Trabalho de Construção de Compiladores

Igor Benaventana Alves
Márcio Scofoni Manfré

Contents

Chapter 1

Introdução

Este trabalho consiste em constuir um compilador da linguagem de programação MiniJava para a plataforma CIL(Commom Intermediate Language) utilizando para isto OCaml ou Racket.

1.1 Commom Intermediate Language

É uma linguagem de programação de baixo nível(assembly) utilizada pela NET Framework e Mono.

1.1.1 Instalação

O comando para instalar o CIL no Ubuntu 14.04 é:

```
1 sudo apt-get install mono-runtime monodevelop
```

1.1.2 Compilação

A extensão compilável é ***.il** e para compilar um arquivo que esteja nesta extensão o comando é:

```
1 ilasm "nomedoarquivo".il
2 Exemplo:
3 ilasm alg1.il
```

A saída será um arquivo no formato ***.exe**

1.1.3 Execução

Para executar o ***.exe** gerado o comando é:

```
1 mono "nomedoarquivo".exe
2 Exemplo:
3 mono alg1.exe
```

Chapter 2

Programas

2.1 1 - Módulo Mínimo

MiniJava

```
1
2  public class Alg1 {
3    public static void main (String argv[])
4    {
5
6    }
7  }
```

CIL

```
1  .assembly extern mscorlib {}
2  .assembly alg1 {}
3  .method static void main() cil managed //metodo main
4  {
5      .entrypoint //para iniciar o programa
6      ret //finalizar programa
7  }
```

2.2 2 - Declaração de uma váriavel

MiniJava

```
1  public class Alg2 {
2    public static void main (String argv[])
3    {
4      int n;
5    }
6  }
```

CIL

```
1  .assembly extern mscorlib {}
2  .assembly alg2 {}
```

```
.method static void main() cil managed
{
    .entrypoint
    .locals init (int32) //quais vari veis locais iremos
       utilizar no programa
    .maxstack 1 //numero de itens..
    ret
}
```

2.3 3 - Atribuição de um inteiro a uma váriavel

MiniJava

```
public class Alg3 {
  public static void main (String argv[])
  {
    int n;
    n = 1;
  }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly alg3 {}
.method static void main() cil managed
{
    .entrypoint
    .maxstack 1
    .locals init (int32)
    ldc.i4 1 //coloca numero 1 na pilha como int32
    stloc.0 //tira da pilha e coloca na variavel local 0
    ret
}
```

2.4 4 - Atribuição de uma soma de inteiros a uma váriavel

MiniJava

```
public class Alg4 {
  public static void main (String argv[])
  {
    int n;
    n = 1+2;
  }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly alg4 {}
.method static void main() cil managed
```

```
{
    .locals init (int32,int32,int32)
    .maxstack 3
    .entrypoint


    ldc.i4 2 //coloca numero 2 na pilha como int32
    stloc.0 //tira da pilha e coloca na variavel local 0
    ldc.i4 1 //coloca numero 1 na pilha como int32
    stloc.1 //tira da pilha e coloca na variavel local 1
    ldloc.0 //carrega variavel local 0 para pilha
    ldloc.1 //carrega variavel local 1 para pilha
    add // add
    stloc.2
    ret

}
```

2.5 5 - Inclusão do comando de impressão

MiniJava

```
public class Alg5 {
  public static void main (String argv[])
  {
    int n;
    n = 2;
    System.out.println(n);
  }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly alg5 {}
.method static void main() cil managed
{
    .maxstack 1
    .entrypoint
    ldc.i4 2
    call void [mscorlib]System.Console::WriteLine(int32) //
        imprimir int32 da pilha
    ret
}
```

2.6 6 - Atribuição de uma subtração de interios a uma váriavel

MiniJava

```
public class Alg6 {
  public static void main (String argv[])
  {
```

```
    int n;
    n = 1 - 2;
    System.out.println(n);
  }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly alg6 {}
.method static void main() cil managed
{
    .locals init (int32,int32,int32)
    .maxstack 3
    .entrypoint


    ldc.i4 1
    stloc.0
    ldc.i4 2
    stloc.1
    ldloc.0
    ldloc.1
    sub
    stloc.2
    ret
}
```

2.7 7 - Inclusão do comando condicional

MiniJava

```
public class Alg7 {
  public static void main (String argv[])
  {
    int n;
    n = 1;
    if(n==1){
        System.out.println(n);
    }
  }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly alg7 {}
.method static void main() cil managed
{
    .maxstack 3
    .locals init (int32,int32,int32)
    .entrypoint
    ldc.i4 1
    stloc.0
```

```
    ldc.i4 1
    stloc.1
    ldloc.0
    ldloc.1
    beq IGUAL // se for igual vai para IGUAL

    IGUAL:
        ldloc.0 //carega variavel local 0 para pilha
        call void [mscorlib]System.Console::WriteLine(int32)
            //imprime
        ret

}
```

2.8 8 - Inclusão do comando condicional com parte senão

MiniJava

```
public class Alg8 {
  public static void main (String argv[])
  {
    int n;
    n = 1;
    if(n==1){
        System.out.println(n);
    }
    else{
        System.out.println(0);
    }
  }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly alg8 {}
.method static void main() cil managed
{
    .maxstack 3
    .locals init (int32,int32,int32)
    .entrypoint
    ldc.i4 2
    stloc.0
    ldc.i4 1
    stloc.1
    ldc.i4 0
    stloc.2
    ldloc.0
    ldloc.1
    beq IGUAL // se for igual vai para IGUAL
    ldloc.2
    call void [mscorlib]System.Console::WriteLine(int32) //
        imprime
```

```
19      ret
20      IGUAL:
21          ldloc.0 //carega variavel local 0 para pilha
22          call void [mscorlib]System.Console::WriteLine(int32)
                //imprime
23          ret
24
25  }
```

2.9 9 - Atribuição de duas operações aritméticas sobre inteiro a uma váriavel

MiniJava

```
1   public class Alg9 {
2     public static void main (String argv[])
3     {
4       int n;
5       n = 1+1/2;
6       if(n==1){
7           System.out.println(n);
8       }
9       else{
10          System.out.println(0);
11      }
12    }
13  }
```

CIL

```
1   .assembly extern mscorlib {}
2   .assembly alg9 {}
3   .method static void main() cil managed
4   {
5       .maxstack 4
6       .locals init (int32,int32,int32,int32)
7       .entrypoint
8       ldc.i4 2
9       stloc.0
10      ldc.i4 2
11      stloc.1
12      ldloc.0
13      ldloc.1
14      div
15      stloc.2
16      ldc.i4 1
17      stloc.3
18      ldloc.2
19      ldloc.3
20      add
21      stloc.0
22      ldc.i4 1
23      stloc.1
```

```
    ldloc.0
    ldloc.1
    beq IGUAL // se for igual vai para IGUAL
    ldstr"0"
    call void [mscorlib]System.Console::WriteLine(string) //
        imprime
    ret
    IGUAL:
    ldloc.0
    call void [mscorlib]System.Console::WriteLine(int32) //
        imprime
    ret

}
```

2.10 10 - Atribuição de duas váriaveis inteiras

MiniJava

```
public class Alg10 {
  public static void main (String argv[])
  {
    int n,m;
    n = 1;
    m = 2;
    if(n==m){
        System.out.println(n);
    }
    else{
        System.out.println(0);
    }
  }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly alg10 {}
.method static void main() cil managed
{
    .maxstack 3
    .locals init (int32,int32,int32)
    .entrypoint
    ldc.i4 1
    stloc.0
    ldc.i4 2
    stloc.1
    ldc.i4 0
    stloc.2
    ldloc.0
    ldloc.1
    beq IGUAL // se for igual vai para IGUAL
    ldloc.2
```

```
18      call void [mscorlib]System.Console::WriteLine(int32) //
            imprime
19      ret
20      IGUAL:
21          ldloc.0 //carega variavel local 0 para pilha
22          call void [mscorlib]System.Console::WriteLine(int32)
                //imprime
23          ret
24
25  }
```

2.11 11 - Atribuição de um comando de repetição enquanto

MiniJava

```
1   public class Alg11 {
2     public static void main (String argv[])
3     {
4       int n,m,x;
5       n = 1;
6       m = 2;
7       x = 5;
8       while(x > n){
9           n = n + m;
10          System.out.println(n);
11      }
12    }
13  }
```

CIL

```
1   .assembly extern mscorlib {}
2   .assembly alg11 {}
3   .method static void main() cil managed
4   {
5       .maxstack 4
6       .locals init (int32, int32,int32,int32)
7       .entrypoint
8       ldc.i4 1
9       stloc.0 // n
10      ldc.i4 2
11      stloc.1 // m
12      ldc.i4 5
13      stloc.2 // x
14
15      LOOP:
16          ldloc.2
17          ldloc.0
18          bgt MAIOR
19      Exit:
20          ret
21
```

```
22
23      MAIOR:
24          ldloc.0
25          ldloc.1
26          add
27          stloc.0
28          ldloc.0
29          call void [mscorlib]System.Console::WriteLine(int32)
                //imprime
30          br LOOP
31
32  }
```

2.12 12 - Comando condicional aninhado em um comando de repetição

MiniJava

```
1   public class Alg12 {
2     public static void main (String argv[])
3     {
4       int n,m,x;
5       n = 1;
6       m = 2;
7       x = 5;
8       while(x > n){
9           if(n == m){
10              System.out.println(n);
11          }
12          else{
13              System.out.println(0);
14          }
15          x = x - 1;
16
17      }
18    }
19  }
```

CIL

```
1   .assembly extern mscorlib {}
2   .assembly alg12 {}
3   .method static void main() cil managed
4   {
5       .maxstack 4
6       .locals init (int32, int32,int32,int32)
7       .entrypoint
8       ldc.i4 1
9       stloc.0 // n
10      ldc.i4 2
11      stloc.1 // m
12      ldc.i4 5
13      stloc.2 // x
```

```
14
15      LOOP:
16          ldloc.2
17          ldloc.0
18          bgt MAIOR
19
20
21      Exit:
22          ret
23
24
25      MAIOR:
26          ldloc.0
27          ldloc.1
28          beq IGUAL
29          ldstr"0"
30          call void [mscorlib]System.Console::WriteLine(string
                )
31          ldc.i4 1
32          stloc.3
33          ldloc.2
34          ldloc.3
35          sub
36          stloc.2
37          br LOOP
38
39      IGUAL:
40          ldloc.0
41          ldc.i4 1
42          stloc.3
43          ldloc.2
44          ldloc.3
45          sub
46          stloc.2
47          br LOOP
48
49  }
```

2.13 13 - Converte graus Celsius para Fahrenheit

MiniJava

```
1   import java.util.Scanner;
2   public class Alg13 {
3     public static void main (String argv[])
4     {
5       Scanner dados = new Scanner(System.in);
6       float = cel,far;
7       System.out.println(" Tabela de convers o: Celsius ->
            Fahrenheit");
8       System.out.println("Digite a temperatura em Celsius: ");
9       cel = dados.nextDouble();
```

```
10      far = (9*cel+160)/5;
11      System.out.println("A nova temperatura eh: "+far);
12    }
13  }
```

CIL

```
1  .assembly extern mscorlib {}
2  .assembly Tarefa1PrintInt {}
3
4  .method static void main() cil managed
5  {
6      .locals init (float32, float32, float32, float32)
7      .entrypoint
8      ldstr "Tabela de convers o Celsius -> Fahrenheit: "
9      call void [mscorlib] System.Console :: WriteLine(string)
10
11     ldstr "Digite a temperatura em Celsius: "
12     call void [mscorlib] System.Console :: WriteLine(string)
13     call string [mscorlib] System.Console :: ReadLine()
14     call float32 [mscorlib] System.Single :: Parse(string)
15     stloc.0 // atribui a temperatura Celsius a variavel
16     ldloc.0 // Coloca a temperatua Celsius na pilha
17     ldc.r4 9.0 // Coloca o 9 na pilha
18     mul // Multiplica 9*TemperaturaCelsius
19     ldc.r4 160.0 // Coloca 160.0 na pilha
20     add // 9*TemperaturaCelsius + 160
21     ldc.r4 5.0 // Coloca o 5.0 na pilha
22     div // (Realiza 9*TemperaturaCelsius +160 ) /5
23     ldstr "Temperatura em Fahrenheit: "
24     call void [mscorlib]System.Console::WriteLine(string)
25     call void [mscorlib]System.Console::Write(float32) //
           printa na tela o resultado em
26     ret //encerra programa
27
28 }
```

2.14 14 - Decide qual maior

MiniJava

```
1  public class Alg14 {
2    public static void main (String argv[])
3    {
4      Scanner dados = new Scanner(System.in);
5      int num1, num2;
6      System.out.println("Digite o primeiro numero: ");
7      num1 = dados.nextInt();
8      System.out.println("Digite o segundo numero: ");
9      num2 = dados.nextInt();
10
11     if(num1>num2){
12         System.out.println("O primeiro numero "+num1+" eh
               maior que o segundo "+num2);
```

```
    }
    else{
        System.out.println("O segundo numero "+num2+" eh
            maior que o primeiro "+num1);
    }
  }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly Tarefa1PrintInt {}

.method static void main() cil managed
{
.locals init (int32, int32)
.entrypoint

LOOP:
 ldstr "Digite um numero: "
 call void [mscorlib] System.Console :: WriteLine(string)
 call string [mscorlib] System.Console :: ReadLine()
 call int32 [mscorlib] System.Int32 :: Parse(string)
 stloc.0

 ldstr "Digite outro numero: "
 call void [mscorlib] System.Console :: WriteLine(string)
 call string [mscorlib] System.Console :: ReadLine()
 call int32 [mscorlib] System.Int32 :: Parse(string)
 stloc.1

 ldloc.0
 ldloc.1
 bgt MAIOR

 ldstr "O segundo n mero "
 call void [mscorlib] System.Console :: Write(string)
 ldloc.1
 call void [mscorlib]System.Console::Write(int32) //imprime
 ldstr " eh maior que o primeiro "
 call void [mscorlib] System.Console :: Write(string)
 ret

 MAIOR:
 ldstr "O primeiro n mero "
 call void [mscorlib] System.Console :: Write(string)
 ldloc.0
 call void [mscorlib]System.Console::Write(int32) //imprime
 ldstr " eh maior que o segundo "
 call void [mscorlib] System.Console :: Write(string)

 Exit:

ret
```

```

}
```

2.15 15 - Lê o número e verifica se está entre 100 e 200

MiniJava

```
public class Alg15 {
  public static void main (String argv[])
  {
    Scanner dados = new Scanner(System.in);
    int numero;
    System.out.println("Digite o numero: ");
    numero = dados.nextInt();
    if(numero >=100 && numero <=200){
        System.out.println("O numero esta no intervalo de
            100 a 200");
    }
    else{
        System.out.println("O numero nao esta no intervalo
            de 100 a 200");
    }

  }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly alg15 {}
.method static void main() cil managed
{
    .locals init (int32)
    .entrypoint
    LOOP:
        ldstr "Digite um numero: "
        call void [mscorlib] System.Console :: WriteLine(
            string)
        call string [mscorlib] System.Console :: ReadLine()
        call int32 [mscorlib] System.Int32 :: Parse(string)
        stloc.0 // atribui valor a primeira variavel
        ldloc.0 // Carrega o valor da primeira variavel
            local na pilha.
        ldc.i4 100 // Carrega o valor 100 na pilha.
        bge MAIOR //verdade se o primeiro a ser colocado na
            pilha    maior que o segundo
        ldstr"O valor nao esta entre 100 e 200"
        call void [mscorlib] System.Console::WriteLine(
            string)
        br Exit
        MAIOR:
        ldloc.0 // carrega o valor digitado na pilha
            novamente pois o bgt retira os valores da pilha
```

```
        ldc.i4 200 // carrega o valor 200 na pilha
        ble CORRETO
        ldstr"O valor nao esta entre 100 e 200"
        call void [mscorlib] System.Console::WriteLine(
            string)
        br Exit

    CORRETO:
        ldstr"O valor esta entre 100 e 200"
        call void [mscorlib] System.Console::WriteLine(
            string)
        br Exit
    Exit:
        ret
}
```

2.16 16 - Lê números e informa quais estão entre 10 e 150

MiniJava

```
public class Alg16 {
  public static void main (String argv[])
  {
    Scanner dados = new Scanner(System.in);
    int x,num,intervalo;
    for(x=0;x<5;x++){
        System.out.println("Digite um numero: ");
        num = dados.nextInt();
        if(num>=10 && num <= 150){
            intervalo = intervalo+1;
        }
    }
    System.out.println("Ao total foram digitados "+intervalo
        +" numeros no intervalo de 10 a 150");
 }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly Tarefa1PrintInt {}

.method static void main() cil managed
{

    .locals init (int32, int32, int32)
    .entrypoint
    ldc.i4 0 // Carrega na pilha o valor zero
    stloc.0 // Associa o valor na pilha a variavel 0
    ldc.i4 0
    stloc.2

```

```
14      LOOP:
15          ldloc.0 //Coloca a variavel x na pilha
16          ldc.i4 4 //Coloca o valor 80 na pilha
17          bgt CONTINUE // se x>80 vai para CONTINUE
18          ldstr "Digite um numero: "
19          call void [mscorlib] System.Console :: WriteLine(
                string)
20          call string [mscorlib] System.Console :: ReadLine()
21          call int32 [mscorlib] System.Int32 :: Parse(string)
22          stloc.1 // atribui valor a variavel 1
23          ldloc.1 // Carrega o valor da variavel 1 na pilha.
24          ldc.i4 10 // Carrega o valor 10 na pilha.
25          bge MAIOR //verdade se o primeiro a ser colocado na
                pilha    maior ou igual ao segundo
26          ldloc.0
27          ldc.i4 1
28          add
29          stloc.0 // Variavel 0 incrementada de 1
30          br LOOP
31
32      MAIOR:
33          ldloc.0 // carrega o valor digitado na pilha
                novamente pois o bgt retira os valores da pilha
34          ldc.i4 150 // carrega o valor 150 na pilha
35          ble CORRETO
36          ldloc.0
37          ldc.i4 1
38          add
39          stloc.0 // Variavel 0 incrementada de 1
40          br LOOP
41
42      CORRETO:
43          ldloc.2
44          ldc.i4 1
45          add
46          stloc.2
47          ldloc.0
48          ldc.i4 1
49          add
50          stloc.0 // Variavel 0 incrementada de 1
51          br LOOP
52      CONTINUE:
53          ldstr "Ao total foram digitados "
54          call void [mscorlib] System.Console :: WriteLine(
                string)
55          ldloc.2
56          call void [mscorlib]System.Console::Write(int32)
57          ldstr "numeros entre 10 e 150."
58          call void [mscorlib] System.Console :: WriteLine(
                string)
59          ret
60  }
```

2.17 17 - Lê strings e caracteres

MiniJava

```
public class Alg17 {
  public static void main (String argv[])
  {
    Scanner dados = new Scanner(System.in);
    string nome,sexo;
    int x,h,m

    for(x=0;x<5;x++){
        System.out.println("Digite o nome: ");
        nome = dados.nextLine();
        System.out.println("H - Homem e M - Mulher");
        sexo = dados.nextLine();
        if(sexo.equals("H")){
            h = h+1;
        }else if(sexo.equals("M")){
            m = m+1;
        }else{
            System.out.println("Sexo so pode ser H ou M");
        }
    }
    System.out.println("Foram inseridos "+h+" Homens" );
    System.out.prinln("Foram inseridos "+m+" Mulheres");
 }
}
```

CIL

2.18 18 - Escreve um número lido por extenso

MiniJava

```
public class Alg18 {
  public static void main (String argv[])
  {
    Scanner dados = new Scanner(System.in);

    int numero;

    System.out.println("Digite um numero de 1 a 5");
    numero = dados.nextInt();
    if(numero==1){
        System.out.println("UM");
    }else if(numero==2){
        System.out.println("DOIS");
    }else if(numero==3){
        System.out.println("TRES");
    }else if(numero==4){
        System.out.println("QUATRO");
    }else if(numero==5){
        System.out.println("CINCO");
```

```
    }else{
        System.out.println("Numero invalido");
    }
 }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly Tarefa1PrintInt {}

.method static void main() cil managed
{
    .locals init (int32)
    .entrypoint
    ldstr "Digite um numero: "
    call void [mscorlib] System.Console :: WriteLine(string)
    call string [mscorlib] System.Console :: ReadLine()
    call int32 [mscorlib] System.Int32 :: Parse(string)
    stloc.0 // atribui valor a variavel 1
    ldloc.0 // Carrega o valor da variavel 1 na pilha.
    ldc.i4 1 // Carrega o valor 1 na pilha.
    beq UM
    ldloc.0 // Carrega o valor da variavel 1 na pilha.
    ldc.i4 2 // Carrega o valor 2 na pilha.
    beq DOIS
    ldloc.0 // Carrega o valor da variavel 1 na pilha.
    ldc.i4 3 // Carrega o valor 10 na pilha.
    beq TRES
    ldloc.0 // Carrega o valor da variavel 1 na pilha.
    ldc.i4 4 // Carrega o valor 10 na pilha.
    beq QUATRO
    ldloc.0 // Carrega o valor da variavel 1 na pilha.
    ldc.i4 5 // Carrega o valor 10 na pilha.
    beq CINCO
    ldstr "Numero nao esta entre 1 e 5."
    call void [mscorlib] System.Console :: WriteLine(string)
    br EXIT

    UM:
        ldstr "UM"
        call void [mscorlib] System.Console :: WriteLine(
            string)
        br EXIT

    DOIS:
        ldstr "DOIS"
        call void [mscorlib] System.Console :: WriteLine(
            string)
        br EXIT

    TRES:
        ldstr "TRES"
```

```
44          call void [mscorlib] System.Console :: WriteLine(
                string)
45          br EXIT
46
47      QUATRO:
48          ldstr "QUATRO"
49          call void [mscorlib] System.Console :: WriteLine(
                string)
50          br EXIT
51
52      CINCO:
53          ldstr "CINCO"
54          call void [mscorlib] System.Console :: WriteLine(
                string)
55          br EXIT
56
57      EXIT:
58          ret
59
60  }
```

2.19 19 - Decide se os números são positivos, negativos ou zero

MiniJava

```
1  public class Alg19 {
2    public static void main (String argv[])
3    {
4      Scanner dados = new Scanner(System.in);
5
6      int numero, programa;
7      String opc;
8
9      programa =1;
10     while(programa == 1){
11         System.out.println("Escreva um numero: ");
12         numero = dados.nextInt();
13
14         if(numero > 0){
15             System.out.println("Numero positivo");
16         }else if(numero < 0){
17             System.out.println("Numero negativo");
18         }else{
19             System.out.println("Numero igual a 0");
20         }
21         System.out.println("Deseja finalizar? (S/N)");
22         opc = dados.nextLine();
23         if(opc.equals("S")){
24             programa = 0;
25         }
26     }
27  }
```

```
}
```

CIL

```
.assembly extern mscorlib {}
.assembly Tarefa2Exerc {}

.method static void main() cil managed

{
    .locals init (int32, int32,int32)
    .entrypoint

    LOOP:
        ldstr "Digite um numero: "
        call void [mscorlib] System.Console :: WriteLine(
            string)
        call string [mscorlib] System.Console :: ReadLine()
        call int32 [mscorlib] System.Int32 :: Parse(string)
        stloc.0 // atribui valor a primeira variavel
        ldloc.0 //
        ldc.i4 0 // Carrega o valor 0 na pilha. Item na
            pilha=2
        bgt MAIOR
        ldloc.0
        ldloc.1
        beq IGUAL
        ldstr "O valor eh negativo"
        call void [mscorlib] System.Console::WriteLine(
            string)
        ldstr "Deseja finalizar? 1 para sim e 0 para nao: "
        call void [mscorlib] System.Console::WriteLine(
            string)
        call string [mscorlib] System.Console :: ReadLine()
        call int32 [mscorlib] System.Int32 :: Parse(string)
        stloc.1 //guarda na variavel
        ldloc.1 //coloca o valor na pilha
        ldc.i4 0 // coloca o 0 na pilha
        beq Exit // se for igual encerra o programa
        br LOOP

    MAIOR:
        ldstr"O valor eh positivo"
        call void [mscorlib] System.Console::WriteLine(
            string)
        ldstr "Deseja finalizar? 1 para sim e 0 para nao: "
        call void [mscorlib] System.Console::WriteLine(
            string)
        call string [mscorlib] System.Console :: ReadLine()
        call int32 [mscorlib] System.Int32 :: Parse(string)
        stloc.1 //guarda na variavel
        ldloc.1 //coloca o valor na pilha
        ldc.i4 0 // coloca o 0 na pilha
        beq Exit // se for igual encerra o programa
```

```
45          br LOOP
46
47      IGUAL:
48          ldstr"O valor eh igual a zero"
49          call void [mscorlib] System.Console::WriteLine(
                string)
50          ldstr "Deseja finalizar? 1 para nao e 0 para sim: "
51          call void [mscorlib] System.Console::WriteLine(
                string)
52          call string [mscorlib] System.Console :: ReadLine()
53          call int32 [mscorlib] System.Int32 :: Parse(string)
54          stloc.1 //guarda na variavel
55          ldloc.1 //coloca o valor na pilha
56          ldc.i4 0 // coloca o 0 na pilha
57          beq Exit // se for igual encerra o programa
58          br LOOP
59
60      Exit:
61          ret
62
63  }
```

2.20 20 - Decide se um número é maior que 10

MiniJava

```
1  public class Alg20 {
2    public static void main (String argv[])
3    {
4      Scanner dados = new Scanner(System.in);
5
6      int numero;
7
8      while(numero != 0){
9          System.out.println("Escreva um numero: ");
10         numero = dados.nextInt();
11         if(numero > 10){
12             System.out.println("O numero "+numero+" e maior
                   que 10");
13         }else if(numero < 10){
14             System.out.println("O numero "+numero+" e menor
                   que 10");
15         }
16     }
17   }
18 }
```

CIL

```
1  .assembly extern mscorlib {}
2  .assembly Tarefa2Exerc {}
3
4  .method static void main() cil managed
5  {
```

```
6       .locals init (int32,int32)
7       .entrypoint
8       LOOP:
9           ldstr "Digite um numero: "
10          call void [mscorlib] System.Console :: WriteLine(
                string)
11          call string [mscorlib] System.Console :: ReadLine()
12          call int32 [mscorlib] System.Int32 :: Parse(string)
13          stloc.0 // atribui valor a primeira variavel
14          ldloc.0 //
15          ldc.i4 10// Carrega o valor 10 na pilha. Item na
                pilha=2
16          bgt MAIOR //verdade se o primeiro a ser colocado na
                pilha    maior que osegundo
17          ldstr "O numero eh menor que 10"
18          call void [mscorlib] System.Console::WriteLine(
                string)
19          ldloc.0
20          ldc.i4 0
21          beq Exit // verdade se ambos valores forem iguais,
                finaliza o programa
22          br LOOP // se nao forem iguais continua o programa
23
24      MAIOR:
25          ldstr"O numero eh maior que 10"
26          call void [mscorlib] System.Console::WriteLine(
                string)
27          br LOOP
28
29      Exit:
30          ret
31
32  }
```

2.21 22 - Calcule o fatorial de um número

MiniJava

```
1   public class Alg22 {
2     public static void main (String argv[])
3     {
4       Scanner dados = new Scanner(System.in);
5
6       int numero,fat;
7       System.out.println("Digite um numero: ");
8       numero = dados.nextInt();
9       fat = fatorial(numero);
10      System.out.println("O fatorial de "+numero+" eh "+fat);
11    }
12    public static int fatorial(int num){
13      if(num==0){
14          return 1;
15      }else{
16          return (num*fatorial(num-1));
```

```
    }
 }
}
```

CIL

```
.assembly extern mscorlib {}
.assembly alg22{}
.method static void main() cil managed
{
    .locals init (int32, int32)
    .entrypoint
    ldstr "Digite um numero : "
    call void [mscorlib]System.Console::WriteLine(string)
    call string [mscorlib]System.Console::ReadLine()
    call int32 [mscorlib]System.Int32::Parse(string)
    stloc.0 //atribui o valor ao n
    ldc.i4 1 //coloca 1 na pilha
    stloc.1 // grava na variavel fat o valor 1
    LOOP:
        ldloc.0 // carrega o valor de n
        brfalse CONTINUE // verifica se n = 0
        ldloc.0 // coloca n na pilha
        ldloc.1 // coloca fat na pilha
        mul.ovf // multiplica n pelo fat e poe o resultado
            no topo
        stloc.1 // grava na variavel fat
        ldloc.0
        ldc.i4 1
        sub.ovf
        stloc.0
        br LOOP
    CONTINUE:
        ldstr"O fatorial eh "
        call void [mscorlib]System.Console::WriteLine(string
            )
        ldloc.1
        call void [mscorlib]System.Console::WriteLine(int32)
        ret
}
```

Chapter 3

Compilador

Neste capítulo é mostrado todos os códigos utilizados na construção de um compilador.

O compilador em questão foi divido em partes: Léxico, Sintático, Semântico, Representação Intermediária e Gerador.

Na Representação Intermediária colocamos somente um esboço para ser aproveitado futuramente.

E também existe o código do Interpretador.

3.1 Gerador

Antes das etapas dos analisadores, precisamos automatizar alguns comandos para facilitar o desenvolvimento e execucação do compilador.

Um deles é o carregador que organiza todas as funções de execução dos analisadores em um único arquivo.

```
1  (* arquivo carregador.ml *)
2
3  #load "sintatico.cmo";;
4  #load "lexico.cmo";;
5  #load "arvSint.cmo";;
6  #load "semantico.cmo";;
7  #load "gerador.cmo";;
8
9
10 open Sintatico;;
11 open ArvSint;;
12 open Semantico;;
13 open Gerador;;
14 open Printf;;
15
16 let sintatico lexbuf =
17   try
18     Sintatico.programa Lexico.token lexbuf
19   with exn ->
20     begin
21       let tok  = Lexing.lexeme lexbuf in
22       let pos  = lexbuf.Lexing.lex_curr_p in
```

```
23        let nlin = pos.Lexing.pos_lnum in
24        let ncol = pos.Lexing.pos_cnum - pos.Lexing.pos_bol -
              String.length tok
25        in
26        let msg1 = sprintf "Erro na linha %d, coluna %d" nlin
              ncol in
27        let msg2 = sprintf "\tA palavra \"%s\" nao era
              esperada aqui." tok in
28          print_endline msg1;
29          print_endline msg2;
30          flush stdout;
31          raise exn
32      end
33
34  let sint_str str =
35    let lexbuf = Lexing.from_string str in
36    sintatico lexbuf
37
38  let sint_arq arq =
39     let ic = open_in arq in
40     let lexbuf = Lexing.from_channel ic in
41     let arv = sintatico lexbuf in
42     close_in ic;
43     arv
44
45  let sem_arq arq =
46    let arv = sint_arq arq in
47    semantico arv
48
49
50  let gera arq =
51    let arv = sint_arq arq in
52    let amb = semantico arv in
53    gera amb arv
54
55  (*
56  let gera arq saida =
57    let ic = open_in arq in
58    let arv = sint_arq arq in
59    let amb = semantico arv in
60    let arquivo = gera amb arv saida in
61    close_in ic;
62    (arquivo, arv)
63  *)
```

3.2 Makefile

Para ajudar na compilação dos arquivos, fizemos um Makefile para facilitar a execução desses comandos.

Segue seu código:

```
CAMLC=ocamlc
CAMLLEX=ocamllex
CAMLYACC=ocamlyacc

compInter: compSem gerador.cmo

compSem: compSint semantico.cmo

compSint: lexico.cmo sintatico.cmo

gerador.cmo: arvSint.cmi gerador.ml
    $(CAMLC) -c gerador.ml

semantico.cmo: arvSint.cmi semantico.ml
    $(CAMLC) -c semantico.ml

arvSint.cmi: arvSint.ml
    $(CAMLC) -c arvSint.ml

sintatico.cmo: sintatico.cmi sintatico.ml
    $(CAMLC) -c sintatico.ml

sintatico.cmi: sintatico.mli
    $(CAMLC) -c sintatico.mli

sintatico.ml: arvSint.cmi sintatico.mly
    $(CAMLYACC) -v sintatico.mly

sintatico.mli: arvSint.cmi sintatico.mly
    $(CAMLYACC) -v sintatico.mly

lexico.cmo: sintatico.cmi lexico.ml
    $(CAMLC) -c lexico.ml

lexico.cmi: sintatico.cmi lexico.ml
    $(CAMLC) -c lexico.ml

lexico.ml: lexico.mll
    $(CAMLLEX) lexico.mll

clean:
    rm -f *.cmo *.cmi lexico.ml sintatico.ml sintatico.mli
```

3.3 Analisador Léxico

A primeira etapa de um compilador é verificar se os caracteres do programa estão no alfabeto da linguagem.

Então o analisador léxico recebe essas linhas de caracteres e transformam em tokens que são manipulados mais facilmente por um parser (leitor de saída).

```
(* Para compilar digite:
ocamlyacc -v sintatico.mly
ocamlc -c sintatico.mli
ocamllex lexico.mll
ocamlc -c lexico.ml
ocamlc -c sintatico.ml
 *)
{
  open Sintatico   (* o tipo token    definido em sintatico.
      mli *)
  open Lexing
  open Printf

  let incr_num_linha lexbuf =
    let pos = lexbuf.lex_curr_p in
    lexbuf.lex_curr_p <- { pos with
       pos_lnum = pos.pos_lnum + 1;
       pos_bol = pos.pos_cnum;
    }

  let msg_erro lexbuf c =
    let pos = lexbuf.lex_curr_p in
    let lin = pos.pos_lnum
    and col = pos.pos_cnum - pos.pos_bol - 1 in
    sprintf "%d-%d: caracter desconhecido %c" lin col c
}

let digito = ['0' - '9']
let inteiro = '-'? digito+
let float = ( '-'? digito+ '.' digito+ | '.' digito+ )
let identificador  = ['a'-'z' 'A'-'Z'] ['_' 'a'-'z' 'A'-'Z'
    '0'-'9']*

let importt = "import java.util.Scanner;"

rule token = parse
  [' ' '\t' ]         { token lexbuf }  (* ignora espa os
      *)
| "\r\n"             { incr_num_linha lexbuf; token lexbuf }
      (* ignora fim de linha *)
| ['\n' ]             { incr_num_linha lexbuf; token lexbuf
    }  (* ignora fim de linha *)

| inteiro as num { let numeroint = int_of_string num in
    printf "Int %d\n" numeroint; LitInt ( int_of_string num)
    }
| float as num { let numerofloat = float_of_string num in
```

```
    printf "Float %f\n" numerofloat; LitFloat (
    float_of_string num) }
41
42  | "=="              { printf "IgualI\n"; IgualI }
43  | "!="                 { printf "Diferente\n"    ; Dif    }
44  | "<="              { printf "MenorQ\n"; MenorI }
45  | ">="              { printf "MaiorQ\n"; MaiorI }
46  | "="                  { printf "Atrib\n" ; Atrib  }
47  | '\'' ([^ '\r'] | '\r' as c) '\'' { LitChar (c) }
48  | '<'               { printf "Menor\n"; Menor }
49  | '>'                  { printf "Maior\n"; Maior }
50
51  | "||"              { printf "Or\n"; Or }
52  | "&&"              { printf "And\n"; And }
53  | '&'               { printf "ECom\n"; ECom }
54
55  | "++"              { printf "Incre\n" ; Inc  }
56
57  | '+'                  { printf "OpSoma\n"; OpSoma }
58  | '-'                  { printf "OpSub\n" ; OpSub  }
59  | '*'                  { printf "OpMul\n"; OpMul }
60  | '/'               { printf "OpDiv\n"; OpDiv }
61
62  | '('                  { printf "AParen\n"; AParen }
63  | ')'                  { printf "FParen\n"; FParen }
64  | '{'                  { printf "AChave\n"; AChave }
65  | '}'                  { printf "FChave\n"; FChave }
66  | '['                  { printf "AColc\n"; AColc  }
67  | ']'                  { printf "FColc\n"; FColc  }
68  | ';'                  { printf "PTVirg\n"; PTVirg }
69  | ','                  { printf "Virg\n"  ; Virg   }
70  | '.'                  { printf "Ponto\n" ; Ponto  }
71  | ':'                  { printf "DoisPont\n" ; DoisPont  }
72  | "'"                  { printf "Apost\n" ; Apost }
73  | '"'            { printf "String: " ; let buffer = Buffer.
    create 1 in STRING (cadeia buffer lexbuf) }
74
75  | '?'               { printf "PInter\n" ; PInter }
76  | '!'                  { printf "PExcla\n" ; Not }
77
78
79  | "public"          { printf "Public\n"; Public }
80  | "static"          { printf "Static\n"; Static }
81  | "void"            { printf "Void\n" ; Void }
82  | "String argv[]"      { printf "StrArgv\n" ; StrArgv }
83  | "int"                { printf "Int\n";   Int  }
84  | "float"           { printf "Float\n"; Float }
85  | "char"            { printf "Char\n"; Char}
86  | "class"           { printf "Class\n";  Class  }
87  | "String"          { printf "String\n"; String }
88  | "System.out.print"  { printf "Print\n"; Print }
89  | "System.out.println"  { printf "Println\n"; Println }
90  | "Scanner"         { printf "Scanner\n"; Scanner}
91  | "new"             { printf "New\n"; New}
```

```
92  | "System.in"          { printf "SystemIn\n"; SystemIn}
93
94  | "nextFloat()"       { printf "NextDouble\n"; NextFloat}
95  | "nextInt()"       { printf "NextInt\n"; NextInt}
96  | "nextLine()"       { printf "NextLine\n"; NextLine}
97
98  | "switch"           { printf "Switch\n"; Switch}
99  | "default"          { printf "Default\n"; Default}
100 | "case"             { printf "Case\n"; Case}
101 | "break"          { printf "Break\n"   ; Break  }
102
103
104 | "return"         { printf "Return\n"; Return }
105 | "if"                { printf "If\n";    If        }
106 | "else"              { printf "Else\n";    Else    }
107 | "true"              { printf "True\n";    True    }
108 | "false"             { printf "False\n";    False  }
109 | "main"              { printf "Main\n";    Main    }
110 | "while"          { printf "While\n";    While    }
111 | "for"            { printf "For\n";    For    }
112 | importt as imp      { printf "Import %s\n" imp; Import (
      imp)}
113
114 | identificador as id { printf "Ident %s\n" id; Ident (id) }
115
116 | _ as c              { failwith (msg_erro lexbuf c); }
117 | eof                 { printf "EOF\n"   ; EOF    }
118
119 and multilinha = parse
120   | "*/"           { token lexbuf }
121   | _             { multilinha lexbuf }
122   | eof            { failwith "comentario n fechado" }
123
124 and cadeia buffer = parse
125 | '"'            { Buffer.contents buffer }
126 | "\\t"          { Buffer.add_char buffer '\t'; cadeia buffer
      lexbuf }
127 | "\\n"          { Buffer.add_char buffer '\n'; printf "
     PulaLinha\n" ; cadeia buffer lexbuf }
128 | '\\' '"'       { Buffer.add_char buffer '"'; cadeia buffer
     lexbuf }
129 | '\\' '\\'      { Buffer.add_char buffer '\\'; cadeia buffer
      lexbuf }
130 | _ as c         { Buffer.add_char buffer c; printf "%c" c;
       cadeia buffer lexbuf }
131 | eof            { failwith "String nao foi fechada" }
```

3.4 Analisador Sintático

Nessa etapa, o analisador sintático recebe os tokens vindos do analisador léxico e processa de acordo com um conjunto de regras.

O analisador sintático durante esse processamento, transforma os tokens em uma árvore sintática, que representa a estrutura sintatica de uma cadeia de acordo com alguma gramática formal.

```
%{
    open ArvSint;;
%}

%token <int> LitInt
%token <float> LitFloat
%token <char> LitChar
%token <string> STRING
%token <bool> LitBool
%token <string> Ident Frase
%token IgualI MenorI MaiorI Menor Maior Or And ECom Inc Dif
    Not
%token OpSoma OpSub OpMul OpDiv
%token AParen FParen AChave FChave AColc FColc
%token Virg PTVirg Ponto Aspas Apost DoisPont PInter PExcla
%token <string> Import
%token Public Class New Scanner Static Void Int Float Char
    Bool SystemIn Return
%token String Print StrArgv Println
%token Atrib NextFloat NextInt NextLine
%token If Else True False Main While For
%token Switch Case Break Default DoisPont
%token EOF

%start programa /* simbolo inicial da gramatica */
%type <ArvSint.programa> programa

%%
programa: imports decl_class EOF { Programa ($1, $2) };

imports: /* nada */             { [] }
        | importt imports  { $1 :: $2 };

importt: Import { Imp $1 };

decl_class: Public Class Ident AChave decl_fun_main FChave {
    DeclClass($5)};

decl_fun_main: Public Static Void Main AParen StrArgv FParen
     AChave comandos FChave{DeclFun($9)};

comandos: /* nada */            { [] }
          | comando comandos            { $1 :: $2 };

comando: cmd_print  { $1 }
       | cmd_incr       { $1 }
```

```
43        | cmd_dec    { $1 }
44        | cmd_if     { $1 }
45        | cmd_while  { $1 }
46        | cmd_atrib  { $1 }
47        | cmd_switch     { $1 }
48        | cmd_for    { $1 }
49        | cmd_return { $1 };
50
51
52
53  cmd_print: Print AParen argumentos FParen PTVirg {CmdPrint (
        $3 )};
54
55  cmd_incr: Ident Inc { CmdInc(ExpVar $1) }
56
57  cmd_dec:  tipo Ident inicial PTVirg {CmdDecl (ExpVar $2, $1,
        $3) };
58      | tipo Ident inicia_construtor PTVirg       {
           CmdDeclConstrut (ExpVar $2, $1, $3)};
59
60  inicial: /* nada */ {None}
61      | Atrib expressao {Some($2)};
62
63
64
65  inicia_construtor: Atrib New tipo AParen SystemIn FParen {
       $3 };
66
67
68  tipo: Int   { Int }
69      | Bool  { Bool }
70      | Char  { Char }
71      | String    { String }
72      | Scanner   { Scanner }
73      | Float     { Float }   ;
74
75  cmd_if: If AParen expressao FParen AChave comandos FChave
       parte_else { CmdIf ($3, $6, $8) }
76
77  parte_else: /*vazio*/ { None }
78      | Else AChave comandos FChave { Some($3) };
79      | Else cmd_if   { Some([$2]) };
80
81
82
83  cmd_while: While AParen expressao FParen AChave comandos
       FChave { CmdWhile ($3, $6) };
84
85  cmd_for: For AParen cmd_atrib expressao PTVirg comando
       FParen AChave comandos FChave { CmdFor ($3,$4,$6,$9) };
86
87  cmd_return: Return expressao PTVirg { CmdReturn( $2 ) };
88
89  cmd_atrib: Ident Atrib expressao PTVirg { CmdAtrib (ExpVar
```

```
    $1, $3) }
90      |  Ident Atrib Ident Ponto NextFloat PTVirg {
            CmdAtribNextFloat (ExpVar $1, ExpVar $3)}
91    |  Ident Atrib Ident Ponto NextInt PTVirg{ CmdAtribNextInt
          (ExpVar $1, ExpVar $3)}
92      |  Ident Atrib Ident Ponto NextLine PTVirg{
            CmdAtribNextLine (ExpVar $1, ExpVar $3)};
93
94  cmd_switch: Switch AParen expressao FParen AChave cases
        default FChave { CmdSwitch($3,$6,Some($7)) };
95
96
97  cases: /* nada */       { [] }
98       | case cases   { $1 :: $2 };
99
100 case: Case expressao DoisPont comandos Break PTVirg { Case (
        $2,$4) }
101      | Case Apost expressao Apost DoisPont comandos Break
            PTVirg { Case ( $3,$6 ) }
102      ;
103
104 default: Default DoisPont comandos { Default ($3) };
105
106
107
108 expressao: expressao And expr10 { ExpBin (And, $1, $3) }
109      | expressao Or  expr10 { ExpBin (Or , $1, $3) }
110          | expr10 { $1 }
111          ;
112
113
114 expr10: expr10 IgualI expr20 { ExpBin (IgualI, $1, $3) }
115  | expr10 Dif expr20 { ExpBin (Dif, $1, $3) }
116  | expr20 { $1 }
117  ;
118
119 expr20: expr20 Maior  expr30   { ExpBin (Maior , $1, $3) }
120      | expr20 Menor  expr30   { ExpBin (Menor , $1, $3) }
121      | expr20 MaiorI expr30  { ExpBin (MaiorI, $1, $3) }
122      | expr20 MenorI expr30  { ExpBin (MenorI, $1, $3) }
123      | expr30 { $1 }
124      ;
125
126
127 expr30: expr30 OpSoma expr40        { ExpBin (Soma, $1, $3)
        }
128      | expr30 OpSub  expr40         { ExpBin (Sub , $1, $3)
            }
129      | expr40 { $1 }
130      ;
131
132
133 expr40: expr40 OpMul expr50     { ExpBin (Mul, $1, $3) }
134      | expr40 OpDiv expr50      { ExpBin (Div, $1, $3) }
```

```
135      | expr50 { $1 }
136      ;
137
138
139 expr50: Not expr50 { ExpUn (Not, $2) }
140      | expr60 { $1 }
141      ;
142
143 expr60: LitInt    { ExpInt    $1 }
144       | LitFloat  { ExpFloat  $1 }
145       | LitChar   { ExpChar   $1 }
146       | LitBool   { ExpBool   $1 }
147       | STRING    { ExpString $1 }
148       | Ident  { ExpVar    $1 }
149       | AParen expressao FParen { $2 };
150
151 argumentos: /* nada */                  { [] }
152           | seq_expressoes              { $1 }
153       | seq_tipo_expressoes              { $1 }
154           ;
155
156 seq_expressoes: expressao                     { [$1] }
157           | seq_expressoes Virg expressao { $1 @ [$3] }
158               ;
159
160 seq_tipo_expressoes: Int expressao                    { [$2
    ] }
161           | seq_tipo_expressoes Virg Int expressao { $1 @ [
                $4] }
162               ;
```

3.4.1 Árvore Sintática

```
type programa = Programa of imports * decl_class

and imports = importt list
and importt =  Imp of imp
and imp = string

and decl_class = DeclClass of decl_fun
and decl_fun = DeclFun of comandos

and comandos = comando list
and comando = CmdPrint of expressao list
        | CmdPrintln of expressao list
        | CmdInc of expressao
        | CmdDecl of  expressao * tipo * expressao option
        | CmdDeclConstrut of expressao * tipo * tipo
        | CmdIf of expressao * comandos * comandos option
        | CmdWhile of expressao * comandos
        | CmdFor of comando * expressao * comando * comandos
        | CmdAtrib of expressao * expressao
        | CmdSwitch of expressao * cases * default option
        | CmdAtribNextFloat of expressao * expressao
        | CmdAtribNextInt of expressao * expressao
        | CmdAtribNextLine of expressao * expressao
        | CmdReturn of expressao

and expressao = ExpInt of int
              | ExpVar of string
          | ExpFloat of float
          | ExpChar  of char
          | ExpString  of string
              | ExpBin of operador * expressao * expressao
          | ExpUn of operador * expressao
          | ExpBool   of bool



and operador = Soma | Sub | Mul | Div
         | Maior | Menor |  MaiorI | MenorI
         | IgualI | Dif  | Or | And | Not

and tipo = Int | Bool | Char | String | Scanner | Float

and inicia_construtor = tipo

and cases = case list
and case = Case of expressao * comandos
and default = Default of comandos

type tvalor = VInt of int | VFloat of float | VBool of bool
    | VString of string | VChar of char
type info = { tipos: tipo;
              inicializada: bool;
```

```
52                  valor: tvalor option;
53                                 mutable endereco: int option
54                        }
```

3.5 Analisador Semântico

Na terceira etapa, de análise semântica, verifica-se os erros semânticos (por exemplo, a soma de um inteiro e um char) e coleta-se os requisitos necessários para a fase de geração do código objeto.

O analisador semântico trata a entrada sintática e transforma em uma representação mais simples e mais adaptada para a geração do código.

```
open ArvSint;;
open Printf;;

let fun_Int =
        let amb = Hashtbl.create 23 in
        Hashtbl.add amb Soma   [ (Int, Int, Int)];
        Hashtbl.add amb Sub    [ (Int, Int, Int)];
        Hashtbl.add amb Mul    [ (Int, Int, Int)];
        Hashtbl.add amb Div    [ (Int, Int, Int)];
        Hashtbl.add amb Or     [ (Bool, Bool, Bool);
                                 (Int, Int, Bool)];
        Hashtbl.add amb And    [ (Bool, Bool, Bool);
                                 (Int, Int, Bool)];
        Hashtbl.add amb Maior  [ (Int, Int, Bool)];
        Hashtbl.add amb Menor  [ (Int, Int, Bool)];
        Hashtbl.add amb IgualI [ (Int, Int, Bool)];
        Hashtbl.add amb Dif    [ (Int, Int, Bool)];
        Hashtbl.add amb MaiorI [ (Int, Int, Bool)];
    Hashtbl.add amb MenorI [ (Int, Int, Bool)];
        amb

let fun_Float =
            let amb = Hashtbl.create 23 in
            Hashtbl.add amb Soma   [ (Float, Float, Float)];
      Hashtbl.add amb Sub    [ (Float, Float, Float)];
      Hashtbl.add amb Mul    [ (Float, Float, Float)];
      Hashtbl.add amb Div    [ (Float, Float, Float)];
      Hashtbl.add amb Or     [ (Bool, Bool, Bool);
                               (Float, Float, Bool)];
      Hashtbl.add amb And    [ (Bool, Bool, Bool);
                               (Float, Float, Bool)];
      Hashtbl.add amb Maior  [ (Float, Float, Bool)];
      Hashtbl.add amb Menor  [ (Float, Float, Bool)];
      Hashtbl.add amb IgualI   [ (Float, Float, Bool)];
      Hashtbl.add amb Dif    [ (Float, Float, Bool)];
      Hashtbl.add amb MaiorI [ (Float, Float, Bool)];
      Hashtbl.add amb MenorI [ (Float, Float, Bool)];
            amb

let fun_Bool =
    let amb = Hashtbl.create 23 in
        Hashtbl.add amb Or   [ (Bool, Bool, Bool)];
        Hashtbl.add amb And  [ (Bool, Bool, Bool)];
        Hashtbl.add amb IgualI [ (Bool, Bool, Bool)];
        Hashtbl.add amb Dif  [ (Bool, Bool, Bool)];
        amb
```

```
47
48  let fun_Char =
49              let amb = Hashtbl.create 23 in
50              Hashtbl.add amb Soma   [ (Char, Char, Char)];
51              Hashtbl.add amb Sub    [ (Char, Char, Char)];
52              Hashtbl.add amb Mul    [ (Char, Char, Char)];
53              Hashtbl.add amb Div    [ (Char, Char, Char)];
54              Hashtbl.add amb Or     [ (Bool, Bool, Bool);
55                                       (Char, Char, Bool)];
56              Hashtbl.add amb And    [ (Bool, Bool, Bool);
57                                       (Char, Char, Bool)];
58              Hashtbl.add amb Maior  [ (Char, Char, Bool)];
59              Hashtbl.add amb Menor  [ (Char, Char, Bool)];
60              Hashtbl.add amb IgualI   [ (Char, Char, Bool)];
61              Hashtbl.add amb Dif    [ (Char, Char, Bool)];
62              Hashtbl.add amb MaiorI [ (Char, Char, Bool)];
63              Hashtbl.add amb MenorI [ (Char, Char, Bool)];
64              amb
65
66  let fun_String =
67          let amb = Hashtbl.create 23 in
68          Hashtbl.add amb Soma [ (String, String, String)];
69          Hashtbl.add amb IgualI [ (String, String, Bool  )];
70          Hashtbl.add amb Dif    [ (String, String, Bool)];
71              amb
72
73  let fun_Scanner =
74           let amb = Hashtbl.create 23 in
75           amb
76
77
78  let fun_un_Bool =
79              let amb = Hashtbl.create 10 in
80              Hashtbl.add amb Not [(Bool,Bool)];
81              amb
82
83  let insere_var amb v tp =
84      if(Hashtbl.mem amb v) then
85          failwith "Variavel ja declarada"
86      else
87        let entrada = { tipos = tp; inicializada = false;
              valor=None;endereco=None;} in
88              Hashtbl.add amb v entrada
89
90  let rec analisa_op t1 t2 ls =
91      match ls with
92          | (tp1,tp2,tp3)::ls ->
93                  if ((t1 == tp1) && (t2 == tp2)) then tp3
94                  else analisa_op t1 t2 ls
95          | [] -> failwith "O tipo dos operandos deveria ser o
                mesmo"
96
97  let rec analisa_op_un t1 ls =
98      match ls with
```

```
99          | (tp1,tp3)::ls ->
100                 if (t1 == tp1) then tp3
101                 else analisa_op_un t1 ls
102         | [] -> failwith "O tipo dos operandos deveria ser o
                mesmo"
103
104 and verifica_exp op t1 t2 =
105     if (t1 <> t2) then
106         failwith "Erro ao verificar Expressao os tipos
                deveriam ser iguais!"
107     else
108     (match t1 with
109     |   Int    -> let tipo_op = Hashtbl.find fun_Int op   in
             analisa_op t1 t2 tipo_op
110         | Float  -> let tipo_op = Hashtbl.find fun_Float op
                in analisa_op t1 t2 tipo_op
111         | Char   -> let tipo_op = Hashtbl.find fun_Char op
                in analisa_op t1 t2 tipo_op
112         | Bool   -> let tipo_op = Hashtbl.find fun_Bool op
                in analisa_op t1 t2 tipo_op
113         | String -> let tipo_op = Hashtbl.find fun_String op
                  in analisa_op t1 t2 tipo_op
114         | Scanner -> let tipo_op = Hashtbl.find fun_Scanner
                op in analisa_op t1 t2 tipo_op)
115
116 and verifica_exp_un op t1 =
117     match t1 with
118         | Bool  -> let tipo_op = Hashtbl.find fun_un_Bool op
                  in analisa_op_un t1 tipo_op
119         | _ -> failwith "erro"
120
121
122
123 and analisa_exp amb exp =
124   match exp with
125   | ExpInt _     -> Int
126   | ExpFloat _   -> Float
127   | ExpChar _    -> Char
128   | ExpBool _    -> Bool
129   | ExpString v ->
130             if(v="%d") then Int
131             else if (v = "%f") then Float
132             else if (v = "%c") then Char
133             else String
134
135   | ExpVar v ->
136      (try
137           let entrada = Hashtbl.find amb v in
138                 if (entrada.inicializada) then
139                         entrada.tipos
140               else
141                   let msg = Printf.sprintf "A vari vel %s
                        n o foi inicializada" v in
142                     failwith msg
```

```
143       with Not_found ->
144             let msg = Printf.sprintf "Nome %s invalido.
                  Tente a,b,c ou d" v in
145             failwith msg)
146   | ExpBin (op, e1, e2) ->
147         let _tipo1 = analisa_exp amb e1
148         and _tipo2 = analisa_exp amb e2 in
149         verifica_exp op _tipo1 _tipo2
150   | ExpUn (op, e1) ->
151         let _tipo1 = analisa_exp amb e1 in
152               verifica_exp_un op _tipo1
153
154 let rec analisa_cmds amb cmds =
155     match cmds with
156     |  cmd :: cmds -> analisa_cmd amb cmd; analisa_cmds amb
          cmds
157         | [] -> ignore()
158
159 and analisa_cmd amb cmd =
160   match cmd with
161     | CmdAtrib (v, exp) ->
162         (match v with
163             | ExpVar v ->
164         if(Hashtbl.mem amb v) then
165             let _tipo = analisa_exp amb exp in
166             let entrada = Hashtbl.find amb v in
167             if(_tipo <> entrada.tipos) then
168                 failwith "Erro na Atribuicao: Os tipos
                      deveriam ser iguais!"
169             else
170                 Hashtbl.replace amb v { entrada with
                      inicializada = true};
171                 Printf.printf "%s alterada\n" v
172         else
173                 failwith "Variavel nao foi declarada!"
174             | _ -> ignore())
175
176     | CmdDecl (nome,ttipo,exp) ->
177         (match nome with
178             | ExpVar nome ->
179             insere_var amb nome ttipo;
180                     let entrada = Hashtbl.find amb nome in
181                         Hashtbl.replace amb nome { entrada
                              with tipos = ttipo};
182                         Printf.printf "declaracao\n";
183                         (match exp with
184                             | None -> ignore()
185                             | Some exp ->
186                         let _tipo1 = analisa_exp amb exp in
187                         if(_tipo1 <> entrada.tipos) then
188                             failwith "A expressao atribuida
                                  a essa variavel nao
                                  corresponde com o seu tipo"
189                         else
```

```
190                             Hashtbl.replace amb nome {
                                    entrada with inicializada =
                                    true};
191                             Printf.printf "%s Variavel
                                    declarada e inicializada\n"
                                    nome)
192                             | _ -> ignore())
193
194
195     | CmdPrint exps -> ignore ()
196
197     | CmdPrintln exps -> ignore()
198
199     | CmdAtribNextInt (v, exps)->
200       (match v with
201         | ExpVar v ->
202           if(Hashtbl.mem amb v) then
203             let entrada = Hashtbl.find amb v in
204             if(entrada.tipos <> Int) then
205               failwith "Esta variavel nao e do tipo Int para
                        receber NextInt()"
206             else
207               Hashtbl.replace amb v { entrada with
                        inicializada = true};
208               Printf.printf "%s -> NextInt()\n" v
209           else
210             failwith "Variavel nao foi declarada"
211          | _ -> ignore())
212
213
214     | CmdAtribNextLine (v, exps)->
215     (match v with
216         | ExpVar v ->
217           if(Hashtbl.mem amb v) then
218             let entrada = Hashtbl.find amb v in
219             if(entrada.tipos <> Char || entrada.tipos <>
                    Char) then
220               failwith "Esta variavel nao e do tipo String
                        para receber NextString()"
221             else
222               Hashtbl.replace amb v { entrada with
                        inicializada = true};
223               Printf.printf "%s -> NextLine()\n" v
224           else
225             failwith "Variavel nao foi declarada"
226          | _ -> ignore())
227
228   | CmdAtribNextFloat (v, exps)->
229     (match v with
230         | ExpVar v ->
231           if(Hashtbl.mem amb v) then
232             let entrada = Hashtbl.find amb v in
233             if(entrada.tipos <> Float) then
234               failwith "Esta variavel nao e do tipo Float
```

```
                para receber NextFloat()"
235           else
236             Hashtbl.replace amb v { entrada with
                    inicializada = true};
237             Printf.printf "%s -> NextFloat()\n" v
238         else
239           failwith "Variavel nao foi declarada"
240         | _ -> ignore())
241
242   | CmdWhile (exp,cmds) ->
243           let t = analisa_exp amb exp in
244                   if(t <> Bool) then
245                       let msgerro = Printf.sprintf "O tipo
                             de expressao dentro do While
                            deveria ser booleana!" in
246                   failwith msgerro
247                   else
248                   analisa_cmds amb cmds
249
250   | CmdFor (atrib,exp1,exp2,cmds) ->
251           analisa_cmd amb atrib;
252           let _tipo1 = analisa_exp amb exp1 in
253           let _tipo2 = analisa_cmd amb exp2 in
254                   let t = analisa_exp amb exp1 in
255                       if(t <> Bool) then
256                           let msgerro = Printf.sprintf "O
                                tipo de expressao dentro do
                                For deveria ser booleana" in
257                            failwith msgerro
258                       else
259                           analisa_cmds amb cmds
260   | CmdIf (exp,cmds1,cmds2) ->
261           let t = analisa_exp amb exp in
262               if(t <> Bool) then
263               let msgerro = Printf.sprintf "O tipo da
                      expressao deveria ser booleana!" in
264               failwith msgerro
265               else
266                    analisa_cmds amb cmds1;
267                       (match cmds2 with
268                           | None -> ignore()
269                           | Some cmds2 -> analisa_cmds amb
                                cmds2)
270
271
272   | CmdSwitch (exp, cases, default) ->
273           let tiposwitch = analisa_exp amb exp in
274               (match cases with
275                 | [] -> ignore()
276                 | Case(a,b)::cases ->
277                   let tipocase = analisa_exp amb a in
278                   if(tipocase <> tiposwitch) then
279                           failwith "O tipo do case e da
                               funcao switch deveriam ser
```

```
                                iguais!"
280                         else
281                             analisa_cmds amb b);
282                         (match default with
283                             | None -> ignore()
284                             | Some Default(a) ->
                                    analisa_cmds amb a )
285
286
287     | CmdInc (v) -> ignore()
288
289     | CmdDeclConstrut (nome, ttipo, tipodonew) ->
290         (match nome with
291             | ExpVar nome ->
292             insere_var amb nome ttipo;
293                     let entrada = Hashtbl.find amb nome in
294                         Hashtbl.replace amb nome { entrada
                            with tipos = ttipo};
295                         if (ttipo <> tipodonew) then
296                         let msgerro = Printf.sprintf "
                            Construtor deveria ser igual ao
                            tipo da variavel %s" nome in
297                                 failwith msgerro
298                     else
299                                 Printf.printf "declaracao
                                    com construtor (%s)\n"
                                    nome;
300         | _ -> ignore())
301
302   | CmdReturn (exp) -> ignore()
303
304
305
306
307 let semantico arv =
308   let (ambiente : (string, info) Hashtbl.t) = Hashtbl.create
          5 in
309     match arv with
310     Programa(imports,DeclClass(DeclFun(arv)))-> analisa_cmds
          ambiente arv;
311     ambiente
```

3.6 Interpretador

O Interpretador é opcional em um compilador. Ele interpreta (traduz) o código e o programa vai sendo utilizado na medida em que vai sendo traduzido.

O interpretador analisa sintaticamente e semanticamente o código, se estas duas etapas forem realizadas e executadas de forma correta o código está pronto para funcionar.

```
1  open ArvSint;;
2  open Printf;;
3
4  let rec analisa_exp amb exp =
5    match exp with
6    | ExpInt i    -> VInt i
7    | ExpFloat i  -> VFloat i
8    | ExpBool i   -> VBool i
9    | ExpChar i   -> VChar i
10   | ExpString v -> VString v
11   | ExpVar v ->
12         let entrada = Hashtbl.find amb v in
13                 (match entrada.valor with
14                 | Some v -> v
15                 | None -> failwith "expressao nao
                       inicializada")
16   | ExpBin (Soma,e1,e2) ->
17      let arg1 = analisa_exp amb e1
18      and arg2 = analisa_exp amb e2 in
19      (match (arg1,arg2) with
20          | (VInt a1, VInt a2) -> VInt (a1+a2)
21          | (VInt a1, VFloat a2) -> VFloat (float(a1) +. a2)
22          | (VFloat a1, VInt a2) -> VFloat (float(a2) +. a1)
23          | (VFloat a1, VFloat a2) -> VFloat (a1 +. a2)
24          | _ -> failwith "Erro soma nao envolve numeros")
25   | ExpBin (Sub,e1,e2) ->
26         let arg1 = analisa_exp amb e1 in
27     let arg2 = analisa_exp amb e2 in
28     (match (arg1,arg2) with
29         | (VInt a1, VInt a2) -> VInt (a1-a2)
30         | (VInt a1, VFloat a2) -> VFloat (float(a1) -. a2)
31         | (VFloat a1, VInt a2) -> VFloat (float(a2) -. a1)
32         | (VFloat a1, VFloat a2) -> VFloat (a1 -. a2)
33         | _ -> failwith "Erro subtra o nao envolve
              numeros")
34   | ExpBin (Mul,e1,e2) ->
35         let arg1 = analisa_exp amb e1
36     and arg2 = analisa_exp amb e2 in
37         (match (arg1,arg2) with
38         | (VInt a1, VInt a2) -> VInt (a1*a2)
39         | (VInt a1, VFloat a2) -> VFloat (float(a1) *. a2)
40         | (VFloat a1, VInt a2) -> VFloat (float(a2) *. a1)
41         | (VFloat a1, VFloat a2) -> VFloat (a1 *. a2)
42         | _ -> failwith "Erro multiplica o nao envolve
              numeros")
43   | ExpBin (Div,e1,e2) ->
```

```
44          let arg1 = analisa_exp amb e1
45          and arg2 = analisa_exp amb e2 in
46          (match (arg1,arg2) with
47          | (VInt a1, VInt a2) -> VInt (a1/a2)
48          | (VInt a1, VFloat a2) -> VFloat (float(a1) /. a2)
49          | (VFloat a1, VInt a2) -> VFloat (float(a2) /. a1)
50          | (VFloat a1, VFloat a2) -> VFloat (a1 /. a2)
51          | _ -> failwith "Erro divis o nao envolve numeros")
52    | ExpBin (Maior,e1,e2) ->
53          let arg1 = analisa_exp amb e1
54          and arg2 = analisa_exp amb e2 in
55          (match (arg1,arg2) with
56          | (VInt a1, VInt a2) -> VBool (a1>a2)
57          | (VInt a1, VFloat a2) -> VBool (float(a1) > a2)
58          | (VFloat a1, VInt a2) ->  VBool (float(a2) > a1)
59          | (VFloat a1, VFloat a2) -> VBool (a1 > a2)
60          | _ -> failwith "Erro maior nao envolve numeros")
61    | ExpBin (Menor,e1,e2) ->
62          let arg1 = analisa_exp amb e1
63          and arg2 = analisa_exp amb e2 in
64          (match (arg1,arg2) with
65          | (VInt a1, VInt a2) -> VBool (a1<a2)
66          | (VInt a1, VFloat a2) -> VBool (float(a1) < a2)
67          | (VFloat a1, VInt a2) ->  VBool (float(a2) < a1)
68          | (VFloat a1, VFloat a2) -> VBool (a1 < a2)
69          | _ -> failwith "Erro menor esta fazendo uma
                operacao errada")
70          | ExpBin (MaiorI,e1,e2) ->
71          let arg1 = analisa_exp amb e1
72       and arg2 = analisa_exp amb e2 in
73          (match (arg1,arg2) with
74          | (VInt a1, VInt a2) -> VBool (a1>=a2)
75          | (VInt a1, VFloat a2) -> VBool (float(a1) >= a2)
76          | (VFloat a1, VInt a2) ->  VBool (float(a2) >= a1)
77          | (VFloat a1, VFloat a2) -> VBool (a1 >= a2)
78          | _ -> failwith "Erro maior igual nao envolve
                numeros")
79    | ExpBin (MenorI,e1,e2) ->
80          let arg1 = analisa_exp amb e1
81          and arg2 = analisa_exp amb e2 in
82          (match (arg1,arg2) with
83          | (VInt a1, VInt a2) -> VBool (a1<=a2)
84          | (VInt a1, VFloat a2) -> VBool (float(a1) <= a2)
85          | (VFloat a1, VInt a2) ->  VBool (float(a2) <= a1)
86          | (VFloat a1, VFloat a2) -> VBool (a1 <= a2)
87          | _ -> failwith "Erro menor igual nao envolve
                numeros")
88    | ExpBin (IgualI,e1,e2) ->
89        let arg1 = analisa_exp amb e1
90          and arg2 = analisa_exp amb e2 in
91          (match (arg1,arg2) with
92          | (VInt a1, VInt a2) -> VBool (a1=a2)
93          | (VInt a1, VFloat a2) -> VBool (float(a1) = a2)
94          | (VFloat a1, VInt a2) ->  VBool (float(a2) = a1)
```

```
95          | (VFloat a1, VFloat a2) -> VBool (a1 = a2)
96          | (VChar a1, VChar a2) -> VBool (a1 = a2)
97          | (VString a1, VString a2) -> VBool (a1 = a2)
98          | _ -> failwith "Erro igual igual nao envolve
                numeros")
99    | ExpBin (Dif,e1,e2) ->
100         let arg1 = analisa_exp amb e1
101         and arg2 = analisa_exp amb e2 in
102         (match (arg1,arg2) with
103         | (VInt a1, VInt a2) -> VBool (not(a1=a2))
104         | (VInt a1, VFloat a2) -> VBool (not(float(a1) = a2)
                )
105         | (VFloat a1, VInt a2) ->  VBool (not(float(a2) = a1
                ))
106         | (VFloat a1, VFloat a2) -> VBool (not(a1 = a2))
107         | _ -> failwith "Erro diferente nao envolve numeros"
                )
108   | ExpBin (And,e1,e2) ->
109         let arg1 = analisa_exp amb e1
110         and arg2 = analisa_exp amb e2 in
111         (match (arg1,arg2) with
112         | (VBool a1, VBool a2) -> VBool (a1 && a2)
113         | _ -> failwith "Erro and nao envolve numeros")
114   | ExpBin (Or,e1,e2) ->
115        let arg1 = analisa_exp amb e1
116         and arg2 = analisa_exp amb e2 in
117         (match (arg1,arg2) with
118         | (VBool a1, VBool a2) -> VBool (a1 || a2)
119         | _ -> failwith "Erro or nao envolve numeros")
120   | ExpBin (Not, e1, e2) -> failwith ("O operador Not nao e
      uma expressao binaria")
121   | ExpUn (Not, e1) ->
122      let arg1 = analisa_exp amb e1 in
123            (match arg1 with
124                | (VBool a1) -> VBool ( not(a1))
125                | _ -> failwith("Erro negacao envolvendo
                       dados errados"))
126
127
128   | ExpUn(Soma, e1) -> failwith ("O operador Soma nao e uma
      expressao unaria")
129   | ExpUn(Sub, e1) -> failwith ("O operador Sub nao e uma
      expressao unaria")
130   | ExpUn(Mul, e1) -> failwith ("O operador Mul nao e uma
      expressao unaria")
131   | ExpUn(Div, e1) -> failwith ("O operador Div nao e uma
      expressao unaria")
132   | ExpUn(Maior, e1) -> failwith ("O operador Maior nao e uma
       expressao unaria")
133   | ExpUn(Menor, e1) -> failwith ("O operador Menor nao e uma
       expressao unaria")
134   | ExpUn(MaiorI, e1) -> failwith ("O operador MaiorI nao e
      uma expressao unaria")
135   | ExpUn(MenorI, e1) -> failwith ("O operador MenorI nao e
```

```
    uma expressao unaria")
 | ExpUn(IgualI, e1) -> failwith ("O operador IgualI nao e
    uma expressao unaria")
 | ExpUn(Dif, e1) -> failwith ("O operador Dif nao e uma
    expressao unaria")
 | ExpUn(Or, e1) -> failwith ("O operador Or nao e uma
    expressao unaria")
 | ExpUn(And, e1) -> failwith ("O operador And nao e uma
    expressao unaria")

and analisa_args amb nome args =
        (match args with
         | a::rgs ->
             let exp = analisa_exp amb a in
             let entrada = Hashtbl.find amb nome in
             Hashtbl.replace amb nome { entrada with valor =
                 Some exp };
             analisa_args amb nome rgs
            | [] -> ignore())


let rec imprime_exps amb exps =
    (match exps with
        | [] -> ignore()
        | e::xps -> let cabeca = analisa_exp amb e in
        (match cabeca with
            | VInt cabeca -> print_int cabeca;
            | VFloat cabeca -> print_float cabeca;
            | VBool true -> print_string "true";
            | VBool false -> print_string "false";
            | VChar cabeca -> print_string (Char.escaped
               cabeca);
            | VString cabeca -> print_string cabeca;


        )
    )


let rec analisa_cmds amb cmds =
    match cmds with
    |  cmd :: cmds -> analisa_cmd amb cmd; analisa_cmds amb
       cmds
        | [] -> ignore()

and executar_while amb exp cmds =
    let entrada = analisa_exp amb exp in
    (match entrada with
  | VBool e1->
  if (e1==true) then
    begin
        analisa_cmds amb cmds;
        executar_while amb exp cmds
    end
```

```
182     else
183         ignore()
184   | _ -> failwith "Erro While")
185
186 and executar_for amb atrib exp1 exp2 cmds =
187         let entrada = analisa_exp amb exp1 in
188                 (match entrada with
189                 | VBool e1 ->
190         if(e1 == true) then
191                 begin
192                     analisa_cmds amb cmds;
193                     analisa_cmd amb exp2;
194                     executar_for amb atrib exp1 exp2 cmds
195                 end
196                 else
197                     ignore()
198                 | _ -> failwith "Erro For")
199
200
201 and analisa_cmd amb cmd =
202     match cmd with
203     | CmdAtrib (v, exp) ->
204     (match v with
205         | ExpVar v->
206             let _arg = analisa_exp amb exp in
207             let entrada = Hashtbl.find amb v in
208             Hashtbl.replace amb v { entrada with valor =
                    Some _arg};
209         | _ -> ignore())
210
211     | CmdDecl (nome,ttipo,exp) ->
212         (match nome with
213             | ExpVar nome ->
214             (match exp with
215                 | None ->
216                 let entrada = Hashtbl.find amb nome in
217                     Hashtbl.replace amb nome { entrada with
                            valor = None}
218                 | Some exp ->
219                 let _arg = analisa_exp amb exp in
220                     let entrada = Hashtbl.find amb nome in
221                         Hashtbl.replace amb nome { entrada
                                with valor = Some _arg};)
222         | _ -> ignore())
223
224     | CmdDeclConstrut (nome,ttipo, exp) ->
225     (match nome with
226         | ExpVar nome ->
227             let entrada = Hashtbl.find amb nome in
228                 Hashtbl.replace amb nome { entrada with
                        tipos = ttipo}
229         | _ -> ignore());
230
231
```

```
232     | CmdWhile (exp,cmds) ->
233         executar_while amb exp cmds
234
235     | CmdFor (atrib,exp1,exp2,cmds) ->
236         analisa_cmd amb atrib;
237         executar_for amb atrib exp1 exp2 cmds;
238
239     | CmdIf (exp,cmds1,cmds2) ->
240             let entrada = analisa_exp amb exp in
241             (match entrada with
242                 | VBool e1 ->
243                     if(e1) then
244                         analisa_cmds amb cmds1
245                     else
246                         (match cmds2 with
247                         | None -> ignore()
248                         | Some cmds2 -> analisa_cmds amb
                              cmds2)
249                 | _ -> failwith "Erro no if")
250
251     | CmdInc (v) ->
252         (match v with
253                 | ExpVar v ->
254                 let entrada = Hashtbl.find amb v in
255                 (match entrada.valor with
256                     | Some valor ->
257                     (match valor with
258                         | VInt valor ->
259                             let valorint = valor + 1 in
260                             Hashtbl.replace amb v { entrada
                                  with valor = Some (VInt(
                                  valorint)) };
261                         | VFloat valor ->
262                             let valorfloat = valor +. 1.0 in
263                             Hashtbl.replace amb v { entrada
                                  with valor = Some (VFloat(
                                  valorfloat)) };
264                         | _ -> failwith "incr nao funciona
                              com este tipo!")
265                         | None -> failwith "erro no
                              incrementa")
266                             | _ -> ignore())
267
268     | CmdSwitch (exp, cases, default) ->
269             (match cases with
270                 | [] -> ignore()
271                 | Case(a,b)::cases ->
272                     analisa_cmds amb b);
273             (match default with
274                 | None -> ignore()
275                 | Some Default(a) -> analisa_cmds amb a )
276     | CmdPrint(exps) -> imprime_exps amb exps
277
278     | CmdPrintln(exps) ->   imprime_exps amb exps
```

```
279
280     | CmdAtribNextFloat (exp, v)->
281         (match exp with
282             | ExpVar exp ->
283                 let entrada = Hashtbl.find amb exp in
284                     let valorFloat = VFloat (read_float())in
285                         Hashtbl.replace amb exp { entrada
                                with valor = Some valorFloat};
286             | _ -> ignore()
287         )
288     | CmdAtribNextLine (exp, v)->
289         (match exp with
290         | ExpVar exp ->
291             let entrada = Hashtbl.find amb exp in
292                 let valorString = VString (read_line())in
293                     Hashtbl.replace amb exp { entrada with
                            valor = Some valorString};
294         | _ -> ignore()
295         )
296     | CmdAtribNextInt (exp, v)->
297         (match exp with
298             | ExpVar exp ->
299                 let entrada = Hashtbl.find amb exp in
300                     let valorInt = VInt (read_int())in
301                         Hashtbl.replace amb exp { entrada
                                with valor = Some valorInt};
302             | _ -> ignore()
303         )
304
305     | CmdReturn (exp) -> ignore()
306
307
308
309
310
311
312
313
314
315 (* Printf.printf "System.out.print(%s)\n" exps *)
316
317
318
319 let rec interpretador ambiente arv =
320     match arv with
321     Programa(imports,DeclClass(DeclFun(arv)))-> analisa_cmds
            ambiente arv;
322     Printf.printf "\n--------------C digo interpretado com
        sucesso!---------------\n"
```

3.7 Gerador

O gerador recebe o resultado do sintático (árvore) e o semântico (ambiente) e traduz para a linguagem de máquina (cil).

```
open ArvSint;;
open Printf;;
open List;;

let contador_registrador = ref 0 (*Contador para variaveis*)
let contador_label = ref 0 (*Contador para labels*)
let contador_arg = ref 0 (*Contador para localsinit *)

let op inst a = sprintf "%s %d" inst a (*Print operador*)

let opr_int op = (*Operacoes para int *)
  match op with
    | Soma -> "add"
    | Mul -> "mul"
    | Sub -> "sub"
    | Div -> "div"
    | Maior -> "bgt"
    | Menor -> "blt"
    | IgualI -> "beq"
    | MaiorI-> "bge"
    | MenorI -> "ble"
    | Dif -> "bne.un"
    | And -> "and"
    | Or -> "or"
    | Not -> "not"

let opr_float op = (*Operacoes para float*)
  match op with
    | Soma -> "add"
    | Mul -> "mul"
    | Sub -> "sub"
    | Div -> "div"
    | Maior -> "bgt"
    | Menor -> "blt"
    | IgualI -> "beq"
    | MaiorI-> "bge"
    | MenorI -> "ble"
    | Dif -> "bne.un"
    | And -> "and"
    | Or -> "or"
    | Not -> "not"

let opr_bool op = (*Operacoes para booleano*)
    match op with
      | Soma -> "add"
      | Mul -> "mul"
      | Sub -> "sub"
      | Div -> "div"
      | Maior -> "bgt"
      | Menor -> "blt"
```

```
51        | IgualI -> "beq"
52        | MaiorI-> "bge"
53        | MenorI -> "ble"
54        | Dif -> "bne.un"
55        | And -> "and"
56        | Or -> "or"
57        | Not -> "not"
58
59  let opr_char op = (*Operacoes para char*)
60    match op with
61      | Maior -> "bgt"
62      | Menor -> "blt"
63      | IgualI -> "beq"
64      | MaiorI-> "bge"
65      | MenorI -> "ble"
66      | Dif -> "bne.un"
67      | And -> "and"
68      | Or -> "or"
69      | Not -> "not"
70      | _ -> failwith "Operacao nao reconhecida para char"
71
72  let operador t op = (*Seleciona o operador de acordo com o
        tipo da var*)
73    match t with
74      | Int -> opr_int op
75      | Float -> opr_float op
76      | Bool -> opr_int op
77      | String -> opr_int op
78      | Char -> opr_char op
79      | _ -> failwith "Tipo nao esperado aqui"
80
81  let prox_registrador() = (*Incrementa a variavel global que
        conta a quantidade de registrador*)
82        incr(contador_registrador);
83        !contador_registrador
84
85  let prox_label() = (*Incrementa a variavel global que conta
        a quantidade de label*)
86        incr(contador_label);
87        sprintf "Label%d" !contador_label
88
89  let prox_arg() =(*Incrementa a variavel global que conta a
        quantidade de argumentos no localinit*)
90      incr(contador_arg);
91      !contador_arg
92
93  let ant_arg() = (*Decrementa a variavel global que conta a
        quantidade de argumentos no localinit*)
94      decr(contador_arg)
95
96  let rec numera exp = (*Contador para a maxstack*)
97      (match exp with
98      | ExpInt _ -> 2
99      | ExpFloat _ -> 2
```

```
100     | ExpString _ -> 2
101     | ExpUn (_,e) -> let arv = numera e in
102       arv
103     | ExpBin (_,esq,dir) ->
104         let aesq = numera esq
105         and adir = numera dir in
106           let n = aesq
107           and m = adir in
108             let nr = if n == m then n + 2 else (max n m)
109             in nr
110     | _ -> 0)
111
112 (*and get_tipo exp = (*retorna o tipo da variavel*)
113       match exp with
114       | t -> t *)
115
116 and emite_maxstack amb arv = (*pegar o valor do maxstack
        verifica comandos*)
117   (match arv with
118       Programa(imports,DeclClass(DeclFun(arv))) ->
119       gen_cmds_maxstack amb arv)
120
121 and gen_maxstack ts cs = (*gerador maxstack*)
122   (match cs with
123   | CmdAtrib (esq , dir) -> let result = numera dir + numera
          esq in result
124   | CmdDecl (esq, tipo, dir) -> let result = numera esq in
125     (match dir with
126     | None ->
127       (result)
128     | Some dir ->
129       (result + numera dir))
130   | CmdIf (exp, cmd_entao, cmd_senao) -> let result = numera
          exp in result
131   | CmdWhile (exp,cmds) -> let result = numera exp in result
132   | CmdFor (atrib,exp1,exp2,cmds) -> let result = numera
        exp1 in result
133   | _ -> 0)
134
135 and gen_cmds_maxstack amb cmds = (* verifica max stack nos
        comanods*)
136     match cmds with
137       |  cmd :: cmds -> gen_maxstack amb cmd +
            gen_cmds_maxstack amb cmds
138       | [] -> 0
139
140 let endereco ts e = (*atribui um enderaco a uma variavel*)
141         (match e with
142        | ExpVar v ->
143           let entrada = Hashtbl.find ts v in
144           (match entrada.endereco with
145            | Some endr -> string_of_int endr
146            | None ->
147                let reg = prox_registrador() in
```

```
148                 Hashtbl.add ts v { entrada with endereco =
                        Some reg};
149                 string_of_int reg)
150         | _ -> failwith "esse endereco nao aponta para uma
                variavel")
151
152  let pegatipo amb exp = (* retorna o tipo da variavel*)
153    match exp with
154      |  ExpInt i -> Int
155      | ExpFloat f -> Float
156      | ExpVar v ->
157        let entrada = Hashtbl.find amb v in
158        entrada.tipos
159      | ExpBool i -> Bool
160      | ExpString i -> String
161      |_ -> String
162
163
164 let emite_ldloc ts var = [ sprintf "ldloc.%s\n" var ](*
        concatena ldloc na saida*)
165
166 let emite_op ts op tipo = (*concatena operador*)
167   let operacao = operador tipo op in
168   [sprintf " %s " operacao]
169
170 let emite_stloc ts var = [ sprintf "stloc.%s\n" var ](*
        concatena stloc na saida*)
171
172 let emite_relacional ts op label tipo =
173     let operacao = operador tipo op in
174     [ sprintf "%s %s\n" operacao label]
175
176 let rec gen_expressao ts exp = (*gerador de expressoes*)
177   match exp with
178   | ExpInt i -> [ sprintf "ldc.i4 %d\n" i ]
179   | ExpFloat f -> [ sprintf "ldc.r4 %e\n" f ]
180   | ExpBool b -> [ sprintf "testebool\n" ]
181   | ExpVar i ->
182     let endr = endereco ts exp in
183     emite_ldloc ts endr
184   | ExpBin(op,a1,a2) ->
185     gen_expressao ts a1 @
186     gen_expressao ts a2 @
187     let tipo = pegatipo ts a1 in
188     emite_op ts op tipo
189   | ExpChar c -> let codascii = pega_ascii c in [ sprintf "
        ldc.i4 %d\n" codascii ]
190   | ExpString s -> [ sprintf "ldstr \"%s\"\n" s ]
191   | ExpUn(op, a1) ->
192     (match op with
193     | Not -> gen_expressao ts a1 @ ["not"]
194     | _ -> [sprintf "ldc.i4 0\n"] @ gen_expressao ts a1 @
           emite_op ts op Int)  (* ERRO: usar match q resolve*)
195
```

```
196 and pega_ascii a = (*pega o valor ASCII de um char*)
197         Char.code a
198
199 let rec gen_atrib ts esq dir = (*gera comando atrib*)
200   let endr = endereco ts esq
201   and cod_dir = gen_expressao ts dir in
202       cod_dir @ emite_stloc ts endr
203
204 and gen_dec ts esq tipo dir = (*gera comando declaracao*)
205   let endr = endereco ts esq
206   and cod_dir = gen_expressao ts dir in
207     cod_dir @ emite_stloc ts endr
208
209
210
211 and gen_cmd ts cs = (*verifica qual comando eh e chama sua
      respc. funcao*)
212   (match cs with
213   | CmdAtrib (esq , dir) -> gen_atrib ts esq dir
214   | CmdDecl (esq, tipo, dir) ->
215     (match dir with
216     | None ->
217     (*  let endr = endereco ts esq in
218       emite_stloc ts endr *)
219       [""]
220     | Some dir ->
221       gen_dec ts esq tipo dir)
222
223   | CmdIf (exp, cmd_entao, cmd_senao) -> gen_se ts exp
        cmd_entao cmd_senao
224   | CmdWhile (exp,cmds) -> gen_while ts exp cmds
225   | CmdInc(v) ->
226       let endr = endereco ts v in
227       emite_ldloc ts endr @ [ sprintf "ldc.i4 1\n" ] @ [
            sprintf "add\n" ] @ emite_stloc ts endr
228   | CmdFor (atrib,exp1,exp2,cmds) -> gen_for ts atrib exp1
        exp2 cmds
229   | CmdPrint (exps) -> gen_imprima ts exps
230   | CmdPrintln (exps) -> gen_imprima ts exps
231   | CmdDeclConstrut (nome,ttipo, exp) -> []
232
233   | CmdAtribNextInt (exp, v)-> let leitura = endereco ts exp
        in
234   ["call string [mscorlib]System.Console::ReadLine()\n"] @ [
        "call int32 [mscorlib]System.Int32::Parse(string)\n"] @
        [sprintf "stloc.%s\n" leitura]
235
236   | CmdAtribNextFloat(exp, v) -> let leitura = endereco ts
        exp in
237   ["call string [mscorlib]System.Console::ReadLine()\n"] @ [
        "call float32 [mscorlib]System.Single::Parse(string)\n"
        ] @ [sprintf "stloc.%s\n" leitura]
238
```

```
239   | CmdAtribNextLine(exp, v) -> let leitura = endereco ts
          exp in
240   ["call string [mscorlib]System.Console::ReadLine()\n"] @ [
          "call char [mscorlib]System.Char::Parse(string)\n"] @ [
          sprintf "stloc.%s\n" leitura]
241
242   | _ -> failwith "Comando nao encontrado!")
243
244 and gen_imprima amb exp = (*gera comando imprime*)
245     (match exp with
246       |e::xp ->
247     let tipoExp =  pegatipo amb e in
248     (match tipoExp with
249       |Int -> gen_expressao amb e @ ["call void [mscorlib]
              System.Console::WriteLine(int32)\n"]
250       |Float -> gen_expressao amb e @ ["call void [mscorlib]
              System.Console::WriteLine(float32)\n"]
251     |Char -> gen_expressao amb e @ ["call void [mscorlib]
            System.Console::WriteLine(string)\n"]
252     |String ->gen_expressao amb e @ ["call void [mscorlib]
            System.Console::WriteLine(string)\n"]
253     |_ -> failwith "tipo nao reconhecido")
254     |[] -> [])
255
256
257 and gen_cmds amb cmds = (*chama o gera comando*)
258   match cmds with
259     |  cmd :: cmds -> gen_cmd amb cmd @ gen_cmds amb cmds
260     | [] -> []
261
262 and gen_while ts exp cmd = (* gerador para while*)
263   let label_faca = prox_label() in
264   let label_inicio = prox_label() in
265   let label_fim = prox_label() in
266   let recebe =
267   (match exp with
268   | ExpBin(op,a1,a2) -> gen_expressao ts a1 @ gen_expressao
          ts a2 @ emite_relacional ts op label_faca Int @ [
          sprintf "br %s\n" label_fim]
269   | _ -> failwith "Erro ao gerar codigo para o while")
270   and res_faca = gen_cmds ts cmd in
271   [sprintf "\n%s:\n" label_inicio] @ recebe @ [sprintf "%s:\
          n" label_faca] @ res_faca @ [sprintf "br %s\n"
          label_inicio] @ [sprintf "%s:\n" label_fim]
272
273 and gen_for ts atrib exp1 exp2 cmds = (*gerar comandor for*)
274   let label_faca = prox_label() in
275   let label_faca_verif = prox_label() in
276   let label_inicio = prox_label() in
277   let label_fim = prox_label() in
278   (*let res1 = gen_cmd ts atrib in*)
279   let (res1, var) =
280   (match atrib with
281   | CmdAtrib (esq , dir) ->
```

```
282     let endr = endereco ts esq
283     and cod_dir = gen_expressao ts dir in
284     (cod_dir @ emite_stloc ts endr, endr)
285   | _ -> failwith "erro ao gerar codigo no for" )
286   in
287   let res2 =
288   (match exp1 with
289   | ExpVar _     | ExpInt _ | ExpBool _ -> gen_expressao ts
          exp1 @ emite_stloc ts var
290   | ExpBin(op,a1,a2) -> gen_expressao ts a1 @ gen_expressao
          ts a2 @ emite_relacional ts op label_faca Int @ [
          sprintf "br %s\n" label_fim]
291   | _ -> failwith "erro na expressao binaria")
292   in
293   let res3 =
294    gen_cmd ts exp2
295   in
296   let res5 = gen_cmds ts cmds in
297   res1 @ [sprintf "\n%s:\n" label_faca_verif] @ res2 @ [
          sprintf "\n%s:\n" label_inicio] @ res3 @ [sprintf "br %
          s\n" label_faca_verif] @ [sprintf "\n%s:\n" label_faca]
           @ res5 @ [sprintf "br %s\n" label_inicio] @ [sprintf "
          \n%s:" label_fim]
298
299 and gen_se ts exp cmd_entao cmd_senao = (*gerar comando if*)
300   let label1 = prox_label() in
301   let label2 = prox_label() in
302   let recebe =
303   (match exp with
304   | ExpBin (op, a1, a2) ->
305      gen_expressao ts a1 @ gen_expressao ts a2 @
             emite_relacional ts op label1 Int
306   | _ -> failwith "erro no if")
307   and res_entao = gen_cmds ts cmd_entao @ [sprintf "br %s\n"
          label2]
308   and res_senao =
309   (match cmd_senao with
310   | Some ter -> gen_cmds ts ter @ [sprintf "br %s\n" label2]
311   | None -> [sprintf "br %s\n" label2])
312     in
313   recebe @ res_senao @ [sprintf "%s:\n" label1] @ res_entao
          @ [sprintf "%s:\n" label2]
314
315   and emite_prologo amb arv saida = (*gera cabecalho*)
316     (*  emitir saida *)
317       let pro = [".assembly extern mscorlib {}\n"] @ [".
              assembly codigoGerado\n"] @ ["{\n"] @ ["        .
              ver 1:0:1:0\n"] @ ["}\n"] @ [".module codigoGerado.
              exe\n"] @ [".method static void main() cil managed\
              n"] @ ["{\n"] @ rev(emite_vars amb arv) @ ["\n.
              entrypoint\n"] @ [sprintf ".maxstack %d\n" (
              emite_maxstack amb arv)] in
318       emitirlista saida pro
319
```

```
320 and geraCod amb arv = (*funcao que chama os gera comandos*)
321   (match arv with
322           Programa(imports,DeclClass(DeclFun(ls))) ->
323           (match ls with
324           | l::s ->
325         gen_cmd amb l @ gen_cmds amb s
326       | [] -> [])
327   )
328
329 and emitir saida str =
330   Printf.fprintf saida "%s" str
331
332 and emitirlista saida str =
333   match str with
334   | [] -> []
335   | a::b -> Printf.fprintf saida "%s" a; emitirlista saida b
336
337 and emite_fim amb arv saida = (*final do programa*)
338     let fim =
339     ["ret\n"] @ ["}\n"] in
340     emitirlista saida fim
341
342 and emite_vars amb arv = (*gera o locals init*)
343   let teste = [")"] in
344   (match arv with
345       Programa(imports,DeclClass(DeclFun(ls)))  ->
346     teste @ gen_cmdslocalsinit amb ls @ [".locals init("])
347
348 and gen_cmdslocalsinit amb cmds = (*gera locals init
      recursivamente*)
349     match cmds with
350       | cmd :: cmds -> gen_localsinit amb cmd @
            gen_cmdslocalsinit amb cmds
351           | [] -> []
352
353 and gen_localsinit amb comando= (*verifica cada declaracao
      para colocar o tipo no  localsinit*)
354     let cont = prox_arg() in
355       (match comando with
356           | CmdDecl (esq, tipo, dir) ->
357             (match tipo with
358             |   Int ->
359               if cont > 1 then
360                 [",int32"]
361               else
362                 ["int32"]
363             | Float ->
364                 if cont > 1 then
365                 [",float32"]
366               else
367                 ["float32"]
368             | Char ->
369                 if cont > 1 then
370                 [",char"]
```

```
371                 else
372                   ["char"]
373           | String ->
374             if cont > 1 then
375               [",string"]
376             else
377              ["string"]
378           | _ -> failwith "Erro. Tipo nao reconhecido!!!")
379         | _ -> ant_arg(); [])
380
381 let rec gera amb arv = (* chama todas as outras funcoes eh
        chamado no carregador.ml *)
382   contador_registrador := -1;
383   let saida = open_out ("codigoGerado.il") in
384   emite_fim amb arv saida @ emitirlista saida (geraCod amb
        arv) @ emite_prologo amb arv saida
```

3.8 Representação Intermediária

A Representação Intermediária ela independe da linguagem de máquina. O código é traduzido para um grafo intermediário que permite a análise do fluxo de exeucação antes de se criar as intruções para o CPU.

Segue o código inicial da RI, pois houve cancelamento dessa etapa, pois nosso assembly é de pilha e essa etapa não seria muito importante.

Segue o código:

```
1  open ArvSint;;
2  open ArvRI;;
3  open Printf;;
4
5  let rec analisa_exp amb exp =
6    match exp with
7    | ExpInt i    -> ConstI i
8    | ExpFloat i  -> ConstF i
9    | ExpBool i   -> ConstB i
10   | ExpChar i   -> ConstC i
11   | ExpString v -> ConstS v
12   | ExpVar v -> Mem(v)
13   | ExpBin (Soma,e1,e2) ->
14      let arg1 = analisa_exp amb e1
15      and arg2 = analisa_exp amb e2 in
16      (match (arg1,arg2) with
17          | (ConstI a1, ConstI a2) -> BinOp (Soma,ConstI a1,
                ConstI a2)
18          | (ConstI a1, ConstF a2) -> BinOp (Soma,ConstI a1,
                ConstF a2)
19          | (ConstF a1, ConstI a2) -> BinOp (Soma,ConstF a1,
                ConstI a2)
20          | (ConstF a1, ConstF a2) -> BinOp (Soma,ConstF a1,
                ConstF a2)
21          | (Mem a1, Mem a2) -> BinOp (Soma,Mem a1, Mem a2)
22          | _ -> failwith "Erro soma nao envolve tipos
                corretos")
23   | ExpBin (Sub,e1,e2) ->
24     let arg1 = analisa_exp amb e1 in
25         let arg2 = analisa_exp amb e2 in
26         (match (arg1,arg2) with
27             | (ConstI a1, ConstI a2) -> BinOp (Sub, ConstI
                  a1, ConstI a2)
28             | (ConstI a1, ConstF a2) -> BinOp (Sub, ConstI
                  a1, ConstF a2)
29             | (ConstF a1, ConstI a2) -> BinOp (Sub, ConstF
                  a1, ConstI a2)
30             | (ConstF a1, ConstF a2) -> BinOp (Sub, ConstF
                  a1, ConstF a2)
31             | (Mem a1, Mem a2) -> BinOp (Sub,Mem a1, Mem a2)
32             | _ -> failwith "Erro sub nao envolve tipos
                  corretos")
33   | ExpBin (Mul,e1,e2) ->
34     let arg1 = analisa_exp amb e1
35         and arg2 = analisa_exp amb e2 in
```

```
36          (match (arg1,arg2) with
37              | (ConstI a1, ConstI a2) -> BinOp (Mul, ConstI
                    a1, ConstI a2)
38              | (ConstI a1, ConstF a2) -> BinOp (Mul, ConstI
                    a1, ConstF a2)
39              | (ConstF a1, ConstI a2) -> BinOp (Mul, ConstF
                    a1, ConstI a2)
40              | (ConstF a1, ConstF a2) -> BinOp (Mul, ConstF
                    a1, ConstF a2)
41              | (Mem a1, Mem a2) -> BinOp (Mul,Mem a1, Mem a2)
42              | _ -> failwith "Erro mul nao envolve tipos
                    corretos")
43    | ExpBin (Div,e1,e2) ->
44      let arg1 = analisa_exp amb e1
45          and arg2 = analisa_exp amb e2 in
46          (match (arg1,arg2) with
47              | (ConstI a1, ConstI a2) -> BinOp (Div, ConstI
                    a1, ConstI a2)
48              | (ConstI a1, ConstF a2) -> BinOp (Div, ConstI
                    a1, ConstF a2)
49              | (ConstF a1, ConstI a2) -> BinOp (Div, ConstF
                    a1, ConstI a2)
50              | (ConstF a1, ConstF a2) -> BinOp (Div, ConstF
                    a1, ConstF a2)
51              | (Mem a1, Mem a2) -> BinOp (Div,Mem a1, Mem a2)
52              | _ -> failwith "Erro div nao envolve tipos
                    corretos")
53    | ExpBin (Maior,e1,e2) ->
54      let arg1 = analisa_exp amb e1
55          and arg2 = analisa_exp amb e2 in
56          (match (arg1,arg2) with
57              | (ConstI a1, ConstI a2) -> BinOp (Maior, ConstI
                      a1, ConstI a2)
58              | (ConstI a1, ConstF a2) -> BinOp (Maior, ConstI
                      a1, ConstF a2)
59              | (ConstF a1, ConstI a2) -> BinOp (Maior, ConstF
                      a1, ConstI a2)
60              | (ConstF a1, ConstF a2) -> BinOp (Maior, ConstF
                      a1, ConstF a2)
61              | (Mem a1, Mem a2) -> BinOp (Maior,Mem a1, Mem
                    a2)
62              | _ -> failwith "Erro maior nao envolve tipos
                    corretos")
63    | ExpBin (Menor,e1,e2) ->
64      let arg1 = analisa_exp amb e1
65          and arg2 = analisa_exp amb e2 in
66          (match (arg1,arg2) with
67              | (ConstI a1, ConstI a2) -> BinOp (Menor, ConstI
                      a1, ConstI a2)
68              | (ConstI a1, ConstF a2) -> BinOp (Menor, ConstI
                      a1, ConstF a2)
69              | (ConstF a1, ConstI a2) -> BinOp (Menor, ConstF
                      a1, ConstI a2)
70              | (ConstF a1, ConstF a2) -> BinOp (Menor, ConstF
```

```
                a1, ConstF a2)
71          | (Mem a1, Mem a2) -> BinOp (Menor,Mem a1, Mem
                a2)
72          | _ -> failwith "Erro maior nao envolve tipos
                corretos")
73  | ExpBin (MaiorI,e1,e2) ->
74    let arg1 = analisa_exp amb e1
75        and arg2 = analisa_exp amb e2 in
76        (match (arg1,arg2) with
77            | (ConstI a1, ConstI a2) -> BinOp (MaiorI,
                  ConstI a1, ConstI a2)
78            | (ConstI a1, ConstF a2) -> BinOp (MaiorI,
                  ConstI a1, ConstF a2)
79            | (ConstF a1, ConstI a2) -> BinOp (MaiorI,
                  ConstF a1, ConstI a2)
80            | (ConstF a1, ConstF a2) -> BinOp (MaiorI,
                  ConstF a1, ConstF a2)
81            | (Mem a1, Mem a2) -> BinOp (MaiorI,Mem a1, Mem
                  a2)
82            | _ -> failwith "Erro maior nao envolve tipos
                  corretos")
83  | ExpBin (MenorI,e1,e2) ->
84    let arg1 = analisa_exp amb e1
85        and arg2 = analisa_exp amb e2 in
86        (match (arg1,arg2) with
87            | (ConstI a1, ConstI a2) -> BinOp (MenorI,
                  ConstI a1, ConstI a2)
88            | (ConstI a1, ConstF a2) -> BinOp (MenorI,
                  ConstI a1, ConstF a2)
89            | (ConstF a1, ConstI a2) -> BinOp (MenorI,
                  ConstF a1, ConstI a2)
90            | (ConstF a1, ConstF a2) -> BinOp (MenorI,
                  ConstF a1, ConstF a2)
91            | (Mem a1, Mem a2) -> BinOp (MenorI,Mem a1, Mem
                  a2)
92            | _ -> failwith "Erro maior nao envolve tipos
                  corretos")
93
94  | ExpBin (IgualI,e1,e2) ->
95    let arg1 = analisa_exp amb e1
96        and arg2 = analisa_exp amb e2 in
97        (match (arg1,arg2) with
98            | (ConstI a1, ConstI a2) -> BinOp (IgualI,
                  ConstI a1, ConstI a2)
99            | (ConstI a1, ConstF a2) -> BinOp (IgualI,
                  ConstI a1, ConstF a2)
100           | (ConstF a1, ConstI a2) -> BinOp (IgualI,
                  ConstF a1, ConstI a2)
101           | (ConstF a1, ConstF a2) -> BinOp (IgualI,
                  ConstF a1, ConstF a2)
102           | (ConstC a1, ConstC a2) -> BinOp (IgualI,
                  ConstC a1, ConstC a2)
103           | (ConstS a1, ConstS a2) -> BinOp (IgualI,
                  ConstS a1, ConstS a2)
```

```
104             | (Mem a1, Mem a2) -> BinOp (IgualI,Mem a1, Mem
                    a2)
105             | _ -> failwith "Erro Igual Igual envolve tipos
                    desconhecidos!")
106   | ExpBin (Dif,e1,e2) ->
107     let arg1 = analisa_exp amb e1
108         and arg2 = analisa_exp amb e2 in
109         (match (arg1,arg2) with
110             | (ConstI a1, ConstI a2) -> BinOp (Dif, ConstI
                    a1, ConstI a2)
111             | (ConstI a1, ConstF a2) -> BinOp (Dif, ConstI
                    a1, ConstF a2)
112             | (ConstF a1, ConstI a2) ->  BinOp (Dif, ConstF
                    a1, ConstI a2)
113             | (ConstF a1, ConstF a2) -> BinOp (Dif, ConstF
                    a1, ConstF a2)
114             | (Mem a1, Mem a2) -> BinOp (Dif,Mem a1, Mem a2)
115             | _ -> failwith "Erro diferente nao envolve
                    tipos corretos")
116   | ExpBin (And,e1,e2) ->
117     let arg1 = analisa_exp amb e1
118         and arg2 = analisa_exp amb e2 in
119         (match (arg1,arg2) with
120             | (ConstB a1, ConstB a2) -> BinOp (And, ConstB
                    a1, ConstB a2)
121             | (Mem a1, Mem a2) -> BinOp (And,Mem a1, Mem a2)
122             | _ -> failwith "Erro and nao envolve tipos
                    corretos")
123   | ExpBin (Or,e1,e2) ->
124     let arg1 = analisa_exp amb e1
125         and arg2 = analisa_exp amb e2 in
126         (match (arg1,arg2) with
127             | (ConstB a1, ConstB a2) -> BinOp (Or, ConstB a1
                    , ConstB a2)
128             | (Mem a1, Mem a2) -> BinOp (Or,Mem a1, Mem a2)
129             | _ -> failwith "Erro or nao envolve numeros")
130   | ExpBin (Not, e1, e2) -> failwith ("O operador Not nao e
       uma expressao binaria")
131 (*  | ExpBin (Inc, e1, e2) -> failwith ("O operador
      Incrementa nao e uma expressao binaria")*)
132   | _ -> failwith "Error"
133
134
135 let rec analisa_cmds amb cmds =
136     match cmds with
137     |  cmd :: cmds -> analisa_cmd amb cmd; analisa_cmds amb
           cmds
138         | [] -> ignore()
139
140
141
142 and analisa_cmd amb cmd =
143     (match cmd with
144     | CmdAtrib (v, exp) ->
```

```
145         (match v with
146         | ExpVar v ->
147                 let _arg = analisa_exp amb exp in
148                 Printf.printf "\nComando Atribuicao\n";
149              Move (Mem v, Some _arg)
150         | _ -> failwith "Erro na Atribuicao" )
151
152     | CmdDecl (nome,ttipo,exp) ->
153         (match nome with
154         | ExpVar nome ->
155                 (match exp with
156             | None ->
157                                 Printf.printf "\nComando
                                        declaracao sem inicializacao\
                                        n";
158                                 Move (Mem nome, None)
159             | Some exp ->
160                                 let _arg2 = analisa_exp amb exp
                                        in
161                                 Move (Mem nome, Some _arg2)
162                                 )
163         | _ -> failwith "erro dec")
164     | CmdDeclConstrut (nome,ttipo, exp) ->
165     (match nome with
166         | ExpVar nome ->
167                 MoveConst (Mem nome, exp)
168         | _ -> failwith "Erro na Declaracao com Contrutor");
169
170     | CmdAtribNextFloat (exp, v)->
171         (match v with
172             | ExpVar v ->
173                     let _arg = analisa_exp amb exp in
174                     Printf.printf "\nComando Atribuicao
                            NextFloat\n";
175                     Move (Mem v, Some _arg)
176
177             | _ -> failwith "Erro na atribuicao NextFloat");
178
179     | CmdAtribNextInt (exp, v)->
180         (match v with
181             | ExpVar v ->
182                     let _arg = analisa_exp amb exp in
183                     Printf.printf "\nComando Atribuicao
                            NextInt\n";
184                     Move (Mem v, Some _arg)
185
186             | _ -> failwith "Erro na atribuicao NextFloat");
187     | CmdAtribNextLine (exp, v)->
188         (match v with
189             | ExpVar v ->
190                     let _arg = analisa_exp amb exp in
191                     Printf.printf "\nComando Atribuicao
                            NextLine\n";
192                     Move (Mem v, Some _arg)
```

```
193
194             | _ -> failwith "Erro na atribuicao NextFloat");
195
196     | CmdIf (exp,cmds1,cmds2) ->
197                 let arg = analisa_exp amb exp in
198                 Cjump (arg, Label newlabel, Label newlabel)
199     | CmdInc (v) ->
200         (match v with
201         | ExpVar v -> Move (Mem v, Some (BinOp (Soma, Mem v,
                ConstI 1)))
202         | _ -> failwith "erro incrementa")
203     | _ -> failwith "Este comando nao foi implementado")
204
205
206
207 and newlabel =
208     "label"
209
210 let rec repi amb arv =
211     match arv with
212     Programa(imports,DeclClass(DeclFun(arv)))-> analisa_cmds
            amb arv;
```

3.8.1 Árvore da Representação Intermediária

```
1  open ArvSint;;
2
3  type expressao = ConstI of int
4                     | ConstV of string
5                     | ConstF of float
6                         | ConstC of char
7                     | ConstS of string
8                | ConstB of bool
9                             | Mem of string
10               | BinOp  of operador * expressao * expressao
11                       | UnOp of operador * expressao
12
13 and operador = Soma  | Sub   | Mul     | Div
14                      | Maior | Menor | MaiorI  | MenorI
15                      | IgualI  | Dif   | Or      | And | Not
16
17 and tipori = Int | Float | Char | Bool | Scanner
18
19
20 and comandos = comando list
21 and comando = Move of expressao * expressao option
22               | MoveConst of expressao * tipo
23               | Temp of expressao * tipori
24               | Label of string
25               | Cjump of expressao * comando * comando
```

Chapter 4

Códigos Resultantes Gerados

Segue os códigos na linguagem de máquina cil, gerados pelo nosso compilador dos exemplos anteriores.

4.1 1 - Módulo Mínimo

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init()
.entrypoint
.maxstack 0
ret
}
```

4.2 2 - Declaração de uma váriavel

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32)
.entrypoint
.maxstack 0
ret
```

```
}
```

4.3 3 - Atribuição de um inteiro a uma váriavel

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32)
.entrypoint
.maxstack 2
ldc.i4 1
stloc.0
ret
}
```

4.4 4 - Atribuição de uma soma de inteiros a uma váriavel

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32)
.entrypoint
.maxstack 4
ldc.i4 1
ldc.i4 2
 add stloc.0
ret
}
```

4.5 5 - Inclusão do comando de impressão

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32)
.entrypoint
```

```
.maxstack 2
ldc.i4 2
stloc.0
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
ret
}
```

4.6 6 - Atribuição de uma subtração de interios a uma váriavel

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32)
.entrypoint
.maxstack 4
ldc.i4 1
ldc.i4 2
 sub stloc.0
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
ret
}
```

4.7 7 - Inclusão do comando condicional

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32)
.entrypoint
.maxstack 4
ldc.i4 1
stloc.0
ldloc.0
ldc.i4 1
beq Label1
br Label2
Label1:
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
```

```
br Label2
Label2:
ret
}
```

4.8 8 - Inclusão do comando condicional com parte senão

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32)
.entrypoint
.maxstack 4
ldc.i4 1
stloc.0
ldloc.0
ldc.i4 1
beq Label1
ldc.i4 0
call void [mscorlib]System.Console::WriteLine(int32)
br Label2
Label1:
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
br Label2
Label2:
ret
}
```

4.9 9 - Atribuição de duas operações aritméticas sobre inteiro a uma váriavel

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32)
.entrypoint
.maxstack 6
ldc.i4 1
ldc.i4 1
ldc.i4 2
```

```
 div  add stloc.0
ldloc.0
ldc.i4 1
beq Label1
ldc.i4 0
call void [mscorlib]System.Console::WriteLine(int32)
br Label2
Label1:
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
br Label2
Label2:
ret
}
```

4.10 10 - Atribuição de duas váriaveis inteiras

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32,int32)
.entrypoint
.maxstack 6
ldc.i4 1
stloc.1
ldc.i4 2
stloc.0
ldloc.1
ldloc.0
beq Label1
ldc.i4 0
call void [mscorlib]System.Console::WriteLine(int32)
br Label2
Label1:
ldloc.1
call void [mscorlib]System.Console::WriteLine(int32)
br Label2
Label2:
ret
}
```

4.11 11 - Atribuição de um comando de repetição enquanto

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
```

```
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32,int32,int32)
.entrypoint
.maxstack 8
ldc.i4 1
stloc.0
ldc.i4 2
stloc.2
ldc.i4 5
stloc.1

Label2:
ldloc.1
ldloc.0
bgt Label1
br Label3
Label1:
ldloc.0
ldloc.2
 add stloc.0
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
ldstr " "
call void [mscorlib]System.Console::WriteLine(string)
br Label2
Label3:
ret
}
```

4.12 12 - Comando condicional aninhado em um comando de repetição

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32,int32,int32)
.entrypoint
.maxstack 8
ldc.i4 1
stloc.0
ldc.i4 2
stloc.2
ldc.i4 5
stloc.1
```

```
18
19 Label2:
20 ldloc.1
21 ldloc.0
22 bgt Label1
23 br Label3
24 Label1:
25 ldloc.0
26 ldloc.2
27 beq Label4
28 ldc.i4 0
29 call void [mscorlib]System.Console::WriteLine(int32)
30 br Label5
31 Label4:
32 ldloc.0
33 call void [mscorlib]System.Console::WriteLine(int32)
34 br Label5
35 Label5:
36 ldloc.1
37 ldc.i4 1
38  sub stloc.1
39 br Label2
40 Label3:
41 ret
42 }
```

4.13 13 - Converte graus Celsius para Fahrenheit

```
1  .assembly extern mscorlib {}
2  .assembly codigoGerado
3  {
4          .ver 1:0:1:0
5  }
6  .module codigoGerado.exe
7  .method static void main() cil managed
8  {
9  .locals init(float32,float32)
10 .entrypoint
11 .maxstack 8
12 ldc.r4 0.000000e+00
13 stloc.1
14 ldc.r4 0.000000e+00
15 stloc.0
16 ldstr "Tabela de conversao: Celsius -> Fahrenheit
17 "
18 call void [mscorlib]System.Console::WriteLine(string)
19 ldstr "Digite a temperatura em Celsius:
20 "
21 call void [mscorlib]System.Console::WriteLine(string)
22 call string [mscorlib]System.Console::ReadLine()
23 call float32 [mscorlib]System.Single::Parse(string)
24 stloc.1
```

```
ldc.r4 9.000000e+00
ldloc.1
 mul ldc.r4 1.600000e+02
 add ldc.r4 5.000000e+00
 div stloc.0
ldstr "A nova temperatura eh: "
call void [mscorlib]System.Console::WriteLine(string)
ldloc.0
call void [mscorlib]System.Console::WriteLine(float32)
ret
}
```

4.14 14 - Decide qual maior

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32,int32,int32)
.entrypoint
.maxstack 2
ldstr "Digite o primeiro numero: "
call void [mscorlib]System.Console::WriteLine(string)
call string [mscorlib]System.Console::ReadLine()
call int32 [mscorlib]System.Int32::Parse(string)
stloc.1
ldstr "Digite o segundo numero: "
call void [mscorlib]System.Console::WriteLine(string)
call string [mscorlib]System.Console::ReadLine()
call int32 [mscorlib]System.Int32::Parse(string)
stloc.0
ldloc.1
ldloc.0
bgt Label1
ldstr "O segundo numero "
call void [mscorlib]System.Console::WriteLine(string)
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
ldstr " eh maior que o primeiro "
call void [mscorlib]System.Console::WriteLine(string)
ldloc.1
call void [mscorlib]System.Console::WriteLine(int32)
br Label2
Label1:
ldstr "O primeiro numero "
call void [mscorlib]System.Console::WriteLine(string)
ldloc.1
call void [mscorlib]System.Console::WriteLine(int32)
ldstr " eh maior que o segundo "
call void [mscorlib]System.Console::WriteLine(string)
```

```
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
br Label2
Label2:
ret
}
```

4.15 15 - Lê o número e verifica se está entre 100 e 200

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32)
.entrypoint
.maxstack 2
ldstr "Digite o numero: "
call void [mscorlib]System.Console::WriteLine(string)
call string [mscorlib]System.Console::ReadLine()
call int32 [mscorlib]System.Int32::Parse(string)
stloc.0
ldloc.0
ldc.i4 100
bge Label1
ldstr "O numero nao esta no intervalo entre 100 e 200."
call void [mscorlib]System.Console::WriteLine(string)
br Label2
Label1:
ldloc.0
ldc.i4 200
ble Label3
ldstr "O numero nao esta no intervalo entre 100 e 200."
call void [mscorlib]System.Console::WriteLine(string)
br Label4
Label3:
ldstr "O numero esta no intervalo entre 100 e 200."
call void [mscorlib]System.Console::WriteLine(string)
br Label4
Label4:
br Label2
Label2:
ret
}
```

4.16 16 - Lê números e informa quais estão entre 10 e 150

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32,int32,int32)
.entrypoint
.maxstack 6
ldc.i4 0
stloc.2
ldc.i4 0
stloc.0
ldc.i4 0
stloc.1

Label2:
ldloc.1
ldc.i4 5
blt Label1
br Label4

Label3:
ldloc.1
ldc.i4 1
add
stloc.1
br Label2

Label1:
ldstr "Digite um numero: "
call void [mscorlib]System.Console::WriteLine(string)
call string [mscorlib]System.Console::ReadLine()
call int32 [mscorlib]System.Int32::Parse(string)
stloc.2
ldloc.2
ldc.i4 10
bge Label5
br Label6
Label5:
ldloc.2
ldc.i4 150
ble Label7
br Label8
Label7:
ldloc.0
ldc.i4 1
 add stloc.0
br Label8
Label8:
br Label6
Label6:
```

```
br Label3

Label4:ldstr "Ao total foram digitados "
call void [mscorlib]System.Console::WriteLine(string)
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
ldstr " numeros no intervalo de 10 a 150"
call void [mscorlib]System.Console::WriteLine(string)
ret
}
```

4.17 17 - Lê strings e caracteres

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32,int32,int32,char)
.entrypoint
.maxstack 8
ldc.i4 0
stloc.2
ldc.i4 0
stloc.1
ldc.i4 0
stloc.0
ldc.i4 0
stloc.2

Label2:
ldloc.2
ldc.i4 5
blt Label1
br Label4

Label3:
ldloc.2
ldc.i4 1
add
stloc.2
br Label2

Label1:
ldstr "H - Homem e M - Mulher "
call void [mscorlib]System.Console::WriteLine(string)
call string [mscorlib]System.Console::ReadLine()
call char [mscorlib]System.Char::Parse(string)
stloc.3
ldloc.3
ldc.i4 104
```

```
beq Label5
ldloc.0
ldc.i4 1
 add stloc.0
br Label6
Label5:
ldloc.1
ldc.i4 1
 add stloc.1
br Label6
Label6:
br Label3

Label4:ldstr "Foram inseridos "
call void [mscorlib]System.Console::WriteLine(string)
ldloc.1
call void [mscorlib]System.Console::WriteLine(int32)
ldstr " Homens
"
call void [mscorlib]System.Console::WriteLine(string)
ldstr "Foram inseridos "
call void [mscorlib]System.Console::WriteLine(string)
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
ldstr " Mulheres"
call void [mscorlib]System.Console::WriteLine(string)
ret
}
```

4.18 19 - Decide se os números são positivos, negativos ou zero

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(,char,int32,int32)
.entrypoint
.maxstack 8
ldc.i4 0
stloc.2
ldc.i4 0
stloc.0
ldc.i4 97
stloc.1
ldc.i4 1
stloc.0

Label2:
```

```
ldloc.0
ldc.i4 1
beq Label1
br Label3
Label1:
ldstr "Escreva um numero:
"
call void [mscorlib]System.Console::WriteLine(string)
call string [mscorlib]System.Console::ReadLine()
call int32 [mscorlib]System.Int32::Parse(string)
stloc.2
ldloc.2
ldc.i4 0
bgt Label6
ldloc.2
ldc.i4 0
beq Label10
br Label11
Label10:
ldstr "Numero igual a 0
"
call void [mscorlib]System.Console::WriteLine(string)
br Label11
Label11:
ldloc.2
ldc.i4 0
blt Label8
br Label9
Label8:
ldstr "Numero Negativo
"
call void [mscorlib]System.Console::WriteLine(string)
br Label9
Label9:
br Label7
Label6:
ldstr "Numero Positivo
"
call void [mscorlib]System.Console::WriteLine(string)
br Label7
Label7:
ldstr "Deseja finalizar? (S/N)
"
call void [mscorlib]System.Console::WriteLine(string)
call string [mscorlib]System.Console::ReadLine()
call char [mscorlib]System.Char::Parse(string)
stloc.1
ldloc.1
ldc.i4 115
beq Label4
br Label5
Label4:
ldc.i4 0
stloc.0
```

```
br Label5
Label5:
br Label2
Label3:
ret
}
```

4.19 20 - Decide se um número é maior que 10

```
.assembly extern mscorlib {}
.assembly codigoGerado
{
        .ver 1:0:1:0
}
.module codigoGerado.exe
.method static void main() cil managed
{
.locals init(int32)
.entrypoint
.maxstack 4
ldc.i4 1
stloc.0

Label2:
ldloc.0
ldc.i4 0
bne.un Label1
br Label3
Label1:
ldstr "Escreva um numero: "
call void [mscorlib]System.Console::WriteLine(string)
call string [mscorlib]System.Console::ReadLine()
call int32 [mscorlib]System.Int32::Parse(string)
stloc.0
ldloc.0
ldc.i4 10
bgt Label4
ldloc.0
ldc.i4 10
blt Label6
ldstr "O numero "
call void [mscorlib]System.Console::WriteLine(string)
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
ldstr " e igual a 10
"
call void [mscorlib]System.Console::WriteLine(string)
br Label7
Label6:
ldstr "O numero "
call void [mscorlib]System.Console::WriteLine(string)
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
ldstr " e menor que 10
```

```
"
call void [mscorlib]System.Console::WriteLine(string)
br Label7
Label7:
br Label5
Label4:
ldstr "O numero "
call void [mscorlib]System.Console::WriteLine(string)
ldloc.0
call void [mscorlib]System.Console::WriteLine(int32)
ldstr " e maior que 10
"
call void [mscorlib]System.Console::WriteLine(string)
br Label5
Label5:
br Label2
Label3:
ret
}
```

Chapter 5

Considerações Finais

Os algoritimos 18 e 21 não foram implemantados pois o comando switch e funções não foram implementados.