# Construção de um compilador de MiniC para Common Language Runtime (CLR) usando Objective Caml

Bruno Sérgio Cardoso Vieira
brunoscvieira@gmail.com

Faculdade de Computação
Universidade Federal de Uberlândia

12 de junho de 2018

# Lista de Figuras

# Lista de Tabelas

# Lista de Listagens

# Sumário

# Capítulo 1

# Introdução

## 1.1 Compilador

Um compilador é um programa de computador que transforma código escrito em uma linguagem de programação (linguagem fonte, normalmente uma linguagem de alto nível - Java, C, Python) em outra linguagem de programação (linguagem objetivo, normalmente uma linguagem de baixo nível, como Assembly e código de máquina).

## 1.2 Visão geral

Este documento apresenta informações iniciais sobre a construção de um compilador cuja linguagem fonte é MiniC (versão com instruções reduzidas da linguagem C) e linguagem objetivo a CIL, que é a linguagem utilizada pela Common Language Runtime para execução.

## 1.3 Cenário

A seguir, serão expostos algumas informações sobre o cenário em que este trabalho foi desenvolvido.

### 1.3.1 MiniC

A linguagem MiniC é constituída a partir de uma seção de instruções mais simples que compõem a linguagem C original, ou seja, o conjunto de instruções foi limitado às instruções necessárias para a elaboração de programas simples e de forma suficiente para entender o processo de construção de um compilador.

### 1.3.2 Common Intermediate Language - CIL

A Common Intermediate Language, inicialmente chamada de Microsoft Intermediate Language (MSIL), é a linguagem de baixo nível utilizada pelo framework .NET, da Microsoft, e pela Mono runtime. Durante a compilação, a linguagem de origem é traduzida para CIL ao invés de código-objeto específico de uma plataforma ou processador. O código CIL é, então, assemblado em bytecode, que é posteriormente executado pela máquina virtual (CLR).

#### Pilha

A CIL é uma linguagem orientada a objetos e é executada utilizando pilha, ou seja, as instruções utilizam os valores armazenados na pilha (removendo-os na execução do comando) e armazenam o valor de retorno novamente na pilha. Exemplo:
**...**
**.maxstack 2** // define o numero máximo de valores na pilha
**...**

**ldc.i4 5** // carrega o valor 5 na pilha
**ldc.i4 20** // carrega o valor 20 na pilha
**sub** // remove os valores 5 e 20 da pilha, calcula (5 - 20) e adiciona o resultado -15 na pilha
**call void [mscorlib]System.Console::WriteLine (int32)** // printa o resultado presente na pilha
**...**

#### Lista de instruções CIL

Lista com algumas instruções mais simples e comuns em programas na linguagem CIL.

1. **add** - adiciona dois valores, retornando o resultado da soma na pilha
2. **and** - operação AND bit a bit
3. **beq LABEL** - direciona o fluxo de execução para LABEL se valores são iguais
4. **bge LABEL** - direciona o fluxo de execução para LABEL se valor1 >= valor2
5. **bgt LABEL** - direciona o fluxo de execução para LABEL se valor1 > valor2
6. **ble LABEL** - direciona o fluxo de execução para LABEL se valor1 <= valor2
7. **blt LABEL** - direciona o fluxo de execução para LABEL se valor1 < valor2
8. **bne.un LABEL** - direciona o fluxo de execução para LABEL se valor1 != valor2
9. **br LABEL** - direciona, incondicionalmente, o fluxo de execução para LABEL
10. **call METHOD** - chamada de função descrita por METHOD
11. **div** - divide dois valores e retorna o quociente na pilha

12. **ldarg.0** - carrega na pilha o primeiro argumento da função
13. **ldc.i4 VALUE** - carrega na pilha o inteiro de 4 bytes (int32) de valor VALUE
14. **ldc.r4 VALUE** - carrega na pilha o real de 4 bytes -float- (float32) de valor VALUE
15. **ldc.r8 VALUE** - carrega na pilha o real de 8 bytes -double- (float64) de valor VALUE
16. **ldloc.0** - carrega na pilha o valor da variavel local 0
17. **ldloc VARIABLE** - carrega na pilha o valor da variavel local VARIABLE
18. **ldstr "text"** - carrega na pilha a string text
19. **mul** - multiplica dois valores e retorna o resultado na pilha
20. **not** - operador não lógico
21. **or** - operação OR bit a bit
22. **pop** - remover valor da pilha
23. **rem** - divide dois valores e retorna o resto na pilha
24. **stloc.0** - salva o valor da pilha na variável 0
25. **stloc** VARIABLE - salva o valor da pilha na variável VARIABLE
26. **sub** - subtrai o valor2 do valor1, retornando o resultado da subtração na pilha

### Exemplo de programa em linguagem CIL

A seguir, segue um exemplo de um programa em linguagem CIL com comentários que descrevem a função dos comandos utilizados.

Listagem 1.1: Nano 6 - CIL

```
1  // instruções iniciadas com . indicam instruções especiais (diretivas)
2
3  .assembly extern mscorlib {} // importação da biblioteca pré-compilada
       para operações de entrada e saída
4
5  .assembly Function // declaração do nome do Assembly
6  {
7    .ver 1:0:1:0 // versão do código
8  }
9  .module function.exe // declaração do módulo (mínimo 1 por assembly)
10
11 .method static void main() cil managed // declaração do método main
12 {
13   .maxstack 2 // número máximo de valores na pilha simultaneamente
14   .entrypoint // indica que esse método é a entrada do assembly (main)
15   .locals init (int32 numero, int32 x) // declaração de variáveis (int32,
         char, float32)
16
17   ldstr "Digite um número: " // (LOAD STRING) carregar string para função
         print na pilha
```

```
18
19    // chamada da biblioteca para printar (recebendo parâmetro string)
20    call void [mscorlib]System.Console::Write (string)
21
22    // chamada da biblioteca para leitura de uma linha (coloca a string lida
          na pilha)
23    call string [mscorlib]System.Console::ReadLine ()
24
25    // chamada da biblioteca para parsear a string entrada em um inteiro (
          coloca o int32 resultante na pilha)
26    call int32 [mscorlib]System.Int32::Parse (string)
27
28    stloc numero // (STORE LOCAL) salva o int32 da pilha na variavel numero
29
30    ldloc numero // (LOAD LOCAL) carrega o valor da variavel numero na pilha
31    call int32 verifica(int32) // chama a função verifica (com parametro
          int32) e coloca o int32 de retorno na pilha
32    stloc x // salva o int32 da pilha (resultado da função) na variável x
33
34    ldloc x // carrega o valor da variavel x na pilha
35    ldc.i4 1 // (LOAD CONSTANT) alocação de um inteiro de 4 bytes (int32)
          com valor 1 - colocado na pilha
36    beq IF1 // compara os dois valores na pilha (X e 1) - pula para IF1 se X
          == 1 (os dois valores são removidos da pilha)
37      ldloc x // carrega o valor de x na pilha
38      ldc.i4 0 // carrega um int32 de valor 0 na pilha
39      beq IF2 // pula para IF2 se X == 0 (remove os dois valores da pilha)
40        ldstr "Número negativo" // carrega a string na pilha
41        call void [mscorlib]System.Console::WriteLine (string) // printa a
              string
42        br END // pula para o fim dos testes
43      IF2: // label utilizado nos jumps
44        ldstr "Zero" // carrega a string na pilha
45        call void [mscorlib]System.Console::WriteLine (string) // printa a
              string
46        br END // pula para o fim dos testes
47    IF1: // label utilizado nos jumps
48      ldstr "Número positivo" // carrega a string na pilha
49      call void [mscorlib]System.Console::WriteLine (string) // printa a
            string
50      br END // pula para o fim dos testes
51
52    END: // fim dos testes
53    ret // retorno (termino) da função
54  }
55
56  // declaração de uma função chamada verifica, que recebe um parametro
        int32 e retorna um int32
57  .method public static int32 verifica (int32) cil managed
58  {
59    .maxstack 2 // número máximo de valores na pilha
60
61    .locals init (int32 res) // inicialização de uma váriavel int32 de nome
          res
62
63    ldarg.0 // (LOAD ARGUMENT) carrega na pilha o valor do primeiro
          argumento
64    ldc.i4 0 // carrega para pilha um inteiro de 4 bytes de valor 0
```

```
65    ble ELSE // compara o parametro da função com 0 (remove os dois valores
          da pilha)
66      ldc.i4 1 // carrega para a pilha um inteiro de 4 bytes de valor 1
67      stloc res // salva o valor 1 na variavel res
68      br END // pula para o fim dos testes
69    ELSE: // arg <= 0
70      ldarg.0 // carrega novamente o valor do argumento na pilha
71      ldc.i4 0 // carrega na pilha um int32 de valor 0
72      bge ELSE2 // pula para ELSE2 se arg >= 0
73        ldc.i4 -1 // carrega na pilha um int32 de valor -1
74        stloc res // salva o valor -1 na variavel res
75        br END // pula para o fim dos testes
76      ELSE2: // arg >= 0
77        ldc.i4 0 // carrega 0 na pilha
78        stloc res // salva 0 na variavel res
79        br END // pula para o fim dos testes
80    END: // fim dos testes
81    ldloc res // carrega o valor da variável res na pilha antes de retornar
82    ret // retorna da função
83  }
```

# Capítulo 2

# Configuração

## 2.1 Introdução

Neste capítulo, serão apresentadas as formas de instalação e configuração necessários para execução do projeto.

## 2.2 Instalação

Foi utilizado o Sistema Operacional Ubuntu 16.04 para a realização deste trabalho.

### 2.2.1 Ocaml

Embora ainda não tenha sido utilizado nesta seção do projeto, o Ocaml será utilizado futuramente como linguagem para o desenvolvimento do compilador. No terminal, digite:

**sudo apt-get install ocaml ocaml-native-compilers ocaml-doc tuareg-mode ocaml-findlib oasis libpcre-ocaml-dev**

### 2.2.2 Mono

No terminal, digite:

**sudo apt-get install mono-runtime monodevelop**

## 2.3 Execução

### 2.3.1 Compilar arquivo .il

Para compilar um arquivo com extensão .il, siga os seguintes passos:

1. Abra o Terminal
2. Navegue até a pasta na qual está localizado o arquivo .il
3. Digite: ilasm nome-do-arquivo.il

### 2.3.2 Executar arquivo .exe

Para executar o arquivo com extensão .exe gerado pela compilação, siga os seguintes passos:

1. Abra o Terminal
2. Navegue até a pasta na qual está localizado o arquivo .exe
3. Digite: mono nome-do-arquivo.exe

# Capítulo 3

# Programas de teste

Neste capítulo, serão apresentados os algoritmos propostos em linguagem C e suas respectivas traduções para a linguagem CIL.

## 3.1 Nano Programas

São programas simples, de caráter exploratório, para verificar as estruturas básicas e a combinação destas estruturas em programas de baixa complexidade.

### 3.1.1 Nano 1

Módulo mínimo que caracteriza um programa.

**Listagem 3.1: Nano 1 - Linguagem C**

```
1 int main(){}
```

**Listagem 3.2: Nano 1 - CIL**

```
1  .assembly extern mscorlib {}
2
3  .assembly Vazio
4  {
5    .ver 1:0:1:0
6  }
7  .module vazio.exe
8
9  .method static void main() cil managed
10 {
11   .maxstack 1
12   .entrypoint
13   ret
14 }
```

Saída:
Em branco

### 3.1.2 Nano 2

Declaração de uma variável.

**Listagem 3.3: Nano 2 - Linguagem C**

```
1 int main() {
2   int n;
3 }
```

**Listagem 3.4: Nano 2 - CIL**

```
1 .assembly extern mscorlib {}
2
3 .assembly DefineVariable
4 {
5   .ver 1:0:1:0
6 }
7 .module defineVariable.exe
8
9 .method static void main() cil managed
10 {
11   .maxstack 1
12   .entrypoint
13   .locals init (int32 n)
14   ret
15 }
```

Saída:
Em branco

### 3.1.3 Nano 3

Atribuição de um inteiro a uma variável.

**Listagem 3.5: Nano 3 - Linguagem C**

```
1 int main() {
2   int n;
3   n = 1;
4 }
```

**Listagem 3.6: Nano 3 - CIL**

```
1 .assembly extern mscorlib {}
2 https://www.overleaf.com/14835423cnbfnrsvtqwt#
3 .assembly Assign
4 {
```

```
5   .ver 1:0:1:0
6 }
7 .module assign.exe
8
9 .method static void main() cil managed
10 {
11   .maxstack 1
12   .entrypoint
13   .locals init (int32 n)
14   ldc.i4 1
15   call void [mscorlib]System.Console::Write (int32)
16   ret
17 }
```

Saída:
1

### 3.1.4 Nano 4

Atribuição de uma soma de inteiro a uma variável.

**Listagem 3.7: Nano 4 - Linguagem C**

```
1 int main() {
2   int n;
3   n = 1 + 2;
4 }
```

**Listagem 3.8: Nano 4 - CIL**

```
1 .assembly extern mscorlib {}
2
3 .assembly Add
4 {
5   .ver 1:0:1:0
6 }
7 .module add.exe
8
9 .method static void main() cil managed
10 {
11   .maxstack 2
12   .entrypoint
13   .locals init (int32 n)
14   ldc.i4 1
15   ldc.i4 2
16   add
17   call void [mscorlib]System.Console::Write (int32)
18   ret
19 }
```

Saída:
3

### 3.1.5 Nano 5

Inclusão do comando de impressão.

**Listagem 3.9: Nano 5 - Linguagem C**

```
#import <stdio.h>

int main(){
  int n;
  n = 2;
  printf("%d", n);
}
```

**Listagem 3.10: Nano 5 - CIL**

```
.assembly extern mscorlib {}

.assembly AssignAndPrint
{
  .ver 1:0:1:0
}
.module assignAndPrint.exe

.method static void main() cil managed
{
  .maxstack 1
  .entrypoint
  .locals init (int32 n)
  ldc.i4 2
  call void [mscorlib]System.Console::Write (int32)
  ret
}
```

Saída:
2

### 3.1.6 Nano 6

Atribuição de uma subtração de inteiro a uma variável.

**Listagem 3.11: Nano 6 - Linguagem C**

```
#import <stdio.h>

int main(){
  int n;
  n = 1 - 2;
  printf("%d", n);
}
```

**Listagem 3.12: Nano 6 - CIL**

```
.assembly extern mscorlib {}

.assembly SubAndPrint
{
  .ver 1:0:1:0
}
.module subAndPrint.exe

.method static void main() cil managed
{
  .maxstack 2
  .entrypoint
  .locals init (int32 n)
  ldc.i4 1
  ldc.i4 2
  sub
  call void [mscorlib]System.Console::Write (int32)
  ret
}
```

Saída:
-1

### 3.1.7 Nano 7

Inclusão do comando condicional.

**Listagem 3.13: Nano 7 - Linguagem C**

```
#import <stdio.h>

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

**Listagem 3.14: Nano 7 - CIL**

```
.assembly extern mscorlib {}

.assembly IfEqual
{
  .ver 1:0:1:0
}
.module ifEqual.exe

.method static void main() cil managed
{
  .maxstack 2
  .entrypoint
  .locals init (int32 n)
  ldc.i4 1
```

```
15   stloc n
16   ldloc n
17   ldc.i4 1
18   bne.un Different
19     ldloc n
20     call void [mscorlib]System.Console::Write (int32)
21     br EndIf
22 Different:
23 EndIf:
24   ret
25 }
```

Saída:
1

### 3.1.8 Nano 8

Inclusão do comando condicional com parte senão.

**Listagem 3.15: Nano 8 - Linguagem C**

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

**Listagem 3.16: Nano 8 - CIL**

```
1 .assembly extern mscorlib {}
2
3 .assembly IfEqualElse
4 {
5   .ver 1:0:1:0
6 }
7 .module ifEqualElse.exe
8
9 .method static void main() cil managed
10 {
11   .maxstack 2
12   .entrypoint
13   .locals init (int32 n)
14   ldc.i4 1
15   stloc n
16   ldloc n
17   ldc.i4 1
18   bne.un Different
19     ldloc n
20     call void [mscorlib]System.Console::Write (int32)
```

```
21     br EndIf
22 Different:
23     ldc.i4 0
24     call void [mscorlib]System.Console::Write (int32)
25 EndIf:
26   ret
27 }
```

Saída:
0

### 3.1.9 Nano 9

Atribuição de duas operações aritméticas sobre inteiros a uma variável.

**Listagem 3.17: Nano 9 - Linguagem C**

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

**Listagem 3.18: Nano 9 - CIL**

```
1 .assembly extern mscorlib {}
2
3 .assembly AssignIfElse
4 {
5   .ver 1:0:1:0
6 }
7 .module assignIfElse.exe
8
9 .method static void main() cil managed
10 {
11   .maxstack 2
12   .entrypoint
13   .locals init (int32 n)
14   ldc.i4 1
15   ldc.i4 1
16   add
17   ldc.i4 2
18   div
19   stloc n
20   ldloc n
21   ldc.i4 1
22   bne.un Different
23     ldloc n
24     call void [mscorlib]System.Console::Write (int32)
```

```
25     br EndIf
26 Different:
27     ldc.i4 0
28     call void [mscorlib]System.Console::Write (int32)
29 EndIf:
30   ret
31 }
```

Saída:
0

### 3.1.10 Nano 10

Atribuição de duas variáveis inteiras.

**Listagem 3.19: Nano 10 - Linguagem C**

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

**Listagem 3.20: Nano 10 - CIL**

```
1 .assembly extern mscorlib {}
2
3 .assembly CompareVariables
4 {
5   .ver 1:0:1:0
6 }
7 .module compareVariables.exe
8
9 .method static void main() cil managed
10 {
11   .maxstack 2
12   .entrypoint
13   .locals init (int32 n, int32 m)
14   ldc.i4 1
15   stloc n
16   ldc.i4 2
17   stloc m
18   ldloc n
19   ldloc m
20   bne.un Different
21     ldloc n
22     call void [mscorlib]System.Console::Write (int32)
23     br EndIf
```

```
24 Different:
25     ldc.i4 0
26     call void [mscorlib]System.Console::Write (int32)
27 EndIf:
28   ret
29 }
```

Saída:
0

### 3.1.11 Nano 11

Introdução do comando de repetição enquanto.

**Listagem 3.21: Nano 11 - Linguagem C**

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

**Listagem 3.22: Nano 11 - CIL**

```
1 .assembly extern mscorlib {}
2
3 .assembly While
4 {
5   .ver 1:0:1:0
6 }
7 .module while.exe
8
9 .method static void main() cil managed
10 {
11   .maxstack 3
12   .entrypoint
13   .locals init (int32 n, int32 m, int32 x)
14   ldc.i4 1
15   stloc n
16   ldc.i4 2
17   stloc m
18   ldc.i4 5
19   stloc x
20
21 StartWhile:
22   ldloc x
23   ldloc n
24   ble ExitWhile
```

```
25
26    ldloc n
27    ldloc m
28    add
29    stloc n
30    ldloc n
31    call void [mscorlib]System.Console::Write (int32)
32    br StartWhile
33
34  ExitWhile:
35    ret
36  }
```

Saída:
35

### 3.1.12 Nano 12

Comando condicional aninhado em um comando de repetição.

**Listagem 3.23: Nano 12 - Linguagem C**

```
1  #import <stdio.h>
2
3  int main(){
4    int n, m, x;
5    n = 1;
6    m = 2;
7    x = 5;
8    while(x > n){
9      if(n == m){
10       printf("%d", n);
11     } else {
12       printf("0");
13     }
14     x = x - 1;
15   }
16 }
```

**Listagem 3.24: Nano 12 - CIL**

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

```
15   stloc n
16   ldc.i4 2
17   stloc m
18   ldc.i4 5
19   stloc x
20
21   StartWhile:
22   ldloc x
23   ldloc n
24   ble ExitWhile
25
26     ldloc n
27     ldloc m
28     bne.un ElseIf
29       ldloc n
30       call void [mscorlib]System.Console::Write (int32)
31       br EndIf
32     ElseIf:
33       ldc.i4 0
34       call void [mscorlib]System.Console::Write (int32)
35       br EndIf
36     EndIf:
37     ldloc x
38     ldc.i4 1
39     sub
40     stloc x
41     br StartWhile
42
43   ExitWhile:
44   ret
45 }
```

Saída:
0000

## 3.2 Micro Programas

São programas de maior complexidade, envolvendo cálculos e operações de entrada e saída mais elaboradas.

### 3.2.1 Micro 1

Converte graus Celcius para Fahreinheit.

**Listagem 3.25: Micro 1 - Linguagem C**

```
1 #import <stdio.h>
2
3 int main(){
4   float cel, far;
5   printf(" Tabela de conversão: Celsius -> Fahrenheit\n");
```

**Figura 3.1:** *Saída do programa Micro 1*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro01.exe
   Tabela de conversão: Celsius -> Fahrenheit
Digite a temperatura em Celsius: 25.4
A nova temperatura é: 77.7200012207031 F
```

```
6   printf("Digite a temperatura em Celsius: ");
7   scanf("%f", &cel);
8   far = (9 * cel + 160) / 5;
9   printf("A nova temperatura é: %f F\n", far);
10 }
```

**Listagem 3.26: Micro 1 - CIL**

```
1  .assembly extern mscorlib {}
2
3  .assembly CelsiusFahrenheit
4  {
5    .ver 1:0:1:0
6  }
7  .module celsiusFahrenheit.exe
8
9  .method static void main() cil managed
10 {
11   .maxstack 2
12   .entrypoint
13   .locals init (float32 cel, float32 far)
14   ldstr "   Tabela de conversão: Celsius -> Fahrenheit"
15   call void [mscorlib]System.Console::WriteLine (string)
16   ldstr "Digite a temperatura em Celsius: "
17   call void [mscorlib]System.Console::Write (string)
18   call string [mscorlib]System.Console::ReadLine ()
19   call float32 [mscorlib]System.Single::Parse (string)
20   stloc cel
21   ldc.r4 9
22   ldloc cel
23   mul
24   ldc.r4 160
25   add
26   ldc.r4 5
27   div
28   stloc far
29   ldstr "A nova temperatura é: "
30   call void [mscorlib]System.Console::Write (string)
31   ldloc far
32   call void [mscorlib]System.Console::Write (float64)
33   ldstr " F"
34   call void [mscorlib]System.Console::WriteLine (string)
35   ret
36 }
```

Saída:

### 3.2.2 Micro 2

Ler dois inteiros e decide qual é maior.

**Listagem 3.27: Micro 2 - Linguagem C**

```c
#import <stdio.h>

int main(){
  int num1, num2;
  printf("Digite o primeiro número: ");
  scanf("%d", &num1);
  printf("Digite o segundo número: ");
  scanf("%d", &num2);
  if(num1 > num2){
    printf("O primeiro número %d é maior que o segundo %d", num1, num2);
  } else {
    printf("O segundo número %d é maior que o primeiro %d", num2, num1);
  }
}
```

**Listagem 3.28: Micro 2 - CIL**

```
.assembly extern mscorlib {}

.assembly InformBigger
{
  .ver 1:0:1:0
}
.module informBigger.exe

.method static void main() cil managed
{
  .maxstack 2
  .entrypoint
  .locals init (int32 num1, int32 num2)

  ldstr "Digite o primeiro número: "
  call void [mscorlib]System.Console::Write (string)
  call string [mscorlib]System.Console::ReadLine ()
  call int32 [mscorlib]System.Int32::Parse (string)
  stloc num1

  ldstr "Digite o segundo número: "
  call void [mscorlib]System.Console::Write (string)
  call string [mscorlib]System.Console::ReadLine ()
  call int32 [mscorlib]System.Int32::Parse (string)
  stloc num2

  ldloc num1
  ldloc num2
  ble ElseIf
    ldstr "O primeiro número "
    call void [mscorlib]System.Console::Write (string)
    ldloc num1
    call void [mscorlib]System.Console::Write (int32)
    ldstr " é maior que o segundo "
    call void [mscorlib]System.Console::Write (string)
```

**Figura 3.2:** *Saída do programa Micro 2*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro02.exe
Digite o primeiro número: 10
Digite o segundo número: 20
O segundo número 20 é maior que o primeiro 10
```

```
36      ldloc num2
37      call void [mscorlib]System.Console::Write (int32)
38      br EndIf
39    ElseIf:
40      ldstr "O segundo número "
41      call void [mscorlib]System.Console::Write (string)
42      ldloc num2
43      call void [mscorlib]System.Console::Write (int32)
44      ldstr " é maior que o primeiro "
45      call void [mscorlib]System.Console::Write (string)
46      ldloc num1
47      call void [mscorlib]System.Console::Write (int32)
48      br EndIf
49    EndIf:
50    ret
51  }
```

Saída:

### 3.2.3 Micro 3

Lê um número e verifica se ele está entre 100 e 200.

**Listagem 3.29: Micro 3 - Linguagem C**

```
1  #import <stdio.h>
2
3  int main(){
4    int numero;
5    printf("Digite um número: ");
6    scanf("%d", &numero);
7    if(numero >= 100){
8      if(numero <= 200){
9        printf("O número está no intervalo entre 100 e 200\n");
10     } else {
11       printf("O número não está no intervalo entre 100 e 200\n");
12     }
13   } else {
14     printf("O número não está no intervalo entre 100 e 200\n");
15   }
16 }
```

**Listagem 3.30: Micro 3 - CIL**

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

**Figura 3.3:** *Saída do programa Micro 3*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro03.exe
Digite um número: 10
O número não está no intervalo entre 100 e 200
```

```
2
3  .assembly InInterval
4  {
5    .ver 1:0:1:0
6  }
7  .module inInterval.exe
8
9  .method static void main() cil managed
10 {
11   .maxstack 2
12   .entrypoint
13   .locals init (int32 numero)
14   ldstr "Digite um número: "
15   call void [mscorlib]System.Console::Write (string)
16   call string [mscorlib]System.Console::ReadLine ()
17   call int32 [mscorlib]System.Int32::Parse (string)
18   stloc numero
19
20   ldloc numero
21   ldc.i4 100
22
23   blt ElseIf1
24     ldloc numero
25     ldc.i4 200
26     bgt ElseIf2
27       ldstr "O número está no intervalo entre 100 e 200"
28       call void [mscorlib]System.Console::WriteLine (string)
29       br EndIf2
30     ElseIf2:
31       ldstr "O número não está no intervalo entre 100 e 200"
32       call void [mscorlib]System.Console::WriteLine (string)
33       br EndIf2
34     EndIf2:
35     br EndIf1
36   ElseIf1:
37     ldstr "O número não está no intervalo entre 100 e 200"
38     call void [mscorlib]System.Console::WriteLine (string)
39     br EndIf1
40   EndIf1:
41   ret
42 }
```

Saída:

### 3.2.4 Micro 4

Lê números inteiros e informa quantos estão entre 10 e 150.

**Listagem 3.31: Micro 4 - Linguagem C**

```c
#import <stdio.h>

int main(){
  int x, num, intervalo;
  intervalo = 0; // a inicialização da variável intervalo não estava
      presente no algoritmo em Portugol
  for(x = 1; x <= 5; x++){
    printf("Digite um número: ");
    scanf("%d", &num);
    if(num >= 10){
      if(num <= 150){
        intervalo = intervalo + 1;
      }
    }
  }
  printf("Ao total, foram digitados %d números no intervalo entre 10 e
      150\n", intervalo);
}
```

**Listagem 3.32: Micro 4 - CIL**

```
.assembly extern mscorlib {}

.assembly CountInInterval
{
  .ver 1:0:1:0
}
.module countInInterval.exe

.method static void main() cil managed
{
  .maxstack 2
  .entrypoint
  .locals init (int32 x, int32 num, int32 intervalo)
  ldc.i4 0
  stloc intervalo

  ldc.i4 1
  stloc x
  StartFor:
    ldloc x
    ldc.i4 5
    bgt EndFor

    ldstr "Digite um número: "
    call void [mscorlib]System.Console::Write (string)
    call string [mscorlib]System.Console::ReadLine ()
    call int32 [mscorlib]System.Int32::Parse (string)
    stloc num

    ldloc num
    ldc.i4 10
```

**Figura 3.4:** *Saída do programa Micro 4*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro04.exe
Digite um número: 50
Digite um número: 60
Digite um número: 10
Digite um número: 190
Digite um número: 200
Ao total, foram digitados 3 números no intervalo entre 10 e 150
```

```
32      blt EndIf1
33        ldloc num
34        ldc.i4 150
35        bgt EndIf2
36          ldloc intervalo
37          ldc.i4 1
38          add
39          stloc intervalo
40          br EndIf2
41        EndIf2:
42        br EndIf1
43      EndIf1:
44      ldloc x
45      ldc.i4 1
46      add
47      stloc x
48      br StartFor
49    EndFor:
50    ldstr "Ao total, foram digitados "
51    call void [mscorlib]System.Console::Write (string)
52    ldloc intervalo
53    call void [mscorlib]System.Console::Write (int32)
54    ldstr " números no intervalo entre 10 e 150"
55    call void [mscorlib]System.Console::WriteLine (string)
56    ret
57  }
```

Saída:

### 3.2.5 Micro 5

Lê strings e caracteres (nome e gênero) e conta a quantidade de inserções de cada gênero.

**Listagem 3.33: Micro 5 - Linguagem C**

```
1 #import <stdio.h>
2
3 int main(){
4   char nome[15];
5   char sexo;
6   char newLine;
7   int x, h, m;
```

```
8   // a inicialização das variáveis h e m não estava presente no algoritmo
        em Portugol
9   h = 0;
10  m = 0;
11  for(x = 1; x <= 5; x++){
12    printf("Digite o nome: ");
13    scanf("%s", nome);
14    scanf("%c", &newLine);
15    printf("H - Homem ou M - Mulher: ");
16    scanf("%c", &sexo);
17    scanf("%c", &newLine);
18    switch(sexo){
19      case 'H':
20        h = h + 1;
21        break;
22      case 'M':
23        m = m + 1;
24        break;
25      default:
26        printf("Sexo só pode ser H ou M!\n");
27    }
28  }
29  printf("Foram inseridos %d Homens\n", h);
30  printf("Foram inseridos %d Mulheres\n", m);
31 }
```

**Listagem 3.34: Micro 5 - CIL**

```
1  .assembly extern mscorlib {}
2
3  .assembly CountGender
4  {
5    .ver 1:0:1:0
6  }
7  .module countGender.exe
8
9  .method static void main() cil managed
10 {
11   .maxstack 2
12   .entrypoint
13   .locals init (string nome, char sexo, int32 x, int32 h, int32 m)
14   ldc.i4 0
15   stloc h
16   ldc.i4 0
17   stloc m
18
19   ldc.i4 1
20   stloc x
21   StartFor:
22     ldloc x
23     ldc.i4 5
24     bgt EndFor
25
26     ldstr "Digite o nome: "
27     call void [mscorlib]System.Console::Write (string)
28     call string [mscorlib]System.Console::ReadLine ()
29     stloc nome
30     ldstr "H - Homem ou M - Mulher: "
31     call void [mscorlib]System.Console::Write (string)
```

```
32      call string [mscorlib]System.Console::ReadLine ()
33      call char [mscorlib]System.Convert::ToChar (string)
34      stloc sexo
35
36      ldloc sexo
37      ldc.i4.s 0x48 // char H in hexadecimal
38      beq HOMEM
39      ldloc sexo
40      ldc.i4.s 0x4D // char M in hexadecimal
41      beq MULHER
42      ldstr "Sexo só pode ser H ou M"
43      call void [mscorlib]System.Console::WriteLine (string)
44      br ExitSwitch
45
46    HOMEM:
47      ldloc h
48      ldc.i4 1
49      add
50      stloc h
51      br ExitSwitch
52    MULHER:
53      ldloc m
54      ldc.i4 1
55      add
56      stloc m
57      br ExitSwitch
58
59    ExitSwitch:
60      ldloc x
61      ldc.i4 1
62      add
63      stloc x
64      br StartFor
65
66    EndFor:
67    ldstr "Foram inseridos "
68    call void [mscorlib]System.Console::Write (string)
69    ldloc h
70    call void [mscorlib]System.Console::Write (int32)
71    ldstr " Homens"
72    call void [mscorlib]System.Console::WriteLine (string)
73    ldstr "Foram inseridos "
74    call void [mscorlib]System.Console::Write (string)
75    ldloc m
76    call void [mscorlib]System.Console::Write (int32)
77    ldstr " Mulheres"
78    call void [mscorlib]System.Console::WriteLine (string)
79    ret
80  }
```

Saída:

**Figura 3.5:** *Saída do programa Micro 5*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro05.exe
Digite o nome: Teste1
H - Homem ou M - Mulher: H
Digite o nome: Teste2
H - Homem ou M - Mulher: M
Digite o nome: Teste3
H - Homem ou M - Mulher: K
Sexo só pode ser H ou M
Digite o nome: Teste4
H - Homem ou M - Mulher: H
Digite o nome: Teste5
H - Homem ou M - Mulher: H
Foram inseridos 3 Homens
Foram inseridos 1 Mulheres
```

### 3.2.6 Micro 6

Escreve um número lido por extenso.

**Listagem 3.35: Micro 6 - Linguagem C**

```
#import <stdio.h>

int main(){
  int numero;
  printf("Digite um número de 1 a 5: ");
  scanf("%d", &numero);
  switch(numero){
    case 1:
      printf("Um\n");
      break;
    case 2:
      printf("Dois\n");
      break;
    case 3:
      printf("Três\n");
      break;
    case 4:
      printf("Quatro\n");
      break;
    case 5:
      printf("Cinco\n");
      break;
    default:
      printf("Número Inválido!!!\n");
  }
}
```

**Listagem 3.36: Micro 6 - CIL**

```
.assembly extern mscorlib {}

.assembly PrintNumber
{
  .ver 1:0:1:0
```

```
}
.module printNumber.exe

.method static void main() cil managed
{
  .maxstack 2
  .entrypoint
  .locals init (int32 numero)

  ldstr "Digite um número de 1 a 5: "
  call void [mscorlib]System.Console::Write (string)
  call string [mscorlib]System.Console::ReadLine ()
  call int32 [mscorlib]System.Int32::Parse (string)
  stloc numero

  ldloc numero
  ldc.i4 1
  beq UM
  ldloc numero
  ldc.i4 2
  beq DOIS
  ldloc numero
  ldc.i4 3
  beq TRES
  ldloc numero
  ldc.i4 4
  beq QUATRO
  ldloc numero
  ldc.i4 5
  beq CINCO
  ldstr "Número Inválido!!!"
  call void [mscorlib]System.Console::WriteLine (string)
  br EndSwitch

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

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

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

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

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

```

**Figura 3.6:** *Saída do programa Micro 6*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro06.exe
Digite um número de 1 a 5: 4
QUATRO
```

```
65    EndSwitch:
66    ret
67 }
```

Saída:

### 3.2.7 Micro 7

Decide se um número lido é positivo, zero ou negativo.

**Listagem 3.37: Micro 7 - Linguagem C**

```
1  #import <stdio.h>
2
3  int main(){
4    int programa, numero;
5    char opc;
6    programa = 1;
7    while(programa == 1){
8      printf("Digite um número: ");
9      scanf("%d", &numero);
10     if(numero > 0){
11       printf("Positivo\n");
12     } else {
13       if(numero == 0){
14         printf("O número é igual a 0\n");
15       } else {
16         printf("Negativo\n");
17       }
18     }
19     scanf("%c", &opc);
20     printf("Deseja finalizar? (S/N) ");
21     scanf("%c", &opc);
22     if(opc == 'S'){
23       programa = 0;
24     }
25   }
26 }
```

**Listagem 3.38: Micro 7 - CIL**

```
1  .assembly extern mscorlib {}
2
3  .assembly PositiveNegative
4  {
5    .ver 1:0:1:0
```

```
}
.module positiveNegative.exe

.method static void main() cil managed
{
  .maxstack 3
  .entrypoint
  .locals init (int32 numero, int32 programa, char opc)

  ldc.i4 1
  stloc programa

  StartWhile:
    ldloc programa
    ldc.i4 1
    bne.un ExitWhile

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

    ldloc numero
    ldc.i4 0
    ble ElseIf1
      ldstr "Positivo"
      call void [mscorlib]System.Console::WriteLine (string)
      br EndIf1
    ElseIf1:
      ldloc numero
      ldc.i4 0
      beq If2
        ldstr "Negativo"
        call void [mscorlib]System.Console::WriteLine (string)
        br EndIf2
      If2:
        ldstr "O número é igual a 0"
        call void [mscorlib]System.Console::WriteLine (string)
        br EndIf2
      EndIf2:
      br EndIf1
    EndIf1:

    ldstr "Deseja finalizar? (S/N) "
    call void [mscorlib]System.Console::Write (string)
    call string [mscorlib]System.Console::ReadLine ()
    call char [mscorlib]System.Convert::ToChar (string)
    stloc opc

    ldloc opc
    ldc.i4.s 0x53
    bne.un EndIf
      ldc.i4 0
      stloc programa
    EndIf:
    br StartWhile
  ExitWhile:
  ret
```

**Figura 3.7:** *Saída do programa Micro 7*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro07.exe
Digite um número: 10
Positivo
Deseja finalizar? (S/N) N
Digite um número: 0
O número é igual a 0
Deseja finalizar? (S/N) N
Digite um número: -10
Negativo
Deseja finalizar? (S/N) S
```

```
65 }
```

Saída:

### 3.2.8 Micro 8

Decide se um número é maior ou menor que 10.

**Listagem 3.39: Micro 8 - Linguagem C**

```
1 #import <stdio.h>
2
3 int main(){
4   int numero;
5   numero = 1;
6   while(numero != 0){
7     printf("Digite um número: ");
8     scanf("%d", &numero);
9     if(numero > 10){
10      printf("O número %d é maior que 10\n", numero);
11    } else {
12      printf("O número %d é menor que 10\n", numero);
13    }
14  }
15 }
```

**Listagem 3.40: Micro 8 - CIL**

```
1 .assembly extern mscorlib {}
2
3 .assembly Compare10
4 {
5   .ver 1:0:1:0
6 }
7 .module compare10.exe
8
9 .method static void main() cil managed
10 {
11   .maxstack 2
```

**Figura 3.8:** *Saída do programa Micro 8*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro08.exe
Digite um número: 50
O número 50 é maior que 10
Digite um número: 5
O número 5 é menor que 10
Digite um número: 0
O número 0 é menor que 10
```

```
12    .entrypoint
13    .locals init (int32 numero)
14    ldc.i4 1
15    stloc numero
16
17    StartWhile:
18    ldloc numero
19    ldc.i4 0
20    beq ExitWhile
21
22      ldstr "Digite um número: "
23      call void [mscorlib]System.Console::Write (string)
24      call string [mscorlib]System.Console::ReadLine ()
25      call int32 [mscorlib]System.Int32::Parse (string)
26      stloc numero
27
28      ldloc numero
29      ldc.i4 10
30      ble ElseIf
31        ldstr "O número "
32        call void [mscorlib]System.Console::Write (string)
33        ldloc numero
34        call void [mscorlib]System.Console::Write (int32)
35        ldstr " é maior que 10"
36        call void [mscorlib]System.Console::WriteLine (string)
37        br EndIf
38      ElseIf:
39        ldstr "O número "
40        call void [mscorlib]System.Console::Write (string)
41        ldloc numero
42        call void [mscorlib]System.Console::Write (int32)
43        ldstr " é menor que 10"
44        call void [mscorlib]System.Console::WriteLine (string)
45        br EndIf
46      EndIf:
47      br StartWhile
48    ExitWhile:
49    ret
50  }
```

Saída:

### 3.2.9 Micro 9

Cálculo de alteração de preço baseado em preço e número de vendas.

**Listagem 3.41: Micro 9 - Linguagem C**

```c
#import <stdio.h>

int main(){
  float preco, venda, novo_preco;
  printf("Digite o preço: ");
  scanf("%f", &preco);
  printf("Digite a venda: ");
  scanf("%f", &venda);
  if(venda < 500 || preco < 30){
    novo_preco = preco + (10.0/100 * preco);
  } else {
    if((venda >= 500 && venda < 1200) || (preco >= 30 && preco < 80)){
      novo_preco = preco + (15.0/100 * preco);
    } else {
      if(venda >= 1200 || preco >= 80){
        novo_preco = preco - (20.0/100 * preco);
      }
    }
  }
  printf("O novo preço é %f\n", novo_preco);
}
```

**Listagem 3.42: Micro 9 - CIL**

```
.assembly extern mscorlib {}

.assembly NewPrice
{
  .ver 1:0:1:0
}
.module newPrice.exe

.method static void main() cil managed
{
  .maxstack 4
  .entrypoint
  .locals init (float32 preco, float32 venda, float32 novo_preco)

  ldc.r4 0.0
  stloc novo_preco

  ldstr "Digite o preço: "
  call void [mscorlib]System.Console::Write (string)
  call string [mscorlib]System.Console::ReadLine ()
  call float32 [mscorlib]System.Single::Parse (string)
  stloc preco

  ldstr "Digite a venda: "
  call void [mscorlib]System.Console::Write (string)
  call string [mscorlib]System.Console::ReadLine ()
  call float32 [mscorlib]System.Single::Parse (string)
  stloc venda
```

```
29
30    ldloc venda
31    ldc.r4 500
32    blt NEWPRICE1
33
34    ldloc preco
35    ldc.r4 30
36    blt NEWPRICE1
37
38    // SecondIf
39    // FirstCase
40    ldloc venda
41    ldc.r4 500
42    blt SecondCase
43
44    ldloc venda
45    ldc.r4 1200
46    bge SecondCase
47
48    br NEWPRICE2
49
50    SecondCase:
51    ldloc preco
52    ldc.r4 30
53    blt SecondElse
54
55    ldloc preco
56    ldc.r4 80
57    bge SecondElse
58
59    br NEWPRICE2
60
61    SecondElse:
62    ldloc venda
63    ldc.r4 1200
64    bge NEWPRICE3
65
66    ldloc preco
67    ldc.r4 80
68    bge NEWPRICE3
69
70    br AfterNewPrice
71
72  NEWPRICE1:
73    ldc.r4 10
74    ldc.r4 100
75    div
76    ldloc preco
77    mul
78    ldloc preco
79    add
80    stloc novo_preco
81    br AfterNewPrice
82
83  NEWPRICE2:
84    ldc.r4 15
85    ldc.r4 100
86    div
87    ldloc preco
```

**Figura 3.9:** *Saída do programa Micro 9*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro09.exe
Digite o preço: 100
Digite a venda: 200
O novo preço é 110
```

```
88    mul
89    ldloc preco
90    add
91    stloc novo_preco
92    br AfterNewPrice
93
94  NEWPRICE3:
95    ldloc preco
96    ldc.r4 20
97    ldc.r4 100
98    div
99    ldloc preco
100   mul
101   sub
102   stloc novo_preco
103   br AfterNewPrice
104
105   AfterNewPrice:
106   ldstr "O novo preço é "
107   call void [mscorlib]System.Console::Write (string)
108   ldloc novo_preco
109   call void [mscorlib]System.Console::WriteLine (float32)
110   ret
111 }
```

Saída:

### 3.2.10 Micro 10

Cálculo de fatorial usando recursão.

**Listagem 3.43: Micro 10 - Linguagem C**

```
1  #import <stdio.h>
2
3  int fatorial(int n);
4
5  int main(){
6    int numero, fat;
7    printf("Digite um número: ");
8    scanf("%d", &numero);
9    fat = fatorial(numero);
10   printf("O fatorial de ");
11   printf("%d", numero);
12   printf(" é ");
```

```
13   printf("%d\n", fat);
14 }
15
16 int fatorial(int n){
17   if(n <= 0){
18     return 1;
19   } else {
20     return n * fatorial(n-1);
21   }
22 }
```

**Listagem 3.44: Micro 10 - CIL**

```
1  .assembly extern mscorlib {}
2
3  .assembly WhileIf
4  {
5    .ver 1:0:1:0
6  }
7  .module whileIf.exe
8
9  .method static void main() cil managed
10 {
11   .maxstack 2
12   .entrypoint
13   .locals init (int32 numero, int32 fat)
14
15   ldstr "Digite um número: "
16   call void [mscorlib]System.Console::Write (string)
17   call string [mscorlib]System.Console::ReadLine ()
18   call int32 [mscorlib]System.Int32::Parse (string)
19   stloc numero
20
21   ldloc numero
22   call int32 fatorial(int32)
23   stloc fat
24
25   ldstr "O fatorial de "
26   call void [mscorlib]System.Console::Write (string)
27   ldloc numero
28   call void [mscorlib]System.Console::Write (int32)
29   ldstr " é "
30   call void [mscorlib]System.Console::Write (string)
31   ldloc fat
32   call void [mscorlib]System.Console::WriteLine (int32)
33   ret
34 }
35
36 .method public static int32 fatorial (int32) cil managed
37 {
38   .maxstack 2
39
40   ldarg.0
41   ldc.i4 0
42   bgt ELSE
43     ldc.i4 1
44     br ENDIF
45   ELSE:
46     ldarg.0
```

**Figura 3.10:** *Saída do programa Micro 10*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro10.exe
Digite um número: 5
O fatorial de 5 é 120
```

```
47      ldc.i4 1
48      sub
49      call int32 fatorial(int32)
50      ldarg.0
51      mul
52      br ENDIF
53    ENDIF:
54    ret
55  }
```

Saída:

### 3.2.11 Micro 11

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

**Listagem 3.45: Micro 11 - Linguagem C**

```c
#import <stdio.h>

int verifica(int n);

int main(){
  int numero, x;
  printf("Digite um número: ");
  scanf("%d", &numero);
  x = verifica(numero);
  if(x == 1){
    printf("Número positivo\n");
  } else {
    if(x == 0){
      printf("Zero\n");
    } else {
      printf("Número negativo\n");
    }
  }
}

int verifica(int n){
  int res;
  if(n > 0){
    res = 1;
  } else {
    if(n < 0){
      res = -1;
    } else {
```

```
29        res = 0;
30      }
31    }
32    return res;
33  }
```

Listagem 3.46: Micro 11 - CIL

```
1  .assembly extern mscorlib {}
2
3  .assembly Function
4  {
5    .ver 1:0:1:0
6  }
7  .module function.exe
8
9  .method static void main() cil managed
10 {
11   .maxstack 2
12   .entrypoint
13   .locals init (int32 numero, int32 x)
14
15   ldstr "Digite um número: "
16   call void [mscorlib]System.Console::Write (string)
17   call string [mscorlib]System.Console::ReadLine ()
18   call int32 [mscorlib]System.Int32::Parse (string)
19   stloc numero
20
21   ldloc numero
22   call int32 verifica(int32)
23   stloc x
24
25   ldloc x
26   ldc.i4 1
27   beq IF1
28     ldloc x
29     ldc.i4 0
30     beq IF2
31       ldstr "Número negativo"
32       call void [mscorlib]System.Console::WriteLine (string)
33       br END
34     IF2:
35       ldstr "Zero"
36       call void [mscorlib]System.Console::WriteLine (string)
37       br END
38   IF1:
39     ldstr "Número positivo"
40     call void [mscorlib]System.Console::WriteLine (string)
41     br END
42
43   END:
44   ret
45 }
46
47 .method public static int32 verifica (int32) cil managed
48 {
49   .maxstack 2
50
51   .locals init (int32 res)
```

**Figura 3.11:** *Saída do programa Micro 11*

```
compilers@ubuntu:~/Desktop/compiler_construction/ilasm/micro$ mono micro11.exe
Digite um número: 50
Número positivo
```

```
52
53    ldarg.0
54    ldc.i4 0
55    ble ELSE
56      ldc.i4 1
57      stloc res
58      br END
59    ELSE:
60      ldarg.0
61      ldc.i4 0
62      bge ELSE2
63        ldc.i4 -1
64        stloc res
65        br END
66      ELSE2:
67        ldc.i4 0
68        stloc res
69        br END
70    END:
71    ldloc res
72    ret
73  }
```

Saída:

# Capítulo 4

# Analisador Léxico

Nesta seção, será apresentada a construção do analisador léxico para a linguagem miniC, incluindo a definição dos tokens e o código desenvolvido para retornar a lista de tokens à partir de um programa escrito em miniC. Também serão apresentados os resultados da execução do analisador léxico sobre os programas de teste utilizados neste relatório (nano e micro).

### 4.0.1 Lista de tokens

A lista de tokens do analisador léxico foi definida com base nas palavras reservadas, operadores lógicos, operadores aritméticos e outros símbolos de marcação da linguagem miniC. A lista é apresentada abaixo na forma (TOKEN - "sequência de caracteres reconhecidos"):

**Palavras-chave:**

1. BREAK - "break"
2. CASE - "case"
3. DEFAULT - "default"
4. DO - "do"
5. ELSE - "else"
6. FALSE - "false"
7. FOR - "for"
8. IF - "if"
9. INCLUDE - "include"
10. NULL - "null"
11. RETURN - "return"

12. SWITCH - "switch"
13. TRUE - "true"
14. WHILE - "while"

**Delimitadores de bloco:**

1. OPEN_PARENTHESIS - ’(’
2. CLOSE_PARENTHESIS - ’)’
3. OPEN_BRACKETS - ’[’
4. CLOSE_BRACKETS - ’]’
5. OPEN_CURLED_BRACKETS - ’’
6. CLOSE_CURLED_BRACKETS - ’’

**Marcadores:**

1. ADDRESS - ’&’
2. COLON - ’:’
3. COMA - ’,’
4. EOF - ’eof’ (delimita final do arquivo)
5. SEMICOLON - ’;’

**Tipos:**

1. BOOL - ’bool’
2. CHAR - ’char’
3. FLOAT - ’float’
4. INTEGER - ’integer’
5. VOID - ’void’

**Operações lógicas:**

1. EQUALS - ’==’
2. DIFFERENT - ’!=’
3. LESS_THAN - ’<’

4. MORE_THAN - '>'
5. LESS_EQUAL_THAN - '<='
6. MORE_EQUAL_THAN - '>='
7. NOT - '!'
8. OR - '||'
9. AND - '&&'

**Operações aritméticas:**

1. ATTRIBUTION - '='
2. ADDITION - '+'
3. SUBTRACTION - '-'
4. MULTIPLICATION - '*'
5. DIVISION - '/'
6. MODULE - '%'
7. INCREMENT - '++'
8. DECREMENT - '–'
9. ADD_ATTRIBUTION - '+='
10. SUB_ATTRIBUTION - '-='
11. MUL_ATTRIBUTION - '*='
12. DIV_ATTRIBUTION - '/='

**Literais:**

1. LITERAL_INTEGER - número inteiro
2. LITERAL_FLOAT - número decimal com ponto
3. LITERAL_CHAR - único caracter entre aspas simples
4. LITERAL_STRING - sequência de caracteres entre aspas duplas
5. HEADER_FILE - cabeçalho de arquivo local ou biblioteca terminado em ".h"
6. 

**Outros:**

1. ID - nome de variáveis ou funções (sequência de caracteres não iniciados com número)
2. ARROW_OPERATION - '->'

### 4.0.2 Código do analisador léxico

A seguir, é apresentado o código do gerador analisador léxico, considerando os tokens definidos na seção anterior. Neste código, são definidos tokens e suas regras de conversão. A execução do gerador (.mll) pela ferramenta Ocamllex produz um arquivo .ml, que é posteriormente utilizado pelo compilador Ocaml.

**Listagem 4.1: Lexical.mll**

```
{
  open Lexing
  open Printf

  type tokens =

              (* BLOCKS *)
              | OPEN_PARENTHESIS
              | CLOSE_PARENTHESIS
              | OPEN_BRACKETS
              | CLOSE_BRACKETS
              | OPEN_CURLED_BRACKETS
              | CLOSE_CURLED_BRACKETS

              (* MARKERS *)
              | ADDRESS (* & *)
              | COLON
              | COMA
              | EOF
              | SEMICOLON

              (* KEYWORDS *)
              | BREAK
              | CASE
              | DEFAULT
              | DO
              | ELSE
              | FALSE
              | FOR
              | IF
              | INCLUDE
              | NULL
              | RETURN
              | SWITCH
              | TRUE
              | WHILE

              (* TYPES *)
              | BOOL
              | CHAR
              | FLOAT
              | INTEGER
              | VOID

              (* LOGIC OPERATIONS *)
              | EQUALS
              | DIFFERENT
              | LESS_THAN
              | MORE_THAN
```

```
50                | LESS_EQUAL_THAN
51                | MORE_EQUAL_THAN
52                | NOT
53                | OR (* || *)
54                | AND (* && *)
55
56                (* BINARY OPERATIONS *)
57                | ATTRIBUTION
58                | ADDITION
59                | SUBTRACTION
60                | MULTIPLICATION
61                | DIVISION
62                | MODULE
63                | INCREMENT (* incrementar (++) *)
64                | DECREMENT (* decrementar (--) *)
65                | ADD_ATTRIBUTION (* += *)
66                | SUB_ATTRIBUTION (* -= *)
67                | MUL_ATTRIBUTION (* *= *)
68                | DIV_ATTRIBUTION (* /= *)
69
70                (* LITERALS *)
71                | LITERAL_INTEGER of int
72                | LITERAL_FLOAT of float
73                | LITERAL_CHAR of char
74                | LITERAL_STRING of string
75                | HEADER_FILE of string
76
77                (* OTHERS *)
78                | ID of string
79                | ARROW_OPERATION (* -> *)
80
81
82    let incr_num_linha lexbuf =
83      let pos = lexbuf.lex_curr_p in
84       lexbuf.lex_curr_p <- { pos with
85          pos_lnum = pos.pos_lnum + 1;
86          pos_bol = pos.pos_cnum;
87       }
88
89    let msg_erro lexbuf c =
90      let pos = lexbuf.lex_curr_p in
91      let lin = pos.pos_lnum
92      and col = pos.pos_cnum - pos.pos_bol - 1 in
93      sprintf "%d-%d: caracter desconhecido %c" lin col c
94
95    let erro lin col msg =
96      let mensagem = sprintf "%d-%d: %s" lin col msg in
97         failwith mensagem
98
99  }
100
101 let digito = ['0' - '9']
102 let inteiro = digito+
103 let numero = (digito* '.' digito+) | (digito+ '.' digito+)
104
105 let letra = ['a' - 'z' 'A' - 'Z']
106 let identificador = letra ( letra | digito | '_')*
107 let caracter = '\'' letra '\''
108
```

```
109 let brancos = [' ' '\t']+
110 let novalinha = '\r' | '\n' | "\r\n"
111
112 let comentario = "//" [^ '\r' '\n' ]*
113
114 let header_file = ('<' letra+ '.' 'h' '>') | ('"' letra+ '.' 'h' '"')
115
116 rule token = parse
117   brancos     { token lexbuf }
118 | novalinha   { incr_num_linha lexbuf; token lexbuf }
119 | comentario { token lexbuf }
120 | "/*"        { comentario_bloco 0 lexbuf }
121 | '('         { OPEN_PARENTHESIS }
122 | ')'         { CLOSE_PARENTHESIS }
123 | '{'         { OPEN_CURLED_BRACKETS }
124 | '}'         { CLOSE_CURLED_BRACKETS }
125 | '['         { OPEN_BRACKETS }
126 | ']'         { CLOSE_BRACKETS }
127 | '+'         { ADDITION }
128 | '-'         { SUBTRACTION }
129 | '*'         { MULTIPLICATION }
130 | '/'         { DIVISION }
131 | '%'         { MODULE }
132 | '='         { ATTRIBUTION }
133 | ','         { COMA }
134 | ';'         { SEMICOLON }
135 | ':'         { COLON }
136 | '&'         { ADDRESS }
137 | '>'         { MORE_THAN }
138 | '<'         { LESS_THAN }
139 | '!'         { NOT }
140 | "=="        { EQUALS }
141 | "!="        { DIFFERENT }
142 | ">="        { MORE_EQUAL_THAN }
143 | "<="        { LESS_EQUAL_THAN }
144 | "||"        { OR }
145 | "&&"        { AND }
146 | "++"        { INCREMENT }
147 | "--"        { DECREMENT }
148 | "+="        { ADD_ATTRIBUTION }
149 | "-="        { SUB_ATTRIBUTION }
150 | "*="        { MUL_ATTRIBUTION }
151 | "/="        { DIV_ATTRIBUTION }
152 | "->"        { ARROW_OPERATION }
153 | "int"       { INTEGER }
154 | "float"     { FLOAT }
155 | "char"      { CHAR }
156 | "bool"      { BOOL }
157 | "if"        { IF }
158 | "else"      { ELSE }
159 | "while"     { WHILE }
160 | "do"        { DO }
161 | "for"       { FOR }
162 | "switch"    { SWITCH }
163 | "case"      { CASE }
164 | "break"     { BREAK }
165 | "default"   { DEFAULT }
166 | "null"      { NULL }
167 | "true"      { TRUE }
```

```
168 | "false"    { FALSE }
169 | "void"     { VOID }
170 | "return"   { RETURN }
171 | "#include" { token lexbuf }
172
173 | '"'        { let pos = lexbuf.lex_curr_p in
174                 let lin = pos.pos_lnum
175                 and col = pos.pos_cnum - pos.pos_bol - 1 in
176                 let buffer = Buffer.create 1 in
177                 let str = leia_string lin col buffer lexbuf in
178                   LITERAL_STRING str }
179
180 | header_file as hf { token lexbuf }
181
182 | inteiro as num { let numero = int_of_string num in
183                     LITERAL_INTEGER numero }
184
185 | numero as num { let n = float_of_string num in
186                     LITERAL_FLOAT n }
187
188 | caracter as char_string { if ((String.length char_string) == 3) then
      LITERAL_CHAR (char_string.[1])
189                             else let pos = lexbuf.lex_curr_p in
190                                  let lin = pos.pos_lnum
191                                  and col = pos.pos_cnum - pos.pos_bol - 1
                                         in
192                                  erro lin col "Caracter não fechado"}
193
194 | identificador as id { ID id }
195
196 | _ as c  { failwith (msg_erro lexbuf c) }
197
198 | eof        { EOF }
199
200 (* AUXILIAR RULES *)
201
202 and comentario_bloco n = parse
203    "*/"      { if n=0 then token lexbuf
204                else comentario_bloco (n-1) lexbuf }
205 | "/*"       { comentario_bloco (n+1) lexbuf }
206 | novalinha  { incr_num_linha lexbuf; comentario_bloco n lexbuf }
207 | _          { comentario_bloco n lexbuf }
208 | eof        { failwith "Comentário não fechado" }
209
210 and leia_string lin col buffer = parse
211   '"'       { Buffer.contents buffer }
212 | "\\t"     { Buffer.add_char buffer '\t'; leia_string lin col buffer
      lexbuf }
213 | "\\n"     { Buffer.add_char buffer '\n'; leia_string lin col buffer
      lexbuf }
214 | '\\' '"'  { Buffer.add_char buffer '"'; leia_string lin col buffer
      lexbuf }
215 | '\\' '\\' { Buffer.add_char buffer '\\'; leia_string lin col buffer
      lexbuf }
216 | novalinha       { erro lin col "A string não foi fechada"}
217 | _ as c    { Buffer.add_char buffer c; leia_string lin col buffer lexbuf
      }
218 | eof       { erro lin col "A string não foi fechada"}
```

### 4.0.3 Código auxiliar

A seguir, é apresentado o código de um módulo de carregamento para facilitar a construção e execução do analisador léxico:

**Listagem 4.2: Loader.ml**

```
#load "lexical.cmo";;

open Lexical;;

let rec tokens lexbuf =
  let tok = Lexical.token lexbuf in
  match tok with
  | Lexical.EOF -> [Lexical.EOF]
  | _ -> tok :: tokens lexbuf
;;

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

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

### 4.0.4 Execução

Para compilar o gerador do analisador léxico, use o comando:
**ocamllex lexical.mll**

Para compilar o arquivo .ml gerado, use o comando:
**ocamlc -c lexical.ml**

Para iniciar o intérprete do ocaml, use o comando:
**ocaml**

Dentro do intérprete, utilize o módulo de carregamento com o seguinte comando:
**#use "loader.ml";;**

Para executar o analisador léxico sobre um arquivo de entrada, use o comando:
**lex "nome_do_arquivo";;**

### 4.0.5 Resultados

A seguir, são apresentados os resultados da execução do analisador léxico nos programas de teste utilizados durante o desenvolvimento.

**Figura 4.1:** *Saída do analisador léxico para o programa Nano 01*

```
# lex "miniC//nano01.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.2:** *Saída do analisador léxico para o programa Nano 02*

```
# lex "miniC//nano02.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; SEMICOLON; CLOSE_CURLED_BRACKETS;
 EOF]
```

**Figura 4.3:** *Saída do analisador léxico para o programa Nano 03*

```
# lex "miniC//nano03.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; SEMICOLON; ID "n"; ATTRIBUTION;
 LITERAL_INTEGER 1; SEMICOLON; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.4:** *Saída do analisador léxico para o programa Nano 04*

```
# lex "miniC//nano04.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; SEMICOLON; ID "n"; ATTRIBUTION;
 LITERAL_INTEGER 1; ADDITION; LITERAL_INTEGER 2; SEMICOLON;
 CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.5:** *Saída do analisador léxico para o programa Nano 05*

```
# lex "miniC//nano05.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; SEMICOLON; ID "n"; ATTRIBUTION;
 LITERAL_INTEGER 2; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%d"; COMA; ID "n"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.6:** *Saída do analisador léxico para o programa Nano 06*

```
# lex "miniC//nano06.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; SEMICOLON; ID "n"; ATTRIBUTION;
 LITERAL_INTEGER 1; SUBTRACTION; LITERAL_INTEGER 2; SEMICOLON; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ID "n"; CLOSE_PARENTHESIS;
 SEMICOLON; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.7:** *Saída do analisador léxico para o programa Nano 07*

```
# lex "miniC//nano07.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; SEMICOLON; ID "n"; ATTRIBUTION;
 LITERAL_INTEGER 1; SEMICOLON; IF; OPEN_PARENTHESIS; ID "n"; EQUALS;
 LITERAL_INTEGER 1; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ID "n"; CLOSE_PARENTHESIS;
 SEMICOLON; CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.8:** *Saída do analisador léxico para o programa Nano 08*

```
# lex "miniC//nano08.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; SEMICOLON; ID "n"; ATTRIBUTION;
 LITERAL_INTEGER 1; SEMICOLON; IF; OPEN_PARENTHESIS; ID "n"; EQUALS;
 LITERAL_INTEGER 1; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ID "n"; CLOSE_PARENTHESIS;
 SEMICOLON; CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "0"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.9:** *Saída do analisador léxico para o programa Nano 09*

```
# lex "miniC//nano09.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; SEMICOLON; ID "n"; ATTRIBUTION;
 LITERAL_INTEGER 1; ADDITION; LITERAL_INTEGER 1; DIVISION; LITERAL_INTEGER 2;
 SEMICOLON; IF; OPEN_PARENTHESIS; ID "n"; EQUALS; LITERAL_INTEGER 1;
 CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%d"; COMA; ID "n"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "0"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.10:** *Saída do analisador léxico para o programa Nano 10*

```
# lex "miniC//nano10.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; COMA; ID "m"; SEMICOLON; ID "n";
 ATTRIBUTION; LITERAL_INTEGER 1; SEMICOLON; ID "m"; ATTRIBUTION;
 LITERAL_INTEGER 2; SEMICOLON; IF; OPEN_PARENTHESIS; ID "n"; EQUALS;
 ID "m"; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ID "n"; CLOSE_PARENTHESIS;
 SEMICOLON; CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "0"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.11:** *Saída do analisador léxico para o programa Nano 11*

```
# lex "miniC//nano11.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; COMA; ID "m"; COMA; ID "x";
 SEMICOLON; ID "n"; ATTRIBUTION; LITERAL_INTEGER 1; SEMICOLON; ID "m";
 ATTRIBUTION; LITERAL_INTEGER 2; SEMICOLON; ID "x"; ATTRIBUTION;
 LITERAL_INTEGER 5; SEMICOLON; WHILE; OPEN_PARENTHESIS; ID "x"; MORE_THAN;
 ID "n"; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "n"; ATTRIBUTION;
 ID "n"; ADDITION; ID "m"; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%d"; COMA; ID "n"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.12:** *Saída do analisador léxico para o programa Nano 12*

```
# lex "miniC//nano12.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "n"; COMA; ID "m"; COMA; ID "x";
 SEMICOLON; ID "n"; ATTRIBUTION; LITERAL_INTEGER 1; SEMICOLON; ID "m";
 ATTRIBUTION; LITERAL_INTEGER 2; SEMICOLON; ID "x"; ATTRIBUTION;
 LITERAL_INTEGER 5; SEMICOLON; WHILE; OPEN_PARENTHESIS; ID "x"; MORE_THAN;
 ID "n"; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; IF; OPEN_PARENTHESIS;
 ID "n"; EQUALS; ID "m"; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS;
 ID "printf"; OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ID "n";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; ELSE;
 OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS; LITERAL_STRING "0";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; ID "x"; ATTRIBUTION;
 ID "x"; SUBTRACTION; LITERAL_INTEGER 1; SEMICOLON; CLOSE_CURLED_BRACKETS;
 CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.13:** *Saída do analisador léxico para o programa Micro 01*

```
# lex "miniC//micro01.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; FLOAT; ID "cel"; COMA; ID "far"; SEMICOLON;
 ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING " Tabela de convers\195\163o: Celsius -> Fahrenheit\n";
 CLOSE_PARENTHESIS; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Digite a temperatura em Celsius: "; CLOSE_PARENTHESIS;
 SEMICOLON; ID "scanf"; OPEN_PARENTHESIS; LITERAL_STRING "%f"; COMA; ADDRESS;
 ID "cel"; CLOSE_PARENTHESIS; SEMICOLON; ID "far"; ATTRIBUTION;
 OPEN_PARENTHESIS; LITERAL_INTEGER 9; MULTIPLICATION; ID "cel"; ADDITION;
 LITERAL_INTEGER 160; CLOSE_PARENTHESIS; DIVISION; LITERAL_INTEGER 5;
 SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "A nova temperatura \195\169: %f F\n"; COMA; ID "far";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.14:** *Saída do analisador léxico para o programa Micro 02*

```
# lex "miniC//micro02.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "num1"; COMA; ID "num2"; SEMICOLON;
 ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Digite o primeiro n\195\186mero: "; CLOSE_PARENTHESIS;
 SEMICOLON; ID "scanf"; OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ADDRESS;
 ID "num1"; CLOSE_PARENTHESIS; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Digite o segundo n\195\186mero: "; CLOSE_PARENTHESIS;
 SEMICOLON; ID "scanf"; OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ADDRESS;
 ID "num2"; CLOSE_PARENTHESIS; SEMICOLON; IF; OPEN_PARENTHESIS; ID "num1";
 MORE_THAN; ID "num2"; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS;
 LITERAL_STRING "O primeiro n\195\186mero %d \195\169 maior que o segundo %d";
 COMA; ID "num1"; COMA; ID "num2"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS;
 LITERAL_STRING "O segundo n\195\186mero %d \195\169 maior que o primeiro %d";
 COMA; ID "num2"; COMA; ID "num1"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.15:** *Saída do analisador léxico para o programa Micro 03*

```
# lex "miniC//micro03.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "numero"; SEMICOLON; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "Digite um n\195\186mero: ";
 CLOSE_PARENTHESIS; SEMICOLON; ID "scanf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%d"; COMA; ADDRESS; ID "numero"; CLOSE_PARENTHESIS;
 SEMICOLON; IF; OPEN_PARENTHESIS; ID "numero"; MORE_EQUAL_THAN;
 LITERAL_INTEGER 100; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; IF;
 OPEN_PARENTHESIS; ID "numero"; LESS_EQUAL_THAN; LITERAL_INTEGER 200;
 CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "O n\195\186mero est\195\161 no intervalo entre 100 e 200\n";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; ELSE;
 OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING
  "O n\195\186mero n\195\163o est\195\161 no intervalo entre 100 e 200\n";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS;
 ELSE; OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING
  "O n\195\186mero n\195\163o est\195\161 no intervalo entre 100 e 200\n";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS;
 EOF]
```

**Figura 4.16:** *Saída do analisador léxico para o programa Micro 04*

```
# lex "miniC//micro04.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "x"; COMA; ID "num"; COMA; ID "intervalo";
 SEMICOLON; ID "intervalo"; ATTRIBUTION; LITERAL_INTEGER 0; SEMICOLON; FOR;
 OPEN_PARENTHESIS; ID "x"; ATTRIBUTION; LITERAL_INTEGER 1; SEMICOLON;
 ID "x"; LESS_EQUAL_THAN; LITERAL_INTEGER 5; SEMICOLON; ID "x"; INCREMENT;
 CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Digite um n\195\186mero: "; CLOSE_PARENTHESIS; SEMICOLON;
 ID "scanf"; OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ADDRESS; ID "num";
 CLOSE_PARENTHESIS; SEMICOLON; IF; OPEN_PARENTHESIS; ID "num";
 MORE_EQUAL_THAN; LITERAL_INTEGER 10; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; IF; OPEN_PARENTHESIS; ID "num"; LESS_EQUAL_THAN;
 LITERAL_INTEGER 150; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS;
 ID "intervalo"; ATTRIBUTION; ID "intervalo"; ADDITION; LITERAL_INTEGER 1;
 SEMICOLON; CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS;
 CLOSE_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING
  "Ao total, foram digitados %d n\195\186meros no intervalo entre 10 e 150\n";
 COMA; ID "intervalo"; CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS;
 EOF]
```

**Figura 4.17:** *Saída do analisador léxico para o programa Micro 05*

```
# lex "miniC//micro05.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; CHAR; ID "nome"; OPEN_BRACKETS; LITERAL_INTEGER 15;
 CLOSE_BRACKETS; SEMICOLON; CHAR; ID "sexo"; SEMICOLON; CHAR; ID "newLine";
 SEMICOLON; INTEGER; ID "x"; COMA; ID "h"; COMA; ID "m"; SEMICOLON; ID "h";
 ATTRIBUTION; LITERAL_INTEGER 0; SEMICOLON; ID "m"; ATTRIBUTION;
 LITERAL_INTEGER 0; SEMICOLON; FOR; OPEN_PARENTHESIS; ID "x"; ATTRIBUTION;
 LITERAL_INTEGER 1; SEMICOLON; ID "x"; LESS_EQUAL_THAN; LITERAL_INTEGER 5;
 SEMICOLON; ID "x"; INCREMENT; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS;
 ID "printf"; OPEN_PARENTHESIS; LITERAL_STRING "Digite o nome: ";
 CLOSE_PARENTHESIS; SEMICOLON; ID "scanf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%s"; COMA; ID "nome"; CLOSE_PARENTHESIS; SEMICOLON;
 ID "scanf"; OPEN_PARENTHESIS; LITERAL_STRING "%c"; COMA; ADDRESS;
 ID "newLine"; CLOSE_PARENTHESIS; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "H - Homem ou M - Mulher: "; CLOSE_PARENTHESIS; SEMICOLON;
 ID "scanf"; OPEN_PARENTHESIS; LITERAL_STRING "%c"; COMA; ADDRESS; ID "sexo";
 CLOSE_PARENTHESIS; SEMICOLON; ID "scanf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%c"; COMA; ADDRESS; ID "newLine"; CLOSE_PARENTHESIS;
 SEMICOLON; SWITCH; OPEN_PARENTHESIS; ID "sexo"; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; CASE; LITERAL_CHAR 'H'; COLON; ID "h"; ATTRIBUTION;
 ID "h"; ADDITION; LITERAL_INTEGER 1; SEMICOLON; BREAK; SEMICOLON; CASE;
 LITERAL_CHAR 'M'; COLON; ID "m"; ATTRIBUTION; ID "m"; ADDITION;
 LITERAL_INTEGER 1; SEMICOLON; BREAK; SEMICOLON; DEFAULT; COLON; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "Sexo s\195\179 pode ser H ou M!\n";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS;
 ID "printf"; OPEN_PARENTHESIS; LITERAL_STRING "Foram inseridos %d Homens\n";
 COMA; ID "h"; CLOSE_PARENTHESIS; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Foram inseridos %d Mulheres\n"; COMA; ID "m";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.18:** *Saída do analisador léxico para o programa Micro 06*

```
# lex "miniC//micro06.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "numero"; SEMICOLON; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "Digite um n\195\186mero de 1 a 5: ";
 CLOSE_PARENTHESIS; SEMICOLON; ID "scanf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%d"; COMA; ADDRESS; ID "numero"; CLOSE_PARENTHESIS;
 SEMICOLON; SWITCH; OPEN_PARENTHESIS; ID "numero"; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; CASE; LITERAL_INTEGER 1; COLON; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "Um\n"; CLOSE_PARENTHESIS; SEMICOLON;
 BREAK; SEMICOLON; CASE; LITERAL_INTEGER 2; COLON; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "Dois\n"; CLOSE_PARENTHESIS; SEMICOLON;
 BREAK; SEMICOLON; CASE; LITERAL_INTEGER 3; COLON; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "Tr\195\170s\n"; CLOSE_PARENTHESIS;
 SEMICOLON; BREAK; SEMICOLON; CASE; LITERAL_INTEGER 4; COLON; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "Quatro\n"; CLOSE_PARENTHESIS; SEMICOLON;
 BREAK; SEMICOLON; CASE; LITERAL_INTEGER 5; COLON; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "Cinco\n"; CLOSE_PARENTHESIS; SEMICOLON;
 BREAK; SEMICOLON; DEFAULT; COLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "N\195\186mero Inv\195\161lido!!!\n"; CLOSE_PARENTHESIS;
 SEMICOLON; CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.19:** *Saída do analisador léxico para o programa Micro 07*

```
# lex "miniC//micro07.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "programa"; COMA; ID "numero"; SEMICOLON;
 CHAR; ID "opc"; SEMICOLON; ID "programa"; ATTRIBUTION; LITERAL_INTEGER 1;
 SEMICOLON; WHILE; OPEN_PARENTHESIS; ID "programa"; EQUALS;
 LITERAL_INTEGER 1; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "Digite um n\195\186mero: ";
 CLOSE_PARENTHESIS; SEMICOLON; ID "scanf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%d"; COMA; ADDRESS; ID "numero"; CLOSE_PARENTHESIS;
 SEMICOLON; IF; OPEN_PARENTHESIS; ID "numero"; MORE_THAN; LITERAL_INTEGER 0;
 CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Positivo\n"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; IF; OPEN_PARENTHESIS;
 ID "numero"; EQUALS; LITERAL_INTEGER 0; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "O n\195\186mero \195\169 igual a 0\n"; CLOSE_PARENTHESIS;
 SEMICOLON; CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "Negativo\n"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS; ID "scanf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%c"; COMA; ADDRESS; ID "opc"; CLOSE_PARENTHESIS; SEMICOLON;
 ID "printf"; OPEN_PARENTHESIS; LITERAL_STRING "Deseja finalizar? (S/N) ";
 CLOSE_PARENTHESIS; SEMICOLON; ID "scanf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%c"; COMA; ADDRESS; ID "opc"; CLOSE_PARENTHESIS; SEMICOLON;
 IF; OPEN_PARENTHESIS; ID "opc"; EQUALS; LITERAL_CHAR 'S'; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; ID "programa"; ATTRIBUTION; LITERAL_INTEGER 0;
 SEMICOLON; CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS;
 CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.20:** *Saída do analisador léxico para o programa Micro 08*

```
# lex "miniC//micro08.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; INTEGER; ID "numero"; SEMICOLON; ID "numero";
 ATTRIBUTION; LITERAL_INTEGER 1; SEMICOLON; WHILE; OPEN_PARENTHESIS;
 ID "numero"; DIFFERENT; LITERAL_INTEGER 0; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Digite um n\195\186mero: "; CLOSE_PARENTHESIS; SEMICOLON;
 ID "scanf"; OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ADDRESS;
 ID "numero"; CLOSE_PARENTHESIS; SEMICOLON; IF; OPEN_PARENTHESIS;
 ID "numero"; MORE_THAN; LITERAL_INTEGER 10; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "O n\195\186mero %d \195\169 maior que 10\n"; COMA;
 ID "numero"; CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; ELSE;
 OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "O n\195\186mero %d \195\169 menor que 10\n"; COMA;
 ID "numero"; CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS;
 CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.21:** *Saída do analisador léxico para o programa Micro 09*

```
# lex "miniC//micro09.c";;
- : Lexical.tokens list =
[INTEGER; ID "main"; OPEN_PARENTHESIS; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; FLOAT; ID "preco"; COMA; ID "venda"; COMA;
 ID "novo_preco"; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Digite o pre\195\167o: "; CLOSE_PARENTHESIS; SEMICOLON;
 ID "scanf"; OPEN_PARENTHESIS; LITERAL_STRING "%f"; COMA; ADDRESS;
 ID "preco"; CLOSE_PARENTHESIS; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Digite a venda: "; CLOSE_PARENTHESIS; SEMICOLON; ID "scanf";
 OPEN_PARENTHESIS; LITERAL_STRING "%f"; COMA; ADDRESS; ID "venda";
 CLOSE_PARENTHESIS; SEMICOLON; IF; OPEN_PARENTHESIS; ID "venda"; LESS_THAN;
 LITERAL_INTEGER 500; OR; ID "preco"; LESS_THAN; LITERAL_INTEGER 30;
 CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "novo_preco"; ATTRIBUTION;
 ID "preco"; ADDITION; OPEN_PARENTHESIS; LITERAL_FLOAT 10.; DIVISION;
 LITERAL_INTEGER 100; MULTIPLICATION; ID "preco"; CLOSE_PARENTHESIS;
 SEMICOLON; CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; IF;
 OPEN_PARENTHESIS; OPEN_PARENTHESIS; ID "venda"; MORE_EQUAL_THAN;
 LITERAL_INTEGER 500; AND; ID "venda"; LESS_THAN; LITERAL_INTEGER 1200;
 CLOSE_PARENTHESIS; OR; OPEN_PARENTHESIS; ID "preco"; MORE_EQUAL_THAN;
 LITERAL_INTEGER 30; AND; ID "preco"; LESS_THAN; LITERAL_INTEGER 80;
 CLOSE_PARENTHESIS; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "novo_preco";
 ATTRIBUTION; ID "preco"; ADDITION; OPEN_PARENTHESIS; LITERAL_FLOAT 15.;
 DIVISION; LITERAL_INTEGER 100; MULTIPLICATION; ID "preco";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; ELSE;
 OPEN_CURLED_BRACKETS; IF; OPEN_PARENTHESIS; ID "venda"; MORE_EQUAL_THAN;
 LITERAL_INTEGER 1200; OR; ID "preco"; MORE_EQUAL_THAN; LITERAL_INTEGER 80;
 CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "novo_preco"; ATTRIBUTION;
 ID "preco"; SUBTRACTION; OPEN_PARENTHESIS; LITERAL_FLOAT 20.; DIVISION;
 LITERAL_INTEGER 100; MULTIPLICATION; ID "preco"; CLOSE_PARENTHESIS;
 SEMICOLON; CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS;
 CLOSE_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "O novo pre\195\167o \195\169 %f\n"; COMA; ID "novo_preco";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.22:** *Saída do analisador léxico para o programa Micro 10*

```
# lex "miniC//micro10.c";;
- : Lexical.tokens list =
[INTEGER; ID "fatorial"; OPEN_PARENTHESIS; INTEGER; ID "n";
 CLOSE_PARENTHESIS; SEMICOLON; INTEGER; ID "main"; OPEN_PARENTHESIS;
 CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; INTEGER; ID "numero"; COMA;
 ID "fat"; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Digite um n\195\186mero: "; CLOSE_PARENTHESIS; SEMICOLON;
 ID "scanf"; OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ADDRESS;
 ID "numero"; CLOSE_PARENTHESIS; SEMICOLON; ID "fat"; ATTRIBUTION;
 ID "fatorial"; OPEN_PARENTHESIS; ID "numero"; CLOSE_PARENTHESIS; SEMICOLON;
 ID "printf"; OPEN_PARENTHESIS; LITERAL_STRING "O fatorial de ";
 CLOSE_PARENTHESIS; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%d"; COMA; ID "numero"; CLOSE_PARENTHESIS; SEMICOLON;
 ID "printf"; OPEN_PARENTHESIS; LITERAL_STRING " \195\169 ";
 CLOSE_PARENTHESIS; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "%d\n"; COMA; ID "fat"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; INTEGER; ID "fatorial"; OPEN_PARENTHESIS; INTEGER;
 ID "n"; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; IF; OPEN_PARENTHESIS;
 ID "n"; LESS_EQUAL_THAN; LITERAL_INTEGER 0; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; RETURN; LITERAL_INTEGER 1; SEMICOLON;
 CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; RETURN; ID "n";
 MULTIPLICATION; ID "fatorial"; OPEN_PARENTHESIS; ID "n"; SUBTRACTION;
 LITERAL_INTEGER 1; CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS;
 CLOSE_CURLED_BRACKETS; EOF]
```

**Figura 4.23:** *Saída do analisador léxico para o programa Micro 11*

```
# lex "miniC//micro11.c";;
- : Lexical.tokens list =
[INTEGER; ID "verifica"; OPEN_PARENTHESIS; INTEGER; ID "n";
 CLOSE_PARENTHESIS; SEMICOLON; INTEGER; ID "main"; OPEN_PARENTHESIS;
 CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; INTEGER; ID "numero"; COMA;
 ID "x"; SEMICOLON; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "Digite um n\195\186mero: "; CLOSE_PARENTHESIS; SEMICOLON;
 ID "scanf"; OPEN_PARENTHESIS; LITERAL_STRING "%d"; COMA; ADDRESS;
 ID "numero"; CLOSE_PARENTHESIS; SEMICOLON; ID "x"; ATTRIBUTION;
 ID "verifica"; OPEN_PARENTHESIS; ID "numero"; CLOSE_PARENTHESIS; SEMICOLON;
 IF; OPEN_PARENTHESIS; ID "x"; EQUALS; LITERAL_INTEGER 1; CLOSE_PARENTHESIS;
 OPEN_CURLED_BRACKETS; ID "printf"; OPEN_PARENTHESIS;
 LITERAL_STRING "N\195\186mero positivo\n"; CLOSE_PARENTHESIS; SEMICOLON;
 CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; IF; OPEN_PARENTHESIS;
 ID "x"; EQUALS; LITERAL_INTEGER 0; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS;
 ID "printf"; OPEN_PARENTHESIS; LITERAL_STRING "Zero\n"; CLOSE_PARENTHESIS;
 SEMICOLON; CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; ID "printf";
 OPEN_PARENTHESIS; LITERAL_STRING "N\195\186mero negativo\n";
 CLOSE_PARENTHESIS; SEMICOLON; CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS;
 CLOSE_CURLED_BRACKETS; INTEGER; ID "verifica"; OPEN_PARENTHESIS; INTEGER;
 ID "n"; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; INTEGER; ID "res";
 SEMICOLON; IF; OPEN_PARENTHESIS; ID "n"; MORE_THAN; LITERAL_INTEGER 0;
 CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "res"; ATTRIBUTION;
 LITERAL_INTEGER 1; SEMICOLON; CLOSE_CURLED_BRACKETS; ELSE;
 OPEN_CURLED_BRACKETS; IF; OPEN_PARENTHESIS; ID "n"; LESS_THAN;
 LITERAL_INTEGER 0; CLOSE_PARENTHESIS; OPEN_CURLED_BRACKETS; ID "res";
 ATTRIBUTION; SUBTRACTION; LITERAL_INTEGER 1; SEMICOLON;
 CLOSE_CURLED_BRACKETS; ELSE; OPEN_CURLED_BRACKETS; ID "res"; ATTRIBUTION;
 LITERAL_INTEGER 0; SEMICOLON; CLOSE_CURLED_BRACKETS; CLOSE_CURLED_BRACKETS;
 RETURN; ID "res"; SEMICOLON; CLOSE_CURLED_BRACKETS; EOF]
```

# Capítulo 5

# Analisador Sintático

Nesta seção, será apresentada a construção do analisador sintático para a linguagem miniC, incluindo a definição da gramática, da árvore sintática abstrada e as alterações realizadas no código do analizador léxico para comportar a adição do analisador sintático. Também serão apresentados os resultados da execução do analisador sintático sobre os programas de teste utilizados neste relatório (nano e micro).

### 5.0.1 Código do analisador sintático

A seguir, é apresentado o código do gerador analisador sintático, contendo a nova lista dos tokes (removida do analisador léxico).

**Listagem 5.1: Analisador Sintático**

```
1 %{
2   open Sintatico_arvore
3
4 %}
5
6 %token <string> ID
7 %token <string> LITERAL_STRING
8 %token <int> LITERAL_INTEGER
9 %token <float> LITERAL_FLOAT
10 %token <char> LITERAL_CHAR
11 %token <bool> LITERAL_BOOL
12 %token INTEGER
13 %token FLOAT
14 %token CHAR
15 %token BOOL
16 %token MAIN
17 %token ATTRIBUTION
18 %token OPEN_PARENTHESIS
19 %token CLOSE_PARENTHESIS
20 %token OPEN_BRACKETS
21 %token CLOSE_BRACKETS
22 %token OPEN_CURLED_BRACKETS
23 %token CLOSE_CURLED_BRACKETS
24 %token ADDITION
25 %token SUBTRACTION
```

```
%token MULTIPLICATION
%token DIVISION
%token COMA
%token SEMICOLON
%token COLON
%token SINGLE_QUOTE
%token MORE_THAN
%token LESS_THAN
%token NOT
%token ADDRESS
%token EQUALS
%token MORE_EQUAL_THAN
%token LESS_EQUAL_THAN
%token DIFFERENT
%token INCREMENT
%token DECREMENT
%token OR
%token AND
%token MODULE
%token IF
%token ELSE
%token WHILE
%token FOR
%token DO
%token SWITCH
%token CASE
%token BREAK
%token DEFAULT
%token RETURN
%token <string> HEADER_FILE
%token EOF
%token SCANF PRINTF

%left OR
%left AND
%left MORE_THAN LESS_THAN MORE_EQUAL_THAN LESS_EQUAL_THAN EQUALS DIFFERENT
%left ADDITION SUBTRACTION
%left MULTIPLICATION DIVISION MODULE
%left NOT


%start <Sintatico_arvore.programa> programa

%%

programa: includes funcoes EOF{ $2 }

includes: /* nada */  { None }
        | HEADER_FILE { Some(Includes $1) };

funcoes: /* nada */ { [] }
       | funcao funcoes { $1 :: $2 }
       ;

funcao: INTEGER MAIN OPEN_PARENTHESIS argumentos CLOSE_PARENTHESIS
     OPEN_CURLED_BRACKETS comandos CLOSE_CURLED_BRACKETS { Funcao(TINTEGER,
     ExpVar "main", [], $7, ExpInt 1) }
     | tipo ID OPEN_PARENTHESIS argumentos CLOSE_PARENTHESIS
         OPEN_CURLED_BRACKETS comandos CLOSE_CURLED_BRACKETS { Funcao($1,
```

```
          ExpVar $2, $4, $7, ExpInt 1) }
82      ;
83
84  tipo: INTEGER { TINTEGER }
85      | FLOAT   { TFLOAT }
86      | CHAR    { TCHAR  }
87      | BOOL    { TBOOL  }
88      ;
89
90  argumentos: /* nada */ { [] }
91          | seq { $1 }
92
93  seq: argumento { [$1] }
94     | seq COMA argumento { $1 @ [$3] }
95
96  argumento: tipo ID { CmdDec(ExpVar $2, $1, None) }
97
98  comandos: /* nada */              { [] }
99    | comando SEMICOLON comandos { $1 :: $3 }
100   | comando comandos           { $1 :: $2 }
101   ;
102
103 comando: cmd_atrib   { $1 }
104         | cmd_dec     { $1 }
105         | cmd_printf  { $1 }
106         | cmd_scanf   { $1 }
107         | cmd_for     { $1 }
108         | cmd_do      { $1 }
109         | cmd_while   { $1 }
110         | cmd_if      { $1 }
111         | cmd_switch  { $1 }
112         | cmd_incr    { $1 }
113         | cmd_decr    { $1 }
114         | cmd_return  { $1 }
115         ;
116
117 cmd_atrib: id=ID ATTRIBUTION exp=expressao SEMICOLON { CmdAtrib (ExpVar id
        , exp) };
118
119 cmd_dec:
120         | tipo ID inicial SEMICOLON { CmdDec(ExpVar $2, $1, $3) }
121         | t=tipo id=ID OPEN_BRACKETS LITERAL_INTEGER CLOSE_BRACKETS
              SEMICOLON { CmdDec(ExpVar id, t, None) }
122         ;
123
124 inicial:
125         | { None }
126         | ATTRIBUTION expressao { Some($2) }
127         ;
128
129 cmd_printf: PRINTF OPEN_PARENTHESIS args CLOSE_PARENTHESIS SEMICOLON {
        CmdPrintf($3) };
130
131 cmd_scanf: SCANF OPEN_PARENTHESIS expressao COMA ADDRESS ID
        CLOSE_PARENTHESIS SEMICOLON { CmdScanf($3, ExpVar $6) };
132
133 cmd_while: WHILE OPEN_PARENTHESIS expressao CLOSE_PARENTHESIS
        OPEN_CURLED_BRACKETS comandos CLOSE_CURLED_BRACKETS { CmdWhile($3, $6)
        };
```

```
134
135 cmd_for: FOR OPEN_PARENTHESIS cmd_atrib expressao SEMICOLON comando
      CLOSE_PARENTHESIS OPEN_CURLED_BRACKETS comandos CLOSE_CURLED_BRACKETS {
       CmdFor($3, $4, $6, $9) };
136
137 cmd_do: DO OPEN_CURLED_BRACKETS comandos CLOSE_CURLED_BRACKETS WHILE
      OPEN_PARENTHESIS expressao CLOSE_PARENTHESIS SEMICOLON { CmdDo($3, $7)
      };
138
139 cmd_if: IF OPEN_PARENTHESIS expressao CLOSE_PARENTHESIS
      OPEN_CURLED_BRACKETS comandos CLOSE_CURLED_BRACKETS elsee { CmdIf($3,
      $6, $8) };
140
141 cmd_incr: ID INCREMENT { CmdIncr(ExpVar $1) };
142
143 cmd_decr: ID DECREMENT { CmdDecr(ExpVar $1) };
144
145 elsee: /* nada */ { None }
146      | ELSE OPEN_CURLED_BRACKETS comandos CLOSE_CURLED_BRACKETS { Some($3)
           }
147      | ELSE cmd_if { Some([$2]) }
148      ;
149
150 cmd_switch: SWITCH OPEN_PARENTHESIS expressao CLOSE_PARENTHESIS
      OPEN_CURLED_BRACKETS cases default CLOSE_CURLED_BRACKETS { CmdSwitch($3
      , $6, Some($7)) };
151
152 cases: /* nada */ { [] }
153      | case cases { $1 :: $2 }
154      ;
155
156 case: CASE expressao COLON comandos BREAK SEMICOLON { CASE($2, $4) }
157     | CASE SINGLE_QUOTE expressao SINGLE_QUOTE COLON comandos BREAK
          SEMICOLON { CASE($3, $6) }
158     ;
159
160 default: DEFAULT COLON comandos { DEFAULT($3) };
161
162 cmd_return:
163           | RETURN exp=expressao SEMICOLON { CmdReturn(exp) };
164
165 args: /* nada */ { [] }
166     | seqs { $1 }
167     ;
168
169 seqs: expressao { [$1] }
170     | seqs COMA expressao { $1 @ [$3] }
171     ;
172
173 expressao:
174          | id=ID              { ExpVar     id }
175          | i=LITERAL_INTEGER { ExpInt      i }
176          | f=LITERAL_FLOAT   { ExpFloat    f }
177          | c=LITERAL_CHAR    { ExpChar     c }
178          | s=LITERAL_STRING  { ExpString   s }
179          | b=LITERAL_BOOL    { ExpBool     b }
180          | NOT e=expressao   { ExpUn(Not, e) }
181          | le=expressao op=oper re=expressao { ExpBin (op, le, re) }
182          | OPEN_PARENTHESIS e=expressao CLOSE_PARENTHESIS { e }
```

```
183           | chama_func { $1 }
184           ;
185
186
187 %inline oper:
188               | OR { Or }
189               | AND { And }
190               | ADDITION { Add }
191               | SUBTRACTION { Sub }
192               | MULTIPLICATION { Mul }
193               | DIVISION { Div }
194               | MORE_THAN { More_Than }
195               | LESS_THAN { Less_Than }
196               | MORE_EQUAL_THAN { More_Equal_Than }
197               | LESS_EQUAL_THAN { Less_Equal_Than }
198               | EQUALS { Eq }
199               | DIFFERENT { Dif }
200               | MODULE { Mod }
201               ;
202
203 chama_func: ID OPEN_PARENTHESIS args CLOSE_PARENTHESIS { ChamaFunc(ExpVar
      $1, $3) };
```

### 5.0.2 Árvore Sintática Abstrata

A seguir, é apresentada a definição da árvore sintática abstrada utiizada pelo analisador sintático.

**Listagem 5.2: Árvore Sintática**

```
1 type programa = funcoes
2
3 and funcoes = funcao list
4 and funcao = Funcao of tipo * expressao * comando list * comandos *
      expressao
5
6 and includes = includee option
7 and includee = Includes of string
8
9 and comandos = comando list
10 and comando = CmdAtrib  of expressao * expressao
11              | CmdDec    of expressao * tipo * expressao option
12              | CmdPrintf of expressao list
13              | CmdScanf  of expressao * expressao
14              | CmdWhile  of expressao * comandos
15              | CmdFor    of comando * expressao * comando * comandos
16              | CmdDo     of comandos * expressao
17              | CmdIf     of expressao * comandos * comandos option
18              | CmdSwitch of expressao * cases * default option
19              | CmdGetC   of string
20              | CmdIncr   of expressao
21              | CmdDecr   of expressao
22              | CmdReturn of expressao
23
24 and expressao = ExpInt     of int
25                | ExpVar     of string
```

```
26                | ExpFloat   of float
27                | ExpChar    of char
28                | ExpString  of string
29                | ExpBool    of bool
30                | ExpBin     of operador * expressao * expressao
31                | ExpUn      of operador * expressao
32                | ChamaFunc  of expressao * expressao list
33
34 and expr = { valor: expressao;
35               mutable tipoexp: tipo option
36             }
37
38 and operador = Add | Sub | Mul | Div | Mod
39               | More_Than | Less_Than | More_Equal_Than | Less_Equal_Than
40               | Eq | Dif | Or | And | Not
41
42 and tipo = TINTEGER | TFLOAT | TCHAR | TBOOL | TSTRING
43
44 and cases = case list
45 and case = CASE of expressao * comandos
46 and default = DEFAULT of comandos
47
48 and variavel = VarSimples of string
49
50 type tvalor = VInt of int | VFloat of float | VBool of bool | VString of
       string
51             | VChar of char
52
53 type info = { tipos: tipo;
54               inicializada: bool;
55               valor: tvalor option;
56               mutable endereco: int option
57             }
```

### 5.0.3 Código auxiliar

A seguir, são apresentados códigos auxiliares para teste do analisador sintático:

**Listagem 5.3: Teste Analisador Sintático**

```
1 open Printf
2 open Lexing
3
4 open Sintatico_arvore
5 open Sintatico_erros (* nome do módulo contendo as mensagens de erro *)
6
7 exception Erro_Sintatico of string
8
9 module S = MenhirLib.General (* Streams *)
10 module I = Sintatico.MenhirInterpreter
11
12 let posicao lexbuf =
13     let pos = lexbuf.lex_curr_p in
14     let lin = pos.pos_lnum
15     and col = pos.pos_cnum - pos.pos_bol - 1 in
16     sprintf "linha %d, coluna %d" lin col
17
```

```
18 (* [pilha checkpoint] extrai a pilha do autômato LR(1) contida em
      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 : Sintatico_arvore.programa I.checkpoint) =
35   let estado_atual = estado checkpoint in
36   let msg = message estado_atual in
37   raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
38                                         (Lexing.lexeme_start lexbuf) msg))
39
40 let loop lexbuf resultado =
41 let fornecedor = I.lexer_lexbuf_to_supplier Lexico.token lexbuf in
42   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
43
44
45
46 let parse_com_erro lexbuf =
47   try
48     Some (loop lexbuf (Sintatico.Incremental.programa lexbuf.lex_curr_p))
49   with
50   | Lexico.Erro msg ->
51      printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
52      None
53   | Erro_Sintatico msg ->
54      printf "Erro sintático na %s %s\n" (posicao lexbuf) msg;
55      None
56
57 let parse s =
58   let lexbuf = Lexing.from_string s in
59   let ast = parse_com_erro lexbuf in
60   ast
61
62 let parse_arq nome =
63   let ic = open_in nome in
64   let lexbuf = Lexing.from_channel ic in
65   let ast = parse_com_erro lexbuf in
66   let _ = close_in ic in
67   ast
68
69
70 (* Para compilar:
71      menhir -v --list-errors sintatico.mly > sintatico.msg
72      menhir -v --list-errors sintatico.mly --compile-errors sintatico.msg
            > erroSint.ml
73      ocamlbuild -use-menhir sintaticoTest.byte
74  *)
```

**Listagem 5.4: Carregador de módulos**

```
#use "topfind";;
#require "menhirLib";;
#directory "_build";;
#load "lexico.cmo";;
#load "sintatico.cmo";;
#load "sintatico_erros.cmo";;
#load "sintatico_arvore.cmo";;
#load "sintatico_teste.cmo";;
open Sintatico_arvore
open Sintatico_teste
```

### 5.0.4 Execução

Gerar arquivo com as mensangens de erro do analisador sintático:
**menhir -v –list-errors sintatico.mly > sintatico.msg**

Compilar o arquivo de mensagens de erro:
**menhir sintatico.mly –compile-errors sintatico.msg > sintatico_erros.ml**

Compilar o analisador sintático:
**ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir –table-package menhirLib sintatico_teste.byte**

Para iniciar o intérprete do ocaml, use o comando:
**rlwrap ocaml**

Para executar o analisador sintático sobre um arquivo de entrada, use o comando:
**parse_arq "nome_do_arquivo";;**

### 5.0.5 Resultados

A seguir, são apresentados os resultados da execução do analisador léxico nos programas de teste utilizados durante o desenvolvimento.

**Figura 5.1:** *Saída do analisador sintático para o programa Nano 01*

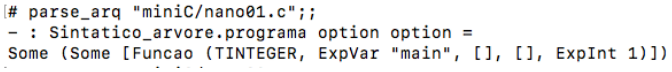

**Figura 5.2:** *Saída do analisador sintático para o programa Nano 02*

```
# parse_arq "miniC/nano02.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None)], ExpInt 1)])
```

**Figura 5.3:** *Saída do analisador sintático para o programa Nano 03*

```
# parse_arq "miniC/nano03.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None); CmdAtrib (ExpVar "n", ExpInt 1)],
     ExpInt 1)])
```

**Figura 5.4:** *Saída do analisador sintático para o programa Nano 04*

```
# parse_arq "miniC/nano04.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None);
      CmdAtrib (ExpVar "n", ExpBin (Add, ExpInt 1, ExpInt 2))],
     ExpInt 1)])
```

**Figura 5.5:** *Saída do analisador sintático para o programa Nano 05*

```
# parse_arq "miniC/nano05.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None); CmdAtrib (ExpVar "n", ExpInt 2);
      CmdPrintf [ExpString "%d"; ExpVar "n"]],
     ExpInt 1)])
```

**Figura 5.6:** *Saída do analisador sintático para o programa Nano 06*

```
# parse_arq "miniC/nano06.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None);
      CmdAtrib (ExpVar "n", ExpBin (Sub, ExpInt 1, ExpInt 2));
      CmdPrintf [ExpString "%d"; ExpVar "n"]],
     ExpInt 1)])
```

**Figura 5.7:** *Saída do analisador sintático para o programa Nano 07*

```
# parse_arq "miniC/nano07.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None); CmdAtrib (ExpVar "n", ExpInt 1);
      CmdIf (ExpBin (Eq, ExpVar "n", ExpInt 1),
       [CmdPrintf [ExpString "%d"; ExpVar "n"]], None)],
     ExpInt 1)])
```

**Figura 5.8:** *Saída do analisador sintático para o programa Nano 08*

```
# parse_arq "miniC/nano08.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None); CmdAtrib (ExpVar "n", ExpInt 1);
      CmdIf (ExpBin (Eq, ExpVar "n", ExpInt 1),
       [CmdPrintf [ExpString "%d"; ExpVar "n"]],
       Some [CmdPrintf [ExpString "0"]])],
     ExpInt 1)])
```

**Figura 5.9:** *Saída do analisador sintático para o programa Nano 09*

```
# parse_arq "miniC/nano09.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None);
      CmdAtrib (ExpVar "n",
       ExpBin (Add, ExpInt 1, ExpBin (Div, ExpInt 1, ExpInt 2)));
      CmdIf (ExpBin (Eq, ExpVar "n", ExpInt 1),
       [CmdPrintf [ExpString "%d"; ExpVar "n"]],
       Some [CmdPrintf [ExpString "0"]])],
     ExpInt 1)])
```

**Figura 5.10:** *Saída do analisador sintático para o programa Nano 10*

```
# parse_arq "miniC/nano10.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None);
      CmdDec (ExpVar "m", TINTEGER, None); CmdAtrib (ExpVar "n", ExpInt 1);
      CmdAtrib (ExpVar "m", ExpInt 2);
      CmdIf (ExpBin (Eq, ExpVar "n", ExpVar "m"),
       [CmdPrintf [ExpString "%d"; ExpVar "n"]],
       Some [CmdPrintf [ExpString "0"]])],
     ExpInt 1)])
```

**Figura 5.11:** *Saída do analisador sintático para o programa Nano 11*

```
# parse_arq "miniC/nano11.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None);
      CmdDec (ExpVar "m", TINTEGER, None);
      CmdDec (ExpVar "x", TINTEGER, None); CmdAtrib (ExpVar "n", ExpInt 1);
      CmdAtrib (ExpVar "m", ExpInt 2); CmdAtrib (ExpVar "x", ExpInt 5);
      CmdWhile (ExpBin (More_Than, ExpVar "x", ExpVar "n"),
       [CmdAtrib (ExpVar "n", ExpBin (Add, ExpVar "n", ExpVar "m"));
        CmdPrintf [ExpString "%d"; ExpVar "n"]])],
     ExpInt 1)])
```

**Figura 5.12:** *Saída do analisador sintático para o programa Nano 12*

```
# parse_arq "miniC/nano12.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "n", TINTEGER, None);
      CmdDec (ExpVar "m", TINTEGER, None);
      CmdDec (ExpVar "x", TINTEGER, None); CmdAtrib (ExpVar "n", ExpInt 1);
      CmdAtrib (ExpVar "m", ExpInt 2); CmdAtrib (ExpVar "x", ExpInt 5);
      CmdWhile (ExpBin (More_Than, ExpVar "x", ExpVar "n"),
       [CmdIf (ExpBin (Eq, ExpVar "n", ExpVar "m"),
         [CmdPrintf [ExpString "%d"; ExpVar "n"]],
         Some [CmdPrintf [ExpString "0"]]);
        CmdAtrib (ExpVar "x", ExpBin (Sub, ExpVar "x", ExpInt 1))])],
     ExpInt 1)])
```

**Figura 5.13:** *Saída do analisador sintático para o programa Micro 01*

```
# parse_arq "miniC/micro01.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "cel", TFLOAT, None);
      CmdDec (ExpVar "far", TFLOAT, None);
      CmdPrintf [ExpString " Tabela de conversão: Celsius -> Fahrenheit\n"];
      CmdPrintf [ExpString "Digite a temperatura em Celsius: "];
      CmdScanf (ExpString "%f", ExpVar "cel");
      CmdAtrib (ExpVar "far",
       ExpBin (Div,
        ExpBin (Add, ExpBin (Mul, ExpInt 9, ExpVar "cel"), ExpInt 160),
        ExpInt 5));
      CmdPrintf [ExpString "A nova temperatura é: %f F\n"; ExpVar "far"]],
     ExpInt 1)])
```

**Figura 5.14:** *Saída do analisador sintático para o programa Micro 02*

```
# parse_arq "miniC/micro02.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "num1", TINTEGER, None);
      CmdDec (ExpVar "num2", TINTEGER, None);
      CmdPrintf [ExpString "Digite o primeiro número: "];
      CmdScanf (ExpString "%d", ExpVar "num1");
      CmdPrintf [ExpString "Digite o segundo número: "];
      CmdScanf (ExpString "%d", ExpVar "num2");
      CmdIf (ExpBin (More_Than, ExpVar "num1", ExpVar "num2"),
       [CmdPrintf
         [ExpString "O primeiro número %d é maior que o segundo %d";
          ExpVar "num1"; ExpVar "num2"]],
       Some
        [CmdPrintf
          [ExpString "O segundo número %d é maior que o primeiro %d";
           ExpVar "num2"; ExpVar "num1"]])],
     ExpInt 1)])
```

**Figura 5.15:** *Saída do analisador sintático para o programa Micro 03*

```
# parse_arq "miniC/micro03.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "numero", TINTEGER, None);
      CmdPrintf [ExpString "Digite um número: "];
      CmdScanf (ExpString "%d", ExpVar "numero");
      CmdIf (ExpBin (More_Equal_Than, ExpVar "numero", ExpInt 100),
       [CmdIf (ExpBin (Less_Equal_Than, ExpVar "numero", ExpInt 200),
         [CmdPrintf
           [ExpString "O número está no intervalo entre 100 e 200\n"]],
         Some
          [CmdPrintf
            [ExpString "O número não está no intervalo entre 100 e 200\n"]])],
       Some
        [CmdPrintf
          [ExpString "O número não está no intervalo entre 100 e 200\n"]])],
     ExpInt 1)])
```

**Figura 5.16:** *Saída do analisador sintático para o programa Micro 04*

```
# parse_arq "miniC/micro04.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "x", TINTEGER, None);
      CmdDec (ExpVar "num", TINTEGER, None);
      CmdDec (ExpVar "intervalo", TINTEGER, None);
      CmdAtrib (ExpVar "intervalo", ExpInt 0);
      CmdFor (CmdAtrib (ExpVar "x", ExpInt 1),
       ExpBin (Less_Equal_Than, ExpVar "x", ExpInt 5), CmdIncr (ExpVar "x"),
       [CmdPrintf [ExpString "Digite um número: "];
        CmdScanf (ExpString "%d", ExpVar "num");
        CmdIf (ExpBin (More_Equal_Than, ExpVar "num", ExpInt 10),
         [CmdIf (ExpBin (Less_Equal_Than, ExpVar "num", ExpInt 150),
           [CmdAtrib (ExpVar "intervalo",
             ExpBin (Add, ExpVar "intervalo", ExpInt 1))],
           None)],
         None)]);
      CmdPrintf
       [ExpString
         "Ao total, foram digitados %d números no intervalo entre 10 e 150\n";
        ExpVar "intervalo"]],
     ExpInt 1)])
```

**Figura 5.17:** *Saída do analisador sintático para o programa Micro 05*

```
# parse_arq "miniC/micro05.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "nome", TCHAR, None);
      CmdDec (ExpVar "sexo", TCHAR, None);
      CmdDec (ExpVar "newLine", TCHAR, None);
      CmdDec (ExpVar "x", TINTEGER, None);
      CmdDec (ExpVar "h", TINTEGER, None);
      CmdDec (ExpVar "m", TINTEGER, None); CmdAtrib (ExpVar "h", ExpInt 0);
      CmdAtrib (ExpVar "m", ExpInt 0);
      CmdFor (CmdAtrib (ExpVar "x", ExpInt 1),
       ExpBin (Less_Equal_Than, ExpVar "x", ExpInt 5), CmdIncr (ExpVar "x"),
       [CmdPrintf [ExpString "Digite o nome: "];
        CmdScanf (ExpString "%s", ExpVar "nome");
        CmdScanf (ExpString "%c", ExpVar "newLine");
        CmdPrintf [ExpString "H - Homem ou M - Mulher: "];
        CmdScanf (ExpString "%c", ExpVar "sexo");
        CmdScanf (ExpString "%c", ExpVar "newLine");
        CmdSwitch (ExpVar "sexo",
         [CASE (ExpChar 'H',
           [CmdAtrib (ExpVar "h", ExpBin (Add, ExpVar "h", ExpInt 1))]);
          CASE (ExpChar 'M',
           [CmdAtrib (ExpVar "m", ExpBin (Add, ExpVar "m", ExpInt 1))])],
         Some (DEFAULT [CmdPrintf [ExpString "Sexo só pode ser H ou M!\n"]]))]);
      CmdPrintf [ExpString "Foram inseridos %d Homens\n"; ExpVar "h"];
      CmdPrintf [ExpString "Foram inseridos %d Mulheres\n"; ExpVar "m"]],
    ExpInt 1)])
```

**Figura 5.18:** *Saída do analisador sintático para o programa Micro 06*

```
# parse_arq "miniC/micro06.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "numero", TINTEGER, None);
      CmdPrintf [ExpString "Digite um número de 1 a 5: "];
      CmdScanf (ExpString "%d", ExpVar "numero");
      CmdSwitch (ExpVar "numero",
       [CASE (ExpInt 1, [CmdPrintf [ExpString "Um\n"]]);
        CASE (ExpInt 2, [CmdPrintf [ExpString "Dois\n"]]);
        CASE (ExpInt 3, [CmdPrintf [ExpString "Três\n"]]);
        CASE (ExpInt 4, [CmdPrintf [ExpString "Quatro\n"]]);
        CASE (ExpInt 5, [CmdPrintf [ExpString "Cinco\n"]])],
       Some (DEFAULT [CmdPrintf [ExpString "Número Inválido!!!\n"]]))],
     ExpInt 1)])
```

**Figura 5.19:** *Saída do analisador sintático para o programa Micro 07*

```
# parse_arq "miniC/micro07.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "programa", TINTEGER, None);
      CmdDec (ExpVar "numero", TINTEGER, None);
      CmdDec (ExpVar "opc", TCHAR, None);
      CmdAtrib (ExpVar "programa", ExpInt 1);
      CmdWhile (ExpBin (Eq, ExpVar "programa", ExpInt 1),
       [CmdPrintf [ExpString "Digite um número: "];
        CmdScanf (ExpString "%d", ExpVar "numero");
        CmdIf (ExpBin (More_Than, ExpVar "numero", ExpInt 0),
         [CmdPrintf [ExpString "Positivo\n"]],
         Some
          [CmdIf (ExpBin (Eq, ExpVar "numero", ExpInt 0),
            [CmdPrintf [ExpString "O número é igual a 0\n"]],
            Some [CmdPrintf [ExpString "Negativo\n"]])]);
        CmdScanf (ExpString "%c", ExpVar "opc");
        CmdPrintf [ExpString "Deseja finalizar? (S/N) "];
        CmdScanf (ExpString "%c", ExpVar "opc");
        CmdIf (ExpBin (Eq, ExpVar "opc", ExpChar 'S'),
         [CmdAtrib (ExpVar "programa", ExpInt 0)], None)])],
     ExpInt 1)])
```

**Figura 5.20:** *Saída do analisador sintático para o programa Micro 08*

```
# parse_arq "miniC/micro08.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "numero", TINTEGER, None);
      CmdAtrib (ExpVar "numero", ExpInt 1);
      CmdWhile (ExpBin (Dif, ExpVar "numero", ExpInt 0),
       [CmdPrintf [ExpString "Digite um número: "];
        CmdScanf (ExpString "%d", ExpVar "numero");
        CmdIf (ExpBin (More_Than, ExpVar "numero", ExpInt 10),
         [CmdPrintf
           [ExpString "O número %d é maior que 10\n"; ExpVar "numero"]],
         Some
          [CmdPrintf
            [ExpString "O número %d é menor que 10\n"; ExpVar "numero"]])])],
     ExpInt 1)])
```

**Figura 5.21:** *Saída do analisador sintático para o programa Micro 09*

```
# parse_arq "miniC/micro09.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "preco", TFLOAT, None);
      CmdDec (ExpVar "venda", TFLOAT, None);
      CmdDec (ExpVar "novo_preco", TFLOAT, None);
      CmdPrintf [ExpString "Digite o preço: "];
      CmdScanf (ExpString "%f", ExpVar "preco");
      CmdPrintf [ExpString "Digite a venda: "];
      CmdScanf (ExpString "%f", ExpVar "venda");
      CmdIf
       (ExpBin (Or, ExpBin (Less_Than, ExpVar "venda", ExpInt 500),
         ExpBin (Less_Than, ExpVar "preco", ExpInt 30)),
       [CmdAtrib (ExpVar "novo_preco",
         ExpBin (Add, ExpVar "preco",
          ExpBin (Mul, ExpBin (Div, ExpFloat 10., ExpInt 100),
           ExpVar "preco")))],
       Some
        [CmdIf
          (ExpBin (Or,
            ExpBin (And,
             ExpBin (More_Equal_Than, ExpVar "venda", ExpInt 500),
             ExpBin (Less_Than, ExpVar "venda", ExpInt 1200)),
            ExpBin (And, ExpBin (More_Equal_Than, ExpVar "preco", ExpInt 30),
             ExpBin (Less_Than, ExpVar "preco", ExpInt 80))),
          [CmdAtrib (ExpVar "novo_preco",
            ExpBin (Add, ExpVar "preco",
             ExpBin (Mul, ExpBin (Div, ExpFloat 15., ExpInt 100),
              ExpVar "preco")))],
          Some
           [CmdIf
             (ExpBin (Or,
               ExpBin (More_Equal_Than, ExpVar "venda", ExpInt 1200),
               ExpBin (More_Equal_Than, ExpVar "preco", ExpInt 80)),
             [CmdAtrib (ExpVar "novo_preco",
               ExpBin (Sub, ExpVar "preco",
                ExpBin (Mul, ExpBin (Div, ExpFloat 20., ExpInt 100),
                 ExpVar "preco")))],
             None)])]);
      CmdPrintf [ExpString "O novo preço é %f\n"; ExpVar "novo_preco"]],
     ExpInt 1)])
```

**Figura 5.22:** *Saída do analisador sintático para o programa Micro 10*

```
# parse_arq "miniC/micro10.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "fatorial",
     [CmdDec (ExpVar "n", TINTEGER, None)],
     [CmdIf (ExpBin (Less_Equal_Than, ExpVar "n", ExpInt 0),
       [CmdReturn (ExpInt 1)],
       Some
        [CmdReturn
          (ExpBin (Mul, ExpVar "n",
            ChamaFunc (ExpVar "fatorial",
             [ExpBin (Sub, ExpVar "n", ExpInt 1)])))])],
     ExpInt 1);
    Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "numero", TINTEGER, None);
      CmdDec (ExpVar "fat", TINTEGER, None);
      CmdPrintf [ExpString "Digite um número: "];
      CmdScanf (ExpString "%d", ExpVar "numero");
      CmdAtrib (ExpVar "fat",
       ChamaFunc (ExpVar "fatoria", [ExpVar "numero"]));
      CmdPrintf [ExpString "O fatorial de "];
      CmdPrintf [ExpString "%d"; ExpVar "numero"];
      CmdPrintf [ExpString " é "];
      CmdPrintf [ExpString "%d\n"; ExpVar "fat"]],
     ExpInt 1)])
```

**Figura 5.23:** *Saída do analisador sintático para o programa Micro 11*

```
# parse_arq "miniC/micro11.c";;
- : Sintatico_arvore.programa option option =
Some
 (Some
   [Funcao (TINTEGER, ExpVar "verifica",
     [CmdDec (ExpVar "n", TINTEGER, None)],
     [CmdDec (ExpVar "res", TINTEGER, None);
      CmdIf (ExpBin (More_Than, ExpVar "n", ExpInt 0),
       [CmdAtrib (ExpVar "res", ExpInt 1)],
       Some
        [CmdIf (ExpBin (Less_Than, ExpVar "n", ExpInt 0),
          [CmdAtrib (ExpVar "res", ExpInt (-1))],
          Some [CmdAtrib (ExpVar "res", ExpInt 0)])]);
      CmdReturn (ExpVar "res")],
     ExpInt 1);
    Funcao (TINTEGER, ExpVar "main", [],
     [CmdDec (ExpVar "numero", TINTEGER, None);
      CmdDec (ExpVar "x", TINTEGER, None);
      CmdPrintf [ExpString "Digite um número: "];
      CmdScanf (ExpString "%d", ExpVar "numero");
      CmdAtrib (ExpVar "x", ChamaFunc (ExpVar "verifica", [ExpVar "numero"]));
      CmdIf (ExpBin (Eq, ExpVar "x", ExpInt 1),
       [CmdPrintf [ExpString "Número positivo\n"]],
       Some
        [CmdIf (ExpBin (Eq, ExpVar "x", ExpInt 0),
          [CmdPrintf [ExpString "Zero\n"]],
          Some [CmdPrintf [ExpString "Número negativo\n"]])])],
   _  ExpInt 1)])
```