Apêndice A. Pseudo-Linguagem

A.1 Considerações Preliminares

Os computadores convencionais se baseiam no conceito de uma memória principal que consiste de células elementares, cada qual identificada por um endereço. O conteúdo de uma célula é o valor da mesma. O valor de uma célula pode ser lido e/ou modificado. Esta modificação é feita pela substituição de um valor por outro. Além disso, circuitos permitem o acesso a uma célula de cada vez. Com poucas exceções, as linguagens de programação podem ser consideradas como abstrações, em níveis diferentes, do comportamento destes computadores convencionais. Em particular, o conceito de variável é introduzido como uma abstração de células de memória, e o conceito de comando de atribuição como uma abstração destrutiva destas células.

Uma variável é caracterizada por um **nome** e dois atributos básicos: **valor** e **tipo**. O **nome** é usado para identificar e fazer referência à variável. O **valor** de uma variável é representado, de forma codificada, na área de memória amarrada à variável. Este código é interpretado de acordo com o **tipo** da variável. O **tipo** de uma variável pode ser considerado como uma especificação da classe de valores que podem ser associados à variável, bem como das operações que podem ser usadas para criar, acessar e modificar estes valores.

A amarração entre uma variável e o valor armazenado na área de memória correspondente é, em geral, dinâmica, já que este valor pode ser modificado por operações de atribuição. Uma atribuição como **b ← a** causa o armazenamento de uma cópia do valor da variável **a** na área de memória amarrada à variável **b**.

Algumas linguagens, entretanto, permitem o congelamento da amarração entre uma variável e o seu valor quando a amarração é estabelecida. A entidade resultante é, sob qualquer aspecto, uma constante simbólica definida pelo programador.

Por exemplo, em Pascal se pode escrever:

```
const
  pi = 3.1416;
```

e então usar **pi** em uma expressão como **circunferência:= 2 * pi * raio;.**

A variável **pi** está amarrada ao valor **3.1416**, e este valor não pode ser modificado; isto é, o compilador acusa erro se existir uma atribuição à **pi**.

Poderíamos usar o valor **3.1416** diretamente na expressão acima, dispensando o uso da constante simbólica **pi** (**circunferência:= 2 * 3.1416 * raio;**). Neste caso, chamamos **3.1416** de constante literal.

A.2 Pseudo-linguagem LPE

Nos itens subseqüentes iremos descrever as regras sintáticas e semânticas de uma linguagem de construção de algoritmos chamada Linguagem de Programação Estruturada (LPE). Esta linguagem tem por objetivo permitir a elaboração de um algoritmo sem termos que nos preocupar com as regras rígidas de uma linguagem de

programação real. Apesar da LPE possuir um certo formalismo, nos é permitido "afrouxar" as suas regras sintáticas quando for conveniente. Isto pode ser feito pelo fato da LPE não ser uma linguagem de programação real; isto é, não existe nenhum tradutor de LPE que nos permita executar um programa escrito nesta linguagem em um computador real. Um programa em LPE terá o seguinte lay-out:

```
constantes
  <nome> = <valor>;
  …
  <nome> = <valor>;
variáveis
  <nome> : <tipo>;
  …
  <nome> : <tipo>;
início
  <comando>;
  …
  <comando>;
fim.
```

No lay-out acima, tudo o que estiver em negrito faz parte da sintaxe da linguagem; e o que estiver entre < > será posteriormente substituído por construções dos programadores. Esta construções, porém, terão também que obedecer às regras da LPE.

A.2.1 Nomes

Um nome de uma variável, ou de uma constante simbólica, é uma seqüência de no máximo 32 caracteres alfanuméricos, além do caracter sublinhado (_). O primeiro caracter tem que ser obrigatoriamente alfabético. A linguagem LPE não faz distinção entre caracteres maiúsculos e minúsculos.

Exemplo:

```
a
nome_aluno
c8
valor
xyz
```

A.2.2 Tipos

A.2.2.1 Inteiro

O tipo **inteiro** é usado para representar valores inteiros positivos e negativos. As operações disponíveis para o tipo **inteiro** são representadas pelos operadores +

(soma), **-** (subtração), ***** (multiplicação), **div** (quociente da divisão inteira), e **mod** (resto da divisão inteira).

A.2.2.2 Real

O tipo **real** é usado para representar valores decimais com representação finita. As operações disponíveis para o tipo **real** são representadas pelos operadores + (soma), **-** (subtração), ***** (multiplicação), / (divisão).

A.2.2.3 Lógico

O tipo lógico é usado para representar os valores lógicos **verdadeiro** e **falso**. As operações disponíveis para o tipo lógico são representadas pelos operadores ∧ (e), ∨ (ou), e ~ (não).

A.2.2.4 Caracter

O tipo **caracter** é usado para representar qualquer cadeia de caracteres tais como nomes, endereços e etc.

A LPE não oferece nenhuma operação sobre os valores do tipo **caracter**.

A.2.3 Constantes Literais

Uma constante literal é uma constante cujo nome é a representação escrita do seu valor. Por exemplo, **21** é a representação decimal de um objeto de dados cujo valor é 21.

Exemplo:

```
12.450  -0.5  1.0                    /* tipo real */
verdadeiro  falso                    /* tipo lógico */
1  -12  234                          /* tipo inteiro */
'PUC-Rio'  'A'  '123'  '     '       /* tipo caracter */
```

A.2.4 Comandos e Expressões

A.2.4.1 Expressões

Uma expressão é uma fórmula ou regra de computação que sempre determina um valor ou resultado. Uma expressão consiste de operandos e operadores. Os operadores da LPE já foram apresentados nos itens relativos aos tipos de dados. Os operandos são constantes e variáveis. Se vários operadores ocorrem em uma expressão, a ordem de execução das operações precisa ser especificada; quer seja pelo emprego explícito de parênteses, quer seja pelas regras implícitas da linguagem (veja a tabela abaixo).

Exemplo:

```
3               /* uma constante é uma expressão */
abc             /* uma variável é uma expressão */
n mod 2
salário * 1.25
(renda_bruta - desconto) * 0.15
```

Neste ponto iremos apresentar os **operadores relacionais**. Os operadores relacionais são amplamente usados na matemática e dispensam apresentações. Estes operadores são usados para relacionar duas expressões; criando uma nova expressão. O valor desta nova expressão é sempre uma valor lógico; isto é, **verdadeiro** ou **falso**. Os operadores relacionais são os seguintes:

= (igual)
≠ (diferente)
> (maior)
< (menor)
>= (maior ou igual)
<= (menor ou igual)

As regras implícitas de precedência dos operadores da LPE são mostradas na tabela abaixo (do maior para o menor):

1. Parênteses
2. Operadores Aritméticos
   - 2.1. +, - (unários)
   - 2.2. *, /, div, mod
   - 2.3. +, - (binários)
3. Operadores Relacionais
4. ~ (não)
5. ∧ (e)
6. ∨ (ou)

A.2.4.2 Comando de Atribuição

O comando de atribuição tem a seguinte sintaxe:

```
<variável> ← <expressão>;
```

onde o termo <variável> dever ser substituído por uma variável declarada na seção **variáveis**, e o termo <expressão> deve ser substituído por uma expressão válida na linguagem LPE. Esta expressão tem que ser de um tipo que seja compatível com o tipo da variável à esquerda do operador de atribuição (←).

O comando de atribuição irá avaliar a expressão à direita do operador de atribuição, e substituirá o valor da variável à esquerda do operador pelo valor da expressão.

Exemplo:

```
idade ← 3;          /* uma constante é uma expressão */
x ← n mod 2;
salário ← salário * 1.25;
imposto ← (renda_bruta - desconto) * 0.15;
endereço ← 'Rua JK, 23';
```

A.2.4.3 Comando de Leitura

O comando de leitura tem a seguinte sintaxe:

```
leia(<var1>,<var2>,…,<varN>);
```

onde os termos <varK> devem ser substituídos por variáveis declaradas na seção **variáveis**, **exceto** por variáveis do tipo **lógico**. O comando de leitura tem por objetivo transferir dados de um periférico, por exemplo o teclado, e armazená-los nas variáveis fornecidas no comando.

Exemplo:

```
leia(matrícula,nome,idade,sexo);
```

A.2.4.4 Comando de Escrita

O comando de escrita tem a seguinte sintaxe:

```
escreva(<exp1>,<exp2>,…,<expN>);
```

onde os termos <expK> devem ser substituídos por expressões válidas na linguagem LPE, **exceto** por expressões que têm valor do tipo **lógico**. O comando de escrita tem por objetivo transferir dados da memória principal para um periférico; por exemplo um monitor de vídeo.

Exemplo:

```
escreva('O Nome do Aluno e ',nome);
escreva('Salário - ',salário,' Imposto - ',salário*0.1);
```

A.2.5 Estrutura de Controle

A.2.5.1 Bloco

Um bloco consiste de um conjunto de comandos delimitados pelas palavras **início** e **fim**. Um bloco pode ser interpretado como sendo um comando composto por vários

outros comandos, e cuja a execução tem efeito igual ao obtido pela execução dos vários comandos nele inseridos.

Na sintaxe da LPE, todas as vezes que aparecer o símbolo <comando>, indicando a obrigatoriedade de se codificar um comando; este poderá se substituído por um bloco.

Exemplo:

```
variáveis
  i, j, k : inteiro;
  abc: real;
início
  i ← 0;
  j ←0;
  início
    k ← (i + j) * 3;
    abc ← k * 8.5;
    escreva(abc);
  fim;
  escreva(i,j,k);
fim.
```

A.2.5.2 Seleção

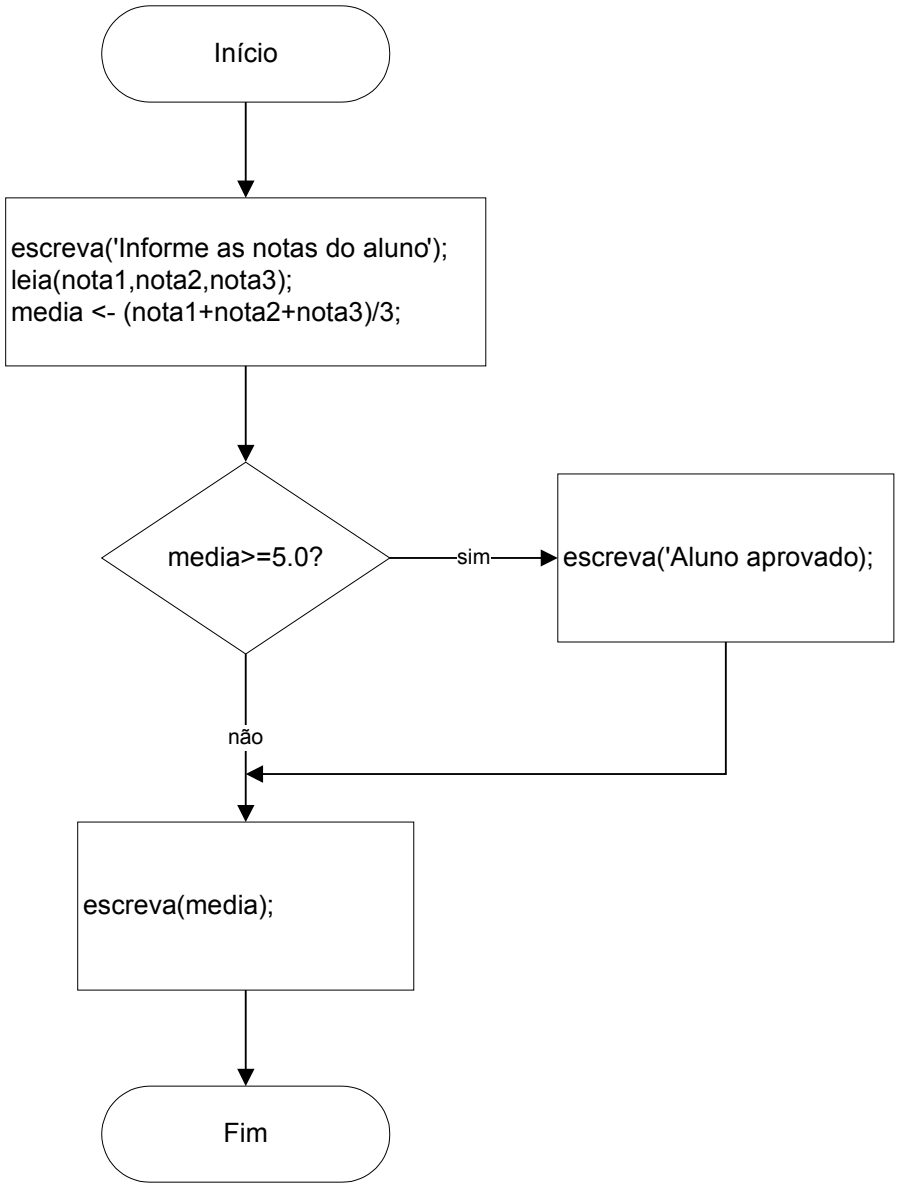
O comando de seleção tem duas formas:

```
se <expressão lógica> então
    <comando>;
```

nesta forma a <expressão lógica> é avaliada inicialmente. Se o valor da expressão for **verdadeiro**, o <comando> é executado. Se o valor da expressão for **falso**, o <comando> não é executado.

Exemplo

```
variáveis
  média, nota1, nota2, nota3 : real;
início
  escreva('Informe as notas do aluno');
  leia(nota1, nota2, nota3);
  média ← (nota1+nota2+nota3)/3;
  se média >= 5.0 então
    escreva('Aluno Aprovado');
  escreva(média);
fim.
```



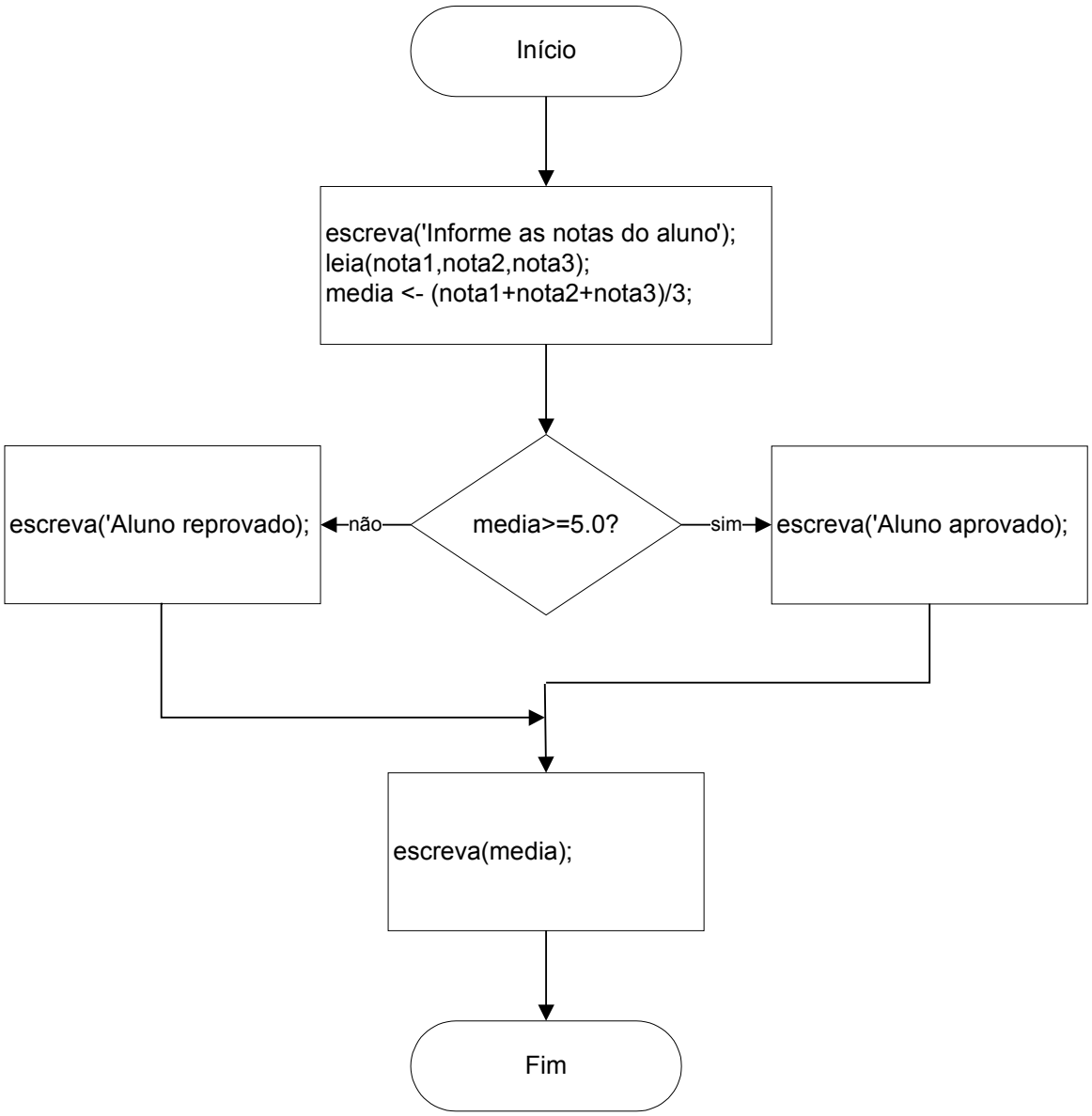

```
se <expressão lógica> então
  <comando1>;
senão
  <comando2>;
```

nesta forma a <expressão lógica> é avaliada inicialmente. Se o valor da expressão for **verdadeiro**, o <comando1> é executado e o <comando2> não. Se o valor da expressão for **falso**, o <comando2> é executado e o <comando1> não.

Exemplo

```
variáveis
```

```
  média, nota1, nota2, nota3 : real;
início
  escreva('Informe as notas do aluno');
  leia(nota1, nota2, nota3);
  média ← (nota1+nota2+nota3)/3;
  se média >= 5.0 então
    escreva('Aluno Aprovado');
  senão
    escreva('Aluno Reprovado');
  escreva(média);
fim.
```



A.2.5.3 Repetição

O comando de repetição tem a seguinte sintaxe:

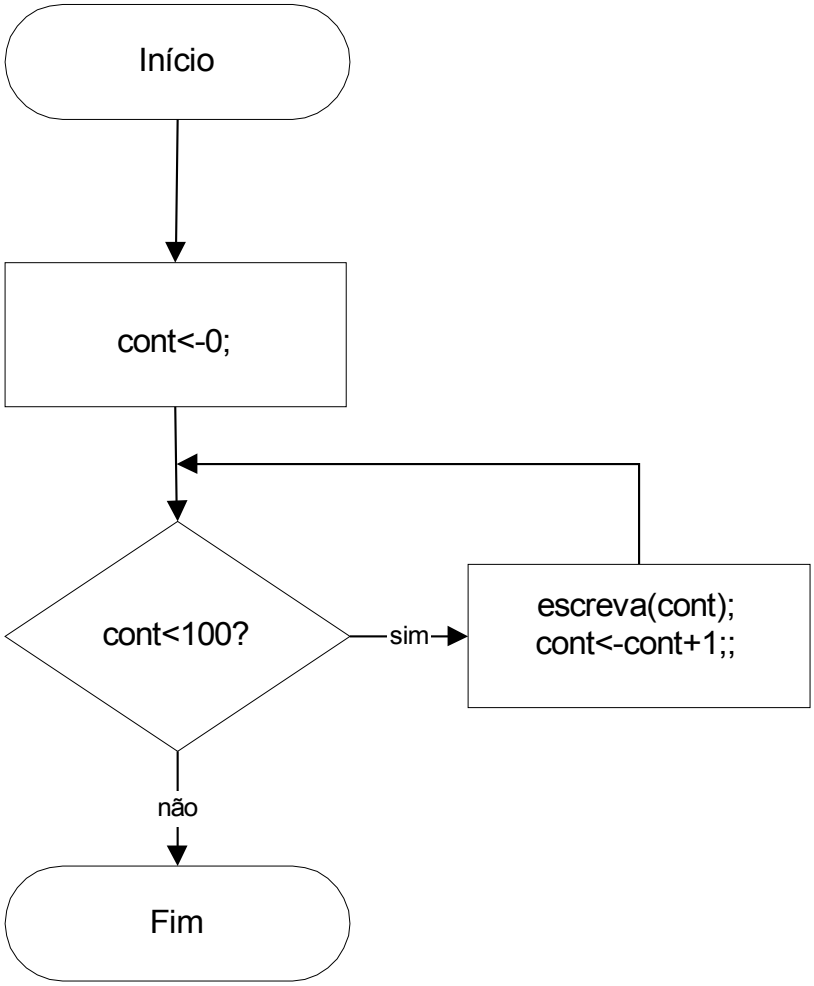
```
enquanto <expressão lógica> faça
  <comando>;
```

neste comando a <expressão lógica> é avaliada inicialmente. Se o valor da expressão for **verdadeiro**, o <comando> é executado e a <expressão lógica> é avaliada novamente. Este ciclo é repetido até que a <expressão lógica> seja avaliada como **falso**. É importante notar que se a <expressão lógica> for avaliada inicialmente como **falso** o <comando> não será executado nenhuma vez. Outra observação importante é que o <comando> deverá alterar de alguma forma as variáveis presentes na <expressão lógica>. Se isto não ocorrer, a <expressão lógica> será sempre avaliada como **verdadeiro** (supondo que a <expressão lógica> seja avaliada como **verdadeiro** na primeira avaliação) e o <comando> será executado continuamente. Diremos então que o "programa entrou em um looping infinito".

Exemplo

O exemplo a seguir exibirá no monitor de vídeo todos os números inteiros entre 0 e 99.

```
variáveis
  cont : inteiro;
início
  cont ← 0;
  enquanto cont < 100 faça
    início
      escreva(cont);
      cont ← cont + 1;
    fim;
fim.
```


Início
cont<-0;
cont<100?
sim
escreva(cont);
cont<-cont+1;;
não
Fim