

Construção de um compilador de MiniPascal para Dalvik usando Objective Caml

Guilherme de Souza Silva

`guilherme.souza.silva95@gmail.com`

Faculdade de Computação
Universidade Federal de Uberlândia

1 de agosto de 2017

Lista de Listagens

2.1	Módulo Mínimo Pascal	12
2.2	Módulo Mínimo Java	12
2.3	Módulo Mínimo Smali	12
2.4	declaração de variável em Pascal	13
2.5	declaração de variável em Java	13
2.6	declaração de variável em Smali	13
2.7	Atribuição de valores em Pascal	14
2.8	Atribuição de valores em Java	14
2.9	Atribuição de valores em Smali	14
2.10	Soma em Pascal	15
2.11	Soma em Java	15
2.12	Soma em Smali	15
2.13	Print em Pascal	16
2.14	Print em Java	16
2.15	Print em Smali	16
2.16	Subtração em Pascal	17
2.17	Subtração em Java	17
2.18	Subtração em Smali	17
2.19	Condicional if em Pascal	18
2.20	Condicional if em Java	19
2.21	Condicional if em Smali	19
2.22	Condicional else em Pascal	20
2.23	Condicional else em Java	20
2.24	Condicional else em Smali	20
2.25	Soma e divisão em Pascal	21
2.26	Soma e divisão em Java	21
2.27	Soma e divisão em Smali	22
2.28	Duas variáveis em Pascal	23
2.29	Duas variáveis em Java	23
2.30	Duas variáveis em Smali	23
2.31	Laço while em Pascal	24
2.32	Laço while em Java	25
2.33	Laço while em Smali	25
2.34	Repetição e condicionais em Pascal	26
2.35	Repetição e condicionais em Java	26
2.36	Repetição e condicionais em Smali	26
2.37	Celsius em Fahrenheit Pascal	27
2.38	Celsius em Fahrenheit Java	28
2.39	Celsius em Fahrenheit Smali	28
2.40	Verificar maior número Pascal	30

2.41	Verificar maior número Java	30
2.42	Verificar maior número Smali	30
2.43	Verifica se está entre 100 e 200 Pascal	33
2.44	Verifica se está entre 100 e 200 Java	34
2.45	Verifica se está entre 100 e 200 Smali	34
2.46	Verifica vários valores Pascal	36
2.47	Verifica vários valores Java	36
2.48	Verifica vários valores Smali	37
2.49	Strings e Chars Pascal	38
2.50	Strings e Chars Java	39
2.51	Strings e Chars Smali	40
2.52	Switch em Pascal	43
2.53	Switch em Java	44
2.54	Switch em Smali	45
2.55	Verifica positivo negativo e zero Pascal	47
2.56	Verifica positivo negativo e zero Java	47
2.57	Verifica positivo negativo e zero Smali	48
2.58	maior ou menor que 10 Pascal	50
2.59	maior ou menor que 10 Java	50
2.60	maior ou menor que 10 Smali	51
2.61	Calcula preços Pascal	53
2.62	Calcula preços Java	54
2.63	Calcula preços Smali	54
2.64	Fatorial em Pascal	57
2.65	Fatorial em Java	57
2.66	Fatorial em Smali	58
2.67	Celsius em Fahrenheit Pascal	60
2.68	Celsius em Fahrenheit Java	60
2.69	Celsius em Fahrenheit Smali	61
3.1	Analizador Léxico	72
3.2	Carregador	76
3.3	Fatorial em Pascal	76
3.4	Erro de Caractere	77
3.5	Parser LR(1)	78
3.6	Analizador Léxico	81
3.7	Gerador Árvore Sintática	84
3.8	Erros	85
3.9	Teste	89
3.10	Ocamlinit	92
3.11	Analizador Semantico	93
3.12	Ambiente	100
3.13	ast	100
3.14	Sast	102
3.15	Tast	102
3.16	Semântico Teste	102
3.17	Interpretador	105
3.18	Ambiente Interpretador	112
3.19	Interpretador Teste	113
3.20	.ocamlinit	114

Sumário

1	Introdução	6
1.1	Sistema Operacional	6
1.2	Instalação Java JDK	6
1.3	Dalvik	6
1.4	Instalação da Dalvik	7
1.5	Compilando Java em dex	7
1.6	DEX e SMALI	7
1.7	SMALI e DEX	8
1.8	Executando o arquivo .dex no Android	8
1.8.1	Utilizando um Smartphone	8
1.8.2	Utilizando Emulador AVD	8
1.9	A linguagem Pascal	10
1.10	Compilando e executando um código Pascal	10
2	Programas de Teste	12
2.1	Nano Programas	12
2.2	Micro Programas	27
2.3	Mais informações sobre SMALI	63
2.3.1	Registradores	63
2.3.2	Tipos	64
2.3.3	Métodos	65
2.4	Instruções Importantes	66
3	Construção de um Compilador	72
3.1	Analizador Léxico	72
3.2	Análise Sintática	78
3.2.1	Analizador	78
3.2.2	Códigos	78
3.2.3	Árvore Sintática Abstrata	84
3.2.4	Tratamento de erros	85
3.2.5	Testes	89
3.3	Tópicos Complementares	91
3.3.1	Extras Analizador Léxico e Ocaml	91
3.3.2	Extras Analizador Sintático e Ocaml	92
3.4	Análise Semântica	92
3.4.1	Códigos	93
3.4.2	Testes	102
3.4.3	Extras	105
3.5	Interpretador	105

3.5.1	Testes	112
3.5.2	Extras	114
3.6	Conclusão	115
4	Referências	116

Capítulo 1

Introdução

Este documento foi escrito como relatório de diversas tarefas propostas pela matéria de construção de compiladores, o intuito dessas tarefas é aumentar minha familiaridade com a linguagem MiniPascal, melhorar o entendimento do que é e como é um código gerado para a máquina virtual Dalvik e também de como utilizar essa máquina virtual.

1.1 Sistema Operacional

Para esse trabalho foi utilizado o sistema operacional Linux Ubuntu 16.04, sua instalação é fácil e rápida, basta acessar o [site](#) e seguir os passos descritos pela desenvolvedora do sistema.

1.2 Instalação Java JDK

Para realizar instalação da JDK, basta que os seguintes comandos sejam executados em um terminal linux:

```
> sudo add-apt-repository ppa:webupd8team/java  
> sudo apt-get update  
> sudo apt-get install oracle-java9-installer
```

1.3 Dalvik

Construída por Dan Bornstein e outros engenheiros da google é uma máquina virtual baseada no uso de registradores reais ao invés de virtuais, foi projetada para ser utilizada no sistema operacional Android, é muito conhecida pelo seu desempenho bom e pelo baixo consumo de memória, seu projeto também permite que várias instâncias da máquina rodem ao mesmo tempo. Essa máquina pode ser confundida com a JVM (Java Virtual Machine) porém o bytecode utilizado por ela é bastante diferente.

1.4 Instalação da Dalvik

A Dalvik já vem instalada em todos os aparelhos/emuladores que rodam android, logo a instalação dela é bastante simples. Inicialmente devemos acessar a página do android studio e baixar a versão mais recente do android studio para linux, isso pode ser feito por esse [link](#). Para rápida instalação basta que os passos desse [link](#) sejam seguidos (selecione a opção linux). Observação importante, a google recomenda a execução do comando abaixo no caso de seu sistema operacional ser de 64 bits.

```
> sudo apt-get install lib32z1 lib32ncurses5 lib32bz2-1.0 lib32stdc++6
```

1.5 Compilando Java em dex

Um código Java não é compilado diretamente para .dex, inicialmente precisaremos gerar .class do java utilizando o compilador java comum e depois poderemos então realizar a compilação para o formato suportado pela Dalvik, uma observação importante a ser feita é que caso o .class seja gerado utilizando java8 a compilação para .dex não irá funcionar. Para que seja feita a compilação, basta que os dois comandos abaixo sejam executados.

```
> javac source 1.7 target 1.7 Exemplo.java
> android-studio-path/Sdk/build-tools/version/dx --dex --output=exemplo.dex Exemplo.class
```

Onde android-studio-path deve ser substituído pelo caminho onde a sdk do android foi instalado (em meu caso o caminho foi /home/compiladores/Android/Sdk/build-tools/25.0.2/dx). Observe que o nome do arquivo .dex não precisa necessariamente ter o mesmo nome que o .class

1.6 DEX e SMALI

A máquina virtual Dalvik é capaz de executar códigos compilados em .dex (bytecodes), por tanto, é nestes arquivos que se encontra o código mais importante para um programa. Apesar disso ao abrir em um editor texto, fica impossível compreender o que está escrito nestes arquivos, isso ocorre pois estes já são arquivos montados em bytecodes, para solucionar esse problema e transformar o código em um assembly legível, utilizei uma ferramenta chamada [baksmali](#) essa ferramenta tem módulos para fazer o desassembly e o assembly de um arquivo .dex, em nosso caso utilizaremos o módulo para fazer o desassembly, para isso baixar que realize-se o download da ferramenta [aqui](#), e que código a seguir seja executado no terminal, na pasta onde se encontra o arquivo .dex, e que o arquivo baksmali-2.1.3.jar e o .dex que será convertido estejam juntos na mesma pasta.

```
> java -jar baksmali-2.1.3.jar micro01.dex
```

Uma observação importante a ser feita é que a versão utilizada da ferramenta baksmali nesse trabalho foi a 2.1.3, fiz essa escolha pois ao utilizar a versão 2.2.0 tive erros na execução da

ferramenta. Seguindo essas passos, conseguiremos converter nossos arquivos .dex em .smali que podem ser compreendidos por nós.

1.7 SMALI e DEX

Simples passos podem ser seguidos para se gerar um .dex a partir de um arquivo .smali, faça o download do programa smali2.1.3.jar através desse [link](#). Coloque o arquivo na mesma pasta em que se encontra o arquivo .smali e utilize o seguinte comando:

```
> java -jar smali-2.1.3.jar arquivo.smali
```

1.8 Executando o arquivo .dex no Android

Após a compilação para que executemos o .dex necessitaremos de um aparelho android ou de um emulador que rode o sistema operacional android, para esse momento, estou utilizando meu próprio aparelho rodando Android 6.0.

1.8.1 Utilizando um Smartphone

Executando os comandos abaixo em um terminal linux o programa antes feito em java será executado pelo smartphone e responderá para o terminal através do ADB (Android Debug bridge). O sistema operacional utilizado no smartphone é Android 6.0 o aparelho utilizado para este trabalho é conhecido como Motorola Xt1097 (moto X2)

```
> ./adb devices
> ./adb push /home/compiladores/Desktop/CC-Pascal/nanos\ e\ micros/micro1/
  Fa.dex /sdcard/
> ./adb shell dalvikvm -cp /sdcard/Fa.dex Fa
```

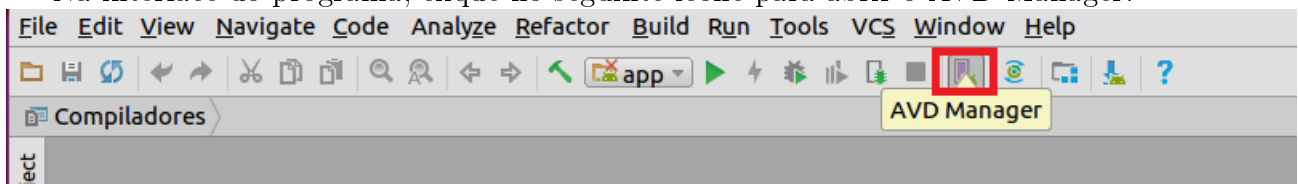
A execução da ferramenta adb deve ser feita dentro da pasta androidpath/sdk/platform-tools/, por tanto é onde todos os comandos descritos acima devem ser executados

1.8.2 Utilizando Emulador AVD

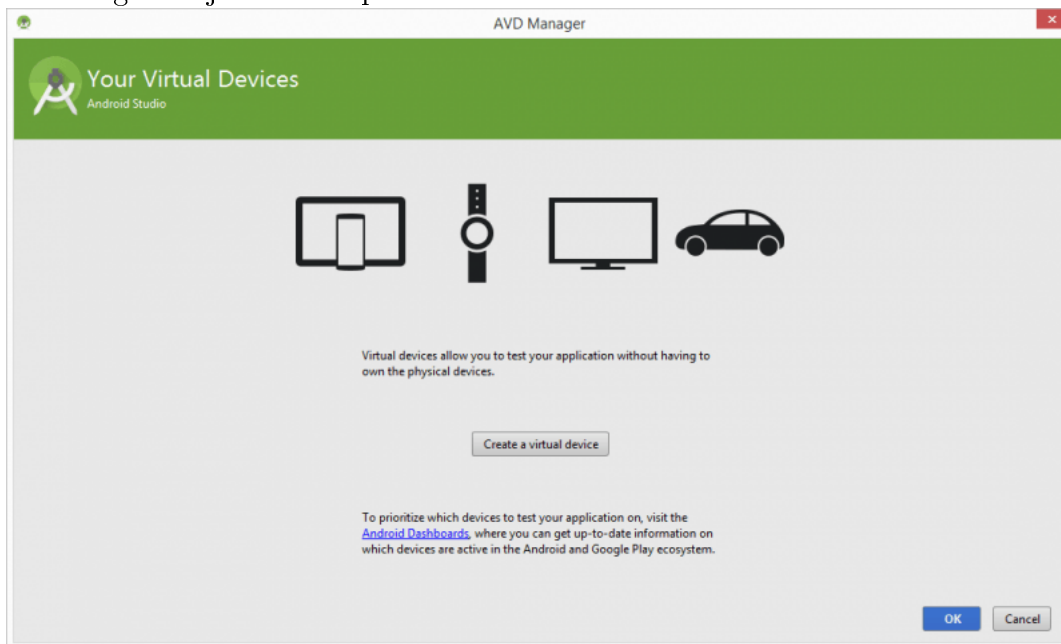
Após a instalação do Android Studio descrito acima, execute-o na pasta onde ele foi instalado (provavelmente /usr/locals/android-studio) utilizando o comando abaixo e crie um novo projeto em branco.

```
> ./studio.sh
```


Na interface do programa, clique no seguinte ícone para abrir o AVD Manager:

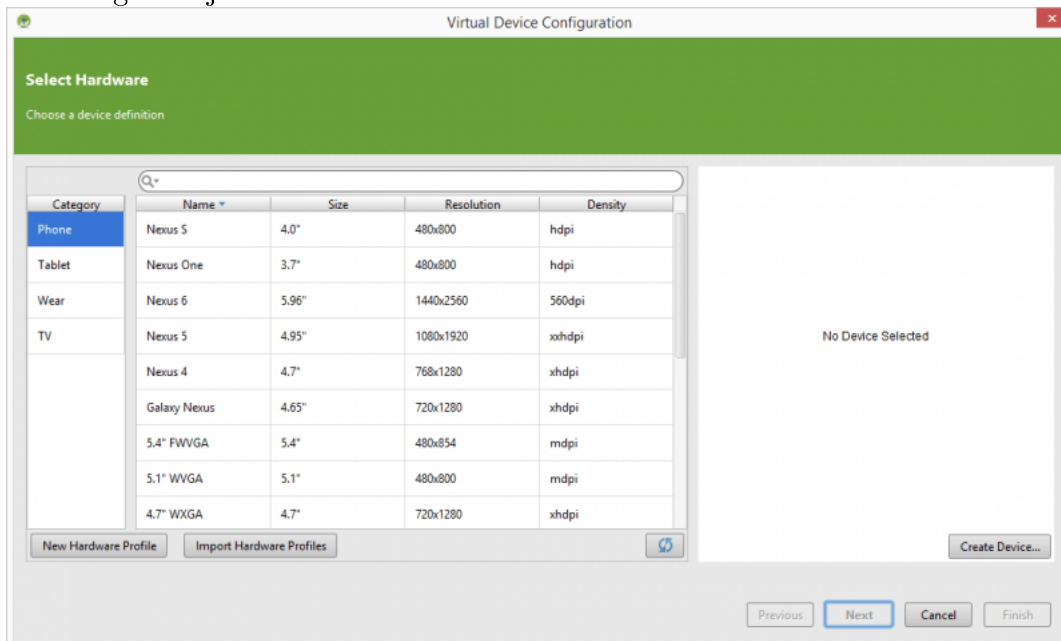


A seguinte janela irá aparecer:



Nessa janela, clique em create a virtual device.

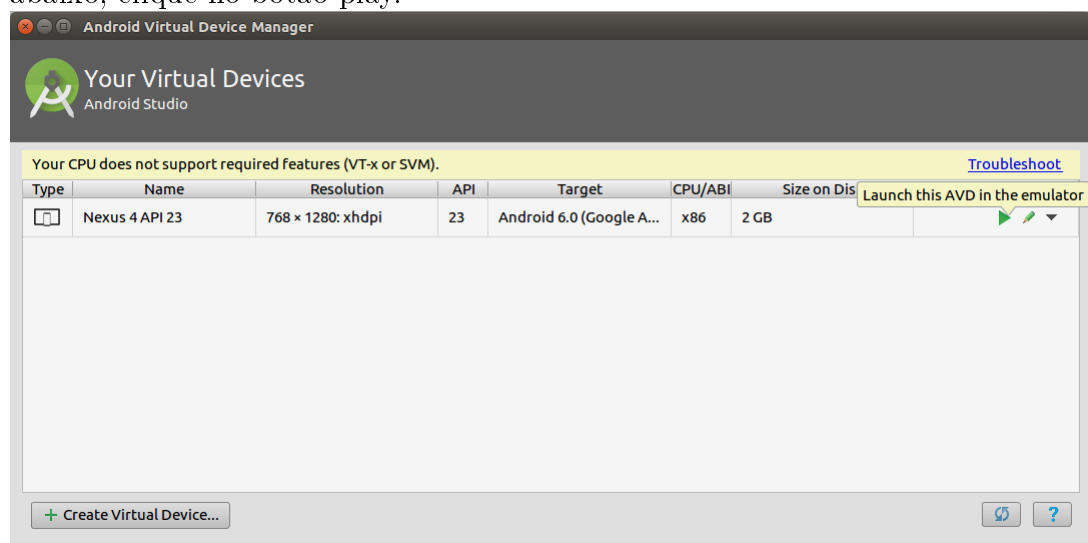
A seguinte janela será exibida:



Selecione o dispositivo que deseja emular (recomendo os da linha nexus!) continue para as próximas janelas clicando em next. E repita isso até que o botão Finish apareça. Isso criará seu emulador.

Abra o Avd manager novamente, sua máquina criada deverá estar listada como na figura

abaixo, clique no botão play.



Com isso você já tem um emulador rodando em sua máquina, agora para rodar o seu código .dex nela, basta realizar os seguintes comandos:

```
> ./adb devices
> ./adb push /home/compiladores/Desktop/CC-Pascal/nanos\ e\ micros/micro1/
  Fa.zip /data/local
> ./adb shell dalvikvm -cp /data/local/Fa.zip Fa
```

A execução da ferramenta adb deve ser feita dentro da pasta androidpath/sdk/platform-tools/, por tanto é onde todos os comandos descritos acima devem ser executados

Lembrete: Antes de fazer esses comandos, o arquivo .dex deve ser zipado em um arquivo .zip, utilizando comando em terminal descrito abaixo na pasta onde se encontra o .dex:

```
> zip -r Fa.zip Fa.dex
```

1.9 A linguagem Pascal

É uma linguagem de programação estruturada desenvolvida por Niklaus Wirth que era um encorajador do uso de códigos estruturados.

Devida a grande variedade de dialetos diferentes para Pascal, esta ficou conhecida como uma família de linguagens de programação, seu sucesso se deu na década de 1980, sendo hoje utilizada mais para ensino de programação básica. Apesar de menos utilizada hoje, Pascal continua forte em produtos da Embarcadero como o Delphi que utiliza Object Pascal

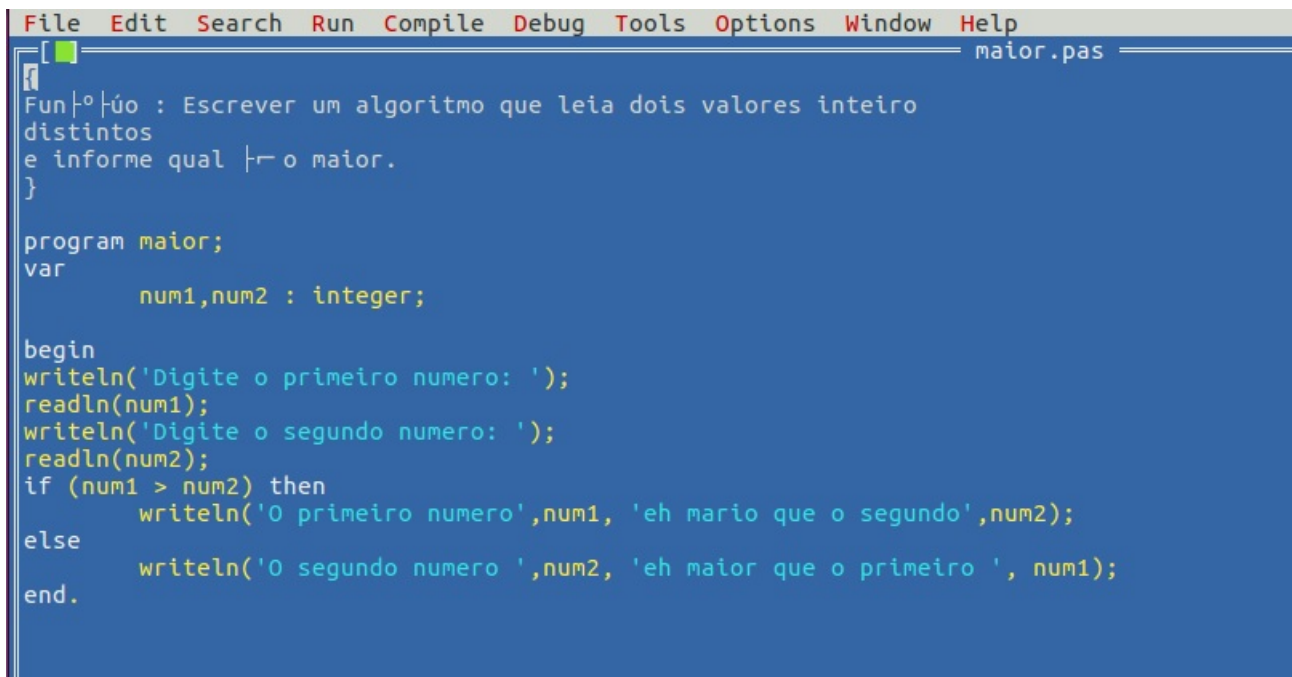
1.10 Compilando e executando um código Pascal

Para realizar a execução e compilação de um código em pascal utilizei o compilador Free Pascal que pode ser baixado utilizando o comando:

```
> sudo apt-get install fpc
```

Após a instalação basta que digite o código abaixo, com isso uma interface irá se abrir em linha de comando com o arquivo aberto, para compilar basta aperta-se `alt+f9` e para executar `ctrl+f9`

```
> fp nomedoarquivo.pas
```



The screenshot shows the FPC IDE interface with a menu bar (File, Edit, Search, Run, Compile, Debug, Tools, Options, Window, Help) and a toolbar. The main window displays the source code for 'maior.pas'. The code is as follows:

```

Função : Escrever um algoritmo que leia dois valores inteiro
distintos
e informe qual o maior.
}

program maior;
var
    num1,num2 : integer;

begin
    writeln('Digite o primeiro numero: ');
    readln(num1);
    writeln('Digite o segundo numero: ');
    readln(num2);
    if (num1 > num2) then
        writeln('O primeiro numero',num1, 'eh maior que o segundo',num2);
    else
        writeln('O segundo numero ',num2, 'eh maior que o primeiro ', num1);
    end.
  
```

Outra forma de se utilizar a compilação de um arquivo .pas é digita apenas o código descrito abaixo no terminal linux, se tudo ocorrer bem você terá seu executável.

```
> fpc nomedoarquivo.pas
```

Capítulo 2

Programas de Teste

A seguir, os programas em Pascal e Java, seguidos pelos bytecodes da Dalvik traduzidos desassemblados para .smali:

2.1 Nano Programas

Listagem 2.1: Módulo Mínimo Pascal

```
1 program nano01;  
2  
3 begin  
4 end.
```

Listagem 2.2: Módulo Mínimo Java

```
1  
2 public class nano01{  
3  
4     public static void main(String[] args){  
5  
6  
7  
8     }  
9 }
```

Listagem 2.3: Módulo Mínimo Smali

```
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 2
```

2.1

```
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 8
22     return-void
23 .end method
```

Listagem 2.4: declaração de variável em Pascal

```
1 program atrib;
2 var
3     n : integer;
4
5 begin
6
7 end.
```

Listagem 2.5: declaração de variável em Java

```
1
2 public class atrib{
3     public int n;
4
5     public static void main(String[] args){
6
7     }
8 }
```

Listagem 2.6: declaração de variável em Smali

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

```

23
24     .prologue
25     .line 7
26     return-void
27 .end method

```

Listagem 2.7: Atribuição de valores em Pascal

```

1 program atribval;
2 var
3     x : integer;
4 begin
5 x := 1;
6 end.

```

Listagem 2.8: Atribuição de valores em Java

```

1
2 public class atribval{
3
4     public static int x;
5     public static void main(String[] args){
6         x = 1;
7     }
8 }

```

Listagem 2.9: Atribuição de valores em Smali

```

1 .class public Latribval;
2 .super Ljava/lang/Object;
3 .source "atribval.java"
4
5
6 # static fields
7 .field public static x:I
8
9
10 # direct methods
11 .method public constructor <init>()V
12     .registers 1
13
14     .prologue
15     .line 2
16     invoke-direct {p0}, Ljava/lang/Object;-><init>()V
17
18     return-void
19 .end method
20
21 .method public static main([Ljava/lang/String;)V
22     .registers 2
23
24     .prologue
25     .line 6
26     const/4 v0, 0x1
27
28     sput v0, Latribval;->x:I
29
30     .line 7

```

2.1

```
31     return-void
32 .end method
```

Listagem 2.10: Soma em Pascal

```
1 program somaval;
2 var
3     x : integer;
4 begin
5 x := 1+2;
6 end.
```

Listagem 2.11: Soma em Java

```
1
2 public class somaval{
3
4     public static int x;
5     public static void main(String[] args){
6         x = 1+2;
7     }
8 }
```

Listagem 2.12: Soma em Smali

```
1 .class public Lsomaval;
2 .super Ljava/lang/Object;
3 .source "somaval.java"
4
5
6 # static fields
7 .field public static x:I
8
9
10 # direct methods
11 .method public constructor <init>()V
12     .registers 1
13
14     .prologue
15     .line 2
16     invoke-direct {p0}, Ljava/lang/Object;-><init>()V
17
18     return-void
19 .end method
20
21 .method public static main([Ljava/lang/String;)V
22     .registers 2
23
24     .prologue
25     .line 6
26     const/4 v0, 0x3
27
28     sput v0, Lsomaval;~x:I
29
30     .line 7
31     return-void
32 .end method
```

Listagem 2.13: Print em Pascal

```

1 program printa;
2 var
3     x : integer;
4 begin
5 x := 1+2;
6 writeln(x);
7 end.

```

Listagem 2.14: Print em Java

```

1
2 public class printa{
3
4     public static int x;
5     public static void main(String[] args){
6         x = 2;
7         System.out.print(""+x);
8     }
9 }

```

Listagem 2.15: Print em Smali

```

1 .class public Lprinta;
2 .super Ljava/lang/Object;
3 .source "printa.java"
4
5
6 # static fields
7 .field public static x:I
8
9
10 # direct methods
11 .method public constructor <init>()V
12     .registers 1
13
14     .prologue
15     .line 2
16     invoke-direct {p0}, Ljava/lang/Object; -><init>()V
17
18     return-void
19 .end method
20
21 .method public static main([Ljava/lang/String;)V
22     .registers 4
23
24     .prologue
25     .line 6
26     const/4 v0, 0x2
27
28     sput v0, Lprinta; ->x:I
29
30     .line 7
31     sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
32
33     new-instance v1, Ljava/lang/StringBuilder;
34
35     invoke-direct {v1}, Ljava/lang/StringBuilder; -><init>()V

```


2.1

```
36
37     const-string v2, ""
38
39     invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
40
41     move-result-object v1
42
43     sget v2, Lprinta;->x:I
44
45     invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
46
47     move-result-object v1
48
49     invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
        String;
50
51     move-result-object v1
52
53     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->print(Ljava/lang/
        String;)V
54
55     .line 8
56     return-void
57 .end method
```

Listagem 2.16: Subtração em Pascal

```
1 program subval;
2 var
3     x : integer;
4 begin
5 x := 1-2;
6 writeln(x);
7 end.
```

Listagem 2.17: Subtração em Java

```
1
2 public class subval{
3
4     public static int x;
5     public static void main(String[] args){
6         x = 1-2;
7         System.out.print(""+x);
8     }
9 }
```

Listagem 2.18: Subtração em Smali

```
1 .class public Lsubval;
2 .super Ljava/lang/Object;
3 .source "subval.java"
4
5
6 # static fields
7 .field public static x:I
```

```

8
9
10 # direct methods
11 .method public constructor <init>()V
12     .registers 1
13
14     .prologue
15     .line 2
16     invoke-direct {p0}, Ljava/lang/Object;--><init>()V
17
18     return-void
19 .end method
20
21 .method public static main([Ljava/lang/String;)V
22     .registers 4
23
24     .prologue
25     .line 6
26     const/4 v0, -0x1
27
28     sput v0, Lsubval;-->x:I
29
30     .line 7
31     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
32
33     new-instance v1, Ljava/lang/StringBuilder;
34
35     invoke-direct {v1}, Ljava/lang/StringBuilder;--><init>()V
36
37     const-string v2, ""
38
39     invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;-->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
40
41     move-result-object v1
42
43     sget v2, Lsubval;-->x:I
44
45     invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;-->append(I)Ljava/
        lang/StringBuilder;
46
47     move-result-object v1
48
49     invoke-virtual {v1}, Ljava/lang/StringBuilder;-->toString()Ljava/lang/
        String;
50
51     move-result-object v1
52
53     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->print(Ljava/lang/
        String;)V
54
55     .line 8
56     return-void
57 .end method

```

Listagem 2.19: Condicional if em Pascal

```

1 program atribval;
2 var

```

2.1

```
3         x : integer;
4 begin
5 x := 2;
6 if x = 1 then
7     writeln(x);
8 end.
```

Listagem 2.20: Condicional if em Java

```
1
2 public class ifs{
3
4     public static int x;
5     public static void main(String[] args){
6         x = 2;
7         if(x == 1)
8             System.out.print(x);
9     }
10 }
```

Listagem 2.21: Condicional if em Smali

```
1 .class public Lifs;
2 .super Ljava/lang/Object;
3 .source "ifs.java"
4
5
6 # static fields
7 .field public static x:I
8
9
10 # direct methods
11 .method public constructor <init>()V
12     .registers 1
13
14     .prologue
15     .line 2
16     invoke-direct {p0}, Ljava/lang/Object;-><init>()V
17
18     return-void
19 .end method
20
21 .method public static main([Ljava/lang/String;)V
22     .registers 3
23
24     .prologue
25     .line 6
26     const/4 v0, 0x2
27
28     sput v0, Lifs;->x:I
29
30     .line 7
31     sget v0, Lifs;->x:I
32
33     const/4 v1, 0x1
34
35     if-ne v0, v1, :cond_f
36
37     .line 8
```

```

38     sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
39
40     sget v1, Lifs; ->x:I
41
42     invoke-virtual {v0, v1}, Ljava/io/PrintStream; ->print(I)V
43
44     .line 9
45     :cond_f
46     return-void
47 .end method

```

Listagem 2.22: Condicional else em Pascal

```

1 program else;
2 var
3     x : integer;
4 begin
5     x := 2;
6     if x = 1 then
7         writeln(x)
8     else
9         writeln(0);
10 end.

```

Listagem 2.23: Condicional else em Java

```

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

```

Listagem 2.24: Condicional else em Smali

```

1 .class public Lelses;
2 .super Ljava/lang/Object;
3 .source "elses.java"
4
5
6 # static fields
7 .field public static x:I
8
9
10 # direct methods
11 .method public constructor <init>()V
12     .registers 1
13
14     .prologue
15     .line 2
16     invoke-direct {p0}, Ljava/lang/Object; -><init>()V
17

```

2.1

```
18     return-void
19 .end method
20
21 .method public static main([Ljava/lang/String;)V
22     .registers 3
23
24     .prologue
25     const/4 v1, 0x1
26
27     .line 6
28     sput v1, Lelses;->x:I
29
30     .line 7
31     sget v0, Lelses;->x:I
32
33     if-ne v0, v1, :cond_f
34
35     .line 8
36     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
37
38     sget v1, Lelses;->x:I
39
40     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->print(I)V
41
42     .line 11
43     :goto_e
44     return-void
45
46     .line 10
47     :cond_f
48     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
49
50     const/4 v1, 0x0
51
52     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->print(I)V
53
54     goto :goto_e
55 .end method
```

Listagem 2.25: Soma e divisão em Pascal

```
1 program somadiv;
2 var
3     x : double;
4 begin
5     x := (1+1)/ 2;
6     if x = 1 then
7         writeln(x)
8     else
9         writeln(0);
10 end.
```

Listagem 2.26: Soma e divisão em Java

```
1
2 public class somadiv{
3
4     public static int x;
5     public static void main(String[] args){
```

```

6         x = 1 + (1/2);
7         if(x == 1)
8             System.out.print(x);
9         else
10            System.out.print(0);
11     }
12 }

```

Listagem 2.27: Soma e divisão em Smali

```

1 .class public Lsomadiv;
2 .super Ljava/lang/Object;
3 .source "somadiv.java"
4
5
6 # static fields
7 .field public static x:I
8
9
10 # direct methods
11 .method public constructor <init>()V
12     .registers 1
13
14     .prologue
15     .line 2
16     invoke-direct {p0}, Ljava/lang/Object; -><init>()V
17
18     return-void
19 .end method
20
21 .method public static main([Ljava/lang/String;)V
22     .registers 3
23
24     .prologue
25     const/4 v1, 0x1
26
27     .line 6
28     sput v1, Lsomadiv; ->x:I
29
30     .line 7
31     sget v0, Lsomadiv; ->x:I
32
33     if-ne v0, v1, :cond_f
34
35     .line 8
36     sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
37
38     sget v1, Lsomadiv; ->x:I
39
40     invoke-virtual {v0, v1}, Ljava/io/PrintStream; ->print(I)V
41
42     .line 11
43     :goto_e
44     return-void
45
46     .line 10
47     :cond_f
48     sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
49

```

2.1

```
50      const/4 v1, 0x0
51
52      invoke-virtual {v0, v1}, Ljava/io/PrintStream;->print(I)V
53
54      goto :goto_e
55 .end method
```

Listagem 2.28: Duas variáveis em Pascal

```
1 program atribduas;
2 var
3     x : double;
4     y : integer;
5 begin
6     x := (1+1)/2;
7     y := 2;
8     if (x = y) then
9         writeln(x)
10    else
11        writeln(0);
12 end.
```

Listagem 2.29: Duas variáveis em Java

```
1
2 public class atribduas{
3
4     public static int x, y;
5     public static void main(String[] args){
6         x = 1;
7         y = 2;
8         if(x == y)
9             System.out.print(x);
10        else
11            System.out.print(0);
12    }
13 }
```

Listagem 2.30: Duas variáveis em Smali

```
1 .class public Latribduas;
2 .super Ljava/lang/Object;
3 .source "atribduas.java"
4
5
6 # static fields
7 .field public static x:I
8
9 .field public static y:I
10
11
12 # direct methods
13 .method public constructor <init>()V
14     .registers 1
15
16     .prologue
17     .line 2
18     invoke-direct {p0}, Ljava/lang/Object;-><init>()V
```

```

19     return-void
20 .end method
21
22
23 .method public static main([Ljava/lang/String;)V
24     .registers 3
25
26     .prologue
27     .line 6
28     const/4 v0, 0x1
29
30     sput v0, Latribduas;->x:I
31
32     .line 7
33     const/4 v0, 0x2
34
35     sput v0, Latribduas;->y:I
36
37     .line 8
38     sget v0, Latribduas;->x:I
39
40     sget v1, Latribduas;->y:I
41
42     if-ne v0, v1, :cond_14
43
44     .line 9
45     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
46
47     sget v1, Latribduas;->x:I
48
49     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->print(I)V
50
51     .line 12
52     :goto_13
53     return-void
54
55     .line 11
56     :cond_14
57     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
58
59     const/4 v1, 0x0
60
61     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->print(I)V
62
63     goto :goto_13
64 .end method

```

Listagem 2.31: Laço while em Pascal

```

1 program whiles;
2 var
3     x,y,z : integer;
4 begin
5     x := 1;
6     y := 2;
7     z := 5;
8     while (z > x) do
9     begin
10        x := x+y;

```



```

11  writeln(x);
12  end;
13  end.

```

Listagem 2.32: Laço while em Java

```

1
2  public class whiles{
3
4      public static void main(String[] args){
5          int z,y,x;
6          x= 1;
7          y = 2;
8          z = 5;
9          while(z > x){
10             x = x+y;
11             System.out.print(x);
12         }
13     }
14 }

```

Listagem 2.33: Laço while em Smali

```

1  .class public Lwhiles;
2  .super Ljava/lang/Object;
3  .source "whiles.java"
4
5
6  # direct methods
7  .method public constructor <init>()V
8      .registers 1
9
10     .prologue
11     .line 2
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 6
22     const/4 v0, 0x1
23
24     .line 7
25     const/4 v1, 0x2
26
27     .line 8
28     const/4 v2, 0x5
29
30     .line 9
31     :goto_3
32     if-le v2, v0, :cond_c
33
34     .line 10
35     add-int/2addr v0, v1
36

```

```

37     .line 11
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 13
45     :cond_c
46     return-void
47 .end method

```

Listagem 2.34: Repetição e condicionais em Pascal

```

1 program ifandwhile;
2 var
3     x,y,z : integer;
4 begin
5     x := 1;
6     y := 2;
7     z := 5;
8     while (z > x) do
9     begin
10        if (x = y) then
11            writeln(x)
12        else
13            writeln(0);
14        z := z-1;
15    end;
16 end.

```

Listagem 2.35: Repetição e condicionais em Java

```

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

```

Listagem 2.36: Repetição e condicionais em Smali

```

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

```

2.2

```
8      .registers 1
9
10     .prologue
11     .line 2
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 6
22     const/4 v1, 0x1
23
24     .line 8
25     const/4 v0, 0x5
26
27     .line 9
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 12
39     add-int/lit8 v0, v0, -0x1
40
41     goto :goto_2
42
43     .line 14
44     :cond_d
45     return-void
46 .end method
```

2.2 Micro Programas

Listagem 2.37: Celsius em Fahrenheit Pascal

```
1 {
2  Função: Ler uma temperatura em graus Celsius e apresentá-la
3  convertida
4  em graus Fahrenheit. A fórmula de conversão é:
5   $F = (9 * C + 160) / 5$ ,
6  sendo F a temperatura em Fahrenheit e C a temperatura em
7  Celsius.
8 }
9 program micro01;
10 var
11     cel, far: real;
12 begin
13     writeln('Tabela de conversao: Celsius -> Fahrenheit');
```

```

14 writeln('Digite a temperatura em Celsius: ');
15 readln(cel);
16 far := (9*cel+160)/5;
17 writeln('A nova temperatura eh:', far, 'F');
18 end.

```

Listagem 2.38: Celsius em Fahrenheit Java

```

1
2 public class Fa{
3
4     public static void main(String[] args){
5         double cel, far;
6         System.out.println("Tabela de conversão: Celsius -> Fahrenheit");
7         System.out.print("Digite a temperatura em Celsius: ");
8         cel = Double.parseDouble(System.console().readLine());
9         far = ((9 * cel) + 160) / 5;
10        System.out.println("A nova temperatura é: "+far+" F");
11
12    }
13 }

```

Listagem 2.39: Celsius em Fahrenheit Smali

```

1 .class public LFa;
2 .super Ljava/lang/Object;
3 .source "Fa.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 2
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 6
22     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
23
24     const-string v1, "Tabela de convers\u00e3o: Celsius -> Fahrenheit"
25
26     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
27
28     .line 7
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

```

2.2

```
34
35 .line 8
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 9
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 10
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 \u00e9: "
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 12
91     return-void
92 .end method

```

Listagem 2.40: Verificar maior número Pascal

```

1 {
2  Função : Escrever um algoritmo que leia dois valores inteiro
3  distintos
4  e informe qual é o maior.
5 }
6
7 program maior;
8 var
9     num1,num2 : integer;
10
11 begin
12     writeln('Digite o primeiro numero: ');
13     readln(num1);
14     writeln('Digite o segundo numero: ');
15     readln(num2);
16     if (num1 > num2) then
17         writeln('O primeiro numero', num1, 'eh mario que o segundo', num2);
18     else
19         writeln('O segundo numero ', num2, 'eh maior que o primeiro ', num1);
20 end.

```

Listagem 2.41: Verificar maior número Java

```

1 public class maior{
2
3     public static void main(String[] args){
4         double num1, num2;
5         System.out.println("Digite o primeiro número: ");
6         num1 = Integer.parseInt(System.console().readLine());
7         System.out.println("Digite o segundo número: ");
8         num2 = Integer.parseInt(System.console().readLine());
9
10        if(num1 > num2)
11            System.out.print("O primeiro número "+num1+" é maior que o segundo"+
                num2);
12        else
13            System.out.print("O segundo número "+num2+" é maior que o primeiro"+
                num1);
14
15
16    }
17 }

```

Listagem 2.42: Verificar maior número Smali

```

1 .class public Lmaior;
2 .super Ljava/lang/Object;
3 .source "maior.java"
4

```

2.2

```
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 8
19
20     .prologue
21     .line 5
22     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
23
24     const-string v1, "Digite o primeiro n\u00f3fameiro: "
25
26     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->println(Ljava/lang/
        String;)V
27
28     .line 6
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     int-to-double v0, v0
42
43     .line 7
44     sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
45
46     const-string v3, "Digite o segundo n\u00f3fameiro: "
47
48     invoke-virtual {v2, v3}, Ljava/io/PrintStream;-->println(Ljava/lang/
        String;)V
49
50     .line 8
51     invoke-static {}, Ljava/lang/System;-->console()Ljava/io/Console;
52
53     move-result-object v2
54
55     invoke-virtual {v2}, Ljava/io/Console;-->readLine()Ljava/lang/String;
56
57     move-result-object v2
58
59     invoke-static {v2}, Ljava/lang/Integer;-->parseInt(Ljava/lang/String;)I
60
61     move-result v2
```

```

62
63     int-to-double v2, v2
64
65     .line 10
66     cmpl-double v4, v0, v2
67
68     if-lez v4, :cond_4f
69
70     .line 11
71     sget-object v4, Ljava/lang/System; ->out:Ljava/io/PrintStream;
72
73     new-instance v5, Ljava/lang/StringBuilder;
74
75     invoke-direct {v5}, Ljava/lang/StringBuilder; -><init>()V
76
77     const-string v6, "O primeiro n\u00f3fameiro "
78
79     invoke-virtual {v5, v6}, Ljava/lang/StringBuilder; ->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
80
81     move-result-object v5
82
83     invoke-virtual {v5, v0, v1}, Ljava/lang/StringBuilder; ->append(D)Ljava
        /lang/StringBuilder;
84
85     move-result-object v0
86
87     const-string v1, " \u00e9 maior que o segundo"
88
89     invoke-virtual {v0, v1}, Ljava/lang/StringBuilder; ->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
90
91     move-result-object v0
92
93     invoke-virtual {v0, v2, v3}, Ljava/lang/StringBuilder; ->append(D)Ljava
        /lang/StringBuilder;
94
95     move-result-object v0
96
97     invoke-virtual {v0}, Ljava/lang/StringBuilder; ->toString()Ljava/lang/
        String;
98
99     move-result-object v0
100
101     invoke-virtual {v4, v0}, Ljava/io/PrintStream; ->print(Ljava/lang/
        String;)V
102
103     .line 16
104     :goto_4e
105     return-void
106
107     .line 13
108     :cond_4f
109     sget-object v4, Ljava/lang/System; ->out:Ljava/io/PrintStream;
110
111     new-instance v5, Ljava/lang/StringBuilder;
112
113     invoke-direct {v5}, Ljava/lang/StringBuilder; -><init>()V
114

```



```

115     const-string v6, "O segundo n\u00f3famer0 "
116
117     invoke-virtual {v5, v6}, Ljava/lang/StringBuilder;-->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
118
119     move-result-object v5
120
121     invoke-virtual {v5, v2, v3}, Ljava/lang/StringBuilder;-->append(D)Ljava
        /lang/StringBuilder;
122
123     move-result-object v2
124
125     const-string v3, " \u00e9 maior que o primeiro"
126
127     invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;-->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
128
129     move-result-object v2
130
131     invoke-virtual {v2, v0, v1}, Ljava/lang/StringBuilder;-->append(D)Ljava
        /lang/StringBuilder;
132
133     move-result-object v0
134
135     invoke-virtual {v0}, Ljava/lang/StringBuilder;-->toString()Ljava/lang/
        String;
136
137     move-result-object v0
138
139     invoke-virtual {v4, v0}, Ljava/io/PrintStream;-->print(Ljava/lang/
        String;)V
140
141     goto :goto_4e
142 .end method

```

Listagem 2.43: Verifica se est entre 100 e 200 Pascal

```

1 {
2 algoritmo "micro03"
3 /* Funo : Faa um algoritmo que receba um nmero e diga se este n
4 mero
5 est no intervalo entre 100 e 200.
6 /
7 *
8 var
9 numero: inteiro
10 incio
11 escreva("Digite um nmero: ")
12 leia(numero)
13 se numero >= 100 ento
14 se numero <= 200 ento
15 escreval("O nmero est no intervalo entre 100 e 200")
16 seno
17 escreval("O nmero no est no intervalo entre 100 e 200")
18 }
19
20 program entre;
21 var
22     numero : integer;

```

```

23 begin
24 write('Digite um numero :');
25 readln(numero);
26 if (numero >= 100) then
27 begin
28   if (numero <= 200) then
29     writeln('O numero esta no intervalo entre 100 e 200')
30   else
31     writeln('O numero nao esta no intervalo entre 100 e 200');
32   end
33 else
34   writeln('O numero nao esta no intervalo entre 100 e 200');
35 end.

```

Listagem 2.44: Verifica se está entre 100 e 200 Java

```

1
2 public class entre{
3
4     public static void main(String[] args){
5         int numero;
6         System.out.println("Digite um número: ");
7         numero = Integer.parseInt(System.console().readLine());
8
9         if(numero >= 100)
10            if(numero <= 200)
11                System.out.println("O número está no intervalo entre 100 e 200");
12            else
13                System.out.println("O número não está no intervalo entre 100 e 200");
14
15        else
16            System.out.println("O número não está no intervalo entre 100 e 200");
17
18    }
19 }
20 }

```

Listagem 2.45: Verifica se está entre 100 e 200 Smali

```

1 .class public Lentre;
2 .super Ljava/lang/Object;
3 .source "entre.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10     .prologue
11     .line 2
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

```

2.2

```
19
20 .prologue
21 .line 6
22 sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
23
24 const-string v1, "Digite um n\u00fameiro: "
25
26 invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->println(Ljava/lang/
    String;)V
27
28 .line 7
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 9
42 const/16 v1, 0x64
43
44 if-lt v0, v1, :cond_2b
45
46 .line 10
47 const/16 v1, 0xc8
48
49 if-gt v0, v1, :cond_23
50
51 .line 11
52 sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
53
54 const-string v1, "O n\u00fameiro est\u00e1 no intervalo entre 100 e 200
    "
55
56 invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->println(Ljava/lang/
    String;)V
57
58 .line 19
59 :goto_22
60 return-void
61
62 .line 13
63 :cond_23
64 sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
65
66 const-string v1, "O n\u00fameiro n\u00e3o est\u00e1 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 16
```

```

73      :cond_2b
74      sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
75
76      const-string v1, "O n\u00famero n\u00e3o est\u00e1 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 2.46: Verifica vários valores Pascal

```

1  {
2  algoritmo "micro04"
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  program micro04;
7  var
8      x,num,intervalo : integer;
9  begin
10     for x := 1 to 5 do
11     begin
12         write('Digite um numero: ');
13         readln(num);
14         if (num >= 10) then
15             begin
16                 if(num <= 150) then
17                     intervalo := intervalo + 1;
18             end;
19     end;
20     writeln('Ao total, foram digitados', intervalo, ' numeros no intervalo
        entre 10 e 150');
21 end.

```

Listagem 2.47: Verifica vários valores Java

```

1
2 public class verificaintervalos{
3
4     public static void main(String[] args){
5         int x, num, intervalo = 0;
6
7
8
9         for(x = 0; x < 5; x++){
10             System.out.println("Digite um número: ");
11             num = Integer.parseInt(System.console().readLine());
12
13             if(num >= 10)
14                 if(num <= 150)
15                 intervalo++;
16
17
18
19     }

```

```

20         System.out.println("Ao total, foram digitados "+intervalo+" nú
           meros no intervalo entre 10 e 150");
21     }
22 }

```

Listagem 2.48: Verifica vários valores Smali

```

1  .class public Lverificaintervalos;
2  .super Ljava/lang/Object;
3  .source "verificaintervalos.java"
4
5
6  # direct methods
7  .method public constructor <init>()V
8      .registers 1
9
10     .prologue
11     .line 2
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     const/4 v0, 0x0
22
23     .line 5
24     move v1, v0
25
26     .line 9
27     :goto_2
28     const/4 v2, 0x5
29
30     if-ge v1, v2, :cond_25
31
32     .line 10
33     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
34
35     const-string v3, "Digite um número: "
36
37     invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
38
39     .line 11
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 13
53     const/16 v3, 0xa
54
55     if-lt v2, v3, :cond_22
56
57     .line 14
58     const/16 v3, 0x96
59
60     if-gt v2, v3, :cond_22
61
62     .line 15
63     add-int/lit8 v0, v0, 0x1
64
65     .line 9
66     :cond_22
67     add-int/lit8 v1, v1, 0x1
68
69     goto :goto_2
70
71     .line 20
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, " n\u00f3meros 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 21
102     return-void
103 .end method

```

```

1 {
2  algoritmo "micro05"
3  /* Função : Escrever um algoritmo que leia o nome e o sexo de 56
4  pessoas
5  e informe o nome e se ela é homem ou mulher. No final
6  informe
7  o total de homens e de mulheres.
8  }
9  program micro05;
10 var
11  nome : string;
12  sexo : char;
13  x, h, m : integer;
14 begin
15
16  for x := 1 to 5 do
17  begin
18      write('Digite o nome: ');
19      readln(nome);
20      write('H - Homem ou M - Mulher: ');
21      readln(sexo);
22      case sexo of
23          'H' : h := h+1;
24          'M' : m := m+1;
25      end;
26  end;
27  writeln('Foram inseridos ',h,' Homens');
28  writeln('Foram inseridas ',m,' Mulheres');
29 end.

```

Listagem 2.50: Strings e Chars Java

```

1
2 class leaisexo{
3
4     public static void main(String[] args){
5         int x,m = 0, h = 0;
6         String sexo, nome;
7
8         for(x = 0; x < 5; x++){
9             System.out.println("Digite o nome: ");
10            nome = System.console().readLine();
11
12            System.out.println("H - homem ou M - mulher: ");
13            sexo = System.console().readLine();
14
15            switch(sexo){
16                case "M":
17                    m += 1;
18                    break;
19
20                case "H":
21                    h += 1;
22                    break;
23
24                default:
25                    System.out.println("Sexo só pode ser H ou M!");
26            }
27

```

```

28         }
29
30         System.out.println("Foram inseridos "+h+" homens.");
31         System.out.println("Foram inseridos "+m+" mulheres.");
32     }
33 }

```

Listagem 2.51: Strings e Chars Smali

```

1 .class Lleaisexo;
2 .super Ljava/lang/Object;
3 .source "leiasexo.java"
4
5
6 # direct methods
7 .method constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 2
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 8
19
20     .prologue
21     const/4 v3, 0x0
22
23     .line 5
24     move v0, v3
25
26     move v1, v3
27
28     move v4, v3
29
30     .line 8
31     :goto_4
32     const/4 v2, 0x5
33
34     if-ge v4, v2, :cond_54
35
36     .line 9
37     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
38
39     const-string v5, "Digite o nome: "
40
41     invoke-virtual {v2, v5}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
42
43     .line 10
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

```


2.2

```
50 .line 12
51 sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
52
53 const-string v5, "H - homem ou M - mulher: "
54
55 invoke-virtual {v2, v5}, Ljava/io/PrintStream;-->println(Ljava/lang/
    String;)V
56
57 .line 13
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 v5
65
66 .line 15
67 const/4 v2, -0x1
68
69 invoke-virtual {v5}, Ljava/lang/String;-->hashCode()I
70
71 move-result v6
72
73 sparse-switch v6, :sswitch_data_92
74
75 :cond_2c
76 :goto_2c
77 packed-switch v2, :pswitch_data_9c
78
79 .line 25
80 sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
81
82 const-string v5, "Sexo s\u00f3 pode ser H ou M!"
83
84 invoke-virtual {v2, v5}, Ljava/io/PrintStream;-->println(Ljava/lang/
    String;)V
85
86 .line 8
87 :goto_36
88 add-int/lit8 v2, v4, 0x1
89
90 move v4, v2
91
92 goto :goto_4
93
94 .line 15
95 :sswitch_3a
96 const-string v6, "M"
97
98 invoke-virtual {v5, v6}, Ljava/lang/String;-->equals(Ljava/lang/Object
    ;)Z
99
100 move-result v5
101
102 if-eqz v5, :cond_2c
103
104 move v2, v3
105
```

```

106     goto :goto_2c
107
108     :sswitch_44
109     const-string v6, "H"
110
111     invoke-virtual {v5, v6}, Ljava/lang/String;->equals(Ljava/lang/Object
        ;)Z
112
113     move-result v5
114
115     if-eqz v5, :cond_2c
116
117     const/4 v2, 0x1
118
119     goto :goto_2c
120
121     .line 17
122     :pswitch_4e
123     add-int/lit8 v1, v1, 0x1
124
125     .line 18
126     goto :goto_36
127
128     .line 21
129     :pswitch_51
130     add-int/lit8 v0, v0, 0x1
131
132     .line 22
133     goto :goto_36
134
135     .line 30
136     :cond_54
137     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
138
139     new-instance v3, Ljava/lang/StringBuilder;
140
141     invoke-direct {v3}, Ljava/lang/StringBuilder;-<init>()V
142
143     const-string v4, "Foram inseridos "
144
145     invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
146
147     move-result-object v3
148
149     invoke-virtual {v3, v0}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
150
151     move-result-object v0
152
153     const-string v3, " homens."
154
155     invoke-virtual {v0, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
156
157     move-result-object v0
158
159     invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
        String;

```

```

160
161     move-result-object v0
162
163     invoke-virtual {v2, v0}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
164
165     .line 31
166     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
167
168     new-instance v2, Ljava/lang/StringBuilder;
169
170     invoke-direct {v2}, Ljava/lang/StringBuilder;-<init>()V
171
172     const-string v3, "Foram inseridos "
173
174     invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
175
176     move-result-object v2
177
178     invoke-virtual {v2, v1}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
179
180     move-result-object v1
181
182     const-string v2, " mulheres."
183
184     invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
185
186     move-result-object v1
187
188     invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
        String;
189
190     move-result-object v1
191
192     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
193
194     .line 32
195     return-void
196
197     .line 15
198     nop
199
200     :sswitch_data_92
201     .sparse-switch
202         0x48 -> :sswitch_44
203         0x4d -> :sswitch_3a
204     .end sparse-switch
205
206     :pswitch_data_9c
207     .packed-switch 0x0
208         :pswitch_4e
209         :pswitch_51
210     .end packed-switch
211 .end method

```

Listagem 2.52: Switch em Pascal

```

1 {
2 algoritmo "micro06"
3 /* Função : Faça um algoritmo que leia um número de 1 a 5 e o
4 escreva por
5 extenso. Caso o usuário digite um número que não esteja
6 neste intervalo, exibir mensagem: número inválido.
7
8 }
9 program micro05;
10 var
11     numero : integer;
12 begin
13     write('Digite um numero de 1 a 5: ');
14     readln(numero);
15     case numero of
16         1 : writeln('Um');
17         2 : writeln('Dois');
18         3 : writeln('Tres');
19         4 : writeln('Quatro');
20         5 : writeln('Cinco');
21         else writeln('Numero invalido!!!');
22     end;
23 end.

```

Listagem 2.53: Switch em Java

```

1
2 public class extenso{
3
4     public static void main(String[] args){
5         int num;
6
7
8         System.out.println("Digite um número de 1 a 5: ");
9         num = Integer.parseInt(System.console().readLine());
10
11
12         switch(num) {
13             case 1:
14                 System.out.println("Um");
15                 break;
16
17             case 2:
18                 System.out.println("Dois");
19                 break;
20
21             case 3:
22                 System.out.println("Três");
23                 break;
24
25             case 4:
26                 System.out.println("Quatro");
27                 break;
28
29             case 5:
30                 System.out.println("Cinco");
31                 break;
32

```

```

33         default:
34             System.out.println("Número inválido");
35     }
36 }
37 }

```

Listagem 2.54: Switch em Smali

```

1  .class public Lextenso;
2  .super Ljava/lang/Object;
3  .source "extenso.java"
4
5
6  # direct methods
7  .method public constructor <init>()V
8      .registers 1
9
10     .prologue
11     .line 2
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 8
22     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
23
24     const-string v1, "Digite um número de 1 a 5: "
25
26     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->println(Ljava/lang/
        String;)V
27
28     .line 9
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 12
42     packed-switch v0, :pswitch_data_46
43
44     .line 34
45     sget-object v0, Ljava/lang/System;-->out:Ljava/io/PrintStream;
46
47     const-string v1, "Número inválido"
48
49     invoke-virtual {v0, v1}, Ljava/io/PrintStream;-->println(Ljava/lang/
        String;)V

```

```

50
51     .line 36
52     :goto_ld
53     return-void
54
55     .line 14
56     :pswitch_1e
57     sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
58
59     const-string v1, "Um"
60
61     invoke-virtual {v0, v1}, Ljava/io/PrintStream; ->println(Ljava/lang/
        String;)V
62
63     goto :goto_ld
64
65     .line 18
66     :pswitch_26
67     sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
68
69     const-string v1, "Dois"
70
71     invoke-virtual {v0, v1}, Ljava/io/PrintStream; ->println(Ljava/lang/
        String;)V
72
73     goto :goto_ld
74
75     .line 22
76     :pswitch_2e
77     sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
78
79     const-string v1, "Tr\u00eas"
80
81     invoke-virtual {v0, v1}, Ljava/io/PrintStream; ->println(Ljava/lang/
        String;)V
82
83     goto :goto_ld
84
85     .line 26
86     :pswitch_36
87     sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
88
89     const-string v1, "Quatro"
90
91     invoke-virtual {v0, v1}, Ljava/io/PrintStream; ->println(Ljava/lang/
        String;)V
92
93     goto :goto_ld
94
95     .line 30
96     :pswitch_3e
97     sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
98
99     const-string v1, "Cinco"
100
101     invoke-virtual {v0, v1}, Ljava/io/PrintStream; ->println(Ljava/lang/
        String;)V
102
103     goto :goto_ld

```

```

104
105     .line 12
106     :pswitch_data_46
107     .packed-switch 0x1
108         :pswitch_1e
109         :pswitch_26
110         :pswitch_2e
111         :pswitch_36
112         :pswitch_3e
113     .end packed-switch
114 .end method

```

Listagem 2.55: Verifica positivo negativo e zero Pascal

```

1
2 (* Função : Faça um algoritmo que receba N números e mostre positivo
3 ,
4 negativo ou zero para cada número.
5 *)
6
7 program micro07;
8 var
9     programa: boolean;
10    numero : integer;
11    opc :char;
12 begin
13     programa := true;
14     while (programa) do
15     begin
16         write('Digite um numero: ');
17         readln(numero);
18         if (numero > 0 ) then
19         begin
20             writeln('o numero eh positivo');
21         end
22         else
23         begin
24             if (numero = 0) then
25             begin
26                 writeln('o numero eh 0')
27             end
28             else
29             begin
30                 writeln('o numero eh negativo');
31             end;
32
33         end;
34         writeln('Deseja finalizar o programa? R: (S/N) ');
35         readln(opc);
36         if opc = 'S' then
37         begin
38             programa := false;
39         end
40         end;
41     end.

```

Listagem 2.56: Verifica positivo negativo e zero Java

```

2 public class posnegzero{
3
4     public static void main(String[] args){
5         int numero, programa;
6         String opc;
7
8         programa = 1;
9
10        while(programa == 1){
11            System.out.println("Digite um número: ");
12            numero = Integer.parseInt(System.console().readLine());
13
14            if(numero > 0 ) System.out.println("Positivo");
15            else if (numero == 0) System.out.println("O número é igual a
16                zero.");
17            else System.out.println("Negativo");
18
19            System.out.println("Deseja finalizar? (S/N)");
20            opc = System.console().readLine();
21
22            if(opc.equals("S")) programa = 0;
23        }
24 }

```

Listagem 2.57: Verifica positivo negativo e zero Smali

```

1 .class public Lposnegzero;
2 .super Ljava/lang/Object;
3 .source "posnegzero.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 2
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     const/4 v1, 0x1
22
23     .line 8
24     move v0, v1
25
26     .line 10
27     :cond_2
28     :goto_2
29     if-ne v0, v1, :cond_4b
30
31     .line 11
32     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;

```


2.2

```
33
34     const-string v3, "Digite um número: "
35
36     invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
37
38     .line 12
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 14
52     if-lez v2, :cond_39
53
54     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
55
56     const-string v3, "Positivo"
57
58     invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
59
60     .line 18
61     :goto_20
62     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
63
64     const-string v3, "Deseja finalizar? (S/N) "
65
66     invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
67
68     .line 19
69     invoke-static {}, Ljava/lang/System;->console() Ljava/io/Console;
70
71     move-result-object v2
72
73     invoke-virtual {v2}, Ljava/io/Console;->readLine() Ljava/lang/String;
74
75     move-result-object v2
76
77     .line 21
78     const-string v3, "S"
79
80     invoke-virtual {v2, v3}, Ljava/lang/String;->equals(Ljava/lang/Object
        ;) Z
81
82     move-result v2
83
84     if-eqz v2, :cond_2
85
86     const/4 v0, 0x0
87
```

```

88     goto :goto_2
89
90     .line 15
91     :cond_39
92     if-nez v2, :cond_43
93
94     sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
95
96     const-string v3, "O n\u00famero \u00e9 igual a zero."
97
98     invoke-virtual {v2, v3}, Ljava/io/PrintStream;-->println(Ljava/lang/
        String;)V
99
100    goto :goto_20
101
102    .line 16
103    :cond_43
104    sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
105
106    const-string v3, "Negativo"
107
108    invoke-virtual {v2, v3}, Ljava/io/PrintStream;-->println(Ljava/lang/
        String;)V
109
110    goto :goto_20
111
112    .line 23
113    :cond_4b
114    return-void
115 .end method

```

Listagem 2.58: maior ou menor que 10 Pascal

```

1 program micro08;
2 var
3     numero: integer;
4 begin
5     numero := 1;
6     while numero <> 0 do
7     begin
8         write('Digite um numero: ');
9         readln(numero);
10        if (numero > 10) then
11        begin
12            writeln('O numero', numero, ' eh maior que 10');
13        end
14        else
15        begin
16            writeln('O numero ', numero, ' eh menor que 10');
17        end;
18    end;
19 end.

```

Listagem 2.59: maior ou menor que 10 Java

```

1
2 public class maior10{
3
4     public static void main(String[] args){

```

```

5      int numero;
6
7      numero = 1;
8
9      while(numero != 0){
10         System.out.println("Digite um número: ");
11         numero = Integer.parseInt(System.console().readLine());
12
13         if(numero > 10 )
14             System.out.println("O numero "+numero+" é maior que dez.");
15         else
16             System.out.println("O numero "+numero+" é menor que dez.");
17
18     }
19 }
20 }

```

Listagem 2.60: maior ou menor que 10 Smali

```

1 .class public Lmaior10;
2 .super Ljava/lang/Object;
3 .source "maior10.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 2
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 7
22     const/4 v0, 0x1
23
24     .line 9
25     :goto_1
26     if-eqz v0, :cond_58
27
28     .line 10
29     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
30
31     const-string v1, "Digite um n\u00famero: "
32
33     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
34
35     .line 11
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/Integer;->parseInt(Ljava/lang/String;)I
45
46     move-result v0
47
48     .line 13
49     const/16 v1, 0xa
50
51     if-le v0, v1, :cond_39
52
53     .line 14
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, " \u00e9 maior que dez."
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 16
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
98     invoke-virtual {v2, v0}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
99
100    move-result-object v2
101
102    const-string v3, " \u00e9 menor que dez."
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 19
117    :cond_58
118    return-void
119 .end method

```

Listagem 2.61: Calcula preços Pascal

```

1  {testando comentário de forma diferente dando erro}
2
3  {erro de bloco
4
5
6  asdasdas
7  }
8
9  //TESTANDO
10 program micro09;
11 var
12 preco,venda,novo_preco : real;
13
14 begin
15     write('Digite o preco: ');
16     readln(preco);
17     write('Digite a venda: ');
18     readln(venda);
19     if(venda < 500) or (preco < 30) then
20         novo_preco := preco + 10/100 * preco
21     else if (((vendas >= 500) and (venda < 1200)) or ((preco >= 30) and (
        preco < 80))) then
22         novo_preco := preco + 15/100*preco
23     else if (venda >= 1200) or (preco >= 80) then
24         novo_preco := preco - 20/100*preco;

```

```

25     writeln('O novo preco eh', novo_preco);
26 end.

```

Listagem 2.62: Calcula preços Java

```

1
2 public class loja{
3
4     public static void main(String[] args){
5         double preco, venda, novo_preco;
6
7         System.out.println("Digite o preço: ");
8         preco = Double.parseDouble(System.console().readLine());
9
10        System.out.println("Digite a venda: ");
11        venda = Double.parseDouble(System.console().readLine());
12
13        if((venda < 500) || (preco < 30))
14            novo_preco = preco + ((10/100) * preco);
15        else{
16
17            if(((venda >= 500) && (venda < 1200)) || ((preco >= 30) && (
18                preco < 80)))
19                novo_preco = preco - ((15/100) * preco);
20            else if((venda >= 1200) || (preco >= 80))
21                novo_preco = preco - ((20/100)*preco);
22        }
23    }
24 }
25 }

```

Listagem 2.63: Calcula preços Smali

```

1 .class public Lloja;
2 .super Ljava/lang/Object;
3 .source "loja.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 2
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 7
19
20     .prologue
21     .line 7
22     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
23
24     const-string v1, "Digite o pre\u00e7o: "
25

```

2.2

```
26  invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
    String;)V
27
28  .line 8
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/Double;->parseDouble(Ljava/lang/String
    ;)D
38
39  move-result-wide v0
40
41  .line 10
42  sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
43
44  const-string v3, "Digite a venda: "
45
46  invoke-virtual {v2, v3}, Ljava/io/PrintStream;->println(Ljava/lang/
    String;)V
47
48  .line 11
49  invoke-static {}, Ljava/lang/System;->console()Ljava/io/Console;
50
51  move-result-object v2
52
53  invoke-virtual {v2}, Ljava/io/Console;->readLine()Ljava/lang/String;
54
55  move-result-object v2
56
57  invoke-static {v2}, Ljava/lang/Double;->parseDouble(Ljava/lang/String
    ;)D
58
59  move-result-wide v2
60
61  .line 13
62  const-wide v4, 0x407f400000000000L    # 500.0
63
64  cmpg-double v4, v2, v4
65
66  if-ltz v4, :cond_35
67
68  const-wide/high16 v4, 0x403e000000000000L    # 30.0
69
70  cmpg-double v4, v0, v4
71
72  if-gez v4, :cond_3a
73
74  .line 14
75  :cond_35
76  const-wide/16 v2, 0x0
77
78  mul-double/2addr v2, v0
79
80  add-double/2addr v0, v2
```

```

81
82     .line 24
83     :cond_39
84     :goto_39
85     return-void
86
87     .line 17
88     :cond_3a
89     const-wide v4, 0x407f400000000000L    # 500.0
90
91     cmpl-double v4, v2, v4
92
93     if-ltz v4, :cond_4c
94
95     const-wide v4, 0x4092c00000000000L    # 1200.0
96
97     cmpg-double v4, v2, v4
98
99     if-ltz v4, :cond_58
100
101     :cond_4c
102     const-wide/high16 v4, 0x403e000000000000L    # 30.0
103
104     cmpl-double v4, v0, v4
105
106     if-ltz v4, :cond_5d
107
108     const-wide/high16 v4, 0x4054000000000000L    # 80.0
109
110     cmpg-double v4, v0, v4
111
112     if-gez v4, :cond_5d
113
114     .line 18
115     :cond_58
116     const-wide/16 v2, 0x0
117
118     mul-double/2addr v2, v0
119
120     sub-double/2addr v0, v2
121
122     goto :goto_39
123
124     .line 19
125     :cond_5d
126     const-wide v4, 0x4092c00000000000L    # 1200.0
127
128     cmpl-double v2, v2, v4
129
130     if-gez v2, :cond_6c
131
132     const-wide/high16 v2, 0x4054000000000000L    # 80.0
133
134     cmpl-double v2, v0, v2
135
136     if-ltz v2, :cond_39
137
138     .line 20
139     :cond_6c

```



```

140     const-wide/16 v2, 0x0
141
142     mul-double/2addr v2, v0
143
144     sub-double/2addr v0, v2
145
146     goto :goto_39
147 .end method

```

Listagem 2.64: Fatorial em Pascal

```

1 {
2 algoritmo "micro10"
3 /* Função : recebe um número e calcula recursivamente o fatorial
4 desse número.
5 */
6 var
7 numero: inteiro
8 fat: inteiro
9 início
10 escreva("Digite um número: ")
11 leia(numero)
12 fat      fatorial(numero)
13 escreva("O fatorial de ")
14 escreva(numero)
15 escreva(" é ")
16 escreval(fat)
17 fim_algoritmo
18
19
20 função fatorial(n: inteiro) : inteiro
21 se n <= 0 então
22 retorne 1
23 senão
24 retorne n * fatorial(n-1)
25 fim_função fatorial
26 }
27 program micro10;
28
29 function fatorial (n:integer) : integer;
30 begin
31 if(n <= 0) then
32     fatorial := 1
33 else
34     fatorial := n * fatorial(n-1);
35 end;
36
37 var
38     numero,fat : integer;
39
40 begin
41     write('Digite um numero: ');
42     readln(numero);
43     fat := fatorial(numero);
44     writeln('O fatorial de ', numero, 'eh', fat);
45 end.

```

Listagem 2.65: Fatorial em Java

```

1
2 public class fat{
3
4     public static void main(String[] args){
5         int numero, fat;
6
7         System.out.println("Digite um numero: ");
8         numero = Integer.parseInt(System.console().readLine());
9
10        fat = fatorial(numero);
11
12        System.out.println("O fatorial de "+ numero);
13        System.out.println("é");
14        System.out.println(fat);
15    }
16
17    public static int fatorial(int numero){
18        if(numero <= 0)
19            return 1;
20        else
21            return numero * fatorial(numero - 1);
22    }
23 }

```

Listagem 2.66: Fatorial em Smali

```

1 .class public Lfat;
2 .super Ljava/lang/Object;
3 .source "fat.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 2
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 18
22    if-gtz p0, :cond_4
23
24    .line 19
25    const/4 v0, 0x1
26
27    .line 21
28    :goto_3
29    return v0
30
31    :cond_4
32    add-int/lit8 v0, p0, -0x1
33

```

2.2

```
34     invoke-static {v0}, Lfat;->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 6
45
46     .prologue
47     .line 7
48     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
49
50     const-string v1, "Digite um numero: "
51
52     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
53
54     .line 8
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 10
68     invoke-static {v0}, Lfat;->fatorial(I)I
69
70     move-result v1
71
72     .line 12
73     sget-object v2, Ljava/lang/System;->out:Ljava/io/PrintStream;
74
75     new-instance v3, Ljava/lang/StringBuilder;
76
77     invoke-direct {v3}, Ljava/lang/StringBuilder;-><init>()V
78
79     const-string v4, "O fatorial de "
80
81     invoke-virtual {v3, v4}, Ljava/lang/StringBuilder;->append(Ljava/lang/
        String;)Ljava/lang/StringBuilder;
82
83     move-result-object v3
84
85     invoke-virtual {v3, v0}, Ljava/lang/StringBuilder;->append(I)Ljava/
        lang/StringBuilder;
86
87     move-result-object v0
88
89     invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/
```

```

        String;
90
91     move-result-object v0
92
93     invoke-virtual {v2, v0}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
94
95     .line 13
96     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
97
98     const-string v2, "\u00e9"
99
100    invoke-virtual {v0, v2}, Ljava/io/PrintStream;->println(Ljava/lang/
        String;)V
101
102    .line 14
103    sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
104
105    invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(I)V
106
107    .line 15
108    return-void
109 .end method

```

Listagem 2.67: Celsius em Fahrenheit Pascal

```

1 program microll;
2
3 function verifica (n:integer) : integer;
4 begin
5 if n > 0 then
6 verifica := 1
7 else if n < 0 then
8 verifica := -1
9 else
10 verifica := 0;
11 end;
12
13 var
14     numero,x : integer;
15
16 begin
17     write('Digite um numero: ');
18     readln(numero);
19     x := verifica(numero);
20     if x = 1 then
21         writeln('Numero positivo')
22     else if x = 0 then
23         writeln('zero')
24     else writeln('numero negativo');
25 end.

```

Listagem 2.68: Celsius em Fahrenheit Java

```

1
2 public class verificanumbers{
3
4     public static void main(String[] args){
5         int x, numero;

```

2.2

```
6      System.out.print("Digite um número: ");
7      numero = Integer.parseInt(System.console().readLine());
8
9
10     x = verifica(numero);
11
12     if(x == 1)
13         System.out.println("Número positivo");
14     else if(x == 0)
15         System.out.println("Zero");
16     else
17         System.out.println("Número negativo");
18
19
20
21 }
22
23 public static int verifica(int numero){
24     int res;
25     if(numero > 0) res = 1;
26     else{
27         if(numero < 0 )
28             res = -1;
29         else
30             res = 0;
31     }
32     return res;
33 }
34 }
```

Listagem 2.69: Celsius em Fahrenheit Smali

```
1 .class public Lverificanumbers;
2 .super Ljava/lang/Object;
3 .source "verificanumbers.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10     .prologue
11     .line 2
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 6
22     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
23
24     const-string v1, "Digite um n\u00famero: "
25
26     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->print(Ljava/lang/
        String;)V
```

```

27
28 .line 7
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 10
42 invoke-static {v0}, Lverificanumbers; ->verifica(I)I
43
44 move-result v0
45
46 .line 12
47 const/4 v1, 0x1
48
49 if-ne v0, v1, :cond_22
50
51 .line 13
52 sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
53
54 const-string v1, "N\u00famero positivo"
55
56 invoke-virtual {v0, v1}, Ljava/io/PrintStream; ->println(Ljava/lang/
    String;)V
57
58 .line 21
59 :goto_21
60 return-void
61
62 .line 14
63 :cond_22
64 if-nez v0, :cond_2c
65
66 .line 15
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 17
76 :cond_2c
77 sget-object v0, Ljava/lang/System; ->out:Ljava/io/PrintStream;
78
79 const-string v1, "N\u00famero 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 25
91     if-lez p0, :cond_4
92
93     const/4 v0, 0x1
94
95     .line 32
96     :goto_3
97     return v0
98
99     .line 27
100    :cond_4
101    if-gez p0, :cond_8
102
103    .line 28
104    const/4 v0, -0x1
105
106    goto :goto_3
107
108    .line 30
109    :cond_8
110    const/4 v0, 0x0
111
112    goto :goto_3
113 .end method

```

2.3 Mais informações sobre SMALI

2.3.1 Registradores

Em um bytecode para dalvik os registradores sempre serão de 32 bits e podem guardar qualquer tipo de valor, porém 2 registradores podem ser usados para guardar tipos de 64 bits.

Existem duas formas de se especificar quantos registradores serão utilizados em um método:

.registers que deve ter o número total de registradores.

.locals que terá o número de registradores utilizados no método sem contar os utilizados como parâmetros.

Como parâmetros são passados: Quando um método é invocado, os parâmetros são colocados nos últimos n registradores. Se um método tem dois argumentos e 5 registradores (v0-v4), os argumentos serão colocados nos 2 últimos: v3 e v4.

O primeiro parâmetro de um método não-estático sempre será o objeto em que o método foi invocado.

Por exemplo, vamos dizer que você escreve o seguinte método não estático:

```
LMyObject;->callMe(II)V.
```

Observe que nesse método você tem 2 inteiros como parâmetro, porém também tem LMyObject; como parâmetro implícito antes dos dois inteiros, então esse método tem um total de 3 argumentos.

Digamos que você especifique 5 registradores no método (v0-v4), você terá então o .registers 5 (total de 5 registradores) .locals 2 (2 registradores locais + 3 registradores de parâmetros). Dessa forma o objeto da onde veio a invocação ficará em v2, o argumento1 em v3 e o argumento2 em v4.

Para métodos estáticos o argumento implícito (nome do objeto que invoca) não será passado.

Nomeando os registradores: Existem duas nomenclaturas para os registradores, os normais "v" e os "p" que são registradores de parâmetros. Isso somente para facilitar a utilização de registradores, logo você pode chamar os registradores pelo seu próprio nome (v2 não necessita ser p0 por exemplo).

Local	Param	
v0		O primeiro registrador local
v1		O segundo registrador local
v2	p0	O primeiro parametro registrador
v3	p1	O secundo parametro registrador
v4	p2	O terceiro parametro registrador

Valores Long/Double: Como dito anteriormente, as primitivas long e double (J e D) são de 64 bits e requerem portanto 2 registradores.

Exemplo:

Register	Type
p0	this
p1	I
p2, p3	J
p4	Z

Lembre-se que então será necessário especificar os dois registradores para qualquer double/-longe para realizar uma instrução de invocação do método.

2.3.2 Tipos

Dalvik possui basicamente duas classes de tipos, os tipos primitivos e os referenciados. Dos referenciados se diz respeito a arrays e objetos, todo o resto é caracterizado como tipo primitivo.

Os tipos primitivos são representados com uma única letra:

Observação: As extremidades são representadas sempre em Little-endian!!! Ou seja, Byte menos significativo no menor endereço

V	void - Podem somente ser usado como retorno
Z	boolean
B	byte
S	short
C	char
I	int
J	long (64 bits)
F	float
D	double (64 bits)

Objetos tomam o seguinte formato:

```
Lpackage/name/ObjectName;
```

Onde L indica o tipo desse objeto, package/name/ é o pacote cujo o objeto pertence, ObjectName é o nome do objeto onde ";"denota o fim do nome do objeto. A linha cima tem o seguinte equivalente em java:

```
package.name.ObjectName
```

Outro exemplo:

```
Ljava/lang/String;
```

equivale ao seguinte em java:

```
java.lang.String
```

Vetores tomam a forma de por exemplo [I para um vetor de inteiros int[]]. Para vetores de múltiplas dimensões basta que se adicione "["a mais antes do tipo, [III = int[][][]]. Número máximo de dimensões: 255

2.3.3 Métodos

Métodos são especificados de forma bastante verbosa, em um método contém: O tipo que contém o método, seu nome, seus tipos de parâmetros e seu tipo de retorno. Esses dados são necessários para melhorar a performance da máquina virtual.

Eles com uma forma parecida com a seguinte:

```
Lpackage/name/ObjectName;->MethodName(III)Z
```

Lpackage/name/ObjectName é reconhecido como um tipo. O nome do método está obviamente descrito, na assinatura do método é possível encontrar (III)Z o que representam 3 inteiros como parâmetro e o Z ao final representa o tipo de retorno, que no caso apresentado é um booleano.

Observe que os parâmetros são listados um a direita do outro sem separadores.

Um exemplo mais complexo:

```
method(I[[IILjava/lang/String;[Ljava/lang/Object;)Ljava/lang/String;
```

Em java isso deve se parecer com:

```
String method(int, int[][], int, String, Object[])
```

Muito mais informações podem ser encontradas na wiki da máquina virtual Dalvik, o link se encontra nas referências.

2.4 Instruções Importantes

Mnemonic / Syntax	Description
<code>nop</code>	Waste cycles.
<code>move vA, vB</code>	Move the contents of one non-object register to another.
<code>move/from16 vAA, vBBBB</code>	Move the contents of one non-object register to another.
<code>move/16 vAAAA, vBBBB</code>	Move the contents of one non-object register to another.
<code>move-wide vA, vB</code>	Move the contents of one register-pair to another. Note: It is legal to move from <i>vN</i> to either <i>vN-1</i> or <i>vN+1</i> , so implementations must arrange for both halves of a register pair to be read before anything is written.
<code>move-wide/from16 vAA, vBBBB</code>	Move the contents of one register-pair to another. Note: Implementation considerations are the same as <code>move-wide</code> , above.
<code>move-wide/16 vAAAA, vBBBB</code>	Move the contents of one register-pair to another. Note: Implementation considerations are the same as <code>move-wide</code> , above.
<code>move-object vA, vB</code>	Move the contents of one object-bearing register to another.
<code>move-object/from16 vAA, vBBBB</code>	Move the contents of one object-bearing register to another.
<code>move-object/16 vAAAA, vBBBB</code>	Move the contents of one object-bearing register to another.
<code>move-result vAA</code>	Move the single-word non-object result of the most recent <i>invoke-kind</i> into the indicated register. This must be done as the instruction immediately after an <i>invoke-kind</i> whose (single-word, non-object) result is not to be ignored; anywhere else is invalid.
<code>move-result-wide vAA</code>	Move the double-word result of the most recent <i>invoke-kind</i> into the indicated register pair. This must be done as the instruction immediately after an <i>invoke-kind</i> whose (double-word) result is not to be ignored; anywhere else is invalid.
<code>move-result-object vAA</code>	Move the object result of the most recent <i>invoke-kind</i> into the indicated register. This must be done as the instruction immediately after an <i>invoke-kind</i> or <i>filled-new-array</i> whose (object) result is not to be ignored; anywhere else is invalid.
<code>move-exception vAA</code>	Save a just-caught exception into the given register. This should be the first instruction of any exception handler whose caught exception is not to be ignored, and this instruction may <i>only</i> ever occur as the first instruction of an exception handler; anywhere else is invalid.
<code>return-void</code>	Return from a void method.
<code>return vAA</code>	Return from a single-width (32-bit) non-object value-returning method.
<code>return-wide vAA</code>	Return from a double-width (64-bit) value-returning method.
<code>return-object vAA</code>	Return from an object-returning method.
<code>const/4 vA, #B</code>	Move the given literal value (sign-extended to 32 bits) into the specified register.
<code>const/16 vAA, #BBBB</code>	Move the given literal value (sign-extended to 32 bits) into the specified register.
<code>const vAA, #BBBBBBBB</code>	Move the given literal value into the specified register.
<code>const/high16 vAA, #BBBB0000</code>	Move the given literal value (right-zero-extended to 32 bits) into the specified register.

const-wide/16 vAA, #+BBBB	Move the given literal value (sign-extended to 64 bits) into the specified register-pair.
const-wide/32 vAA, #+BBBBBBBB	Move the given literal value (sign-extended to 64 bits) into the specified register-pair.
const-wide vAA, #+BBBBBBBBBBBBBBBB	Move the given literal value into the specified register-pair.
const-wide/high16 vAA, #+BBBB000000000000	Move the given literal value (right-zero-extended to 64 bits) into the specified register-pair.
const-string vAA, string@BBBB	Move a reference to the string specified by the given index into the specified register.
const-string/jumbo vAA, string@BBBBBBBB	Move a reference to the string specified by the given index into the specified register.
const-class vAA, type@BBBB	Move a reference to the class specified by the given index into the specified register. In the case where the indicated type is primitive, this will store a reference to the primitive type's degenerate class.
monitor-enter vAA	Acquire the monitor for the indicated object.
monitor-exit vAA	Release the monitor for the indicated object. Note: If this instruction needs to throw an exception, it must do so as if the pc has already advanced past the instruction. It may be useful to think of this as the instruction successfully executing (in a sense), and the exception getting thrown <i>after</i> the instruction but <i>before</i> the next one gets a chance to run. This definition makes it possible for a method to use a monitor cleanup catch-all (e.g., <code>finally</code>) block as the monitor cleanup for that block itself, as a way to handle the arbitrary exceptions that might get thrown due to the historical implementation of <code>Thread.stop()</code> , while still managing to have proper monitor hygiene.
check-cast vAA, type@BBBB	Throw if the reference in the given register cannot be cast to the indicated type. The type must be a reference type (not a primitive type).
instance-of vA, vB, type@CCCC	Store in the given destination register 1 if the indicated reference is an instance of the given type, or 0 if not. The type must be a reference type (not a primitive type).
array-length vA, vB	Store in the given destination register the length of the indicated array, in entries
new-instance vAA, type@BBBB	Construct a new instance of the indicated type, storing a reference to it in the destination. The type must refer to a non-array class.
new-array vA, vB, type@CCCC	Construct a new array of the indicated type and size. The type must be an array type.
filled-new-array {vD, vE, vF, vG, vA}, type@CCCC	Construct an array of the given type and size, filling it with the supplied contents. The type must be an array type. The array's contents must be single-word (that is, no arrays of <code>long</code> or <code>double</code>). The constructed instance is stored as a "result" in the same way that the method invocation instructions store their results, so the constructed instance must be moved to a register with a subsequent <code>move-result-object</code> instruction (if it is to be used).
filled-new-array/range {vCCCC .. vNNNN}, type@BBBB	Construct an array of the given type and size, filling it with the supplied contents. Clarifications and restrictions are the same as <code>filled-new-array</code> , described above.
fill-array-data vAA, +BBBBBBBB (with supplemental data as specified below in "fill-array-data Format")	Fill the given array with the indicated data. The reference must be to an array of primitives, and the data table must match it in type and size. Note: The address of the table is guaranteed to be even (that is, 4-byte aligned). If the code size of the method is otherwise odd, then an extra code unit is inserted between the main code and the table whose value is the same as a <code>nop</code> .
throw vAA	Throw the indicated exception.
goto +AA	Unconditionally jump to the indicated instruction.

goto/16 +AAAA	Unconditionally jump to the indicated instruction. Note: The branch offset may not be 0. (A spin loop may be legally constructed either with goto/32 or by including a nop as a target before the branch.)
goto/32 +AAAAAAAA	Unconditionally jump to the indicated instruction.
packed-switch vAA, +BBBBBBBB (with supplemental data as specified below in "packed-switch Format")	Jump to a new instruction based on the value in the given register, using a table of offsets corresponding to each value in a particular integral range, or fall through to the next instruction if there is no match. Note: The address of the table is guaranteed to be even (that is, 4-byte aligned). If the code size of the method is otherwise odd, then an extra code unit is inserted between the main code and the table whose value is the same as a nop.
sparse-switch vAA, +BBBBBBBB (with supplemental data as specified below in "sparse-switch Format")	Jump to a new instruction based on the value in the given register, using an ordered table of value-offset pairs, or fall through to the next instruction if there is no match. Note: Alignment and padding considerations are identical to packed-switch, above.
cmpkind vAA, vBB, vCC 2d: cmp1-float (lt bias) 2e: cmpg-float (gt bias) 2f: cmp1-double (lt bias) 30: cmpg-double (gt bias) 31: cmp-long	Perform the indicated floating point or long comparison, storing 0 if the two arguments are equal, 1 if the second argument is larger, or -1 if the first argument is larger. The "bias" listed for the floating point operations indicates how NaN comparisons are treated: "Gt bias" instructions return 1 for NaN comparisons, and "lt bias" instructions return -1. For example, to check to see if floating point $a < b$, then it is advisable to use cmpg-float; a result of -1 indicates that the test was true, and the other values indicate it was false either due to a valid comparison or because one or the other values was NaN.
if-test vA, vB, +CCCC 32: if-eq 33: if-ne 34: if-lt 35: if-ge 36: if-gt 37: if-le	Branch to the given destination if the given two registers' values compare as specified. Note: The branch offset may not be 0. (A spin loop may be legally constructed either by branching around a backward goto or by including a nop as a target before the branch.)
if-testz vAA, +BBBB 38: if-eqz 39: if-nez 3a: if-ltz 3b: if-gez 3c: if-gtz 3d: if-lez	Branch to the given destination if the given register's value compares with 0 as specified. Note: The branch offset may not be 0. (A spin loop may be legally constructed either by branching around a backward goto or by including a nop as a target before the branch.)
(unused)	(unused)
arrayop vAA, vBB, vCC 44: aget 45: aget-wide 46: aget-object 47: aget-boolean 48: aget-byte 49: aget-char 4a: aget-short 4b: aput 4c: aput-wide 4d: aput-object 4e: aput-boolean 4f: aput-byte 50: aput-char 51: aput-short	Perform the identified array operation at the identified index of the given array, loading or storing into the value register.

<i>instanceop</i> vA, vB, field@CCCC 52: <i>iget</i> 53: <i>iget-wide</i> 54: <i>iget-object</i> 55: <i>iget-boolean</i> 56: <i>iget-byte</i> 57: <i>iget-char</i> 58: <i>iget-short</i> 59: <i>iput</i> 5a: <i>iput-wide</i> 5b: <i>iput-object</i> 5c: <i>iput-boolean</i> 5d: <i>iput-byte</i> 5e: <i>iput-char</i> 5f: <i>iput-short</i>	Perform the identified object instance field operation with the identified field, loading or storing into the value register. Note: These opcodes are reasonable candidates for static linking, altering the field argument to be a more direct offset.
<i>sstaticop</i> vAA, field@BBBB 60: <i>sget</i> 61: <i>sget-wide</i> 62: <i>sget-object</i> 63: <i>sget-boolean</i> 64: <i>sget-byte</i> 65: <i>sget-char</i> 66: <i>sget-short</i> 67: <i>sput</i> 68: <i>sput-wide</i> 69: <i>sput-object</i> 6a: <i>sput-boolean</i> 6b: <i>sput-byte</i> 6c: <i>sput-char</i> 6d: <i>sput-short</i>	Perform the identified object static field operation with the identified static field, loading or storing into the value register. Note: These opcodes are reasonable candidates for static linking, altering the field argument to be a more direct offset.
<i>invoke-kind</i> {vD, vE, vF, vG, vA}, meth@CCCC 6e: <i>invoke-virtual</i> 6f: <i>invoke-super</i> 70: <i>invoke-direct</i> 71: <i>invoke-static</i> 72: <i>invoke-interface</i>	Call the indicated method. The result (if any) may be stored with an appropriate <i>move-result*</i> variant as the immediately subsequent instruction. <i>invoke-virtual</i> is used to invoke a normal virtual method (a method that is not <i>static</i> or <i>final</i> , and is not a constructor). <i>invoke-super</i> is used to invoke the closest superclass's virtual method (as opposed to the one with the same <i>method_id</i> in the calling class). <i>invoke-direct</i> is used to invoke a non-static direct method (that is, an instance method that is by its nature non-overridable, namely either a <i>private</i> instance method or a constructor). <i>invoke-static</i> is used to invoke a static method (which is always considered a direct method). <i>invoke-interface</i> is used to invoke an interface method, that is, on an object whose concrete class isn't known, using a <i>method_id</i> that refers to an interface. Note: These opcodes are reasonable candidates for static linking, altering the method argument to be a more direct offset (or pair thereof).
(unused)	(unused)
<i>invoke-kind/range</i> {vCCCC .. vNNNN}, meth@BBBB 74: <i>invoke-virtual/range</i> 75: <i>invoke-super/range</i> 76: <i>invoke-direct/range</i> 77: <i>invoke-static/range</i> 78: <i>invoke-interface/range</i>	Call the indicated method. See first <i>invoke-kind</i> description above for details, caveats, and suggestions.

<i>unop</i> vA, vB 7b: neg-int 7c: not-int 7d: neg-long 7e: not-long 7f: neg-float 80: neg-double 81: int-to-long 82: int-to-float 83: int-to-double 84: long-to-int 85: long-to-float 86: long-to-double 87: float-to-int 88: float-to-long 89: float-to-double 8a: double-to-int 8b: double-to-long 8c: double-to-float 8d: int-to-byte 8e: int-to-char 8f: int-to-short	Perform the identified unary operation on the source register, storing the result in the destination register.
<i>binop</i> vAA, vBB, vCC 90: add-int 91: sub-int 92: mul-int 93: div-int 94: rem-int 95: and-int 96: or-int 97: xor-int 98: shl-int 99: shr-int 9a: ushr-int 9b: add-long 9c: sub-long 9d: mul-long 9e: div-long 9f: rem-long a0: and-long a1: or-long a2: xor-long a3: shl-long a4: shr-long a5: ushr-long a6: add-float a7: sub-float a8: mul-float a9: div-float aa: rem-float ab: add-double ac: sub-double ad: mul-double ae: div-double af: rem-double	Perform the identified binary operation on the two source registers, storing the result in the first source register.

<code>binop/2addr vA, vB</code> <code>b0: add-int/2addr</code> <code>b1: sub-int/2addr</code> <code>b2: mul-int/2addr</code> <code>b3: div-int/2addr</code> <code>b4: rem-int/2addr</code> <code>b5: and-int/2addr</code> <code>b6: or-int/2addr</code> <code>b7: xor-int/2addr</code> <code>b8: shl-int/2addr</code> <code>b9: shr-int/2addr</code> <code>ba: ushr-int/2addr</code> <code>bb: add-long/2addr</code> <code>bc: sub-long/2addr</code> <code>bd: mul-long/2addr</code> <code>be: div-long/2addr</code> <code>bf: rem-long/2addr</code> <code>c0: and-long/2addr</code> <code>c1: or-long/2addr</code> <code>c2: xor-long/2addr</code> <code>c3: shl-long/2addr</code> <code>c4: shr-long/2addr</code> <code>c5: ushr-long/2addr</code> <code>c6: add-float/2addr</code> <code>c7: sub-float/2addr</code> <code>c8: mul-float/2addr</code> <code>c9: div-float/2addr</code> <code>ca: rem-float/2addr</code> <code>cb: add-double/2addr</code> <code>cc: sub-double/2addr</code> <code>cd: mul-double/2addr</code> <code>ce: div-double/2addr</code> <code>cf: rem-double/2addr</code>	<p>Perform the identified binary operation on the two source registers, storing the result in the first source register.</p>
<code>binop/lit16 vA, vB, #+CCCC</code> <code>d0: add-int/lit16</code> <code>d1: rsub-int (reverse subtract)</code> <code>d2: mul-int/lit16</code> <code>d3: div-int/lit16</code> <code>d4: rem-int/lit16</code> <code>d5: and-int/lit16</code> <code>d6: or-int/lit16</code> <code>d7: xor-int/lit16</code>	<p>Perform the indicated binary op on the indicated register (first argument) and literal value (second argument), storing the result in the destination register.</p> <p>Note: <code>rsub-int</code> does not have a suffix since this version is the main opcode of its family. Also, see below for details on its semantics.</p>
<code>binop/lit8 vAA, vBB, #+CCC</code> <code>d8: add-int/lit8</code> <code>d9: rsub-int/lit8</code> <code>da: mul-int/lit8</code> <code>db: div-int/lit8</code> <code>dc: rem-int/lit8</code> <code>dd: and-int/lit8</code> <code>de: or-int/lit8</code> <code>df: xor-int/lit8</code> <code>e0: shl-int/lit8</code> <code>e1: shr-int/lit8</code> <code>e2: ushr-int/lit8</code>	<p>Perform the indicated binary op on the indicated register (first argument) and literal value (second argument), storing the result in the destination register.</p> <p>Note: See below for details on the semantics of <code>rsub-int</code>.</p>

Capítulo 3

Construção de um Compilador

Um compilador é um programa que traduz um código fonte que está em uma linguagem em outra. Para este trabalho estarei construindo um compilador que traduz um arquivo escrito em pascal em um assembly equivalente que funcione em uma máquina virtual Dalvik. Para se construir um compilador são necessários alguns passos, abaixo esses passos são descritos e exemplificados.

3.1 Analisador Léxico

Em uma linguagem de programação define-se uma gramática, um alfabeto. Um analisador léxico lê todos os caracteres contidos em um código fonte, busca suas separações e então classifica cada elemento do código fonte como tokens, estes tokens podem ser palavras reservadas, identificadores, operadores aritméticos, operadores lógicos dentre outras possibilidades da linguagem. Além dessa classificação, é nesta fase que elementos que não serão utilizados, como comentários serão descartados e, alguns tipos de erros serão detectados, como por exemplo, o uso de caracteres não pertencentes ao alfabeto. Abaixo segue o código do analisador léxico escrito em OCAML.

Listagem 3.1: Analisador Léxico

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


3.1

```
17
18 let erro lin col msg =
19     let mensagem = sprintf "%d-%d: %s" lin col msg in
20     failwith mensagem
21
22 let erroComentario lin col msg =
23     let mensagem = sprintf "%d-%d: %s" lin col msg in
24     failwith mensagem
25
26 type tokens = APAR
27             | FPAR
28             | ATRIB
29             | VAR
30             | PONTOVIRG
31             | DOISPONTOS
32             | REAL
33             | INTEGER
34             | PONTO
35             | CHAR
36             | STRING
37             | PROGRAM
38             | BEGIN
39             | END
40             | ENDPROG
41             | LEIA
42             | LEIALN
43             | PRINT
44             | PRINTLN
45             | VIRG
46             | MAIS
47             | MENOS
48             | DIVIDE
49             | MULTIPLICA
50             | MODULO
51             | IGUAL
52             | MAIOR
53             | MENOR
54             | MAIORIGUAL
55             | MENORIGUAL
56             | DIFERENTE
57             | OU
58             | E
59             | IF
60             | THEN
61             | ELIFE
62             | ELSE
63             | FOR
64             | WHILE
65             | TO
66             | DO
67             | CASE
68             | OF
69             | BOOLEAN
70             | OF
71             | FUNCAO
72             | PROCEDURE
73             | LITINT of int
74             | LITBOOL of bool
75             | LITREAL of float
```

```

76         | LITSTRING of string
77         | ID of string
78         | EOF
79     }
80     let digito = ['0' - '9']
81     let inteiro = digito+
82     let real = digito* '.' digito+
83
84     let letra = ['a' - 'z' 'A' - 'Z']
85     let identificador = letra ( letra | digito | '_' ) *
86
87     let brancos = [' ' '\t'] +
88     let novalinha = '\r' | '\n' | "\r\n" | "\\n"
89
90     let comentario = "//" [ ^ '\r' '\n' ] *
91
92     rule token = parse
93     brancos      { token lexbuf }
94 | novalinha     { incr_num_linha lexbuf; token lexbuf }
95
96 | comentario { token lexbuf }
97 | "(" {
98     let pos = lexbuf.lex_curr_p in
99     let lin = pos.pos_lnum
100    and col = pos.pos_cnum - pos.pos_bol - 1 in
101    comentario_bloco lin col 0 lexbuf }
102 | "{" {
103     let pos = lexbuf.lex_curr_p in
104     let lin = pos.pos_lnum
105    and col = pos.pos_cnum - pos.pos_bol - 1 in
106    comentario_bloco_dif lin col 0 lexbuf }
107 | '(' { APAR }
108 | ')' { FPAR }
109 | "program" { PROGRAM }
110 | "var" { VAR }
111 | ',' { VIRG }
112 | ';' { PONTOVIRG }
113 | '.' { PONTO }
114 | ':' { DOISPONTOS }
115 | "real" { REAL }
116 | "boolean" { BOOLEAN }
117 | "of" { OF }
118 | "integer" { INTEGER }
119 | "string" { STRING }
120 | "char" { CHAR }
121 | "begin" { BEGIN }
122 | "end" { END }
123 | "end." { ENDPROG }
124 | "write" { PRINT }
125 | "writeln" { PRINTLN }
126 | "read" { LEIA }
127 | "readln" { LEIALN }
128 | '+' { MAIS }
129 | '-' { MENOS }
130 | '/' { DIVIDE }
131 | '*' { MULTIPLICA }
132 | "div" { MODULO }
133 | '>' { MAIOR }
134 | '<' { MENOR }

```

3.1

```

135 | '='          { IGUAL }
136 | ">="         { MAIORIGUAL }
137 | "<="         { MENORIGUAL }
138 | "<>"        { DIFERENTE }
139 | ":@"        { ATRIB }
140 | "or"         { OU }
141 | "and"        { E }
142 | "function"   { FUNCAO }
143 | "procedure"  { PROCEDURE }
144 | "if"         { IF }
145 | "else if"    { ELIFE }
146 | "then"       { THEN }
147 | "else"       { ELSE }
148 | "while"      { WHILE }
149 | "for"        { FOR }
150 | "to"         { TO }
151 | "do"         { DO }
152 | "case"       { CASE }
153 | "of"         { OF }
154 | inteiro as num { let numero = int_of_string num in
155 |                  LITINT numero }
156 | real as r {let r = float_of_string r in LITREAL r}
157 | identificador as id { ID id }
158 | '\\'       { let pos = lexbuf.lex_curr_p in
159 |               let lin = pos.pos_lnum
160 |               and col = pos.pos_cnum - pos.pos_bol - 1 in
161 |               let buffer = Buffer.create 1 in
162 |               let str = leia_string lin col buffer lexbuf in
163 |               LITSTRING str }
164 | _ as c { failwith (msg_erro lexbuf c) }
165 | eof    { EOF }
166
167
168 and comentario_bloco lin col n = parse
169   "*" ) { if n=0 then token lexbuf
170         else comentario_bloco lin col (n-1) lexbuf }
171 | "("   { comentario_bloco lin col (n+1) lexbuf }
172 | novalinha { incr_num_linha lexbuf; comentario_bloco lin col n lexbuf }
173 | _       { comentario_bloco lin col n lexbuf }
174 | eof     { erroComentario lin col "Comentario nao fechado" }
175
176 and comentario_bloco_dif lin col n = parse
177   ")" { if n=0 then token lexbuf
178         else comentario_bloco_dif lin col (n-1) lexbuf }
179 | "{" { comentario_bloco_dif lin col (n+1) lexbuf }
180 | novalinha { incr_num_linha lexbuf; comentario_bloco_dif lin col n
181 |             lexbuf }
182 | _       { comentario_bloco_dif lin col n lexbuf }
183 | eof     { erroComentario lin col "Comentario nao fechado" }
184
184 and leia_string lin col buffer = parse
185   '\\' { Buffer.contents buffer}
186 | "\\t" { Buffer.add_char buffer '\t'; leia_string lin col buffer
187 |         lexbuf }
187 | "\\n" { Buffer.add_char buffer '\n'; leia_string lin col buffer
188 |         lexbuf }
188 | '\\\'' { Buffer.add_char buffer '\''; leia_string lin col buffer
189 |         lexbuf }

```

```

189 | '\\\' '\\\' { Buffer.add_char buffer '\\\' ; leia_string lin col buffer
      lexbuf }
190 | _ as c      { Buffer.add_char buffer c ; leia_string lin col buffer lexbuf
      }
191 | eof          { erro lin col "A string nao foi fechada"}

```

Foi construído em sala de aula um programa responsável por carregar o analisador lexico junto a um arquivo do formato desejado e então obter a resposta do analisador lexico dentro de uma lista, segue o código desse programa:

Listagem 3.2: Carregador

```

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

```

Como exemplo, farei a execução do analisador léxico sobre o seguinte código:

Listagem 3.3: Fatorial em Pascal

```

1 program micro10;
2
3 function fatorial (n:integer) : integer;
4 begin
5   if(n <= 0) then
6     fatorial := 1
7   else
8     fatorial := n * fatorial(n-1);
9   end;
10
11 var
12   numero,fat : integer;
13
14 begin
15   write('Digite um numero: ');
16   readln(numero);
17   fat := fatorial(numero);
18   writeln('O fatorial de ', numero, 'eh', fat);
19 end.

```

Obtendo como saída:

```

- : Lexico.tokens list =
[Lexico.PROGRAM; Lexico.ID "micro10"; Lexico.PONTOVIRG; Lexico.FUNCAO;
 Lexico.ID "fatorial"; Lexico.APAR; Lexico.ID "n"; Lexico.DOISPONTOS;
 Lexico.INTEGER; Lexico.FPAR; Lexico.DOISPONTOS; Lexico.INTEGER;
 Lexico.PONTOVIRG; Lexico.BEGIN; Lexico.IF; Lexico.APAR; Lexico.ID "n";
 Lexico.MENORIGUAL; Lexico.LITINT 0; Lexico.FPAR; Lexico.THEN;
 Lexico.ID "fatorial"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.ELSE;
 Lexico.ID "fatorial"; Lexico.ATRIB; Lexico.ID "n"; Lexico.MULTIPLICA;
 Lexico.ID "fatorial"; Lexico.APAR; Lexico.ID "n"; Lexico.MENOS;
 Lexico.LITINT 1; Lexico.FPAR; Lexico.PONTOVIRG; Lexico.END;
 Lexico.PONTOVIRG; Lexico.VAR; Lexico.ID "numero"; Lexico.VIRG;
 Lexico.ID "fat"; Lexico.DOISPONTOS; Lexico.INTEGER; Lexico.PONTOVIRG;
 Lexico.BEGIN; Lexico.PRINT; Lexico.APAR;
 Lexico.LITSTRING "Digite um numero: "; Lexico.FPAR; Lexico.PONTOVIRG;
 Lexico.LEIALN; Lexico.APAR; Lexico.ID "numero"; Lexico.FPAR;
 Lexico.PONTOVIRG; Lexico.ID "fat"; Lexico.ATRIB; Lexico.ID "fatorial";
 Lexico.APAR; Lexico.ID "numero"; Lexico.FPAR; Lexico.PONTOVIRG;
 Lexico.PRINTLN; Lexico.APAR; Lexico.LITSTRING "O fatorial de "; Lexico.
  VIRG;
 Lexico.ID "numero"; Lexico.VIRG; Lexico.LITSTRING "eh"; Lexico.VIRG;
 Lexico.ID "fat"; Lexico.FPAR; Lexico.PONTOVIRG; Lexico.ENDPROG; Lexico.
  EOF]

```

Um exemplo de detecção de erro no código fonte encontrado pelo analisador léxico pode ser visto aplicando-o no código a seguir:

Listagem 3.4: Erro de Caractere

```

1
2 @
3 program micro05;
4 var
5   nome : string;
6   sexo : char;
7   x, h, m : integer;
8 begin
9
10  for x := 1 to 5 do
11  begin
12    write('Digite o nome: ');
13    readln(nome);
14    write('H - Homem ou M - Mulher: ');
15    readln(sexo);
16    case sexo of
17      'H' : h := h+1;
18      'M' : m := m+1;
19    end;
20  end;
21  writeln('Foram inseridos ', h, ' Homens');
22  writeln('Foram inseridas ', m, ' Mulheres');
23 end.

```

Ao executar o analisador léxico nesse código, obtemos a saída:

```
Exception: Failure "9-0: caracter desconhecido @".
```

3.2 Análise Sintática

Após concluída a análise léxica e com os tokens classificados, é necessário dar a "ordem" do que irá ocorrer e verificar se tudo foi utilizado da forma correto, segundo a gramática definida, no caso para este trabalho, a gramática da linguagem Mini Pascal, essa tarefa é dada ao analisador sintático.

3.2.1 Analisador

Responsável por consumir os tokens gerados pelo analisado léxico e reconhecer as "frases" produzidas pela linguagem, junto a esse reconhecimento, o analisador sintático gera a árvore sintática abstrata que dá a ideia do que será executado primeiro no programa. Para facilitar, implementaremos um parser preditivo (algoritmo responsável pela leitura da gramática), utilizando a ferramenta Menhir implementada para a linguagem Ocaml.

3.2.2 Códigos

Implementação do analisador(parser LR(1)):

Listagem 3.5: Parser LR(1)

```

1
2 %{
3 open Ast
4
5 %}
6
7 %token <int> LITINT
8 %token <float> LITREAL
9 %token <string> ID
10 %token <string> LITSTRING
11 %token PROGRAM
12 %token BEGIN
13 %token END
14 %token ENDPROG
15 %token VAR
16 %token VIRG DOISPONTOS PONTO PONTOVIRG
17 %token APAR FPAR
18 %token INTEGER STRING CHAR REAL
19 %token IF THEN ELSE
20 %token FOR WHILE DO TO
21 %token LEIA LEIALN
22 %token PRINT PRINTLN
23 %token ATRIB
24 %token MAIS
25 %token MENOS
26 %token MULTIPLICA
27 %token DIVIDE
28 %token MODULO

```

3.2

```
29 %token MENOR
30 %token IGUAL
31 %token MENORIGUAL
32 %token MAIORIGUAL
33 %token DIFERENTE
34 %token MAIOR
35 %token E
36 %token OU
37 %token CASE
38 %token OF
39 %token FUNCAO
40 %token PROCEDURE
41 %token EOF
42 %token BOOLEAN
43
44 %left OU
45 %left E
46 %left IGUAL DIFERENTE MAIOR MENOR MAIORIGUAL MENORIGUAL
47 %left MAIS MENOS
48 %left MULTIPLICA DIVIDE MODULO
49
50 %start <Ast.programa> programa
51
52 %%
53
54 programa: PROGRAM id=ID PONTOVIRG
55     fs = declaracao_funcao*
56         ds = declaracao*
57         BEGIN
58             cs = comando*
59         ENDPROG
60         EOF { Programa (id,fs, List.flatten ds, cs) }
61
62 declaracao: |VAR ids = separated_nonempty_list(VIRG, ID) DOISPONTOS t =
63     tipo PONTOVIRG {List.map (fun id -> DecVar (id,t)) ids }
64
65 |ids = separated_nonempty_list(VIRG, ID) DOISPONTOS t = tipo
66     PONTOVIRG {List.map (fun id -> DecVar (id,t)) ids }
67
68 (* Declaração de variáveis ok!*)
69
70 tipo: t=tipo_simples { t }
71
72 tipo_simples: |INTEGER { TipoInt }
73 | REAL { TipoReal }
74 | STRING { TipoString }
75 | CHAR { TipoChar }
76 | BOOLEAN {TipoBool}
77
78 (* TIPOS ok!*)
79
80 parametros: ids = separated_list(VIRG, ID) DOISPONTOS t=tipo_simples {
81     List.map (fun id -> Parametros (id,t)) ids }
82
83 declaracao_funcao: FUNCAO id = ID APAR p=parametros FPAR DOISPONTOS tp =
84     tipo_simples PONTOVIRG
85     bv = declaracao*
86     BEGIN
87     lc = comando*
```

```

84         END PONTOVIRG {Funcao(id, p, tp,List.flatten bv, lc) }
85     (*|PROCEDURE id = ID APAR p = parametros FPAR tp= PONTOVIRG
86     bv = declaracao*
87     BEGIN
88     c = comando*
89     END PONTOVIRG {Funcao (id,p,tp,bv,c)}*)
90
91 comando: c=comando_atribuicao { c } (*ok*)
92     | c=comando_se { c } (*ok*)
93     | c=comando_entrada { c } (*ok*)
94     | c=comando_saida { c } (*ok*)
95     | c=comando_for { c } (*ok*)
96     | c=comando_while { c } (*ok*)
97     | c=comando_case { c } (*ok*)
98     | c = comando_funcao {c }
99
100 comando_atribuicao: v=variavel ATRIB e=expressao PONTOVIRG {CmdAtrib (v,e)
101     }
102 (* Atribuição ok!*)
103
104 comando_funcao: |id = ID APAR p=option(arg=separated_nonempty_list(VIRG,
105     expressao) {arg}) FPAR {CmdChamadaFuncao (id, p)}
106
107 comando_se:IF teste=expressao THEN BEGIN entao = comando* END
108     senao=option(ELSE BEGIN cs=comando* END PONTOVIRG {cs} )
109     {CmdSe (teste, entao, senao)}
110 (* IFS OK *)
111
112 comando_entrada: LEIA xs=expressao PONTOVIRG {CmdEntrada xs}
113     |LEIALN xs=expressao PONTOVIRG {CmdEntradaLn xs}
114 (* Leitura ok!*)
115
116 comando_saida: PRINT APAR xs=separated_nonempty_list(VIRG, expressao) FPAR
117     PONTOVIRG { CmdSaida xs}
118     |PRINTLN APAR cs=separated_nonempty_list(VIRG, expressao) FPAR
119     PONTOVIRG { CmdSaidaLn cs}
120 (* Saída ok!*)
121
122 comando_for: FOR v=variavel ATRIB ex=expressao TO e=expressao DO BEGIN c=
123     comando* END PONTOVIRG { CmdFor(v,ex,e,c) }
124
125 comando_while: WHILE teste=expressao DO BEGIN c=comando* END PONTOVIRG {
126     CmdWhile(teste,c)}
127
128 comando_case: CASE v=variavel OF c = cases+ default=option(ELSE cs=comando
129     {cs}) END PONTOVIRG {CmdCase(v,c,default)} (*Shift-reduce a corrigir
130     *)
131
132 cases: e = expressao DOISPONTOS c = comando {Case(e,c)}
133
134 expressao:
135     | v=variavel { ExpVar v }
136     | i=LITINT { ExpInt i }
137     | s=LITSTRING { ExpString s }
138     | r=LITREAL { ExpReal r }
139     | c = comando_funcao {ExpChamadaF c}
140     | e1=expressao op=oper e2=expressao { ExpOp (op, e1, e2) }

```


3.2

```
135     | APAR e=expressao FPAR { Expar(e) }
136
137 (*chamada_func:
138     | f = comando_funcao {ExprChamadaFuncao } *)
139
140 %inline oper:
141     | MAIS { Mais }
142     | MENOS { Menos }
143     | MULTIPLICA { Mult }
144     | DIVIDE { Div }
145     | MODULO { Mod }
146     | MENOR { Menor }
147     | IGUAL { Igual }
148     | MENORIGUAL { MenorIgual }
149     | MAIORIGUAL { MaiorIgual }
150     | DIFERENTE { Difer }
151     | MAIOR { Maior }
152     | E { And }
153     | OU { Or }
154
155 variavel:
156     | x=ID { VarSimples x }
157     | v=variavel PONTO x=ID { VarCampo (v,x) }
```

Algumas modificações tiveram de ser feitas no analisador léxico para que o mesmo enviasse mensagens de erro da forma correta (para tratamento posterior dos erros) e para que o arquivo sintatico.mly ficasse responsável pela definição inicial dos tokens:

Listagem 3.6: 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   (* let msg_erro lexbuf c =
16     let pos = lexbuf.lex_curr_p in
17     let lin = pos.pos_lnum
18     and col = pos.pos_cnum - pos.pos_bol - 1 in
19     sprintf "%d-%d: caracter desconhecido %c" lin col c
20
21   let erro lin col msg =
22     let mensagem = sprintf "%d-%d: %s" lin col msg in
23     failwith mensagem
24
25   let erroComentario lin col msg =
26     let mensagem = sprintf "%d-%d: %s" lin col msg in
27     failwith mensagem
28 *)
```

```

29 }
30 let digito = ['0' - '9']
31 let inteiro = digito+
32 let real = digito* '.' digito+
33
34 let letra = ['a' - 'z' 'A' - 'Z']
35 let identificador = letra ( letra | digito | '_' ) *
36
37 let brancos = [' ' '\t'] +
38 let novalinha = '\r' | '\n' | "\r\n" | "\\n"
39
40 let comentario = "//" [ ^ '\r' '\n' ] *
41
42 rule token = parse
43   brancos      { token lexbuf }
44 | novalinha    { incr_num_linha lexbuf; token lexbuf }
45
46 | comentario { token lexbuf }
47 | "(" {
48   let pos = lexbuf.lex_curr_p in
49   let lin = pos.pos_lnum
50   and col = pos.pos_cnum - pos.pos_bol - 1 in
51   comentario_bloco lin col 0 lexbuf }
52 | "{" {
53   let pos = lexbuf.lex_curr_p in
54   let lin = pos.pos_lnum
55   and col = pos.pos_cnum - pos.pos_bol - 1 in
56   comentario_bloco_dif lin col 0 lexbuf }
57 | '(' { APAR }
58 | ')' { FPAR }
59 | "program" { PROGRAM }
60 | "var" { VAR }
61 | ',' { VIRG }
62 | ';' { PONTOVIRG }
63 | '.' { PONTO }
64 | ':' { DOISPONTOS }
65 | "real" { REAL }
66 | "boolean" { BOOLEAN }
67 | "of" { OF }
68 | "integer" { INTEGER }
69 | "string" { STRING }
70 | "char" { CHAR }
71 | "begin" { BEGIN }
72 | "end" { END }
73 | "end." { ENDPROG }
74 | "write" { PRINT }
75 | "writeln" { PRINTLN }
76 | "read" { LEIA }
77 | "readln" { LEIALN }
78 | '+' { MAIS }
79 | '-' { MENOS }
80 | '/' { DIVIDE }
81 | '*' { MULTIPLICA }
82 | "div" { MODULO }
83 | '>' { MAIOR }
84 | '<' { MENOR }
85 | '=' { IGUAL }
86 | ">=" { MAIORIGUAL }
87 | "<=" { MENORIGUAL }

```

```

88 | "<>"          { DIFERENTE }
89 | "[:="         { ATRIB }
90 | "or"          { OU }
91 | "and"         { E }
92 | "function"    { FUNCAO }
93 | "procedure"   { PROCEDURE }
94 | "if"          { IF }
95 | "then"        { THEN }
96 | "else"        { ELSE }
97 | "while"       { WHILE }
98 | "for"         { FOR }
99 | "to"          { TO }
100 | "do"          { DO }
101 | "case"        { CASE }
102 | "of"          { OF }
103 | inteiro as num { let numero = int_of_string num in
104 |                 LITINT numero }
105 | real as r {let r = float_of_string r in LITREAL r}
106 | identificador as id { ID id }
107 | '\\'        { let pos = lexbuf.lex_curr_p in
108 |                 let lin = pos.pos_lnum
109 |                 and col = pos.pos_cnum - pos.pos_bol - 1 in
110 |                 let buffer = Buffer.create 1 in
111 |                 let str = leia_string lin col buffer lexbuf in
112 |                 LITSTRING str }
113 | _ { raise (Erro ("Caracter desconhecido: " ^ Lexing.lexeme lexbuf)) }
114 | eof { EOF }
115
116
117 and comentario_bloco lin col n = parse
118   "*" ) { if n=0 then token lexbuf
119         else comentario_bloco lin col (n-1) lexbuf }
120 | "("   { comentario_bloco lin col (n+1) lexbuf }
121 | novalinha { incr_num_linha lexbuf; comentario_bloco lin col n lexbuf }
122 | _       { comentario_bloco lin col n lexbuf }
123 | eof     { raise (Erro "Comentário não terminado") }
124
125 and comentario_bloco_dif lin col n = parse
126   "}"   { if n=0 then token lexbuf
127         else comentario_bloco_dif lin col (n-1) lexbuf }
128 | "{"   { comentario_bloco_dif lin col (n+1) lexbuf }
129 | novalinha { incr_num_linha lexbuf; comentario_bloco_dif lin col n
130 |           lexbuf }
131 | _       { comentario_bloco_dif lin col n lexbuf }
132 | eof     { raise (Erro "Comentário não terminado") }
133
134 and leia_string lin col buffer = parse
135   '\\' { Buffer.contents buffer}
136 | "\\t" { Buffer.add_char buffer '\t'; leia_string lin col buffer
137 |       lexbuf }
138 | "\\n" { Buffer.add_char buffer '\n'; leia_string lin col buffer
139 |       lexbuf }
140 | '\\\'' { Buffer.add_char buffer '\''; leia_string lin col buffer
141 |       lexbuf }
142 | '\\\'' { Buffer.add_char buffer '\\\''; leia_string lin col buffer
143 |       lexbuf }
144 | _ as c { Buffer.add_char buffer c; leia_string lin col buffer lexbuf
145 |       }
146 | eof { raise (Erro "A string não foi fechada")}

```

3.2.3 Árvore Sintática Abstrata

O resultado de uma análise sintática deve ser uma árvore sintática abstrata, que será consumida pelo próximo passo conhecido como análise semântica. A árvore é gerada utilizando o seguinte código:

Listagem 3.7: Gerador Árvore Sintática

```

1  (* The type of the abstract syntax tree (AST). *)
2  type ident = string
3
4  type programa = Programa of ident * funcoes * declaracoes * comandos
5  and declaracoes = declaracao list
6  and comandos = comando list
7  and declaracao = DecVar of ident * tipo
8
9  and mult_param = declaracao_parametros list
10 and declaracao_parametros = parametros list
11 and parametros = Parametros of ident * tipo
12
13 and listarparametros = multi_param list
14 and multi_param = Listaparam of declaracao_parametros
15
16 and funcoes = declaracao_funcao list
17 and declaracao_funcao = Funcao of ident * declaracao_parametros * tipo *
    declaracoes * comandos
18
19 and tipo = TipoInt
20           | TipoReal
21           | TipoString
22           | TipoChar
23           | TipoBool
24
25
26 and campos = campo list
27 and campo = ident * tipo
28
29 and cases = Case of expressao * comando
30
31
32 and comando = CmdAtrib of variavel * expressao
33               | CmdSe of expressao * comandos * comandos option
34               | CmdEntrada of expressao
35               | CmdSaida of expressoes
36               | CmdEntradaln of expressao
37               | CmdSaidaln of expressoes
38               | CmdFor of variavel * expressao * expressao * comandos
39               | CmdWhile of expressao * comandos
40               | CmdCase of variavel * cases list * comando option
41               | CmdChamadaFuncao of ident * expressoes option
42
43 and variaveis = variavel list
44 and variavel = VarSimples of ident
45               | VarCampo of variavel * ident
46
47
48 and expressao = ExpVar of variavel

```

3.2

```
49         | ExpInt of int
50         | ExpString of string
51         | ExpBool of bool
52         | ExpReal of float
53         | ExpOp of oper * expressao * expressao
54     | Expar of expressao
55     | ExpChamadaF of comando
56
57 and expressoes = expressao list
58
59 and oper = Mais
60         | Menos
61         | Mult
62         | Div
63         | Mod
64         | Menor
65         | Igual
66         | Difer
67         | Maior
68         | MaiorIgual
69     | MenorIgual
70         | And
71         | Or
```

Observe que, para cada retorno do arquivo `sintatico.mly`, existe uma interpretação para árvore utilizar.

3.2.4 Tratamento de erros

Um processo final, porém com grande importância é o tratamento de erros, é nessa parte que definimos os erros que serão exibidos ao usuário de nosso compilador quando o código digitado não estiver de acordo com a linguagem. Para geração dos arquivos de erro utilize o comando em terminal:

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

Com o arquivo gerado, editei as linhas escritas: . Com todas as linhas editadas, gere o arquivo de erro final utilizando o comando:

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

O arquivo final de erro gerado em meu trabalho:

Listagem 3.8: Erros

```
1
2 (* This file was auto-generated based on "sintatico.msg". *)
3
4 (* Please note that the function [message] can raise [Not_found]. *)
5
6 let message =
```

```

7  fun s ->
8      match s with
9      | 0 ->
10         "0: Esperava 'program'\n"
11      | 1 ->
12         "1: Esperava nome do programa\n"
13      | 2 ->
14         "2: Esperava ';' \n"
15      | 3 ->
16         "3: Esperava declaracao de variavel ou declaracao de funcao ou um
          'begin'\n"
17      | 21 ->
18         "21: Esperava identificadores para as variaveis\n"
19      | 23 ->
20         "23: Esperava um tipo\n"
21      | 25 ->
22         "25: Esperava ';' \n"
23      | 7 ->
24         "7: Esperava ',' ou ':' \n"
25      | 8 ->
26         "8: Esperava identificador\n"
27      | 28 ->
28         "28: Esperava um tipo\n"
29      | 29 ->
30         "29: Esperava ';' \n"
31      | 155 ->
32         "<155: Esperava declaracao de variaveis ou 'begin'\n"
33      | 4 ->
34         "4: Esperava identificador\n"
35      | 5 ->
36         "5: Esperava '(' \n"
37      | 6 ->
38         "6: Esperava lista de parametros\n"
39      | 158 ->
40         "158: Esperava um tipo\n"
41      | 11 ->
42         "11: Esperava ')' \n"
43      | 12 ->
44         "12: Esperava ':' \n"
45      | 13 ->
46         "13: Esperava um tipo\n"
47      | 19 ->
48         "19: Esperava ';' \n"
49      | 20 ->
50         "20: Esperava por declaracao de variaveis ou 'begin'\n"
51      | 32 ->
52         "32: Esperava comandos ou 'end'\n"
53      | 152 ->
54         "152: Esperava 'end'\n"
55      | 153 ->
56         "153: Esperava ';' \n"
57      | 166 ->
58         "166: Esperava declaracao de funcao ou declaracao de variaveis ou
          'begin'\n"
59      | 33 ->
60         "33: Esperava expressao ou '(' seguido de expressao\n"
61      | 78 ->
62         "78: Esperava operador ou 'DO'\n"
63      | 44 ->

```

3.2

```
64         "44: Esperava expressao ou '('\n"
65 | 45 ->
66         "45: Esperava operador, 'TO', 'DO', ':', 'Then', ')'\n"
67 | 46 ->
68         "46: Esperava expressao\n"
69 | 49 ->
70         "49: Esperava expressao\n"
71 | 51 ->
72         "51: Esperava expressao\n"
73 | 52 ->
74         "52: Esperava operador, 'TO', 'DO', ':', 'Then', ')', ';' \n"
75 | 55 ->
76         "55: Esperava expressao\n"
77 | 56 ->
78         "56: Esperava operador, 'TO', 'DO', ':', 'Then', ')', ';' \n"
79 | 59 ->
80         "59: Esperava expressao ou '('\n"
81 | 60 ->
82         "60: Esperava operador, 'TO', 'DO', ':', 'Then', ')', ';' \n"
83 | 57 ->
84         "57: Esperava expressao ou '('\n"
85 | 58 ->
86         "58: Esperava operador, 'TO', 'DO', ':', 'Then', ')', ';' \n"
87 | 61 ->
88         "61: Esperava expressao ou '('\n"
89 | 62 ->
90         "62: Esperava operador, 'TO', 'DO', ':', 'Then', ')', ';' \n"
91 | 63 ->
92         "63: Esperava expressao ou '('\n"
93 | 64 ->
94         "64: Esperava operador, 'TO', 'DO', ':', 'Then', ')', ';' \n"
95 | 65 ->
96         "65: Esperava expressao ou '('\n"
97 | 66 ->
98         "66: Esperava operador, 'TO', 'DO', ':', 'Then', ')', ';' \n"
99 | 67 ->
100        "67: Esperava expressao ou '('\n"
101 | 68 ->
102        "68: Esperava operador, 'TO', 'DO', ':', 'Then', ')', ';' \n"
103 | 79 ->
104        "79: Esperava 'begin'\n"
105 | 80 ->
106        "80: Esperava comandos ou 'end'\n"
107 | 149 ->
108        "149: Esperava 'end'\n"
109 | 150 ->
110        "150: Esperava ';' \n"
111 | 53 ->
112        "53: Esperava expressao ou '('\n"
113 | 69 ->
114        "69: Esperava expressao ou '('\n"
115 | 70 ->
116        "70: Esperava operador, 'TO', 'DO', ':', 'Then', ')', ';' \n"
117 | 40 ->
118        "40: Esperava operador, 'TO', 'DO', ':', 'Then', ')', ';' \n"
119 | 39 ->
120        "39: Esperava expressao ou '('\n"
121 | 43 ->
122        "43: Esperava expressao\n"
```

```

123 | 162 ->
124 |     "162: Esperava comandos ou 'end.'\n"
125 | 81 ->
126 |     "81: Esperava '('\n"
127 | 82 ->
128 |     "82: Esperava expressao ou '('\n"
129 | 75 ->
130 |     "75: Esperava Operador, ')' ou ', '\n"
131 | 76 ->
132 |     "76: Esperava expressao ou '('\n"
133 | 84 ->
134 |     "84: Esperava ';\n"
135 | 86 ->
136 |     "86: Esperava '('\n"
137 | 87 ->
138 |     "87: Esperava expressao ou '('\n"
139 | 89 ->
140 |     "89: Esperava ';\n"
141 | 91 ->
142 |     "91: Esperava expressao ou '('\n"
143 | 92 ->
144 |     "92: Esperava operador ou ';\n"
145 | 139 ->
146 |     "139: Esperava comandos ou 'end'\n"
147 | 163 ->
148 |     "163: Esperava 'end.'\n"
149 | 94 ->
150 |     "94: Esperava expressao ou '('\n"
151 | 95 ->
152 |     "95: Esperava operador ou ';\n"
153 | 97 ->
154 |     "97: Esperava expressao ou '('\n"
155 | 98 ->
156 |     "98: Esperava operador ou 'then'\n"
157 | 99 ->
158 |     "99: Esperava 'begin'\n"
159 | 100 ->
160 |     "100: Esperava comandos ou 'end'\n"
161 | 141 ->
162 |     "141: Esperava 'end'\n"
163 | 142 ->
164 |     "142: Esperava 'else' ou 'end' \n"
165 | 143 ->
166 |     "143: Esperava 'begin'\n"
167 | 144 ->
168 |     "144: Esperava comandos ou 'end'\n"
169 | 145 ->
170 |     "145: Esperava 'end'\n"
171 | 146 ->
172 |     "146: Esperava ';\n"
173 | 37 ->
174 |     "37: Esperava '(' seguido ou nao de parametros \n"
175 | 115 ->
176 |     "115: Esperava ':=\n"
177 | 41 ->
178 |     "41: Esperava um identificador\n"
179 | 116 ->
180 |     "116: Esperava expressao ou '('\n"
181 | 117 ->

```



```

182         "117: Esperava ';' ou operador\n"
183     | 38 ->
184         "38: Esperava expressao ou ')' \n"
185     | 101 ->
186         "101: Esperava identificador\n"
187     | 103 ->
188         "103: Esperava ':='\n"
189     | 104 ->
190         "104: Esperava expressao \n"
191     | 105 ->
192         "105: Esperava operador ou 'TO'\n"
193     | 106 ->
194         "106: Esperava expressao\n"
195     | 107 ->
196         "107: Esperava operador ou 'DO'\n"
197     | 108 ->
198         "108: Esperava 'begin'\n"
199     | 109 ->
200         "109: Esperava lista de comandos ou 'end'\n"
201     | 136 ->
202         "136: Esperava 'end'\n"
203     | 137 ->
204         "137: Esperava ';\n"
205     | 164 ->
206         "164: Esperava EOF Fim de arquivo\n"
207     | 110 ->
208         "110: Esperava um identificador\n"
209     | 111 ->
210         "111: Esperava 'OF'\n"
211     | 112 ->
212         "112: Esperava cases\n"
213     | 131 ->
214         "131: Esperava ':'\n"
215     | 132 ->
216         "132: Esperava um comando\n"
217     | 134 ->
218         "134: Esperava ':'\n"
219     | 129 ->
220         "129: Esperava ';\n"
221     | 114 ->
222         "114: Esperava ':='\n"
223     | 128 ->
224         "128: Esperava 'end'\n"
225     | _ ->
226         raise Not_found

```

3.2.5 Testes

Para os testes com erro, utilizei os arquivos criados pelo professor em aula, o código e como compilá-lo pode ser encontrado abaixo:

Listagem 3.9: Teste

```

1 open Printf
2 open Lexing

```

```

3
4 open Ast
5 open ErroSint (* nome do módulo 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 autômato 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 não pode acontecer *)
24
25 let estado checkpoint : int =
26   match Lazy.force (pilha checkpoint) with
27   | S.Nil -> (* O parser está 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"
38                                         (Lexing.lexeme_start lexbuf) msg))
39
40 let loop lexbuf resultado =
41   let fornecedor = I.lexer_lexbuf_to_supplier Lexico.token lexbuf in
42   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
43
44
45 let parse_com_erro lexbuf =
46   try
47     Some (loop lexbuf (Sintatico.Incremental.programa lexbuf.lex_curr_p))
48   with
49   | Lexico.Erro msg ->
50     printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
51     None
52   | Erro_Sintatico msg ->
53     printf "Erro sintático na %s %s\n" (posicao lexbuf) msg;
54     None
55
56 let parse s =
57   let lexbuf = Lexing.from_string s in
58   let ast = parse_com_erro lexbuf in
59   ast
60

```

3.3

```
61 let parse_arq nome =
62   let ic = open_in nome in
63   let lexbuf = Lexing.from_channel ic in
64   let result = parse_com_erro lexbuf in
65   let _ = close_in ic in
66   match result with
67   | Some ast -> ast
68   | None -> failwith "A analise sintatica falhou"
69
70
71 (* Para compilar:
72    menhir -v --list-errors sintatico.mly > sintatico.msg
73    menhir -v sintatico.mly --compile-errors sintatico.msg > erroSint.ml
74    ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -
75    package menhirLib sintaticoTest.byte
76 *)
```

Utilizando o código `whiles.pas` descrito neste relatório, realizei o teste do analisador sintático obtendo o seguinte resultado como resposta:

```
- : Ast.programa option =
Some
  (Programa ("whiles", [],
    [DecVar ("x", TipoInt); DecVar ("y", TipoInt); DecVar ("z", TipoInt)],
    [CmdAtrib (VarSimples "x", ExpInt 1); CmdAtrib (VarSimples "y", ExpInt
      2);
    CmdAtrib (VarSimples "z", ExpInt 5);
    CmdWhile
      (Expar (ExpOp (Maior, ExpVar (VarSimples "z"), ExpVar (VarSimples "x"
        )),
      [CmdAtrib (VarSimples "x",
        ExpOp (Mais, ExpVar (VarSimples "x"), ExpVar (VarSimples "y"))];
      CmdSaidaln [ExpVar (VarSimples "x")]])))))
```

Ao remover um ";" qualquer neste programa, obtive o seguinte erro:

```
Erro sintático na linha 12, coluna 2 117 - 84: Esperava ';'
.

Exception: Failure "A analise sintatica falhou".
```

3.3 Tópicos Complementares

3.3.1 Extras Analisador Léxico e Ocaml

Para executar o analisador léxico, tenha instalado o Ocaml em seu computador, para fazer a instalação no Ubuntu utilize:

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

Em seguida gere os arquivos necessários de compilação com os seguintes comandos:

```
ocamllex lexico.mll
```

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

Com os arquivos gerados execute o ocaml em um terminal e então utilize os seguintes comandos:

```
ocaml
#use "arquivocarregador.ml";;
lex "nomearquivo.extensao";;
```

3.3.2 Extras Analisador Sintático e Ocaml

Para configuração do Ocaml para execução correta do Menhir, utilizei os seguintes comandos em um terminal linux:

```
>sudo apt-get install menhir
>sudo apt-get install libmenhir-ocaml-dev
```

Um arquivo .ocamlinit deve ser criado e colocado na mesma pasta onde se encontram os códigos que serão utilizados, este arquivo deve ser dessa forma:

Listagem 3.10: Ocamlinit

```
1 let () =
2   try Topdirs.dir_directory (Sys.getenv "OCAML_TOPLEVEL_PATH")
3   with Not_found -> ()
4 ;;
5
6 #use "topfind";;
7 #require "menhirLib";;
8 #directory "_build";;
9 #load "erroSint.cmo";;
10 #load "sintatico.cmo";;
11 #load "lexico.cmo";;
12 #load "ast.cmo";;
13 #load "sintaticoTest.cmo";;
14 open Ast
15 open SintaticoTest
```

Para compilação dos códigos, os comandos necessários são informados no arquivo acima sintaticoTeste.ml

Para execução do código deve-se abrir o Ocaml no terminal e então utilizar:

```
parse_arq "nomedoarquivo.formato";;
```

3.4 Análise Semântica

O analisador semântico dará significado as instruções, além de ocorrer a validação de diversas regras da linguagem. A análise semântica percorre a árvore sintática relaciona os

identificadores com seus dependentes de acordo com a estrutura hierárquica.

Essa etapa também captura informações sobre o programa fonte para que as fases subsequentes gerar o código objeto, um importante componente da análise semântica é a verificação de tipos, nela o compilador verifica se cada operador recebe os operandos permitidos e especificados na linguagem fonte(johnidm.gitbooks.io). Outra ação realizada na análise semântica é diferenciar os escopos entre global e local, para utilizar corretamente os elementos no código.

3.4.1 Códigos

E composto pelos arquivos `semantico.ml`, responsável por classificar os operadores da linguagem (Aritmético, Lógico ou Relacional), inferir os tipos para cada variável e anotá-los na tabela de símbolos, inferir tipos para expressões e verificar se os operadores estão relacionados a tipos compatíveis. Por fim, nesse arquivo são verificados os comandos da linguagem e se suas regras semânticas são respeitadas, Como por exemplo se em uma condição para um WHILE há uma expressão que resulta em um tipo booleano.

Listagem 3.11: 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 (_, pos) -> pos
9   | ExpInt (_, pos) -> pos
10  | ExpString (_, pos) -> pos
11  | ExpReal (_, pos) -> pos
12  | ExpBool (_, pos) -> pos
13  | ExpOp ((_, pos), _, _) -> pos
14  | ExpChamada ((_, pos), _, _) -> pos
15
16 type classe_op = Aritmetico | Relacional | Logico
17
18 let classifica op =
19   let open A in
20   match op with
21   | Or
22   | And -> Logico
23   | Menor
24   | Maior
25   | Igual
26   | MaiorIgual
27   | MenorIgual
28   | Difer -> Relacional
29   | Mais
30   | Menos
31   | Mult
32   | Mod
33   | Div -> Aritmetico
34
35 let msg_erro_pos pos msg =
36   let open Lexing in
37   let lin = pos.pos_lnum

```

```

38 and col = pos.pos_cnum - pos.pos_bol - 1 in
39 Printf.sprintf "Semantico -> linha %d, coluna %d: %s" lin col msg
40
41 let msg_erro nome msg =
42   let pos = snd nome in
43   msg_erro_pos pos msg
44
45 let nome_tipo t =
46   let open A in
47   match t with
48     | TipoInt -> "inteiro"
49     | TipoString -> "string"
50     | TipoBool -> "bool"
51     | TipoVoid -> "void"
52     | TipoReal -> "real"
53     | TipoChar -> "char"
54
55 let mesmo_tipo pos msg tinf tdec =
56   if tinf <> tdec
57   then
58     let msg = Printf.sprintf msg (nome_tipo tinf) (nome_tipo tdec) in
59     failwith (msg_erro_pos pos msg)
60
61 let rec infere_exp amb exp =
62   match exp with
63     | S.ExpInt n -> (T.ExpInt (fst n, A.TipoInt), A.TipoInt)
64     | S.ExpString s -> (T.ExpString (fst s, A.TipoString), A.TipoString)
65     | S.ExpReal r -> (T.ExpReal (fst r, A.TipoReal), A.TipoReal)
66     | S.ExpBool b -> (T.ExpBool (fst b, A.TipoBool), A.TipoBool)
67     | S.ExpVar v ->
68       let id = fst v in
69       (try (match (Amb.busca amb id) with
70         | Amb.EntVar tipo -> (T.ExpVar (id, tipo), tipo)
71         | Amb.EntFun _ ->
72           let msg = "nome de funcao usado como nome de variavel: "
73             ^ id in
74           failwith (msg_erro v msg)
75       )
76       with Not_found ->
77         let msg = "A variavel " ^ id ^ " nao foi declarada" in
78         failwith (msg_erro v msg)
79       (*| _ -> failwith "infere_exp: não implementado"*)
80
81 | S.ExpOp (op, esq, dir) ->
82   let (esq, tesq) = infere_exp amb esq
83   and (dir, tdir) = infere_exp amb dir in
84
85   let verifica_aritmetico () =
86     (match tesq with
87       | A.TipoInt
88       | A.TipoReal ->
89         let _ = mesmo_tipo (snd op)
90           "O operando esquerdo eh do tipo %s mas o direito eh
91             do tipo %s"
92           tesq tdir
93         in tesq (* O tipo da expressão aritmética como um todo *)
94   | t -> let msg = "um operador aritmetico nao pode ser usado com o
95     tipo " ^

```

```

94         (nome_tipo t)
95     in failwith (msg_erro_pos (snd op) msg)
96 )
97
98 and verifica_relacional () =
99     (match tesq with
100     | A.TipoInt
101     | A.TipoReal
102     | A.TipoString ->
103         let _ = mesmo_tipo (snd op)
104             "O operando esquerdo eh do tipo %s mas o direito eh do
              tipo %s"
105             tesq tdir
106     in A.TipoBool (* O tipo da expressão relacional é sempre booleano
                    *)
107
108     | t -> let msg = "um operador relacional nao pode ser usado com o
                    tipo " ^
109                    (nome_tipo t)
110     in failwith (msg_erro_pos (snd op) msg)
111 )
112
113 and verifica_logico () =
114     (match tesq with
115     | A.TipoBool ->
116         let _ = mesmo_tipo (snd op)
117             "O operando esquerdo eh do tipo %s mas o direito eh do
              tipo %s"
118             tesq tdir
119     in A.TipoBool (* O tipo da expressão lógica é sempre booleano *)
120
121     | t -> let msg = "um operador logico nao pode ser usado com o tipo
                    " ^
122                    (nome_tipo t)
123     in failwith (msg_erro_pos (snd op) msg)
124 )
125
126 in
127 let op = fst op in
128 let tinf = (match (classifica op) with
129             | Aritmetico -> verifica_aritmetico ()
130             | Relacional -> verifica_relacional ()
131             | Logico -> verifica_logico ()
132             )
133 in
134     (T.ExpOp ((op,tinf), (esq, tesq), (dir, tdir)), tinf)
135
136 | S.ExpChamada (nome, args) ->
137     let rec verifica_parametros ags ps fs =
138         match (ags, ps, fs) with
139         (a::ags), (p::ps), (f::fs) ->
140             let _ = mesmo_tipo (posicao a)
141                 "O parametro eh do tipo %s mas deveria ser do tipo %s
                  " p f
142             in verifica_parametros ags ps fs
143         | [], [], [] -> ()
144         | _ -> failwith (msg_erro nome "Numero incorreto de parametros")
145     in
146     let id = fst nome in

```

```

147     try
148     begin
149         let open Amb in
150
151         match (Amb.busca amb id) with
152         (* verifica se 'nome' está associada a uma função *)
153         Amb.EntFun {tipo_fn; formais} ->
154             (* Infere o tipo de cada um dos argumentos *)
155             let argst = List.map (infere_exp amb) args
156             (* Obtem o tipo de cada parâmetro formal *)
157             and tipos_formais = List.map snd formais in
158             (* Verifica se o tipo de cada argumento confere com o tipo
159                declarado *)
160             (* do parâmetro formal correspondente. *)
161             let _ = verifica_parametros args (List.map snd argst)
162                 tipos_formais
163             in (T.ExpChamada (id, (List.map fst argst), tipo_fn), tipo_fn)
164 | Amb.EntVar _ -> (* Se estiver associada a uma variável, falhe *)
165             let msg = id ^ " eh uma variavel e nao uma funcao" in
166             failwith (msg_erro nome msg)
167     end
168 with Not_found ->
169     let msg = "Nao existe a funcao de nome " ^ id in
170     failwith (msg_erro nome msg)
171
172 let rec verifica_cmd amb tiporet cmd =
173     let open A in
174     match cmd with
175     CmdRetorno exp ->
176         (match exp with
177         (* Se a função não retornar nada, verifica se ela foi declarada como
178            void *)
179         None ->
180             let _ = mesmo_tipo (Lexing.dummy_pos)
181                 "O tipo retornado eh %s mas foi declarado como %s"
182                 TipoVoid tiporet
183             in CmdRetorno None
184         | Some e ->
185             (* Verifica se o tipo inferido para a expressão de retorno confere
186                com o *)
187             (* tipo declarado para a função. *)
188             let (e1,tinf) = infere_exp amb e in
189             let _ = mesmo_tipo (posicao e)
190                 "O tipo retornado eh %s mas foi declarado
191                 como %s"
192                 tinf tiporet
193             in CmdRetorno (Some e1)
194         )
195     | CmdSe (teste, entao, senao) ->
196         let (testel,tinf) = infere_exp amb teste in
197         (* O tipo inferido para a expressão 'teste' do condicional deve ser
198            booleano *)
199         let _ = mesmo_tipo (posicao teste)
200             "O teste do if deveria ser do tipo %s e nao %s"
201             TipoBool tinf in
202         (* Verifica a validade de cada comando do bloco 'então' *)

```



```

197 let entao1 = List.map (verifica_cmd amb tiporet) entao in
198 (* Verifica a validade de cada comando do bloco 'senão', se houver *)
199 let senao1 =
200     match senao with
201     | None -> None
202     | Some bloco -> Some (List.map (verifica_cmd amb tiporet) bloco)
203 in
204 CmdSe (testel, entao1, senao1)
205
206 | CmdWhile (teste, cmd) ->
207 let (testel,tinf) = infere_exp amb teste in
208 (* O tipo inferido para a expressão 'teste' do condicional deve ser
209     booleano *)
210 let _ = mesmo_tipo (posicao teste)
211     "O teste do while deveria ser do tipo %s e nao %s"
212     TipoBool tinf in
213 (* Verifica a validade de cada comando do bloco 'então' *)
214 let cmd1 = (List.map(verifica_cmd amb tiporet) cmd) in
215 CmdWhile(testel, cmd1)
216
217 | CmdFor (cmdAtrib, num, cmd) ->
218 let cmdAtrib1 = verifica_cmd amb tiporet cmdAtrib in
219 let (num1,tinf) = infere_exp amb num in
220 (* O tipo inferido para a expressão 'teste' do condicional deve ser
221     booleano *)
222 let _ = mesmo_tipo (posicao num)
223     "O teste do for deveria ser do tipo %s e nao %s"
224     TipoInt tinf in
225 (* Verifica a validade de cada comando do bloco 'então' *)
226 let cmd1 = verifica_cmd amb tiporet cmd in
227 CmdFor(cmdAtrib1, num1, cmd1)
228
229 (* | CmdAtrib (elem, exp) ->
230     (* Infere o tipo da expressão no lado direito da atribuição *)
231     let (exp, tdir) = infere_exp amb exp
232     (* Faz o mesmo para o lado esquerdo *)
233     and (elem1, tesq) = infere_exp amb elem in
234     (* Os dois tipos devem ser iguais *)
235     let _ = mesmo_tipo (posicao elem)
236         "Atribuicao com tipos diferentes: %s = %s" tesq
237         tdir
238     in CmdAtrib (elem1, exp)
239
240 | CmdChamada exp ->
241     let (exp,tinf) = infere_exp amb exp in
242     CmdChamada exp
243 *)
244 | CmdAtrib (elem, exp) -> let open Amb in
245     let (exp2, tdir) = infere_exp amb exp in
246     (match elem with
247     | S.ExpVar v ->
248         ( try
249             (match (Amb.busca amb (fst v)) with
250             | Amb.EntVar tipo -> let (elem_tip, telem) =
251                 infere_exp amb elem
252                 and (exp_tip, ttip) = infere_exp amb exp
253                 in

```

```

251         let _ = mesmo_tipo (posicao elem) "
                Atribuicao com tipos diferentes: %s = %
                s"
252         telem ttip
253         in CmdAtrib (elem_tip, exp_tip)
254
255     | Amb.EntFun { tipo_fn; _} -> ( match tipo_fn
                with
256         TipoVoid -> let _ = mesmo_tipo (
                posicao elem) "Funcao do tipo %
                s nao pode receber o valor do
                tipo %s"
257         TipoVoid tdir in
258             CmdRetorno (None)
259         | tipo -> let _ = mesmo_tipo (
                posicao elem) "Funcao do tipo %s
                nao pode receber o valor do tipo
                %s"
260             tipo tdir in
261                 CmdRetorno (Some exp2)
262
263         )
264     )
265     with Not_found -> failwith ("A variável " ^ fst v ^ "
                não foi declarada")
266
267     | _ -> failwith ""
268 )
269
270 | CmdChamada exp ->
271     let (exp,tinf) = infere_exp amb exp in
272     CmdChamada exp
273
274 | CmdEntrada exps ->
275     (* Verifica o tipo de cada argumento da função 'entrada' *)
276     let exps = List.map (infere_exp amb) exps in
277     CmdEntrada (List.map fst exps)
278
279 | CmdSaidaln exps ->
280     (* Verifica o tipo de cada argumento da função 'saida' *)
281     let exps = List.map (infere_exp amb) exps in
282     CmdSaidaln (List.map fst exps)
283
284 | CmdEntradaln exps ->
285     (* Verifica o tipo de cada argumento da função 'entrada' *)
286     let exps = List.map (infere_exp amb) exps in
287     CmdEntradaln (List.map fst exps)
288
289 | CmdSaida exps ->
290     (* Verifica o tipo de cada argumento da função 'saida' *)
291     let exps = List.map (infere_exp amb) exps in
292     CmdSaida (List.map fst exps)
293
294
295 and verifica_fun amb ast =
296     let open A in
297     match ast with
298     A.Funcao {fn_nome; fn_prms; fn_tiporeturn; fn_locais; fn_cmds} ->
299         (* Estende o ambiente global, adicionando um ambiente local *)

```

```

300 let ambfn = Amb.novo_escopo amb in
301 (* Insere os parâmetros no novo ambiente *)
302 let insere_parametro (v,t) = Amb.insere_param ambfn (fst v) t in
303 let _ = List.iter insere_parametro fn_prms in
304 (* Insere as variáveis locais no novo ambiente *)
305 let insere_local = function
306   (DecVar (v,t)) -> Amb.insere_local ambfn (fst v) t in
307 let _ = List.iter insere_local fn_locais in
308 (* Verifica cada comando presente no corpo da função usando o novo
    ambiente *)
309 let corpo_tipado = List.map (verifica_cmd ambfn fn_tiporeturn) fn_cmds
    in
310   A.Funcao {fn_nome; fn_prms; fn_tiporeturn; fn_locais; fn_cmds =
    corpo_tipado}
311
312
313 let rec verifica_dup xs =
314   match xs with
315   [] -> []
316 | (nome,t)::xs ->
317   let id = fst nome in
318   if (List.for_all (fun (n,t) -> (fst n) <> id) xs)
319   then (id, t) :: verifica_dup xs
320   else let msg = "Parametro duplicado " ^ id in
321     failwith (msg_erro nome msg)
322
323 let insere_declaracao_var amb dec =
324   let open A in
325     match dec with
326     DecVar (nome, tipo) -> Amb.insere_local amb (fst nome) tipo
327
328 let insere_declaracao_fun amb dec =
329   let open A in
330     match dec with
331     Funcao {fn_nome; fn_prms; fn_tiporeturn; fn_locais; fn_cmds} ->
332       (* Verifica se não há parâmetros duplicados *)
333       let formais = verifica_dup fn_prms in
334       let nome1 = fst fn_nome in
335       Amb.insere_fun amb nome1 formais fn_tiporeturn
336
337
338 (* Lista de cabeçalhos das funções pré definidas *)
339 let fn_predefs = let open A in [
340   ("write", [("x", TipoInt); ("y", TipoInt)], TipoVoid);
341   ("read",  [("x", TipoInt); ("y", TipoInt)], TipoVoid);
342   ("writeln", [("x", TipoInt); ("y", TipoInt)], TipoVoid);
343   ("readln",  [("x", TipoInt); ("y", TipoInt)], TipoVoid)
344 ]
345
346 (* insere as funções pré definidas no ambiente global *)
347 let declara_predefinidas amb =
348   List.iter (fun (n,ps,tr) -> Amb.insere_fun amb n ps tr) fn_predefs
349
350 let semantico ast =
351   (* cria ambiente global inicialmente vazio *)
352   let amb_global = Amb.novo_amb [] in
353   let _ = declara_predefinidas amb_global in
354   let (A.Programa (ident,decs_funs, decs_globais,corpo)) = ast in
355   let _ = List.iter (insere_declaracao_var amb_global) decs_globais in

```

```

356 let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
357 (* Verificação de tipos nas funções *)
358 let decs_funs = List.map (verifica_fun amb_global) decs_funs in
359 (* Verificação de tipos na função principal *)
360 let corpo = List.map (verifica_cmd amb_global A.TipoVoid) corpo in
361 (A.Programa (ident,decs_funs, decs_globais,corpo), amb_global)

```

O arquivo Ambiente.ml implementa as operações necessárias para a manipulações dos ambientes, dividindo-os entre global e local.

Listagem 3.12: 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
8 type entrada = EntFun of entrada_fn
9               | EntVar of A.tipo
10
11 type t = {
12   ambv : entrada Tab.tabela
13 }
14
15 let novo_amb xs = { ambv = Tab.cria xs }
16
17 let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }
18
19 let busca amb ch = Tab.busca amb.ambv ch
20
21 let insere_local amb ch t =
22   Tab.insere amb.ambv ch (EntVar t)
23
24 let insere_param amb ch t =
25   Tab.insere amb.ambv ch (EntVar t)
26
27 let insere_fun amb nome params resultado =
28   let ef = EntFun { tipo_fn = resultado;
29                     formais = params }
30   in Tab.insere amb.ambv nome ef

```

Para o analisador semantico, se fez necessário parametrizar as expressões, o resultado final pode ser visto abaixo

Listagem 3.13: ast

```

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

```

3.4

```

9  and 'expr comandos = ('expr comando) list
10 and declaracao = DecVar of (ident pos) * tipo
11
12 and declaracao_parametros = parametros list
13 and parametros = Parametros of (ident pos) * tipo
14
15
16 and 'expr funcoes = ('expr declaracao_funcao) list
17 and 'expr declaracao_funcao = Funcao of ('expr regFunc)
18 and 'expr regFunc = {
19     fn_nome: (ident pos);
20     fn_prms: ((ident pos) * tipo) list;
21     fn_tiporeturn: tipo;
22     fn_locais: declaracoes;
23     fn_cmds: 'expr comandos;
24 }
25
26 and tipo = TipoInt
27           | TipoReal
28           | TipoString
29           | TipoChar
30           | TipoBool
31           | TipoVoid
32
33 and campos = campo list
34 and campo = ident * tipo
35
36 (*and 'expr cases = Case of 'expr * ('expr comando)*)
37
38
39 and 'expr comando = CmdAtrib of 'expr * 'expr
40                   | CmdSe of 'expr * ('expr comandos) * ('expr comandos option)
41                   | CmdFor of 'expr comando * 'expr * 'expr comando
42                   | CmdEntrada of ('expr expressoes)
43                   | CmdSaida of ('expr expressoes)
44                   | CmdEntradaLn of ('expr expressoes)
45                   | CmdSaidaLn of ('expr expressoes)
46                   (*| CmdFor of ('expr variavel) * 'expr * 'expr * ('expr
47                      comandos)*)
48                   | CmdWhile of 'expr * ('expr comandos)
49                   (*| CmdCase of ('expr variavel) * ('expr cases list) * ('expr
50                      comando option)*)
51                   (*| CmdChamadaFuncao of ident * ('expr expressoes option)*)
52                   | CmdChamada of 'expr
53                   | CmdRetorno of 'expr option
54
55 and 'expr variaveis = ('expr variavel) list
56 and 'expr variavel = VarSimples of ident pos
57                   | VarCampo of ('expr variavel) * (ident pos)
58
59 and 'expr expressoes = 'expr list
60
61 and oper = Mais
62           | Menos
63           | Mult
64           | Div
65           | Mod
66           | Menor
67           | Igual

```

```

66         | Difer
67         | Maior
68         | MaiorIgual
69     | MenorIgual
70         | And
71         | Or

```

Com a parametrização das expressões, fiz o arquivo `sast.ml`, onde coloquei as expressões seguidas dos parâmetros de posição.

Listagem 3.14: Sast

```

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

```

Outro arquivo parecido com o `sast.ml` mas com outros parâmetros é o `tast.ml`, a diferença entre eles é que em `sast` as expressões carregam uma posição e no arquivo `tast.ml` carregam um tipo.

Listagem 3.15: Tast

```

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

```

3.4.2 Testes

O código para execução do analisador semântico pode ser visto abaixo:

Listagem 3.16: Semântico Teste

```

1 open Printf
2 open Lexing
3 open Errosint
4 open Ast
5
6 exception Erro_Sintatico of string
7
8 module S = MenhirLib.General (* Streams *)

```

3.4

```

9 module I = Sintatico.MenhirInterpreter
10
11 let posicao lexbuf =
12     let pos = lexbuf.lex_curr_p in
13     let lin = pos.pos_lnum
14     and col = pos.pos_cnum - pos.pos_bol - 1 in
15     sprintf "linha %d, coluna %d" lin col
16
17 (* [pilha checkpoint] extrai a pilha do autômato LR(1) contida em
    checkpoint *)
18
19 let pilha checkpoint =
20     match checkpoint with
21     | I.HandlingError amb -> I.stack amb
22     | _ -> assert false (* Isso não pode acontecer *)
23
24 let estado checkpoint : int =
25     match Lazy.force (pilha checkpoint) with
26     | S.Nil -> (* O parser está no estado inicial *)
27         0
28     | S.Cons (I.Element (s, _, _, _), _) ->
29         I.number s
30
31 let sucesso v = Some v
32
33 let falha lexbuf (checkpoint : (Sast.expressao Ast.programa) I.checkpoint)
34     =
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
38         lexbuf) msg))
39
40 let loop lexbuf resultado =
41     let fornecedor = I.lexer_lexbuf_to_supplier Lexico.token lexbuf in
42     I.loop_handle sucesso (falha lexbuf) fornecedor resultado
43
44 let parse_com_erro lexbuf =
45     try
46         Some (loop lexbuf (Sintatico.Incremental.programa lexbuf.lex_curr_p))
47     with
48     | Lexico.Erro msg ->
49         printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
50         None
51     | Erro_Sintatico msg ->
52         printf "Erro sintático na %s %s\n" (posicao lexbuf) msg;
53         None
54
55 let parse s =
56     let lexbuf = Lexing.from_string s in
57     let ast = parse_com_erro lexbuf in
58     ast
59
60 let parse_arq nome =
61     let ic = open_in nome in
62     let lexbuf = Lexing.from_channel ic in
63     let ast = parse_com_erro lexbuf in
64     let _ = close_in ic in
65     ast

```

```

65
66 let verifica_tipos nome =
67   let ast = parse_arq nome in
68   match ast with
69     Some (Some ast) -> Semantico.semantico ast
70   | _ -> failwith "Nada a fazer!\n"
71
72 (* Para compilar:
73    ocamlbuild -use-menhir sintaticoTest.byte
74 *)

```

Para os testes utilizei o arquivo Fa.pas, o mesmo pode ser encontrado nesse relatório. Ao executar o analisador semântico com esse arquivo, obtive o seguinte resultado:

```

- : Tast.expressao Ast.programa * Ambiente.t =
(Programa ("program micro01", [],
  [DecVar
    (("cel",
      {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 232; pos_cnum =
        233}),
      TipoReal);
    DecVar
      (("far",
        {Lexing.pos_fname = ""; pos_lnum = 11; pos_bol = 232; pos_cnum =
          237}),
        TipoReal)],
  [CmdSaidaln
    [Tast.ExpString ("Tabela de conversao: Celsius -> Fahrenheit",
      TipoString)];
    CmdSaidaln
      [Tast.ExpString ("Digite a temperatura em Celsius: ", TipoString)];
    CmdEntradaln [Tast.ExpVar ("cel", TipoReal)];
    CmdAtrib (Tast.ExpVar ("far", TipoReal),
      Tast.ExpOp ((Div, TipoReal),
        (Tast.ExpOp ((Mais, TipoReal),
          (Tast.ExpOp ((Mult, TipoReal),
            (Tast.ExpReal (9., TipoReal), TipoReal),
            (Tast.ExpVar ("cel", TipoReal), TipoReal)),
            TipoReal),
            (Tast.ExpReal (160., TipoReal), TipoReal)),
            TipoReal),
            (Tast.ExpReal (5., TipoReal), TipoReal))));
    CmdSaidaln
      [Tast.ExpString ("A nova temperatura eh:", TipoString);
        Tast.ExpVar ("far", TipoReal); Tast.ExpString ("F", TipoString)]],
  <abstr>)

```

"Ao introduzir um erro semântico, onde declarei duas vezes a variável "far", obtive o seguinte erro:

```
Exception: Tabsimb.Entrada_existente "far".
```


3.4.3 Extras

O arquivo .ocamlinit será apresentado em Interpretador!

Para executar a análise semântica basta o que vem a seguir no ocaml.

```
verifica_arq "nomearquivo.formato";;
```

3.5 Interpretador

Com um interpretador é possível realizar as operações da linguagem de programação sem realmente gerar código de máquina, no nosso caso, utilizamos o interpretador ocaml para executar os comandos descritos pelos programas.

O interpretador é uma modificação do analisador semântico, onde além de tipos, também se guarde valores associados a variáveis e funções. Isso pode ser visto no arquivo interprete.ml abaixo:

Listagem 3.17: 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 (nome,tipo) -> (nome, tipo)
11  | _ -> failwith "obtem_nome_tipo_var: nao eh variavel"
12
13 let pega_int exp =
14   match exp with
15   | T.ExpInt (i,_) -> i
16   | _ -> failwith "pega_int: nao eh inteiro"
17
18 let pega_string exp =
19   match exp with
20   | T.ExpString (s,_) -> s
21   | _ -> failwith "pega_string: nao eh string"
22
23 let pega_real exp =
24   match exp with
25   | T.ExpReal (s,_) -> s
26   | _ -> failwith "pega_real: nao eh real"
27
28 let pega_bool exp =
29   match exp with
30   | T.ExpBool (b,_) -> b
31   | _ -> failwith "pega_bool: nao eh booleano"
32
33 type classe_op = Aritmetico | Relacional | Logico
34
```

```

35 let classifica op =
36   let open A in
37   match op with
38     Or
39   | And -> Logico
40   | Menor
41   | Maior
42   | Igual
43   | MaiorIgual
44   | MenorIgual
45   | Difer -> Relacional
46   | Mais
47   | Menos
48   | Mult
49   | Div -> Aritmetico
50
51
52 let rec interpreta_exp amb exp =
53   let open A in
54   let open T in
55   match exp with
56   | ExpVoid
57   | ExpInt _
58   | ExpString _
59   | ExpReal _
60   | ExpBool _ -> exp
61   | ExpVar _ ->
62     let (id,tipo) = obtem_nome_tipo_var exp in
63     (* Tenta encontrar o valor da variável no escopo local, se não *)
64     (* encontrar, tenta novamente no escopo que engloba o atual. Prossegue *)
65     (* assim até encontrar o valor em algum escopo englobante ou até *)
66     (* encontrar o escopo global. Se em algum lugar for encontrado, *)
67     (* devolve-se o valor. Em caso contrário, devolve uma exceção *)
68     (match (Amb.busca amb id) with
69      | Amb.EntVar (tipo, v) ->
70        (match v with
71         | None -> failwith ("variável nao inicializada: " ^ id)
72         | Some valor -> valor
73        )
74      | _ -> failwith "interpreta_exp: expvar"
75     )
76   | ExpOp ((op,top), (esq, tesq), (dir,tdir)) ->
77     let vesq = interpreta_exp amb esq
78     and vdir = interpreta_exp amb dir in
79
80     let interpreta_aritmetico () =
81       (match tesq with
82        | TipoInt ->
83          (match op with
84           | Mais -> ExpInt (pega_int vesq + pega_int vdir, top)
85           | Menos -> ExpInt (pega_int vesq - pega_int vdir, top)
86           | Mult -> ExpInt (pega_int vesq * pega_int vdir, top)
87           | Div -> ExpInt (pega_int vesq / pega_int vdir, top)
88          )
89        | Mod -> ExpInt (pega_int vesq mod pega_int vdir, top)
90        | _ -> failwith "interpreta_aritmetico"
91       )
92     | TipoReal ->

```

```

92     (match op with
93     | Mais -> ExpReal (pega_real vesq +. pega_real vdir, top)
94     | Menos -> ExpReal (pega_real vesq -. pega_real vdir, top)
95     | Mult -> ExpReal (pega_real vesq *. pega_real vdir, top)
96     | Div -> ExpReal (pega_real vesq /. pega_real vdir, top)
97     | _ -> failwith "interpreta_aritmetico"
98     )
99 | _ -> failwith "interpreta_aritmetico"
100 )
101
102 and interpreta_relacional () =
103     (match tesq with
104     | TipoInt ->
105         (match op with
106         | Menor -> ExpBool (pega_int vesq < pega_int vdir, top)
107         | Maior -> ExpBool (pega_int vesq > pega_int vdir, top)
108         | Igual -> ExpBool (pega_int vesq == pega_int vdir, top)
109         | Difer -> ExpBool (pega_int vesq != pega_int vdir, top)
110         | MenorIgual -> ExpBool (pega_int vesq <= pega_int vdir, top)
111         | MaiorIgual -> ExpBool (pega_int vesq >= pega_int vdir, top)
112         | _ -> failwith "interpreta_relacional"
113         )
114     | TipoString ->
115         (match op with
116         | Menor -> ExpBool (pega_string vesq < pega_string vdir, top)
117         | Maior -> ExpBool (pega_string vesq > pega_string vdir, top)
118         | Igual -> ExpBool (pega_string vesq == pega_string vdir, top)
119         | Difer -> ExpBool (pega_string vesq != pega_string vdir, top)
120         | MenorIgual -> ExpBool (pega_string vesq <= pega_string vdir,
121                                 top)
122         | MaiorIgual -> ExpBool (pega_string vesq >= pega_string vdir,
123                                 top)
124         | _ -> failwith "interpreta_relacional"
125         )
126     | TipoReal ->
127         (match op with
128         | Menor -> ExpBool (pega_real vesq < pega_real vdir, top)
129         | Maior -> ExpBool (pega_real vesq > pega_real vdir, top)
130         | Igual -> ExpBool (pega_real vesq == pega_real vdir, top)
131         | Difer -> ExpBool (pega_real vesq != pega_real vdir, top)
132         | MenorIgual -> ExpBool (pega_real vesq <= pega_real vdir, top)
133         | MaiorIgual -> ExpBool (pega_real vesq >= pega_real vdir, top)
134         | _ -> failwith "interpreta_relacional"
135         )
136     | TipoBool ->
137         (match op with
138         | Menor -> ExpBool (pega_bool vesq < pega_bool vdir, top)
139         | Maior -> ExpBool (pega_bool vesq > pega_bool vdir, top)
140         | Igual -> ExpBool (pega_bool vesq == pega_bool vdir, top)
141         | Difer -> ExpBool (pega_bool vesq != pega_bool vdir, top)
142         | MenorIgual -> ExpBool (pega_bool vesq <= pega_bool vdir, top)
143         | MaiorIgual -> ExpBool (pega_bool vesq >= pega_bool vdir, top)
144         | _ -> failwith "interpreta_relacional"
145         )
146     | _ -> failwith "interpreta_relacional"
147     )
148
149 and interpreta_logico () =
150     (match tesq with

```

```

149     | TipoBool ->
150       (match op with
151       | Or -> ExpBool (pega_bool vesq || pega_bool vdir, top)
152       | And -> ExpBool (pega_bool vesq && pega_bool vdir, top)
153       | _ -> failwith "interpreta_logico"
154       )
155     | _ -> failwith "interpreta_logico"
156   )
157
158   in
159   let valor = (match (classifica op) with
160     Aritmetico -> interpreta_aritmetico ()
161     | Relacional -> interpreta_relacional ()
162     | Logico -> interpreta_logico ()
163   )
164   in
165   valor
166
167 | ExpChamada (id, args, tipo) ->
168   let open Amb in
169   ( match (Amb.busca amb id) with
170   | Amb.EntFun {tipo_fn; formais; locais; corpo} ->
171     (* Interpreta cada um dos argumentos *)
172     let vargs = List.map (interpreta_exp amb) args in
173     (* Associa os argumentos aos parâmetros formais *)
174     let vformais = List.map2 (fun (n,t) v -> (n, t, Some v))
175       formais vargs
176     in interpreta_fun amb id vformais locais corpo
177   | _ -> failwith "interpreta_exp: expchamada"
178   )
179
180 and interpreta_fun amb fn_nome fn_formais fn_locais fn_corpo =
181   let open A in
182   (* Estende o ambiente global, adicionando um ambiente local *)
183   let ambfn = Amb.novo_escopo amb in
184   let insere_local d =
185     match d with
186     (DecVar (v,t)) -> Amb.insere_local ambfn (fst v) t None
187   in
188   (* Associa os argumentos aos parâmetros e insere no novo ambiente *)
189   let insere_parametro (n,t,v) = Amb.insere_param ambfn n t v in
190   let _ = List.iter insere_parametro fn_formais in
191   (* Insere as variáveis locais no novo ambiente *)
192   let _ = List.iter insere_local fn_locais in
193   (* Interpreta cada comando presente no corpo da função usando o novo
194     ambiente *)
195   try
196     let _ = List.iter (interpreta_cmd ambfn) fn_corpo in T.ExpVoid
197     with
198     Valor_de_retorno expret -> expret
199
200 and interpreta_cmd amb cmd =
201   let open A in
202   let open T in
203   match cmd with
204   CmdRetorno exp ->
205     (* Levantar uma exceção foi necessária pois, pela semântica do comando
206       de
207       retorno, sempre que ele for encontrado em uma função, a computação

```

```

206         deve parar retornando o valor indicado, sem realizar os demais
           comandos.
207     *)
208     (match exp with
209     (* Se a função não retornar nada, então retorne ExpVoid *)
210     None -> raise (Valor_de_retorno ExpVoid)
211     | Some e ->
212     (* Avalia a expressão e retorne o resultado *)
213     let e1 = interpreta_exp amb e in
214     raise (Valor_de_retorno e1)
215     )
216
217 | CmdSe (teste, entao, senao) ->
218 let testel = interpreta_exp amb teste in
219 (match testel with
220 ExpBool (true,_) ->
221 (* Interpreta cada comando do bloco 'então' *)
222 List.iter (interpreta_cmd amb) entao
223 | _ ->
224 (* Interpreta cada comando do bloco 'senão', se houver *)
225 (match senao with
226 None -> ()
227 | Some bloco -> List.iter (interpreta_cmd amb) bloco
228 )
229 )
230
231 | CmdAtrib (elem, exp) ->
232 (* Interpreta o lado direito da atribuição *)
233 let exp = interpreta_exp amb exp
234 (* Faz o mesmo para o lado esquerdo *)
235 and (elem1, tipo) = obtem_nome_tipo_var elem in
236 Amb.atualiza_var amb elem1 tipo (Some exp)
237
238 | CmdFor (cmd1, exp, cmd2) ->
239 (match cmd1 with
240 | CmdAtrib(v, exp1) ->
241 (* Interpreta o lado direito da atribuição *)
242 let exp2 = interpreta_exp amb exp1
243 (* Faz o mesmo para o lado esquerdo *)
244 and (elem1, tipo) = obtem_nome_tipo_var v in
245 Amb.atualiza_var amb elem1 tipo (Some exp2);
246
247 (match exp2 with
248 | ExpInt (val1, _) ->
249 (match (interpreta_exp amb exp) with
250 | ExpInt (val2, _) ->
251 for var = val1 to val2 do
252
253 (*
254
255 (* Faz o mesmo para o lado esquerdo *)
256 and (elemAux, tipoAux) =
257 obtem_nome_tipo_var v in
258
259 Amb.atualiza_var amb elemAux tipoAux (Some
260 (ExpInt (var, )))
261 *)
262 *)
263
264 interpreta_cmd amb cmd2
265 done
266 | _ -> failwith "segunda expressao invalida"

```

```

262         )
263         | _ -> failwith "expressao invalida"
264     )
265     | _ -> failwith "comando invalido"
266 )
267
268 | CmdWhile (exp, cmd) ->
269   (match (interpreta_exp amb exp) with
270   | ExpBool (v, _) ->
271     (let value = ref v in
272      while !value do
273        List.iter (interpreta_cmd amb) cmd;
274        match (interpreta_exp amb exp) with
275        | ExpBool(v, _) -> value := v
276        | _ -> failwith "Condicao nao satisfeita"
277      done
278     )
279   | _ -> failwith "Condicao invalida"
280 )
281
282 | CmdChamada exp -> ignore( interpreta_exp amb exp)
283
284 | CmdEntrada exps ->
285   (* Obtem os nomes e os tipos de cada um dos argumentos *)
286   let nts = List.map (obtem_nome_tipo_var) exps in
287   let leia_var (nome,tipo) =
288     let valor =
289       (match tipo with
290       | A.TipoInt -> T.ExpInt (read_int (), tipo)
291       | A.TipoReal -> T.ExpReal (read_float (), tipo)
292       | A.TipoString -> T.ExpString (read_line (), tipo)
293       | _ -> failwith "leia_var: nao implementado"
294       )
295     in Amb.atualiza_var amb nome tipo (Some valor)
296   in
297   (* Lê o valor para cada argumento e atualiza o ambiente *)
298   List.iter leia_var nts
299
300 | CmdEntradaLn exps ->
301   (* Obtem os nomes e os tipos de cada um dos argumentos *)
302   let nts = List.map (obtem_nome_tipo_var) exps in
303   let leia_var (nome,tipo) =
304     let valor =
305       (match tipo with
306       | A.TipoInt -> T.ExpInt (read_int (), tipo)
307       | A.TipoReal -> T.ExpReal (read_float (), tipo)
308       | A.TipoString -> T.ExpString (read_line (), tipo)
309       | _ -> failwith "leia_var: nao implementado"
310       )
311     in Amb.atualiza_var amb nome tipo (Some valor)
312   in
313   (* Lê o valor para cada argumento e atualiza o ambiente *)
314   List.iter leia_var nts
315
316 | CmdSaida exps ->
317   (* Interpreta cada argumento da função 'saida' *)
318   let exps = List.map (interpreta_exp amb) exps in
319   let imprima exp =
320     (match exp with

```

```

321 | T.ExpInt (n,_) -> let _ = print_int n in print_string " "
322 | T.ExpString (s,_) -> let _ = print_string s in print_string " "
323 | T.ExpReal (r, _) -> let _ = print_float r in print_string " "
324 | T.ExpBool (b,_) ->
325   let _ = print_string (if b then "true" else "false")
326   in print_string " "
327 | _ -> failwith "imprima: nao implementado"
328 )
329 in
330 List.iter imprima exps
331
332 | CmdSaidaln exps ->
333   (* Interpreta cada argumento da função 'saida' *)
334   let exps = List.map (interpreta_exp amb) exps in
335   let imprima exp =
336     (match exp with
337     | T.ExpInt (n,_) -> let _ = print_int n in print_string " "
338     | T.ExpString (s,_) -> let _ = print_string s in print_string " "
339     | T.ExpReal (r, _) -> let _ = print_float r in print_string " "
340     | T.ExpBool (b,_) ->
341       let _ = print_string (if b then "true" else "false")
342       in print_string " "
343     | _ -> failwith "imprima: nao implementado"
344     )
345   in
346   let _ = List.iter imprima exps in
347   print_newline ()
348
349 let insere_declaracao_var amb dec =
350   match dec with
351   | A.DecVar (nome, tipo) -> Amb.insere_local amb (fst nome) tipo
352   | None
353
354 let insere_declaracao_fun amb dec =
355   let open A in
356   match dec with
357   | Funcao {fn_nome; fn_prms; fn_tiporeturn; fn_locais; fn_cmds} ->
358     let nome = fst fn_nome in
359     let formais = List.map (fun (n,t) -> ((fst n), t)) fn_prms in
360     Amb.insere_fun amb nome formais fn_locais fn_tiporeturn fn_cmds
361
362 (* Lista de cabeçalhos das funções pré definidas *)
363 let fn_predefs = let open A in [
364   ("read", [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
365   ("write", [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
366   ("readln", [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
367   ("writeln", [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
368 ]
369
370 (* insere as funções pré definidas no ambiente global *)
371 let declara_predefinidas amb =
372   List.iter (fun (n,ps,tr,c) -> Amb.insere_fun amb n ps [] tr c)
373   fn_predefs
374
375 let interprete ast =
376   (* cria ambiente global inicialmente vazio *)
377   let amb_global = Amb.novo_amb [] in
378   let _ = declara_predefinidas amb_global in
379   let (A.Programa (ident,decs_funs,decs_globais,corpo)) = ast in

```

```

378 let _ = List.iter (insere_declaracao_var amb_global) decs_globais in
379 let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
380 (* Interpreta a função principal *)
381 let resultado = List.iter (interpreta_cmd amb_global) corpo in
382 resultado

```

Também foram feitas modificações no arquivo ambiente, gerando o arquivo ambienteInterp, que agora gerencia os ambientes e mantém os valores das variáveis nos locais corretos.

Listagem 3.18: Ambiente Interpretador

```

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

```

Com essas simples modificações já é possível testar o interpretador.

3.5.1 Testes

O código para execução do interpretador pode ser visto abaixo:

Listagem 3.19: Interpretador Teste

```

1 open Printf
2 open Lexing
3 open Errosint
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
13 let posicao lexbuf =
14     let pos = lexbuf.lex_curr_p in
15     let lin = pos.pos_lnum
16     and col = pos.pos_cnum - pos.pos_bol - 1 in
17     sprintf "linha %d, coluna %d" lin col
18
19 (* [pilha checkpoint] extrai a pilha do autômato LR(1) contida em
    checkpoint *)
20
21 let pilha checkpoint =
22     match checkpoint with
23     | I.HandlingError amb -> I.stack amb
24     | _ -> assert false (* Isso não pode acontecer *)
25
26 let estado checkpoint : int =
27     match Lazy.force (pilha checkpoint) with
28     | S.Nil -> (* O parser está no estado inicial *)
29         0
30     | S.Cons (I.Element (s, _, _, _), _) ->
31         I.number s
32
33 let sucesso v = Some v
34
35 let falha lexbuf (checkpoint : (Sast.expressao Ast.programa) I.checkpoint)
36     =
37     let estado_atual = estado checkpoint in
38     let msg = message estado_atual in
39     raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
40         (Lexing.lexeme_start lexbuf) msg))
41
42 let loop lexbuf resultado =
43     let fornecedor = I.lexer_lexbuf_to_supplier Lexico.token lexbuf in
44     I.loop_handle sucesso (falha lexbuf) fornecedor resultado
45
46 let parse_com_erro lexbuf =
47     try
48         Some (loop lexbuf (Sintatico.Incremental.programa lexbuf.lex_curr_p))
49     with
50     | Lexico.Erro msg ->
51         printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
52         None
53     | Erro_Sintatico msg ->
54         printf "Erro sintático na %s %s\n" (posicao lexbuf) msg;
55         None
56

```

```

57 let parse s =
58   let lexbuf = Lexing.from_string s in
59   let ast = parse_com_erro lexbuf in
60   ast
61
62 let parse_arq nome =
63   let ic = open_in nome in
64   let lexbuf = Lexing.from_channel ic in
65   let ast = parse_com_erro lexbuf in
66   let _ = close_in ic in
67   ast
68
69 let verifica_tipos nome =
70   let ast = parse_arq nome in
71   match ast with
72   | Some (Some ast) -> semantico ast
73   | _ -> failwith "Nada a fazer!\n"
74
75
76 let interprete nome =
77   let tast,amb = verifica_tipos nome in
78   Interprete.interprete tast

```

Ao executar o arquivo `leiasexo.pas` que pode ser encontrado nesse relatório, obtive a seguinte tela de execução:

```

guilherme@ubuntu: ~/Desktop/CC-Pascal/semantico
Findlib has been successfully loaded. Additional directives:
#require "package";;      to load a package
#list;;                   to list the available packages
#camlp4o;;                to load camlp4 (standard syntax)
#camlp4r;;                to load camlp4 (revised syntax)
#predicates "p,q,...";;   to set these predicates
Topfind.reset();;         to force that packages will be reloaded
#thread;;                 to enable threads

/usr/lib/ocaml/menhirLib: added to search path
/usr/lib/ocaml/menhirLib/menhirLib.cmo: loaded
# interprete "leiasexo.pas";;
Digite o nome: Jose
1 - Homem ou 2 - Mulher: 1
Digite o nome: maria
1 - Homem ou 2 - Mulher: 2
Digite o nome: joao
1 - Homem ou 2 - Mulher: 1
Digite o nome: mario
1 - Homem ou 2 - Mulher: 1
Foram inseridos 3 Homens
Foram inseridas 1 Mulheres
- : unit = ()
#

```

Os er-

ros são praticamente os mesmos da análise semântica.

3.5.2 Extras

O arquivo `.ocamlinit` final ficou da seguinte forma:

Listagem 3.20: `.ocamlinit`

```

1 let () =

```

```

2  try Topdirs.dir_directory (Sys.getenv "OCAML_TOPLEVEL_PATH")
3  with Not_found -> ()
4  ;;
5
6  #use "topfind";;
7  #require "menhirLib";;
8  #directory "_build";;
9  #load "sintatico.cmo";;
10 #load "lexico.cmo";;
11 #load "ast.cmo";;
12 #load "sast.cmo";;
13 #load "tast.cmo";;
14 #load "tabsimb.cmo";;
15 #load "ambiente.cmo";;
16 #load "semantico.cmo";;
17 #load "errosint.cmo";;
18 #load "ambInterp.cmo";;
19 #load "interprete.cmo";;
20 #load "interpreteTeste.cmo";;
21
22 open Ast
23 open AmbInterp
24 open InterpreteTeste

```

Para executar o interpretador, utilize o seguinte comando em um terminal do ocaml:

```
interprete "nomearquivo.formato";;
```

3.6 Conclusão

Ao desenvolver o analisador léxico, o analisador sintático, o analisador semântico e um interpretador em Ocaml, pude compreender melhor como funciona os processos para construção de um compilador que gera código de máquina a partir de uma linguagem de programação. Este novo conhecimento é crucial para qualquer programador, com esse tipo de experiência cria-se uma nova visão de como programar, de como compreender melhor os erros dos compiladores e até mesmo uma nova forma de pensar.

Capítulo 4

Referências

[Baksmali-Smali Wiki](#)
[Dalvik Wiki](#)
[Formato dex](#)
[Instruções](#)
[Análise semântica](#)
Slides do professor
Slides de semestres anteriores.