Tutorial do Interpretador de Linguagem Algorítmica (ILA)

Preparado por Elvio Leonardo, a partir do tutorial fornecido com o *software* ILA
e do material produzido pelo Prof. Marcelo Azambuja

I. Introdução

O *software* ILA é um interpretador, pequeno e versátil, que permite o teste de algoritmos descritos em *português estruturado*. Este *software* pode ser obtido gratuitamente a partir da página do responsável por seu desenvolvimento, Prof. Sérgio Crespo, no endereço:
`www.inf.unisinos.br/~crespo/ila/ila.htm`.
Os arquivos para instalação estão compactados e ocupam apenas 97 K*bytes*. Descompactados, eles ocupam menos de 250 K*bytes*, o que permite a execução a partir de um disquete. O programa roda em MS-DOS, ou em DOS executando sobre MS-Windows. Na medida do possível, é aconselhável a instalação deste *software* em suas máquinas de trabalho particulares (em casa, no trabalho, etc.) para facilitar o desenvolvimento dos trabalhos extra-classe. Alternativamente, os estudantes podem utilizar-se das instalações oferecidas pela universidade, como os laboratórios de informática para a graduação.

II. Regras Gerais

Bem vindo ao ILA, o Interpretador de Linguagem Algoritmica, que o auxiliará a demonstrar a praticidade das estruturas algorítmicas abordadas nas disciplinas de ALGORITMOS e ESTRUTURA DE DADOS.

A estrutura básica de um programa em ILA deve seguir o padrão apresentado na Fig. 1.

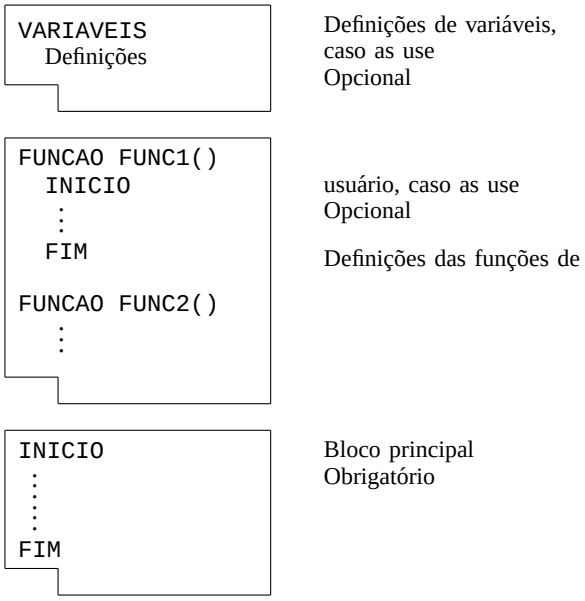

Fig. 1. Estrutura do programa em ILA.

A. Declaração de Variáveis

Todas as variáveis definidas pelo usuário devem ser declaradas em separado, em um bloco especial denominado `VARIAVEIS`. Podem ser definidas duas classes distintas de variáveis: variáveis simples e compostas (ou indexáveis).

B. Variáveis Simples

Sintaxe:
`numerico` <*nome*>`[,` <*nome*> `[,`<*nome*>`]]` ...
`logico` <*nome*>`[,` <*nome*> `[,`<*nome*>`]]` ...
`caracter` <*nome*>`[,` <*nome*> `[,`<*nome*>`]]` ...

C. Variáveis Compostas

Sintaxe:
`matriz numerico` <*nome*>`[`<*expr*>`[,` <*expr*>`]]` ...]

Exemplo 1:

```
// Algoritmo para criar e imprimir um
// vetor de 100 posicoes
// --------------------------------
variaveis
    numerico valor, i
    matriz numerico vet[100]

inicio
    limpar
    para i=1 ate 100
        posicionar 10, 10
        escrever "Entre com um numero = "
        posicionar 10, 40
        ler valor
        vet[i] = valor
    proximo
    para i=1 ate 100
        escrever vet[i]
    proximo
fim
```

Exemplo 2:

```
// Algoritmo para criar e imprimir
// uma matriz mat(3,3)
// -------------------------------
variaveis
    numerico val, i, j
    matriz numerico mat[3,3]
inicio
    para i=1 ate 3
        para j=1 ate 3
            posicionar 10, 10
            escrever "Digite um valor"
```

```
            posicionar 10, 40
            ler val
            mat[i,j] = val
        proximo
    proximo
    // impressao da matriz - nao formatada
    para i=1 ate 3
        para j=1 ate 3
            escrever mat[i,j]
        proximo
    proximo
fim
```

D. Declaração de Funções

Uma chamada à uma função pode ser feita de dentro de uma expressão, ou na forma de uma chamada de subrotina.

Sintaxe:
<*identificador*>([<*parâmetro*> [, <*parâmetro*>]] ...)

Exemplo 1:

```
// Algoritmo para ler um nome usando
// uma funcao para imprimi-lo
// --------------------------------
variaveis
    caracter  nome
funcao imprime(nome)
    // parametro = nome
    inicio
        posicionar 10, 10
        escrever "nome = " , nome
    fim
// Inicio do algoritmo principal
inicio
    limpar
    posicionar 5, 10
    escrever "Digite um nome: "
    ler nome
    imprime(nome)
fim
```

Exemplo 2:

```
// Algoritmo que calcula o fatorial de um
// numero usando uma funcao recursiva
// -------------------------------------
Variaveis
    Numerico  n
    Caracter  c
//-------------
Funcao Fat(n)
// Funcao recursiva que calcula fatorial
Inicio
    Se n < 0 entao
        Retornar 0
    Senao
        Se (n = 0) ou (n = 1) entao
            Retornar 1
        Senao
            Retornar n * Fat(n-1)
        Fim_se
    Fim_se
Fim
// Inicio do algoritmo principal
Inicio
    cor 1,3
    limpar
    janela 01, 01, 24, 79
    posicionar 02, 02
    escrever "CALCULO DE FATORIOAL - RECURSIVO"
    posicionar 04, 02
    escrever "Digite 999 para sair"
    n = 0
    faca enquanto n <> 999
        posicionar 10, 10
        escrever "Digite um nº : "
        posicionar 10, 26
        ler n
        se n <> 999 entao
            posicionar 11, 10
            escrever "fat(", n, ") e' ", Fat(n)
        Fim_se
    Fim_enquanto
fim
```

E. Atribuições

As atribuições são valores recebidos pelas variáveis definidas pelo usuário.

Sintaxe:
<*variável*> = <*expressão*>

Exemplo:

```
variaveis
    numerico num, outro
    numerico num, outro
inicio
    num = 12
    outro = 123
    num = outro
    num = outro + num
fim
```

F. Operadores

A prioridade dos operadores obecede as regras matemáticas.

Operador Unário	
número negativo	-
Operadores Aritméticos	
potenciação	^
multiplicação	*
divisão	/
adição	+
subtração	-
Operadores Lógicos	
multiplicação (AND)	E
adição (OU)	OU
complemento (NOT)	NAO
Operadores Relacionais	
maior	>
menor	<
igual	=
diferente	<>
maior ou igual	>=
menor ou igual	<=
Operador de Campos Caracter	
concatenação	+

Exemplo:

```
// Concatenar dois campos tipo caracter
// em um terceiro
// ------------------------------------
variaveis
    caracter  pre_nome, sobrenome, nome
inicio
    limpar
    ler pre_nome
    ler sobrenome
    nome = pre_nome + sobrenome
    escrever "aluno = ", nome
fim
```

G. Comentários

O simbolo `//` faz com o que tudo que estiver a sua direita seja ignorado pelo interpretador.

Exemplo:

```
// Isto e' apenas um comentario
```

III. Constantes

A. FALSO

Constante lógica com valor “falso”.

Sintaxe:
`FALSO`

B. NP

Constante numérica com valor do número neperiano (2.71828182846...).

Sintaxe:
`NP`

C. PI

Constante numérica com o valor de π (3.14159265359...).

Sintaxe:
`PI`

D. VERDADEIRO

Constante lógica com valor “verdadeiro”.

Sintaxe:
`VERDADEIRO`

IV. Funções Matemáticas

A. ACOS

Calcula o arco-cosseno de um número, com a resposta dada em radianos.

Sintaxe:
`ACOS(<`*expressão*`>)`

Exemplo:

```
// Algoritmo para calcular o arco cosseno de 1
// ------------------------------------------
inicio
    limpar
    escrever "Arco-cosseno de 1 = " , ACOS(1)
fim
```

ALEATORIO

Devolve um número entre 0 e 1, gerado ao acaso, seguindo a distribuição uniforme.

Sintaxe:
`ALEATORIO()`

Exemplo:

```
// Algorítmo que imprime um número aleatório
// ----------------------------------------
variaveis
    numerico num
inicio
    num = aleatorio()
    escrever "um numero aletatorio = ", num
fim
```

B. ASEN

Calcula o arco-seno de um número, com a resposta dada em radianos.

Sintaxe:
`ASEN(<`*expressão*`>)`

C. ATAN

Calcula o arco tangente de um número, com a resposta dada em radianos.

Sintaxe:
`ATAN(<`*expressão*`>)`

D. COS

Calcula o cosseno de um ângulo expresso em radianos.

Sintaxe:
`COS(<`*expressão*`>)`

E. INTEIRO

Extrai de um número qualquer somente a sua parte inteira.

Sintaxe:
`INTEIRO(<`*expressão*`>)`

Exemplo:

```
// Algoritmo para imprimir um numero
// fracionario e sua parte inteira
// -------------------------------
variaveis
    numerico num
inicio
    num = 3.1416
    escrever "numero fracionario = ", num
    escrever "parte inteira      = ", inteiro(num)
fim
```

F. LOG

Calcula o logarítmo na base 10 de um número.

Sintaxe:
`LOG(<`*expressão*`>)`

G. RAIZ

Calcula a raiz de um número.

Sintaxe:
`RAIZ(<`*expr_1*`>, <`*expr_2*`>)`

onde *expr_1* corresponde ao radicando, e *expr_2* corresponde ao índice do radical.

Exemplo:

```
// Algoritmo que extrai a raiz de numero e radical dados
// ---------------------------------------------------
variaveis
    numerico num, resp
inicio
    ler num
    ler indice
    resp = raiz(num,indice)
    escrever "Numero lido = ", num
    escrever "Raiz ", indice, " = ", resp
fim
```

H. RESTO

Calcula o resto da divisão de *expr_1* por *expr_2*.

Sintaxe:
RESTO(<*expr_1*>, <*expr_2*>)

I. SEN

Calcula o seno de um ângulo expresso em radianos.

Sintaxe:
SEN(<*expressão*>)

J. TAN

Calcula o tangente de um ângulo expresso em radianos.

Sintaxe:
TAN(<*expressão*>)

V. FUNÇÕES DE ENTRADA E SAÍDA DO ILA

A. COR

Muda o atributo de cor no vídeo.

Sintaxe:
COR [<*fore*>, <*back*>]

onde *fore* identifica a cor de frente do vídeo (letras, por exemplo); e *back* identifica a cor de fundo. Para ter efeito, este comando deve ser usado antes dos comandos ESCREVER, JANELA e LIMPAR.

Tabela de Cores	
0	preto
1	azul
2	verde
3	ciano
4	vermelho
5	magenta
6	marron
7	cinza
8	preto_intenso
11	azul_intenso
10	verde_intenso
12	ciano_intenso
13	vermelho_intenso
14	magenta_intenso
15	amarelo
16	branco

Exemplo 1:

```
cor                    assume o default preto e branco
cor "preto", "azul"    preto para frente e azul para fundo
cor 1, 3               preto para frente e azul para fundo
cor frente, fundo      conteudo das variáveis define cores
```

Exemplo 2:

```
// Algoritmo para gerar uma cor aleatória no video
// ---------------------------------------------
variaveis
    numerico frente, fundo
inicio
    frente = aleatorio() * 10
    fundo  = aleatorio() * 10
    cor frente, fundo
    limpar
    esperar(200)
fim
```

B. ESCREVER

Envia para o dispositivo de saída (por *default* o vídeo) um texto, uma variável, uma expressão aritmética, ou uma combinação destes.

Sintaxe:
ESCREVER [<*variável*>,] ["<*texto*>",] [<*expressão*>]

Exemplo:

```
// Algoritmo para ler um nome e numero,
// e imprimir: nome e numero elevado ao
// quadrado
// ----------------------------------
variaveis
    caracter nome
    numerico num
inicio
    ler nome
    ler numero
    escrever "Nome = ", nome
    escrever "Numero = ", num * num
fim
```

C. IMPRESSORA

Especifica a impressora como dispositivo de saída. Este dispositivo é utilizado pelo comando ESCREVER para saída de informação e, para ter efeito, deve ser usado antes do comando ESCREVER.

Sintaxe:
IMPRESSORA()

Exemplo:

```
// Algoritmo para ler um nome e numero,
// e imprimir: nome e numero na impressora
// -------------------------------------
variaveis
    caracter nome
    numerico num
inicio
    impressora()
    ler nome
    ler numero
    escrever "Nome = ", nome, " Numero = ", num
fim
```

D. JANELA

Desenha uma moldura na tela.

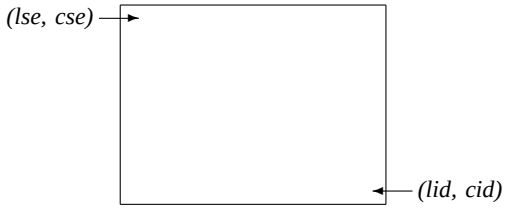

Fig. 2. Referências para a tela de vídeo.

Sintaxe:
JANELA [<*lse*>, <*cse*>, <*lid*>, <*cid*>]

onde *lse* e *cse* especificam, respectivamente, a linha e a coluna do canto superior esquerdo da área se ser criada; e *lid* e *cid* a linha e a coluna do canto inferior direito desta mesma área, conforme ilustrado na Fig. 2. Caso *lse*, *cse*, *lid* e *cid* sejam omitidos, o comando executa com os valores *default*: `JANELA(1,1,24,80)`.

Exemplo:

```
// Algoritmo para criar uma moldura no video
// ----------------------------------------
inicio
    limpar
    janela 10, 10, 20, 50
fim
```

E. LER

Lê uma entrada feita através do teclado coloca-a em uma variável.

Sintaxe:
LER(<*variável*>)

F. LIMPAR

Limpa a área especificada da tela de vídeo.

Sintaxe:
LIMPAR [<*lse*>, <*cse*>, <*lid*>, <*cid*>]

onde *lse* e *cse* correspondem, respectivamente, a linha e a coluna do canto superior esquerdo da área se ser limpa; e *lid* e *cid* a linha e a coluna do canto inferior direito desta mesma área, conforme ilustrado na Fig. 2. Caso *lse*, *cse*, *lid* e *cid* sejam omitidos, o comando executa com os valores *default*: `LIMPA(1,1,24,80)`.

Exemplo:

```
// Algoritmo para limpar todo o video,
// escrever uma frase e após limpar
// somente a metade inferior
// ---------------------------------
inicio
    limpar
    escrever "O valor de 7*45 = ", 7 * 45
    limpar 15, 01, 24, 79
fim
```

G. POSICIONAR

Posiciona o cursor na tela de vídeo.

Sintaxe:
POSICIONAR <*lin*>, <*col*>

onde *lin* e *col* correspondem, respectivamente, à linha e à coluna na tela.

H. VIDEO

Especifica a tela de vídeo como dispositivo de saída. Este dispositivo é utilizado pelo comando ESCREVER para saída de informação e, para ter efeito, deve ser usado antes do comando ESCREVER. O vídeo é o dispositivo *default* de saída.

Sintaxe:
VIDEO()

VI. Funções Para o Tipo Caracter

A. COMPRIMENTO

Informa o tamanho (comprimento) de uma variável do tipo caracter.

Sintaxe:
COMPRIMENTO(<*variável*>)

Exemplo:

```
// Algoritmo para ler um nome e
// imprimir o seu comprimento
// --------------------------
variaveis
    caracter nome
    numerico tam
inicio
    ler nome
    tam = comprimento(nome)
    escrever "Nome lido = ", nome
    escrever "Tamanho do campo = ", tam
fim
```

B. VALOR

Converte o conteúdo da variável do tipo caracter *var* em um valor numérico.

Sintaxe:
VALOR(<*var*>)

Exemplo:

```
// Algoritmo que converte uma "string" em numero
// ---------------------------------------------
variaveis
    caracter string
    numerico numero
inicio
    string = "123.89"
    numero = valor(string)
    escrever "Caracter = ", string
    escrever "Numero   = ", numero
fim
```

VII. Comandos de Fluxo

A. FACA CASO

Especifica uma estrutura de decisão. Provoca o desvio do fluxo do programa dependendo de uma lista de condições. O bloco de comandos cuja condição é satisfeita é aquele executado. Na hipótese de nenhuma das condições ser satisfeita, então o bloco referente à condição OUTRO_CASO

é aquele executado, desde que ele tenha sido declarado.

Sintaxe:	
FACA CASO	
CASO <*expressão*>:	
<*bloco de comandos*>	
⋮	
CASO <*expressão*>:	*opcional*
<*bloco de comandos*>	*opcional*
OUTRO_CASO [<*expressão*>]:	*opcional*
<*bloco de comandos*>	*opcional*
FIM_CASO	

Exemplo:

```
// Algoritmo para ler um valor numerico
// se igual a 1 entao imprimir o valor
// se igual a 2 entao imprimir o dobro
// se igual a 3 entäo imprimir o triplo
// se igual a 4 entäo imprimir o quadruplo
// -------------------------------------
// Primeira Versao
// --------------
variaveis
    numerico val
inicio
    limpar
    ler val
    faca caso
    caso val=1:
        escrever "simples   = ", val
    caso val=2:
        escrever "dobro     = ", val * 2
    caso val=3:
        escrever "triplo    = ", val * 3
    caso val=4:
        escrever "quadruplo = ", val * 4
    fim_caso
fim
//
// Segunda Versao
// --------------
variaveis
    numerico val
inicio
    ler val
    faca caso
    caso val=1:
        escrever "simples   = ", val
    caso val=2:
        escrever "dobro     = ", val * 2
    caso val=3:
        escrever "triplo    = ", val * 3
    caso val=4:
        escrever "quadruplo = ", val * 4
    outro_caso:
        escrever "outros casos ", val
    fim_caso
fim
```

B. FACA ENQUANTO

Especifica um laço de repetição. Um bloco de comandos é executado repetitivamente enquanto o resultado da condição especificada for verdadeiro.

Sintaxe:
FACA ENQUANTO <*expressao*>
<*bloco de comandos*>
FIM_ENQUANTO

Exemplo:

```
// Algoritmo para ler um nome e 3 notas
// calcular e imprimir nome e media aritmetica
// das notas
// se o nome for igual a fim, terminar o algoritmo
// -----------------------------------------------
variaveis
    numerico not1, not2, not3, med
    caracter nome
inicio
    limpar
    ler nome
    faca enquanto nome <> "fim"
    ler not1
    ler not2
    ler not3
    med = (not1 + not2 + not3) / 3
    escrever "nome  = ", nome
    escrever "nota1 = ", not1
    escrever "nota2 = ", not2
    escrever "nota3 = ", not3
    escrever "media = ", med
    ler nome
    Fim_enquanto
fim
```

C. FIM

Indica ao interpretador o final da função ou de um bloco de comandos no algorítmo principal.

Sintaxe:
FIM

Exemplo:

```
// O menor algoritmo em ILA
// ------------------------
inicio
fim
```

D. INICIO

Indica ao interpretador o início da função ou de um bloco de comandos no algorítmo principal.

Sintaxe:
INICIO

E. PARA PROXIMO

Especifica um laço de repetição. Um bloco de comandos é executado repetitivamente enquanto a variável *var* for menor que a condição determinada pela expressão *final*.
No início, a variável *var* tem seu valor determinado pela expressão *inicial*. A cada ciclo no laço, o valor de *var* é incrementado pelo conteúdo expressão *incr*. Desta maneira a execução do laço se repete até que o limite especificado por *final* é atingido. Caso a expressão *incr* seja omitida, a variável *var* passa a ter incremento unitário.

Sintaxe:
PARA <*var*> = <*inicial*> ATE <*final*> [PASSO <*incr*>]
<*bloco de comandos*>
PROXIMO

Exemplo 1:

```
// Algoritmo para gerar os numeros entre 1 e 10
// --------------------------------------------
variaveis
    numerico i
inicio
    para i=1 ate 10
        escrever "numero gerado = ", i
```

```
    proximo
fim
```

Exemplo 2:

```
// Algoritmo para gerar os numeros pares entre 1 e 25
// -------------------------------------------------
variaveis
    numerico i
inicio
    para i=2 ate 24 passo 2
        escrever "numero gerado = ", i
    proximo
fim
```

F. RETORNAR

Termina a execução de uma função (subrotina) e retorna um valor, quando este é especificado.

Sintaxe:
RETORNAR([<*expressão*>])
ou
RETORNAR <*expressão*>

G. SE

Especifica uma estrutura de decisão. Provoca o desvio do fluxo do programa dependendo da condição especificada por *expressão*. Se a condição é satisfeita, o bloco de comandos referente ao SE é executado; caso contrário, é executado aquele referente ao SENAO.

Sintaxe:	
SE <*expressão*> ENTAO	
<*bloco de comandos*>	
SENAO	*opcional*
<*bloco de comandos*>	*opcional*
FIM_SE	

Exemplo 1:

```
// Algoritmo para ler um valor e, se o valor
// for 0, entao imprimir o valor lido
// caso contrario, imprimir o valor 10
// ----------------------------------------
variaveis
    numerico val
inicio
    ler val
    se val = 0 entao
        escrever "valor lido =", val
    senao
        escrever 10
    Fim_se
fim
```

Exemplo 2:

```
// Algoritmo que le um numero e
// imprime se ele for maior que 10
// -------------------------------
variaveis
    numerico val
inicio
    ler val
    se val > 10 entao
    escrever "este numero é maior do 10 => ", val
    Fim_se
fim
```

VIII. OUTRAS FUNÇÕES

A. ESPERAR

Suspende a execução do algorítmo durante o intervalo de tempo especificado em segundos.

Sintaxe:
ESPERAR(<expressão>)

B. TESTEI

Habilita a execução simultânea de um depurador durante a interpretação. Esta função mostra, durante a execução do programa, na parte inferior da tela, o nome da variável corrente e seu conteúdo. Independentemente da habilitação da função TESTEI(), o interpretador gera um arquivo, denominado MESA.MAP, que contém informação sobre as variáveis modificadas durante a execução do programa.

Sintaxe:
TESTEI()

C. TESTEF

Desabilita a execução simultânea do depurador durante a interpretação.

Sintaxe:
TESTEF()