

Construção de um compilador de Python para Dalvik usando Objective Caml

Miguel Henrique de Brito Pereira

miguelhbrito@gmail.com

Faculdade de Computação
Universidade Federal de Uberlândia

18 de maio de 2019

Lista de Figuras

2.1	ADV Manager	12
2.2	Criar um novo device	13
2.3	Modelo Device	13
2.4	Manager Devices	14

Lista de Tabelas

4.1 Tabela de Tokens 29

Lista de Listagens

3.1	Nano01.py	15
3.2	Nano01.java	15
3.3	Nano01.smali	15
3.4	Nano02.py	16
3.5	Nano02.java	16
3.6	Nano02.smali	16
3.7	Nano03.py	16
3.8	Nano03.java	17
3.9	Nano03.smali	17
3.10	Nano04.py	17
3.11	Nano04.java	17
3.12	Nano04.smali	18
3.13	Nano05.py	18
3.14	Nano05.java	18
3.15	Nano05.smali	18
3.16	Nano06.py	19
3.17	Nano06.java	19
3.18	Nano06.smali	19
3.19	Nano07.py	20
3.20	Nano07.java	20
3.21	Nano07.smali	20
3.22	Nano08.py	21
3.23	Nano08.java	21
3.24	Nano08.smali	21
3.25	Nano09.py	22
3.26	Nano09.java	22
3.27	Nano09.smali	23
3.28	Nano10.py	23
3.29	Nano10.java	24
3.30	Nano10.smali	24
3.31	Nano11.py	24
3.32	Nano11.java	25
3.33	Nano11.smali	25
3.34	Nano12.py	26
3.35	Nano12.java	26
3.36	Nano12.smali	26
4.1	lexico.mll	30
4.2	pre-processador.ml	33
4.3	carregador.ml	36
4.4	teste.py	36

4.5	nano01.py	37
4.6	nano02.py	37
4.7	nano03.py	37
4.8	nano04.py	37
4.9	nano05.py	37
4.10	nano06.py	38
4.11	nano07.py	38
4.12	nano08.py	38
4.13	nano09.py	38
4.14	nano10.py	39
4.15	nano11.py	39
4.16	nano12.py	39
5.1	lexico.mll	41
5.2	sintatico.mly	44
5.3	ast.ml	48
5.4	sintaticoTest.ml	49
5.5	astMicro01	51
5.6	astMicro02	52
5.7	astMicro03	52
5.8	astMicro04	53
5.9	astMicro05	53
5.10	astMicro06	54
5.11	astMicro07	54
5.12	astMicro08	55
5.13	astMicro09	55
5.14	astMicro10	56
5.15	astMicro11	57

Sumário

Lista de Figuras	2
Lista de Tabelas	3
1 Introdução	8
1.1 Sistema Operacional	8
1.2 Python	8
1.3 Dalvik	8
1.4 Smali/Baksmali	9
1.5 OCaml	9
2 Instalações	10
2.1 Python	10
2.2 Java	10
2.3 Dalvik	10
2.4 OCaml	11
2.5 Compilação	11
2.5.1 Compilando Java em .dex	12
2.5.2 Complilando .dex em .smali	12
2.5.3 Compilando .smali code em .dex	12
2.6 Executando o arquivo .dex no Android	12
2.6.1 Utilizando Emulador AVD	12
3 Nano Programas	15
3.1 Nano01	15
3.2 Nano02	16
3.3 Nano03	16
3.4 Nano04	17
3.5 Nano05	18
3.6 Nano06	19
3.7 Nano07	20
3.8 Nano08	21
3.9 Nano09	22
3.10 Nano10	23
3.11 Nano11	24
3.12 Nano12	26

4	Analizador Léxico	28
4.1	Lista de Tokens	29
4.2	Códigos	30
4.3	Compilação e execução	36
4.4	Análise léxica Nanos	37
5	Análise sintática	41
5.1	Código	41
5.2	Execução	51
5.3	Análise sintática dos programas micro	51
6	Referências	58

Capítulo 1

Introdução

Este documento foi escrito para auxiliar na confecção do relatório da disciplina de Construção de Compiladores com a finalidade de detalhar todo o trabalho desenvolvido e os processos envolvidos da Construção de um Compilador, mais especificamente, um Compilador de Python para Dalvik, utilizando a linguagem OCaml para a construção do mesmo.

1.1 Sistema Operacional

Para esse trabalho foi utilizado o sistema operacional *Fedora 28*, sua instalação é fácil e rápida, basta acessar o [site](#) e seguir os passos descritos pela desenvolvedora do sistema.

1.2 Python

Python é uma ótima linguagem de programação orientada a objetos, interpretada e interativa é uma linguagem de programação orientada a objetos, interpretada, de script, interativa, funcional e de tipagem dinâmica. Criada por Guido van Rossum em 1991, hoje segue o modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos *Python Software Foundation*.

1.3 Dalvik

Desenvolvida por Dan Bornstein e com contribuições de outros engenheiros do Google, é uma máquina virtual baseada em registradores e foi projetada para ser utilizada no sistema operacional Android. É muito conhecida pelo seu bom desempenho, pelo baixo consumo de memória e foi projetada para permitir que múltiplas instâncias da máquina virtual rodem ao mesmo tempo. A *Dalvik* é frequentemente confundida com uma Java Virtual Machine, porém, o bytecode que ela opera é bastante diferente do bytecode da JVM. A VM do Dalvik, executa um bytecode no formato `.dex` (Dalvik Executable), códigos em `.dex`

são ilegíveis aos humanos, portanto, neste trabalho usaremos Smali Code para apresentar os códigos.

1.4 Smali/Baksmali

O *smali/baksmali* é um *assembler/disassembler* para o formato *.dex* usado pela *Dalvik*, que gera um arquivo *SmaliCode*. A sintaxe é vagamente baseada na sintaxe do *Jasmin*, e suporta a funcionalidades do formato *.dex* (anotações, informações de depuração, informações de linha, etc.)

1.5 OCaml

Objective Caml, ou somente *OCaml*, é uma linguagem de programação funcional e fortemente tipada, da família ML com ênfase na expressividade e na segurança. É usada em aplicações sensíveis onde um único erro pode custar milhões. Será utilizada na implementação de nosso compilador.

Capítulo 2

Instalações

2.1 Python

Já vem instalado por padrão em sistemas *GNU/Linux*, para conferir a versão, digite no terminal:

```
> which python
```

2.2 Java

Para checar as versões disponíveis, digite no terminal:

```
> sudo dnf search openjdk
```

Instale a versão desejada digitando no terminal:

```
> sudo dnf install <openjdk-package-name>
```

Por exemplo:

```
> sudo dnf install java-1.8.0-openjdk.x86_64
```

Para verificar se foi instalado com sucesso digite:

```
> java -version
```

2.3 Dalvik

O Dalvik é uma VM executada em android, então neste trabalho usaremos o Android Studio. Para instalar basta ir no [site](#) baixar a versão que se aplica ao seu SO e configurar o PATH no terminal:

```
>export PATH=\$PATH:/diretoriolocal/android-studio/bin
```

Para executar o Android Studio, entre no diretório android-studio/bin e digite no terminal:

```
>sh studio.sh
```

Os componentes adicionais serão instalados com a ajuda do assistente de configuração na primeira execução do programa.

2.4 OCaml

Versão utilizada: 4.07.0

```
>sudo dnf install wget
>sudo dnf install git m4 mercurial darcs
>wget https://raw.githubusercontent.com/ocaml/opam/master/shell/
>opam_installer.sh -O - | sh -s /dev/bin
>opam init
>eval `opam config env`
>opam repository add git git+https://github.com/ocaml/
>opam-repository
>opam update
```

Para saber qual a versão mais atual:

```
>opam switch
```

Instalando a versão 4.0.7:

```
>opam switch 4.07.0
>eval `opam config env`
```

Instalar o rlwrap para trabalhar melhor com o OCaml:

```
> sudo dnf update
> sudo dnf install rlwrap
```

Executar o OCaml com o rlwrap:

```
>rlwraper ocaml
```

2.5 Compilação

Como gerar .smeli a partir do .dex.

2.5.1 Compilando Java em .dex

Dentro do diretório execute no terminal:

```
> javac file.java  
> diretorioSDK/build-tools/version/dx --dex --output=file.dex file.class
```

Onde esta "diretorioSDK" é o caminho no qual foi instalado o SDK com o assistente de configuração do Android Studio, assim, como "version" a versão que esta sendo utilizada. Baixe o baksmali e smali, os dois na versão 2.2.6, no [site](#) para fazer o desassembly. É importante ressaltar que tem que deixar o .dex/.smali e o baksmali/smali no mesmo diretório.

2.5.2 Complilando .dex em .smali

Dentro do diretório execute no terminal:

```
> java -jar baksmali-2.2.6.jar disassemble file.dex
```

2.5.3 Compilando .smali code em .dex

Dentro do diretório execute no terminal:

```
> java -jar smali-2.2.6.jar assemble file.smali -o file.dex
```

2.6 Executando o arquivo .dex no Android

Depois de compilado o .dex, para executa-lo usaremos um emulador que rode o sistema operacional android. Usaremos o modelo Nexus 5, android 5.1.

2.6.1 Utilizando Emulador AVD

Após instalado o Android Studio, execute-o e crie um novo projeto em branco. Na interface do programa, vá em: *Tools > ADV Manager*.

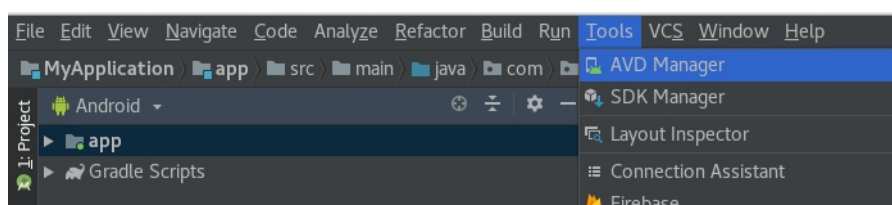


Figura 2.1: *ADV Manager*

A seguinte tela irá aparecer:

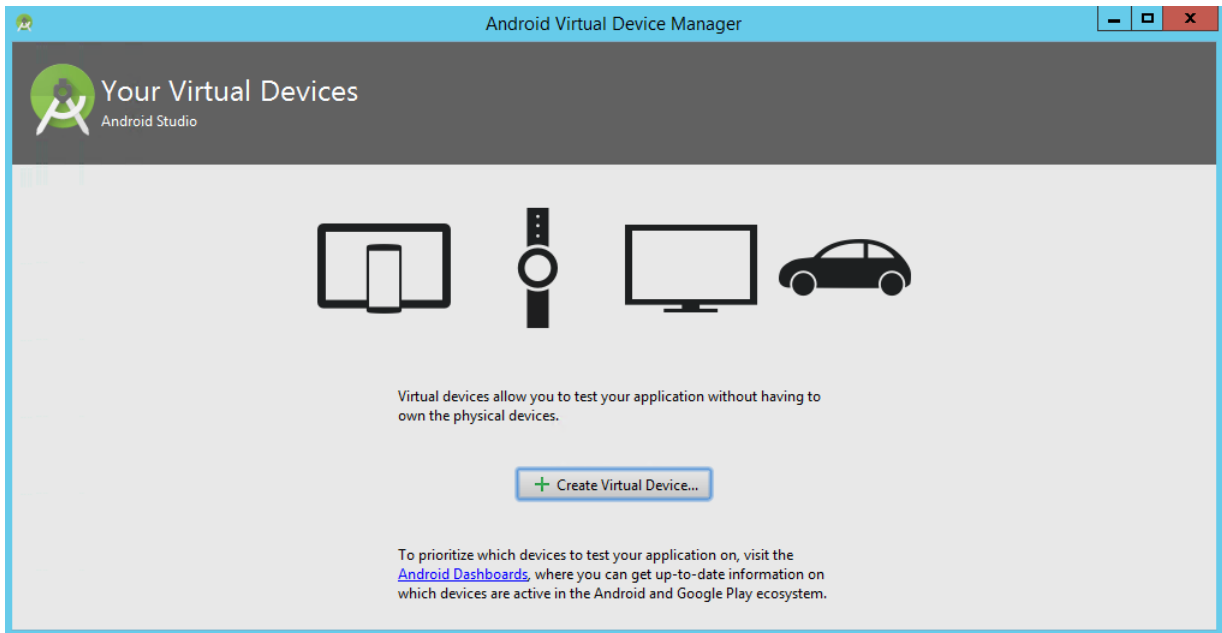


Figura 2.2: Criar um novo device

Nessa janela, selecione *Create Virtual Device*.

Logo em seguida irá aparecer essa janela:

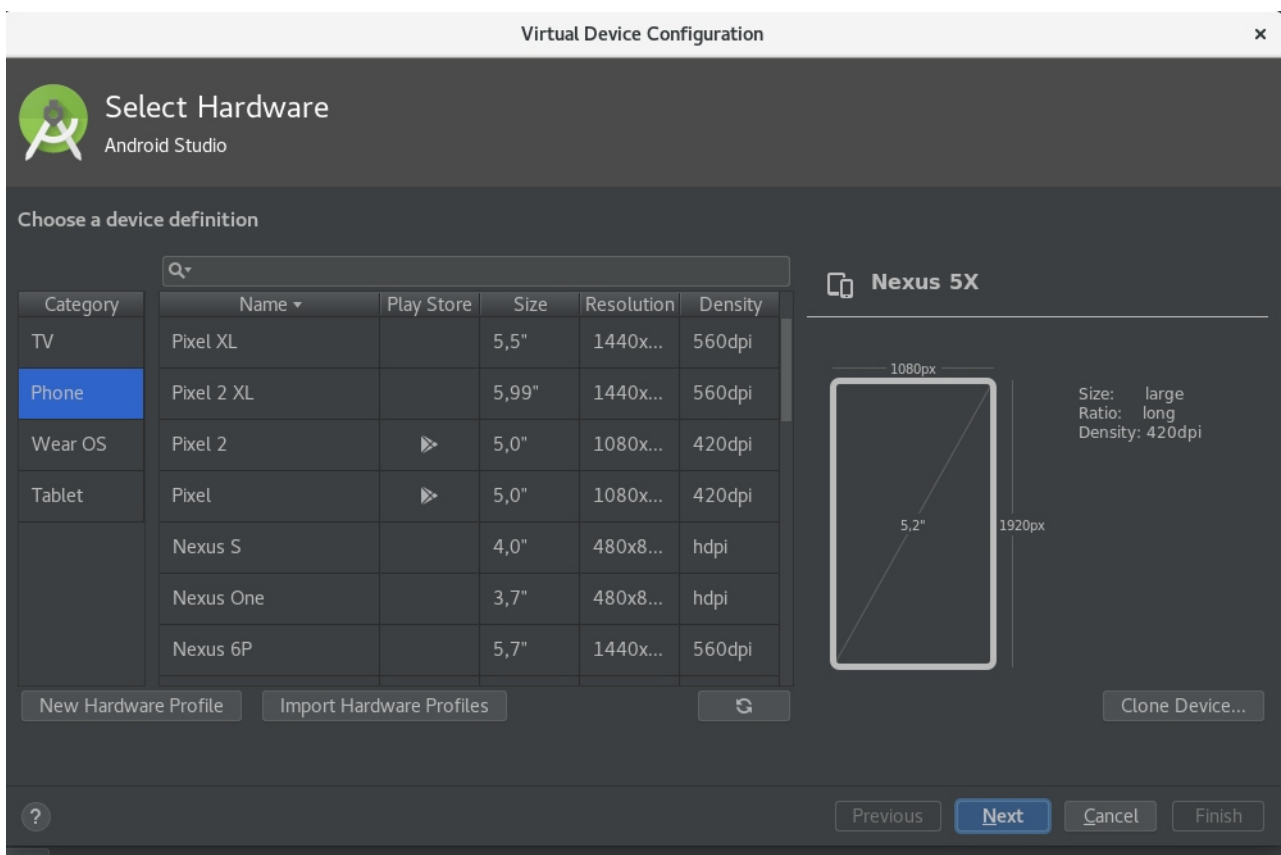


Figura 2.3: Modelo Device

Selecione o modelo que deseja emular, indico usar modelos Nexus, continue para as próximas

janelas clicando em next.

Abrindo o ADV Manager novamente, seu modelo emulado deverá aparecer como mostrado na figura abaixo, para iniciar, clique no ícone *Play*.

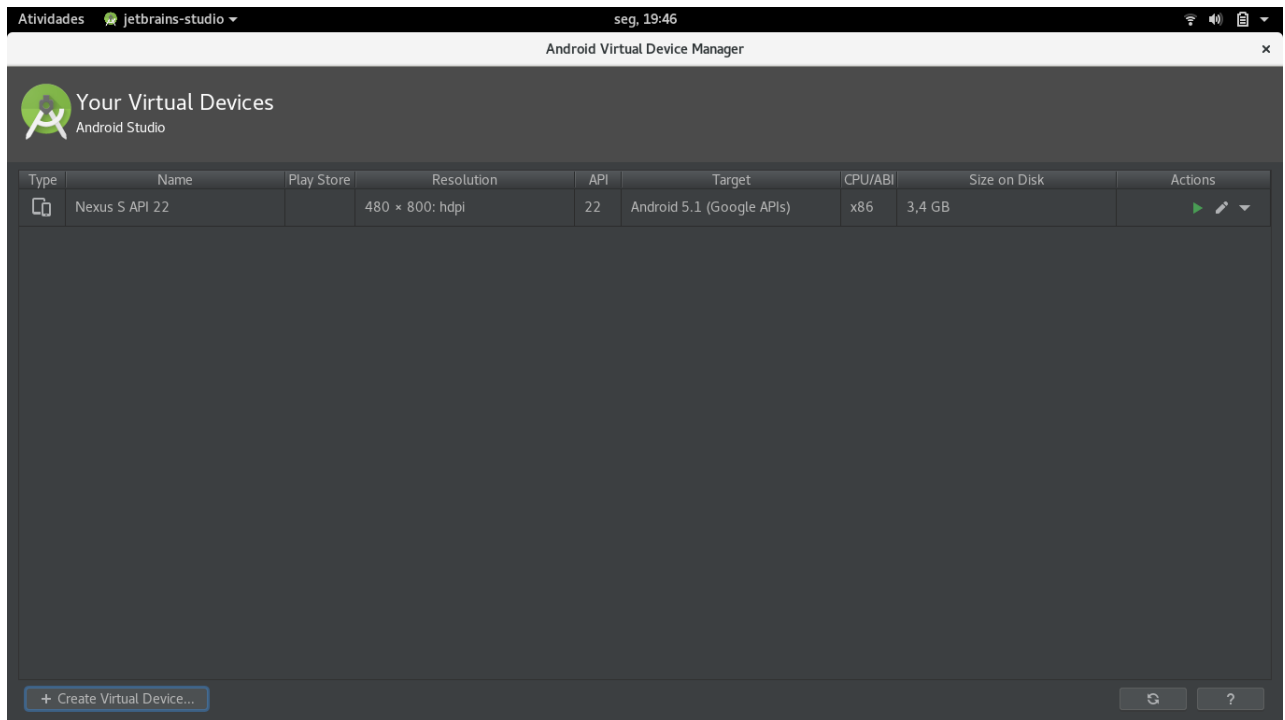


Figura 2.4: *Manager Devices*

Ao fazer esses passos irá aparecer um emulador da tela do celular, com isso já podemos rodar arquivos .dex.

No terminal digite:

```
> ./adb devices
> ./adb push /home/fenrir/dev/Projetos/compilados/nano01.dex /data/local
> ./adb shell dalvikvm -cp /data/local/nano01.dex nano01
```

A execução da ferramenta adb deve ser feita dentro do diretório *androidpath/sdk/platform-tools/*.

Capítulo 3

Nano Programas

Neste capítulo será apresentado alguns programas e suas respectivas versões em Python, Java e Smali Code.

3.1 Nano01

Listagem 3.1: Nano01.py

```
1 def main() -> None:
2     return
```

Listagem 3.2: Nano01.java

```
1 public class Nano01 {
2     public static void main(String[] args) {
3     }
4 }
```

Listagem 3.3: Nano01.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 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 3
22     return-void
23 .end method

```

Saída : nenhuma.

3.2 Nano02

Listagem 3.4: Nano02.py

```

1 def main() -> None:
2     n: int = 0

```

Listagem 3.5: Nano02.java

```

1 public class Nano02 {
2     public static void main(String[] args) {
3         int n;
4     }
5 }

```

Listagem 3.6: Nano02.smali

```

1 .class public LNano02;
2 .super Ljava/lang/Object;
3 .source "Nano02.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 1
19
20     .prologue
21     .line 4
22     return-void
23 .end method

```

Saída : nenhuma.

3.3 Nano03

Listagem 3.7: Nano03.py

```

1 def main() -> None:

```


3.4

```
2      n: int = 1
```

Listagem 3.8: Nano03.java

```
1 public class Nano03 {  
2     public static void main(String[] args) {  
3         int n;  
4         n = 1;  
5     }  
6 }
```

Listagem 3.9: Nano03.smali

```
1 .class public LNano03;  
2 .super Ljava/lang/Object;  
3 .source "Nano03.java"  
4  
5  
6 # direct methods  
7 .method public constructor <init>()V  
8     .registers 1  
9  
10    .prologue  
11    .line 1  
12    invoke-direct {p0}, Ljava/lang/Object;--><init>()V  
13  
14    return-void  
15 .end method  
16  
17 .method public static main([Ljava/lang/String;)V  
18     .registers 1  
19  
20     .prologue  
21     .line 4  
22     .line 5  
23     return-void  
24 .end method
```

Saída : nenhuma.

3.4 Nano04

Listagem 3.10: Nano04.py

```
1 def main() -> None:  
2     n: int = 1 + 2
```

Listagem 3.11: Nano04.java

```
1 public class Nano04 {  
2     public static void main(String[] args) {  
3         int n;  
4         n = 1+2;  
5     }  
6 }
```

Listagem 3.12: Nano04.smali

```

1 .class public LNano04;
2 .super Ljava/lang/Object;
3 .source "Nano04.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object; -> <init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 1
19
20    .prologue
21    .line 4
22    .line 5
23    return-void
24 .end method

```

Saída : nenhuma.

3.5 Nano05

Listagem 3.13: Nano05.py

```

1 def main() -> None:
2     n: int = 2
3     print(n)
4
5 main()

```

Listagem 3.14: Nano05.java

```

1 public class Nano05 {
2     public static void main(String[] args) {
3         int n;
4         n = 2;
5         System.out.println(n);
6     }
7 }

```

Listagem 3.15: Nano05.smali

```

1 .class public LNano05;
2 .super Ljava/lang/Object;
3 .source "Nano05.java"
4
5
6 # direct methods

```

3.6

```
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;--><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20     .prologue
21     .line 4
22     const/4 v0, 0x2
23
24     .line 5
25     sget-object v1, Ljava/lang/System;-->out:Ljava/io/PrintStream;
26
27     invoke-virtual {v1, v0}, Ljava/io/PrintStream;-->println(I)V
28
29     .line 6
30     return-void
31 .end method
```

Saída : 2.

3.6 Nano06

Listagem 3.16: Nano06.py

```
1 def main() -> None:
2     n: int = 1 - 2
3     print(n)
4
5 main()
```

Listagem 3.17: Nano06.java

```
1 public class Nano06 {
2     public static void main(String[] args) {
3         int n;
4         n = 1-2;
5         System.out.println(n);
6     }
7 }
```

Listagem 3.18: Nano06.smali

```
1 .class public LNano06;
2 .super Ljava/lang/Object;
3 .source "Nano06.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
```

```

8      .registers 1
9
10     .prologue
11     .line 1
12     invoke-direct {p0}, Ljava/lang/Object; -><init>()V
13
14     return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20     .prologue
21     .line 4
22     const/4 v0, -0x1
23
24     .line 5
25     sget-object v1, Ljava/lang/System; ->out:Ljava/io/PrintStream;
26
27     invoke-virtual {v1, v0}, Ljava/io/PrintStream; ->println(I)V
28
29     .line 6
30     return-void
31 .end method

```

Saída : -1.

3.7 Nano07

Listagem 3.19: Nano07.py

```

1 def main() -> None:
2     n = 1
3     if n == 1:
4         print(n)
5
6 main()

```

Listagem 3.20: Nano07.java

```

1 public class Nano07 {
2     public static void main(String[] args) {
3         int n;
4         n = 1;
5         if(n==1) {
6             System.out.println(n);
7         }
8     }
9 }

```

Listagem 3.21: Nano07.smali

```

1 .class public LNano07;
2 .super Ljava/lang/Object;
3 .source "Nano07.java"
4
5

```

```

6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10     .prologue
11     .line 1
12     invoke-direct {p0}, Ljava/lang/Object;--><init>()V
13
14     return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20     .prologue
21     .line 4
22     const/4 v0, 0x1
23
24     .line 6
25     sget-object v1, Ljava/lang/System;-->out:Ljava/io/PrintStream;
26
27     invoke-virtual {v1, v0}, Ljava/io/PrintStream;-->println(I)V
28
29     .line 8
30     return-void
31 .end method

```

Saída : 1.

3.8 Nano08

Listagem 3.22: Nano08.py

```

1 def main() -> None:
2     n: int = 1
3     if n == 1:
4         print(n)
5     else:
6         print(0)
7
8 main()

```

Listagem 3.23: Nano08.java

```

1 public class Nano08 {
2     public static void main(String[] args) {
3         int n;
4         n = 1;
5         if(n==1){
6             System.out.println(n);
7         }else{
8             System.out.println(0);
9         }
10    }
11 }

```

Listagem 3.24: Nano08.smali

```

1 .class public LNano08;
2 .super Ljava/lang/Object;
3 .source "Nano08.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object; -> <init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18    .registers 3
19
20    .prologue
21    .line 4
22    const/4 v0, 0x1
23
24    .line 6
25    sget-object v1, Ljava/lang/System; -> out:Ljava/io/PrintStream;
26
27    invoke-virtual {v1, v0}, Ljava/io/PrintStream; -> println(I)V
28
29    .line 10
30    return-void
31 .end method

```

Saída : 1.

3.9 Nano09

Listagem 3.25: Nano09.py

```

1 def main() -> None:
2     n: int = 1 + (1 / 2)
3     if n == 1:
4         print(n)
5     else:
6         print(0)
7
8 main()

```

Listagem 3.26: Nano09.java

```

1 public class Nano09 {
2     public static void main(String[] args) {
3         int n;
4
5         n = 1 + 1 / 2;
6         if(n==1) {

```

3.10

```
7         System.out.println(n);
8     } else {
9         System.out.println(0);
10    }
11 }
12 }
```

Listagem 3.27: Nano09.smali

```
1 .class public LNano09;
2 .super Ljava/lang/Object;
3 .source "Nano09.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20    .prologue
21    .line 5
22    const/4 v0, 0x1
23
24    .line 7
25    sget-object v1, Ljava/lang/System;->out:Ljava/io/PrintStream;
26
27    invoke-virtual {v1, v0}, Ljava/io/PrintStream;->println(I)V
28
29    .line 11
30    return-void
31 .end method
```

Saída : 0.

3.10 Nano10

Listagem 3.28: Nano10.py

```
1 def main() -> None:
2     n: int = 1
3     m: int = 2
4     if n == m:
5         print(n)
6     else:
7         print(0)
8
9 main()
```

Listagem 3.29: Nano10.java

```

1 public class Nano10 {
2     public static void main(String[] args) {
3         int n, m;
4         n = 1;
5         m = 2;
6         if(n==m) {
7             System.out.println(n);
8         }else{
9             System.out.println(0);
10        }
11    }
12
13 }

```

Listagem 3.30: Nano10.smali

```

1 .class public LNano10;
2 .super Ljava/lang/Object;
3 .source "Nano10.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 3
19
20     .prologue
21     .line 4
22     .line 9
23     sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
24
25     const/4 v1, 0x0
26
27     invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(I)V
28
29     .line 11
30     return-void
31 .end method

```

Saída : 0.

3.11 Nano11

Listagem 3.31: Nano11.py

```

1 def main() -> None:

```



```

2      n: int = 1
3      m: int = 2
4      x: int = 5
5      while x > n:
6          n = n + m
7          print(n)
8
9 main()

```

Listagem 3.32: Nano11.java

```

1 public class Nano11 {
2     public static void main(String[] args) {
3         int n, m, x;
4
5         n = 1;
6         m = 2;
7         x = 5;
8         while (x>n) {
9             n = n + m;
10            System.out.println(n);
11        }
12    }
13 }

```

Listagem 3.33: Nano11.smali

```

1 .class public LNano11;
2 .super Ljava/lang/Object;
3 .source "Nano11.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;-><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 5
19
20     .prologue
21     .line 5
22     const/4 v0, 0x1
23
24     .line 6
25     const/4 v1, 0x2
26
27     .line 7
28     const/4 v2, 0x5
29
30     .line 8
31     :goto_3
32     if-le v2, v0, :cond_c

```

```

33
34     .line 9
35     add-int/2addr v0, v1
36
37     .line 10
38     sget-object v3, Ljava/lang/System; ->out:Ljava/io/PrintStream;
39
40     invoke-virtual {v3, v0}, Ljava/io/PrintStream; ->println(I)V
41
42     goto :goto_3
43
44     .line 12
45     :cond_c
46     return-void
47 .end method

```

Saída : 3.

Saída : 5.

3.12 Nano12

Listagem 3.34: Nano12.py

```

1 def main() -> None:
2     n: int = 1
3     m: int = 2
4     x: int = 5
5     while x > n:
6         if n == m:
7             print(n)
8         else:
9             print(0)
10        x = x - 1
11
12 main()

```

Listagem 3.35: Nano12.java

```

1 public class Nano12 {
2     public static void main(String[] args) {
3         int n, m, x;
4         n = 1;
5         m = 2;
6         x = 5;
7         while(x>n) {
8             if(n==m)
9                 System.out.println(n);
10            else
11                System.out.println(0);
12            x = x - 1;
13        }
14    }
15 }

```

Listagem 3.36: Nano12.smali

```

1 .class public LNano12;

```

```

2 .super Ljava/lang/Object;
3 .source "Nano12.java"
4
5
6 # direct methods
7 .method public constructor <init>()V
8     .registers 1
9
10    .prologue
11    .line 1
12    invoke-direct {p0}, Ljava/lang/Object;--><init>()V
13
14    return-void
15 .end method
16
17 .method public static main([Ljava/lang/String;)V
18     .registers 5
19
20    .prologue
21    .line 4
22    const/4 v1, 0x1
23
24    .line 6
25    const/4 v0, 0x5
26
27    .line 7
28    :goto_2
29    if-le v0, v1, :cond_d
30
31    .line 11
32    sget-object v2, Ljava/lang/System;-->out:Ljava/io/PrintStream;
33
34    const/4 v3, 0x0
35
36    invoke-virtual {v2, v3}, Ljava/io/PrintStream;-->println(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

```

Saída : 0.

Saída : 0.

Saída : 0.

Saída : 0.

Capítulo 4

Analizador Léxico

Esse capítulo irá abordar como foi criado o analisador léxico para a linguagem Python.

A análise léxica é a primeira fase do compilador, e tem como tarefa analisar um alfabeto de uma determinada linguagem. Após receber uma sequência de caracteres, ele produz uma sequência de nomes, palavras-chaves e sinais de pontuação chamados de *tokens*. Ainda nessa fase, é de responsabilidade do analisador o descarte de elementos "decorativos" do programa, tais como espaços em branco e comentários entre os tokens.

A ferramenta utilizada para construir o analisador léxico foi o *ocamllex*, que cria um analisador, muito semelhante ao funcionamento de um Automato Finito Determinístico, a partir de um conjunto de expressões regulares e ações semânticas para tais regras.

4.1 Lista de Tokens

Tipo	Representação	Tipo	Representação
FALSE	False	MODULO	%
NONE	None	EXP	**
TRUE	True	EQUIVALENTE	==
AND	and	NAOEQUIVALENTE	!=
AS	as	MENOR	«
BREAK	break	MAIOR	»
CONTINUE	continue	MENORIGUAL	«=
DEF	def	MAIORIGUAL	»=
DEL	del	IGUAL	=
ELIF	elif	APAR	(
ELSE	else	FPAR)
EXCEPT	except	ACOLCHETE	[
FOR	for	FCOLCHETE]
FROM	from	ACHAVE	{
IF	if	FCHAVE	}
IMPORT	import	PONTO	.
IN	in	VIRG	,
IS	is	DPONTOS	:
NOT	not	PVIRG	;
OR	or	ARROBA	@
RETURN	return	SOMAIGUAL	+=
WHILE	while	SUBIGUAL	-=
WITH	with	MULTIGUAL	*=
SOMA	+	DIVIGUAL	/=
SUB	-	DIVINTIGUAL	//=
MULT	*	MODULOIGUAL	%=
DIV	/	EXPIGUAL	**/
DIVINT		SETA	->

Tabela 4.1: *Tabela de Tokens*

4.2 Códigos

Foi incluído o código `pre_processador.ml`, como foi pedido pelo professor para quem estivesse fazendo o trabalho em Python.

Listagem 4.1: `lexico.ml`

```

1 {
2
3   open Lexing
4   open Printf
5   open Sintatico
6
7   type token =
8   | LITINT of (int)
9   | LITSTRING of (string)
10  | ID of (string)
11  | FALSE
12  | NONE
13  | TRUE
14  | AND
15  | AS
16  | BREAK
17  | CONTINUE
18  | DEF
19  | DEL
20  | ELIF
21  | ELSE
22  | EXCEPT
23  | FOR
24  | FROM
25  | IF
26  | IMPORT
27  | IN
28  | IS
29  | NOT
30  | OR
31  | RETURN
32  | WHILE
33  | WITH
34  | SOMA
35  | SUB
36  | MULT
37  | DIV
38  | DIVINT
39  | MODULO
40  | EXP
41  | EQUIVALENTE
42  | NAOEQUIVALENTE
43  | MENOR
44  | MAIOR
45  | MENORIGUAL
46  | MAIORIGUAL
47  | IGUAL
48  | APAR
49  | FPAR
50  | ACOLCHETE
51  | FCOLCHETE
52  | ACHAVE

```

```

53 | FCHAVE
54 | PONTO
55 | VIRG
56 | DPONTOS
57 | PVIRG
58 | ARROBA
59 | SOMAIGUAL
60 | SUBIGUAL
61 | MULTIGUAL
62 | DIVIGUAL
63 | DIVINTIGUAL
64 | MODULOIGUAL
65 | EXPIGUAL
66 | SETA
67 | E
68 | ATRIB
69 | EOF
70 (* Os tokens a seguir são importantes para o pré processador *)
71 | Linha of (int * int * token list)
72 | INDENTA
73 | DEDENTA
74 | NOVALINHA
75
76
77 let nivel_par = ref 0
78
79 let incr_num_linha lexbuf =
80     let pos = lexbuf.lex_curr_p in
81     lexbuf.lex_curr_p <- { pos with
82         pos_lnum = pos.pos_lnum + 1;
83         pos_bol = pos.pos_cnum;
84     }
85
86 let msg_erro lexbuf c =
87     let pos = lexbuf.lex_curr_p in
88     let lin = pos.pos_lnum
89     and col = pos.pos_cnum - pos.pos_bol - 1 in
90     sprintf "%d-%d: caracter desconhecido %c" lin col c
91
92 }
93
94 let digito = ['0' - '9']
95 let int = digito+
96
97 let comentario = "#"[ ^ '\n' ]*
98
99 let linha_em_branco = [' ' '\t']* comentario
100 let restante = [^ ' ' '\t' '\n' ] [^ '\n']+
101 let brancos = [' ' '\t']+
102 let novalinha = '\r' | '\n' | "\r\n"
103
104 let letra = [ 'a'-'z' 'A' - 'Z' ]
105 let identificador = letra ( letra | digito | '_' )*
106
107 (* O pré processador necessário para contabilizar a indentação *)
108 rule preprocessador indentacao = parse
109     linha_em_branco          { preprocessador 0 lexbuf } (* ignora brancos *)
110 | [' ' '\t' ]+ '\n'        { incr_num_linha lexbuf;
                             preprocessador 0 lexbuf } (* ignora brancos *)

```

```

112 | ' ' { preprocessador (indentacao + 1) lexbuf }
113 | '\t' { let nova_ind = indentacao + 8 - (indentacao
      mod 8)
114 | in preprocessador nova_ind lexbuf }
115 | novalinha { incr_num_linha lexbuf;
116 | preprocessador 0 lexbuf }
117 | restante as linha {
118 | let rec tokenize lexbuf =
119 | let tok = token lexbuf in
120 | match tok with
121 | EOF -> []
122 | _ -> tok :: tokenize lexbuf in
123 | let toks = tokenize (Lexing.from_string linha) in
124 | (* A impressão a seguir serve apenas para depuração. Retirar depois!
      *)
125 | printf "Linha(indentacao=%d,nivel_par=%d)\n" indentacao (!nivel_par);
126 | Linha(indentacao,!nivel_par, toks)
127 | }
128 | eof { nivel_par := 0; EOF }
129 |
130 | (* O analisador léxico a ser chamado após o pré processador *)
131 | and token = parse
132 | brancos { token lexbuf }
133 | "False" { FALSE }
134 | "None" { NONE }
135 | "True" { TRUE }
136 | "and" { AND }
137 | "as" { AS }
138 | "break" { BREAK }
139 | "continue" { CONTINUE }
140 | "def" { DEF }
141 | "del" { DEL }
142 | "elif" { ELIF }
143 | "else" { ELSE }
144 | "except" { EXCEPT }
145 | "for" { FOR }
146 | "from" { FROM }
147 | "if" { IF }
148 | "import" { IMPORT }
149 | "in" { IN }
150 | "is" { IS }
151 | "not" { NOT }
152 | "or" { OR }
153 | "return" { RETURN }
154 | "while" { WHILE }
155 | "with" { WITH }
156 | "+" { SOMA }
157 | "-" { SUB }
158 | "*" { MULT }
159 | "/" { DIV }
160 | "//" { DIVINT }
161 | "%" { MODULO }
162 | "*" { EXP }
163 | "==" { EQUIVALENTE }
164 | "!=" { NAOEQUIVALENTE }
165 | "<" { MENOR }
166 | ">" { MAIOR }
167 | "<=" { MENORIGUAL }
168 | ">=" { MAIORIGUAL }

```



```

169 | "="                { IGUAL }
170 | "("                { APAR }
171 | ")"                { FPAR }
172 | "["                { ACOLCHETE }
173 | "]"                { FCOLCHETE }
174 | "{"                { ACHAVE }
175 | "}"                { FCHAVE }
176 | "."                { PONTO }
177 | ","                { VIRG }
178 | ":"                { DPONTOS }
179 | ";"                { PVIRG }
180 | "@"                { ARROBA }
181 | "+="               { SOMAIGUAL }
182 | "-="               { SUBIGUAL }
183 | "*="               { MULTIGUAL }
184 | "/="               { DIVIGUAL }
185 | "//="              { DIVINTIGUAL }
186 | "%="               { MODULOIGUAL }
187 | "**/"               { EXPIGUAL }
188 | comentario         { token lexbuf }
189 | "'''"              { comentario_bloco 0 lexbuf }
190
191 | int as num          { let numero = int_of_string num in
192                       LITINT numero }
193
194 | '""'                { let buffer = Buffer.create 1 in
195                       let str = leia_string buffer lexbuf in
196                       LITSTRING str }
197
198 | identificador as id { ID id }
199 | _ as c              { failwith (msg_erro lexbuf c); }
200 | eof                 { EOF }
201
202 and comentario_bloco n = parse
203     "'''" { if n=0 then token lexbuf
204             else comentario_bloco (n - 1) lexbuf }
205     | "'''" { comentario_bloco (n + 1) lexbuf; }
206     | _     { comentario_bloco n lexbuf }
207     | eof   { failwith "Comentário não fechado "}
208
209 and leia_string buffer = parse
210     | '""'      { Buffer.contents buffer }
211     | "\\t"     { Buffer.add_char buffer '\t'; leia_string buffer lexbuf }
212     | "\\n"     { Buffer.add_char buffer '\n'; leia_string buffer lexbuf }
213     | '\\\'' '\\\'' { Buffer.add_char buffer '\\\''; leia_string buffer lexbuf }
214     | '\\\'' '\\\'' { Buffer.add_char buffer '\\\''; leia_string buffer lexbuf }
215     | _ as c    { Buffer.add_char buffer c; leia_string buffer lexbuf }
216     | eof       { failwith "A string não foi fechada." }

```

Listagem 4.2: pre-processor.ml

```

1 open Lexico
2 open Printf
3
4 (* Pré processa o arquivo gerando os tokens de indenta e dedenta *)
5
6 let preprocessa lexbuf =
7     let pilha = Stack.create ()
8     and npar = ref 0 in

```

```

9      let _ = Stack.push 0 pilha in
10     let off_side toks nivel =
11     let _ = printf "Nivel: %d\n" nivel in
12     if !npar != 0 (* nova linha entre parenteses *)
13     then toks      (* nao faz nada *)
14     else if nivel > Stack.top pilha
15         then begin
16             Stack.push nivel pilha;
17             INDENTA :: toks
18         end
19     else if nivel = Stack.top pilha
20         then toks
21     else begin
22         let prefixo = ref toks in
23         while nivel < Stack.top pilha do
24             ignore (Stack.pop pilha);
25             if nivel > Stack.top pilha
26                 then failwith "Erro de indentacao"
27             else prefixo := DEDENTA :: !prefixo
28         done;
29         !prefixo
30     end
31 in
32
33 let rec dedenta sufixo =
34     if Stack.top pilha != 0
35     then let _ = Stack.pop pilha in
36         dedenta (DEDENTA :: sufixo)
37     else sufixo
38 in
39 let rec get_tokens () =
40     let tok = Lexico.preprocessador 0 lexbuf in
41     match tok with
42     | Linha(nivel,npars,toks) ->
43         let new_toks = off_side toks nivel in
44         npar := npars;
45         new_toks @ (if npars = 0
46                     then NOVALINHA :: get_tokens ()
47                     else get_tokens ())
48     | _ -> dedenta []
49 in get_tokens ()
50
51
52 (* Chama o analisador léxico *)
53 let lexico =
54     let tokbuf = ref None in
55     let carrega lexbuf =
56         let toks = preprocessa lexbuf in
57         (match toks with
58          | tok::toks ->
59              tokbuf := Some toks;
60              tok
61          | [] -> print_endline "EOF";
62              EOF)
63     in
64     fun lexbuf ->
65     match !tokbuf with
66     | Some tokens ->
67         (match tokens with

```

4.2

```
68         tok::toks ->
69         tokbuf := Some toks;
70         tok
71     | [] -> carrega lexbuf)
72 | None -> carrega lexbuf
```

Listagem 4.3: carregador.ml

```

1 #load "lexico.cmo"
2 #load "pre_processador.cmo"
3
4 type nome_arq = string
5 type tokens = Lexico.token list
6
7 let rec tokens lexbuf =
8   let tok = Pre_processador.lexico lexbuf in
9   match tok with
10  | Lexico.EOF -> (|Lexico.EOF):tokens)
11  | _ -> tok :: tokens lexbuf
12 ;;
13
14 let lexico str =
15   let lexbuf = Lexing.from_string str in
16   tokens lexbuf
17 ;;
18
19 let lex (arq:nome_arq)=
20   let ic = open_in arq in
21   let lexbuf = Lexing.from_channel ic in
22   let toks = tokens lexbuf in
23   let _ = close_in ic in
24   toks

```

4.3 Compilação e execução

Para compilar navegue pelo terminal ate o diretório dos arquivos e execute os comandos:

```

>ocamllex lexico.mll
>ocamlc -c lexico.ml
>ocamlc -c pre_processador.ml

```

Após a compilação execute:

```
>lrwrap ocaml
```

Dentro do *O caml* execute os comandos:

```

# #use "carregador.ml";;
# lex "codigo.py";;

```

Um exemplo com o arquivo teste.py com o código:

Listagem 4.4: teste.py

```

1 def paco(x):
2   x = x + 1
3   y = x + 2
4   y = x + 3 + ( 4 + 5)
5   return x

```

A saída do analisador léxico:

```
- : tokens =
  [Lexico.DEF; Lexico.ID "paco"; Lexico.APAR; Lexico.ID "x"; Lexico.FPAR;
  Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.INDENTA; Lexico.ID "x";
  Lexico.IGUAL; Lexico.ID "x"; Lexico.SOMA; Lexico.LITINT 1; Lexico.
    NOVALINHA;
  Lexico.ID "y"; Lexico.IGUAL; Lexico.ID "x"; Lexico.SOMA; Lexico.LITINT
    2;
  Lexico.NOVALINHA; Lexico.ID "y"; Lexico.IGUAL; Lexico.ID "x"; Lexico.
    SOMA;
  Lexico.LITINT 3; Lexico.SOMA; Lexico.APAR; Lexico.LITINT 4; Lexico.SOMA;
  Lexico.LITINT 5; Lexico.FPAR; Lexico.NOVALINHA; Lexico.RETURN;
  Lexico.ID "x"; Lexico.NOVALINHA; Lexico.DEDENTA; Lexico.EOF]
```

4.4 Analise léxica Nanos

Analise léxica dos nanos programas em python vistos no capítulo anterior.

Listagem 4.5: nano01.py

```
1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.RETURN; Lexico.NOVALINHA; Lexico.DEDENTA; Lexico.EOF]
```

Listagem 4.6: nano02.py

```
1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.ID "n"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
4 Lexico.LITINT 0; Lexico.NOVALINHA; Lexico.DEDENTA; Lexico.EOF]
```

Listagem 4.7: nano03.py

```
1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.ID "n"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
4 Lexico.LITINT 1; Lexico.NOVALINHA; Lexico.DEDENTA; Lexico.EOF]
```

Listagem 4.8: nano04.py

```
1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.ID "n"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
4 Lexico.LITINT 1; Lexico.SOMA; Lexico.LITINT 2; Lexico.NOVALINHA;
5 Lexico.DEDENTA; Lexico.EOF]
```

Listagem 4.9: nano05.py

```
1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
```

```

3 Lexico.ID "n"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
4 Lexico.LITINT 2; Lexico.NOVALINHA; Lexico.ID "print"; Lexico.APAR;
5 Lexico.ID "n"; Lexico.FPAR; Lexico.NOVALINHA; Lexico.DEDENTA;
6 Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.NOVALINHA; Lexico.EOF]

```

Listagem 4.10: nano06.py

```

1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.ID "n"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
4 Lexico.LITINT 1; Lexico.SUB; Lexico.LITINT 2; Lexico.NOVALINHA;
5 Lexico.ID "print"; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR;
6 Lexico.NOVALINHA; Lexico.DEDENTA; Lexico.ID "main"; Lexico.APAR;
7 Lexico.FPAR; Lexico.NOVALINHA; Lexico.EOF]

```

Listagem 4.11: nano07.py

```

1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.ID "n"; Lexico.IGUAL; Lexico.LITINT 1; Lexico.NOVALINHA; Lexico.IF
  ;
4 Lexico.ID "n"; Lexico.EQUIVALENTE; Lexico.LITINT 1; Lexico.DPONTOS;
5 Lexico.NOVALINHA; Lexico.INDENTA; Lexico.ID "print"; Lexico.APAR;
6 Lexico.ID "n"; Lexico.FPAR; Lexico.NOVALINHA; Lexico.DEDENTA;
7 Lexico.DEDENTA; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR;
8 Lexico.NOVALINHA; Lexico.EOF]

```

Listagem 4.12: nano08.py

```

1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.ID "n"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
4 Lexico.LITINT 1; Lexico.NOVALINHA; Lexico.IF; Lexico.ID "n";
5 Lexico.EQUIVALENTE; Lexico.LITINT 1; Lexico.DPONTOS; Lexico.NOVALINHA;
6 Lexico.INDENTA; Lexico.ID "print"; Lexico.APAR; Lexico.ID "n"; Lexico.
  FPAR;
7 Lexico.NOVALINHA; Lexico.DEDENTA; Lexico.ELSE; Lexico.DPONTOS;
8 Lexico.NOVALINHA; Lexico.INDENTA; Lexico.ID "print"; Lexico.APAR;
9 Lexico.LITINT 0; Lexico.FPAR; Lexico.NOVALINHA; Lexico.DEDENTA;
10 Lexico.DEDENTA; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR;
11 Lexico.NOVALINHA; Lexico.EOF]

```

Listagem 4.13: nano09.py

```

1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.ID "n"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
4 Lexico.LITINT 1; Lexico.SOMA; Lexico.APAR; Lexico.LITINT 1; Lexico.DIV;
5 Lexico.LITINT 2; Lexico.FPAR; Lexico.NOVALINHA; Lexico.IF; Lexico.ID "n";
6 Lexico.EQUIVALENTE; Lexico.LITINT 1; Lexico.DPONTOS; Lexico.NOVALINHA;
7 Lexico.INDENTA; Lexico.ID "print"; Lexico.APAR; Lexico.ID "n"; Lexico.
  FPAR;
8 Lexico.NOVALINHA; Lexico.DEDENTA; Lexico.ELSE; Lexico.DPONTOS;
9 Lexico.NOVALINHA; Lexico.INDENTA; Lexico.ID "print"; Lexico.APAR;

```

4.4

```
10 Lexico.LITINT 0; Lexico.FPAR; Lexico.NOVALINHA; Lexico.DEDENTA;
11 Lexico.DEDENTA; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR;
12 Lexico.NOVALINHA; Lexico.EOF]
```

Listagem 4.14: nano10.py

```
1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.ID "n"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
4 Lexico.LITINT 1; Lexico.NOVALINHA; Lexico.ID "m"; Lexico.DPONTOS;
5 Lexico.ID "int"; Lexico.IGUAL; Lexico.LITINT 2; Lexico.NOVALINHA; Lexico.
  IF;
6 Lexico.ID "n"; Lexico.EQUIVALENTE; Lexico.ID "m"; Lexico.DPONTOS;
7 Lexico.NOVALINHA; Lexico.INDENTA; Lexico.ID "print"; Lexico.APAR;
8 Lexico.ID "n"; Lexico.FPAR; Lexico.NOVALINHA; Lexico.DEDENTA; Lexico.ELSE
  ;
9 Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.INDENTA; Lexico.ID "print";
10 Lexico.APAR; Lexico.LITINT 0; Lexico.FPAR; Lexico.NOVALINHA; Lexico.
  DEDENTA;
11 Lexico.DEDENTA; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR;
12 Lexico.NOVALINHA; Lexico.EOF]
```

Listagem 4.15: nano11.py

```
1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.ID "n"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
4 Lexico.LITINT 1; Lexico.NOVALINHA; Lexico.ID "m"; Lexico.DPONTOS;
5 Lexico.ID "int"; Lexico.IGUAL; Lexico.LITINT 2; Lexico.NOVALINHA;
6 Lexico.ID "x"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
7 Lexico.LITINT 5; Lexico.NOVALINHA; Lexico.WHILE; Lexico.ID "x";
8 Lexico.MAIOR; Lexico.ID "n"; Lexico.DPONTOS; Lexico.NOVALINHA;
9 Lexico.INDENTA; Lexico.ID "n"; Lexico.IGUAL; Lexico.ID "n"; Lexico.SOMA;
10 Lexico.ID "m"; Lexico.NOVALINHA; Lexico.ID "print"; Lexico.APAR;
11 Lexico.ID "n"; Lexico.FPAR; Lexico.NOVALINHA; Lexico.DEDENTA;
12 Lexico.DEDENTA; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR;
13 Lexico.NOVALINHA; Lexico.EOF]
```

Listagem 4.16: nano12.py

```
1 [Lexico.DEF; Lexico.ID "main"; Lexico.APAR; Lexico.FPAR; Lexico.SUB;
2 Lexico.MAIOR; Lexico.NONE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.
  INDENTA;
3 Lexico.ID "n"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
4 Lexico.LITINT 1; Lexico.NOVALINHA; Lexico.ID "m"; Lexico.DPONTOS;
5 Lexico.ID "int"; Lexico.IGUAL; Lexico.LITINT 2; Lexico.NOVALINHA;
6 Lexico.ID "x"; Lexico.DPONTOS; Lexico.ID "int"; Lexico.IGUAL;
7 Lexico.LITINT 5; Lexico.NOVALINHA; Lexico.WHILE; Lexico.ID "x";
8 Lexico.MAIOR; Lexico.ID "n"; Lexico.DPONTOS; Lexico.NOVALINHA;
9 Lexico.ID "n"; Lexico.INDENTA; Lexico.ID "n"; Lexico.IGUAL; Lexico.ID "n";
  Lexico.EQUIVALENTE; Lexico.ID "m";
10 Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.INDENTA; Lexico.ID "print";
11 Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.NOVALINHA; Lexico.DEDENTA
  ;
12 Lexico.ELSE; Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.INDENTA;
13 Lexico.ID "print"; Lexico.APAR; Lexico.LITINT 0; Lexico.FPAR;
```

```
14 Lexico.NOVALINHA; Lexico.DEDENTA; Lexico.ID "x"; Lexico.IGUAL;
15 Lexico.ID "x"; Lexico.SUB; Lexico.LITINT 1; Lexico.NOVALINHA;
16 Lexico.DEDENTA; Lexico.DEDENTA; Lexico.ID "main"; Lexico.APAR; Lexico.
    FPAR;
17 Lexico.NOVALINHA; Lexico.EOF]
```

Capítulo 5

Análise sintática

O processo de ler uma sequência de tokens de entrada e determinar uma estrutura gramatical, definida através de uma gramática, e ainda verificar se esta correta, é uma tarefa para o analisador sintático.

No processo seguinte, o analisador sintático transforma a sequência de tokens em uma árvore sintática.

O analisador sintático preditivo para o reconhecimento da gramática do Python foi construído com o auxílio do Menhir, que é um gerador de parser, que tem suporte para Ocaml.

Para instalar o Menhir digite no terminal:

```
opam menhir
```

5.1 Código

Algumas alterações foram feitas no lexico.mll:

Listagem 5.1: lexico.mll

```
1 {
2   open Lexing
3   open Printf
4   open Sintatico
5
6   exception Erro of string
7
8   let nivel_par = ref 0
9
10  let incr_num_linha lexbuf =
11    let pos = lexbuf.lex_curr_p in
12    lexbuf.lex_curr_p <-
13      { pos with pos_lnum = pos.pos_lnum + 1;
14          pos_bol = pos.pos_cnum
15      }
16
```

```

17  let msg_erro lexbuf c =
18      let pos = lexbuf.lex_curr_p in
19      let lin = pos.pos_lnum
20      and col = pos.pos_cnum - pos.pos_bol - 1 in
21      sprintf "%d-%d: caracter desconhecido %c" lin col c
22
23  }
24
25  let digito = ['0' - '9']
26  let inteiro = '-'? digito+
27  let frac      = '.'digito*
28  let exp       = ['e' 'E'] ['- ' '+']?digito+
29  let float     = digito* frac exp?
30
31  let letra = ['a' - 'z' 'A' - 'Z']
32  let identificador = letra ( letra | digito | '_' ) *
33
34  let restante      = [^ ' ' '\t' '\n' ] [^ '\n']+
35  let brancos       = [ ' ' '\t' ] +
36  let novalinha     = '\r' | '\n' | "\r\n"
37
38  let comentario = "#" [ ^ '\n' ] *
39  let comentario_bloco = "/*" [ ^ "*/" ] *
40  let linha_em_branco = [ ' ' '\t' ] * comentario
41
42  (* O pré processador necessário para contabilizar a indentação *)
43  rule preprocessador indentacao = parse
44      linha_em_branco          { preprocessador 0 lexbuf } (* ignora brancos *)
45  | [ ' ' '\t' ] + '\n'        { incr_num_linha lexbuf;
46                                preprocessador 0 lexbuf } (* ignora brancos *)
47  | ' '                        { preprocessador (indentacao + 1) lexbuf }
48  | '\t'                       { let nova_ind = indentacao + 8 - (indentacao
49                                mod 8)
50                                in preprocessador nova_ind lexbuf }
51  | novalinha                  { incr_num_linha lexbuf;
52                                preprocessador 0 lexbuf }
53  | restante as linha {
54      let rec tokenize lexbuf =
55          let tok = token lexbuf in
56          match tok with
57          | EOF -> []
58          | _ -> tok :: tokenize lexbuf in
59      let toks = tokenize (Lexing.from_string linha) in
60      (* A impressão a seguir serve apenas para depuração. Retirar depois! *)
61      printf "Linha(indentacao=%d,nivel_par=%d)\n" indentacao (!nivel_par);
62      Linha(indentacao,!nivel_par, toks)
63  }
64
65  and token = parse
66  (* PALAVRAS RESERVADAS *)
67      brancos          { token lexbuf }
68  | novalinha          { incr_num_linha lexbuf; token lexbuf }
69  | comentario         { token lexbuf }
70  | comentario_bloco   { comentario_bloco 0 lexbuf }
71  | "not"              { NAO }
72  | "and"              { ANDLOG }
73  | "or"               { ORLOG }

```

```

74 | "def"                { DEF }
75 | "return"            { RETURN }
76 | "while"             { WHILE }
77 | "for"               { FOR }
78 | "in"                { IN }
79 | "range"             { RANGE }
80 | "input"             { INPUT }
81 | "print"             { PRINT }
82 | "str"               { STRING }
83 | "int"               { INTEIRO }
84 | "bool"              { BOOL }
85 | "char"              { CHAR }
86 | "float"             { FLOAT }
87 | "None"              { NONE }
88 | "if"                { IF }
89 | "elif"              { ELIF }
90 | "else"              { ELSE }
91 | "True"              { LITBOOL true }
92 | "False"             { LITBOOL false }
93 (* OPERADORES ARITMETICOS *)
94 (* | "+="            { SOMAIGUAL }
95 | "-="              { SUBIGUAL }
96 | "*="              { MULTIGUAL }
97 | "/="              { DIVIGUAL }
98 | "%="              { MODIGUAL }*)
99 | "++"              { INCREMENTAR }
100 | '+'               { SOMA }
101 | "--"              { MENOS }
102 | '-'               { SUB }
103 | '*'               { MULT }
104 | '/'               { DIV }
105 | '%'               { MOD }
106 (* OPERADORES RELACIONAIS *)
107 | "<="              { MENORIGUAL }
108 | ">="              { MAIORIGUAL }
109 | "=="              { EQUIVALENTE }
110 | "!="              { NAOEQUIVALENTE }
111 | '<'               { MENOR }
112 | '>'               { MAIOR }
113 (* SEPARADORES *)
114 | '('               { incr(nivel_par); APAR }
115 | ')'               { decr(nivel_par); FPAR }
116 | ','               { VIRG }
117 | ':'               { DPONTOS }
118 (* OUTROS *)
119 | '='               { ATRIB }
120 | "->"             { SETA }
121
122 | "'_'_" as s       { let c = String.get s 1 in LITCHAR c }
123 | inteiro as num    { let numero = int_of_string num in LITINT numero }
124 | float as num      { let numero = float_of_string num in LITFLOAT numero }
125 | ""               { let buffer = Buffer.create 1 in
126 |                   let str = leia_string buffer lexbuf in
127 |                   LITSTRING str }
128 | identificador as id { ID (id) }
129 | _                 { raise ( Erro ("Caracter desconhecido: " ^ Lexing.
130 | eof                { EOF }

```

```

131
132 and comentario_bloco n = parse
133     comentario_bloco { token lexbuf }
134     | _ { comentario_bloco n lexbuf }
135     | eof { raise (Erro "Comentário não terminado") }
136
137 and leia_string buffer = parse
138     | '"' { Buffer.contents buffer }
139     | "\\t" { Buffer.add_char buffer '\t'; leia_string buffer lexbuf }
140     | "\\n" { Buffer.add_char buffer '\n'; leia_string buffer lexbuf }
141     | '\\\'' { Buffer.add_char buffer '\''; leia_string buffer lexbuf }
142     | '\\\'' { Buffer.add_char buffer '\\\''; leia_string buffer lexbuf }
143     | _ as c { Buffer.add_char buffer c; leia_string buffer lexbuf }
144     | eof { raise (Erro "A string não foi fechada.") }

```

Código do sintatico.mly:

Listagem 5.2: sintatico.mly

```

1 %{
2 open Ast
3 %}
4 (* literais *)
5 %token <int * int * token list> Linha
6 %token <int> LITINT
7 %token <char> LITCHAR
8 %token <bool> LITBOOL
9 %token <float> LITFLOAT
10 %token <string> LITSTRING
11 %token <string> ID
12 (* palavras reservadas *)
13 %token INTEIRO
14 %token FLOAT
15 %token CHAR
16 %token STRING
17 %token BOOL
18 %token NONE
19 %token DEF
20 %token RETURN
21 %token RANGE
22 %token INPUT
23 %token PRINT
24 %token ATRIB
25 %token EOF
26 %token ORLOG
27 %token ANDLOG
28 %token NAO
29 (* loops *)
30 %token IF
31 %token ELIF
32 %token ELSE
33 %token WHILE
34 %token FOR
35 %token IN
36 (* operadores *)
37 %token SOMA
38 %token MULT
39 %token DIV
40 %token MOD

```

5.1

```
41 %token SUB
42 %token MENOS
43 %token INCREMENTAR
44 (*%token SOMAIGUAL
45 %token SUBIGUAL
46 %token MULTIGUAL
47 %token DIVIGUAL
48 %token MODIGUAL*)
49 (* comparadores *)
50 %token MENOR
51 %token MAIOR
52 %token EQUIVALENTE
53 %token MAIORIGUAL
54 %token MENORIGUAL
55 %token NAOEQUIVALENTE
56 (* simbolos *)
57 %token APAR
58 %token FPAR
59 %token VIRG
60 %token DPONTOS
61 %token SETA
62 (* leiaute *)
63 %token INDENTA
64 %token DEDENTA
65 %token NOVALINHA
66 (* tokens left *)
67 %left ORLOG
68 %left ANDLOG
69 %left EQUIVALENTE
70 %left NAOEQUIVALENTE
71 %left MAIOR
72 %left MENOR
73 %left MAIORIGUAL
74 %left MENORIGUAL
75 %left SOMA
76 %left MENOS
77 %left MULT
78 %left DIV
79 %left MOD
80
81 %start <Ast.programa> programa
82
83 %%
84
85 programa: ins=instrucao*
86           EOF
87           { Programa ins }
88
89 instrucao:
90           func = funcao { func }
91           | cmd = comando { Cmd cmd }
92
93 comandos: cmd = comando+ { cmd }
94
95 funcao:
96         DEF nome=variavel APAR args=separated_list(VIRG, parametro) FPAR SETA
97         retorno=tipo DPONTOS
98         NOVALINHA INDENTA cmd=comandos DEDENTA
99         {
```

```

99         Funcao {
100             fn_nome = nome;
101             fn_tiporet = retorno;
102             fn_formais = args;
103             fn_corpo = cmd
104         }
105     }
106
107 parametro:
108     nome = variavel DPONTOS tp=tipo { (nome, tp) }
109
110 expressoes:
111     exp = separated_list(VIRG, expressao) { exp }
112
113 comando:
114     c=comando_declaracao { c }
115     | c=comando_atribuicao { c }
116     | c=comando_input { c }
117     | c=comando_input_declaracao { c }
118     | c=comando_if { c }
119     | c=comando_if2 { c }
120     | c=comando_while { c }
121     | c=comando_for { c }
122     | c=comando_for_dec { c }
123     | c=comando_print { c }
124     | c=comando_chamada { c }
125     | c=comando_retorno { c }
126     | c=comando_incrementar { c }
127     | c=comando_decrementar { c }
128
129 comando_declaracao:
130     nome = parametro option(ATRIB) exp=option(expressao) NOVALINHA {
131         CmdDeclaracao (nome, exp) }
132
133 comando_atribuicao:
134     nome = variavel ATRIB exp=expressao NOVALINHA { CmdAtrib (nome, exp) }
135
136 comando_input:
137     nome = variavel ATRIB tp = tipo APAR INPUT APAR exp = option(expressao
138         ) FPAR FPAR
139     NOVALINHA { CmdInput (nome, (exp, tp)) }
140
141 comando_input_declaracao:
142     nome=parametro ATRIB tp=tipo APAR INPUT APAR exp=option(expressao)
143     FPAR FPAR
144     NOVALINHA { CmdInputDeclaracao (nome, (exp, tp)) }
145
146 comando_if:
147     IF exp=expressao DPONTOS
148     NOVALINHA INDENTA cmd1=comandos DEDENTA cmd2=option(comando_if2) {
149         CmdIf (exp, cmd1, cmd2) }
150
151 comando_if2:
152     ELSE DPONTOS NOVALINHA INDENTA cmd2=comandos DEDENTA { CmdElse cmd2
153         | ELIF cond1=expressao DPONTOS NOVALINHA INDENTA entao1=
154             comandos DEDENTA cmd1=option(comando_if2) {
155         CmdIf (cond1, entao1, cmd1)
156     }

```

5.1

```
153
154 comando_while:
155     WHILE exp=expressao DPONTOS NOVALINHA INDENTA cmd=comandos DEDENTA {
156         CmdWhile (exp, cmd) }
157
158 comando_for_dec:
159     FOR p=parametro IN RANGE APAR exp1=expressao VIRG exp2=expressao FPAR
160         DPONTOS
161     NOVALINHA INDENTA cmd=comandos DEDENTA { CmdFor_Dec (p, (exp1, exp2),
162         cmd) }
163
164 comando_print:
165     PRINT APAR args=separated_list(VIRG, expressao) FPAR NOVALINHA {
166         CmdPrint args }
167
168 comando_for:
169     FOR v=variavel IN RANGE APAR exp1 = expressao VIRG exp2 = expressao
170         FPAR DPONTOS
171     NOVALINHA INDENTA cmd=comandos DEDENTA { CmdFor (v, (exp1, exp2), cmd)
172         }
173
174 comando_chamada:
175     exp=chamada NOVALINHA { CmdChmd exp }
176
177 comando_incrementar:
178     op=operI exp=expressao { CmdIncr(op, exp) }
179
180 comando_decrementar:
181     op=operM exp=expressao { CmdDecr(op, exp) }
182
183 chamada:
184     nome = variavel APAR args=separated_list(VIRG, expressao) FPAR {
185         ExpChmd (nome, args) }
186
187 comando_retorno: RETURN exp=option(expressao) NOVALINHA { CmdReturn exp }
188
189 tipo:
190     INTEIRO { TipoInt }
191     | STRING { TipoStr }
192     | BOOL { TipoBool }
193     | CHAR { TipoChar }
194     | FLOAT { TipoFloat }
195     | NONE { TipoNone }
196
197 expressao:
198     f=chamada { f }
199     | v=variavel { ExpVar v }
200     | i=LITINT { ExpInt i }
201     | c=LITCHAR { ExpChar c }
202     | f=LITFLOAT { ExpFloat f }
203     | s=LITSTRING { ExpStr s }
204     | b=LITBOOL { ExpBool b }
205     | e1=expressao op=operB e2=expressao { ExpOperB(op,e1,e2) }
206     | APAR exp=expressao FPAR { exp }
207     | e1=expressao op=operC e2=expressao { ExpComp(op,e1,e2) }
208     | n=operN exp=expressao { ExpNot(n,exp) }
209
210 %inline operB:
211     ANDLOG { ANDlog }
```

```

205     | ORLOG { Orlog }
206     | SOMA { Soma }
207     | SUB { Sub }
208     | MULT { Mult }
209     | DIV { Div }
210     | MOD { Mod }
211
212 %inline operC:
213     NAOEQUIVALENTE { NaoEquivalente }
214     | EQUIVALENTE { Equivalente }
215     | MAIOR { Maior }
216     | MENOR { Menor }
217     | MAIORIGUAL { MaiorIgual }
218     | MENORIGUAL { MenorIgual }
219
220 operM:
221     MENOS { Menos }
222
223 operI:
224     INCREMENTAR { Incrementar }
225
226 operN:
227     NAO { Not }
228
229 variavel:
230     | x = ID {VarSimples x}

```

Código da árvore sintática, ast.ml:

Listagem 5.3: ast.ml

```

1 (* The type of the abstract syntax tree (AST). *)
2 type identificador = string
3
4 type programa = Programa of instrucoes
5 and comandos    = comando list
6 and instrucoes = instrucao list
7 and expressoes = expressao list
8 and instrucao  =
9     Funcao of decfn
10    | Cmd    of comando
11
12 and decfn = {
13     fn_nome: variavel;
14     fn_tiporet: tipo;
15     fn_formais: (variavel * tipo) list;
16     fn_corpo: comandos
17 }
18
19 and tipo =
20     TipoInt
21     | TipoStr
22     | TipoBool
23     | TipoChar
24     | TipoFloat
25     | TipoNone
26
27 and comando =
28     CmdDeclaracao of (variavel * tipo) * expressao option

```



```

29 | CmdAtrib  of variavel * expressao
30 | CmdInputDeclaracao of (variavel * tipo) * (expressao option * tipo)
31 | CmdInput  of variavel * (expressao option * tipo)
32 | CmdPrint  of expressao list
33 | CmdIf     of expressao * comandos * (comando option)
34 | CmdElse   of comandos
35 | CmdReturn of expressao option
36 | CmdWhile  of expressao * comandos
37 | CmdFor    of variavel * (expressao * expressao) * comandos
38 | CmdFor_Dec of (variavel * tipo) * (expressao * expressao) * comandos
39 | CmdChmd   of expressao
40 | CmdIncr   of (operador_incr * expressao)
41 | CmdDecr   of (operador_decr * expressao)
42
43 and variaveis = variavel list
44
45 and variavel = VarSimples of identificador
46
47 and expressao =
48     ExpVar of variavel
49 | ExpInt  of int
50 | ExpStr  of string
51 | ExpChar of char
52 | ExpBool of bool
53 | ExpFloat of float
54 | ExpOperB of operador * expressao * expressao
55 | ExpOperU of operador * expressao
56 | ExpChmd  of variavel * expressoes
57 | ExpComp  of (operador_comparacao * expressao * expressao)
58 | ExpNot   of (operador_not * expressao)
59
60 and operador =
61     Soma
62 | Sub
63 | Mult
64 | Div
65 | Mod
66 | ANDlog
67 | Orlog
68
69 and operador_comparacao =
70     Maior
71 | Menor
72 | MaiorIgual
73 | MenorIgual
74 | Equivalente
75 | NaoEquivalente
76
77 and operador_not =
78     Not
79
80 and operador_decr =
81     Menos
82
83 and operador_incr =
84     Incrementar

```

Código do sintático para execução dos arquivos:

Listagem 5.4: sintaticoTest.ml

```

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 Pre_processador.lexico
42     lexbuf in
43   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
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 =

```

5.3

```
57 let lexbuf = Lexing.from_string s in
58 let ast = parse_com_erro lexbuf in
59 ast
60
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"
```

5.2 Execução

Para compilar os arquivos do projeto, primeiro gere:

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

Depois modifique o arquivo sintatico.msg com as suas mensagens de erro. Agora basta gerar o arquivo erroSint.ml que contém as mensagens de erro:

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

Para usar o ocamlbuild para compilar todo o projeto, digite:

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

Entre no Ocaml:

```
rlwrap ocaml
```

Entre com o arquivo e extensao dentro do Ocaml:

```
parse_arq "nome_arquivo.extensao"
```

5.3 Análise sintática dos programas micro

Listagem 5.5: astMicro01

```
1 Ast.programa option =
2 Some
3 (Programa
4   [Funcao
5     {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6     fn_corpo =
7       [CmdDeclaracao ((VarSimples "cel", TipoFloat), Some (ExpFloat 0.));
8       CmdDeclaracao ((VarSimples "far", TipoFloat), Some (ExpFloat 0.));
9       CmdPrint [ExpStr "Tabela de conversao: Celsius -> Fahrenheit\n"];
```

```

10     CmdPrint [ExpStr "Digite a temperatura em Celsius: "];
11     CmdInput (VarSimples "cel", (None, TipoFloat));
12     CmdAtrib (VarSimples "far",
13         ExpOperB (Div,
14             ExpOperB (Soma,
15                 ExpOperB (Mult, ExpFloat 9., ExpVar (VarSimples "cel")),
16                 ExpFloat 160.),
17             ExpFloat 5.));
18     CmdPrint
19         [ExpStr "A nova temperatura e: "; ExpVar (VarSimples "far");
20         ExpStr " F"]];
21     Cmd (CmdChmd (ExpChmd (VarSimples "main", [])))

```

Listagem 5.6: astMicro02

```

1 Ast.programa option =
2 Some
3   (Programa
4     [Funcao
5       {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6       fn_corpo =
7         [CmdPrint [ExpStr "Digite o primeiro numero: "];
8         CmdInputDeclaracao ((VarSimples "num1", TipoInt), (None, TipoInt))
9         ;
10        CmdPrint [ExpStr "Digite o segundo numero: "];
11        CmdInputDeclaracao ((VarSimples "num2", TipoInt), (None, TipoInt))
12        ;
13        CmdIf
14          (ExpComp
15            (Maior, ExpVar (VarSimples "num1"), ExpVar (VarSimples "num2"))
16          ,
17          [CmdPrint
18            [ExpStr "\nO primeiro numero"; ExpVar (VarSimples "num1");
19            ExpStr "e maior que o segundo"; ExpVar (VarSimples "num2")]],
20          Some
21            (CmdElse
22              [CmdPrint
23                [ExpStr "O segundo numero"; ExpVar (VarSimples "num2");
24                ExpStr "e maior que o primeiro"; ExpVar (VarSimples "num1")]
25              ])))]
26        ]))];
27 Cmd (CmdChmd (ExpChmd (VarSimples "main", [])))

```

Listagem 5.7: astMicro03

```

1 Ast.programa option =
2 Some
3   (Programa
4     [Funcao
5       {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6       fn_corpo =
7         [CmdPrint [ExpStr "Digite um numero: "];
8         CmdInputDeclaracao ((VarSimples "numero", TipoInt), (None, TipoInt
9         ));
10        CmdIf
11          (ExpComp (MaiorIgual, ExpVar (VarSimples "numero"), ExpInt 100),
12          [CmdIf
13            (ExpComp (MenorIgual, ExpVar (VarSimples "numero"), ExpInt 200)
14            ,
15            [CmdPrint [ExpStr "\n numero entre 100 e 200"]],

```

5.3

```
14         Some (CmdElse [CmdPrint [ExpStr "\n numero maior que 200"]]))],
15         Some (CmdElse [CmdPrint [ExpStr "\n numero menor que 100"]]))];
16     CmdReturn None];];
17 Cmd (CmdChmd (ExpChmd (VarSimples "main", []))))]
```

Listagem 5.8: astMicro04

```
1 Ast.programa option =
2 Some
3   (Programa
4     [Funcao
5       {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6        fn_corpo =
7          [CmdDeclaracao ((VarSimples "intervalo", TipoInt), Some (ExpInt 0))
8            ;
9            CmdDeclaracao ((VarSimples "x", TipoInt), None);
10            CmdDeclaracao ((VarSimples "num", TipoInt), None);
11            CmdFor (VarSimples "x", (ExpInt 0, ExpInt 5),
12              [CmdPrint [ExpStr "Digite o numero: "];
13                CmdInput (VarSimples "num", (None, TipoInt));
14                CmdIf (ExpComp (MaiorIgual, ExpVar (VarSimples "num"), ExpInt
15                  10),
16                  [CmdIf
17                    (ExpComp (MenorIgual, ExpVar (VarSimples "num"), ExpInt 150),
18                      [CmdAtrib (VarSimples "intervalo",
19                        ExpOperB (Soma, ExpVar (VarSimples "intervalo"), ExpInt 1))
20                      ],
21                    None)],
22                  None)]);
23            CmdPrint
24              [ExpStr "Ao total, foram digitados ";
25                ExpVar (VarSimples "intervalo");
26                ExpStr " numeros no intervalo entre 10 e 150\n"];
27            CmdReturn None];];
28 Cmd (CmdChmd (ExpChmd (VarSimples "main", []))))]
```

Listagem 5.9: astMicro05

```
1 Ast.programa option =
2 Some
3   (Programa
4     [Funcao
5       {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6        fn_corpo =
7          [CmdDeclaracao ((VarSimples "x", TipoInt), None);
8            CmdDeclaracao ((VarSimples "m", TipoInt), Some (ExpInt 0));
9            CmdDeclaracao ((VarSimples "h", TipoInt), Some (ExpInt 0));
10            CmdDeclaracao ((VarSimples "nome", TipoStr), Some (ExpStr ""));
11            CmdDeclaracao ((VarSimples "sexo", TipoStr), Some (ExpStr ""));
12            CmdFor (VarSimples "x", (ExpInt 0, ExpInt 1),
13              [CmdInput (VarSimples "nome",
14                (Some (ExpStr "Digite o nome: "), TipoStr));
15                CmdInput (VarSimples "sexo",
16                  (Some (ExpStr "H - Homem ou M - Mulher: "), TipoStr));
17                CmdIf
18                  (ExpComp (Equivalente, ExpVar (VarSimples "sexo"), ExpChar 'H')
19                    ,
20                  [CmdAtrib (VarSimples "h",
21                    ExpOperB (Soma, ExpVar (VarSimples "h"), ExpInt 1))],
22                ]);
23            ]);
24        ]);
25 Cmd (CmdChmd (ExpChmd (VarSimples "main", []))))]
```

```

21         Some
22         (CmdIf
23         (ExpComp (Equivalente, ExpVar (VarSimples "sexo"), ExpChar '
24             M'),
25         [CmdAtrib (VarSimples "m",
26             ExpOperB (Soma, ExpVar (VarSimples "m"), ExpInt 1)]),
27         Some (CmdElse [CmdPrint [ExpStr "Sexo so pode ser H ou M!\n"
28             ]]])))]);
29 CmdPrint
30 [ExpStr "\nForam inseridos "; ExpVar (VarSimples "h");
31 ExpStr " homens"];
32 CmdPrint
33 [ExpStr "\nForam inseridas "; ExpVar (VarSimples "m");
34 ExpStr " mulheres"];
35 CmdReturn None];
36 Cmd (CmdChmd (ExpChmd (VarSimples "main", [])))]

```

Listagem 5.10: astMicro06

```

1 Ast.programa option =
2 Some
3 (Programa
4   [Funcao
5     {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6     fn_corpo =
7       [CmdInputDeclaracao ((VarSimples "numero", TipoInt),
8         (Some (ExpStr "Digite um numero de 1 a 5: "), TipoInt));
9       CmdIf (ExpComp (Equivalente, ExpVar (VarSimples "numero"), ExpInt
10         1),
11         [CmdPrint [ExpStr "Um"]],
12         Some
13         (CmdIf
14           (ExpComp (Equivalente, ExpVar (VarSimples "numero"), ExpInt 2)
15             ,
16           [CmdPrint [ExpStr "Dois"]],
17           Some
18           (CmdIf
19             (ExpComp (Equivalente, ExpVar (VarSimples "numero"), ExpInt
20               3),
21             [CmdPrint [ExpStr "Tres"]],
22             Some
23             (CmdIf
24               (ExpComp
25                 (Equivalente, ExpVar (VarSimples "numero"), ExpInt 4),
26                 [CmdPrint [ExpStr "Quatro"]],
27                 Some
28                 (CmdIf
29                   (ExpComp
30                     (Equivalente, ExpVar (VarSimples "numero"), ExpInt
31                       5),
32                   [CmdPrint [ExpStr "Cinco"]],
33                   Some (CmdElse [CmdPrint [ExpStr "Numero Invalido!!!"
34                     ]]])))])))]);
35 CmdReturn None];
36 Cmd (CmdChmd (ExpChmd (VarSimples "main", [])))]

```

Listagem 5.11: astMicro07

```

1 Ast.programa option =

```

5.3

```

2 Some
3   (Programa
4     [Funcao
5       {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6         fn_corpo =
7           [CmdDeclaracao ((VarSimples "numero", TipoInt), Some (ExpInt 0));
8             CmdDeclaracao ((VarSimples "programa", TipoInt), Some (ExpInt 1));
9             CmdDeclaracao ((VarSimples "opc", TipoStr), Some (ExpStr ""));
10            CmdWhile
11              (ExpComp (Equivalente, ExpVar (VarSimples "programa"), ExpInt 1),
12                [CmdInput (VarSimples "numero",
13                  (Some (ExpStr "Digite um numero: "), TipoInt));
14                  CmdIf (ExpComp (Maior, ExpVar (VarSimples "numero"), ExpInt 0),
15                    [CmdPrint [ExpStr "Positivo"]],
16                      Some
17                        (CmdElse
18                          [CmdIf
19                            (ExpComp
20                              (Equivalente, ExpVar (VarSimples "numero"), ExpInt 0),
21                              [CmdPrint [ExpStr "O numero e igual a 0"]],
22                                Some
23                                  (CmdIf
24                                    (ExpComp (Menor, ExpVar (VarSimples "numero"), ExpInt
25                                      0),
26                                      [CmdPrint [ExpStr "Negativo"]], None)))]));
27                            CmdInput (VarSimples "opc",
28                              (Some (ExpStr "Deseja Finalizar? (S/N) :"), TipoStr));
29                            CmdIf
30                              (ExpComp (Equivalente, ExpVar (VarSimples "opc"), ExpStr "S"),
31                                [CmdAtrib (VarSimples "programa", ExpInt 0)], None)]);
32                            CmdReturn None]);
33                          Cmd (CmdChmd (ExpChmd (VarSimples "main", [])))]))

```

Listagem 5.12: astMicro08

```

1 Ast.programa option =
2 Some
3   (Programa
4     [Funcao
5       {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6         fn_corpo =
7           [CmdDeclaracao ((VarSimples "numero", TipoInt), Some (ExpInt 1));
8             CmdWhile
9               (ExpOperB (Orlog,
10                 ExpComp (Menor, ExpVar (VarSimples "numero"), ExpInt 0),
11                 ExpComp (Maior, ExpVar (VarSimples "numero"), ExpInt 0)),
12               [CmdInput (VarSimples "numero",
13                 (Some (ExpStr "Digite um numero: "), TipoInt));
14                 CmdIf (ExpComp (Maior, ExpVar (VarSimples "numero"), ExpInt 10),
15                   [CmdPrint [ExpStr "Numero maior que 10"]],
16                     Some (CmdElse [CmdPrint [ExpStr "Numero menor que 10"])]));
17                 CmdReturn None]);
18             Cmd (CmdChmd (ExpChmd (VarSimples "main", [])))]))

```

Listagem 5.13: astMicro09

```

1 Ast.programa option =
2 Some
3   (Programa

```

```

4  [Funcao
5      {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6      fn_corpo =
7          [CmdDeclaracao ((VarSimples "novopreco", TipoFloat), None);
8          CmdInputDeclaracao ((VarSimples "preco", TipoFloat),
9              (Some (ExpStr "Digite o preco: "), TipoFloat));
10         CmdInput (VarSimples "venda",
11             (Some (ExpStr "Digite a venda: "), TipoFloat));
12         CmdIf
13             (ExpOperB (Orlog,
14                 ExpComp (Menor, ExpVar (VarSimples "venda"), ExpInt 500),
15                 ExpComp (Menor, ExpVar (VarSimples "preco"), ExpInt 30)),
16             [CmdAtrib (VarSimples "novopreco",
17                 ExpOperB (Soma, ExpVar (VarSimples "preco"),
18                     ExpOperB (Mult, ExpOperB (Div, ExpInt 10, ExpInt 100),
19                         ExpVar (VarSimples "preco")))]],
20         Some
21             (CmdIf
22                 (ExpOperB (Orlog,
23                     ExpOperB (ANDlog,
24                         ExpComp (MaiorIgual, ExpVar (VarSimples "venda"), ExpInt
25                             500),
26                         ExpComp (Menor, ExpVar (VarSimples "venda"), ExpInt 1200)),
27                     ExpOperB (ANDlog,
28                         ExpComp (MaiorIgual, ExpVar (VarSimples "preco"), ExpInt
29                             30),
30                         ExpComp (Menor, ExpVar (VarSimples "preco"), ExpInt 80))),
31                 [CmdAtrib (VarSimples "novopreco",
32                     ExpOperB (Soma, ExpVar (VarSimples "preco"),
33                         ExpOperB (Mult, ExpOperB (Div, ExpInt 15, ExpInt 100),
34                             ExpVar (VarSimples "preco")))]],
35             Some
36                 (CmdIf
37                     (ExpOperB (Orlog,
38                         ExpComp
39                             (MaiorIgual, ExpVar (VarSimples "venda"), ExpInt 1200),
40                         ExpComp (MaiorIgual, ExpVar (VarSimples "preco"), ExpInt
41                             80)),
42                     [CmdAtrib (VarSimples "novopreco",
43                         ExpOperB (Sub, ExpVar (VarSimples "preco"),
44                             ExpOperB (Mult, ExpOperB (Div, ExpInt 20, ExpInt 100),
45                                 ExpVar (VarSimples "preco")))]],
46                     None)))]],
47         CmdPrint [ExpStr "O novo preco e: "; ExpVar (VarSimples "novopreco"
48             )];
49         CmdReturn None]};
50  Cmd (CmdChmd (ExpChmd (VarSimples "main", []))))

```

Listagem 5.14: astMicro10

```

1  Ast.programa option =
2  Some
3      (Programa
4          [Funcao
5              {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6              fn_corpo =
7                  [CmdInputDeclaracao ((VarSimples "numero", TipoInt),
8                      (Some (ExpStr "Digite um numero: "), TipoInt));
9                  CmdDeclaracao ((VarSimples "fat", TipoInt),

```



```

10         Some
11         (ExpChmd (VarSimples "fatorial", [ExpVar (VarSimples "numero")])
12         );
13         CmdPrint [ExpStr "O fatorial eh"; ExpVar (VarSimples "fat")];
14         CmdReturn None];];
15     Funcao
16     {fn_nome = VarSimples "fatorial"; fn_tiporet = TipoInt;
17     fn_formais = [(VarSimples "n", TipoInt)];
18     fn_corpo =
19     [CmdIf (ExpComp (MenorIgual, ExpVar (VarSimples "n"), ExpInt 0),
20     [CmdReturn (Some (ExpInt 1))],
21     Some
22     (CmdElse
23     [CmdReturn
24     (Some
25     (ExpOperB (Mult, ExpVar (VarSimples "n"),
26     ExpChmd (VarSimples "fatorial",
27     [ExpOperB (Sub, ExpVar (VarSimples "n"), ExpInt 1)])))]
28     )))]];];
29     Cmd (CmdChmd (ExpChmd (VarSimples "main", [])))]

```

Listagem 5.15: astMicro11

```

1 Ast.programa option =
2 Some
3 (Programa
4 [Funcao
5 {fn_nome = VarSimples "main"; fn_tiporet = TipoNone; fn_formais = [];
6 fn_corpo =
7 [CmdInputDeclaracao ((VarSimples "numero", TipoInt),
8 (Some (ExpStr "Digite um numero: "), TipoInt));
9 CmdDeclaracao ((VarSimples "x", TipoInt),
10 Some
11 (ExpChmd (VarSimples "verifica", [ExpVar (VarSimples "numero")])
12 ));
13 CmdIf (ExpComp (Equivalente, ExpVar (VarSimples "x"), ExpInt 1),
14 [CmdPrint [ExpStr "Positivo"]],
15 Some
16 (CmdIf (ExpComp (Equivalente, ExpVar (VarSimples "x"), ExpInt 0)
17 ,
18 [CmdPrint [ExpStr "zero"]],
19 Some (CmdElse [CmdPrint [ExpStr "Negativo"]]))));
20 CmdReturn None];];
21 Funcao
22 {fn_nome = VarSimples "verifica"; fn_tiporet = TipoInt;
23 fn_formais = [(VarSimples "n", TipoInt)];
24 fn_corpo =
25 [CmdDeclaracao ((VarSimples "res", TipoInt), None);
26 CmdIf (ExpComp (Maior, ExpVar (VarSimples "n"), ExpInt 0),
27 [CmdAtrib (VarSimples "res", ExpInt 1)],
28 Some
29 (CmdIf (ExpComp (Menor, ExpVar (VarSimples "n"), ExpInt 0),
30 [CmdAtrib (VarSimples "res", ExpInt (-1))],
31 Some (CmdElse [CmdAtrib (VarSimples "res", ExpInt 0)]))));
32 CmdReturn (Some (ExpVar (VarSimples "res")))]];];
33 Cmd (CmdChmd (ExpChmd (VarSimples "main", [])))]

```

Capítulo 6

Referências

Python

Dalvik

Smali

OCaml