

Relatório da Disciplina de Construção de  
Compiladores  
MiniJava para LLVM

Flavia Otoubo Cunha,  
Sidney Ferreira de Lima

24 de fevereiro de 2015

# Sumário

<b>1</b>	<b>Introdução</b>	<b>4</b>
1.1	Compilador . . . . .	4
1.1.1	Estrutura de um Compilador . . . . .	5
1.2	Mini Java . . . . .	6
1.3	LLVM . . . . .	6
<b>2</b>	<b>Ambiente de Trabalho</b>	<b>7</b>
2.1	Instalando o Linux Mint . . . . .	7
2.2	Recomendações . . . . .	7
2.3	Instalação . . . . .	7
2.4	Instalação do LLVM . . . . .	14
2.4.1	Instalação . . . . .	14
2.5	Instalação do OCAML . . . . .	14
<b>3</b>	<b>LLVM</b>	<b>15</b>
3.1	Aprendendo um pouco de LLVM . . . . .	15
3.2	Tipos do LLVM . . . . .	16
<b>4</b>	<b>Ocaml</b>	<b>18</b>
4.1	Aprendendo um pouco de Ocaml . . . . .	18
<b>5</b>	<b>Compilador</b>	<b>21</b>
5.1	Analisador Léxico . . . . .	21
5.2	Analisador Sintático . . . . .	22
5.3	Analisador Semântico . . . . .	22
<b>6</b>	<b>Árvore Sintática Abstrata</b>	<b>23</b>
6.1	Derivação da Árvore . . . . .	23
6.2	Gramáticas Ambíguas . . . . .	24
6.3	Derivações e Precedência . . . . .	24
<b>7</b>	<b>Compilador - MiniJava para LLVM</b>	<b>26</b>
7.1	Makefile . . . . .	26
7.2	Código - Carregador . . . . .	28

7.3	Analizador Léxico . . . . .	30
7.3.1	Descrição - Analizador Léxico . . . . .	30
7.3.2	Código - Analizador Léxico . . . . .	31
7.4	Analizador Sintático . . . . .	34
7.4.1	Descrição - Analizador Sintático . . . . .	34
7.4.2	Código - Analizador Sintático . . . . .	38
7.5	Código - Árvore Sintática Abstrata . . . . .	45
7.6	Analizador Semântico . . . . .	48
7.6.1	Descrição . . . . .	48
7.6.2	Código - Analizador Semântico . . . . .	54
7.7	Código - Árvore Intermediária . . . . .	96
7.8	Código - Gerador de Código . . . . .	97
<b>8</b>	<b>Exercícios</b>	<b>109</b>
8.1	Tarefa 1 . . . . .	109
8.1.1	Listagem 1 - Módulo Mínimo . . . . .	109
8.1.2	Listagem 2 - Declaração de uma variável . . . . .	110
8.1.3	Listagem 3 - Atribuição de um inteiro a uma variável . . . . .	111
8.1.4	Listagem 4 - Atribuição de uma soma de inteiro a uma variável . . . . .	112
8.1.5	Listagem 5 - Inclusão do comando de impressão . . . . .	113
8.1.6	Listagem 6 - Atribuição de uma subtração de inteiros a um variável . . . . .	115
8.1.7	Listagem 7 - Inclusão do comando condicional . . . . .	116
8.1.8	Listagem 8 - Inclusão do comando condicional com parte senão . . . . .	118
8.1.9	Listagem 9 - Atribuição de duas operações aritméticas sobre inteiros a uma variável . . . . .	119
8.1.10	Listagem 10 - Atribuição de duas variáveis inteiras . . . . .	121
8.1.11	Listagem 11 - Introdução do comando de repetição enquanto . . . . .	123
8.1.12	Listagem 12 - Comando condicional aninhado em um comando de repetição . . . . .	125
8.1.13	Listagem 13 - Converte graus Celsius para Fahrenheit . . . . .	127
8.1.14	Listagem 14 - Ler dois inteiros e decide qual é maior . . . . .	129
8.1.15	Listagem 15 - Lê um número e verifica se ele está entre 100 e 200 . . . . .	132
8.1.16	Listagem 16 - Lê um número e informa quais estão entre 10 e 150 . . . . .	134
8.1.17	Listagem 17 - Lê strings e caracteres . . . . .	137
8.1.18	Listagem 18 - Escreve um número lido por extenso . . . . .	140
8.1.19	Listagem 19 - Decide se os números são positivos, zeros ou negativos . . . . .	143
8.1.20	Listagem 20 - DEcide se um número é maior ou menor que 10 . . . . .	145
8.1.21	Listagem 21 - Cálculos de preços . . . . .	148
8.1.22	Listagem 22 - Calcula o fatorial de um número . . . . .	151
8.1.23	Listagem 23 - Decide se um número é positivo, zero ou negativo com auxílio de uma função . . . . .	153
8.1.24	Saídas . . . . .	157

# Lista de Figuras

1.1	Estrutura de um compilador. . . . .	4
1.2	Estrutura de um compilador. . . . .	5
2.1	Selecionando o idioma. . . . .	8
2.2	Verificação do espaço. . . . .	8
2.3	Opções avançadas. . . . .	9
2.4	Escolha da partição. . . . .	9
2.5	Formatação da partição. . . . .	10
2.6	Tipo da instalação. . . . .	10
2.7	Sua localização. . . . .	11
2.8	Escolha do layout do teclado. . . . .	11
2.9	Definição do usuário e sua senha. . . . .	12
2.10	Tela de espera. . . . .	12
2.11	Tela para finalização da instalação. . . . .	13
2.12	Tela de saudações. . . . .	13
6.1	Derivação a esquerda. . . . .	23
6.2	Derivação a direita. . . . .	24
6.3	Exemplo de derivação e precedência. . . . .	25

# Capítulo 1

## Introdução

### 1.1 Compilador

Um compilador é um programa de sistema que traduz um programa descrito em uma linguagem de alto nível para um programa equivalente em código de máquina para um processador. Em geral, um compilador não produz diretamente o código de máquina mais sim um programa em linguagem simbólica (assembly) semanticamente equivalente ao programa em linguagem de alto nível. O programa em linguagem simbólica é então traduzido para o programa em linguagem de máquina através de montadores.

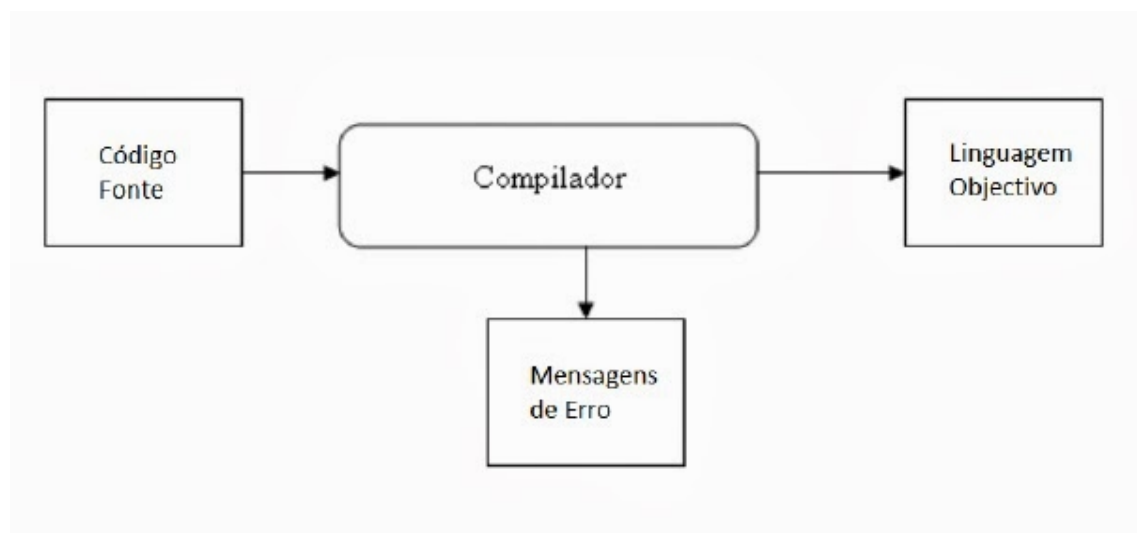


Figura 1.1: Estrutura de um compilador.

### 1.1.1 Estrutura de um Compilador

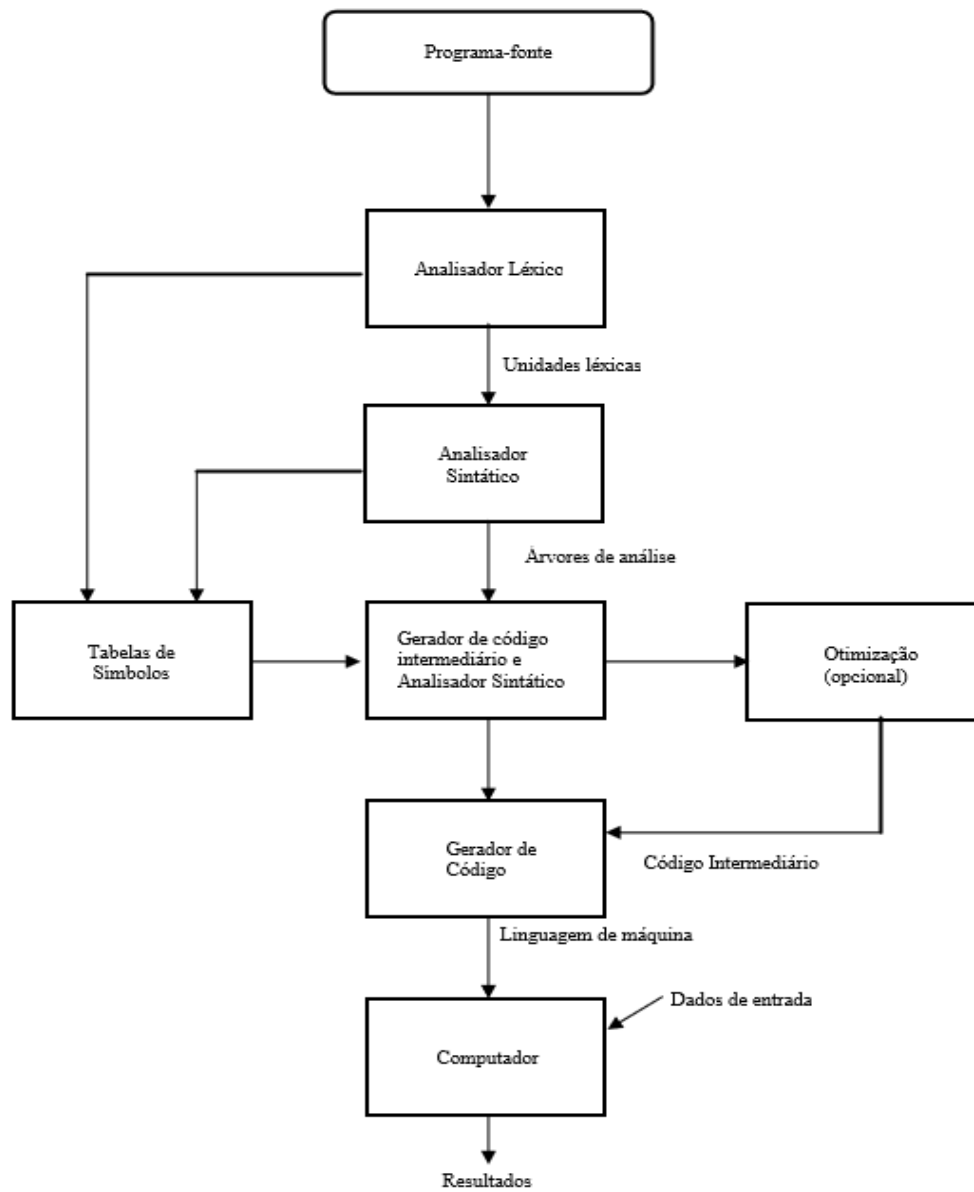


Figura 1.2: Estrutura de um compilador.

## 1.2 Mini Java

MiniJava é um subconjunto de Java. A sobrecarga não é permitido em MiniJava. No MiniJava declarações como `System.out.println(...)`; Só é possibilita imprimir inteiros. A expressão `e.length` MiniJava só se aplica a expressões do tipo `int[]`.

## 1.3 LLVM

Low Level Virtual Machine(LLVM) é uma infraestrutura de compilador escrita em C++, desenvolvida para otimizar em tempos de compilação, ligação e execução de programas escritos em linguagens de programação variadas. Implementada originalmente para c e C++, sua arquitetura permitiu a expansão para outras linguagens posteriormente, incluindo ObjectiveC, Fortran, Ada, Haskell, bytecode Java, Python, Ruby, ActionScript, GLSL, Julia, entre outras. O LLVM pode prover camadas intermediárias de um compilador, lendo a representação intermediária de um compilado e retornando outra representação otimizada, que pode ser então convertida e ligada em código de montagem para determinada plataforma. Ele também consegue gerar código binário otimizado em tempo de execução.

Sua arquitetura é independente de linguagem, conjunto de instruções ou sistema de tipo. Cada instrução é definida numa forma padronizada, permitindo a análise de dependência da árvore de execução do código. Toda forma de conversão de tipo é feita por ele através de instruções `cast`. A infraestrutura fornece alguns tipos básicos, como ponteiros e estruturas.

# Capítulo 2

## Ambiente de Trabalho

### 2.1 Instalando o Linux Mint

Linux Mint é uma das distribuições GNU/Linux mais amigáveis e, por isso, uma das mais recomendadas para iniciantes.

### 2.2 Recomendações

Antes de partirmos para a instalação do Linux Mint, faça o particionamento prévio do seu HD.

Crie uma partição no formato ext4 de mínimo 20 GB para a partição raiz ( / ) e uma partição SWAP (2 - 4GB).

### 2.3 Instalação

1 - Baixe e copie num pendrive o Linux Mint. Vá para o website oficial do Linux Mint em <http://www.linuxmint.com/download.php>, e baixe o .iso para a Edição Principal (Main Edition) mais recente. (Use tanto o “download direto” ou torrente.)

2 - Inicie a partir do pendrive e tente isto. OBS.: isto não vai danificar ou mudar seu computador de maneira alguma. Insira o pendrive no seu computador e reinicie. Se nada de novo acontecer, reinicie novamente e selecione o pendrive como o primeiro dispositivo de inicialização. Isto pode ser feito através da BIOS (pressione DEL assim que o computador iniciar) ou por outra tecla (normalmente ESC) que permite você selecionar o pendrive como um dispositivo de inicialização.



Ao inicializar:

- Selecione seu idioma:

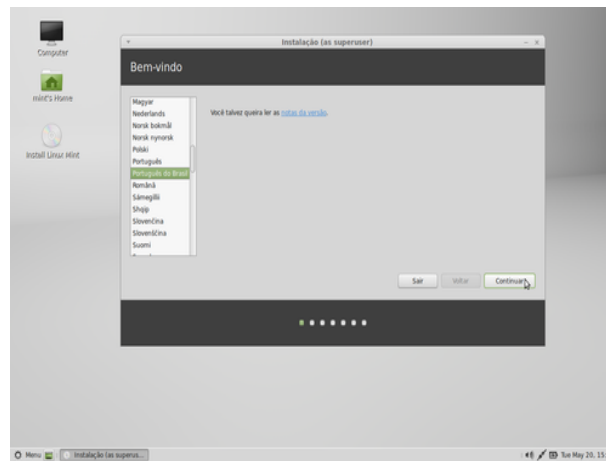


Figura 2.1: Selecionando o idioma.

- Certifique-se de ter espaço suficiente em disco, de estar conectada numa fonte de energia e na internet:

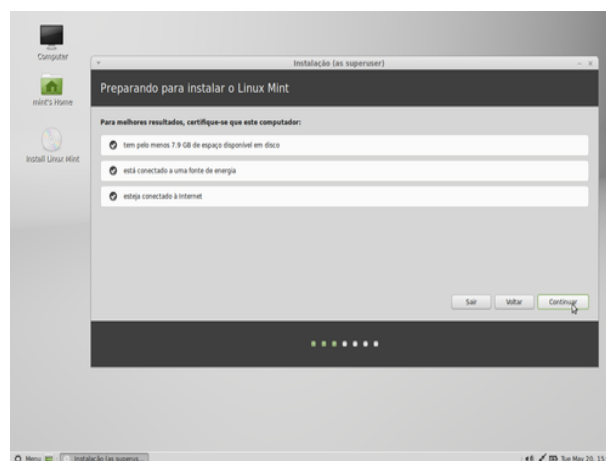


Figura 2.2: Verificação do espaço.

- Como fizemos o particionamento prévio do HD, no Tipo de Instalação, escolha Opção Avançada:

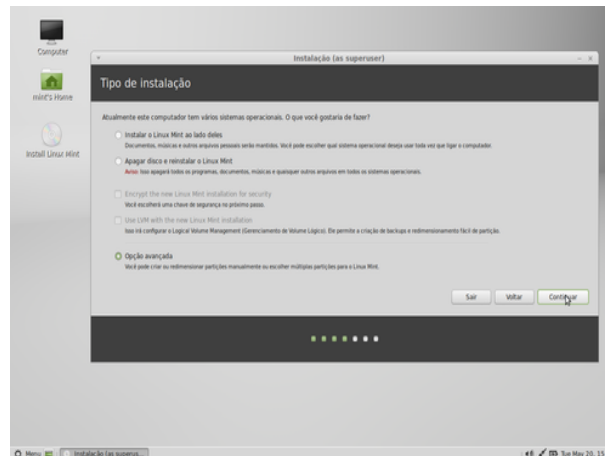


Figura 2.3: Opções avançadas.

- Selecione a partição em que deseja instalar o Linux Mint e clique em Change.

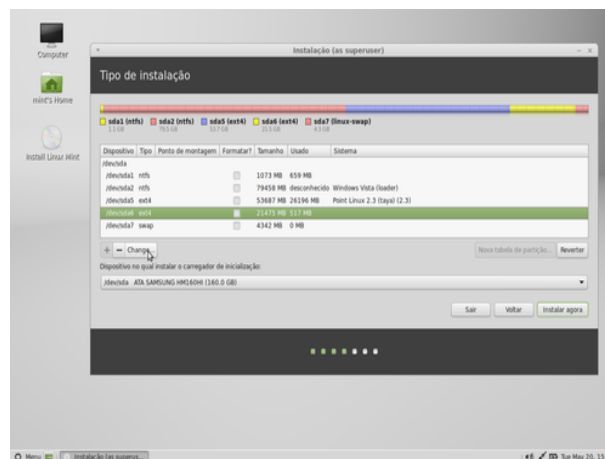


Figura 2.4: Escolha da partição.

- Em Editar Partição, selecione usar como ext4, formatar a partição e ponto de montagem em /:

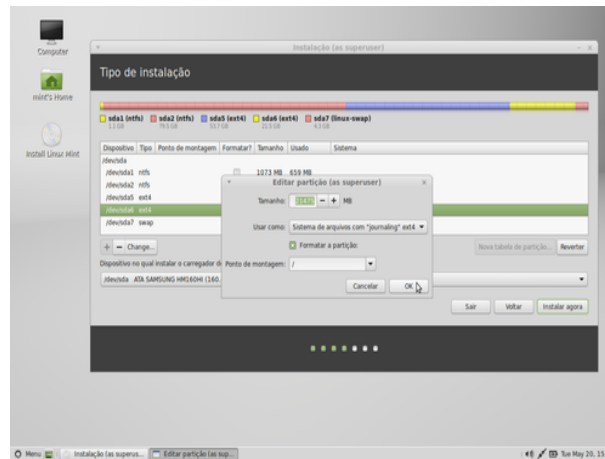


Figura 2.5: Formatação da partição.

- Certifique-se de que escolheu a partição correta e depois, clique em Instalar agora. Repita os mesmos passos para sua /home, se desejar:

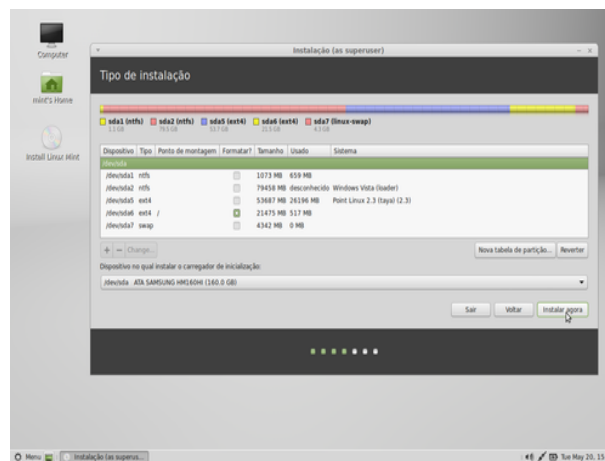


Figura 2.6: Tipo da instalação.

- Selecione sua localização:

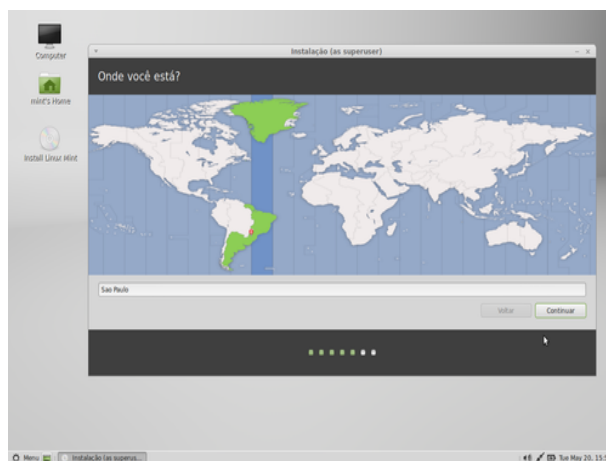


Figura 2.7: Sua localização.

- Selecione o layout de teclado:

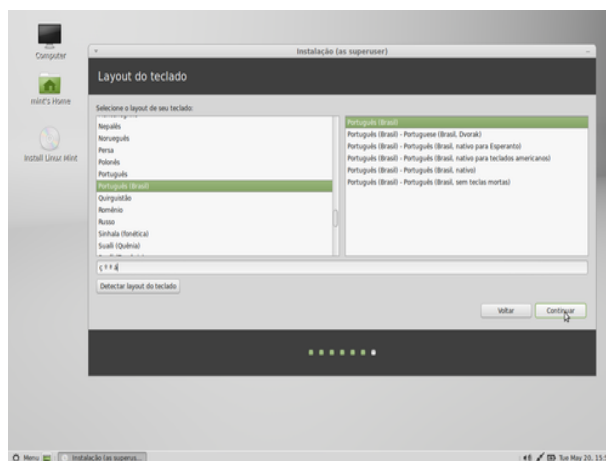


Figura 2.8: Escolha do layout do teclado.

- Crie seu nome de usuário e senha:

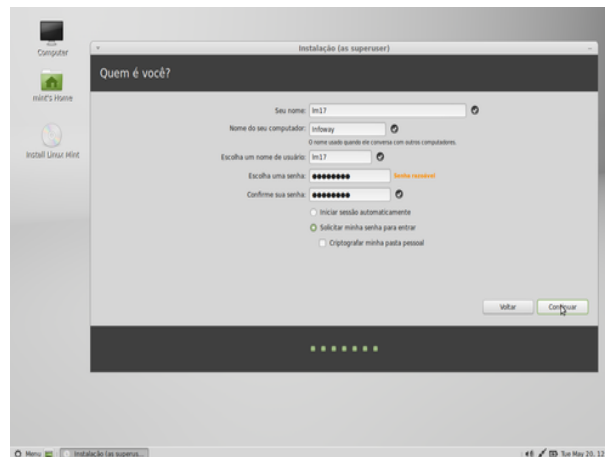


Figura 2.9: Definição do usuário e sua senha.

- Aguarde a instalação:

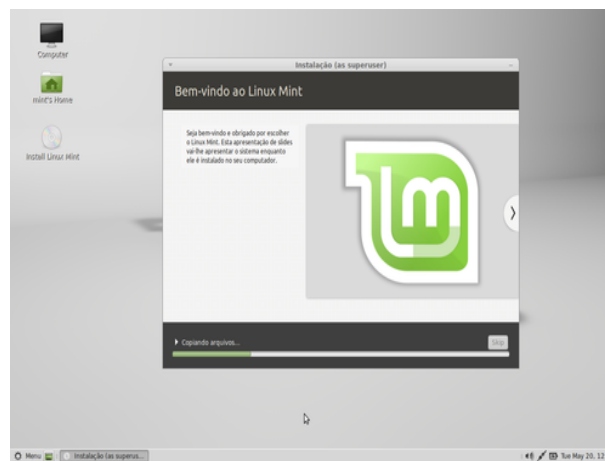


Figura 2.10: Tela de espera.

- Concluída a instalação, clique em Reiniciar Agora:



Figura 2.11: Tela para finalização da instalação.

- Bem-vindo ao Linux Mint 17 Qiana:

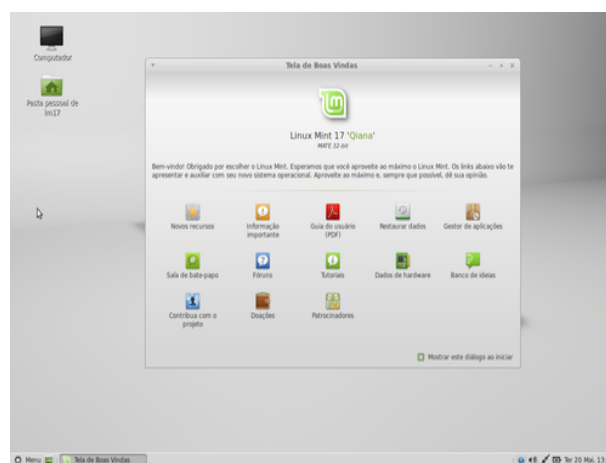


Figura 2.12: Tela de saudações.

## 2.4 Instalação do LLVM

Fácil instalação. O LLVM esta na versão 3.5 estável (versão 3.6 em processo de desenvolvimento).

### 2.4.1 Instalação

```
>sudo apt-get install llvm-3.4-dev llvm-3.4-doc llvm-gcc-4.8
```

Utilizando:

1. Escreva programa na linguagem da plataforma em um arquivo de formato \*.ll (por exemplo, hello.ll).
2. Abra o terminal e insira o seguinte comando:

```
>llvm-as hello.ll  
  
>llc hello.bc  
  
>gcc -o hello.s
```

Também consta disponível arquivo Makefile pré configurado onde basta modificar o nome do arquivo ll e o nome do executável. Para executar esse arquivo no terminal faça:

```
>make -f Makefile
```

## 2.5 Instalação do OCAML

Fácil instalação. Apenas digite no terminal:

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

# Capítulo 3

## LLVM

### 3.1 Aprendendo um pouco de LLVM

LLVM pode fornecer as camadas do meio de um sistema de compilador completo, tomando forma intermediária de código (IF) de um compilador e emitindo um IF otimizado. Este novo IF pode então ser convertido e vinculado a dependente da máquina código de montagem para uma plataforma de destino. LLVM pode aceitar o IF do GCC toolchain, permitindo que ele seja usado com uma grande variedade de compiladores existentes escritas para esse projeto.

LLVM também pode gerar o código de máquina relocatable em tempo de compilação ou link-time ou mesmo código de máquina binária em tempo de execução.

LLVM suporta um independente de linguagem conjunto de instruções e tipo de sistema. Cada instrução é em forma de atribuição única estática (SSA), o que significa que cada variável (chamado de registro digitado) é atribuída uma vez e está congelado. Isso ajuda a simplificar a análise de dependências entre as variáveis. LLVM permite que o código para ser compilado estaticamente, pois é no âmbito do sistema GCC tradicional, ou para a esquerda para a tarde-compilação da IF para código de máquina em um compilador just-in-time (JIT) de forma semelhante ao Java. O tipo de sistema é composto por tipos básicos, como inteiros ou carros alegóricos e cinco tipos derivados: ponteiros, matrizes, vetores, estruturas e funções. A construção tipo em uma linguagem concreta pode ser representada pela combinação desses tipos básicos em LLVM. Por exemplo, uma classe em C++ pode ser representado por uma combinação de estruturas, funções e matrizes de ponteiros de função.

O compilador JIT LLVM pode otimizar ramos estáticos desnecessários fora de um programa em tempo de execução, e, portanto, é útil para a avaliação parcial nos casos em que um programa tem muitas opções, a maioria dos quais pode ser facilmente determinada desnecessários em um ambiente específico. Este recurso é usado no OpenGL gasoduto de Mac OS X Leopard (10.5) para fornecer suporte para a falta de recursos de hardware. Código de gráficos dentro da pilha OpenGL foi deixado em forma intermediária, e depois compilada quando executado na máquina de destino. Em sistemas com high-end GPUs



, o código resultante era muito fina, passando as instruções para a GPU, com alterações mínimas. Em sistemas com GPUs low-end, LLVM seria compilar procedimentos opcionais que são executados no local, unidade central de processamento (CPU) que emular instruções que a GPU não pode ser executado internamente. LLVM melhorou o desempenho em máquinas low-end, usando Intel GMA chipsets. Um sistema similar foi desenvolvido sob a Gallium3D LLVMpipe, e incorporada no GNOME Shell para permitir que ele seja executado sem um driver de hardware 3D adequada carregado.

Quando se trata de desempenho de tempo de execução dos programas compilados, GCC anteriormente superou LLVM por cerca de 10 por cento, em média. resultados mais recentes indicam, no entanto, que LLVM já apanhados com GCC nesta área, e agora está a compilar os binários de aproximadamente igual desempenho, com exceção de programas usando OpenMP .

O núcleo do LLVM é a representação intermediária (IR), uma linguagem de programação de baixo nível semelhante ao de montagem. IR é um conjunto de instruções RISC fortemente tipificada que abstrai os detalhes do alvo. Por exemplo, a convenção de chamada é captada através de chamadas e ret instruções com argumentos explícitos. Além disso, em vez de um conjunto fixo de registros, IR usa um conjunto infinito de temporários do formato %0, %1, etc. LLVM suporta três formas isomórficas do IR: um formato de montagem legível, um formato de C++ objeto adequado para frontends e um formato bitcode densa para serialização. Um simples "Olá, mundo!" programa no formato de montagem:

```
@.Str = Constante interno [14 x i8] c "Olá , mundo \ 0A \ 00"

declararprintf i32 (i8 *, ...)

definirmain i32 (i32\% argc, i8\% ** argv) nounwind {
  entrada:
  \% Tmp1 = getelementptr [14 x i8] * @ .Str, i32 0, i32 0
  \% TMP 2 = chamada i32 (i8 *, ...) *printf (i8 *\% tmp1) nounwind
  ret i32 0
}
```

## 3.2 Tipos do LLVM

Primitivos:

- i1, i2, ... i8, ... i16, ... i32
- label, void
- float e double

Derivados:

- Array: [40 x i32]

Pointers: [4 x i8]\*

Structures: i32, (i32)\*, i1

Function -  $\langle \text{tipo}_{etorno} \rangle (\langle \text{parametro}_{ist} \rangle) : i32(i32)$

getelementptr

$\langle \text{result} \rangle = \text{getelementptr } \langle \text{pty} \rangle^* \langle \text{ptrval} \rangle, \langle \text{ty} \rangle \langle \text{idx} \rangle^*$

O primeiro argumento sempre é um ponteiro

Serve para acessar estruturas a partir do primeiro operando

# Capítulo 4

## Ocaml

### 4.1 Aprendendo um pouco de Ocaml

Em OCaml não há pontos de entrada como em Pascal (com o `begin` principal e `end` principal) ou como em C (a função `int main()`).

Um programa Ocaml é uma sequência de declarações ou de expressões. Uma declaração é definida por:

```
let identificador = expressao
```

As expressões e as declarações podem ser colocadas entre parêntesis ou `begin end`, ou por `;;` finais (esta ultima opção é obrigatória no toplevel)

```
( expressao )

begin

expressao

end

expressao;;
```

As variáveis Ocaml são, por defeito, como em matemática. É uma associação única entre identificador e valor. Uma variável tem um único valor (este não pode mudar). Uma expressão ocaml é uma expressão que devolve um valor (OCaml é uma linguagem fortemente tipada, logo tudo tem um tipo)

Estes exemplos abaixo ilustram como varias variáveis de mesmo nome co-existam e se escondam umas as outras (a ultima tem sempre prioridade)

Mais. Se nenhum valor depende duma determinada variável, digamos `x`, e esta está escondida por outras variáveis, então `x` esta definitivamente inacessível e "inútil". o recuperador de memória libertará automaticamente o espaço ocupado por `x`.

```

let x = 10;;
x;;

let y = 3 + x;;

let x = 6;;

y;;
x;;
let y = y - x;;
y;;

```

OCaml é fortemente tipado. As conversões entre tipos devem ser explicitamente colocadas pelo programador.

Cuidado, como em muitas linguagens (como em C), a aritmética de flutuantes em ocaml, normalizada pelo IEEE 754, é uma aritmética com aproximações.

OCaml é uma linguagem fortemente tipada, mas o tipo não precisa de ser fornecido: o OCaml consegue *\*inferir\** o tipo de cada expressão

Declarações locais:

let identificador = expressão1 in expressão2

O identificador assim definido tem por valor expressão1 e por alcance (i.e. é válido em) expressão2. Fora desta expressão2, identificador já não existe.

```

let x = 9 in x*x;;

let x = 10;;

(* as declarações podem ser aninhadas *)
let x = 9 in let y = 10 in let t = 2 in
(x+y+t)*(x-y-t);;

x;;

```

condicional funcional:

a expressão if then else não foge a regra das outras expressões: tem de devolver um valor. O valor devolvido no then tem de ser do mesmo tipo que o valor devolvido no else.

```

let x = 5;;

let v = if x > 7 then 9 else 1;;

```

```
if v > 3 then print\_int 5 else print\_int 6;;  
  
let w = 1 + (if x = 0 then 2 else 3)
```

Funções são valores como outros.  
Definição simples numa função:

```
let nome lista_de_parametros = expresao
```

se a função for recursiva, utiliza-se a declaração:

```
let rec nome lista_de_parametros = expresao
```

```
let f x y z = x + y + z;;  
  
f 3 4 5;;  
  
((f 4) 5 6);;  
  
let h = f 4;;  
let h y z = f 4 y z;;  
  
h 5 6;;  
  
f 4;;  
  
let u y x z = f x y z;;  
u 5;;  
  
let operacao x y f = f x y;;  
  
operacao 4 8 (+);;  
  
operacao 4.9 8.1 (+.);;  
  
operacao [1;2;3;4] [4;5;6;7] (fun x y -> List.length x + List.  
    length y);;  
  
let g x y z = x +. y +. z;;  
  
let h x y = (int\_of\_float x) + y  
  
let h' (x:int) y = (int\_of\_float x) + y  
  
h 6.0 9;;
```

# Capítulo 5

## Compilador

O compilador traduz o programa de uma linguagem (fonte) para outra (máquina).

- Esse processo demanda sua quebra em várias partes, o entendimento de sua estrutura e significado.
- O front-end do compilador é responsável por esse tipo de análise.

### **Análise Léxica:**

- Quebra a entrada em palavras conhecidas como tokens.

### **Análise Sintática:**

- Analisa a estrutura de frases do programa.

### **Análise Semântica:**

- Calcula o “significado” do programa.

### **Vantagens da divisão em análise léxica e sintática:**

Projeto mais simples. Diminui a complexidade do analisador sintático que não precisa mais lidar com estruturas foras de seu escopo como tratamento de caracteres vazios.

Melhorar a eficiência do compilador. Técnicas de otimização específicas para o analisador léxico.

Melhor portabilidade. Particularidades da linguagem fonte podem ser tratadas diretamente pelo analisador léxico

## 5.1 Analisador Léxico

O analisador léxico separa a seqüência de caracteres que representa o programa fonte em entidades ou tokens, símbolos básicos da linguagem. Durante a análise léxica, os tokens são classificados como palavras reservadas, identificadores, símbolos especiais, constantes de tipos básicos (inteiro real, literal, etc.), entre outras categorias. O analisador léxico é

Lê o texto fonte, carácter por carácter identificando os “elementos léxicos” da linguagem (identificadores), ignorando comentários, brancos, tabse processa includes (se for o caso).

## 5.2 Analisador Sintático

O analisador sintático agrupa os tokens fornecidos pelo analisador léxico em estruturas sintáticas, construindo a árvore sintática correspondente. Para isso, utiliza uma série de regras de sintaxe, que constituem a gramática da linguagem fonte. É a gramática da linguagem que define a estrutura sintática do programa fonte.

O analisador sintático tem também por tarefa o reconhecimento de erros sintáticos, que são construções do programa fonte que não estão de acordo com as regras de formação de estruturas sintáticas como especificado pela gramática.

## 5.3 Analisador Semântico

Embora a análise sintática consiga verificar se uma expressão obedece às regras de formação de uma dada gramática, seria muito difícil expressar através de gramáticas algumas regras usuais em linguagem de programação, como “todas as variáveis devem ser declaradas” e situações onde o contexto em que ocorre a expressão ou o tipo da variável deve ser verificado.

O objetivo da análise semântica é trabalhar nesse nível de inter-relacionamento entre partes distintas do programa. As tarefas básicas desempenhada durante a análise semântica incluem a verificação de tipos, a verificação do fluxo de controle e a verificação da unicidade da declaração de variáveis. Dependendo da linguagem de programação, outros tipos de verificações podem ser necessários.

# Capítulo 6

## Árvore Sintática Abstrata

Uma árvore sintática é uma árvore onde todos os seus nós são nomeados. Os nós internos são nomeados com os não- terminais da gramática e os nós folha são nomeados com os terminais da gramática. Cada filhos de um determinado nó, onde este nó tem associado a ele um não- terminal , representa um passo no processo de derivação de uma expressão reconhecida pela gramática.

### 6.1 Derivação da Árvore

Uma derivação é uma série de passos de reescrita, a cada passo escolhemos um nó não-terminal para reescrever.

Dois processos de derivação de interesse:

- Uma derivação mais à esquerda é o processo de derivação onde o não terminal mais a esquerda é utilizado para seguir no processo de derivação. - Uma derivação mais à direita é o processo de derivação onde o não terminal mais a direita é usado para seguir no processo de derivação.

Mais à esquerda

Regra	Forma Sentencial
—	Expr
0	Expr Op Expr
2	<id,x> Op Expr
4	<id,x> - Expr
0	<id,x> - Expr Op Expr
1	<id,x> - <num,z> Op Expr
5	<id,x> - <num,z> * Expr
2	<id,x> - <num,z> * <id,y>

Isso avalia como  $x - (z * y)$

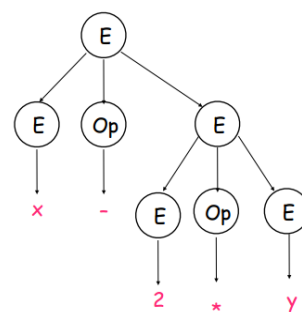


Figura 6.1: Derivação a esquerda.



Mais à droite

Rearra	Forma Sentencial
—	Expr
0	Expr Op Expr
2	Expr Op <id,y>
5	Expr * <id,y>
0	Expr Op Expr * <id,y>
1	Expr Op <num,2> * <id,y>
4	Expr - <num,2> * <id,y>
2	<id,x> - <num,2> * <id,y>

Isso avalia como  $(x - 2) * y$

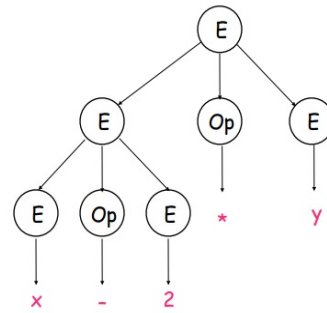


Figura 6.2: Derivação a direita.

## 6.2 Gramáticas Ambíguas

- Se uma gramática tem mais de uma derivação mais à esquerda para uma mesma forma sentencial, a gramática é ambígua. - Se uma gramática tem mais de uma derivação mais à direita para uma mesma forma sentencial, a gramática é ambígua. - As derivações mais à esquerda ou mais à direita podem diferir mesmo em uma gramática não ambígua.

Uma árvore sintática abstrata tem que ser a mesma.

## 6.3 Derivações e Precedência

Para adicionar precedência

- Crie um não-terminal para cada nível de precedência
- Isole a parte correspondente da gramática
- Force a árvore a reconhecer subexpressões de maior precedência primeiro

### Para expressões algébricas

- Parênteses primeiro
- Multiplicação e divisão, depois
- Adição e subtração, por último.

### Derivações e Precedência

Adicionando precedência algébrica tradicional produz:

	0	$\mathcal{G}$	$\rightarrow$	Expr
nível 3	1	Expr	$\rightarrow$	Expr + Termo
	2			Expr - Termo
	3			Termo
nível 2	4	Termo	$\rightarrow$	Termo * Fator
	5			Termo / Fator
	6			Fator
nível 1	7	Fator	$\rightarrow$	( Expr )
	8			<u>num</u>
	9			<u>id</u>

Figura 6.3: Exemplo de derivação e precedência.

# Capítulo 7

## Compilador - MiniJava para LLVM

### 7.1 Makefile

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

geraCodigo: compRepInter geracodigo.cmo

compRepInter: compSem repInter.cmo

compInter: compSem interpretador.cmo

compSem: compSint semantico.cmo

compSint: lexico.cmo sintatico.cmo

geracodigo.cmo: arvRI.cmi geracodigo.ml
$(CAMLC) -c geracodigo.ml

repInter.cmo: arvRI.cmi repInter.ml
$(CAMLC) -c repInter.ml

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

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

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

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

```

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

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

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

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

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

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

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

clean:
rm -f *.cmo *.cmi lexico.ml sintatico.ml sintatico.mli sintatico.output
CAMLC=ocamlc
CAMLLEX=ocamllex
CAMLYACC=ocamlyacc

geraCodigo: compRepInter geracodigo.cmo

compRepInter: compSem repInter.cmo

compInter: compSem interpretador.cmo

compSem: compSint semantico.cmo

compSint: lexico.cmo sintatico.cmo

geracodigo.cmo: arvRI.cmi geracodigo.ml
$(CAMLC) -c geracodigo.ml

repInter.cmo: arvRI.cmi repInter.ml
$(CAMLC) -c repInter.ml

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

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

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

```

```

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

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

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

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

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

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

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

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

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

```

## 7.2 Código - Carregador

```

(* arquivo carregador.ml *)

#load "arvSint.cmo";;
#load "sintatico.cmo";;
#load "lexico.cmo";;
#load "semantico.cmo";;
(*#load "interpretador.cmo";;*)
#load "repInter.cmo";;
#load "geracodigo.cmo";;

open Sintatico;;
open ArvSint;;
open Semantico;;
(*open Interpretador;;*)
open Printf;;
open RepInter;;
open Geracodigo;;

```

```

let sintatico lexbuf =
try
Sintatico.programa Lexico.token lexbuf
with exn ->
begin
let tok  = Lexing.lexeme lexbuf in
let pos  = lexbuf.Lexing.lex_curr_p in
let nlin = pos.Lexing.pos_lnum in
let ncol = pos.Lexing.pos_cnum - pos.Lexing.pos_bol - String.length tok
in
let msg1 = sprintf "Erro na linha %d, coluna %d" nlin ncol in
let msg2 = sprintf "\tA palavra \"%s\" nao era esperada aqui." tok in
print_endline msg1;
print_endline msg2;
flush stdout;
raise exn
end

let sint_str str =
let lexbuf = Lexing.from_string str in
sintatico lexbuf

let sint_arq arq =
let ic = open_in arq in
let lexbuf = Lexing.from_channel ic in
let arv = sintatico lexbuf in
close_in ic;
arv

let sem_arq arq =
let arv = sint_arq arq in
semantico arv
(*)
let interp arq =
let arv = sint_arq arq in
let result  = semantico arv in
interpretador (fst result) (snd result) arv
*)
let repInter arq =
let arv = sint_arq arq in
let result  = semantico arv in
reprIntermediaria (fst result) (snd result) arv

let nomeArquivo arq =
let len = String.length arq in
let posBarra  = (String.rindex arq '/') + 1 in
let directorio = String.sub arq 0 posBarra in
let lenDir    = String.length directorio in
let arquivo   = String.sub arq lenDir (len-lenDir-5) in
(diretorio, arquivo)

```

```
let miniJavaLLVM arq =  
let arqlvm = nomeArquivo arq in  
let arvRI    = repInter arq in  
geraCodigo (fst arqlvm) (snd arqlvm) arvRI
```

## 7.3 Analisador Léxico

### 7.3.1 Descrição - Analisador Léxico

A etapa de análise léxica é a primeira etapa da construção "front-end" de um compilador. Esta etapa é responsável por receber todo o código-fonte digitado pelo usuário e filtrar apenas palavras essenciais para a execução deste código. Para recuperar estas palavras, estas devem estar como uma subsequência de caracteres presentes no código-fonte.

Tais palavras válidas recebem o nome de tokens. Portanto, pertencem a este conjunto de tokens:

- palavras reservadas da linguagem;
- constantes (valores inteiros, flutuantes, caracteres);
- identificadores (nomes de classes, metodos e variaveis);
- símbolos (por exemplo , , ( , ) , ;, ...); operadores unários e binários (por exemplo '+', '-', '\*', '/', '');

Note que palavras como espaços, tabulações e saltos de linha são ignorados pelo analisador léxico.

No nosso compilador, por questões de simplificação, consideramos como tokens comandos de leitura vindos da classe `java.util.Scanner` da linguagem Java (como, por exemplo, os metodos `nextInt()`), assim como os comandos para escrita `System.out.print()` e `System.out.println()`. Para utilizarmos os comandos de leitura, necessitamos de um token representando o comando em Java `import java.util.Scanner`.

Para representar valores constantes (como inteiros, caracteres e números de ponto flutuantes), utilizaremos expressões regulares para descrever tais tipos. Desta forma, se surgir uma subsequência no código-fonte que respeite a expressão regular definida, logo esta subsequência será interpretada como um valor para aquela definição constante. As expressões regulares são formadas de forma semelhante às expressões regulares na teoria das Linguagens Formais e Autômatos.

As expressões regulares utilizadas na análise léxica foram:

- dígitos: valores de '0' a '9'; ['0' - '9']
- identificador: uma letra alfabética (seja maiúscula ou minúscula) seguidas de uma sequência (zero ou mais caracteres) de letras ou dígitos ou underline

```
( _ ); [ 'a' - 'z' 'A' - 'Z' ] [ ' _ ' 'a' - 'z' 'A' - 'Z' '0' - '9' ] }
```

- booleano: true ou false; "true "false"
- numeroFloat: sequência (um ou mais caracteres) de dígitos seguido de um ponto e de outra sequência (um ou mais caracteres) de dígitos digito+ '.' digito+

Para maior entendimento dos padrões necessários para o entendimento, podem acessar tal link como referência: [http://en.wikipedia.org/wiki/Regular\\_expression#Standards](http://en.wikipedia.org/wiki/Regular_expression#Standards)  
Quaisquer subsequência presente no código-fonte que não respeite as restrições de subsequências para formação de tokens é denominada como um erro léxico. Exemplos de erros léxicos são quando palavras reservadas são digitadas de forma incorreta ou quando são inseridos palavras no código-fonte as quais não pertencem à linguagem.

### 7.3.2 Código - Analisador Léxico

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

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

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

  let msg_erro lexbuf c =
```



```

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

let msg_erro_comentario lexbuf s =
let pos = lexbuf.lex_curr_p in
let lin = pos.pos_lnum
and col = pos.pos_cnum - pos.pos_bol - 1 in
sprintf "%d-%d: final de comentario %s utilizado errado" lin col s
}

let digito          = ['0' - '9']
let identificador   = ['a'-'z' 'A'-'Z'] ['_' 'a'-'z' 'A'-'Z' '0'-'9']*
let booleano        = "true" | "false"
let numeroFloat     = digito+ '.' digito+
let strings         = '"' identificador* digito+ '"'
let character       = ['_' 'a'-'z' 'A'-'Z' '0'-'9']

rule token = parse
[' ' '\t' ]          { incr_num_col lexbuf; token lexbuf } (* ignora
    espa os *)
| ['\n']             { incr_num_linha lexbuf; token lexbuf } (* ignora
    fim de linha *)
| " //"             { simple_comment lexbuf } (* ignora comentario *)
| " /*"             { full_comment lexbuf }
| " */"             { failwith (msg_erro_comentario lexbuf "*/"); }
| booleano as bol    { let value = bool_of_string bol in
    LitBoolean (value)}
| digito+ as num     { let numero = int_of_string num in
    LitInt (numero) }
| numeroFloat as num { let value = float_of_string num in
    LitDouble (value) }
| '"'               { let buffer = Buffer.create 1 in
    LitString (cadeia buffer lexbuf) }
| '\\' (character as c) '\\' { LitChar (c) }
| "public static void main" { Main }
| "public"           { Public }
| "private"          { Private }
| "static"           { Static }
| "class"            { Class }
| "new"              { New }
| "return"           { Return }
| "void"             { Void }
| "int"              { Int }
| "char"             { Char }
| "float"            { Float }
| "double"           { Double }
| "String"           { String }
| "boolean"          { Boolean }
| "if"               { If }

```

```

| "else"           { Else           }
| "for"            { For            }
| "do"             { Do             }
| "while"          { While           }
| "switch"         { Switch          }
| "case"           { Case            }
| "default"        { Default         }
| "break"          { Break           }
| "continue"       { Continue        }
| "this"           { This            }
| "null"           { Null            }
| "++"             { OpIncr          }
| "--"            { OpDecr          }
| '+'             { OpSoma           }
| '-'             { OpSub            }
| '*'             { OpMul            }
| '/'             { OpDiv            }
| '%'             { OpMod            }
| '!'             { OpNao            }
| '&&'            { OpE             }
| '||'            { OpOu             }
| '<'            { OpMenor           }
| '<='           { OpMenorIgual      }
| '=='            { OpIgual           }
| '!='            { OpDiferente       }
| '>'            { OpMaior           }
| '>='           { OpMaiorIgual      }
| '='             { Atrib             }
| '('             { AParen            }
| ')'             { FParen            }
| '['             { AColc            }
| ']'             { FColc            }
| '{'             { AChave            }
| '}'             { FChave            }
| ';'             { PTVirg           }
| ','             { Virg             }
| '.'             { Ponto            }
| ':'             { DoisPontos        }
| "System.out.print" { Imprime          }
| "System.out.println" { ImprimeSalta      }
| "import java.util.Scanner" { ImportScanner }
| "Scanner"         { Scanner          }
| "System.in"        { SystemIn          }
| "nextBoolean"      { LeBool            }
| "nextDouble"       { LeDouble           }
| "nextFloat"        { LeFloat            }
| "nextInt"          { LeInt             }
| "nextByte"         { LeByte            }
| "nextLine"         { LeString           }
| identificador as id { Ident (id)         }
| _ as c             { failwith (msg_erro lexbuf c); }

```

```

| eof                { EOF                }

(* para tratar comentarios completos *)
and full_comment = parse
| "*/" { token lexbuf }
| '\n' { incr_num_linha lexbuf; full_comment lexbuf }
| _    { full_comment lexbuf }

(* para tratar comentarios simples *)
and simple_comment = parse
| '\n' { incr_num_linha lexbuf; token lexbuf }
| _    { simple_comment lexbuf }

(* para criar cadeias de strings *)
and cadeia buffer = parse
| '"'      { Buffer.contents buffer }
| '\t'     { Buffer.add_char buffer '\t'; cadeia buffer lexbuf }
| '\n'     { Buffer.add_char buffer '\n'; cadeia buffer lexbuf }
| '\\\ ' "' { Buffer.add_char buffer ' ' ; cadeia buffer lexbuf }
| '\\\ '\\\ { Buffer.add_char buffer '\\\'; cadeia buffer lexbuf }
| _ as c    { Buffer.add_char buffer c ; cadeia buffer lexbuf }
| eof       { failwith "string nao foi fechada" }

```

## 7.4 Analisador Sintático

### 7.4.1 Descrição - Analisador Sintático

Através do analisador léxico, obtemos uma sequência de tokens, na qual tais tokens estão ordenados conforme a ordem das palavras no código-fonte. Nesta etapa, foram removidos caracteres irrelevantes e eliminado erros léxicos. Resta analisar esta sequência de tokens de tal forma que respeite a gramática (sintaxe) da linguagem MiniJava. Tal análise é realizada pelo analisador sintático.

A etapa de análise sintática é a segunda etapa da construção "front-end" de um compilador. Ela é responsável por receber a sequência de tokens e, a partir dela, verificar se a sequência de tokens respeita a sintaxe da linguagem analisada pelo compilador. Portanto, para a linguagem ser aceita pelo analisador sintático, os tokens recebidos pelo analisador léxico devem estar corretamente distribuídos em sequência em relação a gramática da linguagem.

Relacionando a análise sintática com a teoria das Linguagens Formais e Autômatos, a gramática do analisador sintático é uma Gramática Livre de Contexto. Isto significa que uma regra desta gramática é definida no formato de:

$$S \rightarrow \text{Expressao } w;$$

Onde S é um símbolo não-terminal e w uma sequência de símbolos terminais e não-terminais

No caso do nosso analisador sintático, os símbolos terminais são os tokens definidos e os símbolos não-terminais são símbolos que nos auxiliam para a construção das regras. Da mesma forma que uma gramática livre de contexto, uma entrada é válida se ao finalizar a análise não existir símbolos não-terminais do lado direito da regra.

No exemplo a seguir apenas os símbolos +, -, 0 e 1 são símbolos terminais. Os demais símbolos são símbolos não-terminais:

```
S -> Expressao Expressao
S -> Expressao Valor
Expressao -> Expressao Expressao + Valor
Expressao -> Expressao - Valor
Valor -> Expressao 1
Valor -> Expressao 0
```

Para a linguagem MiniJava, as regras utilizadas (em descrição em alto nível) são:

- Um programa deve conter uma ou várias classes. Pode também conter antes das definições de classes o comando para importar classe `java.util.Scanner`;
- Uma classe é formada em sequência por uma declaração de classe, e, entre chaves, as variáveis globais e métodos pertencentes à classe;
- A declaração de classe deve ser:

```
Public Class <nome_classe>
```

```
Class <nome_classe>
```

- As variáveis globais devem ser formadas da seguinte forma:

```
Private Static tipo declaracao de variavel 1, ... ,
declaracao de variavel n;
```

```
Private Static tipo declaracao de variavel;
```

- Uma declaração de variável deve ser formada da seguinte forma:

<nome\_variavel>

<nome\_variavel> = expressao

- Um metodo deve ser formado da seguinte forma:

Public Static tipo <nome\_metodo> (parametros) instrucoes

- Cada parâmetro é separado por virgula (,) e deve estar no seguinte formato:

tipo <nome\_variavel\_parametro>

- As instruções são divididos em comandos e blocos.
- Todo comando deve ser finalizado por ponto-e-vírgula (;) e são estes

<variavel> = expressao

<variavel> ++

<variavel> - -

System.out.print( expressao )

System.out.println( expressao )

Scanner <variavel> = new Scanner(System.in)

<variavel> = <variavel>.nextInt()

<variavel> = <variavel>.nextBoolean()

<variavel> = <variavel>.nextDouble()

<variavel> = <variavel>.nextFloat()

<variavel> = <variavel>.nextByte()

`<variavel> = <variavel>.nextLine()`

`break`

`continue`

`return expressao`

chamada de metodo

- Os blocos devem ser limitados por `{}` . Estes são definidos como:

`if (expressao ) instrucoes`

`if (expressao ) instrucoes else instrucoes`

`while(expressao) instrucoes`

`do instrucoes while(expressao)`

`switch(expressao) cases`

`for(inicializacao ; expressao ; incremento) instrucoes`

- É permitido criar uma variável no bloco `for`. Além disso, as atribuições podem ser tanto incremento da variável, decremento da variável ou uma expressão
- O bloco `switch` deve ao menos incluir um bloco `case` ou um bloco `default`.
- É permitido pela análise sintática blocos `case` não conterem o comando `break`.
- Uma expressão envolve todos os tipos de operações do programa. As operações, em ordem de precedência (serão interpretadas com maior grau de prioridade) são:

literais, chamadas de método e expressões entre parênteses

não lógico, operador unário positivo e operador unário negativo

multiplicação, divisão e módulo

soma e subtração

operadores relacionais

e lógico e ou lógico

- Toda chamada de método deve estar no seguinte formato:

<nome\_classe> . <nome\_método> (argumentos)

- A lista de argumentos pode ser vazia ou conter varias expressões separadas por vírgula.
- Um literal são constantes pertencentes a algum tipo.
- Um tipo pode ser int, char, double, String, boolean ou void.

Em Linguagens Formais e Autômatos, uma gramática pode gerar uma árvore de derivação. Esta árvore é denominada árvore sintática concreta. Para facilitar as futuras análises, esta árvore será simplificada para uma árvore sintática abstrata.

(Explicação sobre arvores de derivação e expressões contidas no relatório)

### 7.4.2 Código - Analisador Sintático

```
%{  
  
open ArvSint;;  
  
%}  
  
%token <int> LitInt  
%token <float> LitFloat  
%token <float> LitDouble  
%token <bool> LitBoolean  
%token <string> LitString  
%token <char> LitChar  
%token <string> Ident  
%token Public Private Static  
%token Class New Return  
%token Main
```

```

/* tipos de variaveis */
%token Void Int Char Float Double String Boolean

/* Controles de fluxo */
%token If Else
%token For Do While
%token Switch Case Default
%token Break Continue

/* Operadores aritmeticos, comparativos e logicos */
%token OpSoma OpSub OpMul OpDiv OpMod
%token OpIncr OpDecr
%token OpNao OpE OpOu
%token OpMenor OpMenorIgual
%token OpIgual OpDiferente
%token OpMaior OpMaiorIgual
%token Atrib

/* Estruturacao */
%token AParen FParen
%token AColc FColc
%token AChave FChave
%token PTVirg Virg Ponto DoisPontos
%token This Null

/* Para escrita System.out.print System.out.println*/
%token Imprime ImprimeSalta

/* Para leitura usando Scanner */
%token ImportScanner
%token Scanner SystemIn
%token LeBool LeDouble LeFloat LeInt LeByte LeString

%token EOF

%start programa /* simbolo inicial da gramatica */

%type <ArvSint.programa> programa

%%

programa: classes EOF { (false, $1)}
| ImportScanner PTVirg classes EOF {(true, $3)}
;

/* ***** declaracao de classe
***** */

classes: classe {[$1]}

```



```

| classe classes {[$1] @ $2}
;

classe: classe_declaracao AChave variaveis_globais metodos FChave {($1, $3,
    $4, Posicao.pos(1))}
;

classe_declaracao: Public Class Ident { $3 }
| Class Ident { $2 }
;

/* ***** declaracao de variaveis
***** */

variaveis_globais: /* nada */ { [] }
| variaveis_tipo_global PTVirg variaveis_globais { $1 @ $3 }
;

variaveis_locais: /* nada */ { [] }
| variaveis_tipo_local PTVirg variaveis_locais { $1 @ $3 }
;

variaveis_tipo_global: Private Static tipo var_declaracoes {List.map(fun (id
    , value, pos) -> (id, $3, value, pos)) $4}
;

variaveis_tipo_local: tipo var_declaracoes {List.map(fun (id, value, pos) ->
    (id, $1, value, pos)) $2}
;

var_declaracoes: var_declaracao {[$1]}
| var_declaracao Virg var_declaracoes {[$1] @ $3}
;

var_declaracao: Ident { ($1, ExpNone, Posicao.pos(1)) }
| Ident Atrib expressao { ($1, $3, Posicao.pos(1)) }
;

/* ***** metodos
***** */

/* *** metodos *** */
metodos: /* nada */ { [] }
| metodo metodos { [$1] @ $2 }
;

metodo : declaracao_metodo AChave variaveis_locais instrucoes FChave

```

```

{Metodo($1, $3, $4, Posicao.pos(1))}
| declaracao_main AChave variaveis_locais instrucoes FChave
{MetodoMain($1,$3, $4, Posicao.pos(1))}
;

declaracao_main: Main AParen FParen {[{}]}
| Main AParen String AColc FColc Ident FParen {[ (TString,$6,Posicao.pos(6))
  ]}
;

declaracao_metodo: Public Static tipo Ident AParen parametros FParen { ($4,
  $3, $6) }
;

/* *** parametros *** */
parametros: /* nada */ { [ ] }
| parametro { [$1] }
| parametro Virg parametros { [$1] @ $3 }
;

parametro: tipo Ident { ($1, $2, Posicao.pos(2)) }
;

/* ***** instrucoes ***** */

instrucoes: /* nada */ { [ ] }
| comando PTVirg instrucoes { [$1] @ $3 }
| bloco instrucoes { [$1] @ $2 }
;

/* ***** bloco ***** */

bloco: bloco_if { $1 }
| bloco_else { $1 }
| bloco_for { $1 }
| bloco_while { $1 }
| bloco_do_while { $1 }
| bloco_switch { $1 }
;

bloco_if: If AParen expressao FParen AChave instrucoes FChave {BlocoIf($3,
  $6, Posicao.pos(3))}
;

bloco_else: If AParen expressao FParen AChave instrucoes FChave Else AChave
  instrucoes FChave {BlocoIfElse($3, $6, $10, Posicao.pos(3))}
;

```

```

bloco_for: For AParen bloco_for_atrib PTVirg expressao PTVirg bloco_for_incr
    FParen AChave instrucoes FChave
{ BlocoFor(($3, ($5,Posicao.pos(5)), $7), $10, Posicao.pos(1)) }
;

bloco_while: While AParen expressao FParen AChave instrucoes FChave {
    BlocoWhile($3,$6,Posicao.pos(3))}
;

bloco_do_while: Do AChave instrucoes FChave While AParen expressao FParen
    PTVirg {BlocoDoWhile($7,$3,Posicao.pos(7))}
;

bloco_for_atrib: Ident Atrib expressao { AtribFor($1, $3, Posicao.pos(1)) }
| tipo Ident Atrib expressao { AtribVarFor($2, $1, $4, Posicao.pos(2)) }
;

bloco_for_incr: Ident Atrib expressao { ExpFor($1, $3, Posicao.pos(1)) }
| Ident OpIncr { IncrFor($1,Posicao.pos(1)) }
| Ident OpDecr { IncrFor($1,Posicao.pos(1)) }
;

bloco_switch: Switch AParen expressao FParen AChave bloco_cases FChave {
    BlocoSwitch($3,$6,Posicao.pos(3)) }
| Switch AParen expressao FParen AChave FChave { BlocoSwitch($3,[],Posicao.
    pos(3)) }
;

bloco_cases: bloco_case                {[$1]}
| bloco_default                {[$1]}
| bloco_case bloco_cases {[$1] @ $2}
;

bloco_case: Case expressao DoisPontos instrucoes {BlocoCase($2,$4,Posicao.
    pos(2))}
;

bloco_default: Default DoisPontos instrucoes {BlocoDefault($3, Posicao.pos
    (1))}
;

/* **** comandos **** */

comando: cmd_return    { $1 }
| cmd_break    { $1 }
| cmd_continue { $1 }
| cmd_read     { $1 }
| cmd_atrib    { $1 }
| cmd_inc_dec  { $1 }
| cmd_print    { $1 }
| cmd_println  { $1 }

```

```

| cmd_scanner { $1 }
| cmd_chamada_metodo { $1 }
;

cmd_atrib: Ident Atrib expressao { CmdAtrib($1, $3, Posicao.pos(1)) }
;

cmd_inc_dec: Ident OpIncr {CmdIncr($1,Posicao.pos(1))}
| Ident OpDecr {CmdDecr($1,Posicao.pos(1))}
;

cmd_print: Imprime AParen expressao FParen { CmdPrint($3,Posicao.pos(3)) }
;

cmd_println: ImprimeSalta AParen expressao FParen { CmdPrintln($3,Posicao.
    pos(3)) }
;

cmd_scanner: Scanner Ident Atrib New Scanner AParen SystemIn FParen {
    CmdScanner($2,Posicao.pos(3)) }
;

cmd_read: Ident Atrib Ident Ponto cmd_read_tipo AParen FParen { CmdRead($1,
    $3,$5,Posicao.pos(1),Posicao.pos(3)) }
;

cmd_read_tipo: LeBool {ReadBool }
| LeDouble {ReadDouble}
| LeFloat {ReadFloat }
| LeInt {ReadInt }
| LeByte {ReadByte }
| LeString {ReadString}
;

cmd_break: Break { CmdBreak(Posicao.pos(1)) }
;

cmd_continue:Continue { CmdContinue(Posicao.pos(1)) }
;

cmd_return: Return expressao { CmdReturn($2, Posicao.pos(2)) }
| Return { CmdReturn(ExpNone, Posicao.pos(1)) }
;

cmd_chamada_metodo: chamada_metodo { CmdChamadaMetodo($1) }
;

/* ***** chamada de metodo
    ***** */

```

```

chamada_metodo: Ident Ponto Ident AParen argumentos FParen { ($3, $1, $5,
    Posicao.pos(3)) }
;

argumentos: /* */ { [ ] }
| expressao { [ ($1,Posicao.pos(1)) ] }
| expressao Virg argumentos { [($1,Posicao.pos(1))] @ $3 }
;

/* ***** expresseoes
***** */

expressao: expressao OpOu exp10 {ExpBin(Ou, $1, $3)}
| expressao OpE exp10 {ExpBin(E, $1, $3)}
| exp10 {$1}
;

exp10: exp10 OpMenor exp20 {ExpBin(Menor, $1, $3) }
| exp10 OpMenorIgual exp20 {ExpBin(MenorIgual, $1, $3)}
| exp10 OpIgual exp20 {ExpBin(Igual, $1, $3) }
| exp10 OpDiferente exp20 {ExpBin(Diferente, $1, $3) }
| exp10 OpMaior exp20 {ExpBin(Maior, $1, $3) }
| exp10 OpMaiorIgual exp20 {ExpBin(MaiorIgual, $1, $3)}
| exp20 {$1}
;

exp20: exp20 OpSoma exp30 {ExpBin(Soma, $1, $3)}
| exp20 OpSub exp30 {ExpBin(Sub, $1, $3)}
| exp30 {$1}
;

exp30: exp30 OpMul exp40 {ExpBin(Mul, $1, $3)}
| exp30 OpDiv exp40 {ExpBin(Div, $1, $3)}
| exp30 OpMod exp40 {ExpBin(Mod, $1, $3)}
| exp40 {$1}
;

exp40: OpNao exp40 {ExpUn(Nao, $2)}
| OpSub exp40 {ExpUn(Negat, $2)}
| OpSoma exp40 {ExpUn(Posit, $2)}
| exp50 {$1}
;

exp50: exp60 OpIncr {ExpUn(Incr, $1)}
| exp60 OpDecr {ExpUn(Decr, $1)}
| exp60 {$1}
;

exp60: operando {$1}
| chamada_metodo { ExpMetodo($1) }

```

```

| AParen expressao FParen {$2}
;

operando: LitString      { ExpString($1)  }
| LitInt                { ExpInt($1)      }
/*| LitFloat            { ExpFloat($1)    }*/
| LitDouble             { ExpDouble($1)   }
| LitBoolean            { ExpBoolean($1)  }
| LitChar               { ExpChar($1)     }
| Ident {ExpVar($1)}
;

/* ***** regras gerais
***** */

tipo: Int      { TInt      }
| Char        { TChar      }
/*| Float      { TFloat    }*/
| Double      { TDouble    }
| String      { TString    }
| Boolean     { TBoolean   }
| Void        { TVoid      }
;

```

## 7.5 Código - Árvore Sintática Abstrata

```

(* utilizado para registrar a posicao de cada item da arvore no
codigo original em casos de erro
ao representar a posicao na arvore, utilizar o tipo Posicao.t nas tuplas
no sintatico, a posicao ser representada pela funcao Posicao.pos(n)
*)

module Posicao =
struct
  type t = { lin_inicial: int;
             col_inicial: int;
             lin_final:   int;
             col_final:   int
           }
  (* n representa qual simbolo da gramatica ser referenciado
  para recuperar a posicao
  Ex: cmd_atrib = exp Atrib exp
  para n = 2

  Parsing.rhs_start_pos 2 retorna a posicao (lexica) onde inicia
  o que corresponderia o simbolo A da gramatica no codigo.

  Assim ele apontar para a linha e coluna onde encontra o

```

simbolo de atribuicao

Parsing.rhs\_start\_pos 2 retorna que a posicao final a do simbolo Atrib. Desta forma, se lancar erro, a posicao a ser apontada ser a posicao onde o simbolo de atribuicao esta no codigo

```
*)
let pos n =
let pos_inicial = Parsing.rhs_start_pos n in
let pos_final   = Parsing.rhs_end_pos n in
let linha_inicial = pos_inicial.Lexing.pos_lnum
and coluna_inicial = pos_inicial.Lexing.pos_cnum -
pos_inicial.Lexing.pos_bol + 1
and linha_final   = pos_final.Lexing.pos_lnum
and coluna_final  = pos_final.Lexing.pos_cnum -
pos_final.Lexing.pos_bol
in
{ lin_inicial = linha_inicial;
  col_inicial = coluna_inicial;
  lin_final   = linha_final;
  col_final   = coluna_final
}
let npos n =
{ lin_inicial = Parsing.rhs_start(n);
  col_inicial = Parsing.rhs_end(n);
  lin_final   = 0;
  col_final   = 0
}
end

(* descrevendo programa , se existe import e a lista de classe*)
type programa      = bool * classes

(* descrevendo classes *)
and classes        = classe list
and classe         = string * vardeclaracao list * metodo list * Posicao.t

(* descrevendo variaveis *)
and vardeclaracao = string * tipo * expressao * Posicao.t

(* descrevendo metodos *)
and metodo         = Metodo      of infometodo * vardeclaracao list *
instrucao list * Posicao.t
| MetodoMain of parametros * vardeclaracao list * instrucao list * Posicao
.t
and infometodo     = string * tipo * parametros
and parametros     = parametro list
and parametro      = tipo * string * Posicao.t

(* * Posicao.t *)
(*descrevendo blocos e comandos*)
```

```

and instrucao      = BlocoIf      of expressao * instrucao list * Posicao.
t
| BlocoIfElse      of expressao * instrucao list * instrucao list * Posicao.t
| BlocoFor          of parametrosFor * instrucao list * Posicao.t
| BlocoWhile        of expressao * instrucao list * Posicao.t
| BlocoDoWhile      of expressao * instrucao list * Posicao.t
| BlocoSwitch       of expressao * casoSwitch list * Posicao.t
| CmdAtrib          of string * expressao * Posicao.t
| CmdChamadaMetodo  of chamadametodo
| CmdIncr           of string * Posicao.t
| CmdDecr           of string * Posicao.t
| CmdPrintln        of expressao * Posicao.t
| CmdPrint          of expressao * Posicao.t
| CmdScanner        of string * Posicao.t
(* vardestino, varscanner, tipoleitura, posicao_variavel_destino,
   posicao_variavel_scanner *)
| CmdRead           of string * string * tipoLeitura * Posicao.t * Posicao.t
| CmdReturn         of expressao * Posicao.t
| CmdBreak          of Posicao.t
| CmdContinue
(* descricao do parametros do for *)
and parametrosFor = inicializacaoFor * expressaoPos * incrementoFor
and inicializacaoFor = AtribFor      of string * expressao * Posicao.t
| AtribVarFor of string * tipo * expressao * Posicao.t
and expressaoPos = expressao * Posicao.t
and incrementoFor = ExpFor          of string * expressao * Posicao.t
| IncrFor      of string * Posicao.t
| DecrFor      of string * Posicao.t
(* descricao do cases no Switch..case *)
and casoSwitch = BlocoCase      of expressao * instrucao list * Posicao.t
| BlocoDefault of instrucao list * Posicao.t
(* descricao dos comandos de leitura *)
and tipoLeitura = ReadBool | ReadDouble | ReadFloat
| ReadInt | ReadByte | ReadString

(* descrevendo chamada de metodo *)
(* nome_metodo, classe referenciada, argumentos *)
and chamadametodo = string * string * expressaoPos list * Posicao.t
(* descrevendo expressoes *)
and expressao = ExpInt      of int
| ExpBoolean of bool
| ExpChar     of char
| ExpString   of string
| ExpFloat    of float
| ExpDouble   of float
| ExpVar      of string
| ExpMetodo   of chamadametodo
| ExpBin      of operador * expressao * expressao
| ExpUn       of operador * expressao
| ExpNone

```



```

and operador = Soma | Sub | Mul | Div | Mod
| Incr | Decr | Nao | E | Ou
| Menor | MenorIgual | Igual
| Diferente | Maior | MaiorIgual
| Posit      | Negat

(* gerais *)
and tipo      = TInt | TBoolean | TChar | TString | TFloat | TDouble |
               TVoid

```

O arquivo que será utilizado para utilizar o compilador ou algumas funções auxiliares será o arquivo a seguir:

## 7.6 Analisador Semântico

### 7.6.1 Descrição

Através do analisador sintático, obtemos a estrutura do código-fonte (gramática) representada através de uma árvore sintática abstrata. Através desta árvore é possível gerar o código-fonte para a máquina virtual correspondente. Entretanto, ainda restam alguns aspectos fundamentais a serem analisados.

Suponha o seguinte exemplo:

```
int n = 2.54 + "carro";
```

Note que no exemplo acima há um erro de tipo, pois uma variável inteira está recebendo por atribuição uma soma entre número de ponto flutuante e uma String. Além disso, não existe operação de soma entre número de ponto flutuante e uma String.

Note no exemplo a seguir que a variável valor nunca foi sequer declarada:

```

public static void imprime()
{
    System.out.println(valor);
}

```

Apesar de estar sintaticamente correto, estes erros não podem ser validados pelo compilador, pois são restrições que, apesar de não pertencerem à análise sintática, determinados contextos (como tipagem de variáveis) não são permitidos por uma linguagem de programação.

Erros como incompatibilidade de tipos, uso de variáveis não declaradas ou não inicializadas, chamadas de métodos não declarados, passagem incorreta de argumentos em

chamadas de métodos e entre outros erros nos quais dependem do contexto onde os tokens estão incluídos são detectados pelo analisador semântico. Portanto, erros que não respeitam as restrições definidas pela etapa de análise semântica são denominados erros semânticos.

Por ser inicializada a partir da análise sintática, isto faz da análise semântica a terceira e a última etapa fundamental na construção do “front-end” de um compilador.

No nosso compilador da linguagem MiniJava para máquina virtual LLVM, a análise semântica, fundamentalmente, detecta caso:

- O método main não estiver sido declarado no programa; método main não estiver sido declarado no programa;
- Declaração de classe já existente (com mesmo nome de uma classe já existente);
- Declaração de método com nome de um outro método já existente em uma mesma classe;
- Caso um método que não possui tipo de retorno void não retornar algum valor ou retornar um valor cujo tipo não é compatível com o retorno do método; Caso uma chamada de método (sem ser utilizado como expressão) retorna tipo diferente de void;
- Se o número de argumentos passados como parâmetro para uma chamada de método (ou uma expressão contendo método) for diferente que a quantidade de argumentos necessárias;
- Se algum argumento passado como parâmetro para uma chamada de método (ou uma expressão contendo método) tiver tipo incompatível com o tipo do parâmetro exigido pelo método;
- Declaração de variável (seja ela global ou local) com o mesmo nome de uma outra variável pré declarada em seu escopo (global ou local);
- Uma variável global, ao ser declarada, for atribuída a um valor que não seja um literal;
- Atribuir a uma variável ou declarar uma variável com uma expressão cujo tipo é incompatível com o tipo da variável;
- Utilizar variáveis que não foram declaradas; Quaisquer instrução após um comando break, garantindo que esta seja a última instrução em seu escopo;
- Realizar leitura com o usuário sem que a classe `java.util.Scanner` ter sido importada;

- Realizar um tipo de leitura com o usuário em uma variável de tipo incompatível;
- Se expressão de condição nos blocos if, if/else, while, do/while e for não for tipo boolean; Se alguma variável utilizada para iteração em um bloco for não for de tipo int;
- Se alguma variável tipo int declarada em algum bloco for for utilizada fora de seu escopo; Se expressão de incremento do bloco for não for do tipo int ou não foi declarada;
- Se variável declarada dentro de um bloco for estiver sido previamente declarada;
- Se comandos break ou continue foram utilizados fora do escopo de algum laço (for, while, do/while) ou fora do bloco switch;
- Se expressão a ser avaliada em um bloco switch não for do tipo int, char, ou String;
- Se o tipo de expressão de algum bloco case não compatível com o tipo da expressão avaliada pelo bloco switch;
- Se expressão em bloco case não for constante (isto é, se expressão contiver variável ou chamada de método);
- Se algum bloco case ou default de um bloco switch não for finalizado pelo comando break;
- Se a variável em um comando de incremento(++) ou decremento(−) não foi declarada, inicializada ou se não é do tipo int;
- Se caso uma expressão for binária, os tipos das expressões forem incompatíveis com os tipos exigidos para a operação binária. Para isso temos a relação de tipos compatíveis:

Soma de expressão de tipo int com expressão de tipo int possui tipo int;

Soma de expressão de tipo int com expressão de tipo double possui tipo double;

Soma de expressão de tipo double com expressão de tipo int possui tipo double;

Soma de expressão de tipo double com expressão de tipo double possui tipo double;

Soma de expressão de tipo String sempre possui tipo String;

Subtração de expressão de tipo int com expressão de tipo int possui tipo int;

Subtração de expressão de tipo int com expressão de tipo double possui tipo double;

Subtração de expressão de tipo double com expressão de tipo int possui tipo double;

Subtração de expressão de tipo double com expressão de tipo double possui tipo double;

Multiplicação de expressão de tipo int com expressão de tipo int possui tipo int;

Multiplicação de expressão de tipo int com expressão de tipo double possui tipo double;

Multiplicação de expressão de tipo double com expressão de tipo int possui tipo double;

Multiplicação de expressão de tipo double com expressão de tipo double possui tipo double;

Divisão de expressão de tipo int com expressão de tipo int possui tipo int;

Divisão de expressão de tipo int com expressão de tipo double possui tipo double;

Divisão de expressão de tipo double com expressão de tipo int possui tipo double;

Divisão de expressão de tipo double com expressão de tipo double possui tipo double;

Módulo de expressão de tipo int com expressão de tipo int possui tipo int;

Operações relacionais Maior, Menor, MaiorIgual e MenorIgual apenas são validadas quando as duas expressões são ou ambas de tipo int ou ambas de tipo double;

Operações relacionais Igual e Diferente apenas são válidas quando as duas expressões são ou ambas de tipo int ou ambas de tipo double ou ambas de tipo char;

Operações lógicas E e Ou apenas são válidas quando as duas expressões são ambas de tipo boolean.

- Se caso uma expressão for unária, o tipo da expressão for incompatível com o tipo exigido para a operação unária. Para isso temos a relação de tipos compatíveis:

Positivo e Negativo são apenas validos para expressão tipo int ou tipo double;

Incremento e Decremento são apenas validos para expressão tipo int;

Operação lógica Não é valida apenas para expressão tipo boolean.

Para efetuar a manipulação e controle da etapa de análise semântica, será utilizada uma tabela de ambiente. Trata de uma variavel do tipo Hashtbl.t onde, dado um valor para chave, esta tabela retorna último conteúdo armazenado por esta variável. A tabela de ambiente está organizada da seguinte maneira:

- A tabela de ambiente possui como chave o nome de uma classe. Dado o nome da classe, esta tabela retorna uma tabela de variáveis globais e uma tabela de métodos com informações dos métodos pertencentes a aquela classe;
- A tabela de métodos possui como chave o nome do método e retorna:

tabela de variáveis locais do método;

tipo do retorno;

lista de parâmetros contendo informações de tipo e nome de cada parâmetro;

quantidade de parâmetros.

- Ambas as tabelas de variáveis globais de uma classe e as tabelas de variáveis locais de um método possuem como chave o nome da variável e retorna:

tipo da variável;

nome da variável em LLVM (endereço);

A ordem de laço for caso esta variável seja declarada dentro de um bloco for. Isto informa qual o for interno esta variável foi declarada, assim prevenindo esta variável existir fora do escopo do laço for. Possui valor 0 caso não foi declarada dentro do

laço for e valor n caso foi declarada no n-ésimo for interno;

Informação se a variável foi inicializada;

nome da classe e do método que a inicializou

Além desta tabela de ambiente, temos uma variável de controle que é um registro de diversas variáveis de controle, as quais são:

- `expbin_consulta`:

Tabela utilizada para relação entre operações binárias e compatibilidade entre seus tipos.

- `expun_consulta`:

Tabela utilizada para relação entre operações unária e compatibilidade entre seus tipos.

- `tblVarGlobaisInic`:

Tabela utilizada armazenar nome de variáveis globais de alguma classe inicializadas dentro do método. Desta forma é possível identificar quais variáveis globais foram inicializadas dentro de qual método.

- `scannerImportado`:

Booleano que afirma se a classe `java.util.Scanner` foi importada.

- `classeMain`:

Nome da classe que contém o método `main`. Caso o método `main` não foi declarado, esta string é vazia.

- `classeAtual`:

Nome da classe que está atualmente sendo analisada.

- `metodoAtual`:

Nome do método que está atualmente sendo analisado.

- tipoSwitch:

Pilha para armazenar os tipos das expressões utilizadas em um bloco switch. Tal informação é útil para testar compatibilidade entre o tipo da expressão em um bloco switch com a expressão em um bloco case.

- emSwitchCaseDefault:

Se está analisando instruções dentro do bloco case ou bloco default.

- caseBreak:

Se comando break foi o mais recente a ser declarado

- emLoopOuSwitch:

Se está analisando instruções dentro de blocos de laço ou switch. É utilizado para a análise de instruções break e continue.

- comandoReturn:

Se o comando return foi o ultimo comando a ser chamado.

- ordemLoopForAtual:

Armazena valor n do n-ésimo for interno a ser analisado. Esta variável é utilizada para controlar a existência de uma variável declarada no n-ésimo for interno.

A análise semântica, caso o programa em MiniJava for válido, retornará uma tupla contendo a tabela de ambiente e o registro com variáveis de controle.

## 7.6.2 Código - Analisador Semântico

```
(***** IMPORTACOES *****)

open ArvSint;;
open Printf;;

(***** VARIAVEIS DO ANALISADOR SEMANTICO *****)

(***** tipos semantico *****)
```

```

type tipo = TipoInt | TipoBoolean | TipoChar | TipoString | TipoFloat |
          TipoDouble | TipoVoid | TipoScanner

type valor_tipo = ValorInt of int
| ValorBoolean of bool
| ValorChar of char
| ValorString of string
| ValorFloat of float
| ValorDouble of float

type info_classe = { tblMetodos: (string, info_metodo) Hashtbl.t;
                    tblVarClasse: (string, info_var) Hashtbl.t
}

and info_metodo = { tblVarMetodos: (string, info_var) Hashtbl.t;
                  tipoRetorno: tipo;
                  listaParametros: info_param list;
                  paramCount: int;
                  valorRetorno: valor_tipo option
}

and info_var = { varTipo : tipo;
                varEndereco: string;
                ordemLoopFor: int;
                mutable varInicializada: bool;
                mutable metodoInicializador: string;
                mutable classeInicializadora: string;
                mutable valorVar: valor_tipo option
}

and info_param = { paramNome: string;
                  paramTipo: tipo
}

(** chave da tabela de consulta de expressoes binarias **)
and expbin_chave = { operadorBin: ArvSint.operador;
                   tipoexp1: tipo;
                   tipoexp2: tipo }

(** chave da tabela de consulta de expressoes unarias **)
and expun_chave = { operadorUn: ArvSint.operador;
                  tipoexp: tipo }

and vars_controle = { expbin_consulta: (expbin_chave, tipo) Hashtbl.t;
                     expun_consulta: (expun_chave, tipo) Hashtbl.t;
                     tblVarGlobaisInic: ((string * string), string) Hashtbl.t;
                     mutable scannerImportado: bool;
                     mutable classeMain: string;
                     mutable classeAtual: string;
                     mutable metodoAtual: string;

```



```

mutable tipoSwitch: tipo list;
mutable emSwitchCaseDefault: bool;
mutable caseBreak: bool;
mutable emLoopOuSwitch: bool;
mutable comandoReturn: bool;
mutable ordemLoopForAtual: int
}

(***** FUNCOES PARA PREENCHER TABELAS DE CONSULTA *****)

(*****
Preenche tabela de consulta de expressoes binarias
*****)

let preenche_expbin_consulta controle =
(** operadores aritmeticos **)
(* Soma *)
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoInt;
    tipoexp2=TipoInt } TipoInt ;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoDouble;
    tipoexp2=TipoDouble} TipoDouble;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoDouble;
    tipoexp2=TipoInt } TipoDouble;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoInt;
    tipoexp2=TipoDouble} TipoDouble;

Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoString;
    tipoexp2=TipoInt } TipoString;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoString;
    tipoexp2=TipoBoolean} TipoString;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoString;
    tipoexp2=TipoChar } TipoString;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoString;
    tipoexp2=TipoString } TipoString;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoString;
    tipoexp2=TipoFloat } TipoString;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoString;
    tipoexp2=TipoDouble } TipoString;

Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoInt;
    tipoexp2=TipoString } TipoString;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoBoolean
    ; tipoexp2=TipoString } TipoString;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoChar;
    tipoexp2=TipoString } TipoString;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoString;
    tipoexp2=TipoString } TipoString;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoFloat;
    tipoexp2=TipoString } TipoString;
Hashtbl.add controle.expbin_consulta {operadorBin=Soma; tipoexpl=TipoDouble;
    tipoexp2=TipoString } TipoString;

```

```

(* Sub *)
Hashtbl.add controle.expbin_consulta {operadorBin=Sub; tipoexpl=TipoInt;
    tipoexp2=TipoInt } TipoInt ;
Hashtbl.add controle.expbin_consulta {operadorBin=Sub; tipoexpl=TipoDouble;
    tipoexp2=TipoDouble} TipoDouble;
Hashtbl.add controle.expbin_consulta {operadorBin=Sub; tipoexpl=TipoDouble;
    tipoexp2=TipoInt } TipoDouble;
Hashtbl.add controle.expbin_consulta {operadorBin=Sub; tipoexpl=TipoInt;
    tipoexp2=TipoDouble} TipoDouble;
(* Mul *)
Hashtbl.add controle.expbin_consulta {operadorBin=Mul; tipoexpl=TipoInt;
    tipoexp2=TipoInt } TipoInt ;
Hashtbl.add controle.expbin_consulta {operadorBin=Mul; tipoexpl=TipoDouble;
    tipoexp2=TipoDouble} TipoDouble;
Hashtbl.add controle.expbin_consulta {operadorBin=Mul; tipoexpl=TipoDouble;
    tipoexp2=TipoInt } TipoDouble;
Hashtbl.add controle.expbin_consulta {operadorBin=Mul; tipoexpl=TipoInt;
    tipoexp2=TipoDouble} TipoDouble;
(* Div *)
Hashtbl.add controle.expbin_consulta {operadorBin=Div; tipoexpl=TipoInt;
    tipoexp2=TipoInt } TipoInt ;
Hashtbl.add controle.expbin_consulta {operadorBin=Div; tipoexpl=TipoDouble;
    tipoexp2=TipoDouble} TipoDouble;
Hashtbl.add controle.expbin_consulta {operadorBin=Div; tipoexpl=TipoDouble;
    tipoexp2=TipoInt } TipoDouble;
Hashtbl.add controle.expbin_consulta {operadorBin=Div; tipoexpl=TipoInt;
    tipoexp2=TipoDouble} TipoDouble;
(* Mod *)
Hashtbl.add controle.expbin_consulta {operadorBin=Mod; tipoexpl=TipoInt;
    tipoexp2=TipoInt } TipoInt ;
(** operadores de comparacao **)
(* Maior *)
Hashtbl.add controle.expbin_consulta {operadorBin=Maior; tipoexpl=TipoInt;
    tipoexp2=TipoInt } TipoBoolean;
Hashtbl.add controle.expbin_consulta {operadorBin=Maior; tipoexpl=TipoDouble
    ; tipoexp2=TipoDouble} TipoBoolean;
(* Menor *)
Hashtbl.add controle.expbin_consulta {operadorBin=Menor; tipoexpl=TipoInt;
    tipoexp2=TipoInt } TipoBoolean;
Hashtbl.add controle.expbin_consulta {operadorBin=Menor; tipoexpl=TipoDouble
    ; tipoexp2=TipoDouble} TipoBoolean;
(* MaiorIgual *)
Hashtbl.add controle.expbin_consulta {operadorBin=MaiorIgual; tipoexpl=
    TipoInt; tipoexp2=TipoInt } TipoBoolean;
Hashtbl.add controle.expbin_consulta {operadorBin=MaiorIgual; tipoexpl=
    TipoDouble; tipoexp2=TipoDouble} TipoBoolean;
(* MenorIgual *)
Hashtbl.add controle.expbin_consulta {operadorBin=MenorIgual; tipoexpl=
    TipoInt; tipoexp2=TipoInt } TipoBoolean;

```

```

Hashtbl.add controle.expbin_consulta {operadorBin=MenorIgual; tipoexp1=
    TipoDouble; tipoexp2=TipoDouble} TipoBoolean;
(* Igual *)
Hashtbl.add controle.expbin_consulta {operadorBin=Igual; tipoexp1=TipoInt;
    tipoexp2=TipoInt } TipoBoolean;
Hashtbl.add controle.expbin_consulta {operadorBin=Igual; tipoexp1=TipoDouble
    ; tipoexp2=TipoDouble} TipoBoolean;
Hashtbl.add controle.expbin_consulta {operadorBin=Igual; tipoexp1=TipoChar;
    tipoexp2=TipoChar } TipoBoolean;
(* Diferente *)
Hashtbl.add controle.expbin_consulta {operadorBin=Diferente; tipoexp1=
    TipoInt; tipoexp2=TipoInt } TipoBoolean;
Hashtbl.add controle.expbin_consulta {operadorBin=Diferente; tipoexp1=
    TipoDouble; tipoexp2=TipoDouble} TipoBoolean;
Hashtbl.add controle.expbin_consulta {operadorBin=Diferente; tipoexp1=
    TipoChar; tipoexp2=TipoChar } TipoBoolean;
(** operadores logicos **)
(* E *)
Hashtbl.add controle.expbin_consulta {operadorBin=E; tipoexp1=TipoBoolean;
    tipoexp2=TipoBoolean } TipoBoolean;
(* Ou *)
Hashtbl.add controle.expbin_consulta {operadorBin=Ou; tipoexp1=TipoBoolean;
    tipoexp2=TipoBoolean } TipoBoolean

(*****
Preenche tabela de consulta de expressoes unarias
*****)

let preenche_expun_consulta controle =
(**operadores aritmeticos**)
(* Posit *)
Hashtbl.add controle.expun_consulta {operadorUn=Posit; tipoexp=TipoInt }
    TipoInt;
Hashtbl.add controle.expun_consulta {operadorUn=Posit; tipoexp=TipoDouble }
    TipoDouble;
(* Negat *)
Hashtbl.add controle.expun_consulta {operadorUn=Negat; tipoexp=TipoInt }
    TipoInt;
Hashtbl.add controle.expun_consulta {operadorUn=Negat; tipoexp=TipoDouble }
    TipoDouble;
(* Incr *)
Hashtbl.add controle.expun_consulta {operadorUn=Incr; tipoexp=TipoInt }
    TipoInt;
(* Decr *)
Hashtbl.add controle.expun_consulta {operadorUn=Decr; tipoexp=TipoInt }
    TipoInt;
(**operadores logicos**)
(* Nao *)
Hashtbl.add controle.expun_consulta {operadorUn=Nao; tipoexp=TipoBoolean }
    TipoBoolean

```

```

(***** FUNCOES UTEIS PARA O ANALISADOR *****)

(* ***** funcoes para alertar warning ***** *)

let exhibe_warning msg pos =
let nlin = pos.Posicao.lin_inicial
and ncol = pos.Posicao.col_inicial in
let msglcl = sprintf "Warning:\nPosicao: linha %d, coluna %d" nlin ncol
in
print_endline msglcl;
print_endline msg

(* ***** funcoes para alertar erro ***** *)
let exhibe_erro msg pos =
let nlin = pos.Posicao.lin_inicial
and ncol = pos.Posicao.col_inicial in
let msglcl = sprintf "Posicao: linha %d, coluna %d" nlin ncol in
print_endline msglcl;
print_endline msg;
failwith "Erro semantico"

(* ***** funcoes para conversao de tipos ***** *)

(* Converte um Semantico.tipo em um ArvSint.tipo *)
let arvSintTipo_of_tipo tipo =
match tipo with
TipoInt      -> TInt
| TipoBoolean -> TBoolean
| TipoChar    -> TChar
| TipoString  -> TString
| TipoFloat   -> TFloat
| TipoDouble  -> TDouble
| TipoVoid    -> TVoid
| _ -> failwith "Tipo Scanner"

(* Converte um Semantico.tipo em um ArvSint.tipo *)
let tipo_of_arvSintTipo tipo =
match tipo with
TInt      -> TipoInt
| TBoolean -> TipoBoolean
| TChar    -> TipoChar
| TString  -> TipoString
| TFloat   -> TipoFloat
| TDouble  -> TipoDouble
| TVoid    -> TipoVoid

(* Converte um ArvSint.tipo em uma string *)
let string_of_arvSintTipo tipo =
match tipo with
TInt      -> "int"

```

```

| TBoolean -> "boolean"
| TChar    -> "char"
| TString  -> "String"
| TFloat   -> "float"
| TDouble  -> "double"
| TVoid    -> "void"

(* Converte um Semantico.tipo em uma string *)
let string_of_tipo tipo =
match tipo with
TipoInt      -> "int"
| TipoBoolean -> "boolean"
| TipoChar    -> "char"
| TipoString  -> "String"
| TipoFloat   -> "float"
| TipoDouble  -> "double"
| TipoVoid    -> "void"
| TipoScanner -> "Scanner"

(* Converte um ArvSint.tipoLeitura para tipo correspondente *)
let tipo_of_tipoLeitura tipoLeitura =
match tipoLeitura with
ReadBool    -> TipoBoolean
| ReadDouble -> TipoDouble
| ReadFloat  -> TipoFloat
| ReadInt    -> TipoInt
| ReadByte   -> TipoChar
| ReadString -> TipoString

(* Converte um ArvSint.operador em uma string *)
let string_of_operador operador =
match operador with
Soma        -> "+"
| Sub        -> "-"
| Mul        -> "*"
| Div        -> "/"
| Mod        -> "%"
| Incr       -> "++"
| Decr       -> "--"
| Nao        -> "!"
| E          -> "&&"
| Ou         -> "||"
| Menor      -> "<"
| MenorIgual -> "<="
| Igual      -> "=="
| Diferente  -> "!="
| Maior      -> ">"
| MaiorIgual -> ">="
| Posit      -> "+"
| Negat      -> "-"

```

```

(* converter uma lista de ArvSint.parametro para uma lista de Semantico.
   info_param *)
let rec converter_lista_parametros (paramlist:ArvSint.parametro list) =
match paramlist with
[]      -> []
| ((t:ArvSint.tipo), s, _) :: ps ->
let e = { paramNome = s;
  paramTipo = (tipo_of_arvSintTipo t)
} in
([e] @ (converter_lista_parametros ps))

(***** FUNCOES PARA VERIFICACAO NA TABELA DE AMBIENTE *****)

(* metodo para obter parametro da lista de parametro *)
let rec obter_parametro controle varnome paramlist =
match paramlist with
[]      -> failwith "Parametro nao existente"
| p::ps ->
begin
if(p.paramNome = varnome) then p
else
obter_parametro controle varnome ps
end

(* metodo para obter parametro da um metodo *)
let obter_parametro_metodo ambiente controle nomeclasse nomemetodo varnome =
let entradaClasse = Hashtbl.find ambiente nomeclasse in
let entradaMetodo = Hashtbl.find entradaClasse.tblMetodos nomemetodo in
obter_parametro controle varnome entradaMetodo.listaParametros

(* retorna o tipo de retorno do metodo atual *)
let obter_tipo_retorno_metodo ambiente controle nomeclasse nomemetodo =
let entradaClasse = Hashtbl.find ambiente nomeclasse in
let entradaMetodo = Hashtbl.find entradaClasse.tblMetodos nomemetodo in
entradaMetodo.tipoRetorno

(* obtem 'info_var' de uma variavel declarada local *)
let obter_variavel_local ambiente controle nomeclasse nomemetodo varnome =
let entradaClasse = Hashtbl.find ambiente nomeclasse in
let entradaMetodo = Hashtbl.find entradaClasse.tblMetodos nomemetodo in
Hashtbl.find entradaMetodo.tblVarMetodos varnome

(* obtem 'info_var' de uma variavel declarada global *)
let obter_variavel_global ambiente controle nomeclasse varnome =
let entradaClasse = Hashtbl.find ambiente nomeclasse in
Hashtbl.find entradaClasse.tblVarClasse varnome

(* verifica se existe a classe na tabela de ambiente *)
let existe_classe ambiente controle nomeclasse =
try

```

```

ignore(Hashtbl.find ambiente nomeclasse);
true
with Not_found ->
false

(* verifica se existe metodo de alguma classe na tabela de ambiente *)
let existe_metodo ambiente controle nomeclasse nomemetodo =
try
let entradaClasse = Hashtbl.find ambiente nomeclasse in
ignore(Hashtbl.find entradaClasse.tblMetodos nomemetodo);
true
with Not_found ->
false

(* verifica se existe a variavel no escopo da classe *)
let existe_variavel_global ambiente controle nomeclasse varnome =
try
let entradaClasse = Hashtbl.find ambiente nomeclasse in
ignore(Hashtbl.find entradaClasse.tblVarClasse varnome);
true
with Not_found ->
false

(* verifica se existe a variavel no escopo da classe *)
let existe_variavel_local ambiente controle nomeclasse nomemetodo varnome =
try
let entradaClasse = Hashtbl.find ambiente nomeclasse in
let entradaMetodo = Hashtbl.find entradaClasse.tblMetodos nomemetodo in
let entradaVar = Hashtbl.find entradaMetodo.tblVarMetodos varnome in
(* se variavel declarada em um for estiver fora de seu escopo, ela nao
existe *)
if(entradaVar.ordemLoopFor > controle.ordemLoopForAtual) then
false
else
true
with Not_found ->
false

(* metodo auxiliar para analisar existencia de parametros *)
let rec existe_parametro_100 controle varnome paramlist existe =
match paramlist with
[] -> existe
| p::ps ->
begin
if(existe == false) then
if(p.paramNome = varnome) then
existe_parametro_100 controle varnome ps true
else
existe_parametro_100 controle varnome ps false
else
true

```

```

end

(* verifica existencia de parametros com tal nome pela lista de parametros*)
let existe_parametro controle varnome paramlist =
existe_parametro_100 controle varnome paramlist false

(* verifica existencia de parametros com tal nome de um metodo de uma classe
*)
let existe_parametro_metodo ambiente controle nomeclasse nomemetodo varnome
=
try
let entradaClasse = Hashtbl.find ambiente nomeclasse in
let entradaMetodo = Hashtbl.find entradaClasse.tblMetodos nomemetodo in
let value = existe_parametro controle varnome entradaMetodo.listaParametros
in
value
with Not_found ->
failwith (sprintf "metodo \'%s\' nao existe" nomemetodo)

(* verifica se a expressao do tipo ExpNone *)
let is_expressao_none controle expressao =
match expressao with
ExpNone -> true
|_ -> false

(* verifica se a expressao constante *)
let rec is_expressao_constante controle expressao =
match expressao with
(ExpMetodo _) -> false
| (ExpVar _) -> false
| (ExpNone) -> false
| (ExpUn (_,e)) -> (is_expressao_constante controle e)
| (ExpBin (_,e1,e2)) -> (is_expressao_constante controle e1) && (
is_expressao_constante controle e2)
| _ -> true

(* verifica se a expressao constante sem operacoes*)
let is_expressao_constante_simples controle expressao =
match expressao with
(ExpMetodo _) -> false
| (ExpVar _) -> false
| (ExpNone) -> false
| (ExpUn (_,_)) -> false
| (ExpBin (_,_,_)) -> false
| _ -> true

(***** FUNCOES PARA MANIPULACAO DO AMBIENTE *****)

(* insere uma classe na tabela de ambiente *)
let insere_classe ambiente controle nomeclasse =
let entrada = { tblMetodos = Hashtbl.create 20;

```



```

    tblVarClasse = Hashtbl.create 20 } in
Hashtbl.add ambiente nomeclasse entrada

(* insere um metodo na tabela de ambiente com base no ponteiro 'classeAtual'
   *)
let insere_metodo ambiente controle nomemetodo tipo (paramlist: (ArvSint.
    parametro list)) =
let entradaClasse = Hashtbl.find ambiente controle.classeAtual in
let lista = (converter_lista_parametros paramlist) in
let entrada = { tblVarMetodos = Hashtbl.create 20;
    tipoRetorno = tipo;
    listaParametros = lista;
    paramCount = (List.length lista);
    valorRetorno = None
}
in
Hashtbl.add entradaClasse.tblMetodos nomemetodo entrada

(* insere variavel global na tabela de ambiente com base no ponteiro '
   classeAtual' *)
let insere_variavel_global ambiente controle varnome tipo expressao =
let entradaClasse = Hashtbl.find ambiente controle.classeAtual in
let inicializada = ((is_expressao_none controle expressao)==false) in
let entrada = { varTipo = tipo;
    (* formato: classe_null_variavel *)
    varEndereco = controle.classeAtual ^ "." ^ "null" ^ "." ^ varnome;
    varInicializada = inicializada;
    ordemLoopFor = 0;
    metodoInicializador = "";
    classeInicializadora = controle.classeAtual;
    valorVar = None }
in
Hashtbl.add entradaClasse.tblVarClasse varnome entrada

(* insere variavel local na tabela de ambiente com base no ponteiro '
   classeAtual'
   e no ponteiro 'metodoAtual'
   *)
let insere_variavel_local ambiente controle varnome tipo expressao ordemFor
    =
let entradaClasse = Hashtbl.find ambiente controle.classeAtual in
let entradaMetodo = Hashtbl.find entradaClasse.tblMetodos controle.
    metodoAtual in
let inicializada = ((is_expressao_none controle expressao)==false) in
let entrada = { varTipo = tipo;
    (* formato: classe_metodo_variavel *)
    varEndereco = varnome;
    ordemLoopFor = ordemFor;
    varInicializada = inicializada;
    metodoInicializador = controle.metodoAtual;
    classeInicializadora = controle.classeAtual;

```

```

    valorVar = None }
in
Hashtbl.add entradaMetodo.tblVarMetodos varnome entrada

(* ***** inicializa uma variavel ***** *)

let inicializa_variavel controle entradaVar =
entradaVar.varInicializada      <- true;
entradaVar.metodoInicializador  <- controle.metodoAtual;
entradaVar.classeInicializadora <- controle.classeAtual

(***** FUNCOES PARA ANALISE SEMANTICA *****)

(* dada uma expressao, esta funcao analisa a chamada de metodo *)

let analisa_expressao_metodo ambiente controle nomemetodo nomeclasse explist
    pos =
(* se metodo nao existe *)
if((existe_classe ambiente controle nomeclasse) == false)
then
let msg = sprintf "A classe \'%s\' para o metodo \'%s\' na expressao deve
    ser criada previamente"
nomeclasse nomemetodo in
exibe_erro msg pos
else
if((existe_metodo ambiente controle nomeclasse nomemetodo) == false)
then
let msg = sprintf "O metodo \'%s\' na expressao deve ser criado previamente
    na classe \'%s\'"
nomemetodo nomeclasse in
exibe_erro msg pos
else
let entradaClasse = Hashtbl.find ambiente nomeclasse in
let entradaMetodo = Hashtbl.find entradaClasse.tblMetodos nomemetodo in
if((List.length explist) > entradaMetodo.paramCount)
then
let msg = sprintf "Muitos argumentos para o metodo \'%s\' "
nomemetodo in
exibe_erro msg pos
else
if((List.length explist) < entradaMetodo.paramCount)
then
let msg = sprintf "Poucos argumentos para o metodo \'%s\' "
nomemetodo in
exibe_erro msg pos

(* dada uma expressao, esta funcao retorna seu tipo *)

let rec analisa_expressao ambiente controle expressao nomeclasse nomemetodo
    pos =
match expressao with

```

```

(ExpInt      _) -> TipoInt
| (ExpBoolean _) -> TipoBoolean
| (ExpChar    _) -> TipoChar
| (ExpString  _) -> TipoString
| (ExpFloat   _) -> TipoFloat
| (ExpDouble  _) -> TipoDouble
| (ExpMetodo (nomemetodo, nomeclasse, explist, pos)) ->
begin
  analisa_expressao_metodo ambiente controle nomemetodo nomeclasse explist pos
  ;
  let entradaClasse = Hashtbl.find ambiente nomeclasse in
  let entradaMetodo = Hashtbl.find entradaClasse.tblMetodos nomemetodo in
  begin
    (* analisa tipo dos parametros de uma funcao *)
    ignore(List.map2
      (fun param (exp,pos) ->
        let tipoarg  = (analisa_expressao ambiente controle exp controle.
          classeAtual controle.metodoAtual pos) in
        let tipoparam = param.paramTipo in
        if(tipoarg <> tipoparam)
        then
          let msg = sprintf "Argumento do metodo \'%s\'      tipo \'%s\' mas deveria ser
            tipo \'%s\'"
            nomemetodo (string_of_tipo tipoarg) (string_of_tipo tipoparam) in
          exibe_erro msg pos
        )
        entradaMetodo.listaParametros explist);
    entradaMetodo.tipoRetorno
  end
end
| (ExpVar varnome) ->
begin
  if((existe_parametro_metodo ambiente controle nomeclasse nomemetodo varnome)
    == false)
  then
    if((existe_variavel_global ambiente controle nomeclasse varnome) == false)
    then
      if((existe_variavel_local ambiente controle nomeclasse nomemetodo varnome)
        == false)
      then
        let msg = sprintf "A variavel \'%s\' na expressao nao foi declarada"
          varnome in
        exibe_erro msg pos
      else
        let entradaVar = obter_variavel_local ambiente controle nomeclasse
          nomemetodo varnome in
        if((entradaVar.varInicializada) == false)
        then
          let msg = sprintf "A variavel \'%s\' na expressao nao foi inicializada"
            varnome in
          exibe_erro msg pos

```

```

else
entradaVar.varTipo
else
let entradaVar = obter_variavel_global ambiente controle nomeclasse varnome
    in
if((entradaVar.varIniciada) == false)
then
let msg = sprintf "A variavel \'%s\' na expressao nao foi inicializada"
varnome in
exibe_erro msg pos
else
entradaVar.varTipo
else
let param = obter_parametro_metodo ambiente controle nomeclasse nomemetodo
    varnome in
param.paramTipo
end
| (ExpBin (op, exp1, exp2)) ->
begin
let tipo1 = (analisa_expressao ambiente controle exp1 nomeclasse nomemetodo
    pos) in
let tipo2 = (analisa_expressao ambiente controle exp2 nomeclasse nomemetodo
    pos) in
try
Hashtbl.find controle.expbin_consulta { operadorBin = op;
    tipoexp1 = tipo1;
    tipoexp2 = tipo2 }
with Not_found ->
let msg = sprintf "Tipos \'%s\' e \'%s\' invalidos para operacao binaria \'%s\'"
    (string_of_tipo tipo1)
    (string_of_tipo tipo2)
    (string_of_operador op) in
exibe_erro msg pos
end
| (ExpUn (op, exp)) ->
begin
let tipo = (analisa_expressao ambiente controle exp nomeclasse nomemetodo
    pos) in
try
Hashtbl.find controle.expun_consulta { operadorUn=op;
    tipoexp=tipo;}
with Not_found ->
let msg = sprintf "Tipo \'%s\' invalido para operacao unaria \'%s\'"
    (string_of_tipo tipo)
    (string_of_operador op) in
exibe_erro msg pos
end
| ExpNone -> failwith "expressao tipo ExpNone"

(* analisa se os argumentos do metodo sao compatíveis *)

```

```

(* ***** analise de expressoes e variaveis ***** *)

let analisa_expressao_declaracao ambiente controle varnome tipovar expressao
    pos =
if((is_expressao_none controle expressao) == false) then
let tpexp = (analisa_expressao ambiente controle expressao controle.
    classeAtual controle.metodoAtual pos) in
if(tpexp != tipovar) then
let msg = sprintf "Variavel \'%s\' e\' tipo \'%s\' mas expressao e\' tipo
    \'%s\'"
varnome (string_of_tipo tipovar) (string_of_tipo tpexp) in
exibe_erro msg pos

(* ** analisa variavel local de diversas ordens e insere *** *)

let analisa_variavel_local_ordem ambiente controle (varnome, (tipo:ArvSint.
    tipo), expressao, pos) ordem =
let tp = (tipo_of_arvSintTipo tipo) in
if(existe_parametro_metodo ambiente controle controle.classeAtual controle.
    metodoAtual varnome) then
let msg = sprintf "Variavel \'%s\'      parametro no metodo \'%s\'"
varnome controle.metodoAtual in
exibe_erro msg pos
else
begin
if (existe_variavel_global ambiente controle controle.classeAtual varnome)
    then
let msg = sprintf "Variavel \'%s\' ja foi declarada global em classe \'%s\'"
varnome controle.classeAtual in
exibe_erro msg pos
else
begin
if(existe_variavel_local ambiente controle controle.classeAtual controle.
    metodoAtual varnome) then
let msg = sprintf "Variavel \'%s\' ja foi declarada local em classe \'%s\'"
varnome controle.metodoAtual in
exibe_erro msg pos
else
analisa_expressao_declaracao ambiente controle varnome tp expressao pos;
insere_variavel_local ambiente controle varnome tp expressao ordem
end
end

(* ***** analisa instrucao incr decr ***** *)

let analisa_incr_decr ambiente controle varnome pos =
if((existe_variavel_global ambiente controle controle.classeAtual varnome)
    == false)
then

```

```

if((existe_variavel_local ambiente controle controle.classeAtual controle.
    metodoAtual varnome) == false)
then
let msg = sprintf "A variavel \'%s\' na expressao nao foi declarada"
varnome in
exibe_erro msg pos
else
begin
let entradaVar = obter_variavel_local ambiente controle controle.classeAtual
    controle.metodoAtual varnome in
if((entradaVar.varInicializada) == false) then
let msg = sprintf "A variavel \'%s\' deve ser inicializada"
varnome in
exibe_erro msg pos
else
if(entradaVar.varTipo <> TipoInt) then
let msg = sprintf "A variavel \'%s\' deve ser \'int\'"
varnome in
exibe_erro msg pos
end
else
begin
let entradaVar = obter_variavel_global ambiente controle controle.
    classeAtual varnome in
if((entradaVar.varInicializada) == false) then
let msg = sprintf "A variavel \'%s\' deve ser inicializada"
varnome in
exibe_erro msg pos
else
if(entradaVar.varTipo <> TipoInt) then
let msg = sprintf "A variavel \'%s\' deve ser \'int\'"
varnome in
exibe_erro msg pos
end

(* ***** analisa instrucao de atribuicao ***** *)

let analisa_atrib ambiente controle (varnome,expressao, pos) =
if((existe_variavel_global ambiente controle controle.classeAtual varnome)
    == false)
then
if((existe_variavel_local ambiente controle controle.classeAtual controle.
    metodoAtual varnome) == false)
then
let msg = sprintf "A variavel \'%s\' na expressao nao foi declarada"
varnome in
exibe_erro msg pos
else
begin
let entradaVar = obter_variavel_local ambiente controle controle.classeAtual
    controle.metodoAtual varnome in

```

```

begin
analisa_expressao_declaracao ambiente controle varnome entradaVar.varTipo
    expressao pos;
inicializa_variavel controle entradaVar;
end
end
else
begin
let entradaVar = obter_variavel_global ambiente controle controle.
    classeAtual varnome in
begin
analisa_expressao_declaracao ambiente controle varnome entradaVar.varTipo
    expressao pos;
inicializa_variavel controle entradaVar;
end
end

(** ANALISE DE BLOCOS DE INSTRUCAO **)

(* ***** analisa expressao condicional ***** *)

let analisa_condicao ambiente controle nomebloco condicao pos =
if((analisa_expressao ambiente controle condicao controle.classeAtual
    controle.metodoAtual pos) <> TipoBoolean)
then
let msg = sprintf "Express o da condi  o do bloco \'%s\' deve ser tipo \'
    boolean\'"
    nomebloco
in
exibe_erro msg pos

(* ***** analisa for incr ***** *)

let analisa_for_incr ambiente controle incremento =
match incremento with
(ExpFor (varnome,expressao,pos)) ->
analisa_atrib ambiente controle (varnome,expressao,pos)
| (IncrFor (varnome,pos))->
analisa_incr_decr ambiente controle varnome pos
| (DecrFor (varnome,pos))->
analisa_incr_decr ambiente controle varnome pos

(* ***** analisa for condicao ***** *)

let analisa_for_cond ambiente controle (condicao, pos) =
analisa_condicao ambiente controle "for" condicao pos

(* ***** analisa for atrib ***** *)

let analisa_for_atrib ambiente controle atribuicao =
match atribuicao with

```

```

(AtribFor (varnome,expressao,pos)) ->
if((existe_variavel_global ambiente controle controle.classeAtual varnome)
== false)
then
if((existe_variavel_local ambiente controle controle.classeAtual controle.
metodoAtual varnome) == false)
then
let msg = sprintf "A variavel \'%s\' na expressao nao foi declarada"
varnome in
exibe_erro msg pos
else
begin
let entradaVar = obter_variavel_local ambiente controle controle.classeAtual
controle.metodoAtual varnome in
begin
analisa_expressao_declaracao ambiente controle varnome entradaVar.varTipo
expressao pos;
inicializa_variavel controle entradaVar;
if(entradaVar.varTipo <> TipoInt)
then
let msg = sprintf "Variavel em \'for\' deve ser tipo \'int\'" in
exibe_erro msg pos
end
end

else
begin
let entradaVar = obter_variavel_global ambiente controle controle.
classeAtual varnome in
begin
analisa_expressao_declaracao ambiente controle varnome entradaVar.varTipo
expressao pos;
inicializa_variavel controle entradaVar;
if(entradaVar.varTipo <> TipoInt)
then
let msg = sprintf "Variavel em \'for\' deve ser tipo \'int\'" in
exibe_erro msg pos
end
end

| (AtribVarFor (varnome,tipo,expressao,pos)) ->
if((tipo_of_arvSintTipo tipo) <> TipoInt)
then
let msg = sprintf "Variavel em \'for\' deve ser tipo \'int\'" in
exibe_erro msg pos
else
analisa_variavel_local_ordem ambiente controle (varnome, (tipo:ArvSint.tipo)
, expressao, pos) controle.ordemLoopForAtual

(* ***** analisa bloco for ***** *)

```



```

let analisa_for ambiente controle (atribuicao,expressao,incremento) =
analisa_for_atrib ambiente controle atribuicao;
analisa_for_cond ambiente controle expressao;
analisa_for_incr ambiente controle incremento

(* ***** analisa expressao switch : case ***** *)

let analisa_case_switch ambiente controle condicao pos =
let tipo = (analisa_expressao ambiente controle condicao controle.
    classeAtual controle.metodoAtual pos) in
if(tipo != (List.hd controle.tipoSwitch))
then
let msg = sprintf "Express o em \'case\' requer tipo \'%s\', mas encontrou
    tipo \'%s\'"
(string_of_tipo (List.hd controle.tipoSwitch)) (string_of_tipo tipo)
in
exibe_erro msg pos
else
if((is_expressao_constante controle condicao) == false)
then
let msg = sprintf "Express o em \'case\' deve ser constante (nao possuir
    variaveis ou metodos)"
in
exibe_erro msg pos

(* ***** analisa expressao switch ***** *)

let analisa_condicao_switch ambiente controle condicao pos =
let tipo = (analisa_expressao ambiente controle condicao controle.
    classeAtual controle.metodoAtual pos) in
if((tipo == TipoBoolean) || (tipo == TipoFloat) || (tipo == TipoDouble))
then
let msg = sprintf "Express o da condi o do bloco \'switch\' n o pode
    ser \'%s\'"
(string_of_tipo tipo)
in
exibe_erro msg pos
else
(* empilha tipo na pilha \'tipoSwitch\' *)
controle.tipoSwitch <- [tipo] @ controle.tipoSwitch

(** ANALISE DE COMANDOS DE INSTRUCAO **)

let analisa_chamada_metodo ambiente controle nomemetodo nomeclasse explist
pos =
ignore(analisa_expressao ambiente controle (ExpMetodo (nomemetodo,
    nomeclasse, explist, pos)) nomemetodo nomeclasse pos);
let entradaClasse = Hashtbl.find ambiente nomeclasse in
let entradaMetodo = Hashtbl.find entradaClasse.tblMetodos nomemetodo in
if(entradaMetodo.tipoRetorno <> TipoVoid) then
let msg = sprintf "Metodo \'%s\' chamado sem atribuicao retorna \'%s\'"

```

```

nomemetodo (string_of_tipo entradaMetodo.tipoRetorno)
in
(*exibe_erro msg pos*)
exibe_warning msg pos

(* ***** analisa instrucao read ***** *)

let analisa_read ambiente controle vardest varscanner tipoLeitura posdest
    posscanner =
(* se nao existe variavel scanner *)
if((existe_variavel_local ambiente controle controle.classeAtual controle.
    metodoAtual varscanner) == false)
then
let msg = sprintf "A variavel \'%s\' de tipo \'Scanner\' nao foi declarada"
varscanner in
exibe_erro msg posscanner
else
(* se nao existe variavel de destino *)
if((existe_variavel_global ambiente controle controle.classeAtual vardest)
    == false)
then
if((existe_variavel_local ambiente controle controle.classeAtual controle.
    metodoAtual vardest) == false)
then
let msg = sprintf "A variavel \'%s\' na expressao nao foi declarada"
vardest in
exibe_erro msg posdest
else
(* se tipos de retorno sao diferentes *)
begin
let entradaVar = obter_variavel_local ambiente controle controle.classeAtual
    controle.metodoAtual vardest in
begin
let tipoVar      = entradaVar.varTipo in
let tipoScanner = (tipo_of_tipoLeitura tipoLeitura) in
if(tipoVar <> tipoScanner)
then
let msg = sprintf "A variavel \'%s\'      tipo \'%s\' mas a leitura retorna
    valor de tipo \'%s\'"
vardest (string_of_tipo tipoVar) (string_of_tipo tipoScanner) in
exibe_erro msg posdest
else
if(entradaVar.varInicializada == false) then
inicializa_variavel controle entradaVar
end
end
else
begin
let entradaVar = obter_variavel_global ambiente controle controle.
    classeAtual vardest in
begin

```

```

let tipoVar      = entradaVar.varTipo in
let tipoScanner = (tipo_of_tipoLeitura tipoLeitura) in
if(tipoVar <> tipoScanner)
then
let msg = sprintf "A variavel \'%s\'      tipo \'%s\' mas a leitura retorna
    valor de tipo \'%s\'"
vardest (string_of_tipo tipoVar) (string_of_tipo tipoScanner) in
exibe_erro msg posdest
else
if(entradaVar.varInicializada == false) then
inicializa_variavel controle entradaVar
end
end

(* ***** analisa instrucao scanner ***** *)

let analisa_scanner ambiente controle varnome pos =
if(controle.scannerImportado == false) then
let msg = sprintf " Classe \'java.util.Scanner\' n o foi importada "in
exibe_erro msg pos
else
if((existe_variavel_local ambiente controle controle.classeAtual controle.
    metodoAtual varnome))
then
let msg = sprintf "A variavel \'%s\' ja foi declarada"
varnome in
exibe_erro msg pos
else
insere_variavel_local ambiente controle varnome TipoScanner ArvSint.ExpNone
    0

(* ***** analisa instrucao return ***** *)

let analisa_return ambiente controle expressao pos =
let tipo_retorno = obter_tipo_retorno_metodo ambiente controle controle.
    classeAtual controle.metodoAtual in
(* se no codigo existe apenas 'return;' e o tipo nao for void *)
if(is_expressao_none controle expressao) then
begin
if(tipo_retorno <> TipoVoid) then
let msg = sprintf "O comando \'return\' deve retornar uma expressao" in
exibe_erro msg pos
end
else
let tipo = (analisa_expressao ambiente controle expressao controle.
    classeAtual controle.metodoAtual pos) in
if(tipo <> tipo_retorno) then
let msg = sprintf "O comando \'return\' possui expressao tipo \'%s\' mas o
    metodo \'%s\' deve retornar \'%s\'"
(string_of_tipo tipo) controle.metodoAtual (string_of_tipo tipo_retorno)
in

```

```

exibe_erro msg pos

(* ***** analisa instrucao apos comando break ***** *)

let analisa_instrucao_apos_break controle pos =
(* se a instrucao esta dentro de um case e est  apos comando break *)
if(controle.emSwitchCaseDefault && controle.caseBreak) then
let msg = sprintf "Instru o ap s instrucao \'break\'" in
exibe_erro msg pos

(* ***** analisa instrucao break ou continue ***** *)
let analisa_instrucao_break_continue controle pos =
if(not controle.emLoopOuSwitch) then
let msg = sprintf "Instru o \'break\' ou \'continue\' fora de loop ou
switch" in
exibe_erro msg pos

(** ANALISE DE INSTRUCOES **)

(* ***** analise de lista de instrucoes ***** *)

let rec analisa_instrucoes ambiente controle instrucaolist =
match instrucaolist with
[] -> ()
| i :: ils ->
match i with
(BlocoIf (condicao, instrucoes, pos)) ->
analisa_instrucao_apos_break controle pos;
analisa_condicao ambiente controle "if" condicao pos;
analisa_instrucoes ambiente controle instrucoes;
analisa_instrucoes ambiente controle ils
| (BlocoIfElse (condicao, instrucoesIf, instrucoesElse, pos)) ->
analisa_instrucao_apos_break controle pos;
analisa_condicao ambiente controle "if" condicao pos;
analisa_instrucoes ambiente controle instrucoesIf;
analisa_instrucoes ambiente controle instrucoesElse;
analisa_instrucoes ambiente controle ils
| (BlocoFor (param, instlist,pos)) ->
analisa_instrucao_apos_break controle pos;
(* incrementa ordem do loop for *)
controle.ordemLoopForAtual <- controle.ordemLoopForAtual + 1;
analisa_for ambiente controle param;
controle.emLoopOuSwitch <- true;
analisa_instrucoes ambiente controle instlist;
(* decrementa ordem do loop for *)
controle.ordemLoopForAtual <- controle.ordemLoopForAtual - 1;
controle.emLoopOuSwitch <- false;
analisa_instrucoes ambiente controle ils
| (BlocoWhile (condicao,instrucoes,pos)) ->
analisa_instrucao_apos_break controle pos;
analisa_condicao ambiente controle "while" condicao pos;

```

```

controle.emLoopOuSwitch <- true;
analisa_instrucoes ambiente controle instrucoes;
controle.emLoopOuSwitch <- false;
analisa_instrucoes ambiente controle ils
| (BlocoDoWhile (condicao,instrucoes,pos)) ->
analisa_instrucao_apos_break controle pos;
controle.emLoopOuSwitch <- true;
analisa_instrucoes ambiente controle instrucoes;
analisa_condicao ambiente controle "do..while" condicao pos;
controle.emLoopOuSwitch <- false;
analisa_instrucoes ambiente controle ils
| (BlocoSwitch (condicao, caselist, pos)) ->
analisa_instrucao_apos_break controle pos;
analisa_condicao_switch ambiente controle condicao pos;
controle.emLoopOuSwitch <- true;
(*analizando caselist*)
(
match caselist with
(*aqui fica a saida do BlocoSwitch*)
[] -> ()

| cs1 -> controle.emSwitchCaseDefault <- true;
ignore(
(* funcao criada para avaliar se casoSwitch BlocoCase ou BlocoDefault.
Aplica funcao criada para cada elemento da lista caselist usando List.map *)
List.map (fun cs ->
match cs with
BlocoCase(condicao,instlist,pos1) ->
controle.caseBreak <- false;
analisa_case_switch ambiente controle condicao pos1;
analisa_instrucoes ambiente controle instlist;
if(controle.caseBreak == false) then
let msg = sprintf "Deve-se adicionar um comando \'break;\' ap s um \'case
\'\'\'n" in
exibe_erro msg pos1
| BlocoDefault(instlist,pos1) ->
controle.caseBreak <- false;
analisa_instrucoes ambiente controle instlist;
if(controle.caseBreak == false) then
let msg = sprintf "Deve-se adicionar um comando \'break;\' ap s um \'
default\'\'\'n" in
exibe_erro msg pos1
) cs1)
);
controle.caseBreak <- false;
controle.emSwitchCaseDefault <- false;
controle.emLoopOuSwitch <- false;
(* desempilha tipo na pilha \'tipoSwitch\' *)
controle.tipoSwitch <- List.tl controle.tipoSwitch;
analisa_instrucoes ambiente controle ils
| (CmdAtrib (varnome, expressao, pos)) ->

```

```

analisa_instrucao_apos_break controle pos;
analisa_atrib ambiente controle (varnome, expressao, pos);
analisa_instrucoes ambiente controle ils
| (CmdChamadaMetodo (nomemetodo, nomeclasse, explist, pos)) ->
analisa_chamada_metodo ambiente controle nomemetodo nomeclasse explist pos;
analisa_instrucoes ambiente controle ils
| (CmdIncr (varnome, pos)) ->
analisa_incr_decr ambiente controle varnome pos;
analisa_instrucoes ambiente controle ils
| (CmdDecr (varnome, pos)) ->
analisa_incr_decr ambiente controle varnome pos;
analisa_instrucoes ambiente controle ils
| (CmdScanner (varnome, pos)) ->
analisa_scanner ambiente controle varnome pos;
analisa_instrucoes ambiente controle ils
| (CmdRead (vardest, varscanner, tipoLeitura, posdest, posscanner)) ->
analisa_read ambiente controle vardest varscanner tipoLeitura posdest
    posscanner;
analisa_instrucoes ambiente controle ils
| (CmdPrint (expressao, pos)) ->
ignore(analisa_expressao ambiente controle expressao controle.classeAtual
    controle.metodoAtual pos);
analisa_instrucoes ambiente controle ils
| (CmdPrintln (expressao, pos)) ->
ignore(analisa_expressao ambiente controle expressao controle.classeAtual
    controle.metodoAtual pos);
analisa_instrucoes ambiente controle ils
| (CmdBreak (pos)) ->
analisa_instrucao_apos_break controle pos;
analisa_instrucao_break_continue controle pos;
if(controle.emSwitchCaseDefault) then
controle.caseBreak <- true;
analisa_instrucoes ambiente controle ils
| (CmdContinue (pos)) ->
analisa_instrucao_break_continue controle pos;
analisa_instrucoes ambiente controle ils
| (CmdReturn (expressao, pos)) ->
analisa_return ambiente controle expressao pos;
controle.comandoReturn <- true;
if(ils <> []) then
controle.comandoReturn <- false;
analisa_instrucoes ambiente controle ils

(** ANALISE DE PARAMETROS DE DECLARACAO DE FUNCAO **)

(* ***** analise de parametros ***** *)

let analisa_parametro ambiente controle ((tipo:ArvSint.tipo), paramnome, pos
) =
if((tipo_of_arvSintTipo tipo) = TipoVoid) then
let msg = sprintf "Parametro \'%s\' do metodo \'%s\' possui tipo \'void\'"

```

```

paramnome controle.metodoAtual in
exibe_erro msg pos

(* ***** analise de lista de parametros ***** *)

let rec analisa_lista_parametros ambiente controle paramlist =
match paramlist with
[] -> ()
| p::ps -> analisa_parametro ambiente controle p;
analisa_lista_parametros ambiente controle ps

(** ANALISE DE VARIAVEIS **)

(* ***** analise de variavel local ***** *)

let analisa_variavel_local ambiente controle (varnome, (tipo:ArvSint.tipo),
    expressao, pos) =
analisa_variavel_local_ordem ambiente controle (varnome, (tipo:ArvSint.tipo)
    , expressao, pos) 0

(* ***** analise de lista de variaveis locais ***** *)

let rec analisa_variaveis_locais ambiente controle varlist =
match varlist with
[] -> ()
| vl::vls -> analisa_variavel_local ambiente controle vl;
analisa_variaveis_locais ambiente controle vls

(* ***** analise de variavel global ***** *)

let analisa_variavel_global ambiente controle (varnome, (tipo:ArvSint.tipo),
    expressao, pos) =
let tp = (tipo_of_arvSintTipo tipo) in
if(existe_variavel_global ambiente controle controle.classeAtual varnome)
then
let msg = sprintf "Variavel \'%s\' ja foi declarada global em classe \'%s\'"
varnome controle.classeAtual in
exibe_erro msg pos
else
begin
analisa_expressao_declaracao ambiente controle varnome tp expressao pos;
if(not (is_expressao_none controle expressao))
then
if(not(is_expressao_constante_simples controle expressao))
then
let msg = sprintf "Expressao em variavel \'%s\' em classe \'%s\' n o
    constante."
    varnome controle.classeAtual in
exibe_erro msg pos
else
insere_variavel_global ambiente controle varnome tp expressao

```

```

else
insere_variavel_global ambiente controle varnome tp expressao
end

(* ***** analise de lista de variaveis globais ***** *)

let rec analisa_variaveis_globais ambiente controle varlist =
match varlist with
[] -> ()
| vg::vgs -> analisa_variavel_global ambiente controle vg;
analisa_variaveis_globais ambiente controle vgs

(** ANALISE DE METODOS **)

(* ***** analise da metodo main ***** *)

let analisa_metodo_main ambiente controle (paramlist, varlist, instlist, pos
) =
if (controle.classeMain <> "") then
let msg = sprintf "Metodo main ja foi declarado na classe \"%s\" \n" controle
.classeMain in
exibe_erro msg pos
else
controle.classeMain <- controle.classeAtual;
insere_metodo ambiente controle "main" TipoVoid paramlist;
controle.metodoAtual <- "main";
analisa_variaveis_locais ambiente controle varlist;
analisa_instrucoes ambiente controle instlist

(* ***** analise do metodo ***** *)

let analisa_metodo ambiente controle ((nomemetodo, (tp:ArvSint.tipo), parlist
), varlist , instlist, pos) =
if(existe_metodo ambiente controle controle.classeAtual nomemetodo) then
let msg = sprintf "Metodo %s ja foi declarado na classe \"%s\" \n"
nomemetodo controle.classeAtual in
exibe_erro msg pos
else
insere_metodo ambiente controle nomemetodo (tipo_of_arvSintTipo tp) parlist;
controle.metodoAtual <- nomemetodo;
analisa_lista_parametros ambiente controle parlist;
analisa_variaveis_locais ambiente controle varlist;
analisa_instrucoes ambiente controle instlist;
if((tipo_of_arvSintTipo tp) <> TipoVoid)
then
if((controle.comandoReturn)) then
controle.comandoReturn <-false
else
let msg = sprintf "Metodo \"%s\" da classe \"%s\" deve retornar um valor \n"
controle.metodoAtual controle.classeAtual in
exibe_erro msg pos

```



```

(* ***** analise da lista de metodos ***** *)

let rec analisa_lista_metodos ambiente controle metodolist =
match metodolist with
[] -> ()
| m :: mls ->
match m with
(Metodo (info,vars,insts,pos)) ->
analisa_metodo ambiente controle (info, vars, insts, pos);
analisa_lista_metodos ambiente controle mls
| (MetodoMain (paramlist,vars,insts,pos)) ->
analisa_metodo_main ambiente controle (paramlist, vars, insts, pos);
analisa_lista_metodos ambiente controle mls

(** ANALISE DE CLASSES **)

(* ***** analise da classe ***** *)

let analisa_classe ambiente controle (nomeclasse, varlist, metodolist, pos)
=
if(existe_classe ambiente controle nomeclasse) then
let msg = sprintf "Ja existe uma classe de nome %s\n" nomeclasse in
exibe_erro msg pos
else
insere_classe ambiente controle nomeclasse;
controle.classeAtual <- nomeclasse;
analisa_variaveis_globais ambiente controle varlist;
analisa_lista_metodos ambiente controle metodolist

(* ***** analise da lista de classes ***** *)

let rec analisa_lista_classes ambiente controle classelist =
match classelist with
[] ->
begin
if (controle.classeMain = "") then
let msg = sprintf "Erro semantico: Metodo main nao foi declarado em alguma
classe" in
failwith msg
else
ambiente
end
| c :: cls -> analisa_classe ambiente controle c;
analisa_lista_classes ambiente controle cls

(** INICIO **)

(* ***** semantico ***** *)

let inicializaCampos =

```

```

let controle = { expbin_consulta = Hashtbl.create 23;
expun_consulta = Hashtbl.create 23;
scannerImportado = false;
classeMain = "";
classeAtual = "";
metodoAtual = "";
tipoSwitch = [];
emSwitchCaseDefault = false;
caseBreak = false;
comandoReturn = false;
ordemLoopForAtual = 0;
emLoopOuSwitch = false;
tblVarGlobaisInic = Hashtbl.create 23
} in
(preenche_expbin_consulta controle;
preenche_expun_consulta controle;
controle)

let semantico arv =
let (ambiente: (string, info_classe) Hashtbl.t) = Hashtbl.create 23
and controle = inicializaCampos
and value = fst arv
and lista = snd arv in
(controle.scannerImportado <- value;
let amb = analisa_lista_classes ambiente controle lista in
(amb, controle))


\subsection{C digo - Interpretador}
\begin{terminal}
open ArvSint;;
open Printf;;
open Scanf;;
open Semantico;;

(** Ponteiros **)

let classeMain = ref ""
let loopInterno = ref false
let sairLoopInterno = ref false
let comandoContinue = ref false
let mudarOrdemForLoop = ref true
let ocorreuAtribFor = ref false

(***** INTERPRETADOR *****)

```

```

let obter_valor_tipo_option optionvalue =
match optionvalue with
Some c -> c
| None -> failwith "expressao none"

let obter_variavel ambiente varnome nomeclasse nomemetodo =
if(Semantico.existe_variavel_global ambiente nomeclasse varnome) then
Semantico.obter_variavel_global ambiente !classeMain varnome
else
Semantico.obter_variavel_local ambiente nomeclasse nomemetodo varnome

let obter_variavel_main ambiente varnome =
obter_variavel ambiente varnome !classeMain "main"

let avalia_op_bin_expressao op arg1 arg2 =
match op with
(** operacoes aritmeticas **)
(* operador soma *)
Soma ->
(* analisa arg1 *)
begin match arg1 with

(ValorInt v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt v2) -> ValorInt (v1 + v2)
| (ValorDouble v2) -> ValorDouble ((float_of_int v1) +. v2)
| (ValorString v2) -> ValorString ((string_of_int v1) ^ v2)
|_ -> failwith "impossivel pegar valor"
end

| (ValorDouble v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt v2) -> ValorDouble (v1 +. (float_of_int v2))
| (ValorDouble v2) -> ValorDouble (v1 +. v2)
| (ValorString v2) -> ValorString ((string_of_float v1) ^ v2)
|_ -> failwith "impossivel pegar valor"
end

| (ValorChar v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorString v2) -> ValorString ((Char.escaped v1) ^ v2)
|_ -> failwith "impossivel pegar valor"
end

| (ValorBoolean v1) ->
(* analisa arg2 *)

```

```

begin match arg2 with
(ValorString v2) -> ValorString ((string_of_bool v1) ^ v2)
|_ -> failwith "impossivel pegar valor"
end

| (ValorString v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt v2) -> ValorString (v1 ^ (string_of_int v2))
| (ValorDouble v2) -> ValorString (v1 ^ (string_of_float v2))
| (ValorChar v2) -> ValorString (v1 ^ (Char.escaped v2))
| (ValorBoolean v2) -> ValorString (v1 ^ (string_of_bool v2))
| (ValorString v2) -> ValorString (v1 ^ v2)
|_ -> failwith "impossivel pegar valor"

end

|_ -> failwith "impossivel pegar valor"
end
(* fim operador soma *)
(* operador sub *)
| Sub ->
(* analisa arg1 *)
begin match arg1 with

(ValorInt v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt v2) -> ValorInt (v1 - v2)
| (ValorDouble v2) -> ValorDouble ((float_of_int v1) -. v2)
|_ -> failwith "impossivel pegar valor"
end

| (ValorDouble v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt v2) -> ValorDouble (v1 -. (float_of_int v2))
| (ValorDouble v2) -> ValorDouble (v1 -. v2)
|_ -> failwith "impossivel pegar valor"
end

|_ -> failwith "impossivel pegar valor"
end

(* fim operador mul *)
(* operador div *)
| Div ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) ->
(* analisa arg2 *)
begin match arg2 with

```

```

(ValorInt    v2)  -> ValorInt    (v1 / v2)
| (ValorDouble v2) -> ValorDouble ((float_of_int v1) /. v2)
|_ -> failwith "impossivel pegar valor"
end

| (ValorDouble v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt    v2)  -> ValorDouble (v1 /. (float_of_int v2))
| (ValorDouble v2) -> ValorDouble (v1 /. v2)
|_ -> failwith "impossivel pegar valor"
end
|_ -> failwith "impossivel pegar valor"
end
(* fim operador div *)
(* operador mod *)
| Mod ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt    v2)  -> ValorInt (v1 mod v2)
|_ -> failwith "impossivel pegar valor"
end
|_ -> failwith "impossivel pegar valor"
end
(* fim operador mod *)

(** operacoes comparacao **)
(* operador Maior *)
| Maior ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt    v2)  -> ValorBoolean (v1 > v2)
|_ -> failwith "impossivel pegar valor"
end
| (ValorDouble v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorDouble    v2)  -> ValorBoolean (v1 > v2)
|_ -> failwith "impossivel pegar valor"
end
|_ -> failwith "impossivel pegar valor"
end
(*fim operador Maior *)

```

```

(* operador Menor *)
| Menor ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt v2) -> ValorBoolean (v1 < v2)
|_ -> failwith "impossivel pegar valor"
end

| (ValorDouble v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorDouble v2) -> ValorBoolean (v1 < v2)
|_ -> failwith "impossivel pegar valor"
end
|_ -> failwith "impossivel pegar valor"
end
(*fim operador Menor *)

(* operador MaiorIgual *)
| MaiorIgual ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt v2) -> ValorBoolean (v1 >= v2)
|_ -> failwith "impossivel pegar valor"
end

| (ValorDouble v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorDouble v2) -> ValorBoolean (v1 >= v2)
|_ -> failwith "impossivel pegar valor"
end
|_ -> failwith "impossivel pegar valor"
end
(*fim operador MaiorIgual *)

(* operador MenorIgual *)
| MenorIgual ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt v2) -> ValorBoolean (v1 <= v2)
|_ -> failwith "impossivel pegar valor"
end

```

```

end

| (ValorDouble v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorDouble v2) -> ValorBoolean (v1 <= v2)
|_ -> failwith "impossivel pegar valor"
end
|_ -> failwith "impossivel pegar valor"
end
(*fim operador Menor *)

(* operador MenorIgual *)
| Igual ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt v2) -> ValorBoolean (v1 = v2)
|_ -> failwith "impossivel pegar valor"
end
end
| (ValorDouble v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorDouble v2) -> ValorBoolean (v1 = v2)
|_ -> failwith "impossivel pegar valor"
end
| (ValorChar v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorChar v2) -> ValorBoolean (v1 = v2)
|_ -> failwith "impossivel pegar valor"
end
|_ -> failwith "impossivel pegar valor"
end
(*fim operador Menor *)

(* operador MenorIgual *)
| Diferente ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorInt v2) -> ValorBoolean (v1 <> v2)
|_ -> failwith "impossivel pegar valor"
end
end
| (ValorDouble v1) ->

```

```

(* analisa arg2 *)
begin match arg2 with
(ValorDouble    v2)  -> ValorBoolean (v1 <> v2)
|_ -> failwith "impossivel pegar valor"
end

| (ValorChar v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorChar    v2)  -> ValorBoolean (v1 <> v2)
|_ -> failwith "impossivel pegar valor"
end
|_ -> failwith "impossivel pegar valor"
end

(*fim operador Menor *)
(** operacoes logicos **)
(* operador E *)
| E ->
(* analisa arg1 *)
begin match arg1 with
(ValorBoolean v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorBoolean    v2)  -> ValorBoolean (v1 && v2)
|_ -> failwith "impossivel pegar valor"
end
end

|_ -> failwith "impossivel pegar valor"
end

(*fim operador E *)
| Ou ->
(* analisa arg1 *)
begin match arg1 with
(ValorBoolean v1) ->
(* analisa arg2 *)
begin match arg2 with
(ValorBoolean    v2)  -> ValorBoolean (v1 || v2)
|_ -> failwith "impossivel pegar valor"
end
end

|_ -> failwith "impossivel pegar valor"
end

| _ -> failwith "impossivel pegar valor"

(*
COMPLETAR COM O RESTANTE DAS EXPRESSOES UNARIAS

*)
let avalia_op_un_expressao op arg1 =
match op with
Nao ->

```



```

(* analisa arg1 *)
begin match arg1 with
(ValorBoolean v1) -> ValorBoolean(not v1)
|_ -> failwith "impossivel pegar valor"
end

|Posit ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) -> ValorInt(+ v1)
|(ValorDouble v1) -> ValorDouble(+. v1)
|_ -> failwith "impossivel pegar valor"
end

|Negat ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) -> ValorInt(- v1)
|(ValorDouble v1) -> ValorDouble(-. v1)
|_ -> failwith "impossivel pegar valor"
end

|Incr ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) -> ValorInt(v1 + 1)
|_ -> failwith "impossivel pegar valor"
end

|Decr ->
(* analisa arg1 *)
begin match arg1 with
(ValorInt v1) -> ValorInt(v1 - 1)
|_ -> failwith "impossivel pegar valor"
end

| _ -> failwith "impossivel pegar valor"

let rec avalia_expressao ambiente nomeclasse nomemetodo expressao =
match expressao with
(ExpInt v) -> Some (ValorInt v)
| (ExpBoolean v) -> Some (ValorBoolean v)
| (ExpChar v) -> Some (ValorChar v)
| (ExpString v) -> Some (ValorString v)
| (ExpFloat v) -> Some (ValorFloat v)
| (ExpDouble v) -> Some (ValorDouble v)
| (ExpBin (op, exp1, exp2)) ->
let arg1 = obter_valor_tipo_option (avalia_expressao ambiente nomeclasse
nomemetodo exp1) in

```

```

let arg2 = obter_valor_tipo_option (avalia_expressao ambiente nomeclasse
    nomemetodo exp2) in
Some (avalia_op_bin_expressao op arg1 arg2)
| (ExpUn (op, exp1)) ->
let arg1 = obter_valor_tipo_option (avalia_expressao ambiente nomeclasse
    nomemetodo exp1) in
Some (avalia_op_un_expressao op arg1)
| (ExpVar (varnome)) ->
let entradaVar = obter_variavel ambiente varnome nomeclasse nomemetodo in
entradaVar.valorVar

| _ -> None

let avalia_print_main ambiente expressao =
let exp = avalia_expressao ambiente !classeMain "main" expressao in
let n = obter_valor_tipo_option exp in
match n with
(ValorInt v) -> let resp = sprintf "%d" v in print_endline resp
| (ValorFloat v) -> let resp = sprintf "%f" v in print_endline resp
| (ValorDouble v) -> let resp = sprintf "%f" v in print_endline resp
| (ValorChar v) -> let resp = sprintf "%c" v in print_endline resp
| (ValorString v) -> let resp = sprintf "%s" v in print_endline resp
| (ValorBoolean v) -> let resp = sprintf "%s" (string_of_bool v)
in print_endline resp

let avalia_println_main ambiente expressao =
avalia_print_main ambiente expressao;
printf "\n"

let leitura_variavel tipoLeitura =
try
match tipoLeitura with
ReadInt ->
let value = scanf "%d" (fun n -> n) in
Some (ValorInt (value))

| ReadFloat ->
let value = scanf "%f" (fun n -> n) in
Some (ValorFloat (value))

| ReadDouble ->
let value = scanf "%f" (fun n -> n) in
Some (ValorDouble (value))

| ReadByte ->
let value = scanf "%c" (fun n -> n) in
Some (ValorChar (value))

| ReadBool ->
let value = scanf "%s" (fun n -> n) in
if ((value = "true") || (value = "false")) then

```

```

Some (ValorBoolean (bool_of_string value))
else
failwith "'boolean' deve ser 'true' ou 'false'"
| ReadString ->
let value = scanf "%S" (fun n -> n) in
Some (ValorString (value))
with Scanf.Scan_failure _->
failwith "Valor da leitura esta vazio"

let avalia_read_main ambiente varnome tipoLeitura =
let entradaVar = obter_variavel_main ambiente varnome in
entradaVar.valorVar <- leitura_variavel tipoLeitura

let avalia_decr_main ambiente varnome =
let entradaVar = obter_variavel_main ambiente varnome in
let valor = avalia_op_un_expressao Decr (obter_valor_tipo_option
    entradaVar.valorVar) in
entradaVar.valorVar <- Some (valor)

let avalia_incr_main ambiente varnome =
let entradaVar = obter_variavel_main ambiente varnome in
let valor = avalia_op_un_expressao Incr (obter_valor_tipo_option
    entradaVar.valorVar) in
entradaVar.valorVar <- Some (valor)

let avalia_atrib_main ambiente varnome expressao =
let entradaVar = obter_variavel_main ambiente varnome in
entradaVar.valorVar <- avalia_expressao ambiente !classeMain "main"
    expressao

(*
let print_int_value ambiente varnome =
let entradaVar = Semantico.obter_variavel_local ambiente !classeMain "main"
    " varnome in
printf "valor atribuido: %d"
((fun c -> match c with ValorInt v -> v | _ -> -123)
(obter_valor_tipo_option entradaVar.valorVar))
*)

let avalia_for_incr_main ambiente incremento =
match incremento with
(ExpFor (varnome,expressao,_)) ->
    avalia_atrib_main ambiente varnome expressao
| (IncrFor (varnome,_))->
    avalia_incr_main ambiente varnome
| (DecrFor (varnome,_))->
    avalia_decr_main ambiente varnome

let avalia_for_atrib_main ambiente atribuicao =

```

```

match atribuicao with
(AtribFor (varnome,expressao,_)) ->
avalia_atrib_main ambiente varnome expressao
| (AtribVarFor (varnome,_,expressao,_)) ->
avalia_atrib_main ambiente varnome expressao

let rec avalia_instrucoes_main ambiente instlist =
match instlist with
[] -> mudarOrdemForLoop := false; ()
| i :: ils ->
match i with
(BlocoIf (condicao, instrucoes,_)) ->
if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
(let cond = avalia_expressao ambiente !classeMain "main" condicao in
if((obter_valor_tipo_option cond) = (ValorBoolean (true))) then
avalia_instrucoes_main ambiente instrucoes;

avalia_instrucoes_main ambiente ils)
| (BlocoIfElse (condicao, instrucoesIf,instrucoesElse,_)) ->
if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
(let cond = avalia_expressao ambiente !classeMain "main" condicao in
if((obter_valor_tipo_option cond) = (ValorBoolean (true))) then
avalia_instrucoes_main ambiente instrucoesIf
else
avalia_instrucoes_main ambiente instrucoesElse;

avalia_instrucoes_main ambiente ils)

| (BlocoFor ((atribuicao,(condicao,_),incremento), instrucoes,_)) ->
(* avisa que esta em um loop interno *)
comandoContinue := false;
loopInterno := true;
if(!sairLoopInterno) then
begin
sairLoopInterno := false;
ocorreuAtribFor := false;
Semantico.ordemLoopForAtual := !(Semantico.ordemLoopForAtual) - 1;
avalia_instrucoes_main ambiente ils

end
else
begin
if(!mudarOrdemForLoop) then
Semantico.ordemLoopForAtual := !(Semantico.ordemLoopForAtual) + 1;
if(not !ocorreuAtribFor) then
(avalia_for_atrib_main ambiente atribuicao;
ocorreuAtribFor := true);
let cond = avalia_expressao ambiente !classeMain "main" condicao in

```

```

if((obter_valor_tipo_option cond) = (ValorBoolean (true)) ) then
begin
mudarOrdemForLoop := true;
avalia_instrucoes_main ambiente instrucoes;
avalia_for_incr_main ambiente incremento;
avalia_instrucoes_main ambiente instlist
end
else
begin
(* avisa que esta em um loop interno *)
loopInterno := false;
ocorreuAtribFor := false;
Semantico.ordemLoopForAtual := Semantico.(!ordemLoopForAtual) - 1;
avalia_instrucoes_main ambiente ils
end
end

| (BlocoWhile (condicao, instrucoes,_)) ->
(* avisa que esta em um loop interno *)
comandoContinue := false;
loopInterno := true;
if(!sairLoopInterno) then
begin
sairLoopInterno := false;
avalia_instrucoes_main ambiente ils
end
else
begin
let cond = avalia_expressao ambiente !classeMain "main" condicao in
if((obter_valor_tipo_option cond) = (ValorBoolean (true)) ) then
begin
avalia_instrucoes_main ambiente instrucoes;
avalia_instrucoes_main ambiente instlist
end
else
begin
(* avisa que esta em um loop interno *)
loopInterno := false;
avalia_instrucoes_main ambiente ils
end
end

| (BlocoDoWhile (condicao, instrucoes,_)) ->
(* avisa que esta em um loop interno *)
comandoContinue := false;
loopInterno := true;
if(!sairLoopInterno) then
begin
sairLoopInterno := false;
avalia_instrucoes_main ambiente ils
end
end

```

```

else
begin
  avalia_instrucoes_main ambiente instrucoes;
  let cond = avalia_expressao ambiente !classeMain "main" condicao in
  if((obter_valor_tipo_option cond) = (ValorBoolean (true)) ) then
  begin
    avalia_instrucoes_main ambiente instrucoes;
    avalia_instrucoes_main ambiente instlist
  end
  else
  begin
    (* avisa que esta em um loop interno *)
    loopInterno := false;
    avalia_instrucoes_main ambiente ils
  end
end

| (BlocoSwitch (condicao, caselist, pos)) ->
if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
  (let valueSwitch = avalia_expressao ambiente !classeMain "main" condicao
   in
   (*if((obter_valor_tipo_option cond) = (ValorBoolean (true)) ) then *)
   (match caselist with
    [] -> ()
    | cls ->
    ignore(List.map (fun cs ->
    match cs with

    BlocoCase(condicao,instlist,_) ->
    let valueCase = avalia_expressao ambiente !classeMain "main" condicao in
    if((obter_valor_tipo_option valueSwitch) = (obter_valor_tipo_option
      valueCase)) then
    avalia_instrucoes_main ambiente instlist;

    | BlocoDefault(instlist,_) ->
    avalia_instrucoes_main ambiente instlist;

    ) cls));

  avalia_instrucoes_main ambiente ils)

| (CmdAtrib (varnome, expressao, _)) ->
if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
  (avalia_atrib_main ambiente varnome expressao;
  avalia_instrucoes_main ambiente ils)
| (CmdIncr (varnome, _)) ->
if(!comandoContinue) then avalia_instrucoes_main ambiente []

```

```

else
  (avalia_incr_main ambiente varnome;
  avalia_instrucoes_main ambiente ils)
  | (CmdDecr(varnome,_)) ->
  if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
  (avalia_decr_main ambiente varnome;
  avalia_instrucoes_main ambiente ils)
  | (CmdPrint (expressao,_)) ->
  if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
  (avalia_print_main ambiente expressao;
  avalia_instrucoes_main ambiente ils)
  | (CmdPrintln (expressao,_)) ->
  if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
  (avalia_println_main ambiente expressao;
  avalia_instrucoes_main ambiente ils)
  | (CmdScanner (_,_)) ->
  if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
  (avalia_instrucoes_main ambiente ils)
  | (CmdRead (varnome, _, tipoLeitura, _, _)) ->
  if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
  (avalia_read_main ambiente varnome tipoLeitura;
  avalia_instrucoes_main ambiente ils)
  | (CmdBreak (_)) ->
  if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
  (if(!loopInterno)
  then
  begin
    (* se ele esta em um loop interno
    alerta que deve sair deste loop antes de
    retornar ao inicio do loop. Para evitar
    que dentro do loop leia o restante das
    instrucoes, a proxima lista de instrucao
    sera uma lista vazia
    *)
    sairLoopInterno := true;
    avalia_instrucoes_main ambiente []
  end
  else
  avalia_instrucoes_main ambiente ils)
  | (CmdContinue) ->
  if(!comandoContinue) then avalia_instrucoes_main ambiente []
else
  (if(!loopInterno)
  then
  begin

```

```

(* se ele esta em um loop interno
apenas nao l restante das instrucoes
internas ao loop, mas continua no loop
*)
comandoContinue := true;
avalia_instrucoes_main ambiente []
end
else
avalia_instrucoes_main ambiente ils)

| _ -> avalia_instrucoes_main ambiente ils

let avalia_variavel_local_main ambiente (varnome, _, expressao, _) =
let entradaVar = Semantico.obter_variavel_local ambiente !classeMain "main"
    " varnome in
entradaVar.valorVar <- avalia_expressao ambiente !classeMain "main"
    expressao

let rec avalia_variaveis_locais_main ambiente varlist =
match varlist with
[] -> ()
| v :: vls ->
avalia_variavel_local_main ambiente v;
avalia_variaveis_locais_main ambiente vls

let avalia_variavel_global_main ambiente (varnome, _, expressao, _) =
let entradaVar = Semantico.obter_variavel_global ambiente !classeMain
    varnome in
entradaVar.valorVar <- avalia_expressao ambiente !classeMain "main"
    expressao

let rec avalia_variaveis_globais_main ambiente varlist =
match varlist with
[] -> ()
| v :: vls ->
avalia_variavel_global_main ambiente v;
avalia_variaveis_globais_main ambiente vls

let rec procura_metodo_main ambiente metodolist =
match metodolist with
[] -> printf "Feito\n"
| m :: mls ->
begin
match m with
(MetodoMain(_, varlist, instlist, _)) ->
avalia_variaveis_locais_main ambiente varlist;
avalia_instrucoes_main ambiente instlist
| (Metodo _) ->
procura_metodo_main ambiente mls
end

```



```

let rec procura_classe_main ambiente classelist =
match classelist with
[] -> printf "Feito\n"
| (nomeclasse,varlist,metodolist,_) :: cls ->
begin
if (nomeclasse = !classeMain)
then
begin
avalia_variaveis_globais_main ambiente varlist;
procura_metodo_main ambiente metodolist
end
else
procura_classe_main ambiente cls
end

(* Interpretador *)
let interpretador ambiente nomeclassemain arv =
classeMain := nomeclassemain;
let lista = snd arv in
(*listaClasses := snd arv;*)
procura_classe_main ambiente lista

```

Para utilizar o arquivo compilador.ml no Ocaml deve-se utilizar o comando a seguir:

```
# #use carregador.ml;;
```

## 7.7 Código - Árvore Intermediária

```

(* arvore de representacao intermediaria *)

type arvoreRI = global list * define list
(* supondo que global seja @.str pelo programa *)
and global      = STR of string * string * tamanho
| VAR of string * tipo * expressao
and define      = DEFINE of tipo * string * paramdecl list * comando
and paramdecl   = tipo * string
and paramcham   = tipo * expressao
and comando     = ALLOCA of string * tipo
| MOVE of expressao * expressao
(* load e store: grave/carregue resultado da segunda expressao
na primeira expressao, ambas de tipo 'tipo'. bool para saber
se variavel      global (true) ou local (false) *)
(* %<temp> = load <tipo>* <var> *)
| LOAD of expressao * string * tipo * bool
(* store <tipo> <valor>, <tipo>* <var> *)
| STORE of expressao * string * tipo * bool
| JUMP of label
| SWITCH of expressao * label * tipo * case list
| CJUMP of oprel * expressao * expressao * label * label

```

```

| LABEL of label
| SEQ   of comando list
| RET   of tipo * expressao
(* contem o nome da variavel, o tipo e
se a variavel   global(true) ou local (false)*)
| SCAN  of string * tipo * bool
| PRINT of expressao * tamanho * tipo
(* temp do double, temp do float *)
| FPEXT of expressao * expressao
| PRINTLN
| FFLUSH_STDIN of expressao
and expressao = ConstInt   of int
| ConstChar of char
| ConstDouble of float
| ConstBoolean of bool
| TEMP   of label
| BINOP  of opbin * expressao * expressao * tipo
| ICMP   of oprel * expressao * expressao * tipo
| CALL   of tipo * string * paramcham list
| ESEQ   of comando * expressao
and opbin = ADD | FADD | SUB | FSUB | MUL | FMUL | DIV | FDIV | SREM
| AND | OR | XOR
and oprel = EQ | NE | SLT | SGT | SLE | SGE
| OEQ | ONE | OLT | OGT | OLE | OGE
and case = expressao * label
and label = string
and tamanho = int
and tipo = RInt | RChar | RDouble | RBoolean | RString | RVoid

```

## 7.8 Código - Gerador de Código

```

open Printf;;
open RepInter;;
open ArvRI;;
open String;;

(***** GERA MAKEFILE *****)

let gera_Makefile diretorio nomearquivo =
let out = open_out (diretorio^"Makefile") in
Printf.fprintf out "EXECUTABLE=%s\n" nomearquivo;
flush out;
Printf.fprintf out "FILENAME=%s\n\n" nomearquivo;
flush out;
Printf.fprintf out "all: $(FILENAME).ll\n";
flush out;
Printf.fprintf out "\tllvm-as $(FILENAME).ll\n";
flush out;

```

```

Printf.fprintf out "\tllc $(FILENAME).bc\n";
flush out;
Printf.fprintf out "\tgcc -o $(EXECUTABLE) $(FILENAME).s\n";
flush out;
Printf.fprintf out "\trm $(FILENAME).s\n";
flush out;
Printf.fprintf out "\trm $(FILENAME).bc\n";
flush out;
close_out out

(***** FUNCOES UTEIS *****)

let ultimo_caracter str =
let ultimaPosicao = (String.length str) - 1 in
str.[ultimaPosicao]

let is_eseq expressao =
match expressao with
ESEQ(_) -> true
| _ -> false

let obter_comando_eseq expressao =
match expressao with
ESEQ(comando,_) -> comando
| _ -> failwith "nao comando eseq"

let obter_align tipo =
match tipo with
RInt -> "align 4"
| RChar -> "align 1"
| RDouble -> "align 4"
| RBoolean -> "align 1"
| RString -> "align 4"
| _ -> failwith "tipo Void"

let string_of_tipo tipo =
match tipo with
RVoid -> "%int"
| RInt -> "%int"
| RChar -> "%char"
| RDouble -> "double"
| RBoolean -> "%boolean"
| _ -> failwith "n o existe tipo para string ainda"

let string_of_opbin op =
match op with
ADD -> "add"
| FADD -> "fadd"
| SUB -> "sub"
| FSUB -> "fsub"

```

```

| MUL  -> "mul"
| FMUL -> "fmul"
| DIV  -> "div"
| FDIV -> "fdiv"
| SREM -> "srem"
| AND  -> "and"
| OR   -> "or"
| XOR  -> "xor"

let string_of_oprel op =
match op with
EQ  -> "eq"
| NE  -> "ne"
| SLT -> "slt"
| SGT -> "sgt"
| SLE -> "sle"
| SGE -> "sge"
| OEQ -> "eq"
| ONE -> "ne"
| OLT -> "slt"
| OGT -> "sgt"
| OLE -> "sle"
| OGE -> "sge"

let descola_ptr_canal out =
let ptr = (pos_out out) in
seek_out out (ptr - 1)

(***** GERACAO DE CODIGO
*****

let gera_expressao_op_str expressao tipo =
match expressao with
ConstInt v      ->
let str = string_of_int v in
if(tipo == RDouble) then (str^".0") else str
| ConstChar v    ->
(string_of_int (Char.code v))
| ConstDouble v  ->
let str = string_of_float v in
if((ultimo_caracter str) == '.') then (str^"0") else str
| ConstBoolean v -> (if(v) then "1" else "0")
| ESEQ(_,TEMP(t))-> "%" ^ t
| TEMP(t)        -> "%" ^ t
|_ -> failwith "EXPRESSAO NAO FOI UTILIZADA"

(* obter o conteudo de uma expressao simples como string *)

let gera_expressao_str expressao =
match expressao with
ConstInt v      ->

```

```

string_of_int v
| ConstChar v    ->
(string_of_int (Char.code v))
| ConstDouble v  ->
let str = string_of_float v in
if((ultimo_caracter str) == '.') then (str^"0") else str
| ConstBoolean v -> (if(v) then "1" else "0")
| ESEQ(_,TEMP(t))-> "%"^t
| TEMP(t)         -> "%"^t
|_ -> failwith "EXPRESSAO NAO FOI UTILIZADA"

(* retorna a string para imprimir saltos de linha *)

let gera_println_str =
"call %int (%char*, ...) * ^"
"@printf(%char* getelementptr ^"
"([2 x %char]* @.print.newline, %int 0, %int 0))\n"

(* retorna a string para imprimir dados *)

let gera_print_str expressao tamanho tipo =
let e      = gera_expressao_str expressao in
let iniStr = "call %int (%char*, ...) * ^"
"@printf(%char* getelementptr inbounds " in
match tipo with
RInt      -> iniStr ^ "([3 x %char]* @.print.int, %int 0, %int 0), %int ^e^"
) \n"
| RChar    -> iniStr ^ "([3 x %char]* @.print.char, %int 0, %int 0), %char ^"
^e^") \n"
| RDouble  -> iniStr ^ "([4 x %char]* @.print.double, %int 0, %int 0), "
double ^e^") \n"
| RBoolean ->
if(e == "1") then
iniStr ^ "([5 x %char]* @.print.true, %int 0, %int 0))\n"
else
iniStr ^ "([6 x %char]* @.print.false, %int 0, %int 0))\n"
| RString  ->
let nomeStr = String.sub e 1 ((String.length e) - 1) in
iniStr ^ "([\"^(string_of_int tamanho)\" x %char]* @\"^nomeStr^", %int 0, %int
0))\n"
| _ -> failwith "tipo Void n o impresso"

(* retorna a string para ler dados *)
let gera_scan_str varnome tipo escopo =
let iniStr = "call %int (%char*, ...) * ^"
"@scanf(%char* getelementptr inbounds " in
match tipo with
RInt      -> iniStr ^ "([3 x %char]* @.print.int, %int 0, %int 0), %int* ^"
escopo^varnome^") \n"
| RChar    -> iniStr ^ "([3 x %char]* @.print.char, %int 0, %int 0), %char* "
"^escopo^varnome^") \n"

```

```

| RDouble  -> iniStr ^ "([4 x %char]* @.print.double, %int 0, %int 0),
    double*  ^escopo^varnome^")\n"
| _ -> failwith "tipo Void e String n o s o lidas ainda"

(** GERA COMANDOS **)

let gera_println out =
Printf.fprintf out "%s" gera_println_str;
flush out

let gera_print out expressao tamanho tipo =
Printf.fprintf out "%s" (gera_print_str expressao tamanho tipo);
flush out

let gera_scan out expressao tipo global =
(if(global) then
Printf.fprintf out "%s" (gera_scan_str expressao tipo "@")
else
Printf.fprintf out "%s" (gera_scan_str expressao tipo "%"));
flush out

let gera_load out expressao varnome tipo global =
let strTipo = (string_of_tipo tipo) in
let strExp  = gera_expressao_str expressao in
let align   = obter_align tipo in
(if(global)
then
Printf.fprintf out "\t%s = load %s* %s, %s\n"
strExp strTipo ("@"^varnome) align
else
Printf.fprintf out "\t%s = load %s* %s, %s\n"
strExp strTipo ("% "^varnome) align
); flush out

(* gera possiveis comandos presentes em eseq.
Tais comandos sao: MOVE, LOAD, PRINT, PRINTLN, SCAN *)
let rec gera_comando_eseq out comando =
match comando with
LOAD(expressao, varnome, tipo, global) ->
gera_load out expressao varnome tipo global
| MOVE(temp, expressao)->
(* GERAR MOVE. SER EXECUTADO DENTRO DO CORPO DESTA FUNCAO POR
PODER POSSUIR UM ESEQ *)
begin
let strTemp = gera_expressao_str temp in
begin
match expressao with
ESEQ(c,e)->
let strTemp = gera_expressao_str temp in
Printf.fprintf out "\t%s = " strTemp; flush out;

```

```

gera_comando_eseq out c
| BINOP(oper,exp1,exp2,tipos) ->
(
  if(is_eseq exp1) then
    gera_comando_eseq out (obter_comando_eseq exp1);
  if(is_eseq exp2) then
    gera_comando_eseq out (obter_comando_eseq exp2);
  let e1      = gera_expressao_op_str exp1 tipos
  and e2      = gera_expressao_op_str exp2 tipos
  and op      = string_of_opbin oper
  and strTipos = string_of_tipos tipos in
  Printf.fprintf out "\t%s = %s %s %s, %s\n"
    strTipos op strTipos e1 e2;
  flush out;
)
| ICMP(oper,exp1,exp2,tipos) ->
(
  if(is_eseq exp1) then
    gera_comando_eseq out (obter_comando_eseq exp1);
  if(is_eseq exp2) then
    gera_comando_eseq out (obter_comando_eseq exp2);
  let e1      = gera_expressao_op_str exp1 tipos
  and e2      = gera_expressao_op_str exp2 tipos
  and op      = string_of_oprel oper
  and strTipos = string_of_tipos tipos in
  (if(tipos == RDouble)
   then
    Printf.fprintf out "\t%s = fcmp %s %s %s, %s\n"
      strTipos op strTipos e1 e2
   else
    Printf.fprintf out "\t%s = icmp %s %s %s, %s\n"
      strTipos op strTipos e1 e2);
  flush out;
)
| CALL(tipos, nomemetodo, paramlist) ->
  if(paramlist <> []) then
    (
      List.iter
        (fun (tp,exp) ->
          if(is_eseq exp) then
            gera_comando_eseq out (obter_comando_eseq exp);
          ) paramlist;
    );
    Printf.fprintf out "\t%s = " strTipos; flush out;
    let strTipos = (string_of_tipos tipos) in
    Printf.fprintf out "call %s @%s("
      strTipos nomemetodo;
    flush out;
    if(paramlist <> []) then
      (
        List.iter

```

```

(fun (tp,exp) ->
let strTipoParam = (string_of_tipo tp) in
let strExp = gera_expressao_str exp in
Printf.fprintf out "%s %s," strTipoParam strExp;
flush out
) paramlist;
descola_ptr_canal out
);
Printf.fprintf out ")\n";
flush out
|_ -> ()
end
end
| PRINTLN ->
gera_println out
| PRINT(expressao,tamanho,tipo) ->
gera_print out expressao tamanho tipo
| SCAN (expressao,tipo,global) ->
gera_scan out expressao tipo global
| _ -> ()

let gera_store out expressao varnome tipo global =
let strTipo = (string_of_tipo tipo) in
let strExp = gera_expressao_str expressao in
let align = obter_align tipo in
(if(is_eseq expressao)
then gera_comando_eseq out (obter_comando_eseq expressao);
(if(global)
then
Printf.fprintf out "\tstore %s %s, %s* %s, %s\n"
strTipo strExp strTipo ("% "^varnome) align
else
Printf.fprintf out "\tstore %s %s, %s* %s, %s\n"
strTipo strExp strTipo ("% "^varnome) align
)); flush out

let gera_fpext out tempdouble tempfloat =
let strTempDouble = gera_expressao_str tempdouble in
let strTempFloat = gera_expressao_str tempfloat in
Printf.fprintf out "\t%s = fpext float %s to double\n"
strTempDouble strTempFloat;
flush out

let gera_ret out tipo expressao =
if(is_eseq expressao) then
gera_comando_eseq out (obter_comando_eseq expressao);
let strTipo = (string_of_tipo tipo) in
let strTemp = gera_expressao_str expressao in
Printf.fprintf out "\tret %s %s\n" strTipo strTemp;
flush out

```



```

let gera_cjump out expressao l1 l2 =
if(is_eseq expressao) then
gera_comando_eseq out (obter_comando_eseq expressao);
let strTemp = gera_expressao_str expressao in
Printf.fprintf out "\tbr %%boolean %s, label %s, label %s\n"
strTemp ("%^l1) (%^l2);
flush out

let rec gera_switch_caselist out strTipo caselist =
match caselist with
[] -> ()
|(e,l)::cls ->
Printf.fprintf out "\t %s %s, label %s\n"
strTipo (gera_expressao_str e) (%^l);
flush out;
gera_switch_caselist out strTipo cls

let gera_switch out expressao labelDefault tipo caselist =
if(is_eseq expressao) then
gera_comando_eseq out (obter_comando_eseq expressao);
let strTemp = gera_expressao_str expressao in
let strTipo = string_of_tipo tipo in
Printf.fprintf out "\tswitch %s %s, label %s \n\t["\n"
strTipo strTemp ("%^labelDefault);
flush out;
gera_switch_caselist out strTipo caselist;
Printf.fprintf out "\t]\n"; flush out

let gera_jump out label =
Printf.fprintf out "\tbr label %s\n" (%^label); flush out

let gera_label out label =
Printf.fprintf out "%s:\n" label; flush out

let gera_alloca out varnome tipo =
let strTipo = (string_of_tipo tipo) in
let align = (obter_align tipo) in
Printf.fprintf out "\t%s = alloca %s, %s\n" (%^varnome) strTipo align;
flush out

let gera_fflush_string out temp =
let strTemp = gera_expressao_str temp in
Printf.fprintf out "\t%s = call %%int @fflush(%%FILE* %%.stdin)\n" strTemp;
flush out

let gera_var_stdin out =
Printf.fprintf out "\t%s\n"

```

```

("%stdin = call %FILE* @fdopen"^
"%int 0, %char* getelementptr inbounds ([2 x %char]* @.r, %int 0, %int 0))"
);
flush out

let rec gera_lista_comandos out comandolist =
match comandolist with
[] -> ()
| c::cls ->
begin match c with
ALLOCA(varnome, tipo) ->
gera_alloca out varnome tipo;
gera_lista_comandos out cls
| JUMP(label) ->
gera_jump out label;
gera_lista_comandos out cls
| LABEL(label) ->
gera_label out label;
gera_lista_comandos out cls
| CJUMP(op, expl,exp2, l1, l2) ->
gera_cjump out expl l1 l2;
gera_lista_comandos out cls
| SWITCH(expressao, labelDefault, tipo, caselist) ->
gera_switch out expressao labelDefault tipo caselist;
gera_lista_comandos out cls
| STORE(expressao, varnome, tipo, global) ->
gera_store out expressao varnome tipo global;
gera_lista_comandos out cls
| LOAD(expressao, varnome, tipo, global) ->
gera_load out expressao varnome tipo global;
gera_lista_comandos out cls
| MOVE(temp, expressao)->
gera_comando_eseq out (MOVE(temp, expressao));
gera_lista_comandos out cls
| RET (tipo, expressao) ->
gera_ret out tipo expressao;
gera_lista_comandos out cls
| FPEXT(tempdouble, tempfloat) ->
gera_fpext out tempdouble tempfloat;
gera_lista_comandos out cls
| FFLUSH_STDIN(temp) ->
gera_fflush_string out temp;
gera_lista_comandos out cls
|_-> gera_lista_comandos out cls
end

(** GERA LISTA DE DEFINES **)

let gera_parametro out (tipo,paramnome) =

```

```

let strTipo = (string_of_tipo tipo) in
Printf.fprintf out "%s %s," strTipo ("%s"^paramnome);
flush out

let rec gera_lista_parametros out paramlist =
match paramlist with
[] -> descola_ptr_canal out
|p::pl ->
gera_parametro out p;
gera_lista_parametros out pl

let gera_define out define =
match define with
DEFINE(tipo, nomemetodo, paramlist, SEQ(comandolist)) ->
let strTipo = string_of_tipo tipo in
Printf.fprintf out "define %s @s(" strTipo nomemetodo; flush out;
(if (paramlist <> [])
then
gera_lista_parametros out paramlist);
Printf.fprintf out ") \n{\n"; flush out;
    gera_var_stdin out;
    gera_lista_comandos out comandolist;
    Printf.fprintf out "}\n\n"; flush out
|_ -> failwith "comando nao e um SEQ"

let rec gera_lista_define out definelist =
match definelist with
[] -> Printf.fprintf out ";fim. ***miniJavaLVM***"; flush out
|df::dfl ->
gera_define out df;
gera_lista_define out dfl

(** GERA VARIÁVEIS GLOBAIS **)

let gera_variavel_global_var out varnome tipo expressao =
let strTipo = (string_of_tipo tipo) in
let strExp = gera_expressao_str expressao in
let align = obter_align tipo in
Printf.fprintf out "@%s = global %s %s, %s\n"
varnome strTipo strExp align; flush out

let gera_variavel_global_string out varnome texto tamanho =
Printf.fprintf out "@%s = private unnamed_addr constant [%d x i8] c\"%s\\00\
", align 1\n"
varnome tamanho texto ; flush out

let gera_variavel_global out varglobal =
match varglobal with
STR (varnome, texto, tamanho) ->
gera_variavel_global_string out varnome texto tamanho
|VAR (varnome,tipo,expressao) ->

```

```

gera_variavel_global_var out varnome tipo expressao

let rec gera_variaveis_globais_100 out globallist =
match globallist with
[] -> Printf.fprintf out "\n"; flush out;
|vg::vgs ->
gera_variavel_global out vg;
gera_variaveis_globais_100 out vgs

let gera_variaveis_globais out globallist =
Printf.fprintf out "%s\n" ";variaveis globais"; flush out;
gera_variaveis_globais_100 out globallist

(** GERA VALORES PRE DETERMINADOS **)

let gera_printf_scanf out =
Printf.fprintf out "%s\n" ";declaracao de funcoes para leitura e escrita";
flush out;
Printf.fprintf out "%s\n" "declare %int @printf(%char*,...)"; flush out;
Printf.fprintf out "%s\n" "declare %int @scanf(i8*, ...)"; flush out;
Printf.fprintf out "%s\n" "declare %FILE* @fdopen(%int, %char*)"; flush out;
Printf.fprintf out "%s\n" "declare %int @fflush(%FILE*)"; flush out;
Printf.fprintf out "\n"; flush out

let gera_str_escrita out =
Printf.fprintf out "%s\n" ";variaveis para leitura e escrita de tipos";
flush out;
(*print int*)
Printf.fprintf out "%s\n"
"@.print.int = private unnamed_addr constant [3 x %char] c \"%d\\00\""; flush
out;
(*print double*)
Printf.fprintf out "%s\n"
"@.print.double = private unnamed_addr constant [4 x %char] c \"%lf\\00\"";
flush out;
(*print char*)
Printf.fprintf out "%s\n"
"@.print.char = private unnamed_addr constant [3 x %char] c \"%c\\00\"";
flush out;
(*print true*)
Printf.fprintf out "%s\n"
"@.print.true = private unnamed_addr constant [5 x %char] c \"true\\00\"";
flush out;
(*print false*)
Printf.fprintf out "%s\n"
"@.print.false = private unnamed_addr constant [6 x %char] c \"false\\00\"";
flush out;
(*print newline*)
Printf.fprintf out "%s\n"

```



## Capítulo 8

# Demonstração do compilador MiniJava para LLVM

Nesta seção mostraremos imagens de como gerar a árvore sintática, intermediária e geração de código.e uma pequena demonstração dos resultados do compilador construído neste trabalho.

### 8.1 Em Funcionamento

Para mostrar o funcionamento do compilador será utilizado como exemplo o código abaixo:

```
import java.util.Scanner;

public class Conversor
{

    public static double celsiusToFahrenheit(double c)
    {
        return (9.0 * c + 160.0) /"5";
    }

    public static double fahrenheitToCelsius(double f)
    {
        return (f - 32.0) / 1.8 ;
    }
}

// ( F      32) / 1,8

public class Principal
{
```

```

public static void main(String[] args)
{
    double c, f;
    char escolha;

    Scanner s = new Scanner(System.in);

    System.out.println("***CONVERSOR DE TEMPERATURA***");
    System.out.print("'C' - CELSIUS; 'F' - Fahrenheit: ");

    escolha = s.nextByte();
    switch(escolha)
    {
        case 'C':
            System.out.print("Digite a temperatura em Celsius: ");

            c = s.nextDouble();
            f = Conversor.celsiusToFahrenheit(c);

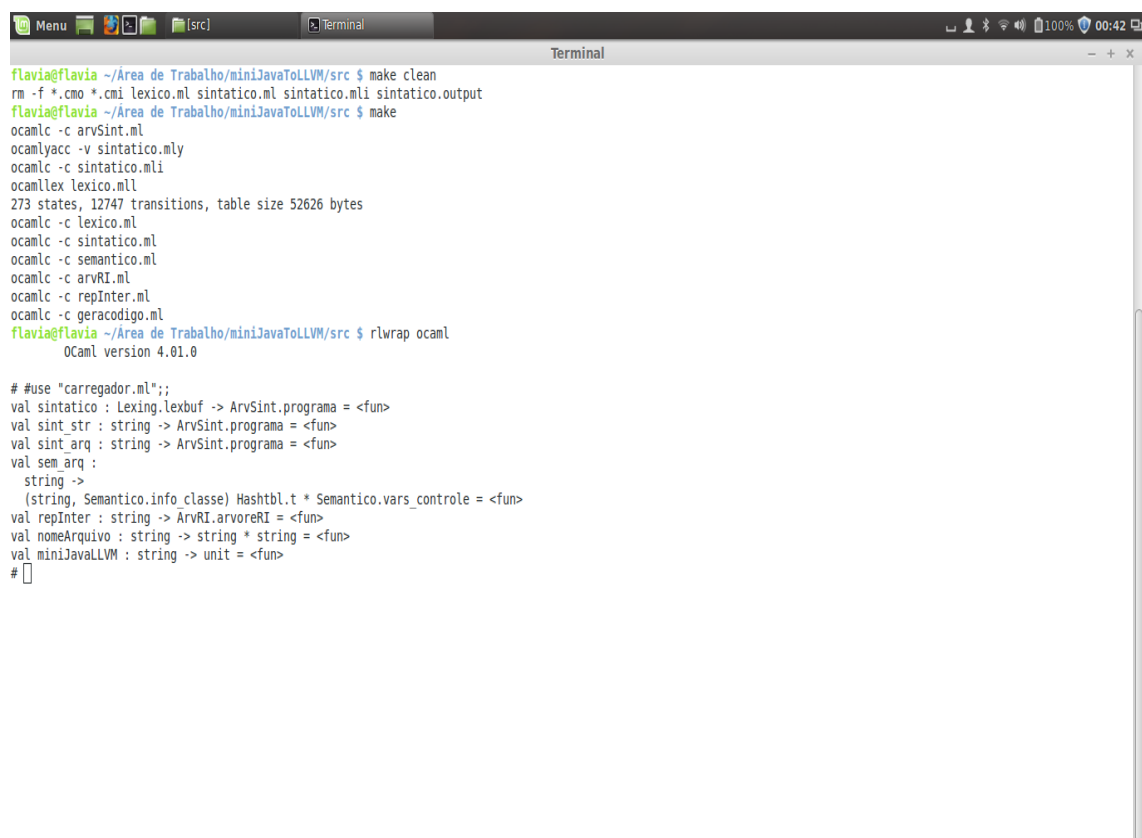
            System.out.print("Temperatura em Fahrenheit :");
            System.out.println(f);
            break;
        case 'F':
            System.out.print("Digite a temperatura em Fahrenheit: ");

            f = s.nextDouble();
            c = Conversor.fahrenheitToCelsius(f);

            System.out.print("Temperatura em Celsius :");
            System.out.println(c);
            break;
        default:
            System.out.println("Escolha inapropriada");
            break;
    }
}
}

```

Abaixo temos um exemplo de como carregar o compilador.



```
flavia@flavia ~/Área de Trabalho/miniJavaToLLVM/src $ make clean
rm -f *.cmo *.cmi lexico.ml sintatico.ml sintatico.mli sintatico.output
flavia@flavia ~/Área de Trabalho/miniJavaToLLVM/src $ make
ocamlc -c arvSint.ml
ocamlyacc -v sintatico.mly
ocamlc -c sintatico.mli
ocamllex lexico.mll
273 states, 12747 transitions, table size 52626 bytes
ocamlc -c lexico.ml
ocamlc -c sintatico.ml
ocamlc -c semantico.ml
ocamlc -c arvRI.ml
ocamlc -c repInter.ml
ocamlc -c geracodigo.ml
flavia@flavia ~/Área de Trabalho/miniJavaToLLVM/src $ rlwrap ocaml
OCaml version 4.01.0

# #use "carregador.ml";;
val sintatico : Lexing.lexbuf -> ArvSint.programa = <fun>
val sint_str : string -> ArvSint.programa = <fun>
val sint_arq : string -> ArvSint.programa = <fun>
val sem_arq :
  string ->
    (string, Semantico.info classe) Hashtbl.t * Semantico.vars_controle = <fun>
val repInter : string -> ArvRI.arvoreRI = <fun>
val nomeArquivo : string -> string * string = <fun>
val miniJavaLLVM : string -> unit = <fun>
#
```

Figura 8.1: Carregamento do Compilador



## 8.2 Árvore Sintática

Abaixo temos uma pequena demonstração da árvore sintática.

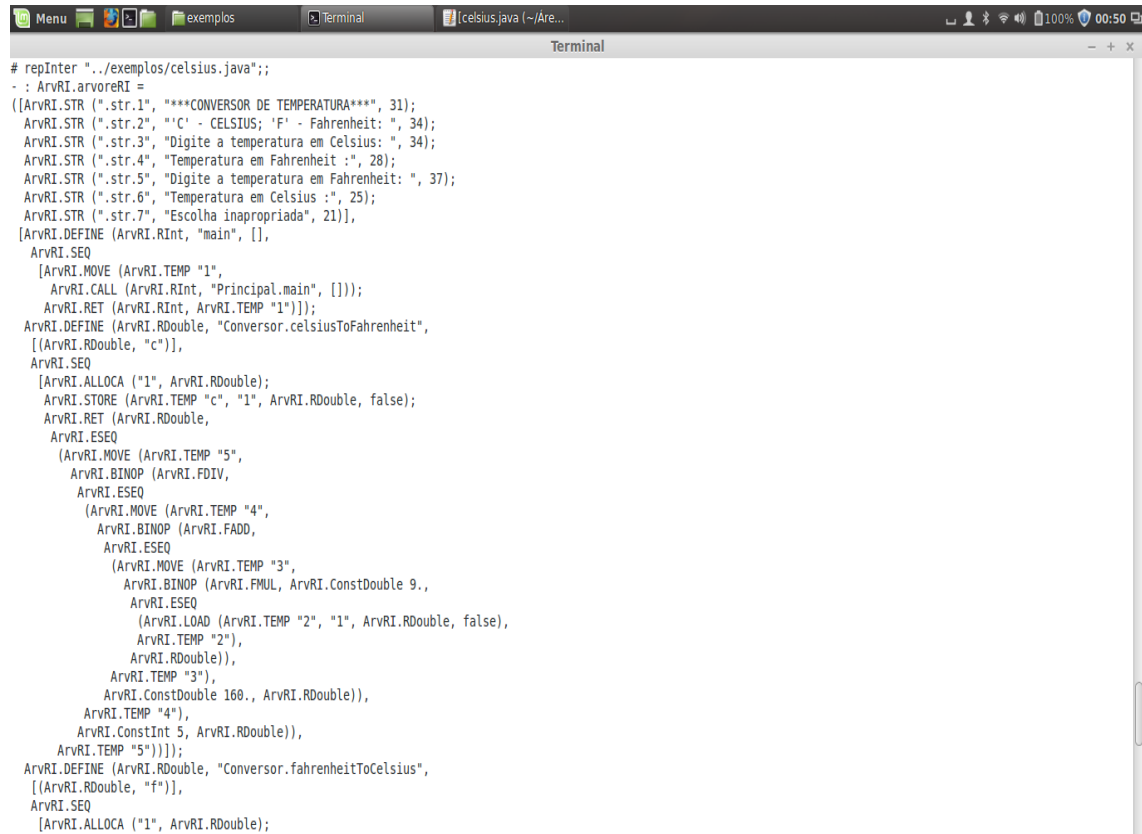


```
# sint_arq "../exemplos/celsius.java";
- : ArvSint.progama =
(true,
[("Conversor", [],
[Metodo
(("celsiusToFahrenheit", TDouble,
[(TDouble, "c",
{ArvSint.Posicao.lin_inicial = 7; col_inicial = 45; lin_final = 7;
col_final = 45})]),
[],
[CmdReturn
(ExpBin (Div,
ExpBin (Soma, ExpBin (Mul, ExpDouble 9., ExpVar "c"),
ExpDouble 160.),
ExpInt 5),
{ArvSint.Posicao.lin_inicial = 9; col_inicial = 7; lin_final = 9;
col_final = 21})]),
{ArvSint.Posicao.lin_inicial = 7; col_inicial = 1; lin_final = 7;
col_final = 46})]),
Metodo
(("fahrenheitToCelsius", TDouble,
[(TDouble, "f",
{ArvSint.Posicao.lin_inicial = 12; col_inicial = 45; lin_final = 12;
col_final = 45})]),
[],
[CmdReturn
(ExpBin (Div, ExpBin (Sub, ExpVar "f", ExpDouble 32.), ExpDouble 1.8),
{ArvSint.Posicao.lin_inicial = 14; col_inicial = 7; lin_final = 14;
col_final = 18})]),
{ArvSint.Posicao.lin_inicial = 12; col_inicial = 1; lin_final = 12;
col_final = 46})]),
{ArvSint.Posicao.lin_inicial = 3; col_inicial = 1; lin_final = 3;
col_final = 28})]),
("Principal", [],
[MetodoMain
([(TString, "args",
{ArvSint.Posicao.lin_inicial = 22; col_inicial = 33; lin_final = 22;
col_final = 36})]),
[("c", TDouble, ExpNone,
{ArvSint.Posicao.lin_inicial = 24; col_inicial = 7; lin_final = 24;
col_final = 7})],
("f", TDouble, ExpNone,
```

Figura 8.2: Árvore Sintática

## 8.3 Árvore Intermediária

Abaixo temos uma pequena demonstração da árvore intermediária.



```
# repInter "../exemplos/celsius.java";
- : ArvRI.arvoreRI =
([ArvRI.STR (".str.1", "****CONVERSOR DE TEMPERATURA****", 31);
ArvRI.STR (".str.2", "'C' - CELSIUS; 'F' - Fahrenheit: ", 34);
ArvRI.STR (".str.3", "Digite a temperatura em Celsius: ", 34);
ArvRI.STR (".str.4", "Temperatura em Fahrenheit :", 28);
ArvRI.STR (".str.5", "Digite a temperatura em Fahrenheit: ", 37);
ArvRI.STR (".str.6", "Temperatura em Celsius :", 25);
ArvRI.STR (".str.7", "Escolha inapropriada", 21)],
[ArvRI.DEFINE (ArvRI.RInt, "main", []),
ArvRI.SEQ
  [ArvRI.MOVE (ArvRI.TEMP "1",
    ArvRI.CALL (ArvRI.RInt, "Principal.main", []));
    ArvRI.RET (ArvRI.RInt, ArvRI.TEMP "1")]);
ArvRI.DEFINE (ArvRI.RDouble, "Conversor.celsiusToFahrenheit",
  [(ArvRI.RDouble, "c")],
ArvRI.SEQ
  [ArvRI.ALLOCA ("1", ArvRI.RDouble);
    ArvRI.STORE (ArvRI.TEMP "c", "1", ArvRI.RDouble, false);
    ArvRI.RET (ArvRI.RDouble,
      ArvRI.ESEQ
        (ArvRI.MOVE (ArvRI.TEMP "5",
          ArvRI.BINOP (ArvRI.FDIV,
            ArvRI.ESEQ
              (ArvRI.MOVE (ArvRI.TEMP "4",
                ArvRI.BINOP (ArvRI.FADD,
                  ArvRI.ESEQ
                    (ArvRI.MOVE (ArvRI.TEMP "3",
                      ArvRI.BINOP (ArvRI.FMUL, ArvRI.ConstDouble 9.,
                        ArvRI.ESEQ
                          (ArvRI.LOAD (ArvRI.TEMP "2", "1", ArvRI.RDouble, false),
                            ArvRI.TEMP "2"),
                            ArvRI.RDouble)),
                        ArvRI.TEMP "3"),
                      ArvRI.ConstDouble 160., ArvRI.RDouble)),
                        ArvRI.TEMP "4"),
                        ArvRI.ConstInt 5, ArvRI.RDouble)),
                        ArvRI.TEMP "5")]);
        ArvRI.DEFINE (ArvRI.RDouble, "Conversor.fahrenheitToCelsius",
          [(ArvRI.RDouble, "f")],
          ArvRI.SEQ
            [ArvRI.ALLOCA ("1", ArvRI.RDouble);
```

Figura 8.3: Árvore Intermediária

## 8.4 Gerador de Código

Abaixo temos a imagem do comando para se gerar um código.



```
flavia@flavia ~/Área de Trabalho/miniJavaToLLVM/src $ make clean
rm -f *.cmo *.cmi lexico.ml sintatico.ml sintatico.mli sintatico.output
flavia@flavia ~/Área de Trabalho/miniJavaToLLVM/src $ make
ocamlc -c arvSint.ml
ocamlc -c arvSint.mli
ocamlc -c sintatico.mli
ocamlc -c sintatico.mli
ocamlc -c lexico.mli
273 states, 12747 transitions, table size 52626 bytes
ocamlc -c lexico.ml
ocamlc -c sintatico.ml
ocamlc -c semantico.ml
ocamlc -c arvRI.ml
ocamlc -c repInter.ml
ocamlc -c geracodigo.ml
flavia@flavia ~/Área de Trabalho/miniJavaToLLVM/src $ rlwrap ocaml
OCaml version 4.01.0

# #use "carregador.ml";;
val sintatico : Lexing.lexbuf -> ArvSint.progama = <fun>
val sint_str : string -> ArvSint.progama = <fun>
val sint_arq : string -> ArvSint.progama = <fun>
val sem_arq :
  string ->
  (string, Semantico.info classe) Hashtbl.t * Semantico.vars_controle = <fun>
val repInter : string -> ArvRI.arvoreRI = <fun>
val nomeArquivo : string -> string * string = <fun>
val miniJavaLLVM : string -> unit = <fun>
# miniJavaLLVM "../exemplos/celsius.java";;
Compilado!!

::Destino em diretorio:../exemplos/
::Nome do arquivo *.ll::celsius.ll

- : unit = ()
#
```

Figura 8.4: Geração de código

Após enviar o comando para gerar o código, obtemos o código gerado para o exemplo e seu make para gerar o seu executável.

O resultado da geração do código está logo abaixo:

```
;definicao de tipos de variaveis
%int = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque
```

```

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais
@.str.1 = private unnamed_addr constant [31 x i8] c"***CONVERSOR DE
    TEMPERATURA***\00", align 1
@.str.2 = private unnamed_addr constant [34 x i8] c"'C' - CELSIUS; 'F' -
    Fahrenheit: \00", align 1
@.str.3 = private unnamed_addr constant [34 x i8] c"Digite a temperatura em
    Celsius: \00", align 1
@.str.4 = private unnamed_addr constant [28 x i8] c"Temperatura em
    Fahrenheit: \00", align 1
@.str.5 = private unnamed_addr constant [37 x i8] c"Digite a temperatura em
    Fahrenheit: \00", align 1
@.str.6 = private unnamed_addr constant [25 x i8] c"Temperatura em Celsius
    : \00", align 1
@.str.7 = private unnamed_addr constant [21 x i8] c"Escolha inapropriada\00"
    , align 1

define %int @main()
{
    %.stdin = call %FILE* @fdopen(%int 0, %char* getelementptr inbounds ([2 x
        %char]* @.r, %int 0, %int 0))
    %1 = call %int @Principal.main()
    ret %int %1
}

define double @Conversor.celsiusToFahrenheit(double %c)
{
    %.stdin = call %FILE* @fdopen(%int 0, %char* getelementptr inbounds ([2 x
        %char]* @.r, %int 0, %int 0))
    %1 = alloca double, align 4
    store double %c, double* %1, align 4
    %2 = load double* %1, align 4
    %3 = fmul double 9.0, %2
    %4 = fadd double %3, 160.0
    %5 = fdiv double %4, 5.0
    ret double %5
}

```

```

}

define double @Conversor.fahrenheitToCelsius(double %f)
{
    %.stdin = call %FILE* @fdopen(%int 0, %char* getelementptr inbounds ([2 x
        %char]* @.r, %int 0, %int 0))
    %1 = alloca double, align 4
    store double %f, double* %1, align 4
    %2 = load double* %1, align 4
    %3 = fsub double %2, 32.0
    %4 = fdiv double %3, 1.8
    ret double %4
}

define %int @Principal.main()
{
    %.stdin = call %FILE* @fdopen(%int 0, %char* getelementptr inbounds ([2 x
        %char]* @.r, %int 0, %int 0))
    %c = alloca double, align 4
    %f = alloca double, align 4
    %escolha = alloca %char, align 1
    %1 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([31 x
        %char]* @.str.1, %int 0, %int 0))
    %2 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    %3 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([34 x
        %char]* @.str.2, %int 0, %int 0))
    %4 = call %int (%char*, ...)* @scanf(%char* getelementptr inbounds ([3 x %
        char]* @.print.char, %int 0, %int 0), %char* %escolha)
    %5 = call %int @fflush(%FILE* %.stdin)
    %6 = load %char* %escolha, align 1
    switch %char %6, label %SWITCH1_DEFAULT
    [
        %char 67, label %SWITCH1_CASE1
        %char 70, label %SWITCH1_CASE2
    ]
    SWITCH1_CASE1:
    %7 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([34 x
        %char]* @.str.3, %int 0, %int 0))
    %8 = call %int (%char*, ...)* @scanf(%char* getelementptr inbounds ([4 x %
        char]* @.print.double, %int 0, %int 0), double* %c)
    %9 = load double* %c, align 4
    %10 = call double @Conversor.celsiusToFahrenheit(double %9)
    store double %10, double* %f, align 4
    %11 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([28
        x %char]* @.str.4, %int 0, %int 0))
    %12 = load double* %f, align 4
    %13 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([4 x
        %char]* @.print.double, %int 0, %int 0), double %12)
    %14 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
        @.print.newline, %int 0, %int 0))

```

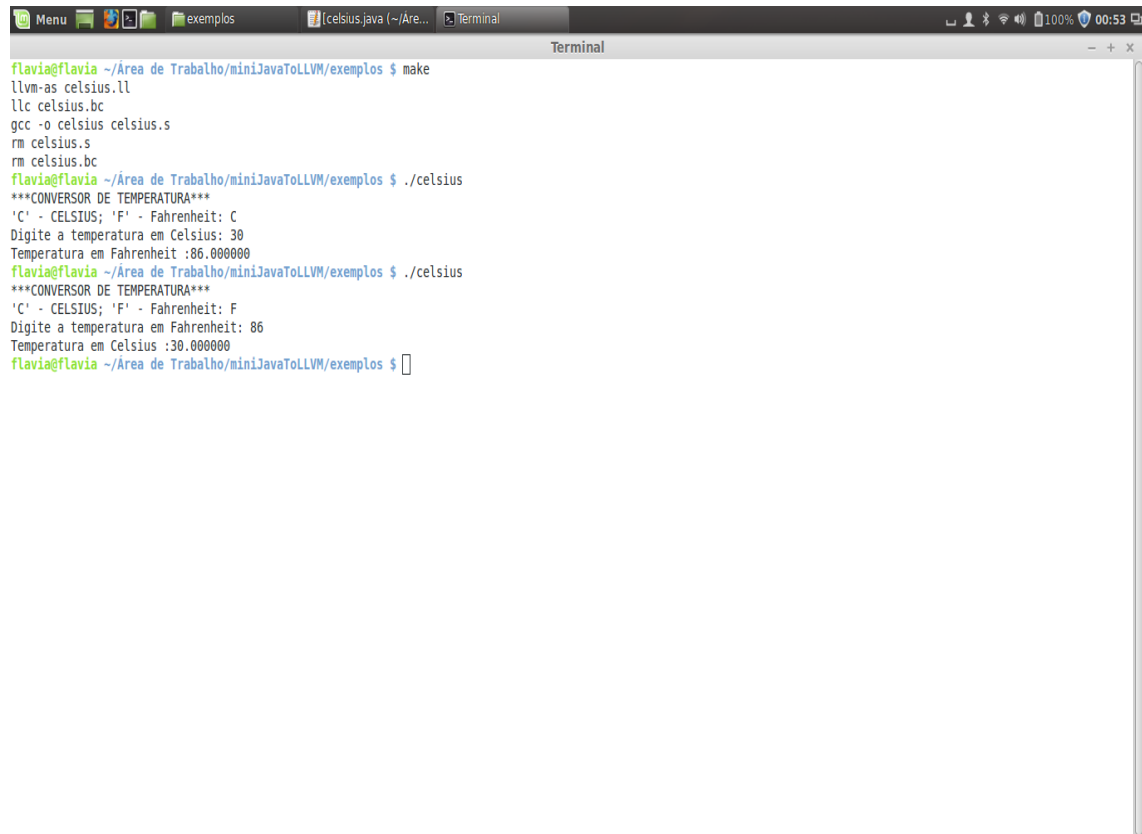
```

br label %END_SWITCH1
SWITCH1_CASE2:
%15 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([37
  x %char]* @.str.5, %int 0, %int 0))
%16 = call %int (%char*, ...)* @scanf(%char* getelementptr inbounds ([4 x
  %char]* @.print.double, %int 0, %int 0), double* %f)
%17 = load double* %f, align 4
%18 = call double @Conversor.fahrenheitToCelsius(double %17)
store double %18, double* %c, align 4
%19 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([25
  x %char]* @.str.6, %int 0, %int 0))
%20 = load double* %c, align 4
%21 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([4 x
  %char]* @.print.double, %int 0, %int 0), double %20)
%22 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
  @.print.newline, %int 0, %int 0))
br label %END_SWITCH1
SWITCH1_DEFAULT:
%23 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([21
  x %char]* @.str.7, %int 0, %int 0))
%24 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
  @.print.newline, %int 0, %int 0))
br label %END_SWITCH1
END_SWITCH1:
ret %int 0
}
;fim. ***miniJavaLLVM***

```

## 8.5 Execução

Abaixo temos a execução do código gerado.



```
flavia@flavia ~/Área de Trabalho/miniJavaToLLVM/exemplos $ make
llvm-as celsius.ll
llc celsius.bc
gcc -o celsius celsius.s
rm celsius.s
rm celsius.bc
flavia@flavia ~/Área de Trabalho/miniJavaToLLVM/exemplos $ ./celsius
***CONVERSOR DE TEMPERATURA***
'C' - CELSIUS; 'F' - Fahrenheit: C
Digite a temperatura em Celsius: 30
Temperatura em Fahrenheit :86.000000
flavia@flavia ~/Área de Trabalho/miniJavaToLLVM/exemplos $ ./celsius
***CONVERSOR DE TEMPERATURA***
'C' - CELSIUS; 'F' - Fahrenheit: F
Digite a temperatura em Fahrenheit: 86
Temperatura em Celsius :30.000000
flavia@flavia ~/Área de Trabalho/miniJavaToLLVM/exemplos $
```

Figura 8.5: Execução do código

# Capítulo 9

## Referências

INSTALLION. To install rlwrap just follow these instructions. 2014. Acessado em: 11 outubro 2014. Disponível em:

<<http://www.installion.co.uk/ubuntu/raring/universe/r/rlwrap/install.html>>.

OCAML. Install ocaml. 2015. Acessado em: 23 fevereiro 2015. Disponível em:

<<http://ocaml.org/docs/install.htm>>

OCAML isnttruction. 2014 Disponível em:

<<http://caml.inria.fr/pub/docs/manual-ocaml-4.02/>>

LLVM. Llvm language reference manual. 2015. Acessado em: 23 fevereiro 2015.

Disponível em: <<http://llvm.org/docs/LangRef.html>>.

PERVASIVES 2015 .Disponível em:

<<http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html>>

RECORDS and VARIANTS 2015. Disponivel em:

<<https://realworldocaml.org/v1/en/html/records.html>>

Regular expression. 2015. Disponível em:

<[http://en.wikipedia.org/wiki/Regular\\_expressionStandards](http://en.wikipedia.org/wiki/Regular_expressionStandards) >

RELATÓRIO LLVM - Grupo: Sérgio e Camila RELATÓRIO MiniJava - Grupo: João Pedro e Felipe Milken Disponíveis em :

<[https://groups.google.com/forum/?hl=pt-BR!topic/comp\\_uvu/](https://groups.google.com/forum/?hl=pt-BR!topic/comp_uvu/)>



# Capítulo 10

## Exercícios

### 10.1 Tarefa 1

#### 10.1.1 Listagem 1 - Módulo Mínimo

Código em java:

```
class main
{
    public static void main(String[] args){
    }
}
```

Código em LLVM:

```
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
```

```

@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @Principal.main()
    ret %int %1
}

define %int @Principal.main()
{
    ret %int 0
}

;fim. ***miniJavaLLVM***

```

## 10.1.2 Listagem 2 - Declaração de uma variável

Código em java:

```

class Main
{
    public static void main(String[] args){
        int n;
    }
}

```

Código em LLVM:

```

;definicao de tipos de variaveis
%int = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos

```

```

@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @Main.main()
    ret %int %1
}

define %int @Main.main()
{
    %n = alloca %int, align 4
    ret %int 0
}

;fim. ***miniJavaLLVM***

```

### 10.1.3 Listagem 3 - Atribuição de um inteiro a uma variável

Código em java:

```

public class exercicio3
{
    public static int main(String[] args)
    {
        int n;
        n = 1;
        return n;
    }
}

```

Código em LLVM:

```

;definicao de tipos de variaveis
%int = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo

```

```

%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @Main.main()
    ret %int %1
}

define %int @Main.main()
{
    %n = alloca %int, align 4
    store %int 1, %int* %n, align 4
    ret %int 0
}

;fim. ***miniJavaLLVM***

```

#### 10.1.4 Listagem 4 - Atribuição de uma soma de inteiro a uma variável

Código em java:

```

class Main
{
    public static void main(String[] args) {
        int n;
        n = 1 + 2;
    }
}

```

## Código em LLVM:

```
;definicao de tipos de variaveis
%int  = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @Main.main()
    ret %int %1
}

define %int @Main.main()
{
    %n = alloca %int, align 4
    %1 = add %int 1, 2
    store %int %1, %int* %n, align 4
    ret %int 0
}

;fim. ***miniJavaLLVM***
```

### 10.1.5 Listagem 5 - Inclusão do comando de impressão

#### Código em java:

```
public class exercicio5
```

```

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

```

## Código em LLVM:

```

;definicao de tipos de variaveis
%int = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @exercicio5.main()
    ret %int %1
}

define %int @exercicio5.main()
{
    %n = alloca %int, align 4
    store %int 2, %int* %n, align 4
    %1 = load %int* %n, align 4
    %2 = call %int (%char*, ...) @printf(%char* getelementptr inbounds ([3 x
        %char]* @.print.int, %int 0, %int 0), %int %1)

```

```

    %3 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    ret %int 0
}

;fim. ***miniJavaLLVM***

```

### 10.1.6 Listagem 6 - Atribuição de uma subtração de inteiros a um variável

Código em java:

```

class Main
{
    public static void main(String[] args) {
        int n;

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

```

Código em LLVM:

```

;definicao de tipos de variaveis
%int = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo

```

```

@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @Main.main()
    ret %int %1
}

define %int @Main.main()
{
    %n = alloca %int, align 4
    %1 = sub %int 1, 2
    store %int %1, %int* %n, align 4
    %2 = load %int* %n, align 4
    %3 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x %char]* @.print.int, %int 0, %int 0), %int %2)
    %4 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @.print.newline, %int 0, %int 0))
    ret %int 0
}

;fim. ***miniJavaLLVM***

```

### 10.1.7 Listagem 7 - Inclusão do comando condicional

Código em java:

```

public class exercicio7
{
    public static void main(String[] args)
    {
        int n;
        n = 1;

        if( n == 1)
        {
            System.out.println(n);
        }

    }
}

```

Código em LLVM:



```

;definicao de tipos de variaveis
%int = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @exercicio7.main()
    ret %int %1
}

define %int @exercicio7.main()
{
    %n = alloca %int, align 4
    store %int 1, %int* %n, align 4
    %1 = load %int* %n, align 4
    %2 = icmp eq %int %1, 1
    br %boolean %2, label %IF1, label %END_IF1
IF1:
    %3 = load %int* %n, align 4
    %4 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x %char]* @.print.int, %int 0, %int 0), %int %3)
    %5 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @.print.newline, %int 0, %int 0))
    END_IF1:
    ret %int 0
}

;fim. ***miniJavaLLVM***

```

## 10.1.8 Listagem 8 - Inclusão do comando condicional com parte senão

Código em java:

```
class Main
{
    public static void main(String[] args) {
        int n;

        n = 1;
        if(n==1) {
            System.out.println(n);
        }
        else {
            System.out.println(0);
        }
    }
}
```

Código em LLVM:

```
;definicao de tipos de variaveis
%int  = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
```

```

{
    %1 = call %int @Main.main()
    ret %int %1
}

define %int @Main.main()
{
    %n = alloca %int, align 4
    store %int 1, %int* %n, align 4
    %1 = load %int* %n, align 4
    %2 = icmp eq %int %1, 1
    br %boolean %2, label %IF1, label %ELSE1
IF1:
    %3 = load %int* %n, align 4
    %4 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
        %char]* @.print.int, %int 0, %int 0), %int %3)
    %5 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    br label %END_IF1
ELSE1:
    %6 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
        %char]* @.print.int, %int 0, %int 0), %int 0)
    %7 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    br label %END_IF1
END_IF1:
    ret %int 0
}

;fim. ***miniJavaLLVM***

```

### 10.1.9 Listagem 9 - Atribuição de duas operações aritméticas sobre inteiros a uma variável

Código em java:

```

public class exercicio9
{
    public static void main(String[] args)
    {
        int n;
        n = 1 + 1 / 2;

        if( n == 1)
        {
            System.out.println(n);
        }
    }
}

```

```

    else
    {
        System.out.println(0);
    }
}
}

```

## Código em LLVM:

```

;definicao de tipos de variaveis
%int  = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @exercicio9.main()
    ret %int %1
}

define %int @exercicio9.main()
{
    %n = alloca %int, align 4
    %1 = div %int 1, 2
    %2 = add %int 1, %1
    store %int %2, %int* %n, align 4
    %3 = load %int* %n, align 4
    %4 = icmp eq %int %3, 1
    br %boolean %4, label %IF1, label %ELSE1
IF1:

```

```

%5 = load %int* %n, align 4
%6 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
    %char]* @.print.int, %int 0, %int 0), %int %5)
%7 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
    .print.newline, %int 0, %int 0))
br label %END_IF1
ELSE1:
%8 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
    %char]* @.print.int, %int 0, %int 0), %int 0)
%9 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
    .print.newline, %int 0, %int 0))
br label %END_IF1
END_IF1:
ret %int 0
}

;fim. ***miniJavaLLVM***

```

### 10.1.10 Listagem 10 - Atribuição de duas variáveis inteiras

Código em java:

```

class Main {
    int n, m;

    public static void main(String[] args) {
        n = 1;
        m = 2;
        if(n==m) {
            System.out.println(n);
        }
        else {
            System.out.println(0);
        }
    }
}

```

Código em LLVM:

```

;definicao de tipos de variaveis
%int  = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

```

```

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @Main.main()
    ret %int %1
}

define %int @Main.main()
{
    %n = alloca %int, align 4
    %m = alloca %int, align 4
    store %int 1, %int* %n, align 4
    store %int 2, %int* %m, align 4
    %1 = load %int* %n, align 4
    %2 = load %int* %m, align 4
    %3 = icmp eq %int %1, %2
    br %boolean %3, label %IF1, label %ELSE1
IF1:
    %4 = load %int* %n, align 4
    %5 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
        %char]* @.print.int, %int 0, %int 0), %int %4)
    %6 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    br label %END_IF1
ELSE1:
    %7 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
        %char]* @.print.int, %int 0, %int 0), %int 0)
    %8 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    br label %END_IF1
END_IF1:
    ret %int 0
}

```

```
;fim. ***miniJavaLLVM***
```

### 10.1.11 Listagem 11 - Introdução do comando de repetição enquanto

Código em java:

```
public class exercicio11
{
    public static void main(String[] args)
    {
        int n, m, x;

        n = 1;
        m = 2;
        x = 5;

        while (x > n)
        {
            if (m == n)
            {
                System.out.println(n);
            }
            else
            {
                System.out.println(0);
            }

            x = x - 1;
        }
    }
}
```

Código em LLVM:

```
;definicao de tipos de variaveis
%int = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*, ...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
```

```

declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @exerciciol1.main()
    ret %int %1
}

define %int @exerciciol1.main()
{
    %n = alloca %int, align 4
    %m = alloca %int, align 4
    %x = alloca %int, align 4
    store %int 1, %int* %n, align 4
    store %int 2, %int* %m, align 4
    store %int 5, %int* %x, align 4
    br label %COND_WHILE1
COND_WHILE1:
    %1 = load %int* %x, align 4
    %2 = load %int* %n, align 4
    %3 = icmp sgt %int %1, %2
    br %boolean %3, label %WHILE1, label %END_WHILE1
WHILE1:
    %4 = load %int* %m, align 4
    %5 = load %int* %n, align 4
    %6 = icmp eq %int %4, %5
    br %boolean %6, label %IF1, label %ELSE1
IF1:
    %7 = load %int* %n, align 4
    %8 = call %int (%char*, ...)@printf(%char* getelementptr inbounds ([3 x
        %char]* @.print.int, %int 0, %int 0), %int %7)
    %9 = call %int (%char*, ...)@printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    br label %END_IF1
ELSE1:
    %10 = call %int (%char*, ...)@printf(%char* getelementptr inbounds ([3 x
        %char]* @.print.int, %int 0, %int 0), %int 0)
    %11 = call %int (%char*, ...)@printf(%char* getelementptr ([2 x %char]*
        @.print.newline, %int 0, %int 0))

```



```

    br label %END_IF1
END_IF1:
%12 = load %int* %x, align 4
%13 = sub %int %12, 1
store %int %13, %int* %x, align 4
br label %COND_WHILE1
END_WHILE1:
ret %int 0
}

;fim. ***miniJavaLLVM***

```

### 10.1.12 Listagem 12 - Comando condicional aninhado em um comando de repetição

Código em java:

```

class Main {
    private static int n, m, x;

    public static void main(String[] args) {
        n = 1;
        m = 2;
        x = 5;
        while (x > n) {
            if (n==m) {
                System.out.println(n);
            }
            else {
                System.out.println(0);
            }
            x = x - 1;
        }
    }
}

```

Código em LLVM:

```

;definicao de tipos de variaveis
%int  = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita

```

```

declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais

define %int @main()
{
    %1 = call %int @Main.main()
    ret %int %1
}

define %int @Main.main()
{
    %n = alloca %int, align 4
    %m = alloca %int, align 4
    %x = alloca %int, align 4
    store %int 1, %int* %n, align 4
    store %int 2, %int* %m, align 4
    store %int 5, %int* %x, align 4
    br label %COND_WHILE1
COND_WHILE1:
    %1 = load %int* %x, align 4
    %2 = load %int* %n, align 4
    %3 = icmp sgt %int %1, %2
    br %boolean %3, label %WHILE1, label %END_WHILE1
WHILE1:
    %4 = load %int* %n, align 4
    %5 = load %int* %m, align 4
    %6 = icmp eq %int %4, %5
    br %boolean %6, label %IF1, label %ELSE1
IF1:
    %7 = load %int* %n, align 4
    %8 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
        %char]* @.print.int, %int 0, %int 0), %int %7)
    %9 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    br label %END_IF1
ELSE1:

```

```

%10 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
    %char]* @.print.int, %int 0, %int 0), %int 0)
%11 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
br label %END_IF1
END_IF1:
%12 = load %int* %x, align 4
%13 = sub %int %12, 1
store %int %13, %int* %x, align 4
br label %COND_WHILE1
END_WHILE1:
ret %int 0
}

;fim. ***miniJavaLLVM***

```

### 10.1.13 Listagem 13 - Converte graus Celsius para Fahrenheit

Código em java:

```

import java.util.Scanner;
public class exercicio13
{
    public static void main(String[] args)
    {
        double c, f;
        Scanner t = new Scanner(System.in);

        System.out.println("Convers o de graus Celsius para Fahrenheit");
        System.out.println("Digite a temperatura em graus Celsius:");

        c = t.nextDouble();
        f = (9.0 * c + 160.0) / 5.0;

        System.out.println("A temperatura ");
        System.out.println( c );
        System.out.println("C convertida para Fahrenheit    ");
        System.out.println( f );
        System.out.println("F" );
    }
}

```

Código em LLVM:

```

;definicao de tipos de variaveis
%int = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais
@.str.1 = private unnamed_addr constant [44 x i8] c"Convers o de graus
    Celsius para Fahrenheit\00", align 1
@.str.2 = private unnamed_addr constant [39 x i8] c"Digite a temperatura em
    graus Celsius:\00", align 1
@.str.3 = private unnamed_addr constant [15 x i8] c"A temperatura \00",
    align 1
@.str.4 = private unnamed_addr constant [33 x i8] c"C convertida para
    Fahrenheit \00", align 1
@.str.5 = private unnamed_addr constant [2 x i8] c"F\00", align 1

define %int @main()
{
    %1 = call %int @exercicio13.main()
    ret %int %1
}

define %int @exercicio13.main()
{
    %c = alloca double, align 4
    %f = alloca double, align 4
    %1 = call %int (%char*,...)* @printf(%char* getelementptr inbounds ([44 x
        %char]* @.str.1, %int 0, %int 0))
    %2 = call %int (%char*,...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    %3 = call %int (%char*,...)* @printf(%char* getelementptr inbounds ([39 x
        %char]* @.str.2, %int 0, %int 0))

```

```

%4 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
.print.newline, %int 0, %int 0))
%5 = call %int (%char*, ...)* @scanf(%char* getelementptr inbounds ([4 x %
char]* @.print.double, %int 0, %int 0), double* %c)
%6 = load double* %c, align 4
%7 = fmul double 9.0, %6
%8 = fadd double %7, 160.0
%9 = fdiv double %8, 5.0
store double %9, double* %f, align 4
%10 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([15
x %char]* @.str.3, %int 0, %int 0))
%11 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
@.print.newline, %int 0, %int 0))
%12 = load double* %c, align 4
%13 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([4 x
%char]* @.print.double, %int 0, %int 0), double %12)
%14 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
@.print.newline, %int 0, %int 0))
%15 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([33
x %char]* @.str.4, %int 0, %int 0))
%16 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
@.print.newline, %int 0, %int 0))
%17 = load double* %f, align 4
%18 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([4 x
%char]* @.print.double, %int 0, %int 0), double %17)
%19 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
@.print.newline, %int 0, %int 0))
%20 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([2 x
%char]* @.str.5, %int 0, %int 0))
%21 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
@.print.newline, %int 0, %int 0))
ret %int 0
}
;fim. ***miniJavaLLVM***

```

### 10.1.14 Listagem 14 - Ler dois inteiros e decide qual é maior

Código em java:

```

/* Função: EScrever um algoritmo que leia dois valores inteiros distintos
e informe qual o maior
*/

import java.util.Scanner;

class Main {

    private static int num1, num2;

```

```

public static void main(String[] args) {

    Scanner s = new Scanner(System.in);

    System.out.println("Digite o primeiro n mero: ");
    num1 = s.nextInt();

    System.out.println("Digite o segundo n mero: ");
    num2 = s.nextInt();

    if(num1 > num2) {
        System.out.println("O primeiro n mero");
        System.out.println(num1);
        System.out.println("    maior que o segundo ");
        System.out.println(num2);
    }
    else {
        System.out.println("O segundo n mero");
        System.out.println(num2 );
        System.out.println("    maior que o primeiro ");
        System.out.println(num1);
    }
}
}

```

## Código em LLVM:

```

;definicao de tipos de variaveis
%int  = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"

```

```

@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais
@.str.1 = private unnamed_addr constant [28 x i8] c"Digite o primeiro
    n mero: \00", align 1
@.str.2 = private unnamed_addr constant [27 x i8] c"Digite o segundo n mero
    : \00", align 1
@.str.3 = private unnamed_addr constant [19 x i8] c"O primeiro n mero\00",
    align 1
@.str.4 = private unnamed_addr constant [24 x i8] c"    maior que o segundo
    \00", align 1
@.str.5 = private unnamed_addr constant [18 x i8] c"O segundo n mero\00",
    align 1
@.str.6 = private unnamed_addr constant [25 x i8] c"    maior que o primeiro
    \00", align 1
@Main.null.num1 = global %int 0, align 4
@Main.null.num2 = global %int 0, align 4

define %int @main()
{
    %1 = call %int @Main.main()
    ret %int %1
}

define %int @Main.main()
{
    %1 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([28 x
        %char]* @.str.1, %int 0, %int 0))
    %2 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    %3 = call %int (%char*, ...)* @scanf(%char* getelementptr inbounds ([3 x %
        char]* @.print.int, %int 0, %int 0), %int* @Main.null.num1)
    %4 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([27 x
        %char]* @.str.2, %int 0, %int 0))
    %5 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    %6 = call %int (%char*, ...)* @scanf(%char* getelementptr inbounds ([3 x %
        char]* @.print.int, %int 0, %int 0), %int* @Main.null.num2)
    %7 = load %int* @Main.null.num1, align 4
    %8 = load %int* @Main.null.num2, align 4
    %9 = icmp sgt %int %7, %8
    br %boolean %9, label %IF1, label %ELSE1
IF1:
    %10 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([19
        x %char]* @.str.3, %int 0, %int 0))
    %11 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
        @.print.newline, %int 0, %int 0))
    %12 = load %int* @Main.null.num1, align 4

```

```

%13 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
    %char]* @.print.int, %int 0, %int 0), %int %12)
%14 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
%15 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([24
    x %char]* @.str.4, %int 0, %int 0))
%16 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
%17 = load %int* @Main.null.num2, align 4
%18 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
    %char]* @.print.int, %int 0, %int 0), %int %17)
%19 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
br label %END_IF1
ELSE1:
%20 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([18
    x %char]* @.str.5, %int 0, %int 0))
%21 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
%22 = load %int* @Main.null.num2, align 4
%23 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
    %char]* @.print.int, %int 0, %int 0), %int %22)
%24 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
%25 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([25
    x %char]* @.str.6, %int 0, %int 0))
%26 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
%27 = load %int* @Main.null.num1, align 4
%28 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
    %char]* @.print.int, %int 0, %int 0), %int %27)
%29 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
br label %END_IF1
END_IF1:
ret %int 0
}
;fim. ***miniJavaLLVM***

```

### 10.1.15 Listagem 15 - Lê um número e verifica se ele está entre 100 e 200

Código em java:

```

import java.util.Scanner;
public class exercicio15
{

```



```

public static void main(String[] args)
{
    int numero;
    Scanner n = new Scanner(System.in);

    System.out.println("Digite um numero:");

    numero = n.nextInt();

    if(numero >= 100 && numero <= 200)
    {
        System.out.println("O n mero est no intervalo entre 100 e 200" );
    }
    else
    {
        System.out.println("O n mero n o est no intervalo entre 100 e 200"
        );
    }
}
}

```

## Código em LLVM:

```

;definicao de tipos de variaveis
%int = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais
@.str.1 = private unnamed_addr constant [18 x i8] c"Digite um numero:\00",
    align 1

```

```

@.str.2 = private unnamed_addr constant [45 x i8] c"O n mero est no
    intervalo entre 100 e 200\00", align 1
@.str.3 = private unnamed_addr constant [50 x i8] c"O n mero n o est no
    intervalo entre 100 e 200\00", align 1

define %int @main()
{
    %1 = call %int @exercicio15.main()
    ret %int %1
}

define %int @exercicio15.main()
{
    %numero = alloca %int, align 4
    %1 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([18 x
        %char]* @.str.1, %int 0, %int 0))
    %2 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    %3 = call %int (%char*, ...)* @scanf(%char* getelementptr inbounds ([3 x %
        char]* @.print.int, %int 0, %int 0), %int* %numero)
    %4 = load %int* %numero, align 4
    %5 = icmp sge %int %4, 100
    %6 = icmp sle %int %4, 200
    %7 = and %boolean %5, %6
    br %boolean %7, label %IF1, label %ELSE1
IF1:
    %8 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([45 x
        %char]* @.str.2, %int 0, %int 0))
    %9 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    br label %END_IF1
ELSE1:
    %10 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([50
        x %char]* @.str.3, %int 0, %int 0))
    %11 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
        @.print.newline, %int 0, %int 0))
    br label %END_IF1
END_IF1:
    ret %int 0
}

;fim. ***miniJavaLLVM***

```

### 10.1.16 Listagem 16 - Lê um número e informa quais estão entre 10 e 150

Código em java:

```

/* Funcao: Ler 5 numeros e ao final informar quantos numeros(s) estao no
   intervalo de 10 (inclusive) e 150 (inclusive)
*/

import java.util.Scanner;

class Main {
    int x, num, intervalo;

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        for(x=0; x<5; x++) {
            System.out.println("Digite um numero ");
            num = s.nextInt();

            if(num >= 10) {
                if(num <= 150) {
                    intervalo = intervalo + 1;
                }
            }
        }

        System.out.println("Ao total, foram digitados " + intervalo + "numeros
            no intervalo entre 10 e 150");

    }
}

```

## Código em LLVM:

```

%int = type i32
%char = type i8
%bool = type i1

@.print.insert = private unnamed_addr constant [16 x %char] c"Digite numero:
    \00";
@.print.answer = private unnamed_addr constant [66 x %char] c"Ao total,
    foram digitados %d numeros no intervalo entre 10 e 150\0A\00"
@.scanf.msg = internal constant [3 x %char] c"%d\00";

declare %int @printf(%char*, ...)
declare %int @scanf(%char*, ...)

define i32 @main()
{
    ;int x
    %x = alloca %int, align 4
    ;int num

```

```

%num = alloca %int, align 4
;int intervalo
%intervalo = alloca %int, align 4

store %int 0, %int* %intervalo ;intervalo = 0

;instrucao for(x=0;x<5;x++)
store %int 0, %int* %x ;x = 0
br label %for.condition

for.condition:
%1 = load %int* %x, align 4 ;recebe valor de x
%2 = icmp ult %int %1, 5 ;testa se x < 5
br %bool %2, label %for.inner, label %for.end

for.inner:
;print "Digite numero"
call %int (%char*, ...)@printf (%char* getelementptr ([16 x %char]* @.
    print.insert, %int 0, %int 0))
;scanf num
call %int (%char*, ...)@scanf (%char* getelementptr ([3 x %char]* @.
    scanf.msg, %int 0, %int 0), %int* %num)

%5 = load %int* %num, align 4 ;ler valor de num

;testa se num >= 10
%6 = icmp sge %int %5, 10 ;sge = signal greater or equal
br %bool %6, label %if.greater10, label %if.endif

if.greater10:
;testa se num <= 150
%7 = icmp sle %int %5, 150 ;sge = signal less or equal
br %bool %7, label %if.less150, label %if.endif

if.less150:
;incrementa intervalo
%8 = load %int* %intervalo, align 4 ;le intervalo
%9 = add %int %8, 1 ;intervalo++
store %int %9, %int* %intervalo

br label %if.endif

if.endif:
%10 = add %int %1, 1 ;x++
store %int %10, %int* %x
br label %for.condition

for.end:
;le variavel intervalo
%11 = load %int* %intervalo, align 4
;imprime resposta

```

```

    call %int (%char*, ...)* @printf (%char* getelementptr ([66 x %char]* @.
        print.answer, %int 0, %int 0), %int %11)

    ret %int 0;
}

```

### 10.1.17 Listagem 17 - Lê strings e caracteres

Código em java:

```

import java.util.Scanner;

public class exercicio17
{
    public void micro05(String args[])
    {
        int x, h = 0, m = 0;

        Scanner s = new Scanner(System.in);

        String nome = new String();
        String sexo = new String();

        for(x = 0; x < 5; x++)
        {
            System.out.println("Digite o nome:");

            nome = s.next();

            System.out.println("Digite o sexo (H- Home ou M- Mulher):");

            sexo = s.next();

            switch(sexo)
            {
                case "H": h++; break;
                case "M": m++; break;
                default: System.out.println("Sexo invalido, so pode ser H ou M!");
            }
        }

        System.out.println("Foram inseridos" + h + "homens.");
        System.out.println("Foram inseridos" + m + "mulheres.");
    }
}

```

## Código em LLVM:

```
@.str = private unnamed_addr constant [16 x i8] c"Digite o nome: \00", align 1
@.str1 = private unnamed_addr constant [8 x i8] c"\0A%[^\\0A]s\00", align 1
@.str2 = private unnamed_addr constant [26 x i8] c"H - Homem ou M - Mulher: \00", align 1
@.str3 = private unnamed_addr constant [8 x i8] c"\0A%[^\\0A]s\00", align 1
@.str4 = private unnamed_addr constant [26 x i8] c"Sexo so pode ser H ou M !\0A\00", align 1
@.str5 = private unnamed_addr constant [29 x i8] c"Foram inseridos, %d, Homens\0A\00", align 1
@.str6 = private unnamed_addr constant [31 x i8] c"Foram inseridos, %d, Mulheres\0A\00", align 1

; Funcoes de leitura e escrita
declare i32 @printf(i8*, ...)
declare i32 @scanf(i8*, ...)

;Le o nome e o sexo da pessoa inserida
;Verifica se sao homens ou mulheres incrementando seus contadores
;A opcao sexo so e aceita com caracteres maiusculos

define i32 @main()
{
    %1 = alloca i32, align 4
    %x = alloca i32, align 4
    %h = alloca i32, align 4
    %m = alloca i32, align 4
    %nome = alloca [30 x i8], align 1; Aloca espaco para % nome string 8bit
    %sexo = alloca i8, align 1; char 8bit

    store i32 0, i32* %1 ; recebe o valor 0
    store i32 0, i32* %h, align 4 ; recebe o valor 0
    store i32 0, i32* %m, align 4 ; recebe o valor 0
    store i32 0, i32* %x, align 4 ; recebe o valor 0

    br label %2; Vai para a istrucao em %2
; <label>:2 ; preds = %22, %0
    %3 = load i32* %x, align 4; %3 recebe o conteudo de %x
    %4 = icmp slt i32 %3, 5; Verifica se o conteudo de %3 e menor que 5
    br i1 %4, label %5, label %24; Se %3 < 5 entao ele pulapra instrucao
        contida em %5, senao pula para a instrucao contida em %2
; <label>;5 ; preds = %2

    %6 = call i32 @printf(i8*, ...) @printf(i8* getelementptr ([16 x i8]* @.str, i32
        0, i32 0)) ; mensagem 0
    %7 = getelementptr [30 x i8]* %nome, i32 0, i32 0; leitura do nome
```

```

%8 = call i32 (i8*, ...)* @scanf(i8* getelementptr ([8 x i8]* @.str1, i32
    0, i32 0), i8* %7); Chama a funcao scanf para a leitura do nome e
    carrega o mesmo no registrador %8
%9 = call i32 (i8*, ...)* @printf(i8* getelementptr([26 x i8]* @.str2, i32
    0, i32 0)) ; mensagem 2

%10 = call i32 (i8*, ...)* @scanf(i8* getelementptr ([8 x i8]* @.str3, i32
    0, i32 0), i8* %sexo) ; Chama a funcao scanf para a leitura do se sexo
    e carrega no registrador %10
%11 = load i8* %sexo, align 1 ;%11 recebe o conteudo de %sexo
%12 = sext i8 %11 to i32 ;%12 pega o valor de %11 para fazer a escolha
    do label a ser executado no switch, 'H', 'M' ou default.

switch i32 %12, label %19[;caso 'M', pula para a istrucao %19
i32 72, label %13 ; caso 'H', pula para a istrucao %13
i32 77, label %16 ;Default, pula para a istrucao %16
]
; <label>:13 ;preds = %5
%14 = load i32* %h, align 4; %14 recebe o conteudo %h
%15 = add nsw i32 %14, 1; incrementa %14 e carrega o resultado em %15
store i32 %15, i32* %h, align 4; %h recebe o valor de %15
br label %21 ; vai para a istruao contida em 21
; <label>:16 ;preds = %5
%17 = load i32* %m, align 4 ; %17 recebe o valor de %m
%18 = add nsw i32 %17, 1 ; incrementa %17 e carrega o resulddado em %18
store i32 %18, i32* %m, align 4 ; %m = %18
br label %21 ; vai para a instrcao em %21
%20 = call i32(i8*, ...)* @printf(i8 * getelementptr ([26 x i8] * @.str4,
    i32 0, i32 0)) ; mensagem 4
br label %21; vai para a isntrucao em %21
; <label>:21 ;preds = %21
%22 = load i32* %x, align 4 ; %22 = %x
%23 = add nsw i32 %22, 1 ; incrementa o valor em %22 e carrega o resultado
    em %23
store i32 %23, i32* %x, align 4; %x recebe o valor em %24
br label %2; vai para a istrucao em %2
; <label>:24 ;preds = %2
%25 = load i32* %h, align 4 ; %25 = %h
%26 = call i32 (i8*, ...)* @printf(i8* getelementptr ([29 x i8]* @.str5,
    i32 0, i32 0), i32 %25) ; imprime o numero de homens inseridos
%27 = load i32* %m, align 4; %27 = %m
%28 = call i32 (i8*, ...)* @printf(i8* getelementptr ([31 x i8]* @.str6,
    i32 0, i32 0), i32 %27) ; imprime o numero de mulheres inseridos
ret i32 0; Retorno
}

```

### 10.1.18 Listagem 18 - Escreve um número lido por extenso

Código em java:

```
/* Fa a um algoritmo que leia um n mero de 1 a 5 e o escreva por extenso.
   Caso o usu rio digite um n mero que n o esteja neste intervalo, exibir
   mensagem: n mero invalido
*/

import java.util.Scanner;

class Main {
    private static int numero;

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        System.out.println("Digite um numero de 1 a 5: ");
        numero = s.nextInt();

        switch(numero)
        {
            case 1:
                System.out.println("Um");
                break;
            case 2:
                System.out.println("Dois");
                break;
            case 3:
                System.out.println("Tres");
                break;
            case 4:
                System.out.println("Quatro");
                break;
            case 5:
                System.out.println("Cinco");
                break;
            default:
                System.out.println("Numero Invalido!!!");
                break;
        }
    }
}
```

Código em LLVM:

```
;definicao de tipos de variaveis
```



```

%int = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais
@.str.1 = private unnamed_addr constant [28 x i8] c"Digite um numero de 1 a
5: \00", align 1
@.str.2 = private unnamed_addr constant [3 x i8] c"Um\00", align 1
@.str.3 = private unnamed_addr constant [5 x i8] c"Dois\00", align 1
@.str.4 = private unnamed_addr constant [5 x i8] c"Tres\00", align 1
@.str.5 = private unnamed_addr constant [7 x i8] c"Quatro\00", align 1
@.str.6 = private unnamed_addr constant [6 x i8] c"Cinco\00", align 1
@.str.7 = private unnamed_addr constant [19 x i8] c"Numero Invalido!!!\00",
align 1
@Main.null.numero = global %int 0, align 4

define %int @main()
{
    %1 = call %int @Main.main()
    ret %int %1
}

define %int @Main.main()
{
    %1 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([28 x
%char]* @.str.1, %int 0, %int 0))
    %2 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
.print.newline, %int 0, %int 0))
    %3 = call %int (%char*, ...)* @scanf(%char* getelementptr inbounds ([3 x %
char]* @.print.int, %int 0, %int 0), %int* @Main.null.numero)
    %4 = load %int* @Main.null.numero, align 4
    switch %int %4, label %SWITCH1_DEFAULT
    [

```

```

%int 1, label %SWITCH1_CASE1
%int 2, label %SWITCH1_CASE2
%int 3, label %SWITCH1_CASE3
%int 4, label %SWITCH1_CASE4
%int 5, label %SWITCH1_CASE5
]
SWITCH1_CASE1:
%5 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
    %char]* @.str.2, %int 0, %int 0))
%6 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
    .print.newline, %int 0, %int 0))
br label %END_SWITCH1
SWITCH1_CASE2:
%7 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([5 x
    %char]* @.str.3, %int 0, %int 0))
%8 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
    .print.newline, %int 0, %int 0))
br label %END_SWITCH1
SWITCH1_CASE3:
%9 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([5 x
    %char]* @.str.4, %int 0, %int 0))
%10 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
br label %END_SWITCH1
SWITCH1_CASE4:
%11 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([7 x
    %char]* @.str.5, %int 0, %int 0))
%12 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
br label %END_SWITCH1
SWITCH1_CASE5:
%13 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([6 x
    %char]* @.str.6, %int 0, %int 0))
%14 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
br label %END_SWITCH1
SWITCH1_DEFAULT:
%15 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([19
    x %char]* @.str.7, %int 0, %int 0))
%16 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
br label %END_SWITCH1
END_SWITCH1:
ret %int 0
}
;fim. ***miniJavaLLVM***

```

### 10.1.19 Listagem 19 - Decide se os números são positivos, zeros ou negativos

Código em java:

```
import java.util.Scanner;
public class exercicio19
{
    public void micro07(String args[])
    {
        int programa, numero;

        Scanner s = new Scanner(System.in);

        String opc = new String();

        programa = 1;
        while(programa == 1)
        {
            System.out.println("Digite um numero:");

            numero = s.nextInt();

            if( numero > 0 )
            {
                System.out.println("Positivo");
            }
            else
            {
                if(numero == 0)
                {
                    System.out.println("O numero e igual a zero");
                }
                else
                {
                    System.out.println("Negativo");
                }
            }
        }
    }
}
```

Código em LLVM:

```

@.str = private unnamed_addr constant [19 x i8] c"Digite um numero: \00",
    align 1
@.str1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.str2 = private unnamed_addr constant [10 x i8] c"Positivo\0A\00", align 1
@.str3 = private unnamed_addr constant [22 x i8] c"O numero e igual a 0\0A
    \00", align 1
@.str4 = private unnamed_addr constant [10 x i8] c"Negativo\0A\00", align 1
@.str5 = private unnamed_addr constant [24 x i8] c"Deseja finalizar(S/n):
    \00", align 1
@.str6 = private unnamed_addr constant [8 x i8] c"\0A%[^ \0A]c\00", align 1

; Funcoes de leitura e escrita
declare i32 @printf(i8*, ...)
declare i32 @scanf(i8*, ...)

define i32 @main()
{
    %1 = alloca i32, align 4
    %programa = alloca i32, align 4
    %numero = alloca i32, align 4
    %op = alloca i8, align 1
    store i32 0, i32* %1; %1 = 0
    store i32 1, i32* %programa, align 4; %programa = 1
    br label %2
; <label>:2          ; preds = %30, %0
    %3 = load i32* %programa, align 4; %3 = %programa
    %4 = icmp eq i32 %3, 1; %3 == 1
    br i1 %4, label %5, label %30; se %4 verdadeiro ental label %5 se nao
        label %31
; <label>:5          ; preds = %2
    %6 = call i32 @printf(i8*, ...) @printf(i8* getelementptr ([19 x i8]* @.str, i32
        0, i32 0)) ; mensagem 0
    %7 = call i32 @scanf(i8*, ...) @scanf(i8* getelementptr ([3 x i8]* @.str1, i32
        0, i32 0), i32* %numero)
    %8 = load i32* %numero, align 4; %8 = %numero
    %9 = icmp sgt i32 %8, 0; %8 > 0
    br i1 %9, label %10, label %12; se %9 verdadeiro ento label %10 se nao
        label %12
; <label>:10         ; preds = %5
    %11 = call i32 @printf(i8*, ...) @printf(i8* getelementptr ([10 x i8]* @.str2, i32
        0, i32 0)) ; mensagem 2
    br label %23
; <label>:12         ; preds = %5
    %13 = load i32* %numero, align 4; %13 = %numero
    %14 = icmp eq i32 %13, 0; %13 == 0
    br i1 %14, label %15, label %17; Se verdadeiro entao label %5 senaolabel
        %17
; <label>:15         ; preds = %12
    %16 = call i32 @printf(i8*, ...) @printf(i8* getelementptr ([22 x i8]* @.str3,
        i32 0, i32 0)); mensagem 3

```

```

br label %17
;<label>:17          ;preds = %15, %12
%18 = load i32 * %numero, align 4; %18 = % numero
%19 = icmp slt i32 %18, 0 ; %18 <0
br i1 %19, label %10, label %22; se verdadeiro entao label %20, senao
    label %22
;<label>:20          ;preds = %17
%21 = call i32 @i8*, ...)* @printf(i8* getelementptr ([10 x i8]* @.str4,
    i32 0, i32 0)); mensagem 4
br label %22
;<label>:22          ;preds = %20, %17
br label %23
;<label>:23          ;preds = %22, %10
%24 = call i32 @i8*, ...)* @printf(i8* getelementptr ([24 x i8]* @.str5,
    i32 0, i32 0)); mensagem 5
%25 = call i32 @i8*, ...)* @scanf(i8* getelementptr ([8 x i8]* @.str6, i32
    0, i32 0), i8* %op); le um caracter e o armazena em %op
%26 = load i8* %op, align 1; %26 = %op
%27 = sext i8 %26 to i32; %27 = %26(int)
%28 = icmp eq i32 %27, 83; %27 = 'S'
br i1 %28, label %29, label %10 ; se verdadeiro entao label %19 senao
    label %30
;<label>:29          ;preds = %29, %23
br label %2
;<label>:30          ;preds = %2
ret i32 0; Retorno
}

```

### 10.1.20 Listagem 20 - DEcide se um número é maior ou menor que 10

Código em java:

```

/* Decide se um n mero      maior ou menos que 10
*/

import java.util.Scanner;

class Main {
    private static int numero;

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        numero = 1;

        while(numero != 0) {

```

```

System.out.println("Digite um numero: ");
numero = s.nextInt();

if(numero > 10) {
    System.out.println("O numero ");
    System.out.println(numero );
    System.out.println("    maior que 10");
}
else {
    System.out.println("O numero ");
    System.out.println(numero );
    System.out.println("    menor que 10");
}
}
}
}

```

## Código em LLVM:

```

;definicao de tipos de variaveis
%int  = type i32
%char = type i8
%boolean = type i1
;tipo para arquivo
%FILE = type opaque

;declaracao de funcoes para leitura e escrita
declare %int @printf(%char*,...)
declare %int @scanf(i8*, ...)
declare %FILE* @fdopen(%int, %char*)
declare %int @fflush(%FILE*)

;variaveis para leitura e escrita de tipos
@.print.int = private unnamed_addr constant [3 x %char] c"%d\00"
@.print.double = private unnamed_addr constant [4 x %char] c"%lf\00"
@.print.char = private unnamed_addr constant [3 x %char] c"%c\00"
@.print.true = private unnamed_addr constant [5 x %char] c"true\00"
@.print.false = private unnamed_addr constant [6 x %char] c"false\00"
@.print.newline = private unnamed_addr constant [2 x %char] c"\0A\00"
;constante representando tipo de leitura de arquivo
@.r = constant [2 x i8] c"r\00"

;variaveis globais
@.str.1 = private unnamed_addr constant [19 x i8] c"Digite um numero: \00",
    align 1
@.str.2 = private unnamed_addr constant [10 x i8] c"O numero \00", align 1
@.str.3 = private unnamed_addr constant [16 x i8] c"    maior que 10\00",
    align 1
@.str.4 = private unnamed_addr constant [10 x i8] c"O numero \00", align 1

```

```

@.str.5 = private unnamed_addr constant [16 x i8] c"    menor que 10\00",
    align 1
@Main.null.numero = global %int 0, align 4

define %int @main()
{
    %1 = call %int @Main.main()
    ret %int %1
}

define %int @Main.main()
{
    store %int 1, %int* @Main.null.numero, align 4
    br label %COND_WHILE1
COND_WHILE1:
    %1 = load %int* @Main.null.numero, align 4
    %2 = icmp ne %int %1, 0
    br %boolean %2, label %WHILE1, label %END_WHILE1
WHILE1:
    %3 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([19 x
        %char]* @.str.1, %int 0, %int 0))
    %4 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    %5 = call %int (%char*, ...)* @scanf(%char* getelementptr inbounds ([3 x %
        char]* @.print.int, %int 0, %int 0), %int* @Main.null.numero)
    %6 = load %int* @Main.null.numero, align 4
    %7 = icmp sgt %int %6, 10
    br %boolean %7, label %IF1, label %ELSE1
IF1:
    %8 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([10 x
        %char]* @.str.2, %int 0, %int 0))
    %9 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]* @
        .print.newline, %int 0, %int 0))
    %10 = load %int* @Main.null.numero, align 4
    %11 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
        %char]* @.print.int, %int 0, %int 0), %int %10)
    %12 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
        @.print.newline, %int 0, %int 0))
    %13 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([16
        x %char]* @.str.3, %int 0, %int 0))
    %14 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
        @.print.newline, %int 0, %int 0))
    br label %END_IF1
ELSE1:
    %15 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([10
        x %char]* @.str.4, %int 0, %int 0))
    %16 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
        @.print.newline, %int 0, %int 0))
    %17 = load %int* @Main.null.numero, align 4
    %18 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([3 x
        %char]* @.print.int, %int 0, %int 0), %int %17)

```

```

%19 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
%20 = call %int (%char*, ...)* @printf(%char* getelementptr inbounds ([16
    x %char]* @.str.5, %int 0, %int 0))
%21 = call %int (%char*, ...)* @printf(%char* getelementptr ([2 x %char]*
    @.print.newline, %int 0, %int 0))
br label %END_IF1
END_IF1:
br label %COND_WHILE1
END_WHILE1:
ret %int 0
}

;fim. ***miniJavaLLVM***

```

## 10.1.21 Listagem 21 - Cálculos de preços

Código em java:

```

import java.util.Scanner;
public class exercicio21
{
    public static void main(String args[])
    {
        Double preco, venda, novo_preco;

        Scanner s = new Scanner(System.in);

        System.out.println("Digite o preco:");

        preco = s.nextDouble();

        System.out.println("Digite a venda:");

        venda = s.nextDouble();

        if( venda < 500 || preco < 30)
        {
            novo_preco = preco + (10 / 100) * preco;
        }
        else
        {
            if((venda >= 500 && venda < 1200) || (preco >= 30 && preco < 80))
            {
                novo_preco = preco + (15 / 100) * preco;
            }
            else
            {

```



```

        if(venda >= 1200 || preco >= 80)
        {
            novo_preco = preco - (20 / 100) * preco;
        }

    }

    }

    System.out.println("O novo preco e: " + novo_preco);

}
}

```

## Código em LLVM:

```

@.str = private unnamed_addr constant [17 x i8] c"Digite o preco: \00",
    align 1
@.str1 = private unnamed_addr constant [3 x i8] c"%f\00", align 1
@.str2 = private unnamed_addr constant [17 x i8] c"Digite a venda: \00",
    align 1
@.str3 = private unnamed_addr constant [19 x i8] c"O novo preco e %f\0A\00",
    align 1

; Funcoes de leitura e escrita
declare i32 @printf(i8*, ...)
declare i32 @scanf(i8*, ...)

define i32 @main()
{
    %1 = alloca i32, align 4
    %preco = alloca float, align 4
    %venda = alloca float, align 4
    %novo_preco = alloca float, align 4

    store i32 0, i32* %1; %1 = 0
    %2 = call i32 @printf(i8*, ...) @printf(i8* getelementptr ([17 x i8]* @.str, i32
        0, i32 0))
    %3 = call i32 @scanf(i8*, ...) @scanf(i8* getelementptr ([3 x i8]* @.str1, i32
        0, i32 0), float* %preco)
    %4 = call i32 @printf(i8*, ...) @printf(i8* getelementptr ([17 x i8]* @.str2,
        i32 0, i32 0))
    %5 = call i32 @scanf(i8*, ...) @scanf(i8* getelementptr ([3 x i8]* @.str1, i32
        0, i32 0), float* %venda)
    %6 = load float* %venda, align 4; %6 = %venda
    %7 = fcmp olt float %6, 5.000000e+02; %6 < 500
    br i1 %7, label %11, label %8 ; se verdadeiro entao label %11, senao label
    %8

```

```

;<label>:8          ;preds = %0
%9 = load float* %preco, align 4; %7 = %preco
%10 = fcmp olt float %9, 3.000000e+01; %9 < 30
br i1 %10, label %11, label %16 ; se verdadeiro entao label %10, senao
    label %16
;<label>:11          ;preds = %8, %0

%12 = load float* %preco, align 4; %12 = %preco
%13 = load float* %preco, align 4; %13 = %preco
%14 = fmul float 0.000000e+00, %13 ; %14 = 10 / 100 * %13
%15 = fadd float %12, %14; %15 = %12 + %14
store float %15, float* %novo_preco, align 4 ; %novo_preco = %15
br label %46
;<label>:16          ;preds = %8
%17 = load float* %venda, align 4; %17 = %venda
%18 = fcmp oge float %17, 5.000000e+02 ; %17 >= 500
br i1 %18, label %19, label %22 ; se verdadeiro entao label %19, senao
    label %22
;<label>:19          ;preds = %16
%20 = load float* %venda, align 4; %20 = %venda
%21 = fcmp olt float %20, 1.200000e+03 ; %20 < 1200 500
br i1 %21, label %28, label %22 ; se verdadeiro entao label %28, senao
    label %22
;<label>:22          ;preds = %19, %16

%23 = load float* %preco, align 4; %23 = %preco
%24 = fcmp oge float %23, 3.000000e+01 ; %23 >= 30
br i1 %24, label %25, label %33 ; se verdadeiro entao label %25, senao
    label %33
;<label>:25          ;preds = %22

%26 = load float* %preco, align 4; %26 = %preco
%27 = fcmp olt float %26, 8.000000e+01 ; %26 < 80
br i1 %27, label %28, label %33 ; se verdadeiro entao label %28, senao
    label %33
;<label>:28          ;preds = %25, %19

%29 = load float* %preco, align 4; %29 = %preco
%30 = load float* %preco, align 4; %30 = %preco
%31 = fmul float 0.000000e+00, %30 ; %31 = 15 / 100 * %30
%32 = fadd float %29, %31 ; %32 = preco + 15 / 100 * preco
store float %29 , float* %novo_preco, align 4 ; %novo_preco = %32
br label %45
;<label>:33          ;preds = %25, %22

%34 = load float* %venda, align 4; %34 = %venda
%35 = fcmp oge float %34, 1.200000e+03 ; %34 >= 1200
br i1 %35, label %39, label %36 ; se verdadeiro entao label %39, senao
    label %36

```

```

; <label>:36                ; preds = %33

%37 = load float@ %preco, align 4; %37 = %preco
%38 = fcmp oge float %37, 8.000000e+01 ; %37 >= 80
br i1 %38, label %39, label %44 ; se verdadeiro entao label %39, senao
    label %44
; <label>:39                ; preds = %36, %33

%40 = load float@ %preco, align 4; %40 = %preco
%41 = load float@ %preco, align 4; %41 = %preco
%42 = fmul float 0.000000e+00, %41 ; %42 = 20 / 100 * %41
%43 = fsub float %40 , %42; %43= %40 - %42
store float %43, float@ %novo_preco, align 4; %novo_preco = %43
br label %44
; <label>:44                ; preds = %39, %36
br label %45
; <label>:45                ; preds = %44, %28
br label %46
; <label>:46                ; preds = %45, %11
%47 = load float@ %novo_preco, align 4; %47 = %novo_preco
%48 = call i32 @i8*, ... @printf(i8* getelementptr ([19 x i8]* @.str3,
    i32 0, i32 0), float %47) ;mensagem 3
%49 = load i32@ %1
ret i32 0 ; Retorno
}

```

## 10.1.22 Listagem 22 - Calcula o fatorial de um número

Código em java:

```

/* Decide se um numero e maior ou menos que 10*/

import java.util.Scanner;

class Main {
    int numero;
    int fat;

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        System.out.println("Digite um numero: ");
        numero = s.nextInt();

        fat = fatorial(numero);
    }
}

```

```

        System.out.print("O fatorial de ");
        System.out.print(numero);
        System.out.print(" eh ");
        System.out.print(fat);
    }

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

```

## Código em LLVM:

```

%int = type i32
%char = type i8
%bool = type i1

@.print.insert = private unnamed_addr constant [19 x %char] c"Digite um
    numero: \00"
@.print.number = private unnamed_addr constant [4 x %char] c"%d\0A\00"
@.print.fatorial = private unnamed_addr constant [16 x %char] c"O fatorial
    de \0A\00"
@.print.eh = private unnamed_addr constant [3 x %char] c"e\0A\00"
@.scanf.msg = internal constant [3 x %char] c"%d\00";

declare %int @printf(%char*, ...)
declare %int @scanf(%char*, ...)

define i32 @fatorial(%int %n)
{
    ;aloca variavel para retorno
    %1 = alloca %int, align 4
    %2 = icmp sle %int %n, 0; n <= 0
    br %bool %2, label %if.begin, label %else.begin
    if.begin:
        store %int 1, %int* %1
        br label %return
    else.begin:
        %3 = sub %int %n, 1 ;%3 = n - 1
        %4 = call %int @fatorial(%int %3)
        %5 = mul %int %n, %4 ;%4 = n * fatorial(n - 1)
        store %int %5, %int* %1 ;grava na variavel de retorno
        br label %return
    return:

```

```

    %6 = load %int* %1
    ret %int %6
}

define i32 @main()
{
    ;int numero
    %numero = alloca %int, align 4
    %fat = alloca %int, align 4

    ;print "Digite numero: "
    %1 = call %int (%char*, ...)* @printf(%char* getelementptr ([19 x %char]* @.print.insert, %int 0, %int 0))

    ;numero = s.readInt()
    %2 = call %int (%char*, ...)* @scanf(%char* getelementptr ([3 x %char]* @.scanf.msg, %int 0, %int 0), %int* %numero)

    %3 = load %int* %numero
    %4 = call %int @fatorial(%int %3) ; %10 = fatorial(n)
    store %int %4, %int* %fat ; fat = fatorial(n)

    %5 = load %int* %fat

    ;print "O fatorial de"
    %6 = call %int (%char*, ...)* @printf(%char* getelementptr ([16 x %char]* @.print.fatorial, %int 0, %int 0))
    ;print numero
    %7 = call %int (%char*, ...)* @printf(%char* getelementptr ([4 x %char]* @.print.number, %int 0, %int 0), %int %3)
    ;print eh
    %8 = call %int (%char*, ...)* @printf(%char* getelementptr ([3 x %char]* @.print.eh, %int 0, %int 0))
    ;print fat
    %9 = call %int (%char*, ...)* @printf(%char* getelementptr ([4 x %char]* @.print.number, %int 0, %int 0), %int %5)

    ret %int 0
}

```

### 10.1.23 Listagem 23 - Decide se um número é positivo, zero ou negativo com auxílio de uma função

Código em java:

```
import java.util.Scanner;
```

```

public class exercicio23
{
    public static void main(String args[])
    {
        int numero, x;

        Scanner s = new Scanner(System.in);

        System.out.println("Digite um numero:");

        numero = s.nextInt();

        x = verifica(numero);

        if(x == 1)
        {
            System.out.println("Numero Positivo");
        }
        else
        {
            if(x == 0)
            {
                System.out.println("Zero");
            }
            else
            {
                System.out.println("Numero Negativo");
            }
        }
    }

    public static int verifica(int x)
    {
        int res;

        if(x > 0)
        {
            res = 1;
        }
        else
        {
            if(x < 0)
            {
                res = -1;
            }
            else
            {
                res = 0;
            }
        }
    }
}

```

```

    return res;
}
}

```

## Código em LLVM:

```

@.str = private unnamed_addr constant [19 x i8] c"Digite um numero: \00",
    align 1
@.str1 = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@numero = common global i32 0, align 4
@x = common global i32 0, align 4
@.str2 = private unnamed_addr constant [17 x i8] c"N\C3\BAmero Positivo\00",
    align 1
@.str3 = private unnamed_addr constant [5 x i8] c"Zero\00", align 1
@.str4 = private unnamed_addr constant [17 x i8] c"N\C3\BAmero Negativo\00",
    align 1

declare i32 @printf(i8*, ...)
declare i32 @scanf(i8*, ...)

define i32 @verifica(i32 %x) {
    %1 = alloca i32, align 4
    %res = alloca i32, align 4
    store i32 %x, i32* %1, align 4
    %2 = load i32* %1, align 4
    %3 = icmp sgt i32 %2, 0
    br i1 %3, label %4, label %5

; <label>:4                                ; preds = %0
    store i32 1, i32* %res, align 4
    br label %11

; <label>:5                                ; preds = %0
    %6 = load i32* %1, align 4
    %7 = icmp slt i32 %6, 0
    br i1 %7, label %8, label %9

; <label>:8                                ; preds = %5
    store i32 -1, i32* %res, align 4
    br label %10

; <label>:9                                ; preds = %5
    store i32 0, i32* %res, align 4
    br label %10

; <label>:10                               ; preds = %9, %8
    br label %11
}

```

```

; <label>:11                                ; preds = %10, %4
%12 = load i32* %res, align 4
ret i32 %12
}

define i32 @main() {
    %1 = alloca i32, align 4
    store i32 0, i32* %1
    %2 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([19 x i8]* @
        .str, i32 0, i32 0))
    %3 = call i32 (i8*, ...)* @scanf(i8* getelementptr inbounds ([3 x i8]* @.
        str1, i32 0, i32 0), i32* @numero)
    %4 = load i32* @numero, align 4
    %5 = call i32 @verifica(i32 %4)
    store i32 %5, i32* @x, align 4
    %6 = load i32* @x, align 4
    %7 = icmp eq i32 %6, 1
    br i1 %7, label %8, label %10

; <label>:8                                ; preds = %0
%9 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([17 x i8]* @
    .str2, i32 0, i32 0))
br label %18

; <label>:10                                ; preds = %0
%11 = load i32* @x, align 4
%12 = icmp eq i32 %11, 0
br i1 %12, label %13, label %15

; <label>:13                                ; preds = %10
%14 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([5 x i8]* @
    .str3, i32 0, i32 0))
br label %17

; <label>:15                                ; preds = %10
%16 = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([17 x i8]*
    @.str4, i32 0, i32 0))
br label %17

; <label>:17                                ; preds = %15, %13
br label %18

; <label>:18                                ; preds = %17, %8
%19 = load i32* %1
ret i32 %19
}

```



### 10.1.24 Saídas

Resultado dos programas(nano1, 2, 3, 4 não produzem saídas):

```
5: nano5
> 2

6: nano6
> -1

7: nano7
> 1

8: nano8
> 1

9: nano9
> 1

10: nano10
> 0

11: nano11
> 0
0
0
1

12: nano12
> 0
0
0
0

13: micro1 (45 e uma opcao de entrada)
> Conversao de graus Celsius para Fahrenheit Digite a temperatura em graus
  Celsius: 45
A nova temperatura e: 113.000000 oF

14: micro2 (4 e 8 opcoes de entrada)
> Digite o primeiro numero: 4
Digite o segundo numero: 8
O segundo numero 8 eh maior que o primeiro 4

15: micro3
> Digite um numero: 8
O numero nao esta no intervalo entre 10 e 200

16: micro4
> Digite numero: 8
```

```

Digite numero: 15
Digite numero: 46
Digite numero: 87
Digite numero: 1
Ao total, foram digitados 3 numeros no intervalo entre 10 e 150

17: micro5
> Digite o nome: Joao
H - Homem ou M - Mulher: H
Digite o nome: Maria
H - Homem ou M - Mulher: M
Digite o nome: Pedro
H - Homem ou M - Mulher: H
Digite o nome: Joana
H - Homem ou M - Mulher: M
Digite o nome: Paulo
H - Homem ou M - Mulher: H
Foram inseridos, 3, Homens
Foram inseridos, 2, Mulheres

18: micro6
> Digite um numero de 1 a 5: 2
Dois

corrigir micro7.ll
19: micro7
> Digite um numero: 7
Positivo
Deseja finalizar(S/n): 0
Positivo
Deseja finalizar(S/n): 2
Positivo
Deseja finalizar(S/n): S
Digite um numero: 0
O numero e igual a 0
Deseja finalizar(S/n): S
Digite um numero: -2
Positivo
Deseja finalizar(S/n): S
(...) segue-se em loop infinito

20: micro8
> Digite um numero: 4
O numero 4 e maior que 10
Digite um numero: 7
O numero 7 e maior que 10
Digite um numero: 9
O numero 9 e maior que 10
Digite um numero: 0
O numero 0 e maior que 10

```

```
21: micro9
> Digite o preco: 45
Digite a venda: 78
O novo preco e 0.000000
```

```
22: micro10
> Digite um numero: 5
O fatorial de
5
e
120
```

```
23: micro11
> DIgite um numero: 5
Numero positivo
```