Construção de um compilador de Python para LLVM usando Objective Caml

Tatiane Fernanda de Souza Silva
tatianefx@gmail.com

Faculdade de Computação
Universidade Federal de Uberlândia

15 de novembro de 2018

Lista de Listagens

Sumário

Capítulo 1

Introdução

Este documento foi escrito como relatório do desenvolvimento de um compilador da linguagem Python que utiliza como assembler a máquina virtual LLVM (Low Level Virtual Machine) para gerar códigos binários executáveis.

O intuito dessas tarefas é aumentar a familiaridade com a linguagem Python, melhorar o entendimento do que é e como é um código gerado para a máquina virtual LLVM e também de como utilizá-la.

1.1 Sitema Operacional e Hardware

O Sistema Operacional (SO) utilizado para realização dos experimentos será o macOS High Sierra 10.13.6. O computador usado para execução das tarefas é um MacBook Air (13-inch, 2017) com processador 1,8 GHz Intel Core i5, 8 GB 1600 MHz DDR3 de memória RAM e 128 GB de espaço em disco SSD.

1.2 OCaml

OCaml é uma linguagem de programação de foco industrial e propósito geral, com ênfase em expressividade e segurança. É a tecnologia escolhida em empresas onde um único erro pode custar milhões. Possui uma comunidade ativa que desenvolveu um rico conjunto de bibliotecas. É também uma linguagem de ensino amplamente usada.

1.3 Low Level Virtual Machine

A *Low Level Virtual Machine* (LLVM) é uma infraestrutura de compilador escrita em C++, desenvolvida para otimizar em tempos de compilação, ligação e execução de programas escritos em linguagens de programação variadas. Implementada originalmente para C e C++, sua arquitetura permitiu a expansão para outras linguagens posteriormente, incluindo

Objective-C, Fortran, Ada, Haskell, bytecode Java, Python, Ruby, ActionScript, GLSL, Julia, entre outras.

1.4 Python

Python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem como um todo não é formalmente especificada. O padrão de fato é a implementação CPython.

Capítulo 2

Instalações

2.1 Homebrew

O Homebrew auxilia na instalação de outras dependências. Para instalá-lo digitamos o seguinte comando no terminal:

Listagem 2.1: Terminal

```
1 $ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install
    /master/install)"
```

2.2 Instalação do OCaml

Para efetuar a instalação o OCaml, basta digitar os seguintes comandos no terminal:

Listagem 2.2: Terminal

```
1 $ brew install ocaml
2 $ brew install opam
3 $ eval $(opam env)
```

Para verificar se tudo ocorreu bem com a instalação, basta digitar o comando a seguir e verificar sua saída.

Listagem 2.3: Terminal

```
1 $ ocaml -version
2 The OCaml toplevel, version 4.07.0
```

2.3 Instalação LLVM

Para a instalação da plataforma LLVM, basta digitar no terminal o seguinte comando:

brewinstall − −with − toolchainllvm

Além da plataforma LLVM, é necessária a instalação do CLANG que irá realizar a conversão do código da linguagem C para o assembly do LLVM. Para a instalação do CLANG, basta digitar no terminal o seguinte comando:

Listagem 2.4: Terminal

```
1 $(brew --prefix llvm)/bin
2 $ echo 'export PATH="/usr/local/opt/llvm/bin:$PATH"' >> ~/.bash_profile
```

2.4 Instalação Menhir

Para a instalação do Menhir, basta digitar no terminal o seguinte comando:

Listagem 2.5: Terminal

```
1 $ opam install menhir
2 $ brew install menhir
```

Capítulo 3

Execução dos programas

Depois de tudo instalado é possível executar os programas. A seguir temos os programas em Python seguidos por sua tradução para assembly LLVM.

3.1 Código feito em Python

Para executar um código feito em Python, basta executar o seguinte comando:

Listagem 3.1: Terminal

```
1 $ python file.py
2 $ python
3 > import file
```

3.2 Clang

Para compilar um arquivo .c para .ll, basta executar o seguinte comando:

Listagem 3.2: Terminal

```
1 $ clang -S -emit-llvm file.c -o file.ll
```

3.3 LLVM

Para compilar um arquivo .ll utilizando o llvm e executá-lo, basta executar os seguintes comandos:

Listagem 3.3: Terminal

```
1 $ llvm-as file.ll
2 $ lli file.bc
```

Para descompilar arquivo binário para geração de código em assembly:

Listagem 3.4: Terminal

```
$ llvm-dis file.bc
```

3.4 Nano Programas

Nesta seção teremos todos os nano programas escritos em Pascal, Java e Jasmin.

3.4.1 Estrutura básica

Listagem 3.5: nano01.py

```
def nano01():
    return

nano01()
```

Listagem 3.6: nano01.c

```
#import<stdio.h>

int main(){
  return 0;
}
```

Listagem 3.7: nano01.ll

```
; ModuleID = 'nano01.c'
source_filename = "nano01.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  ret i32 0
}

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
    divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
    -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
    non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
    fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
    false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
    features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
    soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

```

```
!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.2 Declaração de variável

Listagem 3.8: nano02.py

```
def nano02():
    n = (int)

nano02()
```

Listagem 3.9: nano02.c

```
#import<stdio.h>

int main(){
  int n;
  return 0;
}
```

Listagem 3.10: nano02.ll

```
; ModuleID = 'nano02.c'
source_filename = "nano02.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  ret i32 0
}

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
    divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
    -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
    non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
    fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
    false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
    features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
    soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.3 Atribuição de valor a variável

Listagem 3.11: nano03.py

```
def nano03():
    n = (int)
    n = 1

nano03()
```

Listagem 3.12: nano03.c

```
#import<stdio.h>

int main(){
  int n;
  n = 1;
  return 0;
}
```

Listagem 3.13: nano03.ll

```
; ModuleID = 'nano03.c'
source_filename = "nano03.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 1, i32* %2, align 4
  ret i32 0
}

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
    divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
    -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
    non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
    fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
    false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
    features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
    soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.4 Soma de valores inteiros

Listagem 3.14: nano04.py

```
def nano04():
    n = (int)
    n = 1 + 2

nano04()
```

Listagem 3.15: nano04.c

```
#import<stdio.h>

int main(){
  int n;
  n = 1 + 2;
  return 0;
}
```

Listagem 3.16: nano04.ll

```
; ModuleID = 'nano04.c'
source_filename = "nano04.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 3, i32* %2, align 4
  ret i32 0
}

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
    divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
    -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
    non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
    fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
    false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
    features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
    soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.5 Impressão de inteiro

Listagem 3.17: nano05.py

```
def nano05():
    n = 2
    print(n)

nano05()
```

Listagem 3.18: nano05.c

```
#import<stdio.h>

int main(){
  int n;
```

```
5   n = 2;
6   printf("%d", n);
7   return 0;
8 }
```

Listagem 3.19: nano05.ll

```
1 ; ModuleID = 'nano05.c'
2 source_filename = "nano05.c"
3 target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
4 target triple = "x86_64-apple-macosx10.13.0"
5
6 @.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
7
8 ; Function Attrs: noinline nounwind optnone uwtable
9 define i32 @main() #0 {
10   %1 = alloca i32, align 4
11   %2 = alloca i32, align 4
12   store i32 0, i32* %1, align 4
13   store i32 2, i32* %2, align 4
14   %3 = load i32, i32* %2, align 4
15   %4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
        [3 x i8]* @.str, i32 0, i32 0), i32 %3)
16   ret i32 0
17 }
18
19 declare i32 @printf(i8*, ...) #1
20
21 attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
      divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
      -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
      non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
      fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
      false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
      features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
      soft-float"="false" }
22 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
      -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
      elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
       "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
      trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
      x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"
      ="false" "use-soft-float"="false" }
23
24 !llvm.module.flags = !{!0}
25 !llvm.ident = !{!1}
26
27 !0 = !{i32 1, !"wchar_size", i32 4}
28 !1 = !{i32 7, !"PIC Level", i32 2}
29 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.6 Subtração e impressão do resultado

Listagem 3.20: nano06.py

```
1 def nano06():
2     n = 1 - 2
3     print(n)
```

```
4
5 nano06()
```

Listagem 3.21: nano06.c

```
1 #import<stdio.h>
2
3 int main(){
4   int n;
5   n = 1 - 2;
6   printf("%d", n);
7   return 0;
8 }
```

Listagem 3.22: nano06.ll

```
1 ; ModuleID = 'nano06.c'
2 source_filename = "nano06.c"
3 target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
4 target triple = "x86_64-apple-macosx10.13.0"
5
6 @.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
7
8 ; Function Attrs: noinline nounwind optnone uwtable
9 define i32 @main() #0 {
10   %1 = alloca i32, align 4
11   %2 = alloca i32, align 4
12   store i32 0, i32* %1, align 4
13   store i32 -1, i32* %2, align 4
14   %3 = load i32, i32* %2, align 4
15   %4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
        [3 x i8]* @.str, i32 0, i32 0), i32 %3)
16   ret i32 0
17 }
18
19 declare i32 @printf(i8*, ...) #1
20
21 attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
     divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
     -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
     non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
     fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
     false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
     features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
     soft-float"="false" }
22 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
     -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
     elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
      "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
     trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
     x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"
     ="false" "use-soft-float"="false" }
23
24 !llvm.module.flags = !{!0}
25 !llvm.ident = !{!1}
26
27 !0 = !{i32 1, !"wchar_size", i32 4}
28 !1 = !{i32 7, !"PIC Level", i32 2}
29 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.7 Estrutura condicional e impressão

Listagem 3.23: nano07.py

```
def nano07():
    n = 1
    if n == 1:
        print(n)

nano07()
```

Listagem 3.24: nano07.c

```
#import<stdio.h>

int main(){
  int n;
  n = 1;
  if (n == 1) {
    printf("%d", n);
  }
  return 0;
}
```

Listagem 3.25: nano07.ll

```
; ModuleID = 'nano07.c'
source_filename = "nano07.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

@.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 1, i32* %2, align 4
  %3 = load i32, i32* %2, align 4
  %4 = icmp eq i32 %3, 1
  br i1 %4, label %5, label %8

; <label>:5:                                      ; preds = %0
  %6 = load i32, i32* %2, align 4
  %7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
      [3 x i8]* @.str, i32 0, i32 0), i32 %6)
  br label %8

; <label>:8:                                      ; preds = %5, %0
  ret i32 0
}

declare i32 @printf(i8*, ...) #1

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
    divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
    -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
    non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
```

```
    fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
    false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
    features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
    soft-float"="false" }
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
    -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
    elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
     "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
    trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
    x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"
    ="false" "use-soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.8 Estrutura condicional e impressão

Listagem 3.26: nano08.py

```
def nano08():
    n = 1
    if n == 1:
        print(n)
    else:
        print(0)

nano08()
```

Listagem 3.27: nano08.c

```
#import<stdio.h>

int main(){
  int n;
  n = 1;
  if (n == 1) {
    printf("%d", n);
  } else {
    printf("0");
  }
  return 0;
}
```

Listagem 3.28: nano08.ll

```
; ModuleID = 'nano08.c'
source_filename = "nano08.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

@.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.str.1 = private unnamed_addr constant [2 x i8] c"0\00", align 1

```

```
; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 1, i32* %2, align 4
  %3 = load i32, i32* %2, align 4
  %4 = icmp eq i32 %3, 1
  br i1 %4, label %5, label %8

; <label>:5:                                      ; preds = %0
  %6 = load i32, i32* %2, align 4
  %7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
      [3 x i8]* @.str, i32 0, i32 0), i32 %6)
  br label %10

; <label>:8:                                      ; preds = %0
  %9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8],
      [2 x i8]* @.str.1, i32 0, i32 0))
  br label %10

; <label>:10:                                     ; preds = %8, %5
  ret i32 0
}

declare i32 @printf(i8*, ...) #1

attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
    divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
    -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
    non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
    fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
    false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
    features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
    soft-float"="false" }
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
    -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
    elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
     "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
    trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
    x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"
    ="false" "use-soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.9 Adição com divisão, estrutura condicional e impressão

Listagem 3.29: nano09.py

```
def nano09():
    n = 1 + (1 / 2)
    if n == 1:
        print(n)
```

```
5     else:
6         print(0)
7
8 nano09()
```

Listagem 3.30: nano09.c

```
1 #import<stdio.h>
2
3 int main(){
4   int n;
5   n = 1 + (1 / 2);
6   if (n == 1) {
7     printf("%d", n);
8   } else {
9     printf("0");
10  }
11  return 0;
12 }
```

Listagem 3.31: nano09.ll

```
1 ; ModuleID = 'nano09.c'
2 source_filename = "nano09.c"
3 target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
4 target triple = "x86_64-apple-macosx10.13.0"
5
6 @.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
7 @.str.1 = private unnamed_addr constant [2 x i8] c"0\00", align 1
8
9 ; Function Attrs: noinline nounwind optnone uwtable
10 define i32 @main() #0 {
11   %1 = alloca i32, align 4
12   %2 = alloca i32, align 4
13   store i32 0, i32* %1, align 4
14   store i32 1, i32* %2, align 4
15   %3 = load i32, i32* %2, align 4
16   %4 = icmp eq i32 %3, 1
17   br i1 %4, label %5, label %8
18
19 ; <label>:5:                                      ; preds = %0
20   %6 = load i32, i32* %2, align 4
21   %7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
        [3 x i8]* @.str, i32 0, i32 0), i32 %6)
22   br label %10
23
24 ; <label>:8:                                      ; preds = %0
25   %9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8],
        [2 x i8]* @.str.1, i32 0, i32 0))
26   br label %10
27
28 ; <label>:10:                                     ; preds = %8, %5
29   ret i32 0
30 }
31
32 declare i32 @printf(i8*, ...) #1
33
34 attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
     divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
```

```
    -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
    non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
    fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
    false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
    features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
    soft-float"="false" }
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
    -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
    elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
     "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
    trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
    x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"
    ="false" "use-soft-float"="false" }

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.10 Condição de igualdade

Listagem 3.32: nano10.py

```
def nano10():
    n = 1
    m = 2
    if n == m:
        print(n)
    else:
        print(0)

nano10()
```

Listagem 3.33: nano10.c

```
#import<stdio.h>

int main(){
  int n, m;
  n = 1;
  m = 2;
  if (n == m) {
    printf("%d", n);
  } else {
    printf("0");
  }
  return 0;
}
```

Listagem 3.34: nano10.ll

```
; ModuleID = 'nano10.c'
source_filename = "nano10.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"
```

```
5
6 @.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
7 @.str.1 = private unnamed_addr constant [2 x i8] c"0\00", align 1
8
9 ; Function Attrs: noinline nounwind optnone uwtable
10 define i32 @main() #0 {
11   %1 = alloca i32, align 4
12   %2 = alloca i32, align 4
13   %3 = alloca i32, align 4
14   store i32 0, i32* %1, align 4
15   store i32 1, i32* %2, align 4
16   store i32 2, i32* %3, align 4
17   %4 = load i32, i32* %2, align 4
18   %5 = load i32, i32* %3, align 4
19   %6 = icmp eq i32 %4, %5
20   br i1 %6, label %7, label %10
21
22 ; <label>:7:                                      ; preds = %0
23   %8 = load i32, i32* %2, align 4
24   %9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
        [3 x i8]* @.str, i32 0, i32 0), i32 %8)
25   br label %12
26
27 ; <label>:10:                                     ; preds = %0
28   %11 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8],
        [2 x i8]* @.str.1, i32 0, i32 0))
29   br label %12
30
31 ; <label>:12:                                     ; preds = %10, %7
32   ret i32 0
33 }
34
35 declare i32 @printf(i8*, ...) #1
36
37 attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
      divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
      -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
      non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
      fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
      false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
      features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
      soft-float"="false" }
38 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
      -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
      elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
       "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
      trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
      x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"
      ="false" "use-soft-float"="false" }
39
40 !llvm.module.flags = !{!0}
41 !llvm.ident = !{!1}
42
43 !0 = !{i32 1, !"wchar_size", i32 4}
44 !1 = !{i32 7, !"PIC Level", i32 2}
45 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.11 Estrutura de repetição "while"com impressão

Listagem 3.35: nano11.py

```
def nano11():
    n = 1
    m = 2
    x = 5
    while x > n:
        n = n + m
        print(n)

nano11()
```

Listagem 3.36: nano11.c

```
#import<stdio.h>

int main(){
  int n, m, x;
  n = 1;
  m = 2;
  x = 5;

  while (x > n) {
    n = n + m;
    printf("%d", n);
  }
  return 0;
}
```

Listagem 3.37: nano11.ll

```
; ModuleID = 'nano11.c'
source_filename = "nano11.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

@.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 1, i32* %2, align 4
  store i32 2, i32* %3, align 4
  store i32 5, i32* %4, align 4
  br label %5

; <label>:5:                                      ; preds = %9, %0
  %6 = load i32, i32* %4, align 4
  %7 = load i32, i32* %2, align 4
  %8 = icmp sgt i32 %6, %7
  br i1 %8, label %9, label %15

; <label>:9:                                      ; preds = %5
```

```
27   %10 = load i32, i32* %2, align 4
28   %11 = load i32, i32* %3, align 4
29   %12 = add nsw i32 %10, %11
30   store i32 %12, i32* %2, align 4
31   %13 = load i32, i32* %2, align 4
32   %14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
        [3 x i8]* @.str, i32 0, i32 0), i32 %13)
33   br label %5
34
35 ; <label>:15:                                     ; preds = %5
36   ret i32 0
37 }
38
39 declare i32 @printf(i8*, ...) #1
40
41 attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
      divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
      -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
      non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
      fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
      false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
      features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
      soft-float"="false" }
42 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
      -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
      elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
       "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
      trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
      x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"
      ="false" "use-soft-float"="false" }
43
44 !llvm.module.flags = !{!0}
45 !llvm.ident = !{!1}
46
47 !0 = !{i32 1, !"wchar_size", i32 4}
48 !1 = !{i32 7, !"PIC Level", i32 2}
49 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.4.12 Estruturas de repetição e condicional com impressão de valores

Listagem 3.38: nano12.py

```
1 def nano12():
2     n = 1
3     m = 2
4     x = 5
5     while x > n:
6         if n == m:
7             print(n)
8         else:
9             print(0)
10            x = x -1
11
12 nano12()
```

Listagem 3.39: nano12.c

```c
#import<stdio.h>

int main(){
  int n, m, x;
  n = 1;
  m = 2;
  x = 5;

  while (x > n) {
    if (n == m) {
      printf("%d", n);
    } else {
      printf("0");
      x = x - 1;
    }
  }
  return 0;
}
```

Listagem 3.40: nano12.ll

```
; ModuleID = 'nano12.c'
source_filename = "nano12.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

@.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.str.1 = private unnamed_addr constant [2 x i8] c"0\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 1, i32* %2, align 4
  store i32 2, i32* %3, align 4
  store i32 5, i32* %4, align 4
  br label %5

; <label>:5:                                      ; preds = %20, %0
  %6 = load i32, i32* %4, align 4
  %7 = load i32, i32* %2, align 4
  %8 = icmp sgt i32 %6, %7
  br i1 %8, label %9, label %21

; <label>:9:                                      ; preds = %5
  %10 = load i32, i32* %2, align 4
  %11 = load i32, i32* %3, align 4
  %12 = icmp eq i32 %10, %11
  br i1 %12, label %13, label %16

; <label>:13:                                     ; preds = %9
  %14 = load i32, i32* %2, align 4
  %15 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
      [3 x i8]* @.str, i32 0, i32 0), i32 %14)
  br label %20

```

```
38 ; <label>:16:                                     ; preds = %9
39   %17 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8],
        [2 x i8]* @.str.1, i32 0, i32 0))
40   %18 = load i32, i32* %4, align 4
41   %19 = sub nsw i32 %18, 1
42   store i32 %19, i32* %4, align 4
43   br label %20
44
45 ; <label>:20:                                     ; preds = %16, %13
46   br label %5
47
48 ; <label>:21:                                     ; preds = %5
49   ret i32 0
50 }
51
52 declare i32 @printf(i8*, ...) #1
53
54 attributes #0 = { noinline nounwind optnone uwtable "correctly-rounded-
      divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise
      -fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-
      non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-
      fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="
      false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-
      features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"="false" "use-
      soft-float"="false" }
55 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
      -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
      elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
       "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
      trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
      x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2,+x87" "unsafe-fp-math"
      ="false" "use-soft-float"="false" }
56
57 !llvm.module.flags = !{!0}
58 !llvm.ident = !{!1}
59
60 !0 = !{i32 1, !"wchar_size", i32 4}
61 !1 = !{i32 7, !"PIC Level", i32 2}
62 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5 Micro Programas

Nesta seção teremos todos os micro programas escritos em Pascal e Jasmin.

3.5.1 Converte graus Celsius para Fahrenheit

Listagem 3.41: micro01.py

```
1 def micro01():
2     cel , far = 0.0 , 0.0
3     print(" Tabela de conversao: Celsius -> Fahrenheit")
4     print("Digite a temperatura em Celsius: ")
5     cel = input()
6     far = (9 * cel + 160) / 5
7     print("A nova temperatura e:" + str(far) + "F")
```

```
8
9 micro01()
```

Listagem 3.42: micro01.c

```
1 #import<stdio.h>
2
3 int main(){
4   float cel, far;
5   cel = 0.0;
6   far = 0.0;
7   printf("Tabela de conversao: Celsius -> Fahrenheit");
8
9   printf("Digite a temperatura em Celsius: ");
10  scanf("%f", &cel);
11
12  far = (9.0 * cel + 160.0) / 5.0;
13  printf("%f", far);
14
15  return 0;
16 }
```

Listagem 3.43: micro01.ll

```
1 ; ModuleID = 'micro01.c'
2 source_filename = "micro01.c"
3 target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
4 target triple = "x86_64-apple-macosx10.13.0"
5
6 @.str = private unnamed_addr constant [43 x i8] c"Tabela de conversao:
    Celsius -> Fahrenheit\00", align 1
7 @.str.1 = private unnamed_addr constant [34 x i8] c"Digite a temperatura
    em Celsius: \00", align 1
8 @.str.2 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
9
10 ; Function Attrs: noinline nounwind optnone ssp uwtable
11 define i32 @main() #0 {
12   %1 = alloca i32, align 4
13   %2 = alloca float, align 4
14   %3 = alloca float, align 4
15   store i32 0, i32* %1, align 4
16   store float 0.000000e+00, float* %2, align 4
17   store float 0.000000e+00, float* %3, align 4
18   %4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([43 x i8],
       [43 x i8]* @.str, i32 0, i32 0))
19   %5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([34 x i8],
       [34 x i8]* @.str.1, i32 0, i32 0))
20   %6 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3
        x i8]* @.str.2, i32 0, i32 0), float* %2)
21   %7 = load float, float* %2, align 4
22   %8 = fpext float %7 to double
23   %9 = fmul double 9.000000e+00, %8
24   %10 = fadd double %9, 1.600000e+02
25   %11 = fdiv double %10, 5.000000e+00
26   %12 = fptrunc double %11 to float
27   store float %12, float* %3, align 4
28   %13 = load float, float* %3, align 4
29   %14 = fpext float %13 to double
```

```
30   %15 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
        [3 x i8]* @.str.2, i32 0, i32 0), double %14)
31   ret i32 0
32 }
33
34 declare i32 @printf(i8*, ...) #1
35
36 declare i32 @scanf(i8*, ...) #1
37
38 attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
      -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
      precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
      -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
      nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
      math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
      target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
      " "unsafe-fp-math"="false" "use-soft-float"="false" }
39 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
      -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
      elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
       "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
      trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
      penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
      ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
40
41 !llvm.module.flags = !{!0, !1}
42 !llvm.ident = !{!2}
43
44 !0 = !{i32 1, !"wchar_size", i32 4}
45 !1 = !{i32 7, !"PIC Level", i32 2}
46 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5.2 Ler dois inteiros e decide qual é maior

Listagem 3.44: micro02.py

```
1 def micro02():
2     num1, num2 = 0 , 0
3     print("Digite o primeiro numero: ")
4     num1 = int(input())
5     print("Digite o segundo numero: ")
6     num2 = int(input())
7
8     if num1 > num2:
9         print("O primeiro numero " + str(num1) + " e maior que o segundo "
              + str(num2))
10    else:
11        print("O segundo numero " + str(num2) + " e maior que o primeiro "
              + str(num1))
12
13 micro02()
```

Listagem 3.45: micro02.c

```
1 #import<stdio.h>
2
3 int main(){
4   int num1, num2;
```

```
5   printf("Digite o primeiro numero: ");
6   scanf("%d", &num1);
7   printf("Digite o segundo numero: ");
8   scanf("%d", &num2);
9
10  if(num1 > num2) {
11    printf("O primeiro numero %d e maior que o segundo %d.", num1, num2);
12  } else {
13    printf("O primeiro numero %d e maior que o segundo %d.", num2, num1);
14  }
15
16  return 0;
17 }
```

Listagem 3.46: micro02.ll

```
1  ; ModuleID = 'micro02.c'
2  source_filename = "micro02.c"
3  target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
4  target triple = "x86_64-apple-macosx10.13.0"
5
6  @.str = private unnamed_addr constant [27 x i8] c"Digite o primeiro numero
       : \00", align 1
7  @.str.1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
8  @.str.2 = private unnamed_addr constant [26 x i8] c"Digite o segundo
       numero: \00", align 1
9  @.str.3 = private unnamed_addr constant [47 x i8] c"O primeiro numero %d e
        maior que o segundo %d.\00", align 1
10
11 ; Function Attrs: noinline nounwind optnone ssp uwtable
12 define i32 @main() #0 {
13   %1 = alloca i32, align 4
14   %2 = alloca i32, align 4
15   %3 = alloca i32, align 4
16   store i32 0, i32* %1, align 4
17   %4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([27 x i8],
         [27 x i8]* @.str, i32 0, i32 0))
18   %5 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3
          x i8]* @.str.1, i32 0, i32 0), i32* %2)
19   %6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([26 x i8],
         [26 x i8]* @.str.2, i32 0, i32 0))
20   %7 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3
          x i8]* @.str.1, i32 0, i32 0), i32* %3)
21   %8 = load i32, i32* %2, align 4
22   %9 = load i32, i32* %3, align 4
23   %10 = icmp sgt i32 %8, %9
24   br i1 %10, label %11, label %15
25
26 ; <label>:11:                                     ; preds = %0
27   %12 = load i32, i32* %2, align 4
28   %13 = load i32, i32* %3, align 4
29   %14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([47 x i8],
          [47 x i8]* @.str.3, i32 0, i32 0), i32 %12, i32 %13)
30   br label %19
31
32 ; <label>:15:                                     ; preds = %0
33   %16 = load i32, i32* %3, align 4
34   %17 = load i32, i32* %2, align 4
35   %18 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([47 x i8],
```

```
        [47 x i8]* @.str.3, i32 0, i32 0), i32 %16, i32 %17)
  br label %19

; <label>:19:                                     ; preds = %15, %11
  ret i32 0
}

declare i32 @printf(i8*, ...) #1

declare i32 @scanf(i8*, ...) #1

attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
    -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
    precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
    -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
    nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
    math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
    target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
    " "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
    -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
    elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
     "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
    trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
    penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
    ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }

!llvm.module.flags = !{!0, !1}
!llvm.ident = !{!2}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5.3 Lê um número e verica se ele está entre 100 e 200

Listagem 3.47: micro03.py

```
def micro03():
    numero = 0
    print("Digite um numero: ")
    numero = int(input())
    if numero >= 100:
        if numero<= 200:
            print("O numero esta no intervalo entre 100 e 200")
        else:
            print("O numero nao esta no intervalo entre 100 e 200")
    else:
        print("O numero nao esta no intervalo entre 100 e 200")

micro03()
```

Listagem 3.48: micro03.c

```
#import<stdio.h>

int main(){
  int numero;
```

```
5   printf("Digite um numero: ");
6   scanf("%d", &numero);
7
8   if (numero >= 100) {
9     if (numero <= 200) {
10      printf("O numero esta no intervalo entre 100 e 200");
11    } else {
12      printf("O numero nao esta no intervalo entre 100 e 200");
13    }
14  } else {
15    printf("O numero nao esta no intervalo entre 100 e 200");
16  }
17
18  return 0;
19 }
```

Listagem 3.49: micro03.ll

```
1  ; ModuleID = 'micro03.c'
2  source_filename = "micro03.c"
3  target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
4  target triple = "x86_64-apple-macosx10.13.0"
5
6  @.str = private unnamed_addr constant [19 x i8] c"Digite um numero: \00",
       align 1
7  @.str.1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
8  @.str.2 = private unnamed_addr constant [43 x i8] c"O numero esta no
       intervalo entre 100 e 200\00", align 1
9  @.str.3 = private unnamed_addr constant [47 x i8] c"O numero nao esta no
       intervalo entre 100 e 200\00", align 1
10
11 ; Function Attrs: noinline nounwind optnone ssp uwtable
12 define i32 @main() #0 {
13   %1 = alloca i32, align 4
14   %2 = alloca i32, align 4
15   store i32 0, i32* %1, align 4
16   %3 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([19 x i8],
         [19 x i8]* @.str, i32 0, i32 0))
17   %4 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3
         x i8]* @.str.1, i32 0, i32 0), i32* %2)
18   %5 = load i32, i32* %2, align 4
19   %6 = icmp sge i32 %5, 100
20   br i1 %6, label %7, label %15
21
22 ; <label>:7:                                      ; preds = %0
23   %8 = load i32, i32* %2, align 4
24   %9 = icmp sle i32 %8, 200
25   br i1 %9, label %10, label %12
26
27 ; <label>:10:                                     ; preds = %7
28   %11 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([43 x i8],
         [43 x i8]* @.str.2, i32 0, i32 0))
29   br label %14
30
31 ; <label>:12:                                     ; preds = %7
32   %13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([47 x i8],
         [47 x i8]* @.str.3, i32 0, i32 0))
33   br label %14
34
```

```
; <label>:14:                                     ; preds = %12, %10
  br label %17

; <label>:15:                                     ; preds = %0
  %16 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([47 x i8],
      [47 x i8]* @.str.3, i32 0, i32 0))
  br label %17

; <label>:17:                                     ; preds = %15, %14
  ret i32 0
}

declare i32 @printf(i8*, ...) #1

declare i32 @scanf(i8*, ...) #1

attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
    -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
    precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
    -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
    nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
    math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
    target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
    " "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
    -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
    elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
     "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
    trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
    penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
    ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }

!llvm.module.flags = !{!0, !1}
!llvm.ident = !{!2}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5.4 Lê números e informa quais estão entre 10 e 150

Listagem 3.50: micro04.py

```
def micro04():
    x, num, intervalo = 0,0,0
    for x in range(5):
        print("Digite o numero: ")
        num = int(input())
        if num >= 10:
            if num <= 150:
                intervalo = intervalo + 1
    print("Ao total, foram digitados " + str(intervalo) + " numeros no
        intervalotre 10 e 150")

micro04()
```

Listagem 3.51: micro04.c

```c
#import<stdio.h>

int main(){
  int x, num, intervalo;

  for (x = 0; x < 5; x++) {
    printf("Digite um numero: ");
    scanf("%d", &num);
    if(num >= 10) {
      if (num <= 150) {
        intervalo = intervalo + 1;
      }
    }
  }

  printf("Ao total, foram digitados %d numeros no intervalotre 10 e 150.",
      intervalo);

  return 0;
}
```

Listagem 3.52: micro04.ll

```
; ModuleID = 'micro04.c'
source_filename = "micro04.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

@.str = private unnamed_addr constant [19 x i8] c"Digite um numero: \00",
    align 1
@.str.1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.str.2 = private unnamed_addr constant [63 x i8] c"Ao total, foram
    digitados %d numeros no intervalotre 10 e 150.\00", align 1

; Function Attrs: noinline nounwind optnone ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 0, i32* %2, align 4
  br label %5

; <label>:5:                                      ; preds = %21, %0
  %6 = load i32, i32* %2, align 4
  %7 = icmp slt i32 %6, 5
  br i1 %7, label %8, label %24

; <label>:8:                                      ; preds = %5
  %9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([19 x i8],
      [19 x i8]* @.str, i32 0, i32 0))
  %10 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8],
      [3 x i8]* @.str.1, i32 0, i32 0), i32* %3)
  %11 = load i32, i32* %3, align 4
  %12 = icmp sge i32 %11, 10
  br i1 %12, label %13, label %20

; <label>:13:                                     ; preds = %8
```

```
33   %14 = load i32, i32* %3, align 4
34   %15 = icmp sle i32 %14, 150
35   br i1 %15, label %16, label %19
36
37 ; <label>:16:                                     ; preds = %13
38   %17 = load i32, i32* %4, align 4
39   %18 = add nsw i32 %17, 1
40   store i32 %18, i32* %4, align 4
41   br label %19
42
43 ; <label>:19:                                     ; preds = %16, %13
44   br label %20
45
46 ; <label>:20:                                     ; preds = %19, %8
47   br label %21
48
49 ; <label>:21:                                     ; preds = %20
50   %22 = load i32, i32* %2, align 4
51   %23 = add nsw i32 %22, 1
52   store i32 %23, i32* %2, align 4
53   br label %5
54
55 ; <label>:24:                                     ; preds = %5
56   %25 = load i32, i32* %4, align 4
57   %26 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([63 x i8],
         [63 x i8]* @.str.2, i32 0, i32 0), i32 %25)
58   ret i32 0
59 }
60
61 declare i32 @printf(i8*, ...) #1
62
63 declare i32 @scanf(i8*, ...) #1
64
65 attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
     -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
     precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
     -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
     nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
     math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
     target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
     " "unsafe-fp-math"="false" "use-soft-float"="false" }
66 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
     -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
     elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
      "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
     trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
     penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
     ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
67
68 !llvm.module.flags = !{!0, !1}
69 !llvm.ident = !{!2}
70
71 !0 = !{i32 1, !"wchar_size", i32 4}
72 !1 = !{i32 7, !"PIC Level", i32 2}
73 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5.5 Lê strings e caracteres

Listagem 3.53: micro05.py

```
def micro05():
    x,m,h = 0,0,0
    nome,sexo = "",""
    for x in range(5):
        print("Digite o nome: ")
        nome = input()
        print("H - Homem ou M - Mulher")
        sexo = input()
        if sexo == 'H':
            h = h+1
        elif sexo == 'M':
            m = m+1
        else:
            print("Sexo so pode ser H ou M!")

    print("Foram inseridos " + str(h) + " Homens")
    print("Foram inseridas " + str(m) + " Mulheres")

micro05()
```

Listagem 3.54: micro05.c

```
#import<stdio.h>
#include <string.h>

int main(int argc, char *argv[]){
  int x, m, h;
  char nome, sexo;

  for (x = 0; x < 5; x++) {
    printf("Digite o nome: ");
    scanf("%c", &nome);

    printf("H - Homem ou M - Mulher: ");
    scanf("%c", &sexo);

    switch (sexo) {
      case 'H':
        h = h + 1;
        break;
      case 'M':
        m = m + 1;
        break;
      default:
        printf("Sexo só pode ser H ou M!");
        break;
    }
  }

  printf("Foram inseridos %d Homens", h);
  printf("Foram inseridas %d Mulheres", m);

  return 0;
}
```

Listagem 3.55: micro05.ll

```
; ModuleID = 'micro05.c'
source_filename = "micro05.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

@.str = private unnamed_addr constant [16 x i8] c"Digite o nome: \00", align 1
@.str.1 = private unnamed_addr constant [3 x i8] c"%c\00", align 1
@.str.2 = private unnamed_addr constant [26 x i8] c"H - Homem ou M - Mulher: \00", align 1
@.str.3 = private unnamed_addr constant [26 x i8] c"Sexo s\C3\B3 pode ser H ou M!\00", align 1
@.str.4 = private unnamed_addr constant [26 x i8] c"Foram inseridos %d Homens\00", align 1
@.str.5 = private unnamed_addr constant [28 x i8] c"Foram inseridas %d Mulheres\00", align 1

; Function Attrs: noinline nounwind optnone ssp uwtable
define i32 @main(i32, i8**) #0 {
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i8**, align 8
  %6 = alloca i32, align 4
  %7 = alloca i32, align 4
  %8 = alloca i32, align 4
  %9 = alloca i8, align 1
  %10 = alloca i8, align 1
  store i32 0, i32* %3, align 4
  store i32 %0, i32* %4, align 4
  store i8** %1, i8*** %5, align 8
  store i32 0, i32* %6, align 4
  br label %11

; <label>:11:                                     ; preds = %30, %2
  %12 = load i32, i32* %6, align 4
  %13 = icmp slt i32 %12, 5
  br i1 %13, label %14, label %33

; <label>:14:                                     ; preds = %11
  %15 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([16 x i8], [16 x i8]* @.str, i32 0, i32 0))
  %16 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str.1, i32 0, i32 0), i8* %9)
  %17 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([26 x i8], [26 x i8]* @.str.2, i32 0, i32 0))
  %18 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str.1, i32 0, i32 0), i8* %10)
  %19 = load i8, i8* %10, align 1
  %20 = sext i8 %19 to i32
  switch i32 %20, label %27 [
    i32 72, label %21
    i32 77, label %24
  ]

; <label>:21:                                     ; preds = %14
  %22 = load i32, i32* %8, align 4
  %23 = add nsw i32 %22, 1
  store i32 %23, i32* %8, align 4
  br label %29
```

```
51
52 ; <label>:24:                                     ; preds = %14
53   %25 = load i32, i32* %7, align 4
54   %26 = add nsw i32 %25, 1
55   store i32 %26, i32* %7, align 4
56   br label %29
57
58 ; <label>:27:                                     ; preds = %14
59   %28 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([26 x i8],
         [26 x i8]* @.str.3, i32 0, i32 0))
60   br label %29
61
62 ; <label>:29:                                     ; preds = %27, %24, %21
63   br label %30
64
65 ; <label>:30:                                     ; preds = %29
66   %31 = load i32, i32* %6, align 4
67   %32 = add nsw i32 %31, 1
68   store i32 %32, i32* %6, align 4
69   br label %11
70
71 ; <label>:33:                                     ; preds = %11
72   %34 = load i32, i32* %8, align 4
73   %35 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([26 x i8],
         [26 x i8]* @.str.4, i32 0, i32 0), i32 %34)
74   %36 = load i32, i32* %7, align 4
75   %37 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([28 x i8],
         [28 x i8]* @.str.5, i32 0, i32 0), i32 %36)
76   ret i32 0
77 }
78
79 declare i32 @printf(i8*, ...) #1
80
81 declare i32 @scanf(i8*, ...) #1
82
83 attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
       -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
       precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
       -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
       nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
       math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
       target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
       " "unsafe-fp-math"="false" "use-soft-float"="false" }
84 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
       -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
       elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
        "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
       trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
       penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
       ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
85
86 !llvm.module.flags = !{!0, !1}
87 !llvm.ident = !{!2}
88
89 !0 = !{i32 1, !"wchar_size", i32 4}
90 !1 = !{i32 7, !"PIC Level", i32 2}
91 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5.6 Escreve um número lido por extenso

Listagem 3.56: micro06.py

```
def micro06():
    numero = 0
    print("Digite um numero de 1 a 5: ")
    numero = int(input())
    if numero == 1:
        print("Um")
    elif numero == 2:
        print("Dois")
    elif numero == 3:
        print("Tres")
    elif numero == 4:
        print("Quatro")
    elif numero == 5:
        print("Cinco")
    else:
        print("Numero Invalido!!!")

micro06()
```

Listagem 3.57: micro06.c

```
#import<stdio.h>

int main(){
  int numero;
  printf("Digite um numero de 1 a 5: ");
  scanf("%d", &numero);

  if (numero == 1) {
    printf("Um");
  } else if(numero == 2) {
    printf("Dois");
  } else if(numero == 3) {
    printf("Tres");
  } else if(numero == 4) {
    printf("Quatro");
  } else if(numero == 5) {
    printf("Cinco");
  } else {
    printf("Numero Invalido!!!");
  }

  return 0;
}
```

Listagem 3.58: micro06.ll

```
; ModuleID = 'micro06.c'
source_filename = "micro06.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

@.str = private unnamed_addr constant [28 x i8] c"Digite um numero de 1 a
    5: \00", align 1
@.str.1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
```

```
@.str.2 = private unnamed_addr constant [3 x i8] c"Um\00", align 1
@.str.3 = private unnamed_addr constant [5 x i8] c"Dois\00", align 1
@.str.4 = private unnamed_addr constant [5 x i8] c"Tres\00", align 1
@.str.5 = private unnamed_addr constant [7 x i8] c"Quatro\00", align 1
@.str.6 = private unnamed_addr constant [6 x i8] c"Cinco\00", align 1
@.str.7 = private unnamed_addr constant [19 x i8] c"Numero Invalido!!!\00"
    , align 1

; Function Attrs: noinline nounwind optnone ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  %3 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([28 x i8],
      [28 x i8]* @.str, i32 0, i32 0))
  %4 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3
       x i8]* @.str.1, i32 0, i32 0), i32* %2)
  %5 = load i32, i32* %2, align 4
  %6 = icmp eq i32 %5, 1
  br i1 %6, label %7, label %9

; <label>:7:                                      ; preds = %0
  %8 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8],
      [3 x i8]* @.str.2, i32 0, i32 0))
  br label %35

; <label>:9:                                      ; preds = %0
  %10 = load i32, i32* %2, align 4
  %11 = icmp eq i32 %10, 2
  br i1 %11, label %12, label %14

; <label>:12:                                     ; preds = %9
  %13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8],
      [5 x i8]* @.str.3, i32 0, i32 0))
  br label %34

; <label>:14:                                     ; preds = %9
  %15 = load i32, i32* %2, align 4
  %16 = icmp eq i32 %15, 3
  br i1 %16, label %17, label %19

; <label>:17:                                     ; preds = %14
  %18 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8],
      [5 x i8]* @.str.4, i32 0, i32 0))
  br label %33

; <label>:19:                                     ; preds = %14
  %20 = load i32, i32* %2, align 4
  %21 = icmp eq i32 %20, 4
  br i1 %21, label %22, label %24

; <label>:22:                                     ; preds = %19
  %23 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([7 x i8],
      [7 x i8]* @.str.5, i32 0, i32 0))
  br label %32

; <label>:24:                                     ; preds = %19
  %25 = load i32, i32* %2, align 4
  %26 = icmp eq i32 %25, 5
```

```
60   br i1 %26, label %27, label %29
61
62 ; <label>:27:                                     ; preds = %24
63   %28 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([6 x i8],
        [6 x i8]* @.str.6, i32 0, i32 0))
64   br label %31
65
66 ; <label>:29:                                     ; preds = %24
67   %30 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([19 x i8],
         [19 x i8]* @.str.7, i32 0, i32 0))
68   br label %31
69
70 ; <label>:31:                                     ; preds = %29, %27
71   br label %32
72
73 ; <label>:32:                                     ; preds = %31, %22
74   br label %33
75
76 ; <label>:33:                                     ; preds = %32, %17
77   br label %34
78
79 ; <label>:34:                                     ; preds = %33, %12
80   br label %35
81
82 ; <label>:35:                                     ; preds = %34, %7
83   ret i32 0
84 }
85
86 declare i32 @printf(i8*, ...) #1
87
88 declare i32 @scanf(i8*, ...) #1
89
90 attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
      -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
      precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
      -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
      nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
      math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
      target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
      " "unsafe-fp-math"="false" "use-soft-float"="false" }
91 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
      -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
      elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
       "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
      trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
      penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
      ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
92
93 !llvm.module.flags = !{!0, !1}
94 !llvm.ident = !{!2}
95
96 !0 = !{i32 1, !"wchar_size", i32 4}
97 !1 = !{i32 7, !"PIC Level", i32 2}
98 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5.7 Decide se os números são positivos, zeros ou negativos

Listagem 3.59: micro07.py

```
def micro07():
    numero ,programa = 0,1
    opc = ""
    while programa == 1:
        print("Digite um numero: ")
        numero = int(input())
        if numero > 0:
            print("Positivo")
        else:
            if numero == 0:
                print("O numero e igual a 0")
            elif numero < 0:
                print("Negativo")

        print("Deseja Finalizar? (S/N) ")
        opc = input()
        if opc == "S":
            programa = 0

micro07()
```

Listagem 3.60: micro07.c

```
#import<stdio.h>

int main() {
  int numero = 0;
  int programa = 1;
  char opc;

  while (programa == 1) {
    printf("Digite um numero: ");
    scanf("%d", &numero);

    if(numero > 0) {
      printf("Positivo");
    } else {
      if (numero == 0) {
        printf("O numero e igual a 0");
      } else if (numero < 0) {
        printf("Negativo");
      }
    }
    printf("Deseja Finalizar? (S/N) ");
    scanf("%c", &opc);
    if (opc == 'S') {
      programa = 0;
    }
  }
  return 0;
}
```

Listagem 3.61: micro07.ll

```
; ModuleID = 'micro07.c'
source_filename = "micro07.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

```

```
@.str = private unnamed_addr constant [19 x i8] c"Digite um numero: \00", align 1
@.str.1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.str.2 = private unnamed_addr constant [9 x i8] c"Positivo\00", align 1
@.str.3 = private unnamed_addr constant [21 x i8] c"O numero e igual a 0\00", align 1
@.str.4 = private unnamed_addr constant [9 x i8] c"Negativo\00", align 1
@.str.5 = private unnamed_addr constant [25 x i8] c"Deseja Finalizar? (S/N) \00", align 1
@.str.6 = private unnamed_addr constant [3 x i8] c"%c\00", align 1

; Function Attrs: noinline nounwind optnone ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i8, align 1
  store i32 0, i32* %1, align 4
  store i32 0, i32* %2, align 4
  store i32 1, i32* %3, align 4
  br label %5

; <label>:5:                                      ; preds = %34, %0
  %6 = load i32, i32* %3, align 4
  %7 = icmp eq i32 %6, 1
  br i1 %7, label %8, label %35

; <label>:8:                                      ; preds = %5
  %9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([19 x i8], [19 x i8]* @.str, i32 0, i32 0))
  %10 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str.1, i32 0, i32 0), i32* %2)
  %11 = load i32, i32* %2, align 4
  %12 = icmp sgt i32 %11, 0
  br i1 %12, label %13, label %15

; <label>:13:                                     ; preds = %8
  %14 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([9 x i8], [9 x i8]* @.str.2, i32 0, i32 0))
  br label %27

; <label>:15:                                     ; preds = %8
  %16 = load i32, i32* %2, align 4
  %17 = icmp eq i32 %16, 0
  br i1 %17, label %18, label %20

; <label>:18:                                     ; preds = %15
  %19 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([21 x i8], [21 x i8]* @.str.3, i32 0, i32 0))
  br label %26

; <label>:20:                                     ; preds = %15
  %21 = load i32, i32* %2, align 4
  %22 = icmp slt i32 %21, 0
  br i1 %22, label %23, label %25

; <label>:23:                                     ; preds = %20
  %24 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([9 x i8], [9 x i8]* @.str.4, i32 0, i32 0))
```

```
57   br label %25
58
59 ; <label>:25:                                     ; preds = %23, %20
60   br label %26
61
62 ; <label>:26:                                     ; preds = %25, %18
63   br label %27
64
65 ; <label>:27:                                     ; preds = %26, %13
66   %28 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([25 x i8],
         [25 x i8]* @.str.5, i32 0, i32 0))
67   %29 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8],
        [3 x i8]* @.str.6, i32 0, i32 0), i8* %4)
68   %30 = load i8, i8* %4, align 1
69   %31 = sext i8 %30 to i32
70   %32 = icmp eq i32 %31, 83
71   br i1 %32, label %33, label %34
72
73 ; <label>:33:                                     ; preds = %27
74   store i32 0, i32* %3, align 4
75   br label %34
76
77 ; <label>:34:                                     ; preds = %33, %27
78   br label %5
79
80 ; <label>:35:                                     ; preds = %5
81   ret i32 0
82 }
83
84 declare i32 @printf(i8*, ...) #1
85
86 declare i32 @scanf(i8*, ...) #1
87
88 attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
      -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
      precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
      -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
      nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
      math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
      target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
      " "unsafe-fp-math"="false" "use-soft-float"="false" }
89 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
      -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
      elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
       "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
      trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
      penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
      ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
90
91 !llvm.module.flags = !{!0, !1}
92 !llvm.ident = !{!2}
93
94 !0 = !{i32 1, !"wchar_size", i32 4}
95 !1 = !{i32 7, !"PIC Level", i32 2}
96 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5.8 Decide se um número é maior ou menor que 10

Listagem 3.62: micro08.py

```
def micro08():
    numero = 1
    while numero < 0 or numero > 0:
        print("Digite o numero")
        numero = int(input())
        if numero > 10:
            print("O numero "+str(numero)+" e maior que 10")
        else:
            print("O numero "+str(numero)+" e menor que 10")

micro08()
```

Listagem 3.63: micro08.c

```
#import<stdio.h>

int main() {
  int numero = 1;

  while (numero < 0 || numero > 0) {
    printf("Digite um numero: ");
    scanf("%d", &numero);

    if(numero > 10) {
      printf("O numero %d e maior que 10", numero);
    } else {
      printf("O numero %d e menor que 10", numero);
    }
  }
  return 0;
}
```

Listagem 3.64: micro08.ll

```
; ModuleID = 'micro08.c'
source_filename = "micro08.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.13.0"

@.str = private unnamed_addr constant [19 x i8] c"Digite um numero: \00",
    align 1
@.str.1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.str.2 = private unnamed_addr constant [27 x i8] c"O numero %d e maior
    que 10\00", align 1
@.str.3 = private unnamed_addr constant [27 x i8] c"O numero %d e menor
    que 10\00", align 1

; Function Attrs: noinline nounwind optnone ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 1, i32* %2, align 4
  br label %3

; <label>:3:                                      ; preds = %22, %0
  %4 = load i32, i32* %2, align 4
```

```
21   %5 = icmp slt i32 %4, 0
22   br i1 %5, label %9, label %6
23
24 ; <label>:6:                                     ; preds = %3
25   %7 = load i32, i32* %2, align 4
26   %8 = icmp sgt i32 %7, 0
27   br label %9
28
29 ; <label>:9:                                     ; preds = %6, %3
30   %10 = phi i1 [ true, %3 ], [ %8, %6 ]
31   br i1 %10, label %11, label %23
32
33 ; <label>:11:                                    ; preds = %9
34   %12 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([19 x i8],
         [19 x i8]* @.str, i32 0, i32 0))
35   %13 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8],
        [3 x i8]* @.str.1, i32 0, i32 0), i32* %2)
36   %14 = load i32, i32* %2, align 4
37   %15 = icmp sgt i32 %14, 10
38   br i1 %15, label %16, label %19
39
40 ; <label>:16:                                    ; preds = %11
41   %17 = load i32, i32* %2, align 4
42   %18 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([27 x i8],
         [27 x i8]* @.str.2, i32 0, i32 0), i32 %17)
43   br label %22
44
45 ; <label>:19:                                    ; preds = %11
46   %20 = load i32, i32* %2, align 4
47   %21 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([27 x i8],
         [27 x i8]* @.str.3, i32 0, i32 0), i32 %20)
48   br label %22
49
50 ; <label>:22:                                    ; preds = %19, %16
51   br label %3
52
53 ; <label>:23:                                    ; preds = %9
54   ret i32 0
55 }
56
57 declare i32 @printf(i8*, ...) #1
58
59 declare i32 @scanf(i8*, ...) #1
60
61 attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
       -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
       precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
       -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
       nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
       math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
       target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
       " "unsafe-fp-math"="false" "use-soft-float"="false" }
62 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
       -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
       elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
        "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
       trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
       penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
       ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
```

```
63
64 !llvm.module.flags = !{!0, !1}
65 !llvm.ident = !{!2}
66
67 !0 = !{i32 1, !"wchar_size", i32 4}
68 !1 = !{i32 7, !"PIC Level", i32 2}
69 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5.9 Cálculo de preços

Listagem 3.65: micro09.py

```
1 def micro09():
2     preco, venda, novopreco = 0.0,0.0,0.0
3     print("Digite o preco: ")
4     preco = int(input())
5     print("Digite a venda: ")
6     venda = int(input())
7     if venda < 500 or preco <30:
8         novopreco = preco + 10/100 *preco
9     elif (venda >= 500 and venda <1200) or (preco >= 30 and preco <80):
10        novopreco = preco + 15/100 * preco
11    elif venda >=1200 or preco >=80:
12        novopreco = preco - 20/100 * preco
13
14    print("O novo preco e: "+str(novopreco))
15
16 micro09()
```

Listagem 3.66: micro09.c

```
1 #import<stdio.h>
2
3 int main() {
4   float preco, venda, novopreco;
5
6   printf("Digite o preco: ");
7   scanf("%f", &preco);
8
9   printf("Digite o venda: ");
10  scanf("%f", &venda);
11
12  if(venda < 500 || preco < 30) {
13      novopreco = preco + 10.0 / 100.0 * preco;
14  } else if((venda >= 500 && venda <1200) || (preco >= 30 && preco <80)) {
15      novopreco = preco + 15.0 / 100.0 * preco;
16  } else {
17    novopreco = preco + 20.0 / 100.0 * preco;
18  }
19
20  printf("O novo preco e %f", novopreco);
21  return 0;
22 }
```

Listagem 3.67: micro09.ll

```
1 ; ModuleID = 'micro09.c'
```

```
2 source_filename = "micro09.c"
3 target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
4 target triple = "x86_64-apple-macosx10.13.0"
5
6 @.str = private unnamed_addr constant [17 x i8] c"Digite o preco: \00",
      align 1
7 @.str.1 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
8 @.str.2 = private unnamed_addr constant [17 x i8] c"Digite o venda: \00",
      align 1
9 @.str.3 = private unnamed_addr constant [18 x i8] c"O novo preco e %f\00",
       align 1
10
11 ; Function Attrs: noinline nounwind optnone ssp uwtable
12 define i32 @main() #0 {
13   %1 = alloca i32, align 4
14   %2 = alloca float, align 4
15   %3 = alloca float, align 4
16   %4 = alloca float, align 4
17   store i32 0, i32* %1, align 4
18   %5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([17 x i8],
        [17 x i8]* @.str, i32 0, i32 0))
19   %6 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3
         x i8]* @.str.1, i32 0, i32 0), float* %2)
20   %7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([17 x i8],
        [17 x i8]* @.str.2, i32 0, i32 0))
21   %8 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3
         x i8]* @.str.1, i32 0, i32 0), float* %3)
22   %9 = load float, float* %3, align 4
23   %10 = fcmp olt float %9, 5.000000e+02
24   br i1 %10, label %14, label %11
25
26 ; <label>:11:                                     ; preds = %0
27   %12 = load float, float* %2, align 4
28   %13 = fcmp olt float %12, 3.000000e+01
29   br i1 %13, label %14, label %22
30
31 ; <label>:14:                                     ; preds = %11, %0
32   %15 = load float, float* %2, align 4
33   %16 = fpext float %15 to double
34   %17 = load float, float* %2, align 4
35   %18 = fpext float %17 to double
36   %19 = fmul double 1.000000e-01, %18
37   %20 = fadd double %16, %19
38   %21 = fptrunc double %20 to float
39   store float %21, float* %4, align 4
40   br label %51
41
42 ; <label>:22:                                     ; preds = %11
43   %23 = load float, float* %3, align 4
44   %24 = fcmp oge float %23, 5.000000e+02
45   br i1 %24, label %25, label %28
46
47 ; <label>:25:                                     ; preds = %22
48   %26 = load float, float* %3, align 4
49   %27 = fcmp olt float %26, 1.200000e+03
50   br i1 %27, label %34, label %28
51
52 ; <label>:28:                                     ; preds = %25, %22
53   %29 = load float, float* %2, align 4
```

```
54   %30 = fcmp oge float %29, 3.000000e+01
55   br i1 %30, label %31, label %42
56
57 ; <label>:31:                                     ; preds = %28
58   %32 = load float, float* %2, align 4
59   %33 = fcmp olt float %32, 8.000000e+01
60   br i1 %33, label %34, label %42
61
62 ; <label>:34:                                     ; preds = %31, %25
63   %35 = load float, float* %2, align 4
64   %36 = fpext float %35 to double
65   %37 = load float, float* %2, align 4
66   %38 = fpext float %37 to double
67   %39 = fmul double 1.500000e-01, %38
68   %40 = fadd double %36, %39
69   %41 = fptrunc double %40 to float
70   store float %41, float* %4, align 4
71   br label %50
72
73 ; <label>:42:                                     ; preds = %31, %28
74   %43 = load float, float* %2, align 4
75   %44 = fpext float %43 to double
76   %45 = load float, float* %2, align 4
77   %46 = fpext float %45 to double
78   %47 = fmul double 2.000000e-01, %46
79   %48 = fadd double %44, %47
80   %49 = fptrunc double %48 to float
81   store float %49, float* %4, align 4
82   br label %50
83
84 ; <label>:50:                                     ; preds = %42, %34
85   br label %51
86
87 ; <label>:51:                                     ; preds = %50, %14
88   %52 = load float, float* %4, align 4
89   %53 = fpext float %52 to double
90   %54 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([18 x i8],
         [18 x i8]* @.str.3, i32 0, i32 0), double %53)
91   ret i32 0
92 }
93
94 declare i32 @printf(i8*, ...) #1
95
96 declare i32 @scanf(i8*, ...) #1
97
98 attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
     -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
     precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
     -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
     nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
     math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
     target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
     " "unsafe-fp-math"="false" "use-soft-float"="false" }
99 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
     -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
     elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
      "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
     trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
     penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
```

```
    ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }

!llvm.module.flags = !{!0, !1}
!llvm.ident = !{!2}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5.10 Calcula o fatorial de um número

Listagem 3.68: micro10.py

```
def micro10():
    numero = 0
    fat = 0
    print("Digite um numero: ")
    numero = int(input())
    fat = fatorial(numero)
    print("O fatorial de " + str(numero) + " e " + str(fat))

def fatorial(n):
    if n <= 0:
        return 1
    else:
        return (n * fatorial(n-1))

micro10()
```

Listagem 3.69: micro10.c

```
#import<stdio.h>

int fatorial(int n) {
  if (n <= 0) {
    return 1;
  } else {
    return (n * fatorial(n - 1));
  }
}

int main() {
  int numero, fat;

  printf("Digite um numero: ");
  scanf("%d", &numero);

  fat = fatorial(numero);
  printf("O fatorial de %d e %d", numero, fat);

  return 0;
}
```

Listagem 3.70: micro10.ll

```
; ModuleID = 'micro10.c'
source_filename = "micro10.c"
```

```
3 target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
4 target triple = "x86_64-apple-macosx10.13.0"
5
6 @.str = private unnamed_addr constant [19 x i8] c"Digite um numero: \00",
      align 1
7 @.str.1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
8 @.str.2 = private unnamed_addr constant [22 x i8] c"O fatorial de %d e %d
      \00", align 1
9
10 ; Function Attrs: noinline nounwind optnone ssp uwtable
11 define i32 @fatorial(i32) #0 {
12   %2 = alloca i32, align 4
13   %3 = alloca i32, align 4
14   store i32 %0, i32* %3, align 4
15   %4 = load i32, i32* %3, align 4
16   %5 = icmp sle i32 %4, 0
17   br i1 %5, label %6, label %7
18
19 ; <label>:6:                                      ; preds = %1
20   store i32 1, i32* %2, align 4
21   br label %13
22
23 ; <label>:7:                                      ; preds = %1
24   %8 = load i32, i32* %3, align 4
25   %9 = load i32, i32* %3, align 4
26   %10 = sub nsw i32 %9, 1
27   %11 = call i32 @fatorial(i32 %10)
28   %12 = mul nsw i32 %8, %11
29   store i32 %12, i32* %2, align 4
30   br label %13
31
32 ; <label>:13:                                     ; preds = %7, %6
33   %14 = load i32, i32* %2, align 4
34   ret i32 %14
35 }
36
37 ; Function Attrs: noinline nounwind optnone ssp uwtable
38 define i32 @main() #0 {
39   %1 = alloca i32, align 4
40   %2 = alloca i32, align 4
41   %3 = alloca i32, align 4
42   store i32 0, i32* %1, align 4
43   %4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([19 x i8],
        [19 x i8]* @.str, i32 0, i32 0))
44   %5 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3
         x i8]* @.str.1, i32 0, i32 0), i32* %2)
45   %6 = load i32, i32* %2, align 4
46   %7 = call i32 @fatorial(i32 %6)
47   store i32 %7, i32* %3, align 4
48   %8 = load i32, i32* %2, align 4
49   %9 = load i32, i32* %3, align 4
50   %10 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([22 x i8],
         [22 x i8]* @.str.2, i32 0, i32 0), i32 %8, i32 %9)
51   ret i32 0
52 }
53
54 declare i32 @printf(i8*, ...) #1
55
56 declare i32 @scanf(i8*, ...) #1
```

```
57
58 attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
      -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
      precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
      -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
      nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
      math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
      target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
      " "unsafe-fp-math"="false" "use-soft-float"="false" }
59 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
      -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
      elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
       "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
      trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
      penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
      ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
60
61 !llvm.module.flags = !{!0, !1}
62 !llvm.ident = !{!2}
63
64 !0 = !{i32 1, !"wchar_size", i32 4}
65 !1 = !{i32 7, !"PIC Level", i32 2}
66 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

3.5.11 Decide se um número é positivo, zero ou negativo com auxílio de uma função

Listagem 3.71: micro11.py

```
1 def micro11():
2     numero,x = 0,0
3     print("Digite um numero: ")
4     numero = int(input())
5     x = verifica(numero)
6     if x ==1:
7         print("Numero Positivo")
8     elif x ==0:
9         print("Zero")
10    else:
11        print("Negativo")
12
13 def verifica(n):
14    res = 0
15    if n > 0:
16        res = 1
17    elif n < 0:
18        res = -1
19    else:
20        res = 0
21
22    return res
23
24 micro11()
```

Listagem 3.72: micro11.c

```
1 #import<stdio.h>
```

```
2
3  int verifica(int n) {
4    int res;
5    if (n > 0) {
6      res = 1;
7    } else if (n < 0) {
8      res = -1;
9    } else {
10     res = 0;
11   }
12   return res;
13 }
14
15 int main() {
16   int numero, x;
17
18   printf("Digite um numero: ");
19   scanf("%d", &numero);
20
21   x = verifica(numero);
22   if(x == 1) {
23     printf("Numero Positivo");
24   } else if(x == 0) {
25     printf("Zero");
26   } else {
27     printf("Negativo");
28   }
29
30   return 0;
31 }
```

Listagem 3.73: micro11.ll

```
1  ; ModuleID = 'micro11.c'
2  source_filename = "micro11.c"
3  target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
4  target triple = "x86_64-apple-macosx10.13.0"
5
6  @.str = private unnamed_addr constant [19 x i8] c"Digite um numero: \00",
       align 1
7  @.str.1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
8  @.str.2 = private unnamed_addr constant [16 x i8] c"Numero Positivo\00",
       align 1
9  @.str.3 = private unnamed_addr constant [5 x i8] c"Zero\00", align 1
10 @.str.4 = private unnamed_addr constant [9 x i8] c"Negativo\00", align 1
11
12 ; Function Attrs: noinline nounwind optnone ssp uwtable
13 define i32 @verifica(i32) #0 {
14   %2 = alloca i32, align 4
15   %3 = alloca i32, align 4
16   store i32 %0, i32* %2, align 4
17   %4 = load i32, i32* %2, align 4
18   %5 = icmp sgt i32 %4, 0
19   br i1 %5, label %6, label %7
20
21 ; <label>:6:                                      ; preds = %1
22   store i32 1, i32* %3, align 4
23   br label %13
24
```

```
25 ; <label>:7:                                     ; preds = %1
26   %8 = load i32, i32* %2, align 4
27   %9 = icmp slt i32 %8, 0
28   br i1 %9, label %10, label %11
29
30 ; <label>:10:                                    ; preds = %7
31   store i32 -1, i32* %3, align 4
32   br label %12
33
34 ; <label>:11:                                    ; preds = %7
35   store i32 0, i32* %3, align 4
36   br label %12
37
38 ; <label>:12:                                    ; preds = %11, %10
39   br label %13
40
41 ; <label>:13:                                    ; preds = %12, %6
42   %14 = load i32, i32* %3, align 4
43   ret i32 %14
44 }
45
46 ; Function Attrs: noinline nounwind optnone ssp uwtable
47 define i32 @main() #0 {
48   %1 = alloca i32, align 4
49   %2 = alloca i32, align 4
50   %3 = alloca i32, align 4
51   store i32 0, i32* %1, align 4
52   %4 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([19 x i8],
       [19 x i8]* @.str, i32 0, i32 0))
53   %5 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3
        x i8]* @.str.1, i32 0, i32 0), i32* %2)
54   %6 = load i32, i32* %2, align 4
55   %7 = call i32 @verifica(i32 %6)
56   store i32 %7, i32* %3, align 4
57   %8 = load i32, i32* %3, align 4
58   %9 = icmp eq i32 %8, 1
59   br i1 %9, label %10, label %12
60
61 ; <label>:10:                                    ; preds = %0
62   %11 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([16 x i8],
        [16 x i8]* @.str.2, i32 0, i32 0))
63   br label %20
64
65 ; <label>:12:                                    ; preds = %0
66   %13 = load i32, i32* %3, align 4
67   %14 = icmp eq i32 %13, 0
68   br i1 %14, label %15, label %17
69
70 ; <label>:15:                                    ; preds = %12
71   %16 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8],
       [5 x i8]* @.str.3, i32 0, i32 0))
72   br label %19
73
74 ; <label>:17:                                    ; preds = %12
75   %18 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([9 x i8],
       [9 x i8]* @.str.4, i32 0, i32 0))
76   br label %19
77
78 ; <label>:19:                                    ; preds = %17, %15
```

```
79   br label %20
80
81 ; <label>:20:                                     ; preds = %19, %10
82   ret i32 0
83 }
84
85 declare i32 @printf(i8*, ...) #1
86
87 declare i32 @scanf(i8*, ...) #1
88
89 attributes #0 = { noinline nounwind optnone ssp uwtable "correctly-rounded
     -divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-
     precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer
     -elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-
     nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-
     math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "
     target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87
     " "unsafe-fp-math"="false" "use-soft-float"="false" }
90 attributes #1 = { "correctly-rounded-divide-sqrt-fp-math"="false" "disable
     -tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-
     elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false"
      "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-
     trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="
     penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+
     ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
91
92 !llvm.module.flags = !{!0, !1}
93 !llvm.ident = !{!2}
94
95 !0 = !{i32 1, !"wchar_size", i32 4}
96 !1 = !{i32 7, !"PIC Level", i32 2}
97 !2 = !{!"Apple LLVM version 9.1.0 (clang-902.0.39.2)"}
```

Capítulo 4

Compilador

Um compilador traduz um programa de uma linguagem textual facilmente entendida por um ser humano para uma linguagem de máquina, específica para um processador e sistema operacional.

Para compilar fazemos:

Listagem 4.1: Terminal

```
$ ocamllex lexico.mll
$ ocamlc -c lexico.ml
$ ocamlc -c pre_processador.ml
$ rlwrap ocaml
# #use "carregador.ml";;
# lex "file.py";;
```

4.1 Analisador Léxico

Listagem 4.2: lexico.mll

```
{
  open Lexing
  open Printf

  type token =
  | LITINT of (int)
  | LITFLOAT of (float)
  | LITSTRING of (string)
  | ID of (string)
  | APAR
  | ACOL
  | ACHA
  | FPAR
  | FCOL
  | FCHA
  | VIRG
  | SOMA
  | SUBTRAI
  | DIVIDE
```

```
20    | MULTIPLICA
21    | DOIS_PONTOS
22    | MENOR
23    | MAIOR
24    | MENOR_IGUAL
25    | MAIOR_IGUAL
26    | IGUAL
27    | DIFERENTE
28    | AND
29    | OR
30    | IF
31    | ELSE_IF
32    | ELSE
33    | NOT
34    | TIPO_INT
35    | TIPO_FLOAT
36    | TIPO_STRING
37    | TIPO_BOOL
38    | TIPO_VOID
39    | SETA
40    | TRUE
41    | FALSE
42    | FOR
43    | WHILE
44    | IN
45    | IS
46    | FROM
47    | RANGE
48    | PRINT
49    | INPUT
50    | INT_PARSE
51    | E
52    | ATRIB
53    | ARTIB_SOMA
54    | ARTIB_SUB
55    | ARTIB_MULT
56    | ARTIB_DIV
57    | ARTIB_MOD
58    | MODULO
59    | VIRGULA
60    | PONTO_VIRGULA
61    | PONTO
62    | RETURN
63    | DEF
64    | EOF
65    (* Os tokens a seguir são importantes para o pré processador *)
66    | Linha of (int * int * token list)
67    | INDENTA
68    | DEDENTA
69    | NOVALINHA
70
71  (* contador de nivel de parentizacao - utilizado para identacao *)
72    let nivel_par = ref 0
73
74    (* incrementa a contagem de linhas *)
75    let incr_nlinha lexbuf =
76      let pos = lexbuf.lex_curr_p in
77        lexbuf.lex_curr_p <- { pos with
78                               pos_lnum = pos.pos_lnum + 1;
```

```
79                              pos_bol = pos.pos_cnum;
80                             }
81
82   (* imprime mensagem de erro *)
83   let msg_erro lexbuf c =
84     let pos = lexbuf.lex_curr_p in
85       let lin = pos.pos_lnum
86       and col = pos.pos_cnum - pos.pos_bol - 1 in
87       sprintf "%d-%d: Caracter Desconhecido %c" lin col c
88
89
90   (* cria tabela hasg *)
91   let cria_tab_hash iniciais =
92     let tbl = Hashtbl.create (List.length iniciais) in
93       List.iter (fun (chave, valor) -> Hashtbl.add tbl chave valor)
94         iniciais;
95       tbl
96
97   (* palavras reservadas *)
98   let plv_res =
99     cria_tab_hash
100    [
101    ("def", DEF);
102    ("else", ELSE );
103    ("for", FOR);
104    ("if", IF);
105    ("in", IN);
106    ("not", NOT);
107    ("and", AND);
108    ("or", OR);
109    ("is", IS);
110    ("from", FROM);
111    ("return", RETURN);
112    ("while", WHILE);
113    ("range", RANGE);
114    ("print", PRINT);
115    ("int", TIPO_INT);
116    ("None", TIPO_VOID);
117    ("str", TIPO_STRING);
118    ("float", TIPO_FLOAT);
119    ("bool", TIPO_BOOL);
120    ("raw_input", INPUT);
121    ("int_parse", INT_PARSE)
122    ]
123
124  (* Valores booleanos sao armazenados como 1 para true e 0 para false. *)
125  (* Operacoes com booleanos sao transformadas em operacoes com inteiros
       *)
126  let booleano nbool =
127    match nbool with
128    | "True" -> 1
129    | "False" -> 0
130    | _ -> failwith "Erro: nao eh valor booleano"
131 }
132
133 (* definicoes *)
134 let digito = ['0' - '9']
135 let letra = ['a'-'z' 'A'-'Z']
136 let id = letra ( letra | digito | '_' )*
```

```
137 let comentario = '#' [^ '\n']*
138 let comentario2 = '"' '"' '"' [^ '\n']* '"' '"' '"' | "'" "'" "'" [^ '\n
      ']* "'" "'" "'"
139 let linha_em_branco = [' ' '\t' ]* comentario | [' ' '\t' ]* comentario
140 let restante = [^ ' ' '\t' '\n' ] [^ '\n']+
141 let boolean = "True" | "False"
142 let strings = '"' id* digito* '"' | "'" id* digito* "'"
143 let floats = digito+ '.' digito+
144 let neg = '-' digito+
145
146 (* regras para identificar identacao para gerar tokens de abre e fecha
      escopo *)
147 rule preprocessador indentacao = parse
148   linha_em_branco { preprocessador 0 lexbuf } (* ignora brancos *)
149   | [' ' '\t' ]+ '\n'    { incr_nlinha lexbuf;
150                            preprocessador 0 lexbuf } (* ignora brancos *)
151   | ' '                  { preprocessador (indentacao + 1) lexbuf }
152   | '\t'                 { let nova_ind = indentacao + 8 - (indentacao mod
         8) in
153                            preprocessador nova_ind lexbuf }
154   | '\n'                 { incr_nlinha lexbuf;
155                            preprocessador 0 lexbuf }
156   | eof                  { nivel_par := 0; EOF }
157   | restante as linha    {
158                            let rec tokenize lexbuf =
159                            let tok = token lexbuf in
160                            match tok with
161                              EOF -> []
162                              | _ -> tok :: tokenize lexbuf in
163                            let toks = tokenize (Lexing.from_string linha)
                                 in
164                            Linha(indentacao,!nivel_par, toks)
165                          }
166
167 (* identificacao dos tokens *)
168 and token = parse
169 | ' '
170 | '\t'
171 | comentario             { token lexbuf }
172 | comentario2            { comentario_bloco 0 lexbuf; }
173 | digito+ as numint      { let num = int_of_string numint in LITINT num }
174 | neg as numNeg          { let num = int_of_string numNeg in LITINT num }
175 | digito+ '.' digito+ as numfloat { let num = float_of_string numfloat in
176                                     LITFLOAT num }
177 | boolean as nbool       { LITINT (booleano nbool)}
178 | floats as numfloat     { LITFLOAT (float_of_string numfloat) }
179 | id as palavra          { try Hashtbl.find plv_res palavra
180                            with Not_found -> ID (palavra)}
181 | '"'                    { let buffer = Buffer.create 1 in
182                            LITSTRING (cadeia buffer lexbuf) }
183 | '='                    { ATRIB }
184 | '+'                    { SOMA }
185 | '-'                    { SUBTRAI }
186 | '/'                    { DIVIDE }
187 | '*'                    { MULTIPLICA }
188 | ':'                    { DOIS_PONTOS }
189 | '('                    { APAR }
190 | ')'                    { FPAR }
191 | ';'                    { PONTO_VIRGULA }
```

```
192 | ','                   { VIRGULA }
193 | "=="                   { IGUAL }
194 | '>'                    { MAIOR }
195 | '<'                    { MENOR }
196 | ">="                   { MAIOR_IGUAL }
197 | "<="                   { MENOR_IGUAL }
198 | "!="                   { DIFERENTE }
199 | "and"                  { AND }
200 | "or"                   { OR }
201 | "if"                   { IF }
202 | "else"                 { ELSE }
203 | "not"                  { NOT }
204 | "int"                  { TIPO_INT }
205 | "float"                { TIPO_FLOAT }
206 | "str"                  { TIPO_STRING }
207 | "bool"                 { TIPO_BOOL }
208 | "None"                 { TIPO_VOID }
209 | "def"                  { DEF }
210 | "->"                   { SETA } (* tipo funcao *)
211 | "True"                 { TRUE }
212 | "False"                { FALSE }
213 | "for"                  { FOR }
214 | "while"                { WHILE }
215 | "in"                   { IN }
216 | "range"                { RANGE }
217 | "int_parse"            { INT_PARSE }
218 | "print"                { PRINT }
219 | "+="                   { ARTIB_SOMA }
220 | "-="                   { ARTIB_SUB }
221 | "*="                   { ARTIB_MULT }
222 | "/="                   { ARTIB_DIV }
223 | "%="                   { ARTIB_MOD }
224 | "%"                    { MODULO }
225 | '('                    { incr(nivel_par); APAR }
226 | '['                    { incr(nivel_par); ACOL }
227 | '{'                    { incr(nivel_par); ACHA }
228 | ')'                    { decr(nivel_par); FPAR }
229 | ']'                    { decr(nivel_par); FCOL }
230 | '}'                    { decr(nivel_par); FCHA }
231 | ':'                    { DOIS_PONTOS }
232 | ','                    { VIRG }
233 | ';'                    { PONTO_VIRGULA }
234 | '.'                    { PONTO }
235 | eof                    { EOF }
236 | _ as c                 { failwith (msg_erro lexbuf c); }
237
238 (* para criar comentario de bloco *)
239 and comentario_bloco n = parse
240 | '"' '"' '"' | "'" "'" "'" { if n=0 then token lexbuf
241                             else comentario_bloco (n - 1) lexbuf }
242 | '"' '"' '"' | "'" "'" "'"    { comentario_bloco (n + 1) lexbuf; }
243 | _                      { comentario_bloco n lexbuf }
244 | eof                    { failwith "Comentário não fechado" }
245
246 (* para criar cadeias de strings *)
247 and cadeia buffer = parse
248 | '"'        { Buffer.contents buffer }
249 | "\\t"      { Buffer.add_char buffer '\t'; cadeia buffer lexbuf }
250 | "\\n"      { Buffer.add_char buffer '\n'; cadeia buffer lexbuf }
```

```
251 | '\\' '"'   { Buffer.add_char buffer '"'; cadeia buffer lexbuf }
252 | '\\' '\\' { Buffer.add_char buffer '\\'; cadeia buffer lexbuf }
253 | eof        { failwith "string não foi fechada" }
254 | _ as c     { Buffer.add_char buffer c; cadeia buffer lexbuf }
```

Listagem 4.3: pre$_p$*rocessador.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 *)
```

```
let lexico =
  let tokbuf = ref None in
  let carrega lexbuf =
    let toks = preprocessa lexbuf in
    (match toks with
       tok::toks ->
       tokbuf := Some toks;
       tok
     | [] -> print_endline "EOF";
       EOF)
  in
  fun lexbuf ->
  match !tokbuf with
    Some tokens ->
    (match tokens with
       tok::toks ->
       tokbuf := Some toks;
       tok
     | [] -> carrega lexbuf)
  | None -> carrega lexbuf
```

Listagem 4.4: carregador.ml

```
(* Para compilar:
     ocamllex lexico.mll
     ocamlc -c lexico.ml
     ocamlc -c pre_processador.ml

   Para usar:
     rlwrap ocaml

     #use "carregador.ml";;
     lex "teste.py";;
*)

#load "lexico.cmo"
#load "pre_processador.cmo"

type nome_arq = string
type tokens = Lexico.token list

let rec tokens lexbuf =
  let tok = Pre_processador.lexico lexbuf in
  match tok with
  | Lexico.EOF -> ([Lexico.EOF]:tokens)
  | _ -> tok :: tokens lexbuf
;;

let lexico str =
  let lexbuf = Lexing.from_string str in
  tokens lexbuf
;;

let lex (arq:nome_arq) =
  let ic = open_in arq in
  let lexbuf = Lexing.from_channel ic in
  let toks = tokens lexbuf in
  let _ = close_in ic in
  toks
```

4.1.1 Teste

Para compilar fazemos:

Listagem 4.5: Terminal

```
$ ocamllex lexico.mll
$ ocamlc -c lexico.ml
$ ocamlc -c pre_processador.ml
$ rlwrap ocaml
        OCaml version 4.07.0

# #use "carregador.ml";;
type nome_arq = string
type tokens = Lexico.token list
val tokens : Lexing.lexbuf -> tokens = <fun>
val lexico : string -> tokens = <fun>
val lex : nome_arq -> tokens = <fun>
# lex "tests/micro10.py";;
Linha(identacao=0,nivel_par=0)
Nivel: 0
Linha(identacao=4,nivel_par=0)
Nivel: 4
Linha(identacao=4,nivel_par=0)
Nivel: 4
Linha(identacao=4,nivel_par=0)
Nivel: 4
Linha(identacao=4,nivel_par=0)
Nivel: 4
Linha(identacao=4,nivel_par=0)
Nivel: 4
Linha(identacao=4,nivel_par=0)
Nivel: 4
Linha(identacao=4,nivel_par=0)
Nivel: 4
Linha(identacao=0,nivel_par=0)
Nivel: 0
Linha(identacao=4,nivel_par=0)
Nivel: 4
Linha(identacao=8,nivel_par=0)
Nivel: 8
Linha(identacao=4,nivel_par=0)
Nivel: 4
Linha(identacao=8,nivel_par=0)
Nivel: 8
EOF
- : tokens =
[Lexico.DEF; Lexico.ID "micro10"; Lexico.APAR; Lexico.FPAR; Lexico.DPONTOS
    ;
 Lexico.NOVALINHA; Lexico.INDENTA; Lexico.ID "numero"; Lexico.ATRIB;
 Lexico.LITINT 0; Lexico.NOVALINHA; Lexico.ID "fat"; Lexico.ATRIB;
 Lexico.LITINT 0; Lexico.NOVALINHA; Lexico.PRINT; Lexico.APAR;
 Lexico.LITSTRING "Digite um numero: "; Lexico.FPAR; Lexico.NOVALINHA;
 Lexico.ID "numero"; Lexico.ATRIB; Lexico.INT; Lexico.APAR; Lexico.INPUT;
 Lexico.APAR; Lexico.FPAR; Lexico.FPAR; Lexico.NOVALINHA; Lexico.ID "fat";
 Lexico.ATRIB; Lexico.ID "fatorial"; Lexico.APAR; Lexico.ID "numero";
 Lexico.FPAR; Lexico.NOVALINHA; Lexico.PRINT; Lexico.APAR;
 Lexico.LITSTRING "O fatorial de "; Lexico.MAIS; Lexico.STR; Lexico.APAR;
 Lexico.ID "numero"; Lexico.FPAR; Lexico.MAIS; Lexico.LITSTRING " e ";
 Lexico.MAIS; Lexico.STR; Lexico.APAR; Lexico.ID "fat"; Lexico.FPAR;
 Lexico.FPAR; Lexico.NOVALINHA; Lexico.DEDENTA; Lexico.DEF;
```

```
53 Lexico.ID "fatorial"; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR;
54 Lexico.DPONTOS; Lexico.NOVALINHA; Lexico.INDENTA; Lexico.IF; Lexico.ID "n
       ";
55 Lexico.MENORIGUAL; Lexico.LITINT 0; Lexico.DPONTOS; Lexico.NOVALINHA;
56 Lexico.INDENTA; Lexico.RETURN; Lexico.LITINT 1; Lexico.NOVALINHA;
57 Lexico.DEDENTA; Lexico.ELSE; Lexico.DPONTOS; Lexico.NOVALINHA;
58 Lexico.INDENTA; Lexico.RETURN; Lexico.APAR; Lexico.ID "n"; Lexico.VEZES;
59 Lexico.ID "fatorial"; Lexico.APAR; Lexico.ID "n"; Lexico.MENOS;
60 Lexico.LITINT 1; Lexico.FPAR; Lexico.FPAR; Lexico.NOVALINHA; Lexico.
       DEDENTA;
61 Lexico.DEDENTA; Lexico.EOF]
```

4.2 Analisador Sintático

Listagem 4.6: sintatico.mly

```
1  %{
2    open Ast
3    open Sast
4  %}
5
6  %token <int * int * token list> Linha
7  %token <float * Lexing.position> LITFLOAT
8  %token <string *Lexing.position > ID
9  %token <string *Lexing.position > LITSTRING
10 %token <int * Lexing.position> LITINT
11 %token <bool * Lexing.position>   LITBOOL
12 %token <Lexing.position> DEF SETA DPONTOS
13 %token <Lexing.position> VIRG
14 %token <Lexing.position> ATRIB MAIOR MAIORIGUAL MENOR MENORIGUAL DIFERENTE
       IGUALDADE
15 %token <Lexing.position> OU E NOT MAIS MENOS DIVIDIDO VEZES MODULO
16 %token <Lexing.position> APAR FPAR
17 %token <Lexing.position> PRINT
18 %token <Lexing.position> INPUTI INPUTF INPUTS
19 %token <Lexing.position> WHILE FOR IN RANGE
20 %token <Lexing.position> IF ELIF ELSE
21 %token <Lexing.position> RETURN
22 %token <Lexing.position> NONE
23 %token <Lexing.position> STR
24 %token <Lexing.position> INT
25 %token <Lexing.position> FLOAT
26 %token <Lexing.position> BOOL
27 %token INDENTA DEDENTA NOVALINHA EOF
28
29 %left OU
30 %left E
31 %left IGUALDADE DIFERENTE
32 %left MAIOR MAIORIGUAL MENOR MENORIGUAL
33 %left MAIS MENOS
34 %left VEZES DIVIDIDO MODULO
35
36 %nonassoc unary_minus
37
38 %start <Sast.expressao Ast.programa> programa
39
```

```
%%

programa: ins=instrucao*
     EOF
     {Programa ins }

funcao:
     DEF nome= ID
     APAR args = separated_list(VIRG, parametro) FPAR
     SETA retorno = tipo DPONTOS NOVALINHA
     INDENTA
     cmd = comandos
     DEDENTA
     {
      Funcao {
        fn_nome = nome;
        fn_tiporet = retorno;
        fn_formais = args;
        fn_corpo = cmd
      }
     }


parametro:
    |  id = ID  DPONTOS tp = tipo { (id,tp) }


 /*esse eh o meu stm_block */
instrucao:
  | func = funcao         {     func  }
  | cmd = comando     { ACMD(cmd) }


comandos:
  cmd = comando+ { cmd }

/*esse eh o meu stm_list*/
comando:
  | stm = atribuicao          { stm }
  | stm = chamadafuncao         { stm }
  | stm = loopWhile         { stm }
  | stm = condicaoIF            { stm }
  | stm = loopFOR           { stm }
  | stm = print             { stm }
  | stm = retorno           { stm }
  | stm = leiai NOVALINHA       { stm }
  | stm = leiaf NOVALINHA       { stm }
  | stm = leias NOVALINHA       { stm }
  ;

retorno:
  | RETURN expr = exprLogicoAritmetica? NOVALINHA  { RETORNO(expr) }
  ;

print:
    | PRINT exprla = exprLogicoAritmetica NOVALINHA {PRINT(exprla) }
  ;
/*a sacada eh emcapsular tudo dentro de expressao*/

```

```
99  chamadafuncao:
100   | exp=chamada NOVALINHA  { CHAMADADEFUNCAO(exp) }
101   ;
102
103 chamada : nome=ID APAR args=separated_list(VIRG, exprLogicoAritmetica)
        FPAR { EXPCALL (nome, args) }
104
105 condicaoIF:
106   | IF exprla= exprLogicoAritmetica  DPONTOS NOVALINHA
107     INDENTA stm=comandos DEDENTA
108     cee = condicaoELIFELSE?
109       { CONDICAOIF(exprla,stm,cee) }
110
111
112 condicaoELIFELSE:
113   | ELIF exprla = exprLogicoAritmetica DPONTOS NOVALINHA INDENTA stm =
          comandos DEDENTA condEE = condicaoELIFELSE? { CONDICAOIF (exprla,stm,
           condEE) }
114   | ELSE DPONTOS NOVALINHA INDENTA stm=comandos DEDENTA {CONDICAOElifElse(
           stm ) }
115   ;
116
117 atribuicao: id = ID ATRIB exprla = exprLogicoAritmetica NOVALINHA   {
        ATRIBUICAO (EXPVAR id , exprla) }
118
119 leiai: INPUTI exp=exprLogicoAritmetica  { LEIAI exp }
120 leiaf: INPUTF exp=exprLogicoAritmetica  { LEIAF exp }
121 leias: INPUTS exp=exprLogicoAritmetica  { LEIAS exp }
122
123 loopFOR:
124   | FOR expid=exprLogicoAritmetica IN RANGE APAR exprcomeco =
          exprLogicoAritmetica VIRG exprfim = exprLogicoAritmetica FPAR DPONTOS
           NOVALINHA INDENTA stm = comandos DEDENTA  { FORLOOP(expid,exprcomeco
          ,exprfim,stm)}
125   ;
126
127 loopWhile: WHILE exprla = exprLogicoAritmetica DPONTOS NOVALINHA INDENTA
        stm = comandos DEDENTA { WHILELOOP(exprla,stm) }
128
129 exprLogicoAritmetica:
130   | f = chamada                          { f          }
131   | id = ID                            { EXPVAR(id)       }
132   | i = LITINT                         { EXPINT(i)        }
133   | s = LITSTRING                        { EXPSTRING(s)     }
134   | f = LITFLOAT                         { EXPFLOAT(f)        }
135   | b = LITBOOL                        { EXPBOOL (b)    }
136   | op=opU e=exprLogicoAritmetica %prec unary_minus         { EXPOPU (op,e)
            }
137   | e1=exprLogicoAritmetica op = opB e2 = exprLogicoAritmetica { EXPOPB (
          op,e1,e2) }
138   | APAR e=exprLogicoAritmetica FPAR               { e            }
139   ;
140
141 tipo:
142   | BOOL      { BOOLEAN   }
143   | INT       { INTEIRO   }
144   | FLOAT     { REAL    }
145   | NONE      { NONE    }
146   | STR           { STRING  }
```

```
147   ;
148
149 %inline opB:
150     | pos = MAIS            { (ADICAO, pos) }
151     | pos = MENOS             { (SUBTRACAO, pos)  }
152     | pos = VEZES             { (MULTIPLICACAO,pos) }
153     | pos = DIVIDIDO          { (DIVISAO, pos)    }
154     | pos = MODULO          { (MOD,   pos)    }
155     | pos = IGUALDADE           { (EHIGUAL, pos)    }
156     | pos = MAIOR             { (MAIORQ,  pos)  }
157     | pos = MAIORIGUAL        { (MAIORIGUALQ, pos)  }
158     | pos = MENOR           { (MENORQ,  pos)  }
159     | pos = MENORIGUAL          { (MENORIGUALQ, pos)}
160     | pos = DIFERENTE         { (EHDIFERENTE, pos)  }
161     | pos = E             { (AND,   pos)    }
162     | pos = OU            { (OR,    pos)  }
163     ;
164
165 %inline opU:
166   | pos = NOT   { (NEGACAO, pos)    }
167   | pos = MENOS   { (SUBTRACAO, pos ) }
168   ;
```

Listagem 4.7: ast.ml

```
1  (* The type of the abstract syntax tree (AST). *)
2
3  type identificador = string
4  (*posicao no arquivo*)
5  type 'a pos = 'a * Lexing.position
6
7  type 'expr programa = Programa of 'expr instrucoes
8  and 'expr comandos = 'expr comando list
9  and 'expr instrucoes = 'expr instrucao list
10 and 'expr expressoes = 'expr list
11 and 'expr instrucao =
12   Funcao of 'expr decfn
13   | ACMD of 'expr comando
14 and 'expr decfn = {
15   fn_nome: identificador pos;
16   fn_tiporet: tipo;
17   fn_formais: (identificador pos * tipo) list;
18   fn_corpo: 'expr comandos
19 }
20
21 and tipo =
22         BOOLEAN
23       | INTEIRO
24       | REAL
25       | NONE
26       | STRING
27
28 and 'expr comando =
29           ATRIBUICAO of 'expr * 'expr
30         | CONDICAOIF of 'expr * ('expr comando) list * ('expr comando
             option)
31         | CONDICAOElifElse of 'expr comandos
32         | WHILELOOP of 'expr * ('expr comando) list
33         | FORLOOP of 'expr * 'expr * 'expr * ('expr comando) list
```

```
34          | PRINT of 'expr
35          | RETORNO of 'expr option
36          | LEIAI of 'expr
37          | LEIAF of 'expr
38          | LEIAS of 'expr
39          | CHAMADADEFUNCAO of 'expr
40
41 and operador =
42           ADICAO
43           | SUBTRACAO
44           | MULTIPLICACAO
45           | DIVISAO
46           | MOD
47           | EHIGUAL
48           | MAIORQ
49           | MAIORIGUALQ
50           | MENORQ
51           | MENORIGUALQ
52           | EHDIFERENTE
53           | AND
54           | OR
55           | NEGACAO
```

4.2.1 Teste

Para compilar fazemos:

Listagem 4.8: Terminal

```
1 $ ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -package
     menhirLib sintaticoTeste.byte
2 Finished, 40 targets (37 cached) in 00:00:00.
3 $ rlwrap ocaml
4         OCaml version 4.07.0
5
6 Findlib has been successfully loaded. Additional directives:
7   #require "package";;      to load a package
8   #list;;                   to list the available packages
9   #camlp4o;;                to load camlp4 (standard syntax)
10  #camlp4r;;                to load camlp4 (revised syntax)
11  #predicates "p,q,...";;   to set these predicates
12  Topfind.reset();;         to force that packages will be reloaded
13  #thread;;                 to enable threads
14
15 /Users/tatianefx/.opam/default/lib/menhirLib: added to search path
16 /Users/tatianefx/.opam/default/lib/menhirLib/menhirLib.cmo: loaded
17 # parse_arq "tests/micro10.py";;
```

Listagem 4.9: testeSintatico.txt

```
1 EOF
2 - : Sast.expressao Ast.programa option option =
3 Some
4  (Some
5    (Programa
6      [Funcao
7        {fn_nome =
8          ("main",
```

```
9           {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum =
              4});
10        fn_tiporet = NONE; fn_formais = [];
11        fn_corpo =
12         [ATRIBUICAO
13           (Sast.EXPVAR
14             ("numero",
15              {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
16               pos_cnum = 0}),
17            Sast.EXPINT
18             (0,
19              {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum =
                  8}));
20          ATRIBUICAO
21           (Sast.EXPVAR
22             ("fat",
23              {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
24               pos_cnum = 0}),
25            Sast.EXPINT
26             (0,
27              {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum =
                  6}));
28          PRINT
29           (Sast.EXPSTRING
30             ("Digite um numero: ",
31              {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
32               pos_cnum = 25}));
33          LEIAI
34           (Sast.EXPVAR
35             ("numero",
36              {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
37               pos_cnum = 7}));
38          ATRIBUICAO
39           (Sast.EXPVAR
40             ("fat",
41              {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
42               pos_cnum = 0}),
43            Sast.EXPCALL
44             (("fatorial",
45               {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
46                pos_cnum = 6}),
47              [Sast.EXPVAR
48                ("numero",
49                 {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
50                  pos_cnum = 15})]));
51          PRINT
52           (Sast.EXPSTRING
53             ("O fatorial eh ",
54              {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
55               pos_cnum = 21}));
56          PRINT
57           (Sast.EXPVAR
58             ("fat",
59              {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
60               pos_cnum = 6}))]};
61      Funcao
62       {fn_nome =
63         ("fatorial",
64          {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum =
```

```
             4});
65         fn_tiporet = INTEIRO;
66         fn_formais =
67          [(("n",
68             {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum =
                  13}),
69            INTEIRO)];
70         fn_corpo =
71          [CONDICAOIF
72            (Sast.EXPOPB
73              ((MENORIGUALQ,
74                {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
75                 pos_cnum = 5}),
76               Sast.EXPVAR
77                ("n",
78                 {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
79                  pos_cnum = 3}),
80               Sast.EXPINT
81                (0,
82                 {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
83                  pos_cnum = 8})),
84            [RETORNO
85              (Some
86                (Sast.EXPINT
87                  (1,
88                   {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
89                    pos_cnum = 7}))))],
90            Some
91             (CONDICAOElifElse
92               [RETORNO
93                 (Some
94                   (Sast.EXPOPB
95                     ((MULTIPLICACAO,
96                       {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
97                        pos_cnum = 9}),
98                      Sast.EXPVAR
99                       ("n",
100                       {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
101                        pos_cnum = 7}),
102                     Sast.EXPCALL
103                      (("fatorial",
104                        {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0;
105                         pos_cnum = 11}),
106                       [Sast.EXPOPB
107                         ((SUBTRACAO,
108                           {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol =
                                0;
109                            pos_cnum = 22}),
110                          Sast.EXPVAR
111                           ("n",
112                           {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol =
                                0;
113                            pos_cnum = 20}),
114                          Sast.EXPINT
115                           (1,
116                           {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol =
                                0;
117                            pos_cnum = 24}))])))))])]};
118      ACMD
```

```
119         (CHAMADADEFUNCAO
120           (Sast.EXPCALL
121             (("main",
122               {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum =
                      0}),
123             [])))]))
```

4.3 Analisador Semântico

Listagem 4.10: semantico.ml

```
1  module Amb = Ambiente
2  module A = Ast
3  module S = Sast
4  module T = Tast
5
6  let rec posicao exp =
7    let open S in
8    match exp with
9    | EXPVAR       (_,pos)       -> pos
10   | EXPINT       (_,pos)       -> pos
11   | EXPSTRING    (_,pos)       -> pos
12   | EXPBOOL      (_,pos)       -> pos
13   | EXPFLOAT     (_,pos)       -> pos
14   | EXPOPB    ((_,pos),_,_) -> pos
15   | EXPOPU    ((_,pos),_)   -> pos
16   | EXPCALL     ((_,pos),_)   -> pos
17
18
19 type classe_op = Aritmetico | Relacional | Logico
20
21 let classifica op =
22   let open A in
23     match op with
24       ADICAO
25     | SUBTRACAO
26     | MULTIPLICACAO
27     | DIVISAO
28     | MOD       -> Aritmetico
29     | MAIORQ
30     | MENORQ
31     | MAIORIGUALQ
32     | MENORIGUALQ
33     | EHIGUAL
34     | EHDIFERENTE   -> Relacional
35     | AND
36     | NEGACAO
37     | OR  -> Logico
38
39 let msg_erro_pos pos msg =
40   let open Lexing in
41   let lin = pos.pos_lnum
42   and col = pos.pos_cnum - pos.pos_bol - 1 in
43   Printf.sprintf "Semantico -> linha %d, coluna %d: %s" lin col msg
44
45 (*  argumento nome é do tipo S.tipo  *)
```

```
46 let msg_erro nome msg =
47   let pos = snd nome in
48   msg_erro_pos pos msg
49
50 let nome_tipo t =
51   let open A in
52     match t with
53       INTEIRO     -> "inteiro"
54     | STRING      -> "string"
55     | BOOLEAN     -> "booleano"
56     | REAL    -> "real"
57     | NONE     -> "vazio"
58
59 let mesmo_tipo pos msg tinf tdec =
60   if tinf <> tdec then
61     let msg = Printf.sprintf msg (nome_tipo tinf) (nome_tipo tdec) in
62     failwith (msg_erro_pos pos msg)
63
64 let rec infere_exp amb exp =
65   match exp with
66
67   | S.EXPINT   i -> (T.EXPINT   (fst i, A.INTEIRO  ), A.INTEIRO  )
68   | S.EXPSTRING   s -> (T.EXPSTRING   (fst s, A.STRING  ), A.STRING  )
69   | S.EXPBOOL  b -> (T.EXPBOOL  (fst b, A.BOOLEAN ), A.BOOLEAN )
70   | S.EXPFLOAT f -> (T.EXPFLOAT (fst f, A.REAL), A.REAL)
71   | S.EXPVAR variavel ->
72       let nome = fst variavel in
73         (try begin
74             (match (Amb.busca amb nome) with
75             | Amb.EntVar tipo -> (T.EXPVAR (nome, tipo), tipo)
76             | Amb.EntFun _    ->
77                 let msg = "Nome de funcao usado como nome de variavel: "^
                        nome in
78                   failwith (msg_erro variavel msg))
79         end with Not_found ->
80           let msg = "Variavel "^nome^" nao declarada" in
81             failwith (msg_erro variavel msg))
82   | S.EXPOPB (op, exp_esq, exp_dir) ->
83       let (esq, tesq) = infere_exp amb exp_esq
84       and (dir, tdir) = infere_exp amb exp_dir in
85       let verifica_aritmetico () =
86         (match tesq with
87         | A.INTEIRO
88         | A.REAL ->
89             let _ = mesmo_tipo (snd op)
90                     "Operando esquerdo do tipo %s, mas o tipo do direito
                           eh %s"
91                     tesq tdir
92             in tesq (* Tipo inferido para a operação *)
93         | demais      ->
94             let msg = "O tipo "^
95                       (nome_tipo demais)^
96                       " nao eh valido em um operador aritmético" in
97               failwith (msg_erro op msg))
98       and verifica_relacional () =
99         (match tesq with
100        | A.INTEIRO
101        | A.STRING
102        | A.BOOLEAN
```

```
103         | A.REAL ->
104             (let _ = mesmo_tipo (snd op)
105                     "Operando esquerdo do tipo %s, mas o tipo do direito
                           eh %s"
106                     tesq tdir
107             in A.BOOLEAN) (* Tipo inferido para a operação *)
108         | demais      ->
109             (let msg = "O tipo "^
110                       (nome_tipo demais)^
111                       " nao eh valido em um operador relacional" in
112               failwith (msg_erro op msg)))
113       and verifica_logico () =
114         (match tesq with
115         | A.BOOLEAN ->
116             let _ = mesmo_tipo (snd op)
117                     "Operando esquerdo do tipo %s, mas o tipo do direito
                           eh %s"
118                     tesq tdir
119             in A.BOOLEAN (* Tipo inferido para a operação *)
120         | demais ->
121             let msg = "O tipo "^
122                       (nome_tipo demais)^
123                       " nao eh valido em um operador logico" in
124               failwith (msg_erro op msg))
125       in
126       let oper = fst op in
127       let tinf =
128          (match (classifica oper) with
129         | Aritmetico -> verifica_aritmetico ()
130         | Relacional -> verifica_relacional ()
131         | Logico     -> verifica_logico () )
132       in (T.EXPOPB ((oper, tinf), (esq, tesq), (dir, tdir)), tinf)
133   | S.EXPOPU (op, exp) ->
134     let (exp, texp) = infere_exp amb exp in
135     let verifica_not () =
136       match texp with
137       | A.BOOLEAN ->
138           let _ = mesmo_tipo (snd op)
139                   "O operando eh do tipo %s, mas espera-se um %s"
140                   texp A.BOOLEAN
141           in A.BOOLEAN
142       | demais     ->
143           let msg = "O tipo "^
144                       (nome_tipo demais)^
145                       " nINTEIROao eh valido para o operador not" in
146             failwith (msg_erro op msg)
147     and verifica_negativo () =
148       match texp with
149       | A.REAL ->
150           let _ = mesmo_tipo (snd op)
151                   "O operando eh do tipo %s, mas espera-se um %s"
152                   texp A.REAL
153           in A.REAL
154       | A.INTEIRO ->
155           let _ = mesmo_tipo (snd op)
156                   "O operando eh do tipo %s, mas espera-se um %s"
157                   texp A.INTEIRO
158           in A.INTEIRO
159       | demais     ->
```

```
160           let msg = "O tipo "^
161                       (nome_tipo demais)^
162                       " nao eh valido para o operador menos" in
163               failwith (msg_erro op msg)
164     in
165     let oper = fst op in
166     let tinf =
167       let open A in
168         match oper with
169         | NEGACAO   -> verifica_not ()
170         | SUBTRACAO -> verifica_negativo ()
171         | demais->
172             let msg = "Operador unario indefinido"
173             in failwith (msg_erro op msg)
174     in  (T.EXPOPU ((oper, tinf), (exp, texp)), tinf)
175   | S.EXPCALL (nome, args) ->
176     let rec verifica_parametros ags ps fs =
177       match (ags, ps, fs) with
178       | (a::ags), (p::ps), (f::fs) ->
179           let _ = mesmo_tipo (posicao a)
180                   "O parametro eh do tipo %s mas deveria ser do tipo %s"
181                   p f
182           in verifica_parametros ags ps fs
183       | [], [], [] -> ()
184       | _ -> failwith (msg_erro nome "Numero incorreto de parametros")
185     in
186     let id = fst nome in
187       try
188         begin
189           let open Amb in
190             match (Amb.busca amb id) with
191             | Amb.EntFun {tipo_fn; formais} ->
192               let targs    = List.map (infere_exp amb) args
193               and tformais = List.map snd formais in
194               let _ = verifica_parametros args (List.map snd targs)
                      tformais in
195                 (T.EXPCALL (id, (List.map fst targs), tipo_fn), tipo_fn)
196             | Amb.EntVar _ -> (* Se estiver associada a uma variável,
                   falhe *)
197               let msg = id ^ " eh uma variavel e nao uma funcao" in
198                 failwith (msg_erro nome msg)
199         end
200       with Not_found ->
201         let msg = "Nao existe a funcao de nome " ^ id in
202         failwith (msg_erro nome msg)
203
204 let rec verifica_cmd amb tiporet cmd =
205   let open A in
206     match cmd with
207     | CHAMADADEFUNCAO  exp -> let (exp,tinf) = infere_exp amb exp in
          CHAMADADEFUNCAO exp
208     | PRINT exp -> let expt = infere_exp amb exp in PRINT (fst expt)
209     | WHILELOOP (cond, cmds) ->
210         let (expCond, expT ) = infere_exp amb cond in
211         let comandos_tipados =
212           (match expT with
213             | A.BOOLEAN -> List.map (verifica_cmd amb tiporet) cmds
214             | _ -> let msg = "Condicao deve ser tipo Bool" in
215                       failwith (msg_erro_pos (posicao cond) msg))
```

```
216         in WHILELOOP (expCond,comandos_tipados)
217     | LEIAI exp ->
218         (match exp with
219           S.EXPVAR (id,pos) ->
220            (try
221              begin
222                (match (Amb.busca amb id) with
223                    Amb.EntVar tipo ->
224                      let expt = infere_exp amb exp in
225                      let _ = mesmo_tipo pos
226                        "inputi com tipos diferentes: %s = %s"
227                        tipo (snd expt) in
228                        LEIAI (fst expt)
229                  | Amb.EntFun _ ->
230                      let msg = "nome de funcao usado como nome de
                             variavel: " ^ id in
231                      failwith (msg_erro_pos pos msg) )
232              end
233             with Not_found ->
234               let _ = Amb.insere_local amb id A.INTEIRO in
235               let expt = infere_exp amb exp in
236               LEIAI (fst expt) )
237           | _ -> failwith "Falha Inputi"
238         )
239     | LEIAF exp ->
240         (match exp with
241           S.EXPVAR (id,pos) ->
242            (try
243              begin
244                (match (Amb.busca amb id) with
245                    Amb.EntVar tipo ->
246                      let expt = infere_exp amb exp in
247                      let _ = mesmo_tipo pos
248                        "Inputf com tipos diferentes: %s = %s"
249                        tipo (snd expt) in
250                        LEIAF (fst expt)
251                  | Amb.EntFun _ ->
252                      let msg = "nome de funcao usado como nome de
                             variavel: " ^ id in
253                      failwith (msg_erro_pos pos msg) )
254              end
255             with Not_found ->
256               let _ = Amb.insere_local amb id A.REAL in
257               let expt = infere_exp amb exp in
258               LEIAF (fst expt) )
259           | _ -> failwith "Falha Inputf"
260         )
261     | LEIAS exp ->
262         (match exp with
263           S.EXPVAR (id,pos) ->
264            (try
265              begin
266                (match (Amb.busca amb id) with
267                    Amb.EntVar tipo ->
268                      let expt = infere_exp amb exp in
269                      let _ = mesmo_tipo pos
270                        "Inputs com tipos diferentes: %s = %s"
271                        tipo (snd expt) in
272                        LEIAS (fst expt)
```

```
273                   | Amb.EntFun _ ->
274                       let msg = "nome de funcao usado como nome de
                              variavel: " ^ id in
275                       failwith (msg_erro_pos pos msg) )
276               end
277             with Not_found ->
278               let _ = Amb.insere_local amb id A.STRING in
279               let expt = infere_exp amb exp in
280               LEIAS (fst expt) )
281           | _ -> failwith "Falha Inputs"
282         )
283       | ATRIBUICAO (elem, exp) ->
284         let (var1, tdir) = infere_exp amb exp in
285         ( match elem with
286           S.EXPVAR (id,pos) ->
287            (try
288               begin
289                 (match (Amb.busca amb id) with
290                     Amb.EntVar tipo ->
291                       let _ = mesmo_tipo pos
292                         "Atribuicao com tipos diferentes: %s = %s"
293                         tipo tdir in
294                         ATRIBUICAO (T.EXPVAR (id, tipo), var1)
295                   | Amb.EntFun _ ->
296                       let msg = "nome de funcao usado como nome de
                              variavel: " ^ id in
297                       failwith (msg_erro_pos pos msg) )
298               end
299             with Not_found ->
300               let _ = Amb.insere_local amb id tdir in
301               ATRIBUICAO (T.EXPVAR (id, tdir), var1))
302           | _ -> failwith "Falha CmdAtrib"
303         )
304       | RETORNO exp ->
305         (match exp with
306        (* Se a função não retornar nada, verifica se ela foi declarada como
              void *)
307          None ->
308          let _ = mesmo_tipo (Lexing.dummy_pos)
309                      "O tipo retornado eh %s mas foi declarado como %s"
310
311                      NONE tiporet
312          in RETORNO None
313           | Some e ->
314
315          (* Verifica se o tipo inferido para a expressão de retorno confere
                 com o *)
316          (* tipo declarado para a função.
                                                          *)
317              let (e1,tinf) = infere_exp amb e in
318              let _ = mesmo_tipo (posicao e)
319                                 "O tipo retornado eh %s mas foi declarado
                                        como %s"
320                                 tinf tiporet
321              in RETORNO (Some e1)
322         )
323       | CONDICAOElifElse comandos ->
324           let comandos = List.map (verifica_cmd amb tiporet) comandos in
325             CONDICAOElifElse comandos
```

```
326     | CONDICAOIF (teste, entao, senao) ->
327         let (teste1,tinf) = infere_exp amb teste in
328         let _ = mesmo_tipo (posicao teste)
329                 "O teste do if deveria ser do tipo %s e nao %s"
330                 BOOLEAN tinf in
331         let entao1 = List.map (verifica_cmd amb tiporet) entao in
332         let senao1 =
333           match senao with
334           | None       -> None
335           | Some bloco -> let c = verifica_cmd amb tiporet bloco in Some c
336         in CONDICAOIF (teste1, entao1, senao1)
337     | FORLOOP (idt, int_de,int_ate,bloco) ->
338       let (idt1,tinf) = infere_exp amb idt in
339       let (int_de1,tinf1) = infere_exp amb int_de in
340       let (int_ate1,tinf2) = infere_exp amb int_ate in
341       (* O tipo inferido para o identificador deve ser int *)
342       let _ = mesmo_tipo (posicao idt)
343                 "A variável deveria ser do  tipo %s e nao %s"
344                 INTEIRO tinf in
345       (* O tipo inferido para os ints devem ser inteiros *)
346       let _ = mesmo_tipo (posicao int_de)
347                 "O comando DE deveria ser do  tipo %s e nao %s"
348                 INTEIRO tinf1 in
349       let _ = mesmo_tipo (posicao int_de)
350                 "O comando DE deveria ser do tipo %s e nao %s"
351                 INTEIRO tinf2 in
352       (* Verifica a validade de cada comando do bloco  *)
353       let bloco1 = List.map (verifica_cmd amb tiporet) bloco in
354         FORLOOP (idt1, int_de1,int_ate1,bloco1)
355
356 and verifica_fun amb ast =
357   let open A in
358   match ast with
359   | Funcao {fn_nome; fn_tiporet; fn_formais; fn_corpo} ->
360     (* Estende o ambiente global, adicionando um ambiente local *)
361     let ambfn = Amb.novo_escopo amb in
362     (* Insere os parâmetros no novo ambiente *)
363     let insere_parametro (v,t) = Amb.insere_param ambfn (fst v) t in
364       let _ = List.iter insere_parametro fn_formais in
365     (* Verifica cada comando presente no corpo da função usando o novo
           ambiente *)
366     let corpo_tipado = List.map (verifica_cmd ambfn fn_tiporet) fn_corpo
           in
367       Funcao {fn_nome; fn_tiporet; fn_formais; fn_corpo = corpo_tipado}
368   | ACMD _ -> failwith "Instrucao invalida"
369
370 let rec verifica_dup xs =
371   match xs with
372   | [] -> []
373   | (nome,t)::xs ->
374     let id = fst nome in
375     if (List.for_all (fun (n,t) -> (fst n) <> id) xs)
376     then (id, t) :: verifica_dup xs
377     else let msg = "Parametro duplicado " ^ id in
378       failwith (msg_erro nome msg)
379
380 let insere_declaracao_fun amb dec =
381   let open A in
382     match dec with
```

```
383     | Funcao {fn_nome; fn_tiporet; fn_formais; fn_corpo} ->
384       let formais = verifica_dup fn_formais in
385       let nome = fst fn_nome in
386       Amb.insere_fun amb nome formais fn_tiporet
387     | ACMD _ -> failwith "Instrucao invalida"
388
389 let fn_predefs =
390   let open A in [
391     ("inputi", [("x", INTEIRO  )], NONE);
392     ("inputs", [("x", STRING  )], NONE);
393     ("inputf", [("x", REAL)], NONE)]
394
395 let declara_predefinidas amb =
396   List.iter (fun (n,ps,tr) -> Amb.insere_fun amb n ps tr) fn_predefs
397
398 let semantico ast =
399   let amb_global = Amb.novo_amb [] in
400   let _ = declara_predefinidas amb_global in
401   let A.Programa instr = ast in
402   let decs_funs = List.filter(fun x ->
403     (match x with
404     | A.Funcao _ -> true
405     | _          -> false)) instr in
406     let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
407       let decs_funs = List.map (verifica_fun amb_global) decs_funs in
408       (A.Programa decs_funs, amb_global)
```

Listagem 4.11: sast.ml

```
1 open Ast
2
3 type expressao =
4        EXPOPB of operador pos * expressao * expressao
5       | EXPOPU of operador pos * expressao
6       | EXPVAR of identificador pos
7       | EXPINT of int pos
8       | EXPSTRING of string pos
9       | EXPFLOAT of float pos
10      | EXPBOOL of bool pos
11      | EXPCALL of identificador pos * (expressao expressoes)
```

Listagem 4.12: tast.ml

```
1 open Ast
2
3 type expressao =
4         EXPOPB of (operador * tipo) * (expressao * tipo)  * (expressao *
              tipo)
5       | EXPOPU of (operador * tipo) * (expressao * tipo)
6       | EXPCALL of identificador * (expressao expressoes) * tipo
7       | EXPINT of int * tipo
8       | EXPSTRING of string * tipo
9       | EXPFLOAT of float * tipo
10      | EXPBOOL of bool * tipo
11      | EXPVAR of identificador *  tipo
12      | EXPNONE
```

4.3.1 Testes

Para compilar fazemos:

Listagem 4.13: Terminal

```
$ ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -package
    menhirLib semanticoTeste.byte
Finished, 40 targets (37 cached) in 00:00:00.
$ rlwrap ocaml
        OCaml version 4.07.0

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

/Users/tatianefx/.opam/default/lib/menhirLib: added to search path
/Users/tatianefx/.opam/default/lib/menhirLib/menhirLib.cmo: loaded
# verifica_tipos "tests/micro10.py";;
```

Listagem 4.14: testeSemantico.txt

```
EOF
- : Tast.expressao Ast.programa * Ambiente.t =
(Programa
  [Funcao
    {fn_nome =
      ("main",
       {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 4});
     fn_tiporet = NONE; fn_formais = [];
     fn_corpo =
      [ATRIBUICAO (Tast.EXPVAR ("numero", INTEIRO), Tast.EXPINT (0,
         INTEIRO));
       ATRIBUICAO (Tast.EXPVAR ("fat", INTEIRO), Tast.EXPINT (0, INTEIRO))
          ;
       PRINT (Tast.EXPSTRING ("Digite um numero: ", STRING));
       LEIAI (Tast.EXPVAR ("numero", INTEIRO));
       ATRIBUICAO (Tast.EXPVAR ("fat", INTEIRO),
        Tast.EXPCALL ("fatorial", [Tast.EXPVAR ("numero", INTEIRO)],
           INTEIRO));
       PRINT (Tast.EXPSTRING ("O fatorial eh ", STRING));
       PRINT (Tast.EXPVAR ("fat", INTEIRO))]};
   Funcao
    {fn_nome =
      ("fatorial",
       {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 4});
     fn_tiporet = INTEIRO;
     fn_formais =
      [(("n",
         {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum =
            13}),
        INTEIRO)];
     fn_corpo =
      [CONDICAOIF
```

```
29          (Tast.EXPOPB ((MENORIGUALQ, BOOLEAN),
30            (Tast.EXPVAR ("n", INTEIRO), INTEIRO),
31            (Tast.EXPINT (0, INTEIRO), INTEIRO)),
32          [RETORNO (Some (Tast.EXPINT (1, INTEIRO)))],
33          Some
34           (CONDICAOElifElse
35             [RETORNO
36               (Some
37                 (Tast.EXPOPB ((MULTIPLICACAO, INTEIRO),
38                   (Tast.EXPVAR ("n", INTEIRO), INTEIRO),
39                   (Tast.EXPCALL ("fatorial",
40                     [Tast.EXPOPB ((SUBTRACAO, INTEIRO),
41                       (Tast.EXPVAR ("n", INTEIRO), INTEIRO),
42                       (Tast.EXPINT (1, INTEIRO), INTEIRO))],
43                     INTEIRO),
44                    INTEIRO))))]))]}],
45  <abstr>)
```

4.4 Interprete

Listagem 4.15: interprete.ml

```
1  module Amb = AmbInterp
2  module A = Ast
3  module S = Sast
4  module T = Tast
5
6  exception Valor_de_retorno of T.expressao
7
8  let obtem_nome_tipo_var exp = let open T in
9    match exp with
10     | EXPVAR (nome,tipo) -> (nome,tipo)
11     | _                  -> failwith "obtem_nome_tipo_var1: nao eh
         variavel"
12
13 let pega_int exp =
14   match exp with
15   |  T.EXPINT (i,_) -> i
16   | _ -> failwith "pega_int: nao eh inteiro"
17
18 let pega_float exp = match exp with
19   | T.EXPFLOAT (f,_)-> f
20   | _               -> failwith "pega_float: nao eh inteiro"
21
22 let pega_str exp =
23   match exp with
24   |  T.EXPSTRING (s,_) -> s
25   | _ -> failwith "pega_string: nao eh string"
26
27 let pega_bool exp =
28   match exp with
29   |  T.EXPBOOL (b,_) -> b
30   | _ -> failwith "pega_bool: nao eh booleano"
31
32 type classe_op = Aritmetico | Relacional | Logico
33
```

```
34 let classifica op =
35   let open A in
36   match op with
37     OR
38   | NEGACAO
39   | AND  -> Logico
40   | MENORQ
41   | MAIORQ
42   | MAIORIGUALQ
43   | MENORIGUALQ
44   | EHIGUAL
45   | EHDIFERENTE -> Relacional
46   | ADICAO
47   | SUBTRACAO
48   | MULTIPLICACAO
49   | MOD
50   | DIVISAO -> Aritmetico
51
52
53 let rec interpreta_exp amb exp =
54   let open A in
55   let open T in
56   match exp with
57   | EXPFLOAT  _
58   | EXPINT    _
59   | EXPSTRING _
60   | EXPBOOL   _   -> exp
61   | EXPVAR (nome, tipo) ->
62       (match (Amb.busca amb nome) with
63         | Amb.EntVar (_, v) ->
64           (match v with
65             | Some valor -> valor
66             | None       -> failwith "variável nao inicializada: "
67           )
68         |  _ -> failwith "interpreta_exp: expvar"
69       )
70   | EXPOPB ((op,top), (esq, tesq), (dir,tdir)) ->
71     let  vesq = interpreta_exp amb esq
72     and vdir = interpreta_exp amb dir in
73
74     let interpreta_aritmetico () =
75     (
76       match tesq with
77     | INTEIRO ->
78        (
79         match op with
80         | ADICAO  -> EXPINT (pega_int vesq + pega_int vdir, top)
81         | SUBTRACAO -> EXPINT (pega_int vesq - pega_int vdir, top)
82         | MULTIPLICACAO  -> EXPINT (pega_int vesq * pega_int vdir, top)
83         | DIVISAO   -> EXPINT (pega_int vesq / pega_int vdir, top)
84         | _     -> failwith "interpreta_aritmetico"
85        )
86      | _ -> failwith "interpreta_aritmetico"
87     )
88     and interpreta_relacional () =
89         (match tesq with
90           | INTEIRO ->
91             (match op with
92               | MAIORIGUALQ -> EXPBOOL (pega_int vesq >= pega_int vdir,
```

```
                top)
93            | MENORIGUALQ -> EXPBOOL (pega_int vesq <= pega_int vdir,
                top)
94            | MENORQ      -> EXPBOOL (pega_int vesq <  pega_int vdir,
                top)
95            | MAIORQ      -> EXPBOOL (pega_int vesq >  pega_int vdir,
                top)
96            | EHIGUAL      -> EXPBOOL (pega_int vesq == pega_int vdir,
                top)
97            | EHDIFERENTE      -> EXPBOOL (pega_int vesq != pega_int
                vdir, top)
98            | _          -> failwith "interpreta_relacional"
99          )
100       | STRING ->
101         (match op with
102           | MAIORIGUALQ -> EXPBOOL (pega_str vesq >= pega_str vdir,
                top)
103           | MENORIGUALQ -> EXPBOOL (pega_str vesq <= pega_str vdir,
                top)
104           | MENORQ      -> EXPBOOL (pega_str vesq <  pega_str vdir,
                top)
105           | MAIORQ      -> EXPBOOL (pega_str vesq >  pega_str vdir,
                top)
106           | EHIGUAL      -> EXPBOOL (pega_str vesq == pega_str vdir,
                top)
107           | EHDIFERENTE      -> EXPBOOL (pega_str vesq != pega_str
                vdir, top)
108           | _          -> failwith "interpreta_relacional"
109         )
110       | BOOLEAN ->
111         (match op with
112           | MAIORIGUALQ -> EXPBOOL (pega_bool vesq >= pega_bool vdir,
                top)
113           | MENORIGUALQ -> EXPBOOL (pega_bool vesq <= pega_bool vdir,
                top)
114           | MENORQ      -> EXPBOOL (pega_bool vesq <  pega_bool vdir,
                top)
115           | MAIORQ      -> EXPBOOL (pega_bool vesq >  pega_bool vdir,
                top)
116           | EHIGUAL      -> EXPBOOL (pega_bool vesq == pega_bool vdir,
                 top)
117           | EHDIFERENTE      -> EXPBOOL (pega_bool vesq != pega_bool
                vdir, top)
118           | _          -> failwith "interpreta_relacional"
119         )
120       | REAL ->
121         (match op with
122           | MAIORIGUALQ -> EXPBOOL (pega_float vesq == pega_float vdir
                , top)
123           | MENORIGUALQ -> EXPBOOL (pega_float vesq == pega_float vdir
                , top)
124           | MENORQ      -> EXPBOOL (pega_float vesq <  pega_float vdir
                , top)
125           | MAIORQ      -> EXPBOOL (pega_float vesq >  pega_float vdir
                , top)
126           | EHIGUAL      -> EXPBOOL (pega_float vesq == pega_float
                vdir, top)
127           | EHDIFERENTE      -> EXPBOOL (pega_float vesq != pega_float
                 vdir, top)
```

```
128                | _          -> failwith "interpreta_relacional"
129              )
130            | _ ->  failwith "interpreta_relacional"
131          )
132
133      and interpreta_logico () =
134        (match tesq with
135         | BOOLEAN ->
136           (match op with
137            | OR  ->  EXPBOOL (pega_bool vesq || pega_bool vdir, top)
138            | AND ->  EXPBOOL (pega_bool vesq && pega_bool vdir, top)
139            | _ ->  failwith "interpreta_logico"
140           )
141         | _ ->  failwith "interpreta_logico"
142        )
143
144      in
145      let valor = (match (classifica op) with
146            Aritmetico -> interpreta_aritmetico ()
147          | Relacional -> interpreta_relacional ()
148          | Logico     -> interpreta_logico ()
149        )
150      in
151        valor
152
153    | EXPOPU ((op, top), (exp, texp)) ->
154        let vexp = interpreta_exp amb exp in
155        let interpreta_not () =
156         (match texp with
157          | A.BOOLEAN  -> EXPBOOL (not (pega_bool vexp), top)
158          | _          -> failwith "Operador unario indefinido")
159        and interpreta_negativo () =
160         (match texp with
161          | A.INTEIRO   -> EXPINT   (-1   *  pega_int   vexp, top)
162          | A.REAL -> EXPFLOAT (-1.0 *. pega_float vexp, top)
163          | _           -> failwith "Operador unario indefinido")
164        in
165        let valor =
166         (match op with
167            | NEGACAO   -> interpreta_not ()
168            | SUBTRACAO -> interpreta_negativo ()
169            | _     -> failwith "Operador unario indefinido")
170        in  valor
171    | EXPCALL (id, args, tipo) ->
172      let open Amb in
173        (match (Amb.busca amb id) with
174          | Amb.EntFun {tipo_fn; formais; corpo} ->
175             let vargs    = List.map  (interpreta_exp amb) args in
176             let vformais = List.map2 (fun (n,t) v -> (n, t, Some v))
                   formais vargs
177             in  interpreta_fun amb vformais corpo
178          | _ -> failwith "interpreta_exp: expchamada"
179        )
180    | EXPNONE -> T.EXPNONE
181
182  and interpreta_cmd amb cmd =
183    let open A in
184    let open T in
185    match cmd with
```

```
186     RETORNO exp ->
187     (* Levantar uma exceção foi necessária pois, pela semântica do comando
            de   *)
188     (* retorno, sempre que ele for encontrado em uma função, a computação
              *)
189     (* deve parar retornando o valor indicado, sem realizar os demais
           comandos. *)
190     (match exp with
191       (* Se a função não retornar nada, então retorne ExpVoid *)
192       | None -> raise (Valor_de_retorno EXPNONE)
193       | Some e ->
194         (* Avalia a expressão e retorne o resultado *)
195         let e1 = interpreta_exp amb e in
196         raise (Valor_de_retorno e1))
197   | CONDICAOIF (teste, entao, senao) ->
198       let teste1 = interpreta_exp amb teste in
199       (match teste1 with
200         | EXPBOOL (true,_) ->
201         (* Interpreta cada comando do bloco 'então' *)
202         List.iter (interpreta_cmd amb) entao
203         | _ ->
204           (* Interpreta cada comando do bloco 'senão', se houver *)
205           (match senao with
206             | None -> ()
207             | Some bloco -> interpreta_cmd amb bloco))
208   | CONDICAOElifElse comandos ->
209         List.iter (interpreta_cmd amb ) comandos
210   | ATRIBUICAO (elem, exp) ->
211       let resp = interpreta_exp amb exp in
212         (match elem with
213           | T.EXPVAR (id,tipo) ->
214            (try
215               begin
216                 match (Amb.busca amb id) with
217                   | Amb.EntVar (t, _) -> Amb.atualiza_var amb id tipo (
                          Some resp)
218                   | Amb.EntFun _      -> failwith "falha na atribuicao"
219               end
220             with Not_found ->
221               let _ = Amb.insere_local amb id tipo None in
222               Amb.atualiza_var amb id tipo (Some resp))
223           | _ -> failwith "Falha CmdAtrib"
224         )
225   | CHAMADADEFUNCAO   exp -> ignore( interpreta_exp amb exp )
226   | LEIAI exp
227   | LEIAF exp
228   | LEIAS exp ->
229     (* Obtem os nomes e os tipos de cada um dos argumentos *)
230     let nt = obtem_nome_tipo_var exp in
231     let leia_var (nome,tipo) =
232      let _ =
233        (try
234           begin
235             match (Amb.busca amb nome) with
236               | Amb.EntVar (_,_) -> ()
237               | Amb.EntFun _     -> failwith "falha no input"
238           end
239         with Not_found ->
240           let _ = Amb.insere_local amb nome tipo None in ()
```

```
241         )
242       in
243       let valor =
244        (match tipo with
245           | INTEIRO  -> T.EXPINT  (read_int    ()    , tipo)
246           | STRING   -> T.EXPSTRING   (read_line  ()    , tipo)
247           | REAL     -> T.EXPFLOAT    (read_float ()    , tipo)
248           | _        -> failwith "Fail input")
249       in  Amb.atualiza_var amb nome tipo (Some valor)
250     in leia_var nt
251   | PRINT exp ->
252     let resp = interpreta_exp amb exp in
253       (match resp with
254         | T.EXPINT   (n,_) -> print_int    n
255         | T.EXPFLOAT (n,_) -> print_float  n
256         | T.EXPSTRING   (n,_) -> print_string n
257         | T.EXPBOOL (b,_) ->
258          let _ = print_string (if b then "true" else "false")
259          in print_string " "
260         | _ -> failwith "Fail print"
261       )
262   | WHILELOOP (cond, cmds) ->
263       let rec laco cond cmds =
264         let condResp = interpreta_exp amb cond in
265               (match condResp with
266                 | EXPBOOL (true,_) ->
267                     (* Interpreta cada comando do bloco 'então' *)
268                     let _ = List.iter (interpreta_cmd amb) cmds in
269                       laco cond cmds
270                 | _ -> ())
271       in laco cond cmds
272   | FORLOOP (idt, int_de ,int_ate, bloco) ->
273     let (elem1,tipo) = obtem_nome_tipo_var idt in
274     let rec executa_para amb int_de int_ate bloco elem1 tipo =
275           if (int_de) <= (int_ate)
276           then begin
277                   (*Executa o bloco de código: *)
278                   List.iter (interpreta_cmd amb) bloco;
279                   (*Atualiza o valor da variavel: *)
280                   Amb.atualiza_var amb elem1 tipo (Some ( EXPINT( (int_de
                          + 1 ),INTEIRO) ) );
281                   (*Chamada recursiva:*)
282                   executa_para amb (int_de + 1) int_ate bloco elem1 tipo;
283                end in
284     executa_para amb (pega_int int_de) (pega_int int_ate) bloco elem1 tipo
285
286 and interpreta_fun amb fn_formais fn_corpo =
287   let open A in
288  (* Estende o ambiente global, adicionando um ambiente local *)
289   let ambfn = Amb.novo_escopo amb in
290   (* Associa os argumento
291   s aos parâmetros e insere no novo ambiente *)
292   let insere_parametro (n,t,v) = Amb.insere_param ambfn n t v in
293   let _ = List.iter insere_parametro fn_formais in
294       (* Interpreta cada comando presente no corpo da função usando o novo
              *)
295       (* ambiente
                                                                        *)
296       try
```

```
297         let _ = List.iter (interpreta_cmd ambfn) fn_corpo in T.EXPNONE
298       with
299         Valor_de_retorno expret -> expret
300
301 let insere_declaracao_fun amb dec =
302   let open A in
303     match dec with
304       | Funcao {fn_nome; fn_tiporet; fn_formais; fn_corpo} ->
305         let nome = fst fn_nome in
306         let formais = List.map (fun (n,t) -> ((fst n), t)) fn_formais in
307         Amb.insere_fun amb nome formais fn_tiporet fn_corpo
308       | _ -> failwith "Erro de declaacao de funcao"
309
310
311 let fn_predefs = let open A in [
312     ("inputi", [("x", INTEIRO  )], NONE, []);
313     ("inputf", [("x", REAL     )], NONE, []);
314     ("inputs", [("x", STRING   )], NONE, []);
315
316 ]
317
318 (* insere as funções pré definidas no ambiente global *)
319 let declara_predefinidas amb =
320   List.iter (fun (n,ps,tr,c) -> Amb.insere_fun amb n ps tr c) fn_predefs
321
322 let interprete ast =
323   let open Amb in
324   let amb_global = Amb.novo_amb [] in
325   let _ = declara_predefinidas amb_global in
326   let A.Programa instr = ast in
327     let decs_funs = List.filter (fun x ->
328     (match x with
329     | A.Funcao _ -> true
330     |             _ -> false)) instr in
331     let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
332       (try begin
333         (match (Amb.busca amb_global "main") with
334             | Amb.EntFun { tipo_fn ; formais ; corpo } ->
335               let vformais = List.map (fun (n,t) -> (n, t, None)) formais
                    in
336               let _        = interpreta_fun amb_global vformais corpo in
                    ()
337             | _ -> failwith "variavel declarada como 'main'")
338       end with Not_found -> failwith "Funcao main nao declarada ")
```

Listagem 4.16: ambInterp.ml

```
1 module Tab = Tabsimb
2 module A = Ast
3 module T = Tast
4
5 type entrada_fn = {
6   tipo_fn:  A.tipo;
7   formais: (A.identificador * A.tipo) list;
8   corpo: T.expressao A.comandos
9 }
10
11 type entrada = EntFun of entrada_fn
12             |  EntVar of A.tipo * (T.expressao option)
```

```
13
14
15 type t = {
16   ambv : entrada Tab.tabela
17 }
18
19 let novo_amb xs = { ambv = Tab.cria xs }
20
21 let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }
22
23 let busca amb ch = Tab.busca amb.ambv ch
24
25 let atualiza_var amb ch t v =
26   Tab.atualiza amb.ambv ch (EntVar (t,v))
27
28 let insere_local amb nome t v =
29   Tab.insere amb.ambv nome (EntVar (t,v))
30
31 let insere_param amb nome t v =
32   Tab.insere amb.ambv nome (EntVar (t,v))
33
34 let insere_fun amb nome params resultado corpo =
35   let ef = EntFun { tipo_fn = resultado;
36                     formais = params;
37                     corpo = corpo }
38   in Tab.insere amb.ambv nome ef
```

4.4.1 Teste

Para compilar fazemos:

Listagem 4.17: Terminal

```
1 $ ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -package
     menhirLib interpreteTeste.byte
2 Finished, 40 targets (37 cached) in 00:00:00.
3 $ rlwrap ocaml
4         OCaml version 4.07.0
5
6 Findlib has been successfully loaded. Additional directives:
7   #require "package";;      to load a package
8   #list;;                   to list the available packages
9   #camlp4o;;                to load camlp4 (standard syntax)
10  #camlp4r;;                to load camlp4 (revised syntax)
11  #predicates "p,q,...";;   to set these predicates
12  Topfind.reset();;         to force that packages will be reloaded
13  #thread;;                 to enable threads
14
15 /Users/tatianefx/.opam/default/lib/menhirLib: added to search path
16 /Users/tatianefx/.opam/default/lib/menhirLib/menhirLib.cmo: loaded
17 # interprete "tests/micro10.py";;
```

Listagem 4.18: testeInterprete.txt

```
1 EOF
2 Digite um numero: 5
3 O fatorial eh 120- : unit = ()
```