilador

Tradutor de um código escrito em uma linguagem de programação de alto nível para código de máquina específico a uma arquitetura de computador. O compilador realiza várias etapas fundamentais, através da análise léxica, sintática e semântica. E também pode incluir funcionalidades como relatórios de erros, suporte a bibliotecas padrão e integração com ferramentas de desenvolvimento. Em resumo, facilita a transformação de código legível por humanos em instruções que um computador pode executar eficientemente.

3 Biblioteca Ply

```
import ply.lex as lex
import ply.yacc as yacc
```

Foi importada e utilizada a biblioteca PLY, para auxiliar na construção do compilador, por ser bastante intuitiva.

A biblioteca PLY (Python Lex-Yacc) é uma implementação em Python das ferramentas tradicionais Lex e Yacc, usadas para análise léxica e sintática de linguagens. PLY é útil para criar analisadores de linguagem personalizados e é composta por dois módulos principais: lex para a análise léxica e yacc para a análise sintática.

3.1 Como instalar o ply

```
pip install ply
```

Para instalar o ply, é preciso colocar o código acima no prompt de comando (cmd) ou no terminal de um compilador de sua escolha, porém para isso você precisa ter instalado tanto o Python quanto o pip para poder instalar a biblioteca ply.

Se quiser verificar se possui o pip instalado é possível utilizar o código a seguir no prompt de comando para verificar:

```
pip --version
```

Para verificar se o ply também foi instalado, basta utilizar o código e encontra-lo na lista(caso possua mais de uma biblioteca):

```
pip list
```

4 Arquivos Do Código Padrão

Todos os arquivos de código padrão devem ser executados na mesma pasta de um compilador para garantir que não haja nenhum erro na geração de todos os arquivos necessários, que serão citados mais a frente no documento.

4.1 Arquivo compialdorA3V2

Neste arquivo contém todo o codigo do compilador : Analise Lexica, Sintatica, Semantica, todas as definiçoes dos tokens da nossa linguagem e as declarações do que cada uma faz e nas proximas partes estaram explicados o que cada parte.

4.2 Codigo_input.txt

Este compilador utiliza um arquivo chamado codigo_input.txt que contém o código-fonte de exemplo que será utilizado para a compilação da linguagem para Python.

5 Análise Léxica

Na análise léxica serão efetuadas as seguintes tarefas: definição de tokens, expressões regulares para tokens, funções de tratamento para tokens, ignorar espaços e caracteres irrelevantes, tratamento de erros e a inicialização do lexer.

5.1 Definição de Tokens

```
    tokens = [
INPROGRAMA: inprograma
FMPROGRAMA: fmprograma
NI: ni
VARIAVEL: Identificador (nomes de variáveis)
INTEIRO: Números inteiros
FLOAT: Números de ponto flutuante
OP_ATRIB_IGUAL: =
OP_FINAL_LINHA_PONTO_VIRGULA: ;
PARA: para
LEIA: leia
ESCREVA: escreva
ENQUANTO: enquanto
LESS: <
GREATER: >
OP_PAR_ESQUERDO: (
OP_PAR_DIREITO: )
OP_CHAVE_ESQUERDA: {
OP_CHAVE_DIREITA: }
OP_SOMA: +
IN: in
RANGE: range
```

Os tokens em um compilador são elementos léxicos, ou seja, as menores unidades de significado no código-fonte que possuem um significado específico. São gerados durante a fase de análise léxica.

5.2 Expressões regulares para tokens

```
# Regras de expressão regular para tokens
t_INPROGRAMA = r'inprograma'
t_FMPROGRAMA = r'fmprograma'
t_NI = r'ni'
t_OP_ATRIB_IGUAL = r'='
t_OP_FINAL_LINHA_PONTO_VIRGULA = r';'
t_PARA = r'para'
t_LEIA = r'leia'
t_ESCREVA = r'escreva'
t_ENQUANTO = r'enquanto'
t_LESS = r'<'
t_GREATER = r'>'
t_OP_PAR_ESQUERDO = r'\('
t_OP_PAR_DIREITO = r'\)'
t_OP_CHAVE_ESQUERDA = r'\{'
t_OP_CHAVE_DIREITA = r'\}'
t_OP_SOMA = r'\+'
```

```
t_IN = r'in'
t_RANGE = r'range'
```

O token é definido por uma expressão regular (regex), que especifica como os caracteres do código-fonte correspondem a cada um.

5.3 Funções de tratamento para tokens

```
#Converte o valor do token para um número de ponto flutuante.
    def t_FLOAT(t):
    r'\d+\.\d+'
    t.value = float(t.value)
    return t

#Converte o valor do token para um número inteiro
def t_INTEIRO(t): .V
    r'\d+'
    t.value = int(t.value)
    return t

def t_VARIAVEL(t):
#Verifica se o identificador é uma palavra-chave reservada
(como ni, inprograma, etc.) e, se for, ajusta o tipo do token.
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    if t.value in {'ni', 'inprograma', 'fmprograma', 'escreva', 'leia',
    'para',
    'enquanto', 'in', 'range'}:
        t.type = t.value.upper()

        # Transforma em maiúsculas para tratar como palavra-chave
    return t
```

5.4 Ignorar espaços e caracteres irrelevantes

```
    t_ignore = ' \t\n'
```

A variável t ignore é utilizada para ignorar espaços, tabulações e novas linhas durante a tokenização.

5.5 Tratamento de erros

```
    def t_error(t):
    print("Caractere ilegal '%s'" % t.value[0])
    t.lexer.skip(1)
```

A função t error(t) ocorre quando um caractere ilegal é encontrado no código-fonte.

5.6 inicialização do lexer e finalização da análise léxica

```
    lexer = lex.lex()
```

Análisa se todas as partes que foram citadas está coerente.

5.7 Conjunto de Variáveis Declaradas e Lista de Erros

```
    variaveis_declaradas = set()
erros = []
```

Inicializa um conjunto para armazenar as variáveis declaradas e uma lista para registrar os erros encontrados durante a análise léxica; caso haja falhas, será apontado em qual linha o problema se localiza.

6 Início da análise sintética e semântica

6.1 Regras de parsing

Nos tópicos a seguir, há definições sobre as funções que foram utilizadas com a biblioteca PLY para construir a gramática da linguagem, todas são prefixadas por p_

6.2 p_Programa

```
    def p_programa(p):
    '''
    programa : INPROGRAMA declaracoes FMPROGRAMA
    '''
    p[0] = p[2]
```

Esta regra define a estrutura básica de um programa. Deve começar com INPROGRAMA, seguido por uma série de declarações (declaracoes) e terminar com FMPROGRAMA.

p[0] armazena o resultado final das declarações.

6.3 p_Declaracoes_Single e p_Declaracoes_Mult

```
    def p_declaracoes_single(p):
    '''
    declaracoes : declaracao
    '''
    p[0] = p[1]

def p_declaracoes_mult(p):
    '''
    declaracoes : declaracoes declaracao
    '''
    p[0] = p[1] + p[2]
```

p_Declaracoes_Single lida com uma única declaraçãos.
p_Declaracoes_Mult lida com múltiplas declarações, concatenando-as.

6.4 p_Declaracao_Ni_atribuicao

```
def p_declaracao_ni_atribuicao(p):
'''
declaracao : NI VARIAVEL OP_ATRIB_IGUAL expressao

OP_FINAL_LINHA_PONTO_VIRGULA
'''
variavel = p[2]
if variavel in variaveis_declaradas:
    erro = f"Erro: Variável '{variavel}' já foi declarada."
    print(erro)
    erros.append(erro)
    p[0] = ""
else:
    variaveis_declaradas.add(variavel)
    p[0] = f"{variavel} = {p[4]}\n"
```

Define uma declaração de variável com atribuição (usando NI).
Verifica se a variável já foi declarada. Se sim, adiciona um erro.
Caso contrário, adiciona a variável ao conjunto de variáveis declaradas.

6.5 p_Declaracao_Atribuicao

```
def p_declaracao_atribuicao(p):
'''
declaracao : VARIAVEL OP_ATRIB_IGUAL expressao

OP_FINAL_LINHA_PONTO_VIRGULA
'''
variavel = p[1]
if variavel not in variaveis_declaradas:
    erro = f"Erro: Variável '{variavel}' não foi declarada antes de
    usar."
    print(erro)
    erros.append(erro)
    p[0] = ""
else:
    p[0] = f"{variavel} = {p[3]}\n"
```

Define uma atribuição de valor a uma variável.
Verifica se a variável foi declarada. Se não, adiciona um erro.

6.6 p_Declaracao_Escreva

```
def p_declaracao_escreva(p):
'''
declaracao : ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA
'''
p[0] = f"print({p[2]})\n"
```

Define uma declaração ESCREVA que imprime o valor de uma expressão.

6.7 p_Declaracao_Leia

```
def p_declaracao_leia(p):
'''
declaracao : LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA
'''
variavel = p[2]
if variavel not in variaveis_declaradas:
    variaveis_declaradas.add(variavel)
p[0] = f"{variavel} = float(input('Digite um numero : '))\n"
```

Define uma declaração LEIA, que consulta um valor do usuário e o armazena em uma variável.
Se a variável ainda não foi declarada, é adicionada ao conjunto de variáveis declaradas.

6.8 p_Declaracao_Para

```
def p_declaracao_para(p):
'''
declaracao : PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO

expressao OP_PAR_DIREITO OP_PAR_DIREITO
OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
'''
variavel = p[3]
variaveis_declaradas.add(variavel)
# Implicitamente declarando a variável do loop
p[0] = f"for {variavel} in range({p[7]}):\n{indent(p[11], 4)}\n"
```

Define um loop PARA (equivalente a um loop for).
Adiciona a variável do loop ao conjunto de variáveis declaradas implicitamente.

6.9 p_Declaracao_Enquanto

```
def p_declaracao_enquanto(p):
'''
declaracao :
ENQUANTO OP_PAR_ESQUERDO condicao
OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
'''
p[0] = f"while {p[3]}:\n{indent(p[6], 4)}\n"
```

Define um loop ENQUANTO (equivalente a um loop while).

6.10 p_Expressao

```
def p_expressao(p):
'''
expressao : INTEIRO
          | FLOAT
          | VARIAVEL
          | VARIAVEL OP_SOMA expressao
'''
if len(p) == 2:
    p[0] = f"{p[1]}"
else:
    p[0] = f"{p[1]} + {p[3]}"
```

Define como uma expressão podendo ser um inteiro, um float, uma variável ou uma soma de variável com outra expressão.

6.11 p_Condicao

```
def p_condicao(p):
'''
condicao : expressao LESS expressao
         | expressao GREATER expressao
'''
p[0] = f"{p[1]} {p[2]} {p[3]}"
```

Define uma condição como uma comparação entre duas expressões usando operadores de comparação (¡ ou ¿).

6.12 p_Error

```
def p_error(p):
if p:
    erro = f"Erro de sintaxe em '{p.value}'"
    print(erro)
    erros.append(erro)
else:
    erro = "Erro de sintaxe no final do arquivo"
    print(erro)
    erros.append(erro)
```

Define o tratamento de erros sintáticos, imprimindo uma mensagem de erro e adicionando-a à lista de erros.

7 Fim da análise sintática e semantica

Finalizada a definição de regras de gramatica usando as funcoes que comecam com p_ para definir a estrutura do codigo.

8 Processamento e Análise de Código Python

8.1 Função Auxiliar para Indentação

```
def indent(text, spaces):
return '\n'.join(' ' * spaces + line if line else ''
for line in text.splitlines())
```

O intuito desta função é adicionar indentação ao código gerado.

8.2 Codigo_Fonte

```
    with open('codigo_input.txt', 'r') as file:
linhas = file.readlines()

codigoFonte = ""
```

O arquivo codigo_input.txt é aberto e incluído numa string vazia chamada codigo_fonte. Esta string será usada para armazenar todo o conteúdo do arquivo de entrada em uma única string, removendo caracteres de nova linha.

8.3 Remoção de espaços

```
for linha in linhas:
codigo_fonte += linha.strip() + "\n"
```

A função Strip foi utilizada para remover os espaços em brancos e caracteres de nova linha de cada linha e em seguida é concatenado com a string codigoFonte acrescentando sempre um caracter `\n` ao final de cada linha para manter a formatação.

8.4 Processando o codigo_Fonte com o analisador léxico

```
lexer.input(codigo_fonte)
```

Configurado o analisador léxico (lexer), que processa a string codigo_Fonte. Sendo responsável por dividir a entrada em "tokens" ou unidades léxicas, como palavras-chave, identificadores, operadores, etc.

9 codigo_input.txt

O input, ou entrada, em um contexto de análise léxica e sintática, refere-se ao código-fonte ou texto que está sendo processado pelo compilador ou interpretador. É o conjunto de caracteres ou símbolos que compõem o programa de computador que será traduzido ou interpretado.

```
inprograma
    ni value = 10;
    leia x;
    leia y;
    para (i in range(value)) {
        escreva i;
    }
    enquanto (x < 20) {
        x = x + 1.5;
        escreva x;
    }
    escreva x + y;
fmprograma
```

9.1 Início do Programa:

inprograma
- Indica o início do programa.

9.2 Declaração de Variável:

```
ni value = 10;
```

- Declara uma variável chamada ‘value‘ e a inicializa com o valor ‘10‘.

9.3 Leitura de Entrada:

leia x;
leia y;
- Lê dois valores de entrada e os armazena nas variáveis ‘x‘ e ‘y‘.
Laço ‘para‘:

```
para (i in range(value)) {
    escreva i;
}
```

- Um laço ‘para‘ que itera de ‘0‘ até ‘value - 1‘ (ou seja, de 0 a 9, porque ‘value‘ é 10).
- Em cada iteração, imprime o valor de ‘i‘.

9.4 Laço ‘enquanto‘:

```
enquanto (x < 20) {
    x = x + 1.5;
    escreva x;
}
```

- Um laço ‘enquanto‘ que continua enquanto ‘x‘ for menor que ‘20‘.
- Dentro do laço, ‘x‘ é incrementado em ‘1.5‘ a cada iteração e o novo valor de ‘x‘ é impresso.

9.5 Impressão Final:

escreva x + y;
- Imprime a soma de ‘x‘ e ‘y‘.

9.6 Fim do Programa:

fmprograma
- Indica o fim do programa.

10 Codigos Gerados apos a compilação do arquivo codigo_input.txt com o compilador_a3_v2.py

10.1 Tokens_output.txt

Tokens são unidades léxicas básicas em um programa de computador. Eles são os blocos de construção fundamentais da análise léxica, que é a primeira etapa do processo de compilação. Durante a análise léxica, o código-fonte é dividido em tokens para facilitar a análise posterior.

Arquivo criado para monitorar a saída dos tokens e suas definições que foram geradas após a compilação do arquivo codigo_Input.txt

```
Token tipo = INPROGRAMA, valor = inprograma, linha = 1
Token tipo = NI, valor = ni, linha = 1
Token tipo = VARIAVEL, valor = value, linha = 1
Token tipo = OP_ATRIB_IGUAL, valor = =, linha = 1
Token tipo = INTEIRO, valor = 10, linha = 1
Token tipo = OP_FINAL_LINHA_PONTO_VIRGULA, valor = ;, linha = 1
Token tipo = LEIA, valor = leia, linha = 1
Token tipo = VARIAVEL, valor = x, linha = 1
Token tipo = OP_FINAL_LINHA_PONTO_VIRGULA, valor = ;, linha = 1
Token tipo = LEIA, valor = leia, linha = 1
Token tipo = VARIAVEL, valor = y, linha = 1
Token tipo = OP_FINAL_LINHA_PONTO_VIRGULA, valor = ;, linha = 1
Token tipo = PARA, valor = para, linha = 1
Token tipo = OP_PAR_ESQUERDO, valor = (, linha = 1
Token tipo = VARIAVEL, valor = i, linha = 1
Token tipo = IN, valor = in, linha = 1
Token tipo = RANGE, valor = range, linha = 1
Token tipo = OP_PAR_ESQUERDO, valor = (, linha = 1
Token tipo = VARIAVEL, valor = value, linha = 1
Token tipo = OP_PAR_DIREITO, valor = ), linha = 1
```

```
Token tipo = OP_PAR_DIREITO, valor = ), linha = 1
Token tipo = OP_CHAVE_ESQUERDA, valor = {, linha = 1
Token tipo = ESCREVA, valor = escreva, linha = 1
Token tipo = VARIAVEL, valor = i, linha = 1
Token tipo = OP_FINAL_LINHA_PONTO_VIRGULA, valor = ;, linha = 1
Token tipo = OP_CHAVE_DIREITA, valor = }, linha = 1
Token tipo = ENQUANTO, valor = enquanto, linha = 1
Token tipo = OP_PAR_ESQUERDO, valor = (, linha = 1
Token tipo = VARIAVEL, valor = x, linha = 1
Token tipo = LESS, valor = <, linha = 1
Token tipo = INTEIRO, valor = 20, linha = 1
Token tipo = OP_PAR_DIREITO, valor = ), linha = 1
Token tipo = OP_CHAVE_ESQUERDA, valor = {, linha = 1
Token tipo = VARIAVEL, valor = x, linha = 1
Token tipo = OP_ATRIB_IGUAL, valor = =, linha = 1
Token tipo = VARIAVEL, valor = x, linha = 1
Token tipo = OP_SOMA, valor = +, linha = 1
Token tipo = FLOAT, valor = 1.5, linha = 1
Token tipo = OP_FINAL_LINHA_PONTO_VIRGULA, valor = ;, linha = 1
Token tipo = ESCREVA, valor = escreva, linha = 1
Token tipo = VARIAVEL, valor = x, linha = 1
Token tipo = OP_FINAL_LINHA_PONTO_VIRGULA, valor = ;, linha = 1
Token tipo = OP_CHAVE_DIREITA, valor = }, linha = 1
Token tipo = ESCREVA, valor = escreva, linha = 1
Token tipo = VARIAVEL, valor = x, linha = 1
Token tipo = OP_SOMA, valor = +, linha = 1
Token tipo = VARIAVEL, valor = y, linha = 1
Token tipo = OP_FINAL_LINHA_PONTO_VIRGULA, valor = ;, linha = 1
Token tipo = FMPROGRAMA, valor = fmprograma, linha = 1
```

10.2 parser.out.py

O arquivo parser.out.py é responsável por realizar a análise e validação sintática de um programa, seguindo as regras de uma gramática definida. Ele utiliza o método de análise LALR (Look-Ahead LR) para verificar a conformidade do código com a gramática especificada.

10.2.1 Gramatica

As regras da gramática são listadas no início do código. Elas indicam como cada parte do programa deve ser estruturada. Por exemplo, programa INPROGRAMA declarações FMPROGRAMA significa que um programa começa com INPROGRAMA, seguido por declarações e termina com FMPROGRAMA. Grammar

```
Rule 0     S' -> programa
Rule 1     programa -> INPROGRAMA declaracoes FMPROGRAMA
Rule 2     declaracoes -> declaracao
Rule 3     declaracoes -> declaracoes declaracao
```

```
Rule 4     declaracao -> NI VARIAVEL OP_ATRIB_IGUAL
expressao OP_FINAL_LINHA_PONTO_VIRGULA
Rule 5     declaracao -> VARIAVEL OP_ATRIB_IGUAL
expressao OP_FINAL_LINHA_PONTO_VIRGULA
Rule 6     declaracao -> ESCREVA expressao
OP_FINAL_LINHA_PONTO_VIRGULA
Rule 7     declaracao -> LEIA VARIAVEL
OP_FINAL_LINHA_PONTO_VIRGULA
Rule 8     declaracao -> PARA
OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO expressao
OP_PAR_DIREITO OP_PAR_DIREITO
OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
Rule 9     declaracao -> ENQUANTO
OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
Rule 10    expressao -> INTEIRO
Rule 11    expressao -> FLOAT
Rule 12    expressao -> VARIAVEL
Rule 13    expressao -> VARIAVEL OP_SOMA expressao
Rule 14    condicao -> expressao LESS expressao
Rule 15    condicao -> expressao GREATER expressao
```

10.2.2 Terminais

Terminais referem-se aos símbolos de entrada em uma gramática formal. Eles são os elementos mais básicos que compõem uma linguagem. Durante o processo de análise sintática, os terminais são comparados com os símbolos da gramática para reconhecer padrões e estruturas na entrada fornecida.

```
Terminals, with rules where they appear

ENQUANTO             : 9
ESCREVA              : 6
FLOAT                : 11
FMPROGRAMA           : 1
GREATER              : 15
IN                   : 8
INPROGRAMA           : 1
INTEIRO              : 10
LEIA                 : 7
LESS                 : 14
NI                   : 4
OP_ATRIB_IGUAL       : 4 5
OP_CHAVE_DIREITA     : 8 9
OP_CHAVE_ESQUERDA    : 8 9
OP_FINAL_LINHA_PONTO_VIRGULA : 4 5 6 7
OP_PAR_DIREITO       : 8 8 9
```

```
OP_PAR_ESQUERDO      : 8 8 9
OP_SOMA              : 13
PARA                 : 8
RANGE                : 8
VARIAVEL             : 4 5 7 8 12 13
error                :
```

10.2.3 Não terminais

Os "não-terminais" são símbolos na gramática formal que podem ser substituídos por outros símbolos, incluindo terminais e outros não- terminais, por meio das regras de produção. Eles representam conceitos abstratos ou estruturas na linguagem. Durante o processo de análise sintática, os não-terminais são expandidos ou substituídos por meio das regras de produção até que apenas terminais permaneçam.

```
Nonterminals, with rules where they appear
condicao             : 9
declaracao           : 2 3
declaracoes          : 1 3 8 9
expressao            : 4 5 6 8 13 14 14 15 15
programa             : 0
```

10.2.4 Parsing method: LALR

O método LALR (Look-Ahead LR) é uma técnica de análise sintática eficiente e amplamente utilizada em compiladores e ferramentas de processamento de linguagens. Ele é capaz de lidar com gramáticas complexas e permite analisar uma variedade de linguagens de programação e especificações formais.

```
state 0

    (0) S' -> . programa
    (1) programa -> . INPROGRAMA declaracoes FMPROGRAMA

    INPROGRAMA      shift and go to state 2

    programa                        shift and go to state 1


state 1

    (0) S' -> programa .


state 2

    (1) programa -> INPROGRAMA . declaracoes FMPROGRAMA
    (2) declaracoes -> . declaracao
```

```
    (3) declaracoes -> . declaracoes declaracao
    (4) declaracao -> . NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (5) declaracao -> . VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (6) declaracao -> . ESCREVA expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (7) declaracao -> . LEIA VARIAVEL
    OP_FINAL_LINHA_PONTO_VIRGULA
    (8) declaracao -> . PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO

    expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
    (9) declaracao -> . ENQUANTO OP_PAR_ESQUERDO condicao

    OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    NI              shift and go to state 5
    VARIAVEL        shift and go to state 6
    ESCREVA         shift and go to state 7
    LEIA            shift and go to state 8
    PARA            shift and go to state 9
    ENQUANTO        shift and go to state 10

    declaracoes                   shift and go to state 3
    declaracao                    shift and go to state 4


state 3

    (1) programa -> INPROGRAMA declaracoes . FMPROGRAMA
    (3) declaracoes -> declaracoes . declaracao
    (4) declaracao -> . NI VARIAVEL
    OP_ATRIB_IGUAL expressao OP_FINAL_LINHA_PONTO_VIRGULA
    (5) declaracao -> . VARIAVEL
    OP_ATRIB_IGUAL expressao OP_FINAL_LINHA_PONTO_VIRGULA
    (6) declaracao -> . ESCREVA expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (7) declaracao -> . LEIA VARIAVEL
    OP_FINAL_LINHA_PONTO_VIRGULA
    (8) declaracao -> . PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO
    expressao OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
    OP_CHAVE_DIREITA
    (9) declaracao -> . ENQUANTO OP_PAR_ESQUERDO condicao
    OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    FMPROGRAMA      shift and go to state 11
    NI              shift and go to state 5
    VARIAVEL        shift and go to state 6
```

```
    ESCREVA         shift and go to state 7
    LEIA            shift and go to state 8
    PARA            shift and go to state 9
    ENQUANTO        shift and go to state 10

    declaracao                      shift and go to state 12


state 4

    (2) declaracoes -> declaracao .

    FMPROGRAMA      reduce using rule 2 (declaracoes -> declaracao .)
    NI              reduce using rule 2 (declaracoes -> declaracao .)
    VARIAVEL        reduce using rule 2 (declaracoes -> declaracao .)
    ESCREVA         reduce using rule 2 (declaracoes -> declaracao .)
    LEIA            reduce using rule 2 (declaracoes -> declaracao .)
    PARA            reduce using rule 2 (declaracoes -> declaracao .)
    ENQUANTO        reduce using rule 2 (declaracoes -> declaracao .)
    OP_CHAVE_DIREITA reduce using rule 2 (declaracoes -> declaracao .)


state 5

    (4) declaracao -> NI . VARIAVEL
    OP_ATRIB_IGUAL expressao OP_FINAL_LINHA_PONTO_VIRGULA

    VARIAVEL        shift and go to state 13


state 6

    (5) declaracao -> VARIAVEL .
    OP_ATRIB_IGUAL expressao OP_FINAL_LINHA_PONTO_VIRGULA

    OP_ATRIB_IGUAL  shift and go to state 14


state 7

    (6) declaracao -> ESCREVA . expressao OP_FINAL_LINHA_PONTO_VIRGULA
    (10) expressao -> . INTEIRO
    (11) expressao -> . FLOAT
    (12) expressao -> . VARIAVEL
    (13) expressao -> . VARIAVEL OP_SOMA expressao

    INTEIRO         shift and go to state 16
    FLOAT           shift and go to state 17
    VARIAVEL        shift and go to state 18
```

```
    expressao                      shift and go to state 15


state 8

    (7) declaracao -> LEIA . VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA

    VARIAVEL        shift and go to state 19

state 9

    (8) declaracao -> PARA . OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO
    expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    OP_PAR_ESQUERDO shift and go to state 20

state 10

    (9) declaracao -> ENQUANTO .
    OP_PAR_ESQUERDO condicao
    OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    OP_PAR_ESQUERDO shift and go to state 21

state 11

    (1) programa -> INPROGRAMA declaracoes FMPROGRAMA .

    $end    reduce using rule 1 (programa -> INPROGRAMA declaracoes
    FMPROGRAMA .)

state 12

    (3) declaracoes -> declaracoes declaracao .

    FMPROGRAMA      reduce using rule 3 (declaracoes -> declaracoes
    declaracao .)
    NI              reduce using rule 3 (declaracoes -> declaracoes
    declaracao .)
    VARIAVEL        reduce using rule 3 (declaracoes -> declaracoes
    declaracao .)
    ESCREVA         reduce using rule 3 (declaracoes -> declaracoes
    declaracao .)
    LEIA            reduce using rule 3 (declaracoes -> declaracoes
    declaracao .)
    PARA            reduce using rule 3 (declaracoes -> declaracoes
```

```
    declaracao .)
    ENQUANTO        reduce using rule 3 (declaracoes -> declaracoes
    declaracao .)
    OP_CHAVE_DIREITA reduce using rule 3 (declaracoes -> declaracoes
    declaracao .)


state 13

    (4) declaracao -> NI VARIAVEL .
    OP_ATRIB_IGUAL expressao OP_FINAL_LINHA_PONTO_VIRGULA

    OP_ATRIB_IGUAL  shift and go to state 22


state 14

    (5) declaracao -> VARIAVEL OP_ATRIB_IGUAL .
    expressao OP_FINAL_LINHA_PONTO_VIRGULA
    (10) expressao -> . INTEIRO
    (11) expressao -> . FLOAT
    (12) expressao -> . VARIAVEL
    (13) expressao -> . VARIAVEL OP_SOMA expressao

    INTEIRO         shift and go to state 16
    FLOAT           shift and go to state 17
    VARIAVEL        shift and go to state 18

    expressao                       shift and go to state 23


state 15

    (6) declaracao -> ESCREVA expressao .

    OP_FINAL_LINHA_PONTO_VIRGULA

    OP_FINAL_LINHA_PONTO_VIRGULA shift and go to state 24


state 16

    (10) expressao -> INTEIRO .

    OP_FINAL_LINHA_PONTO_VIRGULA reduce using rule 10 (expressao ->
    INTEIRO .)
    LESS            reduce using rule 10 (expressao -> INTEIRO .)
    GREATER         reduce using rule 10 (expressao -> INTEIRO .)
    OP_PAR_DIREITO  reduce using rule 10 (expressao -> INTEIRO .)
```

```
state 17

    (11) expressao -> FLOAT .

    OP_FINAL_LINHA_PONTO_VIRGULA reduce using rule 11 (expressao -> FLOAT
    .)
    LESS            reduce using rule 11 (expressao -> FLOAT .)
    GREATER         reduce using rule 11 (expressao -> FLOAT .)
    OP_PAR_DIREITO  reduce using rule 11 (expressao -> FLOAT .)


state 18

    (12) expressao -> VARIAVEL .
    (13) expressao -> VARIAVEL . OP_SOMA expressao

    OP_FINAL_LINHA_PONTO_VIRGULA reduce using rule 12 (expressao ->
    VARIAVEL .)
    LESS            reduce using rule 12 (expressao -> VARIAVEL .)
    GREATER         reduce using rule 12 (expressao -> VARIAVEL .)
    OP_PAR_DIREITO  reduce using rule 12 (expressao -> VARIAVEL .)
    OP_SOMA         shift and go to state 25


state 19

    (7) declaracao -> LEIA VARIAVEL . OP_FINAL_LINHA_PONTO_VIRGULA

    OP_FINAL_LINHA_PONTO_VIRGULA shift and go to state 26


state 20

    (8) declaracao -> PARA OP_PAR_ESQUERDO . VARIAVEL IN RANGE
    OP_PAR_ESQUERDO
    expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    VARIAVEL        shift and go to state 27


state 21

    (9) declaracao -> ENQUANTO OP_PAR_ESQUERDO . condicao OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
    (14) condicao -> . expressao LESS expressao
    (15) condicao -> . expressao GREATER expressao
    (10) expressao -> . INTEIRO
    (11) expressao -> . FLOAT
```

```
    (12) expressao -> . VARIAVEL
    (13) expressao -> . VARIAVEL OP_SOMA expressao

    INTEIRO         shift and go to state 16
    FLOAT           shift and go to state 17
    VARIAVEL        shift and go to state 18

    condicao                        shift and go to state 28
    expressao                       shift and go to state 29


state 22

    (4) declaracao -> NI VARIAVEL OP_ATRIB_IGUAL . expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (10) expressao -> . INTEIRO
    (11) expressao -> . FLOAT
    (12) expressao -> . VARIAVEL
    (13) expressao -> . VARIAVEL OP_SOMA expressao

    INTEIRO         shift and go to state 16
    FLOAT           shift and go to state 17
    VARIAVEL        shift and go to state 18

    expressao                       shift and go to state 30


state 23

    (5) declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao .
    OP_FINAL_LINHA_PONTO_VIRGULA

    OP_FINAL_LINHA_PONTO_VIRGULA shift and go to state 31


state 24

    (6) declaracao -> ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA .

    FMPROGRAMA      reduce using rule 6
    (declaracao -> ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA .)
    NI              reduce using rule 6
    (declaracao -> ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA .)
    VARIAVEL        reduce using rule 6
    (declaracao -> ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA .)
    ESCREVA         reduce using rule 6
    (declaracao -> ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA .)
    LEIA            reduce using rule 6
    (declaracao -> ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA .)
```

```
    PARA            reduce using rule 6
    (declaracao -> ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA .)
    ENQUANTO        reduce using rule 6
    (declaracao -> ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA .)
    OP_CHAVE_DIREITA reduce using rule 6
    (declaracao -> ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA .)


state 25

    (13) expressao -> VARIAVEL OP_SOMA . expressao
    (10) expressao -> . INTEIRO
    (11) expressao -> . FLOAT
    (12) expressao -> . VARIAVEL
    (13) expressao -> . VARIAVEL OP_SOMA expressao

    INTEIRO         shift and go to state 16
    FLOAT           shift and go to state 17
    VARIAVEL        shift and go to state 18

    expressao                   shift and go to state 32


state 26

    (7) declaracao -> LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA .

    FMPROGRAMA      reduce using rule 7
    (declaracao -> LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA .)
    NI              reduce using rule 7
    (declaracao -> LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA .)
    VARIAVEL        reduce using rule 7
    (declaracao -> LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA .)
    ESCREVA         reduce using rule 7
    (declaracao -> LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA .)
    LEIA            reduce using rule 7
    (declaracao -> LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA .)
    PARA            reduce using rule 7
    (declaracao -> LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA .)
    ENQUANTO        reduce using rule 7
    (declaracao -> LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA .)
    OP_CHAVE_DIREITA reduce using rule 7
    (declaracao -> LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA .)


state 27

    (8) declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL .
    IN RANGE OP_PAR_ESQUERDO expressao OP_PAR_DIREITO OP_PAR_DIREITO
```

```
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    IN              shift and go to state 33


state 28

    (9) declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao .
    OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    OP_PAR_DIREITO  shift and go to state 34


state 29

    (14) condicao -> expressao . LESS expressao
    (15) condicao -> expressao . GREATER expressao

    LESS            shift and go to state 35
    GREATER         shift and go to state 36


state 30

    (4) declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao .
    OP_FINAL_LINHA_PONTO_VIRGULA

    OP_FINAL_LINHA_PONTO_VIRGULA shift and go to state 37


state 31

    (5) declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .

    FMPROGRAMA      reduce using rule 5
    (declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    NI              reduce using rule 5
    (declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    VARIAVEL        reduce using rule 5
    (declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    ESCREVA         reduce using rule 5
    (declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    LEIA            reduce using rule 5
    (declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao
```

```
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    PARA            reduce using rule 5
    (declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    ENQUANTO        reduce using rule 5
    (declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    OP_CHAVE_DIREITA reduce using rule 5
    (declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)


state 32

    (13) expressao -> VARIAVEL OP_SOMA expressao .

    OP_FINAL_LINHA_PONTO_VIRGULA reduce using rule 13
    (expressao -> VARIAVEL OP_SOMA expressao .)
    LESS            reduce using rule 13
    (expressao -> VARIAVEL OP_SOMA expressao .)
    GREATER         reduce using rule 13
    (expressao -> VARIAVEL OP_SOMA expressao .)
    OP_PAR_DIREITO  reduce using rule 13
    (expressao -> VARIAVEL OP_SOMA expressao .)


state 33

    (8) declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL IN . RANGE
    OP_PAR_ESQUERDO
    expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    RANGE           shift and go to state 38


state 34

    (9) declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao
    OP_PAR_DIREITO . OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    OP_CHAVE_ESQUERDA shift and go to state 39


state 35

    (14) condicao -> expressao LESS . expressao
    (10) expressao -> . INTEIRO
    (11) expressao -> . FLOAT
```

```
    (12) expressao -> . VARIAVEL
    (13) expressao -> . VARIAVEL OP_SOMA expressao

    INTEIRO         shift and go to state 16
    FLOAT           shift and go to state 17
    VARIAVEL        shift and go to state 18

    expressao                   shift and go to state 40


state 36

    (15) condicao -> expressao GREATER . expressao
    (10) expressao -> . INTEIRO
    (11) expressao -> . FLOAT
    (12) expressao -> . VARIAVEL
    (13) expressao -> . VARIAVEL OP_SOMA expressao

    INTEIRO         shift and go to state 16
    FLOAT           shift and go to state 17
    VARIAVEL        shift and go to state 18

    expressao                   shift and go to state 41


state 37

    (4) declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .

    FMPROGRAMA      reduce using rule 4
    (declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    NI              reduce using rule 4
    (declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    VARIAVEL        reduce using rule 4
    (declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    ESCREVA         reduce using rule 4
    (declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    LEIA            reduce using rule 4
    (declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    PARA            reduce using rule 4
    (declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)
    ENQUANTO        reduce using rule 4
```

```
    (declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)

    OP_CHAVE_DIREITA reduce using rule 4
    (declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA .)


state 38

    (8) declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE .
    OP_PAR_ESQUERDO expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    OP_PAR_ESQUERDO shift and go to state 42


state 39

    (9) declaracao -> ENQUANTO OP_PAR_ESQUERDO
    condicao OP_PAR_DIREITO OP_CHAVE_ESQUERDA . declaracoes OP_CHAVE_DIREITA
    (2) declaracoes -> . declaracao
    (3) declaracoes -> . declaracoes declaracao
    (4) declaracao -> . NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (5) declaracao -> . VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (6) declaracao -> . ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA
    (7) declaracao -> . LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA
    (8) declaracao -> . PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
    (9) declaracao -> . ENQUANTO OP_PAR_ESQUERDO condicao
    OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    NI              shift and go to state 5
    VARIAVEL        shift and go to state 6
    ESCREVA         shift and go to state 7
    LEIA            shift and go to state 8
    PARA            shift and go to state 9
    ENQUANTO        shift and go to state 10

    declaracoes                   shift and go to state 43
    declaracao                    shift and go to state 4


state 40

    (14) condicao -> expressao LESS expressao .
```

```
    OP_PAR_DIREITO  reduce using rule 14 (condicao -> expressao LESS expressao .)


state 41

    (15) condicao -> expressao GREATER expressao .

    OP_PAR_DIREITO  reduce using rule 15 (condicao -> expressao GREATER
    expressao .)


state 42

    (8) declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO .
    expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
    (10) expressao -> . INTEIRO
    (11) expressao -> . FLOAT
    (12) expressao -> . VARIAVEL
    (13) expressao -> . VARIAVEL OP_SOMA expressao

    INTEIRO         shift and go to state 16
    FLOAT           shift and go to state 17
    VARIAVEL        shift and go to state 18

    expressao                       shift and go to state 44


state 43

    (9) declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao
    OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes . OP_CHAVE_DIREITA
    (3) declaracoes -> declaracoes . declaracao
    (4) declaracao -> . NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (5) declaracao -> . VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (6) declaracao -> . ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA
    (7) declaracao -> . LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA
    (8) declaracao -> . PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
    (9) declaracao -> . ENQUANTO OP_PAR_ESQUERDO condicao
    OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    OP_CHAVE_DIREITA shift and go to state 45
    NI              shift and go to state 5
```

```
    VARIAVEL        shift and go to state 6
    ESCREVA         shift and go to state 7
    LEIA            shift and go to state 8
    PARA            shift and go to state 9
    ENQUANTO        shift and go to state 10

    declaracao                  shift and go to state 12

state 44

    (8) declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO
    expressao . OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA
    declaracoes OP_CHAVE_DIREITA

    OP_PAR_DIREITO  shift and go to state 46

state 45

    (9) declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA .

    FMPROGRAMA      reduce using rule 9
    (declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA .)
    NI              reduce using rule 9
    (declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA .)
    VARIAVEL        reduce using rule 9
    (declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA .)
    ESCREVA         reduce using rule 9
    (declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA .)
    LEIA            reduce using rule 9
    (declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA .)
    PARA            reduce using rule 9
    (declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA .)
    ENQUANTO        reduce using rule 9
    (declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA .)
    OP_CHAVE_DIREITA reduce using rule 9
    (declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA .)

state 46
```

```
    (8) declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO expressao
    OP_PAR_DIREITO . OP_PAR_DIREITO OP_CHAVE_ESQUERDA
    declaracoes OP_CHAVE_DIREITA

    OP_PAR_DIREITO  shift and go to state 47

state 47

    (8) declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO
    expressao OP_PAR_DIREITO OP_PAR_DIREITO .
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    OP_CHAVE_ESQUERDA shift and go to state 48

state 48

    (8) declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA . declaracoes OP_CHAVE_DIREITA
    (2) declaracoes -> . declaracao
    (3) declaracoes -> . declaracoes declaracao
    (4) declaracao -> . NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (5) declaracao -> . VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (6) declaracao -> . ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA
    (7) declaracao -> . LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA
    (8) declaracao -> . PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
    (9) declaracao -> . ENQUANTO OP_PAR_ESQUERDO condicao
    OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    NI              shift and go to state 5
    VARIAVEL        shift and go to state 6
    ESCREVA         shift and go to state 7
    LEIA            shift and go to state 8
    PARA            shift and go to state 9
    ENQUANTO        shift and go to state 10

    declaracoes                   shift and go to state 49
    declaracao                    shift and go to state 4

state 49

    (8) declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO expressao OP_PAR_DIREITO OP_PAR_DIREITO
```

```
    OP_CHAVE_ESQUERDA declaracoes . OP_CHAVE_DIREITA
    (3) declaracoes -> declaracoes . declaracao
    (4) declaracao -> . NI VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (5) declaracao -> . VARIAVEL OP_ATRIB_IGUAL expressao
    OP_FINAL_LINHA_PONTO_VIRGULA
    (6) declaracao -> . ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA
    (7) declaracao -> . LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA
    (8) declaracao -> . PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA
    (9) declaracao -> . ENQUANTO OP_PAR_ESQUERDO condicao
    OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA

    OP_CHAVE_DIREITA shift and go to state 50
    NI              shift and go to state 5
    VARIAVEL        shift and go to state 6
    ESCREVA         shift and go to state 7
    LEIA            shift and go to state 8
    PARA            shift and go to state 9
    ENQUANTO        shift and go to state 10

    declaracao                  shift and go to state 12

state 50

    (8) declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE
    OP_PAR_ESQUERDO expressao OP_PAR_DIREITO OP_PAR_DIREITO
    OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA .

    FMPROGRAMA      reduce using rule 8 (declaracao -> PARA
    OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO expressao
    OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
    OP_CHAVE_DIREITA .)
    NI              reduce using rule 8 (declaracao -> PARA
    OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO expressao
    OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
    OP_CHAVE_DIREITA .)
    VARIAVEL        reduce using rule 8 (declaracao -> PARA
    OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO expressao
    OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
    OP_CHAVE_DIREITA .)
    ESCREVA         reduce using rule 8 (declaracao -> PARA
    OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO expressao
    OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
    OP_CHAVE_DIREITA .)
    LEIA            reduce using rule 8 (declaracao -> PARA
    OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO expressao
    OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
```

```
OP_CHAVE_DIREITA .)
PARA            reduce using rule 8 (declaracao -> PARA
OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO expressao
OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
OP_CHAVE_DIREITA .)
ENQUANTO        reduce using rule 8 (declaracao -> PARA
OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO expressao
OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
OP_CHAVE_DIREITA .)
OP_CHAVE_DIREITA reduce using rule 8 (declaracao -> PARA
OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO expressao
OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
OP_CHAVE_DIREITA .)
```

10.3 Parser.tab.py

10.3.1 lr_method: Método de análise sintática

O método de análise sintática (parser method) define como a gramática será processada durante a análise do código fonte. Neste caso, o método utilizado é o LALR (Look-Ahead Left-to-Right), que é uma técnica eficiente para analisar gramáticas livres de contexto. Ele trabalha percorrendo a entrada da esquerda para a direita, usando um mecanismo de antecipação (look-ahead) para decidir quais ações tomar durante o processo de análise.

_lr_method: Define o método usado para analisar a gramática. Neste caso, é utilizado o método LALR (Look-Ahead Left-to-Right).

```
_lr_method = 'LALR'
```

_lr_signature: Contém uma assinatura da gramática. Esta é uma representação compacta da gramática definida nas regras abaixo.

```
_lr_signature = 'ENQUANTO ESCREVA FLOAT FMPROGRAMA GREATER IN INPROGRAMA
INTEIRO LEIA LESS NI OP_ATRIB_IGUAL OP_CHAVE_DIREITA OP_CHAVE_ESQUERDA
OP_FINAL_LINHA_PONTO_VIRGULA OP_PAR_DIREITO OP_PAR_ESQUERDO OP_SOMA PARA
RANGE VARIAVEL\n    programa : INPROGRAMA declaracoes FMPROGRAMA\n    \n
declaracoes : declaracao\n    \n    declaracoes : declaracoes
declaracao\n    \n    declaracao : NI VARIAVEL OP_ATRIB_IGUAL expressao
OP_FINAL_LINHA_PONTO_VIRGULA\n    \n    declaracao : VARIAVEL
OP_ATRIB_IGUAL expressao OP_FINAL_LINHA_PONTO_VIRGULA\n    \n
declaracao : ESCREVA expressao OP_FINAL_LINHA_PONTO_VIRGULA\n    \n
declaracao : LEIA VARIAVEL OP_FINAL_LINHA_PONTO_VIRGULA\n    \n
declaracao : PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO
expressao OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
OP_CHAVE_DIREITA\n    \n    declaracao : ENQUANTO OP_PAR_ESQUERDO
condicao OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes OP_CHAVE_DIREITA\n
\n    expressao : INTEIRO\n              | FLOAT\n              |
VARIAVEL\n              | VARIAVEL OP_SOMA expressao\n    \n    condicao
: expressao LESS expressao\n              | expressao GREATER expressao\n
'
```

_lr_action_items: Mapeia os estados e os símbolos de entrada para as ações a serem tomadas pelo analisador. As chaves são os estados do analisador e os valores são pares de listas que mapeiam os símbolos de entrada para as ações. Por exemplo, quando o estado 2 (representando o símbolo "INPROGRAMA") é alcançado, a ação a ser tomada é reduzir para a produção 2.

```
_lr_action_items = {'INPROGRAMA':([0,],[2,]),end':([1,11,],
[0,-1,]),'NI':([2,3,4,12,24,26,31,37,39,43,45,48,49,50,],
[5,5,-2,-3,-6,-7,-5,-4,5,5,-9,5,5,-8,]),'VARIAVEL':
([2,3,4,5,7,8,12,14,20,21,22,24,25,26,31,35,36,37,39,42,43,45,48,49,50,],
[6,6,-2,13,18,19,-3,18,27,18,18,-6,18,-7,-5,18,18,-4,6,18,6,-9,6,6,-8,]),
'ESCREVA':([2,3,4,12,24,26,31,37,39,43,45,48,49,50,],
[7,7,-2,-3,-6,-7,-5,-4,7,7,-9,7,7,-8,]),'LEIA':
([2,3,4,12,24,26,31,37,39,43,45,48,49,50,],
[8,8,-2,-3,-6,-7,-5,-4,8,8,-9,8,8,-8,]),'PARA':
```

```
([2,3,4,12,24,26,31,37,39,43,45,48,49,50,],
[9,9,-2,-3,-6,-7,-5,-4,9,9,-9,9,9,-8,]),'ENQUANTO':
([2,3,4,12,24,26,31,37,39,43,45,48,49,50,],
[10,10,-2,-3,-6,-7,-5,-4,10,10,-9,10,10,-8,]),'FMPROGRAMA':
([3,4,12,24,26,31,37,45,50,],
[11,-2,-3,-6,-7,-5,-4,-9,-8,]),'OP_CHAVE_DIREITA':
([4,12,24,26,31,37,43,45,49,50,],
[-2,-3,-6,-7,-5,-4,45,-9,50,-8,]),'OP_ATRIB_IGUAL':([6,13,],
[14,22,]),'INTEIRO':([7,14,21,22,25,35,36,42,],
[16,16,16,16,16,16,16,16,]),'FLOAT':([7,14,21,22,25,35,36,42,],
[17,17,17,17,17,17,17,17,]),'OP_PAR_ESQUERDO':([9,10,38,],
[20,21,42,]),'OP_FINAL_LINHA_PONTO_VIRGULA':([15,16,17,18,19,23,30,32,],
[24,-10,-11,-12,26,31,37,-13,]),'LESS':([16,17,18,29,32,],
[-10,-11,-12,35,-13,]),'GREATER':([16,17,18,29,32,],
[-10,-11,-12,36,-13,]),'OP_PAR_DIREITO':([16,17,18,28,32,40,41,44,46,],
[-10,-11,-12,34,-13,-14,-15,46,47,]),'OP_SOMA':([18,],[25,]),'IN':([27,],
[33,]),'RANGE':([33,],[38,]),'OP_CHAVE_ESQUERDA':([34,47,],[39,48,]),}
```

_lr_goto_items: Mapeia os estados e os símbolos não-terminais (isto é, símbolos que podem ser expandidos em outras regras) para os próximos estados a serem alcançados. Por exemplo, quando o estado 21 é alcançado, o próximo símbolo não-terminal é uma expressão.

```
\_lr\_goto\_items = {'programa':([0,],[1,]),'declaracoes':([2,39,48,],
[3,43,49,]),'declaracao':([2,3,39,43,48,49,],
[4,12,4,12,4,12,]),'expressao':([7,14,21,22,25,35,36,42,],
[15,23,29,30,32,40,41,44,]),'condicao':([21,],[28,]),}
```

_lr_productions: Lista as produções da gramática, ou seja, as regras que definem como os símbolos podem ser expandidos em outros símbolos. Cada produção consiste em um símbolo não-terminal seguido por uma sequência de símbolos terminais e/ou não-terminais.

```
_lr_productions = [
  ("S' -> programa","S'",1,None,None,None),
  ('programa -> INPROGRAMA declaracoes
  FMPROGRAMA','programa',3,'p_programa','compilador_A3_v2.py',80),
  ('declaracoes ->
  declaracao','declaracoes',1,'p_declaracoes_single','compilador_A3_v2.py
  ',86),
  ('declaracoes -> declaracoes
  declaracao','declaracoes',2,'p_declaracoes_mult','compilador_A3_v2.py',
  92),
  ('declaracao -> NI VARIAVEL OP_ATRIB_IGUAL expressao
  OP_FINAL_LINHA_PONTO_VIRGULA','declaracao',5,'p_declaracao_ni_atribuica
  o','compilador_A3_v2.py',98),
  ('declaracao -> VARIAVEL OP_ATRIB_IGUAL expressao
  OP_FINAL_LINHA_PONTO_VIRGULA','declaracao',4,'p_declaracao_atribuicao',
  'compilador_A3_v2.py',112),
  ('declaracao -> ESCREVA expressao
```

```
  OP_FINAL_LINHA_PONTO_VIRGULA','declaracao',3,'p_declaracao_escreva','co
  mpilador_A3_v2.py',125),
  ('declaracao -> LEIA VARIAVEL
  OP_FINAL_LINHA_PONTO_VIRGULA','declaracao',3,'p_declaracao_leia','compi
  lador_A3_v2.py',131),
  ('declaracao -> PARA OP_PAR_ESQUERDO VARIAVEL IN RANGE OP_PAR_ESQUERDO
  expressao OP_PAR_DIREITO OP_PAR_DIREITO OP_CHAVE_ESQUERDA declaracoes
  OP_CHAVE_DIREITA','declaracao',12,'p_declaracao_para','compilador_A3_v2
  .py',140),
  ('declaracao -> ENQUANTO OP_PAR_ESQUERDO condicao OP_PAR_DIREITO
  OP_CHAVE_ESQUERDA declaracoes
  OP_CHAVE_DIREITA','declaracao',7,'p_declaracao_enquanto','compilador_A3
  _v2.py',148),
  ('expressao ->
  INTEIRO','expressao',1,'p_expressao','compilador_A3_v2.py',154),
  ('expressao ->
  FLOAT','expressao',1,'p_expressao','compilador_A3_v2.py',155),
  ('expressao ->
  VARIAVEL','expressao',1,'p_expressao','compilador_A3_v2.py',156),
  ('expressao -> VARIAVEL OP_SOMA
  expressao','expressao',3,'p_expressao','compilador_A3_v2.py',157),
  ('condicao -> expressao LESS
  expressao','condicao',3,'p_condicao','compilador_A3_v2.py',166),
  ('condicao -> expressao GREATER
  expressao','condicao',3,'p_condicao','compilador_A3_v2.py',167),
]
```

10.4 codigo_output.py

Arquivo criado para monitorar a saída dos tokens e suas definições que foram geradas após a compilação do arquivo codigo_Input.txt

```
value = 10
x = float(input('Digite um numero : '))
y = float(input('Digite um numero : '))
for i in range(value):
    print(i)
while x < 20:
    x = x + 1.5
    print(x)
print(x + y)
```

Este código em Python demonstra algumas estruturas de controle comuns em linguagens de programação. Primeiro, ele define duas variáveis x e y, atribuindo a elas os valores convertidos para ponto flutuante inseridos pelo usuário via entrada (input). Em seguida, entra em um loop for, iterando sobre os valores no intervalo de 0 a 9 (pois value é definido como 10). Dentro deste loop, cada valor de i é impresso. Após isso, há um loop while que

continua enquanto o valor de x for menor que 20. Dentro deste loop, x é incrementado em 1.5 a cada iteração e o novo valor de x é impresso.

Por fim, o código imprime a soma de x e y.

11 Equivalencia entre codigo Input e codigo output

O codigo input :

```
inprograma
    ni value = 10;
    leia x;
    leia y;
    para (i in range(value)) {
        escreva i;
    }
    enquanto (x < 20) {
        x = x + 1.5;
        escreva x;
    }
    escreva x + y;
fmprograma
```

o codigo output :

```
value = 10
x = float(input('Digite um numero : '))
y = float(input('Digite um numero : '))
for i in range(value):
    print(i)
while x < 20:
    x = x + 1.5
    print(x)
print(x + y)
```

12 Conclusão

Este documento abrange integralmente o processo de desenvolvimento de um compilador, desde a concepção até a implementação final, utilizando uma linguagem especifica, projetada para a compilação. Todos os passos, foram meticulosamente planejados e executados, usando a poderosa combinação da biblioteca PLY e Python.

Durante o desenvolvimento, cada aspecto crítico e cada bloco de código nos arquivos gerados após a execução do compilador, foram cuidadosamente comentados. Esse processo, não apenas facilitou a compreensão do funcionamento interno do compilador, mas também permitiu uma análise aprofundada das etapas de análise léxica, análise sintática, analise semântica, geração de código e otimização.

Além disso, enfatizou-se a importância da modularidade e da clareza na estruturação do código-fonte do compilador. Cada componente, foi projetado para ser coeso e reutilizável, seguindo as melhores práticas de engenharia de software.

Ao final, o compilador demonstrou não apenas a capacidade de traduzir eficientemente programas escritos na linguagem de origem para código executável em Python, mas, também serviu como um estudo de caso valioso para entender os princípios fundamentais por trás da construção de compiladores modernos.

Este trabalho, não só expandiu o conhecimento teórico e prático sobre compilação e interpretação de linguagens de programação, mas também estabeleceu uma base sólida para explorar futuras melhorias e extensões no compilador, preparando para enfrentar desafios mais complexos e inovadores no campo da computação.

Em suma, o processo de criação deste compilador não apenas consolidou o entendimento acadêmico, mas também reforçou a habilidade de transformar conceitos abstratos em soluções concretas e funcionais, promovendo assim um ambiente de aprendizado contínuo e crescimento profissional.

13 Referências Bibliográficas

References

[1] Beazley, D. M. (2020). *PLY (Python Lex-Yacc) Documentation.* Acessado a primeira vez em 30/05/2024 `http://www.dabeaz.com/ply/`

[2] Repositorio do Github do criador. "PLY Compiler Example." Acessado a primeira vez em 30/05/2024 `https://github.com/dabeaz/ply`

[3] Levine, John R., Tony Mason, and Doug Brown. *Lex & Yacc.* 2nd ed., O'Reilly Media, 1992. `https://d1.amobbs.com/bbs_upload782111/files_33/ourdev_584393GCYRF3.pdf`

[4] AEDB. (2014). "Desenvolvimento do Compilador da Linguagem Basico." In *Anais do Simpósio de Excelência em Gestão e Tecnologia (SEGeT).*Retirado do site `https://www.aedb.br/seget/arquivos/artigos14/11720295.pdf`