

Construção de um compilador de MiniLua para Parrot usando Objective Caml

Matheus Prado Prandini Faria

matheusprandini.96@gmail.com

Faculdade de Computação
Universidade Federal de Uberlândia

1 de Agosto de 2017

Lista de Figuras

1.1	Arquitetura da Parrot VM	14
2.1	Instalação da Parrot VM	15
3.1	Execução de um código PASM	59
3.2	Conversão e execução de um código PBC	60
3.3	Conversão de um código PIR para PASM	61

Lista de Tabelas

3.1 Principais comandos em PASM 43

Lista de Listagens

3.1	Módulo mínimo que caracteriza um programa	17
3.2	Declaração de uma variável	17
3.3	Atribuição de um inteiro à uma variável	17
3.4	Atribuição de uma soma de inteiros à uma variável	17
3.5	Inclusão do comando de impressão	18
3.6	Atribuição de uma subtração de inteiros à uma variável	18
3.7	Inclusão do comando condicional	18
3.8	Inclusão do comando condicional com parte senão	18
3.9	Atribuição de duas operações aritméticas sobre inteiros a uma variável	18
3.10	Atribuição de duas variáveis inteiras	19
3.11	Introdução do comando de repetição enquanto	19
3.12	Comando condicional aninhado em um comando de repetição	19
3.13	Converte graus Celsius para Fahrenheit	20
3.14	Ler dois inteiros e decide qual é maior	20
3.15	Lê um número e verifica se ele está entre 100 e 200	20
3.16	Lê números e informa quais estão entre 10 e 150	20
3.17	Lê strings e caracteres	21
3.18	Escreve um número lido por extenso	21
3.19	Decide se os números são positivos, zeros ou negativos	22
3.20	Decide se um número é maior ou menor que 10	22
3.21	Cálculo de preços	22
3.22	Calcula o fatorial de um número	23
3.23	Decide se um número é positivo, zero ou negativo com auxílio de uma função	23
3.24	Módulo mínimo que caracteriza um programa	24
3.25	Declaração de uma variável	24
3.26	Atribuição de um inteiro à uma variável	24
3.27	Atribuição de uma soma de inteiros à uma variável	24
3.28	Inclusão do comando de impressão	25
3.29	Atribuição de uma subtração de inteiros à uma variável	25
3.30	Inclusão do comando condicional	25
3.31	Inclusão do comando condicional com parte senão	25
3.32	Atribuição de duas operações aritméticas sobre inteiros a uma variável	25
3.33	Atribuição de duas variáveis inteiras	26
3.34	Introdução do comando de repetição enquanto	26
3.35	Comando condicional aninhado em um comando de repetição	26
3.36	Converte graus Celsius para Fahrenheit	27
3.37	Ler dois inteiros e decide qual é maior	27
3.38	Lê um número e verifica se ele está entre 100 e 200	27
3.39	Lê números e informa quais estão entre 10 e 150	28
3.40	Lê strings e caracteres	28

3.41	Escreve um número lido por extenso	29
3.42	Decide se os números são positivos, zeros ou negativos	29
3.43	Decide se um número é maior ou menor que 10	29
3.44	Cálculo de preços	30
3.45	Calcula o fatorial de um número	30
3.46	Decide se um número é positivo, zero ou negativo com auxílio de uma função	31
3.47	Módulo mínimo que caracteriza um programa	32
3.48	Declaração de uma variável	32
3.49	Atribuição de um inteiro à uma variável	32
3.50	Atribuição de uma soma de inteiros à uma variável	32
3.51	Inclusão do comando de impressão	32
3.52	Atribuição de uma subtração de inteiros à uma variável	32
3.53	Inclusão do comando condicional	33
3.54	Inclusão do comando condicional com parte senão	33
3.55	Atribuição de duas operações aritméticas sobre inteiros a uma variável	33
3.56	Atribuição de duas variáveis inteiras	33
3.57	Introdução do comando de repetição enquanto	34
3.58	Comando condicional aninhado em um comando de repetição	34
3.59	Converte graus Celsius para Fahrenheit	34
3.60	Ler dois inteiros e decide qual é maior	35
3.61	Lê um número e verifica se ele está entre 100 e 200	35
3.62	Lê números e informa quais estão entre 10 e 150	36
3.63	Lê strings e caracteres	36
3.64	Escreve um número lido por extenso	37
3.65	Decide se os números são positivos, zeros ou negativos	38
3.66	Decide se um número é maior ou menor que 10	38
3.67	Cálculo de preços	39
3.68	Calcula o fatorial de um número	40
3.69	Decide se um número é positivo, zero ou negativo com auxílio de uma função	40
3.70	Módulo mínimo que caracteriza um programa	47
3.71	Declaração de uma variável	47
3.72	Atribuição de um inteiro à uma variável	47
3.73	Atribuição de uma soma de inteiros à uma variável	47
3.74	Inclusão do comando de impressão	48
3.75	Atribuição de uma subtração de inteiros à uma variável	48
3.76	Inclusão do comando condicional	48
3.77	Inclusão do comando condicional com parte senão	48
3.78	Atribuição de duas operações aritméticas sobre inteiros a uma variável	49
3.79	Atribuição de duas variáveis inteiras	49
3.80	Introdução do comando de repetição enquanto	50
3.81	Comando condicional aninhado em um comando de repetição	50
3.82	Converte graus Celsius para Fahrenheit	51
3.83	Ler dois inteiros e decide qual é maior	51
3.84	Lê um número e verifica se ele está entre 100 e 200	52
3.85	Lê números e informa quais estão entre 10 e 150	52
3.86	Lê strings e caracteres	53
3.87	Escreve um número lido por extenso	54
3.88	Decide se os números são positivos, zeros ou negativos	55
3.89	Decide se um número é maior ou menor que 10	55

3.90	Cálculo de preços	56
3.91	Calcula o fatorial de um número	57
3.92	Decide se um número é positivo, zero ou negativo com auxílio de uma função	58
3.93	Comando condicional aninhado em um comando de repetição	61
4.1	Código do Analisador Léxico	64
4.2	Saída do analisador léxico para o programa nano01	68
4.3	Saída do analisador léxico para o programa nano02	69
4.4	Saída do analisador léxico para o programa nano03	69
4.5	Saída do analisador léxico para o programa nano04	69
4.6	Saída do analisador léxico para o programa nano05	69
4.7	Saída do analisador léxico para o programa nano06	70
4.8	Saída do analisador léxico para o programa nano07	70
4.9	Saída do analisador léxico para o programa nano08	70
4.10	Saída do analisador léxico para o programa nano09	70
4.11	Saída do analisador léxico para o programa nano10	71
4.12	Saída do analisador léxico para o programa nano11	71
4.13	Saída do analisador léxico para o programa nano12	71
4.14	Saída do analisador léxico para o programa micro01	72
4.15	Saída do analisador léxico para o programa micro02	72
4.16	Saída do analisador léxico para o programa micro03	73
4.17	Saída do analisador léxico para o programa micro04	73
4.18	Saída do analisador léxico para o programa micro05	74
4.19	Saída do analisador léxico para o programa micro06	75
4.20	Saída do analisador léxico para o programa micro07	75
4.21	Saída do analisador léxico para o programa micro08	76
4.22	Saída do analisador léxico para o programa micro09	76
4.23	Saída do analisador léxico para o programa micro10	77
4.24	Saída do analisador léxico para o programa micro11	77
4.25	Comentário não fechado corretamente	78
4.26	Comentário não fechado corretamente	78
4.27	Caracter desconhecido	78
4.28	Caracter desconhecido	79
4.29	String não fechada corretamente	79
4.30	String não fechada corretamente	79
5.1	Código do Analisador Sintático	80
5.2	Código da Árvore Sintática	83
5.3	Módulo mínimo que caracteriza um programa	86
5.4	Saída do analisador sintático para o programa nano01	86
5.5	Declaração de uma variável	86
5.6	Saída do analisador sintático para o programa nano02	86
5.7	Atribuição de um inteiro à uma variável	86
5.8	Saída do analisador sintático para o programa nano03	86
5.9	Atribuição de uma soma de inteiros à uma variável	86
5.10	Saída do analisador sintático para o programa nano04	87
5.11	Inclusão do comando de impressão	87
5.12	Saída do analisador sintático para o programa nano05	87
5.13	Atribuição de uma subtração de inteiros à uma variável	87
5.14	Saída do analisador sintático para o programa nano06	87
5.15	Inclusão do comando condicional	88

5.16	Saída do analisador sintático para o programa nano07	88
5.17	Inclusão do comando condicional com parte senão	88
5.18	Saída do analisador sintático para o programa nano08	88
5.19	Atribuição de duas operações aritméticas sobre inteiros a uma variável	89
5.20	Saída do analisador sintático para o programa nano09	89
5.21	Atribuição de duas variáveis inteiras	89
5.22	Saída do analisador sintático para o programa nano10	89
5.23	Introdução do comando de repetição enquanto	90
5.24	Saída do analisador sintático para o programa nano11	90
5.25	Comando condicional aninhado em um comando de repetição	90
5.26	Saída do analisador sintático para o programa nano12	91
5.27	Converte graus Celsius para Fahrenheit	91
5.28	Saída do analisador sintático para o programa micro01	92
5.29	Ler dois inteiros e decide qual é maior	92
5.30	Saída do analisador sintático para o programa micro02	92
5.31	Lê um número e verifica se ele está entre 100 e 200	93
5.32	Saída do analisador sintático para o programa micro03	93
5.33	Lê números e informa quais estão entre 10 e 150	94
5.34	Saída do analisador sintático para o programa micro04	94
5.35	Lê strings e caracteres	95
5.36	Saída do analisador sintático para o programa micro05	95
5.37	Escreve um número lido por extenso	96
5.38	Saída do analisador sintático para o programa micro06	96
5.39	Decide se os números são positivos, zeros ou negativos	97
5.40	Saída do analisador sintático para o programa micro07	98
5.41	Decide se um número é maior ou menor que 10	98
5.42	Saída do analisador sintático para o programa micro08	98
5.43	Cálculo de preços	99
5.44	Saída do analisador sintático para o programa micro09	99
5.45	Calcula o fatorial de um número	100
5.46	Saída do analisador sintático para o programa micro10	101
5.47	Decide se um número é positivo, zero ou negativo com auxílio de uma função	101
5.48	Saída do analisador sintático para o programa micro11	102
5.49	Comando fora do escopo de uma função	102
5.50	Comando fora do escopo de uma função	102
5.51	Função não estruturada corretamente	103
5.52	Função não estruturada corretamente	103
5.53	String não fechada corretamente	103
5.54	String não fechada corretamente	103
5.55	Atribuição de variável incorreta	103
5.56	Atribuição de variável incorreta	103
5.57	Comando IF incorreto	104
5.58	Comando IF incorreto	104
5.59	Comando FOR incorreto	104
5.60	Comando FOR incorreto	104
5.61	Comando WHILE incorreto	105
5.62	Comando WHILE incorreto	105
5.63	Retorno de função incorreto	105
5.64	Retorno de função incorreto	106

6.1	Módulo mínimo que caracteriza um programa	108
6.2	Saída do analisador semântico para o programa nano01	108
6.3	Saída do interpretador para o programa nano01	108
6.4	Declaração de uma variável	108
6.5	Saída do analisador semântico para o programa nano02	108
6.6	Saída do interpretador para o programa nano02	109
6.7	Atribuição de um inteiro à uma variável	109
6.8	Saída do analisador semântico para o programa nano03	109
6.9	Saída do interpretador para o programa nano03	109
6.10	Atribuição de uma soma de inteiros à uma variável	110
6.11	Saída do analisador semântico para o programa nano04	110
6.12	Saída do interpretador para o programa nano04	110
6.13	Inclusão do comando de impressão	110
6.14	Saída do analisador semântico para o programa nano05	111
6.15	Saída do interpretador para o programa nano05	111
6.16	Atribuição de uma subtração de inteiros à uma variável	112
6.17	Saída do analisador semântico para o programa nano06	112
6.18	Saída do interpretador para o programa nano06	112
6.19	Inclusão do comando condicional	113
6.20	Saída do analisador semântico para o programa nano07	113
6.21	Saída do interpretador para o programa nano07	114
6.22	Inclusão do comando condicional com parte senão	114
6.23	Saída do analisador semântico para o programa nano08	114
6.24	Saída do interpretador para o programa nano08	115
6.25	Atribuição de duas operações aritméticas sobre inteiros a uma variável	115
6.26	Saída do analisador semântico para o programa nano09	115
6.27	Saída do interpretador para o programa nano09	116
6.28	Atribuição de duas variáveis inteiras	116
6.29	Saída do analisador semântico para o programa nano10	117
6.30	Saída do interpretador para o programa nano10	118
6.31	Introdução do comando de repetição enquanto	119
6.32	Saída do analisador semântico para o programa nano11	120
6.33	Saída do interpretador para o programa nano11	121
6.34	Comando condicional aninhado em um comando de repetição	121
6.35	Saída do analisador semântico para o programa nano12	122
6.36	Saída do interpretador para o programa nano12	124
6.37	Converte graus Celsius para Fahrenheit	124
6.38	Saída do analisador semântico para o programa micro01	124
6.39	Saída do interpretador para o programa micro01	126
6.40	Ler dois inteiros e decide qual é maior	126
6.41	Saída do analisador semântico para o programa micro02	126
6.42	Saída do interpretador para o programa micro02	128
6.43	Lê um número e verifica se ele está entre 100 e 200	128
6.44	Saída do analisador semântico para o programa micro03	128
6.45	Saída do interpretador para o programa micro03	129
6.46	Lê números e informa quais estão entre 10 e 150	130
6.47	Saída do analisador semântico para o programa micro04	130
6.48	Saída do interpretador para o programa micro04	132
6.49	Lê strings e caracteres	133

6.50 Saída do analisador semântico para o programa micro05	133
6.51 Saída do interpretador para o programa micro05	136
6.52 Escreve um número lido por extenso	136
6.53 Saída do analisador semântico para o programa micro06	137
6.54 Saída do interpretador para o programa micro06	138
6.55 Decide se os números são positivos, zeros ou negativos	138
6.56 Saída do analisador semântico para o programa micro07	139
6.57 Saída do interpretador para o programa micro07	141
6.58 Decide se um número é maior ou menor que 10	141
6.59 Saída do analisador semântico para o programa micro08	142
6.60 Saída do interpretador para o programa micro08	143
6.61 Cálculo de preços	143
6.62 Saída do analisador semântico para o programa micro09	144
6.63 Saída do interpretador para o programa micro09	146
6.64 Calcula o fatorial de um número	146
6.65 Saída do analisador semântico para o programa micro10	147
6.66 Saída do interpretador para o programa micro10	150
6.67 Decide se um número é positivo, zero ou negativo com auxílio de uma função	150
6.68 Saída do analisador semântico para o programa micro11	150
6.69 Saída do interpretador para o programa micro11	153
6.70 Variável declarada duas vezes com tipos diferentes	153
6.71 Variável declarada duas vezes com tipos diferentes	153
6.72 Atribuição de um inteiro com um valor float	154
6.73 Atribuição de um inteiro com um valor float	154
6.74 Operação envolvendo tipos diferentes	154
6.75 Operação envolvendo tipos diferentes	154
6.76 Chamada de função com número incorreto de parâmetros	155
6.77 Chamada de função com número incorreto de parâmetros	155
6.78 Declaração de parâmetros iguais de uma função	155
6.79 Declaração de parâmetros iguais de uma função	155
6.80 Tipo de retorno diferente do tipo declarado da função	155
6.81 Tipo de retorno diferente do tipo declarado da função	156
A.1 ambiente.ml	157
A.2 ambiente.mli	157
A.3 ambInterp.ml	158
A.4 ambInterp.mli	159
A.5 ast.ml	159
A.6 interprete.ml	161
A.7 interprete.mli	167
A.8 lexico.mll	167
A.9 sast.ml	169
A.10 semantico.ml	169
A.11 semantico.mli	177
A.12 sintatico.mly	177
A.13 tabsimb.ml	181
A.14 tabsimb.mli	182
A.15 tast.ml	182

Sumário

Lista de Figuras	2
Lista de Tabelas	3
1 Introdução	13
1.1 Máquina Virtual Parrot (PVM)	13
1.1.1 Arquitetura da Parrot VM	13
1.2 OCaml	14
2 Instalações	15
2.1 Instalação da Máquina Virtual Parrot	15
2.2 Instalação do OCaml	16
3 Codificação e Tradução de Pseudo-Códigos	17
3.1 Códigos em linguagem Lua	17
3.1.1 Nano Programas	17
3.1.2 Micro Programas	20
3.2 Códigos em linguagem Perl	24
3.2.1 Nano Programas	24
3.2.2 Micro Programas	27
3.3 Códigos em linguagem PIR	32
3.3.1 Nano Programas	32
3.3.2 Micro Programas	34
3.4 Tradução para Parrot Assembly	41
3.4.1 Registradores presentes no PASM	41
3.4.2 Comandos mais utilizados no PASM	42
3.4.3 Nano Programas	47
3.4.4 Micro Programas	51
3.4.5 Execução de um código PASM	59
3.4.6 Conversão de PASM para Bytecode	60
3.4.7 Conversão de PIR para PASM	60
4 Analisador Léxico	62
4.1 Reconhecimento das Palavras Reservadas em Lua	62
4.2 Código do Analisador Léxico	64
4.3 Execução do Analisador Léxico	68
4.4 Testes do Analisador Léxico	68
4.4.1 Nano Programas	68
4.4.2 Micro Programas	72
4.5 Testes de Erros	78

4.5.1	Comentário não fechado	78
4.5.2	Caracter Desconhecido	78
4.5.3	String não fechada corretamente	79
5	Analizador Sintático	80
5.1	Gramática e Código do Analizador Sintático	80
5.2	Árvore Sintática Abstrata	83
5.3	Execução do Analizador Sintático	85
5.4	Testes do Analizador Sintático	85
5.4.1	Nano Programas	85
5.4.2	Micro Programas	91
5.5	Testes de Erros Sintáticos	102
5.5.1	Comandos fora do escopo de uma função	102
5.5.2	Função não estruturada corretamente	103
5.5.3	Declaração incorreta de variável	103
5.5.4	Atribuição de variável incorreta	103
5.5.5	Comando IF em formato incorreto	104
5.5.6	Comando FOR em formato incorreto	104
5.5.7	Comando WHILE em formato incorreto	105
5.5.8	Retorno de função em formato incorreto	105
6	Analizador Semântico e Interpretador	107
6.1	Execução do Analizador Semântico e Interpretador	107
6.2	Testes do Analizador Semântico e Interpretador	108
6.2.1	Nano Programas	108
6.2.2	Micro Programas	124
6.3	Testes de Erros Semânticos	153
6.3.1	Variável declarada duas vezes com tipos diferentes	153
6.3.2	Atribuição de um inteiro com um valor float	154
6.3.3	Operação envolvendo tipos diferentes	154
6.3.4	Chamada de função com número incorreto de parâmetros	154
6.3.5	Declaração de parâmetros iguais de uma função	155
6.3.6	Tipo de retorno diferente do tipo declarado da função	155
	Apêndice	157
A	Códigos Finais	157
A.1	Código "ambiente.ml":	157
A.2	Código "ambiente.mli":	157
A.3	Código "ambInterp.ml":	158
A.4	Código "ambInterp.mli":	159
A.5	Código "ast.ml":	159
A.6	Código "interprete.ml":	161
A.7	Código "interprete.mli":	167
A.8	Código "lexico.mll":	167
A.9	Código "sast.ml":	169
A.10	Código "semantico.ml":	169
A.11	Código "semantico.mli":	177
A.12	Código "sintatico.mly":	177

A.13 Código "tabsimb.ml":	181
A.14 Código "tabsimb.mli":	182
A.15 Código "tast.ml":	182
B Referências Bibliográficas	183

Capítulo 1

Introdução

Este documento apresenta o processo de desenvolvimento de um compilador da linguagem MiniLua para a máquina virtual Parrot utilizando o OCaml. Em um primeiro momento será abordada a instalação das plataformas necessárias, a arquitetura da máquina virtual utilizada, a escrita de códigos nas linguagens Lua e Perl a partir de algoritmos sugeridos pelo Professor Alexsandro Santos Soares, além da tradução dos referidos códigos para a linguagem PASM (Parrot Assembly Language) e a execução dos mesmos. O objetivo principal é o entendimento e aprendizado da linguagem Assembly utilizada pela Máquina Virtual Parrot.

1.1 Máquina Virtual Parrot (PVM)

Parrot VM é uma máquina virtual cujo objetivo é compilar de forma eficiente e executar bytecode para linguagens interpretadas. Projetado para linguagens dinâmicas, é alvo de uma grande variedade de linguagens como Perl, Tcl, Ruby, Python, etc.

1.1.1 Arquitetura da Parrot VM

Atualmente, existem quatro tipos de códigos de programa aceitáveis pelo Parrot, são eles: PIR, PASM, PAST, PBC. O primeiro, Parrot Intermediate Representation, é uma linguagem de mais alto nível comparado com o PASM, pois permite a escrita de códigos de baixo nível com uma linguagem mais próxima do usuário, sua extensão é ".pir". O Segundo, Parrot Assembly Language, é uma linguagem Assembly de mais baixo nível com relação ao PIR, baseada em registradores, sua extensão é representada por ".pasm". O terceiro, Parrot Abstract Syntax Tree, permite o Parrot aceitar uma entrada do tipo de árvore sintática abstrata. O Quarto, Parrot Bytecode, é muito parecido com código de máquina e é o binário interpretado pela máquina virtual, com extensão ".pbc". É importante destacar que as três primeiras formas citadas são transformadas automaticamente para PBC no Parrot. Sua arquitetura pode ser vista na figura [1.1](#).

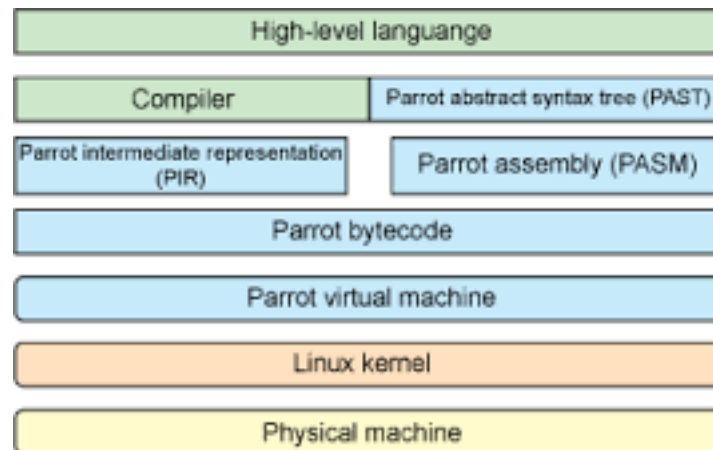


Figura 1.1: *Arquitetura da Parrot VM*

1.2 OCaml

Objective Caml, também conhecida como OCaml (Objective Categorical Abstract Machine Language), é uma linguagem de programação funcional, fortemente e estaticamente tipada, além de possuir adição de suporte de técnicas de orientação a objetos. Ela será utilizada para a construção de um compilador da linguagem MiniLua para a Máquina Virtual Parrot.

Capítulo 2

Instalações

2.1 Instalação da Máquina Virtual Parrot

Para instalar a Parrot VM, primeiramente, foi digitado o seguinte comando no terminal:

```
> sudo software-center
```

Dessa forma, o gerenciador de programas do Ubuntu é aberto. Após isso, procura-se por "Parrot" na caixa de pesquisa e instala-se a máquina virtual juntamente com uma das bibliotecas necessárias, `parrot-doc`, como visto na figura 2.1.

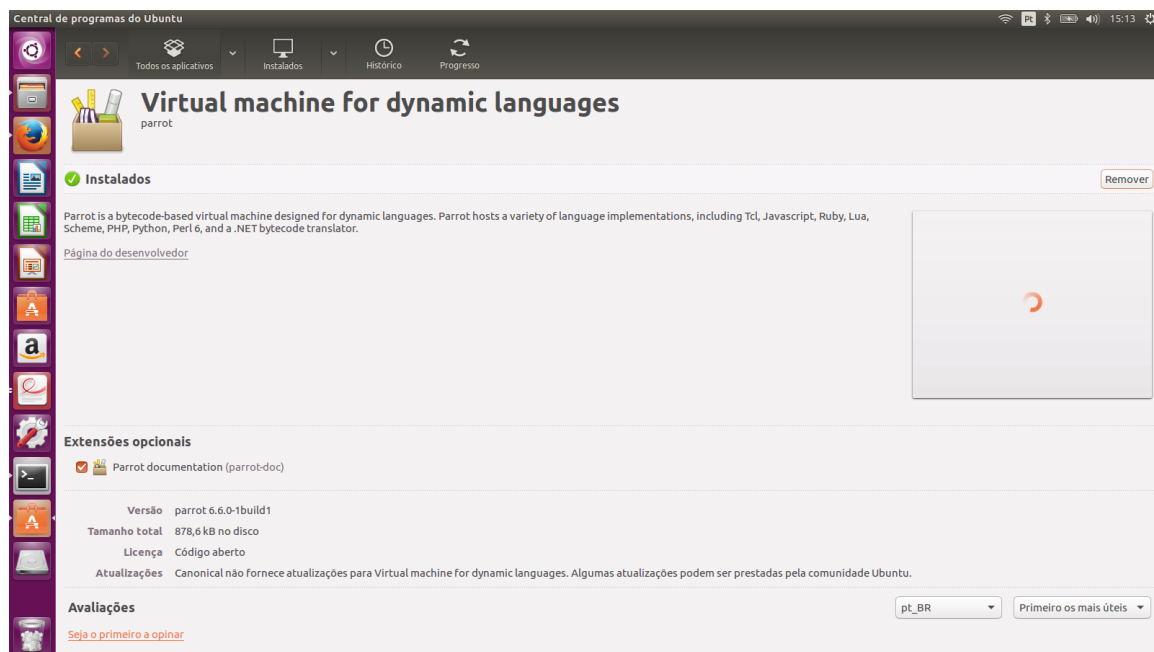


Figura 2.1: *Instalação da Parrot VM*

Além disso, são instaladas também outras duas bibliotecas para auxiliar no uso da máquina virtual, por meio dos seguintes comandos no terminal:

```
> sudo apt-get install libparrot  
> sudo apt-get install libparrot-dev
```

2.2 Instalação do OCaml

Para instalar o OCaml, basta entrar com o seguinte comando no terminal:

```
> sudo apt-get install ocaml
```


Capítulo 3

Codificação e Tradução de Pseudo-Códigos

3.1 Códigos em linguagem Lua

Nesta seção, são apresentados os códigos em linguagem Lua dos pseudo-códigos propostos. Para compilar e executar os arquivos com extensão ".lua", basta utilizar o seguinte comando no terminal:

```
> lua nome_arquivo.lua
```

3.1.1 Nano Programas

Listagem 3.1: Módulo mínimo que caracteriza um programa

```
1 function main()  
2 end
```

Listagem 3.2: Declaração de uma variável

```
1 function main()  
2     local n  
3 end
```

Listagem 3.3: Atribuição de um inteiro à uma variável

```
1 function main()  
2     local n  
3     n = 1  
4 end  
5  
6 main()
```

Listagem 3.4: Atribuição de uma soma de inteiros à uma variável

```
1 function main()
```

```
2     local n
3     n = 1 + 2
4 end
5
6 main()
```

Listagem 3.5: Inclusão do comando de impressão

```
1 function main()
2     local n
3     n = 2
4     print(n)
5 end
6
7 main()
```

Listagem 3.6: Atribuição de uma subtração de inteiros à uma variável

```
1 function main()
2     local n
3     n = 1 - 2
4     print(n)
5 end
6
7 main()
```

Listagem 3.7: Inclusão do comando condicional

```
1 function main()
2     local n
3     n = 1
4     if n == 1 then
5         print(n)
6     end
7 end
8
9 main()
```

Listagem 3.8: Inclusão do comando condicional com parte senão

```
1 function main()
2     local n
3     n = 1
4     if n == 1 then
5         print(n)
6     else
7         print(0)
8     end
9 end
10
11 main()
```

Listagem 3.9: Atribuição de duas operações aritméticas sobre inteiros a uma variável

```
1 function main()
2     local n
3     n = (1 + 1) / 2
```

3.1

```
4     if n == 1 then
5         print(n)
6     else
7         print(0)
8     end
9 end
10
11 main()
```

Listagem 3.10: Atribuição de duas variáveis inteiras

```
1 function main()
2     local n, m
3     n = 1
4     m = 2
5     if n == m then
6         print(n)
7     else
8         print(0)
9     end
10 end
11
12 main()
```

Listagem 3.11: Introdução do comando de repetição enquanto

```
1 function main()
2     local n, m, x
3     n = 1
4     m = 2
5     x = 5
6     while x > n do
7         n = n + m
8         print(n)
9     end
10 end
11
12 main()
```

Listagem 3.12: Comando condicional aninhado em um comando de repetição

```
1 function main()
2     local n, m, x
3     n = 1
4     m = 2
5     x = 5
6     while x > n do
7         if n == m then
8             print(n)
9         else
10            print(0)
11        end
12        x = x - 1
13    end
14 end
15
16 main()
```

3.1.2 Micro Programas

Listagem 3.13: Converte graus Celsius para Fahrenheit

```

1 function main()
2     local cel, far
3     print("Tabela de conversão: Celsius -> Fahrenheit")
4     print("Digite a temperatura em Celsius: ")
5     cel = io
6     .read("*number")
7     far = (9*cel+160)/5
8     print("A nova temperatura eh: "..far.." F")
9 end
10
11 main()

```

Listagem 3.14: Ler dois inteiros e decide qual é maior

```

1 function main()
2     local num1, num2
3     print("Digite o primeiro numero: ")
4     num1 = io.read("*number")
5     print("Digite o segundo numero: ")
6     num2 = io.read("*number")
7
8     if num1 > num2 then
9         print("O primeiro número "..num1.." é maior que o segundo "..num2)
10    else
11        print("O segundo número "..num2.." é maior que o primeiro "..num1)
12    end
13 end
14
15 main()

```

Listagem 3.15: Lê um número e verifica se ele está entre 100 e 200

```

1 function main()
2     local numero
3     print("Digite um número: ")
4     numero = io.read("*number")
5     if numero >= 100 then
6         if numero <= 200 then
7             print("O número está no intervalo entre 100 e 200")
8         else
9             print("O número não está no intervalo entre 100 e 200")
10        end
11    else
12        print("O número não está no intervalo entre 100 e 200")
13    end
14 end
15
16 main()

```

Listagem 3.16: Lê números e informa quais estão entre 10 e 150

```

1 function main()
2     local x, num, intervalo
3     intervalo = 0

```

3.1

```
4     for x=1, 5, 1 do
5         print("Digite um número: ")
6         num = io.read("*number")
7         if num >= 10 then
8             if num <= 150 then
9                 intervalo = intervalo + 1
10            end
11        end
12    end
13    print("Ao total, foram digitados "..intervalo.." números no intervalo
14        entre 10 e 150")
15
16 main()
```

Listagem 3.17: Lê strings e caracteres

```
1 function main()
2     local nome, sexo
3     local x, h, m = 1, 0, 0
4     for x=1, 5, 1 do
5         print("Digite o nome: ")
6         nome = io.read("*line")
7         print("H - Homem ou M - Mulher: ")
8         sexo = io.read("*line")
9         if sexo == 'H' then
10            h = h + 1
11        elseif sexo == 'M' then
12            m = m + 1
13        else
14            print("Sexo só pode ser H ou M!")
15        end
16    end
17    print("Foram inseridos "..h.." Homens")
18    print("Foram inseridos "..m.." Mulheres")
19 end
20
21 main()
```

Listagem 3.18: Escreve um número lido por extenso

```
1 function main()
2     local num
3     print("Digite um numero de 1 a 5: ")
4     num = io.read("*number")
5     if num == 1 then
6         print("Um")
7     elseif num == 2 then
8         print("Dois")
9     elseif num == 3 then
10        print("Tres")
11    elseif num == 4 then
12        print("Quatro")
13    elseif num == 5 then
14        print("Cinco")
15    else
16        print("Numero Invalido!!!")
17    end
18 end
```

```

19
20 main()

```

Listagem 3.19: Decide se os números são positivos, zeros ou negativos

```

1 function main()
2     local programa, numero, opc
3     programa = 1
4     while programa == 1 do
5         print("Digite um número: ")
6         numero = io.read("*n")
7         if numero > 0 then
8             print("Positivo")
9         else
10            if numero == 0 then
11                print("O número é igual a 0")
12            end
13            if numero < 0 then
14                print("Negativo")
15            end
16        end
17
18        print("Deseja finalizar? (S-1/N-2) ")
19        opc = io.read("*n")
20        if opc == 1 then
21            programa = 0
22        end
23    end
24 end
25
26 main()

```

Listagem 3.20: Decide se um número é maior ou menor que 10

```

1 function main()
2     local numero
3     numero = 1
4     while numero ~= 0 do
5         print("Digite um número: ")
6         numero = io.read("*n")
7         if numero > 10 then
8             print("O número "..numero.." é maior que 10")
9         else
10            print("O número "..numero.." é menor que 10")
11        end
12    end
13 end
14
15 main()

```

Listagem 3.21: Cálculo de preços

```

1 function main()
2     local preco, venda, novo_preco
3     print("Digite o preço: ")
4     preco = io.read("*n")
5     print("Digite a venda: ")
6     venda = io.read("*n")

```

3.1

```
7     if venda < 500 or preco < 30 then
8         novo_preco = preco + 10/100 * preco
9     else if (venda >= 500 and venda < 1200) or (preco >= 30 and preco <
10         80) then
11         novo_preco = preco + 15/100 * preco
12     else if venda >= 1200 or preco >= 80 then
13         novo_preco = preco - 20/100 * preco
14     end
15 end
16 print("O novo preço é "..novo_preco)
17 end
18
19 main()
```

Listagem 3.22: Calcula o fatorial de um número

```
1 function fatorial(n)
2     if n <= 0 then
3         return 1
4     else
5         return n * fatorial(n-1)
6     end
7 end
8
9 function main()
10     local numero, fat
11     print("Digite um número: ")
12     numero = io.read("*n")
13     fat = fatorial(numero)
14     print("O fatorial de ")
15     print(numero)
16     print(" é ")
17     print(fat)
18 end
19
20 main()
```

Listagem 3.23: Decide se um número é positivo, zero ou negativo com auxílio de uma função

```
1 function verifica(n)
2     local res
3     if n > 0 then
4         res = 1
5     else if n < 0 then
6         res = -1
7     else
8         res = 0
9     end
10 end
11 return res
12 end
13
14 function main()
15     local numero
16     local x
17     print("Digite um número: ")
18     numero = io.read("*n")
19     x = verifica(numero)
```

```

20     if x == 1 then
21         print("Número positivo")
22     else if x == 0 then
23         print("Zero")
24     else
25         print("Número negativo")
26     end
27 end
28 end
29
30 main()

```

3.2 Códigos em linguagem Perl

Nesta seção, são apresentados os códigos em linguagem Perl dos pseudo-códigos propostos. Para compilar e executar os arquivos com extensão ".pl", basta utilizar o seguinte comando no terminal:

```
> perl nome_arquivo.pl
```

3.2.1 Nano Programas

Listagem 3.24: Módulo mínimo que caracteriza um programa

```

1 sub nano01 {
2
3 }

```

Listagem 3.25: Declaração de uma variável

```

1 use strict;
2
3 sub nano02 {
4     my $n;
5 }

```

Listagem 3.26: Atribuição de um inteiro à uma variável

```

1 use strict;
2
3 sub nano03 {
4     my $n = 1;
5 }

```

Listagem 3.27: Atribuição de uma soma de inteiros à uma variável

```

1 use strict;
2
3 sub nano04 {
4     my $n = 1 + 2;
5 }

```

Listagem 3.28: Inclusão do comando de impressão

```

1 use strict;
2
3 sub nano05 {
4     my $n = 2;
5     print "$n\n";
6 }
7
8 nano05()

```

Listagem 3.29: Atribuição de uma subtração de inteiros à uma variável

```

1 use strict;
2
3 sub nano06 {
4     my $n = 1 - 2;
5     print "$n\n";
6 }
7
8 nano06()

```

Listagem 3.30: Inclusão do comando condicional

```

1 use strict;
2
3 sub nano07 {
4     my $n = 1;
5     if($n==1) {
6         print "$n\n";
7     }
8 }
9
10 nano07()

```

Listagem 3.31: Inclusão do comando condicional com parte senão

```

1 use strict;
2
3 sub nano08 {
4     my $n = 1;
5     if($n==1) {
6         print("$n\n");
7     }
8     else {
9         print("0\n");
10    }
11 }
12
13 nano08()

```

Listagem 3.32: Atribuição de duas operações aritméticas sobre inteiros a uma variável

```

1 use strict;
2
3 sub nano09 {
4     my $n = 1 + 1 / 2;
5     if($n==1) {

```

```

6         print("$n\n");
7     }
8     else {
9         print("0\n");
10    }
11 }
12
13 nano09()

```

Listagem 3.33: Atribuição de duas variáveis inteiras

```

1 use strict;
2
3 sub nano10 {
4     my $n = 1;
5     my $m = 2;
6     if($n==$m) {
7         print("$n\n");
8     }
9     else {
10        print("0\n");
11    }
12 }
13
14 nano10()

```

Listagem 3.34: Introdução do comando de repetição enquanto

```

1 use strict;
2
3 sub nano11 {
4     my $n = 1;
5     my $m = 2;
6     my $x = 5;
7     while($x > $n) {
8         $n = $n + $m;
9         print("$n\n");
10    }
11 }
12
13 nano11()

```

Listagem 3.35: Comando condicional aninhado em um comando de repetição

```

1 use strict;
2
3 sub nano12() {
4     my $n = 1;
5     my $m = 2;
6     my $x = 5;
7     while($x > $n){
8         if($n == $m){
9             print("$n\n");
10        }
11        else {
12            print("0\n");
13        }
14        $x = $x - 1;

```

3.2

```
15     }
16 }
17
18 nano12()
```

3.2.2 Micro Programas

Listagem 3.36: Converte graus Celsius para Fahrenheit

```
1 use strict;
2
3 sub micro01 {
4     my $cel;
5     my $far;
6     print("Tabela de conversão: Celsius -> Fahrenheit\n");
7     print("Digite a temperatura em Celsius: \n");
8     my $cel = <STDIN>;
9     my $far = (9*$cel+160)/5;
10    print("A nova temperatura é $far F\n");
11 }
12
13 micro01()
```

Listagem 3.37: Ler dois inteiros e decide qual é maior

```
1 use strict;
2
3 sub micro02 {
4     my $num1;
5     my $num2;
6     print("Digite o primeiro numero: ");
7     $num1 = <STDIN>;
8     print("Digite o segundo numero: ");
9     $num2 = <STDIN>;
10
11     if($num1 > $num2) {
12         print("O primeiro número $num1 é maior que o segundo $num2\n");
13     }
14     else {
15         print("O segundo número $num2 é maior que o primeiro $num1\n");
16     }
17 }
18
19 micro02()
```

Listagem 3.38: Lê um número e verifica se ele está entre 100 e 200

```
1 use strict;
2
3 sub micro03 {
4     my $numero;
5     print("Digite um número: \n");
6     $numero = <STDIN>;
7     if($numero >= 100) {
8         if($numero <= 200) {
9             print("O número está no intervalo entre 100 e 200\n");
10        }
11    }
12 }
```

```

11         else {
12             print("O número não está no intervalo entre 100 e 200\n")
13         }
14     }
15     else {
16         print("O número não está no intervalo entre 100 e 200\n")
17     }
18 }
19
20 micro03()

```

Listagem 3.39: Lê números e informa quais estão entre 10 e 150

```

1 use strict;
2
3 sub micro04 {
4     my $x;
5     my $num;
6     my $intervalo;
7     $intervalo = 0;
8     for($x=1; $x<=5; $x = $x + 1) {
9         print("Digite um número: ");
10        $num = <STDIN>;
11        if($num >= 10) {
12            if($num <= 150) {
13                $intervalo = $intervalo + 1;
14            }
15        }
16    }
17    print("Ao total, foram digitados $intervalo números no intervalo entre
18        10 e 150\n")
19
20 micro04()

```

Listagem 3.40: Lê strings e caracteres

```

1 use strict;
2
3 sub micro05 {
4     my ($nome, $sexo);
5     my ($x, $h, $m) = (1, 0, 0);
6     for($x=1; $x<=5; $x = $x + 1) {
7         print("Digite o nome: ");
8         $nome = <STDIN>;
9         print("H - Homem ou M - Mulher: ");
10        $sexo = <STDIN>;
11        if(substr($sexo, 0, 1) eq 'H') {
12            $h = $h + 1;
13        }
14        elsif(substr($sexo, 0, 1) eq 'M') {
15            $m = $m + 1;
16        }
17        else {
18            print("Sexo só pode ser H ou M!");
19        }
20    }
21    print("Foram inseridos $h Homens\n");
22    print("Foram inseridos $m Mulheres\n");

```

```

23 }
24
25 micro05()

```

Listagem 3.41: Escreve um número lido por extenso

```

1 use Switch;
2 use strict;
3
4 sub micro06 {
5     my $num;
6     print("Digite um numero de 1 a 5: ");
7     $num = <STDIN>;
8     switch(substr($num, 0, 1)){
9         case "1" { print("Um\n") }
10        case "2" { print("Dois\n") }
11        case "3" { print("Tres\n") }
12        case "4" { print("Quatro\n") }
13        case "5" { print("Cinco\n") }
14        else     { print("Numero Invalido!!!\n") }
15    }
16 }
17
18 micro06()

```

Listagem 3.42: Decide se os números são positivos, zeros ou negativos

```

1 use strict;
2
3 sub micro07 {
4     my ($programa, $numero, $opc);
5     $programa = 1;
6     while($programa == 1) {
7         print("Digite um número: ");
8         $numero = <STDIN>;
9         if($numero > 0) {
10            print("Positivo\n");
11        }
12        else {
13            if($numero == 0) {
14                print("O número é igual a 0\n");
15            }
16            if($numero < 0) {
17                print("Negativo\n");
18            }
19        }
20
21        print("Deseja finalizar? (S/N) ");
22        $opc = <STDIN>;
23        if(substr($opc, 0, 1) eq 'S') {
24            $programa = 0;
25        }
26    }
27 }
28
29 micro07()

```

Listagem 3.43: Decide se um número é maior ou menor que 10

```

1 use strict;
2
3 sub micro08 {
4     my $numero;
5     $numero = 1;
6     while($numero != 0) {
7         print("Digite um número: ");
8         $numero = <STDIN>;
9         if($numero > 10) {
10            print("O número $numero é maior que 10\n");
11        }
12        else {
13            print("O número $numero é menor que 10\n");
14        }
15    }
16 }
17
18 micro08()

```

Listagem 3.44: Cálculo de preços

```

1 use strict;
2
3 sub micro09 {
4     my ($preco, $venda, $novo_preco);
5     print("Digite o preço: ");
6     $preco = <STDIN>;
7     print("Digite a venda: ");
8     $venda = <STDIN>;
9     if($venda < 500 || $preco < 30) {
10        print("a");
11        $novo_preco = $preco + 10/100 * $preco;
12    }
13    else {
14        if(($venda >= 500 and $venda < 1200) or
15           ($preco >= 30 and $preco < 80)) {
16            $novo_preco = $preco + 15/100 * $preco;
17        }
18        else {
19            if($venda >= 1200 or $preco >= 80) {
20                $novo_preco = $preco - 20/100 * $preco;
21            }
22        }
23    }
24    print("O novo preço é $novo_preco\n");
25 }
26
27 micro09()

```

Listagem 3.45: Calcula o fatorial de um número

```

1 use strict;
2
3 sub fatorial {
4
5     my $n = shift;
6
7     if($n <= 0) {
8         return 1;

```

3.3

```
9     }
10     else {
11         return $n * fatorial($n-1);
12     }
13 }
14
15 sub micro10 {
16     my ($numero, $fat);
17     print "Digite um número: ";
18     $numero = <STDIN>;
19     $fat = fatorial($numero);
20     print("O fatorial de ");
21     print($numero);
22     print(" é ");
23     print("$fat\n");
24 }
25
26 micro10()
```

Listagem 3.46: Decide se um número é positivo, zero ou negativo com auxílio de uma função

```
1 use strict;
2
3 sub verifica {
4     my $n = shift;
5     my $res;
6     if($n > 0) {
7         $res = 1;
8     }
9     else {
10    if($n < 0) {
11        $res = -1;
12    }
13    else {
14        $res = 0;
15    }
16    }
17    return $res;
18 }
19
20 sub micro11 {
21     my ($numero, $x);
22     print("Digite um número: ");
23     $numero = <STDIN>;
24     $x = verifica($numero);
25     if($x == 1) {
26         print("Número positivo\n");
27     }
28     else {
29         if($x == 0) {
30             print("Zero\n");
31         }
32         else {
33             print("Número negativo\n");
34         }
35     }
36 }
37
38 micro11()
```

3.3 Códigos em linguagem PIR

Nesta seção, são apresentados os códigos em linguagem PIR dos pseudo-códigos propostos. Tal linguagem é de mais alto nível aceita pela Parrot VM comparada à linguagem PASM, a qual será apresentada mais adiante. Para compilar e executar os arquivos com extensão ".pl", basta utilizar o seguinte comando no terminal:

```
> parrot nome_arquivo.pir
```

3.3.1 Nano Programas

Listagem 3.47: Módulo mínimo que caracteriza um programa

```
1 .sub main
2
3 .end
```

Listagem 3.48: Declaração de uma variável

```
1 .sub main
2
3 .end
```

Listagem 3.49: Atribuição de um inteiro à uma variável

```
1 .sub main
2     .local num n
3     n = 1
4 .end
```

Listagem 3.50: Atribuição de uma soma de inteiros à uma variável

```
1 .sub main
2     .local num n
3     n = 1 + 2
4 .end
```

Listagem 3.51: Inclusão do comando de impressão

```
1 .sub main
2     .local num n
3     n = 2
4     print n
5 .end
```

Listagem 3.52: Atribuição de uma subtração de inteiros à uma variável

```
1 .sub main
2     .local num n
3     n = 1 - 2
4     print n
5 .end
```


Listagem 3.53: Inclusão do comando condicional

```

1 .sub main
2     .local num n
3     n = 1
4     if n == 1 goto PRINT
5
6 PRINT: say n
7
8 .end

```

Listagem 3.54: Inclusão do comando condicional com parte senão

```

1 .sub main
2     .local num n
3     n = 1
4     if n == 1 goto PRINT1
5     goto PRINT0
6
7 PRINT1: say n
8         goto END
9
10 PRINT0: say 0
11         goto END
12
13 END: end
14
15 .end

```

Listagem 3.55: Atribuição de duas operações aritméticas sobre inteiros a uma variável

```

1 .sub main
2     .local num n
3     n = 1 + 1
4     n = n/2
5     if n == 1 goto PRINTN
6     goto PRINT0
7
8 PRINTN: say n
9         goto END
10
11 PRINT0: say 0
12     goto END
13
14 END: end
15
16 .end

```

Listagem 3.56: Atribuição de duas variáveis inteiras

```

1 .sub main
2     .local num n, m
3     n = 1
4     m = 2
5     if n == m goto PRINTN
6     goto PRINT0
7
8 PRINTN: say n
9         goto END

```

```

10
11 PRINT0: say 0
12     goto END
13
14 END: end
15
16 .end

```

Listagem 3.57: Introdução do comando de repetição enquanto

```

1 .sub main
2     .local num n, m, x
3     n = 1
4     m = 2
5     x = 5
6
7 LOOP: if x <= n goto END
8     n = n + m
9     say n
10    goto LOOP
11
12 END: end
13
14 .end

```

Listagem 3.58: Comando condicional aninhado em um comando de repetição

```

1 .sub main
2     .local num n, m, x
3     n = 1
4     m = 2
5     x = 5
6
7 LOOP: if x <= n goto END
8     if n == m goto PRINTN
9     goto PRINT0
10
11 PRINTN: say n
12         dec x
13         goto LOOP
14
15 PRINT0: say 0
16         dec x
17         goto LOOP
18
19
20 END: end
21
22 .end

```

3.3.2 Micro Programas

Listagem 3.59: Converte graus Celsius para Fahrenheit

```

1 .loadlib 'io_ops'
2
3 .sub main

```

3.3

```
4     .local num cel, far
5     say "Tabela de conversao: Celsius -> Fahrenheit"
6     print "Digite a temperatura em Celsius: "
7     read $S1, 3
8     set cel, $S1
9     far = 9*cel
10    far += 160
11    far = far / 5
12    print "A nova temperatura eh: "
13    print far
14    say " F"
15    goto END
16
17 END: end
18
19 .end
```

Listagem 3.60: Ler dois inteiros e decide qual é maior

```
1 .loadlib 'io_ops'
2
3 .sub main
4     .local num num1, num2
5     print "Digite o primeiro numero: "
6     read $S1, 3
7     set num1, $S1
8     print "Digite o segundo numero: "
9     read $S2, 3
10    set num2, $S2
11    if num1 > num2 goto PRINT1
12    goto PRINT2
13
14 PRINT1: print "O primeiro numero "
15         print num1
16         print " eh maior que o segundo "
17         say num2
18    goto END
19
20 PRINT2: print "O segundo numero "
21         print num2
22         print " eh maior que o primeiro "
23         say num1
24    goto END
25
26 END: end
27
28 .end
```

Listagem 3.61: Lê um número e verifica se ele está entre 100 e 200

```
1 .loadlib 'io_ops'
2
3 .sub main
4     .local num numero
5     print "Digite um numero: "
6     read $S1, 3
7     set numero, $S1
8     if numero >= 100 goto VERIF200
9     goto PRINTNAO
```

```

10
11 VERIF200: if numero <= 200 goto PRINTSIM
12         goto PRINTNAO
13
14 PRINTSIM: print "O numero "
15         print numero
16         say " pertence ao intervalo entre 100 e 200"
17         goto END
18
19 PRINTNAO: print "O numero "
20         print numero
21         say " nao pertence ao intervalo entre 100 e 200"
22         goto END
23
24 END: end
25
26 .end

```

Listagem 3.62: Lê números e informa quais estão entre 10 e 150

```

1 .loadlib 'io_ops'
2
3 .sub main
4     .local num x, num1, intervalo
5     x = 1
6     intervalo = 0
7
8 LOOP: if x > 5 goto END
9     inc x
10    print "Digite um numero: "
11    read $S1, 4
12    set num1, $S1
13    if num1 >= 10 goto VERIF150
14    goto LOOP
15
16 VERIF150: if num1 <= 150 goto INC
17         goto LOOP
18
19 INC: inc intervalo
20     goto LOOP
21
22 END: print "Ao total, foram digitados "
23     print intervalo
24     say " numeros no intervalo entre 10 e 150"
25     end
26
27 .end

```

Listagem 3.63: Lê strings e caracteres

```

1 .loadlib 'io_ops'
2
3 .sub main
4     .local string nome, sexo
5     .local num x, h, m
6     x = 1
7     h = 0
8     m = 0
9

```

3.3

```
10 LOOP: if x > 5 goto END
11     inc x
12     print "Digite um nome: "
13     read $S1, 6
14     set nome, $S1
15     print "H - Homem, M - Mulher: "
16     read $S2, 1
17     set sexo, $S2
18     print sexo
19     if sexo == "H" goto INCH
20     if sexo == "M" goto INCM
21     say "Sexo so poder ser H ou M!!!"
22     goto LOOP
23
24 INCH: inc h
25     goto LOOP
26
27 INCM: inc m
28     goto LOOP
29
30
31 END: print "Foram inseridos "
32     print h
33     say " homens"
34     print "Foram inseridos "
35     print m
36     say " mulheres"
37     end
38
39 .end
```

Listagem 3.64: Escreve um número lido por extenso

```
1 .loadlib 'io_ops'
2
3 .sub main
4     .local num numero
5     print "Digite um numero de 1 a 5: "
6     read $S1, 2
7     set numero, $S1
8     if numero == 1 goto PRINT1
9     if numero == 2 goto PRINT2
10    if numero == 3 goto PRINT3
11    if numero == 4 goto PRINT4
12    if numero == 5 goto PRINT5
13    goto PRINT_INVALIDO
14
15 PRINT1: say "Um"
16     goto END
17
18 PRINT2: say "Dois"
19     goto END
20
21 PRINT3: say "Tres"
22     goto END
23
24 PRINT4: say "Quatro"
25     goto END
26
```

```

27 PRINT5: say "Cinco"
28     goto END
29
30 PRINT_INVALIDO: say "Numero invalido!!!"
31     goto END
32
33 END: end
34
35 .end

```

Listagem 3.65: Decide se os números são positivos, zeros ou negativos

```

1 .loadlib 'io_ops'
2
3 .sub main
4     .local num programa, numero
5     .local string opc
6     programa = 1
7
8 LOOP: if programa != 1 goto END
9     print "Digite um numero: "
10    read $S1, 4
11    set numero, $S1
12    if numero > 0 goto PRINT_POS
13    goto VERIF
14
15 VERIF: if numero == 0 goto PRINT_ZERO
16        if numero < 0 goto PRINT_NEG
17        goto LOOP
18
19 PRINT_POS: say "Positivo"
20        goto OPCA0
21
22 PRINT_ZERO: say "Zero"
23        goto OPCA0
24
25 PRINT_NEG: say "Negativo"
26        goto OPCA0
27
28 OPCA0: print "Deseja continuar (S/N): "
29        read $S2, 1
30        set opc, $S2
31        if opc == "N" goto FINALIZA
32        goto LOOP
33
34 FINALIZA: programa = 0
35        goto LOOP
36
37 END: end
38
39 .end

```

Listagem 3.66: Decide se um número é maior ou menor que 10

```

1 .loadlib 'io_ops'
2
3 .sub main
4     .local num numero
5

```

3.3

```
6      numero = 1
7
8 LOOP: if numero == 0 goto END
9      print "Digite um numero: "
10     read $S1, 4
11     set numero, $S1
12     if numero > 10 goto PRINT_MAIOR
13     goto PRINT_MENOR
14
15 PRINT_MAIOR: print "O numero "
16             print numero
17             say " eh maior que 10"
18             goto LOOP
19
20 PRINT_MENOR: print "O numero "
21             print numero
22             say " eh menor que 10"
23             goto LOOP
24
25 END: end
26
27 .end
```

Listagem 3.67: Cálculo de preços

```
1 .loadlib 'io_ops'
2
3 .sub main
4     .local num preco, venda, novo_preco, aux
5
6     print "Digite o preco: "
7     read $S1, 5
8     set preco, $S1
9     print "Digite a venda: "
10    read $S2, 5
11    set venda, $S2
12
13 VERIF1: if venda < 500 goto NOVO1
14     if preco < 30 goto NOVO1
15     goto VERIF2
16
17 VERIF2: if venda >= 500 goto VERIF_NOVO2
18     goto VERIF2_1
19
20 VERIF2_1: if preco >= 30 goto VERIF_NOVO2_2
21     goto VERIF3
22
23 VERIF3: if preco >= 80 goto NOVO3
24     if venda >= 1200 goto NOVO3
25     goto END
26
27 VERIF_NOVO2: if venda < 1200 goto NOVO2
28     goto VERIF2_1
29
30 VERIF_NOVO2_2: if preco < 80 goto NOVO2
31     goto VERIF3
32
33 NOVO1: novo_preco = 10 * preco
34     novo_preco = novo_preco / 100
```

```

35     novo_preco = preco + novo_preco
36     goto END
37
38 NOVO2: novo_preco = 15 * preco
39     novo_preco = novo_preco / 100
40     novo_preco = preco + novo_preco
41     goto END
42
43 NOVO3: aux = 20 * preco
44     aux = aux / 100
45     novo_preco = preco - aux
46     goto END
47
48
49 END: print "O novo preco eh "
50     say novo_preco
51     end
52
53 .end

```

Listagem 3.68: Calcula o fatorial de um número

```

1 .loadlib 'io_ops'
2
3 .sub main
4     .local num numero, fat
5
6     print "Digite um numero: "
7     read $S1, 3
8     set numero, $S1
9     fat = fatorial(numero)
10    print "O fatorial de "
11    print numero
12    print " eh "
13    say fat
14    end
15
16 .end
17
18 .sub fatorial
19     .param int n
20     .local num res
21
22     set res, n
23
24 LOOP: if n <= 1 goto RETURN
25     dec n
26     res = res * n
27     goto LOOP
28
29 RETURN: .return(res)
30
31 .end

```

Listagem 3.69: Decide se um número é positivo, zero ou negativo com auxílio de uma função

```

1 .loadlib 'io_ops'
2
3 .sub main

```


3.4

```
4      .local num numero, x
5
6      print "Digite um numero: "
7      read $S1, 3
8      set numero, $S1
9      x = verifica(numero)
10     if x == 1 goto PRINT_POS
11     if x == 0 goto PRINT_ZERO
12     goto PRINT_NEG
13
14 PRINT_POS: say "Numero positivo"
15           end
16
17 PRINT_NEG: say "Numero negativo"
18           end
19
20 PRINT_ZERO: say "Zero"
21           end
22
23 .end
24
25 .sub verifica
26     .param int n
27     .local num res
28
29     if n > 0 goto POS
30     if n == 0 goto ZERO
31     goto NEG
32
33 POS: res = 1
34     goto RETURN
35
36 ZERO: res = 0
37     goto RETURN
38
39 NEG: res = -1
40     goto RETURN
41
42 RETURN: .return(res)
43
44 .end
```

3.4 Tradução para Parrot Assembly

Nesta seção serão apresentados os códigos mostrados nas duas seções anteriores traduzidos para a linguagem Assembly de mais baixo nível do Parrot, o PASM. Além disso, serão apresentados os tipos de registradores e as principais instruções do PASM, sendo a maioria utilizada para a escrita dos códigos nessa linguagem.

3.4.1 Registradores presentes no PASM

Como já dito na introdução, a Parrot VM é baseada em registradores, não em pilha, como em outras máquinas virtuais, pois trabalha como um processador de forma a ter um

rápido acesso aos registradores. Dessa forma, o assembly adotado pela Parrot VM, a Parrot Assembly ou PASM, possui quatro tipos de registradores:

- **Tipo I:** registradores do tipo I são responsáveis por armazenar dados de valores inteiros.

Exemplo: registradores I1, I2, I3 em um código PASM.

- **Tipo N:** registradores do tipo N são responsáveis por armazenar dados de valores ponto flutuantes.

Exemplo: registradores N1, N2, N3 em um código PASM.

- **Tipo S:** registradores do tipo S são responsáveis por armazenar dados de cadeias de caracteres ou strings.

Exemplo: registradores S1, S2, S3 em um código PASM.

- **Tipo PMC:** registradores do tipo PMC são responsáveis por armazenar dados de objetos ou classes. Este tipo não foi utilizado até o momento para compor os códigos PASM aqui escritos.

3.4.2 Comandos mais utilizados no PASM

Para facilitar a visualização e explicação dos principais comandos utilizados pelo PASM, estes estarão organizados em uma tabela, como visto na tabela 3.1. A primeira coluna, **Instrução**, contém o nome dos comandos utilizados em ordem alfabética. A segunda coluna, **Parâmetros**, contém os parâmetros necessários para utilizar os comandos das respectivas linhas.

Explicação das instruções

A seguir, segue uma pequena explicação da função de cada comando da tabela com alguns exemplos. Destaca-se que o parâmetro "Valor a ser atribuído" pode significar tanto um valor do tipo inteiro, ponto flutuante, string, quanto o valor associado ao registrador invocado pelo comando.

- **Instrução add:** representa a soma de dois valores que são guardados em um registrador de destino.

Instrução	Parâmetros
add	Registrador de Destino, Valor a ser atribuído, Valor a ser atribuído
branch	Nome da função
dec	Registrador a ser decrementado
div	Registrador de Destino, Divisor, Dividendo
end	Não possui parâmetros
eq	Valor a ser atribuído, Valor a ser atribuído, Nome da função
ge	Valor a ser atribuído, Valor a ser atribuído, Nome da função
gt	Valor a ser atribuído, Valor a ser atribuído, Nome da função
inc	Registrador a ser incrementado
le	Valor a ser atribuído, Valor a ser atribuído, Nome da função
lt	Valor a ser atribuído, Valor a ser atribuído, Nome da função
mul	Registrador de Destino, Valor a ser atribuído, Valor a ser atribuído
ne	Valor a ser atribuído, Valor a ser atribuído, Nome da função
print	Valor a ser atribuído
read	Registrador de Destino, Número de caracteres a serem lidos
say	Valor a ser atribuído
set	Registrador de Destino, Valor a ser atribuído
sub	Registrador de Destino, Valor a ser atribuído, Valor a ser atribuído

Tabela 3.1: *Principais comandos em PASM*

Exemplos:

add I1, I2, I3: significa que será armazenado no registrador I1 o resultado da soma dos valores associados aos registradores I2 e I3.

add I1, 2, 1: significa que será armazenado no registrador I1 o resultado da soma dos valores inteiros 1 e 2.

add I1, 1, I2: significa que será armazenado no registrador I1 o resultado da soma do valor inteiro 1 com o valor associado ao registrador I2.

- **Instrução branch:** representa um pulo para o nome da função que recebe como parâmetro.

Exemplo:

branch LOOP: significa que ao chegar na linha dessa instrução, a execução pulará para a função "LOOP:".

- **Instrução dec:** representa o decremento em 1 do valor associado ao registrador que recebe como parâmetro.

Exemplo:

dec I1: significa que o valor associado a I1 será decrementado em 1. Se I1 era igual a 10, após essa instrução I1 será igual a 9.

- **Instrução `div`:** representa a divisão de dois valores e que são guardados em um registrador de destino.

Exemplos:

`div I1, I2, I3`: significa que será armazenado no registrador I1 o resultado da divisão dos valores associados aos registradores I2 e I3.

`div I1, 2, 1`: significa que será armazenado no registrador I1 o resultado da divisão de 2 por 1.

`div I1, I2, 2`: significa que será armazenado no registrador I1 o resultado da divisão do valor associado ao registrador I2 por 2.

- **Instrução `end`:** representa o fim do programa. Não recebe nenhum parâmetro.

Exemplo:

`end`.

- **Instrução `eq` (**equal**):** recebe como parâmetro dois valores a serem comparados e o nome de uma função. Se os valores recebidos forem iguais, o programa pula para o nome da função recebida.

Exemplos:

`eq I1, I2, FIM`: significa que se os valores associados aos registradores I1 e I2 forem iguais, a execução pula para a função "FIM".

`eq I1, 1, LOOP`: significa que se o valor associado ao registrador I1 for igual a 1, o programa pula para a função "LOOP".

- **Instrução `ge` (**greater than or equal**):** recebe como parâmetro dois valores a serem comparados e o nome de uma função. Se o primeiro valor for maior ou igual ao segundo valor recebido, o programa pula para o nome da função recebida.

Exemplo:

`ge I1, I2, FIM`: significa que se o valor associado ao registrador I1 for maior ou igual ao valor em I2, o programa pula para a função "FIM".

- **Instrução `gt` (**greater than**):** recebe como parâmetro dois valores a serem comparados e o nome de uma função. Se o primeiro valor for maior do que o segundo valor recebido, o programa pula para o nome da função recebida.

Exemplo:

`gt I1, I2, FIM`: significa que se o valor associado ao registrador I1 for maior do que o valor em I2, o programa pula para a função "FIM".

- **Instrução `inc`**: ao contrário da instrução `dec`, representa o incremento em 1 do valor associado ao registrador que recebe como parâmetro.

Exemplo:

`inc I1`: significa que o valor associado a I1 será incrementado em 1. Se I1 era igual a 10, após essa instrução I1 será igual a 11.

- **Instrução `le (less than or equal)`**: recebe como parâmetro dois valores a serem comparados e o nome de uma função. Se o primeiro valor for menor ou igual ao segundo valor recebido, o programa pula para o nome da função recebida.

Exemplo:

`le I1, I2, FIM`: significa que se o valor associado ao registrador I1 for menor ou igual ao valor em I2, o programa pula para a função "FIM".

- **Instrução `lt (less than)`**: recebe como parâmetro dois valores a serem comparados e o nome de uma função. Se o primeiro valor for menor do que o segundo valor recebido, o programa pula para o nome da função recebida.

Exemplo:

`lt I1, I2, FIM`: significa que se o valor associado ao registrador I1 for menor do que o valor em I2, o programa pula para a função "FIM".

- **Instrução `mul`**: representa o produto de dois valores e que é guardado em um registrador de destino.

Exemplos:

`mul I1, I2, I3`: significa que será armazenado no registrador I1 o resultado do produto entre os valores associados aos registradores I2 e I3.

`mul I1, 2, 1`: significa que será armazenado no registrador I1 o resultado do produto entre os valores inteiros 1 e 2.

`mul I1, 1, I2`: significa que será armazenado no registrador I1 o resultado do produto entre o valor inteiro 1 com o valor associado ao registrador I2.

- **Instrução `ne (not equal)`**: recebe como parâmetro dois valores a serem comparados e o nome de uma função. Se os valores recebidos forem diferentes, o programa pula para o nome da função recebida.

Exemplos:

ne I1, I2, FIM: significa que se os valores associados aos registradores I1 e I2 forem diferentes, a execução pula para a função "FIM".

ne I1, 1, LOOP: significa que se o valor associado ao registrador I1 não for igual a 1, o programa pula para a função "LOOP".

- **Instrução `print`:** recebe como algum valor a ser escrito na tela para visualização do usuário.

Exemplos:

`print "Fim do Programa":` exibe tal mensagem na tela.

`print I1:` exibe o valor associado ao registrador I1 na tela.

- **Instrução `read`:** recebe como algum valor do tipo String a ser digitado na tela pelo usuário, além do tamanho desse valor. Utiliza-se sempre o registrador do tipo S. Além disso, é necessário importar a biblioteca "io_ops" para ser possível o uso desta instrução.

Exemplos:

`read S1, 1:` lê um valor de até 1 caracter digitado pelo usuário.

`read S1, 5:` lê um valor de até 5 caracteres digitados pelo usuário.

- **Instrução `say`:** mesma função do comando `print`, porém com quebra de linha ao final da impressão.

Exemplos:

`say "Fim do Programa":` exibe tal mensagem na tela e pula uma linha.

`say I1:` exibe o valor associado ao registrador I1 na tela e pula uma linha.

- **Instrução `set`:** armazena no registrador de destino o valor recebido como segundo parâmetro.

Exemplos:

`set I1, 1:` armazena o valor 1 no registrador I1.

`set I1, S1:` armazena o valor associado ao registrador S1 no registrador I1.

- **Instrução `sub`:** representa a subtração de dois valores e que são guardados em um registrador de destino.

Exemplos:

sub I1, I2, I3: significa que será armazenado no registrador I1 o resultado da soma dos valores associados aos registradores I2 e I3.

sub I1, 2, 1: significa que será armazenado no registrador I1 o resultado da soma dos valores inteiros 1 e 2.

sub I1, 1, I2: significa que será armazenado no registrador I1 o resultado da soma do valor inteiro 1 com o valor associado ao registrador I2.

3.4.3 Nano Programas

1 - Nano01:

Listagem 3.70: Módulo mínimo que caracteriza um programa

```
1 end
```

A caracterização mínima de um programa em PASM é representada apenas pelo comando `end`, que significa o fim do programa.

2 - Nano02:

Listagem 3.71: Declaração de uma variável

```
1 end
```

Não é necessário declarar variáveis em PASM, pois é uma linguagem baseada em registradores. Dessa forma, todo registrador inicializado deve conter algum valor referenciado.

3 - Nano03:

Listagem 3.72: Atribuição de um inteiro à uma variável

```
1 MAIN:
2 set I1, 1
3 end
```

Para realizar a atribuição de um valor inteiro a uma variável, utiliza-se a instrução `set`, um registrador do tipo I e um valor inteiro. Dessa forma, a instrução `set` da linha 2 significa que o valor inteiro 1 foi atribuído ao registrador I1.

4 - Nano04:

Listagem 3.73: Atribuição de uma soma de inteiros à uma variável

```
1 MAIN:
2 add I1, 1, 2
3 end
```

É possível representar a atribuição de uma soma de inteiros à um registrador em PASM a partir do comando `add`. Dessa forma, a instrução na linha 2 representa que o registrador I1 foi inicializado com a soma de 1 e 2, ou seja, I1 é igual a 3.

5 - Nano05:

Listagem 3.74: Inclusão do comando de impressão

```
1 MAIN:
2 set I1, 2
3 print I1
4 end
```

Na linha 2 é atribuído o valor 2 ao registrador I1. Na linha 3, o comando `print` exibe o valor do registrador I1 na tela do usuário.

6 - Nano06:

Listagem 3.75: Atribuição de uma subtração de inteiros à uma variável

```
1 MAIN:
2 sub I1, 1, 2
3 print I1
4 end
```

Para representar a atribuição de uma subtração de inteiros a um registrador, utiliza-se a instrução `sub`. Dessa forma, o registrador I1 é inicializado com a subtração do valor 1 e do valor 2 na linha 2, ou seja, I1 é igual a -1. Em seguida, o valor atribuído a I1 é exibido para o usuário.

7 - Nano07:

Listagem 3.76: Inclusão do comando condicional

```
1 MAIN:
2 set I1, 1
3 eq I1, 1, PRINT
4 branch END
5
6 PRINT:
7 print I1
8 branch END
9
10 END:
11 end
```

Neste código, pode-se perceber a presença de três funções: MAIN, PRINT e END. Na primeira função, onde a execução é iniciada, é atribuído o valor 1 ao registrador I1 pela instrução `set`. Em seguida, o comando `eq` faz a verificação se o valor atribuído a I1 é igual a 1. Caso positivo, o código pula para a função PRINT, na qual o valor do registrador I1 é exibido na tela do usuário e, por meio da instrução `branch`, pula-se para a função END, na qual o programa é encerrado. Caso negativo, pula-se para o fim do programa.

8 - Nano08:

Listagem 3.77: Inclusão do comando condicional com parte senão

```
1 MAIN:
2 set I1, 1
3 eq I1, 1, PRINT1
4 branch PRINT2
```


3.4

```
5
6 PRINT1:
7 print I1
8 branch END
9
10 PRINT2:
11 print 0
12 branch END
13
14 END:
15 end
```

Neste código, pode-se perceber que é muito semelhante ao anterior, com a diferença de que quando a condição na linha 3 não for atingida, o código pula para a função PRINT2, na qual é exibido o valor 0 ao usuário e o programa é encerrado.

9 - Nano09:

Listagem 3.78: Atribuição de duas operações aritméticas sobre inteiros a uma variável

```
1 MAIN:
2 div I1, 1, 2
3 add I1, I1, 1
4 eq I1, 1, PRINT
5 print 0
6 branch END
7
8 PRINT:
9 print I1
10 branch END
11
12 END:
13 end
```

Na linha 2, pode-se notar que a instrução `div` atribui o valor da divisão de 1 por 2 ao registrador I1. Após isso, o registrador recebe a soma de 1 ao seu próprio valor na linha 3. Em seguida, é verificado se o valor do registrador I1 é igual a 1. Caso positivo, o valor de I1 é exibido para o usuário. Caso negativo, é exibido o valor 0 para o usuário.

10 - Nano10:

Listagem 3.79: Atribuição de duas variáveis inteiras

```
1 MAIN:
2 set I1, 1
3 set I2, 2
4 eq I1, I2, PRINT
5 print 0
6 branch END
7
8 PRINT:
9 print I1
10 branch END
11
12 END:
13 end
```

Nas linhas 2 e 3, os valores 1 e 2 são atribuídos aos registradores I1 e I2, respectivamente. Na linha 4, é verificado se os valores de I1 e I2 são iguais. Caso positivo, é exibido o valor do registrador I1 ao usuário. Caso negativo, é exibido o valor 0 e encerra-se o programa.

11 - Nano11:

Listagem 3.80: Introdução do comando de repetição enquanto

```

1 MAIN:
2 set I1, 1
3 set I2, 2
4 set I3, 5
5 branch LOOP
6
7 LOOP:
8 ge I1, I3, END
9 add I1, I1, I2
10 print I1
11 branch LOOP
12
13 END:
14 end

```

Nas linhas 2, 3 e 4, os valores 1, 2 e 5 são atribuídos aos registradores I1, I2 e I3, respectivamente e pula-se para a função LOOP. Em seguida, verifica-se se o valor de I1 é maior ou igual ao valor de I3 por meio da instrução ge. Caso positivo, encerra-se o programa. Caso negativo, o registrador I1 recebe a soma do seu conteúdo com o valor associado a I2, o valor de I1 é exibido na tela e pula-se para o começo da função LOOP, na linha 8.

Listagem 3.81: Comando condicional aninhado em um comando de repetição

```

1 MAIN:
2 set I1, 1
3 set I2, 2
4 set I3, 5
5 branch LOOP
6
7 LOOP:
8 ge I1, I3, END
9 eq I1, I2, PRINT
10 print 0
11 dec I3
12 branch LOOP
13
14 PRINT:
15 print I1
16 branch END
17
18 END:
19 end

```

Nas linhas 2, 3 e 4, os valores 1, 2 e 5 são atribuídos aos registradores I1, I2 e I3, respectivamente e pula-se para a função LOOP. Em seguida, verifica-se se o valor de I1 é maior ou igual ao valor de I3 por meio da instrução ge. Caso positivo, encerra-se o programa. Caso negativo, verifica-se se o valor de I1 é igual ao valor de I2 pela instrução eq. Caso positivo, o valor de I1 é exibido para o usuário e encerra-se o programa. Caso negativo, o valor 0 é

exibido para o usuário, o valor de I3 é decrementado em 1 pela instrução `dec` e pula-se para o início do LOOP, na linha 8.

3.4.4 Micro Programas

13 - Micro01:

Listagem 3.82: Converte graus Celsius para Fahrenheit

```

1 .loadlib 'io_ops'
2
3 MAIN:
4 print "Tabela de Conversao: Celsius - Fahrenheit\n"
5 print "Temperatura em Celsius: "
6 read S1, 3
7 set I1, S1
8 mul N1, I1, 9
9 add N1, N1, 160
10 div N1, N1, 5
11 print "\nA nova temperatura eh: "
12 print N1
13 print " F\n"
14 end

```

Na linha 1, pode-se perceber que a biblioteca `io_ops` foi importada para o código, permitindo que seja possível receber o valor da temperatura digitado pelo usuário pelo comando `read` no formato de String, por isso é utilizado um registrador do tipo S. Após isso, transforma-se a String em inteiro pelo comando `set` e a conversão de Celsius para Fahrenheit é realizada por meio de operações de multiplicação, soma e divisão, representadas respectivamente pelos comandos `mul`, `add`, `div`.

14 - Micro02:

Listagem 3.83: Ler dois inteiros e decide qual é maior

```

1 .loadlib 'io_ops'
2
3 MAIN:
4 print "Primeiro numero: "
5 read S1, 1
6 set N1, S1
7 print "Segundo numero: "
8 read S2, 3
9 set N2, S2
10 gt N1, N2, PRINT_MAIOR_I1
11 branch PRINT_MAIOR_I2
12
13 PRINT_MAIOR_I1:
14 print "\nO primeiro numero "
15 print N1
16 print " eh maior que o segundo "
17 print N2
18 branch END
19
20 PRINT_MAIOR_I2:
21 print "\nO segundo numero "

```

```

22 print N2
23 print " eh maior que o primeiro "
24 print N1
25 branch END
26
27 END:
28 end

```

Neste código, na função MAIN são recebidos dois valores digitados pelo usuário e armazenados nos registradores N1 e N2. Na linha 5, a instrução gt verifica se o valor associado a N1 é maior que o valor associado a N2. Caso positivo, é exibida a mensagem que o primeiro número é maior que o segundo. Caso negativo, é exibida a mensagem que o segundo número é maior que o primeiro.

15 - Micro03:

Listagem 3.84: Lê um número e verifica se ele está entre 100 e 200

```

1 .loadlib 'io_ops'
2
3 MAIN:
4 print "Numero: "
5 read S1, 3
6 set I1, S1
7 ge I1, 100, VERIF
8 branch PRINT_NAO_INTERVALO
9
10 VERIF:
11 ge 200, I1, PRINT_INTERVALO
12 branch PRINT_NAO_INTERVALO
13
14 PRINT_NAO_INTERVALO:
15 print "\nO numero nao esta no intervalo de 100 a 200\n"
16 branch END
17
18 PRINT_INTERVALO:
19 print "\nO numero esta no intervalo de 100 a 200\n"
20 branch END
21
22 END:
23 end

```

Neste código, é armazenado um número inteiro digitado pelo usuário no registrador I1. Em seguida, verifica-se se seu valor é maior ou igual a 100. Caso positivo, verifica-se se o número 200 é maior ou igual ao associado a I1, se sim, é exibida a mensagem que o número está no intervalo entre 100 e 200. Caso o valor de I1 não seja maior ou igual a 100 e não seja menor ou igual a 200, é exibida a mensagem que o número não está no intervalo entre 100 e 200.

16 - Micro04:

Listagem 3.85: Lê números e informa quais estão entre 10 e 150

```

1 .loadlib 'io_ops'
2
3 MAIN:
4 set I2, 1

```

3.4

```
5 set I3, 0
6 branch LOOP
7
8 LOOP:
9 gt I2, 5, FIM
10 inc I2
11 print "\nNumero: "
12 read S1, 4
13 set I1, S1
14 print I1
15 ge I1, 10, VERIF
16 branch LOOP
17
18 VERIF:
19 gt I1, 150, LOOP
20 inc I3
21 branch LOOP
22
23 FIM:
24 print "\nAo total, foram digitados "
25 print I3
26 print " numeros no intervalo entre 10 e 150\n"
27 end
```

Neste código, percebe-se um loop de cinco iterações em que cada uma verifica-se se o número digitado pelo usuário está no intervalo entre 10 e 150. Ao final, é exibido ao usuário a quantidade de números digitados e que pertencem a esse intervalo.

17 - Micro05:

Listagem 3.86: Lê strings e caracteres

```
1 .loadlib 'io_ops'
2
3 INICIO:
4 set I2, 1
5 set I3, 0
6 set I4, 0
7 branch LOOP
8
9 LOOP:
10 gt I2, 5, FIM
11 inc I2
12 print "Digite o nome: "
13 read S1, 5
14 print "\nH - Homem ou M - Mulher: "
15 read S2, 1
16 eq S2, "H", INC_H
17 eq S2, "M", INC_M
18 print "\nSexo so pode se H ou M!!"
19 branch LOOP
20
21 INC_H:
22 inc I3
23 branch LOOP
24
25 INC_M:
26 inc I4
27 branch LOOP
```

```

28
29 FIM:
30 print "\n\nForam inseridos "
31 print I3
32 print " homens"
33 print "\nForam inseridos "
34 print I4
35 print " mulheres\n"
36 end

```

Neste programa, percebe-se um loop de cinco iterações em que cada uma o usuário digita um nome e o sexo (H-homem ou M-mulher) de uma pessoa. Ainda no loop, verifica-se o sexo digitado na iteração e incrementa-se o registrador correspondente: I3 armazena o número de vezes que "H" foi digitado e I4 armazena o número de vezes que "M" foi digitado.

18 - Micro06:

Listagem 3.87: Escreve um número lido por extenso

```

1 .loadlib 'io_ops'
2
3 MAIN:
4 print "Numero: "
5 read S1, 2
6 set I1, S1
7 eq I1, 1, UM
8 eq I1, 2, DOIS
9 eq I1, 3, TRES
10 eq I1, 4, QUATRO
11 eq I1, 5, CINCO
12 branch INVALIDO
13
14 UM:
15 print "\nUm\n"
16 branch END
17
18 DOIS:
19 print "\nDois\n"
20 branch END
21
22 TRES:
23 print "\nTres\n"
24 branch END
25
26 QUATRO:
27 print "\nQuatro\n"
28 branch END
29
30 CINCO:
31 print "\nCinco\n"
32 branch END
33
34 INVALIDO:
35 print "\nNumero Invalido!!\n"
36 branch END
37
38 END:
39 end

```

Neste código, o usuário digita um número. Como não há o comando condicional "Switch" em PASM, é necessário usar várias vezes a instrução `eq` de forma a comparar o valor entrado pelo usuário com os números 1, 2, 3, 4 e 5.

19 - Micro07:

Listagem 3.88: Decide se os números são positivos, zeros ou negativos

```

1 .loadlib 'io_ops'
2
3 INICIO:
4 set I1, 1 #programa
5 branch LOOP
6
7 LOOP:
8 ne I1, 1, FIM
9 print "\nNumero: "
10 read S1, 2
11 set I2, S1
12 gt I2, 0, PRINT1
13 eq I2, 0, PRINT2
14 branch PRINT3
15 branch VERIF
16
17 PRINT1:
18 print "\nPositivo"
19 branch VERIF
20
21 PRINT2:
22 print "\nZERO"
23 branch VERIF
24
25 PRINT3:
26 print "\nNEGATIVO"
27 branch VERIF
28
29 VERIF:
30 print "\nDeseja finalizar (S/N) "
31 read S2, 2
32 eq S2, "S\n", FIM
33 branch LOOP
34
35 FIM:
36 print " S\n"
37 end

```

Neste código, percebe-se um loop em que a condição de parada é verificada pela instrução `ne`, ou seja, caso o valor associado a `I1` não for igual a 1, o programa é terminado. Caso contrário, armazena o número digitado pelo usuário no registrador `I2` e exibe para o usuário se o valor é positivo, negativo ou igual a zero. Após isso, verifica-se se o usuário deseja finalizar ou não o programa.

20 - Micro08:

Listagem 3.89: Decide se um número é maior ou menor que 10

```

1 .loadlib 'io_ops'
2

```

```

3 INICIO:
4 set I1, 1
5 branch LOOP
6
7 LOOP:
8 eq I1, 0, FIM
9 print "\nNumero: "
10 read S1, 3
11 set I1, S1
12 gt I1, 10, PRINT_MAIOR
13 branch PRINT_MENOR
14
15 PRINT_MAIOR:
16 print "\nO numero "
17 print I1
18 print " eh maior que 10"
19 branch LOOP
20
21 PRINT_MENOR:
22 print "\nO numero "
23 print I1
24 print " eh menor que 10"
25 branch LOOP
26
27 FIM:
28 end

```

Neste código, o usuário digita um número a cada iteração, então é exibido para o usuário se o número é maior ou menor do que 10. O programa termina caso o usuário entre com o número zero.

21 - Micro09:

Listagem 3.90: Cálculo de preços

```

1 .loadlib 'io_ops'
2
3 INICIO:
4 print "\nPreco: "
5 read S1, 5
6 set N1, S1
7 print "\nVenda: "
8 read S2, 5
9 set N2, S2
10 lt N2, 500, PRECO1 #venda < 500
11 lt N1, 30, PRECO1 #ou preco < 30
12 ge N2, 500, VERIF_SE1 #venda >= 500
13 branch VERIF_SE1_1
14
15 VERIF_SE1:
16 lt N2, 1200, PRECO2 #venda < 1200
17 branch VERIF_SE1_1
18
19 VERIF_SE1_1:
20 ge N1, 30, VERIF_SE1_1_2 #preco >= 30
21 branch VERIF_SE2
22
23 VERIF_SE1_1_2:
24 lt N1, 80, PRECO2 #preco < 80

```


3.4

```
25 branch VERIF_SE2
26
27 VERIF_SE2:
28 ge N2, 1200, PRECO3 #venda >= 1200
29 ge N1, 80, PRECO3 #preco >= 80
30 branch FIM
31
32
33 PRECO1: #(venda < 500 ou preco < 30)
34 set N3, N1
35 mul N3, N3, 10
36 div N3, N3, 100
37 add N3, N3, N1
38 branch FIM
39
40 PRECO2: #(venda <= 500 e venda < 1200)
41 #ou (preco >= 30 e preco < 80)
42 set N3, N1
43 mul N3, N3, 15
44 div N3, N3, 100
45 add N3, N3, N1
46 branch FIM
47
48 PRECO3: #(venda >= 1200 ou preco >= 80)
49 set N3, N1
50 mul N3, N3, 20
51 div N3, N3, 100
52 sub N3, N1, N3
53 branch FIM
54
55
56 FIM:
57 print "\n0 novo preco eh "
58 print N3
59 print "\n"
60 end
```

Neste código, o usuário digita o valor do preço e da venda, sendo retornado o valor do novo preço. O tamanho do código tem muito a ver sobre o fato de não ter como representar os valores booleanos and e or no PASM. Dessa forma, foi necessário implementar vários laços condicionais a partir das instruções lt, ge.

22 - Micro10:

Listagem 3.91: Calcula o fatorial de um número

```
1 .loadlib 'io_ops'
2
3 INICIO:
4 print "\nNumero: "
5 read S1, 2
6 set I0, S1
7 set I1, I0
8 branch INIT_FATORIAL
9
10 INIT_FATORIAL:
11 set I2, I1
12
```

```

13 FATORIAL:
14 eq  I1, 1, FIM
15 dec I1
16 mul I2, I1
17 branch FATORIAL
18
19 FIM:
20 print "\nO fatorial de "
21 print I0
22 print " eh "
23 print I2
24 print "\n"
25 end

```

Neste código, o usuário entra com um valor para calcular o fatorial e ele é gravado nos registradores I0 e I1, vale destacar que I0 é usado somente para imprimir o número ao final do programa. Após isso, o mesmo valor é gravado no registrador I2 e pula-se para a função recursiva FATORIAL. O caso base é quando o valor do registrado I1 é igual a 0, chegando ao fim do programa exibindo o fatorial do número desejado. Caso não seja 1, decrementa-se I1 e faz-se o produto do resultado armazenado até o momento em I2 com I1 e a função FATORIAL é chamada novamente.

23 - Micro11:

Listagem 3.92: Decide se um número é positivo, zero ou negativo com auxílio de uma função

```

1 .loadlib 'io_ops'
2
3 INICIO:
4 print "\nNumero: "
5 read  S1, 2
6 set  I1, S1
7 branch VERIFICA
8
9 VERIFICA:
10 gt  I1, 0, RES1
11 lt  I1, 0, RES2
12 set  I2, 0
13 branch VERIF_RES
14
15 RES1:
16 set  I2, 1
17 branch VERIF_RES
18
19 RES2:
20 set  I2, -1
21 branch VERIF_RES
22
23 VERIF_RES:
24 eq  I2, 1, PRINT1
25 eq  I2, 0, PRINT2
26 print "\nNegativo\n"
27 branch FIM
28
29 PRINT1:
30 print "\nPositivo\n"
31 branch FIM
32

```

```

33 PRINT2:
34 print "\nZero\n"
35 branch FIM
36
37 FIM:
38 end

```

Neste código, percebe-se que há duas funções principais: VERIFICA e VERIFICA_RES. A primeira armazena o número digitado pelo usuário no registrador I1 e verifica se ele é maior que 0, igual a 0 ou menor que 0. Além disso, chama as funções que armazenam no registrador I2 o valor 1 caso o valor associado a I1 é positivo, 0 caso seja o número zero e -1 caso seja um número negativo, depois pula para a função VERIFICA_RES. Nela, é verificado o valor associado a I2 e exibe para o usuário se é positivo, negativo ou igual a zero.

3.4.5 Execução de um código PASM

Para compilar e executar um código em linguagem Assembly do Parrot, PASM, com extensão ".pasm", é necessário digitar o seguinte comando no terminal:

```
> parrot nome_arquivo.pasm
```

Exemplo de execução:

Para compilar e executar o código do arquivo `micro01.pasm`, no qual recebe um valor para converter a temperatura em Celsius para Fahrenheit, basta navega até a pasta onde se localiza o referido arquivo no terminal e executar o comando especificado anteriormente, como visto na figura 3.1.

```

matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-Pasm$ ls
micro01.pasm  micro04.pasm  micro08.pasm  nano01.pasm  nano05.pasm  nano09.pasm
micro01.pbc  micro05.pasm  micro09.pasm  nano02.pasm  nano06.pasm  nano10.pasm
micro02.pasm  micro06.pasm  micro10.pasm  nano03.pasm  nano07.pasm  nano11.pasm
micro03.pasm  micro07.pasm  micro11.pasm  nano04.pasm  nano08.pasm  nano12.pasm
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-Pasm$ parrot micro01.pasm
Tabela de Conversão: Celsius - Fahrenheit
Temperatura em Celsius: 30

A nova temperatura eh: 86 F
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-Pasm$

```

Figura 3.1: Execução de um código PASM

3.4.6 Conversão de PASM para Bytecode

É possível gerar o arquivo bytecode a partir de programas escritos em PASM pelo seguinte comando no terminal:

```
> parrot -o nome_arquivo_destino.pbc nome_arquivo_origem.pasm
```

Para executá-lo, basta realizar a seguinte instrução:

```
> parrot nome_arquivo.pbc
```

Pode-se ver esse processo de conversão e execução de um arquivo bytecode "PBC" na figura 3.2.

```
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-Pasm$ ls
micro01.pasm  micro05.pasm  micro09.pasm  nano02.pasm  nano06.pasm  nano10.pasm
micro02.pasm  micro06.pasm  micro10.pasm  nano03.pasm  nano07.pasm  nano11.pasm
micro03.pasm  micro07.pasm  micro11.pasm  nano04.pasm  nano08.pasm  nano12.pasm
micro04.pasm  micro08.pasm  nano01.pasm  nano05.pasm  nano09.pasm
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-Pasm$ parrot -o micro01.pbc micro01.pasm
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-Pasm$ ls
micro01.pasm  micro04.pasm  micro08.pasm  nano01.pasm  nano05.pasm  nano09.pasm
micro01.pbc   micro05.pasm  micro09.pasm  nano02.pasm  nano06.pasm  nano10.pasm
micro02.pasm  micro06.pasm  micro10.pasm  nano03.pasm  nano07.pasm  nano11.pasm
micro03.pasm  micro07.pasm  micro11.pasm  nano04.pasm  nano08.pasm  nano12.pasm
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-Pasm$ parrot micro01.pbc
Tabela de Conversao: Celsius - Fahrenheit
Temperatura em Celsius: 30

A nova temperatura eh: 86 F
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-Pasm$
```

Figura 3.2: Conversão e execução de um código PBC

3.4.7 Conversão de PIR para PASM

É possível também transformar um código escrito em linguagem PIR para um código em linguagem PASM. Para isso, o primeiro passo é converter o arquivo PIR desejado para um arquivo PBC, bytecode, pelo seguinte comando:

```
> parrot -o nome_arquivo_destino.pbc nome_arquivo_origem.pir
```

Após realizar o comando acima, é necessário desmontar o arquivo PBC gerado para PASM pela instrução:

```
> pbc_disassemble -b nome_arquivo.pbc
```

Ou, então, pode-se desmontar o arquivo PBC para PASM a um arquivo desejado pelo seguinte comando:

```
> pbc_disassemble -b -o nome_arquivo_destino.pasm nome_arquivo_origem.pbc
```

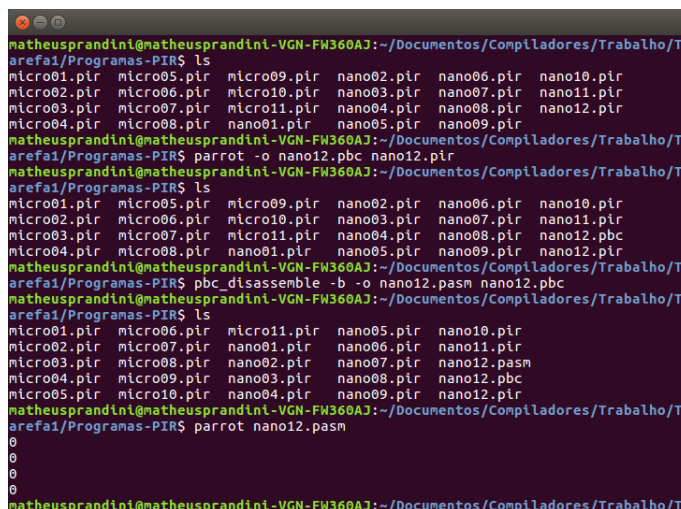
Exemplo código PASM gerado:

Como exemplo, a conversão do código PIR `nano12.pir` gera o seguinte código PASM:

Listagem 3.93: Comando condicional aninhado em um comando de repetição

```
1 =head1 Constant-table
2
3 STR_CONST(0): nano12.pir
4 STR_CONST(1): main
5 STR_CONST(2):
6 STR_CONST(3): parrot
7 PMC_CONST(0):
8
9 =cut
10
11 # Current Source Filename 'nano12.pir'
12 set_n_ic N2,1
13 set_n_ic N1,2
14 set_n_ic N0,5
15 L4: le_n_n_ic N0,N2,L1
16 eq_n_n_ic N2,N1,L2
17 branch_ic L3
18 L2: say_n N2
19 dec_n N0
20 branch_ic L4
21 L3: say_ic 0
22 dec_n N0
23 branch_ic L4
24 L1:      end
```

Toda essa sequência de instruções para converter um código PIR para PASM é ilustrada pela figura 3.3.



```
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-PIRS$ ls
micro01.pir  micro05.pir  micro09.pir  nano02.pir  nano06.pir  nano10.pir
micro02.pir  micro06.pir  micro10.pir  nano03.pir  nano07.pir  nano11.pir
micro03.pir  micro07.pir  micro11.pir  nano04.pir  nano08.pir  nano12.pir
micro04.pir  micro08.pir  nano01.pir  nano05.pir  nano09.pir  nano12.pir
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-PIRS$ parrot -o nano12.pbc nano12.pir
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-PIRS$ ls
micro01.pir  micro05.pir  micro09.pir  nano02.pir  nano06.pir  nano10.pir
micro02.pir  micro06.pir  micro10.pir  nano03.pir  nano07.pir  nano11.pir
micro03.pir  micro07.pir  micro11.pir  nano04.pir  nano08.pir  nano12.pbc
micro04.pir  micro08.pir  nano01.pir  nano05.pir  nano09.pir  nano12.pir
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-PIRS$ pbc_disassemble -b -o nano12.pasm nano12.pbc
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-PIRS$ ls
micro01.pir  micro06.pir  micro11.pir  nano05.pir  nano10.pir
micro02.pir  micro07.pir  nano01.pir  nano06.pir  nano11.pir
micro03.pir  micro08.pir  nano02.pir  nano07.pir  nano12.pasm
micro04.pir  micro09.pir  nano03.pir  nano08.pir  nano12.pbc
micro05.pir  micro10.pir  nano04.pir  nano09.pir  nano12.pir
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
arefa1/Programas-PIRS$ parrot nano12.pasm
0
0
0
0
0
matheusprandini@matheusprandini-VGN-FW360AJ:~/Documentos/Compiladores/Trabalho/T
```

Figura 3.3: Conversão de um código PIR para PASM

Capítulo 4

Analizador Léxico

Nesta seção será mostrado a implementação do analisador sintático da linguagem MiniLua feita em linguagem Ocaml, baseado nos programas nanos e micros encontrados na seção 3.1. Além disso, será executado o analisador léxico para cada um deles e também será realizada análises de erros de forma proposital.

4.1 Reconhecimento das Palavras Reservadas em Lua

Foram criados tokens para cada palavra reservada encontrada em linguagem Lua, da seguinte forma: à esquerda é representado o nome do token criado e à direita é representado o vetor de caracteres (string) do token.

- **AND:** "and";
- **BREAK:** "break";
- **DO:** "do";
- **ELSE:** "else";
- **ELSEIF:** "elseif";
- **EOF:** representa fim do arquivo;
- **END:** "end";
- **FALSE:** "false";
- **FOR:** "for";
- **FUNCAO:** "function";
- **IF:** "if";
- **IN:** "in";
- **IO_READ:** "io.read";

- **LOCAL:** "local";
- **NIL:** "nil";
- **NOT:** "not";
- **NUMBER_INPUT:** *number;
- **OR:** "or";
- **PRINT:** "print";
- **REPEAT:** "repeat";
- **RETURN:** "return";
- **THEN:** "then";
- **TRUE:** "true";
- **UNTIL:** "until";
- **WHILE:** "while";

Da mesma forma, foram criados tokens para os principais operadores aritméticos, lógicos e sinais de pontuação:

- **ABRE_CHAVE:** '{'
- **ABRE_COLCHETE:** '['
- **ADICAO:** '+'
- **AND_BINARIO:** ''
- **APAR:** '('
- **ATRIB:** '='
- **CONCATENA:** '..'
- **DIV_POR_2:** '»'
- **DIVISAO:** '/'
- **DIVISAO_INTEIRO:** '//'
- **DOIS_PONTOS:** ':'
- **EQUIVALENTE:** '=='
- **EXPONENCIACAO:** ''
- **FECHA_CHAVE:** '}'
- **FECHA_COLCHETE:** ']'

- **FPAR:** `') '`
- **MAIOR:** `' > '`
- **MAIOR_OU_IGUAL:** `' >= '`
- **MENOR:** `' < '`
- **MENOR_OU_IGUAL:** `' <= '`
- **MODULO:** `' % '`
- **MULTIPLICACAO:** `' * '`
- **MULT_POR_2:** `' « '`
- **NAO_EQUIVALENTE:** `' = '`
- **OR_BINARIO:** `' | '`
- **PONTO:** `' . '`
- **PONTO_VIRGULA:** `' ; '`
- **RETICENCIAS:** `' ... '`
- **SUBTRACAO:** `' - '`
- **TAMANHO:** `' ' '`
- **VIRGULA:** `' , '`

Outros tokens:

- **LITINT of int:** dígitos;
- **LITSTRING of string:** qualquer expressão entre aspas `()`;
- **ID of string:** qualquer expressão não inicializada por dígitos;

4.2 Código do Analisador Léxico

Listagem 4.1: Código do Analisador Léxico

```

1 {
2   open Lexing
3   open Printf
4
5   let incr_num_linha lexbuf =
6     let pos = lexbuf.lex_curr_p in
7     lexbuf.lex_curr_p <- { pos with
8       pos_lnum = pos.pos_lnum + 1;
9       pos_bol = pos.pos_cnum;
10    }
```


4.2

```
11
12 let msg_erro lexbuf c =
13     let pos = lexbuf.lex_curr_p in
14     let lin = pos.pos_lnum
15     and col = pos.pos_cnum - pos.pos_bol - 1 in
16     sprintf "%d-%d: caracter desconhecido %c" lin col c
17
18 let erro lin col msg =
19     let mensagem = sprintf "%d-%d: %s" lin col msg in
20     failwith mensagem
21
22 type tokens = ABRE_CHAVE
23             | ABRE_COLCHETE
24             | ADICAO
25             | AND
26             | AND_BINARIO
27             | APAR
28             | ATRIB
29             | BREAK
30             | CONCATENA
31             | DIV_POR_2
32             | DIVISAO
33             | DIVISAO_INTEIRO
34             | DO
35             | DOIS_PONTOS
36             | ELSE
37             | ELSEIF
38             | END
39             | EQUIVALENTE
40             | EXPONENCIACAO
41             | FALSE
42             | FECHA_CHAVE
43             | FECHA_COLCHETE
44             | FOR
45             | FPAR
46             | FUNCAO
47             | IF
48             | IN
49             | IO_READ
50             | LOCAL
51             | MAIOR
52             | MAIOR_OU_IGUAL
53             | MENOR
54             | MENOR_OU_IGUAL
55             | MODULO
56             | MULT_POR_2
57             | MULTIPLICACAO
58             | NAO_EQUIVALENTE
59             | NIL
60             | NOT
61             | NUMBER_INPUT
62             | OR
63             | OR_BINARIO
64             | PONTO
65             | PONTO_VIRGULA
66             | PRINT
67             | REPEAT
68             | RETICENCIAS
69             | RETURN
```

```

70      | SUBTRACAO
71      | TAMANHO
72      | THEN
73      | TRUE
74      | UNTIL
75      | VIRGULA
76      | WHILE
77      | LITINT of int
78      | LITSTRING of string
79      | ID of string
80      | EOF
81 }
82
83 let digito = ['0' - '9']
84 let inteiro = digito+
85
86 let letra = ['a' - 'z' 'A' - 'Z']
87 let identificador = letra ( letra | digito | '_' ) *
88
89 let brancos = [' ' '\t'] +
90 let novalinha = '\r' | '\n' | "\r\n"
91
92 let comentario = "--" [ ^ '\r' '\n' ] *
93
94 rule token = parse
95   brancos      { token lexbuf }
96 | novalinha    { incr_num_linha lexbuf; token lexbuf }
97 | "--[" { let pos = lexbuf.lex_curr_p in
98           let lin = pos.pos_lnum
99           and col = pos.pos_cnum - pos.pos_bol - 1 in
100      comentario_bloco lin col 0 lexbuf }
101 | comentario { token lexbuf }
102 | "("        { APAR }
103 | "{"        { ABRE_CHAVE }
104 | "["        { ABRE_COLCHETE }
105 | "+"        { ADICAO }
106 | "-"        { SUBTRACAO }
107 | ")"        { FPAR }
108 | "}"        { FECHA_CHAVE }
109 | "]"        { FECHA_COLCHETE }
110 | ","        { VIRGULA }
111 | "."        { PONTO }
112 | ";"        { PONTO_VIRGULA }
113 | ":"        { DOIS_PONTOS }
114 | "=="       { EQUIVALENTE }
115 | "~="       { NAO_EQUIVALENTE }
116 | ">="       { MAIOR_OU_IGUAL }
117 | "<="       { MENOR_OU_IGUAL }
118 | "/"        { DIVISAO }
119 | "*"        { MULTIPLICACAO }
120 | "%"        { MODULO }
121 | "^"        { EXPONENCIACAO }
122 | ">"        { MAIOR }
123 | "<"        { MENOR }
124 | "="        { ATRIB }
125 | "#"        { TAMANHO }
126 | "<<"       { MULT_POR_2 }
127 | ">>"       { DIV_POR_2 }
128 | "//"       { DIVISAO_INTEIRO }

```

```

129 | "&"          { AND_BINARIO }
130 | "|"          { OR_BINARIO }
131 | ".."         { CONCATENA }
132 | "..."      { RETENCENCIAS }
133 | "and"        { AND }
134 | "break"      { BREAK }
135 | "do"         { DO }
136 | "else"       { ELSE }
137 | "elseif"     { ELSEIF }
138 | "end"        { END }
139 | "false"      { FALSE }
140 | "for"        { FOR }
141 | "function"   { FUNCAO }
142 | "if"         { IF }
143 | "io.read"    { IO_READ }
144 | "in"         { IN }
145 | "local"      { LOCAL }
146 | "nil"        { NIL }
147 | "not"        { NOT }
148 | "print"      { PRINT }
149 | "or"         { OR }
150 | "repeat"     { REPEAT }
151 | "return"     { RETURN }
152 | "then"       { THEN }
153 | "true"       { TRUE }
154 | "until"      { UNTIL }
155 | "while"      { WHILE }
156 | inteiro as num { let numero = int_of_string num in
157 |                 LITINT numero }
158 | identificador as id { ID id }
159 | '"'          { let pos = lexbuf.lex_curr_p in
160 |                 let lin = pos.pos_lnum
161 |                 and col = pos.pos_cnum - pos.pos_bol - 1 in
162 |                 let buffer = Buffer.create 1 in
163 |                 let str = leia_string lin col buffer lexbuf in
164 |                 LITSTRING str }
165 | _ as c      { failwith (msg_erro lexbuf c) }
166 | eof         { EOF }
167
168 and comentario_bloco lin col n = parse
169   "--]]"      { if n=0 then token lexbuf
170 |              else comentario_bloco lin col (n-1) lexbuf }
171 | "--[["      { comentario_bloco lin col (n+1) lexbuf }
172 | novalinha   { incr_num_linha lexbuf; comentario_bloco lin col n lexbuf }
173 | _           { comentario_bloco lin col n lexbuf }
174 | eof         { erro lin col "Comentario nao fechado" }
175
176 and leia_string lin col buffer = parse
177   '"'         { Buffer.contents buffer}
178 | "\\t"       { Buffer.add_char buffer '\t'; leia_string lin col buffer
179 |             lexbuf }
179 | "\\n"       { Buffer.add_char buffer '\n'; leia_string lin col buffer
180 |             lexbuf }
180 | '\\\' \'\' { Buffer.add_char buffer '\''; leia_string lin col buffer
181 |             lexbuf }
181 | '\\\' \'\'\' { Buffer.add_char buffer '\\\''; leia_string lin col buffer
182 |             lexbuf }
182 | _ as c      { Buffer.add_char buffer c; leia_string lin col buffer lexbuf
183 |             }

```

```
183 | eof          { erro lin col "A string nao foi fechada"}
```

4.3 Execução do Analisador Léxico

Os seguintes passos são necessários para executar o analisador léxico escrito na linguagem Ocaml:

1. Entrar no terminal e compilar o arquivo lexico.mll para gerar o arquivo lexico.ml:

```
> ocamllex lexico.mll
```

2. Compilar o código gerado:

```
> ocamlc -c lexico.ml
```

3. Entrar no Ocaml com o programa rlwrap, o qual grava os comandos executados no terminal:

```
> rlwrap ocaml
```

4. Carregar o analisador léxico pelo arquivo carregador.ml:

```
> #use "carregador.ml";;
```

5. Entrar com um código a ser analisado:

```
> lex "nome_codigo";;
```

4.4 Testes do Analisador Léxico

Os testes foram realizados com o uso dos programas apresentados na seção 3.1.

4.4.1 Nano Programas

Nano01

Saída do analisador léxico:

Listagem 4.2: Saída do analisador léxico para o programa nano01

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.END;
3 Lexico.EOF]
```

Nano02

Saída do analisador léxico:

Listagem 4.3: Saída do analisador léxico para o programa nano02

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.END; Lexico.EOF]
```

Nano03

Saída do analisador léxico:

Listagem 4.4: Saída do analisador léxico para o programa nano03

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.ID "n"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.END;
4 Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]
```

Nano04

Saída do analisador léxico:

Listagem 4.5: Saída do analisador léxico para o programa nano04

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.ID "n"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.
  ADICAO;
4 Lexico.LITINT 2; Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR;
5 Lexico.EOF]
```

Nano05

Saída do analisador léxico:

Listagem 4.6: Saída do analisador léxico para o programa nano05

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.ID "n"; Lexico.ATRIB; Lexico.LITINT 2; Lexico.PRINT
  ;
4 Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.END; Lexico.ID "main";
5 Lexico.APAR; Lexico.FPAR; Lexico.EOF]
```

Nano06

Saída do analisador léxico:

Listagem 4.7: Saída do analisador léxico para o programa nano06

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.ID "n"; Lexico.ATRIB; Lexico.LITINT 1;
4 Lexico.SUBTRACAO; Lexico.LITINT 2; Lexico.PRINT; Lexico.APAR; Lexico.ID "
  n";
5 Lexico.FPAR; Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR;
6 Lexico.EOF]
```

Nano07

Saída do analisador léxico:

Listagem 4.8: Saída do analisador léxico para o programa nano07

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.ID "n"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.IF;
4 Lexico.ID "n"; Lexico.EQUIVALENTE; Lexico.LITINT 1; Lexico.THEN;
5 Lexico.PRINT; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.END;
6 Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]
```

Nano08

Saída do analisador léxico:

Listagem 4.9: Saída do analisador léxico para o programa nano08

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.ID "n"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.IF;
4 Lexico.ID "n"; Lexico.EQUIVALENTE; Lexico.LITINT 1; Lexico.THEN;
5 Lexico.PRINT; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.ELSE;
6 Lexico.PRINT; Lexico.APAR; Lexico.LITINT 0; Lexico.FPAR; Lexico.END;
7 Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]
```

Nano09

Saída do analisador léxico:

Listagem 4.10: Saída do analisador léxico para o programa nano09

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.ID "n"; Lexico.ATRIB; Lexico.APAR; Lexico.LITINT 1;
4 Lexico.ADICAO; Lexico.LITINT 1; Lexico.FPAR; Lexico.DIVISAO;
```

```

5 Lexico.LITINT 2; Lexico.IF; Lexico.ID "n"; Lexico.EQUIVALENTE;
6 Lexico.LITINT 1; Lexico.THEN; Lexico.PRINT; Lexico.APAR; Lexico.ID "n";
7 Lexico.FPAR; Lexico.ELSE; Lexico.PRINT; Lexico.APAR; Lexico.LITINT 0;
8 Lexico.FPAR; Lexico.END; Lexico.END; Lexico.ID "main"; Lexico.APAR;
9 Lexico.FPAR; Lexico.EOF]

```

Nano10

Saída do analisador léxico:

Listagem 4.11: Saída do analisador léxico para o programa nano10

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.VIRGULA; Lexico.ID "m"; Lexico.ID "n"; Lexico.ATRIB
4 ;
5 Lexico.LITINT 1; Lexico.ID "m"; Lexico.ATRIB; Lexico.LITINT 2; Lexico.IF;
6 Lexico.ID "n"; Lexico.EQUIVALENTE; Lexico.ID "m"; Lexico.THEN; Lexico.
7 PRINT;
8 Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.ELSE; Lexico.PRINT;
9 Lexico.APAR; Lexico.LITINT 0; Lexico.FPAR; Lexico.END; Lexico.END;
10 Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

Nano11

Saída do analisador léxico:

Listagem 4.12: Saída do analisador léxico para o programa nano11

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.VIRGULA; Lexico.ID "m"; Lexico.VIRGULA; Lexico.ID "
4 x";
5 Lexico.ID "n"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.ID "m"; Lexico.ATRIB
6 ;
7 Lexico.LITINT 2; Lexico.ID "x"; Lexico.ATRIB; Lexico.LITINT 5; Lexico.
8 WHILE;
9 Lexico.ID "x"; Lexico.MAIOR; Lexico.ID "n"; Lexico.DO; Lexico.ID "n";
10 Lexico.ATRIB; Lexico.ID "n"; Lexico.ADICAO; Lexico.ID "m"; Lexico.PRINT;
11 Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.END; Lexico.END;
12 Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

Nano12

Saída do analisador léxico:

Listagem 4.13: Saída do analisador léxico para o programa nano12

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "n"; Lexico.VIRGULA; Lexico.ID "m"; Lexico.VIRGULA; Lexico.ID "
4 x";

```

```

4 Lexico.ID "n"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.ID "m"; Lexico.ATRIB
;
5 Lexico.LITINT 2; Lexico.ID "x"; Lexico.ATRIB; Lexico.LITINT 5; Lexico.
WHILE;
6 Lexico.ID "x"; Lexico.MAIOR; Lexico.ID "n"; Lexico.DO; Lexico.IF;
7 Lexico.ID "n"; Lexico.EQUIVALENTE; Lexico.ID "m"; Lexico.THEN; Lexico.
PRINT;
8 Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.ELSE; Lexico.PRINT;
9 Lexico.APAR; Lexico.LITINT 0; Lexico.FPAR; Lexico.END; Lexico.ID "x";
10 Lexico.ATRIB; Lexico.ID "x"; Lexico.SUBTRACAO; Lexico.LITINT 1; Lexico.
END;
11 Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

4.4.2 Micro Programas

Micro01

Saída do analisador léxico:

Listagem 4.14: Saída do analisador léxico para o programa micro01

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "cel"; Lexico.VIRGULA; Lexico.ID "far"; Lexico.PRINT; Lexico.
APAR;
4 Lexico.LITSTRING "Tabela de convers\195\163o: Celsius -> Fahrenheit";
5 Lexico.FPAR; Lexico.PRINT; Lexico.APAR;
6 Lexico.LITSTRING "Digite a temperatura em Celsius: "; Lexico.FPAR;
7 Lexico.ID "cel"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
8 Lexico.LITSTRING "*number"; Lexico.FPAR; Lexico.ID "far"; Lexico.ATRIB;
9 Lexico.APAR; Lexico.LITINT 9; Lexico.MULTIPLICACAO; Lexico.ID "cel";
10 Lexico.ADICAO; Lexico.LITINT 160; Lexico.FPAR; Lexico.DIVISAO;
11 Lexico.LITINT 5; Lexico.PRINT; Lexico.APAR;
12 Lexico.LITSTRING "A nova temperatura eh: "; Lexico.CONCATENA;
13 Lexico.ID "far"; Lexico.CONCATENA; Lexico.LITSTRING " F"; Lexico.FPAR;
14 Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

Micro02

Saída do analisador léxico:

Listagem 4.15: Saída do analisador léxico para o programa micro02

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "num1"; Lexico.VIRGULA; Lexico.ID "num2"; Lexico.PRINT;
4 Lexico.APAR; Lexico.LITSTRING "Digite o primeiro numero: "; Lexico.FPAR;
5 Lexico.ID "num1"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
6 Lexico.LITSTRING "*number"; Lexico.FPAR; Lexico.PRINT; Lexico.APAR;
7 Lexico.LITSTRING "Digite o segundo numero: "; Lexico.FPAR; Lexico.ID "
num2";
8 Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR; Lexico.LITSTRING "*number";
9 Lexico.FPAR; Lexico.IF; Lexico.ID "num1"; Lexico.MAIOR; Lexico.ID "num2";

```



```

10 Lexico.THEN; Lexico.PRINT; Lexico.APAR;
11 Lexico.LITSTRING "O primeiro n\195\186mero "; Lexico.CONCATENA;
12 Lexico.ID "num1"; Lexico.CONCATENA;
13 Lexico.LITSTRING " \195\169 maior que o segundo "; Lexico.CONCATENA;
14 Lexico.ID "num2"; Lexico.FPAR; Lexico.ELSE; Lexico.PRINT; Lexico.APAR;
15 Lexico.LITSTRING "O segundo n\195\186mero "; Lexico.CONCATENA;
16 Lexico.ID "num2"; Lexico.CONCATENA;
17 Lexico.LITSTRING " \195\169 maior que o primeiro "; Lexico.CONCATENA;
18 Lexico.ID "num1"; Lexico.FPAR; Lexico.END; Lexico.END; Lexico.ID "main";
19 Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

Micro03

Saída do analisador léxico:

Listagem 4.16: Saída do analisador léxico para o programa micro03

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "numero"; Lexico.PRINT; Lexico.APAR;
4 Lexico.LITSTRING "Digite um n\195\186mero: "; Lexico.FPAR;
5 Lexico.ID "numero"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
6 Lexico.LITSTRING "*number"; Lexico.FPAR; Lexico.IF; Lexico.ID "numero";
7 Lexico.MAIOR_OU_IGUAL; Lexico.LITINT 100; Lexico.THEN; Lexico.IF;
8 Lexico.ID "numero"; Lexico.MENOR_OU_IGUAL; Lexico.LITINT 200; Lexico.THEN
9 ;
10 Lexico.PRINT; Lexico.APAR;
11 Lexico.LITSTRING "O n\195\186mero est\195\161 no intervalo entre 100 e
12 200";
13 Lexico.FPAR; Lexico.ELSE; Lexico.PRINT; Lexico.APAR;
14 Lexico.LITSTRING
15 "O n\195\186mero n\195\163o est\195\161 no intervalo entre 100 e 200";
16 Lexico.FPAR; Lexico.END; Lexico.ELSE; Lexico.PRINT; Lexico.APAR;
17 Lexico.LITSTRING
18 "O n\195\186mero n\195\163o est\195\161 no intervalo entre 100 e 200";
19 Lexico.FPAR; Lexico.END; Lexico.END; Lexico.ID "main"; Lexico.APAR;
20 Lexico.FPAR; Lexico.EOF]

```

Micro04

Saída do analisador léxico:

Listagem 4.17: Saída do analisador léxico para o programa micro04

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "x"; Lexico.VIRGULA; Lexico.ID "num"; Lexico.VIRGULA;
4 Lexico.ID "intervalo"; Lexico.ID "intervalo"; Lexico.ATRIB; Lexico.LITINT
5 0;
6 Lexico.FOR; Lexico.ID "x"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.VIRGULA;
7 Lexico.LITINT 5; Lexico.VIRGULA; Lexico.LITINT 1; Lexico.DO; Lexico.PRINT
8 ;
9 Lexico.APAR; Lexico.LITSTRING "Digite um n\195\186mero: "; Lexico.FPAR;
10 Lexico.ID "num"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;

```

```

9 Lexico.LITSTRING "*number"; Lexico.FPAR; Lexico.IF; Lexico.ID "num";
10 Lexico.MAIOR_OU_IGUAL; Lexico.LITINT 10; Lexico.THEN; Lexico.IF;
11 Lexico.ID "num"; Lexico.MENOR_OU_IGUAL; Lexico.LITINT 150; Lexico.THEN;
12 Lexico.ID "intervalo"; Lexico.ATRIB; Lexico.ID "intervalo"; Lexico.ADICAO
    ;
13 Lexico.LITINT 1; Lexico.END; Lexico.END; Lexico.END; Lexico.PRINT;
14 Lexico.APAR; Lexico.LITSTRING "Ao total, foram digitados ";
15 Lexico.CONCATENA; Lexico.ID "intervalo"; Lexico.CONCATENA;
16 Lexico.LITSTRING " n\195\186meros no intervalo entre 10 e 150"; Lexico.
    FPAR;
17 Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

Micro05

Saída do analisador léxico:

Listagem 4.18: Saída do analisador léxico para o programa micro05

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "nome"; Lexico.VIRGULA; Lexico.ID "sexo"; Lexico.LOCAL;
4 Lexico.ID "x"; Lexico.VIRGULA; Lexico.ID "h"; Lexico.VIRGULA; Lexico.ID "
    m";
5 Lexico.ATRIB; Lexico.LITINT 1; Lexico.VIRGULA; Lexico.LITINT 0;
6 Lexico.VIRGULA; Lexico.LITINT 0; Lexico.FOR; Lexico.ID "x"; Lexico.ATRIB;
7 Lexico.LITINT 1; Lexico.VIRGULA; Lexico.LITINT 5; Lexico.VIRGULA;
8 Lexico.LITINT 1; Lexico.DO; Lexico.PRINT; Lexico.APAR;
9 Lexico.LITSTRING "Digite o nome: "; Lexico.FPAR; Lexico.ID "nome";
10 Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR; Lexico.LITSTRING "*line";
11 Lexico.FPAR; Lexico.PRINT; Lexico.APAR;
12 Lexico.LITSTRING "H - Homem ou M - Mulher: "; Lexico.FPAR; Lexico.ID "
    sexo";
13 Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR; Lexico.LITSTRING "*line";
14 Lexico.FPAR; Lexico.IF; Lexico.ID "sexo"; Lexico.EQUIVALENTE;
15 Lexico.LITSTRING "H"; Lexico.THEN; Lexico.ID "h"; Lexico.ATRIB;
16 Lexico.ID "h"; Lexico.ADICAO; Lexico.LITINT 1; Lexico.ELSEIF;
17 Lexico.ID "sexo"; Lexico.EQUIVALENTE; Lexico.LITSTRING "M"; Lexico.THEN;
18 Lexico.ID "m"; Lexico.ATRIB; Lexico.ID "m"; Lexico.ADICAO; Lexico.LITINT
    1;
19 Lexico.ELSE; Lexico.PRINT; Lexico.APAR;
20 Lexico.LITSTRING "Sexo s\195\179 pode ser H ou M!"; Lexico.FPAR; Lexico.
    END;
21 Lexico.END; Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "Foram inseridos
    ";
22 Lexico.CONCATENA; Lexico.ID "h"; Lexico.CONCATENA;
23 Lexico.LITSTRING " Homens"; Lexico.FPAR; Lexico.PRINT; Lexico.APAR;
24 Lexico.LITSTRING "Foram inseridos "; Lexico.CONCATENA; Lexico.ID "m";
25 Lexico.CONCATENA; Lexico.LITSTRING " Mulheres"; Lexico.FPAR; Lexico.END;
26 Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

Micro06

Saída do analisador léxico:

Listagem 4.19: Saída do analisador léxico para o programa micro06

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "num"; Lexico.PRINT; Lexico.APAR;
4 Lexico.LITSTRING "Digite um numero de 1 a 5: "; Lexico.FPAR;
5 Lexico.ID "num"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
6 Lexico.LITSTRING "*number"; Lexico.FPAR; Lexico.IF; Lexico.ID "num";
7 Lexico.EQUIVALENTE; Lexico.LITINT 1; Lexico.THEN; Lexico.PRINT; Lexico.
  APAR;
8 Lexico.LITSTRING "Um"; Lexico.FPAR; Lexico.ELSEIF; Lexico.ID "num";
9 Lexico.EQUIVALENTE; Lexico.LITINT 2; Lexico.THEN; Lexico.PRINT; Lexico.
  APAR;
10 Lexico.LITSTRING "Dois"; Lexico.FPAR; Lexico.ELSEIF; Lexico.ID "num";
11 Lexico.EQUIVALENTE; Lexico.LITINT 3; Lexico.THEN; Lexico.PRINT; Lexico.
  APAR;
12 Lexico.LITSTRING "Tres"; Lexico.FPAR; Lexico.ELSEIF; Lexico.ID "num";
13 Lexico.EQUIVALENTE; Lexico.LITINT 4; Lexico.THEN; Lexico.PRINT; Lexico.
  APAR;
14 Lexico.LITSTRING "Quatro"; Lexico.FPAR; Lexico.ELSEIF; Lexico.ID "num";
15 Lexico.EQUIVALENTE; Lexico.LITINT 5; Lexico.THEN; Lexico.PRINT; Lexico.
  APAR;
16 Lexico.LITSTRING "Cinco"; Lexico.FPAR; Lexico.ELSE; Lexico.PRINT;
17 Lexico.APAR; Lexico.LITSTRING "Numero Invalido!!!"; Lexico.FPAR; Lexico.
  END;
18 Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

Micro07

Saída do analisador léxico:

Listagem 4.20: Saída do analisador léxico para o programa micro07

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "programa"; Lexico.VIRGULA; Lexico.ID "numero"; Lexico.VIRGULA;
4 Lexico.ID "opc"; Lexico.ID "programa"; Lexico.ATRIB; Lexico.LITINT 1;
5 Lexico.WHILE; Lexico.ID "programa"; Lexico.EQUIVALENTE; Lexico.LITINT 1;
6 Lexico.DO; Lexico.PRINT; Lexico.APAR;
7 Lexico.LITSTRING "Digite um n\195\186mero: "; Lexico.FPAR;
8 Lexico.ID "numero"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
9 Lexico.LITSTRING "*n"; Lexico.FPAR; Lexico.IF; Lexico.ID "numero";
10 Lexico.MAIOR; Lexico.LITINT 0; Lexico.THEN; Lexico.PRINT; Lexico.APAR;
11 Lexico.LITSTRING "Positivo"; Lexico.FPAR; Lexico.ELSE; Lexico.IF;
12 Lexico.ID "numero"; Lexico.EQUIVALENTE; Lexico.LITINT 0; Lexico.THEN;
13 Lexico.PRINT; Lexico.APAR;
14 Lexico.LITSTRING "O n\195\186mero \195\169 igual a 0"; Lexico.FPAR;
15 Lexico.END; Lexico.IF; Lexico.ID "numero"; Lexico.MENOR; Lexico.LITINT 0;
16 Lexico.THEN; Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "Negativo";
17 Lexico.FPAR; Lexico.END; Lexico.END; Lexico.PRINT; Lexico.APAR;
18 Lexico.LITSTRING "Deseja finalizar? (S-1/N-2) "; Lexico.FPAR;
19 Lexico.ID "opc"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
20 Lexico.LITSTRING "*n"; Lexico.FPAR; Lexico.IF; Lexico.ID "opc";
21 Lexico.EQUIVALENTE; Lexico.LITINT 1; Lexico.THEN; Lexico.ID "programa";
22 Lexico.ATRIB; Lexico.LITINT 0; Lexico.END; Lexico.END; Lexico.END;
23 Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

Micro08

Saída do analisador léxico:

Listagem 4.21: Saída do analisador léxico para o programa micro08

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "numero"; Lexico.ID "numero"; Lexico.ATRIB; Lexico.LITINT 1;
4 Lexico.WHILE; Lexico.ID "numero"; Lexico.NAO_EQUIVALENTE; Lexico.LITINT
  0;
5 Lexico.DO; Lexico.PRINT; Lexico.APAR;
6 Lexico.LITSTRING "Digite um n\195\186mero: "; Lexico.FPAR;
7 Lexico.ID "numero"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
8 Lexico.LITSTRING "*n"; Lexico.FPAR; Lexico.IF; Lexico.ID "numero";
9 Lexico.MAIOR; Lexico.LITINT 10; Lexico.THEN; Lexico.PRINT; Lexico.APAR;
10 Lexico.LITSTRING "O n\195\186mero "; Lexico.CONCATENA; Lexico.ID "numero"
   ;
11 Lexico.CONCATENA; Lexico.LITSTRING " \195\169 maior que 10"; Lexico.FPAR;
12 Lexico.ELSE; Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "O n\195\186mero
   ";
13 Lexico.CONCATENA; Lexico.ID "numero"; Lexico.CONCATENA;
14 Lexico.LITSTRING " \195\169 menor que 10"; Lexico.FPAR; Lexico.END;
15 Lexico.END; Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR;
16 Lexico.EOF]
```

Micro09

Saída do analisador léxico:

Listagem 4.22: Saída do analisador léxico para o programa micro09

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "preco"; Lexico.VIRGULA; Lexico.ID "venda"; Lexico.VIRGULA;
4 Lexico.ID "novo_preco"; Lexico.PRINT; Lexico.APAR;
5 Lexico.LITSTRING "Digite o pre\195\167o: "; Lexico.FPAR; Lexico.ID "preco
   ";
6 Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR; Lexico.LITSTRING "*n";
7 Lexico.FPAR; Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "Digite a venda:
   ";
8 Lexico.FPAR; Lexico.ID "venda"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR
   ;
9 Lexico.LITSTRING "*n"; Lexico.FPAR; Lexico.IF; Lexico.ID "venda";
10 Lexico.MENOR; Lexico.LITINT 500; Lexico.OR; Lexico.ID "preco"; Lexico.
   MENOR;
11 Lexico.LITINT 30; Lexico.THEN; Lexico.ID "novo_preco"; Lexico.ATRIB;
12 Lexico.ID "preco"; Lexico.ADICAO; Lexico.LITINT 10; Lexico.DIVISAO;
13 Lexico.LITINT 100; Lexico.MULTIPLICACAO; Lexico.ID "preco"; Lexico.ELSE;
14 Lexico.IF; Lexico.APAR; Lexico.ID "venda"; Lexico.MAIOR_OU_IGUAL;
15 Lexico.LITINT 500; Lexico.AND; Lexico.ID "venda"; Lexico.MENOR;
16 Lexico.LITINT 1200; Lexico.FPAR; Lexico.OR; Lexico.APAR; Lexico.ID "preco
   ";
17 Lexico.MAIOR_OU_IGUAL; Lexico.LITINT 30; Lexico.AND; Lexico.ID "preco";
18 Lexico.MENOR; Lexico.LITINT 80; Lexico.FPAR; Lexico.THEN;
19 Lexico.ID "novo_preco"; Lexico.ATRIB; Lexico.ID "preco"; Lexico.ADICAO;
```

4.4

```
20 Lexico.LITINT 15; Lexico.DIVISAO; Lexico.LITINT 100; Lexico.MULTIPLICACAO
    ;
21 Lexico.ID "preco"; Lexico.ELSE; Lexico.IF; Lexico.ID "venda";
22 Lexico.MAIOR_OU_IGUAL; Lexico.LITINT 1200; Lexico.OR; Lexico.ID "preco";
23 Lexico.MAIOR_OU_IGUAL; Lexico.LITINT 80; Lexico.THEN;
24 Lexico.ID "novo_preco"; Lexico.ATRIB; Lexico.ID "preco"; Lexico.SUBTRACAO
    ;
25 Lexico.LITINT 20; Lexico.DIVISAO; Lexico.LITINT 100; Lexico.MULTIPLICACAO
    ;
26 Lexico.ID "preco"; Lexico.END; Lexico.END; Lexico.END; Lexico.PRINT;
27 Lexico.APAR; Lexico.LITSTRING "O novo pre\195\167o \195\169 ";
28 Lexico.CONCATENA; Lexico.ID "novo_preco"; Lexico.FPAR; Lexico.END;
29 Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]
```

Micro10

Saída do analisador léxico:

Listagem 4.23: Saída do analisador léxico para o programa micro10

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "fatorial"; Lexico.APAR; Lexico.ID "n";
3 Lexico.FPAR; Lexico.IF; Lexico.ID "n"; Lexico.MENOR_OU_IGUAL;
4 Lexico.LITINT 0; Lexico.THEN; Lexico.RETURN; Lexico.LITINT 1; Lexico.ELSE
    ;
5 Lexico.RETURN; Lexico.ID "n"; Lexico.MULTIPLICACAO; Lexico.ID "fatorial";
6 Lexico.APAR; Lexico.ID "n"; Lexico.SUBTRACAO; Lexico.LITINT 1; Lexico.
    FPAR;
7 Lexico.END; Lexico.END; Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR;
8 Lexico.FPAR; Lexico.LOCAL; Lexico.ID "numero"; Lexico.VIRGULA;
9 Lexico.ID "fat"; Lexico.PRINT; Lexico.APAR;
10 Lexico.LITSTRING "Digite um n\195\186mero: "; Lexico.FPAR;
11 Lexico.ID "numero"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
12 Lexico.LITSTRING "*n"; Lexico.FPAR; Lexico.ID "fat"; Lexico.ATRIB;
13 Lexico.ID "fatorial"; Lexico.APAR; Lexico.ID "numero"; Lexico.FPAR;
14 Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "O fatorial de "; Lexico.FPAR
    ;
15 Lexico.PRINT; Lexico.APAR; Lexico.ID "numero"; Lexico.FPAR; Lexico.PRINT;
16 Lexico.APAR; Lexico.LITSTRING " \195\169 "; Lexico.FPAR; Lexico.PRINT;
17 Lexico.APAR; Lexico.ID "fat"; Lexico.FPAR; Lexico.END; Lexico.ID "main";
18 Lexico.APAR; Lexico.FPAR; Lexico.EOF]
```

Micro11

Saída do analisador léxico:

Listagem 4.24: Saída do analisador léxico para o programa micro11

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "verifica"; Lexico.APAR; Lexico.ID "n";
3 Lexico.FPAR; Lexico.LOCAL; Lexico.ID "res"; Lexico.IF; Lexico.ID "n";
4 Lexico.MAIOR; Lexico.LITINT 0; Lexico.THEN; Lexico.ID "res"; Lexico.ATRIB
    ;
5 Lexico.LITINT 1; Lexico.ELSE; Lexico.IF; Lexico.ID "n"; Lexico.MENOR;
```

```

6 Lexico.LITINT 0; Lexico.THEN; Lexico.ID "res"; Lexico.ATRIB;
7 Lexico.SUBTRACAO; Lexico.LITINT 1; Lexico.ELSE; Lexico.ID "res";
8 Lexico.ATRIB; Lexico.LITINT 0; Lexico.END; Lexico.END; Lexico.RETURN;
9 Lexico.ID "res"; Lexico.END; Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR
    ;
10 Lexico.FPAR; Lexico.LOCAL; Lexico.ID "numero"; Lexico.LOCAL; Lexico.ID "x
    ";
11 Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "Digite um n\195\186mero: ";
12 Lexico.FPAR; Lexico.ID "numero"; Lexico.ATRIB; Lexico.IO_READ; Lexico.
    APAR;
13 Lexico.LITSTRING "*n"; Lexico.FPAR; Lexico.ID "x"; Lexico.ATRIB;
14 Lexico.ID "verifica"; Lexico.APAR; Lexico.ID "numero"; Lexico.FPAR;
15 Lexico.IF; Lexico.ID "x"; Lexico.EQUIVALENTE; Lexico.LITINT 1; Lexico.
    THEN;
16 Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "N\195\186mero positivo";
17 Lexico.FPAR; Lexico.ELSE; Lexico.IF; Lexico.ID "x"; Lexico.EQUIVALENTE;
18 Lexico.LITINT 0; Lexico.THEN; Lexico.PRINT; Lexico.APAR;
19 Lexico.LITSTRING "Zero"; Lexico.FPAR; Lexico.ELSE; Lexico.PRINT;
20 Lexico.APAR; Lexico.LITSTRING "N\195\186mero negativo"; Lexico.FPAR;
21 Lexico.END; Lexico.END; Lexico.END; Lexico.ID "main"; Lexico.APAR;
22 Lexico.FPAR; Lexico.EOF]

```

4.5 Testes de Erros

Alguns erros léxicos que podem ocorrer são exibidos a seguir.

4.5.1 Comentário não fechado

Abrir comentário de múltiplas linhas e não fechá-lo.

Listagem 4.25: Comentário não fechado corretamente

```

1 function main()
2     local n
3     --[ [
4 end

```

Saída do analisador:

Listagem 4.26: Comentário não fechado corretamente

```

1 Exception: Failure "3-7: Comentario nao fechado".

```

4.5.2 Caracter Desconhecido

Caracter que não se engloba em nenhuma característica de token.

Listagem 4.27: Caracter desconhecido

```

1 function main()

```

4.5

```
2     local n
3     n = @1
4 end
5
6 main()
```

Saída do analisador:

Listagem 4.28: Caracter desconhecido

```
1 Exception: Failure "3-8: caracter desconhecido @".
```

4.5.3 String não fechada corretamente

String é aberta com aspas ("), porém não fechada corretamente.

Listagem 4.29: String não fechada corretamente

```
1 function main()
2     local n
3     n = 2
4     print("n)
5 end
6
7 main()
```

Saída do analisador:

Listagem 4.30: String não fechada corretamente

```
1 Exception: Failure "4-10: A string nao foi fechada".
```

Capítulo 5

Analizador Sintático

Nesta seção será mostrado a implementação do analisador sintático da linguagem MiniLua feita em linguagem Ocaml, baseado nos programas nanos e micros encontrados na seção 3.1 reescritos de forma a serem validados na análise sintática. Além disso, será executado o analisador sintático para cada um deles e também será realizada análises de erros de forma proposital.

5.1 Gramática e Código do Analisador Sintático

Abaixo é definida a gramática do analisador sintático utilizado neste trabalho:

Listagem 5.1: Código do Analisador Sintático

```
1
2 %{
3 open Ast
4
5 %}
6
7 %token <string> ID
8 %token <string> LITSTRING
9 %token <int> LITINT
10 %token <bool> BOOL
11 %token ADICAO
12 %token AND
13 %token AND_BINARIO
14 %token APAR
15 %token ATRIB
16 %token BREAK
17 %token CONCATENA
18 %token DIV_POR_2
19 %token DIVISAO
20 %token DIVISAO_INTEIRO
21 %token DO
22 %token DOIS_PONTOS
23 %token ELSE
24 %token ELSEIF
25 %token END
26 %token EQUIVALENTE
```


5.1

```
27 %token EXPONENCIACAO
28 %token FOR
29 %token FPAR
30 %token FUNCAO
31 %token IF
32 %token IN
33 %token IO_READ
34 %token LOCAL
35 %token MAIOR
36 %token MAIOR_OU_IGUAL
37 %token MENOR
38 %token MENOR_OU_IGUAL
39 %token MODULO
40 %token MULT_POR_2
41 %token MULTIPLICACAO
42 %token NAO_EQUIVALENTE
43 %token NIL
44 %token NOT
45 %token NUMBER_INPUT
46 %token OR
47 %token OR_BINARIO
48 %token OR_BINARIO_EXCLUSIVO
49 %token PONTO
50 %token PONTO_VIRGULA
51 %token PRINT
52 %token REPEAT
53 %token RETICENCIAS
54 %token RETURN
55 %token SUBTRACAO
56 %token TAMANHO
57 %token TIPO_BOOLEAN
58 %token TIPO_INT
59 %token TIPO_STRING
60 %token THEN
61 %token UNTIL
62 %token VIRGULA
63 %token WHILE
64 %token EOF
65
66 %left OR
67 %left AND
68 %left MAIOR MENOR MAIOR_OU_IGUAL MENOR_OU_IGUAL EQUIVALENTE
    NAO_EQUIVALENTE
69 %left OR_BINARIO
70 %left OR_BINARIO_EXCLUSIVO
71 %left AND_BINARIO
72 %left MULT_POR_2 DIV_POR_2
73 %left CONCATENA
74 %left ADICAO SUBTRACAO
75 %left MULTIPLICACAO DIVISAO DIVISAO_INTEIRO MODULO
76 %left NOT TAMANHO
77 %left EXPONENCIACAO
78
79
80 %start <Ast.programa> programa
81
82 %%
83
84 programa: f = funcoes+
```

```

85         EOF { Programa (f) }
86
87 funcoes:
88     | FUNCAO tipo=tipo_simples id=ID APAR args=argumentos* FPAR
89         ds = declaracao*
90         cs = comando*
91         (*ret=retorno*)
92         END { Funcao (tipo, id, args, ds, cs(*, ret*)) }
93     ;
94
95 argumentos:
96     | t=tipo_simples id=ID { Args (t,id) }
97     ;
98
99 declaracao:
100    | t=tipo v=variavel { DecVar (t,v) }
101    ;
102
103 tipo: t=tipo_simples { t }
104
105 tipo_simples: TIPO_INT { TipoInt }
106              | TIPO_STRING { TipoString }
107              | TIPO_BOOLEAN { TipoBool }
108
109
110 comando: c=comando_atribuicao { c }
111         | c=comando_if { c }
112         | c=comando_for { c }
113         | c=comando_while { c }
114         | c=comando_print { c }
115         | c=comando_scan { c }
116         | c=comando_funcao { c }
117         | c=comando_retorno { c }
118
119 comando_atribuicao:
120     | v=variavel ATRIB e=expressao { CmdAtrib (v,e) }
121     | v=variavel ATRIB id=ID APAR args=ID* FPAR { CmdAtribRetorno (v,id
122         ,args) }
123     ;
124
125 comando_if: IF teste=expressao THEN
126             entao=comando+
127             senao=option(ELSE cs=comando+ {cs})
128             END {
129                 CmdIf (teste, entao, senao)
130             }
131
132 comando_for:
133     | FOR v=variavel ATRIB l1=LITINT VIRGULA l2=LITINT VIRGULA l3=LITINT DO
134         cs=comando* END { CmdFor (v, l1, l2, l3, cs) }
135     ;
136
137 comando_while:
138     | WHILE teste=expressao DO cs=comando* END { CmdWhile (teste, cs) }
139     ;
140
141 comando_print:
142     | PRINT APAR teste=expressao FPAR { CmdPrint (teste) }
143     ;

```

```

142
143 comando_scan:
144     | v=variavel ATRIB IO_READ APAR FPAR { CmdScan (v) }
145     ;
146
147 comando_funcao:
148     | id=ID APAR args=ID* FPAR { CmdFunction (id,args) }
149     ;
150
151 comando_retorno:
152     | RETURN exp=expressao { CmdRetorno (exp) }
153     ;
154
155 expressao:
156     | v=variavel { ExpVar v }
157     | i=LITINT { ExpInt i }
158     | s=LITSTRING { ExpString s }
159     | b=BOOL { ExpBool b }
160     | e1=expressao op=oper e2=expressao { ExpOp (op, e1, e2) }
161     | APAR e=expressao FPAR { e }
162
163 %inline oper:
164     | OR { Or }
165     | AND { And }
166     | MAIOR { Maior }
167     | MENOR { Menor }
168     | MAIOR_OU_IGUAL { Maior_ou_Igual }
169     | MENOR_OU_IGUAL { Menor_ou_Igual }
170     | EQUIVALENTE { Equivalente }
171     | NAO_EQUIVALENTE { Nao_Equivalente }
172     | OR_BINARIO { Or_Binario }
173     | OR_BINARIO_EXCLUSIVO { Or_Binario_Exclusivo }
174     | AND_BINARIO { And_Binario }
175     | MULT_POR_2 { Mult_Por_2 }
176     | DIV_POR_2 { Div_Por_2 }
177     | CONCATENA { Concatena }
178     | ADICAO { Adicao }
179     | SUBTRACAO { Subtracao }
180     | MULTIPLICACAO { Multiplicacao }
181     | DIVISAO { Divisao }
182     | DIVISAO_INTEIRO { Divisao_Inteiro }
183     | MODULO { Modulo }
184     | NOT { Not }
185     | TAMANHO { Tamanho }
186     | EXPONENCIACAO { Exponenciacao }
187
188 variavel:
189     | x=ID { VarSimples x }

```

5.2 Árvore Sintática Abstrata

Abaixo é definida a árvore sintática abstrata utilizada no analisador sintático:

Listagem 5.2: Código da Árvore Sintática

```
1 type identificador = string
```

```

2
3 type programa = Programa of funcoes list
4
5 and funcoes = Funcao of tipo * identificador * argumentos list *
  declaracoes * comandos (** retorno*)
6
7 and argumentos = Args of tipo * identificador
8
9 and declaracoes = declaracao list
10
11 and declaracao = DecVar of tipo * variavel
12
13 and comandos = comando list
14
15 and tipo = TipoInt
16           | TipoString
17           | TipoBool
18
19 and comando = CmdAtrib of variavel * expressao
20               | CmdAtribRetorno of variavel * identificador * identificador
21               | CmdIf of expressao * comandos * (comandos option)
22               | CmdFor of variavel * int * int * int * comandos
23               | CmdWhile of expressao * comandos
24               | CmdPrint of expressao
25               | CmdScan of variavel
26               | CmdFunction of identificador * identificador list
27               | CmdRetorno of expressao
28
29 and variaveis = variavel list
30
31 and variavel = VarSimples of identificador
32
33 and expressao = ExpVar of variavel
34               | ExpInt of int
35               | ExpString of string
36               | ExpBool of bool
37               | ExpOp of oper * expressao * expressao
38
39 and oper = Or
40           | And
41           | Maior
42           | Menor
43           | Maior_ou_Igual
44           | Menor_ou_Igual
45           | Equivalente
46           | Nao_Equivalente
47           | Or_Binario
48           | Or_Binario_Exclusivo
49           | And_Binario
50           | Mult_Por_2
51           | Div_Por_2
52           | Concatena
53           | Adicao
54           | Subtracao
55           | Multiplicacao
56           | Divisao
57           | Divisao_Inteiro
58           | Modulo

```

```
59 | Not
60 | Tamanho
61 | Exponenciacao
```

5.3 Execução do Analisador Sintático

Os seguintes passos são necessários para executar o analisador sintático escrito na linguagem Ocaml:

1. Entrar no terminal e gerar as mensagens de erro caso não possua no projeto:

```
> menhir -v --list-errors sintatico.mly > sintatico.msg
```

2. Compilar o arquivo de mensagens de erro sintatico.mly de modo a ser usado pelo analisador sintático:

```
> menhir sintatico.mly --compile-errors sintatico.msg >
erroSint.ml
```

3. Compilar todo o projeto contendo o analisador sintatico:

```
> ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --
table" -package menhirLib main.byte
```

4. Entrar no Ocaml com o programa rlwrap, o qual armazena os comandos executados no terminal:

```
> rlwrap ocaml
```

5. Entrar com o código a ser analisado:

```
> parse_arq "nome_codigo";;
```

5.4 Testes do Analisador Sintático

Os testes realizados tem o objetivo de validar a corretude das árvores geradas pelo analisador sintático. Para tanto, foi necessário o uso dos programas reescritos nanos e micros apresentados na seção 3.1.

5.4.1 Nano Programas

Nano01

Listagem 5.3: Módulo mínimo que caracteriza um programa

```
1 function string main()
2 end
```

Saída do analisador sintático:

Listagem 5.4: Saída do analisador sintático para o programa nano01

```
1 - : Ast.programa option =
2 Some (Programa [Funcao (TipoString, "main", [], [], [])])
```

Nano02

Listagem 5.5: Declaração de uma variável

```
1 function main()
2     local n
3 end
```

Saída do analisador sintático:

Listagem 5.6: Saída do analisador sintático para o programa nano02

```
1 - : Ast.programa option =
2 Some
3   (Programa
4     [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "x")],
5       [CmdRetorno (ExpInt 1)])])
```

Nano03

Listagem 5.7: Atribuição de um inteiro à uma variável

```
1 function int main()
2     int n
3     n=1
4     return 1
5 end
```

Saída do analisador sintático:

Listagem 5.8: Saída do analisador sintático para o programa nano03

```
1 - : Ast.programa option =
2 Some
3   (Programa
4     [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "n")],
5       [CmdAtrib (VarSimples "n", ExpInt 1); CmdRetorno (ExpInt 1)])])
```

Nano04

Listagem 5.9: Atribuição de uma soma de inteiros à uma variável

```
1 function int main()
2
```

5.4

```
3      int n
4      n = 1 + 2
5
6      return 1
7 end
```

Saída do analisador sintático:

Listagem 5.10: Saída do analisador sintático para o programa nano04

```
1 - : Ast.programa option =
2 Some
3 (Programa
4  [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "n")],
5    [CmdAtrib (VarSimples "n", ExpOp (Adicao, ExpInt 1, ExpInt 2));
6    CmdRetorno (ExpInt 1)])])
```

Nano05

Listagem 5.11: Inclusão do comando de impressão

```
1 function int main()
2
3     int n
4     n = 2
5     print(n)
6
7     return 1
8 end
```

Saída do analisador sintático:

Listagem 5.12: Saída do analisador sintático para o programa nano05

```
1 - : Ast.programa option =
2 Some
3 (Programa
4  [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "n")],
5    [CmdAtrib (VarSimples "n", ExpInt 2);
6    CmdPrint (ExpVar (VarSimples "n")); CmdRetorno (ExpInt 1)])])
```

Nano06

Listagem 5.13: Atribuição de uma subtração de inteiros à uma variável

```
1 function int main()
2     int n
3     n = 1 - 2
4     print(n)
5
6     return 1
7 end
```

Saída do analisador sintático:

Listagem 5.14: Saída do analisador sintático para o programa nano06

```

1 - : Ast.programa option =
2 Some
3   (Programa
4     [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "n")],
5       [CmdAtrib (VarSimples "n", ExpOp (Subtracao, ExpInt 1, ExpInt 2));
6         CmdPrint (ExpVar (VarSimples "n")); CmdRetorno (ExpInt 1)])])

```

Nano07

Listagem 5.15: Inclusão do comando condicional

```

1 function int main()
2   int n
3   n = 1
4   if n == 1 then
5     print(n)
6   end
7   return 1
8 end

```

Saída do analisador sintático:

Listagem 5.16: Saída do analisador sintático para o programa nano07

```

1 - : Ast.programa option =
2 Some
3   (Programa
4     [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "n")],
5       [CmdAtrib (VarSimples "n", ExpInt 1);
6         CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "n"), ExpInt 1),
7           [CmdPrint (ExpVar (VarSimples "n"))], None);
8         CmdRetorno (ExpInt 1)])])

```

Nano08

Listagem 5.17: Inclusão do comando condicional com parte senão

```

1 function int main()
2   int n
3   n = 1
4   if n == 1 then
5     print(n)
6   else
7     print(0)
8   end
9
10  return 1
11 end

```

Saída do analisador sintático:

Listagem 5.18: Saída do analisador sintático para o programa nano08

```

1 - : Ast.programa option =
2 Some
3   (Programa
4     [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "n")],

```



```

5      [CmdAtrib (VarSimples "n", ExpInt 1);
6      CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "n"), ExpInt 1),
7      [CmdPrint (ExpVar (VarSimples "n"))], Some [CmdPrint (ExpInt 0)]);
8      CmdRetorno (ExpInt 1)]]])

```

Nano09

Listagem 5.19: Atribuição de duas operações aritméticas sobre inteiros a uma variável

```

1 function int main()
2     int n
3     n = (1 + 1) / 2
4     if n == 1 then
5         print(n)
6     else
7         print(0)
8     end
9 end

```

Saída do analisador sintático:

Listagem 5.20: Saída do analisador sintático para o programa nano09

```

1 - : Ast.programa option =
2 Some
3 (Programa
4  [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "n")],
5  [CmdAtrib (VarSimples "n",
6  ExpOp (Divisao, ExpOp (Adicao, ExpInt 1, ExpInt 1), ExpInt 2));
7  CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "n"), ExpInt 1),
8  [CmdPrint (ExpVar (VarSimples "n"))], Some [CmdPrint (ExpInt 0)])])
9  ])

```

Nano10

Listagem 5.21: Atribuição de duas variáveis inteiras

```

1 function int main()
2     int n
3     int m
4     n = 1
5     m = 2
6     if n == m then
7         print(n)
8     else
9         print(0)
10    end
11
12    return 1
13 end

```

Saída do analisador sintático:

Listagem 5.22: Saída do analisador sintático para o programa nano10

```

1 - : Ast.programa option =
2 Some

```

```

3 (Programa
4   [Funcao (TipoInt, "main", [],
5     [DecVar (TipoInt, VarSimples "n"); DecVar (TipoInt, VarSimples "m")],
6     [CmdAtrib (VarSimples "n", ExpInt 1);
7       CmdAtrib (VarSimples "m", ExpInt 2);
8       CmdIf
9         (ExpOp (Equivalente, ExpVar (VarSimples "n"), ExpVar (VarSimples "m"
10           ")),
11       [CmdPrint (ExpVar (VarSimples "n"))], Some [CmdPrint (ExpInt 0)]];
12   CmdRetorno (ExpInt 1)]]])

```

Nano11

Listagem 5.23: Introdução do comando de repetição enquanto

```

1 function int main()
2
3   int n
4   int m
5   int x
6
7   n = 1
8   m = 2
9   x = 5
10
11  while x > n do
12    n = n + m
13    print(n)
14  end
15
16 end

```

Saída do analisador sintático:

Listagem 5.24: Saída do analisador sintático para o programa nano11

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "main", [],
5     [DecVar (TipoInt, VarSimples "n"); DecVar (TipoInt, VarSimples "m");
6     DecVar (TipoInt, VarSimples "x")],
7     [CmdAtrib (VarSimples "n", ExpInt 1);
8     CmdAtrib (VarSimples "m", ExpInt 2);
9     CmdAtrib (VarSimples "x", ExpInt 5);
10    CmdWhile
11      (ExpOp (Maior, ExpVar (VarSimples "x"), ExpVar (VarSimples "n")),
12      [CmdAtrib (VarSimples "n",
13        ExpOp (Adicao, ExpVar (VarSimples "n"), ExpVar (VarSimples "m")))
14      ];
15    CmdPrint (ExpVar (VarSimples "n"))]]]]])

```

Nano12

Listagem 5.25: Comando condicional aninhado em um comando de repetição

```

1 function int main()

```

```

2      int n
3      int m
4      int x
5
6      n = 1
7      m = 2
8      x = 5
9
10     while x > n do
11         if n == m then
12             print(n)
13         else
14             print(0)
15         end
16         x = x - 1
17     end
18
19 end

```

Saída do analisador sintático:

Listagem 5.26: Saída do analisador sintático para o programa nano12

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "main", [],
5     [DecVar (TipoInt, VarSimples "n"); DecVar (TipoInt, VarSimples "m");
6     DecVar (TipoInt, VarSimples "x")],
7     [CmdAtrib (VarSimples "n", ExpInt 1);
8     CmdAtrib (VarSimples "m", ExpInt 2);
9     CmdAtrib (VarSimples "x", ExpInt 5);
10    CmdWhile
11      (ExpOp (Maior, ExpVar (VarSimples "x"), ExpVar (VarSimples "n")),
12      [CmdIf
13        (ExpOp (Equivalente, ExpVar (VarSimples "n"),
14          ExpVar (VarSimples "m")),
15        [CmdPrint (ExpVar (VarSimples "n"))], Some [CmdPrint (ExpInt 0)])
16        ;
17      CmdAtrib (VarSimples "x",
18      ExpOp (Subtracao, ExpVar (VarSimples "x"), ExpInt 1))]]))]

```

5.4.2 Micro Programas

Micro01

Listagem 5.27: Converte graus Celsius para Fahrenheit

```

1 function int main()
2     int cel
3     int far
4     print("Tabela de conversão: Celsius -> Fahrenheit")
5     print("Digite a temperatura em Celsius: ")
6     cel = io.read()
7     far = (9*cel+160)/5
8     print("A nova temperatura eh: ")
9     print(far)

```

```

10     print(" F")
11 end

```

Saída do analisador sintático:

Listagem 5.28: Saída do analisador sintático para o programa micro01

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "main", [],
5     [DecVar (TipoInt, VarSimples "cel"); DecVar (TipoInt, VarSimples "far
6       ")],
7     [CmdPrint
8       (ExpString "Tabela de convers\195\163o: Celsius -> Fahrenheit");
9       CmdPrint (ExpString "Digite a temperatura em Celsius: ");
10      CmdScan (VarSimples "cel");
11      CmdAtrib (VarSimples "far",
12        ExpOp (Divisao,
13          ExpOp (Adicao,
14            ExpOp (Multiplicacao, ExpInt 9, ExpVar (VarSimples "cel")),
15            ExpInt 160),
16          ExpInt 5));
17      CmdPrint (ExpString "A nova temperatura eh: ");
18      CmdPrint (ExpVar (VarSimples "far")); CmdPrint (ExpString " F")]]])

```

Micro02

Listagem 5.29: Ler dois inteiros e decide qual é maior

```

1 function int main()
2   int num1
3   int num2
4   print("Digite o primeiro numero: ")
5   num1 = io.read()
6   print("Digite o segundo numero: ")
7   num2 = io.read()
8
9   if num1 > num2 then
10     print("O primeiro numero ")
11     print(num1)
12     print(" eh maior que o segundo ")
13     print(num2)
14   else
15     print("O segundo numero ")
16     print(num2)
17     print(" eh maior que o primeiro ")
18     print(num1)
19   end
20 end

```

Saída do analisador sintático:

Listagem 5.30: Saída do analisador sintático para o programa micro02

```

1 - : Ast.programa option =
2 Some
3 (Programa

```

```

4  [Funcao (TipoInt, "main", [],
5  [DecVar (TipoInt, VarSimples "num1");
6  DecVar (TipoInt, VarSimples "num2")],
7  [CmdPrint (ExpString "Digite o primeiro numero: ");
8  CmdScan (VarSimples "num1");
9  CmdPrint (ExpString "Digite o segundo numero: ");
10 CmdScan (VarSimples "num2");
11 CmdIf
12 (ExpOp (Maior, ExpVar (VarSimples "num1"), ExpVar (VarSimples "num2"
13 "))),
14 [CmdPrint (ExpString "O primeiro numero ");
15 CmdPrint (ExpVar (VarSimples "num1"));
16 CmdPrint (ExpString " eh maior que o segundo ");
17 CmdPrint (ExpVar (VarSimples "num2"))],
18 Some
19 [CmdPrint (ExpString "O segundo numero ");
20 CmdPrint (ExpVar (VarSimples "num2"));
21 CmdPrint (ExpString " eh maior que o primeiro ");
22 CmdPrint (ExpVar (VarSimples "num1"))]]]]))

```

Micro03

Listagem 5.31: Lê um número e verifica se ele está entre 100 e 200

```

1  function int main()
2      int numero
3      print("Digite um numero: ")
4      numero = io.read()
5      if numero >= 100 then
6          if numero <= 200 then
7              print("O numero esta no intervalo entre 100 e 200")
8          else
9              print("O numero nao esta no intervalo entre 100 e 200")
10         end
11     else
12         print("O numero nao esta no intervalo entre 100 e 200")
13     end
14 end

```

Saída do analisador sintático:

Listagem 5.32: Saída do analisador sintático para o programa micro03

```

1  - : Ast.programa option =
2  Some
3  (Programa
4  [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "numero")],
5  [CmdPrint (ExpString "Digite um numero: ");
6  CmdScan (VarSimples "numero");
7  CmdIf
8  (ExpOp (Maior_ou_Igual, ExpVar (VarSimples "numero"), ExpInt 100),
9  [CmdIf
10 (ExpOp (Menor_ou_Igual, ExpVar (VarSimples "numero"), ExpInt 200)
11 ,
12 [CmdPrint (ExpString "O numero esta no intervalo entre 100 e 200"
13 )]],
14 Some
15 [CmdPrint

```

```

14         (ExpString "O numero nao esta no intervalo entre 100 e 200"))])
15         ],
16     Some
17     [CmdPrint
18       (ExpString "O numero nao esta no intervalo entre 100 e 200"))]]])
19   ])

```

Micro04

Listagem 5.33: Lê números e informa quais estão entre 10 e 150

```

1  function int main()
2
3      int x
4      int num
5      int intervalo
6
7      intervalo = 0
8
9      for x=1, 5, 1 do
10         print("Digite um numero: ")
11         num = io.read()
12         if num >= 10 then
13             if num <= 150 then
14                 intervalo = intervalo + 1
15             end
16         end
17     end
18
19     print("Ao total, foram digitados ")
20     print(intervalo)
21     print(" numeros no intervalo entre 10 e 150")
22
23 end

```

Saída do analisador sintático:

Listagem 5.34: Saída do analisador sintático para o programa micro04

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "main", [],
5     [DecVar (TipoInt, VarSimples "x"); DecVar (TipoInt, VarSimples "num")
6       ;
7     DecVar (TipoInt, VarSimples "intervalo")],
8   [CmdAtrib (VarSimples "intervalo", ExpInt 0);
9   CmdFor (VarSimples "x", 1, 5, 1,
10    [CmdPrint (ExpString "Digite um numero: ");
11    CmdScan (VarSimples "num");
12    CmdIf (ExpOp (Maior_ou_Igual, ExpVar (VarSimples "num"), ExpInt
13      10),
14    [CmdIf
15      (ExpOp (Menor_ou_Igual, ExpVar (VarSimples "num"), ExpInt 150),
16      [CmdAtrib (VarSimples "intervalo",
17        ExpOp (Adicao, ExpVar (VarSimples "intervalo"), ExpInt 1))],
18      None)],
19    None)])];
20   None)])];

```

```

18 CmdPrint (ExpString "Ao total, foram digitados ");
19 CmdPrint (ExpVar (VarSimples "intervalo"));
20 CmdPrint (ExpString " numeros no intervalo entre 10 e 150"))]]))

```

Micro05

Listagem 5.35: Lê strings e caracteres

```

1  function int main()
2      string nome
3      string sexo
4      int x
5      int h
6      int m
7
8      x = 1
9      h = 0
10     m = 0
11
12     for x=1, 5, 1 do
13         print("Digite o nome: ")
14         nome = io.read()
15         print("H - Homem ou M - Mulher: ")
16         sexo = io.read()
17         if sexo == "H" then
18             h = h + 1
19         else
20             if sexo == "M" then
21                 m = m + 1
22             else
23                 print("Sexo so pode ser H ou M!")
24         end
25     end
26 end
27 print("Foram inseridos ")
28 print(h)
29 print(" Homens")
30 print("Foram inseridos ")
31 print(m)
32 print(" Mulheres")
33 end

```

Saída do analisador sintático:

Listagem 5.36: Saída do analisador sintático para o programa micro05

```

1 - : Ast.programa option =
2 Some
3 (Programa
4  [Funcao (TipoInt, "main", [],
5    [DecVar (TipoString, VarSimples "nome");
6    DecVar (TipoString, VarSimples "sexo");
7    DecVar (TipoInt, VarSimples "x"); DecVar (TipoInt, VarSimples "h");
8    DecVar (TipoInt, VarSimples "m")],
9    [CmdAtrib (VarSimples "x", ExpInt 1);
10   CmdAtrib (VarSimples "h", ExpInt 0);
11   CmdAtrib (VarSimples "m", ExpInt 0);
12   CmdFor (VarSimples "x", 1, 5, 1,

```

```

13      [CmdPrint (ExpString "Digite o nome: "); CmdScan (VarSimples "nome"
14              );
15      CmdPrint (ExpString "H - Homem ou M - Mulher: ");
16      CmdScan (VarSimples "sexo");
17      CmdIf
18      (ExpOp (Equivalente, ExpVar (VarSimples "sexo"), ExpString "H"),
19      [CmdAtrib (VarSimples "h",
20      ExpOp (Adicao, ExpVar (VarSimples "h"), ExpInt 1))],
21      Some
22      [CmdIf
23      (ExpOp (Equivalente, ExpVar (VarSimples "sexo"), ExpString "M"
24              ),
25      [CmdAtrib (VarSimples "m",
26      ExpOp (Adicao, ExpVar (VarSimples "m"), ExpInt 1))],
27      Some [CmdPrint (ExpString "Sexo so pode ser H ou M!")]]]]]);
28      CmdPrint (ExpString "Foram inseridos ");
29      CmdPrint (ExpVar (VarSimples "h")); CmdPrint (ExpString " Homens");
30      CmdPrint (ExpString "Foram inseridos ");
31      CmdPrint (ExpVar (VarSimples "m")); CmdPrint (ExpString " Mulheres")
32      ]))

```

Micro06

Listagem 5.37: Escreve um número lido por extenso

```

1  function int main()
2      int num
3      print("Digite um numero de 1 a 5: ")
4      num = io.read()
5      if num == 1 then
6          print("Um")
7      else
8          if num == 2 then
9              print("Dois")
10         else
11             if num == 3 then
12                 print("Tres")
13             else
14                 if num == 4 then
15                     print("Quatro")
16                 else
17                     if num == 5 then
18                         print("Cinco")
19                     else
20                         print("Numero Invalido!!!")
21                     end
22                 end
23             end
24         end
25     end
26 end

```

Saída do analisador sintático:

Listagem 5.38: Saída do analisador sintático para o programa micro06

```

1 - : Ast.programa option =
2 Some

```


Saída do analisador sintático:

Listagem 5.40: Saída do analisador sintático para o programa micro07

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "main", [],
5     [DecVar (TipoInt, VarSimples "programa");
6     DecVar (TipoInt, VarSimples "numero");
7     DecVar (TipoInt, VarSimples "opc")],
8   [CmdAtrib (VarSimples "programa", ExpInt 1);
9   CmdWhile
10    (ExpOp (Equivalente, ExpVar (VarSimples "programa"), ExpInt 1),
11   [CmdPrint (ExpString "Digite um numero: ");
12   CmdScan (VarSimples "numero");
13   CmdIf (ExpOp (Maior, ExpVar (VarSimples "numero"), ExpInt 0),
14    [CmdPrint (ExpString "Positivo")],
15    Some
16    [CmdIf
17      (ExpOp (Equivalente, ExpVar (VarSimples "numero"), ExpInt 0),
18      [CmdPrint (ExpString "O numero eh igual a 0")], None);
19      CmdIf (ExpOp (Menor, ExpVar (VarSimples "numero"), ExpInt 0),
20      [CmdPrint (ExpString "Negativo")], None)]];
21   CmdPrint (ExpString "Deseja finalizar? (S-1/N-2) ");
22   CmdScan (VarSimples "opc");
23   CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "opc"), ExpInt 1),
24    [CmdAtrib (VarSimples "programa", ExpInt 0)], None)]]))])

```

Micro08

Listagem 5.41: Decide se um número é maior ou menor que 10

```

1 function int main()
2   int numero
3   numero = 1
4   while numero ~= 0 do
5     print("Digite um numero: ")
6     numero = io.read()
7     if numero > 10 then
8       print("O numero ")
9       print(numero)
10      print(" eh maior que 10")
11    else
12      print("O numero ")
13      print(numero)
14      print(" eh menor que 10")
15    end
16  end
17 end

```

Saída do analisador sintático:

Listagem 5.42: Saída do analisador sintático para o programa micro08

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "numero")],

```

```

5      [CmdAtrib (VarSimples "numero", ExpInt 1);
6      CmdWhile
7      (ExpOp (Nao_Equivalente, ExpVar (VarSimples "numero"), ExpInt 0),
8      [CmdPrint (ExpString "Digite um numero: ");
9      CmdScan (VarSimples "numero");
10     CmdIf (ExpOp (Maior, ExpVar (VarSimples "numero"), ExpInt 10),
11     [CmdPrint (ExpString "O numero ");
12     CmdPrint (ExpVar (VarSimples "numero"));
13     CmdPrint (ExpString " eh maior que 10")],
14     Some
15     [CmdPrint (ExpString "O numero ");
16     CmdPrint (ExpVar (VarSimples "numero"));
17     CmdPrint (ExpString " eh menor que 10")]]))]

```

Micro09

Listagem 5.43: Cálculo de preços

```

1  function int main()
2      int preco
3      int venda
4      int novo_preco
5      print("Digite o preco: ")
6      preco = io.read()
7      print("Digite a venda: ")
8      venda = io.read()
9      if venda < 500 or preco < 30 then
10         novo_preco = preco + 10/100 * preco
11     else if (venda >= 500 and venda < 1200) or (preco >= 30 and preco <
12         80) then
13         novo_preco = preco + 15/100 * preco
14         else if venda >= 1200 or preco >= 80 then
15             novo_preco = preco - 20/100 * preco
16         end
17     end
18     print("O novo preco eh ")
19     print(novo_preco)
20 end

```

Saída do analisador sintático:

Listagem 5.44: Saída do analisador sintático para o programa micro09

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "main", [],
5     [DecVar (TipoInt, VarSimples "preco");
6     DecVar (TipoInt, VarSimples "venda");
7     DecVar (TipoInt, VarSimples "novo_preco")],
8     [CmdPrint (ExpString "Digite o preco: "); CmdScan (VarSimples "preco"
9       );
10    CmdPrint (ExpString "Digite a venda: "); CmdScan (VarSimples "venda"
11      );
12    CmdIf
13      (ExpOp (Or, ExpOp (Menor, ExpVar (VarSimples "venda"), ExpInt 500),
14        ExpOp (Menor, ExpVar (VarSimples "preco"), ExpInt 30)),

```

```

13     [CmdAtrib (VarSimples "novo_preco",
14               ExpOp (Adicao, ExpVar (VarSimples "preco"),
15               ExpOp (Multiplicacao, ExpOp (Divisao, ExpInt 10, ExpInt 100),
16               ExpVar (VarSimples "preco")))]],
17   Some
18     [CmdIf
19       (ExpOp (Or,
20         ExpOp (And,
21           ExpOp (Maior_ou_Igual, ExpVar (VarSimples "venda"), ExpInt
22             500),
23           ExpOp (Menor, ExpVar (VarSimples "venda"), ExpInt 1200)),
24         ExpOp (And,
25           ExpOp (Maior_ou_Igual, ExpVar (VarSimples "preco"), ExpInt
26             30),
27           ExpOp (Menor, ExpVar (VarSimples "preco"), ExpInt 80))),
28       [CmdAtrib (VarSimples "novo_preco",
29               ExpOp (Adicao, ExpVar (VarSimples "preco"),
30               ExpOp (Multiplicacao, ExpOp (Divisao, ExpInt 15, ExpInt 100),
31               ExpVar (VarSimples "preco")))]],
32       Some
33         [CmdIf
34           (ExpOp (Or,
35             ExpOp (Maior_ou_Igual, ExpVar (VarSimples "venda"),
36             ExpInt 1200),
37             ExpOp (Maior_ou_Igual, ExpVar (VarSimples "preco"), ExpInt
38               80)),
39           [CmdAtrib (VarSimples "novo_preco",
40                   ExpOp (Subtracao, ExpVar (VarSimples "preco"),
41                   ExpOp (Multiplicacao, ExpOp (Divisao, ExpInt 20, ExpInt
42                     100),
43                   ExpVar (VarSimples "preco")))]],
44           None)]])];
45   CmdPrint (ExpString "O novo preco eh ");
46   CmdPrint (ExpVar (VarSimples "novo_preco")))]])

```

Micro10

Listagem 5.45: Calcula o fatorial de um número

```

1 function int fatorial(int n)
2   int x
3
4   if n <= 1 then
5     return 1
6   else
7     y = n - 1
8     x = fatorial(y)
9   return n * x
10  end
11 end
12
13 function int main()
14   int numero
15   int fat
16   print("Digite um numero: ")
17   numero = io.read()
18   fat = fatorial(numero)
19   print("O fatorial de ")

```

```

20     print(numero)
21     print(" eh ")
22     print(fat)
23
24     return 1
25
26 end

```

Saída do analisador sintático:

Listagem 5.46: Saída do analisador sintático para o programa micro10

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "fatorial", [Args (TipoInt, "n")],
5     [DecVar (TipoInt, VarSimples "x")],
6     [CmdIf (ExpOp (Menor_ou_Igual, ExpVar (VarSimples "n"), ExpInt 1),
7       [CmdRetorno (ExpInt 1)],
8       Some
9         [CmdAtrib (VarSimples "y",
10           ExpOp (Subtracao, ExpVar (VarSimples "n"), ExpInt 1));
11           CmdAtribRetorno (VarSimples "x", "fatorial", ["y"]);
12           CmdRetorno
13             (ExpOp (Multiplicacao, ExpVar (VarSimples "n"),
14               ExpVar (VarSimples "x")))]))]];
15   Funcao (TipoInt, "main", [],
16     [DecVar (TipoInt, VarSimples "numero");
17     DecVar (TipoInt, VarSimples "fat")],
18     [CmdPrint (ExpString "Digite um numero: ");
19     CmdScan (VarSimples "numero");
20     CmdAtribRetorno (VarSimples "fat", "fatorial", ["numero"]);
21     CmdPrint (ExpString "O fatorial de ");
22     CmdPrint (ExpVar (VarSimples "numero")); CmdPrint (ExpString " eh ")
23     ;
24     CmdPrint (ExpVar (VarSimples "fat")); CmdRetorno (ExpInt 1)]])

```

Micro11

Listagem 5.47: Decide se um número é positivo, zero ou negativo com auxílio de uma função

```

1 function int verifica(int n)
2   int res
3   if n > 0 then
4     res = 1
5   else if n < 0 then
6     res = -1
7   else
8     res = 0
9   end
10  end
11  return res
12 end
13
14 function int main()
15   int numero
16   int x
17   print("Digite um numero: ")

```

```

18     numero = io.read()
19     x = verifica(numero)
20     if x == 1 then
21         print("Numero positivo")
22     else if x == 0 then
23         print("Zero")
24     else
25         print("Numero negativo")
26     end
27 end
28 end

```

Saída do analisador sintático:

Listagem 5.48: Saída do analisador sintático para o programa micro11

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "verifica", [Args (TipoInt, "n")],
5     [DecVar (TipoInt, VarSimples "res")],
6     [CmdIf (ExpOp (Maior, ExpVar (VarSimples "n"), ExpInt 0),
7       [CmdAtrib (VarSimples "res", ExpInt 1)],
8       Some
9         [CmdIf (ExpOp (Menor, ExpVar (VarSimples "n"), ExpInt 0),
10           [CmdAtrib (VarSimples "res", ExpInt (-1))],
11           Some [CmdAtrib (VarSimples "res", ExpInt 0)])]),
12       CmdRetorno (ExpVar (VarSimples "res"))]),
13   Funcao (TipoInt, "main", [],
14     [DecVar (TipoInt, VarSimples "numero");
15     DecVar (TipoInt, VarSimples "x")],
16     [CmdPrint (ExpString "Digite um numero: ");
17     CmdScan (VarSimples "numero");
18     CmdAtribRetorno (VarSimples "x", "verifica", ["numero"]);
19     CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "x"), ExpInt 1),
20       [CmdPrint (ExpString "Numero positivo")],
21       Some
22         [CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "x"), ExpInt 0),
23           [CmdPrint (ExpString "Zero")],
24           Some [CmdPrint (ExpString "Numero negativo")])])])])])])])

```

5.5 Testes de Erros Sintáticos

Alguns erros sintáticos que podem ocorrer são exibidos a seguir.

5.5.1 Comandos fora do escopo de uma função

Listagem 5.49: Comando fora do escopo de uma funcao

```

1 while

```

Saída do analisador:

Listagem 5.50: Comando fora do escopo de uma funcao

```

1 Erro sintático na linha 1, coluna 4 0 - "Funcao nao definida"
2 .
3
4 Exception: Failure "A analise sintatica falhou".

```

5.5.2 Função não estruturada corretamente

Listagem 5.51: Função não estruturada corretamente

```

1 function while

```

Saída do analisador:

Listagem 5.52: Função não estruturada corretamente

```

1 Erro sintático na linha 1, coluna 13 9 - "Function espera o seguinte
   formato: 'function tipo nome(argumentos)'"
2 .
3
4 Exception: Failure "A analise sintatica falhou".

```

5.5.3 Declaração incorreta de variável

Listagem 5.53: String não fechada corretamente

```

1 function int main()
2     int
3 end

```

Saída do analisador:

Listagem 5.54: String não fechada corretamente

```

1 Erro sintático na linha 3, coluna 2 28 - "Comando declaracao espera o
   formato: 'tipo_string tipo'"
2 .
3
4 Exception: Failure "A analise sintatica falhou".

```

5.5.4 Atribuição de variável incorreta

Listagem 5.55: Atribuição de variável incorreta

```

1 function int main()
2     int x
3
4     x =
5
6 end

```

Saída do analisador:

Listagem 5.56: Atribuição de variável incorreta

```

1 Erro sintático na linha 6, coluna 2 41 - "Atribuicao espera formato: '
    variavel ATRIB'"
2 .
3
4 Exception: Failure "A analise sintatica falhou".

```

5.5.5 Comando IF em formato incorreto

Comando "if" com comandos e token "end" ausentes.

Listagem 5.57: Comando IF incorreto

```

1 function int main()
2
3     int x
4     x = 9
5
6     if x==9 then
7
8 end

```

Saída do analisador:

Listagem 5.58: Comando IF incorreto

```

1 Erro sintático na linha 8, coluna 2 60 - "Comando if espera o seguinte
    formato: 'if expressao then comandos else comandos end'"
2 .
3
4 Exception: Failure "A analise sintatica falhou".

```

5.5.6 Comando FOR em formato incorreto

Comando "for" com o token "do" ausente.

Listagem 5.59: Comando FOR incorreto

```

1 function int main()
2
3     int x
4     int y
5
6     x = 9
7
8     for y=0,5,1
9         x = x+1
10    end
11
12 end

```

Saída do analisador:

Listagem 5.60: Comando FOR incorreto


```

1 Erro sintático na linha 9, coluna 5 70 - "For espera o seguinte formato: '
  for variavel atrib litint virgula litint virgula litint do comandos end
  '"
2 .
3
4 Exception: Failure "A analise sintatica falhou".

```

5.5.7 Comando WHILE em formato incorreto

Comando "while" com expressão incorreta.

Listagem 5.61: Comando WHILE incorreto

```

1 function int main()
2
3   int x
4
5   x = 9
6
7   while if do
8     x = x+1
9   end
10
11 end

```

Saída do analisador:

Listagem 5.62: Comando WHILE incorreto

```

1 Erro sintático na linha 7, coluna 10 53 - "While espera o seguinte formato
  : 'while expressao do comandos end'"
2 .
3
4 Exception: Failure "A analise sintatica falhou".

```

5.5.8 Retorno de função em formato incorreto

Retorno de função com expressão ausente.

Listagem 5.63: Retorno de função incorreto

```

1 function int main()
2
3   int x
4   x = 9
5
6   return
7
8 end

```

Saída do analisador:

Listagem 5.64: Retorno de função incorreto

```
1 Erro sintático na linha 8, coluna 2 52 - "Comando return espera o formato:  
  'return expressao'"  
2 .  
3  
4 Exception: Failure "A analise sintatica falhou".
```

Capítulo 6

Analizador Semântico e Interpretador

Esta seção tem a finalidade de exibir a execução de testes do analisador semântico e do interpretador da linguagem Minilua feita em linguagem Ocaml. É importante destacar que o analisador sintático construído previamente apenas suportava programas escritos como lista de funções, ou seja, não era possível escrever códigos com variáveis globais e comandos globais, porém em Lua, ambos são permitidos. Dessa forma, o analisador sintático foi refeito para adaptar à seguinte estrutura: variáveis globais, funções e comandos globais. Assim, os programas nanos e micros em Lua foram reescritos de modo a serem executados perfeitamente pelo Analisador Semântico e pelo Interpretador aqui expostos, os quais serão exibidos na seção 6.2. Os códigos finais de todas as etapas estão contidas no apêndice A.

6.1 Execução do Analisador Semântico e Interpretador

Os seguintes passos são necessários para executar o analisador sintático escrito na linguagem Ocaml:

1. Compilar todo o projeto contendo o interpretador:

```
> ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir -  
-table" -package menhirLib interpreteTeste.byte
```

2. Entrar no Ocaml com o programa rlwrap, o qual armazena os comandos executados no terminal:

```
> rlwrap ocaml
```

3. Caso deseje verificar a árvore semântica de um programa, é necessário entrar com o nome de tal programa a ser analisado a partir do comando "verifica_tipos":

```
> verifica_tipos "nome_codigo";;
```

4. Caso deseje executar um programa pelo interpretador, é necessário entrar com o nome de tal programa a partir do comando "interprete":

```
> interprete "nome_codigo";;
```

6.2 Testes do Analisador Semântico e Interpretador

Os testes realizados tem o objetivo de validar a corretude tanto das árvores semânticas geradas pelo analisador semântico quanto das execuções dos programas realizadas pelo interpretador. Para tanto, foi necessário o uso dos programas reescritos nanos e micros em Lua, apresentados abaixo.

6.2.1 Nano Programas

Nano01

Listagem 6.1: Módulo mínimo que caracteriza um programa

Saída do analisador semântico:

Listagem 6.2: Saída do analisador semântico para o programa nano01

```
1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([], [], []), <abstr>)
```

Saída do interpretador:

Listagem 6.3: Saída do interpretador para o programa nano01

```
1 - : unit = ()
```

Nano02

Listagem 6.4: Declaração de uma variável

```
1 function int main()
2   float n
3 end
4
5 main()
```

Saída do analisador semântico:

Listagem 6.5: Saída do analisador semântico para o programa nano02

```
1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([],
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7       fn_tiporet = TipoInt; fn_formais = [];
8       fn_locais =
9         [DecVar
10          (("n",
11            {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
              29})],
```

```

12     TipoFloat)]];
13     fn_corpo = []]],
14     [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]],
15     <abstr>)

```

Saída do interpretador:

Listagem 6.6: Saída do interpretador para o programa nano02

```

1 - : unit = ()

```

Nano03

Listagem 6.7: Atribuição de um inteiro à uma variável

```

1 function int main()
2     int n
3     n=1
4     return 1
5 end
6
7 main()

```

Saída do analisador semântico:

Listagem 6.8: Saída do analisador semântico para o programa nano03

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([],
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7       fn_tiporet = TipoInt; fn_formais = [];
8       fn_locais =
9         [DecVar
10          (("n",
11            {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12              28}),
13            TipoInt)];
14       fn_corpo =
15         [CmdAtrib
16           (Tast.ExpVar
17             (VarSimples
18               ("n",
19                 {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 30;
20                   pos_cnum = 34}),
21               TipoInt),
22             Tast.ExpInt (1, TipoInt));
23           CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
24   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
25   <abstr>)

```

Saída do interpretador:

Listagem 6.9: Saída do interpretador para o programa nano03

```

1 - : unit = ()

```

Nano04

Listagem 6.10: Atribuição de uma soma de inteiros à uma variável

```

1 function int main()
2
3     int n
4     n = 1 + 2
5
6     return 1
7 end
8
9 main()
```

Saída do analisador semântico:

Listagem 6.11: Saída do analisador semântico para o programa nano04

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7       fn_tiporet = TipoInt; fn_formais = [];
8       fn_locais =
9         [DecVar
10          (("n",
11            {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 25; pos_cnum =
12              33}),
13            TipoInt)];
14       fn_corpo =
15         [CmdAtrib
16           (Tast.ExpVar
17             (VarSimples
18               ("n",
19                 {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 35;
20                   pos_cnum = 39}),
21               TipoInt),
22             Tast.ExpOp ((Adicao, TipoInt), (Tast.ExpInt (1, TipoInt), TipoInt)
23               ,
24               (Tast.ExpInt (2, TipoInt), TipoInt)));
25           CmdRetorno (Some (Tast.ExpInt (1, TipoInt))))],
26   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
27   <abstr>)
```

Saída do interpretador:

Listagem 6.12: Saída do interpretador para o programa nano04

```

1 - : unit = ()
```

Nano05

Listagem 6.13: Inclusão do comando de impressão

```

1 function int main()
```

6.2

```
2
3     int n
4     n = 2
5     print(n)
6     print("\n")
7
8     return 1
9 end
10
11 main()
```

Saída do analisador semântico:

Listagem 6.14: Saída do analisador semântico para o programa nano05

```
1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7     fn_tiporet = TipoInt; fn_formais = [];
8     fn_locais =
9       [DecVar
10        (("n",
11          {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 25; pos_cnum =
12            33}),
13        TipoInt)];
14   fn_corpo =
15     [CmdAtrib
16       (Tast.ExpVar
17         (VarSimples
18           ("n",
19             {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 35;
20               pos_cnum = 39}),
21           TipoInt),
22       Tast.ExpInt (2, TipoInt));
23   CmdPrint
24     (Tast.ExpVar
25       (VarSimples
26         ("n",
27           {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 45;
28             pos_cnum = 55}),
29         TipoInt));
30   CmdPrint (Tast.ExpString ("\n", TipoString));
31   CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
32   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
33 <abstr>)
```

Saída do interpretador:

Listagem 6.15: Saída do interpretador para o programa nano05

```
1 2
2 - : unit = ()
```

Nano06

Listagem 6.16: Atribuição de uma subtração de inteiros à uma variável

```

1 function int main()
2     int n
3     n = 1 - 2
4     print(n)
5     print("\n")
6
7     return 1
8 end
9
10 main()

```

Saída do analisador semântico:

Listagem 6.17: Saída do analisador semântico para o programa nano06

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([],
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7       fn_tiporet = TipoInt; fn_formais = [];
8       fn_locais =
9         [DecVar
10          (("n",
11            {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12              28}),
13            TipoInt)]];
14     fn_corpo =
15       [CmdAtrib
16         (Tast.ExpVar
17           (VarSimples
18             ("n",
19               {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 30;
20                 pos_cnum = 34}),
21             TipoInt),
22         Tast.ExpOp ((Subtracao, TipoInt),
23           (Tast.ExpInt (1, TipoInt), TipoInt),
24           (Tast.ExpInt (2, TipoInt), TipoInt)));
25     CmdPrint
26       (Tast.ExpVar
27         (VarSimples
28           ("n",
29             {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 44;
30               pos_cnum = 54}),
31           TipoInt));
32     CmdPrint (Tast.ExpString ("\n", TipoString));
33     CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
34   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
35 <abstr>)

```

Saída do interpretador:

Listagem 6.18: Saída do interpretador para o programa nano06

```

1 -1
2 - : unit = ()

```


Nano07

Listagem 6.19: Inclusão do comando condicional

```

1 function int main()
2     int n
3     n = 1
4     if n == 1 then
5         print(n)
6         print("\n")
7     end
8     return 1
9 end
10
11 main()

```

Saída do analisador semântico:

Listagem 6.20: Saída do analisador semântico para o programa nano07

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7     fn_tiporet = TipoInt; fn_formais = [];
8     fn_locais =
9       [DecVar
10        (("n",
11          {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12            28}),
13        TipoInt)];
14   fn_corpo =
15     [CmdAtrib
16       (Tast.ExpVar
17         (VarSimples
18           ("n",
19             {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 30;
20             pos_cnum = 34}),
21           TipoInt),
22       Tast.ExpInt (1, TipoInt));
23   CmdIf
24     (Tast.ExpOp ((Equivalente, TipoBool),
25       (Tast.ExpVar
26         (VarSimples
27           ("n",
28             {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 40;
29             pos_cnum = 47}),
30           TipoInt),
31       Tast.ExpInt (1, TipoInt), TipoInt)),
32   [CmdPrint
33     (Tast.ExpVar
34       (VarSimples
35         ("n",
36           {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 60;
37           pos_cnum = 74}),
38       TipoInt));

```

```

39         CmdPrint (Tast.ExpString ("\n", TipoString))],
40         None);
41         CmdRetorno (Some (Tast.ExpInt (1, TipoInt))))]],
42     [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]],
43 <abstr>)

```

Saída do interpretador:

Listagem 6.21: Saída do interpretador para o programa nano07

```

1 1
2 - : unit = ()

```

Nano08

Listagem 6.22: Inclusão do comando condicional com parte senão

```

1 function int main()
2     int n
3     n = 1
4     if n == 1 then
5         print(n)
6         print("\n")
7     else
8         print(0)
9         print("\n")
10    end
11
12    return 1
13 end
14
15 main()

```

Saída do analisador semântico:

Listagem 6.23: Saída do analisador semântico para o programa nano08

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7       fn_tiporet = TipoInt; fn_formais = [];
8       fn_locais =
9         [DecVar
10          (("n",
11            {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12              28}),
13          TipoInt)];
14     fn_corpo =
15       [CmdAtrib
16         (Tast.ExpVar
17           (VarSimples
18             ("n",
19               {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 30;
20                 pos_cnum = 34})),
21           TipoInt),

```

```

21     Tast.ExpInt (1, TipoInt));
22   CmdIf
23     (Tast.ExpOp ((Equivalente, TipoBool),
24       (Tast.ExpVar
25         (VarSimples
26           ("n",
27             {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 40;
28              pos_cnum = 47})),
29         TipoInt),
30         TipoInt),
31       (Tast.ExpInt (1, TipoInt), TipoInt))),
32   [CmdPrint
33     (Tast.ExpVar
34       (VarSimples
35         ("n",
36           {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 60;
37            pos_cnum = 74})),
38         TipoInt));
39     CmdPrint (Tast.ExpString ("\n", TipoString))],
40   Some
41     [CmdPrint (Tast.ExpInt (0, TipoInt));
42      CmdPrint (Tast.ExpString ("\n", TipoString))];
43   CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
44   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
45   <abstr>)

```

Saída do interpretador:

Listagem 6.24: Saída do interpretador para o programa nano08

```

1 1
2 - : unit = ()

```

Nano09

Listagem 6.25: Atribuição de duas operações aritméticas sobre inteiros a uma variável

```

1 function int main()
2   int n
3   n = (1 + 1) / 2
4   if n == 1 then
5     print(n)
6     print("\n")
7   else
8     print(0)
9     print("\n")
10  end
11 end
12
13 main()

```

Saída do analisador semântico:

Listagem 6.26: Saída do analisador semântico para o programa nano09

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([],
3   [DecFun

```

```

4 {fn_nome =
5   ("main",
6    {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7   fn_tiporet = TipoInt; fn_formais = [];
8   fn_locais =
9     [DecVar
10      (("n",
11       {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12         28}),
13       TipoInt)];
14   fn_corpo =
15     [CmdAtrib
16      (Tast.ExpVar
17       (VarSimples
18        ("n",
19         {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 30;
20          pos_cnum = 34}),
21         TipoInt),
22       Tast.ExpOp ((Divisao, TipoInt),
23                  (Tast.ExpOp ((Adicao, TipoInt), (Tast.ExpInt (1, TipoInt),
24                    TipoInt),
25                    (Tast.ExpInt (1, TipoInt), TipoInt))),
26                    TipoInt),
27                  (Tast.ExpInt (2, TipoInt), TipoInt))));
28   CmdIf
29     (Tast.ExpOp ((Equivalente, TipoBool),
30                  (Tast.ExpVar
31                   (VarSimples
32                    ("n",
33                     {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 50;
34                      pos_cnum = 57}),
35                     TipoInt),
36                     TipoInt),
37                   (Tast.ExpInt (1, TipoInt), TipoInt))),
38     [CmdPrint
39      (Tast.ExpVar
40       (VarSimples
41        ("n",
42         {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 70;
43          pos_cnum = 81}),
44         TipoInt));
45      CmdPrint (Tast.ExpString ("\n", TipoString))],
46     Some
47       [CmdPrint (Tast.ExpInt (0, TipoInt));
48        CmdPrint (Tast.ExpString ("\n", TipoString))]]],
49   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
50   <abstr>)

```

Saída do interpretador:

Listagem 6.27: Saída do interpretador para o programa nano09

```

1 1
2 - : unit = ()

```

Nano10

Listagem 6.28: Atribuição de duas variáveis inteiras

```

1 function int main()
2     int n, m
3
4     n = 1
5     m = 2
6
7     if n == m then
8         print(n)
9         print("\n")
10    else
11        print(0)
12        print("\n")
13    end
14
15    return 1
16 end
17
18 main()

```

Saída do analisador semântico:

Listagem 6.29: Saída do analisador semântico para o programa nano10

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7     fn_tiporet = TipoInt; fn_formais = [];
8     fn_locais =
9       [DecVar
10        (("n",
11          {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12            28}),
13        TipoInt);
14        DecVar
15          (("m",
16            {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
17              31}),
18          TipoInt)];
19        fn_corpo =
20          [CmdAtrib
21            (Tast.ExpVar
22              (VarSimples
23                ("n",
24                  {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 38;
25                    pos_cnum = 42}),
26                TipoInt),
27            Tast.ExpInt (1, TipoInt));
28          CmdAtrib
29            (Tast.ExpVar
30              (VarSimples
31                ("m",
32                  {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 48;
33                    pos_cnum = 52}),
34                TipoInt),
35            Tast.ExpInt (2, TipoInt));
36          CmdIf

```

```

35     (Tast.ExpOp ((Equivalente, TipoBool),
36       (Tast.ExpVar
37         (VarSimples
38           ("n",
39             {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 59;
40               pos_cnum = 66})),
41         TipoInt),
42         TipoInt),
43       (Tast.ExpVar
44         (VarSimples
45           ("m",
46             {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 59;
47               pos_cnum = 71})),
48         TipoInt),
49         TipoInt))),
50   [CmdPrint
51     (Tast.ExpVar
52       (VarSimples
53         ("n",
54           {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 79;
55             pos_cnum = 93})),
56       TipoInt));
57   CmdPrint (Tast.ExpString ("\n", TipoString))],
58   Some
59     [CmdPrint (Tast.ExpInt (0, TipoInt));
60       CmdPrint (Tast.ExpString ("\n", TipoString))]);
61   CmdRetorno (Some (Tast.ExpInt (1, TipoInt))))],
62   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
63   <abstr>)

```

Saída do interpretador:

Listagem 6.30: Saída do interpretador para o programa nano10

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7       fn_tiporet = TipoInt; fn_formais = [];
8       fn_locais =
9         [DecVar
10           (("n",
11             {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12               28})),
13           TipoInt);
14         DecVar
15           (("m",
16             {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
17               31})),
18           TipoInt)];
19     fn_corpo =
20       [CmdAtrib
21         (Tast.ExpVar
22           (VarSimples
23             ("n",

```

```

24         TipoInt),
25         Tast.ExpInt (1, TipoInt));
26 CmdAtrib
27     (Tast.ExpVar
28         (VarSimples
29             ("m",
30                 {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 48;
31                     pos_cnum = 52})),
32         TipoInt),
33     Tast.ExpInt (2, TipoInt));
34 CmdIf
35     (Tast.ExpOp ((Equivalente, TipoBool),
36         (Tast.ExpVar
37             (VarSimples
38                 ("n",
39                     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 59;
40                         pos_cnum = 66})),
41             TipoInt),
42             TipoInt),
43         (Tast.ExpVar
44             (VarSimples
45                 ("m",
46                     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 59;
47                         pos_cnum = 71})),
48             TipoInt),
49             TipoInt))),
50 [CmdPrint
51     (Tast.ExpVar
52         (VarSimples
53             ("n",
54                 {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 79;
55                     pos_cnum = 93})),
56             TipoInt));
57 CmdPrint (Tast.ExpString ("\n", TipoString))],
58 Some
59     [CmdPrint (Tast.ExpInt (0, TipoInt));
60         CmdPrint (Tast.ExpString ("\n", TipoString))]);
61 CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
62 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
63 <abstr>)

```

Nano11

Listagem 6.31: Introdução do comando de repetição enquanto

```

1 function int main()
2
3     int n, m, x
4
5     n = 1
6     m = 2
7     x = 5
8
9     while x > n do
10         n = n + m
11         print(n)
12         print("\n")
13 end

```

```

14
15 end
16
17 main()

```

Saída do analisador semântico:

Listagem 6.32: Saída do analisador semântico para o programa nano11

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7     fn_tiporet = TipoInt; fn_formais = [];
8     fn_locais =
9       [DecVar
10        ((("n",
11          {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 24; pos_cnum =
12            32}),
13          TipoInt);
14        DecVar
15          ((("m",
16            {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 24; pos_cnum =
17              35}),
18            TipoInt);
19          DecVar
20            ((("x",
21              {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 24; pos_cnum =
22                38}),
23              TipoInt)]);
24        fn_corpo =
25          [CmdAtrib
26            (Tast.ExpVar
27              (VarSimples
28                (("n",
29                  {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 41;
30                    pos_cnum = 45}),
31                  TipoInt),
32              Tast.ExpInt (1, TipoInt));
33            CmdAtrib
34              (Tast.ExpVar
35                (VarSimples
36                  (("m",
37                    {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 51;
38                      pos_cnum = 55}),
39                    TipoInt),
40                Tast.ExpInt (2, TipoInt));
41            CmdAtrib
42              (Tast.ExpVar
43                (VarSimples
44                  (("x",
45                    {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 61;
46                      pos_cnum = 65}),
47                    TipoInt),
48                Tast.ExpInt (5, TipoInt));
49            CmdWhile
50              (Tast.ExpOp ((Maior, TipoBool),

```



```

48      (Tast.ExpVar
49        (VarSimples
50          ("x",
51            {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 72;
52              pos_cnum = 82})),
53        TipoInt),
54      TipoInt),
55      (Tast.ExpVar
56        (VarSimples
57          ("n",
58            {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 72;
59              pos_cnum = 86})),
60        TipoInt),
61      TipoInt)),
62    [CmdAtrib
63      (Tast.ExpVar
64        (VarSimples
65          ("n",
66            {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol = 91;
67              pos_cnum = 99})),
68        TipoInt),
69      Tast.ExpOp ((Adicao, TipoInt),
70        (Tast.ExpVar
71          (VarSimples
72            ("n",
73              {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol = 91;
74                pos_cnum = 103})),
75            TipoInt),
76          TipoInt),
77        (Tast.ExpVar
78          (VarSimples
79            ("m",
80              {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol = 91;
81                pos_cnum = 107})),
82            TipoInt),
83          TipoInt))));
84    CmdPrint
85      (Tast.ExpVar
86        (VarSimples
87          ("n",
88            {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 109;
89              pos_cnum = 123})),
90        TipoInt));
91    CmdPrint (Tast.ExpString ("\n", TipoString))]]]]],
92    [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
93    <abstr>)

```

Saída do interpretador:

Listagem 6.33: Saída do interpretador para o programa nano11

```

1 3
2 5
3 - : unit = ()

```

Nano12

Listagem 6.34: Comando condicional aninhado em um comando de repetição

```

1 function int main()
2     int n, m, x
3
4     n = 1
5     m = 2
6     x = 5
7
8     while x > n do
9         if n == m then
10            print(n)
11            print("\n")
12        else
13            print(0)
14            print("\n")
15        end
16        x = x - 1
17    end
18
19 end
20
21 main()

```

Saída do analisador semântico:

Listagem 6.35: Saída do analisador semântico para o programa nano12

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7     fn_tiporet = TipoInt; fn_formais = [];
8     fn_locais =
9     [DecVar
10      (( "n",
11        {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12          28}),
13      TipoInt);
14      DecVar
15      (( "m",
16        {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
17          31}),
18      TipoInt);
19      DecVar
20      (( "x",
21        {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
22          34}),
23      TipoInt)];
24      fn_corpo =
25      [CmdAtrib
26        (Tast.ExpVar
27          (VarSimples
28            ("n",
29              {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 37;
30                pos_cnum = 41}),
31            TipoInt),
32          Tast.ExpInt (1, TipoInt));
33        CmdAtrib

```

```

31      (Tast.ExpVar
32        (VarSimples
33          ("m",
34            {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 47;
35              pos_cnum = 51}),
36          TipoInt),
37      Tast.ExpInt (2, TipoInt));
38 CmdAtrib
39      (Tast.ExpVar
40        (VarSimples
41          ("x",
42            {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 57;
43              pos_cnum = 61}),
44          TipoInt),
45      Tast.ExpInt (5, TipoInt));
46 CmdWhile
47      (Tast.ExpOp ((Maior, TipoBool),
48        (Tast.ExpVar
49          (VarSimples
50            ("x",
51              {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 68;
52                pos_cnum = 78}),
53            TipoInt),
54          TipoInt),
55        (Tast.ExpVar
56          (VarSimples
57            ("n",
58              {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 68;
59                pos_cnum = 82}),
60            TipoInt),
61          TipoInt))),
62 [CmdIf
63      (Tast.ExpOp ((Equivalente, TipoBool),
64        (Tast.ExpVar
65          (VarSimples
66            ("n",
67              {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 87;
68                pos_cnum = 98}),
69            TipoInt),
70          TipoInt),
71        (Tast.ExpVar
72          (VarSimples
73            ("m",
74              {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 87;
75                pos_cnum = 103}),
76            TipoInt),
77          TipoInt))),
78 [CmdPrint
79      (Tast.ExpVar
80        (VarSimples
81          ("n",
82            {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol = 111;
83              pos_cnum = 122}),
84          TipoInt));
85      CmdPrint (Tast.ExpString ("\n", TipoString))),
86 Some
87      [CmdPrint (Tast.ExpInt (0, TipoInt));
88        CmdPrint (Tast.ExpString ("\n", TipoString))]);
89 CmdAtrib

```

```

90         (Tast.ExpVar
91         (VarSimples
92         ("x",
93         {Lexing.pos_fname = ""; pos_lnum = 16; pos_bol = 212;
94         pos_cnum = 220}),
95         TipoInt),
96         Tast.ExpOp ((Subtracao, TipoInt),
97         (Tast.ExpVar
98         (VarSimples
99         ("x",
100        {Lexing.pos_fname = ""; pos_lnum = 16; pos_bol = 212;
101        pos_cnum = 224}),
102        TipoInt),
103        TipoInt),
104        (Tast.ExpInt (1, TipoInt), TipoInt))))]]],
105 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
106 <abstr>)

```

Saída do interpretador:

Listagem 6.36: Saída do interpretador para o programa nano12

```

1 0
2 0
3 0
4 0
5 - : unit = ()

```

6.2.2 Micro Programas

Micro01

Listagem 6.37: Converte graus Celsius para Fahrenheit

```

1 function int main()
2   int cel, far
3   print("Tabela de conversão: Celsius -> Fahrenheit\n")
4   print("Digite a temperatura em Celsius: ")
5   cel = io.read('n')
6   far = (9*cel+160)/5
7   print("A nova temperatura eh: ")
8   print(far)
9   print(" F\n")
10 end
11
12 main()

```

Saída do analisador semântico:

Listagem 6.38: Saída do analisador semântico para o programa micro01

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3 [DecFun
4 {fn_nome =
5 ("main",
6 {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});

```

6.2

```

7      fn_tiporet = TipoInt; fn_formais = [];
8      fn_locais =
9          [DecVar
10             (("cel",
11                {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12                  28})),
13             TipoInt);
14          DecVar
15             (("far",
16                {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
17                  33})),
18             TipoInt)];
19      fn_corpo =
20          [CmdPrint
21             (Tast.ExpString
22                ("Tabela de convers\195\163o: Celsius -> Fahrenheit\n",
23                 TipoString));
24             CmdPrint
25                (Tast.ExpString ("Digite a temperatura em Celsius: ", TipoString))
26                ;
27             CmdScanInt
28                (Tast.ExpVar
29                   (VarSimples
30                      ("cel",
31                       {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 143;
32                        pos_cnum = 147})),
33                      TipoInt));
34             CmdAtrib
35                (Tast.ExpVar
36                   (VarSimples
37                      ("far",
38                       {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 166;
39                        pos_cnum = 170})),
40                      TipoInt),
41                Tast.ExpOp ((Divisao, TipoInt),
42                             (Tast.ExpOp ((Adicao, TipoInt),
43                                          (Tast.ExpOp ((Multiplicacao, TipoInt),
44                                                         (Tast.ExpInt (9, TipoInt), TipoInt),
45                                                         (Tast.ExpVar
46                                                            (VarSimples
47                                                               ("cel",
48                                                                {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 166;
49                                                                 pos_cnum = 179})),
50                                                                TipoInt),
51                                                                TipoInt))),
52                                                         (Tast.ExpInt (160, TipoInt), TipoInt))),
53                                                         TipoInt),
54                                                         (Tast.ExpInt (5, TipoInt), TipoInt))));
55             CmdPrint (Tast.ExpString ("A nova temperatura eh: ", TipoString));
56             CmdPrint
57                (Tast.ExpVar
58                   (VarSimples
59                      ("far",
60                       {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 227;
61                        pos_cnum = 237})),
62                      TipoInt));
63             CmdPrint (Tast.ExpString (" F\n", TipoString))];
64      [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],

```

62 <abstr>)

Saída do interpretador:

Listagem 6.39: Saída do interpretador para o programa micro01

```
1 Tabela de conversão: Celsius -> Fahrenheit
2 Digite a temperatura em Celsius: 30
3 A nova temperatura eh: 86 F
4 - : unit = ()
```

Micro02

Listagem 6.40: Ler dois inteiros e decide qual é maior

```
1 function int main()
2     int num1
3     int num2
4     print("Digite o primeiro numero: ")
5     num1 = io.read('n')
6     print("Digite o segundo numero: ")
7     num2 = io.read('n')
8
9     if num1 > num2 then
10        print("O primeiro numero ")
11        print(num1)
12        print(" eh maior que o segundo ")
13        print(num2)
14    else
15        print("O segundo numero ")
16        print(num2)
17        print(" eh maior que o primeiro ")
18        print(num1)
19    end
20    print("\n")
21 end
22
23 main()
```

Saída do analisador semântico:

Listagem 6.41: Saída do analisador semântico para o programa micro02

```
1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7     fn_tiporet = TipoInt; fn_formais = [];
8     fn_locais =
9       [DecVar
10        (("num1",
11          {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12            28}),
13        TipoInt);
14        DecVar
15        (("num2",
```

```

15         {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 33; pos_cnum =
16           41}},
17     TipoInt)];
18     fn_corpo =
19     [CmdPrint (Tast.ExpString ("Digite o primeiro numero: ", TipoString)
20       );
21     CmdScanInt
22     (Tast.ExpVar
23     (VarSimples
24     ("num1",
25     {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 86;
26     pos_cnum = 90}),
27     TipoInt));
28     CmdPrint (Tast.ExpString ("Digite o segundo numero: ", TipoString))
29     ;
30     CmdScanInt
31     (Tast.ExpVar
32     (VarSimples
33     ("num2",
34     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 149;
35     pos_cnum = 153}),
36     TipoInt));
37     CmdIf
38     (Tast.ExpOp ((Maior, TipoBool),
39     (Tast.ExpVar
40     (VarSimples
41     ("num1",
42     {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 174;
43     pos_cnum = 181}),
44     TipoInt),
45     TipoInt),
46     (Tast.ExpVar
47     (VarSimples
48     ("num2",
49     {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 174;
50     pos_cnum = 188}),
51     TipoInt),
52     TipoInt)),
53     [CmdPrint (Tast.ExpString ("O primeiro numero ", TipoString));
54     CmdPrint
55     (Tast.ExpVar
56     (VarSimples
57     ("num1",
58     {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 231;
59     pos_cnum = 245}),
60     TipoInt));
61     CmdPrint (Tast.ExpString (" eh maior que o segundo ", TipoString)
62       );
63     CmdPrint
64     (Tast.ExpVar
65     (VarSimples
66     ("num2",
67     {Lexing.pos_fname = ""; pos_lnum = 13; pos_bol = 294;
68     pos_cnum = 308}),
69     TipoInt))],
70     Some
71     [CmdPrint (Tast.ExpString ("O segundo numero ", TipoString));
72     CmdPrint
73     (Tast.ExpVar

```

```

70         (VarSimples
71           ("num2",
72             {Lexing.pos_fname = ""; pos_lnum = 16; pos_bol = 355;
73               pos_cnum = 369}),
74           TipoInt));
75     CmdPrint (Tast.ExpString (" eh maior que o primeiro ",
76                               TipoString));
77   CmdPrint
78     (Tast.ExpVar
79       (VarSimples
80         ("num1",
81           {Lexing.pos_fname = ""; pos_lnum = 18; pos_bol = 420;
82             pos_cnum = 434}),
83         TipoInt))]);
84   CmdPrint (Tast.ExpString ("\n", TipoString))]]],
85   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
86   <abstr>)

```

Saída do interpretador:

Listagem 6.42: Saída do interpretador para o programa micro02

```

1 Digite o primeiro numero: 10
2 Digite o segundo numero: 20
3 O segundo numero 20 eh maior que o primeiro 10
4 - : unit = ()

```

Micro03

Listagem 6.43: Lê um número e verifica se ele está entre 100 e 200

```

1 function int main()
2   int numero
3   print("Digite um numero: ")
4   numero = io.read('n')
5   if numero >= 100 then
6     if numero <= 200 then
7       print("O numero esta no intervalo entre 100 e 200")
8     else
9       print("O numero nao esta no intervalo entre 100 e 200")
10    end
11  else
12    print("O numero nao esta no intervalo entre 100 e 200")
13  end
14  print("\n")
15 end
16
17 main()

```

Saída do analisador semântico:

Listagem 6.44: Saída do analisador semântico para o programa micro03

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",

```



```

6      {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7  fn_tiporet = TipoInt; fn_formais = [];
8  fn_locais =
9      [DecVar
10         (("numero",
11            {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12              28}),
13          TipoInt)];
14  fn_corpo =
15      [CmdPrint (Tast.ExpString ("Digite um numero: ", TipoString));
16       CmdScanInt
17         (Tast.ExpVar
18          (VarSimples
19           ("numero",
20            {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 67;
21              pos_cnum = 71})),
22          TipoInt));
23       CmdIf
24         (Tast.ExpOp ((Maior_ou_Igual, TipoBool),
25                     (Tast.ExpVar
26                      (VarSimples
27                       ("numero",
28                        {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 93;
29                          pos_cnum = 100})),
30                       TipoInt),
31                      TipoInt),
32                     (Tast.ExpInt (100, TipoInt), TipoInt))),
33       [CmdIf
34         (Tast.ExpOp ((Menor_ou_Igual, TipoBool),
35                     (Tast.ExpVar
36                      (VarSimples
37                       ("numero",
38                        {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 119;
39                          pos_cnum = 130})),
40                       TipoInt),
41                      TipoInt),
42                     (Tast.ExpInt (200, TipoInt), TipoInt))),
43       [CmdPrint
44         (Tast.ExpString ("O numero esta no intervalo entre 100 e 200",
45                          TipoString))],
46       Some
47         [CmdPrint
48          (Tast.ExpString
49           ("O numero nao esta no intervalo entre 100 e 200",
50            TipoString))]],
51       Some
52         [CmdPrint
53          (Tast.ExpString ("O numero nao esta no intervalo entre 100 e
54                          200",
55                           TipoString))]];
56  CmdPrint (Tast.ExpString ("\n", TipoString))];
57  [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
58  <abstr>)

```

Saída do interpretador:

Listagem 6.45: Saída do interpretador para o programa micro03

```
1 Digite um numero: 10
```

```

2 O numero nao esta no intervalo entre 100 e 200
3 - : unit = ()

```

Micro04

Listagem 6.46: Lê números e informa quais estão entre 10 e 150

```

1 function int main()
2
3     int x, num, intervalo
4
5     intervalo = 0
6
7     for x=1, 5, 1 do
8         print("Digite um numero: ")
9         num = io.read('n')
10        if num >= 10 then
11            if num <= 150 then
12                intervalo = intervalo + 1
13            end
14        end
15    end
16
17    print("Ao total, foram digitados ")
18    print(intervalo)
19    print(" numeros no intervalo entre 10 e 150\n")
20
21 end
22
23 main()

```

Saída do analisador semântico:

Listagem 6.47: Saída do analisador semântico para o programa micro04

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7     fn_tiporet = TipoInt; fn_formais = [];
8     fn_locais =
9     [DecVar
10      (("x",
11        {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 21; pos_cnum =
12          29}),
13      TipoInt);
14    DecVar
15      (("num",
16        {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 21; pos_cnum =
17          32}),
18      TipoInt);
19    DecVar
20      (("intervalo",
21        {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 21; pos_cnum =
22          37}),
23      TipoInt)];

```

```

21 fn_corpo =
22   [CmdAtrib
23     (Tast.ExpVar
24       (VarSimples
25         ("intervalo",
26           {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 52;
27             pos_cnum = 56})),
28       TipoInt),
29     Tast.ExpInt (0, TipoInt));
30 CmdIf (Tast.ExpBool (true, TipoBool),
31   [CmdAtrib
32     (Tast.ExpVar
33       (VarSimples
34         ("x",
35           {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 75;
36             pos_cnum = 83})),
37       TipoInt),
38     Tast.ExpInt (1, TipoInt));
39 CmdWhile
40   (Tast.ExpOp ((Menor_ou_Igual, TipoBool),
41     (Tast.ExpVar
42       (VarSimples
43         ("x",
44           {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 75;
45             pos_cnum = 83})),
46       TipoInt),
47     TipoInt),
48   (Tast.ExpInt (5, TipoInt), TipoInt)),
49 [CmdPrint (Tast.ExpString ("Digite um numero: ", TipoString));
50 CmdScanInt
51   (Tast.ExpVar
52     (VarSimples
53       ("num",
54         {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 132;
55           pos_cnum = 140})),
56     TipoInt));
57 CmdIf
58   (Tast.ExpOp ((Maior_ou_Igual, TipoBool),
59     (Tast.ExpVar
60       (VarSimples
61         ("num",
62           {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol = 159;
63             pos_cnum = 170})),
64       TipoInt),
65     TipoInt),
66   (Tast.ExpInt (10, TipoInt), TipoInt)),
67 [CmdIf
68   (Tast.ExpOp ((Menor_ou_Igual, TipoBool),
69     (Tast.ExpVar
70       (VarSimples
71         ("num",
72           {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 185;
73             pos_cnum = 193})),
74       TipoInt),
75     TipoInt),
76   (Tast.ExpInt (150, TipoInt), TipoInt)),
77 [CmdAtrib
78   (Tast.ExpVar
79     (VarSimples

```

```

80         ("intervalo",
81         {Lexing.pos_fname = ""; pos_lnum = 12; pos_bol = 209;
82         pos_cnum = 218}),
83         TipoInt),
84     Tast.ExpOp ((Adicao, TipoInt),
85     (Tast.ExpVar
86     (VarSimples
87     ("intervalo",
88     {Lexing.pos_fname = ""; pos_lnum = 12; pos_bol =
89     209;
90     pos_cnum = 230})),
91     TipoInt),
92     (Tast.ExpInt (1, TipoInt), TipoInt)))]],
93     None)],
94     None);
95 CmdAtrib
96 (Tast.ExpVar
97 (VarSimples
98 ("x",
99 {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 75;
100 pos_cnum = 83})),
101 TipoInt),
102 Tast.ExpOp ((Adicao, TipoInt),
103 (Tast.ExpVar
104 (VarSimples
105 ("x",
106 {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 75;
107 pos_cnum = 83})),
108 TipoInt),
109 TipoInt),
110 (Tast.ExpInt (1, TipoInt), TipoInt)))]],
111 None);
112 CmdPrint (Tast.ExpString ("Ao total, foram digitados ", TipoString)
113 );
114 CmdPrint
115 (Tast.ExpVar
116 (VarSimples
117 ("intervalo",
118 {Lexing.pos_fname = ""; pos_lnum = 18; pos_bol = 314;
119 pos_cnum = 324})),
120 TipoInt));
121 CmdPrint
122 (Tast.ExpString (" numeros no intervalo entre 10 e 150\n",
123 TipoString))]],
124 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
125 <abstr>)

```

Saída do interpretador:

Listagem 6.48: Saída do interpretador para o programa micro04

```

1 Digite um numero: 100
2 Digite um numero: 50
3 Digite um numero: 200
4 Digite um numero: 300
5 Digite um numero: 1
6 Ao total, foram digitados 2 numeros no intervalo entre 10 e 150
7 - : unit = ()

```

Micro05

Listagem 6.49: Lê strings e caracteres

```

1  function int main()
2      string nome
3      int sexo, x, h, m
4
5      x = 1
6      h = 0
7      m = 0
8
9      for x=1, 5, 1 do
10         print("Digite o nome: ")
11         nome = io.read('s')
12         print("1 - Homem ou 2 - Mulher: ")
13         sexo = io.read('n')
14
15         if sexo == 1 then
16             h = h + 1
17         else
18             if sexo == 2 then
19                 m = m + 1
20             else
21                 print("Sexo so pode ser 1(Homem) ou 2(Mulher)!\n")
22             end
23         end
24     end
25     print("Foram inseridos ")
26     print(h)
27     print(" Homens\n")
28     print("Foram inseridos ")
29     print(m)
30     print(" Mulheres\n")
31 end
32
33 main()

```

Saída do analisador semântico:

Listagem 6.50: Saída do analisador semântico para o programa micro05

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([],
3  [DecFun
4    {fn_nome =
5      ("main",
6        {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7    fn_tiporet = TipoInt; fn_formais = [];
8    fn_locais =
9      [DecVar
10        (("nome",
11          {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12            31}),
13        TipoString);
14      DecVar
15        (("sexo",
16          {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 36; pos_cnum =
17            44}),

```

```

16     TipoInt);
17   DecVar
18     ("x",
19     {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 36; pos_cnum =
20       50}),
21     TipoInt);
22   DecVar
23     ("h",
24     {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 36; pos_cnum =
25       53}),
26     TipoInt);
27   DecVar
28     ("m",
29     {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 36; pos_cnum =
30       56}),
31     TipoInt)];
32   fn_corpo =
33   [CmdAtrib
34     (Tast.ExpVar
35     (VarSimples
36     ("x",
37     {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 63;
38       pos_cnum = 67}),
39     TipoInt),
40     Tast.ExpInt (1, TipoInt));
41   CmdAtrib
42     (Tast.ExpVar
43     (VarSimples
44     ("h",
45     {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 73;
46       pos_cnum = 77}),
47     TipoInt),
48     Tast.ExpInt (0, TipoInt));
49   CmdAtrib
50     (Tast.ExpVar
51     (VarSimples
52     ("m",
53     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 83;
54       pos_cnum = 87}),
55     TipoInt),
56     Tast.ExpInt (0, TipoInt));
57   CmdIf (Tast.ExpBool (true, TipoBool),
58   [CmdAtrib
59     (Tast.ExpVar
60     (VarSimples
61     ("x",
62     {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 95;
63       pos_cnum = 103}),
64     TipoInt),
65     Tast.ExpInt (1, TipoInt));
66   CmdWhile
67     (Tast.ExpOp ((Menor_ou_Igual, TipoBool),
68     (Tast.ExpVar
69     (VarSimples
70     ("x",
71     {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 95;

```

```

72      (Tast.ExpInt (5, TipoInt), TipoInt)),
73 [CmdPrint (Tast.ExpString ("Digite o nome: ", TipoString));
74 CmdScanString
75   (Tast.ExpVar
76     (VarSimples
77       ("nome",
78        {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 149;
79         pos_cnum = 157})),
80     TipoString));
81 CmdPrint
82   (Tast.ExpString ("1 - Homem ou 2 - Mulher: ", TipoString));
83 CmdScanInt
84   (Tast.ExpVar
85     (VarSimples
86       ("sexo",
87        {Lexing.pos_fname = ""; pos_lnum = 13; pos_bol = 220;
88         pos_cnum = 225})),
89     TipoInt));
90 CmdIf
91   (Tast.ExpOp ((Equivalente, TipoBool),
92     (Tast.ExpVar
93       (VarSimples
94         ("sexo",
95          {Lexing.pos_fname = ""; pos_lnum = 15; pos_bol = 254;
96           pos_cnum = 262})),
97       TipoInt),
98       TipoInt),
99   (Tast.ExpInt (1, TipoInt), TipoInt)),
100 [CmdAtrib
101   (Tast.ExpVar
102     (VarSimples
103       ("h",
104        {Lexing.pos_fname = ""; pos_lnum = 16; pos_bol = 277;
105         pos_cnum = 282})),
106     TipoInt),
107   Tast.ExpOp ((Adicao, TipoInt),
108     (Tast.ExpVar
109       (VarSimples
110         ("h",
111          {Lexing.pos_fname = ""; pos_lnum = 16; pos_bol = 277;
112           pos_cnum = 286})),
113       TipoInt),
114       TipoInt),
115   (Tast.ExpInt (1, TipoInt), TipoInt))],
116 Some
117   [CmdIf
118     (Tast.ExpOp ((Equivalente, TipoBool),
119       (Tast.ExpVar
120         (VarSimples
121           ("sexo",
122            {Lexing.pos_fname = ""; pos_lnum = 18; pos_bol =
123              302;
124              pos_cnum = 317})),
125           TipoInt),
126           TipoInt),
127     (Tast.ExpInt (2, TipoInt), TipoInt)),
128   [CmdAtrib
129     (Tast.ExpVar
130       (VarSimples

```

```

130             ("m",
131             {Lexing.pos_fname = ""; pos_lnum = 19; pos_bol =
              332;
132             pos_cnum = 341})),
133             TipoInt),
134             Tast.ExpOp ((Adicao, TipoInt), (Tast.ExpVar (...), ...),
135             ...));
136             ...],
137             ...);
138             ...]);
139             ...]);
140             ...],
141             ...);
142             ...]};
143             ...],
144             ...),
145             ...)

```

Saída do interpretador:

Listagem 6.51: Saída do interpretador para o programa micro05

```

1 Digite o nome: matheus
2 1 - Homem ou 2 - Mulher: 1
3 Digite o nome: lucas
4 1 - Homem ou 2 - Mulher: 1
5 Digite o nome: paula
6 1 - Homem ou 2 - Mulher: 2
7 Digite o nome: joaquim
8 1 - Homem ou 2 - Mulher: 3
9 Sexo so pode ser 1(Homem) ou 2(Mulher)!
10 Digite o nome: lucia
11 1 - Homem ou 2 - Mulher: 2
12 Foram inseridos 2 Homens
13 Foram inseridos 2 Mulheres
14 - : unit = ()

```

Micro06

Listagem 6.52: Escreve um número lido por extenso

```

1 function int main()
2     int num
3     print("Digite um numero de 1 a 5: ")
4     num = io.read('n')
5     if num == 1 then
6         print("Um\n")
7     else
8         if num == 2 then
9             print("Dois\n")
10        else
11            if num == 3 then
12                print("Tres\n")
13            else
14                if num == 4 then
15                    print("Quatro\n")
16                else
17                    if num == 5 then

```



```

18         print("Cinco\n")
19     else
20         print("Numero Invalido!!!\n")
21     end
22 end
23 end
24 end
25 end
26 end
27
28 main()

```

Saída do analisador semântico:

Listagem 6.53: Saída do analisador semântico para o programa micro06

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7     fn_tiporet = TipoInt; fn_formais = [];
8     fn_locais =
9       [DecVar
10        (("num",
11          {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12            28}),
13        TipoInt)];
14 fn_corpo =
15   [CmdPrint (Tast.ExpString ("Digite um numero de 1 a 5: ", TipoString
16     ));
17   CmdScanInt
18     (Tast.ExpVar
19       (VarSimples
20         ("num",
21           {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 73;
22             pos_cnum = 77}),
23         TipoInt));
24   CmdIf
25     (Tast.ExpOp ((Equivalente, TipoBool),
26       (Tast.ExpVar
27         (VarSimples
28           ("num",
29             {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 96;
30               pos_cnum = 103}),
31           TipoInt),
32         TipoInt),
33       (Tast.ExpInt (1, TipoInt), TipoInt)),
34     [CmdPrint (Tast.ExpString ("Um\n", TipoString))],
35     Some
36       [CmdIf
37         (Tast.ExpOp ((Equivalente, TipoBool),
38           (Tast.ExpVar
39             (VarSimples
40               ("num",
41                 {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 145;
42                   pos_cnum = 156}),
43               TipoInt),

```

```

42         TipoInt),
43         (Tast.ExpInt (2, TipoInt), TipoInt)),
44     [CmdPrint (Tast.ExpString ("Dois\n", TipoString))],
45     Some
46     [CmdIf
47         (Tast.ExpOp ((Equivalente, TipoBool),
48         (Tast.ExpVar
49         (VarSimples
50         ("num",
51         {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 207;
52         pos_cnum = 214})),
53         TipoInt),
54         TipoInt),
55         (Tast.ExpInt (3, TipoInt), TipoInt)),
56     [CmdPrint (Tast.ExpString ("Tres\n", TipoString))],
57     Some
58     [CmdIf
59         (Tast.ExpOp ((Equivalente, TipoBool),
60         (Tast.ExpVar
61         (VarSimples
62         ("num",
63         {Lexing.pos_fname = ""; pos_lnum = 14; pos_bol =
64         271;
65         pos_cnum = 288})),
66         TipoInt),
67         TipoInt),
68         (Tast.ExpInt (4, TipoInt), TipoInt)),
69     [CmdPrint (Tast.ExpString ("Quatro\n", TipoString))],
70     Some
71     [CmdIf
72         (Tast.ExpOp ((Equivalente, TipoBool),
73         (Tast.ExpVar
74         (VarSimples
75         ("num",
76         {Lexing.pos_fname = ""; pos_lnum = 17;
77         pos_bol = 353; pos_cnum = 373})),
78         TipoInt),
79         TipoInt),
80         (Tast.ExpInt (5, TipoInt), TipoInt)),
81     [CmdPrint (Tast.ExpString ("Cinco\n", TipoString))],
82     Some
83     [CmdPrint
84         (Tast.ExpString ("Numero Invalido!!!\n", TipoString
85         ))]]]]]]]]],
86     [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
87     <abstr>)

```

Saída do interpretador:

Listagem 6.54: Saída do interpretador para o programa micro06

```

1 Digite um numero de 1 a 5: 3
2 Tres
3 - : unit = ()

```

Micro07

Listagem 6.55: Decide se os números são positivos, zeros ou negativos

```

1 function int main()
2     int programa
3     int numero
4     int opc
5
6     programa = 1
7
8     while programa == 1 do
9         print("Digite um numero: ")
10        numero = io.read('n')
11        if numero > 0 then
12            print("Positivo")
13        else
14            if numero == 0 then
15                print("0 numero eh igual a 0")
16            end
17            if numero < 0 then
18                print("Negativo")
19            end
20        end
21
22        print("\nDeseja finalizar? (S-1/N-2) ")
23        opc = io.read('n')
24        if opc == 1 then
25            programa = 0
26        end
27    end
28 end
29
30 main()

```

Saída do analisador semântico:

Listagem 6.56: Saída do analisador semântico para o programa micro07

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7     fn_tiporet = TipoInt; fn_formais = [];
8     fn_locais =
9     [DecVar
10      (("programa",
11        {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12          28}),
13      TipoInt);
14      DecVar
15      (("numero",
16        {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 37; pos_cnum =
17          45}),
18      TipoInt);
19      DecVar
20      (("opc",
21        {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 52; pos_cnum =
22          60}),
23      TipoInt)];
24     fn_corpo =

```

```

22 [CmdAtrib
23   (Tast.ExpVar
24     (VarSimples
25       ("programa",
26         {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 65;
27          pos_cnum = 69})),
28     TipoInt),
29   Tast.ExpInt (1, TipoInt));
30 CmdWhile
31   (Tast.ExpOp ((Equivalente, TipoBool),
32     (Tast.ExpVar
33       (VarSimples
34         ("programa",
35           {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 87;
36            pos_cnum = 97})),
37         TipoInt),
38         TipoInt),
39     (Tast.ExpInt (1, TipoInt), TipoInt))),
40 [CmdPrint (Tast.ExpString ("Digite um numero: ", TipoString));
41 CmdScanInt
42   (Tast.ExpVar
43     (VarSimples
44       ("numero",
45         {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol = 147;
46          pos_cnum = 152})),
47     TipoInt));
48 CmdIf
49   (Tast.ExpOp ((Maior, TipoBool),
50     (Tast.ExpVar
51       (VarSimples
52         ("numero",
53           {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 174;
54            pos_cnum = 182})),
55         TipoInt),
56         TipoInt),
57     (Tast.ExpInt (0, TipoInt), TipoInt))),
58 [CmdPrint (Tast.ExpString ("Positivo", TipoString))],
59 Some
60   [CmdIf
61     (Tast.ExpOp ((Equivalente, TipoBool),
62       (Tast.ExpVar
63         (VarSimples
64           ("numero",
65             {Lexing.pos_fname = ""; pos_lnum = 14; pos_bol = 231;
66              pos_cnum = 239})),
67           TipoInt),
68           TipoInt),
69       (Tast.ExpInt (0, TipoInt), TipoInt))),
70     [CmdPrint (Tast.ExpString ("O numero eh igual a 0",
71       TipoString))],
71     None);
72 CmdIf
73   (Tast.ExpOp ((Menor, TipoBool),
74     (Tast.ExpVar
75       (VarSimples
76         ("numero",
77           {Lexing.pos_fname = ""; pos_lnum = 17; pos_bol = 305;
78            pos_cnum = 313})),
79         TipoInt),

```

```

80         TipoInt),
81         (Tast.ExpInt (0, TipoInt), TipoInt)),
82         [CmdPrint (Tast.ExpString ("Negativo", TipoString)], None)])
83     ;
84 CmdPrint
85   (Tast.ExpString ("\nDeseja finalizar? (S-1/N-2) ", TipoString));
86 CmdScanInt
87   (Tast.ExpVar
88     (VarSimples
89       ("opc",
90        {Lexing.pos_fname = ""; pos_lnum = 23; pos_bol = 423;
91         pos_cnum = 428})),
92     TipoInt));
93 CmdIf
94   (Tast.ExpOp ((Equivalente, TipoBool),
95     (Tast.ExpVar
96       (VarSimples
97         ("opc",
98          {Lexing.pos_fname = ""; pos_lnum = 24; pos_bol = 447;
99           pos_cnum = 455})),
100      TipoInt),
101      TipoInt),
102   (Tast.ExpInt (1, TipoInt), TipoInt)),
103 [CmdAtrib
104   (Tast.ExpVar
105     (VarSimples
106       ("programa",
107        {Lexing.pos_fname = ""; pos_lnum = 25; pos_bol = 469;
108         pos_cnum = 474})),
109      TipoInt),
110   Tast.ExpInt (0, TipoInt))],
111 None)]])],
112 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
113 <abstr>

```

Saída do interpretador:

Listagem 6.57: Saída do interpretador para o programa micro07

```

1 Digite um numero: 100
2 Positivo
3 Deseja finalizar? (S-1/N-2) 2
4 Digite um numero: -9
5 Negativo
6 Deseja finalizar? (S-1/N-2) 2
7 Digite um numero: 0
8 O numero eh igual a 0
9 Deseja finalizar? (S-1/N-2) 1
10 - : unit = ()

```

Micro08

Listagem 6.58: Decide se um número é maior ou menor que 10

```

1 function int main()
2   int numero
3   numero = 1
4   while numero ~= 0 do

```

```

5     print("Digite um numero: ")
6     numero = io.read('n')
7     if numero > 10 then
8         print("O numero ")
9             print(numero)
10            print(" eh maior que 10\n")
11    else
12        print("O numero ")
13            print(numero)
14            print(" eh menor que 10\n")
15    end
16 end
17 end
18
19 main()

```

Saída do analisador semântico:

Listagem 6.59: Saída do analisador semântico para o programa micro08

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7     fn_tiporet = TipoInt; fn_formais = [];
8     fn_locais =
9       [DecVar
10        ((("numero",
11          {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12            28}),
13        TipoInt)];
14    fn_corpo =
15      [CmdAtrib
16        (Tast.ExpVar
17          (VarSimples
18            ("numero",
19              {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 35;
20                pos_cnum = 39}),
21            TipoInt),
22        Tast.ExpInt (1, TipoInt));
23    CmdWhile
24      (Tast.ExpOp ((Nao_Equivalente, TipoBool),
25        (Tast.ExpVar
26          (VarSimples
27            ("numero",
28              {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 50;
29                pos_cnum = 60}),
30            TipoInt),
31        TipoInt),
32        (Tast.ExpInt (0, TipoInt), TipoInt)),
33    [CmdPrint (Tast.ExpString ("Digite um numero: ", TipoString));
34    CmdScanInt
35      (Tast.ExpVar
36        (VarSimples
37          ("numero",
38            {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 108;
39              pos_cnum = 113}),

```

```

39         TipoInt));
40     CmdIf
41     (Tast.ExpOp ((Maior, TipoBool),
42     (Tast.ExpVar
43     (VarSimples
44     ("numero",
45     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 135;
46     pos_cnum = 143})),
47     TipoInt),
48     TipoInt),
49     (Tast.ExpInt (10, TipoInt), TipoInt)),
50 [CmdPrint (Tast.ExpString ("O numero ", TipoString));
51 CmdPrint
52 (Tast.ExpVar
53 (VarSimples
54 ("numero",
55 {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 184;
56 pos_cnum = 202})),
57 TipoInt));
58 CmdPrint (Tast.ExpString (" eh maior que 10\n", TipoString))],
59 Some
60 [CmdPrint (Tast.ExpString ("O numero ", TipoString));
61 CmdPrint
62 (Tast.ExpVar
63 (VarSimples
64 ("numero",
65 {Lexing.pos_fname = ""; pos_lnum = 13; pos_bol = 284;
66 pos_cnum = 302})),
67 TipoInt));
68 CmdPrint (Tast.ExpString (" eh menor que 10\n", TipoString))]
69 ]]]],
70 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
71 <abstr>

```

Saída do interpretador:

Listagem 6.60: Saída do interpretador para o programa micro08

```

1 Digite um numero: 100
2 O numero 100 eh maior que 10
3 Digite um numero: 9
4 O numero 9 eh menor que 10
5 Digite um numero: 0
6 O numero 0 eh menor que 10
7 - : unit = ()

```

Micro09

Listagem 6.61: Cálculo de preços

```

1 function int main()
2     float preco, venda, novo_preco
3
4     print("Digite o preco: ")
5     preco = io.read('f')
6     print("Digite a venda: ")
7     venda = io.read('f')
8     if venda < 500.0 or preco < 30.0 then

```

```

9      novo_preco = preco + preco * 0.1
10  else if (venda >= 500.0 and venda < 1200.0) or (preco >= 30.0 and
      preco < 80.0) then
11      novo_preco = preco + preco * 0.15
12      else if venda >= 1200.0 or preco >= 80.0 then
13      novo_preco = preco + preco * 0.2
14      end
15      end
16  end
17  print("O novo preco eh ")
18  print(novo_preco)
19  print("\n")
20 end
21
22 main()

```

Saída do analisador semântico:

Listagem 6.62: Saída do analisador semântico para o programa micro09

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("main",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7       fn_tiporet = TipoInt; fn_formais = [];
8       fn_locais =
9         [DecVar
10          (("preco",
11            {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12              30}),
13            TipoFloat);
14          DecVar
15            (("venda",
16              {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
17                37}),
18              TipoFloat);
19          DecVar
20            (("novo_preco",
21              {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
22                44}),
23              TipoFloat)];
24   fn_corpo =
25     [CmdPrint (Tast.ExpString ("Digite o preco: ", TipoString));
26     CmdScanFloat
27       (Tast.ExpVar
28         (VarSimples
29           ("preco",
30             {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 86;
31               pos_cnum = 90}),
32           TipoFloat));
33     CmdPrint (Tast.ExpString ("Digite a venda: ", TipoString));
34     CmdScanFloat
35       (Tast.ExpVar
36         (VarSimples
37           ("venda",
38             {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 141;
39               pos_cnum = 145}),

```



```

37     TipoFloat));
38 CmdIf
39     (Tast.ExpOp ((Or, TipoBool),
40     (Tast.ExpOp ((Menor, TipoBool),
41     (Tast.ExpVar
42     (VarSimples
43     ("venda",
44     {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 166;
45     pos_cnum = 173})),
46     TipoFloat),
47     TipoFloat),
48     (Tast.ExpFloat (500., TipoFloat), TipoFloat))),
49     TipoBool),
50     (Tast.ExpOp ((Menor, TipoBool),
51     (Tast.ExpVar
52     (VarSimples
53     ("preco",
54     {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 166;
55     pos_cnum = 190})),
56     TipoFloat),
57     TipoFloat),
58     (Tast.ExpFloat (30., TipoFloat), TipoFloat))),
59     TipoBool)),
60 [CmdAtrib
61     (Tast.ExpVar
62     (VarSimples
63     ("novo_preco",
64     {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 208;
65     pos_cnum = 216})),
66     TipoFloat),
67     Tast.ExpOp ((Adicao, TipoFloat),
68     (Tast.ExpVar
69     (VarSimples
70     ("preco",
71     {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 208;
72     pos_cnum = 229})),
73     TipoFloat),
74     TipoFloat),
75     (Tast.ExpOp ((Multiplicacao, TipoFloat),
76     (Tast.ExpVar
77     (VarSimples
78     ("preco",
79     {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 208;
80     pos_cnum = 237})),
81     TipoFloat),
82     TipoFloat),
83     (Tast.ExpFloat (0.1, TipoFloat), TipoFloat))),
84     TipoFloat))],
85 Some
86 [CmdIf
87     (Tast.ExpOp ((Or, TipoBool),
88     (Tast.ExpOp ((And, TipoBool),
89     (Tast.ExpOp ((Maior_ou_Igual, TipoBool),
90     (Tast.ExpVar
91     (VarSimples
92     ("venda",
93     {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol =
94     249;
95     pos_cnum = 262})),

```

```

95         TipoFloat),
96         TipoFloat),
97         (Tast.ExpFloat (500., TipoFloat), TipoFloat)),
98         TipoBool),
99         (Tast.ExpOp ((Menor, TipoBool),
100          (Tast.ExpVar
101           (VarSimples
102            ("venda",
103             {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol =
104              249;
105              pos_cnum = 281})),
106            TipoFloat),
107            TipoFloat),
108            (Tast.ExpFloat (1200., TipoFloat), TipoFloat)),
109            TipoBool)),
110         (Tast.ExpOp ((And, TipoBool),
111          (Tast.ExpOp ((Maior_ou_Igual, TipoBool),
112           (Tast.ExpVar
113            (VarSimples
114             ("preco",
115              {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol =
116               249;
117               pos_cnum = 301})),
118              TipoFloat),
119              TipoFloat),
120              (Tast.ExpFloat (30., TipoFloat), TipoFloat)),
121              TipoBool),
122              (Tast.ExpOp ((Menor, TipoBool),
123               (Tast.ExpVar
124                (VarSimples
125                 ("preco",
126                  {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol =
127                   249;
128                   pos_cnum = 319})),
129                  TipoFloat),
130                  ...)),
131                  ...)),
132                  ...));
133          ...]);
134      ...]];
135      ...],
136      ...),
137      ...)

```

Saída do interpretador:

Listagem 6.63: Saída do interpretador para o programa micro09

```

1 Digite o preco: 50
2 Digite a venda: 500
3 O novo preco eh 57.5
4 - : unit = ()

```

Micro10

Listagem 6.64: Calcula o fatorial de um número

```

1 function int fatorial(int n)
2     int x, y
3
4     if n <= 1 then
5         return 1
6     else
7         y = n - 1
8         x = fatorial(y)
9         return n * x
10    end
11 end
12
13 function int main()
14
15     int numero, fat
16
17     print("Digite um numero: ")
18     numero = io.read('n')
19     fat = fatorial(numero)
20     print("O fatorial de ")
21     print(numero)
22     print(" eh ")
23     print(fat)
24     print("\n")
25
26     return 1
27
28 end
29
30 main()

```

Saída do analisador semântico:

Listagem 6.65: Saída do analisador semântico para o programa micro10

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([,
3   [DecFun
4     {fn_nome =
5       ("fatorial",
6         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7       fn_tiporet = TipoInt;
8       fn_formais =
9         [(("n",
10          {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum =
11            26}),
12          TipoInt)];
13       fn_locais =
14         [DecVar
15           (("x",
16            {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 29; pos_cnum =
17              37}),
18            TipoInt);
19           DecVar
20             (("y",
21              {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 29; pos_cnum =
22                40}),
23              TipoInt)];

```

```

21 fn_corpo =
22   [CmdIf
23     (Tast.ExpOp ((Menor_ou_Igual, TipoBool),
24       (Tast.ExpVar
25         (VarSimples
26           ("n",
27             {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 43;
28               pos_cnum = 50})),
29         TipoInt),
30         TipoInt),
31     (Tast.ExpInt (1, TipoInt), TipoInt)),
32   [CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))],
33   Some
34     [CmdAtrib
35       (Tast.ExpVar
36         (VarSimples
37           ("y",
38             {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 81;
39               pos_cnum = 89})),
40         TipoInt),
41       Tast.ExpOp ((Subtracao, TipoInt),
42         (Tast.ExpVar
43           (VarSimples
44             ("n",
45               {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 81;
46                 pos_cnum = 93})),
47           TipoInt),
48           TipoInt),
49       (Tast.ExpInt (1, TipoInt), TipoInt))];
50   CmdAtrib
51     (Tast.ExpVar
52       (VarSimples
53         ("x",
54           {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 99;
55             pos_cnum = 107})),
56       TipoInt),
57     Tast.ExpChamada ("fatorial",
58       [Tast.ExpVar
59         (VarSimples
60           ("y",
61             {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 99;
62               pos_cnum = 120})),
63         TipoInt)],
64     TipoInt));
65   CmdRetorno
66     (Some
67       (Tast.ExpOp ((Multiplicacao, TipoInt),
68         (Tast.ExpVar
69           (VarSimples
70             ("n",
71               {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 123;
72                 pos_cnum = 131})),
73           TipoInt),
74           TipoInt),
75       (Tast.ExpVar
76         (VarSimples
77           ("x",
78             {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 123;
79               pos_cnum = 135})),

```

```

80         TipoInt),
81         TipoInt))))))]]];
82 DecFun
83 {fn_nome =
84   ("main",
85     {Lexing.pos_fname = ""; pos_lnum = 13; pos_bol = 150; pos_cnum =
86       163});
87   fn_tiporet = TipoInt; fn_formais = [];
88   fn_locais =
89     [DecVar
90       (("numero",
91         {Lexing.pos_fname = ""; pos_lnum = 15; pos_bol = 171;
92           pos_cnum = 179}),
93         TipoInt);
94       DecVar
95         (("fat",
96           {Lexing.pos_fname = ""; pos_lnum = 15; pos_bol = 171;
97             pos_cnum = 187}),
98           TipoInt)];
99   fn_corpo =
100     [CmdPrint (Tast.ExpString ("Digite um numero: ", TipoString));
101     CmdScanInt
102       (Tast.ExpVar
103         (VarSimples
104           ("numero",
105             {Lexing.pos_fname = ""; pos_lnum = 18; pos_bol = 224;
106               pos_cnum = 228}),
107             TipoInt));
108     CmdAtrib
109       (Tast.ExpVar
110         (VarSimples
111           ("fat",
112             {Lexing.pos_fname = ""; pos_lnum = 19; pos_bol = 250;
113               pos_cnum = 254}),
114             TipoInt),
115       Tast.ExpChamada ("fatorial",
116         [Tast.ExpVar
117           (VarSimples
118             ("numero",
119               {Lexing.pos_fname = ""; pos_lnum = 19; pos_bol = 250;
120                 pos_cnum = 269}),
121               TipoInt)],
122         TipoInt));
123     CmdPrint (Tast.ExpString ("O fatorial de ", TipoString));
124     CmdPrint
125       (Tast.ExpVar
126         (VarSimples
127           ("numero",
128             {Lexing.pos_fname = ""; pos_lnum = 21; pos_bol = 305;
129               pos_cnum = 315}),
130             TipoInt));
131     CmdPrint (Tast.ExpString (" eh ", TipoString));
132     CmdPrint
133       (Tast.ExpVar
134         (VarSimples
135           ("fat",
136             {Lexing.pos_fname = ""; pos_lnum = 23; pos_bol = 341;
137               pos_cnum = 351}),
138             TipoInt));

```

```

138         CmdPrint (Tast.ExpString ("\n", TipoString));
139         CmdRetorno (Some (Tast.ExpInt (1, TipoInt))))]],
140     ...),
141     ...)

```

Saída do interpretador:

Listagem 6.66: Saída do interpretador para o programa micro10

```

1 Digite um numero: 10
2 O fatorial de 10 eh 3628800
3 - : unit = ()

```

Micro11

Listagem 6.67: Decide se um número é positivo, zero ou negativo com auxílio de uma função

```

1 function int verifica(int n)
2     int res
3     if n > 0 then
4         res = 1
5     else if n < 0 then
6         res = -1
7     else
8         res = 0
9     end
10    end
11    return res
12 end
13
14 function int main()
15
16     int numero, x
17
18     print("Digite um numero: ")
19     numero = io.read('n')
20
21     x = verifica(numero)
22
23     if x == 1 then
24         print("Numero positivo\n")
25     else if x == 0 then
26         print("Zero")
27     else
28         print("Numero negativo\n")
29     end
30 end
31 end
32
33 main()

```

Saída do analisador semântico:

Listagem 6.68: Saída do analisador semântico para o programa micro11

```

1 - : Tast.expressao Ast.programa * Ambiente.t =
2 (Programa ([],
3  [DecFun

```

```

4      {fn_nome =
5        ("verifica",
6        {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 13});
7      fn_tiporet = TipoInt;
8      fn_formais =
9        [(("n",
10         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum =
11           26}),
12         TipoInt)];
13      fn_locais =
14        [DecVar
15          (("res",
16           {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 29; pos_cnum =
17             37}),
18           TipoInt)];
19      fn_corpo =
20        [CmdIf
21          (Tast.ExpOp ((Maior, TipoBool),
22            (Tast.ExpVar
23              (VarSimples
24                ("n",
25                 {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 41;
26                   pos_cnum = 48}),
27                 TipoInt),
28                 TipoInt),
29                 (Tast.ExpInt (0, TipoInt), TipoInt))),
30          [CmdAtrib
31            (Tast.ExpVar
32              (VarSimples
33                ("res",
34                 {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 59;
35                   pos_cnum = 67}),
36                 TipoInt),
37                 Tast.ExpInt (1, TipoInt))),
38          Some
39          [CmdIf
40            (Tast.ExpOp ((Menor, TipoBool),
41              (Tast.ExpVar
42                (VarSimples
43                  ("n",
44                   {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 75;
45                     pos_cnum = 87}),
46                   TipoInt),
47                   TipoInt),
48                   (Tast.ExpInt (0, TipoInt), TipoInt))),
49            [CmdAtrib
50              (Tast.ExpVar
51                (VarSimples
52                  ("res",
53                   {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 99;
54                     pos_cnum = 100}),
55                   TipoInt),
56                   Tast.ExpInt (-1, TipoInt))),
57            Some
58            [CmdAtrib
59              (Tast.ExpVar
60                (VarSimples
61                  ("res",
62                   {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 123;

```

```

61         pos_cnum = 128)),
62         TipoInt),
63         Tast.ExpInt (0, TipoInt))]]]);
64 CmdRetorno
65     (Some
66         (Tast.ExpVar
67             (VarSimples
68                 ("res",
69                     {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 149;
70                     pos_cnum = 160}),
71                 TipoInt))));
72 DecFun
73 {fn_nome =
74     ("main",
75         {Lexing.pos_fname = ""; pos_lnum = 14; pos_bol = 169; pos_cnum =
76         182});
77 fn_tiporet = TipoInt; fn_formais = [];
78 fn_locais =
79     [DecVar
80         (("numero",
81             {Lexing.pos_fname = ""; pos_lnum = 16; pos_bol = 190;
82             pos_cnum = 198}),
83             TipoInt);
84     DecVar
85         (("x",
86             {Lexing.pos_fname = ""; pos_lnum = 16; pos_bol = 190;
87             pos_cnum = 206}),
88             TipoInt)];
89 fn_corpo =
90     [CmdPrint (Tast.ExpString ("Digite um numero: ", TipoString));
91     CmdScanInt
92         (Tast.ExpVar
93             (VarSimples
94                 ("numero",
95                     {Lexing.pos_fname = ""; pos_lnum = 19; pos_bol = 241;
96                     pos_cnum = 245}),
97                 TipoInt));
98     CmdAtrib
99         (Tast.ExpVar
100             (VarSimples
101                 ("x",
102                     {Lexing.pos_fname = ""; pos_lnum = 21; pos_bol = 272;
103                     pos_cnum = 276}),
104                 TipoInt),
105         Tast.ExpChamada ("verifica",
106             [Tast.ExpVar
107                 (VarSimples
108                     ("numero",
109                         {Lexing.pos_fname = ""; pos_lnum = 21; pos_bol = 272;
110                         pos_cnum = 289}),
111                     TipoInt)],
112             TipoInt));
113 CmdIf
114     (Tast.ExpOp ((Equivalente, TipoBool),
115         (Tast.ExpVar
116             (VarSimples
117                 ("x",
118                     {Lexing.pos_fname = ""; pos_lnum = 23; pos_bol = 302;
119                     pos_cnum = 309})),

```



```

119         TipoInt),
120         TipoInt),
121         (Tast.ExpInt (1, TipoInt), TipoInt)),
122     [CmdPrint (Tast.ExpString ("Numero positivo\n", TipoString))],
123     Some
124     [CmdIf
125       (Tast.ExpOp ((Equivalente, TipoBool),
126         (Tast.ExpVar
127           (VarSimples
128             ("x",
129               {Lexing.pos_fname = ""; pos_lnum = 25; pos_bol = 356;
130                pos_cnum = 368})),
131             TipoInt),
132             TipoInt),
133             (Tast.ExpInt (0, TipoInt), TipoInt)),
134       [CmdPrint (Tast.ExpString ("Zero", TipoString))], Some [...]];
135     ...]);
136     ...]];
137     ...],
138     ...),
139     ...)

```

Saída do interpretador:

Listagem 6.69: Saída do interpretador para o programa micro11

```

1 Digite um numero: 50
2 Numero positivo
3 - : unit = ()

```

6.3 Testes de Erros Semânticos

Alguns erros semânticos que podem ocorrer são exibidos a seguir.

6.3.1 Variável declarada duas vezes com tipos diferentes

Listagem 6.70: Variável declarada duas vezes com tipos diferentes

```

1 function int main()
2
3   int x
4   float x
5
6 end
7
8 main()

```

Saída do analisador:

Listagem 6.71: Variável declarada duas vezes com tipos diferentes

```

1 Exception: Tabsimb.Entrada_existente "x".

```

6.3.2 Atribuição de um inteiro com um valor float

Listagem 6.72: Atribuição de um inteiro com um valor float

```

1 function int main()
2
3     int x
4     float y
5
6     x = y + 1.
7
8 end
9
10 main()

```

Saída do analisador:

Listagem 6.73: Atribuição de um inteiro com um valor float

```

1 Exception:
2 Failure
3 "Semantico -> linha 6, coluna 2: Atribuicao com tipos diferentes: inteiro
   = float".

```

6.3.3 Operação envolvendo tipos diferentes

Listagem 6.74: Operação envolvendo tipos diferentes

```

1 function int main()
2
3     int x
4     float y
5
6     x = 10
7     y = 20.0
8
9     if x==y then
10         print("Igual")
11     else
12         print("Diferente")
13     end
14
15 end
16
17 main()

```

Saída do analisador:

Listagem 6.75: Operação envolvendo tipos diferentes

```

1 Exception:
2 Failure
3 "Semantico -> linha 9, coluna 6: O operando esquerdo eh do tipo inteiro
   mas o direito eh do tipo float".

```

6.3.4 Chamada de função com número incorreto de parâmetros

Listagem 6.76: Chamada de função com número incorreto de parâmetros

```

1 int a, b
2
3 function int main(int z)
4
5     int x
6     float y
7
8     x = 10
9     y = 20.0
10
11 end
12
13 main(a,b)

```

Saída do analisador:

Listagem 6.77: Chamada de função com número incorreto de parâmetros

```

1 # interprete "Programas-Lua/erro5.lua";;
2 Exception:
3 Failure "Semantico -> linha 13, coluna -1: Numero incorreto de parametros"
.

```

6.3.5 Declaração de parâmetros iguais de uma função

Listagem 6.78: Declaração de parâmetros iguais de uma função

```

1 function int main(int x, int x)
2
3     int x
4     float y
5
6     x = 10
7     y = 20.0
8
9 end
10
11 main()

```

Saída do analisador:

Listagem 6.79: Declaração de parâmetros iguais de uma função

```

1 Exception: Failure "Semantico -> linha 1, coluna 21: Parametro duplicado x"
.

```

6.3.6 Tipo de retorno diferente do tipo declarado da função

Listagem 6.80: Tipo de retorno diferente do tipo declarado da função

```

1 function int main()
2
3     int x
4     float y

```

```
5
6     x = 10
7     y = 20.0
8
9     return y
10
11 end
12
13 main()
```

Saída do analisador:

Listagem 6.81: Tipo de retorno diferente do tipo declarado da função

```
1 Exception:
2 Failure
3 "Semantico -> linha 9, coluna 9: O tipo retornado eh float mas foi
   declarado como inteiro".
```

Apêndice A

Códigos Finais

A.1 Código "ambiente.ml":

Listagem A.1: ambiente.ml

```
1 module Tab = Tabsimb
2 module A = Ast
3
4 type entrada_fn = { tipo_fn: A.tipo;
5                     formais: (string * A.tipo) list;
6 }
7
8 type entrada = EntFun of entrada_fn
9               | EntVar of A.tipo
10
11 type t = {
12   ambv : entrada Tab.tabela
13 }
14
15 let novo_amb xs = { ambv = Tab.cria xs }
16
17 let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }
18
19 let busca amb ch = Tab.busca amb.ambv ch
20
21 let insere_local amb ch t =
22   Tab.insere amb.ambv ch (EntVar t)
23
24 let insere_param amb ch t =
25   Tab.insere amb.ambv ch (EntVar t)
26
27 let insere_fun amb nome params resultado =
28   let ef = EntFun { tipo_fn = resultado;
29                     formais = params }
30   in Tab.insere amb.ambv nome ef
```

A.2 Código "ambiente.mli":

Listagem A.2: ambiente.mli

```

1 type entrada_fn = { tipo_fn: Ast.tipo;
2                     formais: (string * Ast.tipo) list;
3                     (*      locais: (string * Asabs.tipo) list *)
4 }
5
6 type entrada = EntFun of entrada_fn
7               | EntVar of Ast.tipo
8
9 type t
10
11 val novo_amb : (string * entrada) list -> t
12 val novo_escopo : t -> t
13 val busca : t -> string -> entrada
14 val insere_local : t -> string -> Ast.tipo -> unit
15 val insere_param : t -> string -> Ast.tipo -> unit
16 val insere_fun : t -> string -> (string * Ast.tipo) list -> Ast.tipo ->
    unit

```

A.3 Código "ambInterp.ml":

Listagem A.3: ambInterp.ml

```

1 module Tab = Tabsimb
2 module A = Ast
3 module T = Tast
4
5 type entrada_fn = {
6   tipo_fn: A.tipo;
7   formais: (A.ident * A.tipo) list;
8   locais: A.declaracoes;
9   corpo: T.expressao A.comandos
10 }
11
12 type entrada = EntFun of entrada_fn
13               | EntVar of A.tipo * (T.expressao option)
14
15 type t = {
16   ambv : entrada Tab.tabela
17 }
18
19 let novo_amb xs = { ambv = Tab.cria xs }
20
21 let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }
22
23 let busca amb ch = Tab.busca amb.ambv ch
24
25 let atualiza_var amb ch t v =
26   Tab.atualiza amb.ambv ch (EntVar (t,v))
27
28 let insere_local amb nome t v =
29   Tab.insere amb.ambv nome (EntVar (t,v))
30
31 let insere_param amb nome t v =
32   Tab.insere amb.ambv nome (EntVar (t,v))
33
34 let insere_fun amb nome params locais resultado corpo =
35   let ef = EntFun { tipo_fn = resultado;

```

```

36             formais = params;
37             locais = locais;
38             corpo = corpo }
39   in Tab.insere amb.ambv nome ef

```

A.4 Código "ambInterp.mli":

Listagem A.4: ambInterp.mli

```

1  type entrada_fn = {
2    tipo_fn: Ast.tipo;
3    formais: (string * Ast.tipo) list;
4    locais: Ast.declaracoes;
5    corpo: Tact.expressao Ast.comandos
6  }
7
8  type entrada =
9    | EntFun of entrada_fn
10   | EntVar of Ast.tipo * (Tact.expressao option)
11
12  type t
13
14  val novo_amb : (string * entrada) list -> t
15  val novo_escopo : t -> t
16  val busca : t -> string -> entrada
17  val atualiza_var : t -> string -> Ast.tipo -> (Tact.expressao option)
18   -> unit
19  val insere_local : t -> string -> Ast.tipo -> (Tact.expressao option)
20   -> unit
21  val insere_param : t -> string -> Ast.tipo -> (Tact.expressao option) ->
22   unit
23  val insere_fun : t ->
24   string ->
25   (string * Ast.tipo) list ->
26   Ast.declaracoes ->
27   Ast.tipo -> (Tact.expressao Ast.comandos) -> unit

```

A.5 Código "ast.ml":

Listagem A.5: ast.ml

```

1  (* The type of the abstract syntax tree (AST). *)
2  open Lexing
3
4  type ident = string
5  type 'a pos = 'a * Lexing.position (* tipo e posição no arquivo fonte *)
6
7  type 'expr programa = Programa of declaracoes * ('expr funcoes) * ('expr
8   comandos)
9  and declaracoes = declaracao list
10 and 'expr funcoes = ('expr funcao) list
11 and 'expr comandos = ('expr comando) list
12
13 and declaracao = DecVar of (ident pos) * tipo

```

```

14 and 'expr funcao = DecFun of ('expr decfn)
15
16 and 'expr decfn = {
17     fn_nome:      ident pos;
18     fn_tiporet: tipo;
19     fn_formais: (ident pos * tipo) list;
20     fn_locais:  declaracoes;
21     fn_corpo:   'expr comandos
22 }
23
24 and tipo = TipoInt
25           | TipoString
26           | TipoBool
27           | TipoFloat
28           | TipoVoid
29
30 and campos = campo list
31 and campo = ident pos * tipo
32
33 and 'expr comando =
34     | CmdAtrib of 'expr * 'expr
35     | CmdIf of 'expr * ('expr comandos) * ('expr comandos option)
36     | CmdRetorno of 'expr option
37     | CmdChamada of 'expr
38     | CmdPrint of 'expr
39     | CmdScanInt of 'expr
40     | CmdScanFloat of 'expr
41     | CmdScanString of 'expr
42     | CmdWhile of 'expr * ('expr comandos)
43
44 and 'expr variaveis = ('expr variavel) list
45 and 'expr variavel =
46     | VarSimples of ident pos
47
48 and 'expr expressoes = 'expr list
49
50 and oper = Or
51           | And
52           | Maior
53           | Menor
54           | Maior_ou_Igual
55           | Menor_ou_Igual
56           | Equivalente
57           | Nao_Equivalente
58           | Or_Binario
59           | Or_Binario_Exclusivo
60           | And_Binario
61           | Mult_Por_2
62           | Div_Por_2
63           | Concatena
64           | Adicao
65           | Subtracao
66           | Multiplicacao
67           | Divisao
68           | Divisao_Inteiro
69           | Modulo
70           | Not
71           | Exponenciacao

```

A.6 Código "interprete.ml":

Listagem A.6: interprete.ml

```

1 module Amb = AmbInterp
2 module A = Ast
3 module S = Sast
4 module T = Tast
5
6 exception Valor_de_retorno of T.expressao
7
8 let obtem_nome_tipo_var exp = let open T in
9   match exp with
10  | ExpVar (v, tipo) ->
11    (match v with
12     | A.VarSimples (nome, _) -> (nome, tipo)
13     )
14  | _ -> failwith "obtem_nome_tipo_var: nao eh variavel"
15
16 let pega_int exp =
17   match exp with
18   | T.ExpInt (i, _) -> i
19   | _ -> failwith "pega_int: nao eh inteiro"
20
21 let pega_float exp =
22   match exp with
23   | T.ExpFloat (f, _) -> f
24   | _ -> failwith "pega_float: nao eh float"
25
26 let pega_string exp =
27   match exp with
28   | T.ExpString (s, _) -> s
29   | _ -> failwith "pega_string: nao eh string"
30
31 let pega_bool exp =
32   match exp with
33   | T.ExpBool (b, _) -> b
34   | _ -> failwith "pega_bool: nao eh booleano"
35
36 type classe_op = Aritmetico | Relacional | Logico | Cadeia
37
38 let classifica op =
39   let open A in
40   match op with
41   | Or
42   | And
43   | Or_Binario
44   | Or_Binario_Exclusivo
45   | And_Binario
46   | Not -> Logico
47   | Menor
48   | Maior
49   | Menor_ou_Igual
50   | Maior_ou_Igual
51   | Equivalente
52   | Nao_Equivalente -> Relacional
53   | Adicao
54   | Subtracao
55   | Multiplicacao

```

```

56 | Divisao
57 | Mult_Por_2
58 | Div_Por_2
59 | Divisao_Inteiro
60 | Modulo
61 | Exponenciacao -> Aritmetico
62 | Concatena -> Cadeia
63
64
65 let rec interpreta_exp amb exp =
66   let open A in
67   let open T in
68   match exp with
69   | ExpVoid
70   | ExpInt _
71   | ExpFloat _
72   | ExpString _
73   | ExpBool _ -> exp
74   | ExpVar _ ->
75     let (id,tipo) = obtem_nome_tipo_var exp in
76     (* Tenta encontrar o valor da variável no escopo local, se não *)
77     (* encontrar, tenta novamente no escopo que engloba o atual. Prossegue *)
78     (* assim até encontrar o valor em algum escopo englobante ou até *)
79     (* encontrar o escopo global. Se em algum lugar for encontrado, *)
80     (* devolve-se o valor. Em caso contrário, devolve uma exceção *)
81     (match (Amb.busca amb id) with
82      | Amb.EntVar (tipo, v) ->
83        (match v with
84         | None -> failwith ("variável nao inicializada: " ^ id)
85         | Some valor -> valor
86        )
87      | _ -> failwith "interpreta_exp: expvar"
88     )
89 | ExpOp ((op,top), (esq, tesq), (dir,tdir)) ->
90   let vesq = interpreta_exp amb esq
91   and vdir = interpreta_exp amb dir in
92
93   let interpreta_aritmetico () =
94     (match top with
95      | TipoInt ->
96        (match op with
97         | Adicao -> ExpInt (pega_int vesq + pega_int vdir, top)
98         | Subtracao -> ExpInt (pega_int vesq - pega_int vdir, top)
99         | Multiplicacao -> ExpInt (pega_int vesq * pega_int vdir,
100          top)
101         | Divisao -> ExpInt (pega_int vesq / pega_int vdir, top)
102         | Modulo -> ExpInt (pega_int vesq mod pega_int vdir, top)
103         | _ -> failwith "interpreta_aritmetico"
104        )
105      | TipoFloat ->
106        (match op with
107         | Adicao -> ExpFloat (pega_float vesq +. pega_float vdir,
108          top)
109         | Subtracao -> ExpFloat (pega_float vesq -. pega_float vdir, top)
110         | Multiplicacao -> ExpFloat (pega_float vesq *. pega_float
111          vdir, top)

```

A.6

```

109         | Divisao -> ExpFloat (pega_float vesq /. pega_float vdir,
110                               top)
111     | _ -> failwith "interpreta_aritmetico"
112 )
113 | _ -> failwith "interpreta_aritmetico"
114 )
115 and interpreta_relacional () =
116   (match tesq with
117   | TipoInt ->
118     (match op with
119     | Menor -> ExpBool (pega_int vesq < pega_int vdir, top)
120     | Maior -> ExpBool (pega_int vesq > pega_int vdir, top)
121     | Equivalente -> ExpBool (pega_int vesq == pega_int vdir, top)
122     | Nao_Equivalente -> ExpBool (pega_int vesq != pega_int vdir,
123                                   top)
124     | Menor_ou_Igual -> ExpBool (pega_int vesq <= pega_int vdir,
125                                   top)
126     | Maior_ou_Igual -> ExpBool (pega_int vesq >= pega_int vdir,
127                                   top)
128     | _ -> failwith "interpreta_relacional"
129     )
130   | TipoFloat ->
131     (match op with
132     | Menor -> ExpBool (pega_float vesq < pega_float vdir, top)
133     | Maior -> ExpBool (pega_float vesq > pega_float vdir, top)
134     | Equivalente -> ExpBool (pega_float vesq == pega_float vdir,
135                                   top)
136     | Nao_Equivalente -> ExpBool (pega_float vesq != pega_float
137                                   vdir, top)
138     | Menor_ou_Igual -> ExpBool (pega_float vesq <= pega_float
139                                   vdir, top)
140     | Maior_ou_Igual -> ExpBool (pega_float vesq >= pega_float
141                                   vdir, top)
142     | _ -> failwith "interpreta_relacional"
143     )
144   | TipoString ->
145     (match op with
146     | Menor -> ExpBool (pega_string vesq < pega_string vdir, top)
147     | Maior -> ExpBool (pega_string vesq > pega_string vdir, top)
148     | Equivalente -> ExpBool (pega_string vesq == pega_string vdir
149                                   , top)
150     | Nao_Equivalente -> ExpBool (pega_string vesq != pega_string
151                                   vdir, top)
152     | Menor_ou_Igual -> ExpBool (pega_string vesq <= pega_string
153                                   vdir, top)
154     | Maior_ou_Igual -> ExpBool (pega_string vesq >= pega_string
155                                   vdir, top)
156     | _ -> failwith "interpreta_relacional"
157     )
158   | TipoBool ->
159     (match op with
160     | Menor -> ExpBool (pega_bool vesq < pega_bool vdir, top)
161     | Maior -> ExpBool (pega_bool vesq > pega_bool vdir, top)
162     | Equivalente -> ExpBool (pega_bool vesq == pega_bool vdir,
163                                   top)
164     | Nao_Equivalente -> ExpBool (pega_bool vesq != pega_bool vdir
165                                   , top)
166     | Menor_ou_Igual -> ExpBool (pega_bool vesq <= pega_bool vdir
167                                   , top)
168     | Maior_ou_Igual -> ExpBool (pega_bool vesq >= pega_bool vdir
169                                   , top)
170     | _ -> failwith "interpreta_relacional"
171     )
172   )

```

```

        , top)
154      | Maior_ou_Igual    -> ExpBool (pega_bool vesq >= pega_bool vdir
        , top)
155      | _ -> failwith "interpreta_relacional"
156    )
157  | _ -> failwith "interpreta_relacional"
158 )
159
160 and interpreta_logico () =
161   (match tesq with
162   | TipoBool ->
163     (match op with
164     | Or -> ExpBool (pega_bool vesq || pega_bool vdir, top)
165     | And -> ExpBool (pega_bool vesq && pega_bool vdir, top)
166     | _ -> failwith "interpreta_logico"
167     )
168   | _ -> failwith "interpreta_logico"
169   )
170 and interpreta_cadeia () =
171   (match tesq with
172   | TipoString ->
173     (match op with
174     | Concatena -> ExpString (pega_string vesq ^ pega_string vdir,
175                               top)
176     | _ -> failwith "interpreta_cadeia"
177     )
178   | _ -> failwith "interpreta_cadeia"
179   )
180 in
181 let valor = (match (classifica op) with
182               Aritmetico -> interpreta_aritmetico ()
183               | Relacional -> interpreta_relacional ()
184               | Logico -> interpreta_logico ()
185               | Cadeia -> interpreta_cadeia ()
186             )
187 in
188   valor
189
190 | ExpChamada (id, args, tipo) ->
191   let open Amb in
192   ( match (Amb.busca amb id) with
193   | Amb.EntFun {tipo_fn; formais; locais; corpo} ->
194     (* Interpreta cada um dos argumentos *)
195     let vargs = List.map (interpreta_exp amb) args in
196     (* Associa os argumentos aos parâmetros formais *)
197     let vformais = List.map2 (fun (n,t) v -> (n, t, Some v))
198                             formais vargs
199     in interpreta_fun amb id vformais locais corpo
200   | _ -> failwith "interpreta_exp: expchamada"
201   )
202
203 and interpreta_fun amb fn_nome fn_formais fn_locais fn_corpo =
204   let open A in
205   (* Estende o ambiente global, adicionando um ambiente local *)
206   let ambfn = Amb.novo_escopo amb in
207   let insere_local d =
208     match d with
209     (DecVar (v,t)) -> Amb.insere_local ambfn (fst v) t None

```

```

209 in
210 (* Associa os argumentos aos parâmetros e insere no novo ambiente *)
211 let insere_parametro (n,t,v) = Amb.insere_param ambfn n t v in
212 let _ = List.iter insere_parametro fn_formais in
213 (* Insere as variáveis locais no novo ambiente *)
214 let _ = List.iter insere_local fn_locais in
215 (* Interpreta cada comando presente no corpo da função usando o novo
216 ambiente *)
217 try
218 let _ = List.iter (interpreta_cmd ambfn) fn_corpo in T.ExpVoid
219 with
220 Valor_de_retorno expret -> expret
221
222 and interpreta_cmd amb cmd =
223 let open A in
224 let open T in
225 match cmd with
226 CmdRetorno exp ->
227 (* Levantar uma exceção foi necessária pois, pela semântica do comando
228 de
229 retorno, sempre que ele for encontrado em uma função, a computação
230 deve parar retornando o valor indicado, sem realizar os demais
231 comandos.
232 *)
233 (match exp with
234 (* Se a função não retornar nada, então retorne ExpVoid *)
235 None -> raise (Valor_de_retorno ExpVoid)
236 | Some e ->
237 (* Avalia a expressão e retorne o resultado *)
238 let e1 = interpreta_exp amb e in
239 raise (Valor_de_retorno e1)
240 )
241 | CmdIf (teste, entao, senao) ->
242 let testel = interpreta_exp amb teste in
243 (match testel with
244 ExpBool (true,_) ->
245 (* Interpreta cada comando do bloco 'então' *)
246 List.iter (interpreta_cmd amb) entao
247 | _ ->
248 (* Interpreta cada comando do bloco 'senão', se houver *)
249 (match senao with
250 None -> ()
251 | Some bloco -> List.iter (interpreta_cmd amb) bloco
252 )
253 )
254 | CmdAtrib (elem, exp) ->
255 (* Interpreta o lado direito da atribuição *)
256 let exp = interpreta_exp amb exp
257 (* Faz o mesmo para o lado esquerdo *)
258 and (elem1, tipo) = obter_nome_tipo_var elem in
259 Amb.atualiza_var amb elem1 tipo (Some exp)
260
261 | CmdChamada exp -> ignore( interpreta_exp amb exp)
262
263 | CmdScanInt exp
264 | CmdScanFloat exp
265 | CmdScanString exp ->

```

```

266 (* Obtem os nomes e os tipos de cada um dos argumentos *)
267 let nt = obtem_nome_tipo_var exp in
268 let leia_var (nome,tipo) =
269     let _ =
270         (try
271             begin
272                 match (Amb.busca amb nome) with
273                 | Amb.EntVar (_,_) -> ()
274                 | Amb.EntFun _ -> failwith "falha no input"
275             end
276         with Not_found ->
277             let _ = Amb.insere_local amb nome tipo None in ()
278         )
279     in
280     let valor =
281         (match tipo with
282         | TipoInt -> T.ExpInt (read_int(), tipo)
283         | TipoFloat -> T.ExpFloat (read_float(), tipo)
284         | TipoString -> T.ExpString (read_line (), tipo)
285         | _ -> failwith "Fail input")
286     in Amb.atualiza_var amb nome tipo (Some valor)
287 in leia_var nt
288
289 | CmdPrint exp ->
290     let resp = interpreta_exp amb exp in
291     (match resp with
292     | T.ExpInt (n,_) -> print_int n
293     | T.ExpFloat (n,_) -> print_float n
294     | T.ExpString (n,_) -> print_string n
295     | _ -> failwith "Fail print"
296     )
297
298 | CmdWhile (cond, cmds) ->
299     let rec laco cond cmds =
300         let condResp = interpreta_exp amb cond in
301         (match condResp with
302         | ExpBool (true,_) ->
303             (* Interpreta cada comando do bloco 'então' *)
304             let _ = List.iter (interpreta_cmd amb) cmds in
305             laco cond cmds
306         | _ -> ())
307     in laco cond cmds
308
309 let insere_declaracao_var amb dec =
310     match dec with
311     A.DecVar (nome, tipo) -> Amb.insere_local amb (fst nome) tipo
312     None
313
314 let insere_declaracao_fun amb dec =
315     let open A in
316     match dec with
317     DecFun {fn_nome; fn_tiporet; fn_formais; fn_locais; fn_corpo} ->
318         let nome = fst fn_nome in
319         let formais = List.map (fun (n,t) -> ((fst n), t)) fn_formais in
320         Amb.insere_fun amb nome formais fn_locais fn_tiporet fn_corpo
321
322 (* Lista de cabeçalhos das funções pré definidas *)
323 let fn_predefs = let open A in [

```

```

324     ("entrada", [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
325     ("saida",      [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
326 ]
327
328 (* insere as funções pré definidas no ambiente global *)
329 let declara_predefinidas amb =
330   List.iter (fun (n,ps,tr,c) -> Amb.insere_fun amb n ps [] tr c)
             fn_predefs
331
332 let interprete ast =
333   (* cria ambiente global inicialmente vazio *)
334   let amb_global = Amb.novo_amb [] in
335   let _ = declara_predefinidas amb_global in
336   let (A.Programa (decs_globais, decs_funs, corpo)) = ast in
337   let _ = List.iter (insere_declaracao_var amb_global) decs_globais in
338   let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
339   (* Interpreta a função principal *)
340   let resultado = List.iter (interpreta_cmd amb_global) corpo in
341   resultado

```

A.7 Código "interprete.mli":

Listagem A.7: interprete.mli

```

1 val interprete : Tast.expressao Ast.programa -> unit

```

A.8 Código "lexico.mll":

Listagem A.8: lexico.mll

```

1 {
2   open Lexing
3   open Printf
4   open Sintatico
5
6   exception Erro of string
7
8   let incr_num_linha lexbuf =
9     let pos = lexbuf.lex_curr_p in
10    lexbuf.lex_curr_p <-
11      { pos with pos_lnum = pos.pos_lnum + 1;
12          pos_bol = pos.pos_cnum
13      }
14
15   let pos_atual lexbuf = lexbuf.lex_start_p
16
17 }
18
19 let digito = ['0' - '9']
20 let inteiro = '-'? digito+
21 let frac = '.'digito*
22 let real = digito* frac
23
24 let letra = ['a' - 'z' 'A' - 'Z']
25 let identificador = letra ( letra | digito | '_' ) *

```

```

26
27 let brancos = [' ' '\t']+
28 let novalinha = '\r' | '\n' | "\r\n"
29
30 let comentario = "--" [^ '\r' '\n' ]*
31
32 rule token =
33   parse
34   | brancos { token lexbuf }
35   | novalinha { incr_num_linha lexbuf; token lexbuf }
36   | comentario { token lexbuf }
37   | "/"*      { comentario_bloco 0 lexbuf }
38   | '+'       { ADICAO (pos_atual lexbuf) }
39   | '&'       { AND_BINARIO (pos_atual lexbuf) }
40   | '('       { APAR (pos_atual lexbuf) }
41   | '='       { ATRIB (pos_atual lexbuf) }
42   | "..."   { CONCATENA (pos_atual lexbuf) }
43   | ">>"      { DIV_POR_2 (pos_atual lexbuf) }
44   | "/"       { DIVISAO (pos_atual lexbuf) }
45   | "//"      { DIVISAO_INTEIRO (pos_atual lexbuf) }
46   | "=="      { EQUIVALENTE (pos_atual lexbuf) }
47   | "^"       { EXPONENCIACAO (pos_atual lexbuf) }
48   | ')'       { FPAR (pos_atual lexbuf) }
49   | '>'       { MAIOR (pos_atual lexbuf) }
50   | ">="      { MAIOR_OU_IGUAL (pos_atual lexbuf) }
51   | '<'       { MENOR (pos_atual lexbuf) }
52   | "<="      { MENOR_OU_IGUAL (pos_atual lexbuf) }
53   | '%'       { MODULO (pos_atual lexbuf) }
54   | "<<"      { MULT_POR_2 (pos_atual lexbuf) }
55   | '*'       { MULTIPLICACAO (pos_atual lexbuf) }
56   | "~="      { NAO_EQUIVALENTE (pos_atual lexbuf) }
57   | "|"       { OR_BINARIO (pos_atual lexbuf) }
58   | "~"       { OR_BINARIO_EXCLUSIVO (pos_atual lexbuf) }
59   | "-"       { SUBTRACAO (pos_atual lexbuf) }
60   | ','       { VIRGULA (pos_atual lexbuf) }
61   | '"'       { let buffer = Buffer.create 1 in
62                 let str = leia_string buffer lexbuf in
63                 LITSTRING (str, pos_atual lexbuf) }
64   | "and"     { AND (pos_atual lexbuf) }
65   | "do"      { DO (pos_atual lexbuf) }
66   | "else"    { ELSE (pos_atual lexbuf) }
67   | "float"   { TIPO_FLOAT (pos_atual lexbuf) }
68   | "for"     { FOR (pos_atual lexbuf) }
69   | "function" { FUNCAO (pos_atual lexbuf) }
70   | "if"      { IF (pos_atual lexbuf) }
71   | "io.read('n')" { IO_READ_INT (pos_atual lexbuf) }
72   | "io.read('f')" { IO_READ_FLOAT (pos_atual lexbuf) }
73   | "io.read('s')" { IO_READ_STRING (pos_atual lexbuf) }
74   | "not"     { NOT (pos_atual lexbuf) }
75   | "or"      { OR (pos_atual lexbuf) }
76   | "print"   { PRINT (pos_atual lexbuf) }
77   | "return"  { RETURN (pos_atual lexbuf) }
78   | "end"     { END (pos_atual lexbuf) }
79   | "bool"    { TIPO_BOOLEAN (pos_atual lexbuf) }
80   | "int"     { TIPO_INT (pos_atual lexbuf) }
81   | "then"    { THEN (pos_atual lexbuf) }
82   | "string"  { TIPO_STRING (pos_atual lexbuf) }
83   | "while"   { WHILE (pos_atual lexbuf) }
84   | "true"    { LITBOOL (true, pos_atual lexbuf) }

```



```

85 | "false"      { LITBOOL (false, pos_atual lexbuf) }
86 | identificador as x      { ID (x, pos_atual lexbuf) }
87 | inteiro as n      { LITINT (int_of_string n, pos_atual lexbuf) }
88 | real as n      { LITFLOAT (float_of_string n, pos_atual lexbuf) }
89 | _ { raise (Erro ("Caracter desconhecido: " ^ Lexing.lexeme lexbuf)) }
90 | eof      { EOF }
91
92 and comentario_bloco n = parse
93   "*/"      { if n=0 then token lexbuf
94               else comentario_bloco (n-1) lexbuf }
95 | "/*"      { comentario_bloco (n+1) lexbuf }
96 | _        { comentario_bloco n lexbuf }
97 | eof      { raise (Erro "Comentário não terminado") }
98
99 and leia_string buffer = parse
100   ""      { Buffer.contents buffer}
101 | "\\t"    { Buffer.add_char buffer '\t'; leia_string buffer lexbuf }
102 | "\\n"    { Buffer.add_char buffer '\n'; leia_string buffer lexbuf }
103 | '\\\'' '\\\'' { Buffer.add_char buffer '\''; leia_string buffer lexbuf }
104 | '\\\'' '\\\'' { Buffer.add_char buffer '\\\''; leia_string buffer lexbuf }
105 | _ as c   { Buffer.add_char buffer c; leia_string buffer lexbuf }
106 | eof      { raise (Erro "A string não foi terminada") }

```

A.9 Código "sast.ml":

Listagem A.9: sast.ml

```

1 open Ast
2
3 type expressao =
4   | ExpVar of (expressao variavel)
5   | ExpInt of int pos
6   | ExpFloat of float pos
7   | ExpString of string pos
8   | ExpBool of bool pos
9   | ExpOp of oper pos * expressao * expressao
10  | ExpChamada of ident pos * (expressao expressoes)

```

A.10 Código "semantico.ml":

Listagem A.10: semantico.ml

```

1 module Amb = Ambiente
2 module A = Ast
3 module S = Sast
4 module T = Tast
5
6 let rec posicao exp = let open S in
7   match exp with
8   | ExpVar v -> (match v with
9                 | A.VarSimples (_,pos) -> pos
10                )
11   | ExpInt (_,pos) -> pos
12   | ExpFloat (_,pos) -> pos
13   | ExpString (_,pos) -> pos

```

```

14 | ExpBool (_,pos) -> pos
15 | ExpOp ((_,pos),_,_) -> pos
16 | ExpChamada ((_,pos), _) -> pos
17
18 type classe_op = Aritmetico | Relacional | Logico | Cadeia
19
20 let classifica op =
21   let open A in
22   match op with
23     Or
24   | And
25   | Or_Binario
26   | Or_Binario_Exclusivo
27   | And_Binario
28   | Not -> Logico
29   | Menor
30   | Maior
31   | Menor_ou_Igual
32   | Maior_ou_Igual
33   | Equivalente
34   | Nao_Equivalente -> Relacional
35   | Adicao
36   | Subtracao
37   | Multiplicacao
38   | Divisao
39   | Mult_Por_2
40   | Div_Por_2
41   | Divisao_Inteiro
42   | Modulo
43   | Exponenciacao -> Aritmetico
44   | Concatena -> Cadeia
45
46 let msg_erro_pos pos msg =
47   let open Lexing in
48   let lin = pos.pos_lnum
49   and col = pos.pos_cnum - pos.pos_bol - 1 in
50   Printf.sprintf "Semantico -> linha %d, coluna %d: %s" lin col msg
51
52 let msg_erro nome msg =
53   let pos = snd nome in
54   msg_erro_pos pos msg
55
56 let nome_tipo t =
57   let open A in
58   match t with
59     TipoInt -> "inteiro"
60   | TipoFloat -> "float"
61   | TipoString -> "string"
62   | TipoBool -> "bool"
63   | TipoVoid -> "void"
64
65 let mesmo_tipo pos msg tinf tdec =
66   if tinf <> tdec
67   then
68     let msg = Printf.sprintf msg (nome_tipo tinf) (nome_tipo tdec) in
69     failwith (msg_erro_pos pos msg)
70
71 let rec infere_exp amb exp =
72   match exp with

```

```

73   S.ExpInt n    -> (T.ExpInt (fst n, A.TipoInt),      A.TipoInt)
74 | S.ExpFloat f -> (T.ExpFloat (fst f, A.TipoFloat),  A.TipoFloat)
75 | S.ExpString s -> (T.ExpString (fst s, A.TipoString), A.TipoString)
76 | S.ExpBool b  -> (T.ExpBool (fst b, A.TipoBool),    A.TipoBool)
77 | S.ExpVar v   ->
78   (match v with
79     A.VarSimples nome ->
80     (* Tenta encontrar a definição da variável no escopo local, se não
81       *)
82     (* encontrar tenta novamente no escopo que engloba o atual.
83       Prossegue-se *)
84     (* assim até encontrar a definição em algum escopo englobante ou at
85       é *)
86     (* encontrar o escopo global. Se em algum lugar for encontrado,
87       *)
88     (* devolve-se a definição. Em caso contrário, devolve uma exceção
89       *)
90     let id = fst nome in
91     (try (match (Amb.busca amb id) with
92       | Amb.EntVar tipo -> (T.ExpVar (A.VarSimples nome, tipo),
93         tipo)
94       | Amb.EntFun _ ->
95         let msg = "nome de funcao usado como nome de variavel: "
96           ^ id in
97         failwith (msg_erro nome msg)
98       )
99     with Not_found ->
100      let msg = "A variavel " ^ id ^ " nao foi declarada" in
101      failwith (msg_erro nome msg)
102    )
103 | _ -> failwith "infere_exp: não implementado"
104 )
105
106 | S.ExpOp (op, esq, dir) ->
107   let (esq, tesq) = infere_exp amb esq
108   and (dir, tdir) = infere_exp amb dir in
109
110   let verifica_aritmetico () =
111     (match tesq with
112     | A.TipoInt
113     | A.TipoFloat ->
114       let _ = mesmo_tipo (snd op)
115       "O operando esquerdo eh do tipo %s mas o direito eh
116         do tipo %s"
117       tesq tdir
118     in tesq (* O tipo da expressão aritmética como um todo *)
119
120   | t -> let msg = "um operador aritmetico nao pode ser usado com o
121     tipo " ^
122       (nome_tipo t)
123     in failwith (msg_erro_pos (snd op) msg)
124   )
125
126 and verifica_relacional () =
127   (match tesq with
128   | A.TipoInt
129   | A.TipoFloat
130   | A.TipoBool
131   | A.TipoString ->

```

```

123     let _ = mesmo_tipo (snd op)
124         "O operando esquerdo eh do tipo %s mas o direito eh do
125         tipo %s"
126         tesq tdir
127     in A.TipoBool (* O tipo da expressão relacional é sempre booleano
128                    *)
129
130 | t -> let msg = "um operador relacional nao pode ser usado com o
131               tipo " ^
132               (nome_tipo t)
133           in failwith (msg_erro_pos (snd op) msg)
134
135 and verifica_logico () =
136     (match tesq with
137     A.TipoBool ->
138         let _ = mesmo_tipo (snd op)
139             "O operando esquerdo eh do tipo %s mas o direito eh do
140             tipo %s"
141             tesq tdir
142         in A.TipoBool (* O tipo da expressão lógica é sempre booleano *)
143
144 | t -> let msg = "um operador logico nao pode ser usado com o tipo
145               " ^
146               (nome_tipo t)
147           in failwith (msg_erro_pos (snd op) msg)
148
149 and verifica_cadeia () =
150     (match tesq with
151     A.TipoString ->
152         let _ = mesmo_tipo (snd op)
153             "O operando esquerdo eh do tipo %s mas o direito eh do
154             tipo %s"
155             tesq tdir
156         in A.TipoString (* O tipo da expressão relacional é sempre string
157                           *)
158
159 | t -> let msg = "um operador relacional nao pode ser usado com o
160               tipo " ^
161               (nome_tipo t)
162           in failwith (msg_erro_pos (snd op) msg)
163
164 )
165
166 in
167 let op = fst op in
168 let tinf = (match (classifica op) with
169             Aritmetico -> verifica_aritmetico ()
170             | Relacional -> verifica_relacional ()
171             | Logico -> verifica_logico ()
172             | Cadeia -> verifica_cadeia ())
173         in
174         (T.ExpOp ((op,tinf), (esq, tesq), (dir, tdir)), tinf)
175
176 | S.ExpChamada (nome, args) ->
177     let rec verifica_parametros ags ps fs =
178         match (ags, ps, fs) with
179         (a::ags), (p::ps), (f::fs) ->
180             let _ = mesmo_tipo (posicao a)

```

```

174         "O parametro eh do tipo %s mas deveria ser do tipo %s
          " p f
175     in verifica_parametros ags ps fs
176 | [], [], [] -> ()
177 | _ -> failwith (msg_erro nome "Numero incorreto de parametros")
178 in
179 let id = fst nome in
180 try
181     begin
182         let open Amb in
183
184         match (Amb.busca amb id) with
185         (* verifica se 'nome' está associada a uma função *)
186         Amb.EntFun {tipo_fn; formais} ->
187             (* Infere o tipo de cada um dos argumentos *)
188             let argst = List.map (infere_exp amb) args
189             (* Obtem o tipo de cada parâmetro formal *)
190             and tipos_formais = List.map snd formais in
191             (* Verifica se o tipo de cada argumento confere com o tipo
              declarado *)
192             (* do parâmetro formal correspondente.
              *)
193             let _ = verifica_parametros args (List.map snd argst)
              tipos_formais
194             in (T.ExpChamada (id, (List.map fst argst), tipo_fn), tipo_fn)
195 | Amb.EntVar _ -> (* Se estiver associada a uma variável, falhe
              *)
196             let msg = id ^ " eh uma variavel e nao uma funcao" in
197             failwith (msg_erro nome msg)
198     end
199 with Not_found ->
200     let msg = "Nao existe a funcao de nome " ^ id in
201     failwith (msg_erro nome msg)
202
203 let rec verifica_cmd amb tiporet cmd =
204     let open A in
205     match cmd with
206     CmdRetorno exp ->
207     (match exp with
208     (* Se a função não retornar nada, verifica se ela foi declarada como
       void *)
209     None ->
210         let _ = mesmo_tipo (Lexing.dummy_pos)
211             "O tipo retornado eh %s mas foi declarado como %s"
212             TipoVoid tiporet
213         in CmdRetorno None
214 | Some e ->
215         (* Verifica se o tipo inferido para a expressão de retorno confere
       com o *)
216         (* tipo declarado para a função.
       *)
217         let (e1,tinf) = infere_exp amb e in
218         let _ = mesmo_tipo (posicao e)
219             "O tipo retornado eh %s mas foi declarado
       como %s"
220             tinf tiporet
221         in CmdRetorno (Some e1)
222     )
223 | CmdIf (teste, entao, senao) ->

```

```

224 let (testel,tinf) = infere_exp amb teste in
225 (* O tipo inferido para a expressão 'teste' do condicional deve ser
    booleano *)
226 let _ = mesmo_tipo (posicao teste)
227      "O teste do if deveria ser do tipo %s e nao %s"
228      TipoBool tinf in
229 (* Verifica a validade de cada comando do bloco 'então' *)
230 let entao1 = List.map (verifica_cmd amb tiporet) entao in
231 (* Verifica a validade de cada comando do bloco 'senão', se houver *)
232 let senao1 =
233     match senao with
234     None -> None
235     | Some bloco -> Some (List.map (verifica_cmd amb tiporet) bloco)
236 in
237 CmdIf (testel, entao1, senao1)
238
239 | CmdAtrib (elem, exp) ->
240     (* Infer o tipo da expressão no lado direito da atribuição *)
241     let (exp, tdir) = infere_exp amb exp
242     (* Faz o mesmo para o lado esquerdo *)
243     and (elem1, tesq) = infere_exp amb elem in
244     (* Os dois tipos devem ser iguais *)
245     let _ = mesmo_tipo (posicao elem)
246           "Atribuicao com tipos diferentes: %s = %s" tesq
247           tdir
248 in CmdAtrib (elem1, exp)
249
250 | CmdChamada exp ->
251     let (exp,tinf) = infere_exp amb exp in
252     CmdChamada exp
253
254 | CmdPrint exp ->
255     let expt = infere_exp amb exp in
256     CmdPrint (fst expt)
257
258 | CmdScanInt exp ->
259     (match exp with
260     ExpVar v -> (match v with
261     | A.VarSimples (id,pos) ->
262         (try
263         begin
264             (match (Amb.busca amb id) with
265             Amb.EntVar tipo ->
266                 let expt = infere_exp amb exp in
267                 let _ = mesmo_tipo pos
268                   "ScanInt com tipos diferentes: %s = %s"
269                   tipo (snd expt) in
270                 CmdScanInt (fst expt)
271             | Amb.EntFun _ ->
272                 let msg = "nome de funcao usado como nome de
273                   variavel: " ^ id in
274                 failwith (msg_erro_pos pos msg) )
275         end
276         with Not_found ->
277             let _ = Amb.insere_local amb id A.TipoInt in
278             let expt = infere_exp amb exp in
279             CmdScanInt (fst expt) )
280     | _ -> failwith "Falha ScanInt"
281     )

```

```

281     )
282
283 | CmdScanFloat exp ->
284   (match exp with
285     ExpVar v -> (match v with
286       | A.VarSimples (id,pos) ->
287         (try
288           begin
289             (match (Amb.busca amb id) with
290               Amb.EntVar tipo ->
291                 let expt = infere_exp amb exp in
292                 let _ = mesmo_tipo pos
293                   "ScanFloat com tipos diferentes: %s = %s"
294                   tipo (snd expt) in
295                 CmdScanFloat (fst expt)
296             | Amb.EntFun _ ->
297               let msg = "nome de funcao usado como nome de
298                 variavel: " ^ id in
299                 failwith (msg_erro_pos pos msg) )
300           end
301         with Not_found ->
302           let _ = Amb.insere_local amb id A.TipoFloat in
303           let expt = infere_exp amb exp in
304           CmdScanFloat (fst expt) )
305   | _ -> failwith "Falha ScanFloat"
306   )
307 )
308
309 | CmdScanString exp ->
310   (match exp with
311     ExpVar v -> (match v with
312       | A.VarSimples (id,pos) ->
313         (try
314           begin
315             (match (Amb.busca amb id) with
316               Amb.EntVar tipo ->
317                 let expt = infere_exp amb exp in
318                 let _ = mesmo_tipo pos
319                   "ScanString com tipos diferentes: %s = %s"
320                   tipo (snd expt) in
321                 CmdScanString (fst expt)
322             | Amb.EntFun _ ->
323               let msg = "nome de funcao usado como nome de
324                 variavel: " ^ id in
325                 failwith (msg_erro_pos pos msg) )
326           end
327         with Not_found ->
328           let _ = Amb.insere_local amb id A.TipoString in
329           let expt = infere_exp amb exp in
330           CmdScanString (fst expt) )
331   | _ -> failwith "Falha ScanString"
332   )
333 )
334
335
336 | CmdWhile (exp_cond, comandos) ->
337   let (expCond, expT ) = infere_exp amb exp_cond in
338   let comandos_tipados =
339     (match expT with

```

```

340     | A.TipoBool -> List.map (verifica_cmd amb tiporet) comandos
341     | _ -> let msg = "Condicao deve ser tipo Bool" in
342         failwith (msg_erro_pos (posicao exp_cond) msg))
343     in CmdWhile (expCond,comandos_tipados)
344
345
346
347 and verifica_fun amb ast =
348     let open A in
349     match ast with
350     A.DecFun {fn_nome; fn_tiporet; fn_formais; fn_locais; fn_corpo} ->
351         (* Estende o ambiente global, adicionando um ambiente local *)
352         let ambfn = Amb.novo_escopo amb in
353         (* Insere os parâmetros no novo ambiente *)
354         let insere_parametro (v,t) = Amb.insere_param ambfn (fst v) t in
355         let _ = List.iter insere_parametro fn_formais in
356         (* Insere as variáveis locais no novo ambiente *)
357         let insere_local = function
358             (DecVar (v,t)) -> Amb.insere_local ambfn (fst v) t in
359         let _ = List.iter insere_local fn_locais in
360         (* Verifica cada comando presente no corpo da função usando o novo
361             ambiente *)
362         let corpo_tipado = List.map (verifica_cmd ambfn fn_tiporet) fn_corpo
363             in
364             A.DecFun {fn_nome; fn_tiporet; fn_formais; fn_locais; fn_corpo =
365                 corpo_tipado}
366
367
368
369 let rec verifica_dup xs =
370     match xs with
371     [] -> []
372     | (nome,t)::xs ->
373         let id = fst nome in
374         if (List.for_all (fun (n,t) -> (fst n) <> id) xs)
375         then (id, t) :: verifica_dup xs
376         else let msg = "Parametro duplicado " ^ id in
377             failwith (msg_erro nome msg)
378
379
380
381 let insere_declaracao_var amb dec =
382     let open A in
383     match dec with
384     DecVar (nome, tipo) -> Amb.insere_local amb (fst nome) tipo
385
386
387
388 let insere_declaracao_fun amb dec =
389     let open A in
390     match dec with
391     DecFun {fn_nome; fn_tiporet; fn_formais; fn_corpo} ->
392         (* Verifica se não há parâmetros duplicados *)
393         let formais = verifica_dup fn_formais in
394         let nome = fst fn_nome in
395         Amb.insere_fun amb nome formais fn_tiporet
396
397
398
399 (* Lista de cabeçalhos das funções pré definidas *)
400 let fn_predefs = let open A in [
401     ("entrada", [("x", TipoInt); ("y", TipoInt)], TipoVoid);
402     ("saida",    [("x", TipoInt); ("y", TipoInt)], TipoVoid)
403 ]
404
405

```



```

396 (* insere as funções pré definidas no ambiente global *)
397 let declara_predefinidas amb =
398   List.iter (fun (n,ps,tr) -> Amb.insere_fun amb n ps tr) fn_predefs
399
400 let semantico ast =
401   (* cria ambiente global inicialmente vazio *)
402   let amb_global = Amb.novo_amb [] in
403   let _ = declara_predefinidas amb_global in
404   let (A.Programa (decs_globais, decs_funs, corpo)) = ast in
405   let _ = List.iter (insere_declaracao_var amb_global) decs_globais in
406   let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
407   (* Verificação de tipos nas funções *)
408   let decs_funs = List.map (verifica_fun amb_global) decs_funs in
409   (* Verificação de tipos na função principal*)
410   let corpo = List.map (verifica_cmd amb_global A.TipoVoid) corpo in
411   (A.Programa (decs_globais, decs_funs, corpo), amb_global)

```

A.11 Código "semantico.mli":

Listagem A.11: semantico.mli

```

1 val semantico : (Sast.expressao Ast.programa) -> Tast.expressao Ast.
   programa * Ambiente.t

```

A.12 Código "sintatico.mly":

Listagem A.12: sintatico.mly

```

1
2 %{
3 open Lexing
4 open Ast
5 open Sast
6 %}
7
8 %token <int * Lexing.position> LITINT
9 %token <float * Lexing.position> LITFLOAT
10 %token <string * Lexing.position> ID
11 %token <string * Lexing.position> LITSTRING
12 %token <bool * Lexing.position> LITBOOL
13 %token <Lexing.position> ADICAO
14 %token <Lexing.position> AND
15 %token <Lexing.position> AND_BINARIO
16 %token <Lexing.position> APAR
17 %token <Lexing.position> ATRIB
18 %token <Lexing.position> CONCATENA
19 %token <Lexing.position> DIV_POR_2
20 %token <Lexing.position> DIVISAO
21 %token <Lexing.position> DIVISAO_INTEIRO
22 %token <Lexing.position> DO
23 %token <Lexing.position> ELSE
24 %token <Lexing.position> END
25 %token <Lexing.position> EQUIVALENTE
26 %token <Lexing.position> EXPONENCIACAO
27 %token <Lexing.position> FOR

```

```

28 %token <Lexing.position> FPAR
29 %token <Lexing.position> FUNCAO
30 %token <Lexing.position> IF
31 %token <Lexing.position> IO_READ_INT
32 %token <Lexing.position> IO_READ_FLOAT
33 %token <Lexing.position> IO_READ_STRING
34 %token <Lexing.position> MAIOR
35 %token <Lexing.position> MAIOR_OU_IGUAL
36 %token <Lexing.position> MENOR
37 %token <Lexing.position> MENOR_OU_IGUAL
38 %token <Lexing.position> MODULO
39 %token <Lexing.position> MULT_POR_2
40 %token <Lexing.position> MULTIPLICACAO
41 %token <Lexing.position> NAO_EQUIVALENTE
42 %token <Lexing.position> NOT
43 %token <Lexing.position> OR
44 %token <Lexing.position> OR_BINARIO
45 %token <Lexing.position> OR_BINARIO_EXCLUSIVO
46 %token <Lexing.position> PRINT
47 %token <Lexing.position> RETURN
48 %token <Lexing.position> SUBTRACAO
49 %token <Lexing.position> TIPO_BOOLEAN
50 %token <Lexing.position> TIPO_INT
51 %token <Lexing.position> TIPO_FLOAT
52 %token <Lexing.position> TIPO_STRING
53 %token <Lexing.position> THEN
54 %token <Lexing.position> VIRGULA
55 %token <Lexing.position> WHILE
56 %token EOF
57
58 %left OR
59 %left AND
60 %left MAIOR MENOR MAIOR_OU_IGUAL MENOR_OU_IGUAL EQUIVALENTE
    NAO_EQUIVALENTE
61 %left OR_BINARIO
62 %left OR_BINARIO_EXCLUSIVO
63 %left AND_BINARIO
64 %left MULT_POR_2 DIV_POR_2
65 %left CONCATENA
66 %left ADICAO SUBTRACAO
67 %left MULTIPLICACAO DIVISAO DIVISAO_INTEIRO MODULO
68 %left NOT
69 %left EXPONENCIACAO
70
71
72 %start <Sast.expressao Ast.programa> programa
73
74 %%
75
76 programa:
77     ds = declaracao_de_variavel*
78     fs = declaracao_de_funcao*
79     cs = comando*
80     EOF { Programa (List.flatten ds, fs, cs) }
81
82
83 declaracao_de_variavel:
84     t=tipo ids=separated_nonempty_list(VIRGULA, ID) {
85         List.map (fun id -> DecVar (id,t)) ids }

```

```

86
87 declaracao_de_funcao:
88     FUNCAO tret=tipo nome=ID APAR formais = separated_list(VIRGULA,
89         parametro) FPAR
90     ds = declaracao_de_variavel*
91     cs = comando*
92     END {
93         DecFun {
94             fn_nome = nome;
95             fn_tiporet = tret ;
96             fn_formais = formais;
97             fn_locais = List.flatten ds;
98             fn_corpo = cs
99         }
100     }
101 parametro: t=tipo nome=ID { (nome, t) }
102
103 tipo: t=tipo_simples { t }
104
105
106 tipo_simples: TIPO_INT      { TipoInt      }
107                | TIPO_FLOAT { TipoFloat    }
108                | TIPO_STRING { TipoString   }
109                | TIPO_BOOLEAN { TipoBool    }
110
111
112 comando: c=comando_atribuicao { c }
113          | c=comando_if      { c }
114          | c=comando_chamada { c }
115          | c=comando_retorno { c }
116          | c=comando_print   { c }
117          | c=comando_scanInt { c }
118          | c=comando_scanFloat { c }
119          | c=comando_scanString { c }
120          | c=comando_while   { c }
121          | c=comando_for     { c }
122
123 comando_atribuicao: v=expressao ATRIB exp=expressao {
124     CmdAtrib (v, exp)
125 }
126
127 comando_if: IF teste=expressao THEN
128             entao=comando+
129             senao=option(ELSE cs=comando+ {cs})
130         END {
131             CmdIf (teste, entao, senao)
132         }
133
134 comando_chamada: exp=chamada { CmdChamada exp }
135
136 comando_retorno: RETURN exp=expressao? { CmdRetorno exp}
137
138 comando_print: PRINT APAR exp=expressao FPAR { CmdPrint exp }
139
140 comando_scanInt: exp=expressao ATRIB IO_READ_INT { CmdScanInt exp }
141
142 comando_scanFloat: exp=expressao ATRIB IO_READ_FLOAT { CmdScanFloat exp }
143

```

```

144 comando_scanString: exp=expressao ATRIB IO_READ_STRING { CmdScanString exp
    }
145
146 comando_while: WHILE exp=expressao DO cs=comando* END { CmdWhile (exp, cs)
    }
147
148 comando_for: FOR v=ID ATRIB init=expressao VIRGULA teste=expressao VIRGULA
    inc=expressao DO cs=comando* END
149 {
150     CmdIf(ExpBool (true, snd v),
151         [
152             CmdAtrib (ExpVar(VarSimples v), init) ;
153             CmdWhile (
154                 ExpOp ((Menor_ou_Igual, snd v),
155                     ExpVar (VarSimples v),
156                     teste
157                 ),
158                 List.append cs [CmdAtrib (ExpVar (VarSimples v),
159                                         ExpOp (
160                         (Adicao, snd v),
161                         ExpVar (VarSimples v),
162                         inc)
163                     )
164                 ]
165             )
166         ],
167         None
168     )
169 }
170
171 expressao:
172     | v=variavel      { ExpVar v      }
173     | i=LITINT        { ExpInt i      }
174     | f=LITFLOAT      { ExpFloat f    }
175     | s=LITSTRING     { ExpString s   }
176     | b=LITBOOL       { ExpBool b     }
177     | e1=expressao op=oper e2=expressao { ExpOp (op, e1, e2) }
178     | c = chamada    { c }
179     | APAR e=expressao FPAR { e }
180
181 chamada : nome=ID APAR args=separated_list(VIRGULA, expressao) FPAR {
182     ExpChamada (nome, args)}
183
184 %inline oper:
185     | pos = OR { (Or, pos) }
186     | pos = AND { (And, pos) }
187     | pos = MAIOR { (Maior, pos) }
188     | pos = MENOR { (Menor, pos) }
189     | pos = MAIOR_OU_IGUAL { (Maior_ou_Igual, pos) }
190     | pos = MENOR_OU_IGUAL { (Menor_ou_Igual, pos) }
191     | pos = EQUIVALENTE { (Equivalente, pos) }
192     | pos = NAO_EQUIVALENTE { (Nao_Equivalente, pos) }
193     | pos = OR_BINARIO { (Or_Binario, pos) }
194     | pos = OR_BINARIO_EXCLUSIVO { (Or_Binario_Exclusivo, pos) }
195     | pos = AND_BINARIO { (And_Binario, pos) }
196     | pos = MULT_POR_2 { (Mult_Por_2, pos) }
197     | pos = DIV_POR_2 { (Div_Por_2, pos) }
198     | pos = CONCATENA { (Concatena, pos) }
199     | pos = ADICAO { (Adicao, pos) }

```

```

200 | pos = SUBTRACAO { (Subtracao, pos) }
201 | pos = MULTIPLICACAO { (Multiplicacao, pos) }
202 | pos = DIVISAO { (Divisao, pos) }
203 | pos = DIVISAO_INTEIRO { (Divisao_Inteiro, pos) }
204 | pos = MODULO { (Modulo, pos) }
205 | pos = NOT { (Not, pos) }
206 | pos = EXPONENCIACAO { (Exponenciacao, pos) }
207
208 variavel:
209 | x=ID          { VarSimples x }

```

A.13 Código "tabsimb.ml":

Listagem A.13: tabsimb.ml

```

1
2 type 'a tabela = {
3     tbl: (string, 'a) Hashtbl.t;
4     pai: 'a tabela option;
5 }
6
7 exception Entrada_existente of string;;
8
9 let insere amb ch v =
10     if Hashtbl.mem amb.tbl ch
11     then raise (Entrada_existente ch)
12     else Hashtbl.add amb.tbl ch v
13
14 let substitui amb ch v = Hashtbl.replace amb.tbl ch v
15
16 let rec atualiza amb ch v =
17     if Hashtbl.mem amb.tbl ch
18     then Hashtbl.replace amb.tbl ch v
19     else match amb.pai with
20         None -> failwith "tabsim atualiza: chave nao encontrada"
21         | Some a -> atualiza a ch v
22
23 let rec busca amb ch =
24     try Hashtbl.find amb.tbl ch
25     with Not_found ->
26         (match amb.pai with
27             None -> raise Not_found
28             | Some a -> busca a ch)
29
30 let rec cria cvs =
31     let amb = {
32         tbl = Hashtbl.create 5;
33         pai = None
34     } in
35     let _ = List.iter (fun (c,v) -> insere amb c v) cvs
36     in amb
37
38 let novo_escopo amb_pai = {
39     tbl = Hashtbl.create 5;
40     pai = Some amb_pai
41 }

```

A.14 Código "tabsimb.mli":

Listagem A.14: tabsimb.mli

```
1
2 type 'a tabela
3
4 exception Entrada_existente of string
5
6 val insere : 'a tabela -> string -> 'a -> unit
7 val substitui : 'a tabela -> string -> 'a -> unit
8 val atualiza : 'a tabela -> string -> 'a -> unit
9 val busca : 'a tabela -> string -> 'a
10 val cria : (string * 'a) list -> 'a tabela
11
12 val novo_escopo : 'a tabela -> 'a tabela
```

A.15 Código "tast.ml":

Listagem A.15: tast.ml

```
1 open Ast
2
3 type expressao = ExpVar of (expressao variavel) * tipo
4           | ExpInt of int * tipo
5           | ExpString of string * tipo
6           | ExpVoid
7           | ExpBool of bool * tipo
8           | ExpFloat of float * tipo
9           | ExpOp of (oper * tipo) * (expressao * tipo) * (expressao *
10              tipo)
11           | ExpChamada of ident * (expressao expressoes) * tipo
```

Apêndice B

Referências Bibliográficas

1. Parrot VM Website, <http://parrot.org/>.
2. Parrot - Programming Examples, https://www.tutorialspoint.com/parrot/parrot_examples.htm.
3. PASM Github, <http://parrot.github.io/parrot-docs1/1.1.0/html/docs/book/ch09pasm.pod.html>.
4. Construção de um Compilador de MiniPython para Parrot VM usando Objective Caml, 2015, Angelo Travizan Neto.
5. MiniPascal e Parrot, 2009, Tiago Ferneda Mansueli e Vinícius Vitor dos Santos Dias.
6. MiniPascal, Java, Parrot, 2009, César Evangelista Borges Júnior e José Augusto Batista Cabral Júnior.
7. <https://www.lua.org/manual/5.3/manual.html#3.1>