

Construção de um compilador de Lua para Dalvik usando Objective Caml

Vitor Martins Basso

`vitorbasso93@gmail.com`

Faculdade de Computação
Universidade Federal de Uberlândia

12 de novembro de 2018

Lista de Listagens

2.1	enunciado do programa micro04 - Lê números e informa quais estão entre 10 e 150	19
2.2	micro04.java - Lê números e informa quais estão entre 10 e 150	19
2.3	micro04.smali - Lê números e informa quais estão entre 10 e 150	20
3.1	nano01.lua - Módulo mínimo que caracteriza um programa	22
3.2	nano01.java - Módulo mínimo que caracteriza um programa	22
3.3	nano01.smali - Módulo mínimo que caracteriza um programa	22
3.4	nano02.lua - Declaração de uma variável	23
3.5	nano02.java - Declaração de uma variável	23
3.6	nano02.smali - Declaração de uma variável	23
3.7	nano03.lua - Atribuição de um inteiro a uma variável	23
3.8	nano03.java - Atribuição de um inteiro a uma variável	23
3.9	nano03.smali - Atribuição de um inteiro a uma variável	24
3.10	nano04.lua - Atribuição de uma soma de inteiros a uma variável	24
3.11	nano04.java - Atribuição de uma soma de inteiros a uma variável	24
3.12	nano04.smali - Atribuição de uma soma de inteiros a uma variável	24
3.13	nano05.lua - Inclusão do comando de impressão	25
3.14	nano05.java - Inclusão do comando de impressão	25
3.15	nano05.smali - Inclusão do comando de impressão	25
3.16	nano06.lua - Atribuição de uma subtração de inteiros a uma variável	26
3.17	nano06.java - Atribuição de uma subtração de inteiros a uma variável	26
3.18	nano06.smali - Atribuição de uma subtração de inteiros a uma variável	26
3.19	nano07.lua - Inclusão do comando condicional	27
3.20	nano07.java - Inclusão do comando condicional	27
3.21	nano07.smali - Inclusão do comando condicional	27
3.22	nano08.lua - Inclusão do comando condicional com parte senão	28
3.23	nano08.java - Inclusão do comando condicional com parte senão	28
3.24	nano08.smali - Inclusão do comando condicional com parte senão	28
3.25	nano09.lua - Atribuição de duas operações aritméticas sobre inteiros a uma variável	29
3.26	nano09.java - Atribuição de duas operações aritméticas sobre inteiros a uma variável	29
3.27	nano09.smali - Atribuição de duas operações aritméticas sobre inteiros a uma variável	29
3.28	nano10.lua - Atribuição de duas variáveis inteiras	30
3.29	nano10.java - Atribuição de duas variáveis inteiras	30
3.30	nano10.smali - Atribuição de duas variáveis inteiras	30
3.31	nano11.lua - Introdução do comando de repetição enquanto	31
3.32	nano11.java - Introdução do comando de repetição enquanto	31
3.33	nano11.smali - Introdução do comando de repetição enquanto	31

3.34	nano12.lua - Comando condicional aninhado em um comando de repetição .	32
3.35	nano12.java - Comando condicional aninhado em um comando de repetição .	33
3.36	nano12.smali - Comando condicional aninhado em um comando de repetição	33
3.37	micro01.lua - Converte graus Celsius para Fahrenheit	34
3.38	micro01.java - Converte graus Celsius para Fahrenheit	34
3.39	micro01.smali - Converte graus Celsius para Fahrenheit	34
3.40	micro02.lua - Ler dois inteiros e decide qual é maior	36
3.41	micro02.java - Ler dois inteiros e decide qual é maior	36
3.42	micro02.smali - Ler dois inteiros e decide qual é maior	37
3.43	micro03.lua - Lê um número e verifica se ele está entre 100 e 200	39
3.44	micro03.java - Lê um número e verifica se ele está entre 100 e 200	40
3.45	micro03.smali - Lê um número e verifica se ele está entre 100 e 200	40
3.46	micro04.lua - Lê números e informa quais estão entre 10 e 150	41
3.47	micro04.java - Lê números e informa quais estão entre 10 e 150	42
3.48	micro04.smali - Lê números e informa quais estão entre 10 e 150	42
3.49	micro05.lua - Lê strings e caracteres	44
3.50	micro05.java - Lê strings e caracteres	44
3.51	micro05.smali - Lê strings e caracteres	45
3.52	micro06.lua - Escreve um número lido por extenso	48
3.53	micro06.java - Escreve um número lido por extenso	48
3.54	micro06.smali - Escreve um número lido por extenso	49
3.55	micro07.lua - Decide se os números são positivos, zeros ou negativos	51
3.56	micro07.java - Decide se os números são positivos, zeros ou negativos	51
3.57	micro07.smali - Decide se os números são positivos, zeros ou negativos	52
3.58	micro08.lua - Decide se um número é maior ou menor que 10	54
3.59	micro08.java - Decide se um número é maior ou menor que 10	54
3.60	micro08.smali - Decide se um número é maior ou menor que 10	55
3.61	micro09.lua - Cálculo de preços	57
3.62	micro09.java - Cálculo de preços	57
3.63	micro09.smali - Cálculo de preços	58
3.64	micro10.lua - Calcula o fatorial de um número	61
3.65	micro10.java - Calcula o fatorial de um número	61
3.66	micro10.smali - Calcula o fatorial de um número	62
3.67	micro11.lua - Decide se um número é positivo, zero ou negativo com auxílio de uma função	63
3.68	micro11.java - Decide se um número é positivo, zero ou negativo com auxílio de uma função	64
3.69	micro11.smali - Decide se um número é positivo, zero ou negativo com auxílio de uma função	64
4.1	lexico.mll - Gera tokens dado código da linguagem LUA	68
4.2	carregador.ml - Programa auxiliar para o analisador léxico	72
4.3	Resultado de passar o analisador léxico no programa nano01.lua	73
4.4	Resultado de passar o analisador léxico no programa nano02.lua	73
4.5	Resultado de passar o analisador léxico no programa nano03.lua	73
4.6	Resultado de passar o analisador léxico no programa nano04.lua	73
4.7	Resultado de passar o analisador léxico no programa nano05.lua	74
4.8	Resultado de passar o analisador léxico no programa nano06.lua	74
4.9	Resultado de passar o analisador léxico no programa nano07.lua	74
4.10	Resultado de passar o analisador léxico no programa nano08.lua	74

4.11	Resultado de passar o analisador léxico no programa nano09.lua	75
4.12	Resultado de passar o analisador léxico no programa nano10.lua	75
4.13	Resultado de passar o analisador léxico no programa nano11.lua	75
4.14	Resultado de passar o analisador léxico no programa nano12.lua	75
4.15	Resultado de passar o analisador léxico no programa micro01.lua	76
4.16	Resultado de passar o analisador léxico no programa micro02.lua	76
4.17	Resultado de passar o analisador léxico no programa micro03.lua	77
4.18	Resultado de passar o analisador léxico no programa micro04.lua	77
4.19	Resultado de passar o analisador léxico no programa micro05.lua	77
4.20	Resultado de passar o analisador léxico no programa micro06.lua	78
4.21	Resultado de passar o analisador léxico no programa micro07.lua	79
4.22	Resultado de passar o analisador léxico no programa micro08.lua	79
4.23	Resultado de passar o analisador léxico no programa micro09.lua	80
4.24	Resultado de passar o analisador léxico no programa micro10.lua	80
4.25	Resultado de passar o analisador léxico no programa micro11.lua	81
4.26	micro05Erro.lua - Lê strings e caracteres - com erro proposital de não fechar um comentário de bloco	81
4.27	Resultado de passar o analisador léxico no programa micro05Erro.lua	82
4.28	micro06Erro.lua - Escreve um número lido por extenso - com erro proposital de adicionar caracter inválido	82
4.29	Resultado de passar o analisador léxico no programa micro06Erro.lua	83
4.30	micro07.lua - Decide se os números são positivos, zeros ou negativos	83
4.31	Resultado de passar o analisador léxico no programa micro07Erro.lua	83
5.1	sintatico.mly - Código com a gramática do analisador sintático	84
5.2	ast.ml - Código da árvore sintática abstrata	88
5.3	sintaticoTest.ml - Código auxiliar	89
5.4	lexico.mll - Novo código para o analisador léxico	90
5.5	nano01.lua - Programa nano01 modificado	94
5.6	nano01.txt - Resultado do analisador sintático executado sobre nano01.lua modificado	94
5.7	nano02.lua - Programa nano02 modificado	94
5.8	nano02.txt - Resultado do analisador sintático executado sobre nano02.lua modificado	94
5.9	nano03.lua - Programa nano03 modificado	95
5.10	nano03.txt - Resultado do analisador sintático executado sobre nano03.lua modificado	95
5.11	nano04.lua - Programa nano04 modificado	95
5.12	nano04.txt - Resultado do analisador sintático executado sobre nano04.lua modificado	95
5.13	nano05.lua - Programa nano05 modificado	95
5.14	nano05.txt - Resultado do analisador sintático executado sobre nano05.lua modificado	95
5.15	nano06.lua - Programa nano06 modificado	96
5.16	nano06.txt - Resultado do analisador sintático executado sobre nano06.lua modificado	96
5.17	nano07.lua - Programa nano07 modificado	96
5.18	nano07.txt - Resultado do analisador sintático executado sobre nano07.lua modificado	96
5.19	nano08.lua - Programa nano08 modificado	96

5.20	nano08.txt - Resultado do analisador sintático executado sobre nano08.lua modificado	97
5.21	nano09.lua - Programa nano09 modificado	97
5.22	nano09.txt - Resultado do analisador sintático executado sobre nano09.lua modificado	97
5.23	nano10.lua - Programa nano10 modificado	97
5.24	nano10.txt - Resultado do analisador sintático executado sobre nano10.lua modificado	98
5.25	nano11.lua - Programa nano11 modificado	98
5.26	nano11.txt - Resultado do analisador sintático executado sobre nano11.lua modificado	98
5.27	nano12.lua - Programa nano12 modificado	99
5.28	nano12.txt - Resultado do analisador sintático executado sobre nano12.lua modificado	99
5.29	micro01.lua - Programa micro01 modificado	100
5.30	micro01.txt - Resultado do analisador sintático executado sobre micro01.lua modificado	100
5.31	micro02.lua - Programa micro02 modificado	100
5.32	micro02.txt - Resultado do analisador sintático executado sobre micro02.lua modificado	101
5.33	micro03.lua - Programa micro03 modificado	101
5.34	micro03.txt - Resultado do analisador sintático executado sobre micro03.lua modificado	102
5.35	micro04.lua - Programa micro04 modificado	102
5.36	micro04.txt - Resultado do analisador sintático executado sobre micro04.lua modificado	102
5.37	micro05.lua - Programa micro05 modificado	103
5.38	micro05.txt - Resultado do analisador sintático executado sobre micro05.lua modificado	103
5.39	micro06.lua - Programa micro06 modificado	104
5.40	micro06.txt - Resultado do analisador sintático executado sobre micro06.lua modificado	105
5.41	micro07.lua - Programa micro07 modificado	105
5.42	micro07.txt - Resultado do analisador sintático executado sobre micro07.lua modificado	106
5.43	micro08.lua - Programa micro08 modificado	106
5.44	micro08.txt - Resultado do analisador sintático executado sobre micro08.lua modificado	106
5.45	micro09.lua - Programa micro09 modificado	107
5.46	micro09.txt - Resultado do analisador sintático executado sobre micro09.lua modificado	107
5.47	micro10.lua - Programa micro10 modificado	108
5.48	micro10.txt - Resultado do analisador sintático executado sobre micro10.lua modificado	109
5.49	micro11.lua - Programa micro11 modificado	109
5.50	micro11.txt - Resultado do analisador sintático executado sobre micro11.lua modificado	110
5.51	teste01.lua - Programa de teste 01	111

5.52	saida01.txt - Resultado do analisador sintático executado sobre teste01.lua modificado	111
5.53	teste02.lua - Programa de teste 02	111
5.54	saida02.txt - Resultado do analisador sintático executado sobre teste02.lua modificado	111
5.55	teste03.lua - Programa de teste 03	111
5.56	saida03.txt - Resultado do analisador sintático executado sobre teste03.lua modificado	111
5.57	teste04.lua - Programa de teste 04	111
5.58	saida04.txt - Resultado do analisador sintático executado sobre teste01.lua modificado	112
5.59	teste05.lua - Programa de teste 05	112
5.60	saida05.txt - Resultado do analisador sintático executado sobre teste05.lua modificado	112
5.61	teste06.lua - Programa de teste 06	112
5.62	saida06.txt - Resultado do analisador sintático executado sobre teste06.lua modificado	113
5.63	teste07.lua - Programa de teste 07	113
5.64	saida07.txt - Resultado do analisador sintático executado sobre teste07.lua modificado	113
5.65	teste08.lua - Programa de teste 08	113
5.66	saida08.txt - Resultado do analisador sintático executado sobre teste08.lua modificado	113
6.1	semantico.ml - Código do analisador semântico	114
6.2	semantico.mli - Código do analisador semântico	122
6.3	ambiente.ml - Código do arquivo auxiliar de ambiente	122
6.4	ambiente.mli - Código do arquivo auxiliar de ambiente	122
6.5	ast.ml - Código da ast	123
6.6	lexico.mll - Código do analisador lexico	124
6.7	sintatico.mly - Código do analisador sintático	126
6.8	semantico.mli - Código do analisador semântico	130
6.9	semanticoTest.ml - Código do arquivo que permite testar o analisador semântico	131
6.10	sast.ml - Código do sast - arquivo auxiliar	135
6.11	tast.ml - Código do tast - arquivo auxiliar	135
6.12	tabsimb.ml - Código do tabsimb - arquivo auxiliar	136
6.13	tabsimb.mli - Código do tabsimb - arquivo auxiliar	137
6.14	nano01.lua - Programa nano01 modificado	138
6.15	nano01.txt - Resultado do analisador semântico executado sobre nano01.lua modificado	138
6.16	nano02.lua - Programa nano01 modificado	138
6.17	nano02.txt - Resultado do analisador semântico executado sobre nano02.lua modificado	139
6.18	nano03.lua - Programa nano03 modificado	139
6.19	nano03.txt - Resultado do analisador semântico executado sobre nano03.lua modificado	139
6.20	nano04.lua - Programa nano04 modificado	140
6.21	nano04.txt - Resultado do analisador semântico executado sobre nano04.lua modificado	140
6.22	nano05.lua - Programa nano05 modificado	141

6.23	nano05.txt - Resultado do analisador semântico executado sobre nano05.lua modificado	141
6.24	nano06.lua - Programa nano01 modificado	142
6.25	nano06.txt - Resultado do analisador semântico executado sobre nano06.lua modificado	142
6.26	nano07.lua - Programa nano07 modificado	143
6.27	nano07.txt - Resultado do analisador semântico executado sobre nano07.lua modificado	143
6.28	nano08.lua - Programa nano08 modificado	144
6.29	nano08.txt - Resultado do analisador semântico executado sobre nano08.lua modificado	144
6.30	nano09.lua - Programa nano01 modificado	145
6.31	nano09.txt - Resultado do analisador semântico executado sobre nano09.lua modificado	145
6.32	nano10.lua - Programa nano01 modificado	147
6.33	nano10.txt - Resultado do analisador semântico executado sobre nano10.lua modificado	147
6.34	nano11.lua - Programa nano11 modificado	148
6.35	nano11.txt - Resultado do analisador semântico executado sobre nano11.lua modificado	148
6.36	nano12.lua - Programa nano12 modificado	150
6.37	nano12.txt - Resultado do analisador semântico executado sobre nano12.lua modificado	151
6.38	micro01.lua - Programa micro01 modificado	152
6.39	micro01.txt - Resultado do analisador semântico executado sobre micro01.lua modificado	153
6.40	micro02.lua - Programa micro02 modificado	154
6.41	micro02.txt - Resultado do analisador semântico executado sobre micro02.lua modificado	154
6.42	micro03.lua - Programa micro03 modificado	156
6.43	micro03.txt - Resultado do analisador semântico executado sobre micro03.lua modificado	157
6.44	micro04.lua - Programa micro04 modificado	158
6.45	micro04.txt - Resultado do analisador semântico executado sobre micro04.lua modificado	158
6.46	micro05.lua - Programa micro05 modificado	160
6.47	micro05.txt - Resultado do analisador semântico executado sobre micro05.lua modificado	161
6.48	micro06.lua - Programa micro06 modificado	164
6.49	micro06.txt - Resultado do analisador semântico executado sobre micro06.lua modificado	164
6.50	micro07.lua - Programa micro07 modificado	166
6.51	micro07.txt - Resultado do analisador semântico executado sobre micro07.lua modificado	166
6.52	micro08.lua - Programa micro08 modificado	168
6.53	micro08.txt - Resultado do analisador semântico executado sobre micro08.lua modificado	169
6.54	micro09.lua - Programa micro09 modificado	170

6.55	micro09.txt - Resultado do analisador semântico executado sobre micro09.lua modificado	171
6.56	micro10.lua - Programa micro10 modificado	173
6.57	micro10.txt - Resultado do analisador semântico executado sobre micro10.lua modificado	174
6.58	micro11.lua - Programa micro11 modificado	176
6.59	micro11.txt - Resultado do analisador semântico executado sobre micro11.lua modificado	177
6.60	teste01.lua - Programa de teste 01	179
6.61	teste01.txt - Resultado do analisador semântico executado sobre teste01.lua modificado	179
6.62	teste02.lua - Programa de teste 02	180
6.63	teste02.txt - Resultado do analisador semântico executado sobre teste02.lua modificado	180
6.64	teste03.lua - Programa de teste 03	180
6.65	teste03.txt - Resultado do analisador semântico executado sobre teste03.lua modificado	180
6.66	teste04.lua - Programa de teste 04	181
6.67	teste04.txt - Resultado do analisador semântico executado sobre teste04.lua modificado	181
6.68	teste05.lua - Programa de teste 05	181
6.69	teste05.txt - Resultado do analisador semântico executado sobre teste05.lua modificado	181
6.70	teste06.lua - Programa de teste 06	181
6.71	teste06.txt - Resultado do analisador semântico executado sobre teste06.lua modificado	182
7.1	interprete.ml - Arquivo com o código do interpretador	183
7.2	interprete.mli - Arquivo com o código do interpretador	189
7.3	interpreteTeste.ml - Arquivo com o código para testar o adaptador interpretador	189
7.4	ambInterp.ml - Arquivo com o código do ambiente do interpretador	193
7.5	ambInterp.mli - Arquivo com o código do ambiente do interpretador	194

Sumário

1	Introdução	11
1.1	Sistema Operacional	11
1.2	Lua	11
1.2.1	Configurando Lua	12
1.3	Java JDK	12
1.3.1	Configurando a Java JDK	12
1.4	Dalvik	12
1.4.1	Configurando a Dalvik	13
1.5	OCaml	13
1.5.1	Configurando OCaml	13
1.6	Smali	13
1.7	Saindo de um arquivo .lua para um arquivo .smali	14
1.8	Saindo de um arquivo .smali para um .dex	15
1.9	Rodando o programa .dex no Dalvik	15
1.9.1	O Android Debug Brigde - ADB	15
1.9.2	Conectando-se a um aparelho	16
1.9.3	Rodando o programa	16
2	A linguagem smali	18
2.1	Entendendo um programa smali	18
3	Programas	22
3.1	Nano Programas	22
3.2	Micro Programas	34
4	Analizador léxico	67
4.1	Reconhecendo os Tokens	67
4.2	Montagem do Analizador Léxico	68
4.2.1	Código do Analizador Léxico	68
4.2.2	Utilização	72
4.3	Testando o Analizador Léxico	73
4.3.1	Programas Nano	73
4.3.2	Programas Micro	76
4.4	Teste de Erros	81
5	Analizador Sintático	84
5.1	Código do Analizador Sintático	84
5.2	Código da Árvore Sintática Abstrata	88
5.3	Código do sintaticoTest	89
5.4	Novo código do analisador léxico	90

5.5	Usando o analisador sintático	93
5.5.1	Pré-requisitos	93
5.5.2	Compilando o analisador sintático	93
5.5.3	Executando o analisador	93
5.6	Testando o Analisador Sintático	94
5.6.1	Programas nano	94
5.6.2	Programas micro	100
5.7	Testes de Erros Sintáticos	110
6	Analisador Semântico	114
6.1	Códigos dessa etapa	114
6.1.1	Código do Analisador Semântico	114
6.1.2	Código do Ambiente	122
6.1.3	Novo código da árvore abstrata	123
6.1.4	Código do novo lexico	124
6.1.5	Código do novo analisador sintático	126
6.1.6	Código do arquivo para testar o analisador semântico	131
6.1.7	Código do Sast	135
6.1.8	Código do Tast	135
6.1.9	Código do Tabsimb	136
6.2	Usando o analisador semântico	137
6.2.1	Compilando o analisador semântico	137
6.2.2	Executando o analisador semântico	137
6.3	Testando o Analisador Semântico	138
6.3.1	Programas nano	138
6.3.2	Programas micro	152
6.4	Testes de Erros Semânticos	179
7	Intérprete	183
7.1	Compilando e Executando	183
7.2	Arquivos	183

Capítulo 1

Introdução

Este Documento está sendo escrito como relatório para as diversas atividades propostas na disciplina de Construção de Compiladores, cuja finalidade é a de desenvolver meu conhecimento acerca do processo de construção de um compilador. Para tal, ao final da disciplina, terei construído um compilador de Lua para Dalvik, utilizando a linguagem OCaml. De forma que será necessário a aprendizagem da linguagem Lua e da linguagem OCaml de programação, assim como o entendimento do código gerado para a máquina virtual Dalvik e sua utilização.

Este capítulo contém informações acerca das tecnologias que serão usadas no decorrer deste trabalho, bem como suas configurações e modo de uso. Dessa forma, serve de base para os capítulos e atividades seguintes, possibilitando a execução deles.

1.1 Sistema Operacional

A versão 18.04.1 LTS da distribuição Ubuntu do sistema operacional Linux está sendo usado para a realização deste trabalho. O download da última versão do ubuntu (18.04.1 LTS quando esse relatório foi escrito) pode ser realizado pelo seguinte [link para a página oficial](#).

1.2 Lua

A linguagem de programação Lua é uma poderosa, leve e eficiente, projetada para entender aplicações. Ela permite programação orientada a objetos, programação orientada a dados e descrição de dados, programação procedural e programação funcional. A linguagem é tipada dinamicamente e é executada via interpretação de bytecodes para uma máquina virtual baseada em registradores. Além disso, Lua é rápida, portátil e de código aberto.

1.2.1 Configurando Lua

Para configurar a linguagem Lua para sua última versão (5.3.5 durante a escrita deste relatório) basta executar os seguintes comandos em um terminal linux

```
$ sudo apt install build-essential libreadline-dev
$ sudo apt-get update
$ mkdir lua_build
$ cd lua_build
$ sudo curl -R -O http://www.lua.org/ftp/lua-5.3.5.tar.gz
$ sudo tar -zxvf lua-5.3.5.tar.gz
$ cd lua-5.3.5
$ sudo make linux test
$ sudo make install
```

Para verificar a versão

```
$ lua -v
```

Para executar programas em .lua, basta executar

```
$ lua arquivo.lua
```

1.3 Java JDK

A Java JDK é um conjunto de utilitários cuja função é permitir criar sistemas de software para a plataforma java. Uma vez que utilizaremos a máquina virtual Dalvik, faz-se necessário instalar java e sua jdk na plataforma.

1.3.1 Configurando a Java JDK

Antes que se possa configurar a Dalvik, faz-se necessário a configuração do Java JDK. Será utilizado a Versão 1.8 da JDK devido a sua maior compatibilidade. Para tal, executa-se os seguintes comandos pelo terminal

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
```

Para verificar a versão instalada

```
$ java -version
```

1.4 Dalvik

A Dalvik é uma máquina virtual projetada e escrita por Dan Bornstein e com contribuições de outros engenheiros do Google para fazer parte da plataforma Android. Ao invés de

usar registradores virtuais, ela usa registradores reais, de forma que foi otimizada para utilizar pouca memória e permitir múltiplas instâncias da máquina virtual rodando ao mesmo tempo. Dalvik possui uma ferramenta chamada dx, incluída no SDK do Android, que permite transformar arquivos do formato .class em arquivos do formato .dex, que é o formato usado por ela.

1.4.1 Configurando a Dalvik

A dalvik está incluída no ambiente de desenvolvimento Android, portanto sua instalação é simples. Basta realizar o download do Android Studio ([link para a versão mais recente](#)) e instalar. Além disso, caso o sistema operacional que estiver sendo usado for de 64 bits, deve-se executar os seguintes comandos no terminal para que algumas bibliotecas de 32 bits necessárias sejam instaladas

```
$ sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386 lib32z1  
libbz2-1.0:i386
```

1.5 OCaml

OCaml (Objective Caml) é uma linguagem de programação de propósito geral, com ênfase em expressividade e segurança. A linguagem permite dois tipos de compilação, uma para bytecode (ocorrendo na máquina virtual zinc) ou para código de máquina nativo para um grande número de plataformas. Suas vantagens advêm de seu compilador, que gera código nativo rapidamente e com excelente desempenho no quesito tempo de execução, além de uma biblioteca base extensa.

1.5.1 Configurando OCaml

Para realizar a instalação da versão mais recente da linguagem OCaml (4.05.0 quando este relatório foi escrito) no Ubuntu, executa-se os seguintes comandos em um terminal Linux

```
$ sudo apt install ocaml-nox  
$ sudo apt install ocaml
```

Para verificar sua versão, executa-se

```
$ ocaml -version
```

1.6 Smali

Como o arquivo .dex não é facilmente legível em sua forma bytecode, que é o formato executado pelo Dalvik, precisamos de uma ferramenta que faça a transformação de um

arquivo .dex para um arquivo .smali, que é um formato mais facilmente legível e desenvolvido justamente para isso. Smali oferece recursos para ir de .dex para .smali (baksmali) e para ir de .smali de volta para .dex (smali). Para realizar o download, entre nesse [link](#) (a versão usada nesse relatório foi a mais recente no tempo de escrita, 2.2.4). Após feito o download, mude o nome dos programas para smali e baksmali respectivamente para facilitar sua execução, de forma que não haja necessidade de digitar a versão nem o formato dos arquivos no terminal.

1.7 Saindo de um arquivo .lua para um arquivo .smali

Como dito anteriormente, um arquivo .smali é uma maneira prática de se ler o bytecode de um programa executável por Dalvik. Isso permite não apenas a fácil alteração de tais arquivos (uma vez que há uma maneira de conversão de volta de .smali para .dex), mas também auxilia o estudo de tal bytecode e do funcionamento de Dalvik. Porém, a linguagem lua não possui uma maneira prática de produzir um executável para Dalvik, de forma que os programas em .lua devam ser traduzidos de forma manual para programas em Java, com formato .java. A partir daí, pode-se compilar tais programas para .class por meio de um compilador java normal (o javac) e, por fim, para o arquivo final da Dalvik com formato .dex utilizando-se de uma ferramenta da própria Dalvik, o dx.

Para fazer isso, primeiramente deve-se compilar os programas em .java já traduzidos dos programas em .lua por meio da seguinte execução em um terminal Linux

```
$ javac arquivo.java
```

Sendo arquivo.java o nome do arquivo a ser compilado. Em seguida, pode-se executar os programas caso desejado com

```
$ java arquivo.class
```

Onde arquivo.class é o nome do arquivo que foi compilado a ser executado.

Após essa etapa, deve-se transformar os arquivos .class para arquivos .dex. Para isso, utiliza-se a ferramenta dx provida pela própria Dalvik. Pode-se fazer isso da seguinte forma

```
$ androidStudio-path/Sdk/build-tools/version/dx --dex --output=arquivo.dex
arquivo.class
```

Onde androidStudio-path é o caminho para a pasta onde está o Android Studio e version o número de versão do programa (home/nomeUsuario/Android/Sdk/build-tools/28.0.2/dx no meu caso, sendo nomeUsuario o nome de usuário da minha máquina).

Até esse momento, temos o arquivo .dex do programa original .java. Agora, para facilitar a leitura de seu código bytecode, vamos transformar esse arquivo para .smali utilizando o baksmali baixado anteriormente. Para isso, é necessário que o programa baksmali esteja na mesma pasta dos arquivos a serem transformados e que se execute o seguinte código

```
$ java -jar baksmali disassemble arquivo.dex
```

Onde arquivo.dex é o arquivo a ser traduzido para .smali.

Pronto! Agora já temos um arquivo de fácil leitura onde podemos analisar o bytecode rodado pela máquina virtual Dalvik. Basta abrir o arquivo .smali com um editor de texto comum qualquer.

1.8 Saindo de um arquivo .smali para um .dex

Com o propósito do trabalho de construir um compilador de lua para dalvik em mente, agora precisamos de uma forma de passar um arquivo do formato .smali para o formato .dex, para que ele possa ser executado pela máquina virtual. Para tal, usaremos o programa smali, baixado anteriormente. Com o arquivo do smali no mesmo endereço onde encontram-se os arquivos .smali a serem traduzidos, basta executar o seguinte comando para obter-se os arquivos .dex correspondentes

```
$ java -jar smali assemble arquivo.smali
```

Onde arquivo.smali é o arquivo a ser traduzido. Pronto! Agora é só mandar os arquivos .dex para a Dalvik para que sejam executados.

1.9 Rodando o programa .dex no Dalvik

Para que possamos executar os arquivos .dex em Dalvik, necessitamos de um aparelho Android, seja um aparelho físico ou um emulador que rode o sistema operacional em questão. O processo para os dois é bastante similar, mudando apenas a forma de se conectar aos dois.

1.9.1 O Android Debug Bridge - ADB

O ADB é um componente da SDK de ferramentas da plataforma do Android. É uma utilidade da linha de comando que permite realizar ações como controlar o seu aparelho de um computador por meio de uma conexão USB, instalar e desinstalar apps, copiar arquivos tanto de quanto para seu aparelho, executar comandos shell e entre outras funções. É uma ferramenta essencial para o desenvolvimento para a plataforma Android.

Configurando o ADB

Para instalar o ADB bastam as instruções a seguir serem executadas em um terminal em seu linux

```
$ sudo apt update
$ sudo apt install android-tools-adb
```

Para verificar a versão instalada

```
$ adb version
```

Por fim, uma função útil é a de verificar a lista de aparelhos disponíveis para uso com a ferramenta, podendo ser executada da seguinte forma

```
$ adb devices
```


1.9.2 Conectando-se a um aparelho

Aqui é onde os processos entre um simulador de aparelho Android e um aparelho real se diferem. Seguem instruções para configurar as duas alternativas.

Aparelho real

Para configurar um aparelho real primeiramente deve-se habilitar a função de desenvolvedor nas configurações. Esse processo pode variar de aparelho para aparelho, porém de forma geral deve-se acessar as configurações e então a sobre o dispositivo. No final da lista deve-se encontrar uma informação com o Número da versão. Pressione repetidas vezes essa informação para habilitar as opções de desenvolvedor. Caso não funcione em seu aparelho Android, faça uma pesquisa na internet de como fazê-lo para seu dispositivo em específico.

Com as Opções do desenvolvedor habilitadas, ache a opção de Depuração USB e a ative. Pronto, agora toda vez que quiser usar seu aparelho Android para realizar operações pela ADB basta conectá-lo à um computador utilizando-se de um cabo USB e mantendo-o desbloqueado.

Simulador Android

Para um simulador Android, use o AVD Manager que já vem com o programa do Android Studio. Por meio dele, basta iniciar um aparelho que ele já estará pronto para ser usado com a ADB.

1.9.3 Rodando o programa

Agora com essas configurações realizadas, os passos para rodar o programa .dex são os mesmos tanto para um aparelho físico quanto para um aparelho simulado. Garantindo-se que há um aparelho disponível para a adb, deve-se primeiramente mandar o arquivo .dex a ser executado para o aparelho. Faz-se isso por meio do comando

```
$ adb push arquivo.dex /sdcard/
```

Onde arquivo.dex é o arquivo a ser executado e /sdcard/ o local no aparelho para onde esse arquivo será enviado. O próximo passo é pedir para a Dalvikvm executar o programa, por meio da adb shell como se segue

```
$ adb shell dalvikvm -cp /sdcard/arquivo.dex arquivoClasse
```

Onde arquivoClasse é a especificação de qual classe do arquivo a Dalvik irá executar. No caso de um exemplo onde o programa micro01 foi mandado para a pasta /sdcard/ do aparelho e sua única classe é também chamada micro01, temos o seguinte comando para executá-lo

```
$ adb shell dalvikvm -cp /sdcard/micro01.dex micro01
```

Pronto, agora a dalvik já está executando o programa .dex utilizando o aparelho Android, seja ele simulado ou real.

Obs: Caso haja mais de um dispositivo disponível para a ADB, faz-se necessário explicitar qual deles está-se usando por meio de parâmetros adicionais.

Capítulo 2

A linguagem smali

O Smali e o Baksmali são um assembler e disassembler para o formato .dex usado pela Dalvik no Android. Sua sintaxe tem base em Jasmim e Dedexer, outros disassemblers, e suporta toda a funcionalidade do formato dex. Por isso, saber usar ambas as ferramentas é de grande valia no entendimento e desenvolvimento do trabalho proposto.

2.1 Entendendo um programa smali

Para entendermos o formato smali, vamos analisar a estrutura de um programa smali e, em seguida, estudar um exemplo do formato.

Estrutura de um programa smali

A estrutura de um programa smali, de forma geral, segue a estrutura apresentada na imagem a seguir

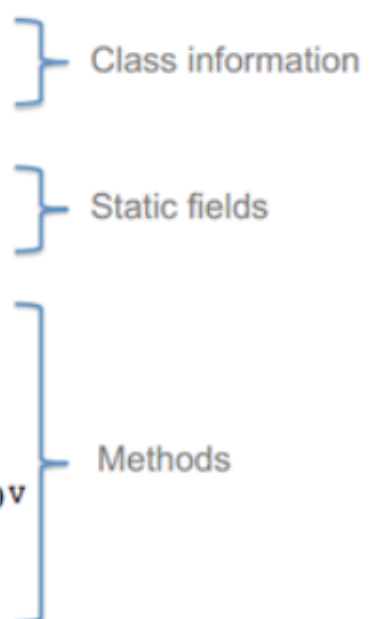
```
.class public Lcom/apkudo/util/Serializer;
.super Ljava/lang/Object;
.source "Serializer.java"

# static fields
.field public static final TAG:Ljava/lang/String; =
"ApkudoUtils"

# direct methods
.method public constructor <init>()V
    .registers 1

    .prologue
    .line 5
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    return-void
.end method
```



The diagram illustrates the structure of a smali program. It uses blue brackets on the right side to group the code into three main sections: 'Class information' (covering the class declaration and source file), 'Static fields' (covering the static field declaration), and 'Methods' (covering the constructor and the main method).

Exemplo de um programa smali

Realizar a análise de um programa smali permitirá melhorar nosso entendimento sobre a sintaxe desse assembler. Para isso, iremos utilizar o pseudocódigo a seguir como exemplo

Listagem 2.1: enunciado do programa micro04 - Lê números e informa quais estão entre 10 e 150

```

1 algoritmo "micro04"
2 /*
3 Função: Ler 5 números e ao final informar quantos número(s)
4 est(á)ão no intervalo entre 10 (inclusive) e 150 (inclusive).
5 */
6
7 var
8 x, num, intervalo: inteiro
9
10 início
11
12 para x de 1 até 5 faça
13   escreva("Digite um número: ")
14   leia(num)
15   se num >= 10 então
16     se num <= 150 então
17       intervalo      intervalo + 1
18   fim_se
19 fim_se
20 fim_para
21
22 escreval("Ao total, foram digitados ", intervalo, " números no
23 intervalo entre 10 e 150")
24
25 fim_algoritmo

```

A tradução desse pseudocódigo para um código escrito em java (micro04.java) é mostrado a seguir

Listagem 2.2: micro04.java - Lê números e informa quais estão entre 10 e 150

```

1 public class micro04{
2     public static void main(String[] args){
3         int x, num, intervalo;
4         intervalo = 0;
5         for (x = 1; x <= 5; x++){
6             System.out.print("Digite um numero: ");
7             num = Integer.parseInt(System.console().readLine());
8             if(num >= 10){
9                 if(num <=150){
10                     intervalo = intervalo + 1;
11                 }
12             }
13         }
14         System.out.println("Ao total, foram digitados " + intervalo + "
15                             numeros no intervalo entre 10 e 150");
16     }
17 }

```

E, finalmente, a tradução desse arquivo .java para o smali resulta no seguinte arquivo

Listagem 2.3: micro04.smali - Lê números e informa quais estão entre 10 e 150

```

1 .class public Lmicro04;
2 .super Ljava/lang/Object;
3 .source "micro04.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object; -> <init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 5
19
20     .prologue
21     .line 4
22     const/4 v0, 0x0
23
24     .line 5
25     const/4 v1, 0x1
26
27     :goto_2
28     const/4 v2, 0x5
29
30     if-gt v1, v2, :cond_25
31
32     .line 6
33     sget-object v2, Ljava/lang/System; -> out:Ljava/io/PrintStream;
34
35     const-string v3, "Digite um numero: "
36
37     invoke-virtual {v2, v3}, Ljava/io/PrintStream; -> print(Ljava/lang/
        String;)V
38
39     .line 7
40     invoke-static {}, Ljava/lang/System; -> console()Ljava/io/Console;
41
42     move-result-object v2
43
44     invoke-virtual {v2}, Ljava/io/Console; -> readLine()Ljava/lang/String;
45
46     move-result-object v2
47
48     invoke-static {v2}, Ljava/lang/Integer; -> parseInt(Ljava/lang/String;)I
49
50     move-result v2
51
52     .line 8
53     const/16 v3, 0xa
54
55     if-lt v2, v3, :cond_22
56
57     .line 9

```

2.1

```
58     const/16 v3, 0x96
59
60     if-gt v2, v3, :cond_22
61
62     .line 10
63     add-int/lit8 v0, v0, 0x1
64
65     .line 5
66     :cond_22
67     add-int/lit8 v1, v1, 0x1
68
69     goto :goto_2
70
71     .line 14
72     :cond_25
73     sget-object v1, Ljava/lang/System;-->out:Ljava/io/PrintStream;
74
75     new-instance v2, Ljava/lang/StringBuilder;
76
77     invoke-direct {v2}, Ljava/lang/StringBuilder;--><init>()V
78
79     const-string v3, "Ao total, foram digitados "
80
81     invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;-->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
82
83     move-result-object v2
84
85     invoke-virtual {v2, v0}, Ljava/lang/StringBuilder;-->append(I)Ljava/
        lang/StringBuilder;
86
87     move-result-object v0
88
89     const-string v2, " numeros no intervalo entre 10 e 150"
90
91     invoke-virtual {v0, v2}, Ljava/lang/StringBuilder;-->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
92
93     move-result-object v0
94
95     invoke-virtual {v0}, Ljava/lang/StringBuilder;-->toString()Ljava/lang/
        String;
96
97     move-result-object v0
98
99     invoke-virtual {v1, v0}, Ljava/io/PrintStream;-->println(Ljava/lang/
        String;)V
100
101     .line 15
102     return-void
103 .end method
```

Capítulo 3

Programas

3.1 Nano Programas

Listagem 3.1: nano01.lua - Módulo mínimo que caracteriza um programa

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

Listagem 3.2: nano01.java - Módulo mínimo que caracteriza um programa

```
1 public class nano01{  
2     public static void main(String[] args){  
3  
4     }  
5 }
```

Listagem 3.3: nano01.smali - Módulo mínimo que caracteriza um programa

```
1 .class public Lnano01;  
2 .super Ljava/lang/Object;  
3 .source "nano01.java"  
4  
5  
6 # direct methods  
7 .method public constructor <init>()V  
8     .registers 1  
9  
10     .prologue  
11     .line 1  
12     invoke-direct {p0}, Ljava/lang/Object; -><init>()V  
13  
14     return-void  
15 .end method  
16  
17 .method public static main([Ljava/lang/String;)V  
18     .registers 1  
19  
20     .prologue  
21     .line 4  
22     return-void
```


23 **.end method**

Listagem 3.4: nano02.lua - Declaração de uma variável

```

1 function main()
2     local n
3 end

```

Listagem 3.5: nano02.java - Declaração de uma variável

```

1 public class nano02{
2     public static void main(String[] args){
3         int n;
4     }
5 }

```

Listagem 3.6: nano02.smali - Declaração de uma variável

```

1 .class public Lnano02;
2 .super Ljava/lang/Object;
3 .source "nano02.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 1
19
20     .prologue
21     .line 4
22     return-void
23 .end method

```

Listagem 3.7: nano03.lua - Atribuição de um inteiro a uma variável

```

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

```

Listagem 3.8: nano03.java - Atribuição de um inteiro a uma variável

```

1 public class nano03{
2     public static void main(String[] args){
3         int n;
4         n = 1;

```

```

5     }
6 }

```

Listagem 3.9: nano03.smali - Atribuição de um inteiro a uma variável

```

1 .class public Lnano03;
2 .super Ljava/lang/Object;
3 .source "nano03.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object; -> <init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 1
19
20     .prologue
21     .line 4
22     .line 5
23     return-void
24 .end method

```

Listagem 3.10: nano04.lua - Atribuição de uma soma de inteiros a uma variável

```

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

```

Listagem 3.11: nano04.java - Atribuição de uma soma de inteiros a uma variável

```

1 public class nano04{
2     public static void main(String[] args){
3         int n;
4         n = 1 + 2;
5     }
6 }

```

Listagem 3.12: nano04.smali - Atribuição de uma soma de inteiros a uma variável

```

1 .class public Lnano04;
2 .super Ljava/lang/Object;
3 .source "nano04.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1

```

3.1

```
9
10     .prologue
11     .line 1
12     invoke-direct {p0}, Ljava/lang/Object;--><init>()V
13
14     return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 1
19
20     .prologue
21     .line 4
22     .line 5
23     return-void
24 .end method
```

Listagem 3.13: nano05.lua - Inclusão do comando de impressão

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

Listagem 3.14: nano05.java - Inclusão do comando de impressão

```
1 public class nano05{
2     public static void main(String[] args){
3         int n;
4         n = 2;
5         System.out.print(n);
6     }
7 }
```

Listagem 3.15: nano05.smali - Inclusão do comando de impressão

```
1 .class public Lnano05;
2 .super Ljava/lang/Object;
3 .source "nano05.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;--><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
```

```

20     .prologue
21     .line 4
22     const/4 v0, 0x2
23
24     .line 5
25     sget-object v1, Ljava/lang/System; ->out:Ljava/io/PrintStream;
26
27     invoke-virtual {v1, v0}, Ljava/io/PrintStream; ->print(I)V
28
29     .line 6
30     return-void
31 .end method

```

Listagem 3.16: nano06.lua - Atribuição de uma subtração de inteiros a uma variável

```

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

```

Listagem 3.17: nano06.java - Atribuição de uma subtração de inteiros a uma variável

```

1 public class nano06{
2     public static void main(String[] args){
3         int n;
4         n = 1 - 2;
5         System.out.print(n);
6     }
7 }

```

Listagem 3.18: nano06.smali - Atribuição de uma subtração de inteiros a uma variável

```

1 .class public Lnano06;
2 .super Ljava/lang/Object;
3 .source "nano06.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object; -><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20    .prologue
21    .line 4
22    const/4 v0, -0x1
23

```

3.1

```
24     .line 5
25     sget-object v1, Ljava/lang/System;->out:Ljava/io/PrintStream;
26
27     invoke-virtual {v1, v0}, Ljava/io/PrintStream;->print(I)V
28
29     .line 6
30     return-void
31 .end method
```

Listagem 3.19: nano07.lua - 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 3.20: nano07.java - Inclusão do comando condicional

```
1 public class nano07{
2     public static void main(String[] args){
3         int n;
4         n = 1;
5         if(n == 1){
6             System.out.print(n);
7         }
8     }
9 }
```

Listagem 3.21: nano07.smali - Inclusão do comando condicional

```
1 .class public Lnano07;
2 .super Ljava/lang/Object;
3 .source "nano07.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20    .prologue
21    .line 4
22    const/4 v0, 0x1
23
```

```

24     .line 6
25     sget-object v1, Ljava/lang/System; ->out:Ljava/io/PrintStream;
26
27     invoke-virtual {v1, v0}, Ljava/io/PrintStream; ->print(I)V
28
29     .line 8
30     return-void
31 .end method

```

Listagem 3.22: nano08.lua - 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 3.23: nano08.java - Inclusão do comando condicional com parte senão

```

1 public class nano08{
2     public static void main(String[] args){
3         int n;
4         n = 1;
5         if(n == 1){
6             System.out.print(n);
7         }else{
8             System.out.print(0);
9         }
10    }
11 }

```

Listagem 3.24: nano08.smali - Inclusão do comando condicional com parte senão

```

1 .class public Lnano08;
2 .super Ljava/lang/Object;
3 .source "nano08.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object; -><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19

```

3.1

```
20 .prologue
21 .line 4
22 const/4 v0, 0x1
23
24 .line 6
25 sget-object v1, Ljava/lang/System;-->out:Ljava/io/PrintStream;
26
27 invoke-virtual {v1, v0}, Ljava/io/PrintStream;-->print(I)V
28
29 .line 10
30 return-void
31 .end method
```

Listagem 3.25: nano09.lua - Atribuição de duas operações aritméticas sobre inteiros a uma variável

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

Listagem 3.26: nano09.java - Atribuição de duas operações aritméticas sobre inteiros a uma variável

```
1 public class nano09{
2     public static void main(String[] args){
3         int n;
4         n = 1 + 1 / 2;
5         if(n == 1){
6             System.out.print(n);
7         }else{
8             System.out.print(0);
9         }
10    }
11 }
```

Listagem 3.27: nano09.smali - Atribuição de duas operações aritméticas sobre inteiros a uma variável

```
1 .class public Lnano09;
2 .super Ljava/lang/Object;
3 .source "nano09.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10 .prologue
11 .line 1
12 invoke-direct {p0}, Ljava/lang/Object;--><init>()V
```



```

13
14     return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20     .prologue
21     .line 4
22     const/4 v0, 0x1
23
24     .line 6
25     sget-object v1, Ljava/lang/System; ->out:Ljava/io/PrintStream;
26
27     invoke-virtual {v1, v0}, Ljava/io/PrintStream; ->print(I)V
28
29     .line 10
30     return-void
31 .end method

```

Listagem 3.28: nano10.lua - Atribuição de duas variáveis inteiras

```

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

```

Listagem 3.29: nano10.java - Atribuição de duas variáveis inteiras

```

1 public class nano10{
2     public static void main(String[] args){
3         int n, m;
4         n = 1;
5         m = 2;
6         if(n == m ){
7             System.out.print(n);
8         }else{
9             System.out.print(0);
10        }
11    }
12 }

```

Listagem 3.30: nano10.smali - Atribuição de duas variáveis inteiras

```

1 .class public Lnano10;
2 .super Ljava/lang/Object;
3 .source "nano10.java"
4
5
6 # direct methods
7 .method public constructor <init>()V

```

3.1

```
8      .registers 1
9
10     .prologue
11     .line 1
12     invoke-direct {p0}, Ljava/lang/Object;--><init>()V
13
14     return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20     .prologue
21     .line 4
22     .line 9
23     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
24
25     const/4 v1, 0x0
26
27     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->print(I)V
28
29     .line 11
30     return-void
31 .end method
```

Listagem 3.31: nano11.lua - Introdução do comando de repetição enquanto

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

Listagem 3.32: nano11.java - Introdução do comando de repetição enquanto

```
1 public class nano11{
2     public static void main(String[] args){
3         int n, m, x;
4         n = 1;
5         m = 2;
6         x = 5;
7         while(x > n){
8             n = n + m;
9             System.out.print(n);
10        }
11    }
12 }
```

Listagem 3.33: nano11.smali - Introdução do comando de repetição enquanto

```
1 .class public Lnano11;
2 .super Ljava/lang/Object;
3 .source "nano11.java"
```

```

4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object; -><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 5
19
20    .prologue
21    .line 4
22    const/4 v0, 0x1
23
24    .line 5
25    const/4 v1, 0x2
26
27    .line 6
28    const/4 v2, 0x5
29
30    .line 7
31    :goto_3
32    if-le v2, v0, :cond_c
33
34    .line 8
35    add-int/2addr v0, v1
36
37    .line 9
38    sget-object v3, Ljava/lang/System; ->out:Ljava/io/PrintStream;
39
40    invoke-virtual {v3, v0}, Ljava/io/PrintStream; ->print(I)V
41
42    goto :goto_3
43
44    .line 11
45    :cond_c
46    return-void
47 .end method

```

Listagem 3.34: nano12.lua - Comando condicional aninhado em um comando de repetição

```

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

```

3.1

```
13
14 main()
```

Listagem 3.35: nano12.java - Comando condicional aninhado em um comando de repetição

```
1 public class nano12{
2     public static void main(String[] args){
3         int n, m, x;
4         n = 1;
5         m = 2;
6         x = 5;
7         while (x > n){
8             if (n == m){
9                 System.out.print(n);
10            }else{
11                System.out.print(0);
12            }
13            x = x - 1;
14        }
15    }
16 }
```

Listagem 3.36: nano12.smali - Comando condicional aninhado em um comando de repetição

```
1 .class public Lnano12;
2 .super Ljava/lang/Object;
3 .source "nano12.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 5
19
20     .prologue
21     .line 4
22     const/4 v1, 0x1
23
24     .line 6
25     const/4 v0, 0x5
26
27     .line 7
28     :goto_2
29     if-le v0, v1, :cond_d
30
31     .line 11
32     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
33
34     const/4 v3, 0x0
35
```

```

36     invoke-virtual {v2, v3}, Ljava/io/PrintStream;->print(I)V
37
38     .line 13
39     add-int/lit8 v0, v0, -0x1
40
41     goto :goto_2
42
43     .line 15
44     :cond_d
45     return-void
46 .end method

```

3.2 Micro Programas

Listagem 3.37: micro01.lua - Converte graus Celsius para Fahrenheit

```

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

```

Listagem 3.38: micro01.java - Converte graus Celsius para Fahrenheit

```

1 public class micro01{
2     public static void main(String[] args){
3         double cel, far;
4         System.out.println("Tabela de conversao: Celsius -> Fahrenheit");
5         System.out.print("Digite a temperatura em Celsius: ");
6         cel = Double.parseDouble(System.console().readLine());
7         far = ((9 * cel) + 160) / 5;
8         System.out.println("A nova temperatura e " + far + " F");
9     }
10 }

```

Listagem 3.39: micro01.smali - Converte graus Celsius para Fahrenheit

```

1 .class public Lmicro01;
2 .super Ljava/lang/Object;
3 .source "micro01.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method

```

3.2

```
16
17 .method public static main([Ljava/lang/String;)V
18   .registers 6
19
20   .prologue
21   .line 4
22   sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
23
24   const-string v1, "Tabela de conversao: Celsius -> Fahrenheit"
25
26   invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->println(Ljava/lang/
    String;)V
27
28   .line 5
29   sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
30
31   const-string v1, "Digite a temperatura em Celsius: "
32
33   invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->print(Ljava/lang/
    String;)V
34
35   .line 6
36   invoke-static {}, Ljava/lang/System;-->console()Ljava/io/Console;
37
38   move-result-object v0
39
40   invoke-virtual {v0}, Ljava/io/Console;-->readLine()Ljava/lang/String;
41
42   move-result-object v0
43
44   invoke-static {v0}, Ljava/lang/Double;-->parseDouble(Ljava/lang/String
    ;)D
45
46   move-result-wide v0
47
48   .line 7
49   const-wide/high16 v2, 0x4022000000000000L    # 9.0
50
51   mul-double/2addr v0, v2
52
53   const-wide/high16 v2, 0x4064000000000000L    # 160.0
54
55   add-double/2addr v0, v2
56
57   const-wide/high16 v2, 0x4014000000000000L    # 5.0
58
59   div-double/2addr v0, v2
60
61   .line 8
62   sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
63
64   new-instance v3, Ljava/lang/StringBuilder;
65
66   invoke-direct {v3}, Ljava/lang/StringBuilder;--><init>()V
67
68   const-string v4, "A nova temperatura e "
69
70   invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;-->append(Ljava/lang/
    String;)Ljava/lang/StringBuilder;
```

```

71
72     move-result-object v3
73
74     invoke-virtual {v3, v0, v1}, Ljava/lang/StringBuilder;->append(D)Ljava
        /lang/StringBuilder;
75
76     move-result-object v0
77
78     const-string v1, " F"
79
80     invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
81
82     move-result-object v0
83
84     invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
        String;
85
86     move-result-object v0
87
88     invoke-virtual {v2, v0}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
89
90     .line 9
91     return-void
92 .end method

```

Listagem 3.40: micro02.lua - 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 3.41: micro02.java - Ler dois inteiros e decide qual é maior

```

1 public class micro02{
2     public static void main(String[] args){
3         int num1, num2;
4         System.out.print("Digite o primeiro numero: ");
5         num1 = Integer.parseInt(System.console().readLine());
6         System.out.print("Digite o segundo numero: ");
7         num2 = Integer.parseInt(System.console().readLine());
8
9         if(num1 > num2){
10             System.out.print("O primeiro numero " + num1 + " e maior que o
                segundo " + num2);
11         }else{

```



```

12         System.out.print("O segundo numero " + num2 + " e maior que o
           primeiro " + num1);
13     }
14 }
15 }

```

Listagem 3.42: micro02.smali - Ler dois inteiros e decide qual é maior

```

1 .class public Lmicro02;
2 .super Ljava/lang/Object;
3 .source "micro02.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 6
19
20     .prologue
21     .line 4
22     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
23
24     const-string v1, "Digite o primeiro numero: "
25
26     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->print (Ljava/lang/
        String;)V
27
28     .line 5
29     invoke-static {}, Ljava/lang/System;-->console()Ljava/io/Console;
30
31     move-result-object v0
32
33     invoke-virtual {v0}, Ljava/io/Console;-->readLine()Ljava/lang/String;
34
35     move-result-object v0
36
37     invoke-static {v0}, Ljava/lang/Integer;-->parseInt (Ljava/lang/String;)I
38
39     move-result v0
40
41     .line 6
42     sget-object v1, Ljava/lang/System;-->out:Ljava/io/PrintStream;
43
44     const-string v2, "Digite o segundo numero: "
45
46     invoke-virtual {v1, v2}, Ljava/io/PrintStream;-->print (Ljava/lang/
        String;)V
47
48     .line 7
49     invoke-static {}, Ljava/lang/System;-->console()Ljava/io/Console;

```

```

50
51     move-result-object v1
52
53     invoke-virtual {v1}, Ljava/io/Console;->readLine()Ljava/lang/String;
54
55     move-result-object v1
56
57     invoke-static {v1}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
58
59     move-result v1
60
61     .line 9
62     if-le v0, v1, :cond_4b
63
64     .line 10
65     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
66
67     new-instance v3, Ljava/lang/StringBuilder;
68
69     invoke-direct {v3}, Ljava/lang/StringBuilder;-<init>()V
70
71     const-string v4, "O primeiro numero "
72
73     invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
74
75     move-result-object v3
76
77     invoke-virtual {v3, v0}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
78
79     move-result-object v0
80
81     const-string v3, " e maior que o segundo "
82
83     invoke-virtual {v0, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
84
85     move-result-object v0
86
87     invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
88
89     move-result-object v0
90
91     invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
        String;
92
93     move-result-object v0
94
95     invoke-virtual {v2, v0}, Ljava/io/PrintStream;->print(Ljava/lang/
        String;)V
96
97     .line 14
98     :goto_4a
99     return-void
100
101     .line 12
102     :cond_4b

```

```

103     sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
104
105     new-instance v3, Ljava/lang/StringBuilder;
106
107     invoke-direct {v3}, Ljava/lang/StringBuilder;--><init>()V
108
109     const-string v4, "O segundo numero "
110
111     invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;-->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
112
113     move-result-object v3
114
115     invoke-virtual {v3, v1}, Ljava/lang/StringBuilder;-->append(I)Ljava/
        lang/StringBuilder;
116
117     move-result-object v1
118
119     const-string v3, " e maior que o primeiro "
120
121     invoke-virtual {v1, v3}, Ljava/lang/StringBuilder;-->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
122
123     move-result-object v1
124
125     invoke-virtual {v1, v0}, Ljava/lang/StringBuilder;-->append(I)Ljava/
        lang/StringBuilder;
126
127     move-result-object v0
128
129     invoke-virtual {v0}, Ljava/lang/StringBuilder;-->toString()Ljava/lang/
        String;
130
131     move-result-object v0
132
133     invoke-virtual {v2, v0}, Ljava/io/PrintStream;-->print(Ljava/lang/
        String;)V
134
135     goto :goto_4a
136 .end method

```

Listagem 3.43: micro03.lua - Lê um número e verifica se ele está entre 100 e 200

```

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

```

Listagem 3.44: micro03.java - Lê um número e verifica se ele está entre 100 e 200

```

1 public class micro03{
2     public static void main(String[] args){
3         int numero;
4         System.out.print("Digite um numero: ");
5         numero = Integer.parseInt(System.console().readLine());
6         if(numero >= 100){
7             if(numero <= 200){
8                 System.out.println("O numero esta no intervalo entre 100 e
9                     200");
10            }else{
11                System.out.println("O numero nao esta no intervalo entre
12                    100 e 200");
13            }
14        }else{
15            System.out.println("O numero nao esta no intervalo entre 100 e
16                200");
17        }
18    }
19 }

```

Listagem 3.45: micro03.smali - Lê um número e verifica se ele está entre 100 e 200

```

1 .class public Lmicro03;
2 .super Ljava/lang/Object;
3 .source "micro03.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20     .prologue
21     .line 4
22     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
23
24     const-string v1, "Digite um numero: "
25
26     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->print(Ljava/lang/
27         String;)V
28
29     .line 5
30     invoke-static {}, Ljava/lang/System;->console()Ljava/io/Console;
31
32     move-result-object v0
33
34     invoke-virtual {v0}, Ljava/io/Console;->readLine()Ljava/lang/String;
35
36     move-result-object v0

```

3.2

```
36
37     invoke-static {v0}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;) I
38
39     move-result v0
40
41     .line 6
42     const/16 v1, 0x64
43
44     if-lt v0, v1, :cond_2b
45
46     .line 7
47     const/16 v1, 0xc8
48
49     if-gt v0, v1, :cond_23
50
51     .line 8
52     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
53
54     const-string v1, "O numero esta no intervalo entre 100 e 200"
55
56     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
57
58     .line 15
59     :goto_22
60     return-void
61
62     .line 10
63     :cond_23
64     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
65
66     const-string v1, "O numero nao esta no intervalo entre 100 e 200"
67
68     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
69
70     goto :goto_22
71
72     .line 13
73     :cond_2b
74     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
75
76     const-string v1, "O numero nao esta no intervalo entre 100 e 200"
77
78     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
79
80     goto :goto_22
81 .end method
```

Listagem 3.46: micro04.lua - Lê números e informa quais estão entre 10 e 150

```
1 function main()
2     local x, num, intervalo
3     intervalo = 0
4     for x = 1, 5, 1
5     do
6         print("Digite um numero: ")
7         num = io.read("*number")
```

```

8         if num >= 10 then
9             if num <= 150 then
10                 intervalo = intervalo + 1
11             end
12         end
13     end
14     print("Ao total, foram digitados "..intervalo.." numeros no intervalo
15         entre 10 e 150")
16 end
17 main()

```

Listagem 3.47: micro04.java - Lê números e informa quais estão entre 10 e 150

```

1 public class micro04{
2     public static void main(String[] args){
3         int x, num, intervalo;
4         intervalo = 0;
5         for (x = 1; x <= 5; x++){
6             System.out.print("Digite um numero: ");
7             num = Integer.parseInt(System.console().readLine());
8             if(num >= 10){
9                 if(num <=150){
10                     intervalo = intervalo + 1;
11                 }
12             }
13         }
14         System.out.println("Ao total, foram digitados " + intervalo + "
15             numeros no intervalo entre 10 e 150");
16     }
17 }

```

Listagem 3.48: micro04.smali - Lê números e informa quais estão entre 10 e 150

```

1 .class public Lmicro04;
2 .super Ljava/lang/Object;
3 .source "micro04.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10     .prologue
11     .line 1
12     invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14     return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 5
19
20     .prologue
21     .line 4
22     const/4 v0, 0x0
23
24     .line 5
25     const/4 v1, 0x1

```

3.2

```
26
27 :goto_2
28 const/4 v2, 0x5
29
30 if-gt v1, v2, :cond_25
31
32 .line 6
33 sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
34
35 const-string v3, "Digite um numero: "
36
37 invoke-virtual {v2, v3}, Ljava/io/PrintStream;-->print(Ljava/lang/
    String;)V
38
39 .line 7
40 invoke-static {}, Ljava/lang/System;-->console()Ljava/io/Console;
41
42 move-result-object v2
43
44 invoke-virtual {v2}, Ljava/io/Console;-->readLine()Ljava/lang/String;
45
46 move-result-object v2
47
48 invoke-static {v2}, Ljava/lang/Integer;-->parseInt(Ljava/lang/String;)I
49
50 move-result v2
51
52 .line 8
53 const/16 v3, 0xa
54
55 if-lt v2, v3, :cond_22
56
57 .line 9
58 const/16 v3, 0x96
59
60 if-gt v2, v3, :cond_22
61
62 .line 10
63 add-int/lit8 v0, v0, 0x1
64
65 .line 5
66 :cond_22
67 add-int/lit8 v1, v1, 0x1
68
69 goto :goto_2
70
71 .line 14
72 :cond_25
73 sget-object v1, Ljava/lang/System;-->out:Ljava/io/PrintStream;
74
75 new-instance v2, Ljava/lang/StringBuilder;
76
77 invoke-direct {v2}, Ljava/lang/StringBuilder;--><init>()V
78
79 const-string v3, "Ao total, foram digitados "
80
81 invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;-->append(Ljava/lang/
    String;)Ljava/lang/StringBuilder;
82
```

```

83     move-result-object v2
84
85     invoke-virtual {v2, v0}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
86
87     move-result-object v0
88
89     const-string v2, " numeros no intervalo entre 10 e 150"
90
91     invoke-virtual {v0, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
92
93     move-result-object v0
94
95     invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
        String;
96
97     move-result-object v0
98
99     invoke-virtual {v1, v0}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
100
101     .line 15
102     return-void
103 .end method

```

Listagem 3.49: micro05.lua - Lê strings e caracteres

```

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

```

Listagem 3.50: micro05.java - Lê strings e caracteres

```

1 public class micro05{
2     public static void main(String[] args){
3         String nome, sexo;
4         int x, h, m;
5         h = 0;
6         m = 0;

```



```

7         for( x=1; x <= 5; x++){
8             System.out.print("Digite o nome: ");
9             nome = System.console().readLine();
10            System.out.print("H - Homem ou M - Mulher:");
11            sexo = System.console().readLine();
12            if(sexo == "H"){
13                h = h + 1;
14            }else if(sexo == "M"){
15                m = m + 1;
16            }else{
17                System.out.println("Sexo so pode ser H ou M!");
18            }
19        }
20        System.out.println("Foram inseridos " + h + " Homens");
21        System.out.println("Foram inseridos " + m + " Mulheres");
22    }
23 }

```

Listagem 3.51: micro05.smali - Lê strings e caracteres

```

1 .class public Lmicro05;
2 .super Ljava/lang/Object;
3 .source "micro05.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 6
19
20     .prologue
21     const/4 v2, 0x0
22
23     .line 5
24     .line 7
25     const/4 v3, 0x1
26
27     move v0, v2
28
29     move v1, v2
30
31     :goto_4
32     const/4 v2, 0x5
33
34     if-gt v3, v2, :cond_3d
35
36     .line 8
37     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
38
39     const-string v4, "Digite o nome: "

```

```

40
41     invoke-virtual {v2, v4}, Ljava/io/PrintStream;->print(Ljava/lang/
        String;)V
42
43     .line 9
44     invoke-static {}, Ljava/lang/System;->console()Ljava/io/Console;
45
46     move-result-object v2
47
48     invoke-virtual {v2}, Ljava/io/Console;->readLine()Ljava/lang/String;
49
50     .line 10
51     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
52
53     const-string v4, "H - Homem ou M - Mulher:"
54
55     invoke-virtual {v2, v4}, Ljava/io/PrintStream;->print(Ljava/lang/
        String;)V
56
57     .line 11
58     invoke-static {}, Ljava/lang/System;->console()Ljava/io/Console;
59
60     move-result-object v2
61
62     invoke-virtual {v2}, Ljava/io/Console;->readLine()Ljava/lang/String;
63
64     move-result-object v2
65
66     .line 12
67     const-string v4, "H"
68
69     if-ne v2, v4, :cond_2e
70
71     .line 13
72     add-int/lit8 v1, v1, 0x1
73
74     .line 7
75     :goto_2a
76     add-int/lit8 v2, v3, 0x1
77
78     move v3, v2
79
80     goto :goto_4
81
82     .line 14
83     :cond_2e
84     const-string v4, "M"
85
86     if-ne v2, v4, :cond_35
87
88     .line 15
89     add-int/lit8 v0, v0, 0x1
90
91     goto :goto_2a
92
93     .line 17
94     :cond_35
95     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
96

```

3.2

```
97     const-string v4, "Sexo so pode ser H ou M!"
98
99     invoke-virtual {v2, v4}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
100
101     goto :goto_2a
102
103     .line 20
104     :cond_3d
105     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
106
107     new-instance v3, Ljava/lang/StringBuilder;
108
109     invoke-direct {v3}, Ljava/lang/StringBuilder;-<init>()V
110
111     const-string v4, "Foram inseridos "
112
113     invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
114
115     move-result-object v3
116
117     invoke-virtual {v3, v1}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
118
119     move-result-object v1
120
121     const-string v3, " Homens"
122
123     invoke-virtual {v1, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
124
125     move-result-object v1
126
127     invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
        String;
128
129     move-result-object v1
130
131     invoke-virtual {v2, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
132
133     .line 21
134     sget-object v1, Ljava/lang/System;->out:Ljava/io/PrintStream;
135
136     new-instance v2, Ljava/lang/StringBuilder;
137
138     invoke-direct {v2}, Ljava/lang/StringBuilder;-<init>()V
139
140     const-string v3, "Foram inseridos "
141
142     invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
143
144     move-result-object v2
145
146     invoke-virtual {v2, v0}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
147
```

```

148     move-result-object v0
149
150     const-string v2, "Mulheres"
151
152     invoke-virtual {v0, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
153
154     move-result-object v0
155
156     invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
        String;
157
158     move-result-object v0
159
160     invoke-virtual {v1, v0}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
161
162     .line 22
163     return-void
164 .end method

```

Listagem 3.52: micro06.lua - Escreve um número lido por extenso

```

1 function main()
2     local numero
3     print("Digite um numero de 1 a 5: ")
4     numero = io.read("*number")
5     if numero == 1 then
6         print("Um\n")
7     elseif numero == 2 then
8         print("Dois\n")
9     elseif numero == 3 then
10        print("Tres\n")
11    elseif numero == 4 then
12        print("Quatro\n")
13    elseif numero == 5 then
14        print("Cinco\n")
15    else
16        print("Numero invalido!!!")
17    end
18 end
19
20 main()

```

Listagem 3.53: micro06.java - Escreve um número lido por extenso

```

1 public class micro06{
2     public static void main(String[] args){
3         int numero;
4         System.out.print("Digite um numero de 1 a 5: ");
5         numero = Integer.parseInt(System.console().readLine());
6         if(numero == 1){
7             System.out.println("Um");
8         }else if(numero == 2){
9             System.out.println("Dois");
10        }else if(numero == 3){
11            System.out.println("Tres");
12        }else if(numero == 4){
13            System.out.println("Quatro");

```

```

14         }else if(numero == 5){
15             System.out.println("Cinco");
16         }else{
17             System.out.println("Numero invalido!!!");
18         }
19     }
20 }

```

Listagem 3.54: micro06.smali - Escreve um número lido por extenso

```

1 .class public Lmicro06;
2 .super Ljava/lang/Object;
3 .source "micro06.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20     .prologue
21     .line 4
22     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
23
24     const-string v1, "Digite um numero de 1 a 5: "
25
26     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->print(Ljava/lang/
        String;)V
27
28     .line 5
29     invoke-static {}, Ljava/lang/System;-->console()Ljava/io/Console;
30
31     move-result-object v0
32
33     invoke-virtual {v0}, Ljava/io/Console;-->readLine()Ljava/lang/String;
34
35     move-result-object v0
36
37     invoke-static {v0}, Ljava/lang/Integer;-->parseInt(Ljava/lang/String;)I
38
39     move-result v0
40
41     .line 6
42     const/4 v1, 0x1
43
44     if-ne v0, v1, :cond_1e
45
46     .line 7
47     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
48

```

```

49     const-string v1, "Um"
50
51     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
52
53     .line 19
54     :goto_ld
55     return-void
56
57     .line 8
58     :cond_1e
59     const/4 v1, 0x2
60
61     if-ne v0, v1, :cond_29
62
63     .line 9
64     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
65
66     const-string v1, "Dois"
67
68     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
69
70     goto :goto_ld
71
72     .line 10
73     :cond_29
74     const/4 v1, 0x3
75
76     if-ne v0, v1, :cond_34
77
78     .line 11
79     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
80
81     const-string v1, "Tres"
82
83     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
84
85     goto :goto_ld
86
87     .line 12
88     :cond_34
89     const/4 v1, 0x4
90
91     if-ne v0, v1, :cond_3f
92
93     .line 13
94     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
95
96     const-string v1, "Quatro"
97
98     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
99
100    goto :goto_ld
101
102    .line 14
103    :cond_3f

```

```

104     const/4 v1, 0x5
105
106     if-ne v0, v1, :cond_4a
107
108     .line 15
109     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
110
111     const-string v1, "Cinco"
112
113     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->println(Ljava/lang/
        String;)V
114
115     goto :goto_1d
116
117     .line 17
118     :cond_4a
119     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
120
121     const-string v1, "Numero invalido!!!"
122
123     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->println(Ljava/lang/
        String;)V
124
125     goto :goto_1d
126 .end method

```

Listagem 3.55: micro07.lua - 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 numero: ")
6         numero = io.read("*n")
7         if numero > 0 then
8             print("Positivo\n")
9         else
10            if numero == 0 then
11                print("0 numero e igual a 0\n")
12            end
13            if numero < 0 then
14                print("Negativo\n")
15            end
16        end
17        print("Deseja finalizar? (S - 1): ")
18        opc = io.read("*n")
19        if opc == 1 then
20            programa = 0
21        end
22    end
23 end
24
25 main()

```

Listagem 3.56: micro07.java - Decide se os números são positivos, zeros ou negativos

```

1 public class micro07{
2     public static void main(String[] args){
3         int programa, numero, opc;

```

```

4
5     programa = 1;
6     while(programa == 1){
7         System.out.print("Digite um numero: ");
8         numero = Integer.parseInt(System.console().readLine());
9         if(numero > 0){
10            System.out.println("Positivo");
11        }else{
12            if(numero == 0){
13                System.out.println("O numero e igual a 0");
14            }
15            if(numero < 0){
16                System.out.println("Negativo");
17            }
18        }
19        System.out.print("Deseja finalizar? (S -1) ");
20        opc = Integer.parseInt(System.console().readLine());
21        if(opc == 1){
22            programa = 0;
23        }
24    }
25 }
26 }
27 }

```

Listagem 3.57: micro07.smali - Decide se os números são positivos, zeros ou negativos

```

1 .class public Lmicro07;
2 .super Ljava/lang/Object;
3 .source "micro07.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 6
19
20     .prologue
21     const/4 v1, 0x1
22
23     .line 5
24     move v0, v1
25
26     .line 6
27     :cond_2
28     :goto_2
29     if-ne v0, v1, :cond_4a
30
31     .line 7
32     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;

```


3.2

```
33
34  const-string v3, "Digite um numero: "
35
36  invoke-virtual {v2, v3}, Ljava/io/PrintStream;->print(Ljava/lang/
    String;)V
37
38  .line 8
39  invoke-static {}, Ljava/lang/System;->console()Ljava/io/Console;
40
41  move-result-object v2
42
43  invoke-virtual {v2}, Ljava/io/Console;->readLine()Ljava/lang/String;
44
45  move-result-object v2
46
47  invoke-static {v2}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
48
49  move-result v2
50
51  .line 9
52  if-lez v2, :cond_37
53
54  .line 10
55  sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
56
57  const-string v3, "Positivo"
58
59  invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/
    String;)V
60
61  .line 19
62  :cond_20
63  :goto_20
64  sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
65
66  const-string v3, "Deseja finalizar? (S -1) "
67
68  invoke-virtual {v2, v3}, Ljava/io/PrintStream;->print(Ljava/lang/
    String;)V
69
70  .line 20
71  invoke-static {}, Ljava/lang/System;->console()Ljava/io/Console;
72
73  move-result-object v2
74
75  invoke-virtual {v2}, Ljava/io/Console;->readLine()Ljava/lang/String;
76
77  move-result-object v2
78
79  invoke-static {v2}, Ljava/lang/Integer;->parseInt(Ljava/lang/String;)I
80
81  move-result v2
82
83  .line 21
84  if-ne v2, v1, :cond_2
85
86  .line 22
87  const/4 v0, 0x0
88
```

```

89     goto :goto_2
90
91     .line 12
92     :cond_37
93     if-nez v2, :cond_40
94
95     .line 13
96     sget-object v3, Ljava/lang/System; ->out:Ljava/io/PrintStream;
97
98     const-string v4, "O numero e igual a 0"
99
100    invoke-virtual {v3, v4}, Ljava/io/PrintStream; ->println(Ljava/lang/
        String;)V
101
102    .line 15
103    :cond_40
104    if-gez v2, :cond_20
105
106    .line 16
107    sget-object v2, Ljava/lang/System; ->out:Ljava/io/PrintStream;
108
109    const-string v3, "Negativo"
110
111    invoke-virtual {v2, v3}, Ljava/io/PrintStream; ->println(Ljava/lang/
        String;)V
112
113    goto :goto_20
114
115    .line 26
116    :cond_4a
117    return-void
118 .end method

```

Listagem 3.58: micro08.lua - 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 numero: ")
6         numero = io.read("*n")
7         if numero > 10 then
8             print("O numero "..numero.." e maior que 10\n")
9         else
10            print("O numero "..numero.." e menor que 10\n")
11        end
12    end
13 end
14
15 main()

```

Listagem 3.59: micro08.java - Decide se um número é maior ou menor que 10

```

1 public class micro08{
2     public static void main(String[] args){
3         int numero;
4         numero = 1;
5         while (numero != 0){
6             System.out.print("Digite um numero: ");

```

3.2

```
7         numero = Integer.parseInt(System.console().readLine());
8         if(numero > 10){
9             System.out.println("O numero " + numero + " e maior que 10
10                ");
11         }else{
12             System.out.println("O numero " + numero + " e menor que 10
13                ");
14         }
15 }
```

Listagem 3.60: micro08.smali - Decide se um número é maior ou menor que 10

```
1 .class public Lmicro08;
2 .super Ljava/lang/Object;
3 .source "micro08.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10     .prologue
11     .line 1
12     invoke-direct {p0}, Ljava/lang/Object;--><init>()V
13
14     return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 5
19
20     .prologue
21     .line 4
22     const/4 v0, 0x1
23
24     .line 5
25     :goto_1
26     if-eqz v0, :cond_58
27
28     .line 6
29     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
30
31     const-string v1, "Digite um numero: "
32
33     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->print(Ljava/lang/
34         String;)V
35
36     .line 7
37     invoke-static {}, Ljava/lang/System;-->console()Ljava/io/Console;
38
39     move-result-object v0
40
41     invoke-virtual {v0}, Ljava/io/Console;-->readLine()Ljava/lang/String;
42
43     move-result-object v0
44
45     invoke-static {v0}, Ljava/lang/Integer;-->parseInt(Ljava/lang/String;)I
```

```

45
46     move-result v0
47
48     .line 8
49     const/16 v1, 0xa
50
51     if-le v0, v1, :cond_39
52
53     .line 9
54     sget-object v1, Ljava/lang/System; ->out:Ljava/io/PrintStream;
55
56     new-instance v2, Ljava/lang/StringBuilder;
57
58     invoke-direct {v2}, Ljava/lang/StringBuilder; -><init>()V
59
60     const-string v3, "O numero "
61
62     invoke-virtual {v2, v3}, Ljava/lang/StringBuilder; ->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
63
64     move-result-object v2
65
66     invoke-virtual {v2, v0}, Ljava/lang/StringBuilder; ->append(I)Ljava/
        lang/StringBuilder;
67
68     move-result-object v2
69
70     const-string v3, " e maior que 10"
71
72     invoke-virtual {v2, v3}, Ljava/lang/StringBuilder; ->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
73
74     move-result-object v2
75
76     invoke-virtual {v2}, Ljava/lang/StringBuilder; ->toString()Ljava/lang/
        String;
77
78     move-result-object v2
79
80     invoke-virtual {v1, v2}, Ljava/io/PrintStream; ->println(Ljava/lang/
        String;)V
81
82     goto :goto_1
83
84     .line 11
85     :cond_39
86     sget-object v1, Ljava/lang/System; ->out:Ljava/io/PrintStream;
87
88     new-instance v2, Ljava/lang/StringBuilder;
89
90     invoke-direct {v2}, Ljava/lang/StringBuilder; -><init>()V
91
92     const-string v3, "O numero "
93
94     invoke-virtual {v2, v3}, Ljava/lang/StringBuilder; ->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
95
96     move-result-object v2
97

```

3.2

```
98     invoke-virtual {v2, v0}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
99
100    move-result-object v2
101
102    const-string v3, " e menor que 10"
103
104    invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
105
106    move-result-object v2
107
108    invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
        String;
109
110    move-result-object v2
111
112    invoke-virtual {v1, v2}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
113
114    goto :goto_1
115
116    .line 14
117    :cond_58
118    return-void
119 .end method
```

Listagem 3.61: micro09.lua - Cálculo de preços

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

Listagem 3.62: micro09.java - Cálculo de preços

```
1 public class micro09{
2     public static void main(String[] args){
3         double preco, venda, novo_preco;
4         novo_preco = 0;
5         System.out.print("Digite o preco: ");
6         preco = Double.parseDouble(System.console().readLine());
7         System.out.print("Digite a venda: ");
8         venda = Double.parseDouble(System.console().readLine());
9         if((venda < 500) || (preco < 30)){
```

```

10         novo_preco = preco + (10/100) * preco;
11     }else if(((venda >= 500) && venda < 1200) || ((preco >= 30) &&
        preco >= 80)){
12         novo_preco = preco + (15/100) * preco;
13     }else if ((venda >= 1200) || (preco >=80)){
14         novo_preco = preco - (20/100) * preco;
15     }
16     System.out.println("O novo preco e " + novo_preco);
17 }
18 }

```

Listagem 3.63: micro09.smali - Cálculo de preços

```

1 .class public Lmicro09;
2 .super Ljava/lang/Object;
3 .source "micro09.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 9
19
20     .prologue
21     .line 4
22     const-wide/16 v0, 0x0
23
24     .line 5
25     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
26
27     const-string v3, "Digite o preco: "
28
29     invoke-virtual {v2, v3}, Ljava/io/PrintStream;->print(Ljava/lang/
        String;)V
30
31     .line 6
32     invoke-static {}, Ljava/lang/System;->console()Ljava/io/Console;
33
34     move-result-object v2
35
36     invoke-virtual {v2}, Ljava/io/Console;->readLine()Ljava/lang/String;
37
38     move-result-object v2
39
40     invoke-static {v2}, Ljava/lang/Double;->parseDouble(Ljava/lang/String
        ;)D
41
42     move-result-wide v2
43
44     .line 7

```

3.2

```
45     sget-object v4, Ljava/lang/System;-->out:Ljava/io/PrintStream;
46
47     const-string v5, "Digite a venda: "
48
49     invoke-virtual {v4, v5}, Ljava/io/PrintStream;-->print(Ljava/lang/
        String;)V
50
51     .line 8
52     invoke-static {}, Ljava/lang/System;-->console()Ljava/io/Console;
53
54     move-result-object v4
55
56     invoke-virtual {v4}, Ljava/io/Console;-->readLine()Ljava/lang/String;
57
58     move-result-object v4
59
60     invoke-static {v4}, Ljava/lang/Double;-->parseDouble(Ljava/lang/String
        ;)D
61
62     move-result-wide v4
63
64     .line 9
65     const-wide v6, 0x407f400000000000L    # 500.0
66
67     cmpg-double v6, v4, v6
68
69     if-ltz v6, :cond_37
70
71     const-wide/high16 v6, 0x403e000000000000L    # 30.0
72
73     cmpg-double v6, v2, v6
74
75     if-gez v6, :cond_54
76
77     .line 10
78     :cond_37
79     const-wide/16 v0, 0x0
80
81     mul-double/2addr v0, v2
82
83     add-double/2addr v0, v2
84
85     .line 16
86     :cond_3b
87     :goto_3b
88     sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
89
90     new-instance v3, Ljava/lang/StringBuilder;
91
92     invoke-direct {v3}, Ljava/lang/StringBuilder;--><init>()V
93
94     const-string v4, "O novo preco e "
95
96     invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;-->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
97
98     move-result-object v3
99
100    invoke-virtual {v3, v0, v1}, Ljava/lang/StringBuilder;-->append(D)Ljava
```

```

        /lang/StringBuilder;
101
102     move-result-object v0
103
104     invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
        String;
105
106     move-result-object v0
107
108     invoke-virtual {v2, v0}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
109
110     .line 17
111     return-void
112
113     .line 11
114     :cond_54
115     const-wide v6, 0x407f400000000000L      # 500.0
116
117     cmpl-double v6, v4, v6
118
119     if-ltz v6, :cond_66
120
121     const-wide v6, 0x4092c00000000000L      # 1200.0
122
123     cmpg-double v6, v4, v6
124
125     if-ltz v6, :cond_72
126
127     :cond_66
128     const-wide/high16 v6, 0x403e000000000000L      # 30.0
129
130     cmpl-double v6, v2, v6
131
132     if-ltz v6, :cond_77
133
134     const-wide/high16 v6, 0x4054000000000000L      # 80.0
135
136     cmpl-double v6, v2, v6
137
138     if-ltz v6, :cond_77
139
140     .line 12
141     :cond_72
142     const-wide/16 v0, 0x0
143
144     mul-double/2addr v0, v2
145
146     add-double/2addr v0, v2
147
148     goto :goto_3b
149
150     .line 13
151     :cond_77
152     const-wide v6, 0x4092c00000000000L      # 1200.0
153
154     cmpl-double v4, v4, v6
155
156     if-gez v4, :cond_86

```


3.2

```
157
158     const-wide/high16 v4, 0x4054000000000000L    # 80.0
159
160     cmpl-double v4, v2, v4
161
162     if-ltz v4, :cond_3b
163
164     .line 14
165     :cond_86
166     const-wide/16 v0, 0x0
167
168     mul-double/2addr v0, v2
169
170     sub-double v0, v2, v0
171
172     goto :goto_3b
173 .end method
```

Listagem 3.64: micro10.lua - Calcula o fatorial de um número

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

Listagem 3.65: micro10.java - Calcula o fatorial de um número

```
1 public class micro10{
2     public static void main(String[] args){
3         int numero, fat;
4         System.out.print("Digite o numero: ");
5         numero = Integer.parseInt(System.console().readLine());
6         fat = fatorial(numero);
7         System.out.print("O fatorial de ");
8         System.out.print(numero);
9         System.out.print(" e ");
10        System.out.print(fat);
11    }
12    public static int fatorial(int n){
13        if(n <= 0){
14            return 1;
15        }else{
16            return (n * fatorial(n-1));
17        }
18    }
```

```

19     }
20 }

```

Listagem 3.66: micro10.smali - Calcula o fatorial de um número

```

1 .class public Lmicro10;
2 .super Ljava/lang/Object;
3 .source "micro10.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object; -> <init>()V
13
14    return-void
15 .end method
16
17 .method public static fatorial(I)I
18     .registers 2
19
20     .prologue
21     .line 13
22     if-gtz p0, :cond_4
23
24     .line 14
25     const/4 v0, 0x1
26
27     .line 16
28     :goto_3
29     return v0
30
31     :cond_4
32     add-int/lit8 v0, p0, -0x1
33
34     invoke-static {v0}, Lmicro10; -> fatorial(I)I
35
36     move-result v0
37
38     mul-int/2addr v0, p0
39
40     goto :goto_3
41 .end method
42
43 .method public static main([Ljava/lang/String;)V
44     .registers 5
45
46     .prologue
47     .line 4
48     sget-object v0, Ljava/lang/System; -> out:Ljava/io/PrintStream;
49
50     const-string v1, "Digite o numero: "
51
52     invoke-virtual {v0, v1}, Ljava/io/PrintStream; -> print(Ljava/lang/
        String;)V
53

```

3.2

```
54 .line 5
55 invoke-static {}, Ljava/lang/System;-->console()Ljava/io/Console;
56
57 move-result-object v0
58
59 invoke-virtual {v0}, Ljava/io/Console;-->readLine()Ljava/lang/String;
60
61 move-result-object v0
62
63 invoke-static {v0}, Ljava/lang/Integer;-->parseInt(Ljava/lang/String;)I
64
65 move-result v0
66
67 .line 6
68 invoke-static {v0}, Lmicro10;-->fatorial(I)I
69
70 move-result v1
71
72 .line 7
73 sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
74
75 const-string v3, "O fatorial de "
76
77 invoke-virtual {v2, v3}, Ljava/io/PrintStream;-->print(Ljava/lang/
    String;)V
78
79 .line 8
80 sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
81
82 invoke-virtual {v2, v0}, Ljava/io/PrintStream;-->print(I)V
83
84 .line 9
85 sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
86
87 const-string v2, " e "
88
89 invoke-virtual {v0, v2}, Ljava/io/PrintStream;-->print(Ljava/lang/
    String;)V
90
91 .line 10
92 sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
93
94 invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->print(I)V
95
96 .line 11
97 return-void
98 .end method
```

Listagem 3.67: micro11.lua - Decide se um número é positivo, zero ou negativo com auxílio de uma função

```
1 function main()
2     local numero, x
3     print("Digite um numero ")
4     numero = io.read("*n")
5     x = verifica(numero)
6     if x == 1 then
7         print("Numero positivo\n")
8     elseif x == 0 then
```

```

9      print("Zero\n")
10     else
11         print("Numero negativo\n")
12     end
13 end
14
15 function verifica(n)
16     local res
17     if n > 0 then
18         res = 1
19     elseif n < 0 then
20         res = -1
21     else
22         res = 0
23     end
24     return res
25 end
26
27 main()

```

Listagem 3.68: micro11.java - Decide se um número é positivo, zero ou negativo com auxílio de uma função

```

1 public class micro11{
2     public static void main(String[] args){
3         int numero, x;
4         System.out.print("Digite um numero: ");
5         numero = Integer.parseInt(System.console().readLine());
6         x = verifica(numero);
7         if(x == 1){
8             System.out.println("Numero positivo");
9         }else if(x == 0){
10             System.out.println("Zero");
11         }else{
12             System.out.println("Numero negativo");
13         }
14     }
15     public static int verifica(int n){
16         int res;
17         if(n > 0){
18             res = 1;
19         }else if (n < 0){
20             res = -1;
21         }else{
22             res = 0;
23         }
24         return res;
25     }
26 }

```

Listagem 3.69: micro11.smali - Decide se um número é positivo, zero ou negativo com auxílio de uma função

```

1 .class public Lmicro11;
2 .super Ljava/lang/Object;
3 .source "micro11.java"
4
5
6 # direct methods

```

3.2

```
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;--><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20     .prologue
21     .line 4
22     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
23
24     const-string v1, "Digite um numero: "
25
26     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->print(Ljava/lang/
        String;)V
27
28     .line 5
29     invoke-static {}, Ljava/lang/System;-->console()Ljava/io/Console;
30
31     move-result-object v0
32
33     invoke-virtual {v0}, Ljava/io/Console;-->readLine()Ljava/lang/String;
34
35     move-result-object v0
36
37     invoke-static {v0}, Ljava/lang/Integer;-->parseInt(Ljava/lang/String;)I
38
39     move-result v0
40
41     .line 6
42     invoke-static {v0}, Lmicroll;-->verifica(I)I
43
44     move-result v0
45
46     .line 7
47     const/4 v1, 0x1
48
49     if-ne v0, v1, :cond_22
50
51     .line 8
52     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
53
54     const-string v1, "Numero positivo"
55
56     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->println(Ljava/lang/
        String;)V
57
58     .line 14
59     :goto_21
60     return-void
61
62     .line 9
63     :cond_22
```

```

64     if-nez v0, :cond_2c
65
66     .line 10
67     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
68
69     const-string v1, "Zero"
70
71     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
72
73     goto :goto_21
74
75     .line 12
76     :cond_2c
77     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
78
79     const-string v1, "Numero negativo"
80
81     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
82
83     goto :goto_21
84 .end method
85
86 .method public static verifica(I)I
87     .registers 2
88
89     .prologue
90     .line 17
91     if-lez p0, :cond_4
92
93     .line 18
94     const/4 v0, 0x1
95
96     .line 24
97     :goto_3
98     return v0
99
100    .line 19
101    :cond_4
102    if-gez p0, :cond_8
103
104    .line 20
105    const/4 v0, -0x1
106
107    goto :goto_3
108
109    .line 22
110    :cond_8
111    const/4 v0, 0x0
112
113    goto :goto_3
114 .end method

```

Capítulo 4

Analizador léxico

Neste capítulo teremos a implementação de um analisador léxico para a linguagem MiniLua. Ela será feita com a linguagem Ocaml, que possui uma ferramenta bastante útil para tal fim, a Ocamllex.

Essa etapa é a primeira no processo de compilação. Sua função é facilitar o trabalho da próxima etapa, a análise sintática, lendo o código e convertendo-o em uma sequência de *tokens*, que são uma espécie de identificadores da estrutura de um programa. Dessa forma, temos *tokens* para simbolizar palavras reservadas da linguagem, variáveis (e seus nomes), valores literais e seus tipos, fim de linha e outros caracteres parte da linguagem. Além disso, o analisador léxico já elimina elementos decorativos, como comentários e espaços em branco.

4.1 Reconhecendo os Tokens

Seguem os *tokens* criados para a linguagem MiniLua na tabela

Tabela 4.1: *Tokens*

Tipo	Representação	Tipo	Representação
AND	<i>and</i>	BREAK	<i>break</i>
DO	<i>do</i>	ELSE	<i>else</i>
ELSEIF	<i>elseif</i>	EOF	<i>representa fim do arquivo</i>
END	<i>end</i>	FALSE	<i>false</i>
FOR	<i>for</i>	FUNCAO	<i>function</i>
IF	<i>if</i>	IN	<i>in</i>
IO_READ	<i>io.read</i>	LOCAL	<i>local</i>
NIL	<i>nil</i>	NOT	<i>not</i>
NUMBER_INPUT	<i>*number</i>	OR	<i>or</i>
PRINT	<i>print</i>	REPEAT	<i>repeat</i>
RETURN	<i>return</i>	THEN	<i>then</i>
TRUE	<i>true</i>	UNTIL	<i>until</i>
WHILE	<i>while</i>	ABRE_CHAVE	<i>{</i>

Continued on next page

Tabela 4.1 – continued from previous page

Tipo	Representação	Tipo	Representação
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	,
LITINT of int	<i>dígitos</i>	LITSTRING of string	<i>"expressao"</i>
ID of string	<i>ex: variavel_nome</i>		

4.2 Montagem do Analisador Léxico

4.2.1 Código do Analisador Léxico

Segue o código do analisador léxico para a linguagem MiniLua. Um sistema de anotação de tipos (inexistente na linguagem atualmente) próprio deste projeto será adicionado futuramente para facilitar o desenvolvimento do restante do compilador.

Listagem 4.1: lexico.mll - Gera tokens dado código da linguagem LUA

```

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    }
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 =

```


4.2

```
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 rule token = parse
93     brancos                { token lexbuf }
94 | novalinha                { incr_num_linha lexbuf; token lexbuf }
95 | "--[" {
96     let pos = lexbuf.lex_curr_p in
97     let lin = pos.pos_lnum
98     and col = pos.pos_cnum - pos.pos_bol - 1 in
99     comentario_bloco lin col lexbuf }
100 | "--"                    { comentario_linha lexbuf }
101 | "("                      { APAR }
102 | "{"                      { ABRE_CHAVE }
103 | "["                      { ABRE_COLCHETE }
104 | "+"                      { ADICAO }
105 | "-"                      { SUBTRACAO }
106 | ")"                      { FPAR }
107 | "}"                      { FECHA_CHAVE }
108 | "]"                      { FECHA_COLCHETE }
109 | ","                      { VIRGULA }
110 | "."                      { PONTO }
111 | ";"                      { PONTO_VIRGULA }
112 | ":"                      { DOIS_PONTOS }
113 | "=="                     { EQUIVALENTE }
114 | "~="                     { NAO_EQUIVALENTE }
115 | ">="                     { MAIOR_OU_IGUAL }
116 | "<="                     { MENOR_OU_IGUAL }
117 | "/"                      { DIVISAO }
118 | "*"                      { MULTIPLICACAO }
119 | "%"                      { MODULO }
120 | "^"                      { EXPONENCIACAO }
121 | ">"                      { MAIOR }
122 | "<"                      { MENOR }
123 | "="                      { ATRIB }
124 | "#"                      { TAMANHO }
125 | "<<"                     { MULT_POR_2 }
126 | ">>"                     { DIV_POR_2 }
127 | "//"                     { DIVISAO_INTEIRO }
128 | "&"                      { AND_BINARIO }
129 | "|"                      { OR_BINARIO }
130 | ".."                     { CONCATENA }
131 | "..."                    { RETICENCIAS }
132 | "and"                    { AND }
133 | "break"                  { BREAK }
134 | "do"                     { DO }
135 | "else"                    { ELSE }
136 | "elseif"                 { ELSEIF }
137 | "end"                    { END }

```

```

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

```

Além do analisador léxico, usaremos um outro programa chamado *carregador.ml*. A função desse programa é automatizar o processo de passar os códigos do arquivo em lua para o programa lexico compilado no formato .cmo, pois nosso analisador léxico por si só não lê o arquivo como um todo, mas palavra por palavra no arquivo de entrada (palavra aqui no

sentido de palavra da linguagem regular que forma a linguagem de programação LUA do arquivo). Segue o código usado pelo *carregador.ml*

Listagem 4.2: carregador.ml - Programa auxiliar para o analisador léxico

```

1 #load "lexico.cmo";;
2
3 let rec tokens lexbuf =
4   let tok = Lexico.token lexbuf in
5   match tok with
6   | Lexico.EOF -> [Lexico.EOF]
7   | _ -> tok :: tokens lexbuf
8 ;;
9
10 let lexico str =
11   let lexbuf = Lexing.from_string str in
12   tokens lexbuf
13 ;;
14
15 let lex arq =
16   let ic = open_in arq in
17   let lexbuf = Lexing.from_channel ic in
18   let toks = tokens lexbuf in
19   let _ = close_in ic in
20   toks

```

4.2.2 Utilização

O código descrito na listagem 6.6 acima nada mais é que um conjunto de expressões regulares e instruções para podermos montar um autômato finito determinístico, usado para fazer o reconhecimento da linguagem regular dos *tokens*. Para criarmos tal autômato, utilizamos de uma extensão do ocaml, chamada ocamllex. Essa ferramenta faz essa tradução das expressões regulares no arquivo .mll para o autômato finito determinístico em um arquivo de saída .ml que pode, então, ser utilizado para produzir os tokens de um arquivo em Lua.

Começamos traduzindo o arquivo .mll para o arquivo .ml por meio do ocamllex rodando o código a seguir no local onde o arquivo se encontra

```
$ ocamllex lexico.mll
```

Isso gerará o arquivo .ml que deverá ser, então, compilado para produzir os formatos .cmi e .cmo da seguinte forma

```
$ ocamlc -c lexico.ml
```

O arquivo *lexico.cmo* gerado após a compilação será usado por nosso programa auxiliar *carregador.ml* 4.2. Para fazermos isso, entramos no interpretador do ocaml por meio do seguinte código

```
$ rlwrap ocaml
```

Para então importarmos o *carregador.ml* da seguinte maneira

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

Agora podemos utilizar a funcionalidade em *carregador.ml*. Faremos isso chamando a função *lex* e passando como argumento o arquivo *.lua* a ser traduzido em tokens pelo analisador léxico.

```
# lex "arquivo.lua";;
```

Onde *arquivo.lua* é o nome do arquivo de onde queremos obter os tokens. Este código terá como resultado a lista de tokens do arquivo.

4.3 Testando o Analisador Léxico

A seguir serão apresentadas as saídas obtidas ao passarmos nossos programas nano e micro pelo nosso analisador léxico.

4.3.1 Programas Nano

Os resultados de passar os programas nano pelo analisador são os seguintes

nano01

Listagem 4.3: Resultado de passar o analisador léxico no programa nano01.lua

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

nano02

Listagem 4.4: Resultado de passar o analisador léxico no programa nano02.lua

```
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

Listagem 4.5: Resultado de passar o analisador léxico no programa nano03.lua

```
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

Listagem 4.6: Resultado de passar o analisador léxico no programa nano04.lua

```
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

Listagem 4.7: Resultado de passar o analisador léxico no programa nano05.lua

```

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

Listagem 4.8: Resultado de passar o analisador léxico no programa nano06.lua

```

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

Listagem 4.9: Resultado de passar o analisador léxico no programa nano07.lua

```

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

Listagem 4.10: Resultado de passar o analisador léxico no programa nano08.lua

```

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

Listagem 4.11: Resultado de passar o analisador léxico no programa nano09.lua

```

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

```

nano10

Listagem 4.12: Resultado de passar o analisador léxico no programa nano10.lua

```

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.
  VIRGULA;
4 Lexico.ID "m"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.VIRGULA;
5 Lexico.LITINT 2; Lexico.IF; Lexico.ID "n"; Lexico.EQUIVALENTE;
6 Lexico.ID "m"; 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]

```

nano11

Listagem 4.13: Resultado de passar o analisador léxico no programa nano11.lua

```

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 "
  x";
4 Lexico.ID "n"; Lexico.VIRGULA; Lexico.ID "m"; Lexico.VIRGULA; Lexico.ID "
  x";
5 Lexico.ATRIB; Lexico.LITINT 1; Lexico.VIRGULA; Lexico.LITINT 2;
6 Lexico.VIRGULA; Lexico.LITINT 5; Lexico.WHILE; Lexico.ID "x"; Lexico.
  MAIOR;
7 Lexico.ID "n"; Lexico.DO; Lexico.ID "n"; Lexico.ATRIB; Lexico.ID "n";
8 Lexico.ADICAO; Lexico.ID "m"; Lexico.PRINT; Lexico.APAR; Lexico.ID "n";
9 Lexico.FPAR; Lexico.END; Lexico.END; Lexico.ID "main"; Lexico.APAR;
10 Lexico.FPAR; Lexico.EOF]

```

nano12

Listagem 4.14: Resultado de passar o analisador léxico no programa nano12.lua

```

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 "
  x";
4 Lexico.ID "n"; Lexico.VIRGULA; Lexico.ID "m"; Lexico.VIRGULA; Lexico.ID "
  x";

```

```

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

```

4.3.2 Programas Micro

Os resultados de passar os programas micro pelo analisador são os seguintes

micro01

Listagem 4.15: Resultado de passar o analisador léxico no programa micro01.lua

```

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 conversao: Celsius -> Fahrenheit\n";
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 e: "; 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

Listagem 4.16: Resultado de passar o analisador léxico no programa micro02.lua

```

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;

```


4.3

```
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

Listagem 4.17: Resultado de passar o analisador léxico no programa micro03.lua

```
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 numero: "; Lexico.FPAR; Lexico.ID "numero";
5 Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR; Lexico.LITSTRING "*number";
6 Lexico.FPAR; Lexico.IF; Lexico.ID "numero"; Lexico.MAIOR_OU_IGUAL;
7 Lexico.LITINT 100; Lexico.THEN; Lexico.IF; Lexico.ID "numero";
8 Lexico.MENOR_OU_IGUAL; Lexico.LITINT 200; Lexico.THEN; Lexico.PRINT;
9 Lexico.APAR;
10 Lexico.LITSTRING "O numero esta no intervalo entre 100 e 200\n";
11 Lexico.FPAR; Lexico.ELSE; Lexico.PRINT; Lexico.APAR;
12 Lexico.LITSTRING "O numero nao esta no intervalo entre 100 e 200\n";
13 Lexico.FPAR; Lexico.END; Lexico.ELSE; Lexico.PRINT; Lexico.APAR;
14 Lexico.LITSTRING "O numero nao esta no intervalo entre 100 e 200\n";
15 Lexico.FPAR; Lexico.END; Lexico.END; Lexico.ID "main"; Lexico.APAR;
16 Lexico.FPAR; Lexico.EOF]
```

micro04

Listagem 4.18: Resultado de passar o analisador léxico no programa micro04.lua

```
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
  0;
5 Lexico.FOR; Lexico.ID "x"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.VIRGULA;
6 Lexico.LITINT 5; Lexico.VIRGULA; Lexico.LITINT 1; Lexico.DO; Lexico.PRINT
  ;
7 Lexico.APAR; Lexico.LITSTRING "Digite um numero: "; Lexico.FPAR;
8 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 " numeros no intervalo entre 10 e 150"; Lexico.FPAR;
17 Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]
```

micro05

Listagem 4.19: Resultado de passar o analisador léxico no programa micro05.lua

```
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.VIRGULA;
4 Lexico.ID "x"; Lexico.VIRGULA; Lexico.ID "h"; Lexico.VIRGULA; Lexico.ID "
  m";
5 Lexico.ID "h"; Lexico.VIRGULA; Lexico.ID "m"; Lexico.ATRIB; 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 so pode ser H ou M!\n"; 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\n"; Lexico.FPAR; Lexico.PRINT; Lexico.APAR;
24 Lexico.LITSTRING "Foram inseridos "; Lexico.CONCATENA; Lexico.ID "m";
25 Lexico.CONCATENA; Lexico.LITSTRING " mulheres\n"; Lexico.FPAR; Lexico.END
  ;
26 Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

micro06

Listagem 4.20: Resultado de passar o analisador léxico no programa micro06.lua

```

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 numero de 1 a 5: "; 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.EQUIVALENTE; Lexico.LITINT 1; Lexico.THEN; Lexico.PRINT; Lexico.
  APAR;
8 Lexico.LITSTRING "Um\n"; Lexico.FPAR; Lexico.ELSEIF; Lexico.ID "numero";
9 Lexico.EQUIVALENTE; Lexico.LITINT 2; Lexico.THEN; Lexico.PRINT; Lexico.
  APAR;
10 Lexico.LITSTRING "Dois\n"; Lexico.FPAR; Lexico.ELSEIF; Lexico.ID "numero"
  ;
11 Lexico.EQUIVALENTE; Lexico.LITINT 3; Lexico.THEN; Lexico.PRINT; Lexico.
  APAR;
12 Lexico.LITSTRING "Tres\n"; Lexico.FPAR; Lexico.ELSEIF; Lexico.ID "numero"
  ;
13 Lexico.EQUIVALENTE; Lexico.LITINT 4; Lexico.THEN; Lexico.PRINT; Lexico.
  APAR;
14 Lexico.LITSTRING "Quatro\n"; Lexico.FPAR; Lexico.ELSEIF; Lexico.ID "
  numero";
15 Lexico.EQUIVALENTE; Lexico.LITINT 5; Lexico.THEN; Lexico.PRINT; Lexico.

```

```

    APAR;
16 Lexico.LITSTRING "Cinco\n"; 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

Listagem 4.21: Resultado de passar o analisador léxico no programa micro07.lua

```

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; Lexico.LITSTRING "Digite um numero:
    ";
7 Lexico.FPAR; 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 0; Lexico.THEN; Lexico.PRINT; Lexico.APAR;
10 Lexico.LITSTRING "Positivo\n"; Lexico.FPAR; Lexico.ELSE; Lexico.IF;
11 Lexico.ID "numero"; Lexico.EQUIVALENTE; Lexico.LITINT 0; Lexico.THEN;
12 Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "O numero e igual a 0\n";
13 Lexico.FPAR; Lexico.END; Lexico.IF; Lexico.ID "numero"; Lexico.MENOR;
14 Lexico.LITINT 0; Lexico.THEN; Lexico.PRINT; Lexico.APAR;
15 Lexico.LITSTRING "Negativo\n"; Lexico.FPAR; Lexico.END; Lexico.END;
16 Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "Deseja finalizar? (S - 1): "
    ;
17 Lexico.FPAR; Lexico.ID "opc"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
18 Lexico.LITSTRING "*n"; Lexico.FPAR; Lexico.IF; Lexico.ID "opc";
19 Lexico.EQUIVALENTE; Lexico.LITINT 1; Lexico.THEN; Lexico.ID "programa";
20 Lexico.ATRIB; Lexico.LITINT 0; Lexico.END; Lexico.END; Lexico.END;
21 Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

micro08

Listagem 4.22: Resultado de passar o analisador léxico no programa micro08.lua

```

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; Lexico.LITSTRING "Digite um numero:
    ";
6 Lexico.FPAR; Lexico.ID "numero"; Lexico.ATRIB; Lexico.IO_READ; Lexico.
    APAR;
7 Lexico.LITSTRING "*n"; Lexico.FPAR; Lexico.IF; Lexico.ID "numero";
8 Lexico.MAIOR; Lexico.LITINT 10; Lexico.THEN; Lexico.PRINT; Lexico.APAR;
9 Lexico.LITSTRING "O numero "; Lexico.CONCATENA; Lexico.ID "numero";
10 Lexico.CONCATENA; Lexico.LITSTRING " e maior que 10\n"; Lexico.FPAR;
11 Lexico.ELSE; Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "O numero ";
12 Lexico.CONCATENA; Lexico.ID "numero"; Lexico.CONCATENA;
13 Lexico.LITSTRING " e menor que 10\n"; Lexico.FPAR; Lexico.END; Lexico.END
    ;
14 Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

micro09

Listagem 4.23: Resultado de passar o analisador léxico no programa micro09.lua

```

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 preco: "; 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.
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.ELSEIF
14 Lexico.APAR; Lexico.ID "venda"; Lexico.MAIOR_OU_IGUAL; Lexico.LITINT 500;
15 Lexico.AND; Lexico.ID "venda"; Lexico.MENOR_OU_IGUAL; Lexico.LITINT 1200;
16 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;
20 Lexico.LITINT 15; Lexico.DIVISAO; Lexico.LITINT 100; Lexico.MULTIPLICACAO
21 Lexico.ID "preco"; Lexico.ELSEIF; Lexico.ID "venda"; Lexico.
22 Lexico.LITINT 1200; Lexico.OR; Lexico.ID "preco"; Lexico.MAIOR_OU_IGUAL;
23 Lexico.LITINT 80; Lexico.THEN; Lexico.ID "novo_preco"; Lexico.ATRIB;
24 Lexico.ID "preco"; Lexico.SUBTRACAO; Lexico.LITINT 20; Lexico.DIVISAO;
25 Lexico.LITINT 100; Lexico.MULTIPLICACAO; Lexico.ID "preco"; Lexico.END;
26 Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "O novo preco e ";
27 Lexico.CONCATENA; Lexico.ID "novo_preco"; Lexico.CONCATENA;
28 Lexico.LITSTRING "\n"; Lexico.FPAR; Lexico.END; Lexico.ID "main";
29 Lexico.APAR; Lexico.FPAR; Lexico.EOF]

```

micro10

Listagem 4.24: Resultado de passar o analisador léxico no programa micro10.lua

```

1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "numero"; Lexico.VIRGULA; Lexico.ID "fat"; Lexico.PRINT;
4 Lexico.APAR; Lexico.LITSTRING "Digite um numero: "; Lexico.FPAR;
5 Lexico.ID "numero"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
6 Lexico.LITSTRING "*n"; Lexico.FPAR; Lexico.ID "fat"; Lexico.ATRIB;
7 Lexico.ID "fatorial"; Lexico.APAR; Lexico.ID "numero"; Lexico.FPAR;
8 Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "O fatorial de ";
9 Lexico.CONCATENA; Lexico.ID "numero"; Lexico.CONCATENA;
10 Lexico.LITSTRING " e "; Lexico.CONCATENA; Lexico.ID "fat"; Lexico.
11 Lexico.LITSTRING "\n"; Lexico.FPAR; Lexico.END; Lexico.FUNCAO;
12 Lexico.ID "fatorial"; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.IF;
13 Lexico.ID "n"; Lexico.MENOR_OU_IGUAL; Lexico.LITINT 0; Lexico.THEN;

```

4.4

```
14 Lexico.RETURN; Lexico.LITINT 1; Lexico.ELSE; Lexico.RETURN; Lexico.ID "n"
    ;
15 Lexico.MULTIPLICACAO; Lexico.ID "fatorial"; Lexico.APAR; Lexico.ID "n";
16 Lexico.SUBTRACAO; Lexico.LITINT 1; Lexico.FPAR; Lexico.END; Lexico.END;
17 Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]
```

micro11

Listagem 4.25: Resultado de passar o analisador léxico no programa micro11.lua

```
1 - : Lexico.tokens list =
2 [Lexico.FUNCAO; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.LOCAL;
3 Lexico.ID "numero"; Lexico.VIRGULA; Lexico.ID "x"; Lexico.PRINT;
4 Lexico.APAR; Lexico.LITSTRING "Digite um numero "; Lexico.FPAR;
5 Lexico.ID "numero"; Lexico.ATRIB; Lexico.IO_READ; Lexico.APAR;
6 Lexico.LITSTRING "*n"; Lexico.FPAR; Lexico.ID "x"; Lexico.ATRIB;
7 Lexico.ID "verifica"; Lexico.APAR; Lexico.ID "numero"; Lexico.FPAR;
8 Lexico.IF; Lexico.ID "x"; Lexico.EQUIVALENTE; Lexico.LITINT 1; Lexico.
    THEN;
9 Lexico.PRINT; Lexico.APAR; Lexico.LITSTRING "Numero positivo\n";
10 Lexico.FPAR; Lexico.ELSEIF; Lexico.ID "x"; Lexico.EQUIVALENTE;
11 Lexico.LITINT 0; Lexico.THEN; Lexico.PRINT; Lexico.APAR;
12 Lexico.LITSTRING "Zero\n"; Lexico.FPAR; Lexico.ELSE; Lexico.PRINT;
13 Lexico.APAR; Lexico.LITSTRING "Numero negativo\n"; Lexico.FPAR; Lexico.
    END;
14 Lexico.END; Lexico.FUNCAO; Lexico.ID "verifica"; Lexico.APAR; Lexico.ID "
    n";
15 Lexico.FPAR; Lexico.LOCAL; Lexico.ID "res"; Lexico.IF; Lexico.ID "n";
16 Lexico.MAIOR; Lexico.LITINT 0; Lexico.THEN; Lexico.ID "res"; Lexico.ATRIB
    ;
17 Lexico.LITINT 1; Lexico.ELSEIF; Lexico.ID "n"; Lexico.MENOR;
18 Lexico.LITINT 0; Lexico.THEN; Lexico.ID "res"; Lexico.ATRIB;
19 Lexico.SUBTRACAO; Lexico.LITINT 1; Lexico.ELSE; Lexico.ID "res";
20 Lexico.ATRIB; Lexico.LITINT 0; Lexico.END; Lexico.RETURN; Lexico.ID "res"
    ;
21 Lexico.END; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.EOF]
```

4.4 Teste de Erros

Nesta seção serão exibidos alguns erros léxicos que podem ocorrer e a resposta devolvida pelo analisador léxico. Detalhe na resposta do erro ele indica a linha e a coluna onde o erro foi encontrado.

Comentário de bloco não fechado

Arquivo de entrada

Listagem 4.26: micro05Erro.lua - Lê strings e caracteres - com erro proposital de não fechar um comentário de bloco

```
1 function main()
2     local nome, sexo, x, h, m
3     h, m = 0, 0
4     for x = 1, 5, 1
```

```

5      do
6          print("Digite o nome: ")
7          nome = io.read("*line")
8          print("H - Homem ou M - Mulher: ")
9          sexo = io.read("*line")
10         --[[if sexo == "H" then
11             h = h + 1
12         elseif sexo == "M" then
13             m = m + 1
14         else
15             print("Sexo so pode ser H ou M!\n")
16         end
17     end
18     print("Foram inseridos "..h.." homens\n")
19     print("Foram inseridos "..m.." mulheres\n")
20 end
21
22 main()

```

Resultado do analisador

Listagem 4.27: Resultado de passar o analisador léxico no programa micro05Erro.lua

```
1 Exception: Failure "10-11: Comentario nao fechado".
```

Carácter invalido

Arquivo de entrada

Listagem 4.28: micro06Erro.lua - Escreve um número lido por extenso - com erro proposital de adicionar carácter inválido

```

1 function main()
2     local numero
3     print("Digite um numero de 1 a 5: ")
4     numero = io.read("*number")
5     if numero == 1 then
6         print("Um\n")
7     elseif numero == 2 then
8         print("Dois\n")
9     elseif numero == 3 then
10        print("Tres\n")
11    elseif numero == 4 then
12        print("Quatro\n")
13    elseif numero == 5 then
14        print("Cinco\n")
15    else
16        @
17        print("Numero invalido!!!")
18    end
19 end
20
21 main()

```

Resultado do analisador

Listagem 4.29: Resultado de passar o analisador léxico no programa micro06Erro.lua

```
1 Exception: Failure "16-1: caracter desconhecido @".
```

String com aspas não fechada corretamente

Arquivo de entrada

Listagem 4.30: micro07.lua - 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 numero: ")
6         numero = io.read("*n")
7         if numero > 0 then
8             print("Positivo\n")
9         else
10            if numero == 0 then
11                print("O numero e igual a 0\n")
12            end
13            if numero < 0 then
14                print("Negativo\n")
15            end
16        end
17        print("Deseja finalizar? (S - 1): ")
18        opc = io.read("*n")
19        if opc == 1 then
20            programa = 0
21        end
22    end
23 end
24
25 main()
```

Resultado do analisador

Listagem 4.31: Resultado de passar o analisador léxico no programa micro07Erro.lua

```
1 Exception: Failure "14-22: A string nao foi fechada".
```

Capítulo 5

Analizador Sintático

O analisador sintático tem o dever de construir uma árvore gramatical para uma dada sentença de entrada. Caso uma dada sentença não pertença a gramática que o analisador está verificando, ele dará, então, uma indicação de erro. Dessa forma, neste capítulo será mostrado a implementação de tal analisador para a linguagem MiniLua, adaptada para permitir tipagem, feita com a linguagem Ocaml. Dessa forma, os programas nanos e micros do capítulo 3 foram reescritos de forma a acomodar essa adaptação, que facilitará as próximas etapas do projeto. Além disso, como a definição da gramática agora está no analisador sintático, não é mais necessária que a mesma esteja duplicada no analisador léxico, portanto reescreveu-se o mesmo afim de simplificá-lo e tornar o projeto mais eficiente de forma geral. Por fim, faz-se a análise sintática de todos esses programas exemplos em lua e alguns testes de erros para confirmar que o analisador sintático está em bom funcionamento.

5.1 Código do Analisador Sintático

Segue o código do analisador sintático para a linguagem MiniLua utilizada nesse trabalho

Listagem 5.1: sintatico.mly - Código com a gramática do analisador sintático

```
1 %{
2   open Ast
3 %}
4
5 %token <string>    ID
6 %token <string>    LITSTRING
7 %token <int>       LITINT
8 %token <bool>      BOOL
9 %token             ABRE_CHAVE
10 %token             ABRE_COLCHETE
11 %token             FECHA_CHAVE
12 %token             FECHA_COLCHETE
13 %token             ADICAO
14 %token             AND
15 %token             AND_BINARIO
16 %token             APAR
17 %token             ATRIB
18 %token             BREAK
```


5.1

19 %token	CONCATENA
20 %token	DIV_POR_2
21 %token	DIVISAO
22 %token	DIVISAO_INTEIRO
23 %token	DO
24 %token	DOIS_PONTOS
25 %token	ELSE
26 %token	ELSEIF
27 %token	END
28 %token	EQUIVALENTE
29 %token	EXPONENCIACAO
30 %token	FOR
31 %token	FPAR
32 %token	FUNCAO
33 %token	IF
34 %token	IN
35 %token	IO_READ
36 %token	LOCAL
37 %token	MAIOR
38 %token	MAIOR_OU_IGUAL
39 %token	MENOR
40 %token	MENOR_OU_IGUAL
41 %token	MODULO
42 %token	MULT_POR_2
43 %token	MULTIPLICACAO
44 %token	NAO_EQUIVALENTE
45 %token	NIL
46 %token	NOT
47 %token	NUMBER_INPUT
48 %token	OR
49 %token	OR_BINARIO
50 %token	OR_BINARIO_EXCLUSIVO
51 %token	PONTO
52 %token	PONTO_VIRGULA
53 %token	PRINT
54 %token	RETICENCIAS
55 %token	RETURN
56 %token	SUBTRACAO
57 %token	TAMANHO
58 %token	TIPO_BOOLEAN
59 %token	TIPO_INT
60 %token	TIPO_STRING
61 %token	THEN
62 %token	UNTIL
63 %token	VIRGULA
64 %token	WHILE
65 %token	EOF
66 %token	FALSE
67 %token	TRUE
68 %token	REPEAT
69	
70 %left	OR
71 %left	AND
72 %left	MAIOR MENOR MENOR_OU_IGUAL MAIOR_OU_IGUAL EQUIVALENTE NAO_EQUIVALENTE
73 %left	OR_BINARIO
74 %left	OR_BINARIO_EXCLUSIVO
75 %left	AND_BINARIO
76 %left	MULT_POR_2 DIV_POR_2

```

77 %left          CONCATENA
78 %left          ADICAO SUBTRACAO
79 %left          MULTIPLICACAO DIVISAO DIVISAO_INTEIRO MODULO
80 %left          NOT TAMANHO
81 %left          EXPONENCIACAO
82
83 %start <Ast.programa> programa
84
85 %%
86
87 programa: f = funcoes+
88     EOF { Programa (f) }
89
90 funcoes:
91     | FUNCAO tipo=tipo_simples id=ID APAR args=argumentos* FPAR ds=
92       declaracao* cs=comando*
93     (*ret=retorno*)
94     END { Funcao (tipo, id, args, ds, cs(*,ret*)) }
95     ;
96
97 argumentos:
98     | t = tipo_simples id = ID { Args (t, id) }
99     ;
100
101 declaracao:
102     | t=tipo v=variavel {DecVar (t,v)}
103     | LOCAL t=tipo v=variavel { DecVar (t,v) }
104     ;
105
106 tipo: t = tipo_simples { t }
107
108 tipo_simples: TIPO_INT          { TipoInt }
109               | TIPO_STRING     { TipoString }
110               | TIPO_BOOLEAN    { TipoBool  }
111
112 comando: c = comando_atribuicao { c }
113          | c = comando_if       { c }
114          | c = comando_for      { c }
115          | c = comando_while    { c }
116          | c = comando_print    { c }
117          | c = comando_scan     { c }
118          | c = comando_funcao   { c }
119          | c = comando_retorno  { c }
120
121 comando_atribuicao:
122     | v = variavel ATRIB e = expressao { CmdAtrib (v,e) }
123     | v = variavel ATRIB id = ID APAR args = ID* FPAR {
124       CmdAtribRetorno (v, id, args) }
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_for:

```

5.1

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

```

190
191 variavel:
192   | x=ID      { VarSimples  x }

```

5.2 Código da Árvore Sintática Abstrata

Segue o código da árvore sintática abstrata, utilizada para construir a árvore sintática dada uma entrada

Listagem 5.2: ast.ml - Código da árvore sintática abstrata

```

1 type identificador = string
2 type programa      = Programa of funcoes list
3
4 and funcoes = Funcao of tipo * identificador * argumentos list *
   declaracoes * comandos (** retorno*)
5
6 and argumentos = Args of tipo * identificador
7
8 and declaracoes = declaracao list
9
10 and declaracao = DecVar of tipo * variavel
11
12 and comandos = comando list
13
14 and tipo =   TipoInt
15             | TipoString
16             | TipoBool
17
18 and comando = CmdAtrib of variavel * expressao
19               | CmdAtribRetorno of variavel * identificador * identificador
20                   list
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 Código do sintaticoTest

Código para auxiliar a análise sintática de um arquivo inteiro

Listagem 5.3: sintaticoTest.ml - Código auxiliar

```

1  open Printf
2  open Lexing
3
4  open Ast
5  open ErroSint (*nome do modulo contendo as mensagens de erro*)
6
7  exception Erro_Sintatico of string
8
9  module S = MenhirLib.General (* Streams *)
10 module I = Sintatico.MenhirInterpreter
11
12 let posicao lexbuf =
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 "linha %d, coluna %d" lin col
17
18 (* [Pilha checkpoint] extrai a pilha do automato LR(1) contida em
19    checkpoint *)
20 let pilha checkpoint =
21     match checkpoint with
22     | I.HandlingError amb -> I.stack amb
23     | _ -> assert false (* Isso n'ao pode acontecer *)
24
25 let estado checkpoint : int =
26     match Lazy.force (pilha checkpoint) with
27     | S.Nil -> (*O parser esta no estado inicial *)
28         0
29     | S.Cons (I.Element (s, _, _, _), _) ->
30         I.number s

```

```

31
32 let sucesso v = Some v
33
34 let falha lexbuf (checkpoint : Ast.programa I.checkpoint) =
35   let estado_atual = estado checkpoint in
36   let msg = message estado_atual in
37   raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n" (Lexing.lexeme_start
    lexbuf) msg))
38
39 let loop lexbuf resultado =
40   let fornecedor = I.lexer_lexbuf_to_supplier Lexico.token lexbuf in
41   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
42
43 let parse_com_erro lexbuf =
44   try
45     Some (loop lexbuf (Sintatico.Incremental.programa lexbuf.lex_curr_p))
46   with
47     | Lexico.Erro msg -> printf "Erro lexico na %s:\n\t%s\n" (posicao
    lexbuf) msg;
48     None
49     | Erro_Sintatico msg ->
50       printf "Erro sintatico na %s %s\n" (posicao lexbuf) msg;
51       None
52
53 let parse s =
54   let lexbuf = Lexing.from_string s in
55   let ast = parse_com_erro lexbuf in
56   ast
57
58 let parse_arq nome =
59   let ic = open_in nome in
60   let lexbuf = Lexing.from_channel ic in
61   let result = parse_com_erro lexbuf in
62   let _ = close_in ic in
63   match result with
64     | Some ast -> ast
65     | None -> failwith "A analise sintatica falhou"
66
67 (* Para compilar:
68   menhir -v --list-errors sintatico.mly > sintatico.msg
69   menhir -v sintatico.mly --compile-errors sintatico.msg > erroSint.ml
70   ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -package
    menhirLib sintaticoTeste.byte
71 *)

```

5.4 Novo código do analisador léxico

Por fim, segue o código do novo analisador léxico, depois de retirado as definições de tokens dele para que não haja duplicatas

Listagem 5.4: lexico.mll - Novo código para o analisador léxico

```

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 <- { pos with
11             pos_lnum = pos.pos_lnum + 1;
12             pos_bol = pos.pos_cnum;
13         }
14
15 }
16
17 let digito = ['0' - '9']
18 let inteiro = '-'? digito+
19
20 let letra = ['a' - 'z' 'A' - 'Z']
21 let identificador = letra ( letra | digito | '_' ) *
22
23 let brancos = [' ' '\t']+
24 let novalinha = '\r' | '\n' | "\r\n"
25
26 rule token = parse
27 | brancos                { token lexbuf }
28 | novalinha              { incr_num_linha lexbuf; token lexbuf }
29 | "--["                 { comentario_bloco lexbuf }
30 | "--"                  { comentario_linha lexbuf }
31 | "int"                  { TIPO_INT }
32 | "bool"                 { TIPO_BOOLEAN }
33 | "string"               { TIPO_STRING }
34 | "("                    { APAR }
35 | "{"                    { ABRE_CHAVE }
36 | "["                    { ABRE_COLCHETE }
37 | "+"                    { ADICAO }
38 | "-"                    { SUBTRACAO }
39 | ")"                    { FPAR }
40 | "}"                    { FECHA_CHAVE }
41 | "]"                    { FECHA_COLCHETE }
42 | ","                    { VIRGULA }
43 | "."                    { PONTO }
44 | ";"                    { PONTO_VIRGULA }
45 | ":"                    { DOIS_PONTOS }
46 | "=="                   { EQUIVALENTE }
47 | "~="                   { NAO_EQUIVALENTE }
48 | ">="                    { MAIOR_OU_IGUAL }
49 | "<="                    { MENOR_OU_IGUAL }
50 | "/"                    { DIVISAO }
51 | "*"                    { MULTIPLICACAO }
52 | "%"                    { MODULO }
53 | "^"                    { EXPONENCIACAO }
54 | ">"                    { MAIOR }
55 | "<"                    { MENOR }
56 | "="                    { ATRIB }
57 | "#"                    { TAMANHO }
58 | "<<"                   { MULT_POR_2 }
59 | ">>"                   { DIV_POR_2 }
60 | "//"                   { DIVISAO_INTEIRO }
61 | "&"                    { AND_BINARIO }
62 | "|"                    { OR_BINARIO }
63 | ".."                   { CONCATENA }

```

```

64 | "..."          { RETICENCIAS }
65 | "and"           { AND }
66 | "break"        { BREAK }
67 | "do"           { DO }
68 | "else"         { ELSE }
69 | "elseif"       { ELSEIF }
70 | "end"          { END }
71 | "false"        { FALSE }
72 | "for"          { FOR }
73 | "function"     { FUNCAO }
74 | "if"           { IF }
75 | "io.read"      { IO_READ }
76 | "in"           { IN }
77 | "local"        { LOCAL }
78 | "nil"          { NIL }
79 | "not"          { NOT }
80 | "print"        { PRINT }
81 | "or"           { OR }
82 | "repeat"       { REPEAT }
83 | "return"       { RETURN }
84 | "then"         { THEN }
85 | "true"         { TRUE }
86 | "until"        { UNTIL }
87 | "while"        { WHILE }
88 | inteiro as num      { let numero = int_of_string num in
89 |                               LITINT numero }
90 | identificador as id { ID id }
91 | '"'           { let buffer = Buffer.create 1 in
92 |               let str = leia_string buffer lexbuf in
93 |               LITSTRING str }
94 | _             { raise (Erro ("Caracter desconhecido: " ^ Lexing.
95 |               lexeme lexbuf))}
96 | eof           { EOF }
97
98 and comentario_bloco = parse
99   "--]]"       { token lexbuf }
100 | novalinha    { incr_num_linha lexbuf; comentario_bloco lexbuf }
101 | _           { comentario_bloco lexbuf }
102 | eof         { raise (Erro "Comentario nao terminado")}
103
104 and leia_string buffer = parse
105   '"'          { Buffer.contents buffer}
106 | "\\t"        { Buffer.add_char buffer '\t'; leia_string buffer lexbuf }
107 | "\\n"        { Buffer.add_char buffer '\n'; leia_string buffer lexbuf }
108 | '\\\'' '\'' { Buffer.add_char buffer '\''; leia_string buffer lexbuf }
109 | '\\\'' '\\\'' { Buffer.add_char buffer '\\\''; leia_string buffer lexbuf }
110 | novalinha    {raise (Erro "A string nao foi fechada")}
111 | _ as c      { Buffer.add_char buffer c; leia_string buffer lexbuf }
112 | eof         { raise (Erro "A string nao foi terminada")}
113
114 and comentario_linha = parse
115   novalinha {incr_num_linha lexbuf; token lexbuf}
116 | _        {comentario_linha lexbuf}

```

5.5 Usando o analisador sintático

5.5.1 Pré-requisitos

Para executar o analisador léxico, são necessários que alguns pacotes do Ocaml extras sejam instalados. Para tal, deve-se começar instalando o ocamlbuild da seguinte maneira

```
$ sudo apt-get update
$ sudo apt-get install ocamlbuild
```

Em seguida, deve-se instalar o pacote *menhir*, pois utilizaremos *menhirLib*. Para tal, deve-se realizar o download do pacote no site oficial, [link para download](#), e então executar os seguintes comandos em seu diretório após descompactá-lo

```
$ make -f Makefile PREFIX=/usr/local USE_OCAMLFIND=true TARGET=byte all
$ sudo make -f Makefile PREFIX=/usr/local TARGET=byte install
```

Pode-se verificar que tal instalação ocorreu corretamente executando

```
$ ocamlfind query menhirLib
```

5.5.2 Compilando o analisador sintático

Antes que se possa executar o analisador sintático, devemos configurar suas mensagens de erro e compilá-lo. Para isso, vamos primeiramente gerar suas mensagens de erro, por meio do comando

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

Este código gerará um arquivo `.msg` contendo os casos de erro e suas respectivas mensagens. Essas mensagens devem ser modificadas nesse arquivo neste momento com mensagens que fazem sentido para cada tipo de erro antes que possamos prosseguir. Após feito isso, compile o arquivo de mensagens de erro para que o mesmo possa ser utilizado pelo analisador sintático da seguinte maneira

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

Agora podemos compilar todo o projeto com o auxílio de ocamlbuild, que criará uma nova pasta `_build` com os arquivos do projeto. Faz-se isso da seguinte maneira

```
$ ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -
package menhirLib sintaticoTest.byte
```

Agora estamos prontos para utilizar o analisador.

5.5.3 Executando o analisador

Para executar o analisador sintático, primeiramente devemos entrar no ambiente Ocaml

```
$ rlwrap ocaml
```

E, finalmente, podemos realizar a análise sintática de um arquivo .lua por meio da função no arquivo auxiliar `sintaticoTest.ml` [5.3](#) `parse_arq` e passar o nome do arquivo a ser analisado como parâmetro. Dessa forma

```
# parse_arq "nome_do_arquivo_a_ser_analisador.lua";;
```

Esse comando retornará o resultado do analisador sintático.

5.6 Testando o Analisador Sintático

Os seguintes testes com os programas nano e micro modificados tem a finalidade de validar a corretude das árvores geradas pelo analisador sintático.

5.6.1 Programas nano

Nano01

Listagem 5.5: nano01.lua - Programa nano01 modificado

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

Saída do analisador sintático:

Listagem 5.6: nano01.txt - Resultado do analisador sintático executado sobre nano01.lua modificado

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

Nano02

Listagem 5.7: nano02.lua - Programa nano02 modificado

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

Saída do analisador sintático:

Listagem 5.8: nano02.txt - Resultado do analisador sintático executado sobre nano02.lua modificado

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

Nano03

Listagem 5.9: nano03.lua - Programa nano03 modificado

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

Saída do analisador sintático:

Listagem 5.10: nano03.txt - Resultado do analisador sintático executado sobre nano03.lua modificado

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

Nano04

Listagem 5.11: nano04.lua - Programa nano04 modificado

```
1 function int main()
2     local int n
3     n = 1 + 2
4 end
```

Saída do analisador sintático:

Listagem 5.12: nano04.txt - Resultado do analisador sintático executado sobre nano04.lua modificado

```
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))])])
```

Nano05

Listagem 5.13: nano05.lua - Programa nano05 modificado

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

Saída do analisador sintático:

Listagem 5.14: nano05.txt - Resultado do analisador sintático executado sobre nano05.lua modificado

```
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"))])]])

```

Nano06

Listagem 5.15: nano06.lua - Programa nano06 modificado

```

1 function int main()
2   local int n
3   n = 1 - 2
4   print(n)
5 end

```

Saída do analisador sintático:

Listagem 5.16: nano06.txt - Resultado do analisador sintático executado sobre nano06.lua modificado

```

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"))])]])

```

Nano07

Listagem 5.17: nano07.lua - Programa nano07 modificado

```

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

```

Saída do analisador sintático:

Listagem 5.18: nano07.txt - Resultado do analisador sintático executado sobre nano07.lua modificado

```

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)])]])

```

Nano08

Listagem 5.19: nano08.lua - Programa nano08 modificado

```

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

```

Saída do analisador sintático:

Listagem 5.20: nano08.txt - Resultado do analisador sintático executado sobre nano08.lua modificado

```

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)])])
9   ])

```

Nano09

Listagem 5.21: nano09.lua - Programa nano09 modificado

```

1 function int main()
2     local 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 5.22: nano09.txt - Resultado do analisador sintático executado sobre nano09.lua modificado

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "n")],
5     [CmdAtrib (VarSimples "n",
6       ExpOp (Adicao, ExpInt 1, ExpOp (Divisao, 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 5.23: nano10.lua - Programa nano10 modificado

```

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

```

Saída do analisador sintático:

Listagem 5.24: nano10.txt - Resultado do analisador sintático executado sobre nano10.lua modificado

```

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 ]

```

Nano11

Listagem 5.25: nano11.lua - Programa nano11 modificado

```

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

```

Saída do analisador sintático:

Listagem 5.26: nano11.txt - Resultado do analisador sintático executado sobre nano11.lua modificado

```

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 5.27: nano12.lua - Programa nano12 modificado

```

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

```

Saída do analisador sintático:

Listagem 5.28: nano12.txt - Resultado do analisador sintático executado sobre nano12.lua modificado

```

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))]]))]

```

5.6.2 Programas micro

micro01

Listagem 5.29: micro01.lua - Programa micro01 modificado

```

1 function int main()
2     local int cel
3     local int far
4     print(" tabela de conversao: Celsius -> Fahrenheit\n")
5     print("Digite a temperatura em Celsius: ")
6     cel = io.read()
7     far = (9*cel + 160)/5
8     print("A nova temperatura e: " ..far.." F")
9 end

```

Saída do analisador sintático:

Listagem 5.30: micro01.txt - Resultado do analisador sintático executado sobre micro01.lua modificado

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "main", [],
5     [DecVar (TipoInt, VarSimples "cel"); DecVar (TipoInt, VarSimples "far
6     ")]],
7     [CmdPrint (ExpString " tabela de conversao: Celsius -> Fahrenheit\n")
8     ;
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
18      (ExpOp (Concatena,
19        ExpOp (Concatena, ExpString "A nova temperatura e: ",
20          ExpVar (VarSimples "far")),
21        ExpString " F")))]])

```

micro02

Listagem 5.31: micro02.lua - Programa micro02 modificado

```

1 function int main()
2     local int num1
3     local 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 número " ..num1.." é maior que o segundo " ..num2)

```



```

11     else
12         print("O segundo número "..num2.." é maior que o primeiro "..num1)
13     end
14 end

```

Saída do analisador sintático:

Listagem 5.32: micro02.txt - Resultado do analisador sintático executado sobre micro02.lua modificado

```

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
15      (ExpOp (Concatena,
16        ExpOp (Concatena,
17          ExpOp (Concatena, ExpString "O primeiro n\195\186mero ",
18            ExpVar (VarSimples "num1")),
19            ExpString " \195\169 maior que o segundo "),
20            ExpVar (VarSimples "num2")))],
21    Some
22      [CmdPrint
23        (ExpOp (Concatena,
24          ExpOp (Concatena,
25            ExpOp (Concatena, ExpString "O segundo n\195\186mero ",
26              ExpVar (VarSimples "num2")),
27              ExpString " \195\169 maior que o primeiro "),
28              ExpVar (VarSimples "num1")))]))]])

```

micro03

Listagem 5.33: micro03.lua - Programa micro03 modificado

```

1 function int main()
2   local 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\n")
8     else
9       print("O numero nao esta no intervalo entre 100 e 200\n")
10    end
11  else
12    print("O numero nao esta no intervalo entre 100 e 200\n")
13  end
14 end

```

Saída do analisador sintático:

Listagem 5.34: micro03.txt - Resultado do analisador sintático executado sobre micro03.lua modificado

```

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              n")]],
14          Some
15            [CmdPrint
16              (ExpString "O numero nao esta no intervalo entre 100 e 200\n")
17            ])],
18          Some
19            [CmdPrint
20              (ExpString "O numero nao esta no intervalo entre 100 e 200\n")
21            ])]])

```

micro04

Listagem 5.35: micro04.lua - Programa micro04 modificado

```

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

```

Saída do analisador sintático:

Listagem 5.36: micro04.txt - Resultado do analisador sintático executado sobre micro04.lua modificado

```

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     CmdPrint
21     (ExpOp (Concatena,
22     ExpOp (Concatena, ExpString "Ao total, foram digitados ",
23     ExpVar (VarSimples "intervalo")),
24     ExpString " numeros no intervalo entre 10 e 150")))]])

```

micro05

Listagem 5.37: micro05.lua - Programa micro05 modificado

```

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

```

Saída do analisador sintático:

Listagem 5.38: micro05.txt - Resultado do analisador sintático executado sobre micro05.lua modificado

```

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!\n")])])]);
28    CmdPrint
29      (ExpOp (Concatena,
30        ExpOp (Concatena, ExpString "Foram inseridos ",
31          ExpVar (VarSimples "h")),
32        ExpString " homens\n"));
33    CmdPrint
34      (ExpOp (Concatena,
35        ExpOp (Concatena, ExpString "Foram inseridos ",
36          ExpVar (VarSimples "m")),
37        ExpString " mulheres\n")))]))];

```

micro06

Listagem 5.39: micro06.lua - Programa micro06 modificado

```

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

```

```

18     print("Cinco\n")
19 else
20     print("Numero invalido!!!")
21 end
22 end

```

Saída do analisador sintático:

Listagem 5.40: micro06.txt - Resultado do analisador sintático executado sobre micro06.lua modificado

```

1 - : Ast.programa option =
2 Some
3 (Programa
4   [Funcao (TipoInt, "main", [], [DecVar (TipoInt, VarSimples "numero")],
5     [CmdPrint (ExpString "Digite um numero de 1 a 5: ");
6       CmdScan (VarSimples "numero");
7       CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "numero"), ExpInt 1),
8         [CmdPrint (ExpString "Um\n")], None);
9       CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "numero"), ExpInt 2),
10        [CmdPrint (ExpString "Dois\n")], None);
11      CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "numero"), ExpInt 3),
12        [CmdPrint (ExpString "Tres\n")], None);
13      CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "numero"), ExpInt 4),
14        [CmdPrint (ExpString "Quatro\n")], None);
15      CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "numero"), ExpInt 5),
16        [CmdPrint (ExpString "Cinco\n")],
17      Some [CmdPrint (ExpString "Numero invalido!!!")])])])

```

micro07

Listagem 5.41: micro07.lua - Programa micro07 modificado

```

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

```

Saída do analisador sintático:

Listagem 5.42: micro07.txt - Resultado do analisador sintático executado sobre micro07.lua modificado

```

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\n")],
15    Some
16    [CmdIf
17     (ExpOp (Equivalente, ExpVar (VarSimples "numero"), ExpInt 0),
18     [CmdPrint (ExpString "O numero e igual a 0\n")], None);
19     CmdIf (ExpOp (Menor, ExpVar (VarSimples "numero"), ExpInt 0),
20     [CmdPrint (ExpString "Negativo\n")], None)]];
21   CmdPrint (ExpString "Deseja finalizar? (S - 1): ");
22   CmdScan (VarSimples "opc");
23   CmdIf (ExpOp (Equivalente, ExpVar (VarSimples "opc"), ExpInt 1),
24    [CmdAtrib (VarSimples "programa", ExpInt 0)], None)]]))])

```

micro08

Listagem 5.43: micro08.lua - Programa micro08 modificado

```

1 function int main()
2   local 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 "..numero.." e maior que 10\n")
9     else
10      print("O numero "..numero.." e menor que 10\n")
11    end
12  end
13 end

```

Saída do analisador sintático:

Listagem 5.44: micro08.txt - Resultado do analisador sintático executado sobre micro08.lua modificado

```

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
12     (ExpOp (Concatena,
13     ExpOp (Concatena, ExpString "O numero ",
14     ExpVar (VarSimples "numero")),
15     ExpString " e maior que 10\n"))],
16     Some
17     [CmdPrint
18     (ExpOp (Concatena,
19     ExpOp (Concatena, ExpString "O numero ",
20     ExpVar (VarSimples "numero")),
21     ExpString " e menor que 10\n"))]]]]))

```

micro09

Listagem 5.45: micro09.lua - Programa micro09 modificado

```

1  function main()
2      local int preco
3      local int venda
4      local 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 < 80)
12         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 e "..novo_preco.."\\n")
19 end

```

Saída do analisador sintático:

Listagem 5.46: micro09.txt - Resultado do analisador sintático executado sobre micro09.lua modificado

```

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

```

```

11      (ExpOp (Or, ExpOp (Menor, ExpVar (VarSimples "venda"), ExpInt 500),
12        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_ou_Igual, ExpVar (VarSimples "venda"), ExpInt
24              1200)),
25          ExpOp (And,
26            ExpOp (Maior_ou_Igual, ExpVar (VarSimples "preco"), ExpInt
27              30),
28            ExpOp (Menor, ExpVar (VarSimples "preco"), ExpInt 80))),
29        [CmdAtrib (VarSimples "novo_preco",
30          ExpOp (Adicao, ExpVar (VarSimples "preco"),
31          ExpOp (Multiplicacao, ExpOp (Divisao, ExpInt 15, ExpInt 100),
32          ExpVar (VarSimples "preco")))]],
33        Some
34          [CmdIf
35            (ExpOp (Or,
36              ExpOp (Maior_ou_Igual, ExpVar (VarSimples "venda"),
37                ExpInt 1200),
38              ExpOp (Maior_ou_Igual, ExpVar (VarSimples "preco"), ExpInt
39                80)),
40            [CmdAtrib (VarSimples "novo_preco",
41              ExpOp (Subtracao, ExpVar (VarSimples "preco"),
42              ExpOp (Multiplicacao, ExpOp (Divisao, ExpInt 20, ExpInt
43                100),
44              ExpVar (VarSimples "preco")))]],
45            None)]])];
46    CmdPrint
47      (ExpOp (Concatena,
48        ExpOp (Concatena, ExpString "O novo preco e ",
49        ExpVar (VarSimples "novo_preco")),
50        ExpString "\n")))]])

```

micro10

Listagem 5.47: micro10.lua - Programa micro10 modificado

```

1 function int fatorial(int n)
2   local 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  local int numero
15  local 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(" e ")
22  print(fat)
23
24  return 1
25 end

```

Saída do analisador sintático:

Listagem 5.48: micro10.txt - Resultado do analisador sintático executado sobre micro10.lua modificado

```

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 "); CmdPrint (ExpString "numero")
22     ;
23     CmdPrint (ExpString " e "); CmdPrint (ExpVar (VarSimples "fat"));
24     CmdRetorno (ExpInt 1)]])

```

micro11

Listagem 5.49: micro11.lua - Programa micro11 modificado

```

1 function int main()
2   local int numero
3   local int x
4   print("Digite um numero ")
5   numero = io.read()
6   x = verifica(numero)
7   if x == 1 then
8     print("Numero positivo\n")
9   else if x == 0 then
10    print("Zero\n")
11  else

```

```

12     print("Numero negativo\n")
13 end
14 end
15 end
16
17 function int verifica(int n)
18     local int res
19     if n > 0 then
20         res = 1
21     else if n < 0 then
22         res = -1
23     else
24         res = 0
25     end
26 end
27 return res
28 end

```

Saída do analisador sintático:

Listagem 5.50: micro11.txt - Resultado do analisador sintático executado sobre micro11.lua modificado

```

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

```

5.7 Testes de Erros Sintáticos

A seguir são alguns dos erros sintáticos que podem ocorrer e gerar uma exceção pelo analisador sintático

Comandos fora do escopo de uma função

Listagem 5.51: teste01.lua - Programa de teste 01

```
1 while
```

Saída do analisador:

Listagem 5.52: saida01.txt - Resultado do analisador sintático executado sobre teste01.lua modificado

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

Função com estrutura incorreta

Listagem 5.53: teste02.lua - Programa de teste 02

```
1 function while
```

Saída do analisador:

Listagem 5.54: saida02.txt - Resultado do analisador sintático executado sobre teste02.lua modificado

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

Declaração de variável incorreta

Listagem 5.55: teste03.lua - Programa de teste 03

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

Saída do analisador:

Listagem 5.56: saida03.txt - Resultado do analisador sintático executado sobre teste03.lua modificado

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

Atribuição incorreta de variável

Listagem 5.57: teste04.lua - Programa de teste 04

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

```

3
4     x =
5
6 end

```

Saída do analisador:

Listagem 5.58: saida04.txt - Resultado do analisador sintático executado sobre teste01.lua modificado

```

1 Erro sintatico na linha 6, coluna 2 40 - "Atribuicao espera formato: '
    variavel ATRIB'"
2 .
3
4 Exception: Failure "A analise sintatica falhou".

```

Formato incorreto: comando IF

Listagem 5.59: teste05.lua - Programa de teste 05

```

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

```

Saída do analisador:

Listagem 5.60: saida05.txt - Resultado do analisador sintático executado sobre teste05.lua modificado

```

1 Erro sintatico na linha 6, 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".

```

Formato incorreto: comando FOR

Listagem 5.61: teste06.lua - Programa de teste 06

```

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

```

Saída do analisador:

Listagem 5.62: saida06.txt - Resultado do analisador sintático executado sobre teste06.lua modificado

```
1 Erro sintatico na linha 8, coluna 8 80 - "For espera o seguinte formato: '
    for variavel atrib litint virgula litint virgula litint do comandos end
    '".
2
3 Exception: Failure "A analise sintatica falhou".
```

Formato incorreto: comando WHILE

Listagem 5.63: teste07.lua - Programa de teste 07

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

Saída do analisador:

Listagem 5.64: saida07.txt - Resultado do analisador sintático executado sobre teste07.lua modificado

```
1 Erro sintatico na linha 6, coluna 11 52 - "While espera o seguinte formato
    : 'while expressao do comandos end'"
2 .
3
4 Exception: Failure "A analise sintatica falhou".
```

Formato incorreto: retorno de função

Listagem 5.65: teste08.lua - Programa de teste 08

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

Saída do analisador:

Listagem 5.66: saida08.txt - Resultado do analisador sintático executado sobre teste08.lua modificado

```
1 Erro sintatico na linha 7, coluna 2 54 - "Comando return espera o formato:
    'return expressao'"
2 .
3
4 Exception: Failure "A analise sintatica falhou".
```

Capítulo 6

Analizador Semântico

A tarefa de um analisador semântico é a de verificar os erros semânticos no código fonte e de coletar as informações pertinentes para a próxima etapa de compilação, a geração do código objeto. Um exemplo de erro semântico é a divisão de um número do tipo inteiro por um número do tipo float. Esse processo modifica a saída da análise sintática, transformando-a numa representação mais simples e mais adaptada a geração de código. Dessa forma, neste capítulo será mostrado a implementação de tal analisador, junto com outros programas auxiliares, específico para a linguagem MiniLua deste projeto, sendo tal implementação realizada em Ocaml. Dessa maneira, os programas nano e micro do capítulo 3 foram adaptados novamente para acomodar os requisitos dessa etapa. Além disso, os códigos do analisador léxico e sintático também sofreram pequenas alterações para acomodar essa nova etapa do trabalho.

6.1 Códigos dessa etapa

6.1.1 Código do Analisador Semântico

Segue o código do analisador semântico para a linguagem MiniLua utilizada nesse trabalho

Listagem 6.1: semantico.ml - Código do analisador semântico

```
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
```

6.1

```
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 -> Logico
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 | S.ExpOp (op, esq, dir) ->
106   let (esq, tesq) = infere_exp amb esq
107   and (dir, tdir) = infere_exp amb dir in
108   let verifica_aritmetico () =
109     (match tesq with
110     A.TipoInt
111     | A.TipoFloat ->
112       let _ = mesmo_tipo (snd op)
113         " (Aritmetico) O operando esquerdo eh do tipo %s mas
114           o direito eh do tipo %s"
115         tesq tdir
116       in tesq (* O tipo da expressão aritmética como um todo *)
117     | t -> let msg = "um operador aritmetico nao pode ser usado com o
118       tipo " ^
119         (nome_tipo t)
120       in failwith (msg_erro_pos (snd op) msg)
121     )
122   and verifica_relacional () =
123     (match tesq with
124     A.TipoInt
125     | A.TipoFloat
126     | A.TipoBool
127     | A.TipoString ->
128       let _ = mesmo_tipo (snd op)
129         " (Relacional) O operando esquerdo eh do tipo %s mas o
130           direito eh do tipo %s"

```



```

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

```

```

175 | [], [], [] -> ()
176 | _ -> failwith (msg_erro nome "Numero incorreto de parametros")
177 in
178 let id = fst nome in
179 try
180   begin
181     let open Amb in
182
183       match (Amb.busca amb id) with
184       (* verifica se 'nome' está associada a uma função *)
185       Amb.EntFun {tipo_fn; formais} ->
186       (* Infere o tipo de cada um dos argumentos *)
187       let argst = List.map (infere_exp amb) args
188       (* Obtem o tipo de cada parâmetro formal *)
189       and tipos_formais = List.map snd formais in
190       (* Verifica se o tipo de cada argumento confere com o tipo
191         declarado *)
192       (* do parâmetro formal correspondente. *)
193
194       let _ = verifica_parametros args (List.map snd argst)
195         tipos_formais
196       in (T.ExpChamada (id, (List.map fst argst), tipo_fn), tipo_fn)
197 | Amb.EntVar _ -> (* Se estiver associada a uma variável, falhe
198   *)
199   let msg = id ^ " eh uma variavel e nao uma funcao" in
200   failwith (msg_erro nome msg)
201 end
202 with Not_found ->
203 let msg = "Nao existe a funcao de nome " ^ id in
204 failwith (msg_erro nome msg)
205
206 let rec verifica_cmd amb tiporet cmd =
207   let open A in
208   match cmd with
209   CmdRetorno exp ->
210   (match exp with
211   (* Se a função não retornar nada, verifica se ela foi declarada como
212     void *)
213   None ->
214   let _ = mesmo_tipo (Lexing.dummy_pos)
215     "O tipo retornado eh %s mas foi declarado como %s"
216     TipoVoid tiporet
217   in CmdRetorno None
218 | Some e ->
219   (* Verifica se o tipo inferido para a expressão de retorno confere
220     com o *)
221   (* tipo declarado para a função. *)
222
223   let (e1,tinf) = infere_exp amb e in
224   let _ = mesmo_tipo (posicao e)
225     "O tipo retornado eh %s mas foi declarado
226     como %s"
227     tinf tiporet
228   in CmdRetorno (Some e1)
229 )
230 | CmdIf (teste, entao, senao) ->
231 let (testel,tinf) = infere_exp amb teste in
232 (* O tipo inferido para a expressão 'teste' do condicional deve ser
233   booleano *)

```

```

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

```

```

282 | A.VarSimples (id,pos) ->
283 (try
284   begin
285     (match (Amb.busca amb id) with
286     Amb.EntVar tipo ->
287       let expt = infere_exp amb exp in
288       let _ = mesmo_tipo pos
289       "ScanFloat com tipos diferentes: %s = %s"
290       tipo (snd expt) in
291       CmdScanFloat (fst expt)
292     | Amb.EntFun _ ->
293       let msg = "nome de funcao usado como nome de variavel: " ^ id
294       in
295       failwith (msg_erro_pos pos msg)
296     end
297   with Not_found ->
298     let _ = Amb.insere_local amb id A.TipoFloat in
299     let expt = infere_exp amb exp in
300     CmdScanFloat (fst expt)
301   | _ -> failwith "Falha ScanFloat"
302 ))
303 | CmdScanString exp ->
304 (match exp with
305   ExpVar v -> (match v with
306   | A.VarSimples (id, pos) ->
307     (try
308       begin
309         (match (Amb.busca amb id) with
310         Amb.EntVar tipo ->
311           let expt = infere_exp amb exp in
312           let _ = mesmo_tipo pos
313           "ScanString com tipos diferentes: %s = %s"
314           tipo (snd expt) in
315           CmdScanString (fst expt)
316         | Amb.EntFun _ ->
317           let msg = "nome de funcao usado como nome de variavel: " ^ id in
318           failwith (msg_erro_pos pos msg)
319         end
320       with Not_found ->
321         let _ = Amb.insere_local amb id A.TipoString in
322         let expt = infere_exp amb exp in
323         CmdScanString (fst expt)
324       | _ -> failwith "Falha ScanString")
325   | CmdWhile (exp_cond, comandos) ->
326     let (expCond, expT) = infere_exp amb exp_cond in
327     let comandos_tipados =
328       (match expT with
329       | A.TipoBool -> List.map (verifica_cmd amb tiporet) comandos
330       | _ -> let msg = "Condicao deve ser tipo Bool" in
331         failwith (msg_erro_pos (posicao exp_cond) msg))
332     in CmdWhile (expCond, comandos_tipados)
333
334
335 and verifica_fun amb ast =
336   let open A in
337   match ast with
338   A.DecFun {fn_nome; fn_tiporet; fn_formais; fn_locais; fn_corpo} ->
339     (* Estende o ambiente global, adicionando um ambiente local *)

```

```

340 let ambfn = Amb.novo_escopo amb in
341 (* Insere os parâmetros no novo ambiente *)
342 let insere_parametro (v,t) = Amb.insere_param ambfn (fst v) t in
343 let _ = List.iter insere_parametro fn_formais in
344 (* Insere as variáveis locais no novo ambiente *)
345 let insere_local = function
346   (DecVar (v,t)) -> Amb.insere_local ambfn (fst v) t in
347 let _ = List.iter insere_local fn_locais in
348 (* Verifica cada comando presente no corpo da função usando o novo
    ambiente *)
349 let corpo_tipado = List.map (verifica_cmd ambfn fn_tiporet) fn_corpo
    in
350   A.DecFun {fn_nome; fn_tiporet; fn_formais; fn_locais; fn_corpo =
    corpo_tipado}
351
352
353 let rec verifica_dup xs =
354   match xs with
355   [] -> []
356 | (nome,t)::xs ->
357   let id = fst nome in
358   if (List.for_all (fun (n,t) -> (fst n) <> id) xs)
359   then (id, t) :: verifica_dup xs
360   else let msg = "Parametro duplicado " ^ id in
361     failwith (msg_erro nome msg)
362
363 let insere_declaracao_var amb dec =
364   let open A in
365     match dec with
366     DecVar (nome, tipo) -> Amb.insere_local amb (fst nome) tipo
367
368 let insere_declaracao_fun amb dec =
369   let open A in
370     match dec with
371     DecFun {fn_nome; fn_tiporet; fn_formais; fn_corpo} ->
372       (* Verifica se não há parâmetros duplicados *)
373       let formais = verifica_dup fn_formais in
374       let nome = fst fn_nome in
375       Amb.insere_fun amb nome formais fn_tiporet
376
377
378 (* Lista de cabeçalhos das funções pré definidas *)
379 let fn_predefs = let open A in [
380   ("entrada", [("x", TipoInt); ("y", TipoInt)], TipoVoid);
381   ("saida",   [("x", TipoInt); ("y", TipoInt)], TipoVoid)
382 ]
383
384 (* insere as funções pré definidas no ambiente global *)
385 let declara_predefinidas amb =
386   List.iter (fun (n,ps,tr) -> Amb.insere_fun amb n ps tr) fn_predefs
387
388 let semantico ast =
389   (* cria ambiente global inicialmente vazio *)
390   let amb_global = Amb.novo_amb [] in
391   let _ = declara_predefinidas amb_global in
392   let (A.Programa (decs_globais, decs_funs, corpo)) = ast in
393   let _ = List.iter (insere_declaracao_var amb_global) decs_globais in
394   let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
395   (* Verificação de tipos nas funções *)

```

```

396 let decs_funs = List.map (verifica_fun amb_global) decs_funs in
397 (* Verificação de tipos na função principal *)
398 let corpo = List.map (verifica_cmd amb_global A.TipoVoid) corpo in
399 (A.Programa (decs_globais, decs_funs, corpo), amb_global)

```

Listagem 6.2: semantico.mli - Código do analisador semântico

```

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

```

6.1.2 Código do Ambiente

Segue o código do arquivo auxiliar que trata dos ambientes de execução

Listagem 6.3: ambiente.ml - Código do arquivo auxiliar de ambiente

```

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 type entrada = EntFun of entrada_fn
8               | EntVar of A.tipo
9
10 type t = {
11   ambv : entrada Tab.tabela
12 }
13
14 let novo_amb xs = { ambv = Tab.cria xs }
15
16 let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }
17
18 let busca amb ch = Tab.busca amb.ambv ch
19
20 let insere_local amb ch t = Tab.insere amb.ambv ch (EntVar t)
21
22 let insere_param amb ch t = Tab.insere amb.ambv ch (EntVar t)
23
24 let insere_fun amb nome params resultado =
25   let ef = EntFun { tipo_fn = resultado;
26                     formais = params }
27   in Tab.insere amb.ambv nome ef

```

Listagem 6.4: ambiente.mli - Código do arquivo auxiliar de ambiente

```

1 type entrada_fn = { tipo_fn: Ast.tipo;
2                     formais: (string * Ast.tipo) list; }
3
4 type entrada = EntFun of entrada_fn
5               | EntVar of Ast.tipo
6
7 type t
8
9 val novo_amb : (string * entrada) list -> t
10 val novo_escopo : t -> t

```

```

11 val busca: t -> string -> entrada
12 val insere_local : t -> string -> Ast.tipo -> unit
13 val insere_param : t -> string -> Ast.tipo -> unit
14 val insere_fun : t -> string -> (string * Ast.tipo) list -> Ast.tipo ->
    unit

```

6.1.3 Novo código da árvore abstrata

Segue o código da nova árvore abstrata sintática para essa etapa do trabalho

Listagem 6.5: ast.ml - Código da ast

```

1 open Lexing
2
3 type ident = string
4 type 'a pos = 'a * Lexing.position
5
6 type 'expr programa = Programa of declaracoes * ('expr funcoes) * ('expr
    comandos)
7     and declaracoes = declaracao list
8     and 'expr funcoes = ('expr funcao) list
9     and 'expr comandos = ('expr comando) list
10
11     and declaracao = DecVar of (ident pos) * tipo
12     and 'expr funcao = DecFun of ('expr decfn)
13
14     and 'expr decfn = {
15         fn_nome:      ident pos;
16         fn_tiporet: tipo;
17         fn_formais: (ident pos * tipo) list;
18         fn_locais: declaracoes;
19         fn_corpo:      'expr comandos
20     }
21
22     and tipo = TipoInt
23               | TipoString
24               | TipoBool
25               | TipoFloat
26               | TipoVoid
27
28     and campos = campo list
29     and campo = ident pos * tipo
30
31     and 'expr comando =
32         | CmdAtrib of 'expr * 'expr
33         | CmdIf of 'expr * ('expr comandos) * ('expr comandos
34             option)
35         | CmdRetorno of 'expr option
36         | CmdChamada of 'expr
37         | CmdPrint of 'expr
38         | CmdScanInt of 'expr
39         | CmdScanFloat of 'expr
40         | CmdScanString of 'expr
41         | CmdWhile of 'expr * ('expr comandos)
42
43     and 'expr variaveis = ('expr variavel) list
44     and 'expr variavel =

```

```

44             | VarSimples of ident pos
45
46         and 'expr expressoes = 'expr list
47
48         and oper = Or
49             | And
50             | Maior
51             | Menor
52             | Maior_ou_Igual
53             | Menor_ou_Igual
54             | Equivalente
55             | Nao_Equivalente
56             | Or_Binario
57             | Or_Binario_Exclusivo
58             | And_Binario
59             | Mult_Por_2
60             | Div_Por_2
61             | Concatena
62             | Adicao
63             | Subtracao
64             | Multiplicacao
65             | Divisao
66             | Divisao_Inteiro
67             | Modulo
68             | Not
69             | Exponenciacao

```

6.1.4 Código do novo lexico

Segue o código para o novo lexico adaptado

Listagem 6.6: lexico.mll - Código do analisador lexico

```

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 <- { pos with
11             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']

```


6.1

```
25 let identificador = letra ( letra | digito | '_' ) *
26
27 let brancos = [ ' ' | '\t' ] +
28 let novalinha = '\r' | '\n' | "\r\n"
29
30 rule token = parse
31 | brancos                { token lexbuf }
32 | novalinha              { incr_num_linha lexbuf; token lexbuf }
33 | "--["                 { comentario_bloco lexbuf }
34 | "--"                  { comentario_linha lexbuf }
35 | "int"                  { TIPO_INT (pos_atual lexbuf)}
36 | "float"                { TIPO_FLOAT (pos_atual lexbuf)}
37 | "bool"                 { TIPO_BOOLEAN (pos_atual lexbuf)}
38 | "string"               { TIPO_STRING (pos_atual lexbuf)}
39 | "("                   { APAR (pos_atual lexbuf)}
40 | "{"                    { ABRE_CHAVE (pos_atual lexbuf)}
41 | "["                   { ABRE_COLCHETE (pos_atual lexbuf)}
42 | "+"                    { ADICAO (pos_atual lexbuf)}
43 | "-"                    { SUBTRACAO (pos_atual lexbuf)}
44 | ")"                   { FPAR (pos_atual lexbuf)}
45 | "}"                    { FECHA_CHAVE (pos_atual lexbuf)}
46 | "]"                   { FECHA_COLCHETE (pos_atual lexbuf)}
47 | ","                   { VIRGULA (pos_atual lexbuf)}
48 | "."                   { PONTO (pos_atual lexbuf)}
49 | ";"                   { PONTO_VIRGULA (pos_atual lexbuf)}
50 | ":"                   { DOIS_PONTOS (pos_atual lexbuf)}
51 | "=="                  { EQUIVALENTE (pos_atual lexbuf)}
52 | "~="                  { NAO_EQUIVALENTE (pos_atual lexbuf)}
53 | ">="                  { MAIOR_OU_IGUAL (pos_atual lexbuf)}
54 | "<="                  { MENOR_OU_IGUAL (pos_atual lexbuf)}
55 | "/"                   { DIVISAO (pos_atual lexbuf)}
56 | "*"                   { MULTIPLICACAO (pos_atual lexbuf)}
57 | "%"                   { MODULO (pos_atual lexbuf)}
58 | "^"                   { EXPONENCIACAO (pos_atual lexbuf)}
59 | ">"                   { MAIOR (pos_atual lexbuf)}
60 | "<"                   { MENOR (pos_atual lexbuf)}
61 | "="                   { ATRIB (pos_atual lexbuf)}
62 | "#"                   { TAMANHO (pos_atual lexbuf)}
63 | "<<"                  { MULT_POR_2 (pos_atual lexbuf)}
64 | ">>"                  { DIV_POR_2 (pos_atual lexbuf)}
65 | "//"                  { DIVISAO_INTEIRO (pos_atual lexbuf)}
66 | "&"                   { AND_BINARIO (pos_atual lexbuf)}
67 | "|"                   { OR_BINARIO (pos_atual lexbuf)}
68 | "~"                   { OR_BINARIO_EXCLUSIVO (pos_atual lexbuf)}
69 | ".."                  { CONCATENA (pos_atual lexbuf)}
70 | "..."                { RETICENCIAS (pos_atual lexbuf)}
71 | "and"                  { AND (pos_atual lexbuf)}
72 | "break"                { BREAK (pos_atual lexbuf)}
73 | "do"                   { DO (pos_atual lexbuf)}
74 | "else"                 { ELSE (pos_atual lexbuf)}
75 | "elseif"               { ELSEIF (pos_atual lexbuf)}
76 | "end"                  { END (pos_atual lexbuf)}
77 | "false"                { LITBOOL (false, pos_atual lexbuf)}
78 | "for"                  { FOR (pos_atual lexbuf)}
79 | "function"             { FUNCAO (pos_atual lexbuf)}
80 | "if"                   { IF (pos_atual lexbuf)}
81 | "io.read('*n')"         { IO_READ_INT (pos_atual lexbuf) }
82 | "io.read('*f')"         { IO_READ_FLOAT (pos_atual lexbuf)}
83 | "io.read('*s')"         { IO_READ_STRING (pos_atual lexbuf)}
```

```

84 | "in"                { IN (pos_atual lexbuf) }
85 | "local"           { LOCAL (pos_atual lexbuf) }
86 | "nil"             { NIL (pos_atual lexbuf) }
87 | "not"             { NOT (pos_atual lexbuf) }
88 | "print"           { PRINT (pos_atual lexbuf) }
89 | "or"              { OR (pos_atual lexbuf) }
90 | "repeat"          { REPEAT (pos_atual lexbuf) }
91 | "return"          { RETURN (pos_atual lexbuf) }
92 | "then"            { THEN (pos_atual lexbuf) }
93 | "true"            { LITBOOL (true, pos_atual lexbuf) }
94 | "until"           { UNTIL (pos_atual lexbuf) }
95 | "while"           { WHILE (pos_atual lexbuf) }
96 | inteiro as n      { LITINT (int_of_string n, pos_atual lexbuf) }
97 | real as n         { LITFLOAT (float_of_string n, pos_atual lexbuf) }
98 | identificador as id { ID (id, pos_atual lexbuf) }
99 | '"'              { let buffer = Buffer.create 1 in
100 |                  let str = leia_string buffer lexbuf in
101 |                  LITSTRING (str, pos_atual lexbuf) }
102 | _                { raise (Erro ("Caracter desconhecido: " ^ Lexing.
lexeme lexbuf)) }
103 | eof              { EOF (pos_atual lexbuf) }
104
105 and comentario_bloco = parse
106   "--]]"           { token lexbuf }
107 | novalinha        { incr_num_linha lexbuf; comentario_bloco lexbuf }
108 | _                { comentario_bloco lexbuf }
109 | eof              { raise (Erro "Comentario nao terminado") }
110
111 and leia_string buffer = parse
112   '"'              { Buffer.contents buffer }
113 | "\\t"            { Buffer.add_char buffer '\t'; leia_string buffer lexbuf }
114 | "\\n"            { Buffer.add_char buffer '\n'; leia_string buffer lexbuf }
115 | '\\' ' "'       { Buffer.add_char buffer '"'; leia_string buffer lexbuf }
116 | '\\' '\\ '      { Buffer.add_char buffer '\\'; leia_string buffer lexbuf }
117 | novalinha        { raise (Erro "A string nao foi fechada") }
118 | _ as c           { Buffer.add_char buffer c; leia_string buffer lexbuf }
119 | eof              { raise (Erro "A string nao foi terminada") }
120
121 and comentario_linha = parse
122   novalinha {incr_num_linha lexbuf; token lexbuf}
123 | _        {comentario_linha lexbuf}

```

6.1.5 Código do novo analisador sintático

Segue o código para o novo analisador sintático adaptado

Listagem 6.7: sintatico.mly - Código do analisador sintático

```

1 %{
2   open Lexing
3   open Ast
4   open Sast
5 %}
6
7 %token <string * Lexing.position> ID
8 %token <string * Lexing.position> LITSTRING
9 %token <int * Lexing.position> LITINT

```

6.1

10 %token <float * Lexing.position>	LITFLOAT
11 %token <bool * Lexing.position>	LITBOOL
12 %token <Lexing.position>	ABRE_CHAVE
13 %token <Lexing.position>	ABRE_COLCHETE
14 %token <Lexing.position>	FECHA_CHAVE
15 %token <Lexing.position>	FECHA_COLCHETE
16 %token <Lexing.position>	ADICAO
17 %token <Lexing.position>	AND
18 %token <Lexing.position>	AND_BINARIO
19 %token <Lexing.position>	APAR
20 %token <Lexing.position>	ATRIB
21 %token <Lexing.position>	BREAK
22 %token <Lexing.position>	CONCATENA
23 %token <Lexing.position>	DIV_POR_2
24 %token <Lexing.position>	DIVISAO
25 %token <Lexing.position>	DIVISAO_INTEIRO
26 %token <Lexing.position>	DO
27 %token <Lexing.position>	DOIS_PONTOS
28 %token <Lexing.position>	ELSE
29 %token <Lexing.position>	ELSEIF
30 %token <Lexing.position>	END
31 %token <Lexing.position>	EQUIVALENTE
32 %token <Lexing.position>	EXPONENCIACAO
33 %token <Lexing.position>	FOR
34 %token <Lexing.position>	FPAR
35 %token <Lexing.position>	FUNCAO
36 %token <Lexing.position>	IF
37 %token <Lexing.position>	IN
38 %token <Lexing.position>	IO_READ_INT
39 %token <Lexing.position>	IO_READ_FLOAT
40 %token <Lexing.position>	IO_READ_STRING
41 %token <Lexing.position>	LOCAL
42 %token <Lexing.position>	MAIOR
43 %token <Lexing.position>	MAIOR_OU_IGUAL
44 %token <Lexing.position>	MENOR
45 %token <Lexing.position>	MENOR_OU_IGUAL
46 %token <Lexing.position>	MODULO
47 %token <Lexing.position>	MULT_POR_2
48 %token <Lexing.position>	MULTIPLICACAO
49 %token <Lexing.position>	NAO_EQUIVALENTE
50 %token <Lexing.position>	NIL
51 %token <Lexing.position>	NOT
52 %token <Lexing.position>	NUMBER_INPUT
53 %token <Lexing.position>	OR
54 %token <Lexing.position>	OR_BINARIO
55 %token <Lexing.position>	OR_BINARIO_EXCLUSIVO
56 %token <Lexing.position>	PONTO
57 %token <Lexing.position>	PONTO_VIRGULA
58 %token <Lexing.position>	PRINT
59 %token <Lexing.position>	RETICENCIAS
60 %token <Lexing.position>	RETURN
61 %token <Lexing.position>	SUBTRACAO
62 %token <Lexing.position>	TAMANHO
63 %token <Lexing.position>	TIPO_BOOLEAN
64 %token <Lexing.position>	TIPO_INT
65 %token <Lexing.position>	TIPO_FLOAT
66 %token <Lexing.position>	TIPO_STRING
67 %token <Lexing.position>	THEN
68 %token <Lexing.position>	UNTIL

```

69 %token <Lexing.position>          VIRGULA
70 %token <Lexing.position>          WHILE
71 %token <Lexing.position>          EOF
72 %token <Lexing.position>          FALSE
73 %token <Lexing.position>          TRUE
74 %token <Lexing.position>          REPEAT
75
76 %left          OR
77 %left          AND
78 %left          MAIOR MENOR MENOR_OU_IGUAL MAIOR_OU_IGUAL EQUIVALENTE
      NAO_EQUIVALENTE
79 %left          OR_BINARIO
80 %left          OR_BINARIO_EXCLUSIVO
81 %left          AND_BINARIO
82 %left          MULT_POR_2 DIV_POR_2
83 %left          CONCATENA
84 %left          ADICAO SUBTRACAO
85 %left          MULTIPLICACAO DIVISAO DIVISAO_INTEIRO MODULO
86 %left          NOT TAMANHO
87 %left          EXPONENCIACAO
88
89 %start <Sast.expressao Ast.programa> programa
90
91 %%
92
93 programa:
94     ds = declaracao_de_variavel*
95     fs = declaracao_de_funcao*
96     cs = comando*
97     EOF { Programa (List.flatten ds, fs, cs) }
98
99 declaracao_de_variavel:
100     t = tipo ids = separated_nonempty_list(VIRGULA, ID){
101     List.map (fun id -> DecVar (id,t)) ids
102     }
103
104 declaracao_de_funcao:
105     FUNCAO tret = tipo nome = ID APAR formais = separated_list (VIRGULA,
      parametro) FPAR
106     ds = declaracao_de_variavel*
107     cs = comando*
108     END {
109     DecFun {
110     fn_nome = nome;
111     fn_tiporet = tret;
112     fn_formais = formais;
113     fn_locais = List.flatten ds;
114     fn_corpo = cs
115     }
116     }
117
118 parametro: t = tipo nome = ID { (nome, t) }
119
120 tipo: t = tipo_simples { t }
121
122 tipo_simples: TIPO_INT { TipoInt }
123               | TIPO_FLOAT { TipoFloat }
124               | TIPO_STRING { TipoString }
125               | TIPO_BOOLEAN { TipoBool }

```

6.1

```
126
127
128 comando:  c = comando_atribuicao  { c }
129           | c = comando_if        { c }
130           | c = comando_chamada   { c }
131           | c = comando_for       { c }
132           | c = comando_while     { c }
133           | c = comando_print     { c }
134           | c = comando_scanInt   { c }
135           | c = comando_scanFloat { c }
136           | c = comando_scanString { c }
137           | c = comando_retorno   { c }
138
139
140
141 comando_atribuicao: v = expressao ATRIB exp = expressao {
142   CmdAtrib (v, exp)
143 }
144
145
146
147 comando_if: IF  teste = expressao THEN
148               entao = comando+
149               senao = option (ELSE cs = comando+ { cs })
150           END {
151               CmdIf (teste, entao, senao)
152           }
153
154 comando_chamada: exp = chamada {  CmdChamada exp  }
155
156 comando_retorno: RETURN  exp = expressao? { CmdRetorno  exp }
157
158 comando_print: PRINT APAR  exp = expressao FPAR  {CmdPrint exp }
159
160 comando_scanInt: exp = expressao ATRIB IO_READ_INT {  CmdScanInt exp }
161
162 comando_scanFloat:  exp = expressao ATRIB IO_READ_FLOAT { CmdScanFloat exp
163   }
164
164 comando_scanString: exp = expressao  ATRIB IO_READ_STRING { CmdScanString
165   exp }
166
166 comando_while: WHILE exp = expressao DO  cs = comando* END { CmdWhile  (
167   exp, cs) }
168
167
168 comando_for: FOR v = ID ATRIB init=expressao VIRGULA teste=expressao
169   VIRGULA inc=expressao DO cs=comando* END {
169   CmdIf (ExpBool (true, snd v),
170   [
171     CmdAtrib (ExpVar (VarSimples v), init) ;
172     CmdWhile (
173       ExpOp ((Menor_ou_Igual, snd v),
174       ExpVar (VarSimples v),
175       teste
176     ),
177     List.append cs [CmdAtrib (ExpVar (VarSimples v),
178     ExpOp (
179       (Adicao, snd v),
180       ExpVar (VarSimples v),
```

```

181         inc)
182     )
183 ]
184 )],
185 None)
186 }
187
188
189
190
191 expressao:
192   | v = variavel { ExpVar v }
193   | i = LITINT   { ExpInt i }
194   | f = LITFLOAT { ExpFloat f }
195   | s = LITSTRING { ExpString s }
196   | b = LITBOOL   { ExpBool b }
197 |e1 = expressao op = oper e2 = expressao { ExpOp (op, e1, e2) }
198   | c = chamada { c }
199 |APAR e = expressao FPAR { e }
200
201
202 chamada: nome = ID APAR args = separated_list(VIRGULA, expressao) FPAR {
203   ExpChamada (nome, args)
204 }
205
206
207
208 %inline oper:
209   | pos = OR { (Or, pos) }
210   | pos = AND { (And, pos) }
211   | pos = MAIOR { (Maior, pos) }
212   | pos = MENOR { (Menor, pos) }
213   | pos = MAIOR_OU_IGUAL { (Maior_ou_Igual, pos) }
214   | pos = MENOR_OU_IGUAL { (Menor_ou_Igual, pos) }
215   | pos = EQUIVALENTE { (Equivalente, pos) }
216   | pos = NAO_EQUIVALENTE { (Nao_Equivalente, pos) }
217   | pos = OR_BINARIO { (Or_Binario, pos) }
218   | pos = OR_BINARIO_EXCLUSIVO { (Or_Binario_Exclusivo, pos) }
219   | pos = AND_BINARIO { (And_Binario, pos) }
220   | pos = MULT_POR_2 { (Mult_Por_2, pos) }
221   | pos = DIV_POR_2 { (Div_Por_2, pos) }
222   | pos = CONCATENA { (Concatena, pos) }
223   | pos = ADICAO { (Adicao, pos) }
224   | pos = SUBTRACAO { (Subtracao, pos) }
225   | pos = MULTIPLICACAO { (Multiplicacao, pos) }
226   | pos = DIVISAO { (Divisao, pos) }
227   | pos = DIVISAO_INTEIRO { (Divisao_Inteiro, pos) }
228   | pos = MODULO { (Modulo, pos) }
229   | pos = NOT { (Not, pos) }
230   | pos = EXPONENCIACAO { (Exponenciacao, pos) }
231
232 variavel:
233   | x=ID { VarSimples x }

```

Listagem 6.8: semantico.mli - Código do analisador semântico

```

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

```

6.1.6 Código do arquivo para testar o analisador semântico

Segue o código do arquivo para testar o analisador semântico nessa etapa do trabalho

Listagem 6.9: `semanticoTest.ml` - Código do arquivo que permite testar o analisador semântico

```

1 open Printf
2 open Lexing
3
4 open Ast
5 exception Erro_Sintatico of string
6
7 module S = MenhirLib.General (* Streams *)
8 module I = Sintatico.MenhirInterpreter
9
10 open Semantico
11
12 let message =
13   fun s ->
14     match s with
15     | 0 ->
16       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
17     | 1 ->
18       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
19     | 34 ->
20       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
21     | 35 ->
22       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
23     | 36 ->
24       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
25     | 72 ->
26       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
27     | 47 ->
28       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
29     | 48 ->
30       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
31     | 49 ->
32       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
33     | 51 ->
34       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
35     | 52 ->
36       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
37     | 55 ->
38       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
39     | 56 ->
40       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
41     | 57 ->
42       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
43     | 58 ->
44       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
45     | 61 ->
46       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
47     | 62 ->
48       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
49     | 63 ->
50       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
51     | 64 ->
52       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"

```

```

53 | 73 ->
54 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
55 | 74 ->
56 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
57 | 95 ->
58 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
59 | 89 ->
60 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
61 | 97 ->
62 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
63 | 98 ->
64 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
65 | 99 ->
66 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
67 | 65 ->
68 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
69 | 66 ->
70 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
71 | 53 ->
72 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
73 | 67 ->
74 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
75 | 68 ->
76 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
77 | 59 ->
78 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
79 | 60 ->
80 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
81 | 42 ->
82 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
83 | 41 ->
84 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
85 | 70 ->
86 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
87 | 75 ->
88 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
89 | 77 ->
90 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
91 | 76 ->
92 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
93 | 105 ->
94 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
95 | 84 ->
96 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
97 | 43 ->
98 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
99 | 85 ->
100 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
101 | 86 ->
102 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
103 | 45 ->
104 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
105 | 46 ->
106 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
107 | 102 ->
108 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
109 | 103 ->
110 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
111 | 81 ->

```



```

112     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
113 | 3 ->
114     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
115 | 2 ->
116     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
117 | 6 ->
118     "estado 6: esperava um tipo. Exemplo:\n  x : inteiro;\n"
119 | 7 ->
120     "estado 7: esperava a definicao de um campo. Exemplo:\n  i:
        registro\n                parte_real: inteiro;\n                parte_imag:
        inteiro;\n                fim registro;\n                "
121 | 8 ->
122     "estado 8: esperava ':'. Exemplo:\n  x: inteiro;\n  "
123 | 9 ->
124     "estado 9: esperava um tipo. Exemplo:\n  x: inteiro;\n"
125 | 25 ->
126     "estado 25: esperva um ';'.'. \n"
127 | 26 ->
128     "estado 26: uma declaracao foi encontrada. Para continuar era\n
        esperado uma outra declara\195\167\195\163o ou a palavra '
        inicio'.'. \n"
129 | 29 ->
130     "estado 29: espera a palavra 'registro'. Exemplo:\n  i: registro\
        n                parte_real: inteiro;\n                parte_imag: inteiro;\n
        fim registro;\n"
131 | 31 ->
132     "estado 31: esperava um ';'.'. \n"
133 | 107 ->
134     "estado 107: uma declaracao foi encontrada. Para continuar era\n
        esperado uma outra declara\195\167\195\163o ou a palavra '
        inicio'.'. \n"
135 | 13 ->
136     "estado 13: esperava um '['. Exemplo:\n  arranjo [1..10] de
        inteiro;\n"
137 | 14 ->
138     "estado 14: esperava os limites do vetor. Exemplo:\n  arranjo
        [1..10] de inteiro;\n"
139 | 15 ->
140     "estado 15: esperava '...'. Exemplo:\n  1 .. 10\n"
141 | 16 ->
142     "estado 16: esperava um numero inteiro. Exemplo:\n  1 .. 10\n"
143 | 18 ->
144     "estado 18: esperava um ']'. Exemplo\n  arranjo [1..10] de
        inteiro;\n"
145 | 19 ->
146     "estado 19: esperava a palavra reservada 'de'. Exemplo:\n
        arranjo [1..10] de inteiro;\n"
147 | 20 ->
148     "estado 20: esperava um tipo. Exemplo\n  arranjo [1..10] de
        inteiro;\n"
149 | _ ->
150     raise Not_found
151
152 let posicao lexbuf =
153     let pos = lexbuf.lex_curr_p in
154     let lin = pos.pos_lnum
155     and col = pos.pos_cnum - pos.pos_bol - 1 in
156     sprintf "linha %d, coluna %d" lin col
157

```

```

158 (* [pilha checkpoint] extrai a pilha do autômato LR(1) contida em
    checkpoint *)
159
160 let pilha checkpoint =
161   match checkpoint with
162   | I.HandlingError amb -> I.stack amb
163   | _ -> assert false (* Isso não pode acontecer *)
164
165 let estado checkpoint : int =
166   match Lazy.force (pilha checkpoint) with
167   | S.Nil -> (* O parser está no estado inicial *)
168       0
169   | S.Cons (I.Element (s, _, _, _), _) ->
170       I.number s
171
172 let sucesso v = Some v
173
174 let falha lexbuf (checkpoint : (Sast.expressao Ast.programa) I.checkpoint)
175   =
176   let estado_atual = estado checkpoint in
177   let msg = message estado_atual in
178   raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
179                                         (Lexing.lexeme_start lexbuf) msg))
179
180 let loop lexbuf resultado =
181   let fornecedor = I.lexer_lexbuf_to_supplier Lexico.token lexbuf in
182   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
183
184
185 let parse_com_erro lexbuf =
186   try
187     Some (loop lexbuf (Sintatico.Incremental.programa lexbuf.lex_curr_p))
188   with
189   | Lexico.Erro msg ->
190     printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
191     None
192   | Erro_Sintatico msg ->
193     printf "Erro sintático na %s %s\n" (posicao lexbuf) msg;
194     None
195
196 let parse s =
197   let lexbuf = Lexing.from_string s in
198   let ast = parse_com_erro lexbuf in
199   ast
200
201 let parse_arq nome =
202   let ic = open_in nome in
203   let lexbuf = Lexing.from_channel ic in
204   let ast = parse_com_erro lexbuf in
205   let _ = close_in ic in
206   ast
207
208 let verifica_tipos nome =
209   let ast = parse_arq nome in
210   match ast with
211   | Some (Some ast) -> semantico ast
212   | _ -> failwith "Nada a fazer!\n"
213
214 (* Para compilar:

```

6.1

```
215     ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -  
        package menhirLib semanticoTest.byte  
216  
217 Para usar, entre no ocaml  
218  
219     rlwrap ocaml  
220  
221 e se desejar ver apenas a árvore sintática que sai do analisador sintá  
    tico, digite  
222  
223     parse_arq "exemplos/ex2.tip";;  
224  
225 Depois, para ver a saída do analisador semântico já com a árvore  
    anotada com  
226 o tipos, digite:  
227  
228     verifica_tipos "exemplos/ex2.tip";;  
229  
230 Note que o analisador semântico está retornando também o ambiente  
    global. Se  
231 quiser separá-los, digite:  
232  
233     let (arv, amb) = verifica_tipos "exemplos/ex2.tip";;  
234  
235  
236  
237 *)
```

6.1.7 Código do Sast

Segue o código do Sast, arquivo auxiliar

Listagem 6.10: sast.ml - Código do sast - arquivo auxiliar

```
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)
```

6.1.8 Código do Tast

Segue o código do Tast, arquivo auxiliar

Listagem 6.11: tast.ml - Código do tast - arquivo auxiliar

```
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 *
              tipo)
10        | ExpChamada of ident * (expressao expressoes) * tipo

```

6.1.9 Código do Tabsimb

Segue o código do Tabsimb, arquivo auxiliar

Listagem 6.12: tabsimb.ml - Código do tabsimb - arquivo auxiliar

```

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 }

```

Listagem 6.13: tabsimb.mli - Código do tabsimb - arquivo auxiliar

```

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

```

6.2 Usando o analisador semântico

6.2.1 Compilando o analisador semântico

Antes que se possa executar o analisador semântico, devemos configurar suas mensagens de erro e, então, compilá-lo. Para tal, vamos, primeiramente, gerar essas mensagens de erro, por meio do comando

```
$ menhir -v --list-errors sintatico.mly > sintatico.messages
```

Este código gerará um arquivo .messages contando os casos de erro e suas respectivas mensagens. Essas mensagens devem ser modificadas nesse arquivo neste momento com mensagens que são pertinentes a cada tipo de erro antes de prosseguirmos. Após isso, compile o arquivo de mensagens de erro para que o mesmo possa ser utilizado, por meio do seguinte comando

```
$ menhir -v --list-errors sintatico.mly --compile-errors sintatico.messages > fnmes.ml
```

Por fim, podemos compilar todo o projeto com o auxílio de ocamlbuild. Para tal, execute o seguinte comando

```
$ ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -package menhirLib semanticoTest.byte
```

Agora estamos prontos para utilizar o analisador semântico

6.2.2 Executando o analisador semântico

Para executar o analisador semântico, devemos, primeiro, entrar no ambiente Ocaml

```
$ rlwrap ocaml
```

Finalmente, podemos realizar a análise semântica de um arquivo .lua por meio da função no arquivo `semanticoTest.ml` *verifica_tipos* e passar o nome do arquivo a ser analisado entre parênteses

```
# verifica_tipos "nome_do_arquivo_a_ser_analisador.lua";;
```

Esse comando retornará o resultado do analisador semântico. Note que ainda é possível usar a função do analisador sintático do arquivo `sintaticoTest.ml`, a *parse_arq* para obter apenas a análise do analisador sintático.

6.3 Testando o Analisador Semântico

Os seguintes testes com os programas nano e micro modificados tem a finalidade de validar a corretude das árvores geradas pelo analisador semântico.

6.3.1 Programas nano

Nano01

Listagem 6.14: nano01.lua - Programa nano01 modificado

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

Saída do analisador semântico:

Listagem 6.15: nano01.txt - Resultado do analisador semântico executado sobre nano01.lua modificado

```
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 = []; fn_locais = []; fn_corpo =
          []}],
8   []),
9   <abstr>)
```

Nano02

Listagem 6.16: nano02.lua - Programa nano01 modificado

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

Saída do analisador semântico:

Listagem 6.17: nano02.txt - Resultado do analisador semântico executado sobre nano02.lua modificado

```

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              30}),
13            TipoFloat)];
14       fn_corpo = []]),
15   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
16 <abstr>)
```

Nano03

Listagem 6.18: nano03.lua - Programa nano03 modificado

```

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

Saída do analisador semântico:

Listagem 6.19: nano03.txt - Resultado do analisador semântico executado sobre nano03.lua modificado

```

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})),
```

```

20         TipoInt),
21         Tast.ExpInt (1, TipoInt));
22     CmdRetorno
23     (Some
24         (Tast.ExpVar
25             (VarSimples
26                 ("n",
27                     {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 40;
28                     pos_cnum = 51}),
29                 TipoInt))))],
30     [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
31     <abstr>)

```

Nano04

Listagem 6.20: nano04.lua - Programa nano04 modificado

```

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

```

Saída do analisador semântico:

Listagem 6.21: nano04.txt - Resultado do analisador semântico executado sobre nano04.lua modificado

```

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                             25}),
13                         TipoInt)];
14             fn_corpo =
15                 [CmdAtrib
16                     (Tast.ExpVar
17                         (VarSimples
18                             ("n",
19                                 {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 27;
20                                 pos_cnum = 28}),
21                                 TipoInt),
22                     Tast.ExpOp ((Adicao, TipoInt), (Tast.ExpInt (1, TipoInt), TipoInt)
23                                     ,
24                                     (Tast.ExpInt (2, TipoInt), TipoInt))));
25             CmdRetorno
26             (Some

```



```

25         (Tast.ExpVar
26         (VarSimples
27         ("n",
28         {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 39;
29         pos_cnum = 47})),
30         TipoInt)))]],
31 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
32 <abstr>)

```

Nano05

Listagem 6.22: nano05.lua - Programa nano05 modificado

```

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

```

Saída do analisador semântico:

Listagem 6.23: nano05.txt - Resultado do analisador semântico executado sobre nano05.lua modificado

```

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 (2, TipoInt));
23 CmdPrint
24 (Tast.ExpVar
25 (VarSimples
26 ("n",
27 {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 40;
28 pos_cnum = 50}),
29 TipoInt));
30 CmdRetorno
31 (Some

```

```

31         (Tast.ExpVar
32         (VarSimples
33         ("n",
34         {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 53;
35         pos_cnum = 64}),
36         TipoInt)))]],
37 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
38 <abstr>)

```

Nano06

Listagem 6.24: nano06.lua - Programa nano01 modificado

```

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

```

Saída do analisador semântico:

Listagem 6.25: nano06.txt - Resultado do analisador semântico executado sobre nano06.lua modificado

```

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));

```

```

31         CmdRetorno
32         (Some
33           (Tast.ExpVar
34             (VarSimples
35               ("n",
36                 {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 57;
37                   pos_cnum = 68}),
38                 TipoInt))))),
39   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
40   <abstr>)

```

Nano01

Listagem 6.26: nano07.lua - Programa nano07 modificado

```

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

```

Saída do analisador semântico:

Listagem 6.27: nano07.txt - Resultado do analisador semântico executado sobre nano07.lua modificado

```

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",

```

```

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 = 59;
37               pos_cnum = 73})),
38           TipoInt))] ,
39     None);
40 CmdRetorno
41 (Some
42   (Tast.ExpVar
43     (VarSimples
44       ("n",
45         {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 84;
46           pos_cnum = 95})),
47       TipoInt)))]],
48 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
49 <abstr>)

```

Nano08

Listagem 6.28: nano08.lua - Programa nano08 modificado

```

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   return n
10 end
11
12 main()

```

Saída do analisador semântico:

Listagem 6.29: nano08.txt - Resultado do analisador semântico executado sobre nano08.lua modificado

```

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})),

```

```

12     TipoInt)];
13     fn_corpo =
14     [CmdAtrib
15       (Tast.ExpVar
16         (VarSimples
17           ("n",
18             {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 30;
19              pos_cnum = 34})),
20         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 = 59;
37              pos_cnum = 73})),
38         TipoInt))],
39     Some [CmdPrint (Tast.ExpInt (0, TipoInt))]);
40     CmdRetorno
41     (Some
42       (Tast.ExpVar
43         (VarSimples
44           ("n",
45             {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 110;
46              pos_cnum = 121})),
47         TipoInt)))]],
48     [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
49     <abstr>)

```

Nano09

Listagem 6.30: nano09.lua - Programa nano01 modificado

```

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      return n
10 end
11
12 main()

```

Saída do analisador semântico:

Listagem 6.31: nano09.txt - Resultado do analisador semântico executado sobre nano09.lua modificado

```

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 ((Adicao, TipoInt), (Tast.ExpInt (1, TipoInt), TipoInt)
23            ,
24            (Tast.ExpOp ((Divisao, TipoInt),
25              (Tast.ExpInt (1, TipoInt), TipoInt),
26              (Tast.ExpInt (2, TipoInt), TipoInt)),
27            TipoInt))));
28    CmdIf
29      (Tast.ExpOp ((Equivalente, TipoBool),
30        (Tast.ExpVar
31          (VarSimples
32            ("n",
33              {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 48;
34                pos_cnum = 55}),
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 = 67;
43                pos_cnum = 81}),
44            TipoInt))],
45      Some [CmdPrint (Tast.ExpInt (0, TipoInt))]);
46    CmdRetorno
47      (Some
48        (Tast.ExpVar
49          (VarSimples
50            ("n",
51              {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 118;
52                pos_cnum = 129}),
53            TipoInt)))]],
54    [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
55    <abstr>)

```

Nano10

Listagem 6.32: nano10.lua - Programa nano01 modificado

```

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    return 1
12 end
13
14 main()

```

Saída do analisador semântico:

Listagem 6.33: nano10.txt - Resultado do analisador semântico executado sobre nano10.lua modificado

```

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 = 3; pos_bol = 30; pos_cnum =
17            38}),
18        TipoInt)];
19    fn_corpo =
20      [CmdAtrib
21        (Tast.ExpVar
22          (VarSimples
23            ("n",
24              {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 41;
25                pos_cnum = 45}),
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 = 52;
33                pos_cnum = 56}),
34            TipoInt),
35        Tast.ExpInt (2, TipoInt));

```

```

34     CmdIf
35     (Tast.ExpOp ((Equivalente, TipoBool),
36     (Tast.ExpVar
37     (VarSimples
38     ("n",
39     {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 62;
40     pos_cnum = 69})),
41     TipoInt),
42     TipoInt),
43     (Tast.ExpVar
44     (VarSimples
45     ("m",
46     {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 62;
47     pos_cnum = 74})),
48     TipoInt),
49     TipoInt)),
50     [CmdPrint
51     (Tast.ExpVar
52     (VarSimples
53     ("n",
54     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 81;
55     pos_cnum = 95})),
56     TipoInt)]],
57     Some [CmdPrint (Tast.ExpInt (0, TipoInt))]);
58     CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
59     [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
60     <abstr>)

```

Nano11

Listagem 6.34: nano11.lua - Programa nano11 modificado

```

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

```

Saída do analisador semântico:

Listagem 6.35: nano11.txt - Resultado do analisador semântico executado sobre nano11.lua modificado

```

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

```


6.3

```

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 = 3; pos_bol = 30; pos_cnum =
17                 38}),
18              TipoInt);
19         DecVar
20            (("x",
21               {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 40; pos_cnum =
22                 48}),
23              TipoInt)];
21  fn_corpo =
22      [CmdAtrib
23         (Tast.ExpVar
24            (VarSimples
25               ("n",
26                  {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 51;
27                    pos_cnum = 55}),
28                  TipoInt),
29            Tast.ExpInt (1, TipoInt));
30         CmdAtrib
31            (Tast.ExpVar
32               (VarSimples
33                  ("m",
34                     {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 61;
35                       pos_cnum = 65}),
36                     TipoInt),
37            Tast.ExpInt (2, TipoInt));
38         CmdAtrib
39            (Tast.ExpVar
40               (VarSimples
41                  ("x",
42                     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 71;
43                       pos_cnum = 75}),
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 = 81;
52                          pos_cnum = 91}),
53                        TipoInt),
54                  TipoInt),
55               (Tast.ExpVar
56                  (VarSimples
57                     ("n",
58                        {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 81;
59                          pos_cnum = 95}),
60                        TipoInt),
61                  TipoInt))),

```

```

62      [CmdAtrib
63        (Tast.ExpVar
64          (VarSimples
65            ("n",
66              {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 100;
67                pos_cnum = 108}),
68            TipoInt),
69        Tast.ExpOp ((Adicao, TipoInt),
70          (Tast.ExpVar
71            (VarSimples
72              ("n",
73                {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 100;
74                  pos_cnum = 112}),
75              TipoInt),
76            TipoInt),
77          (Tast.ExpVar
78            (VarSimples
79              ("m",
80                {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 100;
81                  pos_cnum = 116}),
82              TipoInt),
83            TipoInt)))];
84      CmdPrint
85        (Tast.ExpVar
86          (VarSimples
87            ("n",
88              {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol = 118;
89                pos_cnum = 132}),
90            TipoInt)))];
91      CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
92    [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
93    <abstr>)

```

Nano12

Listagem 6.36: nano12.lua - Programa nano12 modificado

```

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

```

Saída do analisador semântico:

```

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 = 3; pos_bol = 30; pos_cnum =
17            38}),
18        TipoInt);
19      DecVar
20        (("x",
21          {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 40; pos_cnum =
22            48}),
23        TipoInt)];
24   fn_corpo =
25     [CmdAtrib
26       (Tast.ExpVar
27         (VarSimples
28           ("n",
29             {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 50;
30               pos_cnum = 54}),
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 = 60;
38               pos_cnum = 64}),
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 = 70;
46               pos_cnum = 74}),
47           TipoInt),
48       Tast.ExpInt (5, TipoInt));
49     CmdWhile
50       (Tast.ExpOp ((Maior, TipoBool),
51         (Tast.ExpVar
52           (VarSimples
53             ("x",
54               {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 80;
55                 pos_cnum = 90}),
56             TipoInt),
57         TipoInt),

```

```

55     (Tast.ExpVar
56       (VarSimples
57         ("n",
58           {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 80;
59            pos_cnum = 94})),
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 = 99;
68            pos_cnum = 110})),
69       TipoInt),
70       TipoInt),
71     (Tast.ExpVar
72       (VarSimples
73         ("m",
74           {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 99;
75            pos_cnum = 115})),
76       TipoInt),
77       TipoInt))),
78 [CmdPrint
79   (Tast.ExpVar
80     (VarSimples
81       ("n",
82         {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol = 122;
83          pos_cnum = 140})),
84     TipoInt))],
85   Some [CmdPrint (Tast.ExpInt (0, TipoInt))]);
86 CmdAtrib
87   (Tast.ExpVar
88     (VarSimples
89       ("x",
90         {Lexing.pos_fname = ""; pos_lnum = 14; pos_bol = 189;
91          pos_cnum = 197})),
92     TipoInt),
93   Tast.ExpOp ((Subtracao, TipoInt),
94     (Tast.ExpVar
95       (VarSimples
96         ("x",
97           {Lexing.pos_fname = ""; pos_lnum = 14; pos_bol = 189;
98            pos_cnum = 201})),
99       TipoInt),
100     TipoInt),
101     (Tast.ExpInt (1, TipoInt), TipoInt)))]);
102   CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
103 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
104 <abstr>

```

6.3.2 Programas micro

micro01

```

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  return 1
11 end
12
13 main()

```

Saida do analisador sintático:

Listagem 6.39: micro01.txt - Resultado do analisador semântico executado sobre micro01.lua modificado

```

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        (("cel",
11          {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
12            25}),
13        TipoInt);
14        DecVar
15          (("far",
16            {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
17              30}),
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 = 134;
32                      pos_cnum = 135}),
33                  TipoInt));
34            CmdAtrib
35              (Tast.ExpVar
36                (VarSimples
37                  ("far",
38                    {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 155;
39                      pos_cnum = 156}),
40                  TipoInt),

```

```

37     Tast.ExpOp ((Divisao, TipoInt),
38     (Tast.ExpOp ((Adicao, TipoInt),
39     (Tast.ExpOp ((Multiplicacao, TipoInt),
40     (Tast.ExpInt (9, TipoInt), TipoInt),
41     (Tast.ExpVar
42     (VarSimples
43     ("cel",
44     {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 155;
45     pos_cnum = 165})),
46     TipoInt),
47     TipoInt)),
48     TipoInt),
49     (Tast.ExpInt (160, TipoInt), TipoInt)),
50     TipoInt),
51     (Tast.ExpInt (5, TipoInt), TipoInt)));
52 CmdPrint (Tast.ExpString ("A nova temperatura eh: ", TipoString));
53 CmdPrint
54 (Tast.ExpVar
55 (VarSimples
56 ("far",
57 {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 210;
58 pos_cnum = 217})),
59 TipoInt));
60 CmdPrint (Tast.ExpString (" F\n", TipoString));
61 CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
62 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
63 <abstr>)

```

micro02

Listagem 6.40: micro02.lua - Programa micro02 modificado

```

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 número ")
11     print(num1)
12     print(" é maior que o segundo ")
13     print(num2)
14     else
15         print("O segundo número ")
16     print(num2)
17     print(" é maior que o primeiro ")
18     print(num1)
19     end
20     return 1
21 end
22
23 main()

```

Saida do analisador sintático:

Listagem 6.41: micro02.txt - Resultado do analisador semântico executado sobre micro02.lua modificado

```

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",
16            {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 33; pos_cnum =
17              41}),
18          TipoInt)];
19        fn_corpo =
20          [CmdPrint (Tast.ExpString ("Digite o primeiro numero: ", TipoString)
21            );
22          CmdScanInt
23            (Tast.ExpVar
24              (VarSimples
25                ("num1",
26                  {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 86;
27                    pos_cnum = 90}),
28                TipoInt));
29          CmdPrint (Tast.ExpString ("Digite o segundo numero: ", TipoString))
30            ;
31          CmdScanInt
32            (Tast.ExpVar
33              (VarSimples
34                ("num2",
35                  {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 150;
36                    pos_cnum = 154}),
37                TipoInt));
38          CmdIf
39            (Tast.ExpOp ((Maior, TipoBool),
40              (Tast.ExpVar
41                (VarSimples
42                  ("num1",
43                    {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 176;
44                      pos_cnum = 183}),
45                  TipoInt),
46                TipoInt),
47              (Tast.ExpVar
48                (VarSimples
49                  ("num2",
50                    {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 176;
51                      pos_cnum = 190}),
52                  TipoInt),
53                TipoInt));
54          [CmdPrint (Tast.ExpString ("O primeiro n\u00b0mero ", TipoString)
55            );
56          CmdPrint
57            (Tast.ExpVar

```

```

53         (VarSimples
54           ("num1",
55             {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 237;
56               pos_cnum = 244}),
57           TipoInt));
58     CmdPrint
59     (Tast.ExpString (" \195\169 maior que o segundo ", TipoString));
60     CmdPrint
61     (Tast.ExpVar
62       (VarSimples
63         ("num2",
64           {Lexing.pos_fname = ""; pos_lnum = 13; pos_bol = 285;
65             pos_cnum = 292}),
66         TipoInt))] ,
67     Some
68     [CmdPrint (Tast.ExpString ("O segundo n\195\186mero ", TipoString
69       ));
70       CmdPrint
71       (Tast.ExpVar
72         (VarSimples
73           ("num2",
74             {Lexing.pos_fname = ""; pos_lnum = 16; pos_bol = 343;
75               pos_cnum = 350}),
76           TipoInt));
77       CmdPrint
78       (Tast.ExpString (" \195\169 maior que o primeiro ", TipoString)
79         );
80       CmdPrint
81       (Tast.ExpVar
82         (VarSimples
83           ("num1",
84             {Lexing.pos_fname = ""; pos_lnum = 18; pos_bol = 392;
85               pos_cnum = 399}),
86           TipoInt))] );
87     CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
88   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
89   <abstr>)

```

micro03

Listagem 6.42: micro03.lua - Programa micro03 modificado

```

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\n")
8      else
9        print("O numero nao esta no intervalo entre 100 e 200\n")
10     end
11   else
12     print("O numero nao esta no intervalo entre 100 e 200\n")
13   end
14   return 1
15 end
16

```



```
17 main()
```

Saida do analisador sintático:

Listagem 6.43: micro03.txt - Resultado do analisador semântico executado sobre micro03.lua modificado

```
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 = 94;
29         pos_cnum = 101}),
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 = 120;
39         pos_cnum = 131}),
40     TipoInt),
41     TipoInt),
42   (Tast.ExpInt (200, TipoInt), TipoInt)),
43   [CmdPrint
44   (Tast.ExpString ("O numero esta no intervalo entre 100 e 200\n",
45     TipoString))],
46   Some
47   [CmdPrint
48   (Tast.ExpString
49     ("O numero nao esta no intervalo entre 100 e 200\n",
50     TipoString))]]],
51   Some
```

```

51         [CmdPrint
52           (Tast.ExpString
53            ("O numero nao esta no intervalo entre 100 e 200\n",
              TipoString))]];
54     CmdRetorno (Some (Tast.ExpInt (1, TipoInt))))]],
55   [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]],
56   <abstr>)

```

micro04

Listagem 6.44: micro04.lua - Programa micro04 modificado

```

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

```

Saida do analisador sintático:

Listagem 6.45: micro04.txt - Resultado do analisador semântico executado sobre micro04.lua modificado

```

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 = 2; pos_bol = 20; pos_cnum =
12              28}),
13          TipoInt);
14          DecVar
15            (("num",
16              {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
17                31}),
18            TipoInt);
19          DecVar
20            (("intervalo",

```

6.3

```

19         {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
20             36}},
21     TipoInt)];
22     fn_corpo =
23     [CmdAtrib
24         (Tast.ExpVar
25             (VarSimples
26                 ("intervalo",
27                     {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 46;
28                         pos_cnum = 50})),
29             TipoInt),
30         Tast.ExpInt (0, TipoInt));
31     CmdIf (Tast.ExpBool (true, TipoBool),
32         [CmdAtrib
33             (Tast.ExpVar
34                 (VarSimples
35                     ("x",
36                         {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 64;
37                             pos_cnum = 72})),
38                 TipoInt),
39             Tast.ExpInt (1, TipoInt));
40     CmdWhile
41     (Tast.ExpOp ((Menor_ou_Igual, TipoBool),
42         (Tast.ExpVar
43             (VarSimples
44                 ("x",
45                     {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 64;
46                         pos_cnum = 72})),
47                 TipoInt),
48             TipoInt),
49         (Tast.ExpInt (5, TipoInt), TipoInt)),
50     [CmdPrint (Tast.ExpString ("Digite um numero: ", TipoString));
51     CmdScanInt
52     (Tast.ExpVar
53         (VarSimples
54             ("num",
55                 {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 127;
56                     pos_cnum = 135})),
57         TipoInt));
58     CmdIf
59     (Tast.ExpOp ((Maior_ou_Igual, TipoBool),
60         (Tast.ExpVar
61             (VarSimples
62                 ("num",
63                     {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 155;
64                         pos_cnum = 166})),
65                 TipoInt),
66             TipoInt),
67         (Tast.ExpInt (10, TipoInt), TipoInt)),
68     [CmdIf
69     (Tast.ExpOp ((Menor_ou_Igual, TipoBool),
70         (Tast.ExpVar
71             (VarSimples
72                 ("num",
73                     {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 181;
74                         pos_cnum = 196})),
75                 TipoInt),
76                 TipoInt),
77         (Tast.ExpInt (150, TipoInt), TipoInt)),

```

```

77         [CmdAtrib
78             (Tast.ExpVar
79                 (VarSimples
80                     ("intervalo",
81                         {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol = 212;
82                             pos_cnum = 228}),
83                     TipoInt),
84             Tast.ExpOp ((Adicao, TipoInt),
85                 (Tast.ExpVar
86                     (VarSimples
87                         ("intervalo",
88                             {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol =
89                                 212;
90                                 pos_cnum = 240}),
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 = 4; pos_bol = 64;
100                     pos_cnum = 72}),
101             TipoInt),
102     Tast.ExpOp ((Adicao, TipoInt),
103         (Tast.ExpVar
104             (VarSimples
105                 ("x",
106                     {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 64;
107                         pos_cnum = 72}),
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 = 15; pos_bol = 330;
119                     pos_cnum = 340}),
120             TipoInt));
121 CmdPrint
122     (Tast.ExpString (" numeros no intervalo entre 10 e 150",
123         TipoString));
124 CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
125 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
126 <abstr>

```

micro05

Listagem 6.46: micro05.lua - Programa micro05 modificado

```

1 function int main()
2     string nome, sexo

```

```

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

```

Saida do analisador sintático:

Listagem 6.47: micro05.txt - Resultado do analisador semântico executado sobre micro05.lua modificado

```

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 = 2; pos_bol = 20; pos_cnum =
17       37}),
18     TipoString);
19     DecVar
20     (("x",
21     {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 42; pos_cnum =
22       50}),
23     TipoInt);
24     DecVar

```

```

22     ("h",
23     {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 42; pos_cnum =
        53}),
24     TipoInt);
25 DecVar
26     ("m",
27     {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 42; pos_cnum =
        56}),
28     TipoInt)];
29 fn_corpo =
30 [CmdAtrib
31     (Tast.ExpVar
32     (VarSimples
33     ("x",
34     {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 58;
35     pos_cnum = 62}),
36     TipoInt),
37     Tast.ExpInt (1, TipoInt));
38 CmdAtrib
39     (Tast.ExpVar
40     (VarSimples
41     ("h",
42     {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 68;
43     pos_cnum = 72}),
44     TipoInt),
45     Tast.ExpInt (0, TipoInt));
46 CmdAtrib
47     (Tast.ExpVar
48     (VarSimples
49     ("m",
50     {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 78;
51     pos_cnum = 82}),
52     TipoInt),
53     Tast.ExpInt (0, TipoInt));
54 CmdIf (Tast.ExpBool (true, TipoBool),
55 [CmdAtrib
56     (Tast.ExpVar
57     (VarSimples
58     ("x",
59     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 88;
60     pos_cnum = 96}),
61     TipoInt),
62     Tast.ExpInt (1, TipoInt));
63 CmdWhile
64     (Tast.ExpOp ((Menor_ou_Igual, TipoBool),
65     (Tast.ExpVar
66     (VarSimples
67     ("x",
68     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 88;
69     pos_cnum = 96}),
70     TipoInt),
71     TipoInt),
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 = 9; pos_bol = 144;

```

```

79         pos_cnum = 152)),
80     TipoString));
81 CmdPrint
82     (Tast.ExpString ("H - Homem ou M - Mulher: ", TipoString));
83 CmdScanString
84     (Tast.ExpVar
85         (VarSimples
86             ("sexo",
87                 {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 216;
88                     pos_cnum = 224})),
89         TipoString));
90 CmdIf
91     (Tast.ExpOp ((Equivalente, TipoBool),
92         (Tast.ExpVar
93             (VarSimples
94                 ("sexo",
95                     {Lexing.pos_fname = ""; pos_lnum = 12; pos_bol = 245;
96                         pos_cnum = 256})),
97                 TipoString),
98                 TipoString),
99         (Tast.ExpString ("H", TipoString), TipoString))),
100 [CmdAtrib
101     (Tast.ExpVar
102         (VarSimples
103             ("h",
104                 {Lexing.pos_fname = ""; pos_lnum = 13; pos_bol = 273;
105                     pos_cnum = 285})),
106                 TipoInt),
107     Tast.ExpOp ((Adicao, TipoInt),
108         (Tast.ExpVar
109             (VarSimples
110                 ("h",
111                     {Lexing.pos_fname = ""; pos_lnum = 13; pos_bol = 273;
112                         pos_cnum = 289})),
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 = 15; pos_bol =
123                             308;
124                             pos_cnum = 316})),
125                         TipoString),
126                         TipoString),
127                     (Tast.ExpString ("M", TipoString), TipoString))),
128         [CmdAtrib
129             (Tast.ExpVar
130                 (VarSimples
131                     ("m",
132                         {Lexing.pos_fname = ""; pos_lnum = 16; pos_bol =
133                             333;
134                             pos_cnum = 346})),
135                         TipoInt),
136             Tast.ExpOp ((Adicao, TipoInt), (Tast.ExpVar (...), ...),
137                 ...))];

```

```

136         ...],
137         ...);
138         ...]);
139         ...]);
140         ...],
141         ...);
142         ...] });
143     ...],
144     ...),
145     ...)

```

micro06

Listagem 6.48: micro06.lua - Programa micro06 modificado

```

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

```

Saida do analisador sintático:

Listagem 6.49: micro06.txt - Resultado do analisador semântico executado sobre micro06.lua modificado

```

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}),

```



```

12     TipoInt));
13 fn_corpo =
14 [CmdPrint (Tast.ExpString ("Digite um numero de 1 a 5: ", TipoString
15     ));
16     CmdScanInt
17     (Tast.ExpVar
18     (VarSimples
19     ("numero",
20     {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 76;
21     pos_cnum = 80}),
22     TipoInt));
23 CmdIf
24 (Tast.ExpOp ((Equivalente, TipoBool),
25     (Tast.ExpVar
26     (VarSimples
27     ("numero",
28     {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 103;
29     pos_cnum = 110}),
30     TipoInt),
31     TipoInt),
32     (Tast.ExpInt (1, TipoInt), TipoInt))),
33 [CmdPrint (Tast.ExpString ("Um\n", TipoString))], None);
34 CmdIf
35 (Tast.ExpOp ((Equivalente, TipoBool),
36     (Tast.ExpVar
37     (VarSimples
38     ("numero",
39     {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 157;
40     pos_cnum = 164}),
41     TipoInt),
42     TipoInt),
43     (Tast.ExpInt (2, TipoInt), TipoInt))),
44 [CmdPrint (Tast.ExpString ("Dois\n", TipoString))], None);
45 CmdIf
46 (Tast.ExpOp ((Equivalente, TipoBool),
47     (Tast.ExpVar
48     (VarSimples
49     ("numero",
50     {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 213;
51     pos_cnum = 220}),
52     TipoInt),
53     TipoInt),
54     (Tast.ExpInt (3, TipoInt), TipoInt))),
55 [CmdPrint (Tast.ExpString ("Tres\n", TipoString))], None);
56 CmdIf
57 (Tast.ExpOp ((Equivalente, TipoBool),
58     (Tast.ExpVar
59     (VarSimples
60     ("numero",
61     {Lexing.pos_fname = ""; pos_lnum = 14; pos_bol = 269;
62     pos_cnum = 276}),
63     TipoInt),
64     TipoInt),
65     (Tast.ExpInt (4, TipoInt), TipoInt))),
66 [CmdPrint (Tast.ExpString ("Quatro\n", TipoString))], None);
67 CmdIf
68 (Tast.ExpOp ((Equivalente, TipoBool),
69     (Tast.ExpVar
70     (VarSimples

```

```

70         ("numero",
71         {Lexing.pos_fname = ""; pos_lnum = 17; pos_bol = 327;
72         pos_cnum = 334}),
73         TipoInt),
74         TipoInt),
75         (Tast.ExpInt (5, TipoInt), TipoInt)),
76         [CmdPrint (Tast.ExpString ("Cinco\n", TipoString))],
77         Some [CmdPrint (Tast.ExpString ("Numero invalido!!!", TipoString))
78         ]]);
79     CmdRetorno (Some (Tast.ExpInt (1, TipoInt))))],
80 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
81 <abstr>)

```

micro07

Listagem 6.50: micro07.lua - Programa micro07 modificado

```

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

```

Saida do analisador sintático:

Listagem 6.51: micro07.txt - Resultado do analisador semântico executado sobre micro07.lua modificado

```

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 =
          28}),
12      TipoInt);
13  DecVar
14    ("numero",
15      {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
          38}),
16      TipoInt);
17  DecVar
18    ("opc",
19      {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
          46}),
20      TipoInt)];
21  fn_corpo =
22    [CmdAtrib
23      (Tast.ExpVar
24        (VarSimples
25          ("programa",
26            {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 50;
              pos_cnum = 54}),
27            TipoInt),
28          Tast.ExpInt (1, TipoInt));
29      CmdWhile
30        (Tast.ExpOp ((Equivalente, TipoBool),
31          (Tast.ExpVar
32            (VarSimples
33              ("programa",
34                {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 67;
                  pos_cnum = 77}),
35                TipoInt),
36                TipoInt),
37                (Tast.ExpInt (1, TipoInt), TipoInt)),
38                [CmdPrint (Tast.ExpString ("Digite um numero: ", TipoString));
39                CmdScanInt
40                  (Tast.ExpVar
41                    (VarSimples
42                      ("numero",
43                        {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 130;
                          pos_cnum = 138}),
44                        TipoInt));
45                  CmdIf
46                    (Tast.ExpOp ((Maior, TipoBool),
47                      (Tast.ExpVar
48                        (VarSimples
49                          ("numero",
50                            {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 161;
                              pos_cnum = 172}),
51                            TipoInt),
52                            TipoInt),
53                            (Tast.ExpInt (0, TipoInt), TipoInt)),
54                            [CmdPrint (Tast.ExpString ("Positivo\n", TipoString))],
55                            Some
56                              [CmdIf
57                                (Tast.ExpOp ((Equivalente, TipoBool),
58                                  (Tast.ExpVar
59                                    (VarSimples
60                                      ("numero",
61                                        {Lexing.pos_fname = ""; pos_lnum = 10; pos_bol = 233;

```

```

66         pos_cnum = 248}},
67         TipoInt),
68         TipoInt),
69         (Tast.ExpInt (0, TipoInt), TipoInt)),
70     [CmdPrint
71       (Tast.ExpString ("O numero e igual a 0\n", TipoString))],
72     None);
73 CmdIf
74   (Tast.ExpOp ((Menor, TipoBool),
75     (Tast.ExpVar
76       (VarSimples
77         ("numero",
78           {Lexing.pos_fname = ""; pos_lnum = 13; pos_bol = 329;
79             pos_cnum = 344}),
80         TipoInt),
81         TipoInt),
82         (Tast.ExpInt (0, TipoInt), TipoInt))),
83     [CmdPrint (Tast.ExpString ("Negativo\n", TipoString))], None)
84   ];
85 CmdPrint
86   (Tast.ExpString ("Deseja finalizar? (S - 1): ", TipoString));
87 CmdScanInt
88   (Tast.ExpVar
89     (VarSimples
90       ("opc",
91         {Lexing.pos_fname = ""; pos_lnum = 18; pos_bol = 469;
92           pos_cnum = 477}),
93       TipoInt));
94 CmdIf
95   (Tast.ExpOp ((Equivalente, TipoBool),
96     (Tast.ExpVar
97       (VarSimples
98         ("opc",
99           {Lexing.pos_fname = ""; pos_lnum = 19; pos_bol = 497;
100             pos_cnum = 508}),
101         TipoInt),
102         TipoInt),
103         (Tast.ExpInt (1, TipoInt), TipoInt))),
104     [CmdAtrib
105       (Tast.ExpVar
106         (VarSimples
107           ("programa",
108             {Lexing.pos_fname = ""; pos_lnum = 20; pos_bol = 522;
109               pos_cnum = 534}),
110           TipoInt),
111         Tast.ExpInt (0, TipoInt))],
112     None)];
113 CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
114 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
115 <abstr>

```

micro08

Listagem 6.52: micro08.lua - Programa micro08 modificado

```

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(" e maior que 10\n")
11      else
12          print("O numero ")
13          print(numero)
14      print(" e menor que 10\n")
15      end
16  end
17  return 1
18 end
19
20 main()

```

Saida do analisador sintático:

Listagem 6.53: micro08.txt - Resultado do analisador semântico executado sobre micro08.lua modificado

```

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

```

```

36         ("numero",
37         {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 111;
38         pos_cnum = 119}),
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 = 142;
46 pos_cnum = 153}),
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 = 201;
56 pos_cnum = 213}),
57 TipoInt));
58 CmdPrint (Tast.ExpString (" e 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 = 297;
66 pos_cnum = 315}),
67 TipoInt));
68 CmdPrint (Tast.ExpString (" e menor que 10\n", TipoString)))]
69 ];
70 CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
71 [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))],
72 <abstr>

```

micro09

Listagem 6.54: micro09.lua - Programa micro09 modificado

```

1 function int main()
2     int preco, venda, novo_preco
3     print("Digite o preco: ")
4     preco = io.read('*n')
5     print("Digite a venda: ")
6     venda = io.read('*n')
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 < 80)
10         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
end
end

```

```

16     print("O novo preco e ")
17     print(novo_preco)
18     print("\n")
19     return 1
20 end
21
22 main()

```

Saida do analisador sintático:

Listagem 6.55: micro09.txt - Resultado do analisador semântico executado sobre micro09.lua modificado

```

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            28}),
13        TipoInt);
14      DecVar
15        (("venda",
16          {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
17            35}),
18        TipoInt);
19      DecVar
20        (("novo_preco",
21          {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
22            42}),
23        TipoInt)];
24    fn_corpo =
25    [CmdPrint (Tast.ExpString ("Digite o preco: ", TipoString));
26    CmdScanInt
27      (Tast.ExpVar
28        (VarSimples
29          ("preco",
30            {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 83;
31              pos_cnum = 87}),
32        TipoInt));
33    CmdPrint (Tast.ExpString ("Digite a venda: ", TipoString));
34    CmdScanInt
35      (Tast.ExpVar
36        (VarSimples
37          ("venda",
38            {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 139;
39              pos_cnum = 143}),
40        TipoInt));
41    CmdIf
42      (Tast.ExpOp ((Or, TipoBool),
43        (Tast.ExpOp ((Menor, TipoBool),
44          (Tast.ExpVar
45            (VarSimples
46              ("venda",

```

```

44         {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 165;
45         pos_cnum = 172}},
46         TipoInt),
47         TipoInt),
48         (Tast.ExpInt (500, TipoInt), TipoInt)),
49         TipoBool),
50     (Tast.ExpOp ((Menor, TipoBool),
51     (Tast.ExpVar
52     (VarSimples
53     ("preco",
54     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 165;
55     pos_cnum = 187}},
56     TipoInt),
57     TipoInt),
58     (Tast.ExpInt (30, TipoInt), TipoInt)),
59     TipoBool)),
60 [CmdAtrib
61     (Tast.ExpVar
62     (VarSimples
63     ("novo_preco",
64     {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 203;
65     pos_cnum = 211}},
66     TipoInt),
67     Tast.ExpOp ((Adicao, TipoInt),
68     (Tast.ExpVar
69     (VarSimples
70     ("preco",
71     {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 203;
72     pos_cnum = 224}},
73     TipoInt),
74     TipoInt),
75     (Tast.ExpOp ((Multiplicacao, TipoInt),
76     (Tast.ExpOp ((Divisao, TipoInt),
77     (Tast.ExpInt (10, TipoInt), TipoInt),
78     (Tast.ExpInt (100, TipoInt), TipoInt)),
79     TipoInt),
80     (Tast.ExpVar
81     (VarSimples
82     ("preco",
83     {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 203;
84     pos_cnum = 243}},
85     TipoInt),
86     TipoInt)),
87     TipoInt))]],
88 Some
89 [CmdIf
90     (Tast.ExpOp ((Or, TipoBool),
91     (Tast.ExpOp ((And, TipoBool),
92     (Tast.ExpOp ((Maior_ou_Igual, TipoBool),
93     (Tast.ExpVar
94     (VarSimples
95     ("venda",
96     {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 249;
97     pos_cnum = 262}},
98     TipoInt),
99     TipoInt),
100     (Tast.ExpInt (500, TipoInt), TipoInt)),
101     TipoBool),
102     (Tast.ExpOp ((Menor_ou_Igual, TipoBool),

```



```

103         (Tast.ExpVar
104             (VarSimples
105                 ("venda",
106                     {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 249;
107                       pos_cnum = 279})),
108                 TipoInt),
109                 TipoInt),
110             (Tast.ExpInt (1200, TipoInt), TipoInt)),
111             TipoBool)),
112             TipoBool),
113             (Tast.ExpOp ((And, TipoBool),
114                 (Tast.ExpOp ((Maior_ou_Igual, TipoBool),
115                     (Tast.ExpVar
116                         (VarSimples
117                             ("preco",
118                                 {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 249;
119                                   pos_cnum = 298})),
120                                 TipoInt),
121                                 TipoInt),
122                                 (Tast.ExpInt (30, TipoInt), TipoInt)),
123                                 TipoBool),
124                                 (Tast.ExpOp ((Menor, TipoBool), (...), (...), (...)),
125                                     ...))),
126                 ...));
127             ...]);
128         ...]};
129     ...],
130     ...),
131     ...)

```

micro10

Listagem 6.56: micro10.lua - Programa micro10 modificado

```

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     int numero, fat
15     print("Digite um numero: ")
16     numero = io.read('*n')
17     fat = fatorial(numero)
18     print("O fatorial de ")
19     print("numero")
20     print(" e ")
21     print(fat)
22
23     return 1
24 end

```

```

25
26 main()

```

Saida do analisador sintático:

Listagem 6.57: micro10.txt - Resultado do analisador semântico executado sobre micro10.lua modificado

```

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               34}),
18             TipoInt);
19           DecVar
20             (("y",
21               {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 29; pos_cnum =
22                 37}),
23               TipoInt)];
24       fn_corpo =
25         [CmdIf
26           (Tast.ExpOp ((Menor_ou_Igual, TipoBool),
27             (Tast.ExpVar
28               (VarSimples
29                 ("n",
30                   {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 41;
31                     pos_cnum = 45}),
32                   TipoInt),
33                   TipoInt),
34                   (Tast.ExpInt (1, TipoInt), TipoInt))),
35           [CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))],
36           Some
37             [CmdAtrib
38               (Tast.ExpVar
39                 (VarSimples
40                   ("y",
41                     {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 74;
42                       pos_cnum = 76}),
43                     TipoInt),
44                   Tast.ExpOp ((Subtracao, TipoInt),
45                     (Tast.ExpVar
46                       (VarSimples
47                         ("n",
48                           {Lexing.pos_fname = ""; pos_lnum = 7; pos_bol = 74;

```

```

49      (Tast.ExpInt (1, TipoInt), TipoInt)));
50  CmdAtrib
51      (Tast.ExpVar
52          (VarSimples
53              ("x",
54                  {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 86;
55                     pos_cnum = 88})),
56              TipoInt),
57      Tast.ExpChamada ("fatorial",
58          [Tast.ExpVar
59              (VarSimples
60                  ("y",
61                      {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 86;
62                         pos_cnum = 102})),
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 = 105;
72                             pos_cnum = 114})),
73                      TipoInt),
74                      TipoInt),
75              (Tast.ExpVar
76                  (VarSimples
77                      ("x",
78                          {Lexing.pos_fname = ""; pos_lnum = 9; pos_bol = 105;
79                             pos_cnum = 118})),
80                      TipoInt),
81                      TipoInt))))));
82  DecFun
83      {fn_nome =
84          ("main",
85              {Lexing.pos_fname = ""; pos_lnum = 13; pos_bol = 130; pos_cnum =
86                 143});
86      fn_tiporet = TipoInt; fn_formais = [];
87      fn_locais =
88          [DecVar
89              (("numero",
90                  {Lexing.pos_fname = ""; pos_lnum = 14; pos_bol = 150;
91                     pos_cnum = 155})),
92              TipoInt);
93          DecVar
94              (("fat",
95                  {Lexing.pos_fname = ""; pos_lnum = 14; pos_bol = 150;
96                     pos_cnum = 163})),
97              TipoInt)];
98      fn_corpo =
99          [CmdPrint (Tast.ExpString ("Digite um numero: ", TipoString));
100          CmdScanInt
101              (Tast.ExpVar
102                  (VarSimples
103                      ("numero",
104                          {Lexing.pos_fname = ""; pos_lnum = 16; pos_bol = 196;
105                             pos_cnum = 197})),
106                  TipoInt));

```

```

107     CmdAtrib
108     (Tast.ExpVar
109     (VarSimples
110     ("fat",
111     {Lexing.pos_fname = ""; pos_lnum = 17; pos_bol = 220;
112     pos_cnum = 221}),
113     TipoInt),
114     Tast.ExpChamada ("fatorial",
115     [Tast.ExpVar
116     (VarSimples
117     ("numero",
118     {Lexing.pos_fname = ""; pos_lnum = 17; pos_bol = 220;
119     pos_cnum = 236}),
120     TipoInt)],
121     TipoInt));
122     CmdPrint (Tast.ExpString ("O fatorial de ", TipoString));
123     CmdPrint (Tast.ExpString ("numero", TipoString));
124     CmdPrint (Tast.ExpString (" e ", TipoString));
125     CmdPrint
126     (Tast.ExpVar
127     (VarSimples
128     ("fat",
129     {Lexing.pos_fname = ""; pos_lnum = 21; pos_bol = 300;
130     pos_cnum = 307}),
131     TipoInt));
132     CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
133     [CmdChamada (Tast.ExpChamada ("main", [], TipoInt))]),
134     <abstr>)

```

micro11

Listagem 6.58: micro11.lua - Programa micro11 modificado

```

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

```

```

26     end
27     return res
28 end
29
30 main()

```

Saida do analisador sintático:

Listagem 6.59: micro11.txt - Resultado do analisador semântico executado sobre micro11.lua modificado

```

1 #quit- : 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      DecVar
15        (("x",
16          {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol = 20; pos_cnum =
17            36}),
18        TipoInt)];
19      fn_corpo =
20        [CmdPrint (Tast.ExpString ("Digite um numero ", TipoString));
21        CmdScanInt
22          (Tast.ExpVar
23            (VarSimples
24              ("numero",
25                {Lexing.pos_fname = ""; pos_lnum = 4; pos_bol = 69;
26                  pos_cnum = 73}),
27            TipoInt));
28        CmdAtrib
29          (Tast.ExpVar
30            (VarSimples
31              ("x",
32                {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 96;
33                  pos_cnum = 100}),
34            TipoInt),
35          Tast.ExpChamada ("verifica",
36            [Tast.ExpVar
37              (VarSimples
38                ("numero",
39                  {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 96;
40                    pos_cnum = 113}),
41              TipoInt)],
42            TipoInt));
43        CmdIf
44          (Tast.ExpOp ((Equivalente, TipoBool),
45            (Tast.ExpVar
46              (VarSimples
47                ("x",
48                  {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol = 121;

```

```

47         pos_cnum = 128)),
48         TipoInt),
49         TipoInt),
50         (Tast.ExpInt (1, TipoInt), TipoInt)),
51 [CmdPrint (Tast.ExpString ("Numero positivo\n", TipoString))],
52 Some
53 [CmdIf
54   (Tast.ExpOp ((Equivalente, TipoBool),
55   (Tast.ExpVar
56     (VarSimples
57       ("x",
58       {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 175;
59       pos_cnum = 187})),
60     TipoInt),
61     TipoInt),
62     (Tast.ExpInt (0, TipoInt), TipoInt)),
63 [CmdPrint (Tast.ExpString ("Zero\n", TipoString))],
64 Some [CmdPrint (Tast.ExpString ("Numero negativo\n", TipoString
65   ))]]];
66 CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]];
67 DecFun
68 {fn_nome =
69   ("verifica",
70   {Lexing.pos_fname = ""; pos_lnum = 17; pos_bol = 301; pos_cnum =
71     314});
72   fn_tiporet = TipoInt;
73   fn_formais =
74   [(("n",
75     {Lexing.pos_fname = ""; pos_lnum = 17; pos_bol = 301;
76     pos_cnum = 327})),
77     TipoInt)];
78   fn_locais =
79   [DecVar
80     (("res",
81     {Lexing.pos_fname = ""; pos_lnum = 18; pos_bol = 330;
82     pos_cnum = 338})),
83     TipoInt)];
84   fn_corpo =
85   [CmdIf
86     (Tast.ExpOp ((Maior, TipoBool),
87     (Tast.ExpVar
88       (VarSimples
89         ("n",
90         {Lexing.pos_fname = ""; pos_lnum = 19; pos_bol = 342;
91         pos_cnum = 349})),
92       TipoInt),
93       TipoInt),
94       (Tast.ExpInt (0, TipoInt), TipoInt)),
95 [CmdAtrib
96   (Tast.ExpVar
97     (VarSimples
98       ("res",
99       {Lexing.pos_fname = ""; pos_lnum = 20; pos_bol = 360;
100       pos_cnum = 368})),
101     TipoInt),
102     Tast.ExpInt (1, TipoInt))],
103 Some
104 [CmdIf
105   (Tast.ExpOp ((Menor, TipoBool),

```

```

104         (Tast.ExpVar
105         (VarSimples
106         ("n",
107         {Lexing.pos_fname = ""; pos_lnum = 21; pos_bol = 376;
108         pos_cnum = 388})),
109         TipoInt),
110         TipoInt),
111         (Tast.ExpInt (0, TipoInt), TipoInt)),
112     [CmdAtrib
113     (Tast.ExpVar
114     (VarSimples
115     ("res",
116     {Lexing.pos_fname = ""; pos_lnum = 22; pos_bol = 399;
117     pos_cnum = 407})),
118     TipoInt),
119     Tast.ExpInt (-1, TipoInt))],
120     Some
121     [CmdAtrib
122     (Tast.ExpVar
123     (VarSimples
124     ("res",
125     {Lexing.pos_fname = ""; pos_lnum = 24; pos_bol = 425;
126     pos_cnum = 433})),
127     TipoInt),
128     Tast.ExpInt (0, TipoInt))]]]);
129     CmdRetorno (Some (Tast.ExpVar (VarSimples ...))); ...]);
130     ...],
131     ...),
132     ...)

```

6.4 Testes de Erros Semânticos

A seguir são alguns dos erros semânticos que podem ocorrer e gerar uma exceção pelo analisador semântico

Variável declarada duas vezes com tipos diferentes

Listagem 6.60: teste01.lua - Programa de teste 01

```

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

```

Saída do analisador:

Listagem 6.61: teste01.txt - Resultado do analisador semântico executado sobre teste01.lua modificado

```

1 Exception: Tabsimb.Entrada_existente "x".

```

Atribuição de um inteiro com um valor float

Listagem 6.62: teste02.lua - Programa de teste 02

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

Saída do analisador:

Listagem 6.63: teste02.txt - Resultado do analisador semântico executado sobre teste02.lua modificado

```
1 Exception:
2 Failure
3 "Semantico -> linha 4, coluna 6: (Aritmetico) O operando esquerdo eh do
   tipo float mas o direito eh do tipo inteiro".
```

Operação envolvendo tipos diferentes

Listagem 6.64: teste03.lua - Programa de teste 03

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

Saída do analisador:

Listagem 6.65: teste03.txt - Resultado do analisador semântico executado sobre teste03.lua modificado

```
1 Exception:
2 Failure
3 "Semantico -> linha 8, coluna 5: (Relacional) O operando esquerdo eh do
   tipo inteiro mas o direito eh do tipo float".
```

Chamada de função com número incorreto de parâmetros

Listagem 6.66: teste04.lua - Programa de teste 04

```

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

```

Saída do analisador:

Listagem 6.67: teste04.txt - Resultado do analisador semântico executado sobre teste04.lua modificado

```

1 Exception:
2 Failure "Semantico -> linha 11, coluna -1: Numero incorreto de parametros"
.

```

Declaração de parâmetros iguais de uma função

Listagem 6.68: teste05.lua - Programa de teste 05

```

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

```

Saída do analisador:

Listagem 6.69: teste05.txt - Resultado do analisador semântico executado sobre teste05.lua modificado

```

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

```

Tipo de retorno diferente do tipo declarado da função

Listagem 6.70: teste06.lua - Programa de teste 06

```

1 function int main()
2     int x
3     float y
4     x = 10
5     y = 20.0
6     return y
7 end

```

```
8  
9 main()
```

Saída do analisador:

Listagem 6.71: teste06.txt - Resultado do analisador semântico executado sobre teste06.lua modificado

```
1 Exception:  
2 Failure  
3 "Semantico -> linha 6, coluna 7: O tipo retornado eh float mas foi  
   declarado como inteiro".
```

Capítulo 7

Intérprete

O interpretador tem a tarefa de interpretar e executar a saída do analisador semântico. Para tal, seguem os arquivos necessários para o interpretador do analisador semântico da linguagem MiniLua modificada.

7.1 Compilando e Executando

Primeiramente, para compilar o `interpreteTest.ml`, usaremos o *ocamlbuild* como anteriormente. Dessa forma, poderemos usar seu método *interprete*. Para tal, execute

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

Agora podemos testar nosso interpretador da seguinte maneira: Comece entrando no ambiente ocaml

```
$ rlwrap ocaml
```

Em seguida, utilize o método `interprete "nome_do_arquivo.lua";;` para interpretar o arquivo usado como input.

```
# interprete "micro05.lua";;
```

Pronto! Esse código retornará o resultado do `interprete`.

7.2 Arquivos

`interprete.ml`

Listagem 7.1: `interprete.ml` - Arquivo com o código do interpretador

```
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     | _ -> failwith "obtem_nome_tipo_var: nao eh variavel"
14
15 let pega_int exp =
16     match exp with
17     | T.ExpInt (i, _) -> i
18     | _ -> failwith "pega_int: nao eh inteiro"
19
20 let pega_float exp =
21     match exp with
22     | T.ExpFloat (f, _) -> f
23     | _ -> failwith "pega_float: nao eh float"
24
25
26 let pega_string exp =
27     match exp with
28     | T.ExpString (s, _) -> s
29     | _ -> failwith "pega_string: nao eh string"
30
31
32 let pega_bool exp =
33     match exp with
34     | T.ExpBool (b, _) -> b
35     | _ -> failwith "pega_bool: nao eh booleano"
36
37 type classe_op = Aritmetico | Relacional | Logico | Cadeia
38
39 let classifica op =
40     let open A in
41     match op with
42     | Or
43     | And
44     | Or_Binario
45     | Or_Binario_Exclusivo
46     | And_Binario
47     | Not -> Logico
48     | Menor
49     | Maior
50     | Menor_ou_Igual
51     | Maior_ou_Igual
52     | Equivalente
53     | Nao_Equivalente -> Relacional
54     | Adicao
55     | Subtracao
56     | Multiplicacao
57     | Divisao
58     | Mult_Por_2
59     | Div_Por_2
60     | Divisao_Inteiro
61     | Modulo
62     | Exponenciacao -> Aritmetico

```

```

63         | Concatena -> Cadeia
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         (match (Amb.busca amb id) with
77          | Amb.EntVar (tipo, v) ->
78              (match v with
79               | None -> failwith ("variavel nao inicializada: " ^
80                                   id)
81               | Some valor -> valor)
82          | _ -> failwith ("interpreta_exp: expvar"))
83     | ExpOp ((op, top), (esq, tesq), (dir, tdir)) ->
84         let vesq = interpreta_exp amb esq
85         and vdir = interpreta_exp amb dir in
86
87         let interpreta_aritmetico () =
88             (match top with
89              | TipoInt ->
90                  (match op with
91                   | Adicao -> ExpInt (pega_int vesq +
92                                       pega_int vdir, top)
93                   | Subtracao -> ExpInt (pega_int vesq -
94                                             pega_int vdir, top)
95                   | Multiplicacao -> ExpInt (pega_int vesq *
96                                                pega_int vdir, top)
97                   | Divisao -> ExpInt (pega_int vesq /
98                                         pega_int vdir, top)
99                   | Modulo -> ExpInt (pega_int vesq mod
100                                         pega_int vdir, top)
101                   | _ -> failwith ("
102                                     interpreta_aritmetico: int"))
103              | TipoFloat ->
104                  (match op with
105                   | Adicao -> ExpFloat (pega_float vesq +
106                                           pega_float vdir, top)
107                   | Subtracao -> ExpFloat (pega_float vesq -
108                                              pega_float vdir, top)
109                   | Multiplicacao -> ExpFloat (pega_float vesq *
110                                                  pega_float vdir, top)
111                   | Divisao -> ExpFloat (pega_float vesq /
112                                           pega_float vdir, top)
113                   | _ -> failwith ("
114                                     interpreta_aritmetico: float"))
115              | _ -> failwith ("interpreta_aritmetico: type
116                                problem"))
117         and interpreta_relacional () =
118             (match tesq with
119              | TipoInt ->
120                  (match op with
121                   | Menor -> ExpBool (pega_int vesq <

```

```

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

```

```

141         | _ -> failwith ("
142             interpreta_relacional: bool"))
143     | _ -> failwith ("interpreta_relacional: type
144         error"))
145 and interpreta_logico () =
146     (match tesq with
147     | TipoBool ->
148         (match op with
149         | Or -> ExpBool (pega_bool vesq || pega_bool
150             vdir, top)
151         | And -> ExpBool (pega_bool vesq && pega_bool
152             vdir, top)
153         | _ -> failwith ("interpreta_logico: boold"
154             ))
155     | _ -> failwith ("interpreta_logico: type problem
156         "))
157
158 and interpreta_cadeia () =
159     (match tesq with
160     | TipoString ->
161         (match op with
162         | Concatena -> ExpString (pega_string vesq ^
163             pega_string vdir, top)
164         | _ -> failwith ("interpreta_cadeia:
165             string"))
166     | _ -> failwith("interpreta_cadeia: type problem"
167         ))
168
169 in
170 let valor = (match (classifica op) with
171     Aritmetico -> interpreta_aritmetico ()
172     | Relacional -> interpreta_relacional ()
173     | Logico -> interpreta_logico ()
174     | Cadeia -> interpreta_cadeia ())
175
176 in
177     valor
178 | ExpChamada (id, args, tipo) ->
179     let open Amb in
180         (match (Amb.busca amb id) with
181         | Amb.EntFun {tipo_fn; formais; locais; corpo} ->
182             let vargs = List.map (interpreta_exp amb) args in
183             let vformais = List.map2 (fun (n, t) v -> (n, t,
184                 Some v))
185                 formais vargs
186             in interpreta_fun amb id vformais locais corpo
187         | _ -> failwith ("interpreta_exp: expchamada"))
188 and interpreta_fun amb fn_nome fn_formais fn_locais fn_corpo
189 =
190     let open A in
191         let ambfn = Amb.novo_escopo amb in
192         let insere_local d =
193             match d with
194             (DecVar (v, t)) -> Amb.insere_local ambfn (fst
195                 v) t None
196
197         in
198             let insere_parametro (n, t, v) = Amb.insere_param
199                 ambfn n t v in
200             let _ = List.iter insere_parametro fn_formais in

```

```

187         let _ = List.iter insere_local fn_locais in
188     try
189         let _ = List.iter (interpreta_cmd ambfn) fn_corpo
190             in T.ExpVoid
191     with
192         Valor_de_retorno expret -> expret
193
194 and interpreta_cmd amb cmd =
195     let open A in
196     let open T in
197     match cmd with
198     | CmdRetorno exp ->
199         (match exp with
200         | None -> raise (Valor_de_retorno ExpVoid)
201         | Some e ->
202             let e1 = interpreta_exp amb e in
203             raise (Valor_de_retorno e1))
204     | CmdIf (teste, entao, senao) ->
205         let testel = interpreta_exp amb teste in
206         (match testel with
207         | ExpBool (true, _) ->
208             List.iter (interpreta_cmd amb) entao
209         | _ ->
210             (match senao with
211             | None -> ()
212             | Some bloco -> List.iter (interpreta_cmd amb) bloco))
213     | CmdAtrib (elem, exp) ->
214         let exp = interpreta_exp amb exp
215         and (elem1, tipo) = obtem_nome_tipo_var elem in
216         Amb.atualiza_var amb elem1 tipo (Some exp)
217
218     | CmdChamada exp -> ignore (interpreta_exp amb exp)
219
220     | CmdScanInt exp
221     | CmdScanFloat exp
222     | CmdScanString exp ->
223         let nt = obtem_nome_tipo_var exp in
224         let leia_var (nome, tipo) =
225             let _ =
226                 (try
227                     begin
228                         match (Amb.busca amb nome) with
229                         | Amb.EntVar (_,_) -> ()
230                         | Amb.EntFun _ -> failwith ("falha no
231                             input")
232                     end
233                 with Not_found ->
234                     let _ = Amb.insere_local amb nome tipo None in ())
235             in
236             let valor =
237                 (match tipo with
238                 | TipoInt -> T.ExpInt (read_int (), tipo)
239                 | TipoFloat -> T.ExpFloat (read_float (), tipo)
240                 | TipoString -> T.ExpString (read_line (), tipo)
241                 | _ -> failwith ("Fail input"))
242                 in Amb.atualiza_var amb nome tipo (Some valor)
243             in leia_var nt
244
245     | CmdPrint exp ->

```



```

244         let resp = interpreta_exp amb exp in
245         (match resp with
246         | T.ExpInt (n, _) -> print_int n
247         | T.ExpFloat (n, _) -> print_float n
248         | T.ExpString (n, _) -> print_string n
249         | _ -> failwith ("Fail print"))
250     | CmdWhile (cond, cmds) ->
251         let rec laco cond cmds =
252             let condResp = interpreta_exp amb cond in
253             (match condResp with
254             | ExpBool (true, _) ->
255                 let _ = List.iter (interpreta_cmd amb) cmds in
256                 laco cond cmds
257             | _ -> ())
258         in laco cond cmds
259
260 let insere_declaracao_var amb dec =
261     match dec with
262     | A.DecVar (nome, tipo) -> Amb.insere_local amb (fst nome) tipo None
263
264 let insere_declaracao_fun amb dec =
265     let open A in
266     match dec with
267     | DecFun {fn_nome; fn_tiporet; fn_formais; fn_locais; fn_corpo} ->
268         let nome = fst fn_nome in
269         let formais = List.map (fun (n, t) -> ((fst n), t)) fn_formais in
270         Amb.insere_fun amb nome formais fn_locais fn_tiporet fn_corpo
271
272 let fn_predefs = let open A in [
273     ("entrada", [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
274     ("saida", [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
275 ]
276
277 let declara_predefinidas amb =
278     List.iter (fun (n, ps, tr, c) -> Amb.insere_fun amb n ps [] tr c)
279     fn_predefs
280
281 let interprete ast =
282     let amb_global = Amb.novo_amb [] in
283     let _ = declara_predefinidas amb_global in
284     let (A.Programa (decs_globais, decs_funs, corpo)) = ast in
285     let _ = List.iter (insere_declaracao_var amb_global) decs_globais in
286     let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
287     let resultado = List.iter (interpreta_cmd amb_global) corpo in
288     resultado

```

interprete.mli

Listagem 7.2: interprete.mli - Arquivo com o código do interpretador

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

interpreteTeste.ml

Listagem 7.3: interpreteTeste.ml - Arquivo com o código para testar o adaptador interpretador

```
1 open Printf
```

```

2 open Lexing
3
4 open Ast
5 exception Erro_Sintatico of string
6
7 module S = MenhirLib.General (* Streams *)
8 module I = Sintatico.MenhirInterpreter
9
10 open Semantico
11
12 let message =
13   fun s ->
14     match s with
15     | 0 ->
16       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
17     | 1 ->
18       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
19     | 34 ->
20       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
21     | 35 ->
22       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
23     | 36 ->
24       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
25     | 72 ->
26       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
27     | 47 ->
28       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
29     | 48 ->
30       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
31     | 49 ->
32       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
33     | 51 ->
34       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
35     | 52 ->
36       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
37     | 55 ->
38       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
39     | 56 ->
40       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
41     | 57 ->
42       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
43     | 58 ->
44       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
45     | 61 ->
46       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
47     | 62 ->
48       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
49     | 63 ->
50       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
51     | 64 ->
52       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
53     | 73 ->
54       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
55     | 74 ->
56       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
57     | 95 ->
58       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
59     | 89 ->
60       "<YOUR SYNTAX ERROR MESSAGE HERE>\n"

```

```

61 | 97 ->
62 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
63 | 98 ->
64 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
65 | 99 ->
66 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
67 | 65 ->
68 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
69 | 66 ->
70 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
71 | 53 ->
72 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
73 | 67 ->
74 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
75 | 68 ->
76 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
77 | 59 ->
78 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
79 | 60 ->
80 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
81 | 42 ->
82 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
83 | 41 ->
84 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
85 | 70 ->
86 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
87 | 75 ->
88 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
89 | 77 ->
90 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
91 | 76 ->
92 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
93 | 105 ->
94 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
95 | 84 ->
96 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
97 | 43 ->
98 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
99 | 85 ->
100 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
101 | 86 ->
102 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
103 | 45 ->
104 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
105 | 46 ->
106 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
107 | 102 ->
108 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
109 | 103 ->
110 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
111 | 81 ->
112 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
113 | 3 ->
114 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
115 | 2 ->
116 |     "<YOUR SYNTAX ERROR MESSAGE HERE>\n"
117 | 6 ->
118 |     "estado 6: esperava um tipo. Exemplo:\n    x : inteiro;\n"
119 | 7 ->

```

```

120         "estado 7: esperava a definicao de um campo. Exemplo:\n    i:
           registro\n           parte_real: inteiro;\n           parte_imag:
           inteiro;\n           fim registro;\n           "
121 | 8 ->
122     "estado 8: esperava ':'. Exemplo:\n    x: inteiro;\n    "
123 | 9 ->
124     "estado 9: esperava um tipo. Exemplo:\n    x: inteiro;\n"
125 | 25 ->
126     "estado 25: esperva um ';'.'. \n"
127 | 26 ->
128     "estado 26: uma declaracao foi encontrada. Para continuar era\n
           esperado uma outra declara\195\167\195\163o ou a palavra '
           inicio'. \n"
129 | 29 ->
130     "estado 29: espera a palavra 'registro'. Exemplo:\n    i: registro\n
           n           parte_real: inteiro;\n           parte_imag: inteiro;\n
           fim registro;\n"
131 | 31 ->
132     "estado 31: esperava um ';'.'. \n"
133 | 107 ->
134     "estado 107: uma declaracao foi encontrada. Para continuar era\n
           esperado uma outra declara\195\167\195\163o ou a palavra '
           inicio'. \n"
135 | 13 ->
136     "estado 13: esperava um '['. Exemplo:\n    arranjo [1..10] de
           inteiro;\n"
137 | 14 ->
138     "estado 14: esperava os limites do vetor. Exemplo:\n    arranjo
           [1..10] de inteiro;\n"
139 | 15 ->
140     "estado 15: esperava '...'. Exemplo:\n    1 .. 10\n"
141 | 16 ->
142     "estado 16: esperava um numero inteiro. Exemplo:\n    1 .. 10\n"
143 | 18 ->
144     "estado 18: esperava um ']'. Exemplo\n    arranjo [1..10] de
           inteiro;\n"
145 | 19 ->
146     "estado 19: esperava a palavra reservada 'de'. Exemplo:\n
           arranjo [1..10] de inteiro;\n"
147 | 20 ->
148     "estado 20: esperava um tipo. Exemplo\n    arranjo [1..10] de
           inteiro;\n"
149 | _ ->
150     raise Not_found
151
152 let posicao lexbuf =
153     let pos = lexbuf.lex_curr_p in
154     let lin = pos.pos_lnum
155     and col = pos.pos_cnum - pos.pos_bol - 1 in
156     sprintf "linha %d, coluna %d" lin col
157
158 (* [pilha checkpoint] extrai a pilha do autômato LR(1) contida em
    checkpoint *)
159
160 let pilha checkpoint =
161     match checkpoint with
162     | I.HandlingError amb -> I.stack amb
163     | _ -> assert false (* Isso não pode acontecer *)
164

```

```

165 let estado checkpoint : int =
166   match Lazy.force (pilha checkpoint) with
167   | S.Nil -> (* O parser está no estado inicial *)
168       0
169   | S.Cons (I.Element (s, _, _, _), _) ->
170       I.number s
171
172 let sucesso v = Some v
173
174 let falha lexbuf (checkpoint : (Sast.expressao Ast.programa) I.checkpoint)
175   =
176   let estado_atual = estado checkpoint in
177   let msg = message estado_atual in
178   raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
179                                     (Lexing.lexeme_start lexbuf) msg))
179
180 let loop lexbuf resultado =
181   let fornecedor = I.lexer_lexbuf_to_supplier Lexico.token lexbuf in
182   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
183
184
185 let parse_com_erro lexbuf =
186   try
187     Some (loop lexbuf (Sintatico.Incremental.programa lexbuf.lex_curr_p))
188   with
189   | Lexico.Erro msg ->
190     printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
191     None
192   | Erro_Sintatico msg ->
193     printf "Erro sintático na %s %s\n" (posicao lexbuf) msg;
194     None
195
196 let parse s =
197   let lexbuf = Lexing.from_string s in
198   let ast = parse_com_erro lexbuf in
199   ast
200
201 let parse_arq nome =
202   let ic = open_in nome in
203   let lexbuf = Lexing.from_channel ic in
204   let ast = parse_com_erro lexbuf in
205   let _ = close_in ic in
206   ast
207
208 let verifica_tipos nome =
209   let ast = parse_arq nome in
210   match ast with
211   | Some (Some ast) -> semantico ast
212   | _ -> failwith "Nada a fazer!\n"
213
214
215 let interprete nome =
216   let tast, amb = verifica_tipos nome in
217   Interprete.interprete tast

```

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 = { ambv: entrada Tab.tabela}
16
17 let novo_amb xs = { ambv = Tab.cria xs}
18
19 let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }
20
21 let busca amb ch = Tab.busca amb.ambv ch
22
23 let atualiza_var amb ch t v = Tab.atualiza amb.ambv ch (EntVar (t, v))
24
25 let insere_local amb nome t v = Tab.insere amb.ambv nome (EntVar (t, v))
26
27 let insere_param amb nome t v = Tab.insere amb.ambv nome (EntVar (t, v))
28
29 let insere_fun amb nome params locais resultado corpo = let ef = EntFun {
30   tipo_fn = resultado;
31   formais
32   =
33   params
34   ;
35   locais
36   =
37   locais
38   ;
39   corpo
40   =
41   corpo
42   }
43   in Tab.insere amb.
44     ambv nome ef

```

ambInterp.mli

```
1 type entrada_fn = {
2   tipo_fn: Ast.tipo;
3   formais: (string * Ast.tipo) list;
4   locais: Ast.declaracoes;
5   corpo: Tast.expressao Ast.comandos
6 }
7
8 type entrada =
9   | EntFun of entrada_fn
10  | EntVar of Ast.tipo * (Tast.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 -> (Tast.expressao option) ->
    unit
18 val insere_local: t -> string -> Ast.tipo -> (Tast.expressao option) ->
    unit
19 val insere_param: t -> string -> Ast.tipo -> (Tast.expressao option) ->
    unit
20 val insere_fun: t -> string -> (string * Ast.tipo) list -> Ast.declaracoes
    -> Ast.tipo -> (Tast.expressao Ast.comandos) -> unit
```
