

**UNIVERSIDADE TUIUTI DO PARANÁ**

**SABRINA ELOISE NAWCKI**

**VL-D+: UMA FERRAMENTA DE ESTUDO PARA COMPILADORES  
ATRAVÉS DE UMA INTERFACE**

**CURITIBA**

**2020**

**SABRINA ELOISE NAWCKI**

**VL-D+: UMA FERRAMENTA DE ESTUDO PARA COMPILADORES  
ATRAVÉS DE UMA INTERFACE**

Trabalho de Conclusão de Curso apresentado ao  
curso de Bacharelado em Ciência da Computação da  
Faculdade de Ciências Exatas e de Tecnologia, da  
Universidade Tuiuti do Paraná, como requisito à  
obtenção ao grau de Bacharel.

Orientador: Prof. Diógenes Cogo Furlan

**CURITIBA**

**2020**

## LISTA DE FIGURAS

FIGURA 1 - COMPILADOR .....	17
FIGURA 2 - FASES DE UM COMPILADOR.....	18
FIGURA 3 - DECLARAÇÕES NA GRAMÁTICA D+.....	20
FIGURA 4 - COMANDOS NA GRAMÁTICA D+.....	21
FIGURA 5 - COMANDOS DE SELEÇÃO.....	22
FIGURA 6 - COMANDO DE REPETIÇÃO: ENQUANTO .....	23
FIGURA 7 - COMANDO DE REPETIÇÃO: FAÇA .....	23
FIGURA 8 - COMANDO DE REPETIÇÃO: REPITA.....	24
FIGURA 9 - COMANDO DE REPETIÇÃO: PARA .....	24
FIGURA 10 - EXPRESSÕES NA GRAMÁTICA D+ .....	25
FIGURA 11 - ANÁLISE LÉXICA .....	26
FIGURA 12 - CÓDIGO EM D+ PARA CÁLCULO DE FATORIAL .....	28
FIGURA 13 - CÓDIGO EM D+ COM ERRO LÉXICO.....	30
FIGURA 14 - ANÁLISE SINTÁTICA.....	31
FIGURA 15 - ÁRVORE SINTÁTICA CONCRETA.....	32
FIGURA 16 - ÁRVORE CONCRETA DECORADA.....	33
FIGURA 17 - ÁRVORE ABSTRATA .....	34
FIGURA 18 - CÓDIGO EM D+ COM ERROS SINTÁTICOS.....	35
FIGURA 19 - ANÁLISE SEMÂNTICA .....	36
FIGURA 20 - ÁRVORE SINTÁTICA ABSTRATA REDUZIDA.....	37
FIGURA 21 - TABELA DE SÍMBOLOS .....	38
FIGURA 22 - CÓDIGO INTERMEDIÁRIO .....	39
FIGURA 23 – GERADOR DE CÓDIGO .....	40
FIGURA 24 - INTERPRETADOR .....	41
FIGURA 25 - ÁRVORE DE DECISÃO PARA ANÁLISE DE INCLUSIVIDADE ..	44
FIGURA 26 - USABILIDADE .....	46
FIGURA 27 - ELEMENTOS DA EXPERIÊNCIA DO USUÁRIO .....	48
FIGURA 28 - INTERFACE PADRÃO DO SIMULADOR COMPILERSIM .....	50

FIGURA 29 - EXEMPLOS PRONTOS PARA EXECUÇÃO .....	51
FIGURA 30 - ANÁLISE DA LINGUAGEM E RESULTADOS ABORDADOS .....	52
FIGURA 31 - EXECUÇÃO DE UM CÓDIGO E A INDICAÇÃO DE ERROS.....	52
FIGURA 32 - ARQUITETURA DO PROTÓTIPO.....	54
FIGURA 33 - INTERFACE DE EDIÇÃO DE UM AUTÔMATO E RECONHECIMENTO .....	55
FIGURA 34 - INTERFACE DE EDIÇÃO DE GRAMÁTICAS E RECONHECIMENTO .....	56
FIGURA 35 - ESQUEMA DE TRADUÇÃO DO COMPILADOR VERTO .....	57
FIGURA 36 - INTERFACE DE EDIÇÃO DO COMPILADOR VERTO.....	58
FIGURA 37 - ABA: SAÍDA DO SINTÁTICO .....	59
FIGURA 38 - SAÍDA DA ANÁLISE SINTÁTICA.....	60
FIGURA 39 - INTERFACE COM OS AUTÔMATOS DA ANÁLISE SINTÁTICA	61
FIGURA 40 - ÁRVORE SINTÁTICA MONTADA.....	61
FIGURA 41 - TABELA DE SÍMBOLOS .....	62
FIGURA 42 - ÁRVORE SINTÁTICA .....	63
FIGURA 43 - ÁRVORE DE SÍMBOLOS .....	64
FIGURA 44 - JANELA DA IDE DO VISUAL STUDIO .....	67
FIGURA 45 - EXEMPLO DE EXPRESSÕES EM RAZOR .....	68
FIGURA 46 - EXEMPLO DO CANVAS .....	70
FIGURA 47 - DIAGRAMS.NET .....	71
FIGURA 48 - MOCKUP DA TELA INICIAL .....	73
FIGURA 49 - MOCKUP DA ANÁLISE LÉXICA .....	74
FIGURA 50 - MOCKUP DA ANÁLISE SINTÁTICA.....	75
FIGURA 51 - MOCKUP DA TABELA DE SÍMBOLOS.....	76
FIGURA 52 – LEGENDA DA BARRA DE MENUS .....	79
FIGURA 53 - TELA INICIAL SEM INTERAÇÃO DO USUÁRIO.....	80
FIGURA 54 - TELA INICIAL COM INTERAÇÃO DO USUÁRIO .....	80
FIGURA 55 - LEGENDA DO LADO ESQUERDO DA TELA INICIAL .....	81
FIGURA 56 - LEGENDA DO LADO DIREITO DA TELA INICIAL .....	82
FIGURA 57 - ABA DA ANÁLISE LÉXICA COM INTERAÇÃO DO USUÁRIO ...	82

FIGURA 58- ABA DA ANÁLISE SINTÁTICA COM INTERAÇÃO DO USUÁRIO .....	83
FIGURA 59- ABA DA TABELA DE SÍMBOLOS COM INTERAÇÃO DO USUÁRIO .....	83
FIGURA 60- ABA DE ERROS COM INTERAÇÃO DO USUÁRIO .....	84
FIGURA 61- HISTÓRICO DA ANÁLISE LÉXICA COM INTERAÇÃO DO USUÁRIO .....	84
FIGURA 62 - TELA DA GRAMÁTICA D+ SEM INTERAÇÃO DO USUÁRIO ....	85
FIGURA 63 - TELA DA GRAMÁTICA D+ DECLARAÇÕES .....	85
FIGURA 64 - TELA DO MANUAL SEM INTERAÇÃO DO USUÁRIO .....	86
FIGURA 65 - TELA DO MANUAL COMPONENTES DE TELA .....	86
FIGURA 66 - TELA DO MANUAL CAIXA DE TEXTO .....	86
FIGURA 67 - TELA DO MANUAL COMPILADOR .....	87
FIGURA 68 - TELA SOBRE .....	87
FIGURA 69 - CAIXA DE TEXTO .....	88
FIGURA 70 - FUNÇÃO ONKEYUP .....	89
FIGURA 71 - FUNÇÃO ONBACKSPACE .....	90
FIGURA 72 - FUNÇÃO APPLYTEXT .....	91
FIGURA 73 - EXPRESSÕES REGEX .....	91
FIGURA 74 - FUNÇÃO ONPASTE .....	92
FIGURA 75 - FUNÇÃO ONUSERCODEKEYUP .....	92
FIGURA 76 - FUNÇÃO ANALYSIS .....	93
FIGURA 77 - FUNÇÃO ANALYSISCASE1 .....	94
FIGURA 78 - FUNÇÃO STARTANALYSIS .....	95
FIGURA 79 - COORDENADAS PARA O AUTÔMATO FINITO DE STRINGS ....	96
FIGURA 80 - FUNÇÃO DYEIMAGE .....	97
FIGURA 81 - FUNÇÃO BACKSPACEEVENT .....	98
FIGURA 82 - FUNÇÃO ANALYSEENTIRECODE .....	99
FIGURA 83 - FUNÇÃO CALLFUNCTION EXPOSTO .....	100
FIGURA 84 - FUNÇÃO CALLFUNCTION COMPLETA .....	101
FIGURA 85 - FUNÇÃO LISTPARAM .....	102

FIGURA 86 - FUNÇÃO WRITEINTAB.....	103
FIGURA 87 - FUNÇÃO BACKSPACEEVENT ANÁLISE SINTÁTICA .....	104
FIGURA 88 - FUNÇÕES DA TABELA DE SÍMBOLOS .....	105
FIGURA 89 - FUNÇÕES DE INCLUSÃO E REMOÇÃO DE ERROS .....	106
FIGURA 90 - FUNÇÃO HIDEERRORTAB .....	106
FIGURA 91 - EXEMPLO PRÁTICO CÓDIGO COMPLETO.....	107
FIGURA 92 - EXEMPLO PRÁTICO ANÁLISE LÉXICA 'C' .....	108
FIGURA 93 - EXEMPLO PRÁTICO ANÁLISE LÉXICA 'ONST' .....	109
FIGURA 94 - EXEMPLO PRÁTICO ANÁLISE LÉXICA ''.....	111
FIGURA 95 - EXEMPLO PRÁTICO HISTÓRICO DA ANÁLISE LÉXICA 'CONST' .....	111
FIGURA 96 - EXEMPLO PRÁTICO ANÁLISE LÉXICA 'P' .....	112
FIGURA 97 - EXEMPLO PRÁTICO ANÁLISE LÉXICA 'T' .....	112
FIGURA 98 - EXEMPLO PRÁTICO ANÁLISE LÉXICA '=' .....	114
FIGURA 99 - EXEMPLO PRÁTICO HISTÓRICO DA ANÁLISE LÉXICA 'PI' ....	114
FIGURA 100 - EXEMPLO PRÁTICO ANÁLISE LÉXICA '3'.....	115
FIGURA 101 - EXEMPLO PRÁTICO HISTÓRICO DA ANÁLISE LÉXICA '=' ...	115
FIGURA 102 - EXEMPLO PRÁTICO ANÁLISE LÉXICA '.14'.....	116
FIGURA 103 - EXEMPLO PRÁTICO ANÁLISE LÉXICA ';' .....	117
FIGURA 104 - EXEMPLO PRÁTICO ANÁLISE LÉXICA '3.14' .....	118
FIGURA 105 - EXEMPLO PRÁTICO ANÁLISE SINTÁTICA.....	118
FIGURA 106 - EXEMPLO PRÁTICO TABELA DE SÍMBOLOS .....	118
FIGURA 107 - AUTÔMATO PARA SEPARADORES (ESTADO 1) .....	136
FIGURA 108 - AUTÔMATO PARA A VÍRGULA (ESTADO 1).....	136
FIGURA 109 - AUTÔMATO PARA O PONTO E VÍRGULA (ESTADO 1).....	136
FIGURA 110 - AUTÔMATO PARA OS PARÊNTESES (ESTADO 1).....	137
FIGURA 111 - AUTÔMATO PARA OS COLCHETES (ESTADO 1) .....	137
FIGURA 112 - AUTÔMATO PARA O OPERADOR DE ADIÇÃO (ESTADO 1) .	137
FIGURA 113 - AUTÔMATO PARA O OPERADOR DE SUBTRAÇÃO (ESTADO 1) .....	138

FIGURA 114 - AUTÔMATO PARA O OPERADOR DE MULTIPLICAÇÃO (ESTADO 1).....	138
FIGURA 115 - AUTÔMATO PARA NÚMEROS (ESTADO 2) .....	138
FIGURA 116 - AUTÔMATO PARA CARACTERE (ESTADO 4) .....	138
FIGURA 117 - AUTÔMATO PARA STRING (ESTADO 6) .....	139
FIGURA 118 - AUTÔMATO PARA O OPERADOR MAIOR '>' (ESTADO 7) .....	139
FIGURA 119 - AUTÔMATO PARA O OPERADOR MENOR '<' (ESTADO 8) ....	140
FIGURA 120 - AUTÔMATO PARA O OPERADOR IGUAL '=' (ESTADO 9) .....	140
FIGURA 121 - AUTÔMATO PARA VALORES EXCLUSIVOS (ESTADO 10) ...	141
FIGURA 122 - AUTÔMATO PARA DIVISÃO E COMENTÁRIOS (ESTADO 12) .....	141
FIGURA 123 - FORMULÁRIO INTRODUÇÃO .....	142
FIGURA 124 - FORMULÁRIO QUESTÃO 1.....	142
FIGURA 125 - FORMULÁRIO QUESTÃO 2.....	143
FIGURA 126 - FORMULÁRIO QUESTÃO 3.....	143
FIGURA 127 - FORMULÁRIO QUESTÃO 4.....	144
FIGURA 128 - FORMULÁRIO QUESTÃO 5.....	144
FIGURA 129 - FORMULÁRIO QUESTÃO 6.....	145
FIGURA 130 - FORMULÁRIO QUESTÃO 7.....	146
FIGURA 131 - FORMULÁRIO QUESTÃO 8.....	146
FIGURA 132 - FORMULÁRIO QUESTÃO 9.....	147
FIGURA 133 - FORMULÁRIO QUESTÃO 10.....	147
FIGURA 134 - FORMULÁRIO QUESTÃO 11.....	147
FIGURA 135 - FORMULÁRIO QUESTÃO 12.....	148
FIGURA 136 - GRAMÁTICA D+ DECLARAÇÕES .....	149
FIGURA 137 - GRAMÁTICA D+ COMANDOS.....	150
FIGURA 138 - GRAMÁTICA D+ EXPRESSÕES .....	151
FIGURA 139 - GRAMÁTICA D+ COMENTÁRIOS.....	152
FIGURA 140 - MANUAL DO USUÁRIO CAIXA DE TEXTO.....	153
FIGURA 141 - MANUAL DO USUÁRIO BOTÃO DE RECOMPILAR .....	153
FIGURA 142 - MANUAL DO USUÁRIO ABA DE SELEÇÃO .....	153

FIGURA 143 - MANUAL DO USUÁRIO ANÁLISE LÉXICA .....	154
FIGURA 144 - MANUAL DO USUÁRIO HISTÓRICO DA ANÁLISE LÉXICA..	154
FIGURA 145 - MANUAL DO USUÁRIO ANÁLISE SINTÁTICA .....	154
FIGURA 146 - TABELA DE SÍMBOLOS .....	154
FIGURA 147 - MANUAL DO USUÁRIO REGISTRO DE ERROS .....	155

## LISTA DE QUADROS

QUADRO 1 - EXEMPLO DE TOKENS, LEXEMA E PADRÃO .....	27
QUADRO 2 - TOKENS E LEXEMAS OBTIDOS PELA ANÁLISE LÉXICA DO CÓDIGO DE FATORIAL .....	29
QUADRO 3 - PRINCÍPIOS DO DESIGN UNIVERSAL.....	43
QUADRO 4 - CRITÉRIOS DE USABILIDADE.....	45
QUADRO 5 - COMPARAÇÃO DOS TRABALHOS RELACIONADOS .....	49
QUADRO 6 - EXEMPLO PRÁTICO ANÁLISE LÉXICA 'C'.....	108
QUADRO 7 - EXEMPLO PRÁTICO ANÁLISE LÉXICA 'ONST'.....	108
QUADRO 8 - EXEMPLO PRÁTICO ANÁLISE LÉXICA 'CONST' E '' .....	109
QUADRO 9 - EXEMPLO PRÁTICO ANÁLISE SINTÁTICA 'CONST' .....	110
QUADRO 10 - EXEMPLO PRÁTICO ANÁLISE LÉXICA '' E 'P' .....	111
QUADRO 11 - EXEMPLO PRÁTICO ANÁLISE LÉXICA 'T' .....	112
QUADRO 12 - EXEMPLO PRÁTICO ANÁLISE LÉXICA 'PI' E '='.....	113
QUADRO 13 - EXEMPLO PRÁTICO ANÁLISE SINTÁTICA 'PI' .....	113
QUADRO 14 - EXEMPLO PRÁTICO ANÁLISE LÉXICA '=' E '3' .....	114
QUADRO 15 - EXEMPLO PRÁTICO ANÁLISE LÉXICA '.14' .....	115
QUADRO 16 - EXEMPLO PRÁTICO ANÁLISE LÉXICA '3.14' E ';' .....	116
QUADRO 17 - EXEMPLO PRÁTICO ANÁLISE SINTÁTICA '3.14' E ';' .....	117
QUADRO 18 - AUTÔMATO FINITO CORRENTE .....	119
QUADRO 19 - ANÁLISE LÉXICA COM MAIS DE UM LEXEMA .....	120
QUADRO 20 - ANÁLISE LÉXICA DELETAR ÚLTIMO LEXEMA .....	120
QUADRO 21 - SETAS DO HISTÓRICO DA ANÁLISE LÉXICA.....	120
QUADRO 22 – DECLARAÇÃO DE CONSTANTE .....	121
QUADRO 23- DECLARAÇÃO DE VARIÁVEL .....	121
QUADRO 24 - DECLARAÇÃO DE PROCEDIMENTO.....	121
QUADRO 25 - DECLARAÇÃO DE MAIN .....	122
QUADRO 26 - DECLARAÇÃO DE FUNÇÃO .....	122
QUADRO 27 - DECLARAÇÃO DE CONSTANTE (ERRO ID).....	122

QUADRO 28 - DECLARAÇÃO DE CONSTANTE (ERRO ‘=’)	123
QUADRO 29 - DECLARAÇÃO DE CONSTANTE (ERRO LITERAL)	123
QUADRO 30 - DECLARAÇÃO DE CONSTANTE (ERRO ‘;’)	123
QUADRO 31 - DECLARAÇÃO DE VARIÁVEL (ERRO ESPEC-TIPO)	123
QUADRO 32 - DECLARAÇÃO DE VARIÁVEL (ERRO LISTA-VAR)	124
QUADRO 33 - DECLARAÇÃO DE VARIÁVEL (ERRO ‘;’)	124
QUADRO 34 - DECLARAÇÃO DE PROCEDIMENTO (ERRO ESPEC-TIPO)	124
QUADRO 35 - DECLARAÇÃO DE PROCEDIMENTO (ERRO ID)	124
QUADRO 36 - DECLARAÇÃO DE PROCEDIMENTO (ERRO ‘(’ )	125
QUADRO 37 - DECLARAÇÃO DE PROCEDIMENTO (ERRO ‘)’ )	125
QUADRO 38 - DECLARAÇÃO DE PROCEDIMENTO (ERRO BLOCO)	125
QUADRO 39 - DECLARAÇÃO DE PROCEDIMENTO (ERRO ENDSUB)	125

## **LISTA DE GRÁFICOS**

GRÁFICO 1 - Gráfico de Dificuldade de Aprendizado da Disciplina .....	127
GRÁFICO 2 - Gráfico de Métodos de Estudo .....	127
GRÁFICO 3 - Gráfico de Dificuldade no Aprendizado da Disciplina por Processo..	128
GRÁFICO 4 - Gráfico de Utilização de Aplicativo .....	128
GRÁFICO 5 - Gráfico de Satisfação VL-D+ .....	129
GRÁFICO 6 - Gráfico Propósito VL-D+ .....	130
GRÁFICO 7 - Gráfico Relevância VL-D+ .....	130
GRÁFICO 8 - Gráfico Utilização VL-D+.....	131
GRÁFICO 9 - Gráfico Auxílio Futuro VL-D+ .....	131

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	15
<b>2 TEORIA DE COMPILADORES.....</b>	17
<b>2.1 GRAMÁTICA .....</b>	19
2.1.1 Gramática D+.....	19
<b>2.2 ANÁLISE LÉXICA.....</b>	26
2.2.1 Reconhecimento de <i>tokens</i> .....	27
2.2.2 Exemplo de programa e sua análise léxica .....	28
<b>2.3 ANÁLISE SINTÁTICA .....</b>	30
2.3.1 Árvore sintática.....	31
2.3.2 Exemplo prático.....	34
<b>2.4 ANÁLISE SEMÂNTICA.....</b>	35
<b>2.5 TABELA DE SÍMBOLOS .....</b>	37
<b>2.6 GERAÇÃO DE CÓDIGO INTERMEDIÁRIO .....</b>	39
<b>2.7 GERAÇÃO DE CÓDIGO .....</b>	40
<b>2.8 INTERPRETADOR X COMPILADOR .....</b>	40
<b>3 INTERAÇÃO HUMANO-COMPUTADOR .....</b>	42
<b>3.1 ACESSIBILIDADE.....</b>	42
<b>3.2 USABILIDADE.....</b>	44
<b>3.3 INTERFACE NA WEB .....</b>	46
<b>4 TRABALHOS RELACIONADOS .....</b>	49
<b>4.1 AUXÍLIO NO ENSINO EM COMPILADORES: SOFTWARE SIMULADOR     COMO FERRAMENTA DE APOIO NA ÁREA DE COMPILADORES .....</b>	49
<b>4.2 C-GEN – AMBIENTE EDUCACIONAL PARA GERAÇÃO DE     COMPILADORES .....</b>	53

4.3 COMPILADOR EDUCATIVO VERTO: AMBIENTE PARA APRENDIZAGEM DE COMPILADORES .....	56
4.4 INTERPRETADOR DA LINGUAGEM D+ .....	59
4.5 SCC: UM COMPILADOR C COMO FERRAMENTA DE ENSINO DE COMPILADORES .....	62
5 METODOLOGIA .....	66
5.1 TECNOLOGIAS UTILIZADAS .....	66
5.1.1 ASP.NET Core .....	66
5.1.2 Visual Studio .....	67
5.1.3 Razor .....	68
5.1.4 JavaScript .....	69
5.1.5 C Sharp .....	69
5.1.6 Canvas .....	70
5.1.7 Diagrams.net .....	71
5.1.8 Azure .....	71
5.2 MODELO DA INTERFACE DA FERRAMENTA VL-D+ .....	72
5.3 FUNCIONALIDADES DA FERRAMENTA VL-D+ .....	77
5.3.1 Inserção de código .....	77
5.3.2 Aba de seleção .....	77
5.3.3 Análise léxica .....	78
5.3.4 Análise sintática .....	78
5.3.5 Tabela de símbolos .....	78
5.3.6 Registro de erros .....	79
5.4 INTERFACE DA APLICAÇÃO .....	79
5.4.1 Barra de Menus .....	79
5.4.2 Tela Inicial .....	80

5.4.3 Tela de documentação da Gramática D+ .....	84
5.4.4 Tela de documentação do Manual do Usuário .....	85
5.4.5 Tela Sobre .....	87
<b>5.5 DESENVOLVIMENTO DA APLICAÇÃO .....</b>	<b>88</b>
5.5.1 Caixa de texto .....	88
5.5.2 Análise léxica .....	92
5.5.3 Análise sintática.....	99
5.5.4 Tabela de símbolos .....	104
5.5.5 Registro de Erros .....	105
5.5.6 Exemplo prático.....	107
<b>5.6 CASOS DE TESTES.....</b>	<b>119</b>
5.6.1 Análise léxica .....	119
5.6.2 Análise sintática.....	121
<b>6 RESULTADOS E DISCUSSÕES .....</b>	<b>126</b>
6.1 ANÁLISE DOS RESULTADOS .....	126
<b>7 CONCLUSÃO.....</b>	<b>133</b>
<b>REFERÊNCIAS .....</b>	<b>134</b>
<b>APÊNDICE A – AUTÔMATOS DA LINGUAGEM D+ .....</b>	<b>136</b>
<b>APÊNDICE B – FORMULÁRIO DE QUESTÕES SOBRE A EXPERIÊNCIA DO USUÁRIO .....</b>	<b>142</b>
<b>APÊNDICE C – INTERFACE VL-D+: GRAMÁTICA D+ .....</b>	<b>149</b>
<b>APÊNDICE D – INTERFACE VL-D+: MANUAL DO USUÁRIO .....</b>	<b>153</b>

## 1 INTRODUÇÃO

Os programas de computadores são entendidos pela máquina em uma linguagem de baixo nível, normalmente associada à notação binária, com sequências de símbolos 0 e 1 tendo diferentes significados.

Com a sofisticação da produção de programas de computadores estes deixaram de ser escritos diretamente em linguagem de máquina e passaram a ser construídos com o uso de abstrações e outras formas de signos relacionados à mente humana. Isto passou a exigir o uso de ferramentas para criar esta tradução entre as diferentes linguagens, da máquina e do homem. Uma destas ferramentas é chamada de compilador.

De forma simplificada, o compilador é a ferramenta que lê um programa, escrito em uma linguagem fonte, e o traduz em um programa equivalente em outra linguagem, mantendo a semântica original. (Aho, 1995).

Na disciplina de Compiladores são estudados técnicas e métodos para construir um compilador e por esta razão a disciplina se apresenta como uma das que possuem o maior grau de dificuldade de compreensão pelos discentes dos cursos de computação, pois envolve técnicas de difícil visualização e dependências de conteúdos de outras disciplinas.

Por causa desta dependência, o trabalho do docente em Compiladores é deveras complicado, pois exige do discente amplo conhecimento de disciplinas do currículo básico do curso, o que muitas vezes não é obtido com êxito.

Diante dessa premissa, devem-se criar meios que facilitem o aprendizado sem o auxílio direto de um docente, para que o discente obtenha um nível de introspecção maior do conteúdo da disciplina, de forma ampla e prática.

A internet dispõe de diversas formas comunicacionais, como imagens, vídeos, textos e outros, abrindo portas para o compartilhamento de informações. Assim, no meio didático pode-se criar ferramentas que auxiliem a construção do conhecimento com o seu uso.

Sendo assim, o objetivo desse trabalho de pesquisa é desenvolver e disponibilizar na *web* uma ferramenta que possa diminuir as possíveis dificuldades, que os discentes têm na disciplina de Compiladores.

O trabalho é dirigido às pessoas que se interessam em conhecer o funcionamento de um compilador e como o compilador pode ser implementado em uma ferramenta *web*.

A estrutura do trabalho é dividida em seis partes: o capítulo 2 trata sobre teoria de compiladores; o capítulo 3 aborda a interface humano-computador; o capítulo 4 analisa os trabalhos relacionados e o capítulo 5 aborda a metodologia utilizada para o desenvolvimento da ferramenta VL-D+ (*Visual Learning D+*) e o capítulo 6 aborda os resultados e discussões. A conclusão se encontra no capítulo 7. Os anexos pós-textuais se encontram após as referências.

## 2 TEORIA DE COMPILEDORES

Este capítulo apresenta de forma resumida a teoria de compiladores, seu conceito, estrutura e os processos necessários para o seu funcionamento.

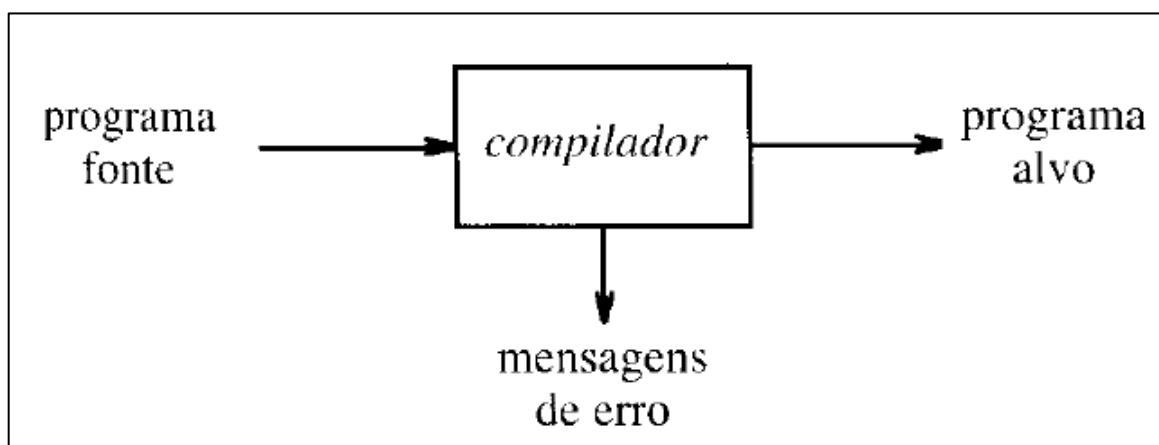
De acordo com Aho (1995), o compilador é um programa que processa comandos escritos numa linguagem fonte e os traduz para uma linguagem equivalente cuja notação possa ser executada no computador.

Para Santos (2018) um compilador é uma aplicação complexa que traduz uma descrição em determinada linguagem fonte para uma descrição equivalente em determinada linguagem de destino. Em outras palavras, um compilador traduz um programa em uma linguagem de programação para um programa executável em linguagem de máquina de uma arquitetura de destino.

Adicionalmente à tradução, os compiladores executam diversas funções auxiliares, que de acordo com Neto (2016) são atividades utilitárias para o usuário, encarregadas pela geração de listagens e a detecção, o diagnóstico e a emissão de mensagens e erros.

A figura 1 apresenta a definição de um compilador.

FIGURA 1 - Compilador



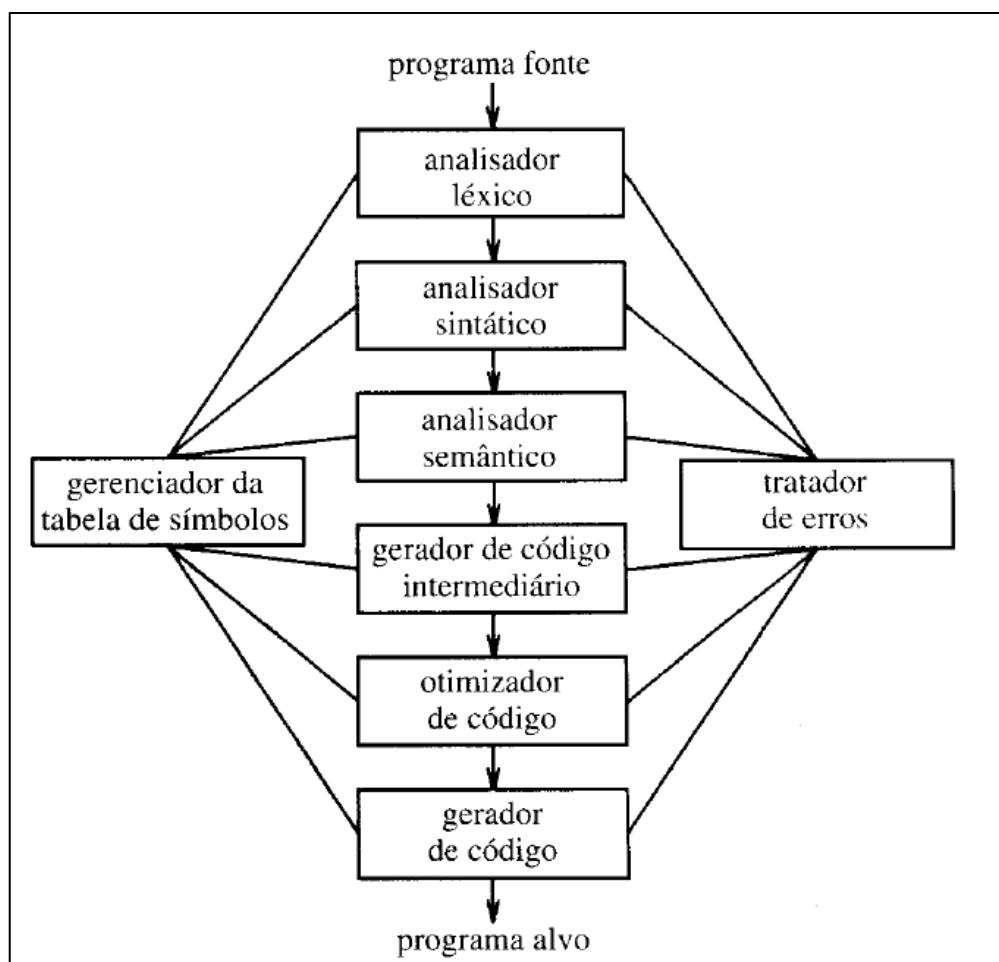
FONTE: AHO, SETHI, ULLMAN (1995)

Como pode ser visto na figura 1 o compilador recebe o programa fonte, que é o programa escrito pelo programador; em seguida é realizado o processo de tradução, que

caso seja bem-sucedido, retornará o programa alvo, e caso falhe, retornará as mensagens de erro ao programador.

Aho (1995) assume que o processo de tradução dos compiladores tipicamente é dividido nas fases: análise léxica, análise sintática, análise semântica, geração de código intermediário, otimização de código e geração de código. A ordem e a relação destas fases são apresentadas na figura 2.

FIGURA 2 - Fases de um compilador



FONTE: AHO, SETHI, ULLMAN (1995)

Este capítulo é dividido da seguinte forma: a seção 2.1 apresenta o conceito de gramática junto à especificação da gramática D+, que é a gramática utilizada pelo compilador desenvolvido neste trabalho; as seções 2.2 a 2.7 apresentam as fases do compilador e a tabela de símbolos e a seção 2.8 apresenta a diferenciação entre compiladores e interpretadores.

## 2.1 GRAMÁTICA

O passo inicial na construção de um compilador é a definição da Linguagem de Programação (LP) que será utilizada na escrita dos programas fontes. A LP utilizada deve ser especificada corretamente e de preferência sem ambiguidades para que seja possível programar um compilador correspondente.

A formalização de uma LP é feita através de regras de formação. O conjunto destas regras é denominado de gramática. De acordo com Neto (2016) são definidas por gramáticas as linguagens construídas a partir de um conjunto de leis ou regras de formação que permitem a produção de textos sintaticamente corretos.

Para os compiladores, a gramática tem um papel importante de definir a estrutura hierárquica das construções da linguagem de programação e permitir a separação de sequências de caracteres como um único *token*, chamados de lexemas.

### 2.1.1 Gramática D+

A linguagem de programação D+, criada pelo professor Diógenes C. Furlan para a disciplina de compiladores, é uma linguagem de cunho didático, cuja sintaxe mescla elementos das linguagens BASIC, Pascal e C.

FIGURA 3 - Declarações na Gramática D+

### Declarações

1. programa  $\rightarrow$  lista-decl
2. lista-decl  $\rightarrow$  lista-decl decl | decl
3. decl  $\rightarrow$  decl-const | decl-var | decl-proc | decl-func
4. decl-const  $\rightarrow$  CONST ID = literal ;
5. decl-var  $\rightarrow$  VAR espec-tipo lista-var ;
6. espec-tipo  $\rightarrow$  INT | FLOAT | CHAR | BOOL | STRING
7. decl-proc  $\rightarrow$  SUB espec-tipo ID ( params ) bloco END-SUB
8. decl-func  $\rightarrow$  FUNCTION espec-tipo ID ( params ) bloco END-FUNCTION
9. params  $\rightarrow$  lista-param |  $\epsilon$
10. lista-param  $\rightarrow$  lista-param , param | param
11. param  $\rightarrow$  VAR espec-tipo lista-var BY mode
12. mode  $\rightarrow$  VALUE | REF

FONTE: FURLAN (2019)

A figura 3 apresenta as regras gramaticais para as declarações de constantes, variáveis, procedimentos, funções e parâmetros da Linguagem D+.

Como pode ser visto nas regras 1 e 2, um programa é uma lista de declarações. A regra 3 apresenta as possíveis declarações que são especificadas nas regras 4, 5, 7 e 8. A regra 6 apresenta os possíveis tipos de variáveis que podem ser declarados. A regra 9 estabelece que os parâmetros podem ser uma lista de parâmetros individuais, podendo esta lista ser vazia. A regra 10 estabelece que uma lista de parâmetros pode ser formada por vários parâmetros separados por vírgula ou apenas um parâmetro. A regra 11 define como declarar um parâmetro. A regra 12 apresenta os possíveis modos de passagem dos parâmetros.

FIGURA 4 - Comandos na Gramática D+

**Comandos**

13. bloco  $\rightarrow$  lista-com
14. lista-com  $\rightarrow$  comando lista-com |  $\epsilon$
15. comando  $\rightarrow$  cham-proc | com-atrib | com-selecao | com-repeticao | com-desvio | com-leitura | com-escrita | decl-var | decl-const
16. com-atrib  $\rightarrow$  var = exp ;
17. com-selecao  $\rightarrow$  IF exp THEN bloco END-IF | IF exp THEN bloco ELSE bloco END-IF
18. com-repeticao  $\rightarrow$  WHILE exp DO bloco LOOP | DO bloco WHILE exp ; | REPEAT bloco UNTIL exp ; | FOR ID = exp-soma TO exp-soma DO bloco NEXT
19. com-desvio  $\rightarrow$  RETURN exp ; | BREAK ; | CONTINUE ;
20. com-leitura  $\rightarrow$  SCAN ( lista-var ) ; | SCANLN ( lista-var ) ;
21. com-escrita  $\rightarrow$  PRINT ( lista-exp ) ; | PRINTLN ( lista-exp ) ;
22. cham-proc  $\rightarrow$  ID ( args ) ;

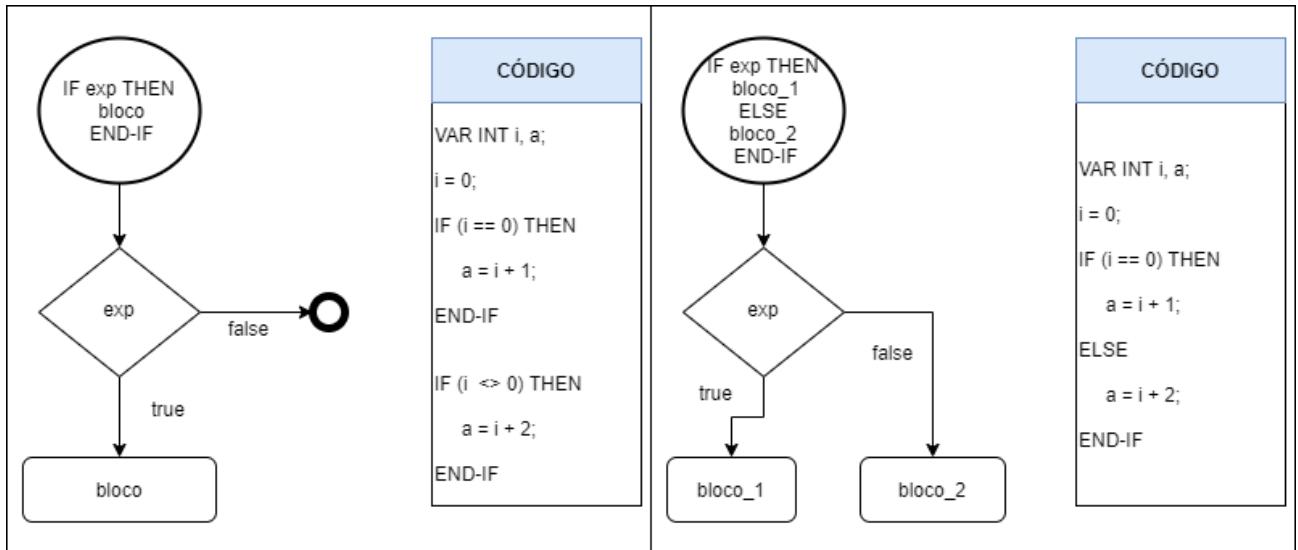
FONTE: FURLAN (2019)

A figura 4 apresenta as regras gramaticais dos comandos da gramática D+, sendo eles: chamada de procedimento, atribuição, seleção, repetição, desvio, leitura, escrita, declaração de variáveis e declaração de constantes.

Como pode ser visto nas regras 13 e 14, um bloco é uma lista de comandos. A regra 15 apresenta os possíveis comandos: chamada de procedimento, atribuição, seleção, repetição, desvio, leitura, escrita, declaração de variáveis e declaração de constantes. As regras 16 até a 22 apresentam como os comandos da regra 15 devem ser escritos.

A programação de um comando de seleção é possível de duas formas diferentes: somente com o caso verdadeiro, e com ambos os casos. A semântica destas formas é apresentada na figura 5.

FIGURA 5 - Comandos de seleção



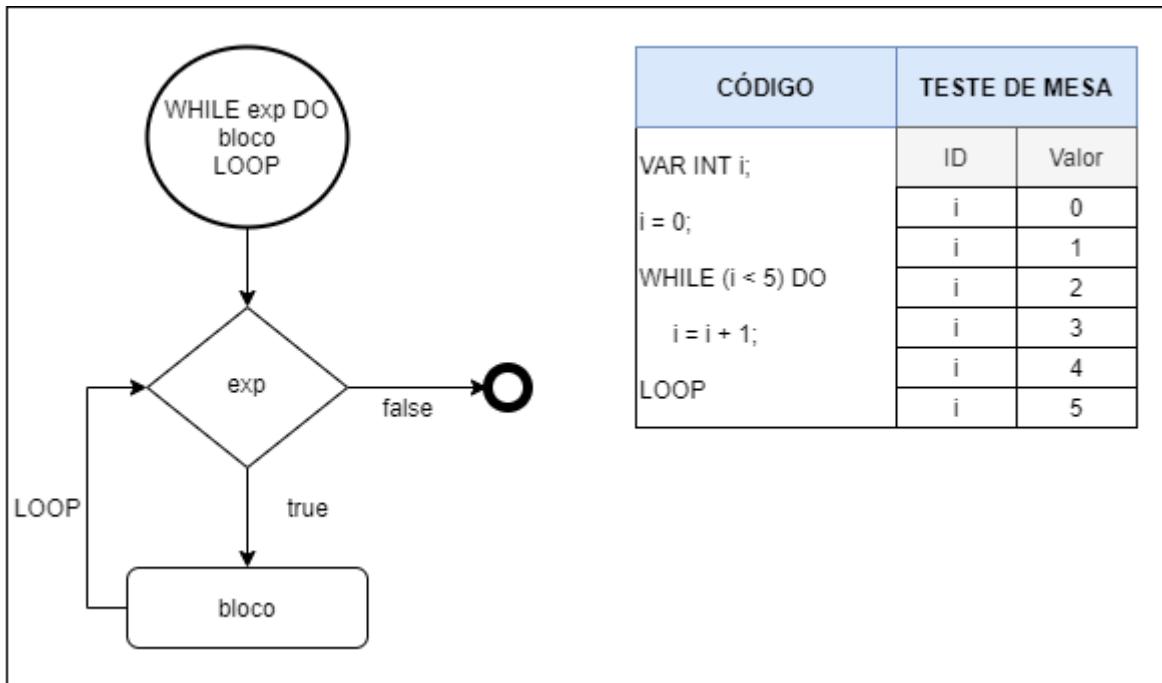
FONTE: a própria autora

A figura 5 apresenta um exemplo de como programar a mesma função utilizando os dois comandos de seleção da gramática D+. Ao lado esquerdo de cada quadrante é apresentado o diagrama do comando e ao lado direito é apresentado o código de exemplo.

Para a programação de um comando de repetição são disponibilizadas quatro maneiras diferentes: 1- enquanto uma condição for verdadeira faça alguma coisa; 2- faça alguma coisa enquanto uma condição for verdadeira; 3- repita alguma coisa até que uma condição seja verdadeira e 4- para um valor até uma expressão ser verdadeira faça alguma coisa.

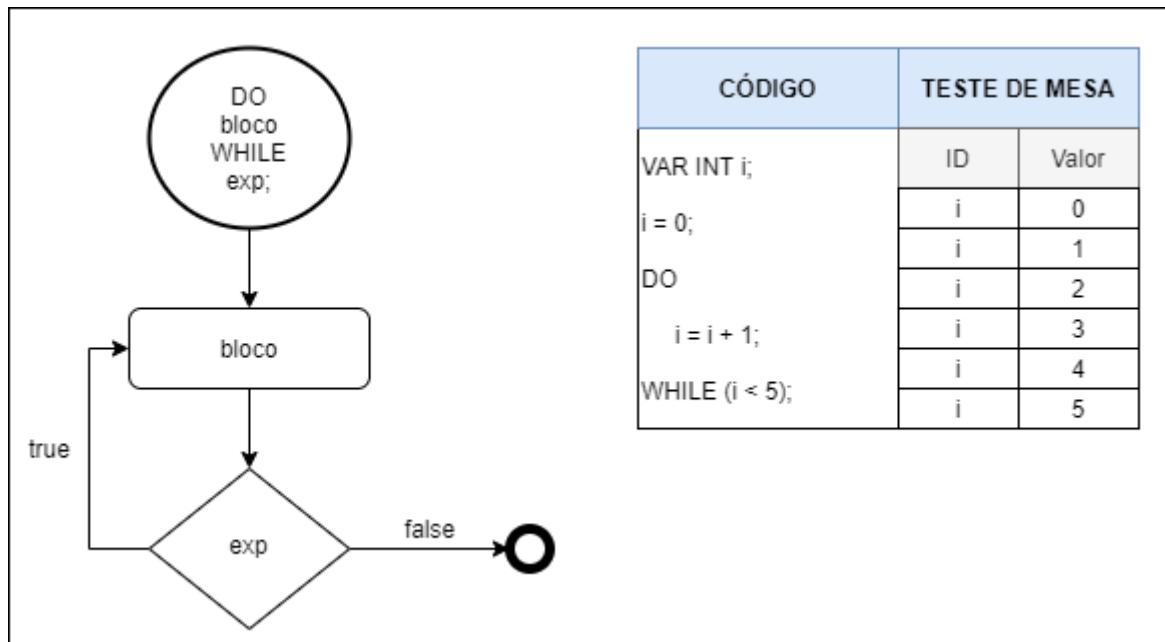
As figuras 6, 7, 8 e 9 exemplificam a semântica de cada uma maneira de realizar o comando de repetição. Todos os exemplos implementam uma função que deve incrementar a variável ‘i’ até que o valor dela seja igual a ‘5’. Ao lado esquerdo de cada figura é apresentado o diagrama do comando e ao lado direito é apresentado o código de exemplo.

FIGURA 6 - Comando de repetição: enquanto



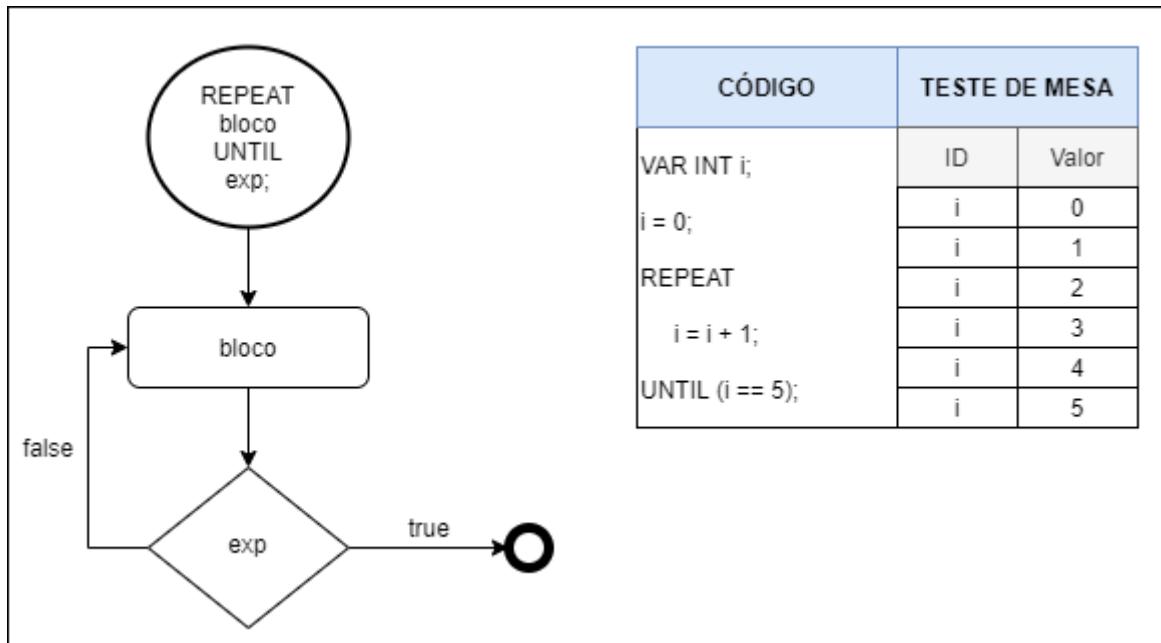
FONTE: a própria autora

FIGURA 7 - Comando de repetição: faça



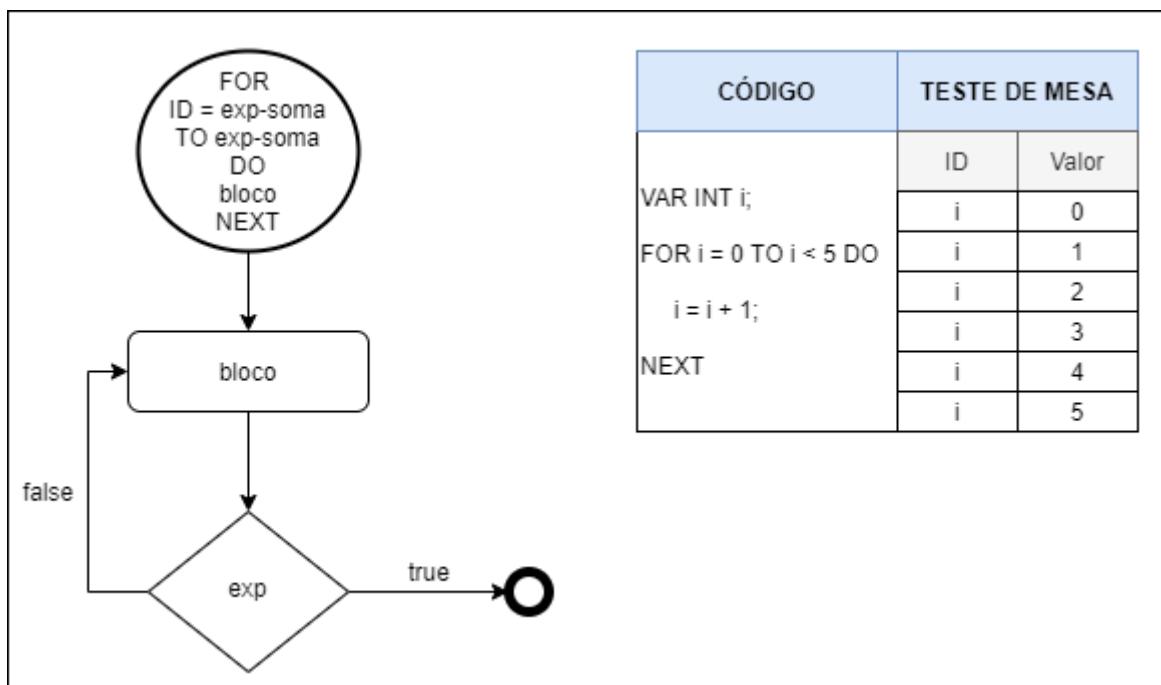
FONTE: a própria autora

FIGURA 8 - Comando de repetição: repita



FONTE: a própria autora

FIGURA 9 - Comando de repetição: para



FONTE: a própria autora

FIGURA 10 - Expressões na Gramática D+

**Expressões**

23. lista-exp → exp , lista-exp | exp
24. exp → exp-soma op-relac exp-soma | exp-soma
25. op-relac → <= | < | > | >= | == | <>
26. exp-soma → exp-mult op-soma exp-soma | exp-mult
27. op-soma → + | - | OR
28. exp-mult → exp-mult op-mult exp-simples | exp-simples
29. op-mult → \* | / | DIV | MOD | AND
30. exp-simples → ( exp ) | var | cham-func | literal | op-unario exp
31. literal → NUMINT | NUMREAL | CARACTERE | STRING | valor-verdade
32. valor-verdade → TRUE | FALSE
33. cham-func → ID ( args )
34. args → lista-exp | ε
35. var → ID | ID [ exp-soma ]
36. lista-var → var , lista-var | var
37. op-unario → + | - | NOT

FONTE: FURLAN (2019)

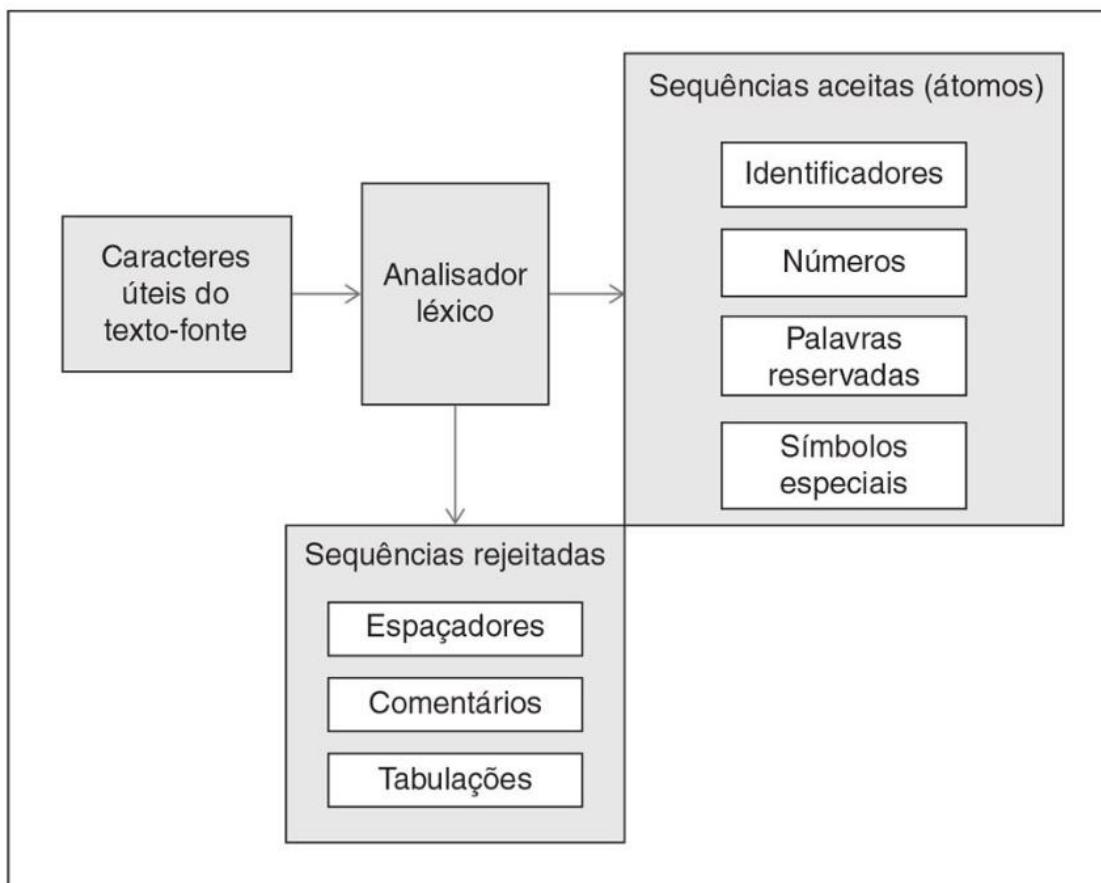
A figura 10 apresenta as regras gramaticais das expressões. As expressões podem conter operações de soma, operações de multiplicação, operações relacionais, operações unárias e chamadas de funções.

A regra 23 estabelece que uma lista de expressões é composta por uma ou mais expressões separadas por vírgula. As regras 24, 26, 28 e 30 definem as expressões. As regras 25, 27, 29 e 37 definem os operadores aceitos pela linguagem. A regra 31 apresenta os possíveis valores para um literal. A regra 32 apresenta os possíveis valores verdade. A regra 33 apresenta a chamada de função que deve ser composta por um identificador seguido por argumentos entre parênteses. A regra 34 apresenta que um argumento é composto por uma lista de expressões ou pode ser vazio. A regra 35 apresenta que uma variável por ser um identificador ou um identificador seguido por uma expressão de soma entre colchetes. A regra 36 apresenta que uma lista de variáveis pode ser composta por uma variável seguida por uma lista de variáveis separadas por vírgula ou por uma variável.

## 2.2 ANÁLISE LÉXICA

A análise léxica é a primeira fase do compilador e de acordo com Neto (2016) é a responsável pelo agrupamento dos caracteres válidos do programa fonte e pela classificação desses caracteres de acordo com a sua constituição.

FIGURA 11 - Análise léxica



FONTE: NETO (2016)

A figura 11 apresenta o processo do analisador léxico, que recebe como entrada o texto fonte e o separa em sequências aceitas e em sequências rejeitadas.

As sequências aceitas são identificadas de acordo com as regras definidas pela gramática da linguagem utilizada pelo programa fonte. As sequências rejeitadas são separadas pela função que descarta as sequências de caracteres que não contribuem na análise do programa fonte, como: tabulações, mudanças de linha, espaços, caracteres de controle, comentários e outros.

Além das funções principais posteriormente citadas, de acordo com Neto (2016) a análise léxica também conduz ações auxiliares, como: conversões numéricas, identificação de palavras reservadas e a criação e manutenção das tabelas de símbolos, definidas na seção 2.6, que são largamente utilizadas no processo da compilação.

### 2.2.1 Reconhecimento de *tokens*

No processo de compilação os termos “*token*”, “padrão” e “lexema” são comumente utilizados. A análise léxica é a fase responsável pela identificação de cada sequência de caracteres como um desses termos.

Aho (1995) descreve o padrão como um conjunto de cadeias de entrada para as quais o *token* é produzido como saída e o lexema é o conjunto de caracteres que é reconhecido pelo padrão de um *token*. A figura 9 exemplifica o uso de cada um dos termos citados.

QUADRO 1 - Exemplo de *tokens*, lexema e padrão

Token	Lexemas Exemplo	Descrição Informal do Padrão
PR_CONST	pi	const travessão opcional seguido por letra seguida por letras, número e/ou travessões
PR_IF	if	if
OP_MAIOR	>	>
IDENTIFICADOR	contador	travessão opcional seguido por letra seguida por letras, número e/ou travessões
NUM_INT	120	qualquer constante numérica inteira
CARACTERE	‘c’	qualquer símbolo entre aspas simples, exceto aspa simples

FONTE: AHO, SETHI, ULLMAN (1995) adaptado

No quadro 1 são apresentados diversos exemplos de *tokens* captados pela análise léxica de um programa escrito na linguagem D+ e seus respectivos lexemas e descrições informais, tais como: os *tokens* identificadores (*id*), que são os valores que começam com um travessão ou uma letra seguida por caracteres alfanuméricos ou travessões; os *tokens* numéricos (*num*), que são qualquer constante numérica inteira; entre outros.

A captura dos lexemas é feita caractere por caractere, de acordo com a sequência lida, e, da gramática da linguagem do programa fonte. Quando um lexema é completado, é possível inferir qual a sua classificação ou gerar o seu respectivo *token*.

## 2.2.2 Exemplo de programa e sua análise léxica

Para exemplificação do funcionamento da análise léxica, foram criados dois cenários de teste na linguagem D+.

O primeiro cenário de teste é um código funcional que efetua o cálculo de fatorial de um número, esse código é apresentado na figura 12.

FIGURA 12 - Código em D+ para cálculo de fatorial

```

1  /* Código na linguagem D+ para efetuar cálculo de fatorial de um número */
2  MAIN ()
3
4  VAR INT fatorial, numero;
5  numero = 3;
6  fatorial = 1;
7  WHILE numero > 1 DO
8      fatorial = fatorial * numero;
9      numero = numero - 1;
10     LOOP
11
12 END

```

FONTE: a própria autora

A análise léxica do código da figura 12 identifica vários *tokens* e seus respectivos lexemas, de acordo com a definição da gramática da linguagem D+, que podem ser vistos no quadro 2.

QUADRO 2 - *Tokens* e lexemas obtidos pela análise léxica do código de fatorial

Token	Lexema
PR_MAIN	MAIN
SIN_PAR_A	(
SIN_PAR_F	)
PR_VAR	VAR
PR_INT	INT
IDENTIFICADOR	fatorial
SIN_V	,
IDENTIFICADOR	numero
SIN_PV	;
IDENTIFICADOR	numero
OP_ATRIBUI	=
NUM_INT	3
SIN_PV	;
IDENTIFICADOR	fatorial
OP_ATRIBUI	=
NUM_INT	1
SIN_PV	;
PR_WHILE	WHILE
IDENTIFICADOR	numero
OP_MAIOR	>
NUM_INT	1
PR_DO	DO
IDENTIFICADOR	fatorial
OP_ATRIBUI	=
IDENTIFICADOR	fatorial
OP_MULTI	*
IDENTIFICADOR	numero
SIN_PV	;
IDENTIFICADOR	numero
OP_ATRIBUI	=
IDENTIFICADOR	numero
OP_SUBTRAI	-
NUM_INT	1
SIN_PV	;
PR_LOOP	LOOP
PR_END	END

FONTE: a própria autora

O quadro 2 apresenta todos os *tokens* e seus respectivos lexemas obtidos através do código de cálculo de fatorial pela análise léxica. Porém, pode-se observar que espaços, quebra de linhas, tabulações e comentários são ignorados.

O segundo cenário de teste é um código que contém um erro léxico, esse código é apresentado na figura 13.

FIGURA 13 - Código em D+ com erro léxico

```

1  MAIN ()
2
3      VAR CHAR a;
4      a = 'erro';
5
6  END

```

FONTE: a própria autora

A análise léxica do código apresentado na figura 13 deve disparar um erro na linha 4 do código, pois o tipo caractere (CHAR) de acordo com a gramática da linguagem D+ suporta apenas um símbolo ou nenhum símbolo entre aspas simples.

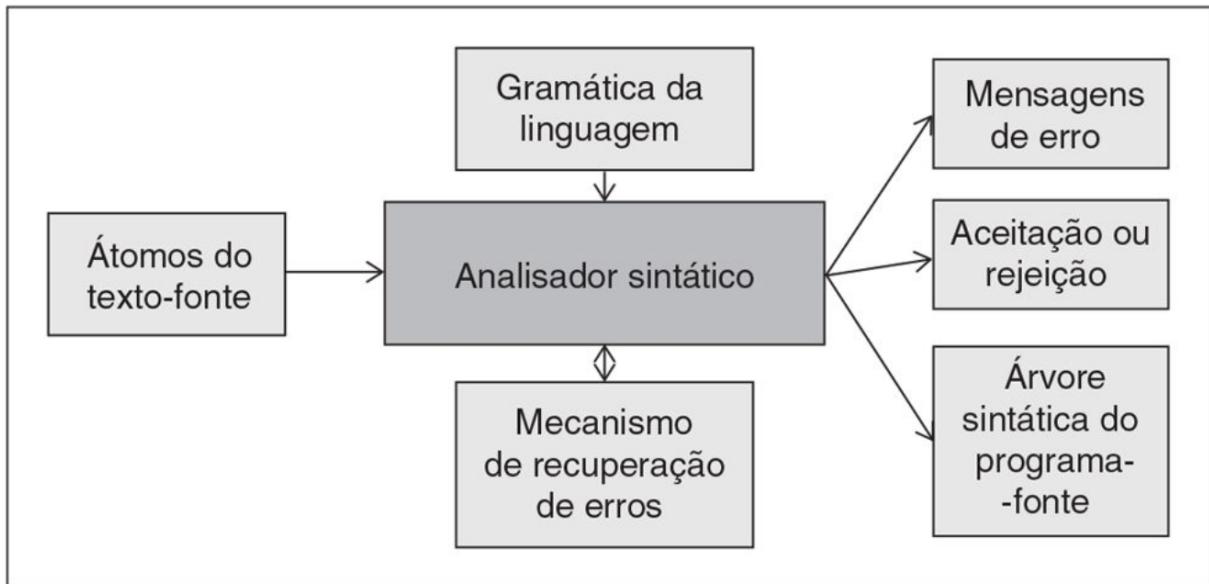
### 2.3 ANÁLISE SINTÁTICA

Após o programa fonte ser devidamente processado pelo analisador léxico, o analisador sintático obtém a sequência de *tokens* gerada na análise léxica e assegura que a sequência seja válida de acordo com a especificação sintática da linguagem fonte.

Para Neto (2016) a função do analisador sintático é construir uma árvore sintática a partir da sequência de *tokens* dada pela análise léxica, associando os ramos que a compõem aos elementos gramaticais correspondentes da linguagem fonte.

Durante o processo de verificação dos *tokens* também é feita a detecção de erros sintáticos encontrados no texto fonte. Os erros devem ser relatados de forma inteligível ao programador e o analisador sintático deve ser capaz de recuperar os erros e continuar o processo de verificação.

FIGURA 14 - Análise sintática



FONTE: NETO (2016)

A figura 14 apresenta o processo do analisador sintático. A entrada do analisador sintático é a sequência de *tokens* do texto fonte e a gramática da linguagem; junto ao analisador sintático existe um mecanismo de recuperação de erros; e a saída do analisador são as mensagens de erro, a aceitação ou rejeição de cada *token* e a árvore sintática do programa fonte.

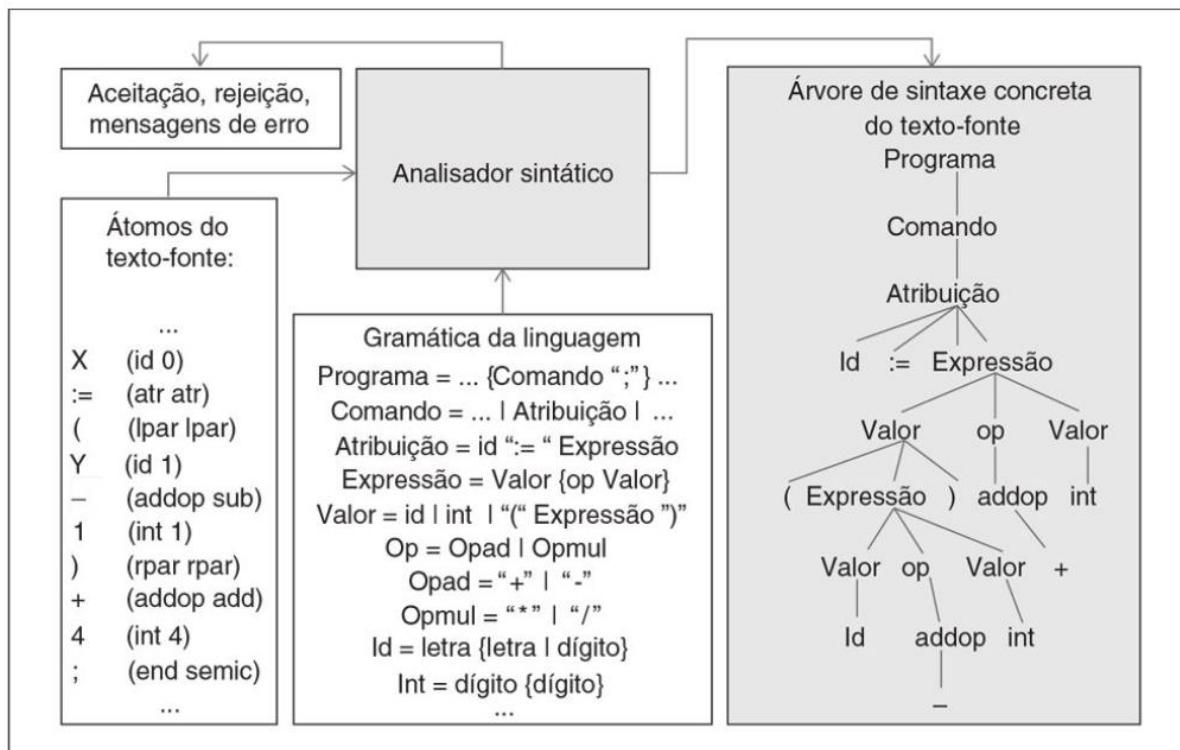
### 2.3.1 Árvore sintática

A árvore sintática descreve a estrutura do texto fonte de acordo com a gramática da linguagem correspondente ao programa fonte.

A construção da árvore sintática pode ser realizada de diversas formas conforme a necessidade do compilador. Usualmente os compiladores constroem uma árvore sintática que exibe apenas as informações essenciais à tradução do programa fonte.

De acordo com Neto (2016) uma árvore sintática montada com base no texto fonte e na gramática da linguagem é denominada *árvore sintática concreta* e ela reflete fielmente a estrutura e a exata composição do texto fonte do programa.

FIGURA 15 - Árvore sintática concreta

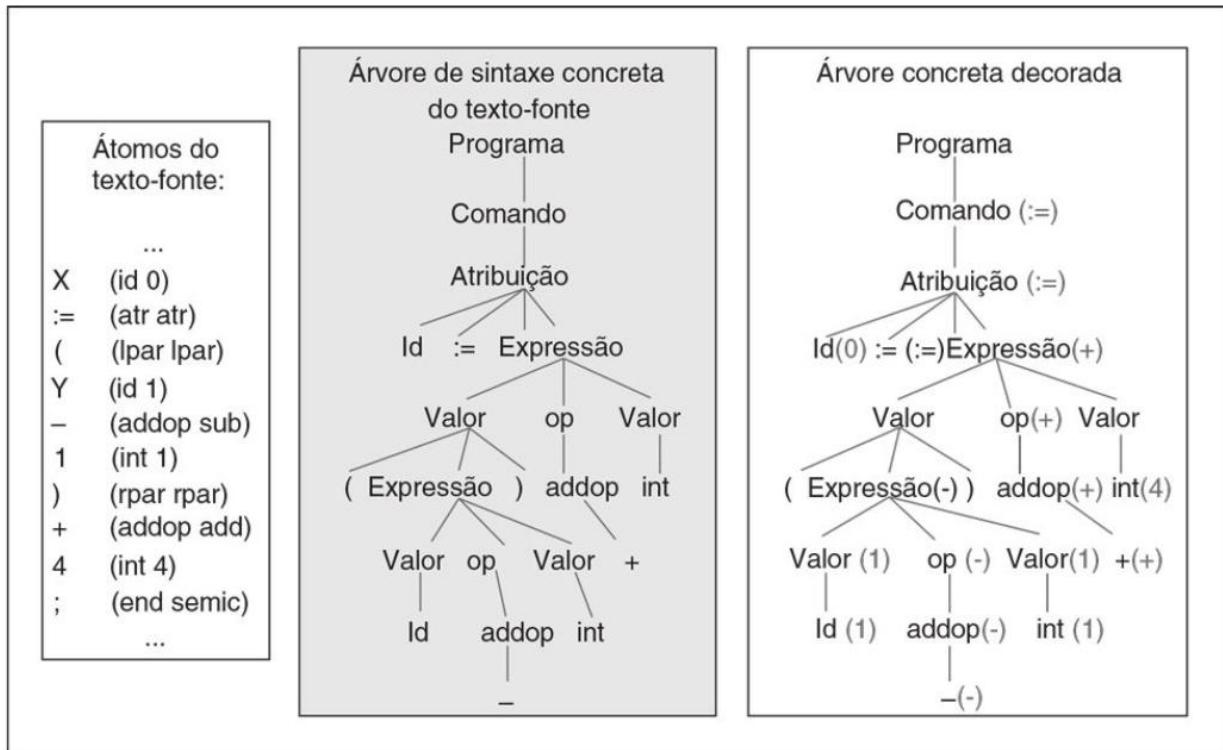


FONTE: NETO (2016)

A figura 15 apresenta um exemplo de processo do analisador sintático. A entrada do analisador sintático é a sequência de *tokens* do texto fonte: identificador ‘X’, atribuição ‘:=’, abre parênteses ‘(’, identificador ‘Y’, subtração ‘-’, número inteiro ‘1’, fecha parênteses ‘)’, adição ‘+’, número inteiro ‘4’, ponto e vírgula ‘;’ e a gramática da linguagem; e a saída do analisador são as mensagens de erro, a aceitação ou rejeição de cada *token* e a árvore sintática concreta gerada através da gramática e do texto fonte recebidos na entrada do analisador.

Quando a árvore sintática concreta é acrescida de informação de cunho semântico ela traz consigo toda a informação disponível sobre o programa que representa e passa a ser chamada de árvore sintática concreta decorada.

FIGURA 16 - Árvore concreta decorada

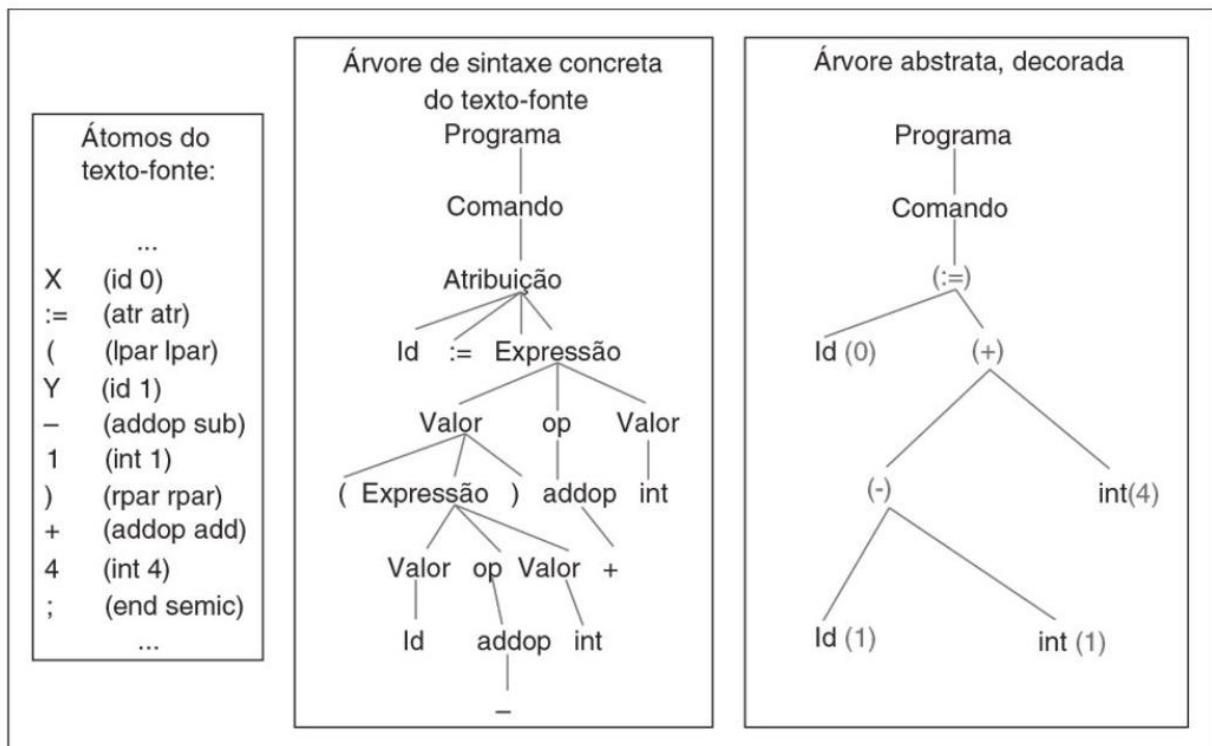


FONTE: NETO (2016)

A figura 16 apresenta um exemplo de construção de uma *árvore sintática concreta* e de uma *árvore sintática concreta decorada*. As duas árvores são construídas a partir da sequência de *tokens* do texto fonte e pode-se observar que a diferença entre elas é que a árvore decorada traz consigo o lexema de cada *token*.

Apesar de a *árvore sintática concreta* apresentar informação abundante sobre o programa fonte ela não é tão útil ao compilador quanto faz parecer. A *árvore sintática concreta* traz consigo inúmeros elementos sintáticos irrelevantes para o compilador, cujo propósito é aumentar o conforto do programador. Pensando em reduzir o espaço de memória e o tempo de processamento a ser executado pelo compilador para produzir a árvore sintática, existe a *árvore sintática abstrata*, uma versão de árvore sintática que elimina todo elemento sintático irrelevante e traz consigo apenas a informação essencial para a construção do código objeto.

FIGURA 17 - Árvore abstrata



FONTE: NETO (2016)

A figura 17 apresenta um exemplo de construção de uma *árvore sintática concreta* e de uma *árvore sintática abstrata decorada*. As duas árvores são construídas a partir da sequência de *tokens* do texto fonte e pode-se observar que a maior diferença entre elas é que a árvore abstrata por conter apenas os nós correspondentes à expressão, é substancialmente menor e traz consigo poucos dados.

### 2.3.2 Exemplo prático

Para exemplificar a análise sintática foi criado um código na linguagem D+ com diversos erros sintáticos, com o intuito de apontar qual deveria ser o resultado do compilador quanto a eles.

FIGURA 18 - Código em D+ com erros sintáticos

```

1  /* Código na linguagem D+ para efetuar cálculo de fatorial de um número */
2  MAIN
3
4      VAR INT factorial, numero
5      numero = 3
6      factorial = 1
7      WHILE int > 1
8          factorial = factorial * numero
9          numero = 1 -
10     LOOP
11
12 END

```

FONTE: a própria autora

A análise sintática do código apresentado na figura 18 deve disparar os seguintes erros: erro de declaração de *main* na linha 2 do código, por falta dos parenteses; erro nas linhas 4, 5, 6, 8 e 9 por falta de ponto e vírgula; erro de comando *while* na linha 7 por falta do comando *do*; e erro na linha 9 pois a operação está incompleta.

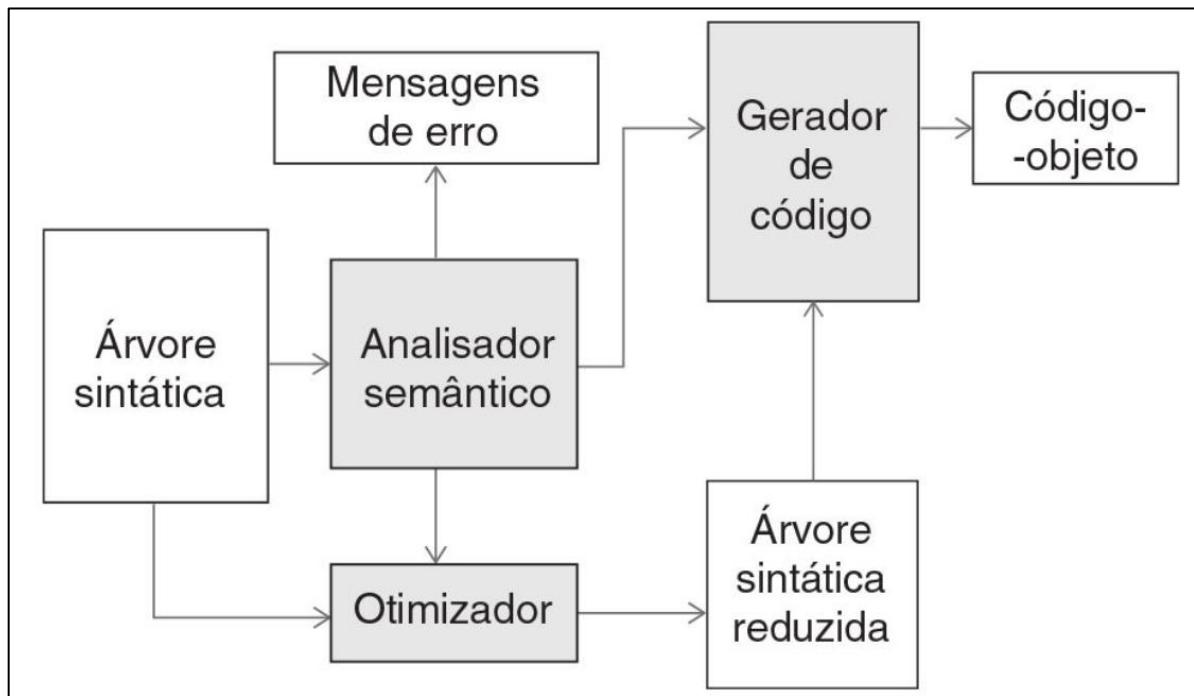
## 2.4 ANÁLISE SEMÂNTICA

Na terceira fase do compilador é realizada a análise da árvore sintática abstrata gerada na fase anterior, em busca de possíveis ineficiências que possam ser removidas sem comprometer o programa.

Essa atividade, de acordo com Neto (2016) é denominada análise semântica e nela podem ser levantadas informações que permitam ao compilador decidir sobre as alterações que podem ou devem ser feitas na árvore.

Para Aho (1995) a análise semântica verifica os erros semânticos no programa fonte e captura as informações de tipo utilizando a estrutura hierárquica determinada pelo analisador sintático, a fim de identificar os operadores e operandos das expressões e enunciados para a fase de geração de código.

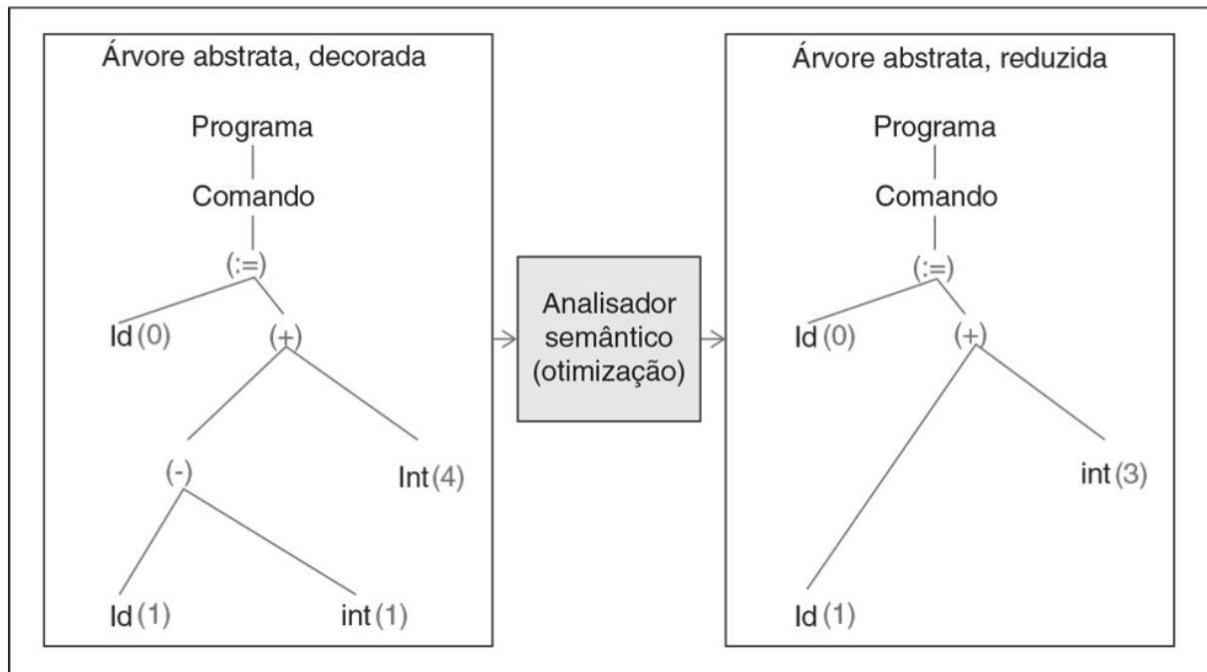
FIGURA 19 - Análise semântica



FONTE: NETO (2016)

A figura 19 apresenta o processo do analisador semântico. O analisador semântico pode realizar dois caminhos diferentes para a geração do código-objeto: o melhor caminho é o compilador enviar a árvore sintática e a análise semântica realizada para um otimizador que remove redundâncias e ineficiências do programa e gera a *árvore sintática abstrata reduzida* e a partir dessa nova árvore enviá-la para o gerador de código, dessa forma, o código-objeto terá menos funções e portanto será mais eficiente; o outro caminho é o resultado do analisador semântico ir direto para o gerador de código e resultar no código-objeto, não otimizado. Caso haja erros nessas etapas, a saída do analisador serão as mensagens de erro.

FIGURA 20 - Árvore sintática abstrata reduzida



FONTE: NETO (2016)

A figura 20 apresenta o analisador semântico recebendo de entrada uma árvore abstrata decorada e retornando uma árvore abstrata reduzida equivalente.

## 2.5 TABELA DE SÍMBOLOS

A criação e manutenção de tabelas de símbolos usualmente é executada pelo analisador léxico. Porém, de acordo com Neto (2016) alguns compiladores constroem os analisadores léxicos e sintáticos limitados estritamente às suas operações mínimas, com o intuito de poder reaproveitá-los. Portanto, as atividades que não podem ser reutilizadas são transferidas para outra parte do compilador.

Neto (2016) define a tabela de símbolos como uma coleção dos identificadores encontrados pelo compilador ao longo da inspeção do texto fonte, relacionados aos atributos necessários para a geração de código.

Para Aho (1995) o compilador utiliza a tabela de símbolos para controlar as informações de escopo e para manter atualizadas as informações acerca de um novo

nome ou de uma nova informação. Santos (2018) relaciona a classe do identificador, o tipo de dados e o tipo de retorno como informações essenciais a serem memorizadas pela tabela de símbolos.

FIGURA 21 - Tabela de símbolos

**TABELA DE SÍMBOLOS**

	nomes						atributos	
1	A	4					Inteiro, 125	
2	W	B	L	D			Real, 126	
3	B	Y	E				Vetor inteiro, 128	
4	S	4	9	2	Z	X	Real, 241	
5								
6								
7								
8								
9								
...							...	
FF								

FONTE: NETO (2016)

A figura 21 apresenta um exemplo de uma tabela de símbolos. Ao lado esquerdo da tabela são colocados os nomes dos identificadores encontrados no texto fonte e ao lado direito são colocados seus respectivos atributos.

Como uma tarefa de controle do compilador, a tabela de símbolos é encarregada de consultar símbolos para o compilador a qualquer momento. De acordo com Neto (2016), caso o símbolo conste na tabela é feita uma consulta em seus atributos com a finalidade de coletar dados; e se o símbolo não consta na tabela, geralmente é uma situação de erro.

Sendo assim, a criação e manutenção da tabela de símbolos são operações fundamentais para o correto funcionamento do compilador.

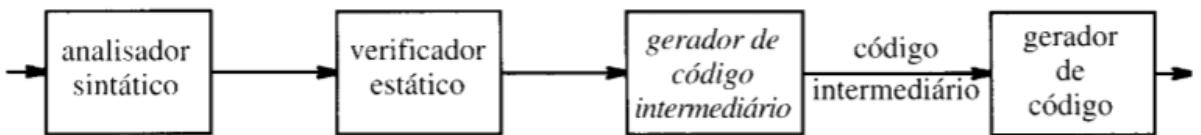
## 2.6 GERAÇÃO DE CÓDIGO INTERMEDIÁRIO

Em alguns modelos de compiladores a penúltima fase do compilador é a geração de código intermediário.

De acordo com Aho (1995) no modelo de análise e síntese de um compilador, o programa fonte é traduzido numa representação intermediária.

Santos (2018) afirma que o código intermediário consiste em um processo de linearização de código que gera uma sequência linear de instruções intermediárias a partir de uma árvore sintática ou qualquer outro tipo de representação intermediária do programa a ser compilado.

FIGURA 22 - Código intermediário



FONTE: AHO, SETHI, ULLMAN (1995)

A figura 22 apresenta em qual local no fluxo de processos do compilador é construído e utilizado o código intermediário.

De acordo com Santos (2018) o principal propósito de um compilador é gerar um código para algum processador e para isso é necessário conhecer os tipos de processadores, ver todos os tipos de representações intermediárias possíveis e definir uma arquitetura que permita gerar, de forma simples e direta, o código não otimizado.

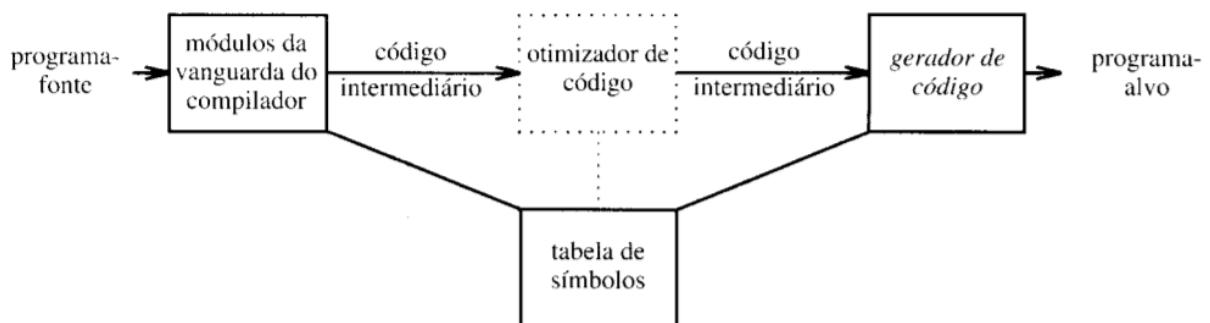
Vide este problema de otimização, para a criação de um código otimizado os compiladores utilizam o código intermediário que pode ser manipulado de modo a gerar instruções mais eficientes para um processador específico.

Aho (1995) afirma que uma vantagem da criação de código intermediário é que outros computadores podem reutilizá-lo, precisando executar o processo de compilação a partir da geração de código.

## 2.7 GERAÇÃO DE CÓDIGO

A fase final do compilador é a tradução propriamente dita do programa fonte para a forma do código objeto, conhecida como geração de código. A entrada do gerador de código é a representação intermediária do programa fonte e a saída é o programa alvo, como indicado na figura 23.

FIGURA 23 – Gerador de código



FONTE: AHO, SETHI, ULLMAN (1995)

A figura 23 apresenta o fluxo completo do compilador dando enfoque à etapa de geração de código. Inicialmente o compilador recebe o programa fonte e o processa até a geração do código intermediário; em seguida o código intermediário pode passar por um otimizador de código; a última etapa é o gerador de código que recebe o código intermediário e a tabela de símbolos como entrada e retorna o programa alvo.

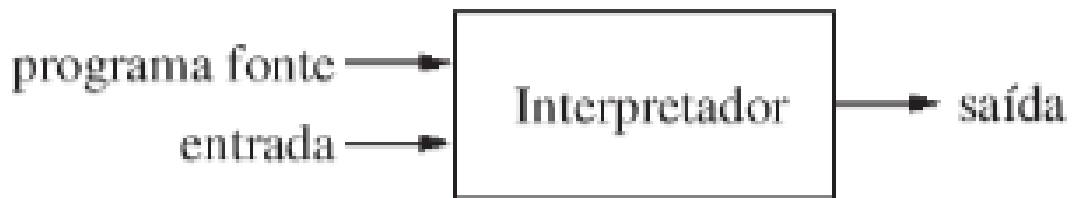
Neto (2016) aponta duas maneiras pelas quais a geração de código pode ser realizada: na primeira maneira, o processo de geração de código é realizado com uma interpretação literal do programa fonte gerando um código objeto de baixa qualidade; e na segunda maneira o código é formatado para que possa ser tratado por uma seção de otimização disponível no compilador.

## 2.8 INTERPRETADOR X COMPILADOR

O interpretador é outro tipo de processador de linguagem. De acordo com Aho (2008), diferentemente do compilador o interpretador executa diretamente as operações

especificadas no programa fonte sobre as entradas fornecidas pelo usuário, como apresentado na figura 24.

FIGURA 24 - Interpretador



FONTE: AHO, LAM, SETHI, ULLMAN (2008)

Como pode ser visto na figura 24 o interpretador recebe o programa fonte, que é o programa escrito pelo programador e a entrada fornecida pelo usuário; em seguida são executadas as operações do programa fonte sobre a entrada e é retornado o programa objeto como saída.

Aho (2008) argumenta que um compilador normalmente produz um programa objeto muito mais rápido no mapeamento de entradas para saídas que um interpretador. Porém, Aho (2008) aponta que um interpretador, ao executar o programa fonte instrução por instrução, é capaz de oferecer um diagnóstico de erro melhor que um compilador.

### 3 INTERAÇÃO HUMANO-COMPUTADOR

Este capítulo aborda alguns princípios da interação humano-computador que, quando considerados no desenvolvimento de um sistema interativo, aumentam a qualidade do sistema.

De acordo com Benyon (2011) o *design* de sistemas interativos é focado na entrega de um sistema de alta qualidade, onde os produtos e serviços oferecidos pelo sistema combinam com as pessoas e com seus modos de vida.

Nas seções 3.1, 3.2, 3.3 são apresentados os conceitos de acessibilidade, usabilidade e aceitabilidade respectivamente; na seção 3.4 são apresentados importantes considerações para o desenvolvimento de uma interface na *web*.

#### 3.1 ACESSIBILIDADE

De acordo com Sobral (2019) a acessibilidade em uma interface tem a função de diminuir as barreiras de interação entre o usuário e o sistema estabelecendo qualidade de interação para a realização de tarefas.

Devido ao aumento crescente de usuários de computadores e tecnologias é cada vez mais comum a necessidade de softwares acessíveis. Para esse requisito Benyon (2011) aponta duas abordagens do *design* que visam a acessibilidade: o *design* universal e o *design* inclusivo.

O *design* universal é um conjunto de conceitos de acessibilidade que um *designer* precisa ter ao desenvolver um produto para torná-lo acessível a qualquer usuário. Os princípios do *design* universal são baseados na abordagem filosófica do *The Center for Universal Design* – numa tradução livre “O Centro do *Design Universal*” apresentados no quadro 3.

### QUADRO 3 - Princípios do Design Universal

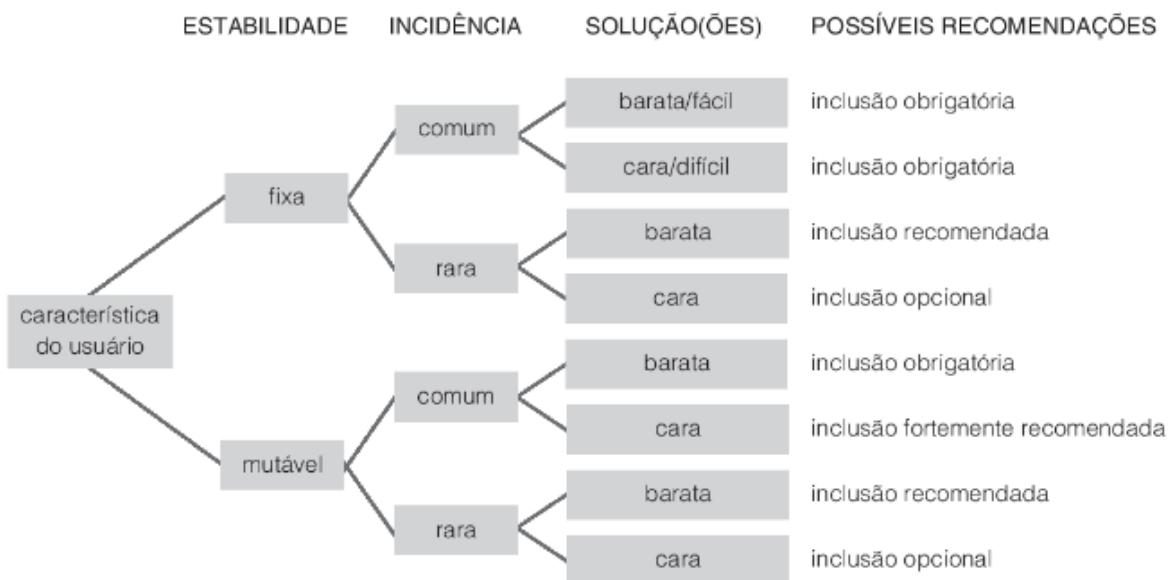
Princípios	Descrição
Uso equitativo	O <i>design</i> não prejudica ou estigmatiza nenhum usuário
Flexibilidade no uso	O <i>design</i> acomoda uma variedade de preferências e habilidades individuais
Uso intuitivo	O uso do <i>design</i> é fácil de entender independentemente do conhecimento do usuário
Informação perceptível	O <i>design</i> transmite a informação corretamente ao usuário, independentemente do ambiente ou das habilidades sensoriais do usuário
Tolerância ao erro	O <i>design</i> minimiza consequências de ações acidentais
Baixo esforço físico	O <i>design</i> não necessita de muito esforço físico
Tamanho e espaço para acesso e uso	O espaço fornecido para aproximação, uso, alcance e manipulação deve ser apropriado independentemente do tamanho do corpo do usuário, postura ou mobilidade

FONTE: The Center for Universal Design adaptado

De acordo com Benyon (2011) o *design* inclusivo é baseado nas seguintes premissas: as diferenças nas habilidades são uma característica comum do ser humano visto que as mudanças físicas e intelectuais ocorrem ao longo da vida; se o *design* é eficaz para pessoas com deficiências será eficaz para todos; a autoestima, identidade e bem-estar das pessoas são profundamente afetados pela capacidade delas de funcionar no ambiente físico, com conforto, independência e controle.

Para Benyon (2011) o *design* inclusivo é uma abordagem que frequentemente por razões técnicas ou financeiras é inatingível. A figura 25 apresenta uma árvore de decisão que demonstra visualmente as condições e probabilidades para alcançar a inclusividade dado um determinado cenário.

FIGURA 25 - Árvore de decisão para análise de inclusividade



FONTE: BENYON (2011)

A figura 25 apresenta uma árvore de decisões para analisar a inclusão necessária no *design* de acordo com as características do usuário com que o software irá trabalhar. Por exemplo, o início da árvore é a característica do usuário, se a característica tiver estabilidade fixa, a incidência de usuários necessitados de inclusão for rara e a solução de software for cara a inclusão de software é opcional.

### 3.2 USABILIDADE

A usabilidade é a principal busca da interação humano-computador. De acordo com Benyon (2011) um sistema com alto grau de usabilidade é eficiente na questão de esforço que o usuário precisa exercer, contém funções e conteúdo de informações adequadas e organizadas, é fácil de aprender e de lembrar, é seguro e entrega alto grau de utilidade para o seu propósito.

#### QUADRO 4 - Critérios de Usabilidade

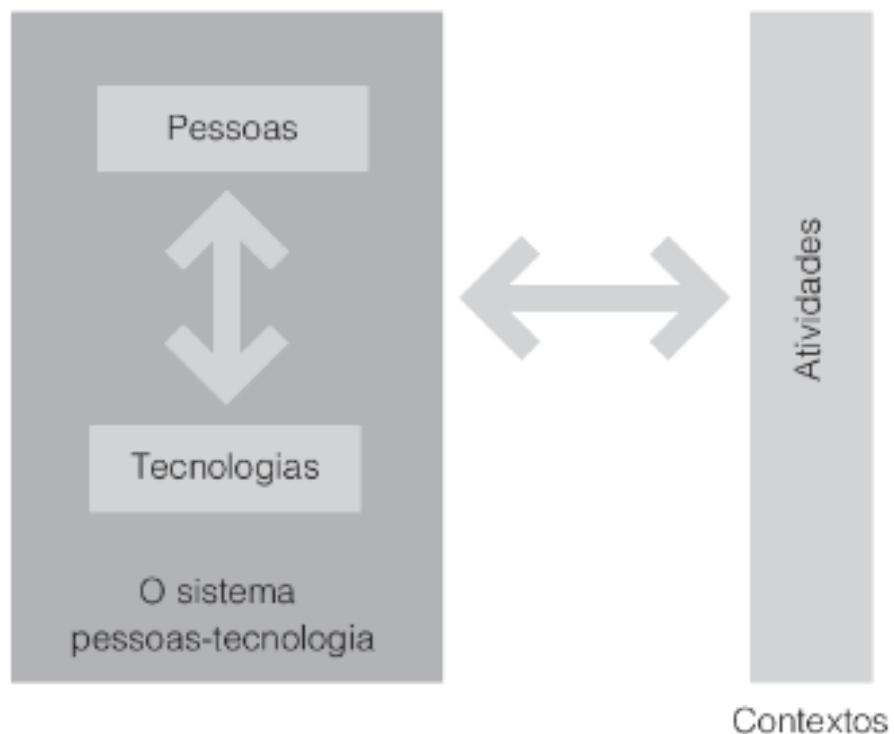
Critério	Descrição
Fácil de aprender	O usuário deve aprender o sistema sem dificuldades
Fácil de lembrar	O sistema deve ser facilmente memorizado de tal forma que um usuário esporádico consiga utilizá-lo após certo período sem a necessidade de reaprendê-lo
Eficiência	Depois que um usuário souber utilizar o sistema a interface deve ser eficiente
Poucos erros	O sistema deve ter baixa taxa de erros durante a interação
Satisfação subjetiva	O sistema deve ser agradável ao uso

FONTE: SOBRAL (2019) adaptado

O quadro 4 apresenta os critérios de um sistema com alto grau de usabilidade. De acordo com Sobral (2019) um sistema seguindo esses critérios fará com que o usuário precise utilizar menos carga de trabalho cognitivo, gerando facilidade de aprendizagem, diminuição dos erros, eficiência na interação e satisfação.

Para Benyon (2011) a usabilidade tem o objetivo de equilibrar os quatro principais fatores do *design* de sistemas interativos centrados no ser humano: pessoas, atividades que as pessoas desejam realizar, contextos nos quais a interação acontece e as tecnologias.

FIGURA 26 - Usabilidade



FONTE: BENYON (2011)

A figura 26 ilustra uma característica importante da interação humano computador. Do lado esquerdo é apresentada a interação entre pessoas e as tecnologias num sistema que as pessoas utilizam; do outro lado é apresentada a relação das pessoas e as atividades que estão sendo realizadas e os contextos dessas atividades.

### 3.3 INTERFACE NA WEB

De acordo com Garret (2000) originalmente a *web* era um espaço de troca de informações hipertextuais. Porém, com o crescente desenvolvimento de tecnologias a *web* começou a ser utilizada como uma interface de software remoto.

O desenvolvimento de uma interface interativa de acordo com (2019) foca em encontrar uma solução que promova a interação da interface com o usuário entre o hardware e o software. Para alcançar uma boa interação é necessário saber quais são as funções do software, quem é o usuário que irá utilizar o software e que tipo de tarefa ele

deseja realizar. Além disso, o desenvolvimento de uma interface interativa deve levar em conta a utilização de usuários de níveis iniciante, intermediário e avançado.

Garret (2000) define considerações-chave que fazem parte do desenvolvimento da experiência do usuário na *web*.

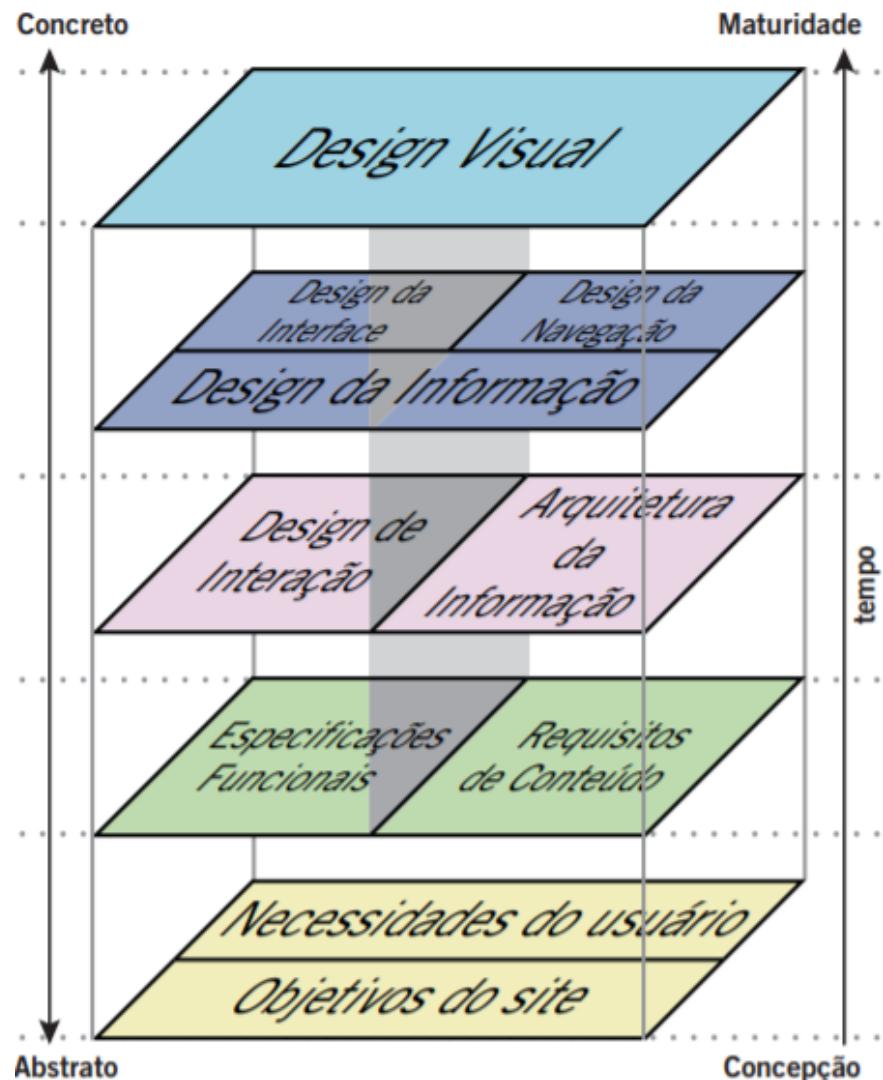
As considerações-chave ambíguas tanto para a *web* como interface de software quanto para a *web* como um sistema de hipertexto são: o *design* visual, responsável pelo tratamento dos elementos gráficos do site; as necessidades do usuário que são identificadas através de pesquisas; e os objetivos do site.

As demais considerações-chave para a *web* como interface de software de acordo com Garret (2000) são: o *design* da interface que trabalha com os elementos da interface para facilitar a interação do usuário com as funcionalidades; o *design* da informação que se preocupa em facilitar a compreensão da informação; o *design* de interação que define como o usuário interage com as funcionalidades do sistema; as especificações funcionais que descrevem detalhadamente as funcionalidades que o site deve incluir para satisfazer as necessidades do usuário.

Para a *web* como um sistema de hipertexto as demais considerações-chave de acordo com Garret (2000) são: o *design* de navegação que facilita a movimentação do usuário em meio a arquitetura da informação; o *design* de informação que se encarrega de apresentar a informação de forma que facilite a compreensão do usuário; a arquitetura da informação que estrutura a informação para facilitar o acesso intuitivo do conteúdo; os requisitos de conteúdo que definem quais são os conteúdos necessários ao site em relação às necessidades do usuário.

A figura 27 apresenta o esquema dos elementos da experiência do usuário definido por Garret (2000).

FIGURA 27 - Elementos da experiência do usuário



FONTE: GARRET (2000)

## 4 TRABALHOS RELACIONADOS

Esse capítulo apresenta em suas seções a síntese dos trabalhos relacionados ao tema do presente projeto. O quadro 5 apresenta um comparativo entre os trabalhos relacionados.

QUADRO 5 - Comparaçāo dos Trabalhos Relacionados

Título	Possui saída visual?	Ilustra a Análise Léxica?	Ilustra a Análise Sintática?	Ilustra a Tabela de Símbolos?	Ilustra código intermediário?	Ilustra os erros de compilação?	É off-line ou on-line?	Ilustração estática ou dinâmica?
Auxílio no ensino em compiladores: software simulador como ferramenta de apoio na área de compiladores	SIM	NÃO	NÃO	SIM	SIM	SIM	OFF	ESTÁTICA
C-gen – ambiente educacional para geraçāo de compiladores	SIM	SIM	SIM	NÃO	NÃO	NÃO	OFF	ESTÁTICA
Compilador educativo verto: ambiente para aprendizagem de compiladores	SIM	SIM	SIM	SIM	SIM	NÃO	OFF	ESTÁTICA
Interpretador da linguagem D+	SIM	SIM	SIM	SIM	NÃO	SIM	OFF	ESTÁTICA
SCC: um compilador C como ferramenta de ensino de compiladores	SIM	NÃO	SIM	SIM	NÃO	NÃO	OFF	ESTÁTICA

FONTE: a própria autora

### 4.1 AUXÍLIO NO ENSINO EM COMPILADORES: SOFTWARE SIMULADOR COMO FERRAMENTA DE APOIO NA ÁREA DE COMPILADORES

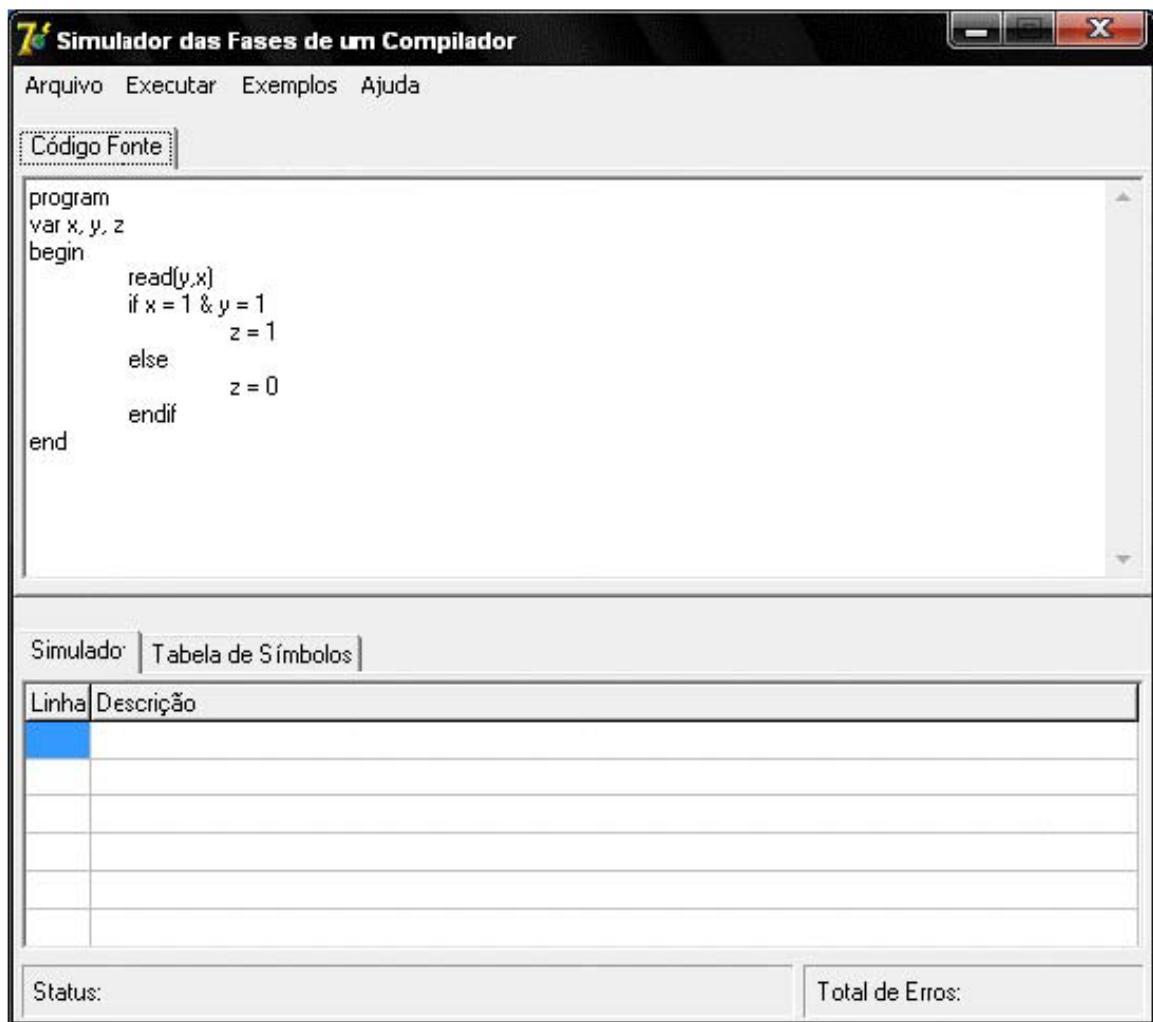
Os pesquisadores Costa, Silva e Britto declararam em seu projeto a necessidade de elaborar uma técnica que pudesse facilitar a compreensão dos discentes, mais especificamente da disciplina de Compiladores, dos cursos que envolvem computação. Com essa premissa, eles projetaram e desenvolveram um software capaz de simular claramente o funcionamento interno das fases de um compilador.

A ferramenta *CompilerSim* foi criada para executar um processo de análise de código baseado na linguagem de programação Pascal, e, posteriormente, para converter

o código para a linguagem de baixo nível Assembly x86. O desenvolvimento da ferramenta se baseou na série de artigos “*Let's Build a Compiler*”, escritos pelo cientista Ph.D. Jack W. Crenshaw na década de 80. A partir dos artigos foi extraído o código original e o mesmo passou por modificações e adaptações com o intuito de que as etapas do processo de compilação fossem demonstradas através de uma interface gráfica.

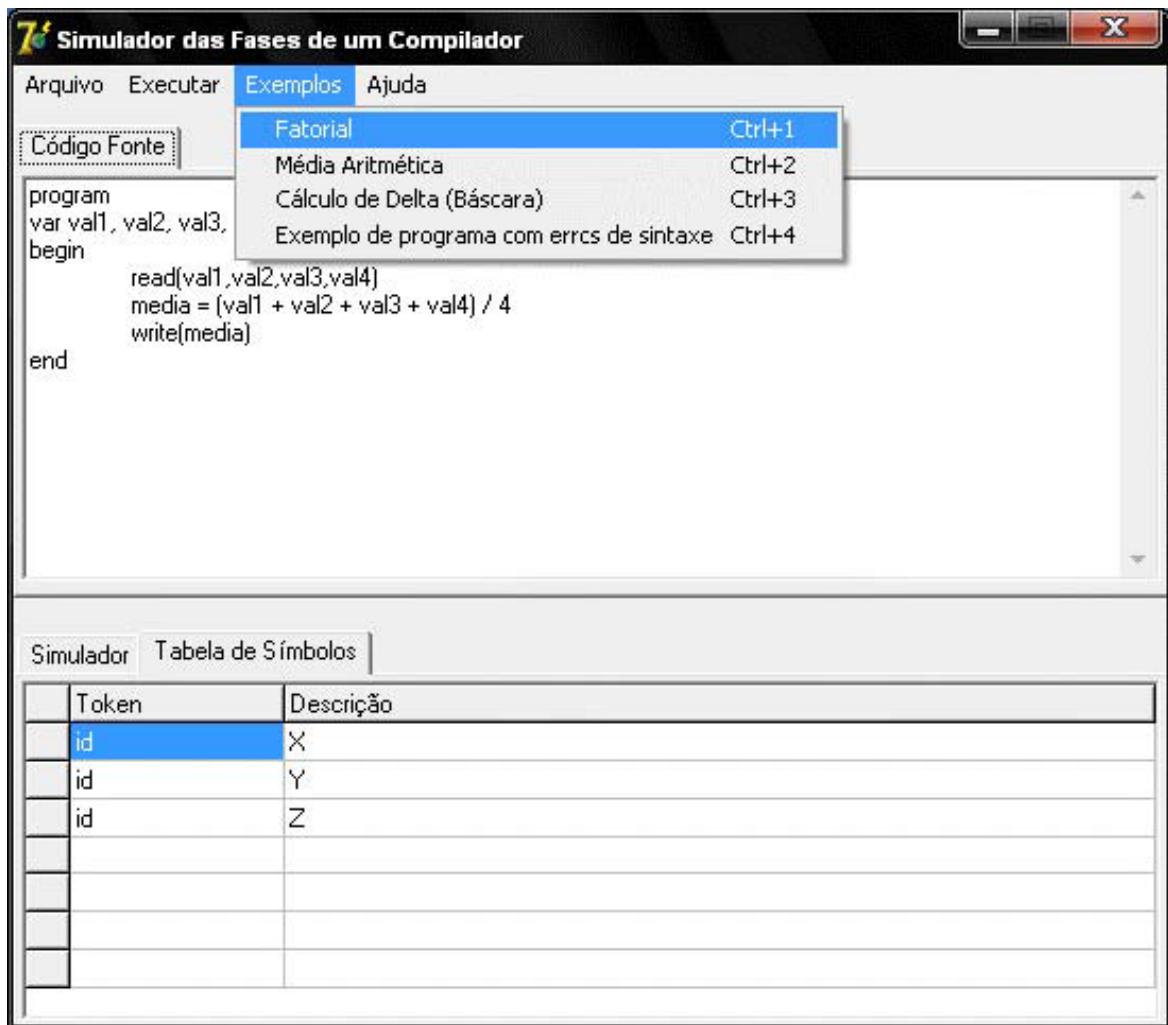
A interface gráfica do software apresenta algumas funcionalidades para o usuário. A primeira etapa da aplicação consiste em inserir o código fonte no programa, podendo ser feito manualmente pelo usuário (figura 28) ou carregado a partir da aba “Exemplos” (figura 29).

FIGURA 28 - Interface padrão do simulador *CompilerSim*



FONTE: COSTA, SILVA, BRITTO (2008)

FIGURA 29 - Exemplos prontos para execução



FONTE: COSTA, SILVA, BRITTO (2008)

Após a execução do código, a ferramenta apresenta os seguintes resultados: os *tokens* gerados a partir da análise léxica do código, a interpretação e geração de código intermediário na linguagem Assembly, e, quando cabível, os erros gerados na compilação. A visualização desses itens pode ser observada nas figuras 30 e 31 respectivamente.

FIGURA 30 - Análise da linguagem e resultados abordados

**Código Fonte:**

```
program
var x, y, z
begin
  read(x,y)
  if x = 1 & y = 1
    z = 1
  else
    z = 0
  endif
end
```

**Tabela de Símbolos:**

Token	Descrição
id	X
id	Y
id	Z

**Código Assembly:**

```
WARMST EQU $A01E
X: DC 0
Y: DC 0
Z: DC 0
MAIN:
BSR READ
LEA Y(PC)A0
MOVE D0,A0
BSR READ
LEA X(PC)A0
MOVE D0,A0
MOVE X(PC)D0
MOVE D0,(SP)
MOVE #1,D0
CMP (SP)+,D0
SEQ D0
EXT D0
MOVE D0,(SP)
MOVE Y(PC),D0
MOVE D0,(SP)
MOVE #1,D0
CMP (SP)+,D0
SEQ D0
EXT D0
AND (SP)+,D0
TST D0
BEQ L0
MOVE #1,D0
LEA Z(PC)A0
```

FONTE: COSTA, SILVA, BRITTO (2008)

FIGURA 31 - Execução de um código e a indicação de erros

**Código Fonte:**

```
program;
var x
begin
  read(x,y)
  if x > y
  else
  endif
end
```

**Tabela de Símbolos:**

Linha	Descrição
2	Erro: 'BEGIN' Esperado
2	Erro: Identificador Desconhecido VAR
3	Erro: '=' Esperado
4	Erro: Identificador Desconhecido BEGIN
4	Erro: Identificador Desconhecido X
4	Erro: Identificador Desconhecido Y

**Status:** Não foi possível compilar, o programa contém erros

**Total de Erros:** 10

FONTE: COSTA, SILVA, BRITTO (2008)

Os autores do projeto *CompilerSim* concluíram através da utilização do software simulador em salas de aula que a ferramenta pode influenciar positivamente e significativamente o processo de ensino e aprendizagem na disciplina de compiladores.

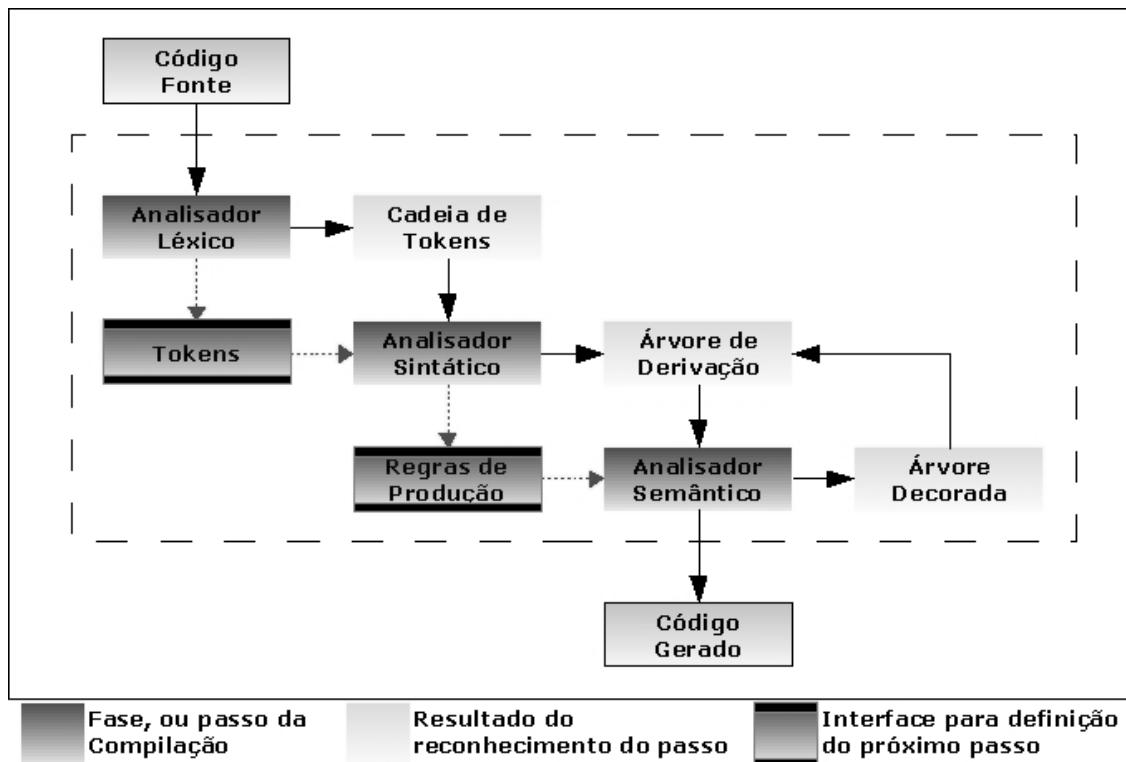
#### 4.2 C-GEN – AMBIENTE EDUCACIONAL PARA GERAÇÃO DE COMPILADORES

O *C-gen* é uma ferramenta, criada pelos pesquisadores Backes e Dahmer, com o intuito de sanar a carência de ferramentas com interface gráfica possíveis de serem utilizadas para orientar os discentes da disciplina de Compiladores, exibindo o funcionamento de todo o processo de compilação.

A ferramenta desenvolvida permite que o discente a utilize enquanto aprende, possibilitando que a teoria de compiladores seja visualizada, assim facilitando a compreensão do conteúdo. Para isso, a ferramenta atende alguns requisitos, dentre eles: possibilitar a definição e disponibilizar uma interface adequada para os analisadores léxicos, sintáticos e semânticos; permitir o acompanhamento do processo de reconhecimento das fases da compilação, passo a passo.

A arquitetura do sistema é organizada para que cada passo do processo de compilação se comunique com o passo posterior através de uma representação intermediária, permitindo ser executado independentemente, conforme apresentado na figura 32.

FIGURA 32 - Arquitetura do protótipo

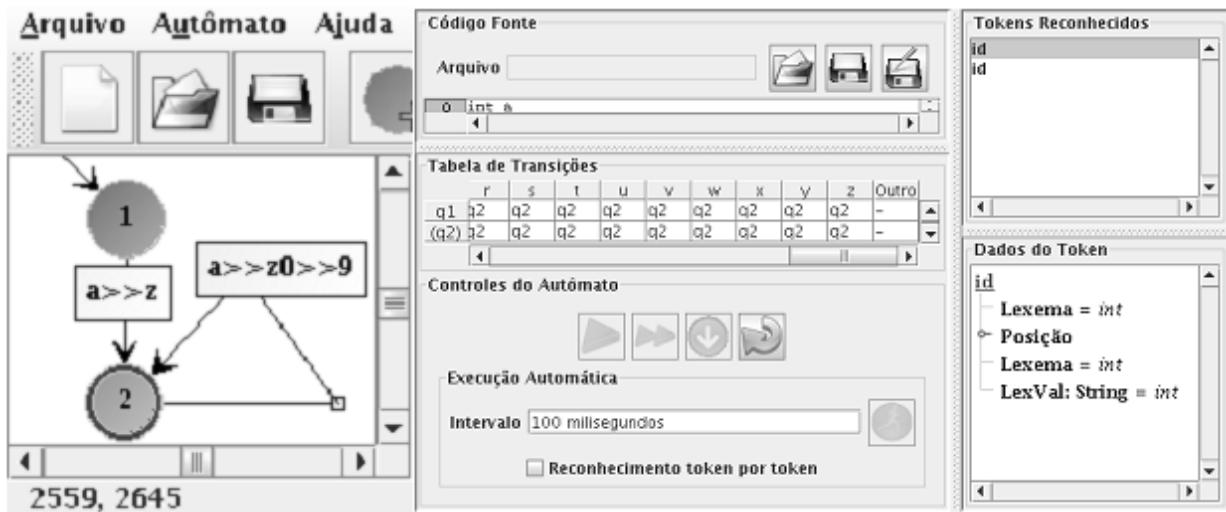


FONTE: BACKES, DAHMER (2006)

As fases da compilação são implementadas com *plug-ins* que reconhecem as suas respectivas entradas e produzem as saídas correspondentes.

O *plug-in* responsável pelo analisador léxico foi implementado de forma que seja produzido um analisador através da definição de autômatos. A definição é informada pelo usuário, que é responsável pela criação de estados e transições do autômato e os caracteres que o mesmo deve reconhecer em cada transição. Após a edição do autômato, quando iniciado o processo de reconhecimento, a ferramenta converte os dados em um autômato finito determinístico e cria a tabela de transições, conforme ilustra a figura 33.

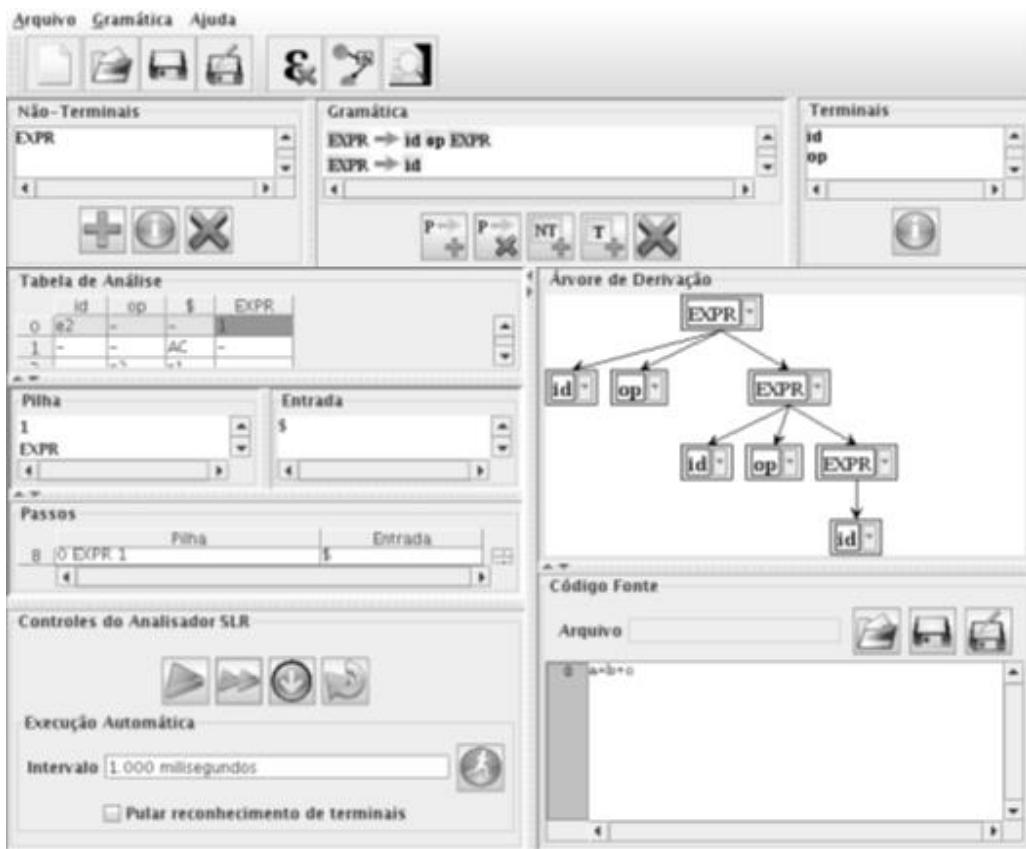
FIGURA 33 - Interface de edição de um autômato e reconhecimento



FONTE: BACKES, DAHMER (2006)

A análise sintática é feita através da especificação da gramática, considerando como terminais os *tokens* gerados pelo analisador léxico. A gramática é gerada pelo usuário no editor, que deve criar os símbolos não-terminais. O propósito do usuário gerar sua própria gramática é justamente para que ele não precise aprender uma nova notação para utilizar a ferramenta. A interface para essa confecção e de resultado da análise é ilustrada pela figura 34.

FIGURA 34 - Interface de edição de gramáticas e reconhecimento



FONTE: BACKES, DAHMER (2006)

Em suma, além do programa auxiliar os discentes no processo de aprendizagem da disciplina de Compiladores, a estrutura do *C-gen* permite expansões de funcionalidades via *plug-ins* ou através da contribuição voluntária de desenvolvedores, visto que o programa é um software livre.

#### 4.3 COMPILADOR EDUCATIVO VERTO: AMBIENTE PARA APRENDIZAGEM DE COMPILADORES

No Centro Universitário Feevale foi notada a necessidade de desenvolver uma ferramenta de apoio pedagógico, para a disciplina de Compiladores após a mesma enfrentar desmotivação e dificuldades de compreensão dos discentes.

Os compiladores são, de forma geral, tradutores de alguma linguagem para um programa. De acordo com os pesquisadores Schneider, Passerino e Oliveira, a

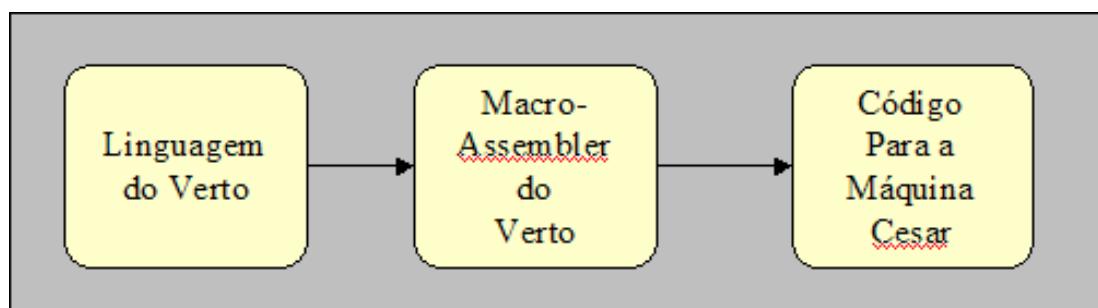
aprendizagem de compiladores consiste em absorver um processo composto por etapas que exigem estratégias e métodos específicos para aprender o processo de compilação, sendo ele: análise léxica, sintática e semântica, tabela de símbolos e geração do código objeto.

A pesquisa ressalta também que, a aprendizagem é a construção de uma representação pessoal de um conteúdo que é objeto de aprendizagem (Coll, 1998 *apud* SCHNEIDER, 2005). Para atingir esse nível de compreensão os estudantes devem ser capacitados a compreender as fases do compilador, sobretudo as fases finais de geração de código intermediário e objeto.

Visto a dificuldade dos discentes e a disciplina ser lecionada em apenas um semestre, foi desenvolvido o Compilador Educativo *Verto*, que faz parte de um ambiente de aprendizagem para compiladores. O compilador é composto por diversas ferramentas computacionais, um docente mediador, os discentes que o utilizam, as sequências didáticas que são planejadas no plano de ensino e aprendizagem apoiada pela metodologia de ensino embasada pelo docente.

O processo de compilação do *Verto* inicia-se com a geração do código intermediário em um formato *macro-assembler*. O formato dispõe de instruções simplificadas para facilitar a compreensão das estruturas compiladas. Após essa etapa, gera-se o arquivo final que contém as instruções no formato *César*, uma linguagem objeto criada com fins didáticos, permitindo que o estudante execute e analise o algoritmo. Na figura 35 pode-se observar o esquema de tradução do compilador *Verto*.

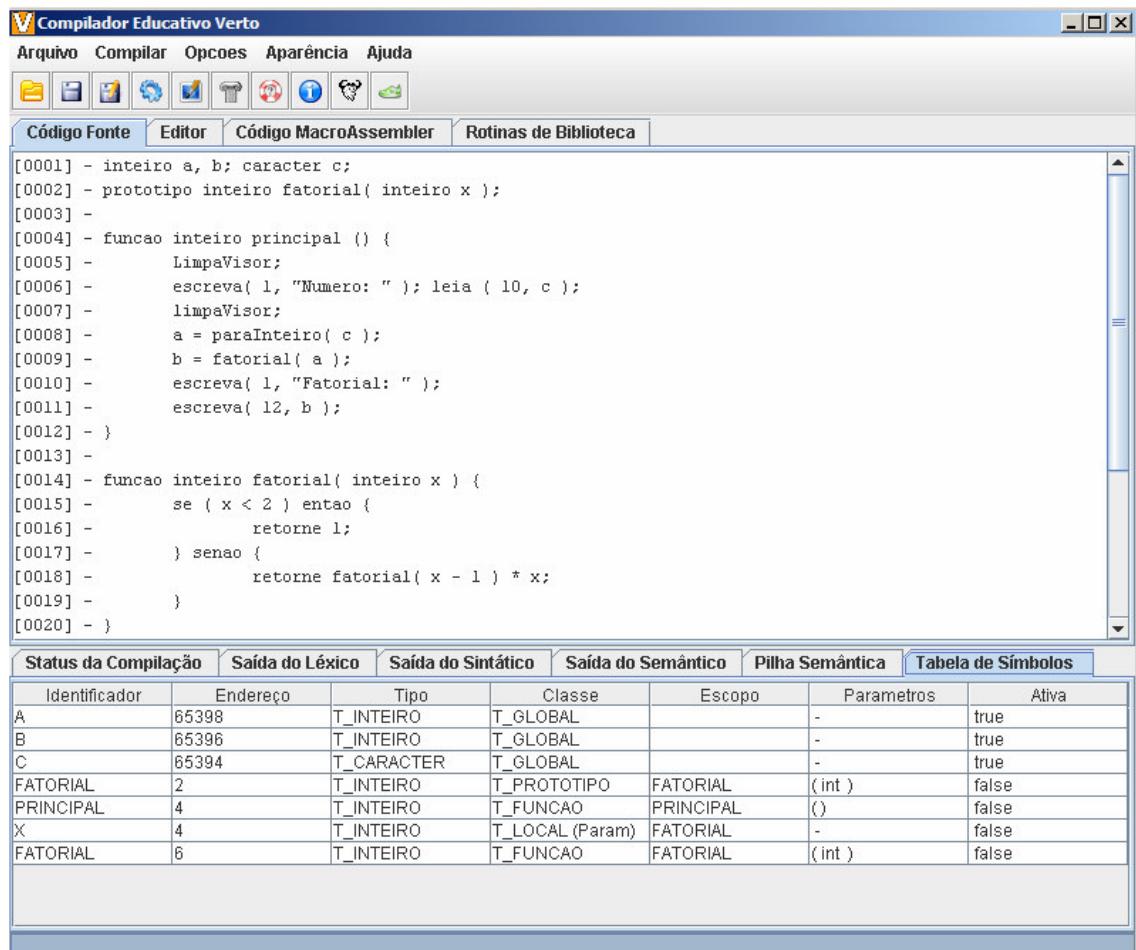
FIGURA 35 - Esquema de Tradução do Compilador *Verto*



FONTE: SCHNEIDER, PASSERINO, OLIVEIRA (2007)

Para o uso inicial da ferramenta, dispõe-se de uma tela para a edição de textos fonte escritos na linguagem *César*. Na figura 36 é possível observar a tela de edição e a tabela de símbolos gerada a partir do código inserido.

FIGURA 36 - Interface de Edição do Compilador *Verto*



FONTE: SCHNEIDER, PASSERINO, OLIVEIRA (2007)

A ferramenta focou as etapas finais da compilação, utilizando uma técnica de análise léxica simples e um método de análise sintática. A análise sintática, onde é feita a geração de erros, análise semântica e geração do código-objeto, é feita de forma recursiva, onde cada regra sintática reconhecida pode levar o compilador a disparar uma rotina de ação semântica ou de geração de código. A ampliação da saída do analisador sintático é ilustrada pela figura 37, onde é registrada a sequência de regras reconhecidas pelo compilador.

FIGURA 37 - Aba: Saída do Sintático

Status da Compilação	Saída do Léxico	Saída do Sintático	Saída do Semântico	Pilha Semântica	Tabela de Símbolos
		<pre> Reconhecida Regra: &lt;tipo inteiro #&gt; Reconhecida Regra: &lt;Entrei no identificador: a&gt; Reconhecida Regra: &lt;identificador&gt; Reconhecida Regra: &lt;declaracaoID&gt; Reconhecida Regra: &lt;Entrei no identificador: b&gt; Reconhecida Regra: &lt;identificador&gt; Reconhecida Regra: &lt;declaracaoID&gt; Reconhecida Regra: &lt;listaDeclaracoesIdentificadores&gt; Reconhecida Regra: &lt;declaracao&gt; Reconhecida Regra: &lt;declaracaoGlobal&gt; </pre>			

FONTE: SCHNEIDER, PASSERINO, OLIVEIRA (2007)

Os autores da ferramenta *Verto* a descrevem como uma ferramenta de auxílio de múltiplas disciplinas, sendo elas: compiladores, arquitetura de computadores e paradigmas de linguagens de programação.

#### 4.4 INTERPRETADOR DA LINGUAGEM D+

O interpretador da linguagem D+ é um software proposto e desenvolvido por Gabriel Ribeiro (2019). Ribeiro enfatiza em seu projeto a dificuldade de aprendizagem dos discentes da disciplina de Compiladores no Bacharelado em Ciência da Computação, ele descreve a disciplina como muito abrangente e profunda. Em sua pesquisa, o autor indica que mais da metade dos estudantes de compiladores tem grande dificuldade de aprender a disciplina.

Visto o problema da aprendizagem dos estudantes, Ribeiro (2019) desenvolveu um software integrado a uma interface com o intuito de amenizar a dificuldade dos discentes no aprendizado e auxiliá-los na absorção do conteúdo de compiladores. O software consiste em uma interface de fácil utilização, onde o usuário insere o código em linguagem D+, pressiona o botão para executar a compilação, e o resultado já é gerado pelo programa contemplando: análise léxica, análise sintática e tabela de símbolos.

A análise léxica é demonstrada tanto por uma tabela de *tokens* e lexemas, quanto por um autômato finito que contém todos os autômatos possíveis para a linguagem D+.

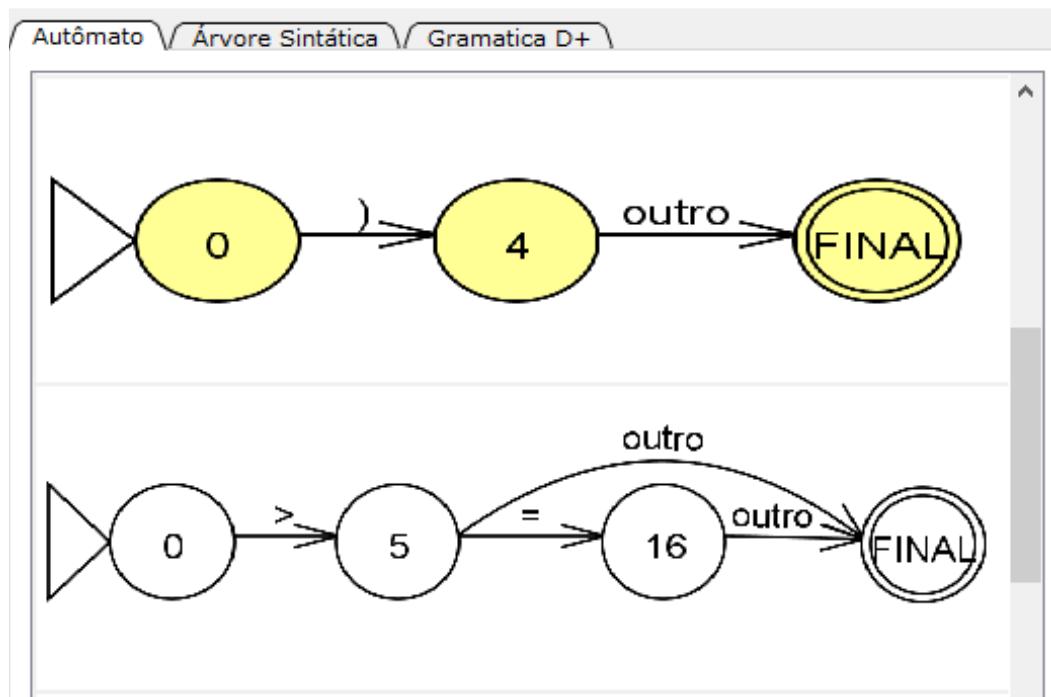
Após a compilação do código inserido pelo usuário, os autômatos utilizados são preenchidos com fundo amarelo. A figura 38 ilustra a tabela de saída da análise léxica e a figura 39 ilustra um autômato colorido quando é acessado, e outro em preto e branco quando não é acessado.

FIGURA 38 - Saída da análise sintática

Lexema	Token
ID_VARIABLE	VAR
ID_INTEGER	INT
IDENTIFICADOR	A
SIGNAL_COMMMA	,
IDENTIFICADOR	B
SIGNAL_SEMICOLON	;
ID_CONST	CONST
IDENTIFICADOR	C
OPERATOR_ATRIBUT	=
NUMREAL	93.5
SIGNAL_SEMICOLON	;
ID_SUB	SUB
ID_FLOAT	FLOAT
IDENTIFICADOR	SOMA
ID_BRACKETRIGHT	(
ID_BRACKETLEFT	)
IDENTIFICADOR	A
OPERATOR_ATRIBUT	=
NUMINT	5
OPERATOR_PLUS	+
NUMINT	6
SIGNAL_SEMICOLON	;
ID_ENDSUB	END-SUB
ID_FUNCTION	FUNCTION
ID_BOOLEAN	BOOL
IDENTIFICADOR	TESTE
ID_BRACKETRIGHT	(
ID_BRACKETLEFT	)
ID_RETURN	RETURN

FONTE: RIBEIRO (2019)

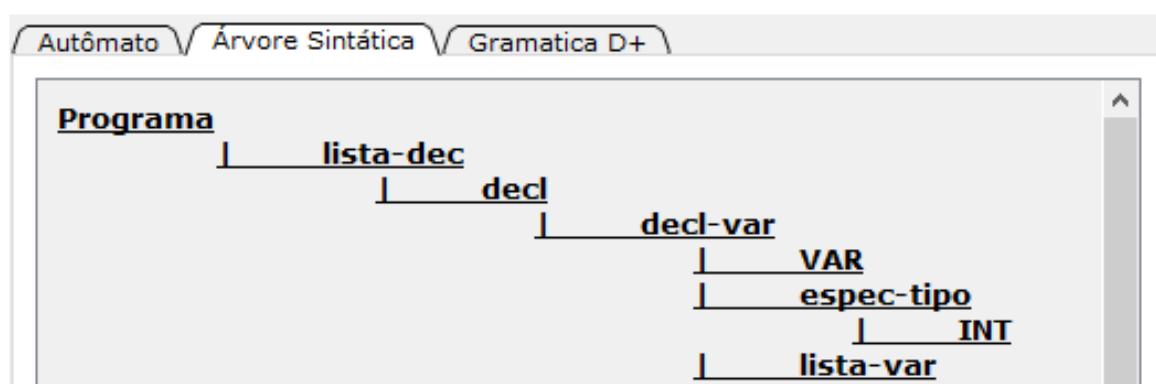
FIGURA 39 - Interface com os autômatos da análise sintática



FONTE: RIBEIRO (2019)

A análise sintática é representada por uma árvore sintática conforme ilustra figura 40.

FIGURA 40 - Árvore Sintática montada



FONTE: RIBEIRO (2019)

A tabela de símbolos ilustra os identificadores localizados pelo interpretador e traz informações que auxiliam na compreensão de cada palavra inserida no código, conforme apresentado na figura 41.

FIGURA 41 - Tabela de símbolos

#	Nome	Tipo	Categoria	Linha
0	A	INT	VARIÁVEL	1
1	C	CONST	CONSTANTE	2
2	SOMA	FLOAT	PROCEDURE	4
3	TESTE	BOOL	FUNÇÃO	8
4	D	BOOL	VARIÁVEL	13

FONTE: RIBEIRO (2019)

As tecnologias utilizadas para o desenvolvimento do interpretador de D+ foram a linguagem de programação C++ para a codificação do compilador, QT *Creator* para criação da interface e JFLAP para a criação dos autômatos finitos.

Para testes de eficiência, o interpretador de D+ foi distribuído junto a um questionário sobre o uso da ferramenta, para discentes da disciplina de Compiladores, que gerou resultados positivos, afirmando que a ferramenta ajuda na compreensão das etapas do compilador. O projeto também constou que para trabalhos futuros poderiam ser implementadas na interface a: análise semântica, geração do código de máquina, dinamicidade na apresentação da informação e o *log* da análise semântica.

#### 4.5 SCC: UM COMPILADOR C COMO FERRAMENTA DE ENSINO DE COMPILADORES

O compilador SOIS C *Compiler* (SCC), de Foleiss, Assunção, Cruz, Gonçalvez e Feltrim (2009), se destaca entre os compiladores de C por contemplar todo o conjunto de instruções de um compilador, assim proporcionando um serviço de ajuda no ensino de disciplinas que abordam o processo de compilação.

O Sistema Operacional Integrado Simulado (SOIS), foi escolhido justamente por ser um ambiente que permite a escrita, execução e depuração de programas em um ambiente simulado. Por sua vez, o montador SASM possui modos de depuração que podem demonstrar todas as fases do processo de compilação, a inter-relação dos seus artefatos e componentes e os resultados obtidos.

O projeto foi desenvolvido com os objetivos de gerar código que permita que o simulador do ambiente execute programas da linguagem C e de ser uma ferramenta de ensino, tendo funcionalidades que auxiliem a compreensão de áreas específicas do processo de um compilador real.

A pesquisa escolheu o processo de compilação proposto por Louden que possui cinco etapas: análise léxica, análise sintática, análise semântica, geração de código intermediário e geração de código final. No SCC, são geradas: a árvore sintática, a árvore de símbolos, a árvore sintática abstrata anotada e um analisador semântico.

A análise sintática tem como principal artefato a árvore sintática abstrata, nela é mostrada a estrutura sintática do programa, sendo possível visualizar o mapeamento do programa-fonte. A figura 42 ilustra um trecho de uma árvore sintática gerada pelo SCC.

FIGURA 42 - Árvore Sintática

```
Análise sintática finalizada. ASA: 0x805c8c0
(0 ) TIPO_NÓ: T_FUNC
  (0 ) TIPO_NÓ: T_T TIPO TIPO: INT (291)
  (1 ) TIPO_NÓ: T_FUNC DECL
    (0 ) TIPO_NÓ: T_ID ID: pot
    (1 ) TIPO_NÓ: T_PARAMETROS
      (0 ) TIPO_NÓ: T_PARAMETRO
        (0 ) TIPO_NÓ: T_T TIPO TIPO: INT (291)
        (1 ) TIPO_NÓ: T_ID ID: base
      (--)> TIPO_NÓ: T_PARAMETRO
        (0 ) TIPO_NÓ: T_T TIPO TIPO: INT (291)
        (1 ) TIPO_NÓ: T_ID ID: exp
  (3 ) TIPO_NÓ: T_COMPOUND_ST
    (1 ) TIPO_NÓ: T_IF
      (0 ) TIPO_NÓ: T_UN_OP UN_OP: !
      (0 ) TIPO_NÓ: T_ID ID: pot
    (1 ) TIPO_NÓ: T_RETURN
      (0 ) TIPO_NÓ: T_CONST_N CONST_N: 1
    (--)> TIPO_NÓ: T_RETURN
      (0 ) TIPO_NÓ: T_BIN_OP OP: *
      (0 ) TIPO_NÓ: T_ID ID: base
      (1 ) TIPO_NÓ: T_ATV
        (0 ) TIPO_NÓ: T_ID ID: pot
        (1 ) TIPO_NÓ: T_ID ID: base
      (--)> TIPO_NÓ: T_BIN_OP OP: -
        (0 ) TIPO_NÓ: T_ID ID: exp
        (1 ) TIPO_NÓ: T_CONST_N CONST_N: 1
```

FONTE: FOLEISSL, ASSUNÇÃO, CRUZ, GONÇALVEZ, FELTRIM (2009)

A árvore de símbolos é outro artefato que explora o entendimento do compilador, é ela a responsável por transcrever a tabela de símbolos gerada a partir do código fonte. A figura 43 ilustra a árvore de símbolos, que é dividida entre os escopos existentes no código.

FIGURA 43 - Árvore de símbolos

```
-----ARVORE DE SIMBOLOS-----
-----
Escopo = global
    Símbolo: main
        Tipo: INT
        Flags: FUNÇÃO
    Símbolo: pot
        Tipo: INT
        Flags: FUNÇÃO

-----
Escopo = pot
    Símbolo: exp
        Tipo: INT
        Flags: PARÂMETRO
    Símbolo: base
        Tipo: INT
        Flags: PARÂMETRO

-----
Escopo = main
    Símbolo: a
        Tipo: INT
        Flags: VARIÁVEL
    Símbolo: b
        Tipo: INT
        Flags: VARIÁVEL
    Símbolo: argc
        Tipo: INT
        Flags: PARÂMETRO
    Símbolo: argv
        Tipo: VOID
        Flags: PARÂMETRO PONTEIRO (Prof = 2)
```

FONTE: FOLEISS, ASSUNÇÃO, CRUZ, GONÇALVEZ, FELTRIM (2009)

A proposta do SCC é apresentar, aos discentes da disciplina de Compiladores, um compilador que mostre os desafios presentes na implementação de um compilador, utilizando métodos empregados no desenvolvimento de compiladores profissionais.

Além dos modos de depuração o SCC provê ao usuário interfaces com algumas partes do processo de compilação. O projeto também citou a possibilidade de incluir um módulo de otimização no sistema e a visualização gráfica da árvore sintática como trabalhos futuros.

## 5 METODOLOGIA

Esse capítulo apresenta a metodologia utilizada e os recursos necessários para o desenvolvimento da ferramenta VL-D+ proposta no presente projeto.

### 5.1 TECNOLOGIAS UTILIZADAS

Nessa seção são apresentados os softwares e linguagens utilizadas como base para o desenvolvimento do sistema proposto, tais como: ASP.NET Core, Visual Studio, Razor, JavaScript, C#, Canvas e Diagrams.net.

#### 5.1.1 ASP.NET Core

De acordo com a documentação da Microsoft (2020) o ASP.NET Core é uma estrutura de software livre de plataforma cruzada, de alto desempenho que fornece uma estrutura para construção de aplicativos modernos, baseados em nuvem e conectados à internet, como aplicativos *web* e aplicativos *IoT*.

Para o desenvolvimento da ferramenta VL-D+ será utilizado o ASP.NET Core MVC, que de acordo com a documentação da Microsoft (2020) é uma estrutura avançada para a criação de aplicativos *web* e APIs usando o padrão e *design* Modelo-Exibição-Controlador.

O padrão Modelo-Exibição-Controlador (MVC) ajuda a tornar as APIs *web* e os aplicativos testáveis e definem um modelo para a codificação.

A Microsoft (2020) aponta vários benefícios da utilização do ASP.NET Core, tais quais: o *Blazor* permite a utilização das linguagens de programação C# e *JavaScript* para construção da lógica do aplicativo do lado cliente e do servidor; o *Razor Pages* facilita a codificação e a torna mais produtiva em cenários focados em interfaces de páginas *web*; existe um *pipeline* de solicitação HTTP leve, modular e de alto desempenho.

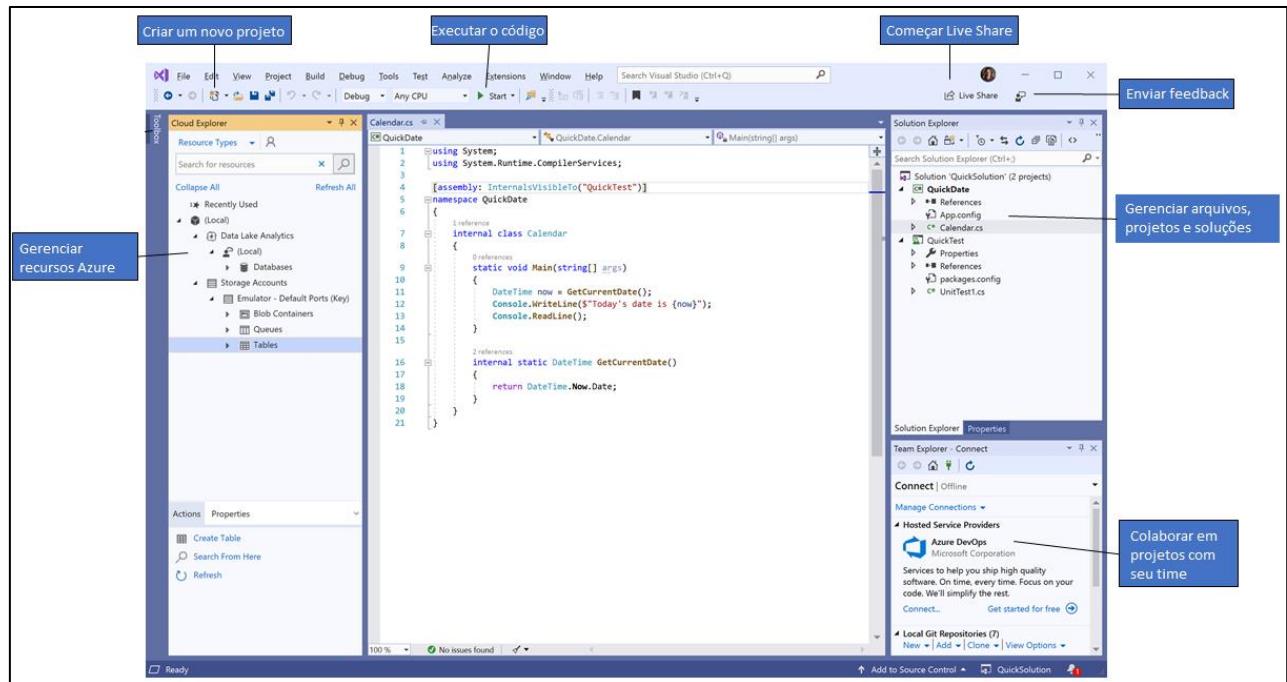
### 5.1.2 Visual Studio

Para utilização do ASP.NET Core é necessário o ambiente de desenvolvimento integrado (IDE) Visual Studio.

De acordo com a documentação da Microsoft (2019) o Visual Studio é um painel de inicialização criativo que permite ao desenvolvedor editar, depurar e compilar o código e, em seguida publicar um aplicativo.

Além do editor e do depurador já fornecidos em diversas IDE's, a Microsoft (2019) reforça que o Visual Studio inclui compiladores, ferramentas de preenchimento de código, *designers* gráficos e outros recursos que facilitam o processo de desenvolvimento do software.

FIGURA 44 - Janela da IDE do Visual Studio



FONTE: Microsoft (2019) adaptada

A figura 44 apresenta o Visual Studio como um projeto aberto e várias janelas de ferramentas que a IDE dispõe. Dois recursos principais que podem ser vistos na figura são: na parte superior direita se encontra o gerenciador de soluções que permite exibir, navegar e gerenciar os arquivos de código e organiza o código agrupando seus arquivos

em soluções e projetos; e na parte central da imagem aparece a janela do editor onde o desenvolvedor edita o código.

### 5.1.3 Razor

De acordo com a Microsoft (2020) o *Razor* é uma sintaxe de marcação para inserir código baseado em servidor em páginas da *web*.

A página *Razor* é composta, de acordo com a documentação da Microsoft (2019), por um par de arquivos: um arquivo *cshtml* que contém a marcação HTML com o código C# usando a sintaxe *Razor* e um arquivo *cshtml.cs* que contém o código C# que manipula os eventos de página.

A linguagem padrão do *Razor* é o HTML, porém, é possível escrever códigos na linguagem C# em conjunto do HTML, isto é definido pela Microsoft (2020) como expressões implícitas e explícitas.

FIGURA 45 - Exemplo de expressões em Razor

<p><b>O dia de hoje é: 25/05/2020 (ex. implícito)</b></p> <p><b>Expressão matemática: 20 (ex. explícito)</b></p>
<pre>&lt;h2&gt;O dia de hoje é: @DateTime.Now.ToShortDateString() (ex. implícito)&lt;/h2&gt; &lt;h2&gt;Expressão matemática: @(2 * (9 + 1)) (ex. explícito)&lt;/h2&gt;</pre>

FONTE: a própria autora

A figura 45 apresenta um exemplo de uso de expressão implícita e explícita. Na metade superior tem-se um exposto retirado de uma página *web* e na metade inferior têm-se o código correspondente escrito em HTML e em C#. A primeira linha do código começa com o código escrito em HTML e a expressão C# é realizada em sequência do '@', a expressão é implícita, pois a expressão C# termina assim que é inserido um separador (quebra de linha, espaçadores ou tabulações); a segunda linha do código começa com o código escrito em HTML, porém se difere da primeira linha de código, pois a expressão C# pode conter separadores enquanto estiver dentro do '@()'.

#### 5.1.4 JavaScript

Para a criação de uma página *web* interativa, dinâmica, responsiva e em tempo real o uso da linguagem de programação *JavaScript* é indispensável para a aplicação.

De acordo com a MDN *web docs* (2019) o *JavaScript* é uma linguagem de *scripting* baseada em protótipos, multi-paradigma e dinâmica, que suporta estilos orientado a objetos, imperativo e funcional.

O *JavaScript* contém diversas funções para manipulação de eventos realizados pelo usuário em alguma parte da página *web*, os mais usados são: o evento *onClick* e o evento *onChange*, geralmente aplicados em botões e em caixas de texto respectivamente.

De acordo com a MDN *web docs* (2019) as capacidades dinâmicas do *JavaScript* incluem a construção de objetos em tempo de execução, listas de parâmetros, variáveis de funções, criação dinâmica de scripts, introspecção de objetos, e recuperação de código fonte. Além disso, é possível obter qualquer valor de um objeto da tela e mudá-lo se necessário.

#### 5.1.5 C Sharp

Além do código C# (C Sharp) que pode ser utilizado nas páginas *Razor*, para as fases da compilação que não foram projetadas para ter uma ilustração dinâmica na interface foi utilizada a linguagem de programação C#.

O C# é uma linguagem de programação criada pela Microsoft como parte da plataforma .NET. A sintaxe do C# é parecida com o Java e C++ e é uma linguagem de programação orientada a objetos.

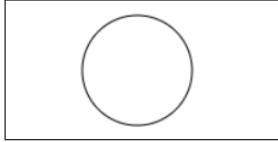
### 5.1.6 Canvas

Para a criação de uma interface dinâmica e em tempo real capaz de ilustrar o funcionamento do compilador, foi escolhida a Interface de Programação de Aplicações (API) *Canvas*.

De acordo com a MDN *web docs* (2020) o *Canvas* é uma API que provê maneiras de desenhar gráficos via *JavaScript* e via elemento HTML.

O *Canvas* é focado em gráficos 2D e pode ser utilizado para animações, gráficos de jogos, visualizações de dados, manipulações de fotos e processamentos de vídeos em tempo real.

FIGURA 46 - Exemplo do Canvas



```

<canvas id="canvas" width="200" height="100" style="border:1px solid black;"></canvas>

<script>
  var canvas = document.getElementById("canvas");
  var contexto = canvas.getContext("2d");
  contexto.beginPath();
  contexto.arc(95, 50, 40, 0, 2 * Math.PI);
  contexto.stroke();
</script>

```

FONTE: w3schools adaptado

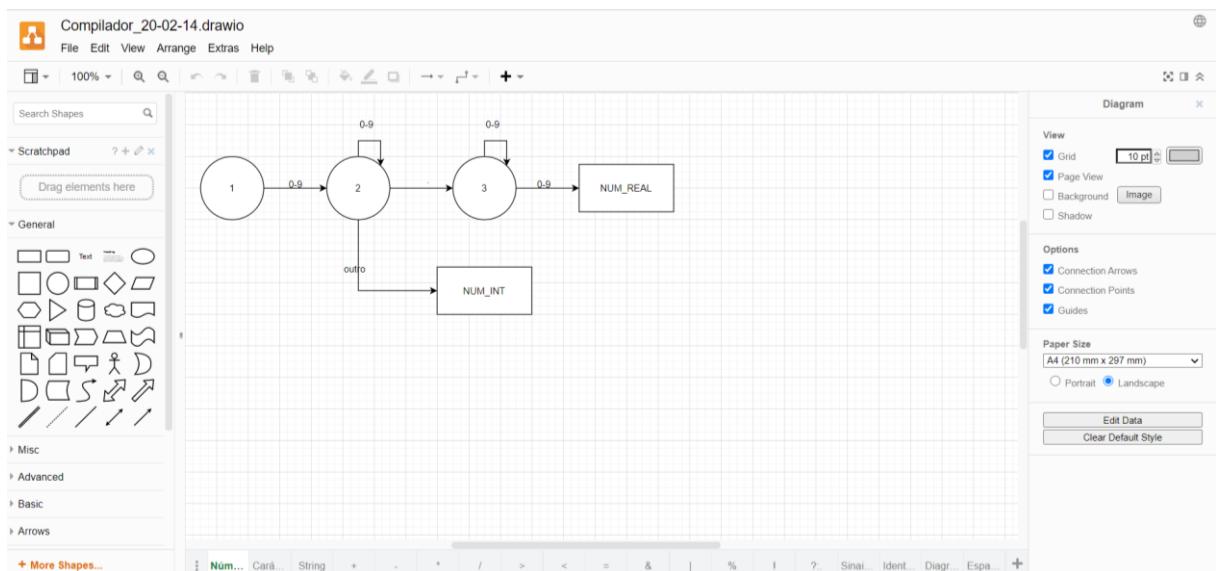
A figura 46 apresenta um exemplo de uso do *Canvas*. A metade superior mostra um exemplo de uma página *web* e a metade inferior mostra o código que foi utilizado para criar o exemplo. A tag HTML ‘`<canvas>`’ possui todas as propriedades necessárias para gerar o retângulo onde será manipulada a imagem que se deseja criar, como a altura e largura que o retângulo deve ter, o estilo da borda e o *id* para que o *JavaScript* consiga obter o elemento; a tag ‘`<script>`’ é onde o código *JavaScript* se responsabiliza por desenhar o círculo.

### 5.1.7 Diagrams.net

O *diagrams.net* é um software gratuito para a geração de diversos tipos de diagramas.

Para o presente trabalho, o *diagrams.net* foi utilizado na criação dos autômatos finitos da linguagem D+ que servem para ilustrar a análise léxica do compilador desenvolvido.

FIGURA 47 - Diagrams.net



FONTE: a própria autora

A figura 47 apresenta a interface do *diagrams.net*. No lado esquerdo são disponibilizados diversos itens de modelagem; no centro da imagem é onde cria-se o diagrama desejado; no lado direito têm-se outras configurações para os itens de modelagem; e no inferior é disponibilizado abas para a criar diagramas de um projeto separadamente.

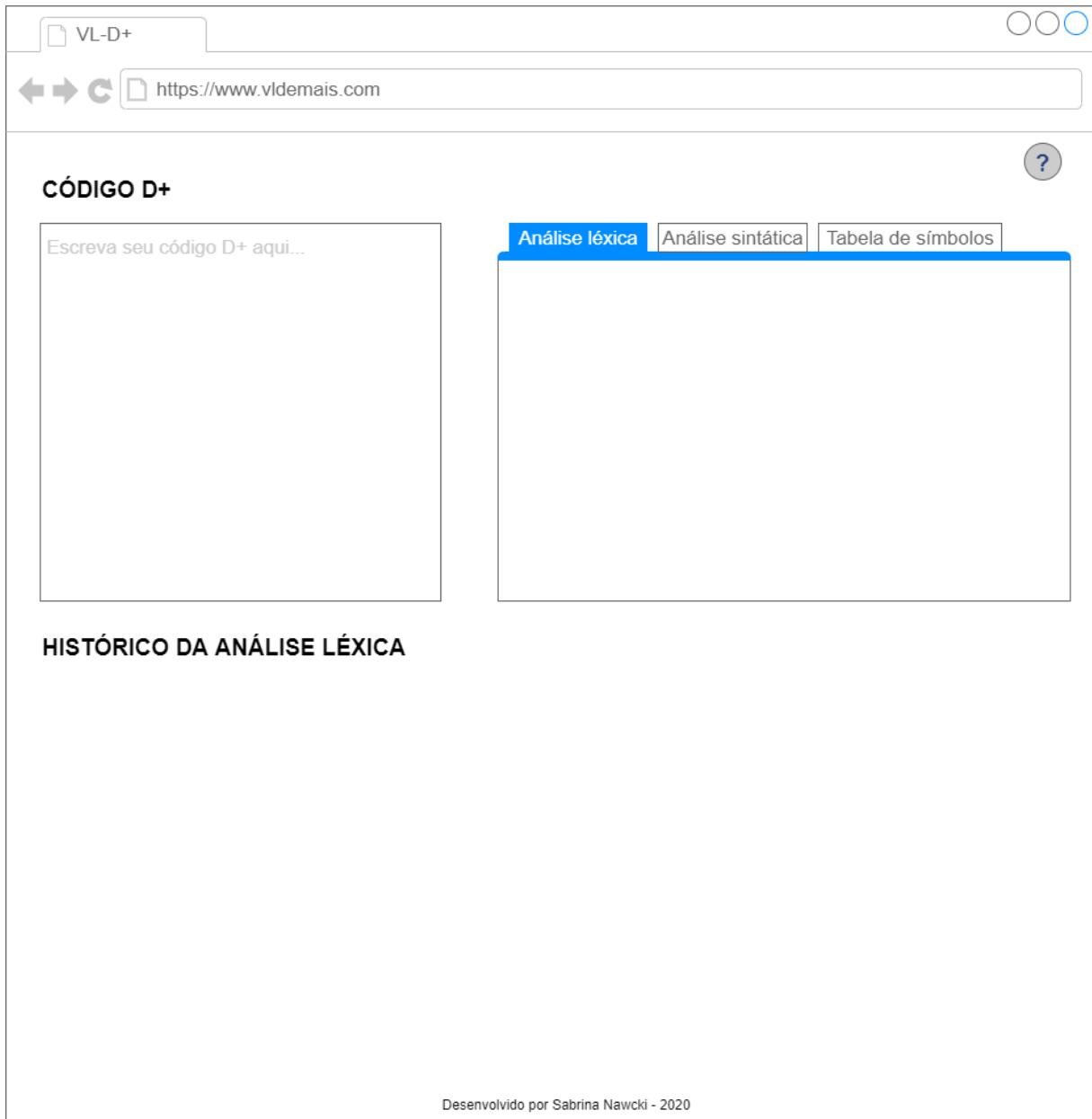
### 5.1.8 Azure

Para publicar a aplicação na *internet* foi utilizado os serviços de um mês grátis da Microsoft Azure. De acordo com a documentação da Microsoft (2020) o APIM (Gerenciamento de API) do Azure ajuda as organizações a liberar o potencial dos seus dados e serviços, permitindo-os a publicar APIs para parceiros externos e desenvolvedores internos.

## 5.2 MODELO DA INTERFACE DA FERRAMENTA VL-D+

Nessa seção são apresentados os modelos, definidos como *mockup's* na área de *design*, das possíveis telas da ferramenta VL-D+ a serem desenvolvidas, tais como: a tela inicial sem nenhuma interação do usuário e as telas da análise léxica, análise sintática e tabela de símbolos, todas apresentando uma interação feita pelo usuário.

A construção da interface da ferramenta VL-D+ foi pensada na usabilidade do usuário, dando-lhe liberdade de visualizar as fases do compilador, disponibilizadas pela ferramenta, quando quiser e sem ter a necessidade de reescrever o código ou abrir outra guia.

FIGURA 48 - *Mockup* da tela inicial

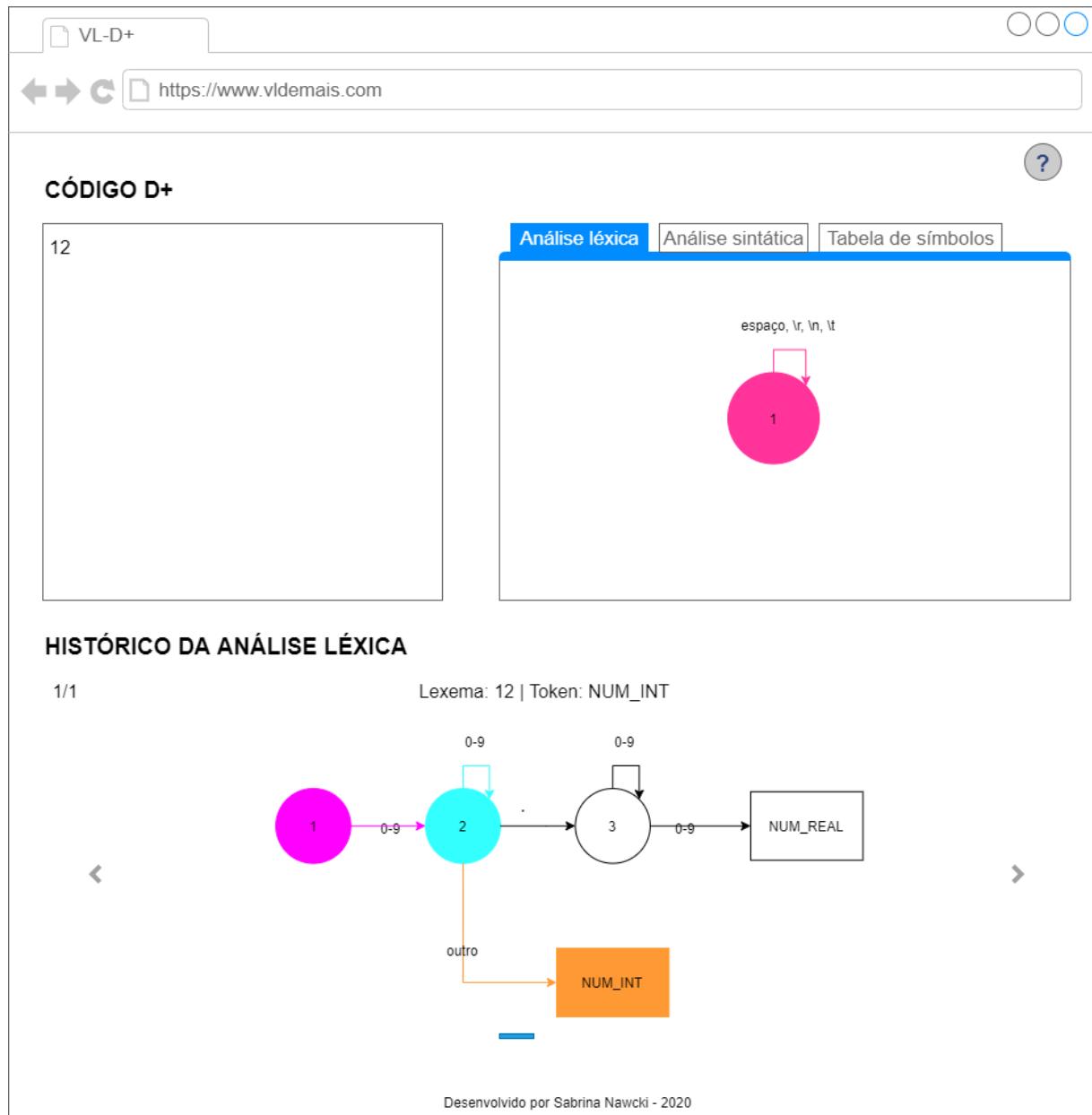
FONTE: a própria autora

A figura 48 apresenta o *mockup* da tela inicial da ferramenta VL-D+ sem nenhuma interação do usuário. Na parte superior da tela à esquerda tem-se uma caixa de texto onde o usuário deve digitar o código na linguagem D+; na parte superior da tela à direita tem-se um espaço destinado às resultantes do compilador separado pelas abas: análise léxica, análise sintática e tabela de símbolos.

Para a representação visual da análise léxica, optou-se a mostrar os autômatos finitos da linguagem D+ correspondentes ao código digitado pelo usuário e para a

melhor usabilidade da ferramenta, foi vista a necessidade de manter um histórico com os autômatos finitos gerados pela análise léxica.

FIGURA 49 - *Mockup* da análise léxica



FONTE: a própria autora

A figura 49 apresenta o *mockup* da tela inicial após uma interação fictícia do usuário na aba da análise léxica. No exemplo o usuário digita na caixa de texto, na parte superior à esquerda da tela, o valor '12' e como resultado da análise léxica são gerados dois autômatos finitos: o primeiro autômato finito pode ser visualizado no *slide show* do histórico da análise léxica, onde as cores simbolizam o percurso que a análise léxica

tomou para concluir que '12' é um número inteiro (NUM\_INT); e como o espaço ' ' é o valor corrente na caixa de texto, o segundo autômato finito gerado pode ser visualizado na parte superior à direita da tela, como a análise léxica ignora separadores o autômato finito só representa um *loop* no ponto inicial.

FIGURA 50 - *Mockup* da análise sintática

VL-D+

https://www.vldemais.com

CÓDIGO D+

```
MAIN ()
  VAR INT a;
  a = 9 + 12 * 4 - 9;
END
```

Análise léxica Análise sintática Tabela de símbolos

PROGRAMA

DECL-MAIN

Palavra(PR\_MAIN, 'main'), OK  
Palavra(SIN\_PAR\_A, '('), OK  
Palavra(SIN\_PAR\_F, ')'), OK

BLOCO

DECL-VAR

Palavra(PR\_VAR, 'var'), OK  
Palavra(PR\_INT, 'int'), OK  
Palavra(DENTIFICADOR, 'a'), OK  
Palavra(SIN\_PV, ';'), OK

FIM DECL-VAR

COM-ATRIB

HISTÓRICO DA ANÁLISE LÉXICA

6/8 Lexema: 12 | Token: NUM\_INT

```

graph LR
    1((1)) -- "0-9" --> 2((2))
    2 -- "0-9" --> 3((3))
    2 -- "outro" --> NUM_INT[“NUM_INT”]
    3 -- "0-9" --> NUM_REAL[“NUM_REAL”]
  
```

Desenvolvido por Sabrina Nawcki - 2020

FONTE: a própria autora

A figura 50 apresenta o *mockup* da tela inicial após uma interação fictícia do usuário na aba da análise sintática. No exemplo o usuário digita na caixa de texto, na parte superior à esquerda da tela, um código válido na linguagem D+ e como resultado

é gerada a árvore sintática correspondente ao código, que pode ser visualizada completamente ao mexer na barra de *scroll* situada à direita. Mesmo com o usuário em outra aba, a análise léxica funciona normalmente e gera o seu histórico.

FIGURA 51 - *Mockup* da tabela de símbolos

VL-D+

https://www.vldemais.com

CÓDIGO D+

```
CONST pi = 3.14;
MAIN ()
  VAR INT a, b;
  a = 9;
  b = 10;
END
```

Análise léxica | Análise sintática | **Tabela de Símbolos**

NOME	TIPO	ESCOPO	VALOR
PI	CONST	GLOBAL	3.14
MAIN	FUNC	GLOBAL	
A	NUM_INT	LOCAL	9
B	NUM_INT	LOCAL	10

HISTÓRICO DA ANÁLISE LÉXICA

6/8 Lexema: 12 | Token: NUM\_INT

```

graph LR
    S1((1)) -- "0-9" --> S2((2))
    S2 -- "0-9" --> S3((3))
    S3 -- "0-9" --> NUM_REAL[“NUM_REAL”]
    S2 -- “outro” --> NUM_INT[“NUM_INT”]
    
```

Desenvolvido por Sabrina Nawcki - 2020

FONTE: a própria autora

A figura 51 apresenta o *mockup* da tela inicial após uma interação fictícia do usuário na aba da tabela de símbolos. No exemplo o usuário digita na caixa de texto, na parte superior à esquerda da tela, um código válido na linguagem D+ e como resultado é gerada a tabela de símbolos correspondente ao código, que pode ser visualizada

completamente ao mexer na barra de *scroll* situada à direita. Mesmo com o usuário em outra aba, a análise léxica funciona normalmente e gera o seu histórico.

### 5.3 FUNCIONALIDADES DA FERRAMENTA VL-D+

Nessa seção são apresentadas as funcionalidades da ferramenta VL-D+ a serem desenvolvidas, mostrando cada componente de tela e como a interação do usuário com esses componentes deve ocorrer.

#### 5.3.1 Inserção de código

A inserção de código deve ser feita numa caixa de texto situada a esquerda da interface e deve possibilitar que o usuário digite qualquer caractere, delete caracteres e utilize os atalhos de copiar (CTRL + V), colar (CTRL + C), recortar (CTRL + X), selecionar tudo (CTRL + A) e desfazer (CTRL + Z).

Quando houver inserção ou remoção de caracteres no final do código o processo do compilador deve continuar sem interrupções, caso contrário um botão de recompilar código deve aparecer na tela e os processos do compilador devem parar até que o botão seja pressionado.

Caso haja erros de compilação no código inserido, o erro deve ser destacado dentro da caixa de texto em outra cor, facilitando o usuário a identificação de erros.

#### 5.3.2 Aba de seleção

A aba de seleção deverá ficar no lado direito da interface e deve possibilitar que o usuário da aplicação consiga selecionar uma das seguintes opções: Análise Léxica, Análise Sintática, Tabela de Símbolos e caso haja erros de compilação deve haver a opção de Erros.

Cada aba deve realizar as suas funções independentemente se esteja selecionada, desta forma a aplicação irá estar com todas as abas atualizadas e demorará menos para apresentar seus resultados caso o usuário queira trocar de aba.

Quando uma aba estiver selecionada a sua cor deve mudar para demonstrar ao usuário qual aba está sendo apresentada no momento.

### 5.3.3 Análise léxica

A aba da Análise Léxica deve apresentar ao usuário o respectivo autômato finito do último lexema que está sendo construído na caixa de texto de inserção de código. As cores do diagrama devem mudar de acordo com a inserção de novos caracteres na caixa de texto mostrando ao usuário em qual passo do diagrama o compilador está.

Quando o compilador identificar um *token* ou um erro sintático o autômato finito deverá ser inserido no *slide show* do histórico da Análise Léxica. Itens que não são considerados como erro, mas fazem parte do código, como espaçadores, tabulações e quebras de linhas também são demonstrados no histórico, porém, não apresentam um *token*.

### 5.3.4 Análise sintática

A aba da Análise Sintática deve apresentar ao usuário a respectiva Árvore Sintática que o compilador está construindo a partir do código digitado na caixa de texto pelo usuário até o momento atual.

### 5.3.5 Tabela de símbolos

A aba da Tabela de Símbolos deve apresentar ao usuário a tabela de símbolos gerada pelo compilador até aquele momento com os atributos que são necessários para o processo de compilação.

### 5.3.6 Registro de erros

A aba de Erros tem o propósito de ajudar o usuário a visualizar quais foram os erros encontrados até o momento atual e deve aparecer apenas quando há algum erro de compilação. Se algum erro for corrigido o mesmo deve desaparecer da aba e caso todos os erros apresentados nessa aba sejam corrigidos, a aba deve desaparecer da tela.

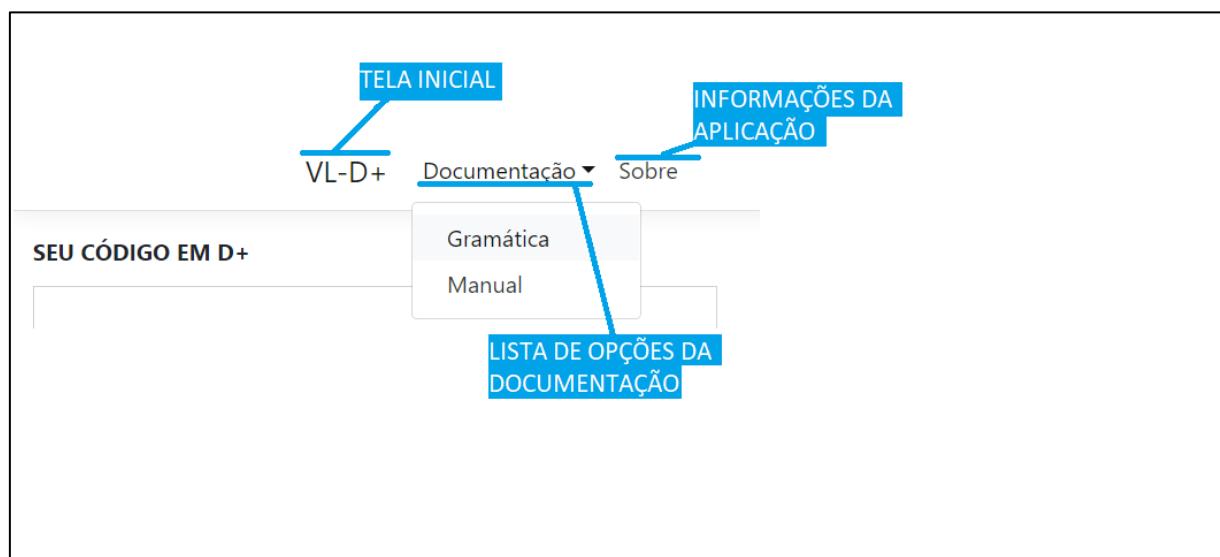
## 5.4 INTERFACE DA APLICAÇÃO

Nessa seção são apresentadas as interfaces de todas as telas da aplicação. A aplicação possui 4 telas: a tela inicial, a tela de documentação da gramática D+, a tela de documentação do manual do usuário e a tela de informação sobre a aplicação.

### 5.4.1 Barra de Menus

A barra de menus está presente em todas as telas da aplicação e permitem a navegação do usuário para cada telas. A figura 52 apresenta a legenda de cada item da barra de menus.

FIGURA 52 – Legenda da Barra de Menus

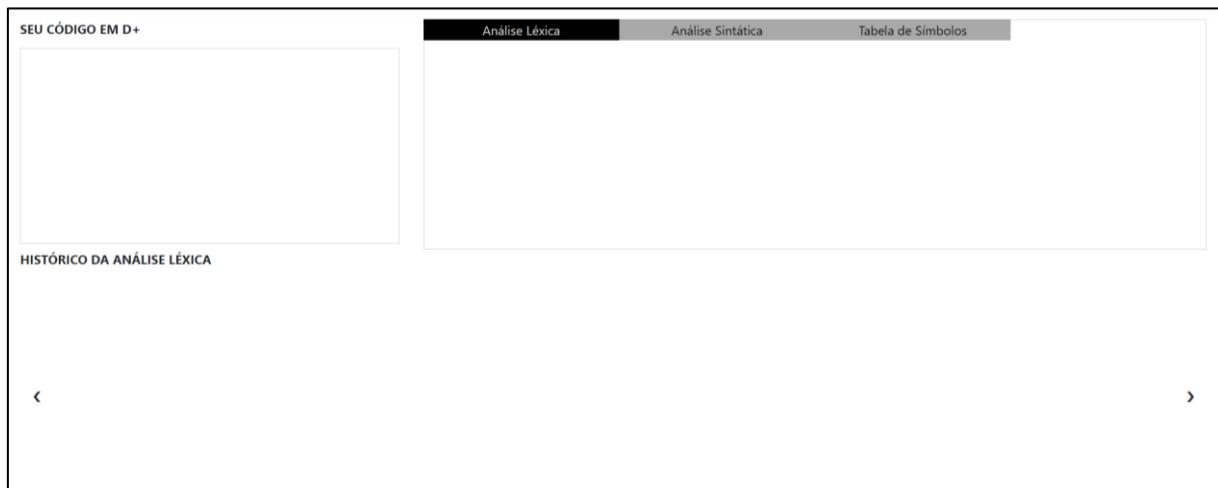


FONTE: a própria autora

#### 5.4.2 Tela Inicial

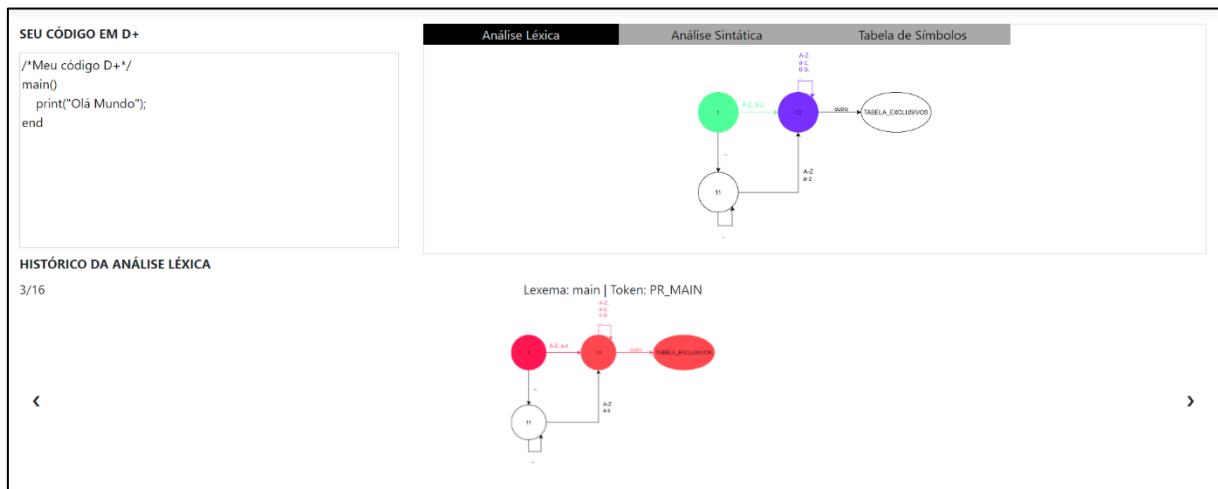
A tela inicial possui várias informações e itens em que o usuário pode navegar. As figuras 53 e 54 apresentam a tela inicial sem interação do usuário e com interação respectivamente.

FIGURA 53 - Tela Inicial sem Interação do Usuário



FONTE: a própria autora

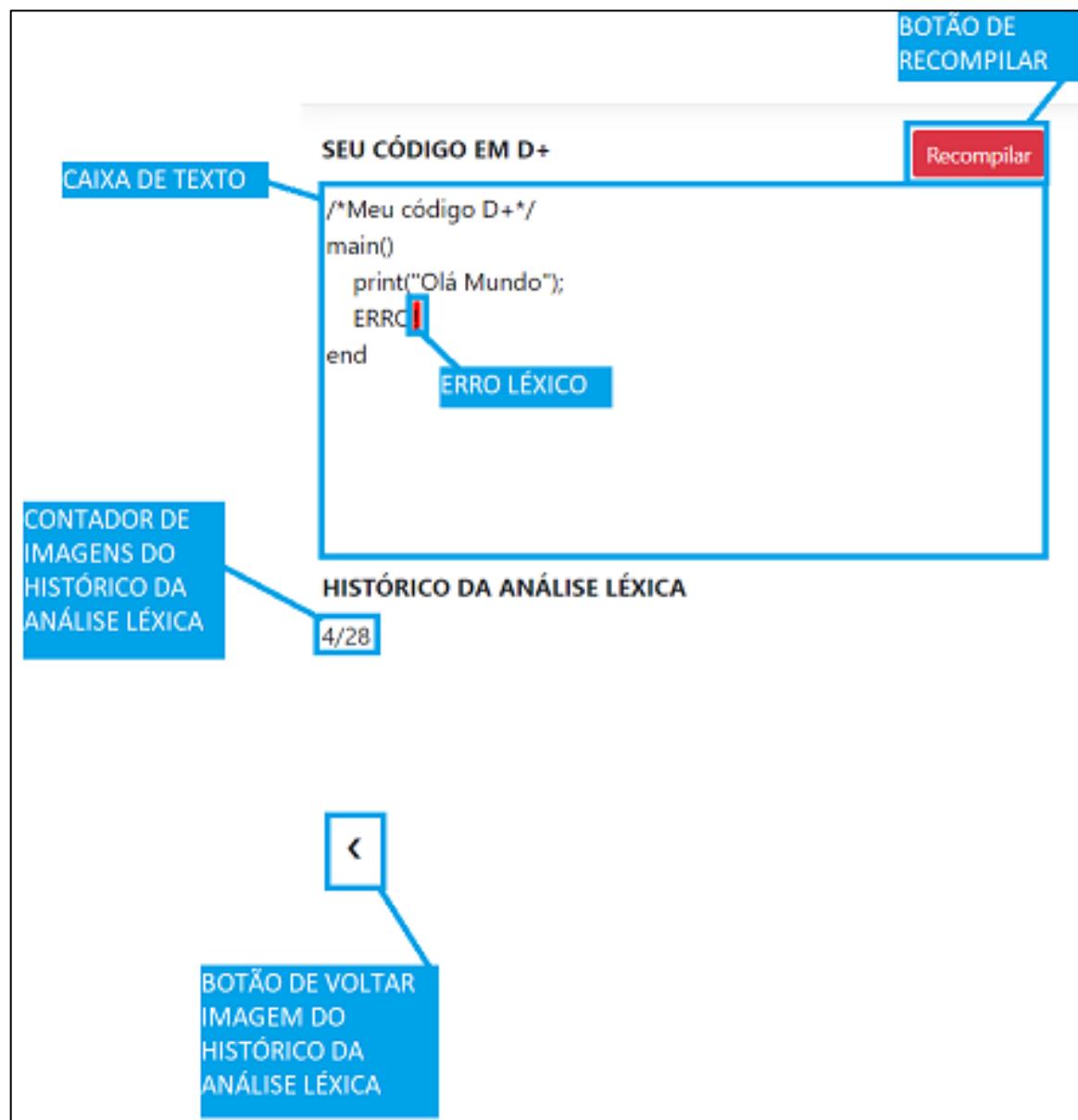
FIGURA 54 - Tela Inicial com Interação do Usuário



FONTE: a própria autora

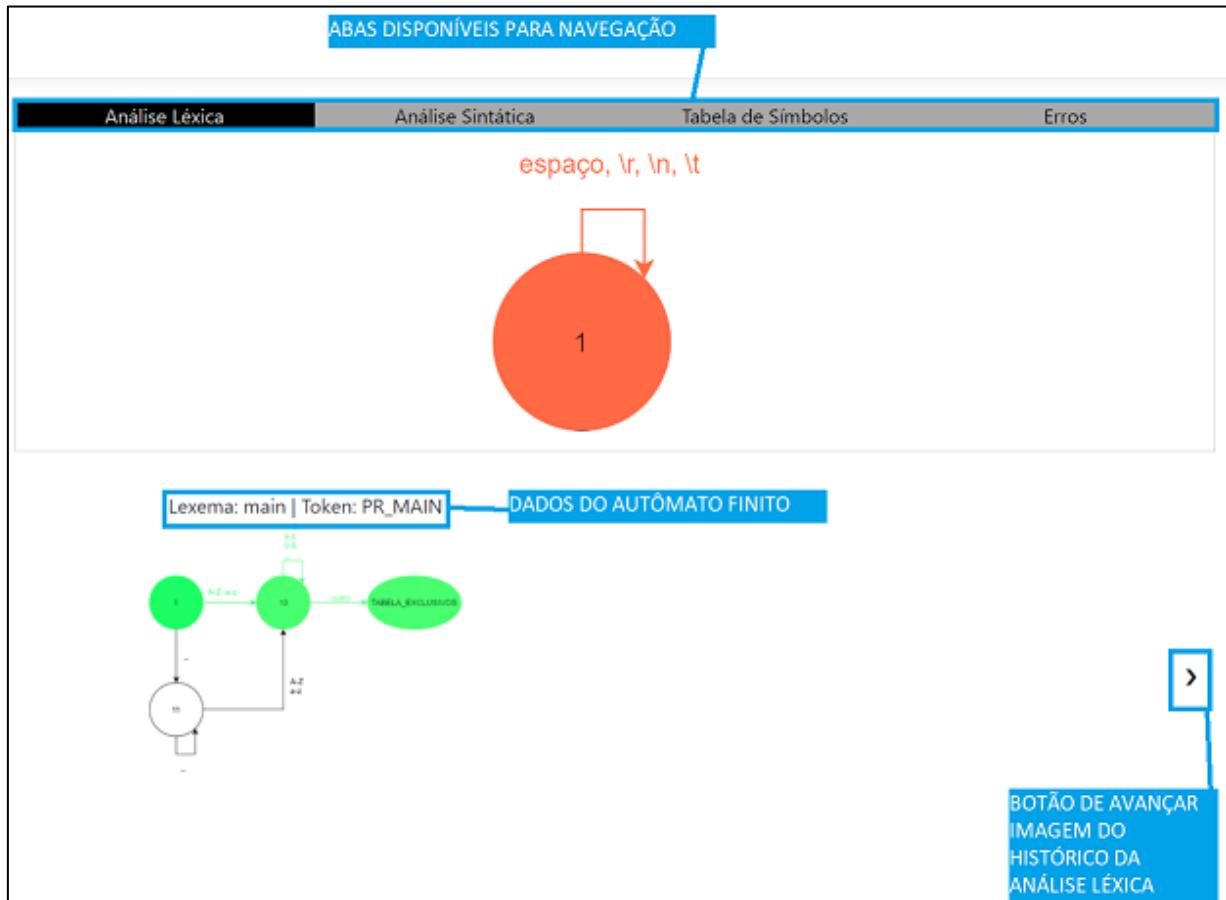
As figuras 55 e 56 apresentam a legenda dos componentes da tela inicial. Para melhor visualização a tela foi dividida no meio e as figuras apresentam o lado esquerdo e o lado direito respectivamente.

FIGURA 55 - Legenda do Lado Esquerdo da Tela Inicial



FONTE: a própria autora

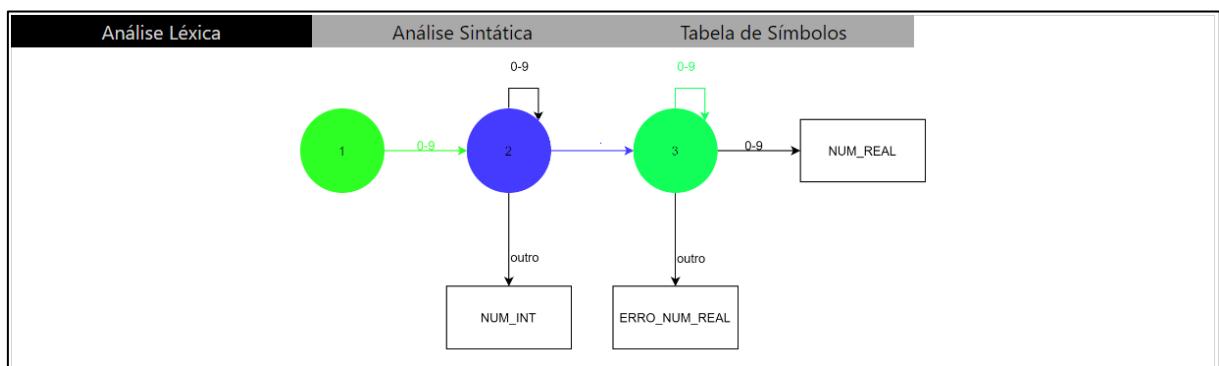
FIGURA 56 - Legenda do Lado Direito da Tela Inicial



FONTE: a própria autora

A figura 57 apresenta a aba da Análise Léxica depois da interação do usuário na caixa de texto com um código D+ lexicalmente válido, no exemplo foi utilizado a construção de um número real.

FIGURA 57 - Aba da Análise Léxica com Interação do Usuário



FONTE: a própria autora

A figura 58 apresenta a aba da Análise Sintática depois da interação do usuário na caixa de texto com um código D+ lexicalmente e sintaticamente válido, no exemplo foi utilizado a declaração da constante *pi*.

FIGURA 58- Aba da Análise Sintática com Interação do Usuário

Análise Léxica	Análise Sintática	Tabela de Símbolos
<b>programa</b> <b>lista-decl</b> <b>decl</b> <b>decl-const</b> $\text{CONST} \rightarrow \text{const}$ $\text{ID} \rightarrow \text{pi}$ $= \rightarrow =$ $\text{literal} \rightarrow 3.14$ $; \rightarrow ;$		

FONTE: a própria autora

A figura 59 apresenta a aba da Tabela de Símbolos depois da interação do usuário na caixa de texto com um código D+ que possua uma variável identificadora, no exemplo foi utilizado a declaração de 4 variáveis.

FIGURA 59- Aba da Tabela de Símbolos com Interação do Usuário

Análise Léxica	Análise Sintática	Tabela de Símbolos
Token		Lexema
IDENTIFICADOR		pi
IDENTIFICADOR		x
IDENTIFICADOR		y
IDENTIFICADOR		z

FONTE: a própria autora

A figura 60 apresenta a aba de Erros depois da interação do usuário na caixa de texto com um código D+ com um erro léxico e um erro sintático, no exemplo foi utilizado o erro sintático de uma declaração inválida no programa e um erro léxico no lexema de um número real.

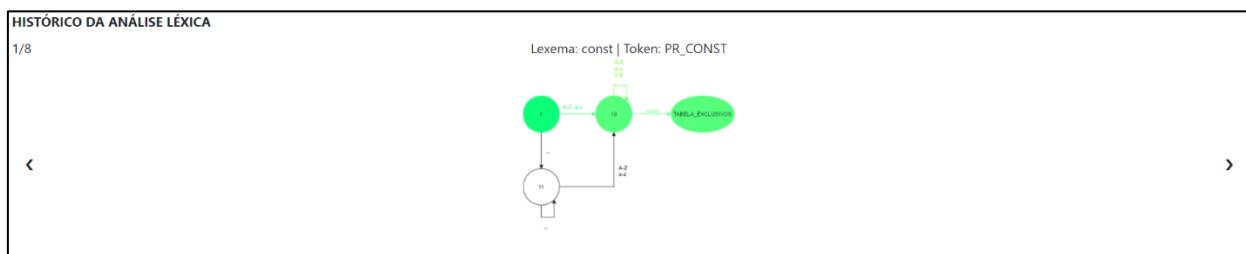
FIGURA 60- Aba de Erros com Interação do Usuário

Análise Léxica	Análise Sintática	Tabela de Símbolos	Erros
Erro sintático na declaração. Esperava-se uma declaração: constante, variável, procedimento ou função.			
Erro no lexema '3.'. O lexema esperava um número e recebeu o valor 'x'			

FONTE: a própria autora

A figura 61 apresenta o Histórico da Análise Léxica depois da interação do usuário na caixa de texto com um código D+ com mais de um lexema, no exemplo foi utilizado a declaração da constante *pi*.

FIGURA 61- Histórico da Análise Léxica com Interação do Usuário



FONTE: a própria autora

### 5.4.3 Tela de documentação da Gramática D+

A tela de documentação da gramática D+ apresenta ao usuário todas as regras da gramática. A figura 62 apresenta a tela sem interação com o usuário.

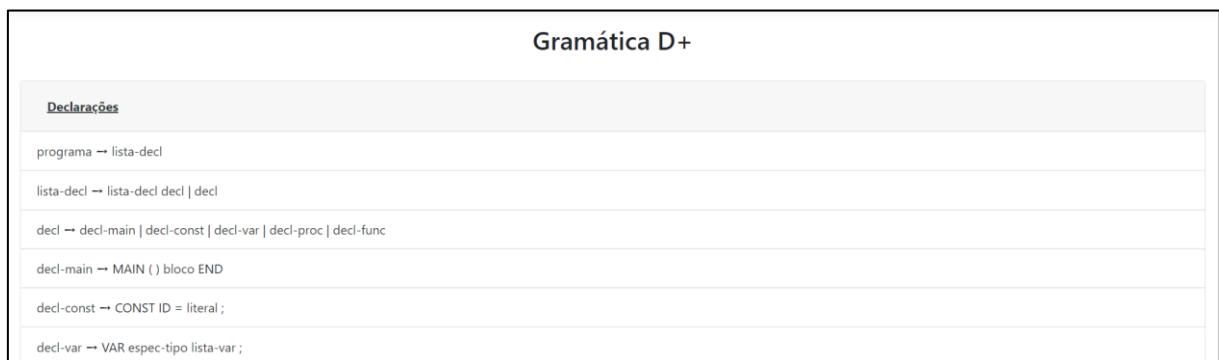
FIGURA 62 - Tela da Gramática D+ sem Interação do Usuário



FONTE: a própria autora

Cada bloco da gramática expande ao ser clicado e apresenta as regras daquele bloco. A figura 63 apresenta um exposto do bloco de Declarações.

FIGURA 63 - Tela da Gramática D+ Declarações



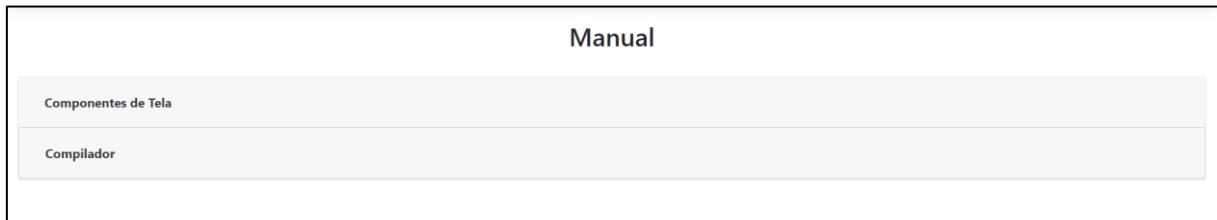
FONTE: a própria autora

Cada regra contida na linha do bloco direciona a tela para a linha correspondente à aquela regra quando clicada. Por exemplo, se a palavra “literal” for clicada na linha da regra da declaração de constante (*decl-const*) a tela irá até a linha da regra de literal que está contida no bloco de Expressões.

#### 5.4.4 Tela de documentação do Manual do Usuário

A tela de documentação do manual do usuário apresenta ao usuário o manual de uso da aplicação, explicando como cada componente da tela inicial deve se comportar. A figura 64 apresenta a tela sem interação com o usuário.

FIGURA 64 - Tela do Manual sem Interação do Usuário



FONTE: a própria autora

Cada bloco do manual expande ao ser clicado e apresenta em sub blocos quais itens são explicados. A figura 65 apresenta o bloco de Componentes de Tela.

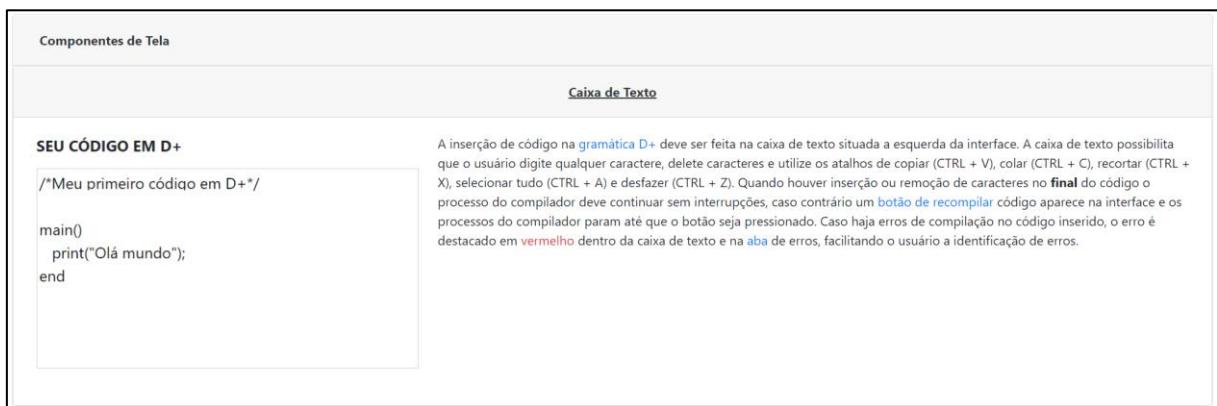
FIGURA 65 - Tela do Manual Componentes de Tela



FONTE: a própria autora

O manual dos componentes de tela explica como é o funcionamento da caixa de texto, do botão de recompilar e da aba de seleção. Cada sub bloco expande ao ser clicado e apresenta o manual daquele item. A figura 66 apresenta o manual da Caixa de Texto.

FIGURA 66 - Tela do Manual Caixa de Texto

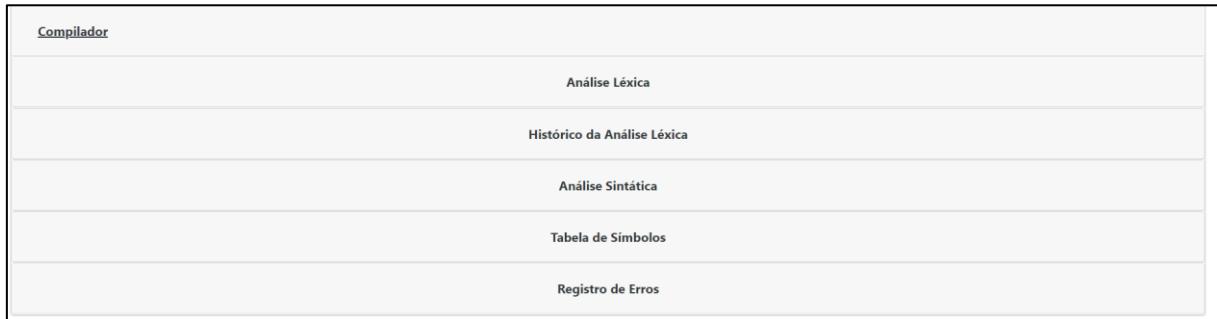


FONTE: a própria autora

No manual foram inseridos *links* destacados em azul claro que quando clicados direcionam a tela até o manual correspondente daquela palavra.

O manual do compilador explica como é o funcionamento da Análise Léxica, do Histórico da Análise Léxica, da Análise Sintática, da Tabela de Símbolos e do Registro de Erros. A figura 67 apresenta o bloco do Compilador.

FIGURA 67 - Tela do Manual Compilador

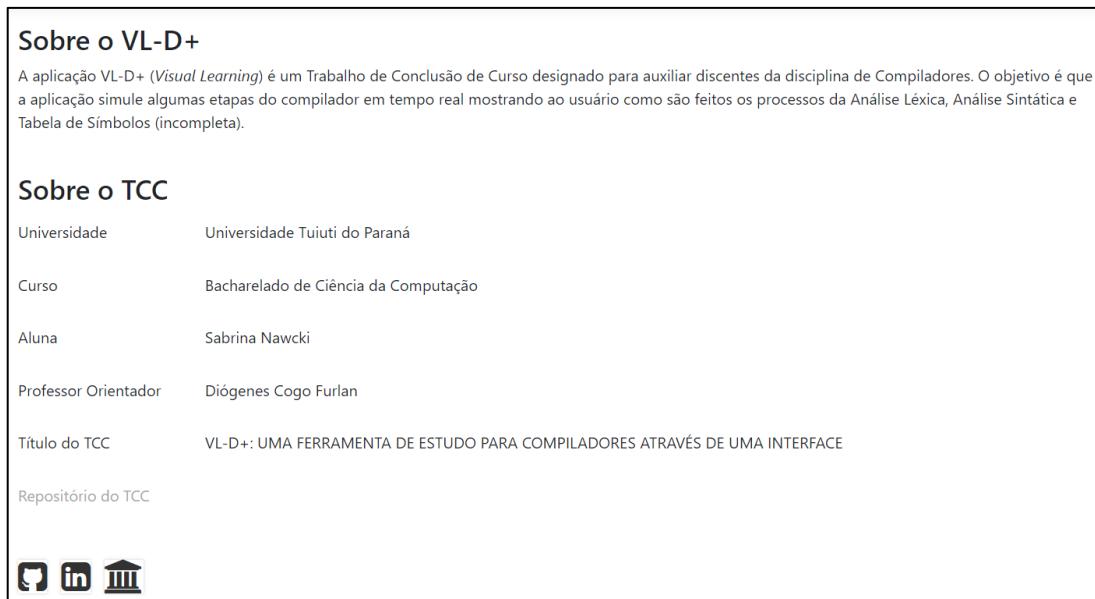


FONTE: a própria autora

#### 5.4.5 Tela Sobre

A tela Sobre apresenta brevemente o que a aplicação VL-D+ trata, apresenta dados gerais do Trabalho de Conclusão de Curso (TCC) e possui o botão do *GitHub* e o botão do *Linkedin* que direcionam o usuário para a página do *GitHub* ou para o *Linkedin* da desenvolvedora e autora do projeto e o botão que direciona o usuário para o *site* oficial da Universidade Tuiuti do Paraná. A figura 68 apresenta a tela.

FIGURA 68 - Tela Sobre



FONTE: a própria autora

## 5.5 DESENVOLVIMENTO DA APLICAÇÃO

Nessa seção é apresentado o desenvolvimento da aplicação, mostrando a implementação do código para cada componente da ferramenta VL-D+.

### 5.5.1 Caixa de texto

A caixa de inserção de código possui dois eventos *javaScript*: o evento *onKeyUp* que é disparado toda vez que uma tecla é pressionada e liberada na caixa de texto e o evento *onPaste* que é disparado toda vez que um texto é colado na caixa de texto. A figura 69 apresenta a implementação do código *cshtml* da caixa de texto.

FIGURA 69 - Caixa de Texto

```
<textarea id="userCode"
          onkeyup="TextBoxComponent.onKeyUp(event)"
          onpaste="TextBoxComponent.onPaste(event)"></textarea>
```

FONTE: a própria autora

A função *onKeyUp* analisa a tecla pressionada pelo usuário e decide quais funções devem ser disparadas a partir dela. A figura 70 apresenta a função.

FIGURA 70 - Função *onKeyUp*

```

onKeyUp: function (e) {
    var key = e.key;
    if (e.ctrlKey)
        return;

    if (key === "Backspace") {
        this.onBackspace();
    } else if (key === "Enter" || key.length === 1) {
        if (!document.getElementById("recompileButton").hasAttribute("hidden"))
            return;

        var entireCodeBefore = this.getEntireCode();
        var entireCodeCurrent = this.getEntireCurrentCode();
        if (entireCodeCurrent.indexOf(entireCodeBefore) !== 0) {
            document.getElementById("recompileButton").removeAttribute("hidden");
            return;
        }
        LexicalAnalysis.onUserCodeKeyUp(key);
    }
    this.applyText(textArea.value);
    SyntaxAnalysisController.scrollBottom();
    SymbolTableController.scrollBottom();
    ErrorManager.scrollBottom();
},

```

FONTE: a própria autora

Caso a tecla pressionada seja a tecla de deletar é disparada a função *onBackspace*, onde podem ocorrer três cenários: se a caixa de texto está vazia todo o conteúdo do Histórico da Análise Léxica e das abas são limpados; se o caractere excluído for o último a função *backspaceEvent* da Análise Léxica é disparado; caso não sejam nenhum dos dois cenários já mencionados o botão de recompilar fica visível ao usuário. A figura 71 apresenta a função.

FIGURA 71 - Função *onBackspace*

```

onBackspace: function () {
    var entireCodeBefore = this.getEntireCode();
    var entireCodeCurrent = this.getEntireCurrentCode();
    if (entireCodeCurrent === "") {
        LexicalAnalysis.clearLexicalAnalysis();
        ErrorManager.clearErrors();
        SymbolTableController.clear();
        SyntaxAnalysisController.clear();
    }
    else {
        if (entireCodeBefore.indexOf(entireCodeCurrent) === 0) {
            LexicalAnalysis.backspaceEvent();
        } else {
            document.getElementById("recompileButton").removeAttribute("hidden");
        }
    }
},

```

FONTE: a própria autora

Se a tecla pressionada seja um caractere ou a tecla *Enter* e o botão de recompilar não esteja escondido para o usuário a função retorna e não faz nada, se o botão de recompilar está escondido para o usuário é verificado se o caractere foi digitado no início ou meio do texto, se for esse o caso o botão de recompilar ficará visível para o usuário e a função é retornada, se não a função *onUserCodeKeyUp* da Análise Léxica é disparada.

Após a análise da tecla são disparadas as funções *applyText* que é responsável por aplicar o mesmo texto da caixa de texto dentro de uma *div* no *cshtml* e colorir os erros léxicos em vermelho e as funções *scrollBottom* que são responsáveis por mover a barra de rolagem para baixo em todas as abas que possuem a barra. A figura 72 apresenta a função *applyText*.

FIGURA 72 - Função *applyText*

```

applyText: function (text) {
  var matches = [];

  matches = matches.concat(text.match(invalidRegex));
  matches = matches.concat(text.match(invalidCharRegex));
  matches = matches.filter((v, i) => matches.indexOf(v) === i && v !== null && !v.match(charRegex));

  for (var i = 0; i < matches.length; i++) {
    text = text.split(matches[i]).join('<span>' + matches[i] + '</span>');
  }

  matches = [];
  matches = matches.concat(text.match(stringRegex));
  matches = matches.concat(text.match(blockCommentRegex));
  matches = matches.concat(text.match(lineCommentRegex));
  matches = matches.filter((v, i) => matches.indexOf(v) === i && v !== null);

  for (var i = 0; i < matches.length; i++) {
    text = text.split(matches[i]).join(matches[i].replace('<span>', '').replace('</span>', ''));
  }

  // replace all the line braks by <br/>, and all the double spaces by the html version &nbsp;
  text = this.replaceAll(text, '\n', '<br/>');
  text = this.replaceAll(text, ' ', '&nbsp;');
  text = this.replaceAll(text, '<span>', '<span style=background-color:red;>');

  // re-inject the processed text into the div
  highlighter.innerHTML = text;
},

```

FONTE: a própria autora

Para aplicar a cor vermelha nos erros léxicos foram utilizadas expressões *Regex* que encontram as palavras que são consideradas como erro na gramática D+. A figura 73 apresenta as expressões *Regex* utilizadas

FIGURA 73 - Expressões *Regex*

```

var invalidRegex = /((?![0-9]).(?![0-9]))|([^\0-9].[0-9])|([0-9].(?![0-9]))|[^ .,;/()^*"]<=_a-zA-Z0-9[\-\]\n]+/gm;
var invalidCharRegex = /('[^']{0,1}?'')|('(?![^']{0,1}?'')')/gm;
var stringRegex = /"(?:[^\\"]|\\.)*/";
var blockCommentRegex = /[/][*](?:[*][/])|.*[*][/]/;
var lineCommentRegex = /[*][/].*/;
var charRegex = /'[^']{0,1}?'gm;

```

FONTE: a própria autora

Na função *onPaste* pode ocorrer três cenários: o botão de recompilar está visível para o usuário então a função retorna e não faz nada; a caixa de texto já possui um código então o botão de recompilar fica visível ao usuário e a função retorna e não faz nada; caso não seja nenhum dos cenários já mencionados a função *analyseEntireCode* é

disparada recebendo o texto colado como parâmetro. A figura 74 apresenta a função *onPaste*.

FIGURA 74 - Função *onPaste*

```
onPaste: function (e) {
    if (!document.getElementById("recompileButton").hasAttribute("hidden"))
        return;

    if (this.getEntireCurrentCode().length !== 0) {
        document.getElementById("recompileButton").removeAttribute("hidden");
        return;
    }
    var code = (e.originalEvent || e).clipboardData.getData('text/plain');
    LexicalAnalysis.analyseEntireCode(code);
},
```

FONTE: a própria autora

### 5.5.2 Análise léxica

A Análise Léxica pode ser inicializada pela função *onUserCodeKeyUp* quando uma tecla de caractere ou a tecla *Enter* é pressionada, pela função *backspaceEvent* quando a tecla de deletar é pressionada ou pela função *analyseEntireCode* quando um texto é colado ou quando o botão de recompilar é pressionado.

A função *onUserCodeKeyUp* dispara a função *startAnalysis* e caso ainda tenha caracteres não analisado ela dispara a função *repeatWordEvent*. A figura 75 apresenta a função.

FIGURA 75 - Função *onUserCodeKeyUp*

```
onUserCodeKeyUp: function (key) {
    this.startAnalysis(key, false);
    if (repeatWord !== "")
        this.repeatWordEvent();
},
```

FONTE: a própria autora

A função *startAnalysis* é responsável por realizar a Análise Léxica, colorir o autômato finito na aba da Análise Léxica e quando um *token* for identificado a função deve adicionar o autômato finito no Histórico da Análise Léxica e disparar as funções da Tabela de Símbolos e da Análise Sintática. A função *repeatWordEvent* é responsável por disparar a função *startAnalysis* até que todos os caracteres sejam examinados.

Para realizar a Análise Léxica a função *startAnalysis* dispara a função *analysis* que baseada no estado corrente do lexema dispara a função de análise léxica correspondente. Os números das funções são os mesmos que foram mapeados no autômato finito completo da gramática D+. A figura 76 apresenta um exposto da função.

FIGURA 76 - Função *analysis*

```
analysis: function (character) {
    var state = this.getState();

    this.setElementAttribute("state", "state_init", state);
    switch (state) {
        /*
         * The 1st case validates all entries
         */
        case "1": {
            this.analysisCase1(character);
            break;
        }
        /*
         * The 2nd case validates integer entries
         */
        case "2": {
            this.analysisCase2(character);
            break;
        }
        /*
         * The 3th case validates decimal entries
         */
        case "3": {
            this.analysisCase3(character);
            break;
        }
        /*
         * The 4th case validates character entries
         */
        case "4": {
            this.analysisCase4(character);
            break;
        }
        /*
         * The 5th case validates character entries
         */
        case "5": {
            this.analysisCase5(character);
            break;
        }
    }
}
```

FONTE: a própria autora

A figura 77 apresenta a função de Análise Léxica para o estado 1. A função compara o caractere recebido com diversas condições e se a condição se adequar a variável *state* (estado) recebe o *token* correspondente ou o próximo estado.

FIGURA 77 - Função *analysisCase1*

```
analysisCase1: function (character) {
    if (separators.includes(character) || separatorsCode.includes(character.charCodeAt()))
        this.setElementAttribute("state", "state", "SEPARADOR");
    else if ("," === character)
        this.setElementAttribute("state", "state", "SIN_V");
    else if (";" === character)
        this.setElementAttribute("state", "state", "SIN_PV");
    else if ("(" === character)
        this.setElementAttribute("state", "state", "SIN_PAR_A");
    else if (")" === character)
        this.setElementAttribute("state", "state", "SIN_PAR_F");
    else if ("[" === character)
        this.setElementAttribute("state", "state", "SIN_COL_A");
    else if ("]" === character)
        this.setElementAttribute("state", "state", "SIN_COL_F");
    else if ("+" === character)
        this.setElementAttribute("state", "state", "OP_SOMA");
    else if ("-" === character)
        this.setElementAttribute("state", "state", "OP_SUBTRAI");
    else if ("*" === character)
        this.setElementAttribute("state", "state", "OP_MULTI");
    else if (Number.isInteger(parseInt(character)))
        this.setElementAttribute("state", "state", "2");
    else if ("'" === character)
        this.setElementAttribute("state", "state", "4");
    else if ('"' === character)
        this.setElementAttribute("state", "state", "6");
    else if (">" === character)
        this.setElementAttribute("state", "state", "7");
```

FONTE: a própria autora

Quando a função *analysis* termina a função *startAnalysis* continua e realiza diversas verificações quando o estado não é o inicial: se a palavra obtida até o momento é vazia o autômato finito daquele estado deve ser inserido no *canvas*; se o estado não for um número significa que a análise léxica daquele lexema já acabou e o estado obtido é analisado: caso o estado seja um *token* de identificador deve-se disparar a função da tabela de símbolos, caso o estado seja qualquer *token* deve-se disparar a função da análise sintática e para qualquer estado deve-se disparar a função de gravar a imagem do *canvas* no histórico da análise léxica. A figura 78 apresenta o exposto da função *startAnalysis* após a chamada da função *analysis*.

FIGURA 78 - Função *startAnalysis*

```
        if (state !== "1") {
            var img_id = this.getImageId(state);
            var image = this.getImage(img_id);

            if (word === "")
                this.drawImage(canvas, context, image);

            var coordinates = this.getCoordinates(state, stateInit);

            this.dyeImage(canvas, context, coordinates);

            if (!needRepeat)
                this.setElementAttribute("state", "word", word + key);
            if (state === "SEPARADOR" && !needRepeat)
                return;

            if (!Number.isInteger(parseInt(state))) {
                if (state !== "SEPARADOR" &&
                    state !== "LOOP1" &&
                    !state.includes("ERRO") &&
                    this.getToken() === "")
                )
                    this.setElementAttribute("state", "token", state);
                var token = this.getToken()
                if (token !== "") {
                    if (token === "IDENTIFICADOR")
                        SymbolTableController.add();
                    SyntaxAnalysisController.callFunction(token, this.getWord(), true);
                }
                this.createImageFromCanvas("slideshow_child", canvas);
                this.resetStateAttributes();
            }
        },
    },
},
```

FONTE: a própria autora

Para a alteração dos *pixels* do autômato finito de um estado para outro foi criada uma variável objeto que armazena as informações de cada autômato finito. A figura 79 apresenta as coordenadas de *pixels* para o autômato finito de *strings*, onde *img\_id* é o identificador da imagem original, *case\_start* e *case\_end* são os estados inicial e final, *width\_start* e *width\_end* são os *pixels* de comprimento inicial e final, *height\_start* e *height\_end* são os *pixels* de altura inicial e final.

FIGURA 79 - Coordenadas para o autômato finito de *strings*

```

img_id: "string_diagram",
coordinates: [
    {
        case_start: "1",
        case_end: "6",
        width_start: [0, 911, 911, 918],
        width_end: [911, 918, 918, 965],
        height_start: [199, 243, 253, 199],
        height_end: [524, 253, 523, 523]
    },
    {
        case_start: "6",
        case_end: "6",
        width_start: [639, 639],
        width_end: [964, 1924],
        height_start: [110, 0],
        height_end: [526, 110]
    },
    {
        case_start: "6",
        case_end: "ERRO_STRING",
        width_start: [639, 911, 911, 918],
        width_end: [911, 918, 918, 1924],
        height_start: [199, 243, 253, 199],
        height_end: [524, 253, 524, 524]
    },
    {
        case_start: "6",
        case_end: "STRING",
        width_start: [639, 911, 911, 918, 555],
        width_end: [911, 918, 918, 965, 1050],
        height_start: [200, 243, 253, 200, 840],
        height_end: [840, 253, 840, 840, 1084]
    },
]

```

FONTE: a própria autora

Para colorir o autômato finito foi criada a função *dyeImage* que coleta a imagem gerada pelo *canvas* e substitui as cores dos *pixels* por uma cor randômica de tom claro ou médio. A figura 80 apresenta a função.

FIGURA 80 - Função *dyeImage*

```

dyeImage: function (canvas, context, coordinates) {
    var imageData = context.getImageData(0, 0, canvas.width, canvas.height);
    var data = imageData.data;
    var mustBe255 = this.getRandomValue(1, 4);
    var random_red = this.getRandomValue(0, 155);
    var random_green = this.getRandomValue(0, 155);
    var random_blue = this.getRandomValue(0, 155);
    mustBe255 === 1 ? random_red = 255 : mustBe255 === 2 ? random_green = 255 : random_blue = 255;
    var h = 0;

    while (coordinates.height_start.length > h) {
        for (var y = coordinates.height_start[h]; y < coordinates.height_end[h]; y++) {
            for (var x = coordinates.width_start[h]; x < coordinates.width_end[h]; x++) {
                var index = (x + canvas.width * y) * 4;
                var r = data[index + 0],
                    g = data[index + 1],
                    b = data[index + 2];
                if (!(r === 0 && g === 0 && b === 1)) {
                    data[index] = random_red;
                    data[index + 1] = random_green;
                    data[index + 2] = random_blue;
                }
            }
        }
        ++h;
    }
    context.putImageData(imageData, 0, 0);
},

```

FONTE: a própria autora

A função *backspaceEvent* verifica a última palavra analisada e caso essa palavra seja vazia a função trabalha com a última imagem do histórico da análise léxica que contém o valor da palavra anterior, com isso, a função é capaz de deletar essa imagem do histórico da análise léxica e deletar as referências à essa palavra de todas as abas. Após esse processo a palavra obtida do histórico da análise léxica sofre a exclusão do seu último caractere e caso a palavra fique vazia o processo é repetido. Finalmente a função *analyseEntireCode* é disparada e refaz o processo da análise léxica para a última palavra obtida. A figura 81 apresenta a função *backspaceEvent*.

FIGURA 81 - Função *backspaceEvent*

```

backspaceEvent: function () {
    var parentOfHistory = document.getElementById("slideshow_child");
    var images = this.getAllHistory();
    var word = this.getWord();
    if (word === "") {
        if (images.length > 0) {
            lastImage = images[images.length - 1];
            word = lastImage.getAttribute("word");
            ErrorManager.clearError(word);
            parentOfHistory.removeChild(parentOfHistory.lastChild);
            token = lastImage.getAttribute("token");
            if (token.length > 0) {
                if (token === "IDENTIFICADOR")
                    SymbolTableController.delete();
                SyntaxAnalysisController.backspaceEvent(token);
            }
        }
    }
    this.setElementAttribute("state", "word", word.substring(0, word.length - 1));

    word = this.getWord();

    if (word === "")...
```

this.clearCanvas();
this.resetStateAttributes();
this.analyseEntireCode(word);
},

FONTE: a própria autora

A função *analyseEntireCode* realiza a análise léxica para um texto recebido por parâmetro, disparando a função *startAnalysis* para cada caractere. A figura 82 apresenta a função *analyseEntireCode*.

FIGURA 82 - Função *analyseEntireCode*

```

analyseEntireCode: function (code) {
  var isRepeat = false;
  for (var i = 0; i < code.length; i++) {
    this.startAnalysis(code[i], isRepeat);
    if (repeatWord !== "") {
      isRepeat = true;
      --i;
    } else {
      isRepeat = false;
    }
  }
},

```

FONTE: a própria autora

### 5.5.3 Análise sintática

Para mapear a Análise Sintática foram criados dois atributos: o atributo *syntaxFunc* que empilha todas as funções que estão pendentes de execução, incluindo todas as funções anteriores à atual separando-as pelo caractere barra (/) e o atributo *syntaxStep* que mapeia o passo corrente correspondente de cada função, ambos são do tipo texto e separam os itens por vírgula.

A função *callFunction* recebe um *token*, um lexema e um valor verdadeiro-falso que determina se a função foi disparada pela Análise Léxica ou não. No começo da função são obtidos os últimos itens mapeados nos atributos e baseado no valor desses itens é disparada a função correspondente, existem casos em que a função precisa ser ignorada e removida da pilha, chamando-se a função e passo anteriores a ela. A figura 83 apresenta a parte inicial da função *callFunction*.

FIGURA 83 - Função *callFunction* exposto

```

callFunction: function (token, lexeme, fromLexic = false) {
  var func = this.getLastSyntaxFunc();
  var step = this.getLastSyntaxStep();
  var [success, hasErrors, deleteRow] = [true, false];

  if (func.includes("ignore")) {
    this.removeLastSyntaxFunc();
    this.removeLastSyntaxStep();
    func = this.getLastSyntaxFunc();
    step = this.getLastSyntaxStep();
  }
  switch (func) {
    //Declarações
    case "programa": ...
    case "lista-decl": ...
    case "decl": ...
    case "decl-const": ...
    case "decl-var": ...
    case "espec-tipo": ...
    case "decl-proc": ...
    case "decl-func": ...
    case "decl-main": ...
    case "params": ...
    case "lista-param": ...
    case "param": ...
    case "mode": ...
    //Comandos
    case "bloco": ...
    case "lista-com": ...
    case "comando": ...
    case "com-atrib": ...
    case "com-selecao": ...
    case "com-repeticao": ...
  }
}

```

FONTE: a própria autora

Após o disparo da função corrente é analisada as saídas da função: a saída de sucesso diz se a função executou algum passo ou não, a saída de erros diz se a função executou e encontrou algum erro e a saída de deletar linha diz que o que foi escrito na aba da Análise Sintática para aquela função precisa ser removido.

Caso a função *callFunction* tenha sido disparada pela Análise Léxica, a saída da função não seja de sucesso e não tenham erros, é necessário disparar a função

*callFunction* novamente pois a função corrente não executou nada. A figura 84 apresenta a função *callFunction* completa com os disparos da função minimizados.

FIGURA 84 - Função *callFunction* completa

```
callFunction: function (token, lexeme, fromLexic = false) {
    var func = this.getLastSyntaxFunc();
    var step = this.getLastSyntaxStep();
    var [success, hasErrors, deleteRow] = [true, false];

    if (func.includes("ignore")) {
        this.removeLastSyntaxFunc();
        this.removeLastSyntaxStep();
        func = this.getLastSyntaxFunc();
        step = this.getLastSyntaxStep();
    }
    switch (func) {
        ...
        if (fromLexic && !success && !hasErrors) {
            return this.callFunction(token, lexeme, true);
        }
    }
    return [success, hasErrors, deleteRow];
},
```

FONTE: a própria autora

Cada regra da gramática D+ possui uma função responsável por validar o *token* enviado pela Análise Léxica, criando a árvore da Análise Sintática e mapeando erros sintáticos se houverem. A figura 85 apresenta a função responsável pela lista de parâmetros.

FIGURA 85 - Função *listParam*

```

listParam: function (step, token, lexeme) {
    //lista-param , param | param
    switch (step) {
        case "1":
            this.replaceLastSyntaxStep("2");
            this.writeInTab(token, "lista-param");
            this.concatenateLastSyntaxFunc(this.getLastSyntaxFunc(true) + "/param");
            this.concatenateLastSyntaxStep("1");
            var [success, hasErrors, deleteRow] = this.callFunction(token, lexeme);
            if (!success) {
                return this.removeLast(false, false, true);
            }
            return [true, false, false];
            break;
        case "2":
            if ("SIN_V" === token) {
                this.replaceLastSyntaxStep("3");
                this.writeInTab(token, ",", lexeme, 1);
                return [true, false, false];
            } else {
                return this.removeLast(false, false, false);
            }
            break;
        case "3":
            this.replaceLastSyntaxStep("2");
            this.writeInTab(token, "hidden");
            this.concatenateLastSyntaxFunc(this.getLastSyntaxFunc(true) + "/param");
            this.concatenateLastSyntaxStep("1");
            var [success, hasErrors, deleteRow] = this.callFunction(token, lexeme);
            if (!success) {
                var error = "Erro sintático na lista de parâmetros. Esperava-se um parâmetro.";
                ErrorManager.addError(error, lexeme);
                this.writeInTab(token, "Erro", error, 1);
                return [false, true, false];
            }
            return [true, false, false];
            break;
        default:
    }
},

```

FONTE: a própria autora

Para a criação da árvore da Análise Sintática é utilizada a função *writeInTab* que cria as linhas para serem anexadas na aba da Análise Sintática e guarda todos os atributos obtidos até aquele momento da análise. Esses atributos são de suma importância para funções que necessitam da exclusão de linhas dessa aba. A figura 86 apresenta a função *writeInTab*.

FIGURA 86 - Função *writeInTab*

```

writeInTab: function (token = "", text, lexeme = "", addTab = 0) {
    var func = this.getLastSyntaxFunc(true);
    var step = this.getLastSyntaxStep();

    var allFuncs = this.getSyntaxFunc();
    var allSteps = this.getSyntaxStep();

    var div = document.getElementById("syntaxAnalysContent");
    var row = document.createElement("div");
    row.setAttribute("syntaxFunc", func);
    row.setAttribute("SyntaxStep", step);
    row.setAttribute("syntaxFuncs", allFuncs);
    row.setAttribute("SyntaxSteps", allSteps);
    row.setAttribute("token", token);

    if (text === "hidden") {
        row.className = "hidden";
    }
    else if (lexeme !== "") {
        row.innerHTML = this.getTabs(addTab) + text + '    -    ' + lexeme;
    }
    else {
        row.className = "font-weight-bold"
        row.innerHTML = this.getTabs(addTab) + text
    }
    div.appendChild(row);
},

```

FONTE: a própria autora

A função de exclusão de caracteres da Análise Sintática é disparada apenas quando um *token* é modificado na caixa de texto. A função procura pelas últimas linhas da aba da Análise Sintática que correspondem à última função do atributo *syntaxFunc* e pelo *token* modificado. Caso todas as linhas da aba tenham sido removidas os atributos *syntaxFunc* e *syntaxStep* voltam ao valor inicial, se não esses atributos recebem os valores que foram previamente mapeados da última linha da aba. A figura 87 apresenta um exemplo da função *backspaceEvent* da Análise Sintática.

FIGURA 87 - Função *backspaceEvent* Análise Sintática

```

do {
    if (index >= 0 && index < syntaxTabDiv.childElementCount) {
        if (index >= 0 && index < syntaxTabDiv.childElementCount) {
            var newElement = syntaxTabDiv.children[index];
            var newFunc = newElement.getAttribute("syntaxFunc").split('/');
            var newLastFunc = newFunc.pop();

            if (lastFunc === newLastFunc && token === newElement.getAttribute("token")) {
                syntaxTabDiv.children[index].remove();
            } else {
                lastFunc = func.pop();
                if (lastFunc === newLastFunc && token === newElement.getAttribute("token")) {
                    syntaxTabDiv.children[index].remove();
                    this.removeLastSyntaxFunc();
                    this.removeLastSyntaxStep();
                }
                else
                    break;
            }
        }
        else
            break;
        --index;
    }
} while (index >= 0 && index < syntaxTabDiv.childElementCount);

if (syntaxTabDiv.childElementCount === 0) {
    this.setSyntaxFunc("programa");
    this.setSyntaxStep("1");
}
else {
    this.setSyntaxFunc(newElement.getAttribute("syntaxFuncs"));
    this.setSyntaxStep(newElement.getAttribute("SyntaxSteps"));
}

```

FONTE: a própria autora

#### 5.5.4 Tabela de símbolos

A Tabela de Símbolos possui apenas as funções de inserção (*add*), remoção de um item da tabela (*delete*) e remoção de todos os itens da tabela (*clear*). A figura 88 apresenta as funções.

FIGURA 88 - Funções da Tabela de Símbolos

```

add: function () {
    var symbolTable = document.getElementById("symbolTable");
    var tbody = symbolTable.getElementsByTagName("tbody");
    var trTokenLexeme = document.createElement("tr");

    var tdToken = document.createElement("td");
    tdToken.textContent = LexicalAnalysis.getToken();
    trTokenLexeme.appendChild(tdToken);

    var tdLexeme = document.createElement("td");
    tdLexeme.textContent = LexicalAnalysis.getWord();
    trTokenLexeme.appendChild(tdLexeme);

    tbody[0].appendChild(trTokenLexeme);
},
clear: function () {
    var symbolTable = document.getElementById("symbolTable");
    var tbody = symbolTable.getElementsByTagName("tbody");
    for (var i = tbody[0].childElementCount - 1; i >= 0; --i) {
        tbody[0].children[i].remove();
    }
},
delete: function () {
    var symbolTable = document.getElementById("symbolTable");
    var tbody = symbolTable.getElementsByTagName("tbody");
    tbody[0].removeChild(tbody[0].lastChild);
}
}

```

FONTE: a própria autora

Como o desenvolvimento da Tabela de Símbolos é incompleta e não considera a Análise Semântica, as funções não precisam lidar com nenhuma regra e apenas disparam funções que lidam com o *html* da aba da Tabela de Símbolos.

#### 5.5.5 Registro de Erros

A aba de Erros possui as funções de inserção, remoção de um erro, remoção de todos os erros e a função de esconder a aba da interface.

A figura 89 apresenta as funções de inserção (*addError*), remoção de um erro (*clearError*) e remoção de todos os erros (*clearErrors*).

FIGURA 89 - Funções de inclusão e remoção de erros

```

addError: function (errorText, lexeme) {
    document.getElementById("generatedErrors").removeAttribute("hidden");

    var divError = document.getElementById("generatedErrorsContent");
    var buttonError = document.getElementById("generatedErrorsButton");
    buttonError.className = buttonError.className.replace("invisible", "visible");

    var tagError = document.createElement("p");
    tagError.textContent = errorText;
    tagError.setAttribute("lexeme", lexeme);
    divError.appendChild(tagError);
},
clearError: function (lexeme) {
    var divError = document.getElementById("generatedErrorsContent");
    if (divError.childElementCount === 0)
        return;
    var errorArray = Array.from(divError.children);
    var index = errorArray.reverse().findIndex(f => f.getAttribute("lexeme") === lexeme);
    var count = errorArray.length - 1
    var lastIndex = index >= 0 ? count - index : index;
    if(lastIndex > -1)
        divError.children[lastIndex].remove();
    if (divError.childElementCount === 0) {
        this.hideErrorTab();
    }
},
clearErrors: function () {
    var divError = document.getElementById("generatedErrorsContent");

    for (var i = divError.childElementCount - 1; i >= 0 ; --i) {
        divError.children[i].remove();
    }
    this.hideErrorTab();
},

```

FONTE: a própria autora

A figura 90 apresenta a função de esconder a aba (*hideErrorTab*).

FIGURA 90 - Função *hideErrorTab*

```

hideErrorTab: function () {
    var buttonError = document.getElementById("generatedErrorsButton");
    document.getElementById("generatedErrors").setAttribute("hidden", "hidden");
    if (!buttonError.className.includes("invisible"))
        buttonError.className = buttonError.className.replace("visible", "invisible");
    if (buttonError.style.backgroundColor === "crimson")
        document.getElementById("defaultOpen").click();
}

```

FONTE: a própria autora

### 5.5.6 Exemplo prático

Para exemplificar o funcionamento do código da aplicação responsável pelas funções do Compilador, essa subseção apresenta uma linha de código em D+ sem erros a ser analisada pela aplicação.

Os quadros da Análise Léxica apresentam quais variáveis a função da Análise Léxica obtém e os quadros da Análise Sintática apresentam quais variáveis a função da Análise Sintática obtém. Em ambos os tipos de quadros, a coluna ‘N’ simboliza qual o número da execução.

A figura 91 apresenta código completo digitado na caixa de texto.

FIGURA 91 - Exemplo Prático Código Completo

**SEU CÓDIGO EM D+**

```
const pi=3.14;
```

FONTE: a própria autora

Ao digitar o caractere ‘c’ na caixa de texto a função da Análise Léxica obtém as variáveis do quadro 6. As funções da Tabela de Símbolos e Análise Sintática não são disparadas.

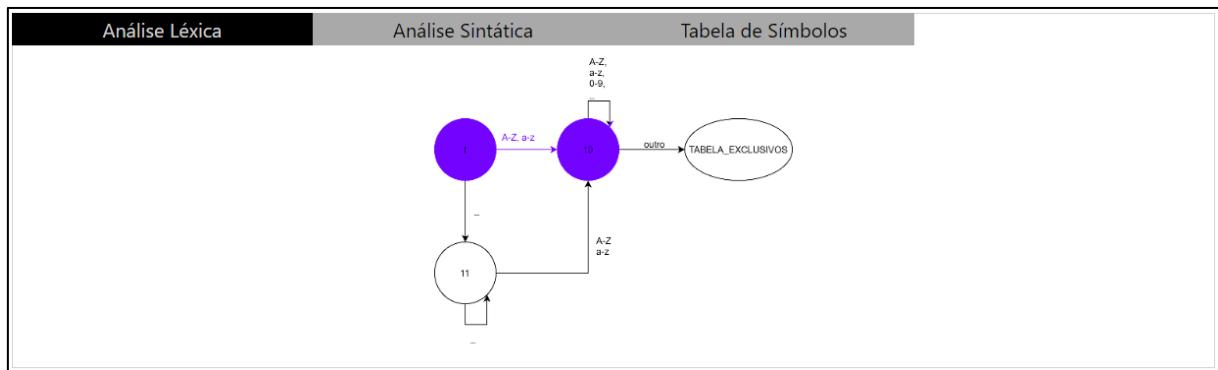
QUADRO 6 - Exemplo Prático Análise Léxica 'c'

N	Variáveis Obtidas pela Análise Léxica					
	Caractere	Palavra	Estado inicial	Estado final	Palavra p/ repetir	Token
1	c			1	10	

FONTE: a própria autora

A figura 92 apresenta a aba da Análise Léxica para o caractere 'c'.

FIGURA 92 - Exemplo Prático Análise Léxica 'c'



FONTE: a própria autora

Ao digitar a sequência de caracteres 'onst' na caixa de texto a função da Análise Léxica obtém as variáveis do quadro 7. As funções da Tabela de Símbolos e Análise Sintática não são disparadas.

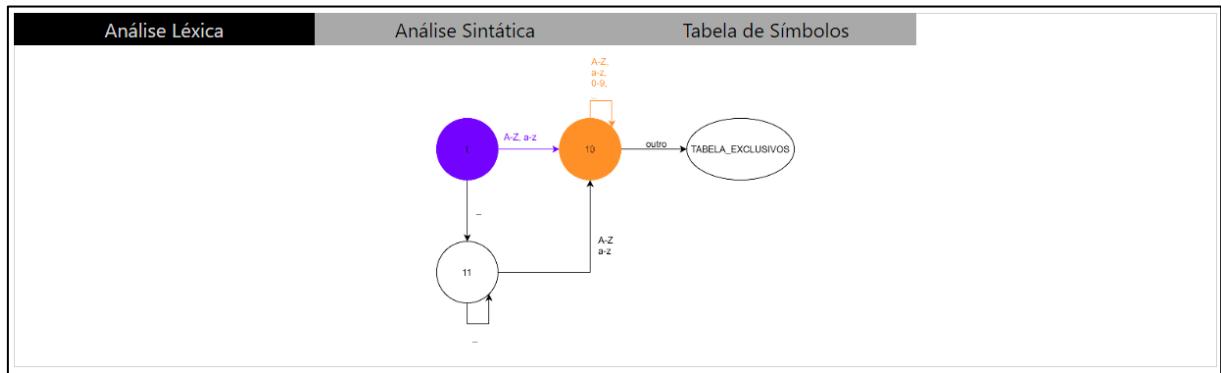
QUADRO 7 - Exemplo Prático Análise Léxica 'onst'

N	Variáveis Obtidas pela Análise Léxica					
	Caractere	Palavra	Estado inicial	Estado final	Palavra p/ repetir	Token
2	o	c		10	10	
3	n	co		10	10	
4	s	con		10	10	
5	t	cons		10	10	

FONTE: a própria autora

A figura 93 apresenta a aba da Análise Léxica para a sequência de caracteres 'onst'.

FIGURA 93 - Exemplo Prático Análise Léxica 'onst'



FONTE: a própria autora

Ao digitar o caractere de espaço ' ' na caixa de texto a função da Análise Léxica é disparada duas vezes e obtém as variáveis do quadro 8.

QUADRO 8 - Exemplo Prático Análise Léxica 'const' e ''

N	Variáveis Obtidas pela Análise Léxica					
	Caractere	Palavra	Estado inicial	Estado final	Palavra p/ repetir	Token
6	ESPAÇO	const		10	TABELA_EXCLUSIVOS	
15	ESPAÇO			1	SEPARADOR	

FONTE: a própria autora

O lexema 'const' apenas é finalizado quando a Análise Léxica recebe um caractere que não faz parte da tabela de exclusivos, por conta disso, a função primeiro termina a análise do lexema e depois repete a função para o último caractere recebido, no caso o caractere de espaço ' '.

A função da Tabela de Símbolos não é disparada e a função da Análise Sintática é disparada 8 vezes para o lexema 'const' antes do segundo disparo da Análise Léxica e obtém as variáveis do quadro 9.

QUADRO 9 - Exemplo Prático Análise Sintática 'const'

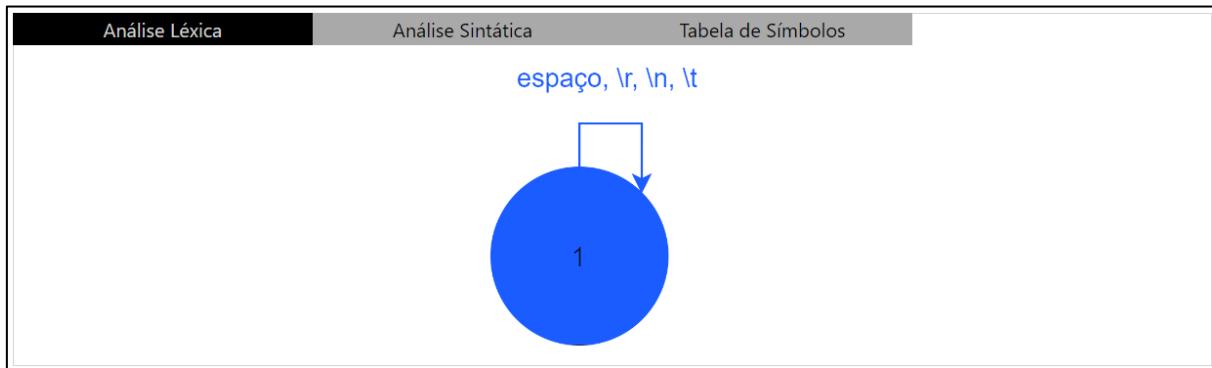
N	Variáveis Obtidas pela Análise Sintática				
	Funções	Passos	Função corrente	Passo corrente	Executou?
7	programa	1	programa	1	Sim
8	programa, programa/lista-decl	2, 1	lista-decl	1	Sim
9	programa, programa/lista-decl, programa/lista-decl/decl	2, 2, 1	decl	1	Sim
10	programa, programa/lista-decl, programa/lista-decl/ignore-decl, programa/lista-decl/ignore-decl/decl-const	2, 2, 1, 1	decl-const	1	Sim
11	programa, programa/lista-decl, programa/lista-decl/ignore-decl, programa/lista-decl/ignore-decl/decl-const, programa/lista-decl/ignore-decl/decl-const/decl-var	2, 2, 1, 2, 1	decl-var	1	Não
12	programa, programa/lista-decl, programa/lista-decl/ignore-decl, programa/lista-decl/ignore-decl/decl-const, programa/lista-decl/ignore-decl/decl-const/decl-proc	2, 2, 1, 2, 1	decl-proc	1	Não
13	programa, programa/lista-decl, programa/lista-decl/ignore-decl, programa/lista-decl/ignore-decl/decl-const, programa/lista-decl/ignore-decl/decl-const/decl-func	2, 2, 1, 2, 1	decl-func	1	Não
14	programa, programa/lista-decl, programa/lista-decl/ignore-decl, programa/lista-decl/ignore-decl/decl-const, programa/lista-decl/ignore-decl/decl-const/decl-main	2, 2, 1, 2, 1	decl-main	1	Não

FONTE: a própria autora

Na coluna funções do quadro 9 é possível analisar que a declaração dispara todas as funções do tipo declaração, mas neste caso apenas a função de declaração de constante é executada pois é a única que pode ser inicializada com o *token* 'PR\_CONST'. Além disso, a função de declaração 'decl' é transformada em 'ignore-decl' pois caso a Análise Sintática não receba uma declaração a função deve retornar para a função anterior.

A figura 94 apresenta a aba da Análise Léxica para o caractere de espaço ' '.

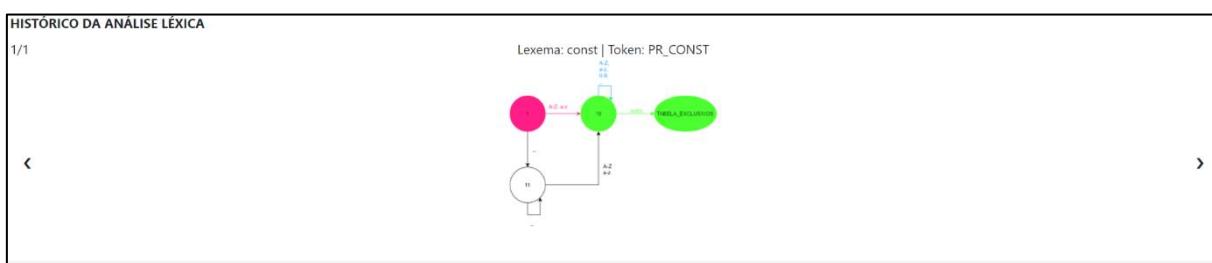
FIGURA 94 - Exemplo Prático Análise Léxica ''



FONTE: a própria autora

A figura 95 apresenta o autômato finito final no Histórico da Análise Léxica para o lexema 'const'.

FIGURA 95 - Exemplo Prático Histórico da Análise Léxica 'const'



FONTE: a própria autora

Ao digitar o caractere 'p' na caixa de texto a função da Análise Léxica é disparada duas vezes e obtém as variáveis do quadro 10. As funções da Tabela de Símbolos e Análise Sintática não são disparadas.

QUADRO 10 - Exemplo Prático Análise Léxica '' e 'p'

N	Variáveis Obtidas pela Análise Léxica					
	Caractere	Palavra	Estado inicial	Estado final	Palavra p/ repetir	Token
16	p	ESPAÇO	SEPARADOR	SEPARADOR		p
17	p			1	10	

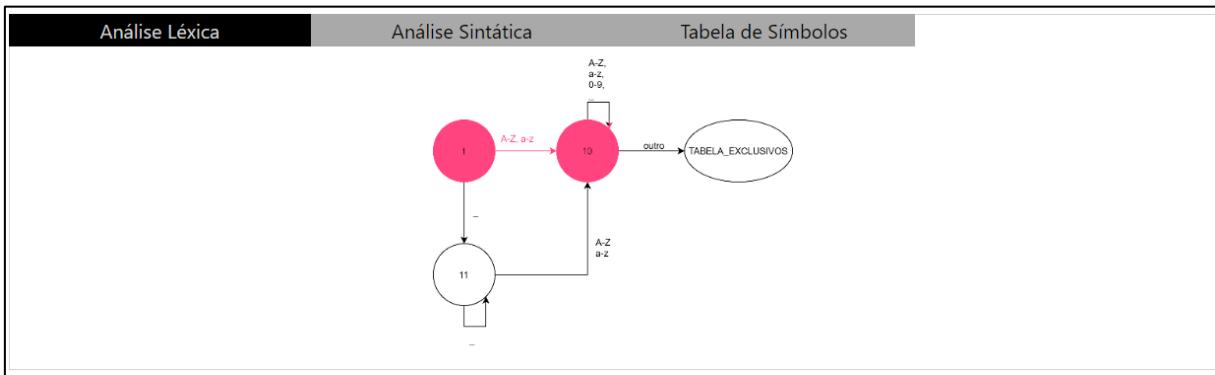
FONTE: a própria autora

A Análise Léxica dos caracteres de espaçamento apenas é finalizada quando a função recebe um caractere que não é um espaçador, por conta disso, a função primeiro

termina a análise do espaço e depois repete a função para o último caractere recebido, no caso o caractere 'p'.

A figura 96 apresenta a aba da Análise Léxica para o caractere 'p'.

FIGURA 96 - Exemplo Prático Análise Léxica 'p'



FONTE: a própria autora

Ao digitar o caractere 'i' na caixa de texto a função da Análise Léxica obtém as variáveis do quadro 11. As funções da Tabela de Símbolos e Análise Sintática não são disparadas.

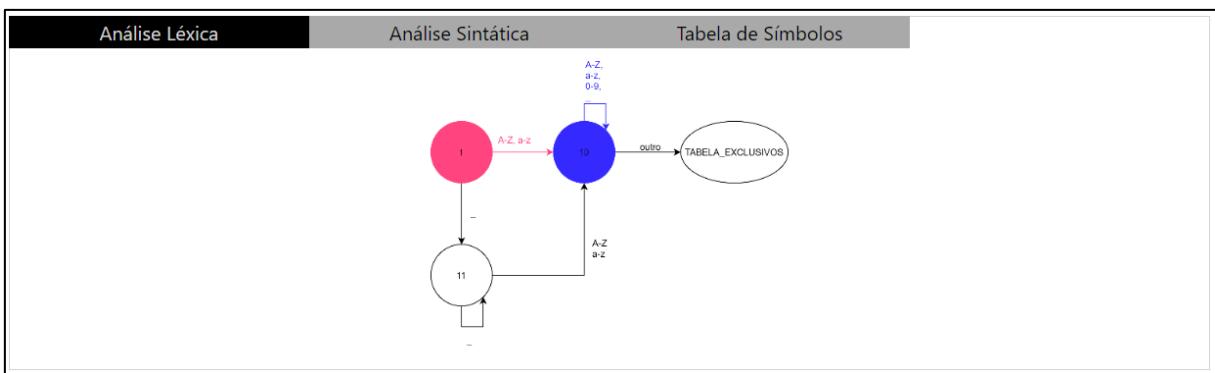
QUADRO 11 - Exemplo Prático Análise Léxica 'i'

N	Variáveis Obtidas pela Análise Léxica					
	Caractere	Palavra	Estado inicial	Estado final	Palavra p/ repetir	Token
18	i	p	10	10		

FONTE: a própria autora

A figura 97 apresenta a aba da Análise Léxica para o caractere 'i'.

FIGURA 97 - Exemplo Prático Análise Léxica 'i'



FONTE: a própria autora

Ao digitar o caractere ‘=’ na caixa de texto a função da Análise Léxica é disparada duas vezes e obtém as variáveis do quadro 12.

QUADRO 12 - Exemplo Prático Análise Léxica 'pi' e '='

N	Variáveis Obtidas pela Análise Léxica					
	Caractere	Palavra	Estado inicial	Estado final	Palavra p/ repetir	Token
19	=	pi		10	TABELA_EXCLUSIVOS	= IDENTIFICADOR
21	=			1	9	

FONTE: a própria autora

O lexema ‘pi’ apenas é finalizado quando a Análise Léxica recebe um caractere que não faz parte da tabela de exclusivos, por conta disso, a função primeiro termina a análise do lexema e depois repete a função para o último caractere recebido, no caso o caractere ‘=’.

A função da Tabela de Símbolos é disparada, pois o lexema ‘pi’ é um *token* identificador. A função da Análise Sintática é disparada uma vez para o lexema ‘pi’ antes do segundo disparo da Análise Léxica e obtém as variáveis do quadro 13.

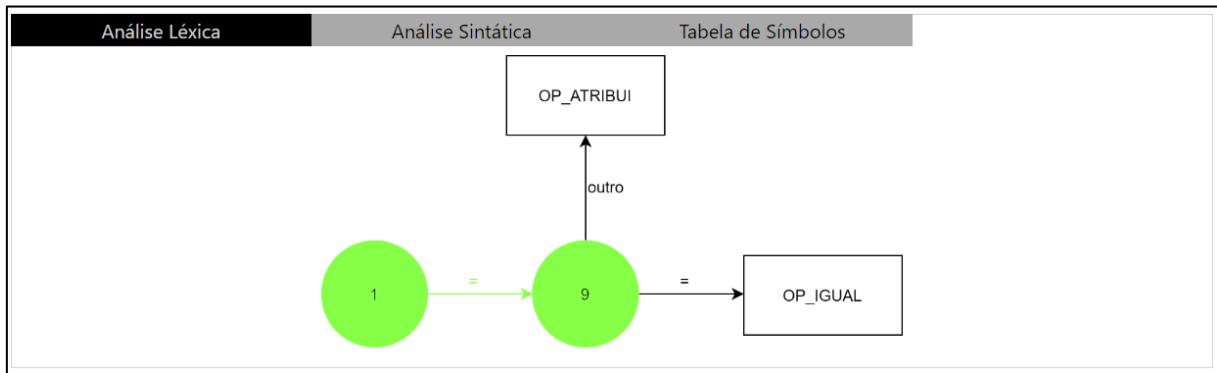
QUADRO 13 - Exemplo Prático Análise Sintática 'pi'

N	Variáveis Obtidas pela Análise Sintática				
	Funções	Passos	Função corrente	Passo corrente	Executou?
20	programa, programa/lista-decl, programa/lista-decl/ignore-decl, programa/lista-decl/ignore-decl/decl-const	2, 2, 1, 2		decl-const	2 Sim

FONTE: a própria autora

A figura 98 apresenta a aba da Análise Léxica para o caractere ‘=’.

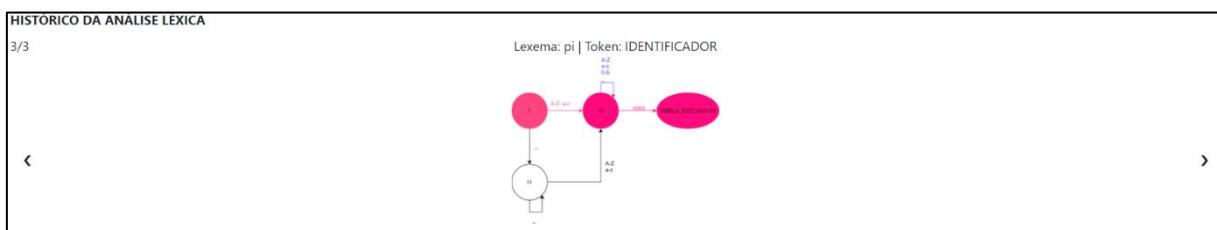
FIGURA 98 - Exemplo Prático Análise Léxica '='



FONTE: a própria autora

A figura 99 apresenta o autômato finito final no Histórico da Análise Léxica para o lexema ‘pi’.

FIGURA 99 - Exemplo Prático Histórico da Análise Léxica 'pi'



FONTE: a própria autora

Ao digitar o caractere ‘3’ na caixa de texto a função da Análise Léxica é disparada duas vezes e obtém as variáveis do quadro 14. As funções da Tabela de Símbolos e Análise Sintática não são disparadas.

#### QUADRO 14 - Exemplo Prático Análise Léxica '=' e '3'

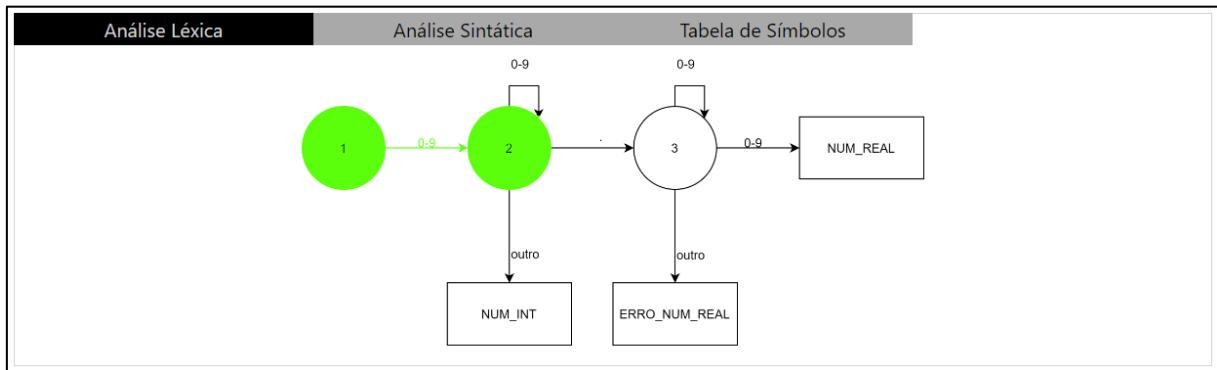
N	Variáveis Obtidas pela Análise Léxica					
	Caractere	Palavra	Estado inicial	Estado final	Palavra p/ repetir	Token
22	3	=		9	OP_ATRIBUI	
23	3			1	2	

FONTE: a própria autora

A Análise Léxica do caractere '=' apenas é finalizada quando a função recebe um caractere que não faz parte dos resultados que são inicializados por '=', por conta disso, a função primeiro termina a análise do '=' e depois repete a função para o último caractere recebido, no caso o caractere '3'.

A figura 100 apresenta a aba da Análise Léxica para o caractere '3'.

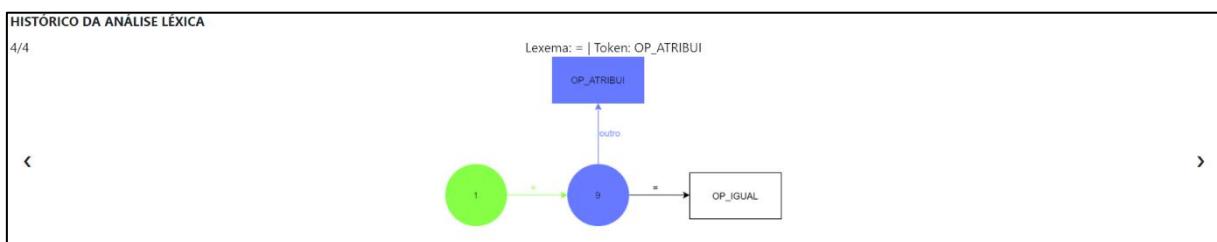
FIGURA 100 - Exemplo Prático Análise Léxica '3'



FONTE: a própria autora

A figura 101 apresenta o autômato finito final no Histórico da Análise Léxica para o lexema '='.

FIGURA 101 - Exemplo Prático Histórico da Análise Léxica '='



FONTE: a própria autora

Ao digitar a sequência de caracteres '.14' na caixa de texto a função da Análise Léxica obtém as variáveis do quadro 15. As funções da Tabela de Símbolos e Análise Sintática não são disparadas.

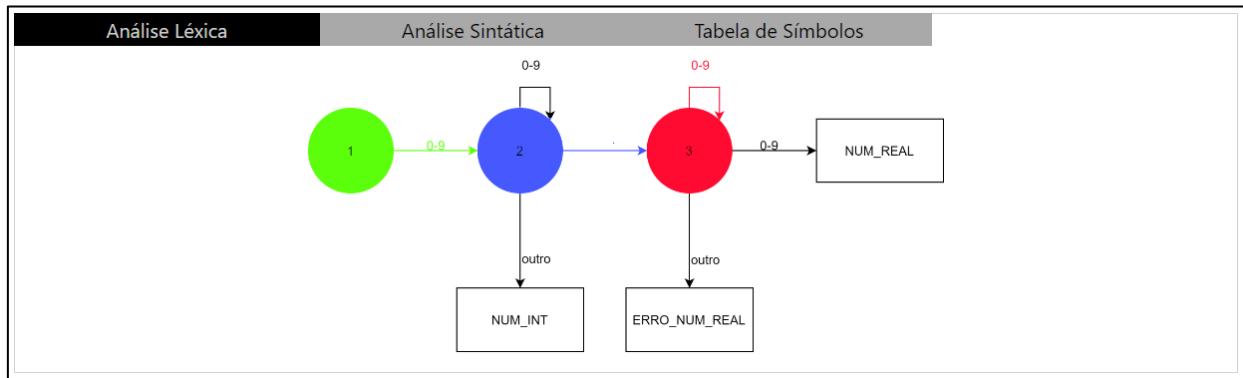
QUADRO 15 - Exemplo Prático Análise Léxica '.14'

N	Variáveis Obtidas pela Análise Léxica					
	Caractere	Palavra	Estado inicial	Estado final	Palavra p/ repetir	Token
24	.	3	2	3		
25	1	3.	3	3		
26	4	3.1	3	3		

FONTE: a própria autora

A figura 102 apresenta a aba da Análise Léxica para a sequência de caracteres ‘.14’.

FIGURA 102 - Exemplo Prático Análise Léxica '.14'



FONTE: a própria autora

Ao digitar o caractere ‘;’ na caixa de texto a função da Análise Léxica é disparada duas vezes e obtém as variáveis do quadro 16.

QUADRO 16 - Exemplo Prático Análise Léxica '3.14' e ';

N	Variáveis Obtidas pela Análise Léxica					
	Caractere	Palavra	Estado inicial	Estado final	Palavra p/ repetir	Token
27	;	3.14		3	NUM_REAL	;
30	;			1	SIN_PV	SIN_PV

FONTE: a própria autora

O lexema ‘3.14’ apenas é finalizado quando a Análise Léxica recebe um caractere que não faz parte dos números reais, por conta disso, a função primeiro termina a análise do lexema e depois repete a função para o último caractere recebido, no caso o caractere ‘;’.

A função da Tabela de Símbolos não é disparada e a função da Análise Sintática é disparada duas vezes para o lexema ‘3.14’ antes do segundo disparo da Análise Léxica e uma vez para o lexema ‘;’ depois do segundo disparo da Análise Léxica e obtém as variáveis do quadro 17.

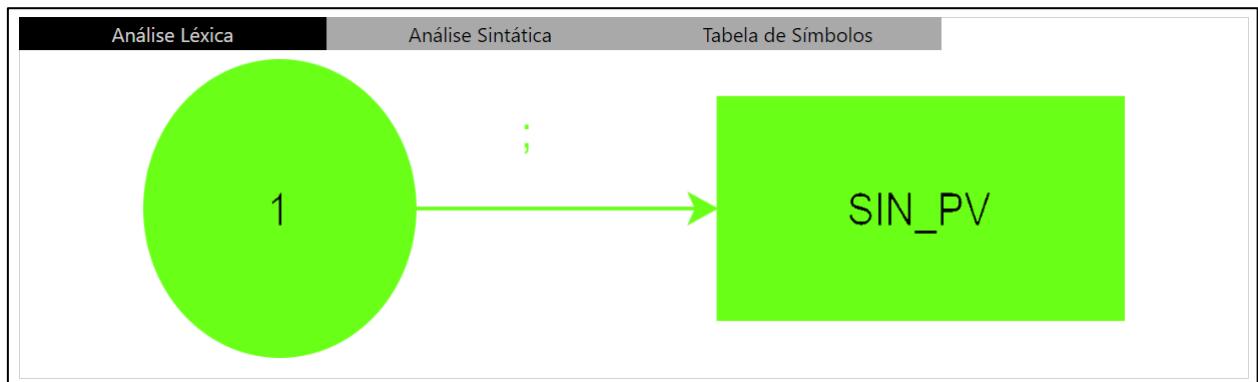
QUADRO 17 - Exemplo Prático Análise Sintática '3.14' e ';

N	Variáveis Obtidas pela Análise Sintática				
	Funções	Passos	Função corrente	Passo corrente	Executou?
28	programa, programa/lista-decl, programa/lista-decl/ignore-decl, programa/lista-decl/ignore-decl/decl-const	2, 2, 1, 4	decl-const	4	Sim
29	programa, programa/lista-decl, programa/lista-decl/ignore-decl, programa/lista-decl/ignore-decl/decl-const, programa/lista-decl/ignore-decl/decl-const/literal	2, 2, 1, 5, 1	literal	1	Sim
31	programa, programa/lista-decl, programa/lista-decl/ignore-decl, programa/lista-decl/ignore-decl/decl-const	2, 2, 1, 5	decl-const	5	Sim

FONTE: a própria autora

A figura 103 apresenta a aba da Análise Léxica para o caractere ';

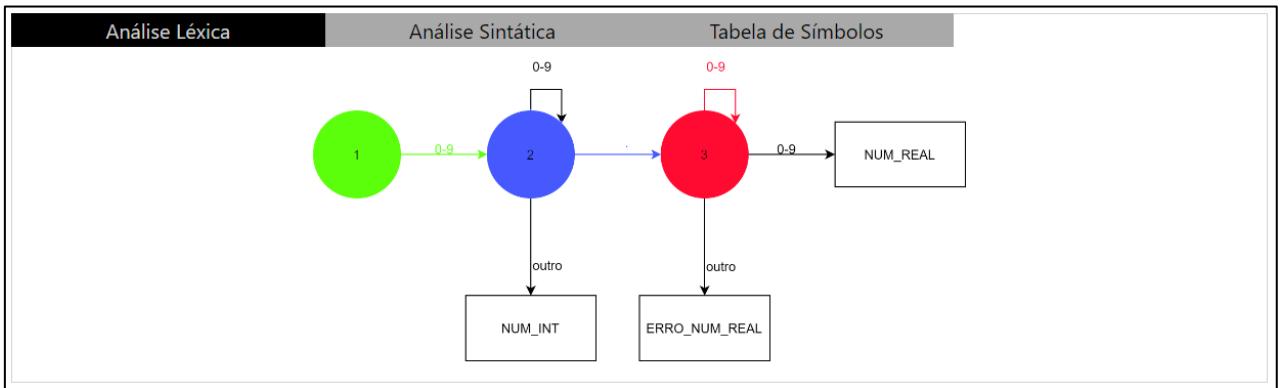
FIGURA 103 - Exemplo Prático Análise Léxica ';



FONTE: a própria autora

A figura 104 apresenta o autômato finito final no Histórico da Análise Léxica para o lexema '3.14'.

FIGURA 104 - Exemplo Prático Análise Léxica '3.14'



FONTE: a própria autora

A figura 105 apresenta como a aba da Análise Sintática ficou após todo o processo.

FIGURA 105 - Exemplo Prático Análise Sintática

Análise Léxica	Análise Sintática	Tabela de Símbolos
<pre> programa lista-decl   decl     decl-const       CONST → const       ID → pi       = → =       literal → 3.14       ; → ;     </pre>		

FONTE: a própria autora

A figura 106 apresenta como a aba da Tabela de Símbolos ficou após todo o processo.

FIGURA 106 - Exemplo Prático Tabela de Símbolos

Análise Léxica	Análise Sintática	Tabela de Símbolos
Token		Lexema
IDENTIFICADOR		pi

FONTE: a própria autora

## 5.6 CASOS DE TESTES

Para a homologação da aplicação VL-D+ foram criados diversos casos de testes, o objetivo dos testes é assegurar que a aplicação funcione sem erros para quaisquer interações do usuário. Os testes foram realizados no navegador *Google Chrome* em notebook e em um computador *desktop*.

Além dos casos de testes, todos os comandos e expressões da gramática D+ foram testados dentro das funções que contém um bloco de código. Os testes de exclusão de caracteres também foram massivamente realizados para vários cenários.

Vale ressaltar que apesar de haver vários cenários testados, é possível que existam erros na aplicação para cenários específicos.

### 5.6.1 Análise léxica

Para a Análise léxica foram realizados 4 casos de testes que englobam todos os cenários possíveis de interação com o usuário. A coluna de Ações a executar apresentam quais ações na aplicação devem ser realizados para alcançar o resultado esperado daquele caso de teste; a coluna Ok apresenta se a aplicação está operando corretamente para aquele caso de teste.

O quadro 18 apresenta o caso de teste para a análise de um lexema digitado. Nesse caso de teste foram testados pelo menos um lexema para cada autômato finito que pode ser gerado na aba da Análise Léxica, incluindo lexemas inválidos.

QUADRO 18 - Autômato Finito Corrente

Ações a executar	Resultado esperado	Ok?
<ol style="list-style-type: none"> <li>1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a></li> <li>2. Digitar um lexema válido</li> </ol>	O sistema deve apresentar na aba da Análise Léxica o autômato finito correspondente ao lexema digitado, colorindo o autômato para cada caractere digitado.	OK

FONTE: a própria autora

O quadro 19 apresenta o caso de teste para a análise de mais de um lexema digitado.

QUADRO 19 - Análise Léxica com Mais de um Lexema

Ações a executar	Resultado esperado	Ok?
<ol style="list-style-type: none"> <li>1. Acessar <a href="http://www.nomedosistema.com.br">www.nomedosistema.com.br</a></li> <li>2. Digitar dois lexemas válidos</li> </ol>	O sistema gera no histórico da Análise Léxica o autômato finito do primeiro lexema digitado e na aba da Análise Léxica gera o autômato finito correspondente ao lexema corrente	OK

FONTE: a própria autora

O quadro 20 apresenta o caso de teste para a análise de dois lexemas tendo o último lexema deletado.

QUADRO 20 - Análise Léxica Deletar Último Lexema

Ações a executar	Resultado esperado	Ok?
<ol style="list-style-type: none"> <li>1. Acessar <a href="http://www.nomedosistema.com.br">www.nomedosistema.com.br</a></li> <li>2. Digitar dois lexemas válidos</li> <li>3. Deletar o último lexema digitado</li> </ol>	O sistema deve remover o diagrama do lexema removido e transferir o diagrama do primeiro lexema do histórico para a aba da Análise Léxica	OK

FONTE: a própria autora

O quadro 21 apresenta o caso de teste para as setas do histórico da Análise Léxica.

QUADRO 21 - Setas do Histórico da Análise Léxica

Ações a executar	Resultado esperado	Ok?
<ol style="list-style-type: none"> <li>1. Acessar <a href="http://www.nomedosistema.com.br">www.nomedosistema.com.br</a></li> <li>2. Digitar três lexemas válidos</li> <li>3. Clicar na seta do histórico da Análise Léxica à esquerda</li> <li>4. Clicar na seta do histórico da Análise Léxica à direita</li> </ol>	O sistema deve mostrar na aba da Análise Léxica o diagrama correspondente ao lexema corrente e no histórico da Análise Léxica ao clicar na seta à esquerda deve mudar do diagrama do primeiro lexema digitado para o diagrama correspondente ao último lexema digitado e ao clicar na seta à direita deve voltar ao diagrama do primeiro lexema.	OK

FONTE: a própria autora

### 5.6.2 Análise sintática

Para a Análise sintática foram realizados 18 casos de testes que englobam vários cenários possíveis de interação com o usuário. A coluna de Ações a executar apresentam quais ações na aplicação devem ser realizados para alcançar o resultado esperado daquele caso de teste; a coluna Ok apresenta se a aplicação está operando corretamente para aquele caso de teste.

O quadro 22 apresenta o caso de teste para a declaração de constante sem erros léxicos ou sintáticos.

QUADRO 22 – Declaração de Constante

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a>	O sistema deve apresentar a árvore sintática correspondente.	OK
2. Digitar uma declaração de constante válida		

FONTE: a própria autora

O quadro 23 apresenta o caso de teste para a declaração de variável sem erros léxicos ou sintáticos.

QUADRO 23- Declaração de Variável

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a>	O sistema deve apresentar a árvore sintática correspondente.	OK
2. Digitar uma declaração de variável válida		

FONTE: a própria autora

O quadro 24 apresenta o caso de teste para a declaração de procedimento sem erros léxicos ou sintáticos.

QUADRO 24 - Declaração de Procedimento

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a>	O sistema deve apresentar a árvore sintática correspondente.	OK
2. Digitar uma declaração de procedimento válida.		

FONTE: a própria autora

O quadro 25 apresenta o caso de teste para a declaração de *main* sem erros léxicos ou sintáticos.

QUADRO 25 - Declaração de *main*

Ações a executar	Resultado esperado	Ok?
<ol style="list-style-type: none"> <li>1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a></li> <li>2. Digitar uma declaração de <i>main</i> válida.</li> </ol>	O sistema deve apresentar a árvore sintática correspondente.	OK

FONTE: a própria autora

O quadro 26 apresenta o caso de teste para a declaração de função sem erros léxicos ou sintáticos.

QUADRO 26 - Declaração de Função

Ações a executar	Resultado esperado	Ok?
<ol style="list-style-type: none"> <li>1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a></li> <li>2. Digitar uma declaração de função válida.</li> </ol>	O sistema deve apresentar a árvore sintática correspondente.	OK

FONTE: a própria autora

O quadro 27 apresenta o caso de teste para a declaração de constante com erro no identificador.

QUADRO 27 - Declaração de Constante (ERRO ID)

Ações a executar	Resultado esperado	Ok?
<ol style="list-style-type: none"> <li>1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a></li> <li>2. Digitar uma declaração de constante com erro no ID.</li> </ol>	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 28 apresenta o caso de teste para a declaração de constante com erro no sinal de igualdade.

QUADRO 28 - Declaração de Constante (ERRO '=')

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de constante com erro no '='.	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 29 apresenta o caso de teste para a declaração de constante com erro no literal.

QUADRO 29 - Declaração de Constante (ERRO LITERAL)

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de constante com erro literal.	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 30 apresenta o caso de teste para a declaração de constante com erro no sinal de ponto e vírgula.

QUADRO 30 - Declaração de Constante (ERRO ';')

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de constante com erro no ';	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 31 apresenta o caso de teste para a declaração de variável com erro na especificação do tipo.

QUADRO 31 - Declaração de Variável (ERRO espec-tipo)

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de variável com erro no espec-tipo.	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 32 apresenta o caso de teste para a declaração de variável com erro na lista de variáveis.

QUADRO 32 - Declaração de Variável (ERRO lista-var)

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de variável com erro de lista-var.	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 33 apresenta o caso de teste para a declaração de variável com erro no sinal de ponto e vírgula.

QUADRO 33 - Declaração de Variável (ERRO ‘;’)

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de variável com erro no ‘;’.	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 34 apresenta o caso de teste para a declaração de procedimento com erro na especificação de tipo.

QUADRO 34 - Declaração de Procedimento (ERRO espec-tipo)

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de procedimento com erro de espec-tipo.	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 35 apresenta o caso de teste para a declaração de procedimento com erro no identificador.

QUADRO 35 - Declaração de Procedimento (ERRO ID)

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de procedimento com erro de ID.	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 36 apresenta o caso de teste para a declaração de procedimento com erro no sinal de abre parênteses.

QUADRO 36 - Declaração de Procedimento (ERRO ‘(‘ )

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de procedimento com erro no ‘(‘ .	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 37 apresenta o caso de teste para a declaração de procedimento com erro no sinal de fecha parênteses.

QUADRO 37 - Declaração de Procedimento (ERRO ‘)’ )

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de procedimento com erro no ‘)’ .	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 38 apresenta o caso de teste para a declaração de procedimento com erro no bloco.

QUADRO 38 - Declaração de Procedimento (ERRO bloco)

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de procedimento com erro no bloco.	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

O quadro 39 apresenta o caso de teste para a declaração de procedimento com erro no *endsub*.

QUADRO 39 - Declaração de Procedimento (ERRO endsub)

Ações a executar	Resultado esperado	Ok?
1. Acessar <a href="http://www.nomodosistema.com.br">www.nomodosistema.com.br</a> 2. Digitar uma declaração de procedimento com erro de endsub.	O sistema deve reportar o erro na aba da análise sintática e na aba de erros.	OK

FONTE: a própria autora

## 6 RESULTADOS E DISCUSSÕES

Para obter os resultados relativos à ferramenta VL-D+ foi elaborado um questionário de 12 questões, divididas em 3 seções: a primeira seção é relativa aos dados pessoais da pessoa, a segunda seção é sobre a experiência da pessoa em relação à disciplina de Compiladores e a terceira seção é sobre a experiência da pessoa em relação à aplicação VL-D+.

O questionário foi criado no *Google Forms* e distribuído aos discentes que estavam cursando ou que já cursaram a disciplina de Compiladores na Universidade Tuiuti do Paraná.

Para acessar a aplicação bastava os discentes que receberam o questionário abrirem o *link* da aplicação em um computador *desktop* ou notebook com acesso à *internet*, preferencialmente no navegador *Google Chrome* onde a aplicação foi homologada.

### 6.1 ANÁLISE DOS RESULTADOS

O formulário distribuído foi respondido por apenas 9 discentes, os resultados apresentados a seguir apresentam os gráficos gerados pelo *Google Forms* a partir das respostas obtidas.

O gráfico 1 apresenta o grau de dificuldade de aprendizado das pessoas na disciplina de Compiladores. A partir do gráfico é possível afirmar que todas as pessoas tiveram dificuldade na disciplina.

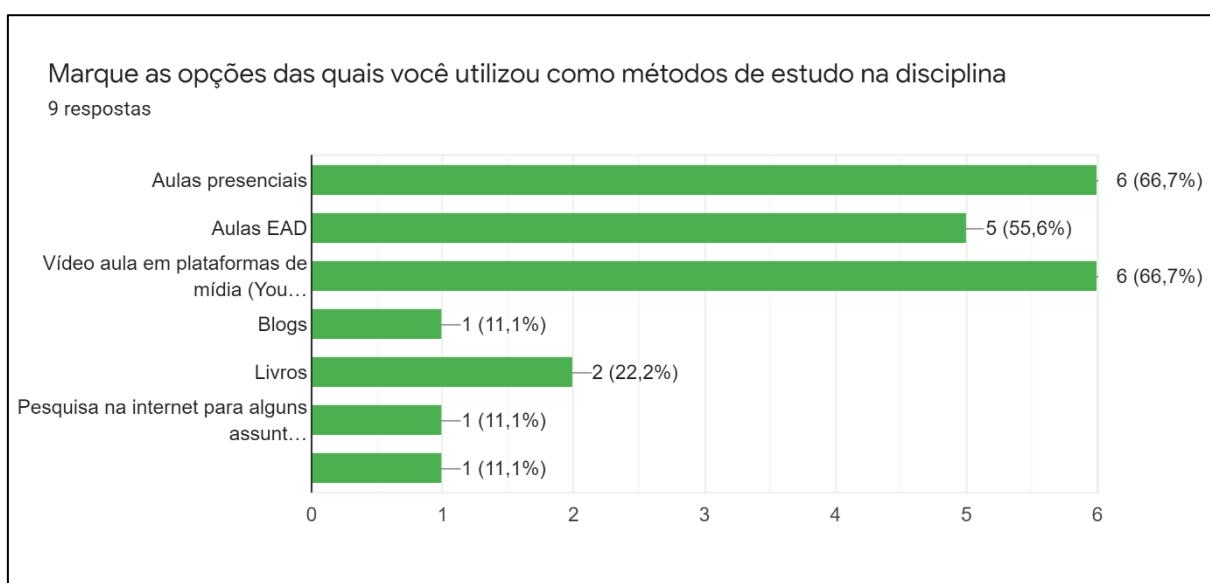
### GRÁFICO 1 - Gráfico de Dificuldade de Aprendizado da Disciplina



FONTE: a própria autora

O gráfico 2 apresenta quais métodos de estudo as pessoas utilizaram para a disciplina de Compiladores. Analisando o gráfico é possível observar que algumas pessoas optaram por buscar auxílio em outros meios de estudo fora das aulas da universidade.

### GRÁFICO 2 - Gráfico de Métodos de Estudo

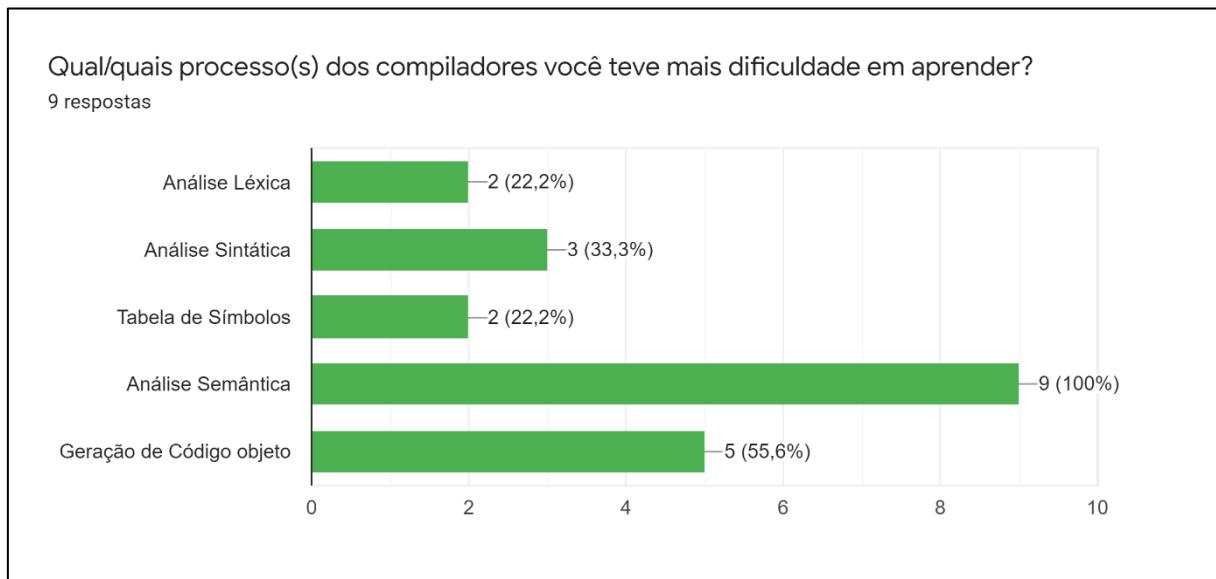


FONTE: a própria autora

O gráfico 3 apresenta quais processos dos compiladores as pessoas tiveram mais dificuldade em aprender. Analisando os resultados pode-se observar que em todos os

processos do Compilador as pessoas tiverem dificuldades e no processo da Análise Semântica todos os alunos tiveram dificuldades.

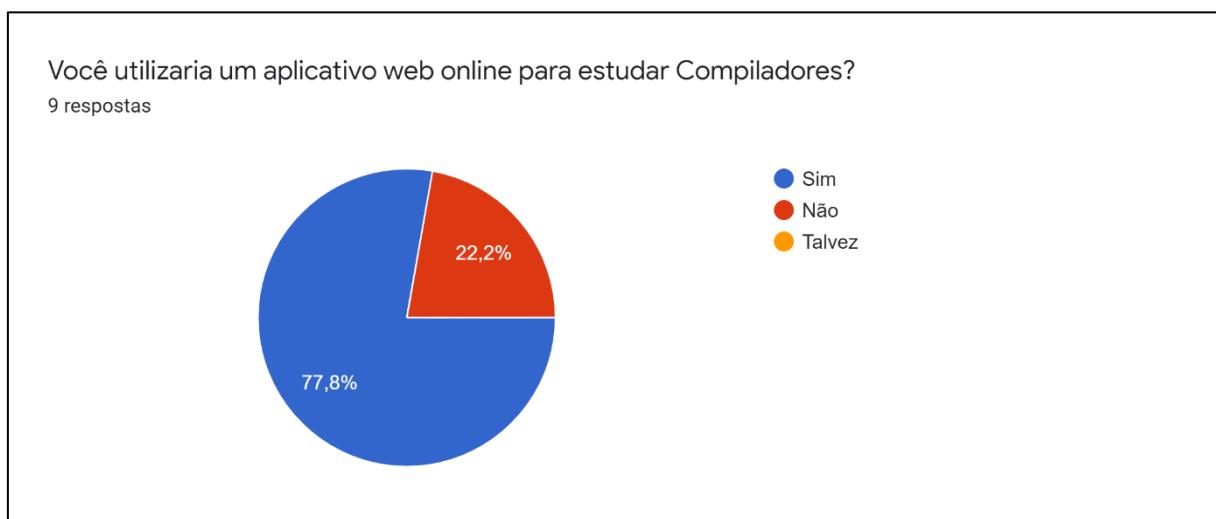
GRÁFICO 3 - Gráfico de Dificuldade no Aprendizado da Disciplina por Processo



FONTE: a própria autora

O gráfico 4 apresenta se as pessoas utilizariam um aplicativo *web* para estudar a disciplina de Compiladores. Analisando as respostas 77,8% das pessoas utilizariam e 22,2% não utilizariam.

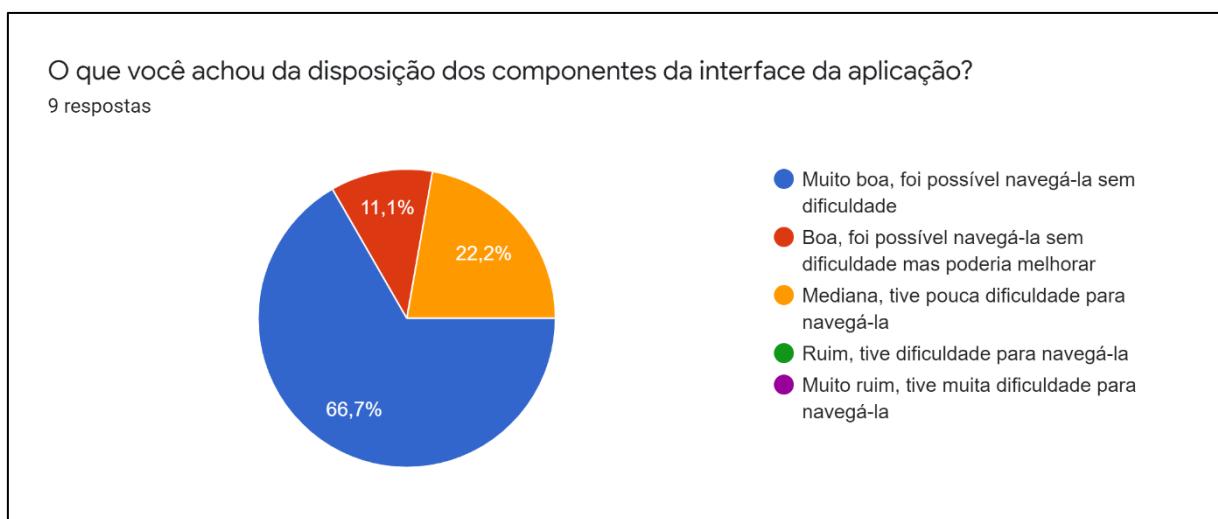
GRÁFICO 4 - Gráfico de Utilização de Aplicativo



FONTE: a própria autora

O gráfico 5 apresenta se a experiência do usuário em relação à interação com a interface da aplicação VL-D+ foi satisfatória. Analisando o gráfico 66,7% das pessoas tiveram um alto nível de satisfação e não tiveram dificuldades em utilizar a aplicação, 11,1% das pessoas não tiveram dificuldades, mas acham que a interface poderia melhorar e 22,2% tiveram um pouco de dificuldade em utilizar a aplicação.

GRÁFICO 5 - Gráfico de Satisfação VL-D+



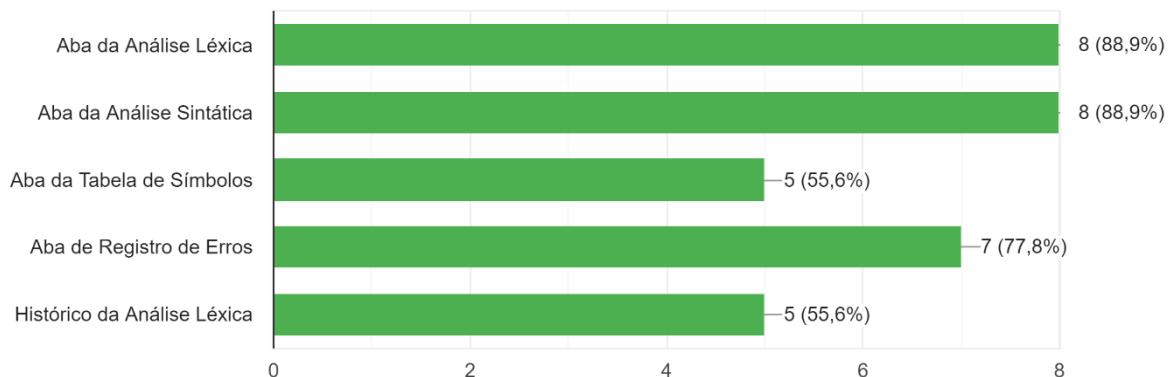
FONTE: a própria autora

O gráfico 6 apresenta quais itens da página inicial da aplicação as pessoas acham que atendem o propósito de ajudar os discentes no aprendizado da disciplina de Compiladores. Analisando o gráfico, a maioria das pessoas (88,9%) acham que as abas da Análise Léxica e Sintática atendem o propósito, 77,8% das pessoas acham que a aba de Registo de Erros atende o propósito e apenas 55,6% das pessoas acham que a aba da Tabela de Símbolos e o Histórico da Análise Léxica atendem o propósito. Esses resultados apontam que a Tabela de Símbolos e o Histórico da Análise Léxica deveriam ser refeitos para terem um propósito maior na aplicação.

### GRÁFICO 6 - Gráfico Propósito VL-D+

Sobre a página principal da aplicação, marque quais itens você acha que atendem o propósito de ajudar no aprendizado da disciplina de compiladores

9 respostas



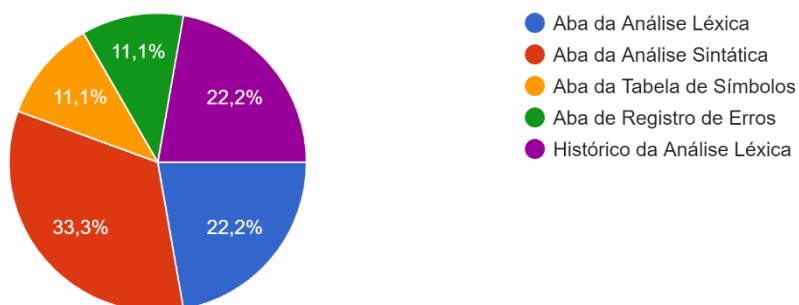
FONTE: a própria autora

O gráfico 7 apresenta qual item da página inicial da interface as pessoas mais gostaram. Observando o gráfico, os resultados ficaram bem divididos e todos os itens foram marcados por pelo menos uma pessoa. 33,3% das pessoas gostaram mais da aba da Análise Sintática, 22,2% gostaram mais da aba da Análise Léxica, 22,2% gostaram mais do Histórico da Análise Léxica, 11,1% gostaram mais da aba de Registro de Erros e 11,1% gostaram mais da aba da Tabela de Símbolos.

### GRÁFICO 7 - Gráfico Relevância VL-D+

Sobre a página principal da aplicação, marque qual item você mais gostou

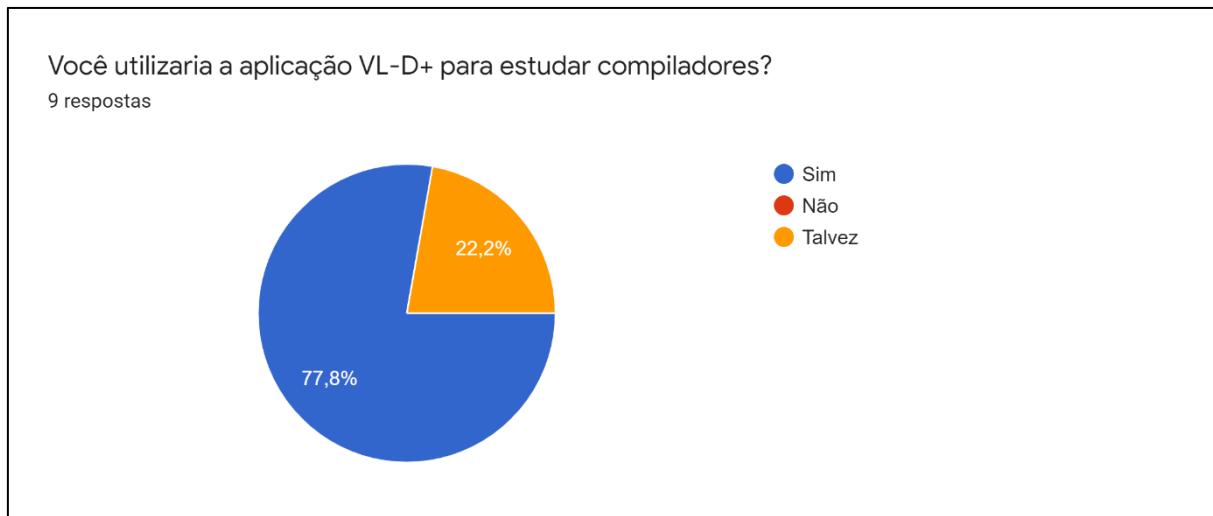
9 respostas



FONTE: a própria autora

O gráfico 8 é similar ao gráfico 4 e apresenta que 77,8% das pessoas usariam a aplicação para estudar a disciplina de Compiladores e 22,2% das pessoas talvez usariam. Em relação ao gráfico 4 o 22,2% das pessoas que marcaram que não usariam uma aplicação *web* para estudar a disciplina agora marcaram que talvez usariam a aplicação VL-D+.

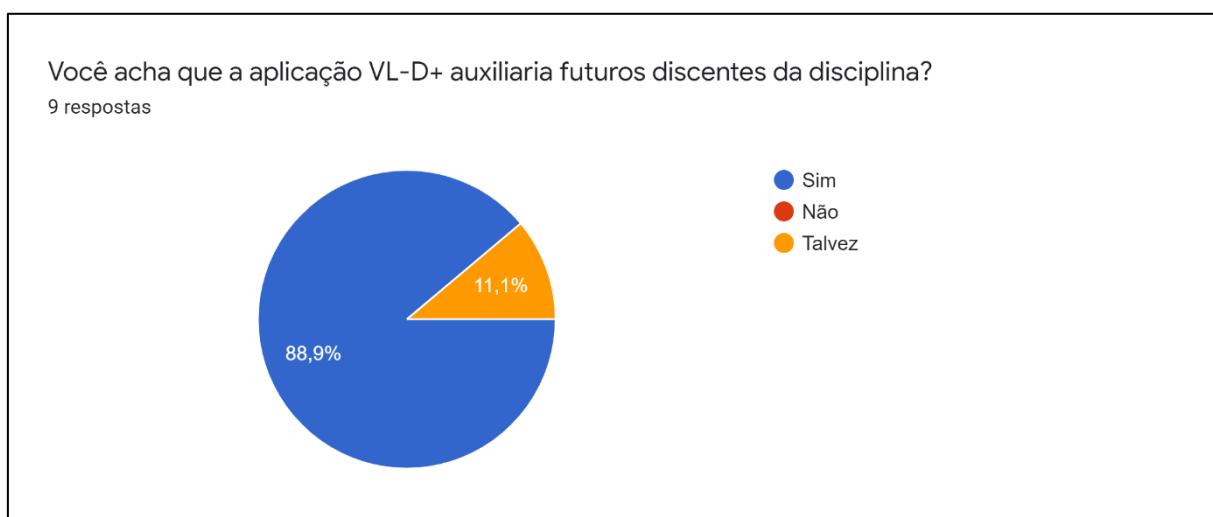
GRÁFICO 8 - Gráfico Utilização VL-D+



FONTE: a própria autora

O gráfico 9 apresenta se as pessoas acham que a aplicação VL-D+ auxiliaria futuros discentes no aprendizado da disciplina de Compiladores. Analisando o gráfico, a maioria das pessoas acreditam que a aplicação auxiliaria e uma pessoa não tem certeza.

GRÁFICO 9 - Gráfico Auxílio Futuro VL-D+



FONTE: a própria autora

Além das perguntas de múltiplas escolhas, foi questionado às pessoas se a aplicação poderia melhorar em algum aspecto, dentre as respostas foram destacadas as seguintes sugestões:

- A aplicação poderia disponibilizar exemplos de códigos na linguagem D+ para que o usuário assimile a linguagem com mais facilidade;
- A aplicação poderia possibilitar que o usuário importasse múltiplos arquivos contendo códigos na linguagem D+ para validação;
- A caixa de texto e as abas da interface da página inicial da aplicação poderiam ser dispostos sem limitação de tamanho máximo, possibilitando a melhor visualização do código e dos resultados das abas;
- As páginas das documentações poderiam por padrão abrir em outra guia ao serem clicadas no menu, facilitando a navegação do usuário entre a página inicial e as páginas de documentações;
- A página da documentação da Gramática D+ poderia disponibilizar a opção de baixar o documento em PDF;
- A aplicação poderia ser responsiva, possibilitando a utilização da aplicação tanto em um dispositivo *desktop* quanto em um dispositivo *mobile*;
- A aplicação poderia ter melhor performance;
- A aba de Erros na página inicial poderia mostrar em qual linha o erro se encontra.

Finalmente, o questionário perguntou opcionalmente às pessoas se haviam comentários, dentre as respostas não houve nenhum comentário negativo e uma das pessoas respondeu que a aplicação é boa no sentido da facilidade da utilização sem a necessidade de baixar algum programa.

De forma geral, os resultados obtidos foram promissores e apresentam que a aplicação pode ser aperfeiçoada, mas já possui conteúdo suficiente para auxiliar o aprendizado de futuros discentes da disciplina de Compiladores.

## 7 CONCLUSÃO

Tendo em vista o exposto, é possível identificar que os discentes da disciplina de Compiladores possuem dificuldades no aprendizado, e fora dos centros de educação existem poucos recursos disponibilizados para o auxílio do discente na compreensão do conteúdo da disciplina.

Conforme apresentado no projeto, já existem ferramentas que reduzem essa problemática, porém todas trazem resultados estáticos das fases do compilador ao usuário e todas necessitam da instalação da ferramenta em um computador. Portanto, a visualização dos resultados se torna mais cansativa ao usuário e a instalação do produto às vezes pode ser algo inviável ou inatingível, tendo em vista que o instalador pode precisar de um sistema operacional específico.

Como alternativa, a ferramenta VL-D+ aborda a tabela de símbolos e as fases da análise léxica e sintática dos compiladores numa interface *web* amigável e intuitiva, que traz os resultados da compilação do código digitado pelo usuário em tempo real, podendo assim ser mais interessante para o usuário e necessitando apenas de um dispositivo *desktop* que consiga acessar a ferramenta através da internet.

A ferramenta VL-D+ foi desenvolvida utilizando o ASP.Net Core, tendo a maioria das funções construídas na linguagem *JavaScript*, pois esta linguagem combinada ao *cshtml* permite o disparo de funções baseadas na interação do usuário e isto apresenta os resultados da interface de forma dinâmica e em tempo real.

Após o desenvolvimento da ferramenta, foi distribuído um formulário de perguntas para discentes que já cursaram ou que estavam cursando a disciplina de Compiladores. O formulário teve o propósito de obter os resultados quanto à aplicabilidade e efetividade da ferramenta construída.

De forma geral, a aplicação já possui conteúdo suficiente para auxiliar os discentes a estudarem a disciplina de Compiladores, porém ainda há espaço para melhorias no que já foi desenvolvido e a aplicação poderia ser expandida para cobrir todas as funções que o compilador realiza, além da Análise Léxica e Análise Sintática já desenvolvidas.

## REFERÊNCIAS

AHO, Alfred V.; SETHI Ravi; ULLMAN Jeffrey D. *Compiladores, Princípios, Técnicas e Ferramentas*. 1. ed. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora S.A., 1995.

AHO, Alfred V. et al. *Compiladores: princípios, técnicas e ferramentas*. 2. ed. São Paulo: Pearson Addison-Wesley, 2008.

BACKES, Jerônimo; DAHMER Alessandra. *C-gen – Ambiente Educacional para Geração de Compiladores*. Santa Cruz do Sul: Universidade de Santa Cruz do Sul, 2006.

BENYON, David. *Interação humano-computador*. 2. ed. São Paulo: Pearson Prentice Hall, 2011.

COSTA, Kelton A. P.; SILVA, Luis Alexandre; BRITO, Talita Pagani. *Auxílio no ensino em compiladores: software simulador como ferramenta de apoio na área de compiladores*. 2. ed. São Paulo: Simpósio Internacional de Educação, 2008.

FOLEISSL, J. H. et al. *SCC: Um Compilador C como Ferramenta de Ensino de Compiladores*. São Paulo: Workshop sobre Educação em Arquitetura de Computadores – WEAC, 2009.

FURLAN, Diógenes Cogo. *Linguagem D+*. Curitiba, 2019.

GARRETT, Jesse James. *Os Elementos da Experiência do Usuário*. 2000. Disponível em: <[http://www.jjjg.net/elements/translations/elements\\_pt.pdf](http://www.jjjg.net/elements/translations/elements_pt.pdf)>. Acesso em: 08 mai. 2020.

MDN web docs, Mozilla. *Canvas*. 2020. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Web/HTML/Canvas>>. Acesso em: 22 mai. 2020.

MDN web docs, Mozilla. *Sobre JavaScript*. 2019. Disponível em: <[https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About\\_JavaScript](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About_JavaScript)>. Acesso em: 22 mai. 2020.

MICROSOFT. *'Início Rápido: Criar uma instância de serviço do Gerenciamento de API do Azure usando o portal do Azure.* 2020. Disponível em: <<https://docs.microsoft.com/pt-br/azure/api-management/get-started-create-service-instance>>. Acesso em 21 nov. 2020.

MICROSOFT. *Documentação do Visual Studio.* Entre 2019 e 2020. Disponível em: <<https://docs.microsoft.com/pt-br/visualstudio/windows/?view=vs-2019>>. Acesso em: 14 mai. 2020.

NETO, João José. *Introdução à compilação.* 2. ed. Rio de Janeiro: Elsevier, 2016.

NC STATE UNIVERSITY, The Center for Universal Design. *The Principles of Universal Design.* 2. ed. 1997. Disponível em <[https://projects.ncsu.edu/ncsu/design/cud/about\\_ud/udprinciplestext.htm](https://projects.ncsu.edu/ncsu/design/cud/about_ud/udprinciplestext.htm)>. Acesso em: 07 mai. 2020.

RIBEIRO, Gabriel Pinto. *Interpretador da linguagem D+.* Trabalho de conclusão de curso (Bacharelado de Ciência da Computação) – Faculdade de Ciências Exatas, Universidade Tuiuti do Paraná, Curitiba, 2019.

SANTOS, Pedro Reis. LANGLOIS Thibault. *Compiladores: da teoria à prática.* 1. ed. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora Ltda., 2018.

SCHNEIDER, Carlos Sérgio; PASSERINO, Liliana Maria; OLIVEIRA, Ricardo Ferreira. *Compilador Educativo VERTO: ambiente para aprendizagem de compiladores.* 3. ed. Rio Grande do Sul: Centro Interdisciplinar de Novas Tecnologias na Educação (CINTED) - Universidade Federal do Rio Grande do Sul (UFRGS), 2005.

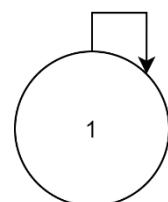
SOBRAL, Wilma Sirlange. *Design de interfaces: introdução.* 1. ed. São Paulo: Érica, 2019.

W3SCHOOLS. *HTML Canvas Graphics.* Disponível em: <[https://www.w3schools.com/html/html5\\_canvas.asp](https://www.w3schools.com/html/html5_canvas.asp)>. Acesso em: 25 mai. 2020.

**APÊNDICE A – AUTÔMATOS DA LINGUAGEM D+**

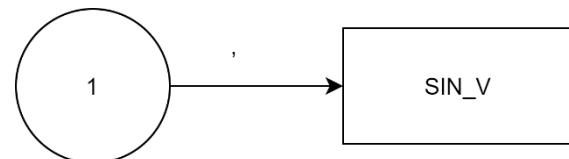
FIGURA 107 - Autômato para separadores (estado 1)

espaço, \r, \n, \t



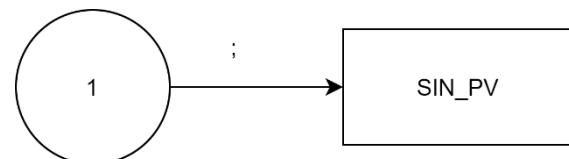
FONTE: a própria autora

FIGURA 108 - Autômato para a vírgula (estado 1)



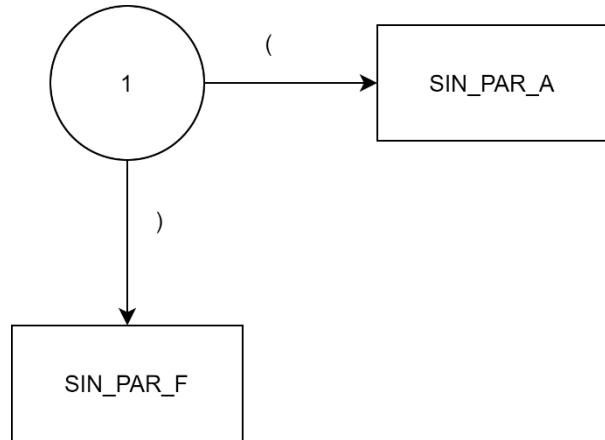
FONTE: a própria autora

FIGURA 109 - Autômato para o ponto e vírgula (estado 1)



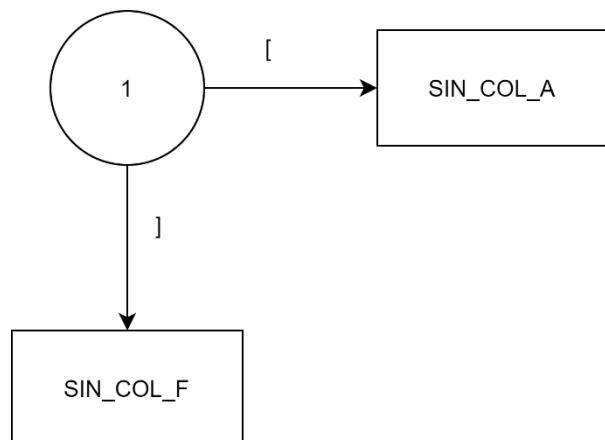
FONTE: a própria autora

FIGURA 110 - Autômato para os parênteses (estado 1)



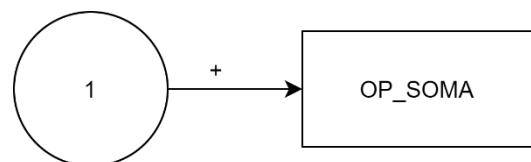
FONTE: a própria autora

FIGURA 111 - Autômato para os colchetes (estado 1)



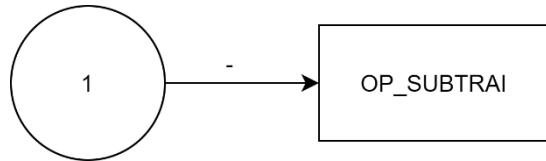
FONTE: a própria autora

FIGURA 112 - Autômato para o operador de adição (estado 1)



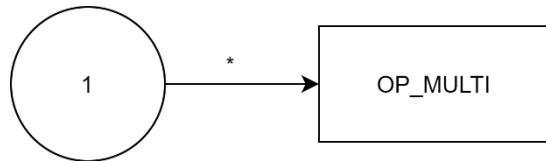
FONTE: a própria autora

FIGURA 113 - Autômato para o operador de subtração (estado 1)



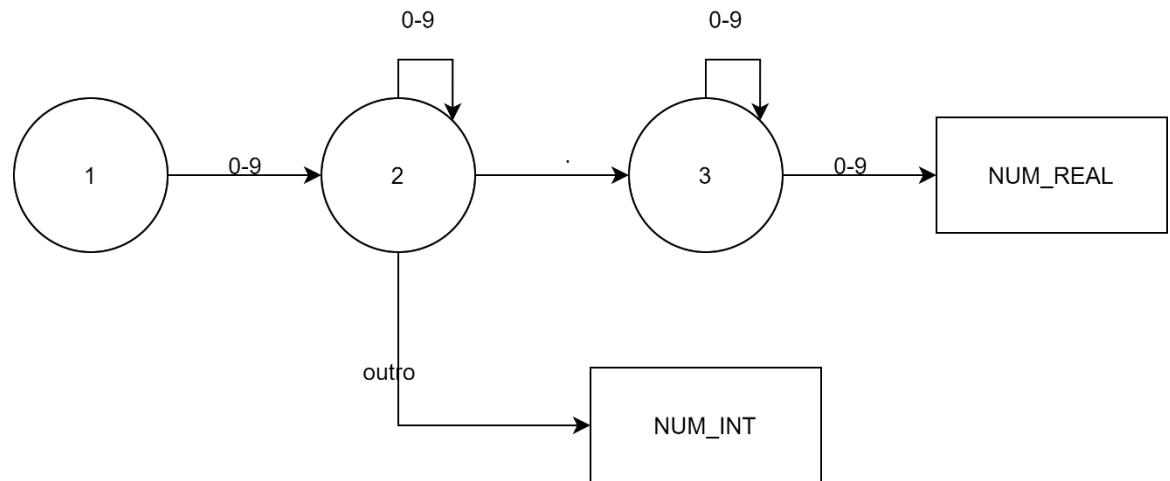
FONTE: a própria autora

FIGURA 114 - Autômato para o operador de multiplicação (estado 1)



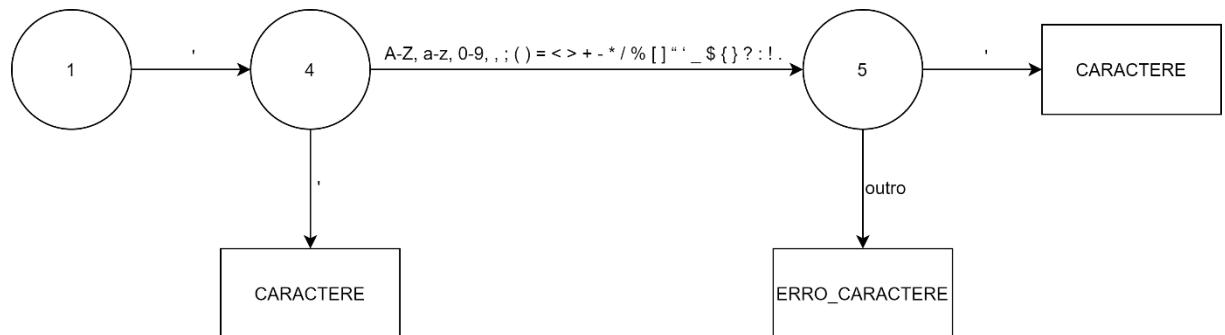
FONTE: a própria autora

FIGURA 115 - Autômato para números (estado 2)



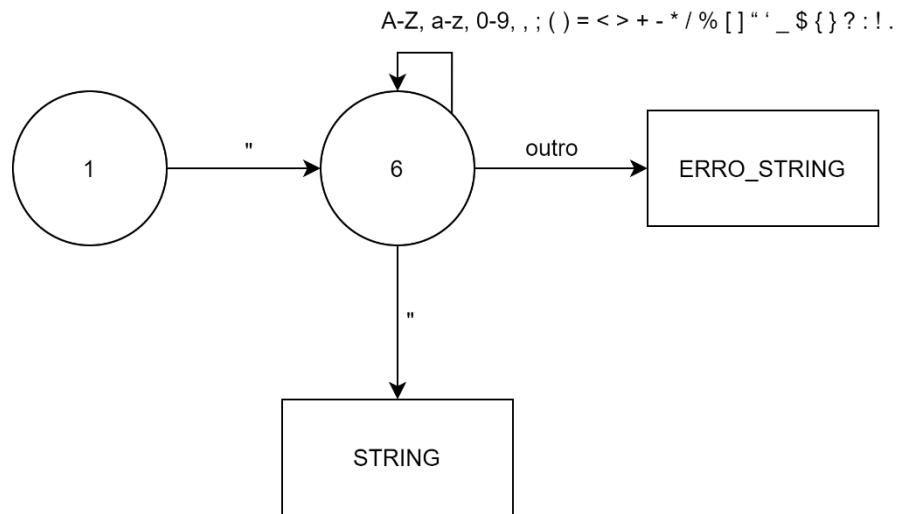
FONTE: a própria autora

FIGURA 116 - Autômato para caractere (estado 4)



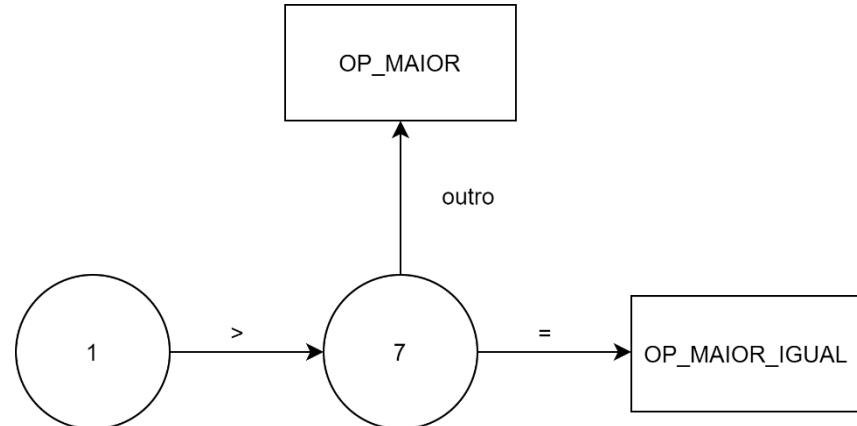
FONTE: a própria autora

FIGURA 117 - Autômato para *string* (estado 6)



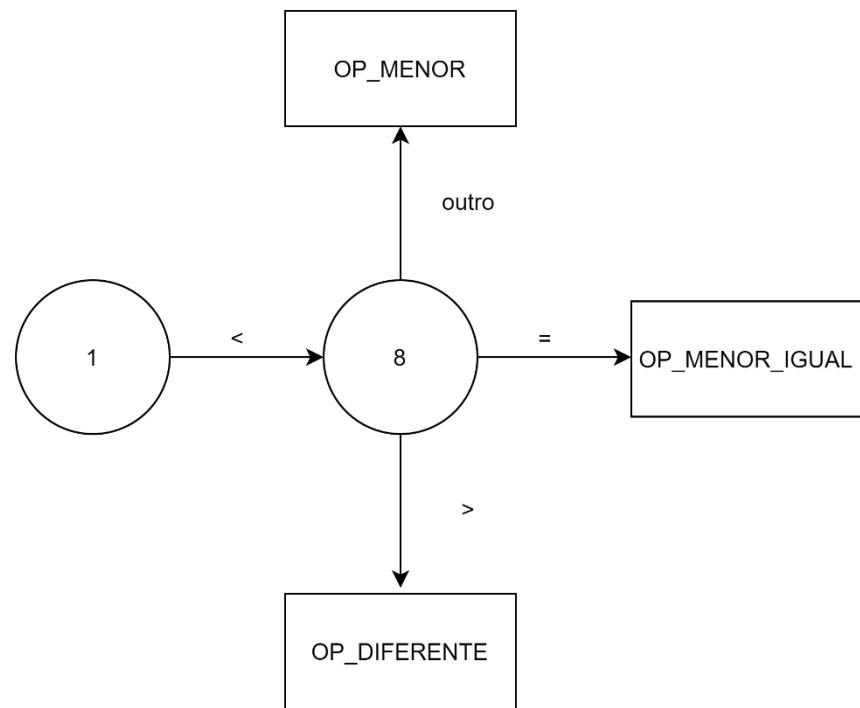
FONTE: a própria autora

FIGURA 118 - Autômato para o operador maior '>' (estado 7)



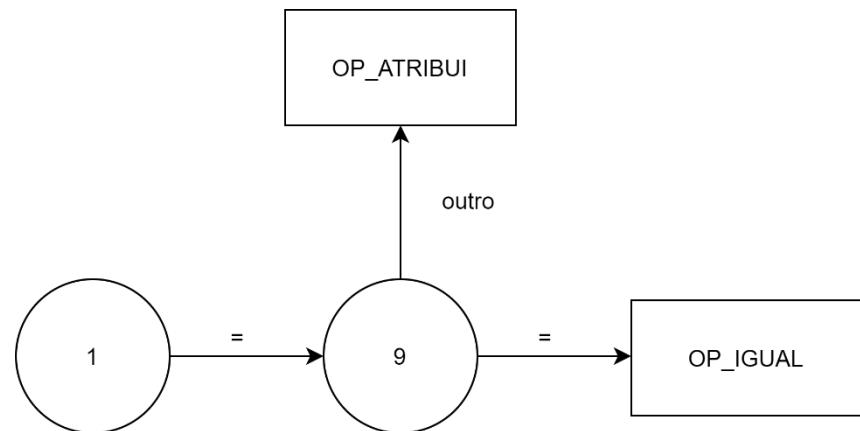
FONTE: a própria autora

FIGURA 119 - Autômato para o operador menor '&lt;' (estado 8)



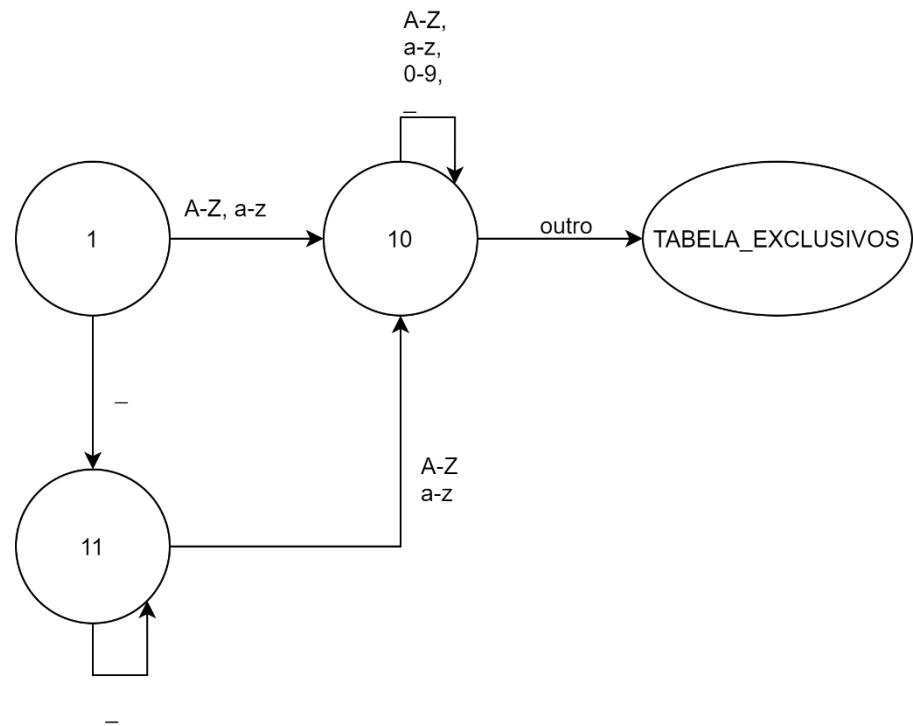
FONTE: a própria autora

FIGURA 120 - Autômato para o operador igual '=' (estado 9)



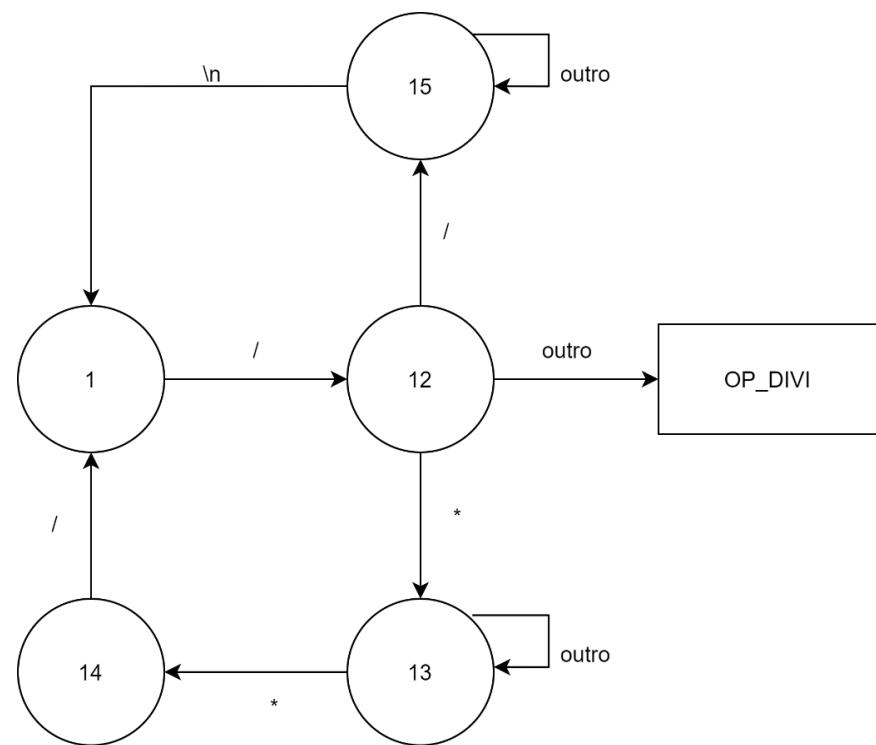
FONTE: a própria autora

FIGURA 121 - Autômato para valores exclusivos (estado 10)



FONTE: a própria autora

FIGURA 122 - Autômato para divisão e comentários (estado 12)



FONTE: a própria autora

## APÊNDICE B – FORMULÁRIO DE QUESTÕES SOBRE A EXPERIÊNCIA DO USUÁRIO

FIGURA 123 - Formulário Introdução

A interface de usuário de um formulário intitulado "Experiência do Usuário VL-D+". O formulário é apresentado em um fundo cinza com uma barra verde no topo. O título "Experiência do Usuário VL-D+" está centralizado na barra verde. O conteúdo principal é contido em uma caixa com fundo branco e bordas cinza. O texto no formulário é o seguinte:

Olá, meu nome é Sabrina Nawcki, aluna do curso de Bacharelado de Ciência da Computação. Meu TCC é relacionado à disciplina de Compiladores e tem o objetivo de auxiliar os discentes no aprendizado.

A aplicação desenvolvida está disponível no link <https://learncompiler20201105190105.azurewebsites.net/> até o início de dezembro/2020.

Para que a aplicação funcione corretamente deve ser utilizada em um computador ou notebook e de preferência utilizando o navegador Google Chrome.

Este formulário deve ser respondido de acordo com a sua experiência de uso na aplicação.

FONTE: a própria autora

FIGURA 124 - Formulário Questão 1

A interface de usuário de um formulário intitulado "Perguntas sobre você". A barra superior é verde com o título "Perguntas sobre você". O formulário contém o seguinte texto:

As próximas perguntas são pessoais e serão usadas para a geração de gráficos mais precisos.

Qual a sua idade? \*

Sua resposta

FONTE: a própria autora

FIGURA 125 - Formulário Questão 2

**Perguntas sobre a disciplina de Compiladores**

As próximas perguntas são pessoais sobre a sua experiência em relação à disciplina e serão usadas para a geração de gráficos mais precisos.

Qual é o seu grau de dificuldade de aprendizado na disciplina? \*

Alto, tenho/tive muita dificuldade

Médio, tenho/tive um pouco de dificuldade

Baixo, não tenho/tive dificuldade

FONTE: a própria autora

FIGURA 126 - Formulário Questão 3

Marque as opções das quais você utilizou como métodos de estudo na disciplina

Aulas presenciais

Aulas EAD

Vídeo aula em plataformas de mídia (YouTube, Udemy, Pluralsight)

Blogs

Livros

Outro: \_\_\_\_\_

FONTE: a própria autora

FIGURA 127 - Formulário Questão 4

Qual/quais processo(s) dos compiladores você teve mais dificuldade em aprender?

- Análise Léxica
- Análise Sintática
- Tabela de Símbolos
- Análise Semântica
- Geração de Código objeto

FONTE: a própria autora

FIGURA 128 - Formulário Questão 5

Você utilizaria um aplicativo web online para estudar Compiladores?

A internet dispõe vários sites educativos que mostram o passo a passo de algum processo, um exemplo é os sites de calculadora de matrizes online que mostram ao usuário todo o passo a passo dos cálculos necessários para obter uma matriz resultante. Se existisse um site que demonstrasse todo o processo do compilador, você utilizaria?

- Sim
- Não
- Talvez

FONTE: a própria autora

FIGURA 129 - Formulário Questão 6

**Perguntas sobre a aplicação VL-D+**

As próximas perguntas são pessoais sobre a sua experiência em relação à aplicação VL-D+ e serão usadas para a geração de gráficos mais precisos.

**O que você achou da disposição dos componentes da interface da aplicação? \***

Essa pergunta é referente ao seu entendimento da disposição dos componentes em tela e seu grau de dificuldade em utilizá-los.

Muito boa, foi possível navegá-la sem dificuldade

Boa, foi possível navegá-la sem dificuldade mas poderia melhorar

Mediana, tive pouca dificuldade para navegá-la

Ruim, tive dificuldade para navegá-la

Muito ruim, tive muita dificuldade para navegá-la

FONTE: a própria autora

FIGURA 130 - Formulário Questão 7

Sobre a página principal da aplicação, marque quais itens você acha que atendem o propósito de ajudar no aprendizado da disciplina de compiladores

A página principal da aplicação mostra o processo de compiladores em tempo real de acordo com o que o usuário digita na caixa de texto. Para você, quais elementos da tela de fato ajudam na compreensão do funcionamento do Compilador?

- Aba da Análise Léxica
- Aba da Análise Sintática
- Aba da Tabela de Símbolos
- Aba de Registro de Erros
- Histórico da Análise Léxica

FONTE: a própria autora

FIGURA 131 - Formulário Questão 8

Sobre a página principal da aplicação, marque qual item você mais gostou

- Aba da Análise Léxica
- Aba da Análise Sintática
- Aba da Tabela de Símbolos
- Aba de Registro de Erros
- Histórico da Análise Léxica

FONTE: a própria autora

FIGURA 132 - Formulário Questão 9

Você utilizaria a aplicação VL-D+ para estudar compiladores? \*

- Sim
- Não
- Talvez

FONTE: a própria autora

FIGURA 133 - Formulário Questão 10

Você acha que a aplicação VL-D+ auxiliaria futuros discentes da disciplina?

- Sim
- Não
- Talvez

FONTE: a própria autora

FIGURA 134 - Formulário Questão 11

Você acha que a aplicação VL-D+ poderia melhorar em algum aspecto? Se sim, qual? \*

Sua resposta

FONTE: a própria autora

FIGURA 135 - Formulário Questão 12

Comentários
Sua resposta

FONTE: a própria autora

## APÊNDICE C – INTERFACE VL-D+: GRAMÁTICA D+

FIGURA 136 - Gramática D+ Declarações

<u>Declarações</u>
programa → lista-decl
lista-decl → lista-decl decl   decl
decl → decl-main   decl-const   decl-var   decl-proc   decl-func
decl-main → MAIN () bloco END
decl-const → CONST ID = literal ;
decl-var → VAR espec-tipo lista-var ;
espec-tipo → INT   FLOAT   CHAR   BOOL   STRING
decl-proc → SUB espec-tipo ID ( params ) bloco ENDSUB
decl-func → FUNCTION espec-tipo ID ( params ) bloco ENDFUNCTION
params → lista-param   $\epsilon$
lista-param → lista-param , param   param
param → VAR espec-tipo lista-var BY mode
mode → VALUE   REF

FONTE: a própria autora

### FIGURA 137 - Gramática D+ Comandos

<u>Comandos</u>
bloco → lista-com
lista-com → comando lista-com   ε
comando → cham-proc   com-atrib   com-selecao   com-repeticao   com-desvio   com-leitura   com-escrita   decl-var   decl-const
com-atrib → var = exp ;
com-selecao → IF exp THEN bloco ENDIF   IF exp THEN bloco ELSE bloco ENDIF
com-repeticao → WHILE exp DO bloco LOOP   DO bloco WHILE exp ;   REPEAT bloco UNTIL exp ;   FOR ID = exp-soma TO exp-soma DO bloco NEXT
com-desvio → RETURN exp ;   BREAK ;   CONTINUE ;
com-leitura → SCAN ( lista-var ) ;   SCANLN ( lista-var ) ;
com-escrita → PRINT ( lista-exp ) ;   PRINTLN ( lista-exp ) ;
cham-proc → ID ( args ) ;

FONTE: a própria autora

FIGURA 138 - Gramática D+ Expressões

<u>Expressões</u>
lista-exp $\rightarrow$ exp , lista-exp   exp
exp $\rightarrow$ exp-soma op-relac exp-soma   exp-soma
op-relac $\rightarrow$ $\leq$   $<$   $>$   $\geq$   $=$   $\neq$
exp-soma $\rightarrow$ exp-mult op-soma exp-soma   exp-mult
op-soma $\rightarrow$ +   -   OR
exp-mult $\rightarrow$ exp-mult op-mult exp-simples   exp-simples
op-mult $\rightarrow$ *   /   DIV   MOD   AND
exp-simples $\rightarrow$ ( exp )   var   cham-func   literal   op-unario exp
literal $\rightarrow$ NUMINT   NUMREAL   CARACTERE   STRING   valor-verdade
valor-verdade $\rightarrow$ TRUE   FALSE
cham-func $\rightarrow$ ID ( args );
args $\rightarrow$ lista-exp   $\epsilon$
var $\rightarrow$ ID   ID [ exp-soma ]
lista-var $\rightarrow$ var , lista-var   var
op-unario $\rightarrow$ +   -   NOT

FONTE: a própria autora

FIGURA 139 - Gramática D+ Comentários

<u>Comentários</u>
comentário-linha → //comentário
comentário-bloco → /*comentário*/

FONTE: a própria autora

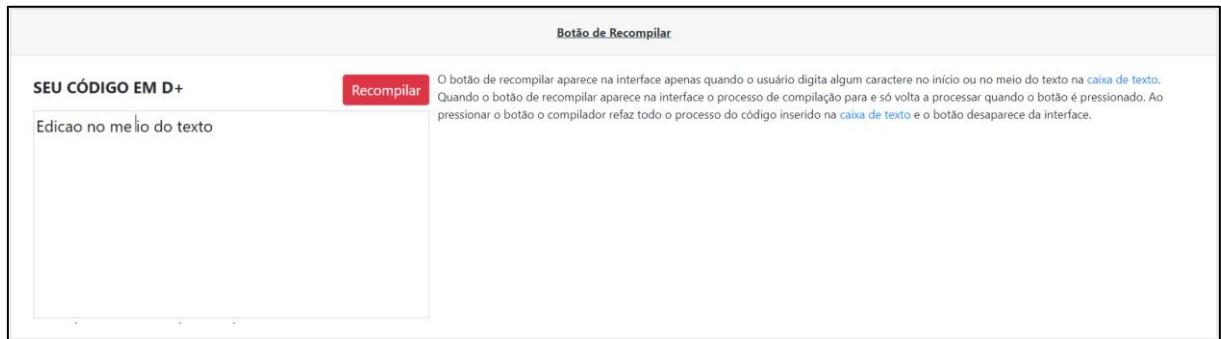
## APÊNDICE D – INTERFACE VL-D+: MANUAL DO USUÁRIO

FIGURA 140 - Manual do Usuário Caixa de Texto



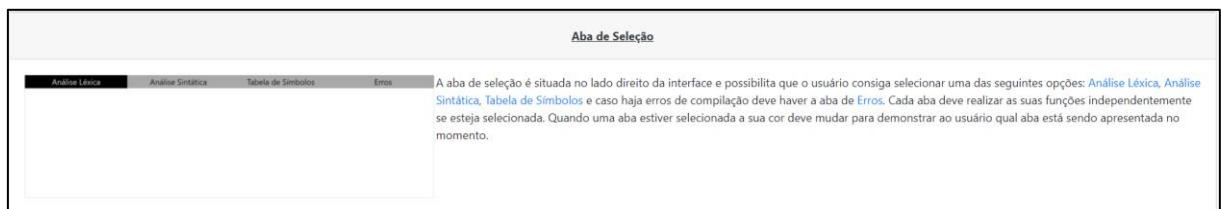
FONTE: a própria autora

FIGURA 141 - Manual do Usuário Botão de Recompilar



FONTE: a própria autora

FIGURA 142 - Manual do Usuário Aba de Seleção



FONTE: a própria autora

FIGURA 143 - Manual do Usuário Análise Léxica



FONTE: a própria autora

FIGURA 144 - Manual do Usuário Histórico da Análise Léxica



FONTE: a própria autora

FIGURA 145 - Manual do Usuário Análise Sintática



FONTE: a própria autora

FIGURA 146 - Tabela de Símbolos

Tabela de Símbolos		
Token	Lexema	
PR_MAIN	main	
SIN_PAR_A	(	
SIN_PAR_F	)	

FONTE: a própria autora

FIGURA 147 - Manual do Usuário Registro de Erros



FONTE: a própria autora