Universidade Federal do Rio Grande do Sul
Instituto de Matemática
Cadernos de Matemática e Estatística
Série B: Trabalho de Apoio Didático

Logo - Manual do Usuário

Paulo Werlang de Oliveira
Maria Alice Gravina

Série B, nº 15
Porto Alegre, março de 1993

Publicações do Instituto de Matemática da UFRGS
Cadernos de Matemática e Estatística

Série B: Trabalho de Apoio Didático

1. Elsa Mundstock - Curso Básico Sobre Wordstar 3.45 - MAR/89.
2. Jaime B. Ripoll - Introdução ao Cálculo Diferencial Via Funções de Uma Variável Real - OUT/89.
3. Edmund R. Puczylowski - Dimension of Modular Lattices - JUN/90
4. Marcos Sebastiani - Geometrias Não Euclidianas - JUL/90
5. Sandra R. C. Pizzato - Cálculo Numérico - AGO/91
6. Vera Clotilde G. Carneiro - Elementos de Cálculo para Biologia - AGO/91
7. Elsa Mundstock - Iniciação ao SPSS/PC - SET/91
8. Elisa Haag, Loiva C. de Zeni, Maria Alice Gravina e Vera Clotilde - Notas da 1ª Oficina de Matemática da UFRGS - JAN/92
9. Paulo Werlang de Oliveira, Elisabete Rambo, Suzana Lima dos Santos. Coordenação: Profa. Maria Alice Gravina - A Tartaruga no Espaço Tridimensional - FEV/92
10. Silvio Possoli - Análise Multivariada - JUL/92
11. Dinara Westphalen Fernandez - Números Índices - OUT/92
12. Maria Teresinha Albanese - Coeficiente de Fidedignidade de um Instrumento de Medida - OUT/92
13. Vera Clotilde Carneiro e Sérgio Cláudio Ramos - Gráficos na Escola - DEZ/92
14. João Riboldi - Elementos Básicos de Estatística - JAN/93
15. Paulo W. de Oliveira e M. Alice Gravina - Logo: Manual do Usuário - MAR/93

LOGO

MANUAL DO USUARIO

Universidade Federal do Rio Grande do Sul

Instituto de Matematica

Departamento de Matematica Pura e Aplicada

LOGO

Manual do Usuario

Paulo Werlang de Oliveira

Orientacao
Profa. Maria Alice Gravina

Apoio: FAPERGS

Porto Alegre
Novembro de 1992

APRESENTAÇÃO

LOGO, Manual do usuário é um guia de referência da linguagem LOGO.

Este manual é uma obra indispensável para professores e alunos de escolas de 1º e 2º graus, pois contém vasto material de consulta à respeito desta linguagem. Todos os comandos básicos do LOGO estão presentes, distribuídos ao longo de 12 capítulos, que foram separados de acordo com a afinidade dos comandos.

A cada primitiva segue-se uma explicação detalhada, além de programas ou exemplos mostrando a sintaxe correta.

Ao terminar de ler este manual, o usuário deverá ter uma idéia geral à respeito do LOGO, estando apto à aproveitar da melhor maneira possível todos os recursos que esta linguagem tem a oferecer.

Quero agradecer à minha coordenadora, Profª Maria Alice Gravina, por todas as sugestões apresentadas e que em muito vieram a aprimorar o meu trabalho, além de ter tornado possível a publicação deste livro pelo Instituto de Matemática da UFRGS.

Porto Alegre, 27 de novembro de 1992.

Paulo Werlang de Oliveira

SUMARIO

- INTRODUÇÃO: 1
- Capítulo 1:
  Comandos Gráficos e de Texto 3
- Capítulo 2:
  Edição de Programas 16
- Capítulo 3:
  Variáveis 24
- Capítulo 4:
  Estruturas de Controle 30
- Capítulo 5:
  Lógica 37
- Capítulo 6:
  Palavras e Listas 43
- Capítulo 7:
  Comandos de INPUT/OUTPUT 51
- Capítulo 8:
  Funções Matemáticas 56
- Capítulo 9:
  Arquivos 66
- Capítulo 10:
  Comandos de Formatação 76
- Capítulo 11:
  Usando a Impressora 82
- Capítulo 12:
  Miscelânea 84
- BIBLIOGRAFIA 93
- INDICE REMISSIVO 95

INTRODUÇÃO

O LOGO é uma linguagem de programação de alto nível derivada do LISP, cuja característica básica é operar com listas de dados. Além do fácil aprendizado, tem várias outras características que tornam atraente o seu uso, tais como:

- permite programação modular e redefinição de primitivas;
- tem estruturas de repetição, iteração e recursão;
- funções podem ser definidas pelo usuário;
- tem precisão múltipla entre 5 e 100 dígitos;
- tem "WINDOWS" para texto e gráficos compartilhados;
- tem um duplo sistema de coordenadas (cartesiano e polar);
- possui um editor de textos, que permite ao usuário editar tanto programas quanto variáveis e arquivos;

Tendo em vista que não há quase nenhum material a disposição dos usuários desta linguagem, este manual foi escrito tentando-se abranger a maioria dos comandos do LOGO.

O objetivo deste manual é servir como um guia de referência aos usuários do LOGO. Ele pode ser usado tanto por programadores mais experientes quanto pelos que querem se iniciar no aprendizado desta linguagem de programação. Aos iniciantes recomendo uma leitura atenta deste livro, começando a partir do primeiro capítulo. Também é necessário, para maior compreensão, fazer como exemplo os programas que são mostrados após a descrição detalhada de cada primitiva. Já os programadores mais experientes podem apenas consultar a primitiva desejada no índice remissivo e então ler somente o capítulo de seu interesse.

Este livro foi dividido em 12 capítulos, de tal forma que as primitivas que se relacionam entre sí façam parte de um mesmo capítulo. No início de cada capítulo temos a lista de todas as primitivas que o compõe.

O primeiro capítulo trata das primitivas gráficas e de texto. A idéia foi começar com estas primitivas pois todas elas podem ser executadas em modo direto, o que facilita o aprendizado, pois o leitor pode visualisar graficamente a atuação de cada primitiva independentemente. Uma vez familiarizado com algumas das primitivas do LOGO é possivel começar a aprender a programar.

O segundo capítulo trata exatamente da parte de programação. Nele são ensinados os rudimentos básicos da programação em LOGO, desde a programação interativa (modo direto) até a programação em lote, terminando com os procedimentos de edição de programas.

No terceiro capítulo falamos sobre as variáveis. As variáveis são um recurso poderoso pois permitem construir procedimentos genéricos que podem ser executados com quaisquer valores de entrada, permitindo assim uma maior maleabilidade da linguagem.

O quarto capítulo trata das estruturas de controle. Com elas podemos criar programas "inteligentes", que podem tomar decisões, a partir dos comandos condicionais.

O quinto capítulo trata dos comandos lógicos. Eles são úteis quando se quer expressões condicionais mais sofisticadas.

O sexto capítulo é destinado às palavras e listas. Palavras e listas são poderosos objetos do LOGO que permitem o armazenamento de dados ou de informações para o seu posterior processamento.

O sétimo capítulo trata dos meios de entrada e saída, que permitem ao usuário acessar dados à um programa durante a sua execução.

O oitavo capítulo é destinado às funções matemáticas do LOGO. Neste capítulo é ensinado ao usuário como fazer para definir as funções que ainda não existem no LOGO, a partir das já existentes.

O nono capítulo traz os comandos que operam sobre arquivos em disco, permitindo criar fichários de dados, além do armazenamento permanente de programas e gráficos. Veremos neste capítulo que com apenas um comando é possível gravar em disco um desenho que está pronto na tela gráfica.

O décimo capítulo trata sobre os comandos de formatação dos atributos gráficos, de texto e de dados numéricos. Estes comandos ensinam por exemplo a mudar o tipo da tela de gráficos ou número de caracteres por linha.

O capítulo 11 ensina como usar a impressora e finalmente o capítulo 12 é uma miscelânea com os demais comandos do LOGO.

Concluindo, gostaria de dizer que procurei tornar este livro o mais abrangente possível, porém mantendo-o sempre bastante acessível, de forma que o leitor, ao terminar sua leitura, possa ter uma visão mais ampla de algumas das capacidades desta linguagem de programação tão poderosa que é o LOGO.

CAPITULO 1

COMANDOS GRAFICOS E DE TEXTO

- BACK (BK)
- CLEAN
- CLEARSCREEN (CS)
- CLEARTEXT (CT)
- DOT
- FORWARD (FD)
- FILL
- HEADING
- HOME
- LEFT (LT)
- PENDOWN (PD)
- PENERASE (PE)
- PENREVERSE (PX)
- PENUP (PU)
- POS
- RIGHT (RT)
- SETHEADING (SETH)
- SETPOS
- SETSHAPE
- SETX
- SETY
- SHAPE
- STAMP
- TOWARDS
- XCOR
- YCOR

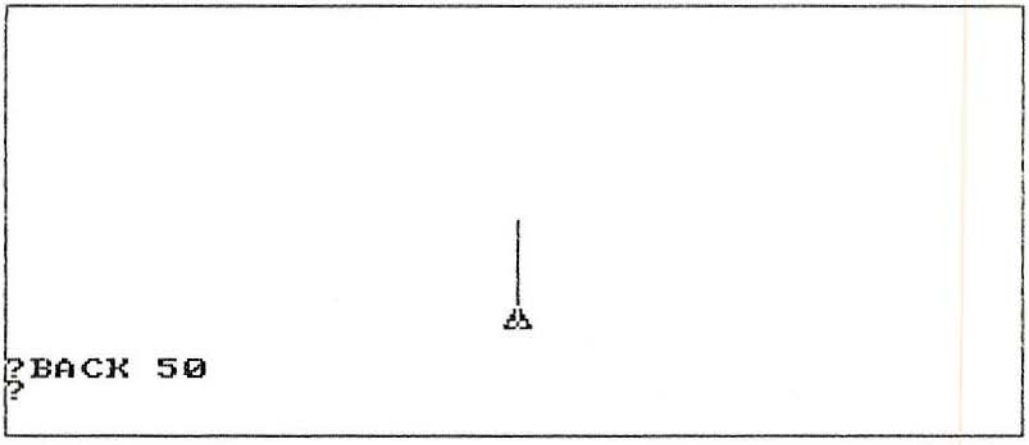
COMANDOS GRAFICOS

Todos os comandos à seguir dizem respeito à mudanças no estado da tartaruga. Essas mudanças podem ocorrer tanto na posição como na orientação da tartaruga. Vejamos tais comandos:

- BACK :n

O comando BACK faz a tartaruga retroceder :n passos de distância, mantendo a sua orientação atual. Note que BACK :n é o mesmo que FORWARD -:n .

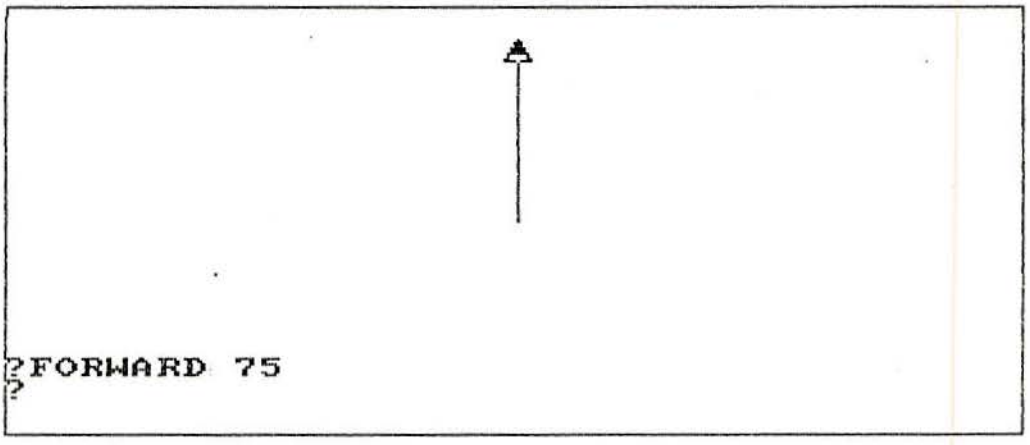
Obs. : O comando BACK pode ser abreviado por BK .
Ex. : BACK 50



- FORWARD :n

O comando FORWARD faz a tartaruga avançar :n passos de distância, mantendo inalterada a sua orientação atual.

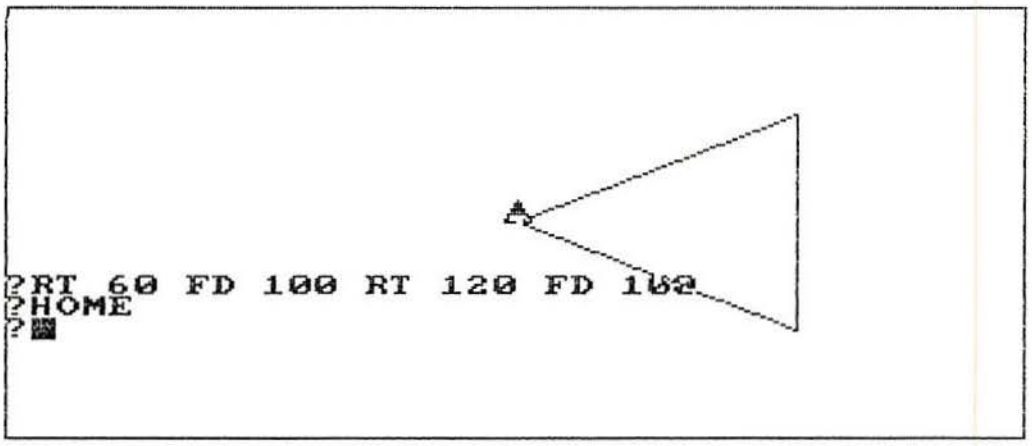
Obs. : O comando FORWARD pode ser abreviado por FD .
Ex. : FORWARD 75



- HOME

O comando HOME leva a tartaruga de volta ao centro da tela e restabelece a orientação em 0 graus (apontando para o "norte"). Os desenhos já existentes não são apagados e a tartaruga vai deixar um rastro se o lápis estiver ativado.

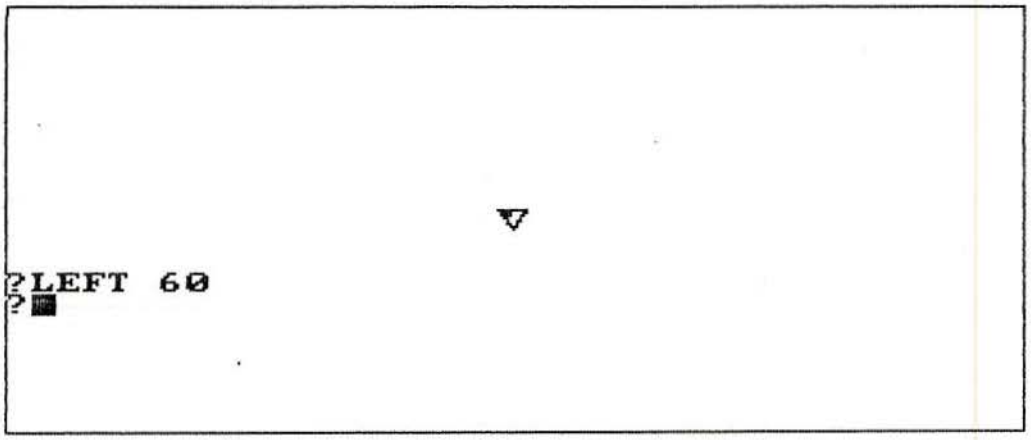
Ex. : HOME



- LEFT :a

O comando LEFT faz a tartaruga girar um ângulo :a para a esquerda, com respeito à sua orientação atual. A sua posição permanece inalterada. O valor do ângulo :a é sempre medido em graus.

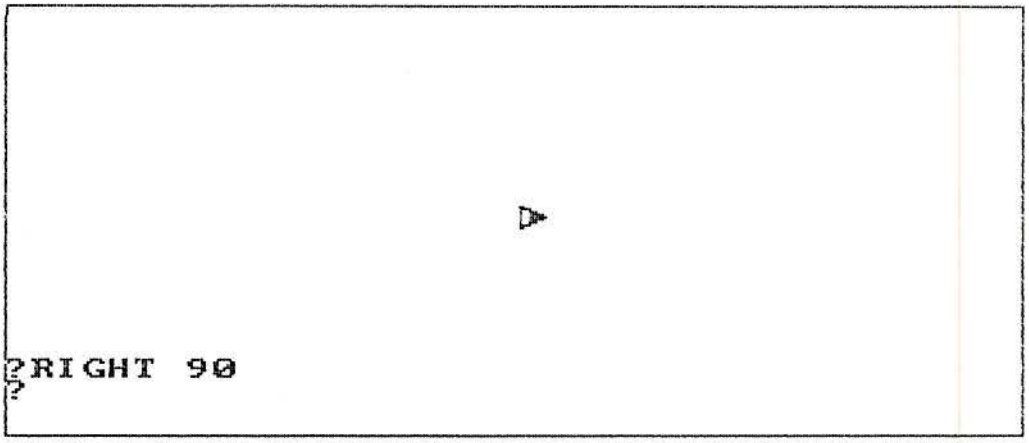
Obs. : O comando LEFT pode ser abreviado por LT .
Ex. : LEFT 60



- RIGHT :a

O Comando RIGHT faz a tartaruga girar para a direita por um ângulo :a dado como entrada. O valor do ângulo :a é sempre medido em graus, com relação à direção atual da tartaruga. Não ocorrem mudanças na posição da tartaruga.

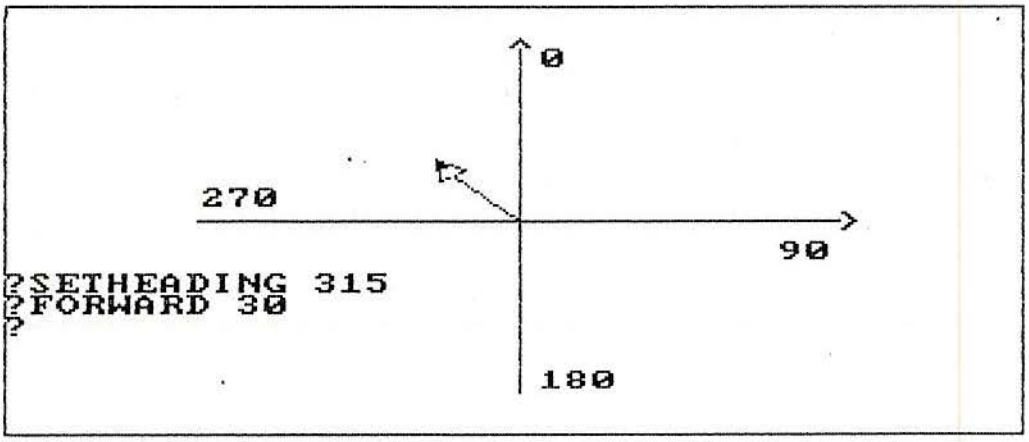
Obs. : O comando RIGHT pode ser abreviado por RT .
Ex. : RIGHT 90



- SETHEADING :a

O comando SETHEADING faz a tartaruga apontar na direção determinada pelo ângulo :a dado como entrada. Este ângulo é medido em graus, no sentido horário e com respeito à orientação dada pela semi-reta OY (orientação padrão "norte"). Não há mudanças na posição da tartaruga.

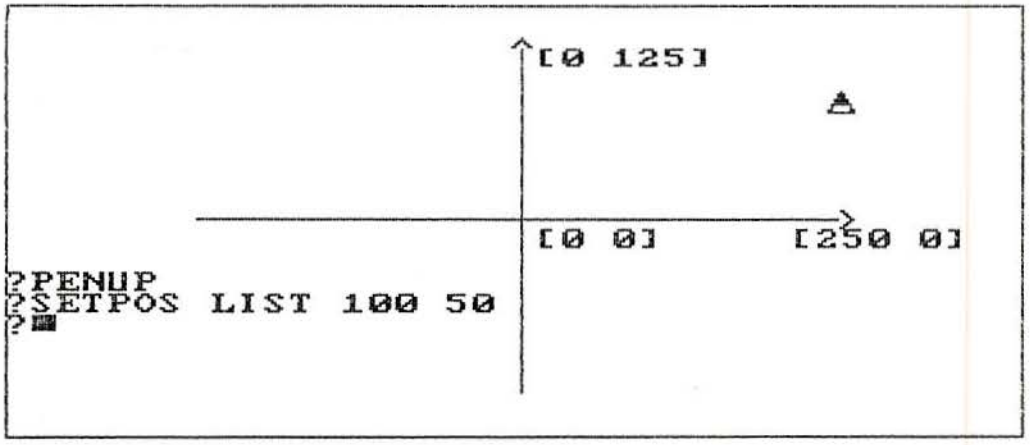
Obs. : O comando SETHEADING pode ser abreviado por SETH .
Ex. : SETHEADING 270



- SETPOS LIST :x :y

O comando SETPOS coloca a tartaruga na posição determinada pela lista de coordenadas :x e :y dada como entrada, não alterando a sua orientação. A tartaruga deixa um rastro se o lápis estiver ativado. A coordenada :x pode assumir valores entre -159 e 160 e a coordenada :y entre -124 e 125. O comando LIST :x :y pode ser substituído pela própria lista contendo os valores das coordenadas, ou seja, SETPOS LIST 10 20 é o mesmo que SETPOS [10 20] .

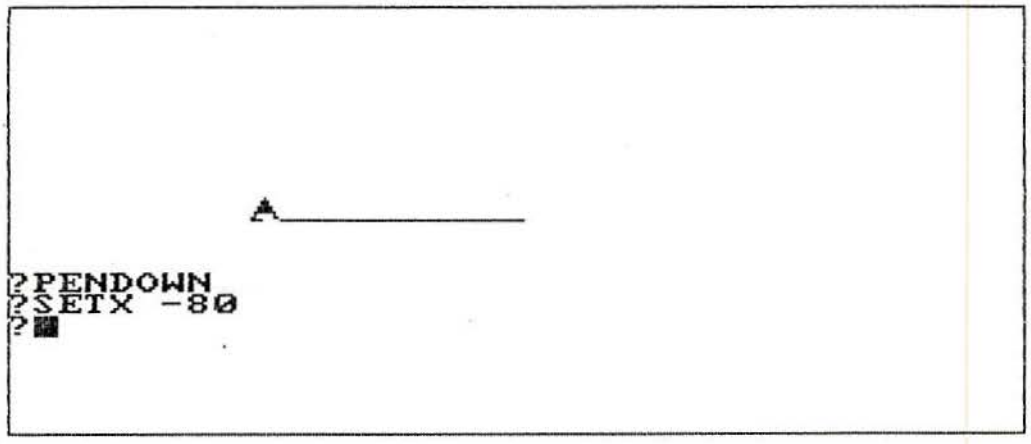
Ex. : SETPOS LIST 100 50



- SETX :x

O comando SETX desloca a tartaruga horizontalmente, colocando-a na coordenada :x dada como entrada. Este comando não afeta a orientação da tartaruga. Será deixado um rastro se o lápis estiver ativado.

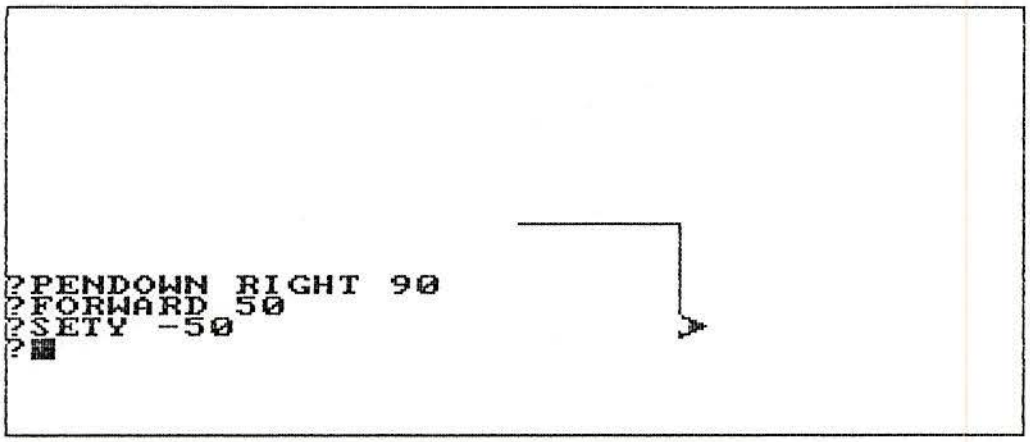
Ex. : SETX -80



- SETY :y

O comando SETY desloca a tartaruga verticalmente, colocando-a na coordenada :y dada como entrada. A orientação da tartaruga não é alterada. Um rastro é deixado se o lápis estiver ativado.

Ex. : SETY 50

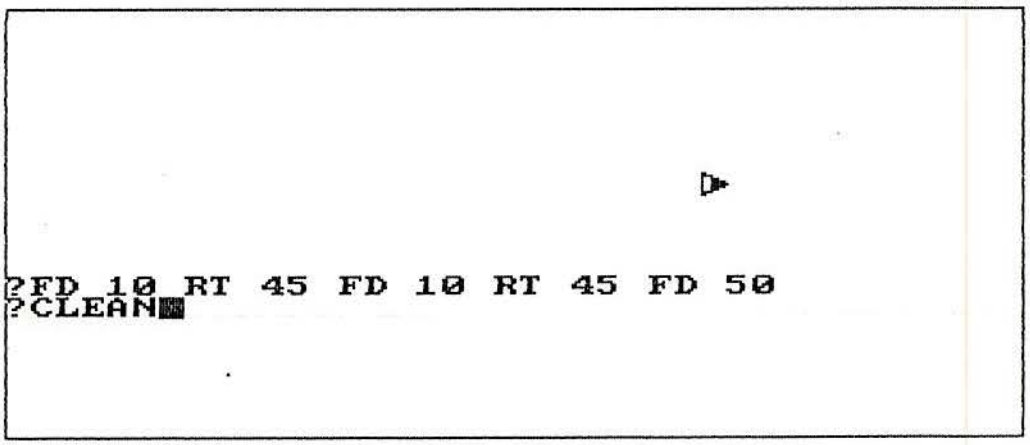


DEMAIS COMANDOS GRAFICOS E DE TEXTO

- CLEAN

O comando CLEAN limpa a tela gráfica, mantendo inalterado o estado da tartaruga.

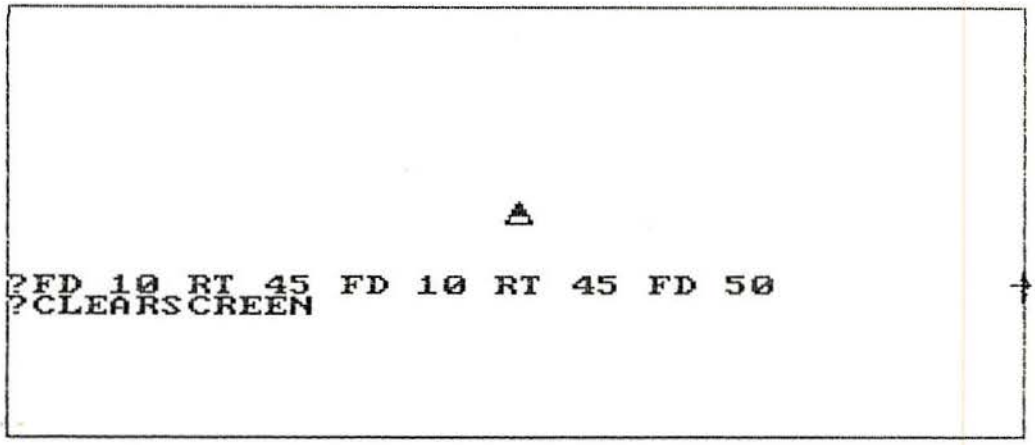
Ex. : CLEAN



- CLEARSCREEN

O comando CLEARSCREEN restaura a tela gráfica, apagando todos os desenhos e levando a tartaruga para o centro da tela e fazendo-a apontar para o "norte". Equivale à executar dois comandos: primeiro HOME e depois CLEAN.

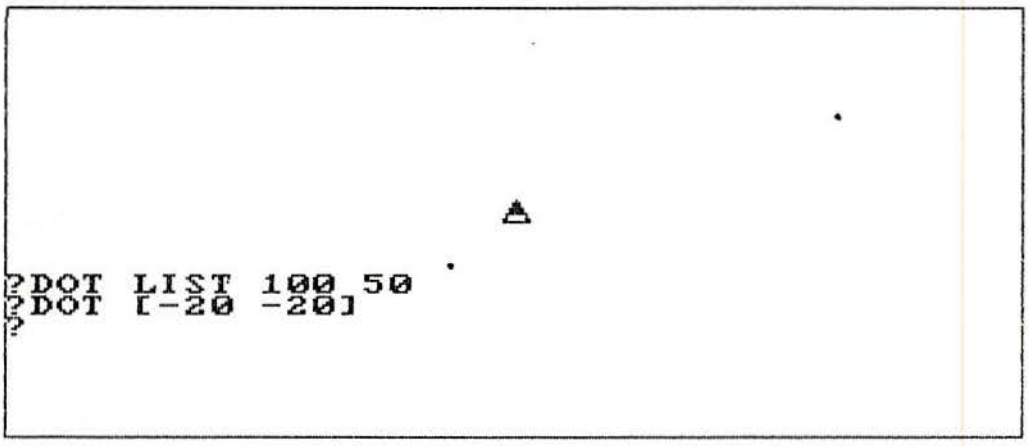
Obs. : O comando CLEARSCREEN pode ser abreviado por CS .
Ex.: CLEARSCREEN



- DOT LIST :x :y

O comando DOT plota um ponto na tela gráfica, na posição determinada pela lista de coordenadas dada como entrada. O estado da tartaruga não é alterado. Se o ponto já estava aceso então ele é apagado. O comando **LIST :x :y** pode ser substituído pela própria lista contendo os valores das coordenadas :x e :y, ou seja, DOT LIST 10 20 é o mesmo que DOT [10 20] .

Ex. : DOT LIST 100 50
DOT [-20 -20]



Programa exemplo:

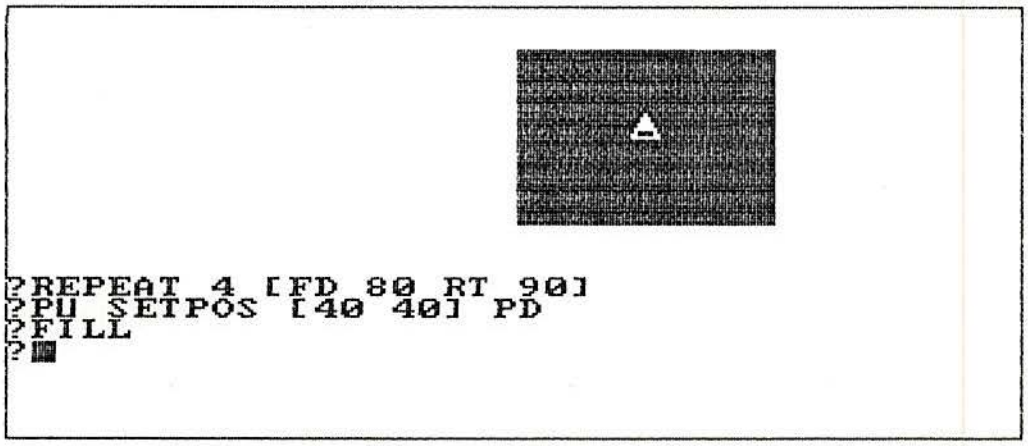
```
TO FLASH
MAKE "X 0  PX
REPEAT 50 [MAKE "X :X+1 DOT LIST :X :X]
FLASH
END
```

- FILL

O comando FILL pinta uma região fechada que rodeie a tartaruga. Para que ele funcione, o lápis precisa estar abaixado e a tartaruga não pode estar sobre o risco que serve como borda da região a ser pintada. A cor da região à ser preenchida deve ser a mesma que foi usada para riscar a fronteira.

Ex.:

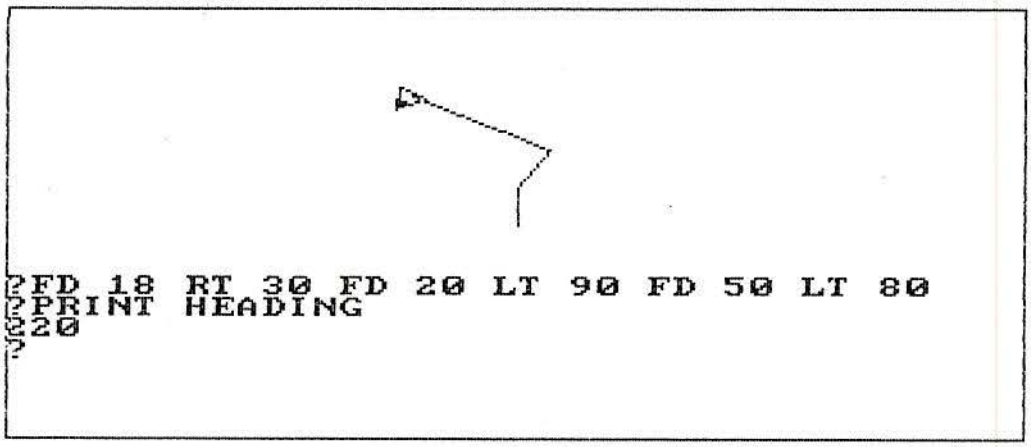
```
FILL
```



- HEADING

O comando HEADING fornece a orientação atual da tartaruga. Veja também o comando SETHEADING.

Ex.: PRINT HEADING



- PENDOWN

O comando PENDOWN baixa o lápis da tartaruga, fazendo com que ela deixe um rastro por onde passar, ao efetuar suas mudanças de posição.

Obs. : O comando PENDOWN pode ser abreviado por PD.

- PENERASE

O comando PENERASE ativa a borracha da tartaruga, fazendo com que ela passe a apagar por onde passar, ao efetuar suas mudanças de posição.

Obs. : O comando PENERASE pode ser abreviado por PE.

- PENREVERSE

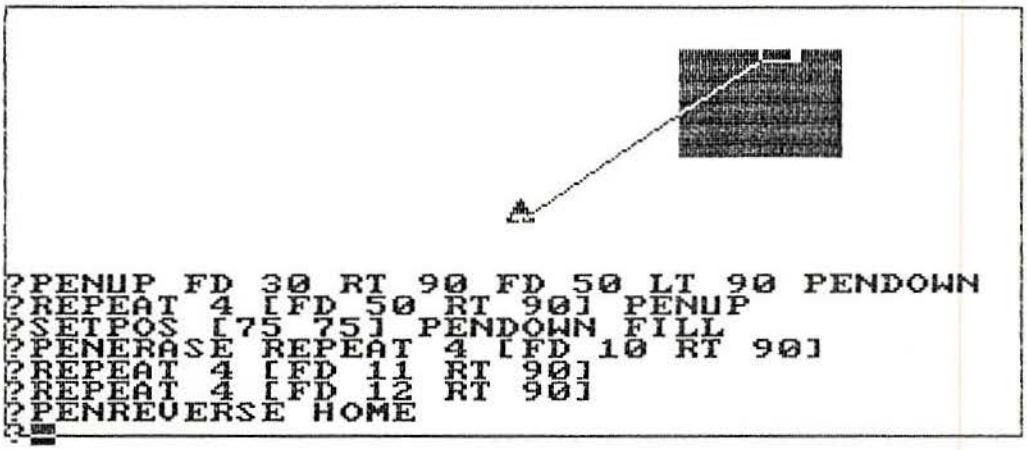
O comando PENREVERSE ativa o inversor da tartaruga, fazendo com que a partir deste momento a tartaruga apague onde está riscado e risque aonde não há nada.

Obs. : O comando PENREVERSE pode ser abreviado por PX.

- PENUP

O comando PENUP levanta o lápis da tartaruga, desativando-o. A partir deste momento a tartaruga não deixa mais um rastro por onde passar, ao efetuar as suas mudanças de posição.

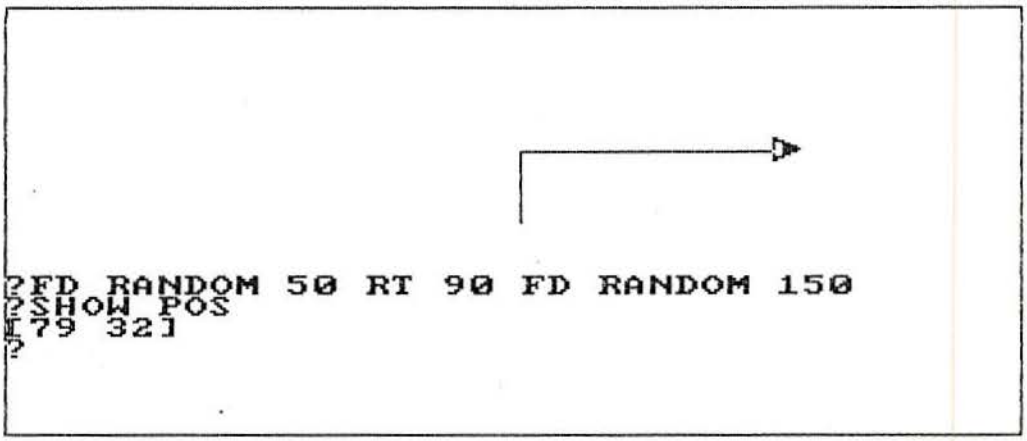
Obs. : O comando PENUP pode ser abreviado por PU.
Ex. : PENUP



- POS

O comando POS fornece a posição atual da tartaruga, na forma de uma lista, onde o primeiro número determina a coordenada x e o segundo a coordenada y.

Ex. : PRINT POS

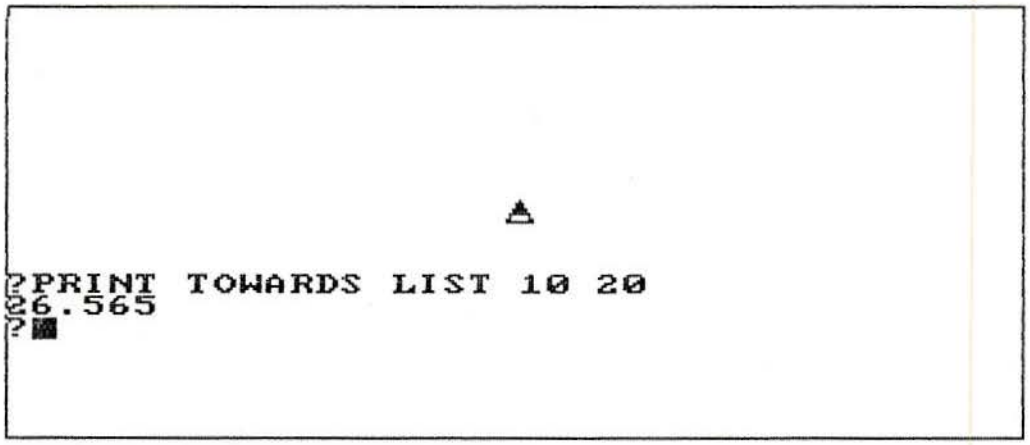


- TOWARDS LIST :x :y

O comando TOWARDS fornece o ângulo com o qual a tartaruga deve ser orientada para que aponte na direção do ponto determinado pela lista de coordenadas dada como entrada. O comando LIST :x :y pode ser substituído pela respectiva lista com os valores das coordenadas x e y , ou seja, TOWARDS LIST 10 20 é o mesmo que TOWARDS [10 20]. Veja também os comandos HEADING e SETHEADING.

Ex. :

```
PRINT TOWARDS LIST 10 20
26.565
```

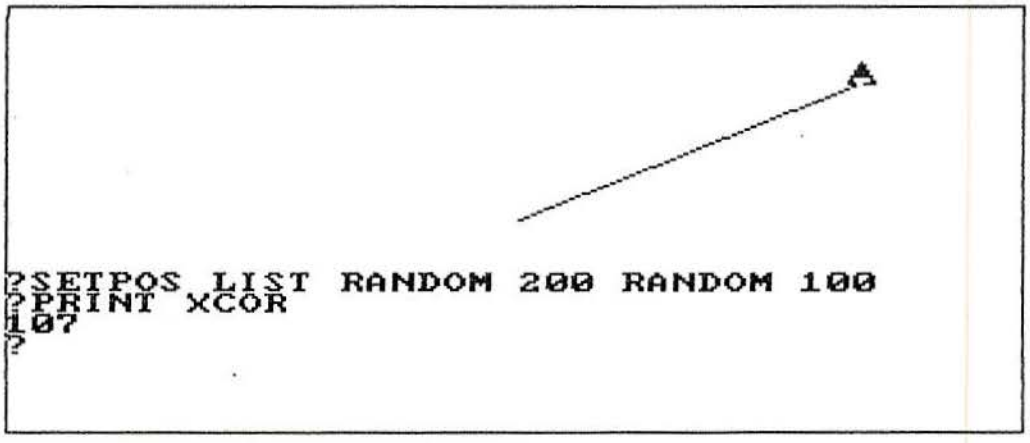


- XCOR

O comando XCOR fornece o valor da coordenada x correspondente à posição atual da tartaruga.

Ex. :

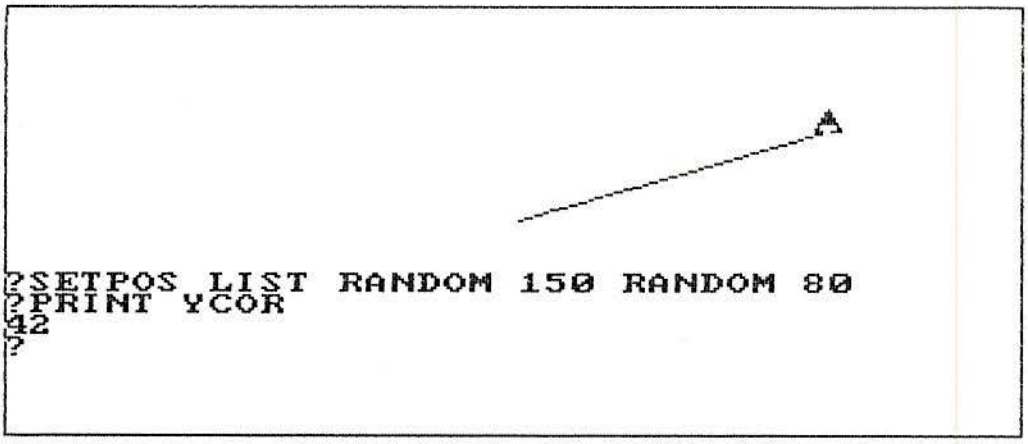
```
PRINT XCOR
```



- YCOR

O comando YCOR fornece o valor da coordenada y correspondente à posição atual da tartaruga.

Ex. : PRINT YCOR



Os seguintes comandos devem ser usados em conjunto e permitem que se escrevam dados ou informações na tela gráfica:

- SETSHAPE :nÚmero

O comando SETSHAPE faz a tartaruga assumir o formato de um dos 256 caracteres especiais da tabela padrão ASCII. O número dado como entrada precisa ser um inteiro entre 0 e 255. Para que a tartaruga volte ao seu estado normal use a palavra reservada "TURTLE.

Ex. :
```
SETSHAPE 97
SETSHAPE "TURTLE
```

Obs. : Para maiores detalhes consulte uma tabela ASCII ou rode o seguinte programa:

```
TO FORMATOS
CT CS MAKE "N -1
REPEAT 256 [MAKE "N :N+1 SETSHAPE :N
CT PR SHAPE WAIT 5]
SETSHAPE "TURTLE
END
```

- SHAPE

O comando SHAPE fornece o codigo ASCII do formato atual da tartaruga. Veja também os comandos SETSHAPE e STAMP.

Ex. : PRINT SHAPE

- STAMP

O comando STAMP deixa uma imagem fixa da tartaruga, de acordo com a sua forma atual, na posição em que ela se encontra. Veja também os comandos SETSHAPE e SHAPE.

O programa exemplo à seguir serve para escrever uma palavra na tela gráfica:

```
TO ESCREVE :PALAVRA
CS ST PU MAKE "I 0
REPEAT COUNT :PALAVRA [MAKE "I :I+1
SETSHAPE ASCII ITEM :I :PALAVRA
STAMP SETX XCOR+8]
SETSHAPE "TURTLE HT
END
```

- CLEARTEXT

O comando CLEARTEXT limpa a tela de textos, colocando o cursor no canto superior esquerdo da respectiva janela de textos. Este comando não afeta a janela de gráficos.

Obs. : O comando CLEARTEXT pode ser abreviado por CT.

Ex. : CLEARTEXT

:

CAPITULO 2

EDIÇÃO DE PROGRAMAS

- COPYDEF
- DEFINE
- DEFINEDP
- EDIT (ED)
- END
- ERALL
- ERASE (ER)
- ERPS
- PO
- POALL
- POPS
- POTS
- REDEFP
- TEXT
- TO

EDIÇÃO DE PROGRAMAS

Como construir um quadrado ?

Você tem duas maneiras de construir um quadrado. A primeira delas é usando o modo direto do LOGO, que consiste em digitar cada comando em uma linha e teclar "RETURN". Vejamos como fica :

```
CS              "RETURN"
FD 50           "RETURN"
RT 90           "RETURN"
FD 50           "RETURN"
RT 90           "RETURN"
FD 50           "RETURN"
RT 90           "RETURN"
FD 50           "RETURN"
RT 90           "RETURN"
```

Note que toda vez que quisermos refazer o quadrado usando o modo direto temos que repetir a seqüência de comandos acima.

A outra maneira de se fazer o quadrado é através do modo de programação. No modo de programação é definido um procedimento que contém a seqüência de comandos que executa o quadrado. Este procedimento irá funcionar tal qual uma nova primitiva na área de trabalho, bastando digitar o nome do procedimento para que tenhamos toda a seqüência de comandos automaticamente executada.

Vejamos como programar em LOGO :

O primeiro passo é escolher um nome para o procedimento que vamos definir. No nosso caso o nome pode ser QUADRADO.

Digite então **TO QUADRADO** e tecle "RETURN". Note que o símbolo especial **>** aparece no canto esquerdo da linha abaixo. Este "prompt" é diferente do padrão normal para nos indicar que estamos dentro do modo de programação.

Digite então a seguinte seqüência de instruções :

```
>REPEAT 4 [FD 50 RT 90]     "RETURN"
```

Note que cada vez que teclamos "RETURN" os comandos de cada linha digitada não são executados automaticamente. É porque estamos no modo de programação e isto significa que as instruções que são digitadas estão sendo guardadas na memória para uso posterior.

Finalmente digite o comando **END** e tecle "RETURN".

O comando **END** indica ao LOGO que terminamos a definição do procedimento. Note que o "prompt" normal ? voltou e aparece a mensagem **QUADRADO DEFINED**. Ou seja, o programa QUADRADO já está definido e pronto para ser usado.

Você tem na tela :

```
?TO QUADRADO
>REPEAT 4 [FD 50 RT 90]
>END
QUADRADO DEFINED
?
```

Nota : O comando END deve ser sempre digitado no final do programa, em linha separada e seguido de "RETURN", pois ele indica o fim do programa.

Vamos agora executar o programa QUADRADO :

Digite CS QUADRADO e tecle "RETURN" para ver o que acontece. Não foi fácil ?

E se agora quisermos fazer um quadrado de tamanho 100. Será que precisamos digitar tudo de novo ?
A resposta é não ! Basta editarmos o programa QUADRADO já existente e trocarmos os valores em cada linha do procedimento.

Digite **EDIT [QUADRADO]** e tecle "RETURN". Vai aparecer uma tela diferente mostrando :

```
TO QUADRADO
REPEAT 4 [FD 50 RT 90]
END
```

O comando **EDIT** coloca o LOGO no modo de edição de programas. Os programas que tem o seu nome colocado dentro da lista de entrada do comando EDIT são transferidos da memória do micro para a janela de edição, onde podem ser facilmente modificados. Agora todas as teclas de controle do teclado podem ser usadas (BACK SPACE, SETAS, DEL, etc...), pois o editor do LOGO funciona exatamente como um editor de textos comum.
Altere os valores de FD 50 para FD 100. Quando terminar acione a tecla de escape **ESC**. Desta forma o LOGO volta ao seu modo direto e vai aparecer uma tela limpa, com a seguinte mensagem :

```
QUADRADO DEFINED
?
```

ESC faz sair do editor do LOGO, voltando ao modo direto.

Execute o procedimento QUADRADO de novo. Veja que ele ficou maior.

Digite **EDIT** (ou simplesmente **ED**) e tecle "RETURN".
O que aconteceu ? Voltamos à janela de edição.

O comando ED digitado sem argumentos de entrada faz o LOGO voltar à janela de edição, trazendo de volta os últimos procedimentos que foram editados. Assim podemos facilmente fazer quadrados de outros tamanhos, apenas trocando os respectivos valores na janela de edição.

Obs. : O comando EDIT pode ser também usado para criar procedimentos novos. Para tanto basta digitar **EDIT "NOMEPROGRAMA** para que o programa cujo nome é dado como entrada possa ser editado.

E se você quizer mudar o nome do procedimento QUADRADO para Q ? Note que Q é bem mais fácil de digitar do que QUADRADO. Use o seguinte comando, que permite copiar procedimentos :

Digite **COPYDEF "Q "QUADRADO** e tecle "RETURN".
Digite **CS Q** e tecle "RETURN". O que aconteceu ?

Agora Q e QUADRADO fazem a mesma coisa.

Uma vez definido um procedimento ele já pode ser usado em outros programas. Veja o seguinte exemplo :

```
?TO POLI
>REPEAT 8 [Q RT 45]
>END
POLI DEFINED
?
```

Digite POLI e tecle "RETURN" para executá-lo.

Após estas considerações gerais vejamos quais são os principais comandos para a edição de procedimentos e sua sintaxe correta.

-COPYDEF "NOVONOME "NOMEANTIGO

COPYDEF permite que se copie a definição de um procedimento. Isto é útil quando queremos ter dois procedimentos, um com o nome por extenso e o outro como uma abreviatura. Note que o programa denominado em "NOMEANTIGO precisa já estar definido, para que possamos copiá-lo.

Ex. :

```
TO QUADRADO :LADO
REPEAT 4 [FORWARD :LADO RT 90]
END

COPYDEF "Q "QUADRADO

POTS
Q :LADO
QUADRADO :LADO
```

Obs. : COPYDEF também pode ser usado para redefinir as primitivas do LOGO. Veja como exemplo o programa abaixo, que serve para redefinir a primitiva PRINT :

```
TO REDEFINE
MAKE "REDEFP "TRUE
COPYDEF "escreva "PRINT
MAKE "REDEFP "FALSE
END
```

Execute o programa REDEFINE.

Agora **escreva** e **PRINT** fazem a mesma coisa. Experimente escreva PI e PRINT PI .

Nota : REDEFP é uma variável do sistema e serve para habilitar a redefinição das primitivas. Para que possamos trocar o nome das primitivas é preciso que ela seja "TRUE.

- DEFINE "NOMEPROG [[LISTA DE VARIAVEIS][LISTA DE COMANDOS]]

DEFINE permite definir um procedimento sem termos que usar os comandos TO e END e nem entrar na janela de edição. Isto é útil pois podemos criar um procedimento durante a execução de outro programa. Se o procedimento que vamos definir não tem variáveis de entrada então a lista de variáveis precisa ser vazia. Não é necessário escrever o comando END na lista de instruções.

Ex. :

```
DEFINE "PENTAGONO [[LADO][REPEAT 5[FD
 :LADO RT 72]]]
```

Verifique que digitando PO "PENTAGONO e teclando "RETURN" vai aparecer :

```
TO PENTAGONO :LADO
REPEAT 5 [FD :LADO RT 72]
END
```

- DEFINEDP "NOMEPROGRAMA

DEFINEDP permite verificar se o procedimento cujo nome é dado como entrada está definido ou não na memória do micro.

Ex. :

```
PR DEFINEDP "PENTAGONO
TRUE
ERALL
PR DEFINEDP "PENTAGONO
FALSE
```

- EDIT "NOMEPROGRAMA
- EDIT [lista com o nome dos programas à serem editados]

EDIT ativa o editor do LOGO, permitindo criar novos programas ou alterar os já existentes. Para editar só um programa digite EDIT seguido do nome do programa desejado, precedido por aspas " . Para editar mais de um programa digite EDIT seguido da lista que contenha o nome de todos os procedimentos que se quer editar ao mesmo tempo. Terminada a edição, saia do editor acionando a tecla ESC de escape. EDIT pode ser abreviado por ED .

Ex. :
```
EDIT "QUADRADO

EDIT [QUADRADO Q PENTAGONO]

ED
```

Obs. : ED sem argumentos edita os últimos programas que estão na memória da janela de edição.

-END

O comando END é usado para se terminar um programa que começou a ser editado com o comando TO. Ele deve ser colocado ao final do procedimento, em uma linha separada, seguido apenas por "RETURN". O comando END além de indicar o fim físico de um programa também suspende a sua execução, quando em processamento.

- ERALL

O comando ERALL é usado para limpar a memória do micro, antes de começarmos a fazer novos programas. Ele apaga tudo, inclusive programas e variáveis globais.

Ex. :
```
ERALL
```

- ERASE "NOMEPROGRAMA

O comando ERASE serve para apagar procedimentos da memória do micro, não afetando as variáveis globais. Para apagar somente um programa digite ERASE seguido do nome do procedimento que se quer apagar, precedido por aspas " . Para apagar mais de um programa digite ERASE seguido da lista que contém o nome de todos os procedimentos que se quer apagar. O comando ERASE pode ser abreviado por ER.

Ex. :
```
ERASE "QUADRADO
ER [Q PENTAGONO]
```

- ERPS

O comando ERPS apaga **todos** os procedimentos da área de trabalho do micro, deixando apenas as variáveis globais presentes na memória.

Ex. : ERPS

- PO "NOMEPROGRAMA
- PO [lista de programas]

O comando PO serve para mostrar a definição de procedimentos que estão na memória do micro. Para ver somente um procedimento digite PO seguido do respectivo nome precedido por aspas " . Para ver mais de um programa digite PO seguido da lista com os respectivos nomes.

Ex. : PO "QUADRADO
PO [Q PENTAGONO]

- POALL

O comando POALL serve para mostrar todo o conteúdo da área de trabalho, tanto programas quanto variáveis globais. Primeiro aparecem os programas que estão presentes na memória do micro, do mais recente ao mais antigo, depois aparecem as variáveis globais, com seus nomes e respectivos conteúdos.

Ex. : POALL

- POPS

O comando POPS serve para mostrar somente os procedimentos da área de trabalho.

Ex. : POPS

- POTS

O comando POTS serve para mostrar os títulos (nome dos procedimentos com suas respectivas entradas) de todos os programas que estão atualmente na área de trabalho do LOGO.

Ex. : POTS

- REDEFP

REDEFP é uma variável do sistema, que aceita somente os valores "TRUE ou "FALSE e permite ou não redefinir as primitivas da linguagem LOGO. Se REDEFP é "TRUE então podemos redefinir as primitivas com o comando COPYDEF. Se ela é "FALSE não é possível redefinir as primitivas, somente copiar procedimentos que foram criados pelo usuário. Seu valor default é "FALSE.

Ex. :
```
MAKE "REDEFP "TRUE
MAKE "REDEFP "FALSE
```

Obs. : Para maiores detalhes veja também o comando COPYDEF.

- TEXT "NOMEPROGRAMA

O comando TEXT fornece como saída uma lista que corresponde aos comandos de definição do procedimento cujo nome é dado como entrada.

Ex. :
```
SHOW TEXT "PENTAGONO
DEFINE "PENT TEXT "PENTAGONO
PO [PENT PENTAGONO]
```

Obs. : O comando SHOW será tratado no capítulo 7.

- TO NOMEPROGRAMA :ENTRADAS

O comando TO permite criar procedimentos sem entrarmos na janela de edição. Para tanto digite o comando TO seguido do título do procedimento que você deseja criar (o título de um procedimento é o nome do procedimento seguido de suas respectivas variáveis de entrada) e tecle "RETURN". Irá aparecer um "prompt" especial (›) que indica que se está definindo um programa em modo direto. Digite a seqüência de instruções que definem o procedimento, sempre teclando "RETURN" quando quiser trocar de linha. Finalize digitando o comando END (sozinho e na última linha) e tecle "RETURN". Terminada a definição do procedimento vai aparecer a respectiva mensagem na tela e o "prompt" normal retornará. O programa já está definido e pronto para ser usado.

Ex. :
```
?TO QUADRADO :LADO
›REPEAT 4 [FD :LADO RT 90]
›END
QUADRADO DEFINED
?
```

CAPITULO 3

VARIAVEIS

- EDNS
- ERN
- ERNS
- LOCAL
- MAKE
- NAME
- NAMEP
- PONS
- THING

VARIAVEIS

O que é uma variável ?

Podemos responder à esta pergunta dizendo que uma variável é um objeto do LOGO que serve para guardar informação e que pode ter o seu conteúdo alterado durante a execução de um programa. O conteúdo (ou informação) a ser armazenado pode ser tanto um número, uma "string" alfanumérica (palavra) ou um "array" de dados (lista). Ao contrário das outras linguagens, no LOGO não é preciso identificar no nome da variável o tipo de dado que ela vai guardar, ou seja, todas as variáveis tem o mesmo tratamento, independente de serem números, palavras ou listas.

A única classificação que o LOGO faz sobre uma variável é quanto ao seu caráter local ou global, ou seja, se ela pertence única e exclusivamente à um procedimento ou a todos os que estão definidos na área de trabalho.

Toda variável **local** fica armazenada temporariamente na memória do micro, sendo usada exclusivamente dentro de um procedimento. Tão logo a execução do procedimento é concluída, automaticamente esta variável é apagada da memória. Todas as variáveis declaradas como entrada de procedimentos (nomes que vão ao lado do título do procedimento, precedidos por :) são de caráter local.

Ex. :

```
TO QUADRADO :LADO
REPEAT 4 [FD :LADO RT 90]
END
```

Aqui, a variável :LADO é uma variável local.

Variáveis locais são muito úteis pois permitem criar procedimentos genéricos que podem ser executados a cada vez com diferentes valores de entrada.

Todas as demais variáveis criadas com os comandos MAKE ou NAME são **globais**. Elas podem ser usadas em qualquer procedimento, a qualquer tempo, pois estão armazenadas de forma permanente na memória do micro. Ao gravarmos programas no disquete elas são gravadas junto com os procedimentos.

As variáveis globais podem ser listadas ou editadas na janela de edição (para podermos alterar o seu conteúdo sem ter que executar nenhum programa) e podem ser apagadas, uma de cada vez ou todas ao mesmo tempo, por comandos específicos. Uma vez criada uma variável global, o seu conteúdo permanece inalterado até que o modifiquemos com um novo comando MAKE ou NAME.

A vantagem de se usar variáveis globais é que elas são um meio fácil de se armazenar dados ou informações.

Vejamos um exemplo:

```
TO TITULO :VARIAVELLOCAL
SHOW :VARIAVELLOCAL
MAKE "VARIAVELGLOBAL :VARIAVELLOCAL
SHOW :VARIAVELGLOBAL
END
```

Este programa apaga todas as variáveis globais, imprime a variável local dada como entrada, cria uma variável global atribuindo-lhe o conteúdo da variável local dada como entrada e finalmente mostra a variável global que foi criada.

Execute-o, digitando TITULO 100 e teclando "RETURN".

Digite POALL + "RETURN" e veja que a variável :VARIAVELGLOBAL aparece listada na área das variáveis e seu valor é 100 ; enquanto que a variável :VARIAVELLOCAL não aparece.

Repita o exercício rodando agora com uma palavra e depois com uma lista.

Ex. :

```
TITULO "PALAVRA

TITULO [ISTO E UMA LISTA]
```

Note que o tratamento é o mesmo para todos os tipos de dados. Uma dica : Não há restrição quanto ao tamanho do nome da variável (quanto menor, mais fácil de digitá-lo) mas procure não usar caracteres especiais, para evitar problemas.

Surge um impasse quando queremos que determinada variável seja local mas precisamos criá-la com um comando MAKE, que a torna automaticamente global. Um exemplo desta situação é quando, ao invés de entrarmos com os dados no título do procedimento, queremos que eles sejam lidos do teclado, como no programa abaixo:

```
TO EXEMPLO
PR [DIGITE UMA PALAVRA E TECLE RETURN]
PR [PARA PARAR DIGITE PARE!]
MAKE "PALAVRA READWORD
PRINT :PALAVRA
IF :PALAVRA="PARE! [STOP]
EXEMPLO
END
```

A solução é simples, basta acrescentar mais uma linha ao programa, ficando :

```
TO EXEMPLO
LOCAL "PALAVRA
PR [DIGITE UMA PALAVRA E TECLE RETURN]
PR [PARA PARAR DIGITE PARE!]
MAKE "PALAVRA READWORD
PRINT :PALAVRA
IF :PALAVRA="PARE! [STOP]
EXEMPLO
END
```

Execute-o e experimente com POALL.

O comando LOCAL "PALAVRA muda a característica da variável. Mesmo sendo criada com o comando MAKE, ela se torna uma variável local como as demais, pois o comando LOCAL prevalece sobre o MAKE.

Finalmente faremos uma breve consideração sobre variáveis locais e recursão. Analise cuidadosamente o seguinte exemplo:

```
TO ITERA :VALOR
PR :VALOR
IF :VALOR>10 [STOP]
ITERA (:VALOR + 1)
PR :VALOR
END
```

Note que a cada vez que o valor é impresso, o procedimento chama de novo a sí mesmo, com um valor diferente (que é o valor antigo mais um). Como o procedimento ITERA não foi concluído e a variável é local então :VALOR+1 é diferente de :VALOR e portanto precisa ser guardado numa posição de memória diferente. Por este motivo é que os procedimentos recursivos geralmente estouram rapidamente o processamento por falta de memória disponível.

Vejamos agora quais são os comandos do LOGO que servem para manipular variáveis e sua sintaxe correta.

- **EDNS**

O comando EDNS coloca todas as variáveis globais na janela de edição, permitindo-se alterar o conteúdo de cada uma delas como se estivessemos editando programas. Podem ser usadas as teclas de controle do teclado, como as setas, BACK SPACE, DEL, PG UP, PG DN, etc... Após atualizarmos o conteúdo das variáveis desejadas, saímos da janela de edição da mesma forma, ou seja, acionando-se a tecla de escape **ESC**. Automaticamente as variáveis serão rearmazenadas na área de memória, com os novos conteúdos.

- **ERN "NOMEVARIAVEL**

ERN apaga da memória do micro a variável global cujo nome é dado como entrada.

- ERNS

O comando ERNS apaga todas as variáveis globais da memória do micro.

- LOCAL "NOMEVARIAVEL

LOCAL permite mudar a característica de uma variável, fazendo com que a variável cujo nome é dado como entrada seja local à um determinado procedimento, mesmo que ela seja criada com um comando MAKE ou NAME. Note que esta primitiva só pode ser usada dentro de um procedimento.

Ex. :

```
TO CONTANDOATE :VALORFINAL
LOCAL "C
TYPE [NUMEROS PARES ATE :]
PR :VALORFINAL
MAKE "C 0
LABEL "CONTANDO
MAKE "C :C+2
IF :C>:VALORFINAL [STOP]
PR :C
GO "CONTANDO
END
```

- MAKE "NOMEVARIAVEL :OBJETO

MAKE é um comando de atribuição. Ele permite criar variáveis globais e/ou atualizar o seu conteúdo.

Ex. :

```
TO CONTARDE :VALORINICIAL :VALORFINAL
MAKE "CONTADOR :VALORINICIAL
LABEL "REPETE
IF :CONTADOR > :VALORFINAL [STOP]
PR :CONTADOR
MAKE "CONTADOR :CONTADOR+1
GO "REPETE
END
```

- NAME :OBJETO "NOMEVARIAVEL

NAME é um comando de atribuição. Seu funcionamento é idêntico ao comando MAKE, só que as entradas ocorrem em ordem inversa.

Ex. :

```
TO CONTAR :VALORINICIAL :VALORFINAL
NAME :VALORINICIAL "CONTADOR
LABEL "REPETIR
IF :CONTADOR > :VALORFINAL [STOP]
PR :CONTADOR
NAME ( :CONTADOR+1 ) "CONTADOR
GO "REPETIR
END
```

- NAMEP "NOMEVARIAVEL

NAMEP permite verificar se a varável cujo nome é dado como entrada existe ou não como uma variável global na memória do micro, retornando sempre um valor booleano "TRUE ou "FALSE.

Ex. :

```
MAKE "RAIZDEDOIS SQRT 2
PR NAMEP "RAIZDEDOIS
TRUE
ERNS
PR NAMEP "RAIZDEDOIS
FALSE
```

- PONS

PONS mostra todas as variáveis globais existentes na memória do micro, dando o nome de cada uma e o seu respectivo conteúdo.

Ex. :

```
PONS
```

- THING "NOMEVARIAVEL

THING fornece o conteúdo da variável cujo nome é dado como entrada. Ela funciona exatamente como o caractere especial : .

Ex. :

```
MAKE "X PI
PR :X
3.141592654
PR THING "X
3.141592654
```

Obs. : THING pode ser usado para dar à uma variável um segundo nível de avaliação.

Ex. :

```
MAKE "FORMULA "H2O
MAKE "H2O "AGUA
MAKE "AGUA "GELO
PR THING "FORMULA
H2O
PR THING :FORMULA
AGUA
PR THING "H2O
AGUA
PR THING :H2O
GELO
```

Em outras palavras, se :FORMULA contém "H2O e :H2O contém "AGUA então :FORMULA contém "AGUA. Isto é útil quando se opera com listas de propriedades.

CAPITULO 4

ESTRUTURAS DE CONTROLE

- CATCH
- CO
- GO
- LABEL
- OUTPUT (OP)
- PAUSE
- REPEAT
- RUN
- STOP
- THROW
- TOPLEVEL
- WAIT

* Condicionais :

- IF
- IFFALSE (IFF)
- IFTRUE (IFT)
- TEST

ESTRUTURAS DE CONTROLE

A linguagem LOGO tem vários comandos que permitem ao usuário definir diferentes estruturas de controle dentro de um programa. A cada estrutura há um ou mais comandos associados que irão controlar o fluxo normal de processamento, fazendo desvios condicionais ou incondicionais, chamando subrotinas externas ao procedimento, repetindo uma seqüência de instruções um número finito de vezes ou até que uma condição seja satisfeita, interrompendo o processamento ou fazendo uma pausa temporária, etc... Neste capítulo veremos quais são as principais estruturas de controle e os seus comandos associados.

- REPEAT :N [lista de instruções]

O comando REPEAT define a estrutura de controle mais simples que há na linguagem LOGO, que é a repetição. Este comando repete por um número finito de vezes uma determinada lista de instruções. Ele é usado em situações onde já sabemos de antemão quantas vezes uma determinada seqüência de instruções deve ser repetida. O valor que :N pode assumir precisa ser um número positivo. Se :N vale zero então a lista de instruções não é executada. Vários comandos REPEAT podem ser encaixados, o que simplifica e facilita a programação. A lista de instruções é sempre executada por completo, não permitindo bifurcações.

Ex. :

```
TO REPETE
PR [VOU REPETIR 5 VEZES O ASTERISCO]
REPEAT 5 [PR [*]]
PR [NAO VOU EXECUTAR A PROXIMA LINHA D
E REPETICAO]
MAKE "N 0
REPEAT :N [PR [NAO VOU IMPRIMIR ESTA L
INHA]]
PR [PARANDO]
END
```

```
TO DESENHO
CS
REPEAT 8 [REPEAT 4 [FD 50 RT 90] RT 45]
END
```

Obs. : Dependendo do tamanho da seqüência de instruções a serem executadas, elas podem não caber dentro da lista, quando estamos definindo o programa com o comando TO. Use a janela de edição.

- **IF (condição) [lista de instruções se a condição for verificada] [lista de instruções se a condição for falsa]**

O comando IF é um comando condicional do tipo SE... ENTÃO... SENÃO... que permite a bifurcação de um programa. Geralmente a condição é uma comparação, usando os operadores relacionais (< = >), mas também podem ser usadas as primitivas lógicas (veja o próximo capitulo). Se a segunda lista de instruções for omitida então o LOGO não fará nada se a condição testada não for verdadeira, pulando automaticamente para a próxima linha do programa.

Ex. :

```
TO TESTANDOIF
MAKE "NUMERO (100-RANDOM 201)
IF (:NUMERO>0) [PR [SORTEEI UM NUMERO
POSITIVO]  PR :NUMERO] [PR [SORTEEI UM
 NUMERO NEGATIVO OU O ZERO]]
TESTANDOIF
END
```

Obs. : Os comandos AND , OR , NOT podem ser usados para alterarmos adequadamente as expressões condicionais.

Ex. :

```
TO REGRADOSSINAIS :X :Y
PR [NAO E PRECISO FAZER O PRODUTO]
PR [PARA SABERMOS O SEU SINAL]
IF (AND (:X<0)(:Y<0))[PR [O PRODUTO E PO
SITIVO]]
IF OR (:X=0)(:Y=0) [PR [O PRODUTO DA ZER
O]]
IF OR (AND(:X<0)(:Y>0))(AND(:X>0)(:Y<0))
 [PR [O PRODUTO E NEGATIVO]]
PR [VAMOS CONFERIR]
PR :X   PR :Y  PR (:X*:Y)
END
```

- **TEST (condição)**
- **IFTRUE [lista de instruções se a condição testada é "TRUE]**
- **IFFALSE [lista de instruções se a condição testada é "FALSE]**

O conjunto de primitivas acima constituem também uma estrutura condicional, cujo funcionamento é parecido com o da primitiva IF. O comando TEST testa se uma determinada condição é verdadeira ou falsa, armazenando um valor booleano "TRUE ou "FALSE, que depois será usado pelos comandos IFTRUE ou IFFALSE. Testada uma condicão, somente um destes dois comandos será executado pelo micro, a cada teste. Se TEST contém "TRUE então a lista de comandos associada ao comando IFTRUE será executada. Se TEST contém "FALSE então será a lista associada ao comando IFFALSE que será executada.

Note que enquanto IF é imediato (no sentido de que se fizermos um teste com IF devemos especificar as respectivas listas de intruções como parte integrante de sua entrada), o comando TEST pode efetuar um teste e ficar retendo um valor booleano até o momento em que o usuário queira que ele seja usado, ou seja, quando for encontrado algum dos comandos IFTRUE ou IFFALSE. Existem situações onde a primitiva IF não pode ser usada e nestas situações é que se torna oportuno o uso dos comandos acima.

Ex. :

```
TO TESTE
PR [DIGITE ALGUMA TECLA PARA PARAR]
TEST KEYP
WAIT 10
PR [AINDA NAO FOI CONFIRMADO SE A]
PR [TECLA FOI ACIONADA]
IFTRUE [PR [OK! ESTOU PARADO] STOP]
TESTE
END
```

Obs. : Os comandos IFTRUE e IFFALSE podem ser abreviados respectivamente por **IFT** e **IFF** .

- WAIT :T

O comando WAIT serve para fazer uma pausa no processamento de um procedimento parando a sua execução por um certo espaço de tempo que é determinado pelo valor da entrada :T. O valor de :T precisa ser um número positivo, e se :T for 1 então a execução fica suspensa por aproximadamente 1/60 de segundo.

Ex. :

```
TO ESPERAR
PR [VOU PARAR POR UM SEGUNDO]
WAIT 60
PR [CONTINUANDO]
END
```

- RUN [lista de instruções]

O comando RUN executa automaticamente uma lista de instruções. É útil quando se quer fazer procedimentos que rodem outros procedimentos sem que eles sejam encarados como subrotinas, ou em procedimentos de finalização de uma sessão de trabalho.

Ex. :

```
TO AUTODESTRUTIVO
RUN LIST "FORWARD 50
RUN [ERASE "AUTODESTRUTIVO  PR [O PROCE
DIMENTO FOI APAGADO] POALL]
END
```

- STOP

O comando STOP suspende a execução de um procedimento, devolvendo o controle ao procedimento que o chamou ou ao modo direto (nível inicial). É usado também para fazer o fim lógico de um programa, junto com o IF. Uma vez parado, o programa não pode ser continuado.

Ex. :

```
TO PARANDO
PR [VOU PARAR]
STOP
PR [ESTA LINHA NAO VAI SER ESCRITA]
END
```

- PAUSE

O comando PAUSE é usado para depurar procedimentos, suspendendo a sua execução para a detecção de erros pelo usuário. Seu funcionamento é idêntico ao do comando STOP, mas permite que o procedimento seja novamente executado, a partir do ponto onde foi parado, mediante o uso do comando de continuação CO. PAUSE é um tipo de "CTRL" + "BREAK" por software pois ele é interno à um procedimento. Quando um procedimento é parado por PAUSE aparece, ao lado do "prompt" de modo direto, uma lista contendo o nome do procedimento que foi interrompido. Para que esta lista desapareça você precisa continuar o processamento do programa que foi parado com o PAUSE e pará-lo com o "CTRL" + "BREAK" manual.

Ex. :

```
TO PAUSA
PR [VOU PARAR]
PR [PARA CONTINUAR DIGITE  CO]
PR [E TECLE RETURN]
PAUSE
PR [VOU CONTINUAR]
PR [DO PONTO ONDE PAREI]
END
```

Obs. : A tecla F5 tem o mesmo efeito que o comando PAUSE e pode ser acionada manualmente quando desejarmos, fazendo uma pausa por hardware (via teclado).

- CO

Continua o processamento de um procedimento que foi parado pelo comando PAUSE ou pelo acionamento da tecla F5 , a partir do ponto onde ele parou.

Ex. :

```
TO CIRCULO
FD 1  RT 1
CIRCULO
END
```

Obs. : Execute o programa acima, acione a tecla F5 manualmente e digite CO para continuar.

- OUTPUT :OBJETO

O comando OUTPUT suspende a execução de um procedimento, enviando um objeto para ser usado como input do procedimento que o chamou. Os objetos podem ser tanto números quanto palavras ou listas. Ele é usado frequentemente para criar funções matemáticas que não existem no LOGO. Para maiores detalhes leia também o capítulo 8.

Ex. :

```
TO LOG :X
IF (:X>0) [OUTPUT (LN :X)/(LN 10)]
OP [ARGUMENTO FORA DO DOMINIO DA FUNCAO]
END

PR LOG 2
0.3010299957

PR LOG -1
ARGUMENTO FORA DO DOMINIO DA FUNCAO
```

Obs. : OUTPUT pode ser abreviado por **OP**.
Nota : Procedimentos que tem OUTPUT precisam ser usados em conjunto com outras primitivas ou procedimentos.

- GO "ROTULO
- LABEL "ROTULO

Os comandos GO - LABEL são usados em conjunto e frequentemente servem para fazer desvios incondicionais ou criar laços repetitivos, internos à um procedimento, que serão repetidos até que uma determinada condição seja satisfeita. O comando LABEL serve para criar um rótulo (nome que indica um determinado trecho do procedimento) que será depois usado pelo comando GO para fazer o desvio. O nome para o rótulo pode ser qualquer, mas uma vez escolhido ele deve ser o mesmo tanto para LABEL quanto para o GO. Pode-se ter mais de um laço interno em um programa e eles podem ser encaixados.

Ex. :

```
TO DESVIANDO_E_CONTANDO_MATRIZ
GO "FIM
LABEL "CONTARX
IF :X>10 [STOP]
MAKE "Y 0
LABEL "CONTARY
MAKE "Y :Y+1
IF :Y>10 [MAKE "X :X+1 PR [] GO "CONTA
RX]
TYPE :X TYPE "\,  TYPE :Y TYPE "\ \ \
GO "CONTARY
LABEL "FIM
MAKE "X 1
PR [ESTOU NA ULTIMA LINHA DO PROGRAMA]
PR [VOU COMECAR A CONTAR]
GO "CONTARX
END
```

- CATCH "ENDEREÇO [lista de instruções]
- THROW "ENDEREÇO
- THROW "TOPLEVEL

Os comandos CATCH e THROW são usados frequentemente para definir subrotinas externas à um procedimento.

O comando CATCH executa a lista de instruções (geralmente nomes de subrotinas) guardando o endereço de retorno à rotina principal na variável "ENDEREÇO. O comando THROW é executado somente quando se deseja que o controle volte para a rotina principal, continuando sua execução na linha posterior ao respectivo CATCH. Os nomes para os endereços de CATCH e THROW podem ser quaisquer palavras a escolha do usuário, bem entendido que devem sempre coincidir. Se queremos que o controle volte para o nível inicial (modo direto) devemos usar a palavra reservada "TOPLEVEL. Analise o seguinte programa com cuidado.

Ex. :

```
TO PRINCIPAL
MAKE "X 1
PR [VOU PARA A SUBROTINA]
CATCH "SUB [SUBROTINA]
PR :X
THROW "TOPLEVEL
PRINCIPAL
END
```

```
TO SUBROTINA
IF :X>0 [AUXILIAR]
PR [ESTOU PRESO NA SUBROTINA]
SUBROTINA
END

TO AUXILIAR
PR :X
IF :X>5 [THROW "SUB]
MAKE "X :X+1
AUXILIAR
END
```

O que aconteceria se ao invés de IF :X>5 [THROW "SUB] tivessemos IF :X>5 [STOP] ?

Como todos os procedimentos chamam a sí próprios, se usassemos o comando STOP o controle ficaria preso, saltando da subrotina para o auxiliar e vice-versa.Agora se usamos THROW a execução é automaticamente desviada para o programa principal independentemente se alguma subrotina intermediária foi concluída ou não.

CAPITULO 5

LOGICA

- AND
- BUTTONP
- EQUALP
- FALSE
- KEYP
- NOT
- OR
- PRIMITIVEP
- SHOWNP
- TRUE

LØGICA

Os comandos deste capítulo são chamados de primitivas lógicas pois fornecem como resultado somente dois valores booleanos "TRUE ou "FALSE e são usados geralmente em testes ou em expressões condicionais, junto com os comandos IF ou TEST. AND e OR são usados para que se tenha mais de uma entrada no comando IF, numa expressão condicional. NOT é usado para fazer a negação de uma expressão. Vejamos tais comandos:

- **AND :condição1 :condição2**

O comando AND faz a operação lógica **e** entre duas condições dadas como entrada, devolvendo um valor booleano "TRUE ou "FALSE. O resultado será "TRUE (verdadeiro) somente se todas as condições dadas como entrada forem verdadeiras, caso contrário será "FALSE. Podemos ter mais de duas expressões condicionais dadas como entrada para este comando, desde que se use adequadamente os parêntesis.

Ex. :

```
PR AND (2<3) (1<2)
TRUE

PR (AND (2<3) (1>2) (-5<0))
FALSE


TO TESTE
MAKE "X 1-RANDOM 3
MAKE "Y 1-RANDOM 3
MAKE "Z 1-RANDOM 3
IF (AND (:X>0)(:Y>0)(:Z>0))[PR [OS
TRES SAO POSITIVOS]][(PR :X :Y :Z)]
TESTE
END
```

- **BUTTONP :n**

O comando BUTTONP verifica se o botão do joystick indicado pelo numero :n foi acionado ou não.

Ex. :

```
TEST BUTTONP 0
```

- EQUALP :expressão1 :expressão2

O comando EQUALP verifica a condição de igualdade entre duas expressões. Seu funcionamento é análogo ao do caractere especial de igualdade = . Esta primitiva aceita somente duas entradas e devolve sempre um valor booleano "TRUE ou "FALSE, segundo os objetos ou expressões dadas como entrada serem iguais ou não.

Ex. :

```
PR EQUALP 2.34 2.34
TRUE

PR EQUALP "PALAVRA "PALAVRAS
FALSE

PR EQUALP 1 [1]
FALSE

PR EQUALP (1<2) (2<3)
TRUE

TO SAOIGUAIS
MAKE "X 1-RANDOM 3
MAKE "Y 1-RANDOM 3
IF EQUALP :X :Y [PR [SAO IGUAIS]
STOP][PR [SAO DIFERENTES]]
PR :X PR :Y
SAOIGUAIS
END
```

- "FALSE

FALSE é uma palavra reservada, usada em expressões condicionais ou em testes e indica sempre o valor booleano FALSO. Note que por ser uma palavra ela precisa estar sempre precedida por aspas " .

Ex. :

```
TO FALSO
IF (2>3)="FALSE [PR [2 NAO E MAIOR QUE
3 ]]
END
```

- KEYP

O comando KEYP serve para verificar se alguma tecla foi acionada ou não. Note que ao contrário do comando READCHAR, o comando KEYP não interrompe o processamento para ficar à espera de que um caractere seja teclado pelo usuário. Ele simplesmente testa se há algum caractere presente no "buffer" do teclado, retornando então um valor booleano "TRUE ou "FALSE .

Ex. :

```
TO TECLA
TEST KEYP
IFTRUE [PR [UMA TECLA FOI ACIONADA]
PR [E O PROCESSAMENTO FOI INTERROMPIDO]
PR [PELO COMANDO STOP A SEGUIR] STOP]
PR [ACIONE ALGUMA TECLA PARA PARAR]
TECLA
END
```

- NOT :expressão

O comando NOT é usado em expressões condicionais e serve para fazer a negação de uma expressão booleana, resultando "TRUE se a expressão for falsa e "FALSE se ela for verdadeira. O comando NOT aceita somente uma expressão como entrada, porém esta expressão pode ser composta por uma ou mais expressões simples, desde que se use adequadamente os comandos AND e/ou OR. Veja o seguinte exemplo:

Como obter a expressão **(x≥0) e (x≤5)** se não temos os operadores ≥ e ≤ nesta linguagem ? Observe que **(x≥0) e (x≤5)** equivale à negação da expressão **(x<0) ou (x>5)** e daí basta usarmos a negação da segunda expressão ao invés da primeira.

Ex. :

```
PR NOT (2<3)
FALSE


PR NOT EQUALP 1 [1]
TRUE
```

:

```
TO INTERVALOFECHADO
MAKE "X 15-RANDOM 26
IF NOT OR ( :X<0)( :X>5) [PR [X PERTENCE
AO INTERVALO FECHADO [0 5]]] [PR [X PE
RTENCE AO COMPLEMENTAR DO INTERVALO FE
CHADO [0 5]]]
PR :X
END
```

- OR :condição1 :condição2

O comando OR faz a operação lógica **ou** entre duas expressões ou condições dadas como entrada, resultando num valor booleano "TRUE ou "FALSE. O resultado será "FALSE somente se todas as condições de entrada forem falsas. O comando OR aceita mais de duas condições como entrada, desde que se use os parêntesis adequadamente.

Ex. :

```
PR OR (1<2) (2<3)
TRUE

PR OR (2<1) (2<3)
TRUE

PR OR (2<1) (3<2)
FALSE


TO TESTAPRODUTO
MAKE "X 1-RANDOM 3
MAKE "Y 1-RANDOM 3
MAKE "Z 1-RANDOM 3
IF (OR (:X=0)(:Y=0)(:Z=0))[PR [O PRODUTO
DA ZERO]]
PR :X PR :Y PR :Z PR :X*:Y*:Z
END
```

- PRIMITIVEP "PALAVRA

O comando PRIMITIVEP é usado para testar se uma determinada palavra dada como entrada é ou não uma primitiva da linguagem LOGO.

Ex. :

```
PR PRIMITIVEP "PALAVRA
FALSE


PR PRIMITIVEP "PRINT
TRUE


TO TESTATITULOS :PALAVRA
TEST PRIMITIVEP :PALAVRA
IFTRUE [PR [ESTA PALAVRA NAO PODE SER US
ADA COMO NOME DE UM PROCEDIMENTO POIS JA
E UMA PRIMITIVA DA LINGUAGEM]]
END
```

- SHOWNP

O comando SHOWNP permite verificar se a tartaruga está visível ou não.

Obs. : Ela pode não estar visível por ter assumido a cor do fundo ou por ter o formato do caractere ASCII 32 (branco).

Ex. :

```
CS SHOWTURTLE
PR SHOWNP
TRUE
HT
PR SHOWNP
FALSE
ST SETSHAPE 32
PR SHOWNP
TRUE
SETSHAPE "TURTLE
```

- "TRUE

TRUE é uma palavra reservada, usada em expressões condicionais ou em testes e indica sempre o valor booleano VERDADEIRO. Note que por ser uma palavra ela precisa sempre ser precedida por aspas " .

Ex. :

```
TO VERDADE
MAKE "X 10-RANDOM 21
IF NOT (:X<0)="TRUE [PR [X E POSITIVO]]
PR :X
END


PR EQUALP (2<3) "TRUE
TRUE
```

CAPITULO 6

PALAVRAS E LISTAS

- ASCII
- BUTFIRST (BF)
- BUTLAST (BL)
- CHAR
- COUNT
- EMPTYP
- FIRST
- FPUT
- ITEM
- LAST
- LIST
- LISTP
- LPUT
- MEMBERP
- NUMBERP
- SENTENCE (SE)
- WORD
- WORDP

:

PALAVRAS E LISTAS

Palavras e listas são objetos do LOGO tais quais as variáveis numéricas. Neste capítulo vamos ver quais são as primitivas que operam sobre palavras e listas e permitem manipular os seus conteúdos. Antes de mais nada devemos esclarecer que palavras são quaisquer seqüências de caracteres alfanuméricos precedidos por aspas " . Note que todo número é também uma palavra. Se uma palavra precisa conter caracteres especiais então eles devem ser precedidos pela barra invertida \. Listas são quaisquer objetos delimitados pelos colchetes [] e separados por espaços em branco. Toda lista pode conter números, palavras ou listas. Vejamos alguns exemplos :

```
                "PALAVRA
                "ALFANUMERICOS\ \E\ NUMEROS\:1234.567
PALAVRAS        "1.4142135
                17424
                "\*\-\*

                [ESTA E UMA LISTA]

                [TODA LISTA PODE CONTER NUMEROS 1 2 3 4
                5 PALAVRAS "MANUAL "DO "LOGO, PRIMITIVAS
LISTAS          FORWARD, PRINT OU LISTAS [LISTA DENTRO D
                E LISTA] [POSICAO [10 20]]]
```

São os seguintes comandos que operam sobre palavras ou listas:

- **ASCII "caractere**

O comando ASCII fornece o código numérico da caractere dado como entrada. Este código segue a tabela padrão do American Standard Code of Information Interchange (ASCII), que é usual aos micros IBM-PC.

Ex. :

```
PR ASCII "A
65
PR ASCII "1
49

TO TABELAASC
PR [DIGITE UM CARACTERE]
MAKE "CARACTERE RC
TYPE [O CARACTERE TECLADO FOI :]
TYPE :CARACTERE
TYPE "\ \
TYPE [E SEU CODIGO E :]
PR ASCII :CARACTERE
TABELAASC
END
```

- BUTFIRST "palavra
- BUTFIRST [lista]

O comando BUTFIRST é usado com palavras ou listas e cria uma nova palavra (lista) que é formada pela antiga (dada como entrada) menos o seu primeiro caractere (respectivamente elemento).

Ex. :
```
PR BUTFIRST "PALAVRA
ALAVRA

PR BF [ESTA E UMA LISTA]
E UMA LISTA
```

Obs. : O comando BUTFIRST pode ser abreviado por **BF**.

- BUTLAST "palavra
- BUTLAST [lista]

O comando BUTLAST fornece uma nova palavra (lista) que é formada pela palavra (lista) que foi dada como entrada menos o seu último caractere (elemento).

Ex. :
```
PR BUTLAST "PALAVRA
PALAVR

PR BUTLAST [ESTA E UMA LISTA]
ESTA E UMA
```

Obs. : O comando BUTLAST pode ser abreviado por **BL**.

- CHAR :número

O comando CHAR fornece o caractere correspondente ao número dado como entrada. O número precisa ser um valor inteiro entre 0 e 255. Consulte uma tabela padrão ASCII para maiores detalhes.

Ex. :
```
PR CHAR 90
Z

TO TABELAASCII
MAKE "X 0
REPEAT 256 [ TYPE "CARACTERE\ TYPE
CHAR :X TYPE "\ \ \   TYPE [CODIGO]
TYPE "\ \   PR :X MAKE "X :X+1]
END
```

- COUNT "palavra
- COUNT [lista]

O comando COUNT fornece o número de letras de uma palavra ou a quantidade de elementos que compõe uma lista.

Ex. :

```
PR COUNT "PALAVRA
7

PR COUNT [ISTO E UMA LISTA]
4

PR COUNT []
0

PR COUNT 13.45
5
```

- EMPTYP [lista]

O comando EMPTYP retorna um valor booleano "TRUE ou "FALSE segundo uma lista esteja vazia ou não.

Ex. :

```
PR EMPTYP []
TRUE

PR EMPTYP [ESTA E UMA LISTA]
FALSE
```

Obs. : As listas vazias podem ser criadas com o comando MAKE da seguinte forma : MAKE "LISTAVAZIA [] .

Nota : Com o devido cuidado o comando EMPTYP também pode ser aplicado à palavras.

Ex. :

```
TO TRIANGULAR :PALAVRA
IF EMPTYP :PALAVRA [STOP]
PR :PALAVRA
TRIANGULAR BUTLAST :PALAVRA
END
```

- FIRST "palavra
- FIRST [lista]

O comando FIRST fornece o primeiro caractere de uma palavra ou o primeiro elemento de uma lista.

Ex. :

```
PR FIRST "PALAVRA
P

PR FIRST [ISTO E UMA LISTA]
ISTO
```

- FPUT :objeto :lista

O comando FPUT cria uma nova lista, que é obtida acrescentando-se o objeto dado como entrada na frente da lista já existente.

Ex. :

```
MAKE "LISTA FPUT "PALAVRA [NA LISTA]
SHOW :LISTA
[PALAVRA NA LISTA]

TO SORTEIA_N_NUMEROS_E_PARA :N
MAKE "LISTA []
LABEL "REPETE
IF (COUNT :LISTA)=:N [SHOW :LISTA STOP]
MAKE "LISTA FPUT RANDOM 101 :LISTA
GO "REPETE
END
```

- ITEM :número :objeto

O comando ITEM fornece o elemento que está na posição :número do objeto dado como entrada. O objeto pode ser tanto um número, uma palavra ou uma lista.

Ex. :

```
PR ITEM 3 [ESTA E UMA LISTA]
UMA

PR ITEM 5 "PALAVRA
V

PR ITEM 1 732
7

TO SORTEIA_UMA_LETRA_ALEATORIA_DO :NOME
PR ITEM (1+RANDOM COUNT :NOME) :NOME
SORTEIA_UMA_LETRA_ALEATORIA_DO :NOME
END
```

- LAST "palavra
- LAST [lista]

O comando LAST fornece o último caractere de uma palavra ou o último elemento de uma lista que é dada como entrada.

Ex. :

```
PR LAST "PALAVRA
A

PR LAST [ESTA E UMA LISTA]
LISTA
```

- LIST :objeto1 :objeto2

O comando LIST cria uma lista com os dois objetos dados como entrada. Qualquer um dos dois objetos pode ser tanto um número, palavra ou lista. O número de argumentos de entrada pode ser aumentado, se usarmos adequadamente os parêntesis.

Ex. :

```
SHOW LIST 2 3
[2 3]

SHOW (LIST 123 "PALAVRA [E UMA LISTA])
[123 PALAVRA [E UMA LISTA]]
```

- LISTP :objeto

O comando LISTP fornece um valor booleano "TRUE ou "FALSE segundo o objeto dado como entrada seja ou não uma lista.

Ex. :

```
PR LISTP [ISTO E UMA LISTA]
TRUE

PR LISTP []
TRUE

PR LISTP "PALAVRA
FALSE
```

- LPUT :objeto :lista

O comando LPUT cria uma nova lista, que é obtida acrescentando-se o objeto dado como entrada ao final da lista já existente.

Ex. :

```
MAKE "LISTA LPUT "PALAVRAS [ESTA E UMA
LISTA DE]
SHOW :LISTA
[ESTA E UMA LISTA DE PALAVRAS]

TO JUNTANOFIM
MAKE "LISTA []
LABEL "REPETE
PR [DIGITE UMA PALAVRA] MAKE "NOME RW
MAKE "LISTA LPUT :NOME :LISTA
PR :LISTA
GO "REPETE
END
```

- MEMBERP :elemento :lista

O comando MEMBERP verifica se um determinado elemento dado como entrada pertence ou não à uma lista, retornando um valor booleano "TRUE ou "FALSE . Este comando também pode ser usado com palavras ou números.

Ex. :

```
PR MEMBERP "A [A B C]
TRUE

PR MEMBERP "A [X Y Z]
FALSE

PR MEMBERP "A "PALAVRA
TRUE

PR MEMBERP "A "ISSO
FALSE

PR MEMBERP 2 1234
TRUE

PR MEMBERP 23 1234
TRUE

MAKE "ELEMENTO "UMA
MAKE "LISTA [ISTO E UMA LISTA]
PR MEMBERP :ELEMENTO :LISTA
TRUE
```

- NUMBERP :objeto

O comando NUMBERP fornece um valor booleano "TRUE ou "FALSE segundo um objeto dado como entrada seja ou não um número.

Ex. :

```
PR NUMBERP 23.5
TRUE

PR NUMBERP "23.5
TRUE

PR NUMBERP PI
TRUE

PR NUMBERP [23]
FALSE

PR NUMBERP "PALAVRA
FALSE
```

-SENTENCE :objeto1 :objeto2

O comando SENTENCE forma uma sentença com os dois objetos dados como entrada, criando uma lista. Quaisquer dos dois objetos podem ser números, palavras ou listas. Se algum dos objetos é uma lista então somente o seu conteúdo é colocado dentro da lista que é formada, ou seja, este comando elimina os colchetes mais externos das listas. O número de argumentos de entrada pode ser aumentado, se usarmos parêntesis.

Obs. : O comando SENTENCE pode ser abreviado por SE.

Ex. :

```
SHOW SENTENCE 2 3
[2 3]

SHOW (SENTENCE 123 "PALAVRA [ISTO E UMA
LISTA])
[123 PALAVRA ISTO E UMA LISTA]

PRINT SENTENCE [ESTA E UMA][SENTENCA DE
PALAVRAS]
ESTA E UMA SENTENCA DE PALAVRAS
```

- WORD "palavra1 "palavra2

O comando WORD forma uma palavra composta que é obtida através da concatenação das duas palavras dadas como entrada. O número de argumentos de entrada pode ser aumentado, usando-se os parêntesis circundando a primitiva e as respectivas entradas.

Ex. :

```
PR WORD "GUARDA "ROUPA
GUARDAROUPA

PR WORD "12 "34.56
1234.56

PR WORD 12 34.56
1234.56

PR (WORD "MAIS "DE "DUAS "ENTRADAS)
MAISDEDUASENTRADAS

PR(WORD CHAR 76 CHAR 79 CHAR 71 CHAR 79)
LOGO
```

- WORDP :objeto

O comando WORDP verifica se um determinado objeto dado como entrada é uma palavra ou não, retornando um valor booleano "TRUE ou "FALSE. Note que números são casos particulares de palavras.

Ex. :

```
PR WORDP "PALAVRA
TRUE

PR WORDP [ESTA E UMA LISTA]
FALSE
```

CAPITULO 7

COMANDOS DE INPUT/OUTPUT

- PRINT (PR)
- READCHAR (RC)
- READCHARS (RCS)
- READLIST (RL)
- READWORD (RW)
- SHOW
- TYPE

COMANDOS DE ENTRADA E SAIDA

Os comandos de INPUT/OUTPUT são responsáveis pela entrada e saída de dados em um programa, durante o seu processamento. A entrada de dados pode ocorrer via teclado ou por leitura de arquivos. A saída dos dados pode ser feita por impressão de caracteres na tela do monitor, na impressora ou escrevendo-se dados num arquivo em disco. No capítulo que trata sobre arquivos veremos como fazer para habilitar a leitura ou escrita de dados em um disco magnético (tanto faz se é disquete ou "winchester"). Embora eles usem as mesmas primitivas de INPUT/OUTPUT, precisam de outras primitivas especiais que preparam e habilitam o transporte dos dados. Neste capítulo nos deteremos apenas nos meios ou dispositivos padrão de entrada e saída, que são o teclado e o vídeo em modo texto. Para saber como imprimir dados simultaneamente na tela e na impressora consulte o capítulo 11, que descreve detalhadamente o uso da impressora serial. Para imprimir caracteres na tela gráfica veja como exemplo o programa "ESCREVE" no capítulo 1.

Vejamos agora quais são os comandos de INPUT/OUTPUT:

- **PRINT :dado**

O comando **PRINT** escreve um dado na tela de texto, fazendo troca de linha, ou seja, numa próxima impressão, o novo dado será impresso no começo da linha seguinte.

Ex. :

```
TO PRINTANDO
PRINT 23
PRINT "PALAVRA
PRINT [ESTA E UMA LISTA]
PR PI
PRINT []
MAKE "DADO SQRT 2
PR :DADO
END
```

Obs. : O comando PRINT pode ser abreviado por PR .

Nota : Para imprimir uma linha em branco use **PR []** .

- READCHAR

O comando **READCHAR** lê um caractere do teclado. Quando o **LOGO** encontra esta primitiva dentro de um procedimento ele suspende a sua execução, ficando à espera de que um caractere seja teclado pelo usuário. Tão logo o caractere é lido, o processamento continua de onde parou. Note que não é necessário teclar "RETURN" após digitar o caractere. Observe também que ele não fica impresso na tela do micro.

Ex. :

```
TO CARACTERES
MAKE "LETRA READCHAR
PRINT :LETRA
CARACTERES
END
```

Obs. : READCHAR pode ser abreviado por **RC** .

- READCHARS :n

READCHARS suspende a execução de um procedimento, ficando à espera para que sejam lidos :n caracteres do teclado. O valor de :n precisa ser um número inteiro positivo. Se :n vale 1 então a primitiva funciona exatamente como o RC. Note que os :n caracteres teclados pelo usuário não aparecem impressos na tela e nem é preciso teclar "RETURN".

Ex. :

```
TO PALAVRASDE8LETRAS
MAKE "PALAVRA READCHARS 8
PR :PALAVRA
PALAVRASDE8LETRAS
END
```

Obs. : READCHARS pode ser abreviado por **RCS** .

- READLIST

O comando **READLIST** lê uma lista de dados do teclado. A lista de dados deve ser toda ela digitada em uma única linha. Isto significa que você não pode teclar "RETURN" enquanto não tiver digitado todos os dados. Note que mesmo que o **LOGO** troque de linha os dados que estão aparacendo na tela, isto não quer dizer que é uma nova lista que está sendo lida. Perceba que neste caso aparece a seta no canto direito da linha atual, indicando que ela ainda continua na próxima. Cada um dos dados precisa estar separado dos outros por meio de um espaço em branco.

Não existem restrições quanto ao tipo de dados que compõe a lista (podem ser números, palavras ou listas). A quantidade de dados que podem ser digitados fica restrita apenas ao volume de memória disponível no momento. Quando todos os dados foram digitados então tecle "RETURN".

Ex. :

```
TO LENDOLISTAS
PR [DIGITE UM NUMERO UMA PALAVRA E UMA
LISTA]
MAKE "LISTA READLIST
PR :LISTA
END
```

Obs. : READLIST pode ser abreviado por **RL** .

Nota : Veja maiores detalhes sobre operações com listas no capítulo 6 .

- READWORD

O comando **READWORD** lê uma palavra do teclado. Não há restrições quanto ao número de caracteres que vão ser teclados, porém não podem ser digitados nem espaços em branco e nem caracteres especiais (a menos que cada um deles esteja precedido pela barra invertida \ ). Depois do usuário ter digitado a palavra desejada, ele deve teclar "RETURN", para indicar ao **LOGO** que deve retornar ao processamento, continuando do ponto de onde ele parou. Todas as palavras que já foram digitadas permanecem visíveis na tela.

Ex. :

```
TO LENDO_PALAVRAS
MAKE "PALAVRA READWORD
PR :PALAVRA
LENDO_PALAVRAS
END
```

Obs. : READWORD pode ser abreviado por **RW** .

- SHOW :dado

O comando **SHOW** visualiza o objeto que está em :dado, mostrando-o na tela texto. Ele funciona exatamente como o comando **PRINT**, diferindo apenas no caso de listas, pois enquanto **PRINT** apenas imprime o conteúdo de uma lista, o comando **SHOW** imprime a lista exatamente como ela é, mostrando inclusive os colchetes mais externos.

Ex. :

```
TO MOSTRANDO
SHOW 23  SHOW "PALAVRA
SHOW [ESTA E UMA LISTA]
SHOW PI  SHOW []
SHOW [[ESTA E][UMA LISTA][DE LISTAS]]
END
```

- TYPE :dado

O comando **TYPE** funciona exatamente como o comando **PRINT** , só que não faz a troca de linha quando termina a impressão. Qualquer outro dado que for impresso depois sairá na mesma linha, ao lado do último dado impresso pelo comando **TYPE** anterior. Analise o segunte exemplo e compare-o com o programa PRINTANDO :

Ex. :

```
TO TYPANDO
TYPE 23
TYPE "PALAVRA
TYPE [ESTA E UMA LISTA]
TYPE PI  TYPE []
MAKE "DADO SQRT 2
TYPE :DADO
PR [VOU MUDAR DE LINHA [E PARAR]]
END
```

Obs. : Usando os comandos PRINT e TYPE adequadamente podemos montar frases com variáveis no meio. Veja o exemplo abaixo :

```
TO CONVERSA
CT
TYPE [COMO E SEU NOME ?]
MAKE "NOME RW
TYPE [Ok! , ]
TYPE :NOME
PR [ . QUERO SER SEU AMIGO. ]
PR [] PR []
END
```

CAPITULO 8

FUNÇÕES MATEMATICAS

- ARCTAN
- COS
- DIFFERENCE
- EXP
- INT
- LN
- PI
- POWER
- PRECISION
- PRODUCT
- QUOTIENT
- RANDOM
- REMAINDER
- RERANDOM
- ROUND
- SETPRECISION
- SIN
- SQRT
- SUM

FUNÇÕES MATEMATICAS

A linguagem LOGO já tem várias funções matemáticas pré-definidas, prontas para serem usadas, como por exemplo o seno, o cosseno, a arco-tangente, logaritmo natural, exponenciação, potenciação, decomposição de um numero em suas partes inteira e fracionária, etc... As funções matemáticas que não estão pré-definidas podem ser facilmente programadas pelo usuário, através do comando **OUTPUT**. Veremos como fazer isto no final deste capítulo. Vejamos agora as principais funções matemáticas pré-definidas do LOGO.

- ARCTAN :X

ARCTAN fornece o valor do arco (ângulo em graus entre -90 e 90) cuja tangente vale :X . O valor do argumento pode ser qualquer número real, ou mesmo uma expressão que resulte num valor numérico.

Ex. :

```
PR ARCTAN 1
45

MAKE "X 2
PR ARCTAN (SQRT :X)/:X
35.26438968
```

- COS :A

COS fornece o valor do cosseno de :A , onde :A é o valor (em graus) da medida de um ângulo.

Ex. :

```
PR COS 60
0.5

MAKE "RAIZDEDOIS (2*COS 45)
PR :RAIZDEDOIS
1.414213562
```

- DIFFERENCE :X :Y

DIFFERENCE fornece a diferença entre dois valores :X e :Y, ou seja, faz o mesmo que (:X - :Y). Os argumentos de entrada :X e :Y podem ser tanto números reais como expressões numéricas.

Ex. :

```
PR DIFFERENCE 3 2
1

PR DIFFERENCE 2 3
-1

PR DIFFERENCE SQRT 3 SQRT 2
0.317837246
```

- EXP :X

EXP fornece o valor da função exponencial de :X , onde :X pode ser tanto um número real como uma expressão numérica.

Ex. :

```
PR EXP 10
22026.46579

MAKE "e EXP COS 0
PR :e
2.718281828
```

- INT :X

INT fornece a parte inteira do número :X . O arredondamento usado é do tipo corte, ou seja, INT despreza a parte fracionária do número dado como entrada, independente de seu valor. O argumento :X pode ser qualquer numero ou expressão numérica.

Ex. :

```
PR INT 2.999
2

PR INT 2.001
2

PR INT EXP 10
22026
```

- LN :X

LN fornece o valor do logaritmo natural de :X . O argumento :X pode ser tanto um número real como uma expressão numérica, mas seu valor tem que ser positivo, caso contrário vai ocorrer um erro, com a interrupção do processamento e com o aparecimento da respectiva mensagem de erro na tela.

Ex. :

```
PR LN 10
2.302585093

PR LN EXP 2.5
2.5

MAKE "LOG_DE_10_NA_BASE_2 (LN 10)/(LN 2)
```

Nota : Para obter o valor do logaritmo de um número em outra base basta observar que $\log_B(X) = LN(X)/LN(B)$.

- PI

PI é a primitiva do LOGO que fornece o valor da constante π.

Ex. :
```
PR PI
3.141592654

PR COS PI
0.9984971499
```

Note que PI é o valor de um ângulo que está sendo medido em radianos e não em graus. É por isso que a resposta para PR COS PI não foi -1. Para conversões entre unidades use a seguinte regra:

ANGULO EM RADIANOS = ANGULO EM GRAUS * PI / 180

- POWER :X :Y

POWER é a função de potenciação universal. Ela fornece o valor de :X elevado ao expoente :Y . Aqui tanto :X como :Y podem ser números ou expressões numéricas, mas o valor de :X precisa ser positivo, caso contrário a operação irá ocasionar uma mensagem de erro.

Ex. :
```
PR POWER 2 10
1024

MAKE "RAIZCUBICADE2 POWER 2 (1/3)
```

Nota : A operação de radiciação pode ser obtida com o mesmo comando POWER pois basta observar que $\sqrt[Y]{X} = X^{1/Y}$ e usar a seguinte regra prática : MAKE "RAIZ_Y_DE_X POWER :X (1/:Y)

- PRECISION

PRECISION é a primitiva que serve para informar a precisão atual com que os cálculos são efetuados, ou seja, ela fornece como resposta a quantidade máxima de dígitos significativos que podem compor a mantissa de um número num dado momento. Para maiores detalhes veja também SETPRECISION.

Ex. :
```
PR PRECISION
10
```

- PRODUCT :X :Y

PRODUCT fornece o produto entre dois valores :X e :Y, ou seja, faz o mesmo que (:X * :Y). Os argumentos de entrada podem ser tanto números reais quanto expressões numéricas.

Ex. :
```
PR PRODUCT 2 3
6
```

- QUOTIENT :X :Y

QUOTIENT fornece o resultado da divisão de :X por :Y, ou seja, o quociente de :X dividido por :Y. Tanto :X como :Y podem ser números reais ou expressões numéricas, mas o valor de :Y precisa ser diferente de zero para não dar erro.

Ex. :

```
PR QUOTIENT 4 3
1.333333333

PR QUOTIENT 8 2.5
3.2

PR QUOTIENT POWER 10 8 INT EXP 10
4540.088986
```

- RANDOM :N

RANDOM é a primitiva que gera números aleatórios inteiros entre 0 e (:N-1) . O valor de :N precisa ser um número inteiro maior que 1.

Ex. :

```
REPEAT 15 [ PR (RANDOM 100)+1 ]

REPEAT 10 [PR RANDOM 2]
```

Obs. : Para gerar números aleatórios negativos basta tomar -RANDOM :N .

Ex. :

```
TO ANOITECER
CS HT CT
REPEAT 4 [REPEAT 4[FD 100 RT 90] RT 90]
PU SETPOS [-110 0] PD FILL
REPEAT 400 [DOT LIST 100-RANDOM 201 100
- RANDOM 201]
END
```

- REMAINDER :X :Y

REMAINDER fornece o resto inteiro da divisão de :X por :Y. Aqui tanto o valor de :X como o de :Y precisam ser números inteiros e , além disto, o valor de :Y tem que ser diferente de zero.

Ex. :

```
PR REMAINDER 4 3
1

PR REMAINDER 2 2
0

PR REMAINDER POWER 10 8 INT EXP 10
1960
```

- RERANDOM

RERANDOM reinicializa a semente que vai gerar os números aleatórios de forma que saia sempre a mesma seqüência gerada.

Ex. :

```
RERANDOM  REPEAT 10 [PR RANDOM 10]
RERANDOM  REPEAT 10 [PR RANDOM 10]
RERANDOM  REPEAT 10 [PR RANDOM 100]
RERANDOM  REPEAT 20 [PR RANDOM 100]
```

- ROUND :X

ROUND arredonda o valor de :X para o inteiro mais próximo.

Ex. :

```
PR ROUND 1
1
PR ROUND 1.25
1
PR ROUND 1.5
2
PR ROUND 1.89
2
PR ROUND -43.6789
-44
```

Obs. : O seguinte programa pode ser útil caso você queira fazer um arredondamento clássico de :X com :N casas após a vírgula.

```
TO ARREDONDE :X :N
OUTPUT (POWER 10 (-:N))*ROUND(:X*POWER 10 :N)
END
```

Ex. :

```
PR ARREDONDE PI 3
3.142

PR ARREDONDE SQRT 2 5
1.41421

MAKE "VALOR ARREDONDE SQRT 654321 2
PR :VALOR
808.9
```

Nota : O valor de :N no programa ARREDONDE precisa ser sempre um número inteiro entre 0 e o valor dado pelo comando PRECISION. Se :N for negativo ou não inteiro vão ocorrer erros sem que apareça a mensagem avisando!

- SETPRECISION :N

SETPRECISION é a primitiva que permite ao usuário alterar a quantidade de dígitos que compõe a mantissa dos números, mudando para :N a sua precisão. O valor de :N precisa ser um número inteiro entre 5 e 100. O valor "default" é 10.

Ex. :

```
TO PRECISAO
PR PI
PR PRECISION
SETPRECISION 25
PR PI
PR PRECISION
END
```

Obs. : Qualquer que seja a precisão adotada pelo usuário, as funções matemáticas básicas do LOGO (SIN,COS,EXP,LN,etc...) sempre oferecem resultados com exatidão de máquina, porém quanto maior for a precisão mais demorados se tornam os cálculos. Veja também o comando PRECISION.

- SIN :A

SIN fornece o valor do seno de :A , onde :A é o valor (em graus) da medida de um ângulo.

Ex. :

```
PR SIN 30
0.5

PR SIN ARCTAN 1
0.7071067812
```

- SQRT :X

SQRT fornece o valor da raiz quadrada de :X . O argumento :X pode ser tanto um número real como uma expressão numérica, mas seu valor não pode ser negativo.

Ex. :

```
PR SQRT 2
1.414213562

PR SQRT EXP COS RANDOM 361
(qualquer valor entre 0.6065 e 1.65)
```

- SUM :X :Y

SUM fornece a soma de dois valores :X e :Y , ou seja , faz a mesma coisa que (:X + :Y) . Os argumentos de entrada :X e :Y podem ser tanto números reais quanto expressões numéricas.

Ex. :

```
PR SUM 1 2
3
```

DEFININDO FUNÇÕES NO LOGO

Como fazer para obter o valor de Arccos 0.7 ?

Não existe primitiva na linguagem LOGO que sirva para calcular o valor do arco-cosseno de um número, mas podemos criá-la, definindo-a a partir das outras primitivas funcionais já existentes. Vejamos como fazer isso.

Existe uma primitiva no LOGO que se chama **OUTPUT**. O comando OUTPUT permite que o usuário defina suas próprias funções. Seu funcionamento é parecido com o comando STOP, pois ele interrompe a execução de um procedimento, só que além disso ele também envia algum objeto para fora do procedimento que foi parado, passando-o para ser usado em algum outro procedimento. Analise a seguinte linha de comando do LOGO :

```
PRINT SIN 30
0.5
```

SIN é uma primitiva que pega o valor 30 dado como entrada, calcula o valor do seno do ângulo de 30 graus (que vale 0.5) e envia este resultado para ser usado pelo comando de impressão PRINT. Na verdade a primitiva SIN tem um OUTPUT interno. Seguindo este raciocínio podemos agora definir nossas próprias funções.

Vejamos um primeiro exemplo, bastante simples. Vamos calcular o módulo de um número através da função MODULO definida da seguinte maneira : $|X| = \sqrt{x^2}$.

Como fica a função :

```
TO MODULO :X
OUTPUT SQRT ( :X*:X)
END
```

Teste, digitando

```
PR MODULO (30-40)
10
```

Note que a característica básica de uma função é que ela sempre precisa de uma outra primitiva atuando em conjunto, como os comandos PRINT, MAKE, etc... Se você tentar executá-la como os demais procedimentos vai notar que ocorre um erro.

Note que a estrutura de uma função é absolutamente a mesma dos demais procedimentos. Ela precisa do delimitador TO que indica o início da definição de um procedimento; ela tem um título que é constituído pelo nome da função e pelos argumentos de entrada e tem o delimitador END, que indica o final da definição da função. A única diferença está mesmo é no comando OUTPUT.

Uma vez definida uma função, ela pode ser automaticamente usada dentro de outras funções. Veja o seguinte exemplo, que serve para calcular a função SINAL:

$$SINAL(X) = \begin{cases} 1 & \text{se } X>0 \\ 0 & \text{se } X=0 \\ -1 & \text{se } X<0 \end{cases}$$

```
TO SINAL :X
IF :X=0 [OUTPUT 0][OUTPUT :X/MODULO :X]
END
```

Aqui foi preciso usar o condicional IF pois se o valor de :X for zero não podemos fazer a divisão :X / MODULO :X pois MODULO 0 dá zero.

Ex. :

```
PR SINAL 0
0

PR SINAL PI
1

PR SINAL 1/2-COS 360
-1
```

Obs. : OUTPUT pode ser abreviado por **OP**.

OUTPUT serve para passar qualquer tipo de objeto, seja ele um número, uma palavra ou uma lista, para um outro procedimento. Desta forma você pode criar não só funções matemáticas, mas também funções que operam sobre palavras e listas. Use a sua criatividade.

Até agora só trabalhamos com funções cuja expressão é simples. Existem situações onde a expressão que define a função é tão complicada ou depende de certos parâmetros de entrada e daí o comando OUTPUT não aceita a fórmula da função. Aí o jeito é partí-la em expressões menores e enviar somente o resultado no final. Veja o exemplo abaixo, que serve para calcular a norma de um vetor do R^n.

```
TO NORMA :VETOR
(LOCAL "Q "I)
MAKE "Q 0
MAKE "I 0
REPEAT COUNT :VETOR [MAKE "I :I+1 MAKE
"Q (:Q + (ITEM :I :VETOR)*(ITEM :I :VE
TOR))]
MAKE "Q SQRT :Q
OUTPUT :Q
END
```

Neste programa o vetor é dado por uma lista e a sua dimensão é determinada pelo tamanho da lista. Teste com alguns vetores.

```
PR NORMA [1 2 3]
3.741657387

PR NORMA [1 1]
1.414213562

PR NORMA [1 -1 0 1 1]
2

PR NORMA [-10]
10
```

A seguir temos uma listagem com programas que servem para calcular algumas das principais funções matemáticas que não existem no LOGO:

```
TO ABS :X
IF (:X<0) [OUTPUT -:X][OUTPUT :X]
END

TO ARCCOS :X
IF ((ABS :X)<1) [OP (90-ARCTAN (:X/SQRT(1-:X*:X)))]
IF (:X=1)  [OUTPUT 0]
IF (:X=-1) [OUTPUT 180]
OP [ARGUMENTO FORA DO DOMINIO DA FUNCAO]
END

TO ARCCOT :X
OUTPUT (90-ARCTAN :X)
END

TO ARCSIN :X
IF ((ABS :X)<1) [OP ARCTAN (:X/SQRT(1-:X*:X))]
IF (:X=1)  [OUTPUT 90]
IF (:X=-1) [OUTPUT -90]
OP [ARGUMENTO FORA DO DOMINIO DA FUNCAO]
END

TO DIV :A :B
IF (:B=0) [OP [DIVISAO POR ZERO]]
OUTPUT INT(:A/:B)
END

TO FAT :X
TEST (:X>0)
IFFALSE [OP 1]
IFTRUE  [OP (:X*FAT(:X-1))]
END

TO FRAC :X
OUTPUT (:X-INT :X)
END

TO LOG :X
IF (:X>0) [OUTPUT (LN :X)/LN 10 ]
OP [ARGUMENTO FORA DO DOMINIO DA FUNCAO]
END

TO SGN :X
IF (:X<0) [OP -1][OP 1]
END

TO TAN :X
IF (COS :X)=0 [OP [ARGUMENTO FORA DO DOMINIO DA
FUNCAO]]
OUTPUT (SIN :X)/COS :X
END
```

CAPITULO 9

ARQUIVOS

- ALLOPEN
- CLOSE
- CLOSEALL
- DIR
- DISK
- EDITFILE
- ERASEFILE
- FILELEN
- FILEP
- LOAD
- LOADPIC
- OPEN
- POFILE
- READEOFP
- READER
- SAVE
- SAVEPIC
- SETDISK
- SETREAD
- SETWRITE
- STARTUP
- WRITEOFP
- WRITER

ARQUIVOS

Neste capítulo veremos quais são os comandos que operam sobre arquivos em disco magnético, permitindo a gravação e o carregamento de programas, gráficos e fichários de dados. Veremos também como se pode criar arquivos auto-executáveis nesta linguagem, além de aprendermos à editar arquivos em modo texto (ASCII) usando o editor de programas do LOGO.

Antes de começarmos a trabalhar com tais recursos convém esclarecer alguns conceitos preliminares.

Um arquivo é qualquer conjunto de dados gravados em um meio magnético (disquete, fita, winchester, etc...). Estes dados podem ser programas, textos, fichários, gráficos, bibliotecas, etc...
No LOGO os arquivos são de três tipos: programas, pictures e linguagem de máquina (binários). Os arquivos de programas podem ser normais ou autoexecutáveis. Um arquivo é dito "auto-executável" quando, logo após o término de seu carregamento, ele é colocado em funcionamento automaticamente, sem ser necessária a intervenção do programador.

Todo arquivo é identificado por um nome (com no máximo 8 caracteres) e por uma extensão (de 3 caracteres) que determina o tipo de dados gravados em tal arquivo. No LOGO a extensão .LF indica programas em modo texto e a extensão .PIC indica gráficos.

Veremos a seguir quais os comandos que operam sobre arquivos, qual sua sintaxe e em que situações devem ser usados.

* GRAVAÇÃO E CARREGAMENTO DE PROGRAMAS:

- SAVE "NOMEARQUIVO

O comando SAVE grava o conteúdo da memória do micro no arquivo cujo nome é dado como entrada. Com este comando são gravados tanto os procedimentos quanto as variáveis globais que foram criados ou carregados durante uma sessão de trabalho. Para evitar gravar repetidamente programas desnecessários, apague-os antes de iniciar o processo de gravação, deixando somente os procedimentos e variáveis que você precisa mesmo gravar. Para maiores detalhes leia também os capítulos 2 e 3.

Ex. : SAVE "MEUARQ

- LOAD "NOMEARQUIVO

O comando LOAD carrega para a memória do micro o conteúdo do arquivo cujo nome é dado como entrada. Se um determinado programa que está na memória do micro já existir gravado no disquete com o mesmo nome, então a sua definição será substituída pela que está sendo lida do disquete.

Ex. : LOAD "MEUARQ

* GRAVAÇÃO E CARREGAMENTO DE GRAFICOS:

- SAVEPIC "NOMEDESENHO

O comando SAVEPIC grava a tela de vídeo no arquivo de gráficos cujo nome é dado como entrada. Se você quer gravar somente os desenhos (sem ter a respectiva janela de textos gravada junto), coloque o vídeo em modo gráfico através do comando FULLSCREEN (tecla F4) antes de iniciar o processo de gravação.

Obs. : Gravar uma tela gráfica ocupa muito espaço do disco. Procure gravar somente os desenhos que são muito complicados ou demorados para serem feitos.

Ex. : SAVEPIC "MEUDESEN

- LOADPIC "NOMEDESENHO

O comando LOADPIC carrega o conteúdo de um arquivo gráfico cujo nome é dado como entrada, fazendo-o aparecer na tela do vídeo, conforme havia sido gravado. Todos os desenhos que estão atualmente na tela gráfica são apagados, sendo substituídos pelos que estão sendo lidos.

Ex. : LOADPIC "MEUDESEN

* MUDANDO DE "DRIVE":

- SETDISK "DRIVE:

O comando SETDISK muda o "drive", passando o controle para o acionador de discos especificado pela entrada.

Ex. :
```
SETDISK "A:

SETDISK "C:
DIR
```

- DISK

O comando DISK fornece o nome do "drive" em uso atualmente.

Ex. :
```
PR DISK
C:
```

* VERIFICANDO O CONTEÚDO DE UM DISCO OU ARQUIVO:

- DIR

O comando DIR fornece uma listagem contendo o nome de todos os arquivos existentes no disquete ou "winchester".

Ex. :
```
DIR
```

- POFILE "NOMEARQUIVO.EXTENSÃO

O comando POFILE permite verificar o conteúdo de um arquivo do disco magnético. Somente são impressos os arquivos em ASCII.

Obs. : O comando POFILE necessita que se especifique a extensão que determina o tipo do arquivo.

Ex. :
```
POFILE "PROGRAMA.LF
```

- FILEP "NOMEARQUIVO.EXTENSÃO

O comando FILEP permite verificar a existência ou não de um determinado arquivo no disco magnético, retornando um valor booleano. O resultado é "TRUE se o arquivo cujo nome foi especificado na entrada existe no disco e "FALSE em caso contrário.

Obs. : É preciso sempre especificar a extensão do tipo do arquivo.

Ex. :
```
PRINT FILEP "LOGO.COM
TRUE
```

* EDITANDO OU APAGANDO ARQUIVOS:

- EDITFILE "NOMEARQUIVO.EXTENSÃO
- EDITFILE "ARQUIVOANTIGO.EXTENSÃO "ARQUIVONOVO.EXTENSÃO

O comando EDITFILE permite que se edite um arquivo em ASCII diretamente no editor de programas do LOGO, sem que ele seja carregado para a memória do micro. Terminada a edição, acione a tecla de escape ESC para que a nova versão do arquivo seja automaticamente gravada no disco magnético, no lugar da antiga.

Obs. : O comando EDITFILE também pode ser usado para se mudar o nome de um arquivo. Para tal copie o arquivo antigo, dando-lhe um novo nome (use o comando no segundo formato acima). Feito isso, apague o arquivo antigo com o comando ERASEFILE.

- ERASEFILE "NOMEARQUIVO.EXTENSÃO

O comando ERASEFILE elimina do disco magnético o arquivo cujo nome é dado como entrada, permitindo que o espaço do arquivo que foi apagado do disco possa ser usado de novo na gravação de um outro arquivo.

Ex. :
```
ERASEFILE "MEUAQR.LF
```

* LENDO E ESCREVENDO DADOS NUM ARQUIVO TIPO FICHARIO:

Para que possamos operar com dados que estão num arquivo tipo fichário precisamos seguir as seguintes etapas:

1) Abrir o arquivo com o comando OPEN ;

2) Habilitar a leitura ou escrita com os comandos SETREAD ou SETWRITE ;

3) Testar o final do arquivo com os comandos READEOFP ou WRITEOFP ;

4) Lêr ou escrever os dados no arquivo com as primitivas de INPUT/OUTPUT (ver maiores detalhes no capítulo 7) ;

5) Terminada a operação de leitura/escrita, fechar o arquivo com o comando CLOSE ;

Vejamos a sintaxe destes comandos:

- OPEN "NOMEARQUIVO.EXTENSÃO
- OPEN "NOMEARQUIVO

O comando OPEN abre o arquivo texto cujo nome foi especificado na entrada para que possamos efetuar operações de leitura ou escrita sobre o mesmo.

Ex. : OPEN "MEUARQ

- SETREAD "NOMEARQUIVO.EXTENSÃO
- SETREAD "NOMEARQUIVO

O comando SETREAD habilita a operação de leitura de dados a partir do arquivo cujo nome foi especificado na entrada. Deste momento em diante, todas as primitivas de INPUT de dados (READCHAR, READWORD, READLIST, etc...) lerão somente as informações contidas neste arquivo.

EX. : SETREAD "MEUARQ

- SETWRITE "NOMEARQUIVO.EXTENSÃO
- SETWRITE "NOMEARQUIVO

O comando SETWRITE habilita a operação de escrita de dados num arquivo. A partir deste momento todas as primitivas de OUTPUT de dados (PRINT, TYPE, SHOW, etc...) passarão a escrever diretamente no arquivo cujo nome foi especificado na entrada. Se o arquivo já existe, os novos dados serão inseridos seqüencialmente no final do arquivo.

Ex. : SETWRITE "DADOS.TXT

- CLOSE "NOMEARQUIVO.EXTENSÃO
- CLOSE "NOMEARQUIVO
- CLOSEALL

O comando CLOSE fecha o arquivo cujo nome é dado pela entrada. O comando CLOSEALL fecha todos os arquivos que estiverem abertos.

Ex. : CLOSE "AGENDA

CLOSEALL

*** DEMAIS COMANDOS QUE OPERAM SOBRE ARQUIVOS:**

- ALLOPEN

:

O comando ALLOPEN fornece uma lista que contém o nome de todos os arquivos que estão atualmente abertos, seja para leitura ou escrita.

Ex. : PRINT ALLOPEN

- FILELEN "NOMEARQUIVO.EXTENSÃO
- FILELEN "NOMEARQUIVO

O comando FILELEN fornece o tamanho (em Bytes) do arquivo cujo nome é dado como entrada.

Obs. : Para que possamos obter o tamanho do arquivo é necessário que ele esteja aberto, senão ocasionará uma mensagem de erro.

Ex. :
```
OPEN "MEUARQ.LF
PRINT FILELEN "MEUARQ.LF
CLOSE "MEUARQ.LF
```

- READER

O comando READER fornece uma lista que contém o nome de todos os arquivos que estão abertos e ativados para leitura de dados.

Ex. :
```
PRINT READER
```

- WRITER

O comando WRITER fornece uma lista que contém o nome de todos os arquivos que estão abertos e ativados para a escrita de dados.

Ex. :
```
PRINT WRITER
```

- READEOFP

O comando READEOFP verifica se foi encontrado ou não o final do arquivo que está aberto para leitura de dados, retornando um valor booleano "TRUE ou "FALSE. Se a resposta for "TRUE significa que não há mais dados para serem lidos.

Ex. :
```
OPEN "MEUARQ.LF
SETREAD "MEUARQ.LF
PRINT READEOFP
FALSE
CLOSEALL
```

- WRITEOFP

O comando WRITEOFP verifica se foi encontrado ou não o final do arquivo que está aberto para a escrita de dados, retornando um valor booleano "TRUE ou "FALSE. Se a resposta for "TRUE significa que não há mais espaço disponível neste disco para se efetuar uma nova operação de escrita de dados, restando só fechar o arquivo.

Ex. :

```
OPEN "MEUARQ.LF
SETWRITE "MEUARQ.LF
TEST WRITEOFP
IFTRUE [PRINT [NOVOS DADOS]]
IFFALSE [CLOSE "MEUARQ.LF]
```

Os programas a seguir ilustram como se pode usar um arquivo em disco para armazenar informações, como elas são escritas e depois recuperadas.

O programa ESCREVENOARQUIVO sorteia uma seqüência de números aleatórios que vai sendo escrita no arquivo de nome "NUMEROS. O programa LEARQUIVO é quem fará a respectiva leitura depois.

Analise-os atentamente.

```
TO ESCREVENOARQUIVO
OPEN "NUMEROS
SETWRITE "NUMEROS
REPEAT (20+RANDOM 30) [PRINT RANDOM 100]
CLOSE "NUMEROS
END
```

```
TO LEARQUIVO :NOMEARQUIVO
TEST FILEP :NOMEARQUIVO
IFFALSE [STOP]
MAKE "LD []
OPEN :NOMEARQUIVO
SETREAD :NOMEARQUIVO
LABEL "REPETE
TEST READEOFP
IFTRUE [CLOSE :NOMEARQUIVO PRINT :LD STOP]
MAKE "LD LPUT READWORD :LD
GO "REPETE
END
```

Obs. : Execute o segundo programa depois do primeiro, digitando LEARQUIVO "NUMEROS e veja o resultado.

* ARQUIVOS AUTO-EXECUTAVEIS:

- STARTUP [LISTA DE PROGRAMAS A SEREM EXECUTADOS]

STARTUP é uma variável do sistema e deve conter a lista com o nome de todos os programas que deverão ser executados automaticamente, após o término da leitura do arquivo que os contém. Esta variável global é criada através do comando MAKE e deve ser gravada com o comando SAVE, junto com os programas que deverão ser auto-executáveis. O seu funcionamento é ativado pelo comando de leitura LOAD.

Obs. : Este comando STARTUP é útil quando queremos que um programa funcione sozinho, sem a intervenção do usuário.

Vejamos um exemplo:

```
TO PROGRAMA1
CS TEXTSCREEN CT
PR [ESTE PROGRAMA E AUTO-EXECUTAVEL]
PR [PELO COMANDO "STARTUP"]
PR [] PR [TAO LOGO SEJA LIDO]
PR [ELE SERA EXECUTADO]
PR [**********] PR []
END

TO PROGRAMA2
PR [ESTE PROGRAMA NAO E AUTO-EXECUTAVEL]
PR [++++++++++]
END

MAKE "STARTUP [PROGRAMA1]

SAVE "ARQUIV01

POALL

ERALL

POALL

LOAD "ARQUIV01

ESTE PROGRAMA E AUTO - EXECUTAVEL
PELO COMANDO "STARTUP"

TAO LOGO SEJA LIDO
ELE SERA EXECUTADO
* * * * * * * * * * * * * * * * * * * *

?
```

CAPITULO 10

COMANDOS DE FORMATAÇÃO

- Formatação de Texto :

  - CURSOR
  - SETCURSOR
  - SETTEXT
  - SETWIDTH
  - WIDTH

- Formatação Gráfica :

  - .SCRUNCH
  - .SETSCRUNCH
  - FENCE
  - WINDOW
  - WRAP

- Formatação de Dados :

  - EFORM
  - FORM

COMANDOS DE FORMATAÇÃO

Existem dois tipos de comandos de formatação. Os comandos de formatação de dados e os comandos de formatação do vídeo.

Os comandos de formatação do vídeo permitem alterar os atributos das telas de texto e gráfica. Entende-se por atributos de uma tela as suas características básicas, como por exemplo o número de caracteres que se pode imprimir em cada linha, no modo texto; o número de linhas que se tem na janela de texto, quando no modo misto (texto e gráficos); a proporção entre os eixos, nos deslocamentos verticais e horizontais da tartaruga, quando no modo gráfico; o formato da tela gráfica (se é plana ou "toro"); etc...

Os comandos de formatação de dados permitem determinar o formato da impressão dos dados numéricos, se em notação científica ou de ponto fixo; número de dígitos após o ponto; etc...

São os seguintes os comandos que permitem alterar ou confirmar os atributos da tela texto:

- CURSOR

CURSOR fornece a posição atual do cursor, na janela de texto. A saída é uma lista que contém dois números; o primeiro indica a coluna em que o cursor se encontra e o segundo indica a respectiva linha.

```
Ex. :                CT  PR CURSOR
                     0 0
```

- SETCURSOR LIST :C :L

SETCURSOR coloca o cursor da tela texto na posição indicada pela lista de coordenadas :C e :L, onde :C é o número da coluna e :L é o número da linha. O valor de :C pode variar de 0 até o valor fixado pelo comando WIDTH e o valor de :L pode variar de 0 até o valor especificado pelo número de linhas da tela mista, pelo comando SETTEXT.

Obs. : O comando LIST :C :L pode ser substituído pela lista [C L] .

```
Ex. :                CT  SETCURSOR LIST 3 5  PR PI
                     CT  SETCURSOR [3 5]  PR PI
```

- SETTEXT :N

SETTEXT fixa em :N o número de linhas da janela de texto, quando no modo misto. O valor para :N deve ser um inteiro entre 1 e 20.

Ex. :
```
SETTEXT 10  MS
```

- SETWIDTH :N

SETWIDTH especifica o tamanho de cada linha no modo texto, ou seja, determina que sejam :N o número máximo de caracters que podem ser impressos numa mesma linha do vídeo. O valor de :N pode ser qualquer inteiro entre 2 e 80, mas a partir de 41 não se pode ter mais o modo misto, com janelas compartilhadas, só o modo texto. Desta forma, antes de colocar SETWIDTH 41, certifique-se de que o micro estará no modo texto digitando CT CS TS e teclando "return".

Ex. :
```
SETWIDTH 10
REPEAT 50 [TYPE "\*]
PR []
```

- WIDTH

WIDTH fornece a largura atual da tela texto, que pode ser de 2 a 80 caracteres por linha.

Ex. :
```
PR WIDTH
10
```

São os seguintes comandos que permitem alterar ou confirmar os atributos da tela gráfica:

- .SCRUNCH

O comando .SCRUNCH fornece a proporção atual entre os eixos X e Y, onde o número resultante é o quociente entre o passo da tartaruga no eixo vertical e o passo no eixo horizontal.

Ex. :

```
PR .SCRUNCH
0.8

TO ASPECTO
CS CT PR .SCRUNCH
MAKE "E 0
REPEAT 6 [MAKE "E :E+.25 .SETSCRUNCH :E
CS CT TYPE "Escala\: PR .SCRUNCH REPEAT
360 [FD 1 RT 1] WAIT 50]
END
```

- .SETSCRUNCH :X

O comando .SETSCRUNCH muda a proporção entre os eixos X e Y na tela gráfica, onde :X é o valor do quociente entre o passo da tartaruga no eixo Y pelo passo no eixo X, ou seja, $\Delta y = :x * \Delta x$. O valor de :X pode ser qualquer número real positivo.

Obs. : Para que o desenho saia em tamanho real na impressora, quando usamos as teclas **SHIFT + PRINT SCREEN**, use a proporção 1.2 em .SETSCRUNCH.

Ex. :

```
TO QUADRADOREAL
.SETSCRUNCH 1.2
CS REPEAT 4 [FD 50 RT 90]
END
```

- FENCE

FENCE coloca a tela gráfica no modo "plano", de tal forma que quando a tartaruga executar um movimento que saia fora da tela por um lado ela não entre pelo outro. Neste caso a atuação da tartaruga fica restrita ao retângulo determinado pelos eixos de coordenadas do vídeo, admitindo valores entre -159 e 160 para as coordenadas no eixo X e entre -124 e 125 para as coordenadas no eixo Y. Obs. : Colocar a tartaruga em qualquer outro ponto fora deste retângulo determina condição de erro e a execução é suspensa, com o aparecimento da respectiva mensagem de erro.

Ex. :

```
TO CIRCULOF
FENCE
CS REPEAT 360 [FD 5 RT 1]
END
```

- WINDOW

O comando WINDOW coloca a tela gráfica no modo "plano" como o comando FENCE, mas os limites de atuação da tartaruga são ampliados. O valor máximo que as coordenadas X e Y podem assumir aumenta, passando a estarem entre -32768 e 32768. Note que a janela visível é a mesma do comando FENCE, porém quando a tartaruga for movimentada para um ponto que não esteja dentro desta janela não irá ocorrer uma mensagem de erro pois com este comando ela vai poder ser movimentada.

Ex. :

```
TO SEMICIRCULO
WINDOW
CS REPEAT 360 [FD 5 RT 1]
END
```

- WRAP

O comando WRAP coloca a tela gráfica no modo "toroidal". Quando a tartaruga sair por um lado da tela ela automaticamente reaparecerá pelo lado oposto, como se estivesse caminhando sobre um toro. Com a tela neste modo, não existem limites para os valores das coordenadas X e Y. Este é o modo "default" da tela gráfica.

Ex. :

```
WRAP
CS RT 15 FD 5000
CS DOT [4000 3800]

TO CIRCULOWRAP
WRAP
CS REPEAT 360 [FD 5 RT 1]
END
```

COMANDOS DE FORMATAÇÃO DE DADOS NUMÉRICOS

São dois os comandos de formatação de dados numericos. Eles permitem ao usuário escolher o formato em que os dados serão apresentados na tela texto: em ponto flutuante (notação científica) ou em ponto fixo, com especificação do número de dígitos na parte inteira e fracionária. São eles:

- EFORM :X :N

EFORM faz com que o valor do número em :X seja impresso em notação científica, com :N algarismos na mantissa. O valor de :N pode ser qualquer número inteiro entre 1 e 1000.

Ex. :

```
PRINT EFORM PI 3
 3.14E+0000
PR EFORM POWER 2 10 20
 1.20400000000000000E+0003
```

- FORM :X :I :F

FORM faz com que o valor do número em :X seja impresso em notação de ponto fixo, com :I dígitos na parte inteira e :F digítos na parte fracionária. Se :F não é especificado, não são impressos os dígitos depois da vírgula, mas o arredondamento clássico sempre é feito.

Ex. :

```
PR FORM PI 3 5
 3.14159

PR FORM POWER 2 10 3 2
1024.00

PR FORM 258/7 5 5
  36.85714

PR FORM 258/7 5 0
  37
```

:

CAPITULO 11

USANDO A IMPRESSORA

- DRIBBLE
- NODRIBBLE

PROCEDIMENTOS DE IMPRESSÃO

Os comandos de impressão permitem que se possa visualizar na tela do micro a definição de procedimentos, o conteúdo das variáveis ou a lista de títulos de todos os procedimentos existentes na memória do micro até o momento. Podemos fazer com que a impressão seja feita simultaneamente na tela e na impressora.

Para que a listagem saia também na impressora, proceda da seguinte maneira :

- ligue a impressora e certifique-se de que o multiplexor (chaveador da impressora) esteja na posição correta;
- ajuste o papél (se necessário);
- digite **DRIBBLE "LPT1** e tecle "ENTER".

Agora a impressora já está pronta para ser usada.

Digite **DIR** e tecle "ENTER" para ver o que acontece.

O comando **DRIBBLE "LPT1** abre um canal da saída simultânea dos caracteres do vídeo para a impressora. A partir deste momento tudo que for digitado pelo teclado e aparecer no vídeo também sairá listado na impressora. Este recurso permite que se faça um histórico de todos os comandos do LOGO que foram digitados durante uma sessão de trabalho. Caso você queira que os comandos sejam gravados em seu disquete ao invés de sairem na impressora, troque a palavra **"LPT1** pelo comando **DRIBBLE "NOMEARQUIVO**, onde "NOMEARQUIVO é uma palavra que indica o nome do arquivo onde os dados devem ser gravados.

Para desativar a impressora ou fechar o arquivo do disco digite **NODRIBBLE** .

Antes de sair do LOGO e desligar o micro, sempre que usar a impressora com o **DRIBBLE "LPT1** use o **NODRIBBLE** , pois se você sair do LOGO sem desativar a impressora poderá causar danos irreparáveis ao seu disquete (note que a luz indicadora do drive acende quando usamos o comando **DRIBBLE "LPT1**).

Ex. :

```
TO USANDO_A_IMPRESSORA
PR [ATIVANDO A IMPRESSORA]
DRIBBLE "LPT1
PR [ESTE PROGRAMA VAI SER IMPRESSO]
PO "USANDO_A_IMPRESSORA
PR [DESATIVANDO A IMPRESSORA]
NODRIBBLE
END
```

CAPITULO 12

MISCELÂNEA

---

* COMANDOS ESPECIAIS :

- .CONTENTS
- .DOS
- NODES
- RECYCLE
- REPARSE
- ERRACT
- ERROR

* CARACTERES ESPECIAIS :

( ) * + - / < = > \ " [ ] : .

* SONS :

- TONE

* DEMAIS COMANDOS :

- HIDETURTLE (HT)
- SHOWTURTLE (ST)
- FULLSCREEN (FS)
- MIXEDSCREEN (MS)
- TEXTSCREEN (TS)

COMANDOS ESPECIAIS

- .CONTENTS

O comando .CONTENTS fornece como saída uma lista que contém todas as primitivas da linguagem LOGO e também o nome de todos os procedimentos e variáveis globais criados pelo usuário.

Ex. : PR .CONTENTS

- .DOS

O comando .DOS encerra uma sessão de trabalho, saindo do LOGO e devolvendo o controle ao sistema operacional. Todos os procedimentos e variáveis serão perdidos, se não forem previamente gravados, pois a memória é automaticamente apagada.

Ex. : .DOS

- NODES

O comando NODES dá como saída o número de bytes livres na área de trabalho, ou seja, fornece a quantidade de memória que ainda está disponível para o usuário.

```
Ex . :          RECYCLE  PR NODES
                31093
```

- RECYCLE

O comando RECYCLE efetua uma limpeza da área de trabalho, reorganizando-a e liberando tantos bytes quanto é possível. O LOGO realiza automaticamente esta limpeza a cada certo tempo, porém é conveniente fazê-lo manualmente antes de executar procedimentos longos para evitar perda de tempo durante o processamento.

Ex. : RECYCLE

- REPARSE

Quando se apaga um procedimento, o espaço de memória que fica disponível pode ser ocupado por outro procedimento. O comando REPARSE serve para reajustar os procedimentos restantes, reocupando o espaço que ficou disponível.

Ex. : REPARSE

- ERRACT

ERRACT é uma variável do sistema, que pode assumir somente as palavras "TRUE ou "FALSE . Ela serve para habilitar ou não o LOGO a fazer uma pausa se ocorrer algum erro durante o processamento. O valor "default" para ERRACT é "FALSE.

Ex. :

```
MAKE "ERRACT "TRUE

MAKE "ERRACT "FALSE
```

- ERROR

O comando ERROR fornece como saída uma lista que contém seis elementos e que descrevem o erro que ocorreu mais recentemente.

Os seis elementos da lista são :

1) o número que identifica o erro;

2) a mensagem explicando o erro;

3) o nome do procedimento onde ocorreu o erro;

4) a linha que contém o erro;

5) o nome da primitiva que causou o erro;

6) o nome do objeto que causou o erro.

Ex. :

```
ERROR
```

:

CARACTERES ESPECIAIS

São os seguintes os caracteres especiais do LOGO:

( ) * + - / < = > \ " [ ] : .

*

Faz o produto de duas ou mais expressões numéricas.

```
Ex. :                PR 3*2
                     6

                     PR 3*2*4*5
                     120

                     MAKE "X 5  PR :X*ARCTAN :X
                     393.4503377
```

+

Faz a soma de duas ou mais expressões numéricas.

```
Ex. :                PR 3+2
                     5

                     PR 3+2+4+5
                     14

                     MAKE "X 5  PR :X+ARCTAN :X
                     83.69006753
```

-

Faz a diferença entre duas expressões numéricas e serve também como o simétrico de um número.

```
Ex. :                PR 3-2
                     1

                     PR -5
                     -5
```

/

Faz a divisão entre duas expressões numéricas.

```
Ex. :                PR 15/8
                     1.875

                     PR SIN 30/COS 30
                     0.5684328556
```

( )

Os parêntesis são usados para alterar a ordem de avaliação de uma expressão, dando-lhe maior prioridade, ou para que possamos acrescentar mais argumentos em algumas primitivas.

Ex. :

```
PR 5+3*2-6
5

PR (5+3)*(2-6)
-32

PR (5+3)*2-6
10

PR 5+3*(2-6)
-7

PR SUM (1+2+3+4) 5
15

PR (SUM 1 2 3 4 5 6)
21

MAKE "5! (PRODUCT 1 2 3 4 5)  PR :5!
120

PR (WORD "ESTA "E "UMA "PALAVRA)
ESTAEUMAPALAVRA

PR (SE [ESTA] [E] [UMA] [FRASE])
ESTA E UMA FRASE
```

.

O ponto decimal é usado ao invés da vírgula, para separar a parte inteira da parte fracionária dos números.

Ex. :

```
PR PI
3.141592654

MAKE "X 34.56  PR :X
34.56
```

:

Os dois pontos servem para indicar que o objeto que se segue é uma variável, enviando automaticamente o seu conteúdo.

Ex. :

```
TO VARIAVEL :X
PR :X
END
```

"

As aspas servem para indicar que o objeto que se segue é uma palavra e não um procedimento. É usada para dar nome às variáveis globais criadas com os comandos MAKE ou NAME.

Ex. :
```
PR "PALAVRA
PALAVRA

MAKE "VARIAVEL  RANDOM 100
PR :VARIAVEL
55
```

[ ]

Os colchetes servem para identificar listas, delimitando o seu conteúdo.

Ex. :
```
SHOW [ISTO E UMA LISTA]
[ISTO E UMA LISTA]
```

<

Operador lógico-aritmético que é usado em expressões condicionais. Verifica a condição de uma expressão numérica ser menor do que a que se segue.

Ex. :
```
IF 2<3 [PR [OK!]]
OK!
```

>

Operador lógico-aritmético usado em expressões condicionais. Verifica se uma determinada expressão numérica é maior ou não que a expressão que a segue.

Ex. :
```
IF 2>3 [PR [2 E MENOR DO QUE 3]][PR [2
NAO E MAIOR DO QUE 3]]
2 NAO E MAIOR DO QUE 3
```

=

Operador lógico que verifica a condição de igualdade entre dois objetos, retornando sempre um valor booleano "TRUE ou "FALSE

Ex. :
```
PR 2=2
TRUE

PR 2=3
FALSE

PR "2=[2]
FALSE
```

\

A barra invertida serve como delimitador para que os demais caracteres especiais sejam interpretados como caracteres e não como operadores ou delimitadores. Sempre que queremos que um caractere especial seja impresso na tela ele deve ser precedido pela barra invertida \ .

Ex. :

```
PR [A\*B]
A*B

PR 3+2
5

PR "3\+2
3+2

PR "\\\ \-\ \\
\ - \
```

SONS

- TONE :frequência :duração

O comando TONE serve para gerar um sinal sonoro numa determinada freqüência e com uma determinada duração.

Ex. :

```
TONE 800 10

TO PLAY
TONE 800 10
TONE 1000 10
TONE 800 10
WAIT 50
TONE 800 10
TONE 600 10
TONE 800 10
END
```

DEMAIS COMANDOS

- HIDETURTLE

O comando HIDETURTLE torna a tartaruga invisível.

Obs. : O comando HIDETURTLE pode ser abreviado por HT.

Ex. : HIDETURTLE

Nota: O programa roda mais rápido se a tartaruga está invisível.

- SHOWTURTLE

O comando SHOWTURTLE serve para tornar a tartaruga visível.

Obs. : O comando SHOWTURTLE pode ser abreviado por ST.

Ex. : SHOWTURTLE

- FULLSCREEN

O comando FULLSCREEN serve para colocar o vídeo em modo gráfico, fazendo com que a janela de textos desapareça, porém o texto não é apagado e pode ser recuperado mediante o uso do comando MIXEDSCREEN.

Obs. : O comando FULLSCREEN pode ser abreviado por FS.

Ex. : FULLSCREEN

Nota: Você pode obter o mesmo efeito acionando a tecla F4.

- MIXEDSCREEN

O comando MIXEDSCREEN põe o vídeo em modo misto, reservando a parte superior da tela para gráficos e a parte inferior para a janela de textos.

Obs. : O comando MIXEDSCREEN pode ser abreviado por MS.

Ex. : MIXEDSCREEN

Nota: Você pode obter o mesmo efeito acionando a tecla F2.

- TEXTSCREEN

O comando TEXTSCREEN põe o vídeo em modo texto, fazendo com que a janela de gráficos desapareça, porém os desenhos não são apagados e podem ser recuperados usando-se o comando MIXEDSCREEN.

Obs. : O comando TEXTSCREEN pode ser abreviado por TS.

Ex. : TEXTSCREEN

Nota: Você pode obter o mesmo efeito acionando a tecla F1.

BIBLIOGRAFIA

BIBLIOGRAFIA

ABELSON,Harold. Apple LOGO. Massachusetts: BYTE, 1982. 222p.

ABELSON,Harold. Turtle Geometry. Cambridge: MIT Press, 1980. 477p.

BERGER, Anne R., CARTER, Richard C., HARRIS, Ross J. et al. PCLOGO: Tutorial. Somerville: Harvard, 1983.

HANNA, Stephen, ING, Carolin K. H., MERTENS, Peter von. PCLOGO: Reference Manual. Somerville: Harward, 1983.

RODRIGUEZ-ROSELLÓ, Luis. LOGO: de la Tortuga a la Inteligencia Artificial. Madrid: Vector Ediciones, 1986. 589p.

INDICE REMISSIVO

ALLOPEN	72	EDITFILE	70
AND	38	EDNS	27
ARCTAN	57	EFORM	81
ASCII	44	EMPTYP	46
BACK,BK	4	END	21
BUTFIRST,BF	45	EQUALP	39
BUTLAST,BL	45	ERALL	21
BUTTONP	38	ERASE,ER	21
CATCH	36	ERASEFILE	70
CHAR	45	ERN	27
CLEAN	8	ERNS	28
CLEARSCREEN,CS	9	ERPS	22
CLEARTEXT,CT	15	ERRACT	86
CLOSE	72	ERROR	86
CLOSEALL	72	EXP	58
CO	34	FALSE	39
COPYDEF	19	FENCE	79
COS	57	FILELEN	73
COUNT	46	FILEP	70
CURSOR	77	FILL	10
DEFINE	20	FIRST	46
DEFINEDP	20	FORM	81
DIFFERENCE	57	FORWARD,FD	4
DIR	69	FPUT	47
DISK	69	FULLSCREEN,FS	91
DOT	9	GO	35
DRIBBLE	83	HEADING	11
EDIT,ED	21	HIDETURTLE,HT	91

HOME 5
IF 32
IFFALSE,IFF 32,33
IFTRUE,IFT 32,33
INT 58
ITEM 47
KEYP 40
LABEL 35
LAST 47
LEFT,LT 5
LIST 48
LISTP 48
LN 58
LOAD 68
LOADPIC 68
LOCAL 28
LPUT 48
MAKE 28
MEMBERP 49
MIXEDSCREEN,MS 91
NAME 28
NAMEP 29
NODES 85
NODRIBBLE 83
NOT 40
NUMBERP 49
OPEN 71
OR 41
OUTPUT,OP 35
PAUSE 34
PENDOWN,PD 11
PENERASE,PE 11
PENREVERSE,PX 11
PENUP,PU 12
PI 59
PO 22
POALL 22
POFILE 69
PONS 29
POPS 22
POS 12
POTS 22
POWER 59
PRECISION 59
PRIMITIVEP 41
PRINT,PR 52
PRODUCT 59
QUOTIENT 60
RANDOM 60
READCHAR,RC 53
READCHARS,RCS 53
READEOFP 73
READER 73
READLIST,RL 53,54
READWORD,RW 54
RECYCLE 85

REDEFP 23
REMAINDER 60
REPARSE 85
REPEAT 31
RERANDOM 61
RIGHT,RT 6
ROUND 61
RUN 33
SAVE 67
SAVEPIC 68
SENTENCE,SE 50
SETCURSOR 77
SETDISK 69
SETHEADING,SETH 6
SETPOS 7
SETPRECISION 62
SETREAD 71
SETSHAPE 14
SETTEXT 78
SETWIDTH 78
SETWRITE 72
SETX 7
SETY 8
SHAPE 15
SHOW 54
SHOWNP 42
SHOWTURTLE,ST 91
SIN 62
SQRT 62
STAMP 15
STARTUP 75
STOP 34
SUM 62
TEST 32
TEXT 23
TEXTSCREEN,TS 92
THING 29
THROW 36
TO 23
TONE 90
TOPLEVEL 36
TOWARDS 13
TRUE 42
TYPE 55
WAIT 33
WIDTH 78
WINDOW 80
WORD 50
WORDP 50
WRAP 80
WRITEOFP 74
WRITER 73
XCOR 13
YCOR 14
.CONTENTS 85
.DOS 85

.SCRUNCH 79

.SETSCRUNCH 79

( ) 88

* 87

+ 87

- 87

/ 87

< 89

> 89

= 89

. 88

: 88

" 89

[ ] 89

\ 90

:

TECLAS DE CONTROLE

ESC	Sai da janela de edição.
F1	Põe o vídeo em modo texto.
F2	Põe o vídeo em modo misto.
F3	Retorna a última linha digitada.
F4	Põe o vídeo em modo gráfico.
F5	Pausa por Hardware (permite alterar um programa e continuá-lo com CO).
DEL	Apaga o caractere sob o cursor.
BACK SPACE	Apaga o caractere imediatamente antes do cursor.
ALT ###	Gera um caractere especial, onde ### representa um número inteiro entre 0 e 255.
PAUSE	Faz uma pausa temporária no processamento (para continuar acione a barra de espaços).
TAB	Move o cursor para o fim da linha.
SHIFT + TAB	Move o cursor para o início da linha.
CTRL + PG UP	Vai para o início do texto.
CTRL + PG DN	Vai para o fim do texto.
CTRL + --->	Apaga a linha inteira.
CTRL + <---	Devolve o que foi apagado.
SHIFT + CTRL + --->	Apaga do cursor ao fim da linha.
SHIFT + CTRL + <---	Retorna o que foi apagado.
CTRL+ALT+DEL	Reseta o computador, reinicializando-o.

Publicações do Instituto de Matemática da UFRGS
Cadernos de Matemática e Estatística

Série B: Trabalho de Apoio Didático

1. Elsa Mundstock - Curso Básico Sobre Wordstar 3.45 - MAR/89

2. Jaime B. Ripoll - Introdução ao Cálculo Diferencial Via Funções de Uma Variável Real - OUT/89

3. Edmund R. Puczylowski - Dimension of Modular Lattices - JUN/90

4. Marcos Sebastiani - Geometrias Não Euclidianas - JUL/90

5. Sandra R. C. Pizzatto - Cálculo Numérico - AGO/91

6. Vera Clotilde G. Carneiro - Elementos de Cálculo para Biologia - AGO/91

7. Elsa Mundstock - Iniciação ao SPSS/PC - SET/91

8. Elisa Hagg, Loiva C. de Zeni, Maria Alice Gravina e Vera Carneiro - Notas da 1ª Oficina de Matemática da UFRGS - JAN/92

9. Paulo Werlang de Oliveira, Elisabete Rambo, Suzana Lima dos Santos, Coordenação: Profª Maria Alice Gravina - A Tartaruga no Espaço Tridimensional - FEV/92

10. Silvio Possoli - Análise Multivariada - JUL/92

11. Dinara Westphalen Fernandez - Números Índices - OUT/92

12. Maria Teresinha Albanese - Coeficiente de Fidedignidade de um Instrumento de Medida - OUT/92

13. Vera Clotilde Carneiro e Sérgio Cláudio Ramos - Gráficos na Escola - DEZ/92

14. João Riboldi - Elementos Básicos de Estatística - JAN/93

15. Paulo W. de Oliveira e M. Alice Gravina - Logo: Manual do Usuário - MAR/93

16. Ruben Markus, Elsa C. de Mundstock, Dinara W. X. Fernandez e João Riboldi - Exercícios de Métodos Estatísticos - AGO/93

17. Loiva C. de Zeni e M. Alice Gravina - Sugestões de Atividades no Ambiente Logo para a Exploração de Conteúdos Matemáticos dos Currículos Escolares de 1º e 2º Grau - SET/93

18. João Riboldi - Delineamentos Experimentais de Campo, Parte 1 - SET/93

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE MATEMÁTICA
NÚCLEO DE ATIVIDADES EXTRA CURRICULARES

Os Cadernos de Matemática e Estatística publicam as seguintes séries:

Série A: Trabalho de Pesquisa

Série B: Trabalho de Apoio Didático

Série C: Colóquio de Matemática SBM/UFRGS

Série D: Trabalho de Graduação

Série F: Trabalho de Divulgação

Série G: Textos para Discussão

Toda correspondência com solicitação de números publicados e demais informações deverá ser enviada para:

NAEC - NÚCLEO DE ATIVIDADES EXTRA CURRICULARES
INSTITUTO DE MATEMATICA - UFRGS
AV. BENTO GONÇALVES, 9500 - PRÉDIO 43111
CEP 91509 - 900 AGRONOMIA - POA/RS
FONE: 336 92 22 OU 339 13 55 OU 228 16 33
RAMAL 6197
FAX: 336 15 12