UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
Construção de Compiladores
Av. João Naves de Ávila 2121, Campus Santa Mônica

Construção de Compiladores JavaScript para CLR

Nome: Lara Carolina Luciana e Oliveira
Matrícula: 11511BCC038
Email: lara.carolinnaa@gmail.com

Sumário

UFU, Universidade Federal de Uberlândia, Minas Gerais, Brasil

1 Introdução

Este relatorio tem como propósito detalhar o processo de instalação e uso das ferramentas relacionadas à plataforma selecionada para o projeto (CLR), apresentar a linguagem Assembly associada a ela (CIL), e fornecer uma possível implementação para cada uma das etapas de construção de um compilador para a linguagem JavaScript utilizando a linguagem Ocaml.

Para contextualização do assunto, inicialmente será apresentada uma introdução sobre a CLR, a linguagem Assembly utilizada e o Javascript. Em seguida, serão apresentadas as ferramentas instaladas. Serão mostrados pseudo códigos convertidos em Javascript e em C#, os quais serão compilados utilizando uma plataforma que segue o modelo de compilação CLR para a geração dos respectivos códigos em Assembly CIL.

Em seguida, será apresentada a implementação de cada etapa referente à construção do compilador: Análise Lexica, Análise Sintática, Análise Semântica, Intérprete e Gerador de Código.

2 CLR

A CLR (Common Language Runtime) é o componente da maquina virtual da plataforma .Net responsável por executar as aplicações desenvolvidas em .Net. Além da compilação, a CLR também oferece outros serviços como segurança e tratamento de exceções. A compilação é feita em tempo de execução.

O processo de compilação executado pela CLR pode ser descrito da seguinte maneira: ao passar pelo compilador, o código fonte dá origem ao código gerenciado, o qual é descrito em uma linguagem de programação de baixo nível (Assembly) chamada CIL (Common Intermediate Language). A CLR então, através da compilação just-in-time (compilação em tempo de execução), converte o código gerenciado para linguagem de máquina.

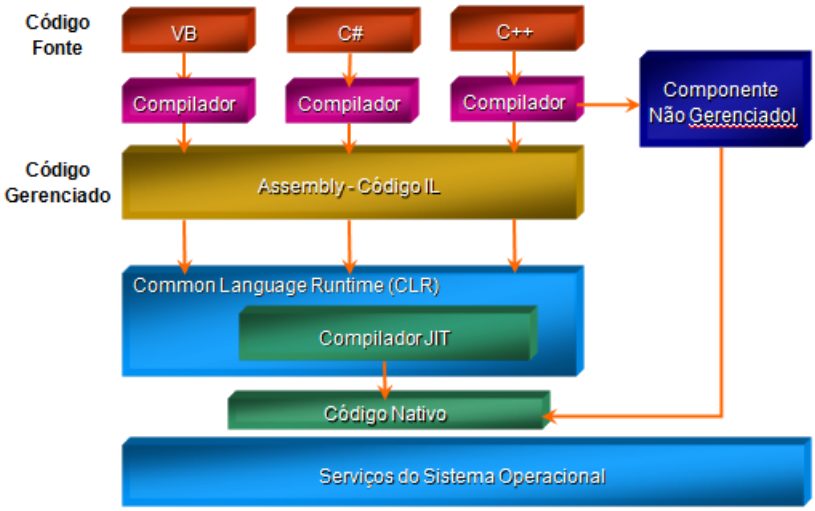

Figura 1: Modelo de Compilação CLR

A CLR suporta algumas linguagens como C#, C++, Visual Basic etc, porém não compila códigos escritos na linguagem selecionada, JavaScript. Desta forma, além dos códigos em Javascript, também serão apresentados no relatório códigos em C#, os quais de fato serão compilados pela CLR para a obtenção do código em linguagem Assembly correspondente.

3 Linguagem Assembly (CIL)

A CIL (Common Intermediate Language) é a linguagem Assembly associada à CLR. As instruções definidas pela CIL são convertidas para linguagem de máquina pelo compilador JIT.

A CIL é uma linguagem orientada a objetos, ou seja, envolve conceitos como criação de objetos, classes e chamada de métodos, e é baseada em pilha (dados são retirados de uma pilha e não de registradores).

A seguir são apresentadas as instruções do conjunto de instruções da CIL.

- add: retira os dois elementos do topo da pilha e coloca o resultado no topo.
- add.ovf: retira os dois elementos do topo da pilha e coloca o resultado no topo, verificando overflow.

- and: retira os dois elementos do topo da pilha, faz a operação and bit a bit e coloca o resultado no topo da pilha
- arglist: retorna o identificador da lista de argumentos para o método atual
- beq: pula para o endereço se os elementos topo da pilha são iguais
- bge: pula para o endereço se o topo da pilha é menor que o seu antecessor
- bge.un. : igual ao bge, mas a comparação é feita sem sinal.
- bgt: pular para o endereço se o segundo elemento da pilha é maior que o topo.
- ble: pula para o endereço se o topo é menor ou igual ao seu antecessor.
- ble.un: igual ao ble, mas a comparação é feita sem sinal.
- blt : pula pra o endereço se o o topo é menor que seu antecessor.
- blt.un: igual ao blt, mas a comparação é feita sem sinal.
- bne: pula para o endereço se o topo não for igual ou não ordenada.
- bne.un: igual ao bne, mas a comparação é feita sem sinal.
- br: salto incondicional
- break: instrução de breakpoint
- brfalse: pula para o endereco se falso, nulo, ou zero.
- brtrue: pula para o endereço se verdadeiro, diferente de zero, ou não nulo.
- call: chamada de método.
- calli: chamada de método indireto.
- ceq: compara se igual
- cgt: compara se maior que.

- cgt.un: igual ao cgt, mas a comparação é feita sem sinal.
- clt: compara se menor que
- clt.un igual ao clt, mas a comparação é feita sem sinal.
- conv : conversão de dados.
- conv.ovf: igual ao conv, mas com sinalização de overflow.
- conv.ovf.un: igual ao conv.of, mas a comparação é feita sem sinal.
- cpbllc: copia dados da memória para a memória.
- div: divisão.
- div.un: igual ao div, mas a comparação é feita sem sinal.
- dup: duplica o valor da pilha.
- jmp: pula para um método.
- jmpi: pular para um ponteiro de método.
- ldarg: carrega um argumento na pilha.
- ldarga: carrega um argumento a partir de um endereço.
- ldc: carrega uma constante numérica.
- Idelem: carrega um elemento do vetor no index para topo da pilha.
- Idelema: carrega o endereco do vetor na posicao index na pilha.
- Idstr: poe no topo da pilha a string.
- mul: multiplica os dois valores de cima da pilha e empilha o resultado.
- mul.ovf: igual ao mul, mas é feita sinalização de overflow.
- neg: altera o sinal do elemento do topo da pilha.
- Newarr: cria um array com o tipo definido onde seu tamanho está no topo da pilha.

- not: inverte os bits do elemento do topo da pilha.
- or: faz um ou lógico bit a bit dos elementos do topo, empilhando o resultado.
- pop: desempilha a pilha.
- rem: computa o resto da divisão do elemento abaixo do topo, pelo topo, e empilha o resultado.
- rem.un: igual ao rem, mas a comparação é feita sem sinal.
- shl: deslocamento de bits a esquerda.
- shr: deslocamento de bits a direita.
- starg: retira o elemento do topo da pilha e coloca em um argumento.
- sizeof: carrega o tamanho em bits do topo na pilha.
- sub: subtrai o segundo valor do primeiro e empilha o resultado.
- sub.ovf: igual ao sub, mas com a sinalização de overflow.
- xor: executaa a operação XOR(bit a bit) entre dois elementos da pilha colocando seu resultado na mesma.

4 Códigos em Assembly CIL (comentados)

4.1 Módulo mínimo que carateriza um programa

Listing 1: Hello World em CIL

```
1  .assembly extern mscorlib
2 {
3 }
4
5 .assembly moduloMinimo
6 {
7 }
8
9 .module moduloMinimo.exe
10
```

```
11 .class private auto ansi beforefieldinit moduloMinimo.Program
       extends [mscorlib]System.Object
12 {
13  .method private hidebysig static void  Main(string[] args)
       cil managed
14   {
15     .entrypoint
16     .maxstack  8
17      ret
18   } // end of method Program::Main
19
20
21   .method public hidebysig specialname rtspecialname
22           instance void  .ctor() cil managed
23   {
24     .maxstack  8
25      ldarg.0
26      call       instance void [mscorlib]System.Object::.ctor
           ()
27     ret
28   } // end of method Program::.ctor
29
30 }
```

Os nomes que se iniciam com o prefixo ".", como .assembly, .method, .class, são chamados de Diretrizes CIL. A primeira diretriz no código, ".assembly extern mscorlib", é usada para informar ao assembler que serão utilizados métodos e objetos de um assembly externo, no caso o "mscorlib".

As próximas duas diretrizes ".assembly moduloMinimo" e ".module moduloMinimo.exe" informam ao programa que estamos criando um código assembly e nomeiam o módulo como moduloMinimo. Sem essas declarações, o assembler não consegue executar o código assembly.

A próxima linha ".class private auto ansi beforefieldinit moduloMinimo.Program extends [mscorlib]System.Object" declara a classe do programa e informa que ela "herda" da classe Object presente em mscorlib.

Após criar a classe é definido o método principal, cuja declaração informa que ele será gerenciado pelo assembly CIL: ".method private hidebysig static void Main(string[] args)cil managed.". O atributo "hidebysig" significa que o membro na classe base com o mesmo nome e assinatura está oculto da classe derivada. É dentro deste método principal que o programa começa de fato. A diretriz .entrypoint é quem marca o início do programa.

Após o .entrypoint deve-se informar o número de itens que estão na pilha

no momento. Para isso, utiliza-se a diretriz .maxstack. Caso esse valor não seja informado, é utilizado um valor default igual a 8. Finalmente as instruções CIL podem ser declaradas.Como o objetivo deste programa é apenas apresentar o módulo mínimo de um código CIL, há apenas a instrução "ret" ou "return" que finaliza o método e o programa.

A diretriz .ctor representa o construtor em nível de instância. Para representar o construtor estático utiliza-se ".cctor" (construtor de classe). O .ctor está sempre qualificado com o atributo specialname e rtspecialname. O specialname é usado para indicar que este token pode ser tratado de forma diferente por diferentes ferramentas, por exemplo, em linguagem c#, o construtor não tem o tipo de retorno, mas no CIL ele tem o tipo de retorno void. A instrução ldarg é usada para carregar o argumento passado no método para a pilha. "ldarg.0" aqui significa carregar o primeiro argumento na pilha.

4.2 Hello World

O código a seguir refere-se a um Hello World escrito em CIL:

Listing 2: Hello World em CIL

```
.assembly extern mscorlib {}
.assembly Hello {}
.module Hello.exe

.class Hello.Program
extends [mscorlib]System.Object
{
 .method static void Main(string[] args) cil managed
 {
  .entrypoint
 .maxstack 8
  ldstr "Hello World"
  call void [mscorlib]System.Console:: WriteLine(string)
  ret
 }

 .method public hidebysig specialname rtspecialname
          instance void  .ctor() cil managed
  {
    .maxstack  8
    ldarg.0
    call       instance void [mscorlib]System.Object::.ctor()
    ret
  } // end of method Program::.ctor
```

```
25 }
```

O código acima possui todas as características estruturais apresentadas anteriormente. A diferença é que este código apresenta instruções declaradas após o .entrypoint. A instrução ldstr carrega a string “Hello World” na pilha e a instrução call chama o método WriteLine da classe Console. O método “pega” seus parâmetros da pilha. Por fim, o “ret” ou “return” finaliza o método e o programa.

4.3 Execução baseada em pilha

Antes do início de cada método, a pilha está vazia e durante a execução do método, as instruções CIL adicionam e/ou removem os itens da pilha, cujo resultado final é uma pilha vazia no final da execução desse método.

As instruções que copiam valores da memória para a pilha são chamados de Load e as instruções que copiam os valores da pilha para a memória são chamados de Store. Todos as instruções iniciadas com ld são usadas para carregar o item na pilha e as que começam com st são usadas para armazenar o item na memória.

Para entender melhor como funciona a pilha durante a execução de um método, considere o seguinte código escrito em C#:

Listing 3: Código em C#

```
1 static void add()
2       {
3           int value1 = 10;
4           int value2 = 20;
5           int value3 = value1 + value2;
6       }
```

O código correspondente em assembly CIL é mostrado a seguir. Os comentários adicionados a este código tem como finalidade mostrar a operação da pilha durante a execução.

Listing 4: Código em CIL

```
1 .method private static void  'add'() cil managed
2 {
3    maxstack  2
4   .locals init ([0] int32 value1,
5            [1] int32 value2,
```

```
6             [2] int32 value3) // 3 variáveis locais do tipo
                 int32 são declaradas
7
8 nop  // sem operação ( sem push ou pop na pilha)
9
10  ldc.i4.s   10  // carrega o valor 10 de tipo int32 no topo
      da pilha. Itens na pilha = 1
11  stloc.0 // pega o item do topo da pilha (10) e armazena na
      primeira variável local. Itens na pilha = 0.
12  ldc.i4.s   20 //  carrega o valor 20 de tipo int32 no topo
      da pilha. Itens na pilha = 1
13  stloc.1 0 // pega o item do topo da pilha (20) e armazena na
       primeira variável local. Itens na pilha = 0.
14 ldloc.0 // carrega o valor da primeira variável local na
     pilha. Itens na pilha = 1
15 ldloc.1 // carrega o valor da primeira variável local na
     pilha. Itens na pilha = 2
16 add // (pega os primeiros dois valores da pilha e envia o
     resultado da soma para a pilha. Itens na pilha = 2-2+1 = 1
17 stloc.2 // pega o valor do topo da pilha e armazena na
     terceira variável local. Itens na pilha = 0
18 ret // fim do método.
19 }
```

Observações:

- No começo do método do programa apresentado acima, temos a diretriz ".locals init". Ela informa que serão declaradas variáveis locais a serem utilizadas no programa.
- Note que a declaração do método é feita de forma diferente do programa Hello World apresentado anteriormente: ".method private hidebysig static void 'add'() cil managed". Isso porque este não é o método principal da classe. A declaração deste método indica que ele é do tipo "private" e "static", não retorna valores (void) e que o seu nome é "add". Como já foi dito, "cil managed" indica que o método será gerenciado pelo assembly CIL. Essa declaração é necessária para qualquer método escrito em código CIL.
- A declaração .entrypoint não está presente neste método pois essa diretriz faz parte apenas do método principal da classe.

```
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 4:0:0:0
}

.assembly HelloWorld
{
 ----lines omitted
}

.module HelloWorld.exe

.class private auto ansi beforefieldinit HelloWorld.Program
       extends [mscorlib]System.Object
{
 .method private hidebysig static void  Main(string[] args) cil managed
  {
    .entrypoint
    // Code size       13 (0xd)
    .maxstack  8
    IL_0000:  nop
    IL_0001:  ldstr      "Hello World"
    IL_0006:  call       void [mscorlib]System.Console::WriteLine(string)
    IL_000b:  nop
    IL_000c:  ret
  } // end of method Program::Main


  .method public hidebysig specialname rtspecialname
          instance void  .ctor() cil managed
  {
    .maxstack  8
    IL_0000:  ldarg.0
    IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
    IL_0006:  ret
  } // end of method Program::.ctor

} // end of class HelloWorld.Program
```

Figura 2: Código Hello World em CIL

Os nomes que se iniciam com o prefixo ".", como .assembly, .method, .class, são chamados de Diretrizes CIL. Os nomes usados juntamente às diretrizes, como extends, implements, public, são chamados de Atributos CIL.

Antes de cada método a pilha está vazia e, durante a execução de cada método, as instruções da CIL adicionam e/ou removem os itens da pilha. No final do método, a pilha volta a ser vazia.

No começo da função, deve-se informar o número de itens que estão na pilha no momento. Para isso, utiliza-se a diretriz .maxstack. Caso esse valor não seja informado, é utilizado um valor default igual a 8.

4.4 Condições e Iterações

Para entender como funcionam condições e iterações em CIL será apresentado um código em C# e seu correspondente em CIL. Cada passo executado no código em assembly apresentado será explicado posteriormente.

Listing 5: Código em C#

```
static void IterationExample( )
{
    int i = 0;
    while (i < 5)
    {
        i++;
    }
}
```

Listing 6: Código em CIL

```
.method private static void  IterationExample() cil managed
{
  .maxstack  2
  .locals init ([0] int32 i,
           [1] bool CS$4$0000) // PASSO 1
 IL_0000: nop                 //PASSO 2
 IL_0001: ldc.i4.0            //PASSO 3
 IL_0002: stloc.0             //PASSO 4
 IL_0003: br.s       IL_000b //PASSO 5
 IL_0005: nop                 //PASSO 12 //PASSO 24 e assim
    por diante...
 IL_0006: ldloc.0             //PASSO 13
 IL_0007: ldc.i4.1            //PASSO 14
 IL_0008: add                 //PASSO 15
 IL_0009: stloc.0             //PASSO 16
 IL_000a:  nop                 //PASSO 17
 IL_000b: ldloc.0             //PASSO 6 //PASSO 18
 IL_000c: ldc.i4.5            //PASSO 7 //PASSO 19
 IL_000d: clt                 //PASSO 8 //PASSO 20
 IL_000f: stloc.1             //PASSO 9 //PASSO 21
 IL_0010: ldloc.1             //PASSO 10 //PASSO 22
 IL_0011: brtrue.s   IL_0005 //PASSO 11 //PASSO 23
 IL_0013: ret
}
```

PASSO 1: declara duas variáveis locais nos índices 0 e 1.

PASSO 2: sem operação na pilha.

PASSO 3: carrega o valor 0 no topo da pilha (itens na pilha = 1).

PASSO 4: pega o valor do topo da pilha e armazena na variável local 0 (itens na pilha = 0, valor = 0)

PASSO 5: ir para "IL000b".

PASSO 6: carrega a variavél 0 na pilha (itens na pilha = 1).

PASSO 7: carrega o valor 5 na pilha (itens na pilha = 2).

PASSO 8: faz a comparaçao "menor que" com dois itens após removê-los da pilha. Carrega o valor 1 na pilha já que clt retorna 1 (verdadeiro) para a comparação (0¡1) (itens na pilha = 1).

PASSO 9: pega o valor da pilha e armazena na variável local 1 (itens na pilha = 0).

PASSO 10: carrega o valor 1 na pilha (itens na pilha = 1, valor = 1).

PASSO 11: brtrue checa se o valor é maior que 0 e depois pula para IL0005. Neste caso, como o valor é 1 a execução vai para IL0005. Se o valor é 0, a execução para a linha que contém "ret" e sai do método.

PASSO 12: sem operação na pilha.

PASSO 13: carrega o valor da variável local 0 na pilha.

PASSO 14: carrega o valor 1 na pilha (itens na pilha = 2, valor = 1)

PASSO 15: pega dois itens na pilha e retorna sua soma para a pilha (itens na pilha = 1, valor = 1).

PASSO 16: armazena a soma na variável local 0 (itens na pilha = 0)

A execução continua até que a condição no PASSO 11 retorne 0.

Observação: nomes como IL0001 e IL0002 que aparecem no código são chamados rótulos CIL. Eles são opcionais, ou seja, podem ser incluídos ou não no código ou podem ser substituídos por qualquer texto. No código acima eles são uteis para direcionar a execução para outra linha do código.

4.5 Fatorial

Aqui será apresentado e explicado um código em assembly CIL um pouco mais complexo. Seja o código em C# que calcula o fatorial de um número:

Listing 7: Calcula o fatorial de um número(C#)

```
1 using System;
2 class calculaFatorial {
3 public static int fatorial (int n){
4 if ( n <= 0)
5 return 1;
```

```
6 else
7 return n * fatorial(n-1);
8
9 }
10
11 public static void Main(){
12 int numero, fat;
13
14 Console.Write("Digite um número: ");
15 numero = Convert.ToInt32(Console.ReadLine());
16 fat = fatorial(numero);
17 Console.Write("O fatorial de ");
18 Console.Write(numero);
19 Console.Write(" é ");
20 Console.WriteLine(fat);
21 }
22 }
```

O código correspondente em CIL é:

Listing 8: Calcula o fatorial de um número(CIL)

```
1 .assembly extern mscorlib
2 {
3
4 }
5 .assembly 'micro10'
6 {
7 }
8 .module calculaFatorial.exe
9
10
11   .class private auto ansi beforefieldinit micro10 extends [
       mscorlib]System.Object
12   {
13
14     // Metodo 1
15     .method public hidebysig specialname rtspecialname
          instance default void '.ctor' ()  cil managed
16     {
17             .maxstack 8
18   IL_0000:  ldarg.0
19   IL_0001:  call instance void object::'.ctor'()
20   IL_0006:  ret
21     } // end of method calculaFAtorial::.ctor
22
```

```
23     // Método 2
24     .method public static hidebysig  default int32 fatorial (
          int32 n)  cil managed
25     {
26   .maxstack 8
27   IL_0000:  ldarg.0  // carrega argumento na pilha
28   IL_0001:  ldc.i4.0  // carrega inteiro na pilha
29   IL_0002:  bgt IL_0009 // pula para IL0009 se o segundo
        elemento da pilha é maior que o topo
30
31   IL_0007:  ldc.i4.1 // carrega inteiro na pilha
32   IL_0008:  ret // fim do método
33   IL_0009:  ldarg.0 // carrega argumento na pilha
34   IL_000a:  ldarg.0 //  carrega argumento na pilha
35   IL_000b:  ldc.i4.1 // carrega inteiro na pilha
36   IL_000c:  sub // subtrai o segundo valor do primeiro e
        empilha o resultado. ( n-1)
37   IL_000d:  call int32 class calculaFAtorial::fatorial(int32)
         // invoca fatorial novamente (recursão)
38   IL_0012:  mul // multiplica os dois valores de cima da
        pilha e empilha o resultado (n*fatorial(n-1))
39   IL_0013:  ret // sai do metodo
40     } // end of method calculaFatorial::fatorial
41
42     // Metodo 3
43     .method public static hidebysig default void Main ()  cil
          managed
44     {
45   .entrypoint
46   .maxstack 1
47   .locals init (
48     int32 V_0,
49     int32 V_1) // declaração de variaveis
50   IL_0000:  ldstr "Digite um numero:" // carrega string na
        pilha
51   IL_0005:  call void class [mscorlib]System.Console::Write(
        string) // chama metodo para escrever a string
52   IL_000a:  call string class [mscorlib]System.Console::
        ReadLine() // chama metodo para ler string
53   IL_000f:  call int32 class [mscorlib]System.Convert::
        ToInt32(string) // converte a string digitada para
        inteiro
54   IL_0014:  stloc.0 // pega o valor do topo da pilha e
        armazena na primeira variável local
55   IL_0015:  ldloc.0 // carrega o valor da primeira variavel
```

```
      local na pilha
56  IL_0016:  call int32 class micro10::fatorial(int32) //
      invoca metodo fatorial
57  IL_001b:  stloc.1 // armazena o resultado na segunda
      variavel local
58  IL_001c:  ldstr "O fatorial de " // carrega string na pilha
59  IL_0021:  call void class [mscorlib]System.Console::Write(
      string) // chama metodo para escrever a string
60  IL_0026:  ldloc.0 // carrega o valor da primeira variavel
      local na pilha
61  IL_0027:  call void class [mscorlib]System.Console::Write(
      int32) // chama metodo para escrever o inteiro (numero
      digitado pelo usuario)
62  IL_002c:  ldstr "é" // carrega string na pilha
63  IL_0031:  call void class [mscorlib]System.Console::Write(
      string) //  chama metodo para escrever a string
64  IL_0036:  ldloc.1 //  carrega o valor da segunda variavel
      local na pilha
65  IL_0037:  call void class [mscorlib]System.Console::
      WriteLine(int32) // chama metodo para escrever inteiro (
      resultado)
66  IL_003c:  ret
67    } // end of method micro10::Main
68
69  } // end of class calculaFatorial
```

O código CIL acima para calcular o fatorial de um número possui 3 métodos: método ".ctor", o método "fatorial"(calcula o fatorial) e o método Main (recebe o número do teclado, invoca o método "fatorial"e mostra o resultado). A explicação das instruções utilizadas está em forma de comentários no código.

5 JavaScript

JavaScript é uma linguagem de programação interpretada baseada em scripts. Os scripts desenvolvidos em JavaScript são executados no interior de programas e são amplamente integrados em páginas web. Tais scripts comumente são incluídos em páginas HTML e interagem com o Modelo de Objeto de Documentos (DOM) da página.

Os códigos em JavaScript apresentados neste relatório irão incluir apenas os scripts, sem a sua integração com outras linguagens.

6 Preparação do Ambiente

Esta seção tem como objetivo apresentar as ferramentas necessárias para este projeto e para a compilação segundo o modelo de compilação CLR dos programas propostos e obtenção dos respectivos códigos em linguagem Assembly CIL.

O sistema operacional utilizado é o Ubuntu 14.04.

6.1 OCaml

Ocaml é a linguagem de programação escolhida para a implementação do compilador. Para instalar a Ocaml no Ubuntu utiliza-se o seguinte comando:

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

6.2 Mono

O Mono é uma plataforma desenvolvida para criar ferramentas compatíveis com a plataforma .Net. Ele implementa um compilador CLR, ou seja, contém um motor de compilação just-in-time, assim como a plataforma .Net.

A .Net é uma plataforma exclusiva da Microsoft e portanto não será possível utilizá-la neste projeto devido à utilização do Ubuntu. Por isso, o Mono será utilizado.

6.3 Instalação

Para instalar o Mono no Ubuntu basta inserir o seguinte comando no terminal:

```
> sudo apt-get install mono-complete
```

6.4 Compilação

Para compilar na plataforma Mono um dado programa “olamundo.cs” escrito em linguagem C#, utiliza-se o seguinte comando:

```
> mcs olamundo.cs
```

6.4.1 Execução

Para executar o programa:

```
> mono olamundo.exe
```

6.4.2 Geração do Assembly

Para obter o Assembly CIL associado ao executável deste programa, utiliza-se o comando:

```
> monodis olamundo.exe
```

6.4.3 Geração do Executável

Para obter o executável associado ao Assembly deste programa, utiliza-se o comando:

```
> ilasm olamundo.il
```

7 Códigos

Esta seção tem como finalidade apresentar os pseudo códigos propostos e convertê-los para linguagem Javascript e linguagem Assembly CIL. Para a compilação no Mono, os códigos em Javascript serão convertidos para C#.

7.1 Nano 01

Listing 9: Módulo mínimo que caracteriza um programa (JavaSript)

Listing 10: Módulo mínimo que caracteriza um programa (C#)

```
1 class nano1 {
2 public static void Main(){
3 }
4 }
```

Listing 11: Módulo mínimo que caracteriza um programa (CIL)

```
.assembly extern mscorlib
{
  .ver 4:0:0:0
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
}
.assembly 'nano01'
{
  .custom instance void class [mscorlib]System.Runtime.
     CompilerServices.RuntimeCompatibilityAttribute::'.ctor
     '() =  (
    01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
       T..WrapNonEx
    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
       ceptionThrows.

  .hash algorithm 0x00008004
  .ver  0:0:0:0
}
.module nano01.exe // GUID = {A5E14623-A7C9-4386-A2AB-
    A4439E4638AE}


  .class private auto ansi beforefieldinit nano01
    extends [mscorlib]System.Object
  {

    // method line 1
    .method public hidebysig specialname rtspecialname
           instance default void '.ctor' ()  cil managed
    {
        // Method begins at RVA 0x2050
  // Code size 7 (0x7)
  .maxstack 8
  IL_0000:  ldarg.0
  IL_0001:  call instance void object::'.ctor'()
  IL_0006:  ret
    } // end of method nano01::.ctor

    // method line 2
    .method public static hidebysig
           default void Main ()  cil managed
    {
        // Method begins at RVA 0x2058
  .entrypoint
```

```
40   // Code size 1 (0x1)
41   .maxstack 8
42   IL_0000:  ret
43     } // end of method nano01::Main
44
45   } // end of class nano01
```

7.2 Nano 02

Listing 12: Declaração de uma variável (JavaScript)

```
1 var n;
```

Listing 13: Declaração de uma variável (C#)

```
1 class nano02 {
2 public static void Main(){
3 int n;
4 }
5 }
```

Listing 14: Declaração de uma variável (CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano02'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9      01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
          T..WrapNonEx
10     63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
          ceptionThrows.
11
12  .hash algorithm 0x00008004
13  .ver  0:0:0:0
14 }
15 .module nano02.exe // GUID = {9801BF40-553F-412A-8756-1
     EDCE3D5872B}
16
17
```

```
18   .class private auto ansi beforefieldinit nano02
19     extends [mscorlib]System.Object
20   {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24            instance default void '.ctor' ()  cil managed
25     {
26         // Method begins at RVA 0x2050
27   // Code size 7 (0x7)
28   .maxstack 8
29   IL_0000:  ldarg.0
30   IL_0001:  call instance void object::'.ctor'()
31   IL_0006:  ret
32     } // end of method nano02::.ctor
33
34     // method line 2
35     .method public static hidebysig
36            default void Main ()  cil managed
37     {
38         // Method begins at RVA 0x2058
39   .entrypoint
40   // Code size 1 (0x1)
41   .maxstack 0
42   .locals init (
43     int32 V_0)
44   IL_0000:  ret
45     } // end of method nano02::Main
46
47   } // end of class nano02
```

7.3 Nano 03

Listing 15: Atribuição de um inteiro a uma variável (JavaScript)

```
1 var n;
2 n=1;
```

Listing 16: Atribuição de um inteiro a uma variável (C#

```
1 class nano03 {
2 public static void Main(){
3 int n;
4 n = 1;
5 }
```

```
6 }
```

Listing 17: Atribuição de um inteiro a uma variável (CIL

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano03'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
        T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
        ceptionThrows.
11
12  .hash algorithm 0x00008004
13  .ver  0:0:0:0
14 }
15 .module nano03.exe // GUID = {7D1CB53A-C8D0-4E7C-8710-6
    F8E3B9028CF}
16
17
18  .class private auto ansi beforefieldinit nano03
19    extends [mscorlib]System.Object
20  {
21
22    // method line 1
23    .method public hidebysig specialname rtspecialname
24           instance default void '.ctor' ()  cil managed
25    {
26        // Method begins at RVA 0x2050
27  // Code size 7 (0x7)
28  .maxstack 8
29  IL_0000:  ldarg.0
30  IL_0001:  call instance void object::'.ctor'()
31  IL_0006:  ret
32    } // end of method nano03::.ctor
33
34    // method line 2
35    .method public static hidebysig
36           default void Main ()  cil managed
```

```
37     {
38         // Method begins at RVA 0x2058
39   .entrypoint
40   // Code size 3 (0x3)
41   .maxstack 1
42   .locals init (
43     int32 V_0)
44   IL_0000:  ldc.i4.1
45   IL_0001:  stloc.0
46   IL_0002:  ret
47     } // end of method nano03::Main
48
49   } // end of class nano03
```

7.4 Nano 04

Listing 18: Atribuição de uma soma de inteiros a uma variável (JavaScript)

```
1 var n;
2 n=1 +2;
```

Listing 19: Atribuição de uma soma de inteiros a uma variável (C#)

```
1 class nano04 {
2 public static void Main(){
3 int n;
4 n = 1 + 2;
5 }
6 }
```

Listing 20: Atribuição de uma soma de inteiros a uma variável (CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano04'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
        T..WrapNonEx
```

```
10      63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
          ceptionThrows.
11
12    .hash algorithm 0x00008004
13    .ver  0:0:0:0
14  }
15  .module nano04.exe // GUID = {F2B9D2CB-2730-4B2A-9ED4-432
      ADD5A882E}
16
17
18    .class private auto ansi beforefieldinit nano04
19      extends [mscorlib]System.Object
20    {
21
22      // method line 1
23      .method public hidebysig specialname rtspecialname
24              instance default void '.ctor' ()  cil managed
25      {
26          // Method begins at RVA 0x2050
27    // Code size 7 (0x7)
28    .maxstack 8
29    IL_0000:  ldarg.0
30    IL_0001:  call instance void object::'.ctor'()
31    IL_0006:  ret
32      } // end of method nano04::.ctor
33
34      // method line 2
35      .method public static hidebysig
36              default void Main ()  cil managed
37      {
38          // Method begins at RVA 0x2058
39    .entrypoint
40    // Code size 3 (0x3)
41    .maxstack 1
42    .locals init (
43      int32 V_0)
44    IL_0000:  ldc.i4.3
45    IL_0001:  stloc.0
46    IL_0002:  ret
47      } // end of method nano04::Main
48
49    } // end of class nano04
```

7.5 Nano 05

Listing 21: Inclusão do comando de impressão (JavaScript)

```
var n;
n = 2;
document.write(n);
```

Listing 22: Inclusão do comando de impressão (C#)

```
using System;
class nano05 {
public static void Main(){
int n;
n = 2;
Console.Write(n);
}
}
```

Listing 23: Inclusão do comando de impressão (CIL)

```
.assembly 'nano05'
{
  .custom instance void class [mscorlib]System.Runtime.
     CompilerServices.RuntimeCompatibilityAttribute::'.ctor
     '() =  (
    01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
       T..WrapNonEx
    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
       ceptionThrows.

  .hash algorithm 0x00008004
  .ver  0:0:0:0
}
.module nano05.exe // GUID = {A4654B95-D993-490C-BB82-2705
    BA3ACD45}


  .class private auto ansi beforefieldinit nano05
    extends [mscorlib]System.Object
  {

    // method line 1
    .method public hidebysig specialname rtspecialname
           instance default void '.ctor' ()  cil managed
    {
        // Method begins at RVA 0x2050
  // Code size 7 (0x7)
```

```
23   .maxstack 8
24   IL_0000:  ldarg.0
25   IL_0001:  call instance void object::'.ctor'()
26   IL_0006:  ret
27     } // end of method nano05::.ctor
28
29     // method line 2
30     .method public static hidebysig
31            default void Main ()  cil managed
32     {
33         // Method begins at RVA 0x2058
34   .entrypoint
35   // Code size 9 (0x9)
36   .maxstack 1
37   .locals init (
38     int32 V_0)
39   IL_0000:  ldc.i4.2
40   IL_0001:  stloc.0
41   IL_0002:  ldloc.0
42   IL_0003:  call void class [mscorlib]System.Console::Write(
       int32)
43   IL_0008:  ret
44     } // end of method nano05::Main
45
46   } // end of class nano05
```

7.6 Nano 06

Listing 24: Atribuição de uma subtração de inteiros a uma variável (JavaScript)

```
1 var n;
2 n=1-2;
3 document.write(n);
```

Listing 25: Atribuição de uma subtração de inteiros a uma variável (C#)

```
1 using System;
2 class nano05 {
3 public static void Main(){
4 int n;
5 n = 1 - 2;
6 Console.Write(n);
7 }
8 }
```

Listing 26: Atribuição de uma subtração de inteiros a uma variável (CIL)

```
.assembly extern mscorlib
{
  .ver 4:0:0:0
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
}
.assembly 'nano06'
{
  .custom instance void class [mscorlib]System.Runtime.
     CompilerServices.RuntimeCompatibilityAttribute::'.ctor
     '() =  (
    01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
       T..WrapNonEx
    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
       ceptionThrows.

  .hash algorithm 0x00008004
  .ver  0:0:0:0
}
.module nano06.exe // GUID = {5201859C-0488-442D-B835-
    AB64377EC240}


  .class private auto ansi beforefieldinit nano06
    extends [mscorlib]System.Object
  {

    // method line 1
    .method public hidebysig specialname rtspecialname
           instance default void '.ctor' ()  cil managed
    {
        // Method begins at RVA 0x2050
  // Code size 7 (0x7)
  .maxstack 8
  IL_0000:  ldarg.0
  IL_0001:  call instance void object::'.ctor'()
  IL_0006:  ret
    } // end of method nano06::.ctor

    // method line 2
    .method public static hidebysig
           default void Main ()  cil managed
    {
        // Method begins at RVA 0x2058
  .entrypoint
```

```
40   // Code size 9 (0x9)
41   .maxstack 1
42   .locals init (
43     int32 V_0)
44   IL_0000:  ldc.i4.m1
45   IL_0001:  stloc.0
46   IL_0002:  ldloc.0
47   IL_0003:  call void class [mscorlib]System.Console::Write(
       int32)
48   IL_0008:  ret
49     } // end of method nano06::Main
50
51   } // end of class nano06
```

7.7 Nano 07

Listing 27: Inclusão do comando condicional (JavaScript)

```
1 var n;
2 n=1;
3 if(n==1)
4 document.write(n);
```

Listing 28: Inclusão de comando condicional (C#)

```
1 using System;
2 class nano07 {
3 public static void Main(){
4 int n;
5 n = 1;
6 if ( n == 1)
7 Console.Write(n);
8
9 }
10 }
```

Listing 29: Inclusão de comando condicional (CIL)

```
1 .assembly 'nano07'
2 {
3   .custom instance void class [mscorlib]System.Runtime.
      CompilerServices.RuntimeCompatibilityAttribute::'.ctor
      '() =  (
4     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
        T..WrapNonEx
```

```
5       63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
            ceptionThrows.
6
7     .hash algorithm 0x00008004
8     .ver  0:0:0:0
9   }
10  .module nano07.exe // GUID = {CCD3B1BD-FA30-4D9A-B653-
        E815EAE203E2}
11
12
13    .class private auto ansi beforefieldinit nano07
14      extends [mscorlib]System.Object
15    {
16
17      // method line 1
18      .method public hidebysig specialname rtspecialname
19              instance default void '.ctor' ()  cil managed
20      {
21          // Method begins at RVA 0x2050
22    // Code size 7 (0x7)
23    .maxstack 8
24    IL_0000:  ldarg.0
25    IL_0001:  call instance void object::'.ctor'()
26    IL_0006:  ret
27      } // end of method nano07::.ctor
28
29      // method line 2
30      .method public static hidebysig
31              default void Main ()  cil managed
32      {
33          // Method begins at RVA 0x2058
34    .entrypoint
35    // Code size 16 (0x10)
36    .maxstack 2
37    .locals init (
38      int32 V_0)
39    IL_0000:  ldc.i4.1
40    IL_0001:  stloc.0
41    IL_0002:  ldloc.0
42    IL_0003:  ldc.i4.1
43    IL_0004:  bne.un IL_000f
44
45    IL_0009:  ldloc.0
46    IL_000a:  call void class [mscorlib]System.Console::Write(
          int32)
```

```
47   IL_000f:  ret
48     } // end of method nano07::Main
49
50   } // end of class nano07
```

7.8 Nano 08

Listing 30: Inclusão do comando condicional com parte senão(JavaScript)

```
1 var n;
2 n=1;
3 if(n=1){
4 document.write(n);
5 }
6 else{
7 document.write(0);
8 }
```

Listing 31: Inclusão do comando condicional com parte senão(C#)

```
1 using System;
2 class nano08 {
3 public static void Main(){
4 int n;
5 n = 1;
6 if ( n == 1)
7 Console.Write(n);
8 else
9 Console.Write(0);
10
11 }
12 }
```

Listing 32: Inclusão do comando condicional com parte senão(CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano08'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
```

```
9       01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
          T..WrapNonEx
10      63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
          ceptionThrows.
11
12    .hash algorithm 0x00008004
13    .ver  0:0:0:0
14  }
15  .module nano08.exe // GUID = {00005516-A6A3-43D8-842A-1
      C4C9041633F}
16
17
18    .class private auto ansi beforefieldinit nano08
19      extends [mscorlib]System.Object
20    {
21
22      // method line 1
23      .method public hidebysig specialname rtspecialname
24              instance default void '.ctor' ()  cil managed
25      {
26          // Method begins at RVA 0x2050
27    // Code size 7 (0x7)
28    .maxstack 8
29    IL_0000:  ldarg.0
30    IL_0001:  call instance void object::'.ctor'()
31    IL_0006:  ret
32      } // end of method nano08::.ctor
33
34      // method line 2
35      .method public static hidebysig
36              default void Main ()  cil managed
37      {
38          // Method begins at RVA 0x2058
39    .entrypoint
40    // Code size 27 (0x1b)
41    .maxstack 2
42    .locals init (
43      int32 V_0)
44    IL_0000:  ldc.i4.1
45    IL_0001:  stloc.0
46    IL_0002:  ldloc.0
47    IL_0003:  ldc.i4.1
48    IL_0004:  bne.un IL_0014
49
50    IL_0009:  ldloc.0
```

```
51   IL_000a:  call void class [mscorlib]System.Console::Write(
        int32)
52   IL_000f:  br IL_001a
53
54   IL_0014:  ldc.i4.0
55   IL_0015:  call void class [mscorlib]System.Console::Write(
        int32)
56   IL_001a:  ret
57     } // end of method nano08::Main
58
59   } // end of class nano08
```

7.9 Nano 09

Listing 33: Atribuição de duas operações aritméticas sobre inteiros a uma variável(JavaScript)

```
1 var n;
2 n=1+1/2;
3 if(n=1){
4 document.write(n);
5 }
6 else{
7 document.write(0);
8 }
```

Listing 34: Atribuição de duas operações aritméticas sobre inteiros a uma variável(C#)

```
1 using System;
2 class nano09{
3 public static void Main(){
4 int n;
5 n = 1 + 1 / 2;
6 if ( n == 1)
7 Console.Write(n);
8 else
9 Console.Write(0);
10
11 }
12 }
```

Listing 35: Atribuição de duas operações aritméticas sobre inteiros a uma variável(CIL)

```
1  .assembly extern mscorlib
2  {
3    .ver 4:0:0:0
4    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5  }
6  .assembly 'nano09'
7  {
8    .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9      01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
         T..WrapNonEx
10     63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
         ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver  0:0:0:0
14 }
15 .module nano09.exe // GUID = {2A0BCBD4-349E-49BB-9AA6-
     FA7742A27A80}
16
17
18   .class private auto ansi beforefieldinit nano09
19     extends [mscorlib]System.Object
20   {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24             instance default void '.ctor' ()  cil managed
25     {
26         // Method begins at RVA 0x2050
27   // Code size 7 (0x7)
28   .maxstack 8
29   IL_0000:  ldarg.0
30   IL_0001:  call instance void object::'.ctor'()
31   IL_0006:  ret
32     } // end of method nano09::.ctor
33
34     // method line 2
35     .method public static hidebysig
36             default void Main ()  cil managed
37     {
38         // Method begins at RVA 0x2058
39   .entrypoint
40   // Code size 27 (0x1b)
```

```
41   .maxstack 2
42   .locals init (
43     int32 V_0)
44   IL_0000:  ldc.i4.1
45   IL_0001:  stloc.0
46   IL_0002:  ldloc.0
47   IL_0003:  ldc.i4.1
48   IL_0004:  bne.un IL_0014
49
50   IL_0009:  ldloc.0
51   IL_000a:  call void class [mscorlib]System.Console::Write(
       int32)
52   IL_000f:  br IL_001a
53
54   IL_0014:  ldc.i4.0
55   IL_0015:  call void class [mscorlib]System.Console::Write(
       int32)
56   IL_001a:  ret
57     } // end of method nano09::Main
58
59   } // end of class nano09
```

7.10 Nano 10

Listing 36: Atribuição de variáveis inteiras(JavaScript)

```
1 var n,m;
2 n=1;
3 m=2;
4 if(n==m){
5 document.write(n);
6 }
7 else{
8 document.write(0);
9 }
```

Listing 37: Atribuição de variáveis inteiras(C#)

```
1 using System;
2 class nano10{
3 public static void Main(){
4 int n, m;
5 n = 1;
6 m = 2;
7
```

```
8 if ( n == m)
9 Console.Write(n);
10 else
11 Console.Write(0);
12
13 }
14 }
```

Listing 38: Atribuição de variáveis inteiras(CIL)

```
1 .assembly 'nano10'
2 {
3   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
4     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
        T..WrapNonEx
5     63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
        ceptionThrows.
6
7   .hash algorithm 0x00008004
8   .ver  0:0:0:0
9 }
10 .module nano10.exe // GUID = {E8BF1B6B-B890-43AC-A3FF-65
     AF216D4F02}
11
12
13   .class private auto ansi beforefieldinit nano10
14     extends [mscorlib]System.Object
15   {
16
17     // method line 1
18     .method public hidebysig specialname rtspecialname
19            instance default void '.ctor' ()  cil managed
20     {
21         // Method begins at RVA 0x2050
22   // Code size 7 (0x7)
23   .maxstack 8
24   IL_0000:  ldarg.0
25   IL_0001:  call instance void object::'.ctor'()
26   IL_0006:  ret
27     } // end of method nano10::.ctor
28
29     // method line 2
30     .method public static hidebysig
```

```
31            default void Main ()  cil managed
32    {
33        // Method begins at RVA 0x2058
34   .entrypoint
35   // Code size 29 (0x1d)
36   .maxstack 2
37   .locals init (
38     int32 V_0,
39     int32 V_1)
40   IL_0000:  ldc.i4.1
41   IL_0001:  stloc.0
42   IL_0002:  ldc.i4.2
43   IL_0003:  stloc.1
44   IL_0004:  ldloc.0
45   IL_0005:  ldloc.1
46   IL_0006:  bne.un IL_0016
47
48   IL_000b:  ldloc.0
49   IL_000c:  call void class [mscorlib]System.Console::Write(
       int32)
50   IL_0011:  br IL_001c
51
52   IL_0016:  ldc.i4.0
53   IL_0017:  call void class [mscorlib]System.Console::Write(
       int32)
54   IL_001c:  ret
55     } // end of method nano10::Main
56
57   } // end of class nano10
```

7.11 Nano 11

Listing 39: Introdução do comando de repetição enquanto(JavaScript)

```
1 var n, m, x;
2 n=1;
3 m=2;
4 x=5;
5 while(x > n){
6 n = n + m;
7 document.write(n);
8 }
```

Listing 40: Introdução do comando de repetição enquanto(C#)

```
using System;
class nano11 {
public static void Main(){
int n, m, x;
n = 1;
m = 2;
x = 5;

while ( x > n){
n = n + m;
Console.Write(n);
}
}
}
```

Listing 41: Introdução do comando de repetição enquanto(CIL)

```
.assembly 'nano11'
{
  .custom instance void class [mscorlib]System.Runtime.
     CompilerServices.RuntimeCompatibilityAttribute::'.ctor
     '() =  (
    01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
       T..WrapNonEx
    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
       ceptionThrows.

  .hash algorithm 0x00008004
  .ver  0:0:0:0
}
.module nano11.exe // GUID = {AF3ECC5A-57D3-401A-9BCE-0
    A8CCFA6FCBB}


  .class private auto ansi beforefieldinit nano11
    extends [mscorlib]System.Object
  {

    // method line 1
    .method public hidebysig specialname rtspecialname
           instance default void '.ctor' ()  cil managed
    {
        // Method begins at RVA 0x2050
  // Code size 7 (0x7)
  .maxstack 8
```

```
24   IL_0000:  ldarg.0
25   IL_0001:  call instance void object::'.ctor'()
26   IL_0006:  ret
27     } // end of method nano11::.ctor
28
29     // method line 2
30     .method public static hidebysig
31            default void Main ()  cil managed
32     {
33         // Method begins at RVA 0x2058
34   .entrypoint
35   // Code size 29 (0x1d)
36   .maxstack 2
37   .locals init (
38     int32 V_0,
39     int32 V_1,
40     int32 V_2)
41   IL_0000:  ldc.i4.1
42   IL_0001:  stloc.0
43   IL_0002:  ldc.i4.2
44   IL_0003:  stloc.1
45   IL_0004:  ldc.i4.5
46   IL_0005:  stloc.2
47   IL_0006:  br IL_0015
48
49   IL_000b:  ldloc.0
50   IL_000c:  ldloc.1
51   IL_000d:  add
52   IL_000e:  stloc.0
53   IL_000f:  ldloc.0
54   IL_0010:  call void class [mscorlib]System.Console::Write(
       int32)
55   IL_0015:  ldloc.2
56   IL_0016:  ldloc.0
57   IL_0017:  bgt IL_000b
58
59   IL_001c:  ret
60     } // end of method nano11::Main
61
62   } // end of class nano11
```

7.12 Nano 12

Listing 42: Comando condicional aninhando em um comando de repetição(JavaScript)

```
var n, m, x;
n = 1;
m = 2;
x = 5;
while(x>n){
if(n == m){
document.write(n);
}
else{
document.write(0);
}
x = x -1;
}
```

Listing 43: Comando condicional aninhando em um comando de repetição(C#)

```
using System;
class nano12 {
public static void Main(){
int n, m, x;
n = 1;
m = 2;
x = 5;

while ( x > n){
if ( n == m)
Console.Write(n);

else
Console.Write(0);
x = x - 1;
}
}
}
```

Listing 44: Comando condicional aninhando em um comando de repetição(CIL)

```
.assembly extern mscorlib
{
  .ver 4:0:0:0
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
}
.assembly 'nano12'
```

```
7  {
8    .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9      01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
         T..WrapNonEx
10     63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
         ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver  0:0:0:0
14 }
15 .module nano12.exe // GUID = {15C6B382-B3A6-4401-8043-
     CB4CD765B5EF}
16
17
18   .class private auto ansi beforefieldinit nano12
19     extends [mscorlib]System.Object
20   {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24            instance default void '.ctor' ()  cil managed
25     {
26         // Method begins at RVA 0x2050
27   // Code size 7 (0x7)
28   .maxstack 8
29   IL_0000:  ldarg.0
30   IL_0001:  call instance void object::'.ctor'()
31   IL_0006:  ret
32     } // end of method nano12::.ctor
33
34     // method line 2
35     .method public static hidebysig
36            default void Main ()  cil managed
37     {
38         // Method begins at RVA 0x2058
39   .entrypoint
40   // Code size 47 (0x2f)
41   .maxstack 2
42   .locals init (
43     int32 V_0,
44     int32 V_1,
45     int32 V_2)
46   IL_0000:  ldc.i4.1
```

```
47    IL_0001:  stloc.0
48    IL_0002:  ldc.i4.2
49    IL_0003:  stloc.1
50    IL_0004:  ldc.i4.5
51    IL_0005:  stloc.2
52    IL_0006:  br IL_0027
53
54    IL_000b:  ldloc.0
55    IL_000c:  ldloc.1
56    IL_000d:  bne.un IL_001d
57
58    IL_0012:  ldloc.0
59    IL_0013:  call void class [mscorlib]System.Console::Write(
          int32)
60    IL_0018:  br IL_0023
61
62    IL_001d:  ldc.i4.0
63    IL_001e:  call void class [mscorlib]System.Console::Write(
          int32)
64    IL_0023:  ldloc.2
65    IL_0024:  ldc.i4.1
66    IL_0025:  sub
67    IL_0026:  stloc.2
68    IL_0027:  ldloc.2
69    IL_0028:  ldloc.0
70    IL_0029:  bgt IL_000b
71
72    IL_002e:  ret
73      } // end of method nano12::Main
74
75    } // end of class nano12
```

7.13 Micro 01

Listing 45: Converte graus Celsius para Fahrenheit(JavaScript)

```
1 var cel, far;
2 document.write("Tabela de conversao: Celsuis -> Fahrenheit");
3 cel = prompt("Digite a temperatura em Celsius: ");
4 far = (9*cel+160)/5
5 document.write("A nova temperatura é: "  +far);
```

Listing 46: Converte graus Celsius para Fahrenheit(C#)

```
1 using System;
```

```
2 class micro01 {
3 public static void Main(){
4 double cel, far;
5 Console.WriteLine("Tabela de conversão: Celsius -> Fahrenheit
    ");
6 Console.Write("Digite a temperatura em Celsius: ");
7 cel = Convert.ToDouble(Console.ReadLine());
8 far = (9*cel+160)/5;
9 Console.WriteLine("A nova temperatura é " + far + " F");
10 }
11 }
```

Listing 47: Converte graus Celsius para Fahrenheit(CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro01'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
        T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
        ceptionThrows.
11
12  .hash algorithm 0x00008004
13  .ver  0:0:0:0
14 }
15 .module micro01.exe // GUID = {BB705713-99F6-4277-9269-743
    CACC6EC45}
16
17
18  .class private auto ansi beforefieldinit micro01
19    extends [mscorlib]System.Object
20  {
21
22    // method line 1
23    .method public hidebysig specialname rtspecialname
24           instance default void '.ctor' ()  cil managed
25    {
26        // Method begins at RVA 0x2050
```

```
27  // Code size 7 (0x7)
28  .maxstack 8
29  IL_0000:  ldarg.0
30  IL_0001:  call instance void object::'.ctor'()
31  IL_0006:  ret
32    } // end of method micro01::.ctor
33
34    // method line 2
35    .method public static hidebysig
36           default void Main ()  cil managed
37    {
38        // Method begins at RVA 0x2058
39  .entrypoint
40  // Code size 90 (0x5a)
41  .maxstack 3
42  .locals init (
43    float64 V_0,
44    float64 V_1)
45  IL_0000:  ldstr bytearray (
46  54 00 61 00 62 00 65 00 6c 00 61 00 20 00 64 00  // T.a.b.e
       .l.a. .d.
47  65 00 20 00 63 00 6f 00 6e 00 76 00 65 00 72 00  // e. .c.o
       .n.v.e.r.
48  73 00 e3 00 6f 00 3a 00 20 00 43 00 65 00 6c 00  // s...o
       .:. .C.e.l.
49  73 00 69 00 75 00 73 00 20 00 2d 00 3e 00 20 00  // s.i.u.s
       . .-.>. .
50  46 00 61 00 68 00 72 00 65 00 6e 00 68 00 65 00  // F.a.h.r
       .e.n.h.e.
51  69 00 74 00 01 )                                 // i.t..
52
53  IL_0005:  call void class [mscorlib]System.Console::
       WriteLine(string)
54  IL_000a:  ldstr "Digite a temperatura em Celsius: "
55  IL_000f:  call void class [mscorlib]System.Console::Write(
       string)
56  IL_0014:  call string class [mscorlib]System.Console::
       ReadLine()
57  IL_0019:  call float64 class [mscorlib]System.Convert::
       ToDouble(string)
58  IL_001e:  stloc.0
59  IL_001f:  ldc.r8 9.
60  IL_0028:  ldloc.0
61  IL_0029:  mul
62  IL_002a:  ldc.r8 160.
```

```
63    IL_0033:  add
64    IL_0034:  ldc.r8 5.
65    IL_003d:  div
66    IL_003e:  stloc.1
67    IL_003f:  ldstr bytearray (
68    41 00 20 00 6e 00 6f 00 76 00 61 00 20 00 74 00  // A. .n.o
         .v.a. .t.
69    65 00 6d 00 70 00 65 00 72 00 61 00 74 00 75 00  // e.m.p.e
         .r.a.t.u.
70    72 00 61 00 20 00 e9 00 20 00 01 )               // r.a.
         ... ..
71
72    IL_0044:  ldloc.1
73    IL_0045:  box [mscorlib]System.Double
74    IL_004a:  ldstr " F"
75    IL_004f:  call string string::Concat(object, object, object
         )
76    IL_0054:  call void class [mscorlib]System.Console::
         WriteLine(string)
77    IL_0059:  ret
78      } // end of method micro01::Main
79
80    } // end of class micro01
```

7.14 Micro 02

Listing 48: Lê dois inteiros e decide qual é maior (JavaScript)

```
1 var num1, num2;
2 num1 = prompt("Digite o primeiro numero: ");
3 num2 = prompt("Digite o segundo numero: ");
4 if(num1 > num2){
5 document.write("O primeiro numero " +num1+ "é maior que o
     segundo" +num2);
6 }
7 else{
8 document.write("O segundo numero " +num2+ "é maior que o
     primeiro" +num1);
9 }
```

Listing 49: Ler dois inteiros e decide qual é maior (C#)

```
1 using System;
2 class micro02 {
3 public static void Main(){
```

```
4 int num1, num2;
5 Console.Write("Digite o primeiro número: ");
6 num1 = Convert.ToInt32(Console.ReadLine());
7 Console.Write("Digite o segundo número: ");
8 num2 = Convert.ToInt32(Console.ReadLine());
9 if ( num1 > num2)
10 Console.Write("O primeiro número " + num1 + " é maior que o
     segundo " + num2);
11 else
12
13 Console.Write("O segundo número " + num2 + " é maior que o
     primeiro" + num2);
14 }
15 }
```

Listing 50: Ler dois inteiros e decide qual é maior (CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro02'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
        T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
        ceptionThrows.
11
12  .hash algorithm 0x00008004
13  .ver  0:0:0:0
14 }
15 .module micro02.exe // GUID = {E62D9F80-4E16-4D3D-9E3A
     -5761710E4171}
16
17
18  .class private auto ansi beforefieldinit micro02
19    extends [mscorlib]System.Object
20  {
21
22    // method line 1
23    .method public hidebysig specialname rtspecialname
```

```
24            instance default void '.ctor' ()  cil managed
25    {
26        // Method begins at RVA 0x2050
27   // Code size 7 (0x7)
28   .maxstack 8
29   IL_0000:  ldarg.0
30   IL_0001:  call instance void object::'.ctor'()
31   IL_0006:  ret
32    } // end of method micro02::.ctor
33
34    // method line 2
35    .method public static hidebysig
36           default void Main ()  cil managed
37    {
38        // Method begins at RVA 0x2058
39   .entrypoint
40   // Code size 155 (0x9b)
41   .maxstack 4
42   .locals init (
43    int32 V_0,
44    int32 V_1)
45   IL_0000:  ldstr bytearray (
46   44 00 69 00 67 00 69 00 74 00 65 00 20 00 6f 00  // D.i.g.i
       .t.e. .o.
47   20 00 70 00 72 00 69 00 6d 00 65 00 69 00 72 00  //  .p.r.i
       .m.e.i.r.
48   6f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // o. .n
       ...m.e.r.o.
49   3a 00 20 00 01 )                                 // :. ..
50
51   IL_0005:  call void class [mscorlib]System.Console::Write(
       string)
52   IL_000a:  call string class [mscorlib]System.Console::
       ReadLine()
53   IL_000f:  call int32 class [mscorlib]System.Convert::
       ToInt32(string)
54   IL_0014:  stloc.0
55   IL_0015:  ldstr bytearray (
56   44 00 69 00 67 00 69 00 74 00 65 00 20 00 6f 00  // D.i.g.i
       .t.e. .o.
57   20 00 73 00 65 00 67 00 75 00 6e 00 64 00 6f 00  //  .s.e.g
       .u.n.d.o.
58   20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 3a 00  //  .n...m
       .e.r.o.:.
59   20 00 01 )                                       //  ..
```

```
60
61   IL_001a:  call void class [mscorlib]System.Console::Write(
        string)
62   IL_001f:  call string class [mscorlib]System.Console::
        ReadLine()
63   IL_0024:  call int32 class [mscorlib]System.Convert::
        ToInt32(string)
64   IL_0029:  stloc.1
65   IL_002a:  ldloc.0
66   IL_002b:  ldloc.1
67   IL_002c:  ble IL_0068
68
69   IL_0031:  ldc.i4.4
70   IL_0032:  newarr [mscorlib]System.Object
71   IL_0037:  dup
72   IL_0038:  ldc.i4.0
73   IL_0039:  ldstr bytearray (
74   4f 00 20 00 70 00 72 00 69 00 6d 00 65 00 69 00  // O. .p.r
        .i.m.e.i.
75   72 00 6f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00  // r.o. .n
        ...m.e.r.
76   6f 00 20 00 01 )                                 // o. ..
77
78   IL_003e:  stelem.ref
79   IL_003f:  dup
80   IL_0040:  ldc.i4.1
81   IL_0041:  ldloc.0
82   IL_0042:  box [mscorlib]System.Int32
83   IL_0047:  stelem.ref
84   IL_0048:  dup
85   IL_0049:  ldc.i4.2
86   IL_004a:  ldstr bytearray (
87   20 00 e9 00 20 00 6d 00 61 00 69 00 6f 00 72 00  //  ... .m
        .a.i.o.r.
88   20 00 71 00 75 00 65 00 20 00 6f 00 20 00 73 00  //  .q.u.e
        . .o. .s.
89   65 00 67 00 75 00 6e 00 64 00 6f 00 20 00 01 )   // e.g.u.n
        .d.o. ..
90
91   IL_004f:  stelem.ref
92   IL_0050:  dup
93   IL_0051:  ldc.i4.3
94   IL_0052:  ldloc.1
95   IL_0053:  box [mscorlib]System.Int32
96   IL_0058:  stelem.ref
```

```
97   IL_0059:  call string string::Concat(object[])
98   IL_005e:  call void class [mscorlib]System.Console::Write(
        string)
99   IL_0063:  br IL_009a
100
101  IL_0068:  ldc.i4.4
102  IL_0069:  newarr [mscorlib]System.Object
103  IL_006e:  dup
104  IL_006f:  ldc.i4.0
105  IL_0070:  ldstr bytearray (
106  4f 00 20 00 73 00 65 00 67 00 75 00 6e 00 64 00  // O. .s.e
        .g.u.n.d.
107  6f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // o. .n
        ...m.e.r.o.
108  20 00 01 )                                       //  ..
109
110  IL_0075:  stelem.ref
111  IL_0076:  dup
112  IL_0077:  ldc.i4.1
113  IL_0078:  ldloc.1
114  IL_0079:  box [mscorlib]System.Int32
115  IL_007e:  stelem.ref
116  IL_007f:  dup
117  IL_0080:  ldc.i4.2
118  IL_0081:  ldstr bytearray (
119  20 00 e9 00 20 00 6d 00 61 00 69 00 6f 00 72 00  //  ... .m
        .a.i.o.r.
120  20 00 71 00 75 00 65 00 20 00 6f 00 20 00 70 00  //  .q.u.e
        . .o. .p.
121  72 00 69 00 6d 00 65 00 69 00 72 00 6f 00 01 )   // r.i.m.e
        .i.r.o..
122
123  IL_0086:  stelem.ref
124  IL_0087:  dup
125  IL_0088:  ldc.i4.3
126  IL_0089:  ldloc.1
127  IL_008a:  box [mscorlib]System.Int32
128  IL_008f:  stelem.ref
129  IL_0090:  call string string::Concat(object[])
130  IL_0095:  call void class [mscorlib]System.Console::Write(
        string)
131  IL_009a:  ret
132    } // end of method micro02::Main
133
134  } // end of class micro02
```

7.15 Micro 03

Listing 51: Lê um número e verifica se ele está entre 100 e 200(JavaScript)

```
var numero;
numero = prompt("Digite um numero: ");
if(numero >= 100){
if(numero <= 200){
document.write("O numero está no intervalo entre 100 e 200");
}
else{
document.write("O numero nao está no intervalo entre 100 e
    200");
}
}
else{
document.write("O numero nao está no intervalo entre 100 e
    200");
}
```

Listing 52: Lê um número e verifica se ele está entre 100 e 200(C#)

```
using System;
class micro03 {
public static void Main(){
int numero;
Console.Write("Digite um número: ");
numero = Convert.ToInt32(Console.ReadLine());
if ( numero >= 100){
if ( numero <= 200)
Console.WriteLine("O número  esta no intervalo entre 100 e
    200");
else
Console.WriteLine("O número não está no intervalo entre 100 e
     200");
}
else
Console.WriteLine("O número não está no intervalo entre 100 e
     200");
}
}
```

Listing 53: Lê um número e verifica se ele está entre 100 e 200(CIL)

```
.assembly extern mscorlib
```

```
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro03'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
        T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
        ceptionThrows.
11
12  .hash algorithm 0x00008004
13  .ver  0:0:0:0
14 }
15 .module micro03.exe // GUID = {E42F4BE8-EE27-49F9-BE32-1
     C00BE87B41B}
16
17
18  .class private auto ansi beforefieldinit micro03
19    extends [mscorlib]System.Object
20  {
21
22    // method line 1
23    .method public hidebysig specialname rtspecialname
24           instance default void '.ctor' ()  cil managed
25    {
26        // Method begins at RVA 0x2050
27  // Code size 7 (0x7)
28  .maxstack 8
29  IL_0000:  ldarg.0
30  IL_0001:  call instance void object::'.ctor'()
31  IL_0006:  ret
32    } // end of method micro03::.ctor
33
34    // method line 2
35    .method public static hidebysig
36           default void Main ()  cil managed
37    {
38        // Method begins at RVA 0x2058
39  .entrypoint
40  // Code size 81 (0x51)
41  .maxstack 2
```

```
42  .locals init (
43    int32 V_0)
44  IL_0000:  ldstr bytearray (
45  44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00  // D.i.g.i
       .t.e. .u.
46  6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // m. .n
       ...m.e.r.o.
47  3a 00 20 00 01 )                                 // :. ..
48
49  IL_0005:  call void class [mscorlib]System.Console::Write(
       string)
50  IL_000a:  call string class [mscorlib]System.Console::
       ReadLine()
51  IL_000f:  call int32 class [mscorlib]System.Convert::
       ToInt32(string)
52  IL_0014:  stloc.0
53  IL_0015:  ldloc.0
54  IL_0016:  ldc.i4.s 0x64
55  IL_0018:  blt IL_0046
56
57  IL_001d:  ldloc.0
58  IL_001e:  ldc.i4 200
59  IL_0023:  bgt IL_0037
60
61  IL_0028:  ldstr bytearray (
62  4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // O. .n
       ...m.e.r.o.
63  20 00 20 00 65 00 73 00 74 00 61 00 20 00 6e 00  //  . .e.s
       .t.a. .n.
64  6f 00 20 00 69 00 6e 00 74 00 65 00 72 00 76 00  // o. .i.n
       .t.e.r.v.
65  61 00 6c 00 6f 00 20 00 65 00 6e 00 74 00 72 00  // a.l.o.
       .e.n.t.r.
66  65 00 20 00 31 00 30 00 30 00 20 00 65 00 20 00  // e.
       .1.0.0. .e. .
67  32 00 30 00 30 00 01 )                           // 2.0.0..
68
69  IL_002d:  call void class [mscorlib]System.Console::
       WriteLine(string)
70  IL_0032:  br IL_0041
71
72  IL_0037:  ldstr bytearray (
73  4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // O. .n
       ...m.e.r.o.
74  20 00 6e 00 e3 00 6f 00 20 00 65 00 73 00 74 00  //  .n...o
```

```
      . .e.s.t.
75    e1 00 20 00 6e 00 6f 00 20 00 69 00 6e 00 74 00  // .. .n.o
      . .i.n.t.
76    65 00 72 00 76 00 61 00 6c 00 6f 00 20 00 65 00  // e.r.v.a
      .l.o. .e.
77    6e 00 74 00 72 00 65 00 20 00 31 00 30 00 30 00  // n.t.r.e
      . .1.0.0.
78    20 00 65 00 20 00 32 00 30 00 30 00 01 )         //  .e.
      .2.0.0..
79
80    IL_003c:  call void class [mscorlib]System.Console::
      WriteLine(string)
81    IL_0041:  br IL_0050
82
83    IL_0046:  ldstr bytearray (
84    4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // O. .n
      ...m.e.r.o.
85    20 00 6e 00 e3 00 6f 00 20 00 65 00 73 00 74 00  //  .n...o
      . .e.s.t.
86    e1 00 20 00 6e 00 6f 00 20 00 69 00 6e 00 74 00  // .. .n.o
      . .i.n.t.
87    65 00 72 00 76 00 61 00 6c 00 6f 00 20 00 65 00  // e.r.v.a
      .l.o. .e.
88    6e 00 74 00 72 00 65 00 20 00 31 00 30 00 30 00  // n.t.r.e
      . .1.0.0.
89    20 00 65 00 20 00 32 00 30 00 30 00 01 )         //  .e.
      .2.0.0..
90
91    IL_004b:  call void class [mscorlib]System.Console::
      WriteLine(string)
92    IL_0050:  ret
93      } // end of method micro03::Main
94
95    } // end of class micro03
```

7.16 Micro 04

Listing 54: Lê números e informa quais estão entre 10 e 150(JavaScript)

```
1 var x, num, intervalo;
2 intervalo = 0;
3 for(x=1;x<=5;x++){
4 num = prompt("Digite um numero");
5 if(num >= 10){
6 if(num  <= 150){
```

```
7 intervalo = intervalo +1;
8 }
9 }
10 }
11 document.write("Ao total foram digitados " +intervalo+ "
     numeros no intervalo entre 10 e 50");
```

Listing 55: Lê números e informa quais estão entre 10 e 150(C#)

```
1 using System;
2 class micro04 {
3 public static void Main(){
4 int x, num, intervalo;
5
6 intervalo = 0;
7
8 for ( x = 1; x <= 5; x++){
9 Console.Write("Digite um número: ");
10 num = Convert.ToInt32(Console.ReadLine());
11 if ( num >= 10 )
12
13 if ( num <= 150 )
14 intervalo = intervalo + 1;
15
16 }
17 Console.WriteLine("Ao total, foram digitados " + intervalo +
     " números no intervalo entre 10 e 150");
18
19 }
20 }
```

Listing 56: Lê números e informa quais estão entre 10 e 150(CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro04'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
         T..WrapNonEx
```

```
10      63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
           ceptionThrows.
11
12    .hash algorithm 0x00008004
13    .ver  0:0:0:0
14  }
15  .module micro04.exe // GUID = {6FD00F32-6B00-44B9-BB2C-278
       DA96AAABA}
16
17
18    .class private auto ansi beforefieldinit micro04
19      extends [mscorlib]System.Object
20    {
21
22      // method line 1
23      .method public hidebysig specialname rtspecialname
24              instance default void '.ctor' ()  cil managed
25      {
26          // Method begins at RVA 0x2050
27    // Code size 7 (0x7)
28    .maxstack 8
29    IL_0000:  ldarg.0
30    IL_0001:  call instance void object::'.ctor'()
31    IL_0006:  ret
32      } // end of method micro04::.ctor
33
34      // method line 2
35      .method public static hidebysig
36              default void Main ()  cil managed
37      {
38          // Method begins at RVA 0x2058
39    .entrypoint
40    // Code size 91 (0x5b)
41    .maxstack 3
42    .locals init (
43      int32 V_0,
44      int32 V_1,
45      int32 V_2)
46    IL_0000:  ldc.i4.0
47    IL_0001:  stloc.2
48    IL_0002:  ldc.i4.1
49    IL_0003:  stloc.0
50    IL_0004:  br IL_0039
51
52    IL_0009:  ldstr bytearray (
```

```
53   44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00  // D.i.g.i
       .t.e. .u.
54   6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // m. .n
       ...m.e.r.o.
55   3a 00 20 00 01 )                                 // :. ..
56
57   IL_000e:  call void class [mscorlib]System.Console::Write(
       string)
58   IL_0013:  call string class [mscorlib]System.Console::
       ReadLine()
59   IL_0018:  call int32 class [mscorlib]System.Convert::
       ToInt32(string)
60   IL_001d:  stloc.1
61   IL_001e:  ldloc.1
62   IL_001f:  ldc.i4.s 0x0a
63   IL_0021:  blt IL_0035
64
65   IL_0026:  ldloc.1
66   IL_0027:  ldc.i4 150
67   IL_002c:  bgt IL_0035
68
69   IL_0031:  ldloc.2
70   IL_0032:  ldc.i4.1
71   IL_0033:  add
72   IL_0034:  stloc.2
73   IL_0035:  ldloc.0
74   IL_0036:  ldc.i4.1
75   IL_0037:  add
76   IL_0038:  stloc.0
77   IL_0039:  ldloc.0
78   IL_003a:  ldc.i4.5
79   IL_003b:  ble IL_0009
80
81   IL_0040:  ldstr "Ao total, foram digitados "
82   IL_0045:  ldloc.2
83   IL_0046:  box [mscorlib]System.Int32
84   IL_004b:  ldstr bytearray (
85   20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00 73 00  //  .n...m
       .e.r.o.s.
86   20 00 6e 00 6f 00 20 00 69 00 6e 00 74 00 65 00  //  .n.o.
       .i.n.t.e.
87   72 00 76 00 61 00 6c 00 6f 00 20 00 65 00 6e 00  // r.v.a.l
       .o. .e.n.
88   74 00 72 00 65 00 20 00 31 00 30 00 20 00 65 00  // t.r.e.
       .1.0. .e.
```

```
89    20 00 31 00 35 00 30 00 01 )                        //
          .1.5.0..
90
91    IL_0050:  call string string::Concat(object, object, object
          )
92    IL_0055:  call void class [mscorlib]System.Console::
          WriteLine(string)
93    IL_005a:  ret
94      } // end of method micro04::Main
95
96    } // end of class micro04
```

7.17 Micro 05

Listing 57: Lê strings e caracteres(JavaScript)

```
1  var nome, sexo, x, h, m;
2  h = 0;
3  m = 0;
4  for(x=1;x<=5;x++){
5  nome = prompt("Digite o nome: ");
6  sexo = prompt("H  Homem ou M  Mulher: ");
7  switch(sexo){
8  case 'H':
9  h = h +1;
10 break;
11 case 'M':
12 m = m +1;
13 break;
14 default:
15 document.write("Sexo só pode ser H ou M!");
16 }
17 }
18 document.write("Foram inseridos " +h+ "homens");
19 document.write("Foram inseridos " +m+ "mulheres");
```

Listing 58: Lê strings e caracteres(C#)

```
1  using System;
2  class micro05 {
3  public static void Main(){
4  string nome;
5  char sexo;
6  int x, h, m;
7
```

```
h = 0;
m = 0;

for ( x = 1; x <= 5; x++){
Console.Write("Digite o nome: ");
nome = Console.ReadLine();
Console.Write("H - Homem ou M - Mulher: ");
sexo = Convert.ToChar(Console.Read());
switch (sexo){
case 'H':
 h = h + 1;
 break;
case 'M':
 m = m + 1;
break;
default:
Console.WriteLine("Sexo só pode ser H ou M!");
break;

}
}
Console.WriteLine("Foram inseridos " + h + " Homens");
Console.WriteLine("Foram inseridos " + m + " Mulheres");
}
}
```

Listing 59: Lê strings e caracteres(CIL)

```
.assembly extern mscorlib
{
  .ver 4:0:0:0
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
}
.assembly 'micro05'
{
  .custom instance void class [mscorlib]System.Runtime.
     CompilerServices.RuntimeCompatibilityAttribute::'.ctor
     '() =  (
    01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
       T..WrapNonEx
    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
       ceptionThrows.

  .hash algorithm 0x00008004
  .ver  0:0:0:0
```

```
14 }
15 .module micro05.exe // GUID = {72D3380D-485B-4ADB-8D92-492
       D7DB0832A}
16
17
18   .class private auto ansi beforefieldinit micro05
19     extends [mscorlib]System.Object
20   {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24             instance default void '.ctor' ()  cil managed
25     {
26         // Method begins at RVA 0x2050
27   // Code size 7 (0x7)
28   .maxstack 8
29   IL_0000:  ldarg.0
30   IL_0001:  call instance void object::'.ctor'()
31   IL_0006:  ret
32     } // end of method micro05::.ctor
33
34     // method line 2
35     .method public static hidebysig
36             default void Main ()  cil managed
37     {
38         // Method begins at RVA 0x2058
39   .entrypoint
40   // Code size 170 (0xaa)
41   .maxstack 3
42   .locals init (
43     string  V_0,
44     char  V_1,
45     int32 V_2,
46     int32 V_3,
47     int32 V_4)
48   IL_0000:  ldc.i4.0
49   IL_0001:  stloc.3
50   IL_0002:  ldc.i4.0
51   IL_0003:  stloc.s 4
52   IL_0005:  ldc.i4.1
53   IL_0006:  stloc.2
54   IL_0007:  br IL_006d
55
56   IL_000c:  ldstr "Digite o nome: "
57   IL_0011:  call void class [mscorlib]System.Console::Write(
```

```
      string)
58    IL_0016:  call string class [mscorlib]System.Console::
         ReadLine()
59    IL_001b:  stloc.0
60    IL_001c:  ldstr "H - Homem ou M - Mulher: "
61    IL_0021:  call void class [mscorlib]System.Console::Write(
         string)
62    IL_0026:  call int32 class [mscorlib]System.Console::Read()
63    IL_002b:  call char class [mscorlib]System.Convert::ToChar(
         int32)
64    IL_0030:  stloc.1
65    IL_0031:  ldloc.1
66    IL_0032:  ldc.i4.s 0x48
67    IL_0034:  beq IL_0046
68
69    IL_0039:  ldloc.1
70    IL_003a:  ldc.i4.s 0x4d
71    IL_003c:  beq IL_004f
72
73    IL_0041:  br IL_005a
74
75    IL_0046:  ldloc.3
76    IL_0047:  ldc.i4.1
77    IL_0048:  add
78    IL_0049:  stloc.3
79    IL_004a:  br IL_0069
80
81    IL_004f:  ldloc.s 4
82    IL_0051:  ldc.i4.1
83    IL_0052:  add
84    IL_0053:  stloc.s 4
85    IL_0055:  br IL_0069
86
87    IL_005a:  ldstr bytearray (
88    53 00 65 00 78 00 6f 00 20 00 73 00 f3 00 20 00  // S.e.x.o
         . .s... .
89    70 00 6f 00 64 00 65 00 20 00 73 00 65 00 72 00  // p.o.d.e
         . .s.e.r.
90    20 00 48 00 20 00 6f 00 75 00 20 00 4d 00 21 00  //  .H. .o
         .u. .M.!.
91    01 )                                             // .
92
93    IL_005f:  call void class [mscorlib]System.Console::
         WriteLine(string)
94    IL_0064:  br IL_0069
```

```
95
96    IL_0069:  ldloc.2
97    IL_006a:  ldc.i4.1
98    IL_006b:  add
99    IL_006c:  stloc.2
100   IL_006d:  ldloc.2
101   IL_006e:  ldc.i4.5
102   IL_006f:  ble IL_000c
103
104   IL_0074:  ldstr "Foram inseridos "
105   IL_0079:  ldloc.3
106   IL_007a:  box [mscorlib]System.Int32
107   IL_007f:  ldstr " Homens"
108   IL_0084:  call string string::Concat(object, object, object
          )
109   IL_0089:  call void class [mscorlib]System.Console::
          WriteLine(string)
110   IL_008e:  ldstr "Foram inseridos "
111   IL_0093:  ldloc.s 4
112   IL_0095:  box [mscorlib]System.Int32
113   IL_009a:  ldstr " Mulheres"
114   IL_009f:  call string string::Concat(object, object, object
          )
115   IL_00a4:  call void class [mscorlib]System.Console::
          WriteLine(string)
116   IL_00a9:  ret
117     } // end of method micro05::Main
118
119   } // end of class micro05
```

7.18 Micro 06

Listing 60: Escreve um número lido por extenso(JavaScript)

```
1  var numero;
2  numero = prompt("Digite um numero de 1 a 5: ");
3  switch(numero){
4  case 1:
5  document.write("Um");
6  break;
7  case 2:
8  document.write("Dois");
9  break;
10 case 3:
11 document.write("Três");
```

```
break;
case 4:
document.write("Quatro");
break;
case 5:
document.write("Cinco");
break;
default:
document.write("Numero invalido!");
break;
}
```

Listing 61: Escreve um número lido por extenso(C#)

```
using System;
class micro06 {
public static void Main(){
int numero;
Console.Write("Digite um número de 1 a 5: ");
numero = Convert.ToInt32(Console.ReadLine());
switch(numero){
case 1:
Console.WriteLine("Um");
break;
case 2:
Console.WriteLine("Dois");
break;
case 3:
Console.WriteLine("Três");
break;
case 4:
Console.WriteLine("Quatro");
break;
case 5:
Console.WriteLine("Cinco");
break;
default:
Console.WriteLine("Número Inválido!!!");
break;
}
}
}
```

Listing 62: Escreve um número lido por extenso(C#)

```
using System;
class micro06 {
public static void Main(){
int numero;
Console.Write("Digite um número de 1 a 5: ");
numero = Convert.ToInt32(Console.ReadLine());
switch(numero){
case 1:
Console.WriteLine("Um");
break;
case 2:
Console.WriteLine("Dois");
break;
case 3:
Console.WriteLine("Três");
break;
case 4:
Console.WriteLine("Quatro");
break;
case 5:
Console.WriteLine("Cinco");
break;
default:
Console.WriteLine("Número Inválido!!!");
break;
}
}
}
```

Listing 63: Escreve um número lido por extenso(CIL)

```
.assembly extern mscorlib
{
  .ver 4:0:0:0
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
}
.assembly 'micro06'
{
  .custom instance void class [mscorlib]System.Runtime.
     CompilerServices.RuntimeCompatibilityAttribute::'.ctor
     '() =  (
    01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
       T..WrapNonEx
    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
       ceptionThrows.
```

```
11
12   .hash algorithm 0x00008004
13   .ver  0:0:0:0
14 }
15 .module micro06.exe // GUID = {C6D580A0-F5D7-4265-A1A7-4
     A6EC7748578}
16
17
18   .class private auto ansi beforefieldinit micro06
19     extends [mscorlib]System.Object
20   {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24            instance default void '.ctor' ()  cil managed
25     {
26         // Method begins at RVA 0x2050
27   // Code size 7 (0x7)
28   .maxstack 8
29   IL_0000:  ldarg.0
30   IL_0001:  call instance void object::'.ctor'()
31   IL_0006:  ret
32     } // end of method micro06::.ctor
33
34     // method line 2
35     .method public static hidebysig
36            default void Main ()  cil managed
37     {
38         // Method begins at RVA 0x2058
39   .entrypoint
40   // Code size 145 (0x91)
41   .maxstack 2
42   .locals init (
43     int32 V_0)
44   IL_0000:  ldstr bytearray (
45   44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00  // D.i.g.i
       .t.e. .u.
46   6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // m. .n
       ...m.e.r.o.
47   20 00 64 00 65 00 20 00 31 00 20 00 61 00 20 00  //  .d.e.
       .1. .a. .
48   35 00 3a 00 20 00 01 )                           // 5.:. ..
49
50   IL_0005:  call void class [mscorlib]System.Console::Write(
       string)
```

```
51   IL_000a:  call string class [mscorlib]System.Console::
        ReadLine()
52   IL_000f:  call int32 class [mscorlib]System.Convert::
        ToInt32(string)
53   IL_0014:  stloc.0
54   IL_0015:  ldloc.0
55   IL_0016:  ldc.i4.1
56   IL_0017:  sub
57   IL_0018:  switch (
58     IL_0036,
59     IL_0045,
60     IL_0054,
61     IL_0063,
62     IL_0072)
63   IL_0031:  br IL_0081
64
65   IL_0036:  ldstr "Um"
66   IL_003b:  call void class [mscorlib]System.Console::
        WriteLine(string)
67   IL_0040:  br IL_0090
68
69   IL_0045:  ldstr "Dois"
70   IL_004a:  call void class [mscorlib]System.Console::
        WriteLine(string)
71   IL_004f:  br IL_0090
72
73   IL_0054:  ldstr bytearray (
74   54 00 72 00 ea 00 73 00 01 )                      // T.r...s
        ..
75
76   IL_0059:  call void class [mscorlib]System.Console::
        WriteLine(string)
77   IL_005e:  br IL_0090
78
79   IL_0063:  ldstr "Quatro"
80   IL_0068:  call void class [mscorlib]System.Console::
        WriteLine(string)
81   IL_006d:  br IL_0090
82
83   IL_0072:  ldstr "Cinco"
84   IL_0077:  call void class [mscorlib]System.Console::
        WriteLine(string)
85   IL_007c:  br IL_0090
86
87   IL_0081:  ldstr bytearray (
```

```
88   4e 00 fa 00 6d 00 65 00 72 00 6f 00 20 00 49 00  // N...m.e
        .r.o. .I.
89   6e 00 76 00 e1 00 6c 00 69 00 64 00 6f 00 21 00  // n.v...l
        .i.d.o.!.
90   21 00 21 00 01 )                                 // !.!..
91
92   IL_0086:  call void class [mscorlib]System.Console::
        WriteLine(string)
93   IL_008b:  br IL_0090
94
95   IL_0090:  ret
96     } // end of method micro06::Main
97
98   } // end of class micro06
```

7.19 Micro 07

Listing 64: Decide se os números são positivos zeros ou negativos(JavaScript)

```
1  var programa, numero, opc;
2  programa = 1;
3  while(programa){
4  numero = prompt("Digite um numero: ");
5  if(numero > 0)
6  document.write("Positivo");
7  else{
8  if(numero == 0)
9  document.write("O numero é igual a 0");
10 else{
11 if(numero < 0)
12 document.write("Negativo");
13 }
14 }
15 opc = prompt("Deseja finalizar? (S/N) ");
16 if(opc == 'S')
17 programa = 0;
18 }
```

Listing 65: Decide se os números são positivos zeros ou negativos(C#)

```
1  using System;
2  class micro07 {
3  public static void Main(){
4
5  int programa, numero;
```

```
6 char opc;
7 programa = 1;
8 while ( programa == 1 ){
9 Console.WriteLine("Digite um número: ");
10 numero = Convert.ToInt32(Console.ReadLine());
11 if ( numero > 0 )
12 Console.WriteLine("Positivo");
13 else {
14 if ( numero == 0 )
15 Console.WriteLine("O número é igual a 0");
16 if ( numero < 0 )
17 Console.WriteLine("Negativo");
18
19 }
20 Console.Write("Desja finalizar? (S/N) ");
21 opc = Convert.ToChar(Console.ReadLine());
22 if ( opc == 'S' )
23 programa = 0;
24
25 }
26 }
27 }
```

Listing 66: Decide se os números são positivos zeros ou negativos(CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro07'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
        T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
        ceptionThrows.
11
12  .hash algorithm 0x00008004
13  .ver  0:0:0:0
14 }
15 .module micro07.exe // GUID = {09933DC8-14E9-4C2F-9CF0-
     B1DBBC62502F}
```

```
16
17
18  .class private auto ansi beforefieldinit micro07
19    extends [mscorlib]System.Object
20  {
21
22    // method line 1
23    .method public hidebysig specialname rtspecialname
24           instance default void '.ctor' ()  cil managed
25    {
26        // Method begins at RVA 0x2050
27  // Code size 7 (0x7)
28  .maxstack 8
29  IL_0000:  ldarg.0
30  IL_0001:  call instance void object::'.ctor'()
31  IL_0006:  ret
32    } // end of method micro07::.ctor
33
34    // method line 2
35    .method public static hidebysig
36           default void Main ()  cil managed
37    {
38        // Method begins at RVA 0x2058
39  .entrypoint
40  // Code size 122 (0x7a)
41  .maxstack 2
42  .locals init (
43    int32 V_0,
44    int32 V_1,
45    char  V_2)
46  IL_0000:  ldc.i4.1
47  IL_0001:  stloc.0
48  IL_0002:  br IL_0072
49
50  IL_0007:  ldstr bytearray (
51  44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00  // D.i.g.i
       .t.e. .u.
52  6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // m. .n
       ...m.e.r.o.
53  3a 00 20 00 01 )                                 // :. ..
54
55  IL_000c:  call void class [mscorlib]System.Console::
       WriteLine(string)
56  IL_0011:  call string class [mscorlib]System.Console::
       ReadLine()
```

```
57   IL_0016:  call int32 class [mscorlib]System.Convert::
        ToInt32(string)
58   IL_001b:  stloc.1
59   IL_001c:  ldloc.1
60   IL_001d:  ldc.i4.0
61   IL_001e:  ble IL_0032
62
63   IL_0023:  ldstr "Positivo"
64   IL_0028:  call void class [mscorlib]System.Console::
        WriteLine(string)
65   IL_002d:  br IL_0053
66
67   IL_0032:  ldloc.1
68   IL_0033:  brtrue IL_0042
69
70   IL_0038:  ldstr bytearray (
71   4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // O. .n
        ...m.e.r.o.
72   20 00 e9 00 20 00 69 00 67 00 75 00 61 00 6c 00  //  ... .i
        .g.u.a.l.
73   20 00 61 00 20 00 30 00 01 )                     //  .a.
        .0..
74
75   IL_003d:  call void class [mscorlib]System.Console::
        WriteLine(string)
76   IL_0042:  ldloc.1
77   IL_0043:  ldc.i4.0
78   IL_0044:  bge IL_0053
79
80   IL_0049:  ldstr "Negativo"
81   IL_004e:  call void class [mscorlib]System.Console::
        WriteLine(string)
82   IL_0053:  ldstr "Desja finalizar? (S/N) "
83   IL_0058:  call void class [mscorlib]System.Console::Write(
        string)
84   IL_005d:  call string class [mscorlib]System.Console::
        ReadLine()
85   IL_0062:  call char class [mscorlib]System.Convert::ToChar(
        string)
86   IL_0067:  stloc.2
87   IL_0068:  ldloc.2
88   IL_0069:  ldc.i4.s 0x53
89   IL_006b:  bne.un IL_0072
90
91   IL_0070:  ldc.i4.0
```

```
92    IL_0071:  stloc.0
93    IL_0072:  ldloc.0
94    IL_0073:  ldc.i4.1
95    IL_0074:  beq IL_0007
96
97    IL_0079:  ret
98      } // end of method micro07::Main
99
100   } // end of class micro07
```

7.20 Micro 08

Listing 67: Decide se um número é maior ou menor que 10(JavaScript)

```
1 var numero;
2 numero = 1;
3 while(numero != 0){
4 numero = prompt("Digite um numero: ");
5 if(numero > 10)
6 document.write("o numero " +numero+ " é maior que 10");
7 else
8 document.write("o numero " +numero+ " é menor que 10");
9 }
```

Listing 68: Decide se um número é maior ou menor que 10(C#)

```
1 using System;
2 class micro08 {
3 public static void Main(){
4 int numero;
5 numero = 1;
6 while ( numero != 0 ) {
7 Console.Write("Digite um número: ");
8 numero = Convert.ToInt32(Console.ReadLine());
9 if ( numero > 10 )
10 Console.WriteLine("O número " + numero + " é maior que 10");
11
12 else
13 Console.WriteLine("O número " + numero + " é menor que 10");
14 }
15 }
16 }
```

Listing 69: Decide se um número é maior ou menor que 10(CIL)

```
1  .assembly extern mscorlib
2  {
3    .ver 4:0:0:0
4    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5  }
6  .assembly 'micro08'
7  {
8    .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9      01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
         T..WrapNonEx
10     63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
         ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver  0:0:0:0
14 }
15 .module micro08.exe // GUID = {24F28363-2C63-4D9B-BB72-8
     CF2C7E16F06}
16
17
18   .class private auto ansi beforefieldinit micro08
19     extends [mscorlib]System.Object
20   {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24            instance default void '.ctor' ()  cil managed
25     {
26         // Method begins at RVA 0x2050
27   // Code size 7 (0x7)
28   .maxstack 8
29   IL_0000:  ldarg.0
30   IL_0001:  call instance void object::'.ctor'()
31   IL_0006:  ret
32     } // end of method micro08::.ctor
33
34     // method line 2
35     .method public static hidebysig
36            default void Main ()  cil managed
37     {
38         // Method begins at RVA 0x2058
39   .entrypoint
40   // Code size 100 (0x64)
```

```
41  .maxstack 3
42  .locals init (
43    int32 V_0)
44  IL_0000:  ldc.i4.1
45  IL_0001:  stloc.0
46  IL_0002:  br IL_005d
47
48  IL_0007:  ldstr bytearray (
49  44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00  // D.i.g.i
       .t.e. .u.
50  6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // m. .n
       ...m.e.r.o.
51  3a 00 20 00 01 )                                 // :. ..
52
53  IL_000c:  call void class [mscorlib]System.Console::Write(
       string)
54  IL_0011:  call string class [mscorlib]System.Console::
       ReadLine()
55  IL_0016:  call int32 class [mscorlib]System.Convert::
       ToInt32(string)
56  IL_001b:  stloc.0
57  IL_001c:  ldloc.0
58  IL_001d:  ldc.i4.s 0x0a
59  IL_001f:  ble IL_0043
60
61  IL_0024:  ldstr bytearray (
62  4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // O. .n
       ...m.e.r.o.
63  20 00 01 )                                       //  ..
64
65  IL_0029:  ldloc.0
66  IL_002a:  box [mscorlib]System.Int32
67  IL_002f:  ldstr bytearray (
68  20 00 e9 00 20 00 6d 00 61 00 69 00 6f 00 72 00  //  ... .m
       .a.i.o.r.
69  20 00 71 00 75 00 65 00 20 00 31 00 30 00 01 )   //  .q.u.e
       . .1.0..
70
71  IL_0034:  call string string::Concat(object, object, object
       )
72  IL_0039:  call void class [mscorlib]System.Console::
       WriteLine(string)
73  IL_003e:  br IL_005d
74
75  IL_0043:  ldstr bytearray (
```

```
76   4f 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // O. .n
       ...m.e.r.o.
77   20 00 01 )                                       //  ..
78
79   IL_0048:  ldloc.0
80   IL_0049:  box [mscorlib]System.Int32
81   IL_004e:  ldstr bytearray (
82   20 00 e9 00 20 00 6d 00 65 00 6e 00 6f 00 72 00  //  ... .m
       .e.n.o.r.
83   20 00 71 00 75 00 65 00 20 00 31 00 30 00 01 )   //  .q.u.e
       . .1.0..
84
85   IL_0053:  call string string::Concat(object, object, object
       )
86   IL_0058:  call void class [mscorlib]System.Console::
       WriteLine(string)
87   IL_005d:  ldloc.0
88   IL_005e:  brtrue IL_0007
89
90   IL_0063:  ret
91     } // end of method micro08::Main
92
93   } // end of class micro08
```

7.21 Micro 09

Listing 70: Cálculo de preços(JavaScript)

```
1 var preco, venda, novo_preco;
2 novo_preco = 0;
3 preco = prompt("Digite o preco: ");
4 venda = prompt("Digite a venda: ");
5 if(venda < 500 | preco < 30)
6 novo_preco = preco + 10/100 * preco;
7 eles if((venda >= 500 & venda < 1200) | (preco >= 30 & preco
      < 80))
8 novo_preco = preco + 15/10 * preco;
9 else if(venda >= 1200 | preco >= 80)
10 novo_preco = preco - 20/100 * preco;
11 document.write("O novo preco é " +novo_preco);
```

Listing 71: Cálculo de preços(C#)

```
1 using System;
2 class micro09 {
```

```
3 public static void Main(){
4 double preco, venda, novo_preco;
5 novo_preco = 0;
6 Console.Write("Digite o preco: ");
7 preco = Convert.ToDouble(Console.ReadLine());
8 Console.Write("Digite a venda: ");
9 venda = Convert.ToDouble(Console.ReadLine());
10 if ( (venda < 500) || ( preco < 30 ) )
11 novo_preco = preco + 10/100 * preco;
12 else if ( (venda >= 500 && venda < 1200) || (preco >= 30 &&
     preco < 80) )
13 novo_preco = preco + 15/100 * preco;
14 else if ( venda >= 1200 || preco >= 80 )
15 novo_preco = preco - 20/100 * preco;
16 Console.Write("O novo preco e " + novo_preco);
17 }
18 }
```

Listing 72: Cálculo de preços(CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro09'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
        T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
        ceptionThrows.
11
12  .hash algorithm 0x00008004
13  .ver  0:0:0:0
14 }
15 .module micro09.exe // GUID = {18F4C6D1-F37A-470C-B18B-
     EFA7F29E4AAB}
16
17
18  .class private auto ansi beforefieldinit micro09
19    extends [mscorlib]System.Object
20  {
```

```
21
22      // method line 1
23      .method public hidebysig specialname rtspecialname
24              instance default void '.ctor' ()  cil managed
25      {
26          // Method begins at RVA 0x2050
27    // Code size 7 (0x7)
28    .maxstack 8
29    IL_0000:  ldarg.0
30    IL_0001:  call instance void object::'.ctor'()
31    IL_0006:  ret
32      } // end of method micro09::.ctor
33
34      // method line 2
35      .method public static hidebysig
36              default void Main ()  cil managed
37      {
38          // Method begins at RVA 0x2058
39    .entrypoint
40    // Code size 246 (0xf6)
41    .maxstack 3
42    .locals init (
43      float64 V_0,
44      float64 V_1,
45      float64 V_2)
46    IL_0000:  ldc.r8 0.
47    IL_0009:  stloc.2
48    IL_000a:  ldstr "Digite o preco: "
49    IL_000f:  call void class [mscorlib]System.Console::Write(
        string)
50    IL_0014:  call string class [mscorlib]System.Console::
        ReadLine()
51    IL_0019:  call float64 class [mscorlib]System.Convert::
        ToDouble(string)
52    IL_001e:  stloc.0
53    IL_001f:  ldstr "Digite a venda: "
54    IL_0024:  call void class [mscorlib]System.Console::Write(
        string)
55    IL_0029:  call string class [mscorlib]System.Console::
        ReadLine()
56    IL_002e:  call float64 class [mscorlib]System.Convert::
        ToDouble(string)
57    IL_0033:  stloc.1
58    IL_0034:  ldloc.1
59    IL_0035:  ldc.r8 500.
```

```
60   IL_003e:  blt IL_0052
61
62   IL_0043:  ldloc.0
63   IL_0044:  ldc.r8 30.
64   IL_004d:  bge.un IL_0065
65
66   IL_0052:  ldloc.0
67   IL_0053:  ldc.r8 0.
68   IL_005c:  ldloc.0
69   IL_005d:  mul
70   IL_005e:  add
71   IL_005f:  stloc.2
72   IL_0060:  br IL_00e0
73
74   IL_0065:  ldloc.1
75   IL_0066:  ldc.r8 500.
76   IL_006f:  blt.un IL_0083
77
78   IL_0074:  ldloc.1
79   IL_0075:  ldc.r8 1200.
80   IL_007e:  blt IL_00a1
81
82   IL_0083:  ldloc.0
83   IL_0084:  ldc.r8 30.
84   IL_008d:  blt.un IL_00b4
85
86   IL_0092:  ldloc.0
87   IL_0093:  ldc.r8 80.
88   IL_009c:  bge.un IL_00b4
89
90   IL_00a1:  ldloc.0
91   IL_00a2:  ldc.r8 0.
92   IL_00ab:  ldloc.0
93   IL_00ac:  mul
94   IL_00ad:  add
95   IL_00ae:  stloc.2
96   IL_00af:  br IL_00e0
97
98   IL_00b4:  ldloc.1
99   IL_00b5:  ldc.r8 1200.
100  IL_00be:  bge IL_00d2
101
102  IL_00c3:  ldloc.0
103  IL_00c4:  ldc.r8 80.
104  IL_00cd:  blt.un IL_00e0
```

```
105
106   IL_00d2:  ldloc.0
107   IL_00d3:  ldc.r8 0.
108   IL_00dc:  ldloc.0
109   IL_00dd:  mul
110   IL_00de:  sub
111   IL_00df:  stloc.2
112   IL_00e0:  ldstr "O novo preco e "
113   IL_00e5:  ldloc.2
114   IL_00e6:  box [mscorlib]System.Double
115   IL_00eb:  call string string::Concat(object, object)
116   IL_00f0:  call void class [mscorlib]System.Console::Write(
        string)
117   IL_00f5:  ret
118     } // end of method micro09::Main
119
120   } // end of class micro09
```

7.22 Micro 10

Listing 73: Calcula o fatorial de um número(JavaScript)

```
1 function fatorial(n){
2 if (n <= 0)
3 return 1;
4 else
5 return n * fatorial(n-1);
6 }
7
8 var numero, fat;
9 numero = prompt("Digite um numero: ");
10 fat = fatorial(numero);
11 document.write("O fatorial de " +numero+ " é " +fat);
```

Listing 74: Calcula o fatorial de um número(C#)

```
1 using System;
2 class micro10 {
3 public static int fatorial (int n){
4 if ( n <= 0)
5 return 1;
6 else
7 return n * fatorial(n-1);
8
9 }
```

```
10
11 public static void Main(){
12 int numero, fat;
13
14 Console.Write("Digite um número: ");
15 numero = Convert.ToInt32(Console.ReadLine());
16 fat = fatorial(numero);
17 Console.Write("O fatorial de ");
18 Console.Write(numero);
19 Console.Write(" é ");
20 Console.WriteLine(fat);
21 }
22 }
```

Listing 75: Calcula o fatorial de um número(CIL)

```
1 .assembly extern mscorlib
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'micro10'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
        T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
        ceptionThrows.
11
12  .hash algorithm 0x00008004
13  .ver  0:0:0:0
14 }
15 .module micro10.exe // GUID = {BBA21109-28D2-406B-8D4B-
    B6D586DD5E21}
16
17
18  .class private auto ansi beforefieldinit micro10
19    extends [mscorlib]System.Object
20  {
21
22    // method line 1
23    .method public hidebysig specialname rtspecialname
24           instance default void '.ctor' ()  cil managed
```

```
25    {
26        // Method begins at RVA 0x2050
27  // Code size 7 (0x7)
28  .maxstack 8
29  IL_0000:  ldarg.0
30  IL_0001:  call instance void object::'.ctor'()
31  IL_0006:  ret
32    } // end of method micro10::.ctor
33
34    // method line 2
35    .method public static hidebysig
36           default int32 fatorial (int32 n)  cil managed
37    {
38        // Method begins at RVA 0x2058
39  // Code size 20 (0x14)
40  .maxstack 8
41  IL_0000:  ldarg.0
42  IL_0001:  ldc.i4.0
43  IL_0002:  bgt IL_0009
44
45  IL_0007:  ldc.i4.1
46  IL_0008:  ret
47  IL_0009:  ldarg.0
48  IL_000a:  ldarg.0
49  IL_000b:  ldc.i4.1
50  IL_000c:  sub
51  IL_000d:  call int32 class micro10::fatorial(int32)
52  IL_0012:  mul
53  IL_0013:  ret
54    } // end of method micro10::fatorial
55
56    // method line 3
57    .method public static hidebysig
58           default void Main ()  cil managed
59    {
60        // Method begins at RVA 0x2070
61  .entrypoint
62  // Code size 61 (0x3d)
63  .maxstack 1
64  .locals init (
65    int32 V_0,
66    int32 V_1)
67  IL_0000:  ldstr bytearray (
68  44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00  // D.i.g.i
       .t.e. .u.
```

```
69   6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // m. .n
       ...m.e.r.o.
70   3a 00 20 00 01 )                                 // :. ..
71
72   IL_0005:  call void class [mscorlib]System.Console::Write(
       string)
73   IL_000a:  call string class [mscorlib]System.Console::
       ReadLine()
74   IL_000f:  call int32 class [mscorlib]System.Convert::
       ToInt32(string)
75   IL_0014:  stloc.0
76   IL_0015:  ldloc.0
77   IL_0016:  call int32 class micro10::fatorial(int32)
78   IL_001b:  stloc.1
79   IL_001c:  ldstr "O fatorial de "
80   IL_0021:  call void class [mscorlib]System.Console::Write(
       string)
81   IL_0026:  ldloc.0
82   IL_0027:  call void class [mscorlib]System.Console::Write(
       int32)
83   IL_002c:  ldstr bytearray (
84   20 00 e9 00 20 00 01 )                           //  ... ..
85
86   IL_0031:  call void class [mscorlib]System.Console::Write(
       string)
87   IL_0036:  ldloc.1
88   IL_0037:  call void class [mscorlib]System.Console::
       WriteLine(int32)
89   IL_003c:  ret
90     } // end of method micro10::Main
91
92   } // end of class micro10
```

7.23 Micro 11

Listing 76: Decide se um número é positivo zero ou negativo com auxílio de uma função(JavaScript)

```
1 function verifica(n){
2 var res;
3 if( n > 0)
4 res = 1;
5 else if(n < 0)
6 res = -1;
7 eles
```

```
8 res = 0;
9 return res;
10 }
11
12 var numero, x;
13 numero = prompt("Digite um numero: ");
14 x = verifica(numero);
15 if(x == 1)
16 document.write("Numero positivo");
17 else if(x == 0)
18 document.write("Zero");
19 else
20 document.write("Numero negativo");
```

Listing 77: Decide se um número é positivo zero ou negativo com auxílio de uma função(C#)

```
1 using System;
2 class micro11 {
3 public static int verifica(int n){
4 int res;
5 if ( n > 0 )
6 res = 1;
7 else if ( n < 0)
8 res = -1;
9 else
10 res = 0;
11 return res;
12 }
13 public static void Main(){
14 int numero, x;
15
16 Console.Write("Digite um número: ");
17 numero = Convert.ToInt32(Console.ReadLine());
18 x = verifica(numero);
19 if ( x == 1 )
20 Console.WriteLine("Numero positivo");
21 else if ( x == 0)
22 Console.WriteLine("Zero");
23 else Console.WriteLine("Numero negativo");
24 }
25 }
```

Listing 78: Decide se um número é positivo zero ou negativo com auxílio de uma função(CIL)

```
1  .assembly extern mscorlib
2  {
3    .ver 4:0:0:0
4    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5  }
6  .assembly 'micro11'
7  {
8    .custom instance void class [mscorlib]System.Runtime.
       CompilerServices.RuntimeCompatibilityAttribute::'.ctor
       '() =  (
9      01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78   // ....
         T..WrapNonEx
10     63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01       ) //
         ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver  0:0:0:0
14 }
15 .module micro11.exe // GUID = {5A9256D0-682E-43BF-AB66-
     CE37CDFF3D0A}
16
17
18   .class private auto ansi beforefieldinit micro11
19     extends [mscorlib]System.Object
20   {
21
22     // method line 1
23     .method public hidebysig specialname rtspecialname
24            instance default void '.ctor' ()  cil managed
25     {
26         // Method begins at RVA 0x2050
27   // Code size 7 (0x7)
28   .maxstack 8
29   IL_0000:  ldarg.0
30   IL_0001:  call instance void object::'.ctor'()
31   IL_0006:  ret
32     } // end of method micro11::.ctor
33
34     // method line 2
35     .method public static hidebysig
36            default int32 verifica (int32 n)  cil managed
37     {
38         // Method begins at RVA 0x2058
```

```
39  // Code size 32 (0x20)
40  .maxstack 2
41  .locals init (
42    int32 V_0)
43  IL_0000:  ldarg.0
44  IL_0001:  ldc.i4.0
45  IL_0002:  ble IL_000e
46
47  IL_0007:  ldc.i4.1
48  IL_0008:  stloc.0
49  IL_0009:  br IL_001e
50
51  IL_000e:  ldarg.0
52  IL_000f:  ldc.i4.0
53  IL_0010:  bge IL_001c
54
55  IL_0015:  ldc.i4.m1
56  IL_0016:  stloc.0
57  IL_0017:  br IL_001e
58
59  IL_001c:  ldc.i4.0
60  IL_001d:  stloc.0
61  IL_001e:  ldloc.0
62  IL_001f:  ret
63    } // end of method micro11::verifica
64
65    // method line 3
66    .method public static hidebysig
67           default void Main ()  cil managed
68    {
69        // Method begins at RVA 0x2084
70  .entrypoint
71  // Code size 82 (0x52)
72  .maxstack 2
73  .locals init (
74    int32 V_0,
75    int32 V_1)
76  IL_0000:  ldstr bytearray (
77  44 00 69 00 67 00 69 00 74 00 65 00 20 00 75 00  // D.i.g.i
       .t.e. .u.
78  6d 00 20 00 6e 00 fa 00 6d 00 65 00 72 00 6f 00  // m. .n
       ...m.e.r.o.
79  3a 00 20 00 01 )                                 // :. ..
80
81  IL_0005:  call void class [mscorlib]System.Console::Write(
```

```
      string)
82    IL_000a:  call string class [mscorlib]System.Console::
         ReadLine()
83    IL_000f:  call int32 class [mscorlib]System.Convert::
         ToInt32(string)
84    IL_0014:  stloc.0
85    IL_0015:  ldloc.0
86    IL_0016:  call int32 class micro11::verifica(int32)
87    IL_001b:  stloc.1
88    IL_001c:  ldloc.1
89    IL_001d:  ldc.i4.1
90    IL_001e:  bne.un IL_0032
91
92    IL_0023:  ldstr "Numero positivo"
93    IL_0028:  call void class [mscorlib]System.Console::
         WriteLine(string)
94    IL_002d:  br IL_0051
95
96    IL_0032:  ldloc.1
97    IL_0033:  brtrue IL_0047
98
99    IL_0038:  ldstr "Zero"
100   IL_003d:  call void class [mscorlib]System.Console::
         WriteLine(string)
101   IL_0042:  br IL_0051
102
103   IL_0047:  ldstr "Numero negativo"
104   IL_004c:  call void class [mscorlib]System.Console::
         WriteLine(string)
105   IL_0051:  ret
106     } // end of method micro11::Main
107
108   } // end of class micro11
```

8 Construção do Compilador

Esta seção apresenta as etapas de implementação necessárias para a construção do compilador.

8.1 Analisador Léxico

A análise léxica é a primeira etapa do processo de compilação. Nesta estapa, o código fonte do programa a ser compilado é varrido caractere por caractere e cada e palavra reservada, constante, identificador ou outras palavras que pertencem a linguagem de programação, são traduzidas para uma sequência de símbolos chamados "tokens".

Além de extrair e classificar tokens, a análise léxica também elimina espaços em branco e comentários e identifica erros léxicos, simplificando a próxima etapa do processo de compilação, a análise sintática.

Neste trabalho será usado o gerador de analisadores lexicais da linguagem Ocaml. o Ocamllex. Este gerador produz o código do analisador léxico a partir de uma especificação lexical. Tal especificação deve conter a definição dos tokens da linguagem (representados por expressões regulares) e a ação tomada para cada token.

8.2 Especificação Lexical para MiniJavaScript

O código em Ocaml a seguir apresenta uma especificação lexical simplificada para a linguagem Javascript. Este código deve ser salvo como um arquivo no formato ".mll".

Listing 79: Especificação Lexical

```
1
2 {
3   open Lexing
4   open Printf
5
6
7   let incr_num_linha lexbuf =
8     let pos = lexbuf.lex_curr_p in
9      lexbuf.lex_curr_p <- { pos with
10        pos_lnum = pos.pos_lnum + 1;
11        pos_bol = pos.pos_cnum;
12     }
```

```
13
14   let msg_erro lexbuf c =
15     let pos = lexbuf.lex_curr_p in
16     let lin = pos.pos_lnum
17     and col = pos.pos_cnum - pos.pos_bol - 1 in
18     sprintf "%d-%d: caracter desconhecido %c" lin col c
19
20   let erro lin col msg =
21     let mensagem = sprintf "%d-%d: %s" lin col msg in
22        failwith mensagem
23
24   let pos_atual lexbuf = lexbuf.lex_start_p
25
26  type tokens =
27 | APAR
28 | FPAR
29 | ACHAVE
30 | FCHAVE
31 | ACOLCH
32 | FCOLCH
33 | MAIS
34 | MENOS
35 | MULT
36 | DIV
37 | MOD
38 | POT
39 | MAIOR
40 | MENOR
41 | ATRIB
42 | IGUAL
43 | MENORIGUAL
44 | MAIORIGUAL
45 | INCR
46 | DECR
47 | AND
48 | OR
49 | NOT
50 | VIRG
51 | PONTOVIRG
52 | DIF
53 | FOR
54 | IF
55 | ELSE
56 | WHILE
57 | SWITCH
```

```
58 | CASE
59 | BREAK
60 | DEFAULT
61 | CONSOLELOG
62 | FUNCTION
63 | RETURN
64 | LITINT of int
65 | LITFLOAT of float
66 | LITSTRING of string
67 | LITBOOL of bool
68 | LITCHAR of char
69 | VAR
70 | ID of string
71 | DOISPTO
72 | DO
73 | PROMPT
74 | INT
75 | FLOAT
76 | STRING
77 | BOOL
78 | TRUE
79 | FALSE
80 | CHAR
81 | VOID
82 %token EOF
83 }
84
85 let digito = ['0' - '9']
86 let int = '-'? digito+
87 let float = '-' ? digito+ '.'? digito+
88
89 let letra = ['a' - 'z' 'A' - 'Z']
90 let char = ''' (letra | digito) '''
91 let identificador = letra ( letra | digito | '_')*
92
93 let brancos = [' ' '\t']+
94 let novalinha = '\r' | '\n' | "\r\n"
95
96 let comentario = "//" [^ '\r' '\n' ]*
97
98 rule token = parse
99   brancos    { token lexbuf }
100 | novalinha  { incr_num_linha lexbuf; token lexbuf }
101 | comentario { token lexbuf }
102 | "/*"          { comentario_bloco 0 lexbuf }
```

```
103 | '('             { APAR   }
104 | ')'             { FPAR   }
105 | '{'             { ACHAVE  }
106 | '}'             { FCHAVE  }
107 | '['             { ACOLCH  }
108 | ']'             { FCOLCH  }
109 | '+'             { MAIS  }
110 | '-'             { MENOS  }
111 | '*'             { MULT  }
112 | '/'             { DIV  }
113 | '%'             { MOD  }
114 | "**"            { POT  }
115 | '='             { ATRIB  }
116 | '>'             { MAIOR  }
117 | '<'             { MENOR  }
118 | "<="            { MENORIGUAL  }
119 | ">="            { MAIORIGUAL  }
120 | "=="            { IGUAL  }
121 | "++"            { INCR  }
122 | "--"            { DECR  }
123 | "&&"            { AND  }
124 | "||"            { OR  }
125 | '!'             { NOT  }
126 | "!="            { DIF  }
127 | ','             { VIRG  }
128 | ';'             { PONTOVIRG  }
129 | ':'             { DOISPTO  }
130 | "for"           { FOR  }
131 | "if"            { IF  }
132 | "else"          { ELSE  }
133 | "switch"        { SWITCH  }
134 | "case"          { CASE  }
135 | "break"         { BREAK  }
136 | "default"       { DEFAULT  }
137 | "console.log"   { CONSOLELOG  }
138 | "function"      { FUNCTION  }
139 | "return"        { RETURN  }
140 | "var"           { VAR  }
141 | "while"         { WHILE  }
142 | "int"           { INT  }
143 | "float"         { FLOAT  }
144 | "char"          { CHAR  }
145 | "string"        { STRING  }
146 | "bool"          { BOOL  }
147 | "void"          { VOID }
```

```
148 | "do"           { DO  }
149 | "prompt"       { PROMPT  }
150 | "true"         { TRUE  }
151 | "false"       { FALSE  }
152 | identificador as id   { ID (id, pos_atual lexbuf) }
153 | int as n { LITINT (int_of_string n) }
154 | float as f { LITFLOAT (float_of_string f)}
155 | char as c { LITCHAR (c.[1])}
156 | '"'         { let buffer = Buffer.create 1 in
157                 let str = leia_string buffer lexbuf in
158                  LITSTRING str}
159 | _  { raise (Erro ("Caracter desconhecido: " ^ Lexing.lexeme
      lexbuf)) }
160 | eof   { EOF }
161
162 and leia_string lin col buffer = parse
163 '"' { Buffer.contents buffer}
164 | "\\t" { Buffer.add_char buffer '\t';
165 leia_string lin col buffer lexbuf }
166 | "\\n" { Buffer.add_char buffer '\n';
167 leia_string lin col buffer lexbuf }
168 | '\\' '"' { Buffer.add_char buffer '"';
169 leia_string lin col buffer lexbuf }
170 | '\\' '\\' { Buffer.add_char buffer '\\';
171 leia_string lin col buffer lexbuf }
172 | _ as c { Buffer.add_char buffer c;
173 leia_string lin col buffer lexbuf }
174 | eof { erro lin col "A string n~ao foi fechada"}
175
176 and comentario_bloco n = parse
177 "*/"        { if n=0 then token lexbuf
178                 else comentario_bloco (n-1) lexbuf }
179 | "/*"      { failwith "Comentarios aninhados nao permitidos"
      }
180 | novalinha { incr_num_linha lexbuf;
181                 comentario_bloco n lexbuf }
182 | _         { comentario_bloco n lexbuf }
183 | eof       { failwith "Coment ário não fechado" }
```

Listing 80: Carregadorl

```
1 {
2 \#load "lexer.cmo";;
3
4 let rec tokens lexbuf =
```

```
5   let tok = Lexer.token lexbuf in
6   match tok with
7   | Lexer.EOF -> [Lexer.EOF]
8   | _ -> tok :: tokens lexbuf
9 ;;
10
11 let lexico str =
12   let lexbuf = Lexing.from_string str in
13   tokens lexbuf
14 ;;
15
16 let lex arq =
17   let ic = open_in arq in
18   let lexbuf = Lexing.from_channel ic in
19   let toks = tokens lexbuf in
20   let _ = close_in ic in
21   toks
```

8.3 Compilação do Analisador Léxico

A geração do código do analisador léxico usando o Ocamllex é feita utilizando os seguintes comandos:

```
> ocamllex lexicoJS.mll
> ocamlc -c lexicoJS.ml
```

Onde "lexicoJS.mll"é o nome do arquivo que contém a especificação lexical. Após utilizar estes comandos será gerado um arquivo "lexicoJS.ml", o qual de fato apresenta o código do analisador léxico.

8.4 Teste do Analisador Léxico

Para testar o funcionamento do analisador léxico foi criado um arquivo "teste .js"contendo um código em JavaScript. O analisador léxico varrerá este arquivo, caractere por caractere, e produzirá uma sequência de tokens. Para isso, utiliza-se os comandos:

```
> rlwrap ocaml
>#use "carregador.ml";;
> lex "teste.js";;
```

```
1 {
2 function factorial(num)
```

```
{
    // coment
    if (num < 0) {
        return -1;
    }
    else if (num == 0) {
        return 1;
    }
    else {
        return (num * factorial(num - 1));
    }
}

var result = factorial(8);
document.write(result);
```

Para o código em JavaScript acima foi gerada a seguinte sequência de tokens:

```
- : LexicoJS.tokens list =
[LexicoJS.FUNCTION; LexicoJS.ID "factorial"; LexicoJS.APAR;
 LexicoJS.ID "num"; LexicoJS.FPAR; LexicoJS.ACHAVE; LexicoJS.
    IF;
 LexicoJS.APAR; LexicoJS.ID "num"; LexicoJS.MENOR; LexicoJS.
    LITINT 0;
 LexicoJS.FPAR; LexicoJS.ACHAVE; LexicoJS.RETURN; LexicoJS.
    MENOS;
 LexicoJS.LITINT 1; LexicoJS.PONTOVIRG; LexicoJS.FCHAVE;
    LexicoJS.ELSE;
 LexicoJS.IF; LexicoJS.APAR; LexicoJS.ID "num"; LexicoJS.
    IGUAL;
 LexicoJS.LITINT 0; LexicoJS.FPAR; LexicoJS.ACHAVE; LexicoJS.
    RETURN;
 LexicoJS.LITINT 1; LexicoJS.PONTOVIRG; LexicoJS.FCHAVE;
    LexicoJS.ELSE;
 LexicoJS.ACHAVE; LexicoJS.RETURN;LexicoJS.APAR; LexicoJS.ID
    "num";
 LexicoJS.MULT; LexicoJS.ID "factorial"; LexicoJS.APAR;
    LexicoJS.ID "num";
 LexicoJS.MENOS;LexicoJS.LITINT 1; LexicoJS.FPAR; LexicoJS.
    FPAR;
 LexicoJS.PONTOVIRG; LexicoJS.FCHAVE; LexicoJS.FCHAVE;
    LexicoJS.VAR;
 LexicoJS.ID "result"; LexicoJS.ATRIB; LexicoJS.ID "factorial
    ";
 LexicoJS.APAR; LexicoJS.LITINT 8; LexicoJS.FPAR; LexicoJS.
```

```
    PONTOVIRG;
  LexicoJS.CONSOLELOG; LexicoJS.APAR; LexicoJS.ID "result";
    LexicoJS.FPAR;
  LexicoJS.PONTOVIRG; LexicoJS.EOF]
```

8.5 Analisador Sintático

A análise sintática é a segunda etapa do processo de compilação. O analisador sintático, também conhecido como parser, recebe os tokens gerados pela análise léxica e sua tarefa principal é determinar se as sequências de tokens recebidas formam sentenças válidas para a linguagem, ou seja, estão de acordo com a sintaxe da linguagem. Caso a combinação de tokens esteja de acordo com a sintaxe, o analisador sintático produzirá como saída uma árvore.

Para implementar um analisador sintático na linguagem Ocaml para a linguagem JavaScript foi utilziado o Menhir, um gerador de analisadores sintáticos LR(1). Primeiramente foram definidas todas as sentenças válidas para a linguagem JavaScript (aquivo parser.mly). No arquivo ast.ml é definida uma árvore sintática abstrata a partir das regras de produção da linguagem. Também foi necessário definir mensagens de erro para possíveis erros de sintaxe no arquivo parser.msg, o qual pode ser facilmente gerado pelo Menhir. Além disso, foram feitas algumas alterações no analisador léxico codificado nas seções anteriores.

8.6 Especificação Sintática para MiniJavaScript

Os arquivos e os comandos necessários para a compilação e execução do analisador sintático são mostrados a seguir.

Listing 81: Arquivo parser.mly

```
1
2 %{
3   open Lexing
4   open Ast
5 %}
6
7 %token  MAIN
8 %token  APAR
9 %token  FPAR
```

```
%token  ACHAVE
%token  FCHAVE
%token  ACOLCH
%token  FCOLCH
%token  MAIS
%token  MENOS
%token  MULT
%token  DIV
%token  MOD
%token  POT
%token  MAIOR
%token  MENOR
%token  ATRIB
%token  IGUAL
%token  MENORIGUAL
%token  MAIORIGUAL
%token  INCR
%token  DECR
%token  AND
%token  OR
%token  NOT
%token  VIRG
%token  PONTOVIRG
%token  DIF
%token  FOR
%token  IF
%token  ELSE
%token  WHILE
%token  SWITCH
%token  CASE
%token  BREAK
%token  DEFAULT
%token  CONSOLELOG
%token  FUNCTION
%token  RETURN
%token <int> LITINT
%token <float> LITFLOAT
%token <string> LITSTRING
%token <bool> LITBOOL
%token <char> LITCHAR
%token  VAR
%token <string> ID
%token  DOISPTO
%token  DO
%token  PROMPT
```

```
55 %token  INT
56 %token  FLOAT
57 %token  STRING
58 %token  BOOL
59 %token  TRUE
60 %token  FALSE
61 %token  CHAR
62 %token  VOID
63 %token EOF
64
65
66 %left OR
67 %left AND
68 %left IGUAL DIF
69 %left MAIOR MAIORIGUAL MENOR MENORIGUAL
70 %left MAIS MENOS
71 %left MULT DIV MOD
72 %right POT
73 %right NOT
74
75
76 %start <Ast.prog> prog
77
78 %%
79
80 prog:
81     | vdb=var_decl* fd=func_decl* stmb=stm_block  EOF { Prog
          (List.flatten vdb,fd,stmb) }
82     ;
83
84 var_decl:
85     | VAR ids = separated_nonempty_list(VIRG, ID) DOISPTO t=
          tp_primitivo PONTOVIRG {  List.map (fun id -> DecVar(
          id,t)) ids }
86     ;
87
88
89 tp_primitivo:
90     | INT { TipoInteiro }
91     | FLOAT { TipoReal }
92     | CHAR { TipoCaractere }
93     | BOOL { TipoBooleano }
94     | VOID { TipoVoid }
95     ;
96
```

```
97  stm_block:
98      | stms=stm_list* { (stms)}
99      ;
100
101 stm_list:
102     | stm=stm_attr {stm}
103     | stm=stm_fcall {stm}
104     | stm=stm_ret {stm}
105     | stm=stm_if {stm}
106     | stm=stm_while {stm}
107    (* | stm=stm_for {stm}*)
108     | stm=stm_print {stm}
109     (*| stm=stm_read {stm}*)
110     | stm=stm_switch {stm}
111     | stm = stm_declara_var { stm }
112     ;
113
114
115 stm_declara_var:
116     VAR ids=separated_nonempty_list(VIRG, ID) DOISPTO t=
           tp_primitivo PONTOVIRG { StmVarDecl(List.map(fun id ->
            VarDecl(id, t)) ids) }
117     ;
118
119 stm_fcall:
120     | exp=fcall PONTOVIRG {Chamada exp}
121    | v=expr ATRIB exp=fcall PONTOVIRG { ChamadaRec(v,exp)}
122     ;
123
124 stm_ret:
125     | RETURN ex=expr? PONTOVIRG { Retorne(ex)}
126     ;
127
128 variable:
129     | id=ID { Var(id) }
130     | v=variable ACOLCH e=expr FCOLCH {VarElement(v,e)}
131     ;
132
133
134 stm_attr:
135     | v=expr ATRIB e=expr PONTOVIRG { Attrib(v,e) }
136     ;
137
138 stm_if:
```

```
139     | IF APAR e=expr FPAR ACHAVE stms=stm_list* FCHAVE senao=
          stm_else? { Se(e,stms,senao)}
140  (*   | IF APAR e=expr FPAR stm=stm_list senao=stm_else { Se(
       e,[stm],senao)}*)
141     ;
142
143 stm_else:
144     | ELSE ACHAVE stm=stm_list* FCHAVE { stm }
145    (* | ELSE stm=smt_list {[stm]}
146       /*| ELSE IF APAR exp=expressao FPAR ACHAVE stms=
             statement* FCHAVE another=stm_else? { StmElseIf(exp,
              stms, another) }
147     | ELSE IF APAR exp=expressao FPAR st=statement another=
          stm_else? { StmElseIf(exp, [st], another) }*/*)
148     ;
149
150 stm_switch:
151     | SWITCH APAR id=ID FPAR ACHAVE c=case+ DEFAULT DOISPTO
          stms=stm_list* FCHAVE {Escolha(id,c,stms) }
152     ;
153
154 case:
155     | CASE c=LITCHAR DOISPTO stms=stm_list* BREAK {CaseChar(c
          ,stms)}
156     | CASE i=LITINT DOISPTO stms=stm_list* BREAK {CaseInt(i,
          stms) }
157     ;
158
159 stm_while:
160     | WHILE APAR e=expr FPAR ACHAVE stm=stm_list* FCHAVE {
          Enquanto (e,stm) }
161     ;
162
163 (*
164 stm_for:
165     | FOR APAR lv=expr DE e1=expr ATE e2=expr p=passo FACA
          stm=stm_list* FIMPARA {Para (lv,e1,e2,p,stm) }
166     ;
167 *)
168
169 (*
170 stm_read:
171     | v=variable ATRIB PROMPT APAR e=expr FPAR PONTOVIRG {
          Leia (v,e)}
172     ;
```

```
173 *)
174 stm_print:
175     | CONSOLELOG APAR  stm=separated_nonempty_list(VIRG, expr
            ) FPAR PONTOVIRG {Escreva stm}
176     ;
177
178
179 expr:
180    | e1=expr o=op e2=expr { ExpOp(o,e1,e2) }
181    | f=fcall { f }
182    | s=LITSTRING { ExpString s}
183    | i=LITINT { ExpInt i}
184    | f=LITFLOAT { ExpFloat f}
185    | c=LITCHAR { ExpChar c}
186    | l=logico_value { ExpBool l}
187    | lv=variable { ExpVar lv }
188    /*| pos=NOT e=expr { ExpNot ( )e, pos)}*/
189    | APAR e=expr FPAR { e }
190    | MAIS v=variable { ExpMaisVar v }
191    ;
192
193
194
195 %inline op:
196      pos = MAIS { (Add, pos) }
197     | pos = MENOS { (Sub, pos) }
198     | pos = MULT { (Mult, pos) }
199     | pos = DIV { (Div, pos) }
200     | pos = MOD { (Mod, pos) }
201     | pos = OR { (Or, pos) }
202     | pos = AND { (And, pos) }
203     | pos = POT { (Pot, pos) }
204     | pos = MENOR { (Menor, pos) }
205     | pos = IGUAL { (Igual, pos) }
206     | pos = DIF { (Dif, pos) }
207     | pos = MAIOR { (Maior, pos) }
208     | pos =  MAIORIGUAL { (MaiorIgual, pos)}
209     | pos = MENORIGUAL { (MenorIgual, pos) }
210     ;
211
212
213 logico_value:
214   | pos=TRUE { (true,pos) }
215   | pos=FALSE  { (false,pos) }
216   ;
```

```
217
218 fcall:
219     | id=ID APAR args=fargs FPAR { ExpFunCall(id, args) }
220     ;
221
222 fargs:
223     | exprs=separated_list(VIRG, expr) {  List.map (fun expr
          -> expr) exprs}
224     ;
225 (*
226 fcall:
227     | id=ID APAR args=fargs? FPAR { ExpFunCall(id,(match args
          with
228                                                   | None ->
                                                        []
229                                                   | Some
                                                        fargs ->
                                                         fargs )
                                                       ) }
230     ;
231
232 fargs:
233     | exprs=separated_nonempty_list(VIRG, expr) {  List.map (
          fun expr -> expr) exprs}
234     ;*)
235
236 /*Declaração de funções*/
237 func_decl:
238     | FUNCTION id=ID APAR fp=fparams? FPAR fy=func_type fv=
          var_decl* fb=func_bloc
239   { FuncDecl {
240                       fn_id = id;
241                       fn_params = (match fp with
242                         | None -> []
243                         | Some args -> args;
244                      fn_tiporet = fy;
245                      fn_locais = List.flatten fv;
246                      fn_corpo = fb }
247                           }
248     ;
249
250 func_type:
251     | DOISPTO t=tp_primitivo { (t) }
252     ;
253 /* */
```

```
254
255 func_bloc:
256     | ACHAVE stm=stm_list* FCHAVE {(stm)}
257     ;
258
259
260 /*Parâmetro de funções*/
261 fparams:
262     | fparam=separated_nonempty_list(VIRG,fparam){ fparam }
263     ;
264
265 fparam:
266     | id=ID DOISPTO t=tp_primitivo {(id,t)}
267     ;
```

Listing 82: Arquivo main.ml

```
1  open Printf
2  open Lexing
3
4  open Ast
5  open ErroSint (* nome do módulo contendo as mensagens de erro
       *)
6
7  exception Erro_Sintatico of string
8
9  module S = MenhirLib.General (* Streams *)
10 module I = Parser.MenhirInterpreter
11
12 let posicao lexbuf =
13     let pos = lexbuf.lex_curr_p in
14     let lin = pos.pos_lnum
15     and col = pos.pos_cnum - pos.pos_bol - 1 in
16     sprintf "linha %d, coluna %d" lin col
17
18 (* [pilha checkpoint] extrai a pilha do autômato LR(1)
       contida em checkpoint *)
19
20 let pilha checkpoint =
21   match checkpoint with
22   | I.HandlingError amb -> I.stack amb
23   | _ -> assert false (* Isso não pode acontecer *)
24
25 let estado checkpoint : int =
26   match Lazy.force (pilha checkpoint) with
```

```
27   | S.Nil -> (* O parser está no estado inicial *)
28       0
29   | S.Cons (I.Element (s, _, _, _), _) ->
30       I.number s
31
32 let sucesso v = Some v
33
34 let falha lexbuf (checkpoint : Ast.prog I.checkpoint) =
35   let estado_atual = estado checkpoint in
36   let msg = message estado_atual in
37   raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
38                                         (Lexing.lexeme_start
                                             lexbuf) msg))
39
40 let loop lexbuf resultado =
41   let fornecedor = I.lexer_lexbuf_to_supplier LexicoJS.token
         lexbuf in
42   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
43
44
45 let parse_com_erro lexbuf =
46   try
47     Some (loop lexbuf (Parser.Incremental.prog lexbuf.
           lex_curr_p))
48   with
49   | LexicoJS.Erro msg ->
50       printf "Erro Lexico na %s:\n\t%s\n" (posicao lexbuf) msg
             ;
51       None
52   | Erro_Sintatico msg ->
53       printf "Erro sintático na %s %s\n" (posicao lexbuf) msg;
54       None
55
56 let parse s =
57   let lexbuf = Lexing.from_string s in
58   let ast = parse_com_erro lexbuf in
59   ast
60
61 let parse_arq nome =
62   let ic = open_in nome in
63   let lexbuf = Lexing.from_channel ic in
64   let result = parse_com_erro lexbuf in
65   let _ = close_in ic in
66   match result with
67   | Some ast -> ast
```

```
68   | None -> failwith "A analise sintatica falhou"
```

Listing 83: Arquivo ast.ml

```
1  open Lexing
2
3  type identificador = string
4
5  and  prog = Prog of var_decl *  (func_decl_list) *
       statements
6  and declaracao_algoritmo = DeclAlg
7  and var_decl = vars list
8  and vars = DecVar of identificador * tipo
9  and func_decl_list = ('expr func_decl) list
10 and func_decl = FuncDecl of decfun
11 and fparams = fparam list
12 and fparam =  identificador * tipo
13 and functype = tipo
14 and funcbloc =  stm_list list
15 and statements = stm_list list
16
17 and  decfun = {
18   fn_id: identificador pos;
19   fn_params: (identificador pos * tipo) list;
20   fn_tiporet: functype;
21   fn_locais: var_decl;
22   fn_corpo: 'expr funcbloc;
23
24 }
25
26
27 and stm_list  = Attrib of expressao  * expressao
28             |Chamada of expressao
29                 |ChamadaRec of expressao * expressao
30             |Retorne of expressao option
31             |Se of expressao * stm_list list * senao option
32             |Escolha of identificador * case list * stm_list
                  list
33             |Leia of variable * expressao
34             |Escreva of expressao list
35             |Enquanto of expressao * stm_list list
36                 |StmVarDecl of varDeclaracao list
37
38 and varDeclaracao = VarDecl of identificador pos * tipo
39 and tipo = TipoInteiro
```

```
40        |TipoReal
41        |TipoBooleano
42        |TipoVoid
43        |TipoCaractere
44
45  and senao = stm_list list
46
47  and 'case = CaseInt of int pos * stm_list list
48        |CaseChar of char pos * stm_list list
49
50  and variable = Var of identificador
51        |VarElement of variable * expressao
52
53  and op =
54        Add
55      | Sub
56      | Pot
57      | Mult
58      | Div
59      | Mod
60      | And
61      | Or
62      | Menor
63      | MenorIgual
64      | Igual
65      | Dif
66      | Maior
67      | MaiorIgual
68
69  open Ast
70
71  and expressao =
72        |ExpOp of op * expressao * expressao
73        |ExpFunCall of identificador * fargs
74        |ExpString of string
75        |ExpInt of int
76        |ExpFloat of float
77        |ExpChar of char
78        |ExpBool of bool
79        |ExpVar of variable
80            |ExpMaisVar of variable
81
82
83
84  and fargs = expressao list
```

```
85
86 and logico_value = Verdadeiro of bool
87           |Falso of bool
```

Listing 84: Especificação Lexical Alterada

```
1
2 {
3   open Parser
4   open Lexing
5   open Printf
6
7
8   let incr_num_linha lexbuf =
9     let pos = lexbuf.lex_curr_p in
10     lexbuf.lex_curr_p <- { pos with
11        pos_lnum = pos.pos_lnum + 1;
12        pos_bol = pos.pos_cnum;
13     }
14
15  let msg_erro lexbuf c =
16    let pos = lexbuf.lex_curr_p in
17    let lin = pos.pos_lnum
18    and col = pos.pos_cnum - pos.pos_bol - 1 in
19    sprintf "%d-%d: caracter desconhecido %c" lin col c
20
21  let erro lin col msg =
22    let mensagem = sprintf "%d-%d: %s" lin col msg in
23       failwith mensagem
24
25  let pos_atual lexbuf = lexbuf.lex_start_p
26
27 }
28
29 let digito = ['0' - '9']
30 let int = '-'? digito+
31 let float = '-' ? digito+ '.'? digito+
32
33 let letra = ['a' - 'z' 'A' - 'Z']
34 let char = ''' (letra | digito) '''
35 let identificador = letra ( letra | digito | '_')*
36
37 let brancos = [' ' '\t']+
38 let novalinha = '\r' | '\n' | "\r\n"
39
```

```
40 let comentario = "//" [^ '\r' '\n' ]*
41
42 rule token = parse
43   brancos     { token lexbuf }
44 | novalinha   { incr_num_linha lexbuf; token lexbuf }
45 | comentario { token lexbuf }
46 | "/*"           { comentario_bloco 0 lexbuf }
47 | '('            { APAR   }
48 | ')'            { FPAR   }
49 | '{'            { ACHAVE  }
50 | '}'            { FCHAVE  }
51 | '['            { ACOLCH  }
52 | ']'            { FCOLCH  }
53 | '+'            { MAIS  }
54 | '-'            { MENOS  }
55 | '*'            { MULT  }
56 | '/'            { DIV  }
57 | '%'            { MOD  }
58 | "**"           { POT  }
59 | '='            { ATRIB  }
60 | '>'            { MAIOR  }
61 | '<'            { MENOR  }
62 | "<="           { MENORIGUAL  }
63 | ">="           { MAIORIGUAL  }
64 | "=="           { IGUAL  }
65 | "++"           { INCR  }
66 | "--"           { DECR  }
67 | "&&"           { AND  }
68 | "||"           { OR  }
69 | '!'            { NOT  }
70 | "!="           { DIF  }
71 | ','            { VIRG  }
72 | ';'            { PONTOVIRG  }
73 | ':'            { DOISPTO  }
74 | "for"          { FOR  }
75 | "if"           { IF  }
76 | "else"         { ELSE  }
77 | "switch"       { SWITCH  }
78 | "case"         { CASE  }
79 | "break"        { BREAK  }
80 | "default"      { DEFAULT  }
81 | "console.log" { CONSOLELOG  }
82 | "function"     { FUNCTION  }
83 | "return"       { RETURN  }
84 | "var"          { VAR  }
```

```
85 | "while"       { WHILE  }
86 | "int"         { INT  }
87 | "float"       { FLOAT  }
88 | "char"        { CHAR  }
89 | "string"      { STRING  }
90 | "bool"        { BOOL  }
91 | "void"        { VOID }
92 | "do"          { DO  }
93 | "prompt"      { PROMPT  }
94 | "true"        { TRUE  }
95 | "false"      { FALSE  }
96 | identificador as id   { ID (id, pos_atual lexbuf) }
97 | int as n { LITINT (int_of_string n) }
98 | float as f { LITFLOAT (float_of_string f)}
99 | char as c { LITCHAR (c.[1])}
100 | '"'        { let buffer = Buffer.create 1 in
101               let str = leia_string buffer lexbuf in
102                LITSTRING str}
103 | _  { raise (Erro ("Caracter desconhecido: " ^ Lexing.lexeme
      lexbuf)) }
104 | eof   { EOF }
105
106 and leia_string lin col buffer = parse
107 '"' { Buffer.contents buffer}
108 | "\\t" { Buffer.add_char buffer '\t';
109 leia_string lin col buffer lexbuf }
110 | "\\n" { Buffer.add_char buffer '\n';
111 leia_string lin col buffer lexbuf }
112 | '\\' '"' { Buffer.add_char buffer '"';
113 leia_string lin col buffer lexbuf }
114 | '\\' '\\' { Buffer.add_char buffer '\\';
115 leia_string lin col buffer lexbuf }
116 | _ as c { Buffer.add_char buffer c;
117 leia_string lin col buffer lexbuf }
118 | eof { erro lin col "A string n~ao foi fechada"}
119
120 and comentario_bloco n = parse
121 "*/"        { if n=0 then token lexbuf
122               else comentario_bloco (n-1) lexbuf }
123 | "/*"      { failwith "Comentarios aninhados nao permitidos"
      }
124 | novalinha { incr_num_linha lexbuf;
125               comentario_bloco n lexbuf }
126 | _         { comentario_bloco n lexbuf }
127 | eof       { failwith "Coment ário não fechado" }
```

8.7 Compilação do Analisador Sintático

Antes de compilar o analisador sintático é necessário obter e editar mensagens para erros sintaticos. Através do Menhir é possível listar os erros sintáticos que podem ocorrer. Com o seguinte comando é gerado o arquivo parser.msg contendo tais erros.

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

Após listados os erros sintáticos é necessário editar as mensagens para cada tipo de erro e então compilar o arquivo com os erros e mensagens editadas.

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

Após estes passos basta compilar o analisador sintático para testá-lo.

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

8.8 Teste do Analisador Sintático

Para testar o analisador sintático é necessário um arquivo de teste contendo algum código na linguagem trabalhada. O teste é feito através dos seguintes comandos, gerando a árvore sintatica abstrata.

```
rlwrap ocaml
parse_arq "teste.js";;
```

8.9 Analisador Semântico

A próxima etapa do processo de compilação é a Analise Semântica, que verifica aspectos relacionados ao significado das sentenças, percorrendo a árvore sintática gerada pelo analisador sintático. Após passar pela análise semântica o programa fonte estará apto para ser convertido para linguagem de máquina.

Uma parte importante desta etapa é a verificação de tipos, onde o compilador verifica regras do tipo:

- os operandos de um operador devem ser do mesmo tipo do operador;

- uma função deve retornar um valor que seja do tipo especificado na declaração da função;
- os parâmetros passados para uma função devem ser do tipo especificado na declaração da função;
- o número de parâmetros passado para uma função deve corresponder ao número de parâmetros especificado na declaração da função;
- etc.

Além disso, a análise semântica é responsável por verificar se variaveis usadas no programa fonte foram declaradas e por determinar o escopo de cada variável. Assim como na análise sintática, ao fim desta etapa será gerada uma árvore, porém uma árvore tipada.

A implementação desta etapa também é feita através do Menhir. A seguir são apresentadas as etapas para a implementação, compilação e teste do Analisador Semântico.

8.10 Especificação Semântica para MiniJavaScript

A seguir são apresentados todos os arquivos necessários para a implementação do analisador semântico. É importante ressaltar que foram realizadas algumas alterações nos arquivos correspondentes as etapas de análise léxica e sintática.

Listing 85: Especificação Lexical Alterada (arquivo lexer.mll)

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

```
15
16   let msg_erro lexbuf c =
17     let pos = lexbuf.lex_curr_p in
18     let lin = pos.pos_lnum
19     and col = pos.pos_cnum - pos.pos_bol - 1 in
20     sprintf "%d-%d: caracter desconhecido %c" lin col c
21
22   let erro lin col msg =
23     let mensagem = sprintf "%d-%d: %s" lin col msg in
24        failwith mensagem
25
26   let pos_atual lexbuf = lexbuf.lex_start_p
27 }
28
29 let digito = ['0' - '9']
30 let int = '-'? digito+
31 let float = '-' ? digito+ '.'? digito+
32
33 let letra = ['a' - 'z' 'A' - 'Z']
34 let char = ''' (letra | digito) '''
35 let identificador = letra ( letra | digito | '_')*
36
37 let brancos = [' ' '\t']+
38 let novalinha = '\r' | '\n' | "\r\n"
39
40 let comentario = "//" [^ '\r' '\n' ]*
41
42 rule token = parse
43   brancos    { token lexbuf }
44 | novalinha  { incr_num_linha lexbuf; token lexbuf }
45 | comentario { token lexbuf }
46 | "/*"          { comentario_bloco 0 lexbuf }
47 | '('           { APAR  (pos_atual lexbuf) }
48 | ')'           { FPAR  (pos_atual lexbuf) }
49 | '{'           { ACHAVE (pos_atual lexbuf) }
50 | '}'           { FCHAVE (pos_atual lexbuf) }
51 | '['           { ACOLCH (pos_atual lexbuf) }
52 | ']'           { FCOLCH (pos_atual lexbuf) }
53 | '+'           { MAIS (pos_atual lexbuf) }
54 | '-'           { MENOS (pos_atual lexbuf) }
55 | '*'           { MULT (pos_atual lexbuf) }
56 | '/'           { DIV (pos_atual lexbuf) }
57 | '%'           { MOD (pos_atual lexbuf) }
58 | "**"          { POT (pos_atual lexbuf) }
59 | '='           { ATRIB (pos_atual lexbuf) }
```

```
60 | '>'            { MAIOR (pos_atual lexbuf) }
61 | '<'            { MENOR (pos_atual lexbuf) }
62 | "<="           { MENORIGUAL (pos_atual lexbuf) }
63 | ">="           { MAIORIGUAL (pos_atual lexbuf) }
64 | "=="           { IGUAL (pos_atual lexbuf) }
65 | "++"           { INCR (pos_atual lexbuf) }
66 | "--"           { DECR (pos_atual lexbuf) }
67 | "&&"           { AND (pos_atual lexbuf) }
68 | "||"           { OR (pos_atual lexbuf) }
69 | '!'            { NOT (pos_atual lexbuf) }
70 | "!="           { DIF (pos_atual lexbuf) }
71 | ','            { VIRG (pos_atual lexbuf) }
72 | ';'            { PONTOVIRG (pos_atual lexbuf) }
73 (*| '.'            { PONTO (pos_atual lexbuf) }*)
74 | ':'            { DOISPTO (pos_atual lexbuf) }
75 | "for"          { FOR (pos_atual lexbuf) }
76 | "if"           { IF (pos_atual lexbuf) }
77 | "else"         { ELSE (pos_atual lexbuf) }
78 | "switch"       { SWITCH (pos_atual lexbuf) }
79 | "case"         { CASE (pos_atual lexbuf) }
80 | "break"        { BREAK (pos_atual lexbuf) }
81 | "default"      { DEFAULT (pos_atual lexbuf) }
82 | "console.log"  { CONSOLELOG (pos_atual lexbuf) }
83 | "function"     { FUNCTION (pos_atual lexbuf) }
84 | "return"       { RETURN (pos_atual lexbuf) }
85 | "var"          { VAR (pos_atual lexbuf) }
86 | "while"        { WHILE (pos_atual lexbuf) }
87 (*| "let"          { LET (pos_atual lexbuf) }*)
88 | "int"          { INT (pos_atual lexbuf) }
89 | "float"        { FLOAT (pos_atual lexbuf) }
90 | "char"         { CHAR (pos_atual lexbuf) }
91 | "string"       { STRING (pos_atual lexbuf) }
92 | "bool"         { BOOL (pos_atual lexbuf) }
93 | "void"         { VOID (pos_atual lexbuf)}
94 | "do"           { DO (pos_atual lexbuf) }
95 | "prompt"       { PROMPT (pos_atual lexbuf) }
96 | "true"         { TRUE (pos_atual lexbuf) }
97 | "false"       { FALSE (pos_atual lexbuf) }
98 | "main"        { MAIN (pos_atual lexbuf) }
99 | identificador as id   { ID (id, pos_atual lexbuf) }
100 | int as n { LITINT (int_of_string n, pos_atual lexbuf) }
101 | float as f { LITFLOAT (float_of_string f, pos_atual lexbuf)
     }
102 | char as c { LITCHAR (c.[1],pos_atual lexbuf)}
103 | '"'         { let buffer = Buffer.create 1 in
```

```
104                 let str = leia_string buffer lexbuf in
105                  LITSTRING (str, pos_atual lexbuf)}
106 | _  { raise (Erro ("Caracter desconhecido: " ^ Lexing.lexeme
      lexbuf)) }
107 | eof    { EOF }
108
109 and comentario_bloco n = parse
110    "}"     { if n=0 then token lexbuf
111              else comentario_bloco (n-1) lexbuf }
112 | "{"      { comentario_bloco (n+1) lexbuf }
113 | "\n"     { incr_num_linha lexbuf; comentario_bloco n lexbuf}
114 | _        { comentario_bloco n lexbuf }
115 | eof      { raise (Erro "Comentário não terminado") }
116 and leia_string buffer = parse
117   '"'     { Buffer.contents buffer}
118 | "\\t"    { Buffer.add_char buffer '\t'; leia_string buffer
     lexbuf }
119 | "\\n"    { Buffer.add_char buffer '\n'; leia_string buffer
     lexbuf }
120 | '\\' '"'  { Buffer.add_char buffer '"'; leia_string buffer
     lexbuf }
121 | '\\' '\\'  { Buffer.add_char buffer '\\'; leia_string
     buffer lexbuf }
122 | _ as c    { Buffer.add_char buffer c; leia_string buffer
     lexbuf }
123 | eof      { raise (Erro "A string não foi fechada") }
```

Listing 86: Especificação Sintática Alterada (arquivo parser.mly)

```
1
2 %{
3   open Lexing
4   open Ast
5   open Sast
6 %}
7
8 %token <Lexing.position> MAIN
9 %token <Lexing.position> APAR
10 %token <Lexing.position> FPAR
11 %token <Lexing.position> ACHAVE
12 %token <Lexing.position> FCHAVE
13 %token <Lexing.position> ACOLCH
14 %token <Lexing.position> FCOLCH
15 %token <Lexing.position> MAIS
16 %token <Lexing.position> MENOS
```

```
17 %token <Lexing.position> MULT
18 %token <Lexing.position> DIV
19 %token <Lexing.position> MOD
20 %token <Lexing.position> POT
21 %token <Lexing.position> MAIOR
22 %token <Lexing.position> MENOR
23 %token <Lexing.position> ATRIB
24 %token <Lexing.position> IGUAL
25 %token <Lexing.position> MENORIGUAL
26 %token <Lexing.position> MAIORIGUAL
27 %token <Lexing.position> INCR
28 %token <Lexing.position> DECR
29 %token <Lexing.position> AND
30 %token <Lexing.position> OR
31 %token <Lexing.position> NOT
32 %token <Lexing.position> VIRG
33 %token <Lexing.position> PONTOVIRG
34 %token <Lexing.position> DIF
35 %token <Lexing.position> FOR
36 %token <Lexing.position> IF
37 %token <Lexing.position> ELSE
38 %token <Lexing.position> WHILE
39 %token <Lexing.position> SWITCH
40 %token <Lexing.position> CASE
41 %token <Lexing.position> BREAK
42 %token <Lexing.position> DEFAULT
43 %token <Lexing.position> CONSOLELOG
44 %token <Lexing.position> FUNCTION
45 %token <Lexing.position> RETURN
46 %token <int * Lexing.position> LITINT
47 %token <float * Lexing.position> LITFLOAT
48 %token <string * Lexing.position> LITSTRING
49 %token <bool * Lexing.position> LITBOOL
50 %token <char * Lexing.position> LITCHAR
51 %token <Lexing.position> VAR
52 (*%token <Lexing.position> LET*)
53 %token <string * Lexing.position> ID
54 (*%token <Lexing.position> PONTO*)
55 %token <Lexing.position> DOISPTO
56 %token <Lexing.position> DO
57 %token <Lexing.position> PROMPT
58 %token <Lexing.position> INT
59 %token <Lexing.position> FLOAT
60 %token <Lexing.position> STRING
61 %token <Lexing.position> BOOL
```

```
%token <Lexing.position> TRUE
%token <Lexing.position> FALSE
%token <Lexing.position> CHAR
%token <Lexing.position> VOID
%token EOF


%left OR
%left AND
%left IGUAL DIF
%left MAIOR MAIORIGUAL MENOR MENORIGUAL
%left MAIS MENOS
%left MULT DIV MOD
%right POT
%right NOT


%start <Sast.expressao Ast.prog> prog

%%

prog:
    | vdb=var_decl* fd=func_decl* stmb=stm_block  EOF { Prog
       (List.flatten vdb,fd,stmb) }
    ;

var_decl:
    | VAR ids = separated_nonempty_list(VIRG, ID) DOISPTO t=
       tp_primitivo PONTOVIRG {  List.map (fun id -> DecVar(
       id,t)) ids }
    ;


tp_primitivo:
    | INT { TipoInteiro }
    | FLOAT { TipoReal }
    | CHAR { TipoCaractere }
    | BOOL { TipoBooleano }
    | VOID { TipoVoid }
    ;

stm_block:
    | stms=stm_list* { (stms)}
    ;

```

```
104 stm_list:
105     | stm=stm_attr {stm}
106     | stm=stm_fcall {stm}
107     | stm=stm_ret {stm}
108     | stm=stm_if {stm}
109     | stm=stm_while {stm}
110    (* | stm=stm_for {stm}*)
111     | stm=stm_print {stm}
112     (*| stm=stm_read {stm}*)
113     | stm=stm_switch {stm}
114     | stm = stm_declara_var { stm }
115     ;
116
117
118 stm_declara_var:
119     VAR ids=separated_nonempty_list(VIRG, ID) DOISPTO t=
            tp_primitivo PONTOVIRG { StmVarDecl(List.map(fun id ->
             VarDecl(id, t)) ids) }
120     ;
121
122 stm_fcall:
123     | exp=fcall PONTOVIRG {Chamada exp}
124    | v=expr ATRIB exp=fcall PONTOVIRG { ChamadaRec(v,exp)}
125     ;
126
127 stm_ret:
128     | RETURN ex=expr? PONTOVIRG { Retorne(ex)}
129     ;
130
131 variable:
132     | id=ID { Var(id) }
133     | v=variable ACOLCH e=expr FCOLCH {VarElement(v,e)}
134     ;
135
136
137 stm_attr:
138     | v=expr ATRIB e=expr PONTOVIRG { Attrib(v,e) }
139     ;
140
141 stm_if:
142     | IF APAR e=expr FPAR ACHAVE stms=stm_list* FCHAVE senao=
            stm_else? { Se(e,stms,senao)}
143  (*   | IF APAR e=expr FPAR stm=stm_list senao=stm_else { Se(
        e,[stm],senao)}*)
144     ;
```

```
145
146 stm_else:
147     | ELSE ACHAVE stm=stm_list* FCHAVE { stm }
148    (* | ELSE stm=smt_list {[stm]}
149       /*| ELSE IF APAR exp=expressao FPAR ACHAVE stms=
              statement* FCHAVE another=stm_else? { StmElseIf(exp,
               stms, another) }
150     | ELSE IF APAR exp=expressao FPAR st=statement another=
           stm_else? { StmElseIf(exp, [st], another) }*/*)
151     ;
152
153 stm_switch:
154     | SWITCH APAR id=ID FPAR ACHAVE c=case+ DEFAULT DOISPTO
           stms=stm_list* FCHAVE {Escolha(id,c,stms) }
155     ;
156
157 case:
158     | CASE c=LITCHAR DOISPTO stms=stm_list* BREAK {CaseChar(c
           ,stms)}
159     | CASE i=LITINT DOISPTO stms=stm_list* BREAK {CaseInt(i,
           stms) }
160     ;
161
162 stm_while:
163     | WHILE APAR e=expr FPAR ACHAVE stm=stm_list* FCHAVE {
           Enquanto (e,stm) }
164     ;
165
166 (*
167 stm_for:
168     | FOR APAR lv=expr DE e1=expr ATE e2=expr p=passo FACA
           stm=stm_list* FIMPARA {Para (lv,e1,e2,p,stm) }
169     ;
170 *)
171
172 (*
173 stm_read:
174     | v=variable ATRIB PROMPT APAR e=expr FPAR PONTOVIRG {
           Leia (v,e)}
175     ;
176 *)
177 stm_print:
178     | CONSOLELOG APAR  stm=separated_nonempty_list(VIRG, expr
           ) FPAR PONTOVIRG {Escreva stm}
179     ;
```

```
180
181
182 expr:
183    | e1=expr o=op e2=expr { ExpOp(o,e1,e2) }
184    | f=fcall { f }
185    | s=LITSTRING { ExpString s}
186    | i=LITINT { ExpInt i}
187    | f=LITFLOAT { ExpFloat f}
188    | c=LITCHAR { ExpChar c}
189    | l=logico_value { ExpBool l}
190    | lv=variable { ExpVar lv }
191    /*| pos=NOT e=expr { ExpNot ( )e, pos)}*/
192    | APAR e=expr FPAR { e }
193    | MAIS v=variable { ExpMaisVar v }
194    ;
195
196
197
198 %inline op:
199      pos = MAIS { (Add, pos) }
200     | pos = MENOS { (Sub, pos) }
201     | pos = MULT { (Mult, pos) }
202     | pos = DIV { (Div, pos) }
203     | pos = MOD { (Mod, pos) }
204     | pos = OR { (Or, pos) }
205     | pos = AND { (And, pos) }
206     | pos = POT { (Pot, pos) }
207     | pos = MENOR { (Menor, pos) }
208     | pos = IGUAL { (Igual, pos) }
209     | pos = DIF { (Dif, pos) }
210     | pos = MAIOR { (Maior, pos) }
211     | pos =  MAIORIGUAL { (MaiorIgual, pos)}
212     | pos = MENORIGUAL { (MenorIgual, pos) }
213     ;
214
215
216 logico_value:
217   | pos=TRUE { (true,pos) }
218   | pos=FALSE  { (false,pos) }
219   ;
220
221 fcall:
222     | id=ID APAR args=fargs FPAR { ExpFunCall(id, args) }
223     ;
224
```

```
225 fargs:
226     | exprs=separated_list(VIRG, expr) {  List.map (fun expr
          -> expr) exprs}
227     ;
228 (*
229 fcall:
230     | id=ID APAR args=fargs? FPAR { ExpFunCall(id,(match args
          with
231                                                   | None ->
                                                       []
232                                                   | Some
                                                      fargs ->
                                                       fargs )
                                                      ) }
233     ;
234
235 fargs:
236     | exprs=separated_nonempty_list(VIRG, expr) {  List.map (
          fun expr -> expr) exprs}
237     ;*)
238
239 /*Declaração de funções*/
240 func_decl:
241     | FUNCTION id=ID APAR fp=fparams? FPAR fy=func_type fv=
          var_decl* fb=func_bloc
242 { FuncDecl {
243                   fn_id = id;
244                   fn_params = (match fp with
245                    | None -> []
246                    | Some args -> args);
247                    fn_tiporet = fy;
248                    fn_locais = List.flatten fv;
249                    fn_corpo = fb }
250                 }
251     ;
252
253 func_type:
254     | DOISPTO t=tp_primitivo { (t) }
255     ;
256 /* */
257
258 func_bloc:
259     | ACHAVE stm=stm_list* FCHAVE {(stm)}
260     ;
261
```

```
262
263 /*Parâmetro de funções*/
264 fparams:
265     | fparam=separated_nonempty_list(VIRG,fparam){ fparam }
266     ;
267
268 fparam:
269     | id=ID DOISPTO t=tp_primitivo {(id,t)}
270     ;
```

Listing 87: Árvore Sintática Abstrata Alterada (arquivo ast.ml)

```
1  open Lexing
2
3  type identificador = string
4  type 'a pos =  'a * Lexing.position (* tipo e posição no
       arquivo fonte *)
5
6  type 'expr prog = Prog of var_decl * ('expr func_decl_list) *
       ('expr statements)
7  and declaracao_algoritmo = DeclAlg
8  and var_decl = vars list
9  and vars = DecVar of (identificador pos) * tipo
10 and 'expr func_decl_list = ('expr func_decl) list
11 and 'expr func_decl = FuncDecl of ('expr decfun)
12 and fparams = fparam list
13 and fparam =  identificador * tipo
14 and functype = tipo
15 and 'expr funcbloc = ('expr stm_list) list
16 and 'expr statements = ('expr stm_list) list
17
18 and 'expr decfun = {
19   fn_id: identificador pos;
20   fn_params: (identificador pos * tipo) list;
21   fn_tiporet: functype;
22   fn_locais: var_decl;
23   fn_corpo: 'expr funcbloc;
24
25 }
26
27
28 and 'expr stm_list  = Attrib of 'expr  * 'expr
29             |Chamada of 'expr
30                              |ChamadaRec of 'expr * 'expr
31             |Retorne of 'expr option
```

```
32            |Se of 'expr * 'expr stm_list list * ('expr senao)
                  option
33            |Escolha of identificador pos * 'expr case list *
                 'expr stm_list list
34          (*   |Para of ('expr) * 'expr * 'expr * ('expr )  * '
                 expr statements *)
35            (* |Leia of 'expr variable * 'expr*)
36            |Escreva of 'expr list
37            |Enquanto of 'expr * 'expr stm_list list
38                          |StmVarDecl of varDeclaracao list
39
40 and varDeclaracao = VarDecl of identificador pos * tipo
41 and tipo = TipoInteiro
42       |TipoReal
43       |TipoBooleano
44       |TipoVoid
45       |TipoCaractere
46
47 and 'expr senao = 'expr stm_list list
48
49 and 'expr case = CaseInt of int pos * 'expr stm_list list
50       |CaseChar of char pos * 'expr stm_list list
51
52 and 'expr variable = Var of identificador pos
53       |VarElement of ('expr variable) * 'expr
54
55 and op =
56       Add
57     | Sub
58     | Pot
59     | Mult
60     | Div
61     | Mod
62     | And
63     | Or
64     | Menor
65     | MenorIgual
66     | Igual
67     | Dif
68     | Maior
69     | MaiorIgual
70
71 (*let arguments:
72   match args with
73     None -> []
```

```
74      Some xs -> xs*)
75 and 'expr fargs = 'expr list
76
77 and logico_value = Verdadeiro of bool
78           |Falso of bool
```

Listing 88: Arquivo sast.ml

```
1 open Ast
2
3 type expressao =
4       |ExpOp of op pos * expressao * expressao
5       |ExpFunCall of identificador pos * expressao fargs
6       |ExpString of string pos
7       |ExpInt of int pos
8       |ExpFloat of float pos
9       |ExpChar of char pos
10      |ExpBool of bool pos
11      |ExpVar of (expressao variable)
12                      |ExpMaisVar of (expressao variable)
```

Listing 89: Arquivo tast.ml

```
1 open Ast
2
3 type expressao = ExpVar of (expressao variable) * tipo
4       |ExpOp of (op * tipo ) * (expressao * tipo) * (
            expressao * tipo)
5       |ExpFunCall of identificador * expressao fargs * tipo
6       |ExpString of string * tipo
7       |ExpInt of int * tipo
8       |ExpFloat of float * tipo
9       |ExpChar of char * tipo
10      |ExpBool of bool * tipo
```

Listing 90: Arquivo tabsimb.ml

```
1
2 type 'a tabela = {
3     tbl: (string, 'a) Hashtbl.t;
4     pai: 'a tabela option;
5 }
6
7 exception Entrada_existente of string;;
8
```

```
let insere amb ch v =
  if Hashtbl.mem amb.tbl ch
  then raise (Entrada_existente ch)
  else Hashtbl.add amb.tbl ch v

let substitui amb ch v = Hashtbl.replace amb.tbl ch v

let rec atualiza amb ch v =
    if Hashtbl.mem amb.tbl ch
    then Hashtbl.replace amb.tbl ch v
    else match amb.pai with
       None -> failwith "tabsim atualiza: chave nao
          encontrada"
     | Some a -> atualiza a ch v

let rec busca amb ch =
  try Hashtbl.find amb.tbl ch
  with Not_found ->
    (match amb.pai with
       None -> raise Not_found
     | Some a -> busca a ch)

let rec cria cvs =
  let amb = {
    tbl = Hashtbl.create 5;
    pai = None
  } in
  let _ = List.iter (fun (c,v) -> insere amb c v) cvs
  in amb

let novo_escopo amb_pai = {
  tbl = Hashtbl.create 5;
  pai = Some amb_pai
}
```

Listing 91: Arquivo ambiente.ml

```
module Tab = Tabsimb
module A = Ast

type entrada_fn = { tipo_fn:  A.tipo;
                    formais: (string * A.tipo) list;
}

type entrada =  EntFun of entrada_fn
```

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

Listing 92: Arquivo semantico.ml

```
1 module Amb = Ambiente
2 module A = Ast
3 module S = Sast
4 module T = Tast
5
6
7 let rec posicao exp = let open S in
8   match exp with
9   | ExpVar v -> (match v with
10       | A.Var (_,pos) -> pos
11       | A.VarElement (_,exp2) -> posicao exp2
12     )
13   (*| ExpNot (_, pos) -> pos *)
14   | ExpInt (_,pos) -> pos
15   | ExpFloat (_,pos) -> pos
16   | ExpChar (_,pos) -> pos
17   | ExpString  (_,pos) -> pos
18   | ExpBool (_,pos) -> pos
19   | ExpOp ((_,pos),_,_)  -> pos
20   | ExpFunCall ((_,pos), _) -> pos
```

```
21    | ExpMaisVar v -> (match v with
22        | A.Var (_,pos) -> pos
23        | A.VarElement (_,exp2) -> posicao exp2
24      )
25
26
27  type classe_op = Aritmetico | Relacional | Logico
28
29  let classifica op =
30    let open A in
31    match op with
32        Add
33      | Sub
34      | Pot
35      | Mult
36      | Div
37      | Mod -> Aritmetico
38      | And
39      | Or -> Logico
40      | Menor
41      | MenorIgual
42      | Igual
43      | Dif
44      | Maior
45      | MaiorIgual -> Relacional
46
47  let msg_erro_pos pos msg =
48    let open Lexing in
49    let lin = pos.pos_lnum
50    and col = pos.pos_cnum - pos.pos_bol - 1 in
51    Printf.sprintf "Semantico -> linha %d, coluna %d: %s" lin
        col msg
52
53  let msg_erro nome msg =
54    let pos = snd nome in
55    msg_erro_pos pos msg
56
57  let nome_tipo t =
58    let open A in
59      match t with
60        TipoInteiro -> "inteiro"
61      | TipoCaractere -> "caractere"
62      | TipoBooleano -> "logico"
63      | TipoReal -> "real"
64      | TipoVoid -> "vazio"
```

```
65
66
67 let mesmo_tipo pos msg tinf tdec =
68   if tinf <> tdec
69   then
70     let msg = Printf.sprintf msg (nome_tipo tinf) (nome_tipo
         tdec) in
71     failwith (msg_erro_pos pos msg)
72
73 let rec infere_exp amb exp =
74   match exp with
75     S.ExpInt n    -> (T.ExpInt (fst n, A.TipoInteiro),
         A.TipoInteiro)
76   | S.ExpString s -> (T.ExpString (fst s, A.TipoCaractere), A
       .TipoCaractere)
77   | S.ExpBool b   -> (T.ExpBool (fst b, A.TipoBooleano),
       A.TipoBooleano)
78   | S.ExpFloat f  -> (T.ExpFloat (fst f, A.TipoReal), A.
       TipoReal)
79   | S.ExpChar c   -> (T.ExpChar (fst c, A.TipoCaractere), A.
       TipoCaractere)
80   | S.ExpVar v ->
81     (match v with
82        A.Var nome ->
83        (* Tenta encontrar a definição da variável no escopo
           local, se não      *)
84        (* encontar tenta novamente no escopo que engloba o
           atual. Prossegue-se *)
85        (* assim até encontrar a definição em algum escopo
           englobante ou até    *)
86        (* encontrar o escopo global. Se em algum lugar for
           encontrado,          *)
87        (* devolve-se a definição. Em caso contrário, devolve
           uma exceção       *)
88        let id = fst nome in
89          (try (match (Amb.busca amb id) with
90                | Amb.EntVar tipo -> (T.ExpVar (A.Var nome,
                    tipo), tipo)
91                | Amb.EntFun _ ->
92                  let msg = "nome de funcao usado como nome de
                        variavel: " ^ id in
93                   failwith (msg_erro nome msg)
94             )
95           with Not_found ->
96                  let msg = "A variavel " ^ id ^ " nao foi
```

```
                    declarada" in
97                  failwith (msg_erro nome msg)
98          )
99       | _ -> failwith "infere_exp: não implementado"
100     )
101
102   | S.ExpOp (op, esq, dir) ->
103     let (esq, tesq) = infere_exp amb esq
104     and (dir, tdir) = infere_exp amb dir in
105
106     let verifica_aritmetico () =
107       (match tesq with
108          A.TipoInteiro
109        |A.TipoReal ->
110          let _ = mesmo_tipo (snd op)
111                      "O operando esquerdo eh do tipo %s mas
                            o direito eh do tipo %s"
112                      tesq tdir
113          in tesq (* O tipo da expressão aritmética como um
                todo *)
114
115        | t -> let msg = "um operador aritmetico nao pode ser
                usado com o tipo " ^
116                         (nome_tipo t)
117          in failwith (msg_erro_pos (snd op) msg)
118       )
119
120     and verifica_relacional () =
121       (match tesq with
122          A.TipoInteiro
123        | A.TipoReal
124        | A.TipoCaractere ->
125          let _ = mesmo_tipo (snd op)
126                    "O operando esquerdo eh do tipo %s mas o
                          direito eh do tipo %s"
127                    tesq tdir
128          in A.TipoBooleano (* O tipo da expressão relacional
                é sempre booleano *)
129
130        | t -> let msg = "um operador relacional nao pode ser
                usado com o tipo " ^
131                         (nome_tipo t)
132          in failwith (msg_erro_pos (snd op) msg)
133       )
134
```

```
135     and verifica_logico () =
136       (match tesq with
137          A.TipoBooleano ->
138          let _ = mesmo_tipo (snd op)
139                    "O operando esquerdo eh do tipo %s mas o
                          direito eh do tipo %s"
140                    tesq tdir
141          in A.TipoBooleano (* O tipo da expressão lógica é
                sempre booleano *)
142
143        | t -> let msg = "um operador logico nao pode ser
                usado com o tipo " ^
144                         (nome_tipo t)
145              in failwith (msg_erro_pos (snd op) msg)
146       )
147   (*  and verifica_cadeia () =
148       (match tesq with
149          A.TipoCaractere ->
150          let _ = mesmo_tipo (snd op)
151                    "O operando esquerdo eh do tipo %s mas o
                          direito eh do tipo %s"
152                    tesq tdir
153          in A.TipoCaractere (* O tipo da expressão relacional
                é sempre string *)
154
155        | t -> let msg = "um operador relacional nao pode ser
                usado com o tipo " ^
156                         (nome_tipo t)
157              in failwith (msg_erro_pos (snd op) msg)
158       )
159   *)
160     in
161     let op = fst op in
162     let tinf = (match (classifica op) with
163           Aritmetico -> verifica_aritmetico ()
164         | Relacional -> verifica_relacional ()
165         | Logico -> verifica_logico ()
166       )
167     in
168       (T.ExpOp ((op,tinf), (esq, tesq), (dir, tdir)), tinf)
169
170   | S.ExpFunCall (nome, args) ->
171      let rec verifica_parametros ags ps fs =
172         match (ags, ps, fs) with
173          (a::ags), (p::ps), (f::fs) ->
```

```
174              let _ = mesmo_tipo (posicao a)
175                       "O parametro eh do tipo %s mas deveria
                             ser do tipo %s" p f
176              in verifica_parametros ags ps fs
177        | [], [], [] -> ()
178        | _ -> failwith (msg_erro nome "Numero incorreto de
               parametros")
179      in
180      let id = fst nome in
181      try
182        begin
183          let open Amb in
184
185          match (Amb.busca amb id) with
186          (* verifica se 'nome' está associada a uma função *)
187            Amb.EntFun {tipo_fn; formais} ->
188            (* Infere o tipo de cada um dos argumentos *)
189            let argst = List.map (infere_exp amb) args
190            (* Obtem o tipo de cada parâmetro formal *)
191            and tipos_formais = List.map snd formais in
192            (* Verifica se o tipo de cada argumento confere
                  com o tipo declarado *)
193            (* do parâmetro formal correspondente.
                                                  *)
194            let _ = verifica_parametros args (List.map snd
                  argst) tipos_formais
195             in (T.ExpFunCall (id, (List.map fst argst),
                  tipo_fn), tipo_fn)
196          | Amb.EntVar _ -> (* Se estiver associada a uma vari
                 ável, falhe *)
197            let msg = id ^ " eh uma variavel e nao uma funcao"
                  in
198            failwith (msg_erro nome msg)
199        end
200      with Not_found ->
201        let msg = "Nao existe a funcao de nome " ^ id in
202        failwith (msg_erro nome msg)
203
204 let rec verifica_cmd amb tiporet cmd =
205   let open A in
206    match cmd with
207     Retorne exp ->
208     (match exp with
209      (* Se a função não retornar nada, verifica se ela foi
            declarada como void *)
```

```
210         None ->
211         let _ = mesmo_tipo (Lexing.dummy_pos)
212                     "O tipo retornado eh %s mas foi declarado
                           como %s"
213                     TipoVoid tiporet
214         in Retorne None
215       | Some e ->
216         (* Verifica se o tipo inferido para a expressão de
              retorno confere com o *)
217         (* tipo declarado para a função.
                                                          *)
218             let (e1,tinf) = infere_exp amb e in
219             let _ = mesmo_tipo (posicao e)
220                                "O tipo retornado eh %s mas foi
                                     declarado como %s"
221                                tinf tiporet
222             in Retorne (Some e1)
223       )
224   | Se (teste, entao, senao) ->
225     let (teste1,tinf) = infere_exp amb teste in
226     (* O tipo inferido para a expressão 'teste' do
          condicional deve ser booleano *)
227     let _ = mesmo_tipo (posicao teste)
228               "O teste do if deveria ser do tipo %s e nao %s"
229               TipoBooleano tinf in
230     (* Verifica a validade de cada comando do bloco 'então'
          *)
231     let entao1 = List.map (verifica_cmd amb tiporet) entao in
232     (* Verifica a validade de cada comando do bloco 'senão',
          se houver *)
233     let senao1 =
234         match senao with
235           None -> None
236         | Some bloco -> Some (List.map (verifica_cmd amb
              tiporet) bloco)
237      in
238      Se (teste1, entao1, senao1)
239
240   | Attrib (elem, exp) ->
241     (* Infere o tipo da expressão no lado direito da atribuiç
          ão *)
242     let (exp,  tdir) = infere_exp amb exp
243     (* Faz o mesmo para o lado esquerdo *)
244     and (elem1, tesq) = infere_exp amb elem in
245     (* Os dois tipos devem ser iguais *)
```

```
246     let _ = mesmo_tipo (posicao elem)
247                        "Atribuicao com tipos diferentes: %s =
                               %s" tesq tdir
248     in Attrib (elem1, exp)
249
250   | Enquanto (teste, corpo) ->
251     let (teste_tipo,tinf) = infere_exp amb teste in
252     let _ = mesmo_tipo (posicao teste)
253                        "O teste do enquanto deveria ser do
                              tipo %s e nao %s"
254                         TipoBooleano tinf in
255     let corpo_tipo = List.map (verifica_cmd amb tiporet)
           corpo in
256     Enquanto (teste_tipo, corpo_tipo)
257
258  (*| Para (var, inicio, fim, avanco, corpo) ->
259     let (var_tipo, tinfv) =  infere_exp amb var in
260     let (inicio_tipo,tinfi) = infere_exp amb inicio in
261     let (fim_tipo,tinff) = infere_exp amb fim in
262     let (avanco_tipo,tinfa) = infere_exp amb avanco in
263
264     let corpo_tipo = List.map (verifica_cmd amb tiporet)
           corpo in
265     Para (var_tipo,inicio_tipo,fim_tipo,avanco_tipo,
           corpo_tipo)*)
266
267  (*
268     let t0 = (match var.tipo with Some t -> t | None ->
           failwith "tipo" )
269     and t1 = (match inicio.tipo with Some t -> t | None ->
           failwith
270                        "verifica_cmd->para->t1: tipo indefinido"
                           )
271     and t2 = (match fim.tipo with Some t -> t | None ->
           failwith
272                        "verifica_cmd->para->t2: tipo indefinido"
                           )
273     and t3 = (match avanco with
274                        Some v -> infere_exp amb v;
275                            (match v.tipo with Some t -> t | None
                                   -> failwith
276                        "verifica_cmd->para->t2: tipo indefinido"
                           )
277                        |None -> TipoBooleano
278                    ) in
```

```
279     begin
280       if ((t0 <> TipoInteiro ) || (t0 <> t1) || (t0 <> t2) )
              then
281         msg_erro " para " cmd.pcmd " Os tipos deveraim ser
                iguais"
282       else
283         begin
284           ( match v3 with
285           | Some v ->
286                 if (t0 <> t3) then
287                                 msg_erro " para " cmd.pcmd " Os
                                      tipos deveraim ser iguais"
288
289           | None -> ignore()
290           );
291           verifica_cmd amb corpo
292         end
293     end *)
294  (* | Para (var,inicio,fim,avanco, corpo) ->
295     let comando = Attrib(var,inicio) in
296       let comando_tipo = verifica_cmd amb tiporet comando in
297         let op = operador_do_for avanco in
298           let expression = S.ExpOp(op, inicio, fim) in
299             let (teste_tipo,tinf) = infere_exp amb expression
                    in
300               let _ = mesmo_tipo (posicao fim)
301                           "O teste do para deveria ser do tip
                                %s e nao %s"
302                            TipoBooleano tinf in
303                 let comando_avanco = Attrib(var, S.ExpOp(Soma
                        ,var,avanco)) in
304                   let comando_avanco_tipo = verifica_cmd amb
                          tiporet comando_avanco in
305                     let corpo_tipo = verifica_cmd amb tiporet
                            corpo in
306                       let bloco = Bloco([], corpo_tipo::
                              comando_avanco_tipo::[]) in
307                       let cmd_tipo = comando_tipo::Enquanto(
                              teste_tipo, bloco)::[] in
308                       print_list(cmd_tipo)
309                       Bloco([], cmd_tipo)*)
310   (*| Escolha (id, caso, corpo) ->
311     let (teste_id, tinf) = infere_exp amb id in
312     let _ = mesmo_tipo (posicao id)
313                   "O teste do escolha deve ser do tipo %s e
```

```
                        nao %s"
314                   TipoInteiro tinf in
315  *)
316 (* | Bloco (decs, cmds) ->
317    let amb_bloco = Amb.novo_escopo amb in
318    let decs_tipo = List.map (verifica_dec amb_bloco) decs in
319    let _ = List.iter (analisa_dec amb_bloco) decs_tipo in
320    let cmds_tipo = List.map (verifica_cmd amb_bloco tiporet)
           cmds in
321    Bloco(decs_tipo, cmds_tipo)*)
322
323  | Chamada exp ->
324    (* Verifica o tipo de cada argumento da função 'entrada'
          *)
325    let (exp,tinf) = infere_exp amb exp in
326    Chamada exp
327
328 | ChamadaRec (elem, exp) ->
329    (* Infere o tipo da expressão no lado direito da atribuiç
          ão *)
330    let (exp,  tdir) = infere_exp amb exp
331    (* Faz o mesmo para o lado esquerdo *)
332    and (elem1, tesq) = infere_exp amb elem in
333    (* Os dois tipos devem ser iguais *)
334    let _ = mesmo_tipo (posicao elem)
335                       "Atribuicao com tipos diferentes: %s =
                              %s" tesq tdir
336    in ChamadaRec (elem1, exp)
337
338  | StmVarDecl decsvars ->
339   let ambfn = Amb.novo_escopo amb in
340   let insere_local = function
341        (VarDecl(v,t)) -> Amb.insere_local ambfn (fst v)  t
             in
342   StmVarDecl decsvars
343
344
345  (*| Leia (var, exps) ->
346    (* Infere o tipo da expressão no lado direito da atribuiç
          ão *)
347    let (exps,  tdir) = infere_exp amb exps
348    (* Faz o mesmo para o lado esquerdo *)
349    and (elem1, tesq) = infere_exp amb var in
350    (* Os dois tipos devem ser iguais *)
351    let _ = mesmo_tipo (posicao var)
```

```
352                         "Atribuicao com tipos diferentes: %s =
                                %s" tesq tdir
353     in Leia (elem1, exps)*)
354
355   | Escreva exps ->
356     (* Verifica o tipo de cada argumento da função 'saida' *)
357     let exps = List.map (infere_exp amb) exps in
358     Escreva (List.map fst exps)
359
360 (*and operador_do_for ini =
361   let open A in
362     let zero = ExpInt( 0 ) in
363       if (ini >= zero) then MenorIgual
364       else MaiorIgual *)
365 and verifica_fun amb ast =
366   let open A in
367   match ast with
368     A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
          fn_corpo} ->
369     (* Estende o ambiente global, adicionando um ambiente
          local *)
370     let ambfn = Amb.novo_escopo amb in
371     (* Insere os parâmetros no novo ambiente *)
372     let insere_parametro (v,t) = Amb.insere_param ambfn (fst
          v) t in
373     let _ = List.iter insere_parametro fn_params in
374     (* Insere as variáveis locais no novo ambiente *)
375     let insere_local = function
376         (DecVar(v,t)) -> Amb.insere_local ambfn (fst v)  t in
377     let _ = List.iter insere_local fn_locais in
378     (* Verifica cada comando presente no corpo da função
          usando o novo ambiente *)
379     let corpo_tipado = List.map (verifica_cmd ambfn
          fn_tiporet) fn_corpo in
380       A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
            fn_corpo = corpo_tipado}
381
382
383 let rec verifica_dup xs =
384   match xs with
385     [] -> []
386   | (nome,t)::xs ->
387     let id = fst nome in
388     if (List.for_all (fun (n,t) -> (fst n) <> id) xs)
389     then (id, t) :: verifica_dup xs
```

```
390     else let msg = "Parametro duplicado " ^ id in
391       failwith (msg_erro nome msg)
392
393 let insere_declaracao_var amb dec =
394   let open A in
395     match dec with
396       DecVar(nome, tipo) ->  Amb.insere_local amb (fst nome)
              tipo
397
398 let insere_declaracao_fun amb dec =
399   let open A in
400     match dec with
401       FuncDecl {fn_id; fn_params; fn_tiporet;  fn_corpo} ->
402         (* Verifica se não há parâmetros duplicados *)
403         let formais = verifica_dup fn_params in
404         let nome = fst fn_id in
405         Amb.insere_fun amb nome formais fn_tiporet
406
407 (*let analisa_dec amb dec =
408   let open A in
409     match dec with
410       DecVar(nome, tipo) ->  Amb.insere_local amb (fst nome)
              tipo
411       |FuncDecl {fn_id; fn_params; fn_tiporet;  fn_corpo} ->
412         (* Verifica se não há parâmetros duplicados *)
413         let _ = verifica_dup fn_params in
414         Amb.insere_fun amb fn_id fn_params fn_tiporet
415
416 let verifica_dec amb dec =
417     let open A in
418     match amb with
419     A.DecVar {nome; tipo}  ->
420         A.DecVar {nome; tipo}
421    | A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
          fn_corpo} ->
422     (* Estende o ambiente global, adicionando um ambiente
          local *)
423     let ambfn = Amb.novo_escopo amb in
424     (* Insere os parâmetros no novo ambiente *)
425     let insere_parametro (v,t) = Amb.insere_param ambfn (fst
          v) t in
426     let _ = List.iter insere_parametro fn_params in
427     (* Insere as variáveis locais no novo ambiente *)
428     let insere_local = function
429         (DecVar(v,t)) -> Amb.insere_local ambfn (fst v)  t in
```

```
430     let _ = List.iter insere_local fn_locais in
431     (* Verifica cada comando presente no corpo da função
           usando o novo ambiente *)
432     let corpo_tipado = List.map (verifica_cmd ambfn
           fn_tiporet) fn_corpo in
433       A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
             fn_corpo = corpo_tipado}
434 *)
435
436 (* Lista de cabeçalhos das funções pré definidas *)
437 let fn_predefs = let open A in [
438    ("escreva", [("x", TipoCaractere); ("y", TipoInteiro)],
          TipoVoid);
439    ("leia",    [("x", TipoReal); ("y", TipoInteiro)], TipoVoid
          )
440 ]
441
442 (* insere as funções pré definidas no ambiente global *)
443 let declara_predefinidas amb =
444   List.iter (fun (n,ps,tr) -> Amb.insere_fun amb n ps tr)
          fn_predefs
445
446 let semantico ast =
447   (* cria ambiente global inicialmente vazio *)
448   let amb_global = Amb.novo_amb [] in
449   let _ = declara_predefinidas amb_global in
450   let (A.Prog (decs_globais, decs_funs, corpo)) = ast in
451   let _ = List.iter (insere_declaracao_var amb_global)
          decs_globais in
452   let _ = List.iter (insere_declaracao_fun amb_global)
          decs_funs in
453   (* Verificação de tipos nas funções *)
454   let decs_funs = List.map (verifica_fun amb_global)
          decs_funs in
455   (* Verificação de tipos na função principal *)
456   let corpo = List.map (verifica_cmd amb_global A.TipoVoid)
          corpo in
457     (A.Prog (decs_globais, decs_funs, corpo),  amb_global)
```

Listing 93: Arquivo semanticoTest.ml

```
1 module Amb = Ambiente
2 module A = Ast
3 module S = Sast
4 module T = Tast
```

```
5
6
7  let rec posicao exp = let open S in
8    match exp with
9    | ExpVar v -> (match v with
10       | A.Var (_,pos) -> pos
11       | A.VarElement (_,exp2) -> posicao exp2
12     )
13   (*| ExpNot (_, pos) -> pos *)
14   | ExpInt (_,pos) -> pos
15   | ExpFloat (_,pos) -> pos
16   | ExpChar (_,pos) -> pos
17   | ExpString  (_,pos) -> pos
18   | ExpBool (_,pos) -> pos
19   | ExpOp ((_,pos),_,_)  -> pos
20   | ExpFunCall ((_,pos), _) -> pos
21   | ExpMaisVar v -> (match v with
22       | A.Var (_,pos) -> pos
23       | A.VarElement (_,exp2) -> posicao exp2
24     )
25
26
27 type classe_op = Aritmetico | Relacional | Logico
28
29 let classifica op =
30   let open A in
31   match op with
32       Add
33     | Sub
34     | Pot
35     | Mult
36     | Div
37     | Mod -> Aritmetico
38     | And
39     | Or -> Logico
40     | Menor
41     | MenorIgual
42     | Igual
43     | Dif
44     | Maior
45     | MaiorIgual -> Relacional
46
47 let msg_erro_pos pos msg =
48   let open Lexing in
49   let lin = pos.pos_lnum
```

```
50   and col = pos.pos_cnum - pos.pos_bol - 1 in
51   Printf.sprintf "Semantico -> linha %d, coluna %d: %s" lin
        col msg
52
53 let msg_erro nome msg =
54   let pos = snd nome in
55   msg_erro_pos pos msg
56
57 let nome_tipo t =
58   let open A in
59     match t with
60       TipoInteiro -> "inteiro"
61     | TipoCaractere -> "caractere"
62     | TipoBooleano -> "logico"
63     | TipoReal -> "real"
64     | TipoVoid -> "vazio"
65
66
67 let mesmo_tipo pos msg tinf tdec =
68   if tinf <> tdec
69   then
70     let msg = Printf.sprintf msg (nome_tipo tinf) (nome_tipo
          tdec) in
71     failwith (msg_erro_pos pos msg)
72
73 let rec infere_exp amb exp =
74   match exp with
75     S.ExpInt n    -> (T.ExpInt (fst n, A.TipoInteiro),
          A.TipoInteiro)
76   | S.ExpString s -> (T.ExpString (fst s, A.TipoCaractere), A
        .TipoCaractere)
77   | S.ExpBool b   -> (T.ExpBool (fst b, A.TipoBooleano),
        A.TipoBooleano)
78   | S.ExpFloat f  -> (T.ExpFloat (fst f, A.TipoReal), A.
        TipoReal)
79   | S.ExpChar c   -> (T.ExpChar (fst c, A.TipoCaractere), A.
        TipoCaractere)
80   | S.ExpVar v ->
81     (match v with
82        A.Var nome ->
83        (* Tenta encontrar a definição da variável no escopo
             local, se não      *)
84        (* encontar tenta novamente no escopo que engloba o
             atual. Prossegue-se *)
85        (* assim até encontrar a definição em algum escopo
```

```
            englobante ou até    *)
86        (* encontrar o escopo global. Se em algum lugar for
            encontrado,          *)
87        (* devolve-se a definição. Em caso contrário, devolve
            uma exceção       *)
88        let id = fst nome in
89          (try (match (Amb.busca amb id) with
90                | Amb.EntVar tipo -> (T.ExpVar (A.Var nome,
                     tipo), tipo)
91                | Amb.EntFun _ ->
92                  let msg = "nome de funcao usado como nome de
                         variavel: " ^ id in
93                   failwith (msg_erro nome msg)
94              )
95           with Not_found ->
96                  let msg = "A variavel " ^ id ^ " nao foi
                       declarada" in
97                  failwith (msg_erro nome msg)
98          )
99       | _ -> failwith "infere_exp: não implementado"
100    )
101
102  | S.ExpOp (op, esq, dir) ->
103    let (esq, tesq) = infere_exp amb esq
104    and (dir, tdir) = infere_exp amb dir in
105
106    let verifica_aritmetico () =
107      (match tesq with
108         A.TipoInteiro
109        |A.TipoReal ->
110         let _ = mesmo_tipo (snd op)
111                      "O operando esquerdo eh do tipo %s mas
                           o direito eh do tipo %s"
112                      tesq tdir
113         in tesq (* O tipo da expressão aritmética como um
              todo *)
114
115        | t -> let msg = "um operador aritmetico nao pode ser
             usado com o tipo " ^
116                         (nome_tipo t)
117         in failwith (msg_erro_pos (snd op) msg)
118      )
119
120    and verifica_relacional () =
121      (match tesq with
```

```
122           A.TipoInteiro
123         | A.TipoReal
124         | A.TipoCaractere ->
125           let _ = mesmo_tipo (snd op)
126                     "O operando esquerdo eh do tipo %s mas o
                          direito eh do tipo %s"
127                     tesq tdir
128           in A.TipoBooleano (* O tipo da expressão relacional
                 é sempre booleano *)
129
130         | t -> let msg = "um operador relacional nao pode ser
                  usado com o tipo " ^
131                          (nome_tipo t)
132           in failwith (msg_erro_pos (snd op) msg)
133       )
134
135     and verifica_logico () =
136       (match tesq with
137          A.TipoBooleano ->
138          let _ = mesmo_tipo (snd op)
139                     "O operando esquerdo eh do tipo %s mas o
                          direito eh do tipo %s"
140                     tesq tdir
141          in A.TipoBooleano (* O tipo da expressão lógica é
                 sempre booleano *)
142
143         | t -> let msg = "um operador logico nao pode ser
                  usado com o tipo " ^
144                          (nome_tipo t)
145               in failwith (msg_erro_pos (snd op) msg)
146       )
147   (*  and verifica_cadeia () =
148       (match tesq with
149          A.TipoCaractere ->
150          let _ = mesmo_tipo (snd op)
151                     "O operando esquerdo eh do tipo %s mas o
                          direito eh do tipo %s"
152                     tesq tdir
153          in A.TipoCaractere (* O tipo da expressão relacional
                 é sempre string *)
154
155         | t -> let msg = "um operador relacional nao pode ser
                  usado com o tipo " ^
156                          (nome_tipo t)
157               in failwith (msg_erro_pos (snd op) msg)
```

```
158       )
159   *)
160     in
161     let op = fst op in
162     let tinf = (match (classifica op) with
163           Aritmetico -> verifica_aritmetico ()
164         | Relacional -> verifica_relacional ()
165         | Logico -> verifica_logico ()
166       )
167     in
168       (T.ExpOp ((op,tinf), (esq, tesq), (dir, tdir)), tinf)
169
170   | S.ExpFunCall (nome, args) ->
171      let rec verifica_parametros ags ps fs =
172         match (ags, ps, fs) with
173          (a::ags), (p::ps), (f::fs) ->
174             let _ = mesmo_tipo (posicao a)
175                       "O parametro eh do tipo %s mas deveria
                             ser do tipo %s" p f
176             in verifica_parametros ags ps fs
177        | [], [], [] -> ()
178        | _ -> failwith (msg_erro nome "Numero incorreto de
              parametros")
179      in
180      let id = fst nome in
181      try
182        begin
183          let open Amb in
184
185          match (Amb.busca amb id) with
186          (* verifica se 'nome' está associada a uma função *)
187            Amb.EntFun {tipo_fn; formais} ->
188            (* Infere o tipo de cada um dos argumentos *)
189            let argst = List.map (infere_exp amb) args
190            (* Obtem o tipo de cada parâmetro formal *)
191            and tipos_formais = List.map snd formais in
192            (* Verifica se o tipo de cada argumento confere
                  com o tipo declarado *)
193            (* do parâmetro formal correspondente.
                                                  *)
194            let _ = verifica_parametros args (List.map snd
                  argst) tipos_formais
195             in (T.ExpFunCall (id, (List.map fst argst),
                  tipo_fn), tipo_fn)
196          | Amb.EntVar _ -> (* Se estiver associada a uma vari
```

```
              ável, falhe *)
197           let msg = id ^ " eh uma variavel e nao uma funcao"
                  in
198           failwith (msg_erro nome msg)
199       end
200     with Not_found ->
201       let msg = "Nao existe a funcao de nome " ^ id in
202       failwith (msg_erro nome msg)
203
204 let rec verifica_cmd amb tiporet cmd =
205   let open A in
206    match cmd with
207     Retorne exp ->
208     (match exp with
209      (* Se a função não retornar nada, verifica se ela foi
            declarada como void *)
210       None ->
211       let _ = mesmo_tipo (Lexing.dummy_pos)
212                   "O tipo retornado eh %s mas foi declarado
                       como %s"
213                   TipoVoid tiporet
214       in Retorne None
215     | Some e ->
216       (* Verifica se o tipo inferido para a expressão de
             retorno confere com o *)
217       (* tipo declarado para a função.
                                                        *)
218           let (e1,tinf) = infere_exp amb e in
219           let _ = mesmo_tipo (posicao e)
220                               "O tipo retornado eh %s mas foi
                                    declarado como %s"
221                               tinf tiporet
222           in Retorne (Some e1)
223      )
224   | Se (teste, entao, senao) ->
225     let (teste1,tinf) = infere_exp amb teste in
226     (* O tipo inferido para a expressão 'teste' do
          condicional deve ser booleano *)
227     let _ = mesmo_tipo (posicao teste)
228              "O teste do if deveria ser do tipo %s e nao %s"
229              TipoBooleano tinf in
230     (* Verifica a validade de cada comando do bloco 'então'
          *)
231     let entao1 = List.map (verifica_cmd amb tiporet) entao in
232     (* Verifica a validade de cada comando do bloco 'senão',
```

```
          se houver *)
233     let senao1 =
234         match senao with
235           None -> None
236         | Some bloco -> Some (List.map (verifica_cmd amb
              tiporet) bloco)
237      in
238      Se (teste1, entao1, senao1)
239
240   | Attrib (elem, exp) ->
241     (* Infere o tipo da expressão no lado direito da atribuiç
          ão *)
242     let (exp,  tdir) = infere_exp amb exp
243     (* Faz o mesmo para o lado esquerdo *)
244     and (elem1, tesq) = infere_exp amb elem in
245     (* Os dois tipos devem ser iguais *)
246     let _ = mesmo_tipo (posicao elem)
247                        "Atribuicao com tipos diferentes: %s =
                             %s" tesq tdir
248     in Attrib (elem1, exp)
249
250   | Enquanto (teste, corpo) ->
251     let (teste_tipo,tinf) = infere_exp amb teste in
252     let _ = mesmo_tipo (posicao teste)
253                        "O teste do enquanto deveria ser do
                             tipo %s e nao %s"
254                         TipoBooleano tinf in
255     let corpo_tipo = List.map (verifica_cmd amb tiporet)
          corpo in
256     Enquanto (teste_tipo, corpo_tipo)
257
258  (*| Para (var, inicio, fim, avanco, corpo) ->
259     let (var_tipo, tinfv) =  infere_exp amb var in
260     let (inicio_tipo,tinfi) = infere_exp amb inicio in
261     let (fim_tipo,tinff) = infere_exp amb fim in
262     let (avanco_tipo,tinfa) = infere_exp amb avanco in
263
264     let corpo_tipo = List.map (verifica_cmd amb tiporet)
          corpo in
265     Para (var_tipo,inicio_tipo,fim_tipo,avanco_tipo,
          corpo_tipo)*)
266
267  (*
268     let t0 = (match var.tipo with Some t -> t | None ->
          failwith "tipo" )
```

```
269     and t1 = (match inicio.tipo with Some t -> t | None ->
            failwith
270                       "verifica_cmd->para->t1: tipo indefinido"
                             )
271     and t2 = (match fim.tipo with Some t -> t | None ->
            failwith
272                       "verifica_cmd->para->t2: tipo indefinido"
                             )
273     and t3 = (match avanco with
274                       Some v -> infere_exp amb v;
275                           (match v.tipo with Some t -> t | None
                                    -> failwith
276                       "verifica_cmd->para->t2: tipo indefinido"
                             )
277                       |None -> TipoBooleano
278                   ) in
279     begin
280       if ((t0 <> TipoInteiro ) || (t0 <> t1) || (t0 <> t2) )
              then
281         msg_erro " para " cmd.pcmd " Os tipos deveraim ser
                iguais"
282       else
283         begin
284           ( match v3 with
285           | Some v ->
286                 if (t0 <> t3) then
287                               msg_erro " para " cmd.pcmd " Os
                                     tipos deveraim ser iguais"
288
289           | None -> ignore()
290           );
291           verifica_cmd amb corpo
292         end
293     end *)
294   (* | Para (var,inicio,fim,avanco, corpo) ->
295     let comando = Attrib(var,inicio) in
296       let comando_tipo = verifica_cmd amb tiporet comando in
297         let op = operador_do_for avanco in
298           let expression = S.ExpOp(op, inicio, fim) in
299             let (teste_tipo,tinf) = infere_exp amb expression
                    in
300               let _ = mesmo_tipo (posicao fim)
301                           "O teste do para deveria ser do tip
                                %s e nao %s"
302                            TipoBooleano tinf in
```

```
303                 let comando_avanco = Attrib(var, S.ExpOp(Soma
                        ,var,avanco)) in
304                   let comando_avanco_tipo = verifica_cmd amb
                          tiporet comando_avanco in
305                     let corpo_tipo = verifica_cmd amb tiporet
                            corpo in
306                       let bloco = Bloco([], corpo_tipo::
                              comando_avanco_tipo::[]) in
307                       let cmd_tipo = comando_tipo::Enquanto(
                              teste_tipo, bloco)::[] in
308                       print_list(cmd_tipo)
309                       Bloco([], cmd_tipo)*)
310  (*| Escolha (id, caso, corpo) ->
311    let (teste_id, tinf) = infere_exp amb id in
312    let _ = mesmo_tipo (posicao id)
313                  "O teste do escolha deve ser do tipo %s e
                         nao %s"
314                   TipoInteiro tinf in
315  *)
316 (* | Bloco (decs, cmds) ->
317    let amb_bloco = Amb.novo_escopo amb in
318    let decs_tipo = List.map (verifica_dec amb_bloco) decs in
319    let _ = List.iter (analisa_dec amb_bloco) decs_tipo in
320    let cmds_tipo = List.map (verifica_cmd amb_bloco tiporet)
           cmds in
321    Bloco(decs_tipo, cmds_tipo)*)
322
323  | Chamada exp ->
324    (* Verifica o tipo de cada argumento da função 'entrada'
         *)
325    let (exp,tinf) = infere_exp amb exp in
326    Chamada exp
327
328 | ChamadaRec (elem, exp) ->
329    (* Infere o tipo da expressão no lado direito da atribuiç
         ão *)
330    let (exp,  tdir) = infere_exp amb exp
331    (* Faz o mesmo para o lado esquerdo *)
332    and (elem1, tesq) = infere_exp amb elem in
333    (* Os dois tipos devem ser iguais *)
334    let _ = mesmo_tipo (posicao elem)
335                        "Atribuicao com tipos diferentes: %s =
                               %s" tesq tdir
336    in ChamadaRec (elem1, exp)
337
```

```
338   | StmVarDecl decsvars ->
339    let ambfn = Amb.novo_escopo amb in
340    let insere_local = function
341         (VarDecl(v,t)) -> Amb.insere_local ambfn (fst v)  t
               in
342    StmVarDecl decsvars
343
344
345   (*| Leia (var, exps) ->
346     (* Infere o tipo da expressão no lado direito da atribuiç
          ão *)
347     let (exps,  tdir) = infere_exp amb exps
348     (* Faz o mesmo para o lado esquerdo *)
349     and (elem1, tesq) = infere_exp amb var in
350     (* Os dois tipos devem ser iguais *)
351     let _ = mesmo_tipo (posicao var)
352                        "Atribuicao com tipos diferentes: %s =
                               %s" tesq tdir
353     in Leia (elem1, exps)*)
354
355   | Escreva exps ->
356     (* Verifica o tipo de cada argumento da função 'saida' *)
357     let exps = List.map (infere_exp amb) exps in
358     Escreva (List.map fst exps)
359
360 (*and operador_do_for ini =
361   let open A in
362     let zero = ExpInt( 0 ) in
363       if (ini >= zero) then MenorIgual
364       else MaiorIgual *)
365 and verifica_fun amb ast =
366   let open A in
367   match ast with
368     A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
          fn_corpo} ->
369     (* Estende o ambiente global, adicionando um ambiente
          local *)
370     let ambfn = Amb.novo_escopo amb in
371     (* Insere os parâmetros no novo ambiente *)
372     let insere_parametro (v,t) = Amb.insere_param ambfn (fst
          v) t in
373     let _ = List.iter insere_parametro fn_params in
374     (* Insere as variáveis locais no novo ambiente *)
375     let insere_local = function
376         (DecVar(v,t)) -> Amb.insere_local ambfn (fst v)  t in
```

```
377     let _ = List.iter insere_local fn_locais in
378     (* Verifica cada comando presente no corpo da função
           usando o novo ambiente *)
379     let corpo_tipado = List.map (verifica_cmd ambfn
           fn_tiporet) fn_corpo in
380       A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
             fn_corpo = corpo_tipado}
381
382
383 let rec verifica_dup xs =
384   match xs with
385     [] -> []
386   | (nome,t)::xs ->
387     let id = fst nome in
388     if (List.for_all (fun (n,t) -> (fst n) <> id) xs)
389     then (id, t) :: verifica_dup xs
390     else let msg = "Parametro duplicado " ^ id in
391       failwith (msg_erro nome msg)
392
393 let insere_declaracao_var amb dec =
394   let open A in
395     match dec with
396       DecVar(nome, tipo) ->  Amb.insere_local amb (fst nome)
             tipo
397
398 let insere_declaracao_fun amb dec =
399   let open A in
400     match dec with
401       FuncDecl {fn_id; fn_params; fn_tiporet;  fn_corpo} ->
402         (* Verifica se não há parâmetros duplicados *)
403         let formais = verifica_dup fn_params in
404         let nome = fst fn_id in
405         Amb.insere_fun amb nome formais fn_tiporet
406
407 (*let analisa_dec amb dec =
408   let open A in
409     match dec with
410       DecVar(nome, tipo) ->  Amb.insere_local amb (fst nome)
             tipo
411       |FuncDecl {fn_id; fn_params; fn_tiporet;  fn_corpo} ->
412         (* Verifica se não há parâmetros duplicados *)
413         let _ = verifica_dup fn_params in
414         Amb.insere_fun amb fn_id fn_params fn_tiporet
415
416 let verifica_dec amb dec =
```

```
417     let open A in
418     match amb with
419     A.DecVar {nome; tipo}  ->
420         A.DecVar {nome; tipo}
421    | A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
          fn_corpo} ->
422     (* Estende o ambiente global, adicionando um ambiente
          local *)
423     let ambfn = Amb.novo_escopo amb in
424     (* Insere os parâmetros no novo ambiente *)
425     let insere_parametro (v,t) = Amb.insere_param ambfn (fst
          v) t in
426     let _ = List.iter insere_parametro fn_params in
427     (* Insere as variáveis locais no novo ambiente *)
428     let insere_local = function
429         (DecVar(v,t)) -> Amb.insere_local ambfn (fst v)  t in
430     let _ = List.iter insere_local fn_locais in
431     (* Verifica cada comando presente no corpo da função
          usando o novo ambiente *)
432     let corpo_tipado = List.map (verifica_cmd ambfn
          fn_tiporet) fn_corpo in
433       A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
            fn_corpo = corpo_tipado}
434 *)
435
436 (* Lista de cabeçalhos das funções pré definidas *)
437 let fn_predefs = let open A in [
438    ("escreva", [("x", TipoCaractere); ("y", TipoInteiro)],
          TipoVoid);
439    ("leia",   [("x", TipoReal); ("y", TipoInteiro)], TipoVoid
          )
440 ]
441
442 (* insere as funções pré definidas no ambiente global *)
443 let declara_predefinidas amb =
444   List.iter (fun (n,ps,tr) -> Amb.insere_fun amb n ps tr)
         fn_predefs
445
446 let semantico ast =
447   (* cria ambiente global inicialmente vazio *)
448   let amb_global = Amb.novo_amb [] in
449   let _ = declara_predefinidas amb_global in
450   let (A.Prog (decs_globais, decs_funs, corpo)) = ast in
451   let _ = List.iter (insere_declaracao_var amb_global)
         decs_globais in
```

```
452   let _ = List.iter (insere_declaracao_fun amb_global)
          decs_funs in
453   (* Verificação de tipos nas funções *)
454   let decs_funs = List.map (verifica_fun amb_global)
          decs_funs in
455   (* Verificação de tipos na função principal *)
456   let corpo = List.map (verifica_cmd amb_global A.TipoVoid)
          corpo in
457      (A.Prog (decs_globais, decs_funs, corpo),  amb_global)
```

8.11 Compilação do Analisador Semântico

Para compilar o analisador semântico implementado de acordo com os arquivos mostrados na seção anterior, será utilizado o Menhir e o arquivo "semanticoTest". A compilação pode ser feita através do sguinte comando:

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

8.12 Teste do Analisador Semântico

Para testar o analisador semântico será usado o seguinte arquivo com o programa fonte escrito em JavaScript:

```
var numero, fat : int;
function fatorial(n){
if (n <= 0)
return 1;
else
return n * fatorial(n-1);
}

numero = 10;
fat = fatorial(numero);
```

O teste é feito a partir dos seguintes comandos:

```
rlwrap ocaml
verifica_tipos "teste.js";;
```

A árvore tipada gerada pelo analisador semântico para o programa fonte apresentado é mostrada a seguir.

Listing 94: Árvore Tipada

```
 Tast.expressao Ast.prog * Ambiente.t =
(Prog
  ([DecVar
     (("x", {Lexing.pos_fname = ""; pos_lnum = 2; pos_bol =
        1; pos_cnum = 5}),
     TipoInteiro);
    DecVar
     (("n",
       {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 14;
          pos_cnum = 18}),
     TipoInteiro)],
  [FuncDecl
    {fn_id =
      ("fatorial",
       {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 28;
          pos_cnum = 37});
     fn_params =
      [(("n",
         {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 28;
            pos_cnum = 46}),
        TipoInteiro)];
     fn_tiporet = TipoInteiro; fn_locais = [];
     fn_corpo =
      [Se
        (Tast.ExpOp ((MenorIgual, TipoBooleano),
          (Tast.ExpVar
            (Var
              ("n",
               {Lexing.pos_fname = ""; pos_lnum = 6; pos_bol
                  = 63;
                pos_cnum = 68}),
            TipoInteiro),
           TipoInteiro),
          (Tast.ExpInt (0, TipoInteiro), TipoInteiro)),
        [Retorne (Some (Tast.ExpInt (1, TipoInteiro)))],
        Some
         [Retorne
           (Some
             (Tast.ExpOp ((Mult, TipoInteiro),
               (Tast.ExpVar
                 (Var
                   ("n",
                    {Lexing.pos_fname = ""; pos_lnum = 10;
                       pos_bol = 96;
```

```
39                      pos_cnum = 104}),
40                  TipoInteiro),
41                 TipoInteiro),
42                (Tast.ExpFunCall ("fatorial",
43                  [Tast.ExpOp ((Sub, TipoInteiro),
44                    (Tast.ExpVar
45                      (Var
46                        ("n",
47                         {Lexing.pos_fname = ""; pos_lnum =
                              10; pos_bol = 96;
48                          pos_cnum = 115}),
49                      TipoInteiro),
50                     TipoInteiro),
51                    (Tast.ExpInt (1, TipoInteiro), TipoInteiro
                        ))],
52                  TipoInteiro),
53                 TipoInteiro))))])]}],
54   [Attrib
55     (Tast.ExpVar
56       (Var
57         ("n",
58          {Lexing.pos_fname = ""; pos_lnum = 14; pos_bol =
                128;
59           pos_cnum = 128}),
60       TipoInteiro),
61     Tast.ExpInt (10, TipoInteiro));
62    ChamadaRec
63     (Tast.ExpVar
64       (Var
65         ("x",
66          {Lexing.pos_fname = ""; pos_lnum = 15; pos_bol =
                134;
67           pos_cnum = 134}),
68       TipoInteiro),
69     Tast.ExpFunCall ("fatorial",
70      [Tast.ExpVar
71        (Var
72          ("n",
73           {Lexing.pos_fname = ""; pos_lnum = 15; pos_bol =
                 134;
74            pos_cnum = 147}),
75        TipoInteiro)],
76      TipoInteiro))]),
77  <abstr>)
```

9 Bibliografia

1. *Guia do .NET - CLR (Common Language Runtime)*
2. *.Net Framework - Common Language Runtime)*
3. *Understanding Common Intermediate Language (CIL)*
4. *Documentação do Mono*
5. *Lista de instruções CIL*
6. *Trabalho de Construção de Compiladores - Portugol para CLR (Eduardo Costa de Paiva)*