Núcleo de Informática Aplicada à Educação
Universidade Estadual de Campinas

RESUMO

Este texto dá continuidade ao trabalho iniciado no Memo Nº 35, intitulado: *Tartarugas, Figuras, Palavras, Listas e Procedimento: Um Primeiro Passeio pelo Logo – SuperLogo*. Agora, são abordados novos tópicos de programação: definição de procedimentos com parâmetros, área de trabalho, comandos condicionais, recursão e programas que "conversam" com o computador. A seleção dos tópicos de programação baseia-se na demanda experenciada pelas autoras durante os cursos de formação de professores na área de Informática e Educação.

NIED - Memo Nº 36
2000

Parâmetros, Condicionais, Recursão... continuando o passeio pelo Logo–SuperLogo 3.0

Heloísa Vieira da Rocha (e-mail: heloisa@dcc.unicamp.br)
Fernanda M. P. Freire (e-mail: ffreire@unicamp.br)
Maria Elisabette B.B. Prado (e-mail: bprado@unicamp.br)

Núcleo de Informática Aplicada à Educação - Nied
Universidade Estadual de Campinas - Unicamp
Cidade Universitária "Prof. Zeferino Vaz"
Prédio V da Reitoria - 2º Piso
13083-970 - Campinas - SP
Telefones: (019) 788-7350 ou 788-8136
Fac-símile: (019) 788-7350 ou 788-8136 (Ramal 30)
http://www.nied.unicamp.br

CONTEÚDO

Parâmetros, Condicionais, Recursão...
Continuando o passeio pelo Logo

INTRODUÇÃO

O conjunto de ações primitivas do Logo é subdividido em duas categorias: COMANDOS e OPERAÇÕES. Comandos são ações que sempre produzem uma ação explícita: um desenho, a impressão de algum valor, etc.. Por exemplo, PF é um comando que resulta em um deslocamento da tartaruga. ROTULE é um comando que resulta na escrita de um valor ou mensagem na Janela Gráfica. Outros exemplos são PD, PE, PT, UL, etc..

Ao contrário dos comandos, as operações não produzem uma ação explícita mas, sim, valores que ficam disponíveis para serem usados por outros comandos. Por exemplo, suponhamos que a tartaruga esteja posicionada na coordenada 80 do eixo X da Janela Gráfica. Para obter essa informação, pode-se pedir para escrever na tela o valor da coordenada X usando:

```
ROTULE COORX
80
```

Se for digitado:

```
COORX
```

o Interpretador Logo devolverá uma mensagem de erro do tipo:

```
não disse o que fazer com 80
```

Isto porque, o valor retornado pela operação COORX deve ser usado como entrada de um comando (por exemplo: rotule coorx).

Uma operação, portanto, sempre é utilizada como parâmetro de um comando ou de uma outra operação[2]. Por exemplo: MO POS. Outras operações análogas são: DÇ, COORY etc..

A partir do conjunto de primitivas da linguagem pode-se definir novos procedimentos para serem usados nos mais variados contextos. Neste texto vamos abordar conceitos computacionais que permitem uma maior amplitude no uso da linguagem Logo.

Procedimentos com parâmetros

Em Logo, há comandos como UN, DT, que não usam parâmetro e comandos, como PF, REPITA, MUDEPOS que precisam de um ou mais parâmetros. A natureza do parâmetro pode ser diversificada: número, palavra ou lista. Há ainda comandos como o ROTULE que aceitam parâmetros de qualquer natureza.

Ao definirmos um procedimento como:

```
APRENDA QUADRADO
REPITA 4 [ PF 70 PD 90 ]
FIM
```

estamos construindo um comando de nome QUADRADO que não precisa de parâmetro e que sempre produz o desenho de um quadrado cujo lado mede 70 "passos de tartaruga". Este tipo de procedimento define um comando que é análogo a comandos como UN ou AT que dispensam o uso de parâmetros.

É possível, também, construir procedimentos análogos ao comando PF, ou seja, que necessitam valores para a sua execução. A definição deste tipo de procedimento possibilita a construção de procedimentos mais genéricos. Pode-se definir um procedimento de nome QUADRADO que desenha a figura de um quadrado cujo tamanho é especificado pelo parâmetro dado:

```
APRENDA QUADRADO :lado [3]
REPITA 4 [ PF :lado PD 90 ]
```

[1] Os exemplos apresentados usam os comandos da versão SuperLogo 3.0. Todos eles podem ser adaptados e utilizados em outras versões de Logo. Nesta versão os comandos com acentuação deverão ser digitadas com letras minúsculas.
[2] Para maiores detalhes sobre esse tópico, consultar: Rocha,H.V., Freire,F.M.P.,Prado,M.E.B.B..(199() Tartarugas, Figuras, Palavras, Listas e Procedimento: Um Primeiro Passeio Pelo Logo. MEMO nº 35. Campinas: NIED - UNICAMP.
[3] Estamos convencionando a escrita do nome do procedimento em letras maiúsculas e o nome do parâmetro em minúsculas.

FIM

Em Logo, há duas notações importantes: uma utiliza aspas (") e, a outra, o sinal de dois pontos (:). Quando se escreve:

```
"lado
```

estamos nos referindo ao **nome lado** ou à palavra lado. Mas, quando se escreve:

```
:lado
```

estamos nos referindo ao **valor representado pelo nome lado** (ou pela palavra lado).

Portanto, no caso de procedimentos com parâmetros, estes sempre serão precedidos por dois pontos (:). Esta notação indica que será usado o valor do nome que representa o parâmetro e que é informado no momento da chamada do procedimento.

O procedimento será utilizado da mesma forma que qualquer comando Logo com parâmetro, isto é, para desenhar um quadrado de lado 50 deve-se digitar:

```
QUADRADO 50
```

e para desenhar um quadrado de lado 25:

```
QUADRADO 25
```

Vamos ver outro exemplo. Suponha que se queira um procedimento que desenhe um quadrado, a sua diagonal e retorne a tartaruga ao ponto inicial. A diagonal do quadrado é a raiz quadrada de 2 multiplicada pelo comprimento do lado. O procedimento faz exatamente isso:

```
APRENDA DIAGONAL
REPITA 4 [ PF 50 PD 90]
PD 45
PF 50 * RAIZQ 2 [4]
PT 50 * RAIZQ 2
```

[4] Neste caso, como o lado do quadrado é 50, o resultado da raiz quadrada de 2 multiplicada por 50 é igual a 70.

```
PE 45
FIM
```

Para desenhar um quadrado de qualquer tamanho e sua respectiva diagonal, pode-se definir o procedimento DIAGONAL com um parâmetro que representa o valor do lado que será desenhado o quadrado:

```
APRENDA DIAGONAL :lado
REPITA 4 [PF :lado PD 90]
PD 45
PF :lado * ( raizq 2 )
PT :lado * ( raizq 2 )
PE 45
FIM
```

```
DIAGONAL 100
```

Definindo operações: o comando envie

O procedimento QUADRADO apresentado no item anterior produz um resultado na tela gráfica: o desenho da figura de um quadrado do tamanho dado como entrada. Por esta razão, este tipo de procedimento é definido como um **comando**.

De forma análoga, pode-se definir operações, como:

```
APRENDA DOBRO :número
ENVIE :número * 2
FIM
```

que retorna o dobro de um número dado como parâmetro.

Assim, se for digitado:

```
ESC DOBRO 173
```

aparecerá escrito na Janela de Comandos o número:

```
346
```

O que torna possível a definição de operações em Logo é o comando ENVIE. O ENVIE é um comando especial que só pode ser utilizado dentro de procedimentos e nunca no modo direto. Ele necessita de um parâmetro que pode ser de qualquer natureza: palavra, número ou lista.

Se o parâmetro do ENVIE for uma palavra, o procedimento que o usa em sua definição, também retornará uma palavra. No nosso exemplo, o parâmetro do ENVIE é um número (resultante da operação de multiplicação) e a saída de DOBRO também é um número. Portanto, o ENVIE retorna como saída do procedimento do qual faz parte, um parâmetro de natureza idêntica à do seu próprio parâmetro.

Todo procedimento que possui o ENVIE é uma operação porque retorna um valor para ser utilizado por outros comandos ou operações:

```
PF DOBRO 100
```

ou mesmo:

```
ESC (DOBRO 148) - (DOBRO 123)
```

DOBRO é análogo a operações primitivas do Logo como: RAIZQ, SEN, COS, entre outras.

Procedimentos como:

```
APRENDA ÉA :qualquer
ENVIE (ULT :qualquer) = "a
FIM
```

são denominados **predicados**. Predicados são operações que retornam valores booleanos, isto é, VERD ou FALSO. O procedimento ÉA retorna VERD se o último elemento do parâmetro dado for o caracter **a**, caso contrário, retorna FALSO. Por exemplo:

```
ESC ÉA "boba

verd
```

```
ESC ÉA  340
falso
```

```
ESC ÉA  [ c b a ]
verd
```

```
ESC ÉA  [ 340 boba ]
falso
```

Procedimentos com mais de um parâmetro

Pode-se definir procedimentos em Logo com mais de um parâmetro. Escolhe-se um nome para cada um dos parâmetros e inclui-se os mesmos no cabeçalho do procedimento. Por exemplo, para se desenhar retângulos de diversos tamanhos pode-se definir:

```
APRENDA RETÂNGULO :altura :comprimento
PF :altura   PD 90
PF :comprimento  PD 90
PF :altura  PD 90
PF :comprimento  PD 90
FIM
```

Com ele, pode-se desenhar um retângulo de qualquer tamanho, de acordo com o valor dado como entrada no momento da chamada:

```
RETÂNGULO 70 10
```

ou, um quadrado, se dermos como valores dos parâmetros algo como:

```
RETÂNGULO 70 70
```

Também as operações podem ser definidas com mais de um parâmetro:

```
APRENDA PORCENTO :porcentagem :número
ENVIE (:porcentagem * :número) / 100
FIM
```

Com esta operação, pode-se obter uma determinada porcentagem a partir dos números dados como entrada:

```
ESC PORCENTO 170 15
25.5
```

```
ESC PORCENTO 50 5
2.5
```

```
ESC PORCENTO 5 50
2.5
```

Nomeação de parâmetros: o conceito de nomes locais

Definir um procedimento Logo significa agrupar uma série de comandos sob um nome escolhido pelo programador. O novo procedimento pode ser visto como parte do vocabulário da linguagem Logo porque ele representa uma nova palavra cujo significado é descrito pelo procedimento. Usar parâmetros também envolve esse processo de nomeação mas, em um sentido diferente. Os nomes dos parâmetros, são locais ao procedimento que os usam como entrada, ou seja, somente o próprio procedimento conhece estes nomes.

Uma vez que os nomes dos parâmetros são locais, diferentes procedimentos podem usar os mesmos nomes de parâmetros sem que haja qualquer interferência entre eles. Isso é análogo a pensarmos que existem diversas famílias com pessoas que possuem um mesmo nome e, na verdade, são pessoas diferentes.

Outra maneira de pensar, é imaginar que cada procedimento tem uma biblioteca particular. Toda vez que o procedimento é chamado ele acessa esta biblioteca particular que associa ao nome dos parâmetros os valores especificados na chamada. Quando o procedimento avalia uma linha que contém um nome de parâmetro (precedido de dois pontos) o Interpretador busca o valor daquele nome na biblioteca e substitui-o pelo valor. Por exemplo, o procedimento RETÂNGULO chamado com:

```
RETÂNGULO 70 10
```

teria associado a ele uma biblioteca como a mostrada na figura 1.

altura	70
comprimento	10

Figura 1: biblioteca particular quando da execução de RETÂNGULO 70 10

Os valores dados de entrada no momento em que o procedimento é chamado são associados aos nomes na ordem em que aparecem na linha título do procedimento. Neste caso, o primeiro valor, 70, é associado ao nome do primeiro parâmetro **altura** e, o segundo valor, 10, é associado ao nome do segundo parâmetro **comprimento**.

Como se sabe, um comando em um procedimento pode ser a chamada de um outro procedimento (subprocedimento). Como cada procedimento tem sua própria biblioteca particular de nomes não existe conflito entre procedimentos que utilizam nomes iguais para seus parâmetros. Por exemplo, o procedimento RETÂNGULO pode ser utilizado como parte de um procedimento para desenhar uma bandeira, como mostra a figura 2:

```
APRENDA BANDEIRA :altura
PF :altura
RETÂNGULO  (:altura/2) :altura [5]
PT :altura
FIM
```

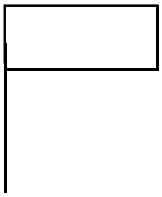


Figura 2: bandeira desenhada por BANDEIRA 70

O procedimento BANDEIRA desenha um "mastro" de tamanho **altura** e então desenha no topo do mastro um retângulo de tamanho **altura**/2 **altura** e, em seguida, move a Tartaruga de volta para a base do mastro.

Vamos examinar detalhadamente como é avaliado o comando BANDEIRA 70:

[5] Observe o uso de parênteses em (:altura /2). Isto não é uma exigência do Logo mas, torna o programa mais fácil de ser lido.

Ele cria uma biblioteca particular para BANDEIRA dentro da qual existe o nome **altura** associado ao valor 70 e inicia a avaliação da definição de BANDEIRA começando pela primeira linha:

```
PF :altura
```

Olhando na biblioteca particular de Bandeira, o Interpretador encontra o valor 70 associado ao nome **altura**, e então ele faz a tartaruga andar PARAFRENTE 70. A seguir é preciso avaliar a próxima linha:

```
RETÂNGULO (:altura / 2) :altura
```

Para executar este comando, primeiro, o Interpretador calcula os valores que serão passados para RETÂNGULO. O primeiro valor é a metade do valor de **altura**, isto é, 35; e, o segundo valor, é o próprio valor de **altura**, que é 70. Para fazer este cálculo ele utiliza a biblioteca particular de RETÂNGULO. A partir daí fica estabelecido que o comando a ser executado é RETÂNGULO 35 70. Isto faz com que seja criada a biblioteca particular de BANDEIRA, na qual os nomes dos parâmetros de RETÂNGULO **altura** e **comprimento** são associados aos valores 35 e 70 respectivamente. Um esquema destas chamadas pode ser visto na figura 3. Embora o nome **altura** seja associado a 70 em BANDEIRA e a 35 em RETÂNGULO, isso não cria qualquer conflito, pois cada procedimento consulta sua própria biblioteca.

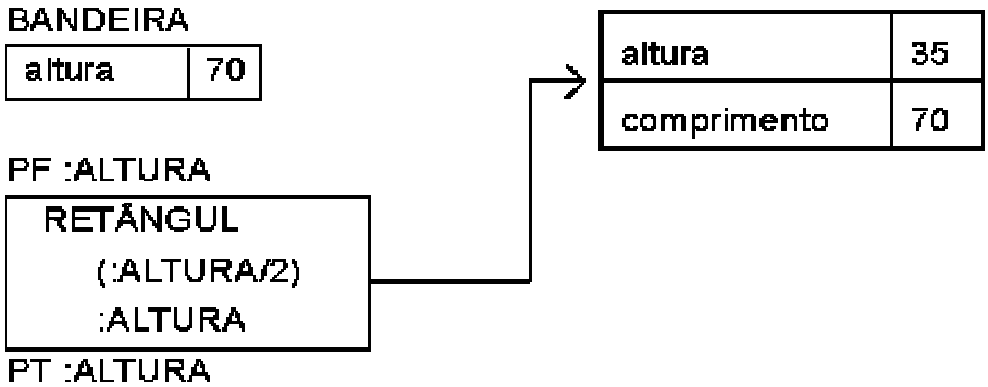


Figura 3: bibliotecas particulares criadas a partir da chamada de BANDEIRA 70

O conceito de nomes locais é importante pelo fato de se poder definir procedimentos com parâmetros sem precisar se preocupar com detalhes relacionados a escrita do procedimento, podendo-se concentrar mais naquilo que o procedimento faz. Quando se está construindo o

procedimento BANDEIRA, pode-se olhar o procedimento RETÂNGULO como um todo, sem observar como foi escrito o procedimento (que nomes foram dados aos seus parâmetros). O interessante é que ao escrever o procedimento BANDEIRA o procedimento RETÂNGULO pode ser tratado como se fosse mais uma primitiva do sistema Logo (se estiver disponível na área de trabalho naquele instante).

A técnica de enxergar um procedimento (mesmo um procedimento complexo) como um módulo cujos detalhes não precisamos nos preocupar é uma idéia crucial em programação. Cada vez que se define um novo procedimento, pode-se usá-lo como uma parte de procedimentos mais complexos e, desta forma, pode-se construir processos muito complexos.

Como ilustração, uma vez definida a BANDEIRA pode-se construir um outro procedimento que faz a tartaruga desenhar uma bandeira e caminhar um pouco para a direita:

```
APRENDA BANDANDE :tamanho :distância
UL    BANDEIRA :tamanho
UN    PD 90
PF :distância  PE 90
FIM
```

e, utilizar este procedimento para desenhar uma linha de diversas bandeiras como mostra a figura 4:

```
APRENDA LINHABAND :quantas :tamanho :distância
REPITA :quantas [ BANDANDE :tamanho :distância]
FIM
```

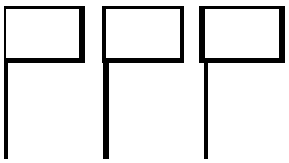


Figura 4: desenho feito por LINHABAND 3 40 50

Procedimentos com parâmetros de natureza diversa

Até agora vimos procedimentos cuja entrada é um número mas, se observarmos o elenco de primitivas do Logo, descobriremos que os parâmetros também podem ser listas ou palavras,

como nos casos de: MUDEPOS OU MUDECL. Isso implica que também é possível construir procedimentos cuja entrada pode ser uma palavra ou uma lista, dependendo do tipo de uso.

Há muitas formas de usar palavras e listas em Logo. Pode-se, para ilustrar melhor essa idéia, definir um procedimento que escreva os ingredientes de um bolo de limão:

```
APRENDA INGREDIENTES
ROTULE [ Bolo de limão ]
UN PT 20
ROTULE [ 4 ovos ]
PT 20
ROTULE [ 2 copos de açúcar ]
PT 20
ROTULE [ 1 copo de leite ]
PT 20
ROTULE [ 3 copos de farinha ]
PT 20
ROTULE [ 1 colher de fermento ]
PT 20
ROTULE [ 2 xícaras de suco de limão ]
FIM
```

Esta relação de ingredientes pode ser considerada "padrão" e, por esta razão, pode ser usada para vários outros bolos: de laranja, de abacaxi, de morango, etc.. Para tanto, pode-se modificar o procedimento de modo que ele escreva os ingredientes de bolos de diferentes sabores:

```
APRENDA INGREDIENTES :sabor
ROTULE SN [ Bolo de ] :sabor
UL PT 20
ROTULE [ 4 ovos ]
PT 20
ROTULE [ 2 copos de açúcar ]
PT 20
ROTULE [ 1 copo de leite ]
PT 20
ROTULE [ 3 copos de farinha ]
PT 20
ROTULE [ 1 colher de fermento ]
PT 20
ROTULE SN [ 2 xícaras de suco de ]  :sabor
FIM
```

```
INGREDIENTES "laranja
```

o computador escreverá a receita colocando a palavra *laranja* nos locais adequados e assinalados pela presença do parâmetro: *Bolo de laranja* e *2 xícaras de suco de laranja*. O mesmo ocorrerá se o parâmetro for:

```
INGREDIENTES "morango
```

Suponha que se queira fazer o desenho do bolo e pintá-lo com a cor do lápis correspondente à cor do sabor do bolo. É preciso definir um procedimento que desenhe e pinte o bolo. Algo como:

```
APRENDA BOLO :cor
...
...                          comandos que desenham um bolo
...
UL
MUDECP :cor
.....
FIM
```

Em seguida, pode-se juntar os dois procedimentos afim de usar uma única chamada e obter os ingredientes e o desenho:

```
APRENDA RECEITA :sabor :cor
```

```
INGREDIENTES :sabor
BOLO :cor
FIM
```

O lugar do desenho na tela também pode ser escolhido. Para tanto é preciso definir mais um parâmetro para representar a posição desejada do desenho (par de coordenadas x e y). Tem-se então:

```
APRENDA RECEITA :sabor  :lugar :cor
INGREDIENTES :sabor
UN MUDEPOS :lugar UL
BOLO :cor
FIM
```

O resultado final é um procedimento que utiliza três parâmetros: uma palavra - que serve para escrever os ingredientes do bolo com o sabor desejado; uma lista - que posiciona a tartaruga em um certo ponto da tela; e um número - que indica a cor correlacionada ao sabor do bolo. Ao efetuarmos a chamada:

```
RECEITA "morango  [-30 0]  4
```

obtém-se a receita do bolo de morango e o desenho de um bolo pintado de vermelho.

Área de trabalho

Quando se define um procedimento em Logo, costuma-se dizer que o novo procedimento passa a fazer parte do elenco de primitivas do Logo. Essa afirmativa é relativamente verdadeira.

A implementação da linguagem Logo reserva um espaço de memória para o usário trabalhar. Esse espaço é denominado de área de trabalho. Assim tudo que é feito pelo usuário durante uma sessão Logo fica disponível, na área de trabalho: procedimentos definidos, figuras definidas, etc..

Quando se sai do Logo toda e qualquer informação existente na área de trabalho é perdida porque ela é volátil. Para que isso não ocorra é necessário organizar e armazenar os conteúdos da área de trabalho de forma permanente, isto é, em disquete ou disco rígido na forma de arquivos.

O gerenciamento da área de trabalho se faz através de comandos primitivos do Logo como: SALVE, CARREGUE, ELTUDO, MOTS, MOPS, entre outros.

Portanto, quando se diz que a definição de procedimentos expande o elenco de primitivas do Logo, isto deve ser compreendido metaforicamente. Na verdade, os procedimentos definidos pelo usuário A poderão fazer parte de um arquivo que, se carregado para a área de de trabalho, estará disponível para ser reutilizado mas não será parte da linguagem Logo propriamente dita. Os procedimentos de um arquivo X do usuário A, por exemplo, não farão parte da área de trabalho do usuário B a não ser que este último carregue o arquivo X para a sua própria área de trabalho.

Comandos condicionais

O comando SE é chamado de comando condicional. Ao ser usado, modifica o fluxo de execução de um programa, como veremos mais adiante. Ele tem a forma:

SE	alguma condição é verdadeira	faça alguma ação

O primeiro parâmetro, a condição, é um predicado que retorna VERD ou FALSO[6]_. Qualquer predicado primitivo do Logo ou qualquer predicado definido pelo usuário pode ser usado como parâmetro do SE. A ação a ser executada no caso de a condição ser verdadeira é uma lista de instruções. Quando o resultado do predicado é FALSO, a lista não é executada e o Interpretador Logo continua a execução do programa no comando imediatamente seguinte ao comando SE

A forma geral do comando SE, portanto, é:

SE <predicado> <lista>

Em algumas versões do Logo existem outros comandos condicionais como SEVERD e SEFALSO. No SuperLogo, por exemplo, pode-se usar o comando SE com 3 parâmetros:

[6] Geralmente em Logo os predicados se iniciam por É, como ÉNÚMERO, ÉVAZIA, ÉPALAVRA. São considerados também predicados os operadores Lógicos: E, NÃO, ALGUM e os operadores relacionais: >, <, =.

```
(SE   <predicado>   <lista1>   <lista2>)
```

Caso o predicado seja VERD é executada a *lista1*, caso contrário, é executada a *lista2.* Esta forma do comando SE é análoga à forma do comando SENÃO usado em outras versões do Logo.

Recapitulemos o exemplo do quadrado com parâmetro. Dependendo do parâmetro dado, o desenho ultrapassa as dimensões da tela. Pode-se "testar" o tamanho do lado para contornar este problema com o auxílio do comando SE:

```
APRENDA QUADRADO :lado
TAT
SE :lado > 200  [  UN  PT 200  UL ]
REPITA 4 [ PF :lado PD 90 ]
FIM
```

Assim, se o procedimento for chamado com:

```
QUADRADO 100
```

a tartaruga desenhará um quadrado de lado igual a 100 passos a partir do centro da tela. Mas, se o procedimento for chamado com:

```
QUADRADO 210
```

a tartaruga andará 200 passos para trás e depois desenhará o quadrado com o tamanho solicitado.

Para desenhar quadrados ainda maiores dentro dos limites da tela, podemos sofisticar a condição a ser testada, acrescentando um número mínimo e máximo do tamanho do lado do quadrado, atrelado a um certo deslocamento da tartaruga:

```
APRENDA QUADRADO :lado
TAT
SE  E  (:lado > 200)  (:lado < 400)   [ DESLOCATAT]
REPITA 4 [ PF :lado PD 90 ]
FIM
```

```
APRENDA DESLOCATAT
UN  PT  200  PE  90
PF 155  PD  90  UL
FIM
```

Neste caso, a tartaruga somente se deslocará na tela caso os dois predicados aritméticos sejam VERD. Os resultados de " > " e de " < " são retornados ao operador lógico E que, por sua vez, retorna o resultado ao comando SE que executa a lista de instruções. Isto mostra que o primeiro parâmetro do comando SE pode ser um encadeamento de vários predicados cujo resultado final é VERD ou FALSO.

O que acontece se for digitado:

```
QUADRADO 500
```

O valor do parâmetro ultrapassa o intervalo de tamanho descrito no programa. Assim, como o resultado do operador lógico é FALSO, a tartaruga permanece no centro da tela. A linha seguinte do programa, que desenha a figura do quadrado é, então, executada. A tartaruga desenha a figura do quadrado de lado 500 que ultrapassa os limites da tela. Para contornar isso, pode-se acrescentar uma nova condição:

```
APRENDA QUADRADO :lado
TAT
SE OU  (:lado > 400) (:lado = 400) [ ESC [ Este quadrado ultrapassa os limites da tela ] ]
SE  E  (:lado > 200)  (:lado < 400)  [  DESLOCATAT ]
REPITA 4 [ PF :lado PD 90 ]
FIM
```

Se, agora, for solicitado:

```
QUADRADO 500
```

o Interpretador imprimirá a mensagem:

```
este quadrado ultrapassa os limites da tela
```

e desenhará um quadrado de lado igual a 500.

Na verdade, o objetivo é que o desenho de QUADRADO 500 não seja feito. Mas, da maneira como o procedimento está definido, o desenho ainda é executado. Vejamos porque isso acontece.

Inicialmente, o Interpretador Logo executa a primeira linha do procedimento:

```
SE ALGUM :lado > 400 :lado = 400 [ ESC [ este quadrado ultrapassa os limites da tela ] ]
```

A condição é satisfeita, retornando ao comando SE o valor booleano VERD, fazendo com que a lista de instruções seja executada. Em seguida, o Interpretador executa a segunda linha do procedimento:

```
SE  E  (:lado > 200)  (:lado < 400)  [  DESLOCATAT ]
```

Como a segunda condição não é satisfeita, o comando SE recebe FALSO e não executa a lista de instruções. Em seguida, o Interpretador executa a linha seguinte:

```
REPITA 4 [ PF :lado PD 90 ]
```

e desenha o quadrado de lado 500 a partir do centro da tela. Portanto, o problema ainda não está resolvido.

É necessário **modificar o fluxo de execução do programa.** No caso do valor do tamanho ser igual ou maior do que 400, o programa deve imprimir uma mensagem e ser **interrompido**. Isso é obtido através do uso do comando PARE, que interrompe a execução do programa no momento em que é executado:

```
APRENDA QUADRADO :lado
SE ALGUM :lado > 400 :lado = 400 [ ESC [ este quadrado ultrapassa
                                        os limites a tela ] PARE ]
SE  E  (:lado > 200)  (:lado < 400)  [ DESLOCATAT ]
REPITA 4 [ PF :lado PD 90 ]
FIM
```

Vejamos o procedimento FIGQUA que usa QUADRADO como subprocedimento:

```
APRENDA FIGQUA :lado
```

```
QUADRADO :lado
ESC [ vamos brincar novamente ]
FIM
```

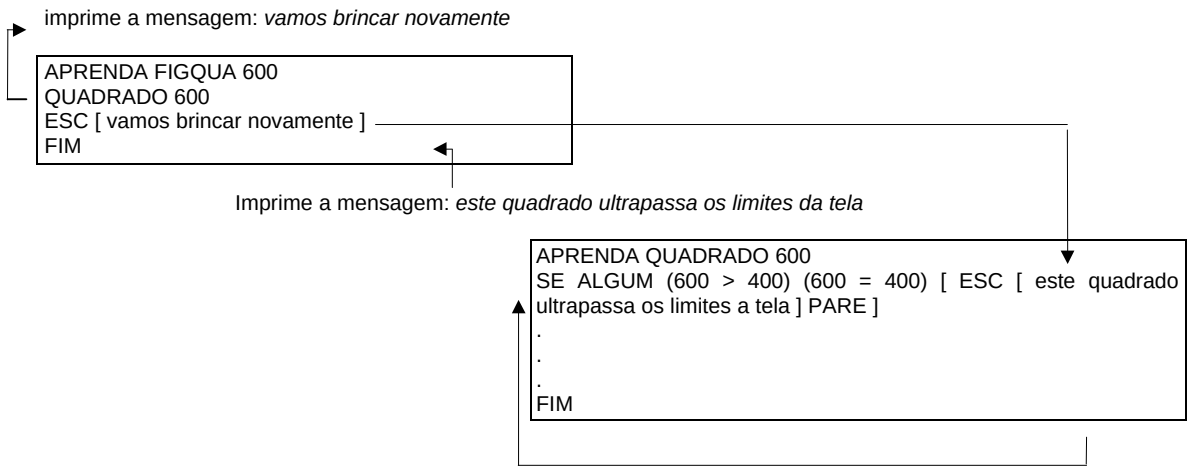
Se for executado FIGQUA 300, o procedimento desenhará um quadrado de lado 300 e imprimirá a mensagem:

```
vamos brincar novamente
```

Se for solicitado FIGQUA 600, o procedimento imprimirá duas mensagens:

```
este quadrado ultrapassa os limites da tela
vamos brincar novamente
```

Observemos o fluxo de execução de FIGQUA 600:



Isto é exatamente o mesmo que acontece quando o procedimento executa a última linha de seu código. Pode-se imaginar que sempre existe um comando PARE antes do FIM, pois o comportamento é o mesmo.

Vejamos no caso das operações o que acontece. A operação abaixo retorna a palavra **começo** quando as suas entradas iniciam com um mesmo caracter, retorna a palavra **final** quando as suas entradas terminam com um mesmo caracter ou retorna a palavra **diferente** quando não ocorre nenhuma das duas situações.

```
APRENDA PRIULT :palavra1 :palavra2
SE ( PRI :palavra1) = ( PRI :palavra2) [ ENVIE "começo ]
```

```
SE ( ULT :palavra1) = (ULT :palavra2) [ENVIE "final ]
ENVIE "diferente
FIM
```

Portanto, se for solicitado:

```
ESC PRIULT "bode "cabra
```

o resultado será:

diferente

Mas, se for pedido:

```
ESC PRIULT "ovo "galo
```

ou

```
ESC PRIULT "elefante "exército
```

serão escritas as palavras **final** e **começo**, respectivamente. Nos dois casos, embora o procedimento ainda tenha outras instruções, elas não são executadas. Isso acontece porque o comando ENVIE tem, também, a função de **interromper** o procedimento, como se ele tivesse um comando PARE na sua definição. Portanto, as operações não necessitam do comando PARE para indicar o final de execução do procedimento em um ponto diferente do FIM. Assim, se utilizarmos:

```
ESC PRIULT "casa "cola
```

o resultado será:

começo

Recursão

Como já se sabe, quando se deseja repetir uma certa ação um determinado número de vezes conhecido, pode-se usar o comando REPITA. Entretanto, há contextos em que não se

sabe "a priori" quantas vezes será necessário repetir uma seqüência de passos para se conseguir um resultado desejado. O programa seguinte é um ótimo exemplo:

```
APRENDA POLI :lado :ângulo
PF :lado
PD :ângulo
POLI :lado :ângulo
FIM
```

A tartaruga fica se movendo muitas e muitas vezes de acordo com os valores dados como entrada, até que se interrompa a execução do procedimento (acionando o botão PARAR da Janela de Comandos). Modificando-se os valores dos parâmetros, pode-se obter muitas figuras diferentes. Todas as figuras produzidas são fechadas mas, o número de lados que são desenhados antes de a figura se fechar depende de uma relação complexa envolvendo o valor de **ângulo** dado como entrada. Usando-se este tipo de repetição indefinida, o procedimento torna-se extremamente simples.

Recursão é o nome que se dá, em programação, ao uso do nome POLI como parte da definição de POLI. Ou, de maneira mais geral, de se escrever **procedimentos que chamam a si mesmos**.

O procedimento recursivo POLI tem uma forma muito simples: ele repete um ciclo (andar e girar) que mantém uma certa regularidade. Recursão é uma idéia muito mais poderosa e pode ser usada para obter efeitos sofisticados. Vejamos um exemplo:

```
APRENDA CONTAR :num
ESC :num
CONTAR :num - 1
FIM
```

Examine o que acontece quando é dado o comando:

```
CONTAR 10
```

Para entender o efeito deste comando, é preciso olhar, novamente, a definição de CONTAR. Pode-se ver que ele tem um parâmetro chamado **num**. Neste caso, foi dado o valor 10 como entrada, e portanto o procedimento assume **num** como sendo 10.

A primeira linha diz:

```
ESC :num
```

então, é escrito o valor 10 e, em seguida, é avaliada a próxima linha,

```
CONTAR :num -1
```

e, neste caso,

```
CONTAR 9
```

Este comando causa o mesmo efeito de se teclar o comando:

```
CONTAR 9
```

que escreveria o valor 9 e, em seguida, avaliar:

```
CONTAR 8
```

e, assim por diante. Resumindo, o resultado de:

```
CONTAR 10
```

é escrever:

10
9
8
7
6
5
4
3
2
1
0
-1
-2
-3 ... até interromper a execução com o botão PARAR da Janela de Comandos

Um outro exemplo da mesma técnica de programação é a seguinte modificação do procedimento POLI:

```
APRENDA POLI :lado :ângulo
PF :lado
PD :ângulo
POLI (:lado + 3 ) :ângulo
FIM
```

Digitando-se o comando:

```
POLI 0 90
```

ocorre a seguinte seqüência de movimentos da tartaruga:

```
PF 0
PD 90
PF 3
PD 90
PF 6
PD 90
PF 9
PD 90 ...
```

que produz uma espiral quadrada. Mudando-se o **ângulo** dado de entrada pode-se obter toda sorte de espirais. Parte do poder da recursão é o fato de programas muito simples, como POLI, conduzirem a uma variedade de resultados, muitos deles não esperados.

Programando a interrupção de um comando recursivo

Os procedimentos recursivos vistos até agora são interrompidos através das teclas CRTL e BREAK. Outra maneira de provocar a interrupção de procedimentos deste tipo é através de "testes". Suponha que se queira parar o procedimento CONTAR antes que ele escreva o valor 0:

```
APRENDA CONTAR :num
SE (:num = 0)  [ PARE]
ESC :num
CONTAR :num -1
FIM
```

O comando SE testa se o valor de **num** é zero. Se for, o procedimento CONTAR pára. Isto é, ao invés de continuar na próxima linha do procedimento, ele retorna o controle para o procedimento que chamou o CONTAR. Portanto em resposta ao comando:

```
CONTAR 8
```

o computador escreve:

8, 7, 6, 5, 4, 3, 2, 1 e está pronto para a execução de um próximo comando.

Operações recursivas

A recursão também pode ser usada na definição de operações e predicados. Nestes casos não é preciso preocupar-se com a interrupção do processo recursivo pois, como já foi dito, o comando ENVIE encarrega-se de interromper o procedimento.

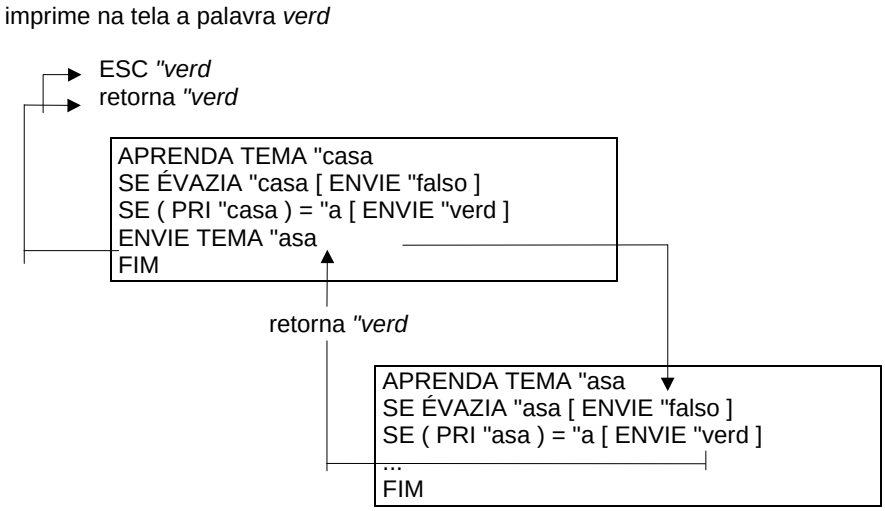
Um predicado que verifica se uma palavra possui a letra **a** poderia ser escrito assim:

```
APRENDA TEMA :p
SE ÉVAZIA :p [ ENVIE "falso ]
SE ( PRI :p ) = "a [ ENVIE "verd ]
ENVIE TEMA SP :p
FIM
```

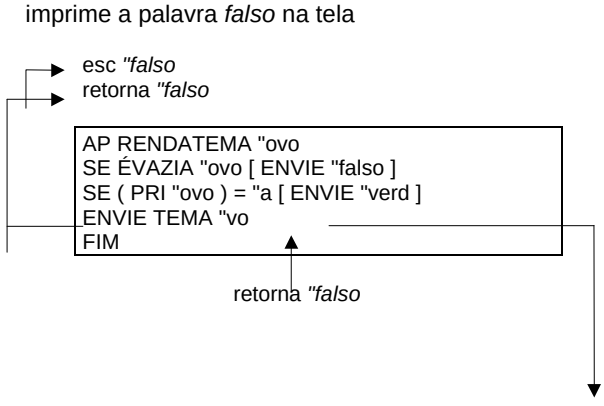
```
ESC TEMA "casa
verd
```

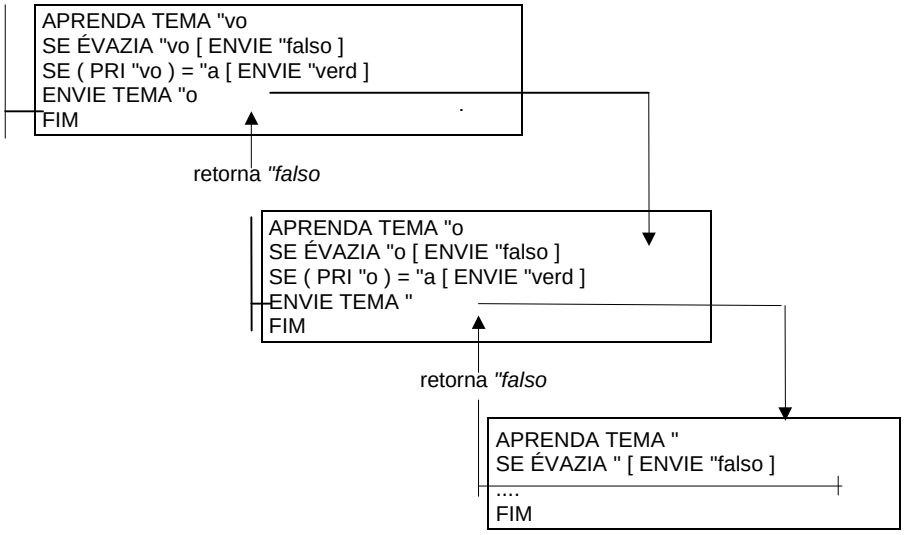
```
ESC TEMA "ovo
falso
```

Vejamos como é o fluxo de execução de ESC TEMA "casa



E o fluxo de execução de ESC TEMA "ovo:





Vejamos um outro exemplo de um predicado bastante útil e que faz parte do elenco de primitivas de algumas versões do Logo. O predicado ÉELEMENTO verifica se um dado objeto é ou não elemento de um outro objeto:

```
APRENDA ÉELEMENTO :objeto1 :objeto2
SE  ÉVAZIA :objeto2 [ ENVIE "falso ]
SE :objeto1 = ( PRI :objeto2 ) [ ENVIE "verd ]
ENVIE ÉELEMENTO :objeto1 SP :objeto2
FIM
```

Se for digitado

```
ESC ÉELEMENTO 3  123
```

obtém-se:

verd

ou, com:

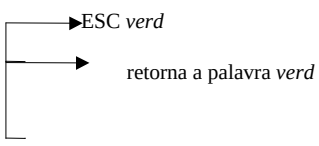
```
ESC ÉELEMENTO 6  [ 66 666 ]
```

obtém-se:

falso

Vejamos como é efetuada a execução de ESC ÉELEMENTO 3 123

imprime a palavra *verd*



```
APRENDA ÉELEMENTO 3  123
SE  ÉVAZIA 123 [ ENVIE "falso ]
SE 3 = ( PRI 123 ) [ ENVIE "verd ]
ENVIE ÉELEMENTO 3  SP 123
FIM
```

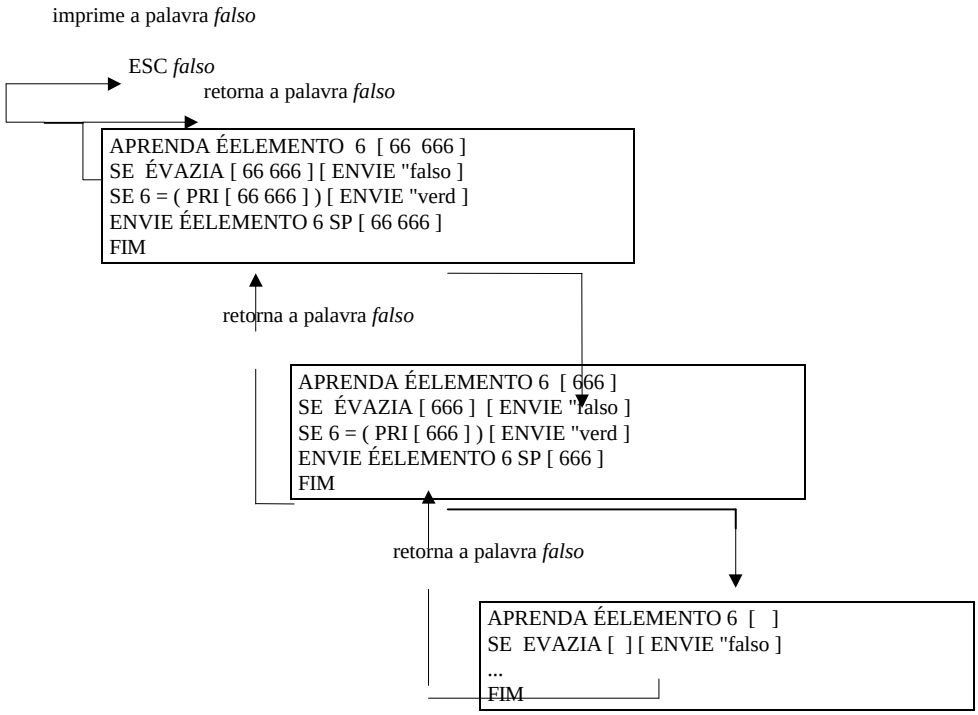
retorna a palavra *verd*

```
APRENDA ÉELEMENTO 3  23
SE  ÉVAZIA 23 [ ENVIE "falso ]
SE 3 = ( PRI 23 ) [ ENVIE "verd ]
ENVIE ÉELEMENTO 3  SP 23
FIM
```

retorna a palavra *verd*

```
APRENDA ÉELEMENTO   3  3
SE  ÉVAZIA 3 [ ENVIE "falso ]
SE 3 = ( PRI 3 ) [ ENVIE "verd ]
.
.FIM
```

e a execução de ESC ÉELEMENTO 6 [66 666] :



Utilizando o retorno da recursão

Os exemplos que vimos até agora, nos quais a chamada recursiva é o último comando, podem ser vistos como casos gerais de repetição. O conceito de recursão se completa quando realizamos comandos ou operações no processo de retorno das sucessivas chamadas recursivas.

Vamos comparar o comando CONTAR já visto:

```
APRENDA CONTAR :num
SE (:num = 0 ) [PARE]
ESC :num
CONTAR :num-1
FIM
```

com o comando seguinte, bastante similar quanto à escrita:

```
APRENDA MIST :num
SE (:num = 0) [ PARE ]
MIST :num-1
ESC :num
FIM
```

Como se pode testar,

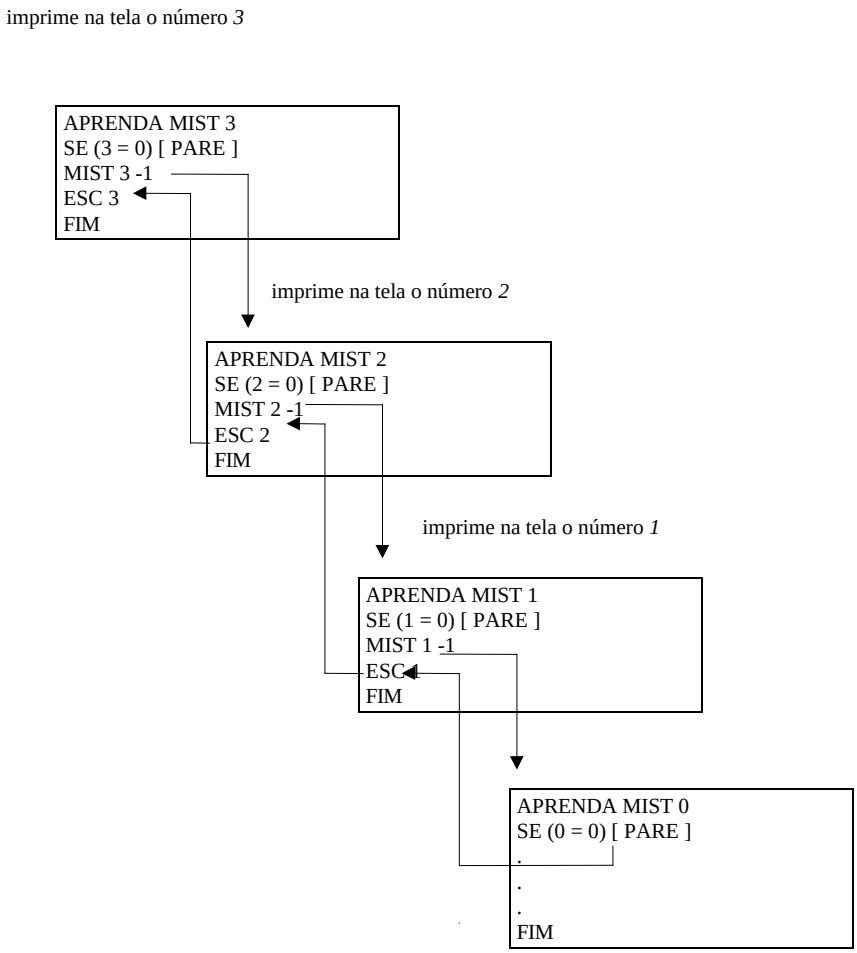
```
CONTAR 3
```

escreve 3, 2, 1. Em contraste,

```
MIST 3
```

escreve 1, 2, 3.

Vamos verificar cuidadosamente como é o processo de avaliação e execução de MIST. Primeiro, chama-se MIST com entrada 3 e MIST inicializa sua biblioteca particular, na qual **num** é associado ao valor 3:



O procedimento MIST 3 tem a instrução MIST 3 -1 e, assim sucessivamente, até chegar a MIST 0. Cada chamada de MIST é um subprocedimento MIST igual ao MIST anterior com um parâmetro de valor diferente. Na quarta cópia de MIST, isto é, em MIST 0, a condição expressa pelo comando SE é satisfeita, sendo interrompida a execução desse procedimento. O controle

da execução do processo todo volta para a cópia anterior, ou seja, ao procedimento que chamou MIST 0 que foi MIST 1. MIST 1 ainda precisa executar a instrução ESC 1 para depois, executar FIM. O controle volta para MIST 2 que também executa ESC 2 antes de executar o comando FIM. E, assim sucessivamente, até a última cópia de MIST chegar ao seu final.

Em suma, este processo respeita a seguinte regra:

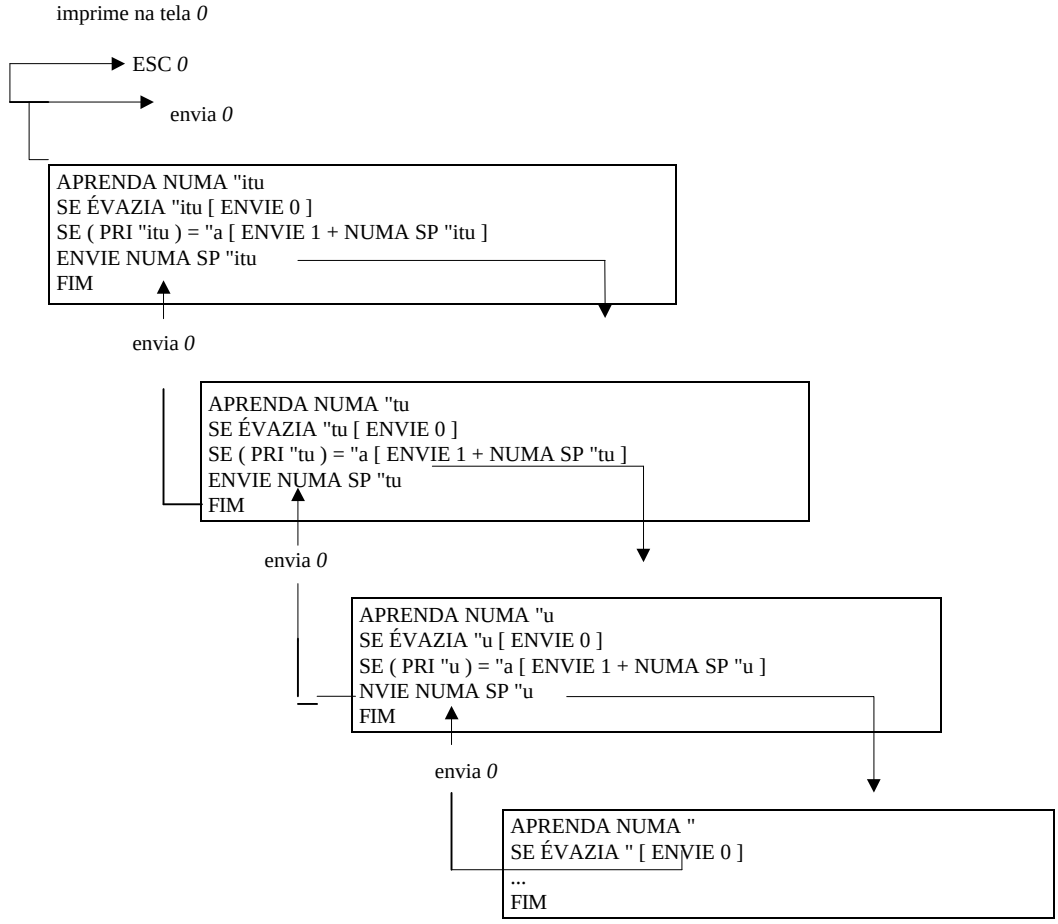
> *Quando um procedimento A chama um procedimento B, ele aguarda o término da execução de B. Quando isto ocorre, o controle é devolvido ao procedimento A que prossegue sua execução a partir do comando seguinte ao da chamada do procedimento B.*

> *Como já foi visto, durante a execução de B existe a biblioteca particular de B e, na volta de B, é a biblioteca particular de A que volta a existir. No caso de chamadas recursivas existe a mesma idéia mas, ao invés de termos procedimentos distintos A e B, temos cópias distintas de um mesmo procedimento A. Cada cópia consulta sua própria biblioteca particular.*

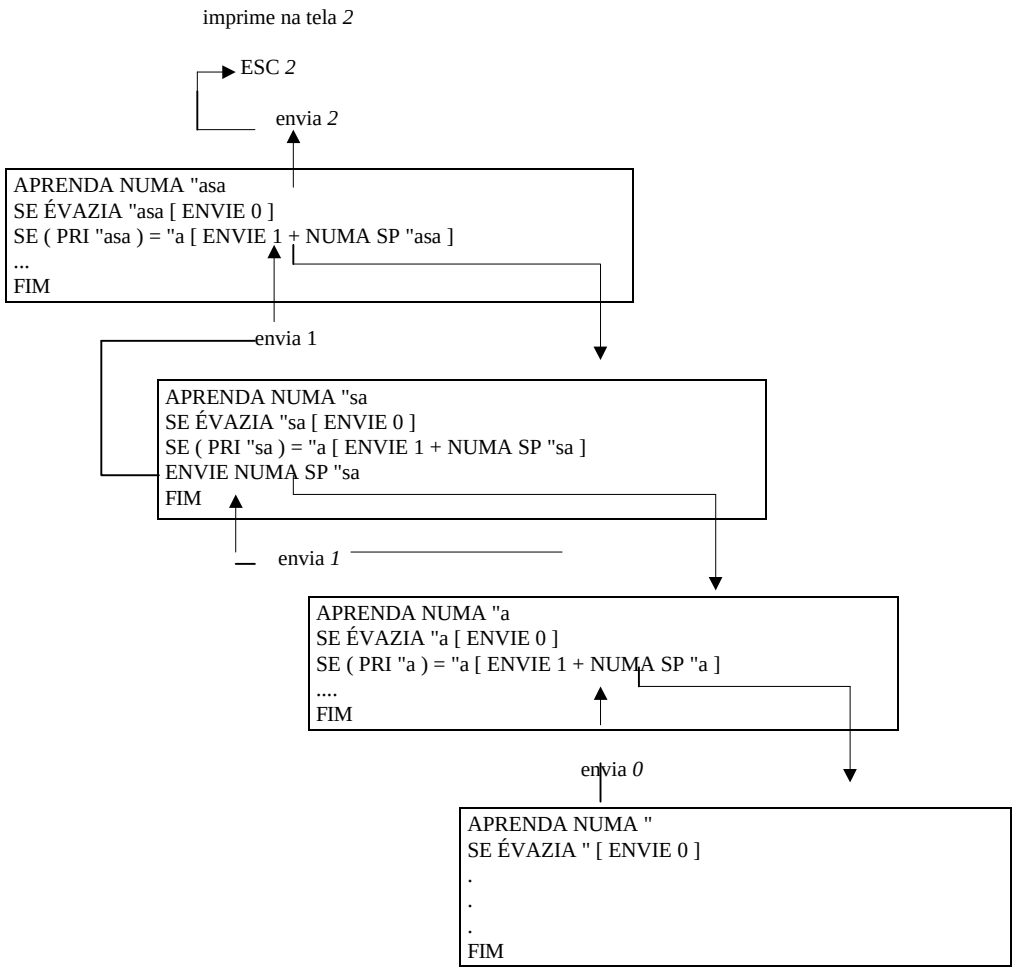
O mesmo processo é verificado nas operações. Para definir uma operação que conte o número de letras **a** de uma palavra pode-se escrever:

```
APRENDA NUMA :p
SE ÉVAZIA :p [ ENVIE 0 ]
SE ( PRI :p ) = "a [ ENVIE 1 + NUMA SP :p ]
ENVIE NUMA SP :p
FIM
```

Vejamos o fluxo de execução de ESC NUMA "itu :



e o fluxo de execução de ESC NUMA "asa :



Conversando com o computador

Em Logo existem operações, como LEIAL e LEIAC, que possibilitam obter informações digitadas por um usuário, via teclado, durante a execução de um procedimento.

A operação LEIAL provoca uma **interrupção** no andamento do programa até que seja digitada qualquer seqüência de caracteres seguida da tecla <enter>. LEIAL retorna o que foi digitado sempre na forma de uma **lista** [7]

```
ESC LEIAL
Maria  < enter >  informação digitada pelo usuário no teclado
```

[7] Mesmo que nada seja digitado, isto é, mesmo que o usuário aperte somente o < enter >, o resultado de LEIAL será uma lista vazia.

```
Maria
```

```
MO LEIAL
Estou com fome...  < enter >  informação digitada pelo usuário no teclado
[ Estou com fome... ]
```

Um exemplo:

```
APRENDA ELE :altura
TAT
PF :altura  PT :altura
PD 90
PF :altura / 2  PT :altura / 2
PE 90
FIM
```

```
APRENDA DESENHAELE
ESC [ Digite um número e aperte a tecla < enter > para desenhar a letra " L " ]
ELE PRI LEIAL
FIM
```

Vejamos como funciona o procedimento DESENHAELE:

O comando ESC escreve na janela gráfica uma instrução para um usuário qualquer;

O procedimento ELE precisa de um parâmetro que é o resultado de duas operações: PRI LEIAL. Portanto, o procedimento ELE não pode ser executado até que as operações retornem um resultado que será usado como entrada de ELE;

A operação PRI aguarda o resultado da operação LEIAL;

A operação LEIAL provoca uma pausa na execução do procedimento DESENHAELE até que o usuário digite alguma coisa e aperte a tecla < enter >.A operação LEIAL retorna aquilo que foi digitado pelo usuário na forma de uma lista;

A operação PRI, então, retira o primeiro elemento da lista retornada pela operação LEIAL;

O resultado da operação PRI é a entrada do procedimento ELE que é, então, chamado. O procedimento ELE é executado com o valor digitado pelo usuário;

Terminada a execução de ELE, o controle é retornado para o procedimento que o chamou, DESENHAELE;

Não há mais nada para ser executado no procedimento DESENHAELE e ele chega ao fim.

O retorno da operação LEIAL pode ser utilizado por qualquer comando, operação ou procedimento definido. No caso de DESENHAELE foi usado como parâmetro do procedimento ELE.

A operação LEIAC é análoga à operação LEIAL. A diferença é que ela é usada quando se deseja que usuário digite um único caracter. O resultado de LEIAC é sempre uma palavra formada por um único caracter. Por exemplo:

```
APRENDA PAPO
ESC [ Oi, meu nome é TATI. Como você se chama ? ]
ESC SN [ Muito prazer, ] LEIAL
ESC [ Eu sei fazer um quadrado...Você quer ver ? Aperte a tecla "s" ou "n" ]
SE  LEIAC  = "s [ REPITA 4 [ PF 80 PD 90 ] PARE]
SE LEIAC   = "n [ESC [ Que pena, até mais!!! ] ]
FIM
```

BIBLIOGRAFIA

Abelson, N.; Abelson, A.. (1992) *Logo for the Macintosh: Na Introduction Trough Object Logo*. Cambridge, MA: Paradigm Software Inc..

Manual do Super Logo (1994) Campinas, SP: NIED / UNICAMP.

PC Logo for Windows: Tutorial, Reference and Glossary. (1994) Cambridge, MA: Harvard Associates, Inc..

Rocha, H.V.; Freire, F.M.P.; Prado, M.E.B.B.. (1999) *Tartaruga, Figuras,Palavras, Listas e Procedimento: Um Primeiro Passeio pelo Logo*. Campinas, SP: MEMO nº 35, NIED / UNICAMP.