UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Ronistone Junior - 11521BCC018

Relatório Parcial 3 - Construção de Compiladores

March 2019

Sumário

0.1 Instalação

0.1.1 Instalando o OCaml

Utilizando os comandos:

```
1 sudo apt install wget
2 sudo apt install git aspcud m4 mercurial darcs
3 wget https://raw.github.com/ocaml/opam/master/shell/opam_installer.sh -O -
    | sh -s /usr/local/bin
4 opam init
5 opam update
6 eval `opam config env`
7 sudo apt-get install rlwrap
```

foi instalado o OPAM, rlwrap e o OCaml, com todas as suas dependências

0.1.2 Instalando JVM e Assembler

- Como já possuía java instalado não foi necessário instalação da JVM, estou utilizando o Java 8.
- Já para instalar o jasmin, que é o assembler da JVM, baixei o source no seguinte link Após o download temos um arquivo zip que contém o jar do jasmin.
- Foi utilizado também o disassembler ClassFileAnalyzer (Can) que pode ser baixado no seguinte link
- Para instalar o Assembler e o Disassembler extrai os dois .zip para o meu /opt utilizando unzip nome-do-arquivo.zip e criei um script no /usr/bin para cara um para executa-los, abaixo estão os dois scripts:

```
1 # /usr/bin/run-jasmin
2 java -jar /opt/jasmin-2.4/jasmin.jar $@
```

```
1 # /usr/bin/generate-dot-j
2 java -jar /opt/classfileanalyzer/classfileanalyzer.jar $@
```

0.2 Gerando e Compilando o Assembly Dos Arquivos

Todos os códigos utilizados foram disponibilizados pelo professor no forum do google groups da matéria, utilizaremos os arquivos "Nanos", que podem ser baixados neste link.

Aqui vamos gerar os arquivos .j que é utilizado pelo Jasmin para gerar como estamos utilizando um disassembler para gerar o assembly de forma mais rápida, seguiremos o seguinte processo:

1. Compilaremos um arquivo .java utilizando o javac
2. Depois de compilado teremos o .class, usaremos ele no disassembler para gerar o .j
3. Agora podemos manipular o assembly da forma que for necessária e ao fim utilizar o Jasmin para compilar o nosso assembly

0.2.1 Compilando

Para compilar entrar na pasta que se encontra os arquivos .java e executar:

```
1       javac *.java
```

0.2.2 Desassemblando

Para desassemblar e gerar os arquivos .j para cada código java, utilizaremos o disassembler ClassFileAnalyzer para isto criei um pequeno script para realizar este processo:

```
1       # ./generate.sh
2       files=`ls *.class`
3
4       for i in $files
5       do
6               name=$( echo "$i" | cut -f 1 -d '.' )
7               echo 'Generating ' ${name} ' ...'
8               generate-dot-j $i > $name.j
9       done;
```

0.2.3 Compilando o Assembly

Após geramos os arquivos .j temos o nosso assembly gerado, caso queiramos compilar este assembly basta utilizarmos o nosso Assembler Jasmin, para realizar este processo criei o script abaixo que irá compilar todos os nossos assembly:

```
1 # ./compile.sh
2 files=`ls *.j`
3
4 for i in $files
5 do
6  name=$( echo "$i" | cut -f 1 -d '.' )
7  echo 'Compiling ' ${name} '...'
8  run-jasmin $i
9 done;
```

0.3 Assembly Gerado

Vamos agora entender um pouco do nosso assembly gerado.

0.3.1 Nano01

No Nano01 tem uma função main que não executa nenhum operação

```
1  ; Nano01.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano01.java
12 .class public Nano01
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 0
26   .limit locals 1
27   .line 3
28   0: return
29 .end method
```

0.3.1.1 Comentários

Aqui temos nosso primeiro assembly, a primeira coisa que podemos notar é que para escrever comentários basta utilizar o ";"(ponto e virgula) e tudo que estiver na frente do ";"faz parte do comentário.

0.3.1.2 Diretivas

Logo depois notamos algumas palavras iniciadas com ".", essas são as chamadas Diretivas. Elas nada mais são que palavras utilizadas para controle pelo Jasmin, ou seja, elas não são instruções da JVM. Podemos ver as seguintes diretivas: .bytecode, .source, .class, .super, .method, .limit, .line, .end. As diretivas estão melhor explicadas na seção [0.4]

0.3.1.3 Métodos

Podemos notar dois métodos, sendo o primeiro o init que é o construtor da classe e o método main.

Dentro do construtor podemos ver a chamada o construtor da classe Object que é uma super classe desta nossa Classe Nano

Dentro do main podemos ver que não é feito quase nada, apenas é definido o tamanho inicial da pilha, o tamanho das variáveis locais e logo em seguida é realizado um return finalizando o método.

0.3.2 Nano02

Declaramos apenas uma variável dentro da nossa classe main.

```
1  ; Nano02.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano02.java
12 .class public Nano02
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 0
26   .limit locals 2
27   .line 4
28   0: return
29 .end method
```

Podemos notar que o este assembly é praticamente identico ao assembly do Nano01 [0.3.1]. Diferindo apenas no .limit locals da função main, podemos ver que agora temos

.limit locals 2 e antes tinhamos .limit locals 1, isto nos mostra que temos mais uma variável local.

0.3.3 Nano03

Declaramos e definimos um valor para uma variável.

```
1  ; Nano03.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano03.java
12 .class public Nano03
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 1
26   .limit locals 2
27   .line 4
28   0: iconst_1
29   1: istore_1
30   .line 5
31   2: return
32 .end method
```

Aqui podemos ver um limite maior para nossa pilha, e duas novas instruções, o limite maior se dá devido a definição de um valor para a variável. Já as duas novas instruções são iconst_1 carrega a constante 1 no topo da pilha. Enquanto a instrução istore_1 carrega o valor do topo da pilha para a variável 1.

0.3.4 Nano04

Declarando uma variável e atribuindo a soma de duas constantes a esta variável.

```
1  ; Nano04.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano04.java
12 .class public Nano04
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 1
26   .limit locals 2
27   .line 4
28   0: iconst_3
29   1: istore_1
30   .line 5
31   2: return
32 .end method
```

Neste exemplo podemos ver que não há apenas uma diferença com o Nano03 0.3.3 que é o valor da constante agora é 3, podemos concluir disto que ouve uma otimização, pois como estávamos somando duas constantes o compilador carregou o resultado da soma direto para não precisar refazer o calculo em tempo de execução.

0.3.5 Nano05

Declarando, atribuindo valor a uma variável e imprimindo o seu valor.

```
1  ; Nano05.j
2
```

```
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano05.java
12 .class public Nano05
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 2
27   .line 4
28   0: iconst_2
29   1: istore_1
30   .line 5
31   2: getstatic java/lang/System/out Ljava/io/PrintStream;
32   5: iload_1
33   6: invokevirtual java/io/PrintStream/println(I)V
34   .line 6
35   9: return
36 .end method
```

Aqui notamos 3 novas instruções, o getstatic que carrega um campo estático para o topo da pilha, o iload_1 que carrega o valor da variável 1 para o topo da pilha e o invokevirtual que realiza a chamada de um método.

0.3.6 Nano06

Declarando variável, atribuindo uma subtração de duas constantes para a variável e por fim imprimindo a variável.

```
1  ; Nano06.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
```

```
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano06.java
12 .class public Nano06
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 2
27   .line 4
28   0: iconst_m1
29   1: istore_1
30   .line 5
31   2: getstatic java/lang/System/out Ljava/io/PrintStream;
32   5: iload_1
33   6: invokevirtual java/io/PrintStream/println(I)V
34   .line 6
35   9: return
36 .end method
```

Podemos notar aqui que ouve a mesma otimização citada no Nano04 0.3.4, sendo utilizado o iconst_m1 para carregar a constante -1 para a pilha. No mais o assembly é identico ao assembly do 0.3.5.

0.3.7 Nano07

Declarando uma variável, atribuindo valor e verificando se o valor é igual a 1, se for imprime a variável.

```
1  ; Nano07.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
```

```
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano07.java
12 .class public Nano07
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 2
27   .line 4
28   0: iconst_1
29   1: istore_1
30   .line 5
31   2: iload_1
32   3: iconst_1
33   4: if_icmpne Label14
34   .line 6
35   7: getstatic java/lang/System/out Ljava/io/PrintStream;
36   10: iload_1
37   11: invokevirtual java/io/PrintStream/println(I)V
38 Label14:
39   .line 8
40   14: return
41   ; append_frame (frameNumber = 0)
42   ; frame_type = 252, offset_delta = 14
43   ; frame bytes: 252 0 14 1
44   .stack
45     offset 14
46     locals Integer
47     .end stack
48 .end method
```

Agora temos uma nova instrução, o if_icmpne verifica se os dois valores do topo da pilha não são iguais se não forem, pula para o Label14. Para a execução desta instrução podemos ver um preparo para tal, onde é carregado a variável 1 e a constante 1 para o topo da pilha, fazendo então o if_icmpne que corresponde a 'if(n==1)'.

0.3.8 Nano08

Verifica se valor na variável é igual a 1 se for imprime a variável, se não imprime 0.

```
1  ; Nano08.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano08.java
12 .class public Nano08
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 2
27   .line 4
28   0: iconst_1
29   1: istore_1
30   .line 5
31   2: iload_1
32   3: iconst_1
33   4: if_icmpne Label17
34   .line 6
35   7: getstatic java/lang/System/out Ljava/io/PrintStream;
36   10: iload_1
37   11: invokevirtual java/io/PrintStream/println(I)V
38   14: goto Label24
39 Label17:
40   .line 8
41   17: getstatic java/lang/System/out Ljava/io/PrintStream;
42   20: iconst_0
43   21: invokevirtual java/io/PrintStream/println(I)V
44 Label24:
```

```
45    .line 10
46    24: return
47    ; append_frame (frameNumber = 0)
48    ; frame_type = 252, offset_delta = 17
49    ; frame bytes: 252 0 17 1
50    .stack
51      offset 17
52      locals Integer
53      .end stack
54    ; same_frame (frameNumber = 1)
55    ; frame_type = 6, offset_delta = 6
56    ; frame bytes: 6
57    .stack
58      offset 24
59      locals Integer
60      .end stack
61  .end method
```

Aqui temos apenas mais uma instrução nova, o goto, ele simplesmente salta para o um label. Assim podemos ver o if feito no Nano07 0.3.7 com a diferença de mais um bloco de instruções e no primeiro bloco que será executado se o if_icmpne falhar temos o goto para pular para o fim do if, enquanto o if_icmpne salta direto para o bloco de código do else.

0.3.9 Nano09

Atribui o valor de uma expressão a variável e depois verifica o valor com um if e else.

```
1  ; Nano09.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano09.java
12 .class public Nano09
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
```

```
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 2
27   .line 5
28   0: iconst_1
29   1: istore_1
30   .line 6
31   2: iload_1
32   3: iconst_1
33   4: if_icmpne Label17
34   .line 7
35   7: getstatic java/lang/System/out Ljava/io/PrintStream;
36   10: iload_1
37   11: invokevirtual java/io/PrintStream/println(I)V
38   14: goto Label24
39 Label17:
40   .line 9
41   17: getstatic java/lang/System/out Ljava/io/PrintStream;
42   20: iconst_0
43   21: invokevirtual java/io/PrintStream/println(I)V
44 Label24:
45   .line 11
46   24: return
47   ; append_frame (frameNumber = 0)
48   ; frame_type = 252, offset_delta = 17
49   ; frame bytes: 252 0 17 1
50   .stack
51     offset 17
52     locals Integer
53     .end stack
54   ; same_frame (frameNumber = 1)
55   ; frame_type = 6, offset_delta = 6
56   ; frame bytes: 6
57   .stack
58     offset 24
59     locals Integer
60     .end stack
61 .end method
```

Vemos um código idêntico ao Nano08 [0.3.8] devido a otimização do compilador resolvendo a expressão em tempo de compilação.

0.3.10 Nano10

Declara duas variáveis, atribui valores para as variáveis e verifica se são iguais

```
; Nano10.j

; Generated by ClassFileAnalyzer (Can)
; Analyzer and Disassembler for Java class files
; (Jasmin syntax 2, http://jasmin.sourceforge.net)
;
; ClassFileAnalyzer, version 0.7.0


.bytecode 52.0
.source Nano10.java
.class public Nano10
.super java/lang/Object

.method public <init>()V
  .limit stack 1
  .limit locals 1
  .line 1
  0: aload_0
  1: invokespecial java/lang/Object/<init>()V
  4: return
.end method

.method public static main([Ljava/lang/String;)V
  .limit stack 2
  .limit locals 3
  .line 4
  0: iconst_1
  1: istore_1
  .line 5
  2: iconst_2
  3: istore_2
  .line 6
  4: iload_1
  5: iload_2
  6: if_icmpne Label19
  .line 7
  9: getstatic java/lang/System/out Ljava/io/PrintStream;
  12: iload_1
  13: invokevirtual java/io/PrintStream/println(I)V
  16: goto Label26
Label19:
  .line 9
  19: getstatic java/lang/System/out Ljava/io/PrintStream;
  22: iconst_0
  23: invokevirtual java/io/PrintStream/println(I)V
Label26:
```

```
48    .line 11
49    26: return
50    ; append_frame (frameNumber = 0)
51    ; frame_type = 253, offset_delta = 19
52    ; frame bytes: 253 0 19 1 1
53    .stack
54      offset 19
55      locals Integer
56      locals Integer
57      .end stack
58    ; same_frame (frameNumber = 1)
59    ; frame_type = 6, offset_delta = 6
60    ; frame bytes: 6
61    .stack
62      offset 26
63      locals Integer
64      locals Integer
65      .end stack
66  .end method
```

Aqui não temos nada novo, a única diferença para o que já foi visto é que agora a comparação é feita utilizando apenas variáveis, ou seja, antes de executar o if_icmpne carregamos as variáveis 1 e 2 para a pilha ao invés de carregar uma variável e uma constante.

0.3.11 Nano11

Declaração de três variáveis e um while verificando se x ¿, n, realizando n = n + m e imprimindo o valor de n a cada interação.

```
1  ; Nano11.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano11.java
12 .class public Nano11
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
```

```
18    .line 1
19    0: aload_0
20    1: invokespecial java/lang/Object/<init>()V
21    4: return
22  .end method
23
24  .method public static main([Ljava/lang/String;)V
25    .limit stack 2
26    .limit locals 4
27    .line 5
28    0: iconst_1
29    1: istore_1
30    .line 6
31    2: iconst_2
32    3: istore_2
33    .line 7
34    4: iconst_5
35    5: istore_3
36  Label6:
37    .line 8
38    6: iload_3
39    7: iload_1
40    8: if_icmple Label25
41    .line 9
42    11: iload_1
43    12: iload_2
44    13: iadd
45    14: istore_1
46    .line 10
47    15: getstatic java/lang/System/out Ljava/io/PrintStream;
48    18: iload_1
49    19: invokevirtual java/io/PrintStream/println(I)V
50    22: goto Label6
51  Label25:
52    .line 12
53    25: return
54    ; append_frame (frameNumber = 0)
55    ; frame_type = 254, offset_delta = 6
56    ; frame bytes: 254 0 6 1 1 1
57    .stack
58      offset 6
59      locals Integer
60      locals Integer
61      locals Integer
62      .end stack
63    ; same_frame (frameNumber = 1)
64    ; frame_type = 18, offset_delta = 18
```

```
65    ; frame bytes: 18
66    .stack
67      offset 25
68      locals Integer
69      locals Integer
70      locals Integer
71      .end stack
72  .end method
```

Agora temos uma estrutura muito similar ao que é realizado nos códigos anteriores onde tinhamos um if, com o while(x¿n) foi gerado um if_icmple que verifica se o primeiro valor é menor ou equal ao segundo, se for pula para um label. Isto é utilizado para marcar o fim do while, caso seja verdadeiro ele pula para o fim do bloco do while, se não ele apenas executa o bloco.

Dentro do bloco vemos uma n = n + m, que é feito da seguinte forma, carrega-se n e m para a pilha depois executamos iadd para somar os dois valores do topo da pilha e salvamos o resultado no topo da pilha, por fim armazenamos o resultado que esta no topo da pilha na variável 1, realizamos a impressão do valor e pulamos de volta para o label que marca a inicialização para o if_icmple.

0.3.12 Nano12

Realiza dentro do while do Nano11 [0.3.11] agora uma verificação se n==m se for imprime n senão imprime 0, e por fim decrementa x.

```
1   ; Nano12.j
2
3   ; Generated by ClassFileAnalyzer (Can)
4   ; Analyzer and Disassembler for Java class files
5   ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6   ;
7   ; ClassFileAnalyzer, version 0.7.0
8
9
10  .bytecode 52.0
11  .source Nano12.java
12  .class public Nano12
13  .super java/lang/Object
14
15  .method public <init>()V
16    .limit stack 1
17    .limit locals 1
18    .line 1
19    0: aload_0
20    1: invokespecial java/lang/Object/<init>()V
21    4: return
```

```
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 4
27   .line 4
28   0: iconst_1
29   1: istore_1
30   .line 5
31   2: iconst_2
32   3: istore_2
33   .line 6
34   4: iconst_5
35   5: istore_3
36 Label6:
37   .line 7
38   6: iload_3
39   7: iload_1
40   8: if_icmple Label40
41   .line 8
42   11: iload_1
43   12: iload_2
44   13: if_icmpne Label26
45   .line 9
46   16: getstatic java/lang/System/out Ljava/io/PrintStream;
47   19: iload_1
48   20: invokevirtual java/io/PrintStream/println(I)V
49   23: goto Label33
50 Label26:
51   .line 11
52   26: getstatic java/lang/System/out Ljava/io/PrintStream;
53   29: iconst_0
54   30: invokevirtual java/io/PrintStream/println(I)V
55 Label33:
56   .line 12
57   33: iload_3
58   34: iconst_1
59   35: isub
60   36: istore_3
61   37: goto Label6
62 Label40:
63   .line 14
64   40: return
65   ; append_frame (frameNumber = 0)
66   ; frame_type = 254, offset_delta = 6
67   ; frame bytes: 254 0 6 1 1 1
68   .stack
```

```
69      offset 6
70      locals Integer
71      locals Integer
72      locals Integer
73      .end stack
74    ; same_frame (frameNumber = 1)
75    ; frame_type = 19, offset_delta = 19
76    ; frame bytes: 19
77    .stack
78      offset 26
79      locals Integer
80      locals Integer
81      locals Integer
82      .end stack
83    ; same_frame (frameNumber = 2)
84    ; frame_type = 6, offset_delta = 6
85    ; frame bytes: 6
86    .stack
87      offset 33
88      locals Integer
89      locals Integer
90      locals Integer
91      .end stack
92    ; same_frame (frameNumber = 3)
93    ; frame_type = 6, offset_delta = 6
94    ; frame bytes: 6
95    .stack
96      offset 40
97      locals Integer
98      locals Integer
99      locals Integer
100     .end stack
101 .end method
```

Aqui temos um código muito parecido com o Nano11 [0.3.11], podemos ver uma if ser realizado dentro do while e ao fim desse if é realizado uma subtração, onde carregamos o valor da variável 3 e a constante 1 para o topo da pilha, depois executamos isub para realizar a subtração dos dois valores do topo da pilha armazenando o resultado no topo da pilha e por fim usamos istore para armazenar o valor do topo da pilha na variável 3.

0.4 Principais Instruções do Jasmin

Abaixo estão listadas as principais instruções da sintaxe do Jasmin

0.4.1 .bytecode

Indica a versão de bytecode, não é necessário utiliza-la

0.4.2 .source

Indica o nome do arquivo source original.

0.4.3 .class

Utilizado para declarar uma classe

0.4.4 .super

Utilizado para declarar uma herança.

0.4.5 .method

Utilizado para declarar um método.

0.4.6 .limit

Utilizado para definir o tamanho da pilha ou o tamanho da variavel local

0.4.7 .end

Indica o final (p.e.: uma Classe ou um método)

0.4.8 .line

Utilizado para marcar a linha do código fonte, não é necessário, mas ajuda no entendimento do assembly.

0.4.9 lload

Carrega o valor de uma variável local para o topo da pilha.

0.4.10 lstore

Carrega o valor do topo da pilha para a variável determinada pelo índice passado como parâmetro.

0.4.11 iaload

Carrega um valor int de um vetor. O valor da referência do vetor e o índice são retirados do topo da pilha. O valor recuperado do vetor é colocado topo da pilha.

0.4.12 iload

Carrega o valor de uma variável local que é um inteiro para o topo da pilha.

0.4.13 istore

Carrega o valor do topo da pilha para a variável.

0.4.14 fload

Carrega o valor de uma variável local que é um float para o topo da pilha.

0.4.15 fstore

Carrega o valor do topo da pilha que é um float para a variável.

0.4.16 ldc

Desempilha a constante da pilha.

0.4.17 dup

A palavra no topo da pilha é duplicado.

0.4.18 pop

Retira o valor que esta no topo da pilha.

0.4.19 swap

Troca dois operandos da pilha (uma troca, o primeiro vira o segundo e o segundo vira o primeiro).

0.4.20 iadd

Adiciona dois inteiros.

0.4.21 idiv

Divide dois inteiros.

0.4.22 imul

Multiplica dois inteiros.

0.4.23 isub

Subtrai dois inteiros.

0.4.24 fadd

Adiciona dois float.

0.4.25 fdiv

Divide dois float.

0.4.26 fmul

Multiplica dois float.

0.4.27 fsub

Subtrai dois float.

0.4.28 goto

Ir para o marcador.

0.4.29 ifeq

Ir para o rótulo (marcador), se o valor no topo da pilha é 0.

0.4.30 ifge

Ir para o rótulo, se o valor no topo da pilha é igual ou superior (maior) a 0.

0.4.31 ifgt

Ir para o rótulo, se o valor no topo da pilha é superior (maior) a 0.

0.4.32 ifle

Ir para o rótulo, se o valor no topo da pilha é inferior (menor) ou igual a 0.

0.4.33 iflt

Ir para o rótulo, se o valor no topo da pilha é inferior (menor) a 0.

0.4.34 ifne

Ir para o rótulo, se o valor no topo da pilha não é igual a 0.

0.4.35 if_icmple

Ir para o rótulo, se o primeiro valor da pilha for menor ou igual ao segundo valor da pilha.

0.4.36 if_icmpne

Ir para o rótulo, se o primeiro valor da pilha for diferente do segundo valor da pilha.

0.4.37 if_icmpe

Ir para o rótulo, se o primeiro valor da pilha for diferente do segundo valor da pilha.

0.4.38 iand

AND (inteiros).

0.4.39 ior

OR (inteiros).

0.4.40 i2f

Converte inteiro para float.

0.4.41 f2i

Converte float para inteiro.

0.4.42 jsr

Retorna o endereço da pilha e ”pula”para subrotina indicada.

0.4.43 ret

Retorna o endereço da subrotina que esta armazenado a variável local.

0.4.44 invokevirtual

É a forma padrão para a chamada de um método.

0.4.45 invokeinterfacel

Realiza a chamada de um método de uma interface procurando o método implementado por um objeto particular de um ambiente de execução.

0.4.46 invokespecial

Realiza a chamada de métodos especiais como métodos privados, ou métodos da super classe.

0.4.47 invostatic

Faz a chamada de métodos de classe (static).

0.4.48 getstatic

Carrega um campo estático para o topo da pilha.

0.5 Analisador Léxico

O analisador léxico é quem realiza o primeiro processamento sobre o código, ele remove todos os espaços em branco e os comentários, além de produzir uma sequência de palavras chaves através do código, que é denominado de token.

0.5.1 Implementação

Para a implementação do analisador léxico para o portugol, foi utilizado uma adaptação dos códigos gerados em aula.

O código abaixou é o código final do 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
26 let digito = ['0' - '9']
27 let inteiro = digito+
28
29 let letra = ['a' - 'z' 'A' - 'Z']
30 let identificador = ('_' | letra) ( letra | digito | '_')*
31
32 let brancos = [' ' '\t']+
33 let novalinha = '\r' | '\n' | "\r\n"
34
35 let comentario = "//" [^ '\r' '\n' ]*
36
37 rule read = parse
38   brancos    { read lexbuf }
39 | novalinha  { incr_num_linha lexbuf; read lexbuf }
40 | comentario { read lexbuf }
41 | "/*"       { comentario_bloco 0 lexbuf }
42 | '('        { APAR }
43 | '+'        { MAIS }
44 | '-'        { MENOS }
45 | '/'        { DIV }
46 | '*'        { MULTI }
47 | '%'        { RESTO }
48 | '^'        { POTEN }
49 | ':'        { DEF }
50 | ')'        { FPAR }
51 | "<-"       { ATRIB }
52 | "var"      { VAR }
53 | "="        { EQUALS }
```

```
54  | "<>"          { DIFER }
55  | ">="          { GTE }
56  | "<="          { LTE }
57  | '>'           { GT }
58  | '<'           { LT }
59  (* | '.'          { DOT } *)
60  | ','           { VIRG }
61  | inteiro as num { let numero = int_of_string num in
62                     LITINT numero  }
63  | "e"           { AND }
64  | "E"           { AND }
65  | "ou"          { OR }
66  | "OU"          { OR }
67  | "se"          { IF }
68  | "ent o"       { ENTAO }
69  | "sen o"       { ELSE }
70  | "fim_se"      { FIF }
71  | "in cio"      { INI }
72  | "fa a"        { DO }
73  | "fim_para"    { FFOR }
74  | "para"        { FOR }
75  | "de"          { BEGIN }
76  | "at "         { END }
77  | "fim_enquanto" { FWHILE }
78  | "enquanto"    { WHILE }
79  | "escolha"     { SWITCH }
80  | "fim_escolha" { FSWITCH }
81  | "caso"        { CASE }
82  | "outrocaso"   { CASE_DEFAULT }
83  | "algoritmo"   { PROGRAMA }
84  | "fim_algoritmo" { FPROGRAMA }
85  | "escreva"     { SAIDA }
86  | "escreval"    { SAIDA_LINHA }
87  | "leia"        { ENTRADA }
88  | "fim_fun  o"      { FFUNCTION }
89  | "retorne"     { RETURN }
90  | "fun  o"      { FUNCTION }
91  | "inteiro"     { INTEGER }
92  | "real"        { REAL }
93  | "caractere"   { CARACTER }
94  | "l gico"      { BOOL }
95  | "verdadeiro"  { LITBOOL(true) }
96  | "falso"       { LITBOOL(false) }
97  | identificador as id { ID id }
98  | "'" _ "'" as s  { let c = String.get s 1 in LITCHAR (c) }
99  | '"'           { let pos = lexbuf.lex_curr_p in
100                    let lin = pos.pos_lnum
```

```
101                 and col = pos.pos_cnum - pos.pos_bol - 1 in
102                 let buffer = Buffer.create 1 in
103                 let str = leia_string lin col buffer lexbuf in
104                   LITSTRING str }
105 | _ as c  { failwith (msg_erro lexbuf c) }
106 | eof        { EOF }
107
108 and comentario_bloco n = parse
109    "*/"      { if n=0 then read lexbuf
110                 else comentario_bloco (n-1) lexbuf }
111 | "/*"       { comentario_bloco (n+1) lexbuf }
112 | novalinha  { incr_num_linha lexbuf; comentario_bloco n lexbuf }
113 | _          { comentario_bloco n lexbuf }
114 | eof        { let pos = lexbuf.lex_curr_p in
115                 let lin = pos.pos_lnum
116                 and col = pos.pos_cnum - pos.pos_bol in
117                  erro lin col  "Comentario nao fechado" }
118
119 and leia_string lin col buffer = parse
120    '"'      { Buffer.contents buffer}
121 | "\\t"     { Buffer.add_char buffer '\t'; leia_string lin col buffer lexbuf
       }
122 | "\\n"     { Buffer.add_char buffer '\n'; leia_string lin col buffer lexbuf
       }
123 | '\\' '"'  { Buffer.add_char buffer '"'; leia_string lin col buffer lexbuf
       }
124 | '\\' '\\' { Buffer.add_char buffer '\\'; leia_string lin col buffer
     lexbuf }
125 | _ as c     { Buffer.add_char buffer c; leia_string lin col buffer lexbuf }
126 | eof       { erro lin col "A string nao foi fechada"}
```

0.5.2 Execução

Para execução utilize o seguinte código do carregador.ml

```
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
```

1. Execute o parse com ocamllex

```
1          ocamllex lexico.mll
```

2. Compile o arquivo .ml gerado pelo parse

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

3. Entre no interpretador do ocaml

```
1          rlwrap ocaml
```

4. No interpretador carregue o carregador.ml

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

5. Execute para o seu arquivo passando o caminho para o mesmo

```
1          lex "../../exemplos/Portugol/nano/nano12.alg";;
```

6. Após a execução você terá uma saida como a abaixo, onde temos os tokens do nosso código de entrada

```
1           lex "../../exemplos/Portugol/nano/nano12.alg";;
2 - : Lexico.tokens list =
3 [Lexico.PROGRAMA; Lexico.LITSTRING "nano12"; Lexico.VAR; Lexico.ID "n
    ";
4  Lexico.VIRG; Lexico.ID "m"; Lexico.VIRG; Lexico.ID "x"; Lexico.DEF;
5  Lexico.INTEGER; Lexico.INI; Lexico.ID "n"; Lexico.ATRIB; Lexico.
     LITINT 1;
6  Lexico.ID "m"; Lexico.ATRIB; Lexico.LITINT 2; Lexico.ID "x"; Lexico.
     ATRIB;
7  Lexico.LITINT 5; Lexico.WHILE; Lexico.ID "x"; Lexico.GT; Lexico.ID "n
     ";
8  Lexico.DO; Lexico.IF; Lexico.ID "n"; Lexico.EQUALS; Lexico.ID "m";
9  Lexico.OR; Lexico.ID "n"; Lexico.GTE; Lexico.LITINT 0; Lexico.ENTAO;
10 Lexico.ID "escreva"; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.
     ELSE;
11 Lexico.ID "escreva"; Lexico.APAR; Lexico.LITINT 0; Lexico.FPAR;
     Lexico.FIF;
```

```
12  Lexico.ID "x"; Lexico.ATRIB; Lexico.ID "x"; Lexico.MENOS; Lexico.
        LITINT 1;
13  Lexico.ID "fim_enquanto"; Lexico.FPROGRAMA; Lexico.EOF]
```

0.6 Analisador Sintático

É a analise realizada pelo compilador para identificar se o código pertence a linguagem.

0.6.1 Preparando o Ambiente

Antes de iniciarmos lembre-se de utilizar o lexico.mll atualizado que se encontrar na seção acima. Precisamos agora instalar o menhir que é quem irá gerar o analisador sintático LR(1) para a gramática que iremos criar. Para isso utilize

```
1     opam install menhir
```

Agora com o menhir instalado crie um arquivo .ocamlinit com o seguinte conteúdo

```
1  #use "topfind";;
2  #require "menhirLib";;
3  #directory "_build";;
4  #load "erroSint.cmo";;
5  #load "lexico.cmo";;
6  #load "sintatico.cmo";;
7  #load "ast.cmo";;
8  #load "sintaticoTest.cmo";;
9
10 open Ast
11 open SintaticoTest
```

0.6.2 Árvore Sintática Abstrata

Crie o arquivo ast.ml

```
1  (* The type of the abstract syntax tree (AST). *)
2  type ident = string
3
4  type programa = Programa of declaracoes * funcoes * comandos
5  and declaracoes = declaracao list
6  and funcoes = Funcoes of funcao list
7  and comandos = comando list
8  and declaracao = DecVar of ident * tipo
9  and expressoes = Expressoes of expressao list
10
11 and tipo = TipoInt
12          | TipoString
13          | TipoReal
```

```
14          | TipoBool
15
16 and campos = campo list
17 and campo = ident * tipo
18
19 and comando = CmdAtrib of variavel * expressao
20             | CmdSe of expressao * comandos * (comandos option)
21             | CmdEnquanto of expressao * comandos
22             | CmdReturn of expressao
23             | CmdEntrada of expressoes
24             | CmdChamada of expressao
25             | CmdSaida of expressoes
26             | CmdSaidaLine of expressoes
27             | CmdFor of variavel * expressao * expressao * comandos
28             | CmdSwitch of variavel * cases * (caseDefault option)
29
30 and cases = case list
31 and case = CmdCase of expressao * comandos
32 and caseDefault = CmdCaseDefault of comandos
33
34 and variaveis = variavel list
35 and variavel = VarSimples of ident
36              | VarCampo of variavel * ident
37              | VarElemento of variavel * expressao
38
39 and expressao = ExpVar of variavel
40                | ExpInt of int
41                | ExpString of string
42                | ExpChar of char
43                | ExpBool of bool
44                | ExpFuncao of ident * expressoes
45                | ExpOp of oper * expressao * expressao
46                | ExpNegativo of expressao
47
48 and funcao = Funcao of ident * ident * tipo * declaracoes * declaracoes *
       comandos
49
50 and oper = Mais
51          | Menos
52          | Mult
53          | Div
54          | Resto
55          | Potencia
56          | Menor
57          | Igual
58          | Difer
59          | Maior
```

```
60              | MaioIgual
61              | MenorIgual
62              | E
63              | Ou
64              | Concat
```

0.6.3 Gramática

Crie o arquivo sintatico.mly com o seguinte conteúdo

```
1  %{
2     open Ast
3  %}
4
5  %token <int> LITINT
6  %token <string> LITSTRING
7  %token <char> LITCHAR
8  %token <string> ID
9  %token <bool> LITBOOL
10 %token FUNCTION FFUNCTION
11 %token PROGRAMA FPROGRAMA INI
12 %token MAIS MENOS MULTI DIV
13 %token INTEGER CARACTER REAL VAR BOOL
14 %token APAR FPAR
15 %token ATRIB
16 %token IF
17 %token ELSE
18 %token EQUALS DIFER GTE GT LTE LT POTEN RESTO
19 %token VIRG
20 %token AND OR
21 %token DEF
22 %token SAIDA ENTRADA SAIDA_LINHA
23 %token WHILE FWHILE FOR FFOR SWITCH FSWITCH CASE CASE_DEFAULT RETURN
24 %token DO
25 %token FIF
26 %token ENTAO BEGIN END
27 %token EOF
28
29 %left OR AND
30 %left EQUALS DIFER
31 %left LT LTE GT GTE
32 %right POTEN
33 %left MAIS MENOS RESTO
34 %left MULTI DIV
35
36
37 %start <Ast.programa> prog
```

```
38
39 %%
40
41 prog:
42     | PROGRAMA LITSTRING
43         ds = declaracao
44       INI
45         cs = comando*
46       FPROGRAMA
47       dFunctions = decFunctions*
48       EOF {
49         let functions = Funcoes(dFunctions) in
50          Programa (List.flatten ds, functions, cs) }
51     ;
52
53 declaracao:
54     | VAR
55       v=variaveis* { v }
56     | { [] }
57     ;
58
59 variaveis:
60     | ids=separated_nonempty_list(VIRG, ID) DEF t=tipo {
61           List.map (fun id -> DecVar (id,t)) ids
62     }
63
64 tipo:
65     | INTEGER   { TipoInt }
66     | REAL      { TipoReal }
67     | CARACTER  { TipoString }
68     | BOOL      { TipoBool }
69     ;
70
71 comando: c=comando_atribuicao { c }
72        | c=comando_se         { c }
73        | c=comando_entrada    { c }
74        | c=comando_saida      { c }
75        | c=comando_saida_l    { c }
76        | c=comando_enquanto   { c }
77        | c=comando_for        { c }
78        | c=comando_switch     { c }
79        | c=comando_return     { c }
80        | c=comando_chamada     { c }
81        ;
82
83 comando_return: RETURN e=expressao {
84       CmdReturn (e)
```

```
85 }
86
87 comando_chamada: exp=chamada { CmdChamada(exp) }
88
89 comando_atribuicao: v=variavel ATRIB e=expressao {
90       CmdAtrib (v,e)
91 }
92
93 comando_se: IF teste=expressao ENTAO
94                entao=comando+
95                senao=option(ELSE cs=comando+ {cs})
96             FIF {
97               CmdSe (teste, entao, senao)
98             }
99
100 comando_entrada: ENTRADA APAR xs=separated_nonempty_list(VIRG, expressao)
    FPAR {
101                  let expressions = Expressoes(xs) in
102                   CmdEntrada expressions
103               }
104
105 comando_saida: SAIDA APAR xs=separated_nonempty_list(VIRG, expressao) FPAR
    {
106                let expressions = Expressoes(xs) in
107                CmdSaida expressions
108             }
109
110 comando_saida_l: SAIDA_LINHA APAR xs=separated_nonempty_list(VIRG,
    expressao) FPAR {
111                let expressions = Expressoes(xs) in
112                CmdSaidaLine expressions
113             }
114
115 comando_enquanto: WHILE teste=expressao DO
116                       entao=comando+
117                  FWHILE {
118                    CmdEnquanto( teste, entao )
119                  }
120
121 comando_for:  FOR v=variavel BEGIN l=expressao END r=expressao DO
122                 c=comando+
123               FFOR {
124                 CmdFor(v, l, r, c)
125               }
126
127 comando_switch: SWITCH v=variavel
128                     c=caso+
```

```
129                     d=option(default)
130                 FSWITCH {
131                   CmdSwitch(v, c, d)
132                 }
133
134 caso: CASE e=expressao
135           c=comando+ {
136             CmdCase(e, c)
137           }
138
139 default: CASE_DEFAULT
140             c=comando+ {
141               CmdCaseDefault(c)
142             }
143
144 decFunctions: FUNCTION a=ID APAR args=variaveis* FPAR DEF t=tipo
145                     variables=declaracao
146                 INI
147                   corpo=comando*
148                 FFUNCTION b=ID {
149                       Funcao( a, b, t, List.flatten args, List.flatten
                              variables, corpo )
150                   }
151
152 chamada: i=ID APAR xs=separated_nonempty_list(VIRG, expressao) FPAR {
153               let expressions = Expressoes(xs) in
154               ExpFuncao(i, expressions)
155           }
156
157 expressao:
158         | c=chamada     { c }
159         | v=variavel    { ExpVar v    }
160         | i=LITINT      { ExpInt i    }
161         | s=LITSTRING   { ExpString s }
162         | c=LITCHAR     { ExpChar c }
163         | b=LITBOOL     { ExpBool b }
164         | e1=expressao op=oper e2=expressao { ExpOp (op, e1, e2) }
165         | APAR e=expressao FPAR { e }
166         | MENOS e=expressao { ExpNegativo(e) }
167
168 %inline oper:
169   | MAIS  { Mais  }
170   | MENOS { Menos }
171   | MULTI { Mult  }
172   | DIV   { Div   }
173   | RESTO { Resto }
174   | POTEN { Potencia }
```

```
175     | LT { Menor }
176     | GT { Maior }
177     | LTE { MenorIgual }
178     | GTE { MaioIgual }
179     | EQUALS { Igual }
180     | DIFER { Difer }
181     | AND  { E      }
182     | OR { Ou     }
183
184
185 variavel:
186           | x=ID        { VarSimples x }
```

0.6.4 Execução

Para execução e detecção de erros cria o arquivo sintaticoTest.ml com o seguinte conteúdo.

```
1  open Printf
2  open Lexing
3
4  open Ast
5  exception Erro_Sintatico of string
6
7  module S = MenhirLib.General (* Streams *)
8  module I = Sintatico.MenhirInterpreter
9
10 open ErroSint
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 aut mato LR(1) contida em
       checkpoint *)
19
20 let pilha checkpoint =
21   match checkpoint with
22   | I.HandlingError amb -> I.stack amb
23   | _ -> assert false (* Isso n o pode acontecer *)
24
25 let estado checkpoint : int =
26   match Lazy.force (pilha checkpoint) with
27   | S.Nil -> (* O parser est  no estado inicial *)
28       0
29   | S.Cons (I.Element (s, _, _, _), _) ->
```

```
30      I.number s
31
32  let sucesso v = 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.read lexbuf in
42    I.loop_handle sucesso (falha lexbuf) fornecedor resultado
43
44
45
46  let parse_com_erro lexbuf =
47    try
48      Some (loop lexbuf (Sintatico.Incremental.prog 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 sint tico 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 rec tokens lexbuf =
70    let tok = Lexico.read lexbuf in
71    match tok with
72    | Sintatico.EOF -> [Sintatico.EOF]
73    | _ -> tok :: tokens lexbuf
74
75  let lexicoArq arq =
76    let ic = open_in arq in
```

```
77    let lexbuf = Lexing.from_channel ic in
78    let toks = tokens lexbuf in
79    let _ = close_in ic in
80    toks
```

0.6.5 Compilando

Para compilarmos o compilador, utilizei os comandos fornecidos pelo professor, são eles:

```
1 menhir -v --list-errors sintatico.mly > sintatico.msg
2
3 menhir sintatico.mly --compile-errors sintatico.msg >
4 erroSint.ml
5
6 ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --
7 table" -package menhirLib sintaticoTest.byte
```

Após isto utilize rlwrap ocaml na pasta do projeto onde se encontra o .ocamlinit para ter acesso ou código do compilador. No ocaml utilize parse_arq 'caminho para um arquivo' para gerar a árvore sintática