

BIBLIOTECA BÁSICA

INFORMÁTICA

LOGO 18

a tartaruga
inteligente



BIBLIOTECA BÁSICA

INFORMÁTICA

LOGO

18

a tartaruga
inteligente

Diretor editor:

M. A. Nieto

Coordenação e supervisão técnica:

Eng.º Sergio Rocha Paggioli

Tradução:

Ideli Novo

Projetos especiais:

Rainer K.E. Ladewig

Editor Gráfico:

Auro Pereira da Silva

Revisão:

Susana M. Amaral Couto

Produção Gráfica:

Luiz Carlos Siqueira Lago

Arte-Final:

Rubens Tadeu Benedito (coordenador)

Luiz Antonio de Andrade

Vadinho de Oliveira

Fotocomposição, fotolito:

Omnicolor Gráfica e Propaganda Ltda - Rua Dr. Virgílio de Carvalho Pinto, 619
Pinheiros - CEP 05415 - São Paulo.

Impressão:

Editora Antártica S.A. - Av. Ramon Freire, 6920 (Pajaritos) - Santiago - Chile

© Antonio M. Ferrer Abello

© Edições Ingelek S.A.

© 1986 para a língua portuguesa Ed. Século Ltda. - Rua Belisário Pena, 821
Penha - R. J. - Fone: 290-6273 - CEP 21020.

A editora Século Futuro mantém todos os direitos reservados sobre esta publicação. Fica proibido assim, sua reprodução total ou parcial por qualquer sistema sem prévia autorização do Diretor.

ÍNDICE

PREFÁCIO

05 Prefácio

CAPÍTULO I

07 A tartaruga, seu rastro e a resolução gráfica

CAPÍTULO II

27 Procedimentos e subprocedimentos

CAPÍTULO III

49 Constantes, variáveis e tabela

CAPÍTULO IV

71 Loops, comparações e recursividade

CAPÍTULO V

103 Operações numéricas

CAPÍTULO VI

123 Círculos, parábolas e elipses

CAPÍTULO VII

133 Os arquivos

CAPÍTULO VIII

139 Commodore 64 e Logo

CAPÍTULO IX

147 Apple II e Logo

APÊNDICE A

157 Mensagens de erro

APÊNDICE B

165 O código ASCII

BIBLIOGRAFIA

175 Bibliografia

PREFÁCIO



Logo é sinônimo de simplicidade e potência, duas características que dão honra aos autores da linguagem e que fazem com que este seja o melhor candidato à utilização, sempre mais próximo, da informática na escola.

A finalidade deste livro da B.B.I. é guiar o leitor na aprendizagem das informações necessárias para poder enfrentar depois problemas mais complexos. Quem já possui conhecimentos neste campo encontrará no texto informações interessantes.

Este livro, como outros, não poderá ensinar completamente o que quer dizer "saber programar bem". Só a experiência conseguida por cada um de nós e completada por uma boa bagagem teórica convertem a um programador qualquer em um programador competente.

Neste livro trataremos das versões da linguagem implantadas nos sistemas COMMODORE 64 e APPLE II; por motivos de clareza, no entanto, os primeiros capítulos que examinam a linguagem a partir do ponto de vista geral, terão seus exemplos baseados no LOGO TERRPIN do CMB 64. Nos capítulos seguintes serão analisados os detalhes técnicos relativos a cada versão.

CAPÍTULO I

A TARTARUGA, SEU RASTRO E A RESOLUÇÃO GRÁFICA



símbolo que, além de ser uma novidade, tem contribuído para a fama no mundo da informática da linguagem “LOGO” é a tartaruga (“turtle”): movendo-a na tela vai deixando uma pista no comprimento de seu percurso, desenhando imagens de diferentes tipos e cores.

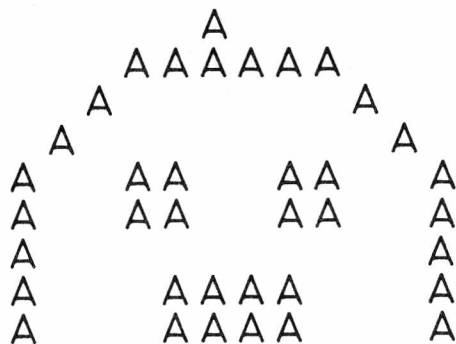
Estudaremos neste capítulo quais são as funções da tartaruga e, em particular, as instruções que estão na base de seus movimentos e do desenho em geral.

Controle direto da tartaruga

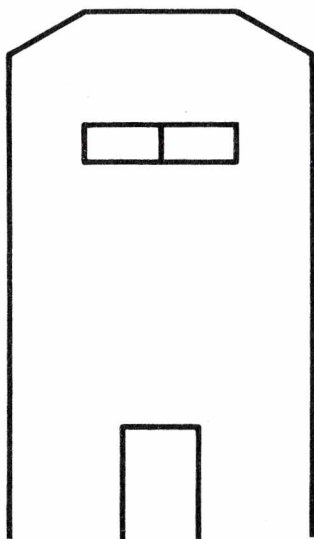
Nosso computador pode trabalhar de duas formas diferentes na hora de visualizar uma imagem: em “BAIXA” e “ALTA” resolução gráfica (falaremos também do modo TEXTO e do modo GRÁFICO).

O primeiro tipo de visualização é típico do computador no momento de ligar; se pressionamos as teclas do console veremos imediatamente na tela os caracteres correspondentes.

Se quisermos fazer na tela imagens a nosso gosto utilizando exclusivamente os caracteres, podemos manejá-los de tal maneira que obteremos uma figura que se aproxime à que desenhemos.



Ao contrário, o modo de “alta resolução” dá possibilidade ao programador de traçar gráficos e figuras com maior precisão: ele tem à sua disposição uma tela gráfica na qual pode desenhar, ponto por ponto, as imagens necessárias.



De um modo geral, a utilização do modo de alta resolução apresenta notáveis dificuldades de utilização por parte do usuário: a linguagem LOGO foi estudada (também) para permitir ao programador controlar individualmente cada ponto da tela de maneira fácil e eficaz, permitindo aproveitar completamente as possibilidades gráficas do computador sem recorrer a pesados, além de difíceis, cálculos de coordenadas.

O LOGO permite a completa (em seguida veremos o que quer dizer este termo) utilização de uma janela de vídeo formada por “X” pontos horizontais e por “Y” verticais (dois parâmetros variáveis segundo o tipo de computador utilizado) sobre a qual podem ser traçados gráficos e desenhos.

A primeira instrução que vamos examinar será, considerando as premissas:

DRAW (return)

que apaga a tela, coloca no centro a tartaruga e predispõe a tela em ALTA RESOLUÇÃO. Em particular, a parte superior da tela visualiza a página gráfica, enquanto a inferior fica de modo texto, isto é, fica em baixa resolução com o fim de poder seguir novas instruções e, eventualmente, permitir a leitura das mensagens de erro.

Esta é uma das três combinações possíveis de tela que podem ser escolhidas por meio das teclas de funções:

F1: Tela completa em forma de texto.

F3: Tela dividida em alta resolução e forma de texto.

F5: Tela completamente em alta resolução.

Nos computadores não dotados de teclas de funções e nos que as tem é possível conseguir também os mesmos resultados pressionando:

TEXTSCREEN (return): forma texto.

SPLITSCEEN (return): gráfica e texto simultaneamente.

FULLSCREEN (return): só gráfica.

Pressionamos agora SPLITSCREEN (return) e poderemos observar e ver os comandos pressionados (na parte baixa da tela) e os resultados relativos (NA PARTE SUPERIOR), como indica a Figura 1.

Escrevemos por exemplo:

FORWARD 50 <RETURN>

Veremos como a tartaruga, desde sua posição imóvel no centro da tela, se move para frente 50 posições, traçando uma linha reta entre o ponto onde anteriormente encontrava-se e o ponto alcançado (Fig. 2).

Portanto, a sintaxe é (na seguinte não explicaremos a necessidade de pressionar a tecla <RETURN> de retorno do carro):

FORWARD (Posições a avançar)

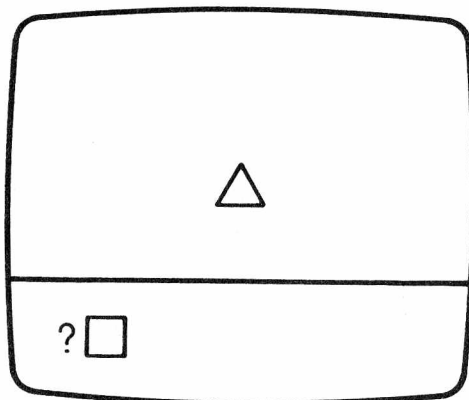


Figura 1. — Ao pressionar SPLITSCREEN a parte superior da tela fica no modo gráfico e a parte inferior em modo texto.

portanto, pressionando:

FORWARD 10 Avança adiante 10 posições

FORWARD 100 Avança adiante 100 posições

FORWARD 120 Avança adiante 120 posições

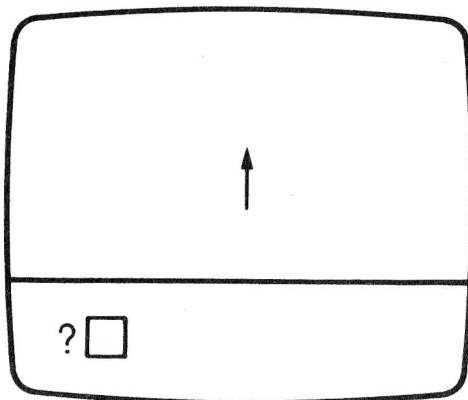


Figura 2. — O comando FORWARD permite que a tartaruga avance em linha reta.

Se agora queremos obter o resultado oposto, fazendo retroceder o cursor escreveremos:

BACK 50

e a tartaruga, se tivermos ficado na Figura 1.2, volta-se a colocar como na Figura 1.1.

A sintaxe é, por conseguinte, similar a adotada por FORWARD:

BACK (Posições a retroceder)

pressionando:

BACK 10 Retroceder 10 posições

BACK 100 Retroceder 100 posições

BACK 120 Retroceder 120 posições

Agora que sabemos como mover em linha reta, vejamos como podemos variar novas trajetórias.

Para fazer isto usamos duas instruções cujo nome explica amplamente sua função: RIGHT (direita) e LEFT (esquerda)

Pressionando:

RIGHT 90

veremos o cursor (a tartaruga) girar para a direita 90 graus.

e com:

LEFT 90

voltará a sua posição original, depois fará girar a esquerda 90 graus.

Recordamos, para aqueles que não tenham muita familiaridade com a anotação sexagesimal dos ângulos que:

Circunferência = 360 graus

Meia circunferência = 180 graus

Ângulo reto = 90 graus

e que: os ângulos internos de um triângulo equilátero regular va-

- lem 120 graus.

- a soma dos ângulos internos de qualquer triângulo é 180 graus;
- os ângulos internos de um quadrado ou de um retângulo valem 90 graus;
- os ângulos internos de um hexágono regular valem 120 graus.

É correto escrever:

RIGHT (graus de rotação a direita)

LEFT (graus de rotação a esquerda)

Expomos, em seguida, um exemplo que mostra a utilidade dos quatro comandos básicos que acabamos de ver e que mais uma vez nos sirva para traçar uma figura geométrica.

Para obter um triângulo equilátero pressionamos:

LEFT 90

gira a tartaruga 90 graus para a esquerda, dispondo-a a escrever em horizontal com relação ao plano visual, pois inicialmente está “olhando” para cima;

FORWARD 50

A avança 50 posições, traçando uma linha reta horizontal que representa a metade esquerda e a base do triângulo;

RIGHT 120

gira o cursor 120 graus a direita. Recordando que é o ângulo plano é de 180 graus e que o ângulo interno de um triângulo equilátero é de 60 graus, o ângulo de rotação exato é o resultado de $180 - 60 = 120$ graus;

FORWARD 100

avança 100 posições em sentido oblíquo com uma inclinação respectiva da base de 60 graus, formando o lado esquerdo do triângulo;

RIGHT 120

gira 120 graus a direita. Para a medida dos graus vale o exemplo

anterior: $180-60 = 120$ graus;

FORWARD 100

avança 100 posições, gerando o lado direito do triângulo;

RIGHT 120

fecha a rotação com um último ângulo de 120 graus a direita. A tartaruga está de novo na horizontal em relação ao respectivo plano;

FORWARD 50

termina a construção do triângulo traçando a segunda metade do lado base (Fig.3).

Desenh mos agora um quadrado.

RIGHT 90

girando a tartaruga à direita a colocamos sobre o plano horizontal (no programa anterior obtivemos o mesmo resultado com a rotação à esquerda).

FORWARD 50

traça a metade do lado da base;

LEFT 90

gira à esquerda 90 graus criando o ângulo de base direita;

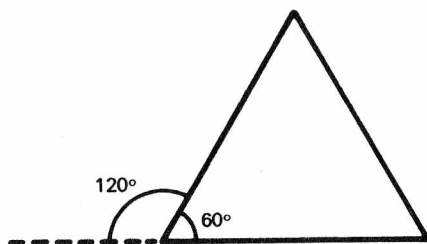


Figura 3. — Triângulo equilátero, facilmente realizável com o LOGO.

FORWARD 100

avança 100 posições desenhando o lado direito;

LEFT 90

90 graus à esquerda, criando vértice superior direito;

FORWARD 100

traça a base superior;

RIGHT 90

vamos fazer esta operação um pouco complicada; para criar o ângulo superior esquerdo seria suficiente escrever LEFT 90, porém, ao contrário, desta maneira poderemos mostrar a utilização da instrução BACK.

BACK 100

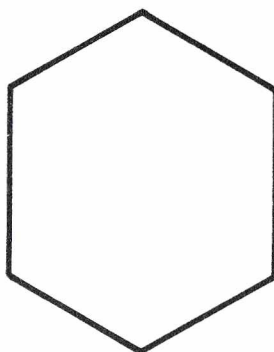
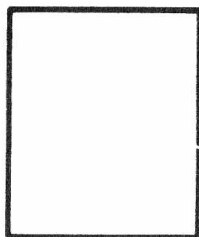
leva a tartaruga até a base do quadrado, traçando o lado esquerdo;

RIGHT 90

último ângulo (inferior esquerdo). É RIGHT, pois temos baixado "de costas";

FORWARD 50

traça a segunda metade da base (Fig. 4).



 *Figura 4. — Quadrado e hexágono regular obtidos com comandos diretos.*

O programa que em continuação veremos, desenha, análogamente aos anteriores, outra figura geométrica regular (Fig.4)

```
LEFT 90
FORWARD 35
RIGHT 60
FORWARD 70
RIGHT 60
FORWARD 70
RIGHT 60
FORWARD 70
RIGHT 60
FORWARD 70
RIGHT 60
FORWARD 70
RIGHT 60
FORWARD 35
```

Estudaremos agora duas instruções de relevante importância para a programação gráfica, que permitem a colocação da tartaruga em qualquer zona da tela.

Vejamos na forma operacional (ou direta)

SETX x

leva a tartaruga até o ponto de abscissa "X", porém não muda a ordenada. Este exemplo é válido, obviamente, pensando na tela como em um plano sobre o qual está fixado um sistema de referência de eixos cartesianos ortogonais, como se mostra na Figura 5.

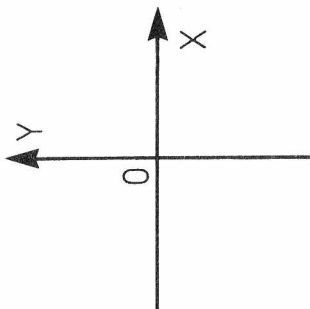
SETY y

força a ordenada da tartaruga ao valor "y", não alterando a posição horizontal (abscissa).

SETXY x y

leva a tartaruga até o ponto de coordenadas cartesianas (x,y).

É importante sublinhar que se fizermos deslocar a tartaruga



 *Figura 5. — Sistema cartesiano de referência considerado para fixar a posição de um ponto na tela.*

desde o ponto “A” até o ponto “B” esta deixará atrás de si a “pista” habitual; seguidamente veremos como evitar isto usando o comando `PENUP`.

Utilizando estes comandos será muito fácil traçar um quadrado:

```
HOME
CLEARSCREEN
SETY 30
SETX 30
SETY 0
SETX 0
```

A primeira instrução **HOME**, leva a tartaruga à posição central da tela (origem dos eixos) e além disso situa a cabeça da tartaruga paralelamente ao eixo das ordenadas positivas ($y +$).

A segunda instrução da lista antes mencionada limpa completamente a tela colocada em alta resolução.

De fato, pressionando somente

CLEARSCREEN

desaparecerão todas as figuras anteriores da área interna gráfica sem que, de qualquer maneira, resultem modificados os parâmetros definidos anteriormente como cor, modo da tela e, sobretudo, posição da tartaruga.

Se escreveu antes a ordem HOME e, em seguida CLEARSCREEN porque fazendo de maneira inversa, depois de apagar a tela (CLEARSCREEN) se traçaria a linha reta que une a posição da tartaruga com o centro da tela (HOME), com o qual esta ficaria "suja" de novo.

Além disso, existe um comando mais complexo capaz de levar a cabo simultaneamente as funções de apagar a tela e de colocar de novo no centro desta a tartaruga. Trata-se (como já dissemos) de:

DRAW

Vamos analisar a seguinte série de instruções, que resumem o que se tem descrito até agora desenhando uma letra e que, ao mesmo tempo, permite o exame de duas novas instruções.

```
HOME
CLEARSCREEN
FORWARD 120
RIGHT 90
FORWARD 40
RIGHT 90
PENUP [1]
FORWARD 60
RIGHT 90
PENDOWN [2]
FORWARD 40
LEFT 90
FORWARD 60
LEFT 90
FORWARD 40
PENDOWN [2]
FORWARD 40
LEFT 90
FORWARD 60
LEFT 90
FORWARD 40
```


A instrução marcada (1), como podem ter notado durante a execução do programa, inibe a “mancha” da tartaruga, que se desloca sem escrever nem apagar; posteriormente as marcadas (2) a reativará para escrever a letra E.

Em resumo:

PENUP

faz que o movimento da tartaruga não fique marcado, e

PENDOWN

é seu contrário ativando de novo o desenho aos movimentos.

Se ao finalizar o programa anterior pressionarmos:

HIDETURTLE

veríamos desaparecer da tela a tartaruga. É importante notar que ainda sendo invisível segue desenhando sua pista (a não ser que antes tivéssemos pressionado PENUP).

Para nos assegurarmos do que foi dito escreveremos:

```
HOME  
CLEARSCREEN  
HIDETURTLE  
FORWARD 100
```

Para visualizar de novo a tartaruga usaremos

SHOWTURTLE

Outra instrução que atua sobre a plumilha é

PENERASE

Como consequência do uso deste comando, a tartaruga, ao mover-se apaga em vez de escrever.

Vejamos um exemplo de como utilizar corretamente esta instrução;

```
FORWARD 100
```

PENERASE BACK 100

Ao final não fica nenhum rastro: tal e qual apareceram, desapareceram.

Pressionando seguidamente

PENCOLOR X

com X inteiro e positivo veremos de novo a pista.

Para que o cursor seja visto de novo tem que escrever

SHOWTURTLE

Finalmente observaremos o efeito de

STAMPCHAR "A"

Esta instrução permite a impressão dos caracteres standard em alta resolução; imprime a letra na posição seguinte à que é encontrada a tartaruga.

CLEARSCREEN STAMPCHAR "A"

veremos a nossa tartaruga desenhar a letra "A" na tela definida em alta resolução

As cores

Neste parágrafo nos ocuparemos de instruções que tenham grande importância no contexto de uma programação agradável e esteticamente válida.

Falamos, evidentemente, de comandos que podem trocar as características cromáticas da tela, do fundo, das linhas traçadas em alta resolução e da tartaruga.

Examinemos, antes de tudo, uma instrução que troca a cor da página de alta resolução. Escreveremos:

BACKGROUND 5 (return)

A tela colocada em alta resolução toma a cor correspondente ao código 5.

CODIGO	COR
00	PRETO
01	BRANCO
02	VERMELHO
03	AZUL ESCURO 1
04	PÚRPURA
05	VERDE 1
06	AZUL ESCURO 2
07	AMARELO
08	LARANJA
09	MARROM
10	VERMELHO 1
11	CINZA 1
12	CINZA 2
13	VERDE 2
14	AZUL
15	CINZA 3

 Tabela 1. — Relação de códigos de cores para o Commodore 64.

A sintaxe, portanto, é:

BACKGROUND (código-cor)

Ao que se refere a relação código-cor, esta varia segundo o computador e a tela utilizados; a tabela 1 é a de referência para o sistema COMMODORE 64.

Se, pelo contrário, desejamos trocar as cores da página texto utilizaremos

TEXTBG X

que em “x” especifica a cor desejada (segundo o conteúdo da tabela 1).

TEXTCOLOR x

Indica a cor dos caracteres standard.

Examinaremos o que conseguimos com esta série de instruções, executadas de maneira direta:

```
HOME  
CLEARSCREEN  
TEXTBG O
```

colore à página de texto de preto (tabela 1).

BACKGROUND 2

colore a página de alta resolução em vermelho.

TEXTCOLOR 13

dá a cor verde (em base à tabela 1) aos caracteres standard.

Arquivar as imagens

Depois de ter construído uma figura surge o problema de poder conservá-la sobre um suporte magnético.

Desta tarefa se ocupam três comandos com funções complementares, que permitem todas as operações de “gravação”, “carregamento” e “cancelamento”.

Escreveremos a seguinte sequência de instruções, que constroem o gráfico de exemplo dos três comandos antes mencionados:

```
SINGLECOLOR
```

FULLSCREEN
PENUP
FORWARD 100
LEFT 90
FORWARD 16 + 100
LEFT 180
STAMPCHAR "S
FORWARD 8
STAMPCHAR "A
FORWARD 8
STAMPCHAR "V
FORWARD 8
STAMPCHAR "E
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 24
LEFT 180
STAMPCHAR "R
FORWARD 8
STAMPCHAR "E
FORWARD 8
STAMPCHAR "A
FORWARD 8
STAMPCHAR "D
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 24
LEFT 180
STAMPCHAR "E
FORWARD 8
STAMPCHAR "R
FORWARD 8

STAMPCHARD "A
FORWARD 8
STAMPCHAR "S
FORWARD 8
STAMPCHAR "E
FORWARD 8
PENDOWN
LEFT 45
FORWARD 142
RIGHT 270
FORWARD 142
BACK 142
RIGHT 135
PENUP
FORWARD 8
STAMPCHAR "P
FORWARD 8
STAMPCHAR "I
FORWARD 8
STAMPCHAR "C
FORWARD 8
STAMPCHAR "T
FORWARD 16

Nesta situação, para salvar sobre o disco a página gráfica resultante (Fig. 6) escreveremos;

SAVEPICT "nome

Seguidamente, para recuperar terá que escrever:

READPICT "nome

Em contrário, se, quiser apagar a página do disco:

ERASEPICT "nome

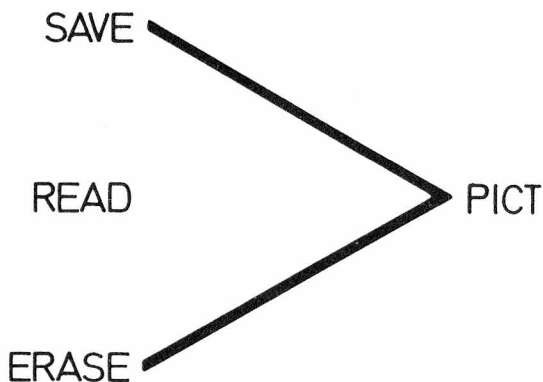


Figura 6. — Página gráfica que vamos gravar.

Resumo das instruções

DRAW

Para entrar em alta resolução.

NODRAW

Contrário de draw.

TEXTSCREEN

Vídeo completo em modo texto.

SPLITSCREEN

Vídeo subdividido em alta resolução e em modo texto.

FULLSCREEN

Vídeo completamente em alta resolução.

FORWARD x

Avança a tartaruga “x” posições.

BACK x

Atrasa a tartaruga “x” posições.

RIGHT x

Gira à direita “x” graus a tartaruga.

LEFT x

Gira à esquerda “x” graus a tartaruga.

SETX x

Situa a tartaruga na abcissa “x”.

SETY x

Situa a tartaruga na ordenada “y”.

CLEARSCREEN

Limpa a área gráfica.

HOME

Coloca a tartaruga na parte central da tela.

PENCOLOR x

Dá a tartaruga e a pista deixada por ela a cor "x".

PENERASE

Sob este comando a tartaruga não escreve, porém apaga

BACKGROUND x

Determina a cor da página gráfica.

TEXTCOLOR x

Dá aos caracteres página texto e cor desejada.

TEXTBG x

Parâmetros de cor para a área do texto.

PENUP

Sob este comando a tartaruga se desloca sem escrever nem apagar.

PENDOWN

Leva a cabo a função contrária a PENUP.

HIDETURTLE

Torna a tartaruga invisível, porém não sua pista.

SHOWTURTLE

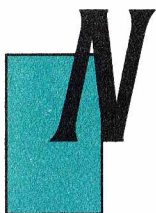
Torna a tartaruga visível.

STAMPCHAR"x

Imprime em alta resolução os caracteres que seguem as aspas.

CAPÍTULO II

PROCEDIMENTOS E SUBPROCEDIMENTOS



este capítulo falaremos de uma das partes mais importantes da linguagem LOGO: os *procedimentos*, isto é, *aquelas "palavras" que nós mesmos podemos definir e que nos permitem usar um mesmo grupo de instruções por meio da simples menção da palavra.*

Definição de novas palavras

Suponhamos que queremos desenhar um retângulo como o da Figura 1.

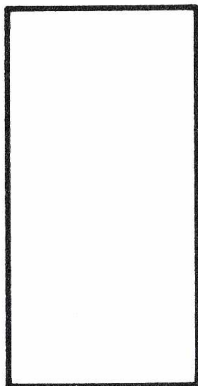


Figura 1. — Retângulo para desenhar.

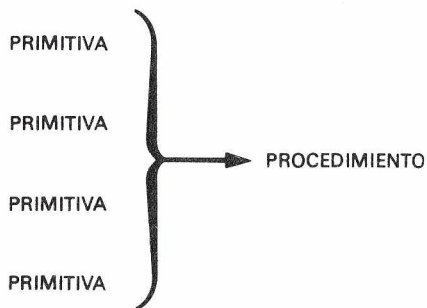
Com os meios atualmente à nossa disposição não podemos fazer outra coisa que pressionar seqüencialmente as instruções:

```
FORWARD 50  
RIGHT 90  
FORWARD 100  
RIGHT 90  
FORWARD 50  
RIGHT 90  
FORWARD 100
```

Nos vemos obrigados a escrever tudo isto já que o LOGO não coloca à nossa disposição, dentro do vocabulário dos comandos fundamentais, uma instrução que traga automaticamente a figura geométrica requerida. De fato, **este vocabulário de comandos resulta estar exclusivamente constituído por simples, ainda que importantes, palavras chamadas PRIMITIVAS**, que são as já conhecidas:

FORWARD, LEFT, RIGHT, PENCOLOR, etc.

Neste contexto ressalta um dos aspectos principais da linguagem LOGO: **o programador, conseqüentemente para suas próprias necessidades, pode definir um conjunto de NOVAS INSTRUÇÕES, chamadas tecnicamente PROCEDIMENTOS, que ampliarão o básico e limitado vocabulário.**



 *Figura 2. — O LOGO nos permite definir um procedimento a partir das primitivas adequadas.*

Vejamos na prática como construir uma nova instrução que, ao ser chamada, desenha na tela o retângulo da Figura 1.

Antes de tudo vamos decidir o nome que lhe daremos; é aconselhável nesta fase ter presente quatro normas gerais boas, aparentemente óbvias, que no aprendizado resultem de evidente utilidade e simplificando muito a compreensão dos programas:

1. Preferir uma etiqueta que ponha evidência e se possível SINTETIZE a função do procedimento.
2. BUSCAR nomes SIMPLES e, portanto, fáceis de recordar.
3. Comprovar que o nome escolhido não se confunda com outros dos procedimentos definidos anteriormente.
4. O intérprete LOGO não permite, que em seguida “batizemos” um procedimento com o apelativo típido de uma primitiva ou com o nome de um Procedimento já presente no vocabulário (por este motivo é impossível chamar um procedimento “FORWARD” ou “PENUP”).

Neste caso examinado por nós necessitamos de um procedimento que traga um retângulo; o nome que pode ser dado e que contenha os ditos requisitos necessários é, por exemplo, “DESENHARETANGULO”.

Fazemos por meio da instrução:

TO DESENHARETANGULO

Neste momento podemos introduzir seqüencialmente as instruções que desejamos que se memorizem em relação ao nosso procedimento.

Agora, já nos encontramos no ambiente EDITOR, os comandos teclados não serão levados a cabo imediatamente. Só se memorizará e se assimilará à palavra que estamos definindo; será depois, que chamarmos o nosso Procedimento, quando esta seqüência de comando será levada a cabo automaticamente e de forma seguida mas sem ter que novamente escrever por inteiro.

Depois de haver escrito o TO DESENHARETANGULO e esclarecido estes pontos, pressionemos:

RIGHT 90; [realiza um giro de 90 graus]

FORWARD 90; [avança 90 posições]

LEFT 90; [giro à esquerda]

FORWARD 40; [avança nessa direção]

LEFT 90; [outros 90 graus]

FORWARD 90; [lado superior]

LEFT 90; [último giro]
FORWARD 40; [último lado]
END

No exemplo podemos observar dois detalhes importantes.

- Na definição de um procedimento qualquer permite-se fazer comentários de qualquer tipo, com a condição de que estão fechados entre colchetes e precedidos por sinal “;”
- A última instrução de um procedimento deve ser necessariamente END; no caso de omissão, será o EDITOR quem se encarregará de acrescentar ao final da lista.

Pressionemos agora, de uma vez, as teclas

<CTRL> e <C>

para que o procedimento em questão se memorize e, portanto, entre no vocabulário LOGO.

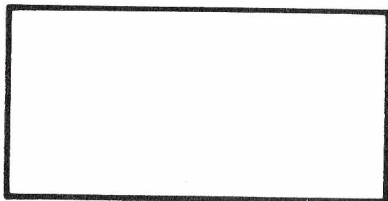
Neste ponto o computador contesta:

PLEASE WAIT...

e em seguida, depois, se tudo foi introduzido corretamente:

DESENHARETANGULO DEFINED

Temos de voltar ao modo indireto, ou seja, a um ambiente de programação no qual os comandos teclados não se memorizam, mas que são levados a cabo imediatamente, ao contrário do que passava no EDITOR.



 *Figura 3. — Retângulo obtido com o procedimento DESENHARETANGULOS.*

Chamamos agora o Procedimento recém-construído com

RUN DESENHARETANGULO

que produz na tela o retângulo da Figura 3.

Agora que temos entendido a utilidade e as operações necessárias em relação à criação de um novo comando podemos, por analogia, discutir os subprocedimentos, isto é, os procedimentos que participam na definição anterior de novas instruções.

Se, por exemplo, queremos desenhar na tela quatro retângulos colocados como na Figura 4, poderemos definir um procedimento que chame quatro vezes o procedimento "DESENHARETÂNGULO" definido anteriormente e que, neste caso, toma função de subprocedimento. O diagrama de sintaxe é o mostrado na Figura 5.

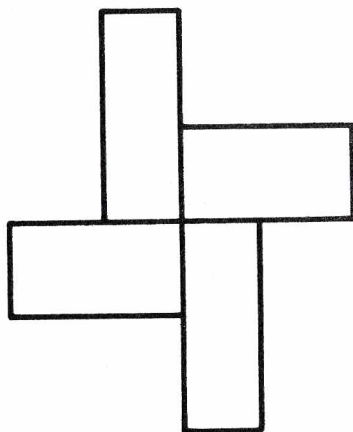


Figura 4. — Um procedimento pode usar de outros, aos quais converte em subprocedimentos, para seus fins.

Portanto podemos escrever

TO QUATRO • RETÂNGULOS

DESENHARETANGULO; [primeiro]

DESENHARETÂNGULO; [segundo]

DESENHARETANGULO; [terceiro]

DESENHARETANGULO; [quarto]

END

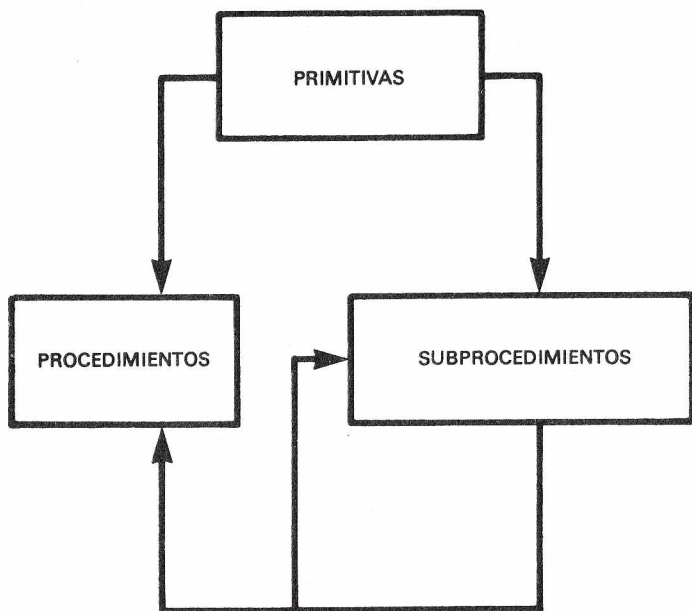


Figura 5. — Gráfico de relação entre primitivas, subprocedimentos e procedimentos.

e pressionando simultaneamente

< CTRL > e < C >

memorizamos a nova palavra-chave encontrada “QUATRO • RETÂNGULOS. O procedimento QUATRO • RETÂNGULOS não explora completamente as capacidades do LOGO; nos capítulos que seguiremos veremos como existe uma instrução que pode evitar a repetição de instruções idênticas.

Assim, para conseguir o que temos na Figura 4 será suficiente escrever:

RUN QUATRO•RETÂNGULOS

Está claro que podendo definir a nosso próprio gosto instruções, a linguagem LOGO permite uma sensível flexibilidade em relação a cada problema de programação.

Eis aqui alguns exemplos, que podem constituir para o leitor interessado uma válida ajuda, resumindo o aprendido até agora.

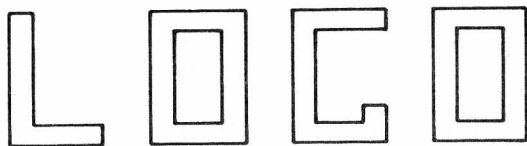
TO LOGO

HOME; [LEVA A TARTARUGA AO CENTRO
DA TELA]
CLEARSCREEN; [APAGA POR COMPLETO TODA A
TELA]
;[COMEÇA O PROGRAMA DE DESENHO DA
PALAVRA LOGO]
PENUP; [ALÇA O LÁPIS PARA QUE O PERCURSO
DA TARTARUGA NÃO SEJA VISUALIZADO]
LEFT 90; [GIRO DE 90 GRAUS À ESQUERDA]
FORWARD 70; [AVANÇA NA DIREÇÃO ADIANTE
70 POSIÇÕES]
PENDOWN; [UMA VEZ SITUADA A TARTARUGA
ONDE QUERÍAMOS ABAIXAMOS O LÁPIS PARA
DESENHAR SEU RASTRO]
FORWARD 40; [DESENHAMOS A ' L ']
RIGHT 90; [GIRO A DIREITA DE 90 GRAUS]
FORWARD 70
RIGHT 90
FORWARD 10
RIGHT 90
FORWARD 60
LEFT 90
FORWARD 30
RIGHT 90
FORWARD 10
PENUP; [ACABADA A ' L ' PREPARAMOS TUDO
PARA O ' O ' LEVANTANDO O LÁPIS ANTES]
LEFT 90
FORWARD 20
LEFT 90
0; [CHAMAMOS AO SUBPROCEDIMENTO ' O '
PENDOWN

FORWARD 40 ; [DESENHANDO A 'G']
LEFT 90
FORWARD 20
LEFT 90
FORWARD 10
LEFT 90
FORWARD 10
RIGHT 90
FORWARD 20
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 30
LEFT 90
FORWARD 10
LEFT 90
FORWARD 40
LEFT 90
FORWARD 70
LEFT 90
PENUP; [PREPARAMOS PARA O SEGUNDO 'O']
FORWARD 60
LEFT 90
O; [NOVA CHAMADA AO SUBPROCEDIMENTO 'O']
END

TO O
PENDOWN; [O PROCEDIMENTO DESENHA A
LETRA 'O']
FORWARD 70
RIGHT 90
FORWARD 40
RIGHT 90
FORWARD 70

```
RIGHT 90
FORWARD 40
PENUP; [LEVANTAMOS O LÁPIS PARA
    SITUARMOS NA PARTE INTERIOR]
RIGHT 90
FORWARD 10
RIGHT 90
FORWARD 10
PENDOWN
FORWARD 20
LEFT 90
FORWARD 50
LEFT 90
FORWARD 20
LEFT 90
FORWARD 50
PENUP; [POSICIONAR A TARTARUGA NO
    COMEÇO DA SEGUINTE LETRA]
FORWARD 10
LEFT 90
FORWARD 50
END
```



 *Figura 6. — Resultado da execução de um procedimento que, por sua vez, faz uso de subprocedimentos.*

O Editor

Com este termo técnico se indica aquela seção da linguagem cuja função específica é a de permitir e ajudar ao máximo o usuário quando este escreve procedimentos, subprocedimentos e, portanto, qualquer programa.

No momento que o computador é ligado situa em modo direto ou de comandos (COMMAND MODE); nele, se pressionamos uma instrução correta, esta levará a cabo imediatamente em quanto pressionamos a tecla RETURN.

Por exemplo, escrevendo:

FORWARD 50 (e o necessário RETURN)

veremos depois de alguns momentos, como a tartaruga move-se na tela 50 posições.

No entanto, se pressionamos

TO NOME PROCEDIMENTO

ou então

EDIT NOME PROCEDIMENTO

ou também

EDIT ALL

entra-se em modo editor, isto é, podemos introduzir e/ou modificar as seqüências de instruções relativas ao procedimento que desejamos.

Quando mais tarde chamarmos este procedimento, as instruções dadas serão levadas a cabo em seqüência fielmente, como se nós as tivéssemos pressionado no modo direto.

Examinaremos agora quais são as opções a disposição do programador LOGO no caso de que tenha entrado em ambiente EDITOR.

À esquerda oferecemos uma lista das teclas a pressionar simultaneamente, e a direita é explicada sua função (para o CBM 64).

CTRL e O	Equivale a pressionar RETURN; se introduz uma linha vazia no texto do Programa e levam a nova linha todos os caracteres seguintes ao cursor.
CTRL e N	Pressionando simultaneamente estas duas teclas, o cursor se situa no início da linha SUCESSIVA com respeito à que se encontra.
CTRL e P	Efetua a função contrária à do comando anterior, movendo o cursor ao início da linha ANTERIOR.

CTRL e L	Coloca o cursor ao final da linha atual.
CTRL e A	Desloca o cursor ao extremo esquerdo da tela.
CTRL e F	Leva o cursor ao final do programa.
CTRL e B	Movimenta o cursor até o primeiro caractere do programa.
CTRL e D	Apaga-se o caractere sobre o qual está colocado o cursor.
CTRL e K	Apaga todos os caracteres à DIREITA do cursor.

Finalmente, as instruções mais importantes:

CTRL e C

CTRL e G

a primeira define o procedimento que está levando a cabo e as eventuais variações, enquanto que a segunda dá por acabada a fase de variação e/ou modificação. Estes dois últimos comandos TRANSFEREM DE NOVO o controle à maneira IMEDIATA. A Figura 7 ilustra este conceito.

Listagem e eliminação de procedimentos

Nos parágrafos anteriores temos falado detalhadamente das operações necessárias para a definição e modificação de um procedimento e/ou subprocedimento. No entanto, muito frequentemente passa que, depois de ter escrito um programa e ter comprovado seus resultados, nasce a necessidade de visualizar velocemente a listagem correspondente.

Para fazer tudo isto é fundamental poder **ter a nossa disposição o nome de todos os procedimentos presentes até o momento no vocabulário**. A instrução à qual se confiou esta incumbência é:

POTS

Se, por exemplo, nosso vocabulário está constituído pelos procedimentos "DESENHARETANGULO", "QUATRORETANGULOS" e "TRIANGULO", pressionando POTS obteremos na tela:

? POTS

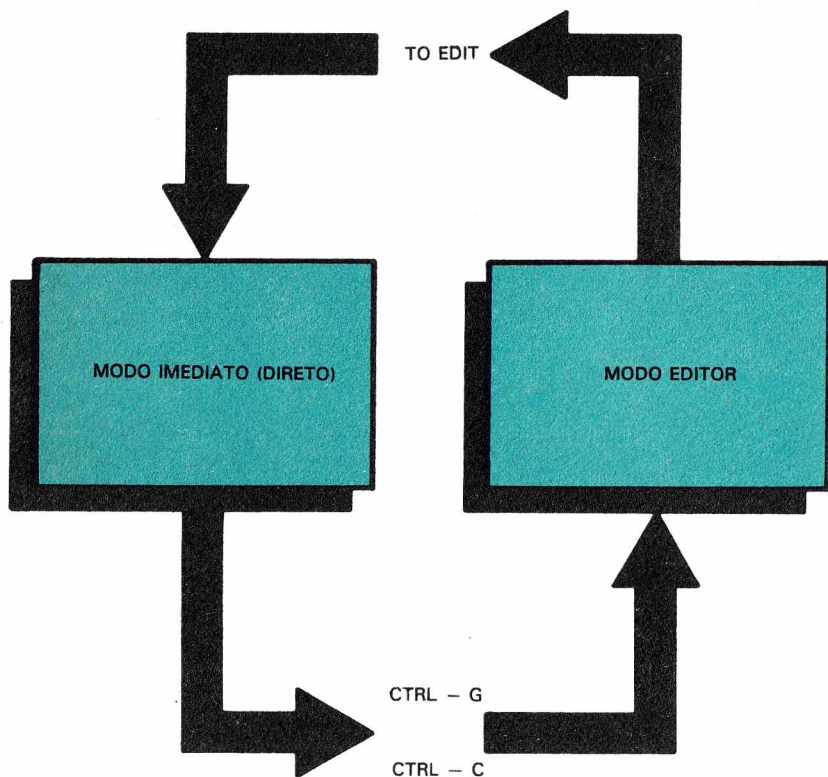


Figura 7. — Esquema da passagem de modo direto a editor e vice-versa. Os comandos são os do CBM 64.

TO DESENHARETANGULO
 TO QUATRORETÂNGULOS
 TO TRIÂNGULO

Para conseguir a listagem completa de procedimento já escrito usa-se o comando PO, cuja sintaxe é:

PO NOMEPROCEDIMENTO

Esta operação pode parecer a princípio igual a instrução TO NOMEPROCEDIMENTO, porém, na realidade, é muito distinta.

De fato, enquanto TO permite mais ou menos a trabalhosa modificação da listagem, já que passa o controle das operações ao EDITOR, PO permite exclusivamente a visualização dessa listagem.

Se pressionamos agora:

PO ALL

aparecerá na tela a listagem de todos os procedimentos residentes em memória, que aparecem na ordem contrária com respeito ao que foi definido. Em outras palavras: a última palavra criada pelo programador será a primeira a ser listada.

Recorde que "scroll" (deslocamento) da tela pode ser momentaneamente interrompido pressionando de uma vez.

CTRL e W

e ativada de novo com um só pressionar de uma tecla qualquer.

Agora que já examinamos como INTRODUZIR, LISTAR e MODIFICAR um procedimento, fica ao nosso critério a fase de anulação, parcial ou total, do VOCABULÁRIO.

Esta incumbência, muito importante (recordemos que anular um procedimento significa ganhar memória), está confiada a instrução ERASE, cuja sintaxe é:

ERASE NOMEPROCEDIMENTO

Com ela eliminamos (e desaparece assim do vocabulário) a palavra da qual especificamos o nome, na condição que não seja uma primitiva.

ERASE ALL

apaga TODOS os procedimentos residentes em memória, levando o vocabulário LOGO às condições típicas que seguem ao funcionamento inicial do nosso computador.

Gravar o vocabulário

Quando se tem definido novas palavras de uso freqüente ou que voltaremos a usar é necessário armazená-las para termos à nossa disposição em qualquer momento sem termos de voltar a escrever e verificar.

O LOGO permite a memorização dos procedimentos em suporte magnético (de maneira particular em disquetes - Flopp Disk

e em fita) para criar um arquivo pessoal de novas instruções. Se quisermos "salvar" um vocabulário (ARQUIVO) criado por nós ou outros deveremos usar o comando:

SAVE "NOMEARQUIVO"

Para averiguar se a gravação do arquivo-vocabulário foi levado a cabo corretamente chamamos ao diretório (directory) do disco pressionando a instrução:

CATALOG

Veremos como aparece o nome de nosso arquivo.

Carregaremos agora na memória o vocabulário salvo.

Para estarmos seguros de que esta operação foi feita e não nos enganamos pelo fato de que o vocabulário já estava em memória, levamos o computador às condições iniciais (de ligado), de forma que esteja sem nenhum procedimento pré-definido:

GOODBYE

A máquina será inicializada. Chegados aqui, utilizamos a instrução

READ "NOME"

para que reapareça o arquivo-vocabulário anteriormente arquivado.

Para fazer mais compreensível o que foi dito até agora pegamos um exemplo prático: consideremos um vocabulário constituído pelos procedimentos DESENHARETANGULO e QUATRORETANGULOS.

Para gravá-los pressionaremos

SAVE "PRIMEIRO-VOCABULÁRIO"

Para carregá-lo:

READ "PRIMEIRO-VOCABULÁRIO"

É importante levar em conta dois casos:

1. É possível salvar o vocabulário com um nome distinto dos pro-

cedimentos neles contidos.

2. No caso de carregar a memória em um segundo vocabulário, este se junta ao anterior sem ser modificado. Agora então, cuidado para não carregar dois procedimentos que tenham o mesmo nome, pois neste caso o último vocabulário será cancelado.

Fica por discutir a eliminação das palavras salvas sobre o disco. Esta operação, chamada SCRATCH, está confiada ao comando:

ERASEARQUIVO "NOME

Programas resumidos

Programa 1 - Demonstração

O seguinte programa traça o gráfico das operações realizadas entre a memória e o arquivo em disco ou cassete seguintes:

Definição dos procedimentos

Salvar arquivo

Carregar arquivo em memória

Recordamos que depois de cada linha existe um <RETURN >

TO D

; [ESTE PROCEDIMENTO ESCRIVE A LETRA 'D'
E MOVE A TARTARUGA EM 8 POSICOES]

STAMPCHAR "D

FORWARD 8

END

TO L

; [ESCREVE A 'L']

STAMPCHAR "L

FORWARD 8

END

TO F

```
    ; [ESCREVE A 'F']  
    STAMPCHAR "F"  
    FORWARD 8  
END
```

```
TO V  
    ; [ESCREVE A 'V']  
    STAMPCHAR "V"  
    FORWARD 8  
END
```

```
TO S  
    ; [ESCREVE A 'S']  
    STAMPCHAR "S"  
    FORWARD 8  
END
```

```
TO A  
    ; [ESCREVE A 'A']  
    STAMPCHAR "A"  
    FORWARD 8  
END
```

```
TO I  
    ; [ESCREVE A 'I']  
    STAMPCHAR "I"  
    FORWARD 8  
END
```

```
TO R  
    ; [ESCREVE A 'R']  
    STAMPCHAR "R"  
    FORWARD 8  
END
```

TO O
; [ESCREVE A 'O']
STAMPCHAR "O
FORWARD 8
END

TO E
; [ESCREVE A 'E']
STAMPCHAR "E
FORWARD 8
END

TO M
; [ESCREVE A 'M']
STAMPCHAR "M
FORWARD 8
END

TO READ

; [ESCREVE A PALAVRA 'READ']
RIGHT 90
FORWARD 70
LEFT 90
BACK 130

R; [CHAMA AO SUBPROCEDIMENTO 'R']
E
A
D
FORWARD 20
END

TO FILE

; [ESCREVE A PALAVRA 'FILE']

FORWARD 65

RIGHT 90

FORWARD 70

LEFT 90

F

I

L

E

END

TO SAVE

; [ESCREVE A PALAVRA 'SAVE']

LEFT 90

FORWARD 70

RIGHT 90

FORWARD 40

S

A

V

E

END

TO MEMORIA

; [ESCREVE A PALAVRA 'MEMORIA']

FORWARD 30

RIGHT 90

FORWARD 205

LEFT 180

M

E
M
O
R
I
A

END

TO RETANGULO

; [DESENHA UM RETANGULO]

RIGHT 90

FORWARD 60

LEFT 90

FORWARD 60

LEFT 90

FORWARD 120

LEFT 90

FORWARD 60

LEFT 90

FORWARD 60

END

TO DEMONSTRAÇÃO

;[CHAMA E CONTROLA TODOS OS

SUBPROCEDIMENTOS ANTERIORES]

HOME; [TARTARUGA AO CENTRO DA TELA]

CLEARSCREEN; [APAGA A TELA]

PENUP; [LEVANTA O LÁPIS PARA NÃO

MARCAR O PERCURSO]

RIGHT 90


```

FORWARD 80
PENDOWN; [SITUADA A TARTARUGA VOLTA A
    MARCAR SEU RASTRO]
LEFT 90
RETANGULO; [CHAMA AO SUBPROCEDIMENTO
    'RETANGULO']
RIGHT 90
FORWARD 30
RIGHT 90
FORWARD 180
RIGHT 90
FORWARD 30
RETANGULO
PENUP
LEFT 90
FORWARD 60

```

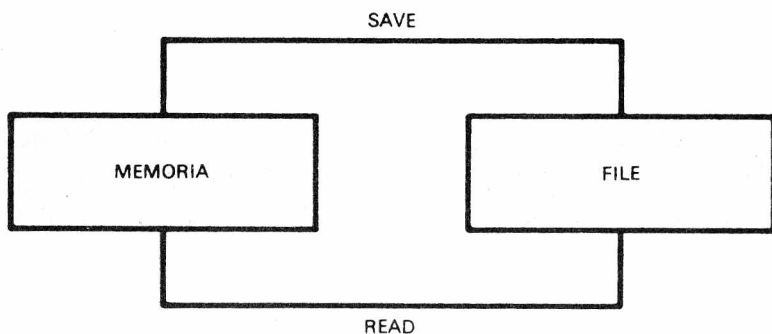


Figura 8. — Resultado do programa DEMONSTRAÇÃO.

```

PENDOWN
FORWARD 30
RIGHT 90
FORWARD 180
RIGHT 90
FORWARD 30

```

```
PENUP  
MEMORIA; [CHAMADA AO SUBPROCEDIMENTO  
  'MEMORIA']  
SAVE; [CHAMADA AO SUBPROCEDIMENTO  
  'SAVE']  
FILE; [CHAMADA AO SUBPROCEDIMENTO  
  'FILE']  
READ; [CHAMADA AO SUBPROCEDIMENTO  
  'READ']
```

END

Programa 2 - Folha

O programa FOLHA desenha, como diz seu nome, uma folha.

Em primeiro lugar cria-se um procedimento que desenhe um losângo e que depois possa ser chamado várias vezes repetidas, com uma única instrução (REPEAT) que veremos posteriormente.

TO FOLHA

```
HOME; [TARTARUGA AO CENTRO]  
CLEARSCREEN; [APAGA TELA]  
PENUP; [LEVANTA A PLUMILHA]  
FORWARD 10  
PENDOWN; [BAIXA A PLUMILHA]  
ROTACAO; [CHAMADA AO SUBPROCEDIMENTO  
  'ROTACAO']
```

END

TO ROTACAO

```
REPEAT 7 [LOSANGO]; [ESPERE PARA VER O  
  CAPITULO 5]
```

```
RIGHT 140  
FORWARD 50  
LEFT 40  
FORWARD 50  
RIGHT 220  
FORWARD 50  
LEFT 40  
FORWARD 50  
END  
TO LOSANGO  
LEFT 150  
FORWARD 60  
LEFT 60  
FORWARD 60  
RIGHT 240  
FORWARD 60  
LEFT 60  
FORWARD 60  
END
```

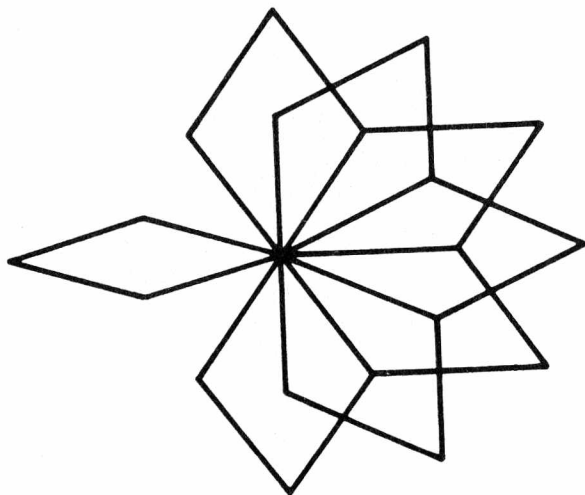


Figura 9. — Execução do programa FOLHA.

CAPÍTULO III

CONSTANTES, VARIÁVEIS E TABELAS

"Make" e "Print"



Como toda linguagem evoluida, o LOGO oferece a possibilidade de tratar a nível de programa constantes e variáveis com relativa simplicidade. Examinaremos em detalhes as palavras da linguagem que desenvolvem este aspecto operacional.

Temos que ter presentes que, ao definir uma constante ou variável não podemos assinalar nenhum dos nomes característicos do LOGO (ou seja, primitivas e procedimentos).

Além disso é necessário recordar que o valor numérico atribuível as constantes ou variáveis deve estar necessariamente compreendidas entre um mínimo e um máximo, típicos da máquina com a qual se trabalha.

Uma variável que no transcurso do programa assume um valor não consentido gera a imediata detenção de execução do procedimento em curso e a impressão da mensagem.

NUMBER TOO BIG (ou TOO SMALL)

Consideremos agora na prática o procedimento necessário para a definição de uma constante, que no exemplo é chamada "CONS" e que deve assumir o valor numérico inteiro 150.

É suficiente pressionar

MAKE "CONS 150

A instrução MAKE atribui à palavra seguinte o valor especí-

fico; esta instrução não precisa ser necessariamente uma constante, já que pode ser comodamente substituída com uma expressão, como no exemplo seguinte:

```
MAKE "ALFA (2 + 3*5)
```

Ou também:

```
MAKE "BETA SIN (180) * COS (0)
```

Outro aspecto fundamental da instrução MAKE é que permite também a criação de constantes e variáveis com argumentos alfanuméricos.

A sintaxe a seguir neste caso é ligeiramente diferente e pode ser sintetizada assim:

```
MAKE "NOME" ARGUMENTO
```

Querendo atribuir a frase "21. Janeiro.1986" à constante "DATA", escreve-se;

```
MAKE "DATA" 21. Janeiro 1986
```

O programador inexperiente deve ter cuidado ao utilizar a instrução do tipo

```
MAKE " ALFA 10  
MAKE " ALFA 20
```

que levaria, em primeiro lugar, a atribuição do valor numérico 10 à constante ALFA e, imediatamente depois, à colocação do valor 20 na mesma constante, com a perda completa da informação anterior (10).

É este um exemplo que, se inicialmente pode parecer simples e sem sentido, pode talvez perder tempo na correção de um programa mesmo para um usuário mais experiente.

Paralelamente à instrução MAKE examinaremos a linguagem primitiva cuja função é praticamente a oposta, ou seja, a de imprimir o valor de argumento e não a de atribuir um valor a uma variável ou constante.

A instrução é do tipo

```
PRINT VALOR
```

ou também

PRINT: NOMEVARIÁVEL

Pode ser utilizada de modo direto ou como programa. Provemos, portanto, o pressionamento de:

```
MAKE "FRASE" QUERIDO AMIGO
```

e

```
PRINT: FRASE
```

imprime-se, evidentemente: Querido amigo.

Para imprimir mensagens diretamente usa-se a sintaxe

PRINT [FRASE A IMPRIMIR]

O usuário mais experiente terá se certificado seguramente do fato de que nós utilizamos repetidamente o comando PRINT, os distintos argumentos imprimem sobre diferentes linhas da tela.

Pressione, por exemplo, o procedimento:

```
TO IMPRIMEFRASE  
  PRINT [ESTOU APRENDENDO]  
  PRINT [A UTILIZAR]  
  PRINT [O LOGO]  
END
```

Depois de ter executado, na tela aparece uma mensagem em três linhas do tipo:

```
ESTOU APRENDENDO  
A UTILIZAR  
O LOGO
```

Se desejarmos que nossa mensagem seja disposta sobre uma só linha, recorreremos à primitiva PRINT1, de toda similar, pelo que concorre à sintaxe, à PRINT examinada anteriormente.

Voltamos a escrever, portanto, nosso procedimento como abaixo indica-se:

```
TO IMPREMEFRASE  
  PRINT1 [ESTOU APRENDENDO]  
  PRINT1 [A UTILIZAR]  
  PRINT1 [O LOGO]  
END
```

A impressão sobre a tela é, como nós esperávamos:

ESTOU APRENDENDO A UTILIZAR O LOGO

Variáveis locais: LOCAL

Concluído o tema MAKE e PRINT, as instruções fundamentais de manipulação das variáveis numéricas e alfanuméricas, passemos a detalhes técnicos e examinemos a utilização de uma nova instrução primitiva (LOCAL) que reveste também uma tarefa de particular interesse.

Ocorre muito freqüentemente que um subprocedimento trabalhe sobre uma ou mais variáveis utilizadas pelo programa principal e cujo valor não deve ser alterado; é, portanto, indispensável que ao final da execução do procedimento a variável em questão reassuma o valor que tem no momento do requerimento.

A linguagem LOGO resolve com extrema simplicidade este problema através da instrução:

```
LOCAL "NOMEVARIABLE
```

Para entender a fundo a função desta instrução primitiva examinaremos o seguinte programa:

```
TO SUBPROCEDIMENTO  
  LOCAL "ALFA  
  MAKE "ALFA 6  
  PRINT :ALFA  
END
```

```
TO PROGRAMAPRINCIPAL
```



```
MAKE "ALFA 5
SUBPROCEDIMENTO
PRINT :ALFA
```

```
END
```

Vamos executar como de costume

```
RUN PROGRAMAPRINCIPAL
```

Os passos a desenvolver serão:

1. MAKE dá o valor 5 a ALFA.
2. Chama-se ao subprocedimento chamado SUBPROCEDIMENTO.
3. A instrução LOCAL indica que nosso subprocedimento trabalhará com a variável ALFA utilizada pelo programa principal e que, portanto, ao final da execução do subprocedimento deverá retomar automaticamente o valor original (que é 5).
4. O subprocedimento está em ação: alfa agora vale 6 (MAKE "ALFA 6).
5. Imprime-se o valor correspondente (6).
6. Volta-se ao programa principal: o valor de ALFA coloca-se automaticamente em 5.
7. Uma vez impresso ALFA o programa termina.

"REQUEST" e "READCHARACTER"

Ao introduzir em um programa a instrução "REQUEST", pára momentaneamente sua execução até que o operador pressione a tecla "RETURN".

A função desta instrução primitiva é pedir ao usuário uma informação que em seguida será utilizada pelo programa; se quisermos definir um procedimento que pergunte um número e que imprima seu quadrado, pressionaremos

```
TO QUADRADO
```

```
MAKE "X REQUEST; [X= TECLAS ANTES DO RETURN]
```

```
PRINT1 [QUADRADO É:]
```

```
PRINT: X * : X
```

```
END
```


Ativamos o procedimento com RUN QUADRADO, pressionamos um oito e "RETURN"; a contestação será:

O QUADRADO É 64

Ao contrário de uma instrução primitiva "REQUEST", a instrução "READCHARACTER" espera um dado só de entrada; o programa pára até que pressionemos uma tecla qualquer.

O exemplo pode sugerir ao leitor uma das distintas maneiras de utilização desta instrução primitiva. O breve programa chamado COMANDOS.DIRETOS, representado abaixo, permite desenhar na tela pressionando as teclas "A" = adiante, "B" = back, atrás; "D" = gira à direita, "I" gira à esquerda, "F" = final do procedimento.

Ao que se refere ao uso da instrução IF...THEN (utilizada no programa) aconselha-se ao leitor esperar e ler o capítulo 4.

TO COMANDOS. DIRETOS

HOME: [TARTARUGA A POSIÇÃO INICIAL]

CLEARSCREEN: [APAGA A TELA]

PRINT [PRESSIONAR A, B, D, I (F = FINAL)]

GETCHARACTER: [SUBPROCEDIMENTO PARA
REALIZAR A LEITURA DOS COMANDOS]

END

TO GETCHARACTER

MAKE "TECLAPRESSIONADA READCHARACTER

IF: TECLAPRESSIONADA = "F THEN NODRAW

PRINT [] PRINT [] PRINT [FIM] STOP

IF: TECLAPRESSIONADA = "A THEN PRINT

[ADIANTE] FORWARD 10

IF: TECLAPRESSIONADA = "B THEN PRINT [ATRAS]

BACK 10

IF: TECLAPULSADA = "D THEN PRINT [DIRETA]

RIGHT 15

IF: TECLAPRESSIONADA = "I THEN PRINT

[ESQUERDA] LEFT 15

GET

END

Já que entendemos como podem ser utilizadas as variáveis, examinaremos agora quais são as técnicas que permitem utilizá-las em um programa completo.

Até agora temos sempre tomado em consideração a utilização de primitivas gráficas com constantes, como: FORWARD 10, LEFT 50, PENCOLOR 1, etc.

O LOGO permite atribuir a uma palavra valores VARIÁVEIS à condição de que o nome da variável em questão esteja precedida por “:”.

Fazendo referência ao exemplo já mencionado obtemos, portanto:

```
FORWARD :X, LEFT :Y, PENCOLOR :Z ...
```

onde as variáveis “X”, “Y”, “Z” valem, respectivamente (MAKE) 10, 50 e 1.

Estão permitidas, além dessas, seqüências de instruções do tipo:

```
MAKE “ALFA 10 MAKE “BETA 40  
PRINT :ALFA — :BETA
```

Lembrando sempre que em uma expressão figura o nome de uma variável, tem que ser colocada necessariamente diante desta o símbolo:

Se intui facilmente a grande eficácia operacional de tal funcionamento.

Por exemplo, se queremos desenhar N figuras similares, porém de diferentes dimensões, apenas devemos definir um único procedimento, capaz de trabalhar sobre PARÂMETROS VARIÁVEIS (ALTURA, LADO); depois será suficiente voltar a chamar N vezes passando os valores dimensionais apropriados para conseguir os N desenhos desejados.

Por exemplo, iremos definir um procedimento que nos permita desenhar facilmente um retângulo com as dimensões desejadas:

```
TO RETANGULO: BASE: ALTURA  
HOME; [CENTRA A TARTARUGA]
```

```

CLEARSCREEN; [LIMPA A TELA]
FORWARD: ALTURA; [LADO ESQUERDO]
RIGHT 90; [GIRA A DIREITA]
FORWARD: BASE; [BASE SUPERIOR]
RIGHT 90; [NOVO GIRO]
FORWARD: ALTURA; [LADO DIREITO]
RIGHT 90
FORWARD: BASE ;[BASE]
RIGHT 90; [TARTARUGA NA POSIÇÃO INICIAL]
END

```

Iremos chamá-lo agora dando como entradas (INPUT) os dados necessários para levar a cabo seu encargo.

Por exemplo, escrevemos: RUN RETÂNGULO 40 90 (Fig. 1 a).

RUN RETANGULO 90 40

ou finalmente, para conseguir um quadrado mediante um retângulo com altura e base iguais, RUN RETÂNGULO 90 90 (Fig. 1 d.).

Tomaremos em consideração agora a utilização das variáveis em um subprocedimento.

O programa apresentado como exemplo (TRES.RETÂNGULOS) chama três vezes o subprocedimento RETÂNGULO anteriormente definido, proporcionando-lhe as dimensões da base e da altura.

```

TO TRÊS.RETANGULO
  RETÂNGULO: BASE: ALTURA
  RIGHT 120
  RETÂNGULO: BASE: ALTURA
  RIGHT 120
  RETÂNGULO: BASE: ALTURA
  RIGHT 120
END

```

A imagem desenhada pelo programa acima citado está reproduzida na Figura 2.

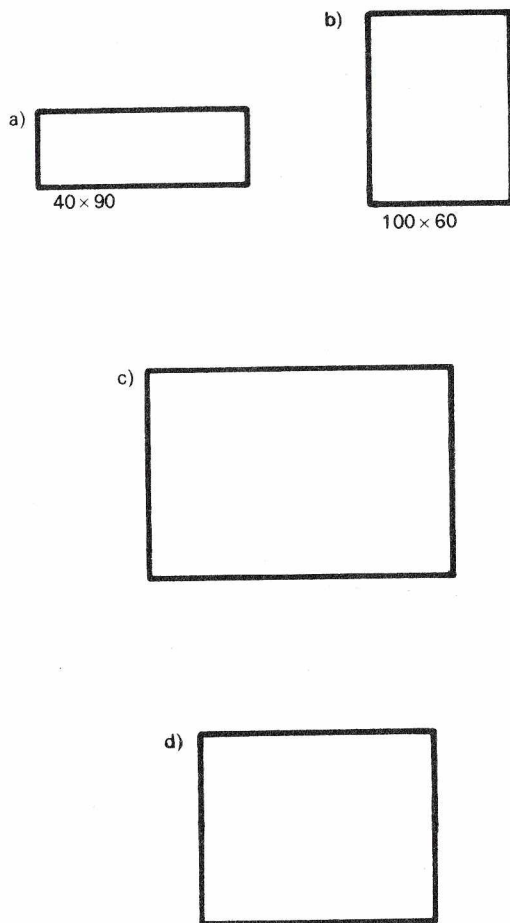


 Figura 1. — Distintos resultados de executar o procedimento RETANGULO com diversos valores.

A instrução "OUTPUT"

A função do comando "OUTPUT" desenvolveu-se unicamente em modo de programação (não é lícito utilizá-lo de maneira direta); **devolve um valor desde o procedimento que o tenha utilizado.**

A sintaxe desta instrução é a seguinte:

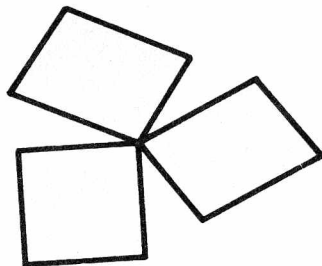


Figura 2. — Resultado do procedimento TRES.RETANGULOS.

OUTPUT CONSTANTE

Ou também

OUTPUT: VARIÁVEL

Se levarmos a cabo a execução do procedimento:

```
TO EXEMPLO
  OUTPUT 10
END
```

com RUN EXEMPLO, na tela aparecerá a mensagem:

RESULT: 10

Leva a cabo a função contrária à primitiva BUTFIRST.

Isto é porque a instrução "OUTPUT" força o valor especificado (10) desde o procedimento que tem utilizado.

Uma maneira de utilizar a instrução OUTPUT poderia ser a seguinte:

```
TO CALCULA. TANGENTE. DE: ARGUMENTO
  OUTPUT (SIN (:ARGUMENTO)/COS
    (:ARGUMENTO))
END
```

Levaremos a cabo o procedimento com:

RUN CALCULA. TANGENTE. DE 130

e o computador contestará com

RESULT: - 1.1918

Nos exemplos anteriores temos visto como o LOGO imprime na tela o valor derivado da chamada a um procedimento que utiliza a instrução OUTPUT.

É possível realizar o processo de maneira que o valor gerado por OUTPUT não seja impresso, porém sim empregado como argumento de posteriores primitivas ou procedimentos, como resulta no exemplo que segue Pressionando de maneira direta.

FORWARD 50 * CALCULA. TANGENTE. DE 140

a tartaruga se deslocará adiante um número de posições iguais a

$12 = 50 * (\text{tangente de } 140)$

No capítulo 4 veremos de quanta potência operacional resulta ser a instrução OUTPUT se são introduzidas em programas que utilizam técnicas recorrentes

Tratamento das listas e tabelas

Uma considerável parte da linguagem LOGO está orientada a permitir ao programador um domínio eficaz e simples das listas ou tabelas. Examinaremos detalhadamente este importante aspecto do LOGO para estudar depois suas possíveis aplicações.

É oportuno esclarecer, antes de tudo, o conceito da lista ou tabela: com este termo indica-se um conjunto de elementos que podem ser expressos de forma tabular. No interior de um par de colchetes (a posição que ocupa um elemento no âmbito de uma tabela deve ser contada a partir do extremo esquerdo da lista).

Por exemplo, se quiser atribuir à listagem “ALFA” o conjunto dos elementos 1,A,3F escreveremos:

MAKE “ALFA (1 A 3F)

Por conjunto de elementos entende-se, portanto, uma coleção, um grupo de números e/ou palavras, constantes e/ou variáveis. É importante sublinhar que uma listagem está vazia () se

não contém elementos.

Para definir uma listagem vazia, a introdução necessária é do tipo:

MAKE "NOMELISTA ()

Examinaremos uma por uma as primitivas dedicadas ao tratamento das listagens.

[...] = [...]

O operador "=" confronta o conteúdo da primeira lista com o da segunda, entendendo por conteúdo o conjunto dos elementos de cada um em sua ordem respectiva. Em outros termos, a listagem [a b c] é distinta [ou não igual] da listagem (c a b): ainda que contenham os mesmos elementos os dois estão caracterizados por uma ordem distinta de tabulação.

Se pressionarmos de maneira direta:

[ALFA BETA GAMA] = [ALFA BETA GAMA]

o computador contestará

RESULT : TRUE

TRUE (= é verdade) confirma a veracidade da igualdade.

Se, pelo contrário, escrevermos

[ALFA BETA GAMA] = [EPSILON GAMA]

A contestação que devemos esperar será:

RESULT : FALSE

já que a igualdade, como claramente coloca em evidência o termo inglês, é falso.

O operador "=" pode ser empregado na comparação de listas e para testemunhar a igualdade entre duas variáveis ou entre uma constante e uma variável.

É lícito escrever, por exemplo, comando do tipo:

PRINT 4 = 2*2

ou também

MAKE "X 2 PRINT 4 =:X*:X + 1
que dão como resultados TRUE e FALSE, respectivamente.
Pelo que se refere aos operadores "menor", "maior" e "distinto" valem as considerações usadas em relação à primitiva "=".

SENTENCE ARGUMENTO 1 ARGUMENTO 2

Esta instrução serve para unir, em uma lista única, os conteúdos das listas específicas (argumento 1 + argumento 2).

Por exemplo:

```
PRINT SENTENCE [ATÉ LOGO] [QUERIDO AMIGO]
```

Determina a impressão de "ATÉ LOGO QUERIDO AMIGO".

Como o operador "=" também a primitiva SENTENCE pode trabalhar com listas e com constantes ou variáveis numéricas, alfanuméricas ou alfabéticas.

Assim estão permitidas instruções do tipo:

```
SENTENCE "ADEUS [QUERIDO AMIGO]
```

(constante alfabética + listagem = lista) ou

```
SENTENCE : NOME [LISTA ELEMENTOS]
```

(variável alfanumérico + listagem = lista) ou também

```
SENTENCE 14 "É um número par
```

(constante numérica + constante alfanumérica = lista).

FIRST argumento

O argumento desta primitiva pode ser indiferentemente uma lista, uma variável ou uma constante.

A instrução FIRST devolve o primeiro elemento do argumento.

Por exemplo:

```
PRINT FIRST [ABC DEF GHI]
```

imprime na tela "ABC"
e também

```
PRINT FIRST "nos veremos
```

determinará a letra "n".

Lembre sempre que qualquer que seja o argumento de FIRST (um número, uma variável ou uma lista) esta primitiva gera sempre como saída uma lista.

Fazendo referência aos exemplos anteriores poderemos estar seguros que "ABC" e "N" são duas listas. Se estabelecemos, portanto, as seguintes igualdades:

```
[ABC] = FIRST [ABC DEF GHI]
[N] = FIRST "NOS VEREMOS"
```

teremos com segurança duas condições "TRUE".

Temos que ter em mente na fase de programação, que se o argumento da primitiva FIRST é nulo (first [], first") irá gerar um erro, com a conseqüente interrupção do procedimento que foi levado a cabo.

LAST argumento

Esta instrução **desenvolve a função oposta do comando FIRST: devolve o último dos elementos que constituem o argumento**, que pode ser de novo, indiferentemente, uma lista, uma variável (numérica, alfanumérica ou alfabética), ou uma constante (numérica, alfanumérica ou alfabética).

Utilizando os exemplos usados anteriormente para esclarecer o conceito da primitiva FIRST, poderemos agora escrever:

```
PRINT LAST [ABC DEF GHI]
```

ou também

```
PRINT LAST "NOS VEREMOS"
```

O computador contestará respectivamente: "GHI" e "S".

Também para a primitiva LAST vale as observações feitas anteriormente ao examinar o comando "FIRST": de qualquer tipo (lista, variável...) que seja o argumento, executando esta primitiva teremos sempre como resultado uma lista.

BUTFIRST argumento

BUTFIRST, termo que pode ser traduzido para o português com a expressão "exceto o primeiro", **gera como saída um valor**

igual ao argumento, porém sem seu primeiro elemento.
Pressionando de maneira direta.

```
[ABC 123] = BUTFIRST [DEF ABC 123]
```

ou

```
[MIGO] = BUTFIRST "AMIGO
```

teremos como resultado duas contestações do tipo:

```
RESULT : TRUE
```

BUTLAST argumento

Leva a cabo a função contrária a primitiva BUTFIRST

Recorrendo a instrução BUTLAST (= exceto o último) gera como saída um valor igual ao argumento, porém sem seu último elemento.

Para ter um exemplo do que foi dito anteriormente pressionaremos

```
PRINT BUTLAST [1 2 B C 3 4]
```

e também

```
PRINT BUTLAST "QUERIDÍSSIMO
```

e teremos, respectivamente, a impressão de "1. 2 B C 3" e "QUERIDÍSSIMO.

LIST argumento 1 argumento 2

Esta é uma instrução básica dedicada ao tratamento das listas.

Requer como entrada dois argumentos; a saída estará constituída por uma nova lista que conterà em sucessão os dois argumentos. Veremos um exemplo que esclarece este conceito.

Pressionaremos de maneira direta.

```
LIST [A B] [C D]
```

O computador contestará com:

```
RESULT : [[A B] [C D]]
```

Ou seja, com uma lista derivada da união dos argumentos, foi gerada uma “lista de duas listas”.

De fato `[[A B] [C D]]` está formada pelas listagens `[A B]` e `[C D]` por sua vez constituídas por elementos `(a,b)` e `(c,d)`.

Para entender a fundo a **substancial diferença existente entre a instrução LIST e a aparentemente igual SENTENCE** observe atentamente o exemplo que segue:

```
PRINT FIRST SENTNCE [A B] [C D]
```

gera

```
RESULTA :A
```

```
PRINT FIRST LIST [A B] [C D]
```

imprime

```
RESULT : [A B]
```

Como foi dito para outras primitivas de manipulação de listas, a instrução LIST admite também argumentos numéricos (exemplo: `LIST 123 456 = [123] [456]`) ou variável.

FPUT argumento 1 argumento 2

Esta introdução exige que o argumento 2 seja necessariamente uma lista.

Já observamos como a lista gerada pela primitiva LIST está constituída pelo conjunto dos argumentos. A função confiada a instrução FPUT é, sem dúvida, aparentemente similar, porém ao examiná-la menos superficialmente o resultado é notavelmente distinto.

Escreveremos de maneira direta:

```
LIST [A B C] [DEF]
```

e teremos

```
RESULT : [A B C] [D E F]
```

Se pressionarmos

```
FPUT [A B C] [D E F]
```

obteremos a impressão de

```
RESULT: [[A B C] [D E F]]
```

é fácil entender que enquanto a primitiva LIST cria uma lista com dois elementos (que por sua vez são listagens),

devolve como saída uma a nova lista cujo primeiro elemento é o ARGUMENTO 1, enquanto os restantes membros são os mesmos que constituem o ARGUMENTO 2.

Fazendo referência ao exemplo de que falamos anteriormente poderemos agora compreender porquê

```
PRINT LAST LIST [A B C] [D E F]
```

dará a mensagem RESULT : [D E F]

enquanto que as instruções

```
PRINT LAST FPUT [A B C] [D E F]
```

dão origem a lista (F).

LPUT argumento 1 argumento 2

O argumento 1 deve ser necessariamente uma lista.

Esta primitiva proporciona como saída uma lista constituída pelos elementos que formam o ARGUMENTO 1 e que tem como último membro ao próprio ARGUMENTO 2.

Tentemos utilizar estas instruções:

```
PRINT LPUT [1 2 3] [5 6 7]
PRINT LAST LPUT [1 2 3] [5 6 7 ]
PRINT FIRST LPUT [1 2 3] [5 6 7]
```

obteremos, respectivamente, as impressões:

```
[1 2 3 ] 5 6 7]
5 6 7
1
```

Comando de comprovação

A primitiva **LIST?** controla que o argumento seja de fato uma

lista; em caso afirmativo devolve a mensagem TRUE e em caso contrário FALSE (falso, o argumento não é uma lista).

Teremos, portanto:

```
LIST? [a b c] → RESULT: TRUE  
LIST? "frase → RESULT : FALSE  
LIST? 134 → RESULT : FALSE
```

Análoga é a função confiada à primitiva **WORD?**: comprova que o argumento é uma variável; nesse caso a saída é TRUE e se é o contrário devolve a mensagem FALSE.

Em correspondência com o exemplo anterior obteremos:

```
WORD? [a b c] → RESULT : FALSE  
WORD? "frase → RESULT : TRUE  
WORD? 123 → RESULT : TRUE
```

Examinaremos finalmente a primitiva **EMPTY?** A sintaxe geral é:

```
PRINT EMPTY? : NOME LISTA
```

ou também

```
PRINT EMPTY? : NOME VARIÁVEL
```

Esta instrução examina o conteúdo do argumento: se é nulo gera a mensagem TRUE, e se não, devolve o valor FALSE.

Programa resumo

Em seguida apresentamos uma listagem do programa REPLAY, que ilustra as possíveis aplicações na programação das listas.

O programa trabalha em alta resolução; permite inicialmente ao operador deslocar-se através da pressão das teclas "A" (= adiante), "B" (= atrás), "I" (= giro à esquerda) e "D" (= giro à direita) com o qual a tartaruga vai deixando uma pista na tela.

Esta é a fase que poderia ser definida como "passiva" do programa: este memoriza em uma lista todos os deslocamentos ordenados pelo usuário.

Quando desenhemos tudo o que nós havíamos proposto pressionamos a tecla "R" (replay) para colocar em marcha a segunda

parte do programa: cancela-se a página gráfica e a tartaruga, por si só, leva a cabo a mesma figura anteriormente traçada por nós.

Isto é possível graças à grande potência da linguagem: é suficiente voltar a levar a cabo (RUN) uma a uma todas as informações de nossos deslocamentos que anteriormente haviam sido memorizados em uma lista.

A listagem do programa é a seguinte (os números de linha que aparecem foram colocados, simplesmente para efeitos de facilitar a posterior explicação; portanto não devem ser editados):

```
01 TO REPLAY
02 MAKE LISTA. COMANDOS [ ] ;[APAGA A LISTA
    QUE CONTEM TODOS OS NOSSOS
    MOVIMENTOS]
03 HOME ; [CENTRA A TARTARUGA]
04 CLEARSCREEN ; [LIMPA A TELA]
05 PRINT [PRESSIONAR A, B, D, I (F = FINAL) OU R
    (REPLAY)]
06 GET
07 END
08 TO GET
09 MAKE "TECLAPRESSIONADA READCHARACTER
10 IF :TECLAPRESSIONADA = "R THEN HOME
    CLEARSCREEN EXECUTA.LISTA.COMANDOS
    NODRAW PRINT [RETORNO] STOP
11 IF TECLAPRESSIONADA = "A THEN PRINT
    [ADIANTE] FORWARD 10
    MAKE "LISTA.COMANDOS SENTENCE
    :LISTA.COMANDOS [FORWARD 10]
12 IF TECLAPRESSIONADA = "B THEN PRINT
    [ATRAS] BACK 10
    MAKE "LISTA.COMANDOS SENTENCE
    :LISTA.COMANDOS [BACK 10]
13 IF TECLAPRESSIONADA = "D THEN PRINT
    [DIREITA] RIGHT 15
    MAKE "LISTA.COMANDOS SENTENCE
```



```

:LISTA.COMANDOS [RIGHT 15]
14 IF TECLAPRESSIONADA = [ THEN PRINT
   [ESQUERDA] LEFT 15
   MAKE "LISTA.COMANDOS SENTENCE
:LISTA.COMANDOS [LEFT 15]
15 GET
16 END
17 TO EXECUTA.LISTA.COMANDOS
18 RUN SENTENCE FIRST :LISTA.COMANDOS
   FIRST BUTFIRST :LISTA.COMANDOS
19 MAKE "LIST.COMANDOS BUTFIRST
   BUTFIRST :LISTA.COMANDOS
20 IF :LISTA.COMANDOS = [ ] THEN REPLAY
21 EXECUTA.LISTA.COMANDOS
22 END

```

Tentemos entender a fundo o funcionamento do programa REPLAY que proporciona uma antecipada e breve análise, pelo que se refere à utilização da primitiva IF, da que falaremos com mais profundidade no próximo capítulo.

- LINHAS 01,02,03 e 04
Auto-explicativas
- LINHA 05
Imprime na tela a mensagem que adverte o usuário quais são as teclas para o movimento e, em que direção geram o deslocamento da tartaruga.
- LINHA 06
Salta incondicionalmente ao subprocedimento GET que inicia (TO GET) na linha 08.
- LINHA 09
A instrução READCHARACTER detém a execução do programa até que o operador pressione uma tecla.
A primitiva MAKE atribui ao argumento a letra correspondente ao deslocamento desejado.
Suponhamos agora que nossa escolha tenha sido a D (giro

à direita).

O programa leva a cabo os testes das linhas 10,11,12, porém os encontrarão todos falsos. Chega, portanto, à linha 13, que realiza o giro na direção à direita.

O IF comprova que a tecla pressionada por nós é, efetivamente "D". Imprime-se portanto, a mensagem (DIREITA) e depois a tartaruga gira 15 graus à direita (RIGHT 15). Nesta fase do procedimento GET nosso movimento memoriza-se na lista: esta função está confiada às instruções:

MAKE "LISTA.COMANDOS SENTENCE : LISTA COMANDOS (RIGHT 15)

Enquanto tenham levado a cabo as instruções que seguem ao IF volta-se a execução normal do programa: a linha 14 leva a cabo; como o teste é falso (nossa escolha é "D" e não "I") passa-se a linha 15.

Neste momento volta a aparecer o procedimento GET e o ciclo se repete. Para as linhas 11,12,14 valem as mesmas observações realizadas para a linha 13.

Se em certo ponto o usuário pressiona a tecla "R" (REPLAY) o teste da linha 10 toma o valor TRUE e as instruções seguintes, portanto, se efetuam. Centraliza-se a tartaruga e limpa-se a página de alta resolução. Depois executa-se o subprocedimento EXECUTA. LISTA. COMANDOS, cujo início está na linha 17.

Suponhamos que no momento desta chamada o : LISTA. COMANDOS tenha o valor:

[RIGHT 15 FORWARD 10 LEFT 15 FORWARD 10 RIGHT 15]

- LINHA 18

A instrução RUN executa o comando presente no argumento.

Porém este argumento não é outra coisa que o conjunto (SENTECE) dos primeiros dois elementos da listagem no suposto (RIGHT 15), obtidos com SENTENCE de FIRST e FIRST BUTFIRST, que indicam qual teria sido nosso primeiro deslocamento.

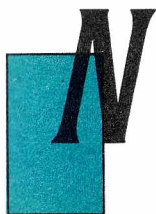
- LINHA 19

Da lista dos deslocamentos elimina-se a codificação da primeira instrução (se tirar os primeiros dois elementos).

- LINHA 20
Se LISTA. COMANDOS tem sido levado a cabo de maneira completa, o programa REPLAY inicia de novo.
- LINHA 21
Em caso contrário efetua-se a execução do deslocamento seguinte.

CAPÍTULO IV

LOOPS, COMPARAÇÕES E RECURSIVIDADE



o segundo e terceiro capítulos vimos como o intérprete LOGO permite a repetida execução de grupos específicos de instruções.

Quando necessitamos que um conjunto de comandos seja executado um número “N” de vezes podemos resolver o problema (pouco elegantemente) através da sucessiva definição de procedimentos.

No entanto, **com o LOGO temos a possibilidade de criar macro-instruções que levam a cabo uma função em um determinado número de vezes que nós especificamos a priori no momento de desenhar o programa.**

Freqüentemente é também necessário poder chamar os comandos particulares um número de vezes variável segundo os casos. Neste capítulo poderemos constatar e estudar como o intérprete LOGO nos permite resolver com simplicidade e potência o problema que tratamos.

Na segunda parte do capítulo serão examinadas as principais primitivas que nos permitem tomar decisões, enquanto que a última parte reservamos para uma fabulosa ferramenta, que permite obter resultados importantes e que é conhecida com o nome de recursividade.

A instrução “repeat”

Escrevemos o programa listado abaixo: ordenando sua exe-

cução nos pede que especifiquemos qual é o caracter que queremos imprimir na tela e o número de vezes que desejamos que se repita a fase de impressão:

TO INPUTNUMERO

```
PRINT [ESCREVE QUANTAS VEZES]
PRINT [ ]
PRINT [DEVO REPETIR A AÇÃO]
PRINT [ ]
PRINT [DE IMPRESSAO]
PRINT [ ]
PRINT [EM SEGUIDA PRESSIONAR <RETURN>]
PRINT [ ]
MAKE "NUMERO REQUEST
OUTPUT FIRST :NUMERO
```

END

TO INPUTCHARACTER

```
PRINT [PRESSIONAR O CARACTER]
PRINT [ ]
PRINT [A IMPRIMIR]
MAKE "CARATER READCHARACTER
PRINT [ ]
PRINT: CHARACTER
PRINT [ ]
```

END

TO PROGRAMA. REPETIR

```
MAKE "CHARACTER
INPUTCHARACTER
```

```
MAKE "CICLOS INPUTNUMERO  
REPEAT: CICLOS [PRINT1: CHARACTER]  
PRINT [ ]
```

END

O funcionamento do programa deveria estar claro, exceto o recurso, na antepenúltima linha, à instrução:

```
REPEAT: CICLOS [PRINT 1: CHARACTER]
```

A primitiva REPEAT efetua um grupo de instruções (Procedimentos e/ou primitivas) o número de vezes especificadas pelo argumento.

De fato, a sintaxe necessária para a utilização da instrução REPEAT requer:

1. Um número colocado imediatamente depois da palavra chave (no exemplo é a variável numérica: CICLOS) que especifica o número de vezes que deverão repetir-se as instruções seguintes.
2. Uma ou mais instruções, sempre fechadas entre um par de colchetes.

Em outras palavras, podemos repetir a execução de alguns subprocedimentos, umas simples primitivas como FORWARD, RIGHT, BACK..., uma série de instruções ou então nada menos que vários subprocedimentos correntes.

Iremos agora analisar seu funcionamento através da ajuda oferecida por uma série de programas de exemplo.

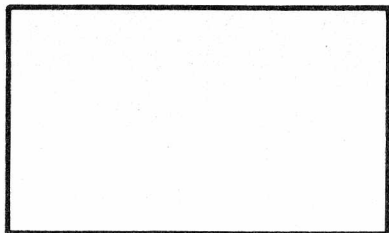
Desenharemos um retângulo (recordaremos como fizemos isto no capítulo terceiro, com um grande esforço de memória):

TO RETANGULO

Chamaremos este procedimento RETANGULO predispondo assim ao intérprete LOGO para que o aceite como uma palavra nova do vocabulário; depois pressionaremos

```
REPEAT 2 [FORWARD 30 LEFT 90 FORWARD 50 LEFT 90]
```

A instrução REPEAT tem como parâmetro o número de ve-



 *Figura 1. — Retângulo obtido mediante a instrução REPEAT.*

zes que a lista de instruções deve repetir (2 vezes no nosso caso) e estas mesmas que, em ordem, são: FORWARD 30, LEFT 90, FORWARD 50, LEFT 90.

Lançaremos agora o programa e entenderemos o que é mostrado na Figura 1.

Existe um método ainda mais correto para obter o mesmo resultado:

Definir um subprocedimento do tipo:

```
TO ARESTA
  FOWARD 30
  LEFT 90
  FORWARD 50
  LEFT 90
END
```

e logo chamá-lo desta maneira:

```
TO RETANGULO
  REPEAT 2 [ARESTA]
END
```

Iremos agora construir outro procedimento que utilize REPEAT e por sua vez o subprocedimento RETANGULO:

```
TO JANELA
  REPEAT 2 [RETANGULO FORWARD 30]
  LEFT 90
  FORWARD 25
  LEFT 90
```



```
FORWARD 60  
END
```

Obteremos o que se vê na Figura 2.

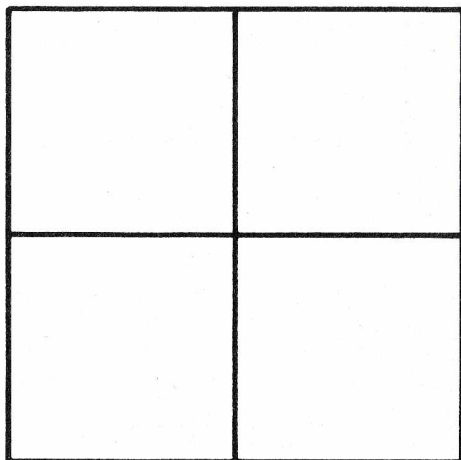


Figura 2. — Chamadas repetidas ao RETANGULO.

Agora vejamos como conseguir uma estrela (Fig. 3):

```
TO ESTRELA
```

```
  REPEAT 18 [FORWARD 100 RIGHT 140]
```

```
END
```

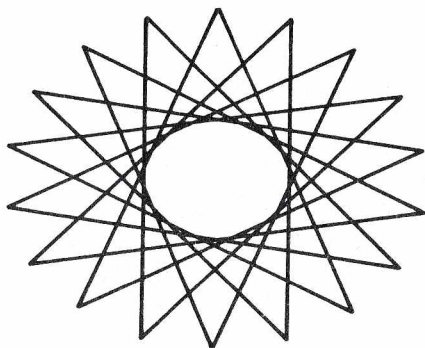


Figura 3. — Outras possibilidades da instrução REPEAT.

E, para acabar, uma espiral:

TO ESPIRAL: RAI0

```
REPEAT 30 [MAKE "RAIO :RAIO + 5  
FORWARD :RAIO RIGHT 90]
```

END

Para ver os resultados (Fig. 4) pressionar:

ESPIRAL 10

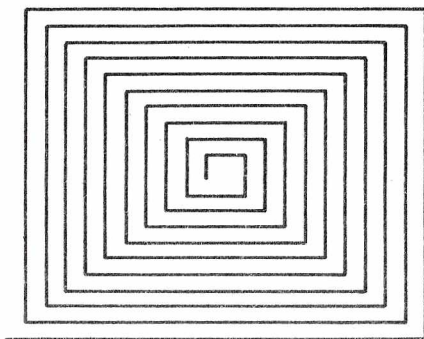


Figura 4. — União de REPEAT e variáveis de entrada.

No procedimento ESPIRAL aparece outra vez a utilização de uma variável. Estas são as artífices de uma programação eficaz, já que, permitindo a variação no interior de um ciclo dos parâmetros essenciais, nos deixam trocar notavelmente as características do que fazemos sem ter que recorrer a utilização de novas versões do programa.

Já vimos no capítulo anterior a utilização das variáveis nos procedimentos e subprocedimentos: vejamos agora como “explorá-las” melhor em relação a instrução REPEAT.

Seja QUADRADO um procedimento simples que traz a figura homônima:

TO QUADRADO :LADO

```
REPEAT 4 [FORWARD: LADO RIGHT 90]
```

END

Para executá-lo terá que dar entrada não só do nome do procedimento, mas também o valor inicial das variáveis associadas. Em outras palavras: se desejamos que o quadrado tenha como lado 10..20..30.. deveremos escrever:

```
RUN QUADRADO 10  
RUN QUADRADO 20
```

e assim seguidamente (no capítulo 3 deveria ter ficado esclarecido este conceito).

Desta maneira é simples traçar figuras similares com um só programa em que tenha sido introduzida de maneira oportuna uma variável e, eventualmente, construir um procedimento de controle com os decréscimo ou incrementos desejados (como fizemos em ESPIRAL).

De forma particular veremos como incrementar uma variável em um ciclo dominado por REPEAT para desenhar uma série de figuras “concêntricas” (Fig.5):

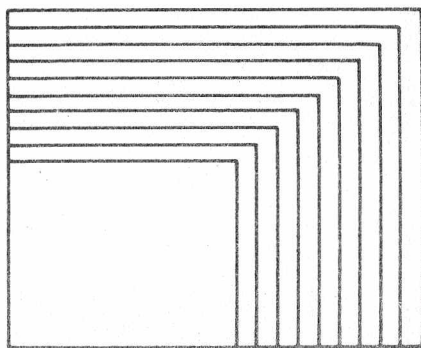


Figura 5. — Combinação de REPEAT e MAKE com variáveis.

A instrução MAKE, como já sabemos, permite efetuar operações com as variáveis (em nosso caso, uma soma) durante a execução do programa.

```
TO QUADRADO :LADO  
  REPEAT 4 [FORWARD :LADO RIGHT 90]
```

```

END
TO FIGURAS. CONCENTRICAS :DIAMETRO
  REPEAT 10 [MAKE "DIAMETRO :DIAMETRO + 10
    QUADRADO :DIAMETRO]
END

```

Finalmente estudaremos uma interessante série de procedimentos que estão em correlação e que desenharam os triângulos da Figura 6.

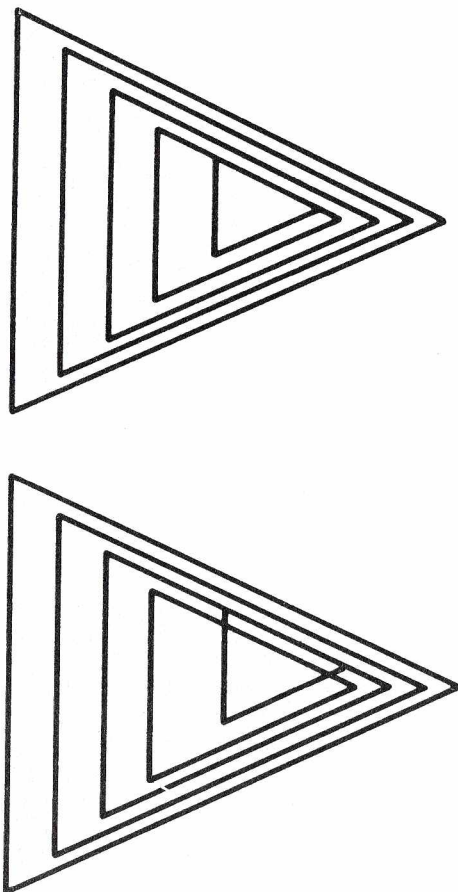


Figura 6. — Dois exemplos do procedimento CINCO. TRIANGULOS.

Pressionando por exemplo, RUN CINCOTRIANGULOS 10 e RUN CINCOTRIANGULOS 30 teremos, sucessivamente, as figuras:

```
TO TRIANGULO :LADO  
  REPEAT 3 (FORWARD :LADO RIGHT 120)  
END
```

```
TO CINCO. TRIANGULOS :LADO
```

```
  REPEAT 5 (MAKE "LADO :LADO + 40  
  TRIANGULO :LADO ALFA :LADO)  
END
```

```
TO ALFA :LADO
```

```
  PENUP  
  HOME  
  LEFT 90  
  FORWARD :LADO / 2  
  LEFT 90  
  FORWARD :LADO / 2  
  RIGHT 180  
  PENDOWN
```

```
END
```

Comparações e bifurcações

A linguagem LOGO permite ao programador efetuar controles do tipo decisivo sobre os valores das variáveis, tanto numéricas como literais.

Para compreender qualitativamente qual é o significado e a função das instruções de teste, tomemos em consideração um exemplo, aparentemente banal, que encerra em si, ao examiná-lo menos superficialmente, vários pontos de reflexão. Suponhamos que estamos viajando por uma autopista com o nosso carro; em certo momento do percurso aparece a nossa frente uma bifurcação na estrada com suas respectivas placas indicadoras (Fig. 7). Se bem que para nós está claro qual é o caminho, nos vemos obrigados a escolher uma direção a tomar, conseqüentemente, o caminho mais oportuno.

Se a nossa meta é, por exemplo, a cidade de Madri, entre as

duas direções (que levam uma a Zaragoza e a outra a Madrid) escolheremos a segunda alternativa.

Qual a operação feita pela nossa mente no momento em que fomos obrigados a tomar a decisão? O processo terá sido, mais ou menos, o seguinte:

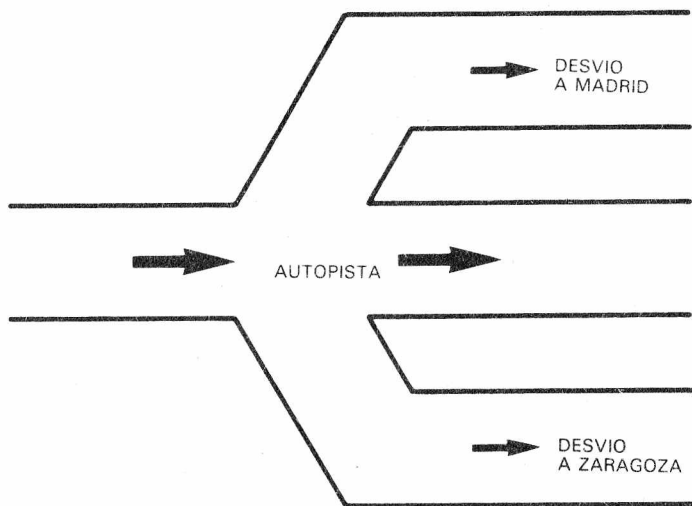


Figura 7. — Uma simples desviada no caminho.

- Se a direção que quero (Madri) = indicação da estrada, ENTÃO pego o desvio correspondente.
- Se a direção que quero (Madri) não = a indicação da estrada, ENTÃO sigo direto.

Em outras conclusões: confrontamos o que queremos com o que vemos.

Da mesma maneira atua o intérprete LOGO no momento de tomar uma decisão.

Se desejamos que certo procedimento ou subprocedimento seja interrompido em um certo momento se foi verificada determinada condição, por exemplo no momento em que a variável A toma o valor 100, teremos que escrever:

```
IF :A = 100 STOP
```

Em consequência, cada vez que o procedimento chegue a esta instrução iremos operar uma comparação entre o valor real e a constante 100; só no caso em que a igualdade seja verdadeira (:A = 100) efetua-se a instrução STOP (ou qualquer outro comando especificado).

No exemplo anterior foi utilizado o operador de igualdade ("="), porém existem outros que nos permitem uma casuística muito completa.

Apresentamos seguidamente na tabela 1 os operadores de relação e as funções que levam a cabo.

Voltando a falar da utilização da instrução IF, sua forma mais simples é:

IF (expressão booleana) THEN (instrução-é-)

Iremos agora analisar detalhadamente a expressão:

1. A primitiva "IF" adverte ao intérprete LOGO que deve ser efetuado uma comparação.
2. Por expressão booleana entende-se qualquer tipo de com-

OPERADOR	SIGNIFICADO
=	IGUAL
>	MAIOR
<	MENOR
>=	MAIOR OU IGUAL
<=	MENOR OU IGUAL
<>	DISTINTOS

Tabela 1. — Operadores de relação e comparações que desempenham.

paração válida ($a = b, a < b, a > b, a < b, \dots$); o termo “booleana” toma seu nome do matemático George Boole. O resultado de uma expressão de Boole pode tomar exclusivamente os valores verdadeiro ou falso (TRUE OU FALSE); por exemplo, se $a = 1$ e $b = 1$, então $a = b$ é “verdadeiro”, porém se $a = 1$ e $b = 2$, $a = b$ é “falso”.

3. O comando “THEN” indica que desenvolvem-se as operações que seguem imediatamente depois exclusivamente no caso de que o resultado da expressão booleana seja “verdadeira”.
4. O termo (instrução-é-) substituirá pela série de operações que nós queremos que levem a cabo no caso de que a expressão booleana seja verdadeira.

Esclarecemos estes conceitos por meio do seguinte exemplo (programa “INTERSECCÕES”). Suponhamos que queremos girar uma figura até o momento em que um contador supere um determinado valor numérico que nós escolhemos igual a 100.

```
TO INTERSECCOES :CONTADOR :GIRO
```

```
  IF :CONTADOR > 100 THEN PRINT “FINAL  
  DA EXECUCAO STOP  
  ROTACAO :GIRO  
  INTERSECCOES ( :CONTADOR + 10) :GIRO
```

```
END
```

```
TO ROTACAO :GIRO
```

```
  QUADRADO  
  LEFT :GIRO
```

```
END
```

```
TO QUADRADO
```

```
  REPEAT 4 [FORWARD 60 RIGHT 90]
```

```
END
```

O programa chama sucessivamente a três subprocedimentos (INTERSECCOES, ROTACAO E QUADRADO) que traçam uma série de quadrados que ao girar alguns com relação aos outros geram o efeito que é observado na Figura 8.

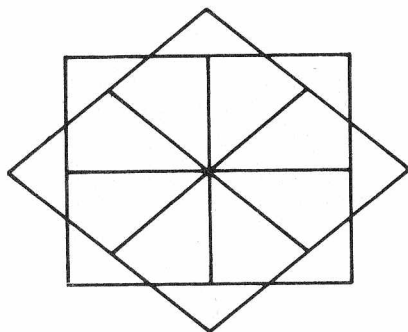


 Figura 8. — Efeito conseguido ao desenhar vários quadrados girados.

A instrução

```
IF CONTADOR > 100 THEN PRINT "FINAL DA EXECUÇÃO"
STOP
```

pára o programa, que de outra maneira seguirá até o infinito.
Estudaremos outro exemplo:

```
TO CHAMADA :PASSO :COMPRIMENTO
```

```
    LEFT 90
    DESENHA :PASSO :COMPRIMENTO
END
```

```
TO DESENHA :PASSO :COMPRIMENTO
    FORWARD 1
    MAKE "COMPRIMENTO :COMPRIMENTO - 1 - :passo
    PENUP
    FORWARD :PASSO
    PENDOW
    IF :COMPRIMENTO < 1 THEN STOP
    DESENHA :PASSO :COMPRIMENTO
END
```

O programa "DESENHA :PASSO :COMPRIMENTO" exige como dados o passo gráfico (a distância entre uma linha e a seguinte) e o comprimento total do desenho.

Graças a utilização do operador de relações "<" a execução do procedimento termina no momento em que a medida do gráfico a traçar toma um valor inferior a 1.

Abaixo apresentamos três tabelas (2,3 e 4) que mostram as combinações possíveis entre os fundamentais operadores de relações e os resultados das aplicações a duas variáveis; a primeira linha de cada tabela constitui a hipótese em base da qual (através dos mesmos operadores) se chega a obter um resultado do tipo booleano (verdadeiro ou falso). Por exemplo, na tabela 2, onde supomos $A = B$, a relação $A > B$ será falsa, no entanto na tabela 3 será verdadeira.

Uma vez entendidos os fundamentos da instrução IF podemos passar agora a ver outra expressão possível.

Escreveremos na seguinte forma e tentaremos compreender as diferenças fundamentais com o anteriormente dito:

IF (EXPRESSÃO BOOLEANA) THEN (Instrução 1) ELSE (instrução 2)

SE $A = B$	
$A = B$	VERDADEIRO
$A > B$	FALSO
$A < B$	FALSO
$A \geq B$	VERDADEIRO
$A \leq B$	VERDADEIRO
$A \neq B$	FALSO

Tabela 2. — Resultados possíveis supondo $A = B$.

SE $A > B$	
$A = B$	FALSO
$A > B$	VERDADEIRO
$A < B$	FALSO
$A \geq B$	VERDADEIRO
$A \leq B$	FALSO
$A \neq B$	VERDADEIRO

Tabela 3. — Partindo de $A > B$ os resultados de combinar os operadores lógicos de relação, são os mostrados.

SE $A < B$	
$A = B$	FALSO
$A > B$	FALSO
$A < B$	VERDADEIRO
$A \geq B$	FALSO
$A \leq B$	VERDADEIRO
$A \neq B$	VERDADEIRO

Tabela 4. — A última hipótese é $A < B$.

- Se a expressão booleana é verdadeira (vale TRUE) levaremos a cabo então a instrução 1 e iremos ignorar a instrução 2;
- Se a expressão booleana é falsa (vale FALSE) levaremos a cabo então a instrução 2 e iremos ignorar a instrução 1.

É evidente que o grupo IF...THEN...ELSE permite uma maior e melhor estruturação do programa.

Na Figura 9 encontramos o diagrama do fluxo para instrução IF...THEN...ELSE.

O procedimento que segue demonstra uma aplicação possível da instrução "ELSE".

O programa pergunta dois números e imprime uma mensagem segundo sua relação (que o primeiro seja maior que o segundo, os dois iguais ou o primeiro é menor que o segundo).

```

TO PROG
  CLEARTEXT
  INSTRUÇÕES ;[ESCREVA AS INSTRUÇÕES PARA
    O USUÁRIO]
  MAKE "X INPUTNUMERO ; [PRIMEIRO NUMERO]
  INSTRUÇÕES
  MAKE "Y INPUTNUMERO ; [SEGUNDO NUMERO]
  IF :X = :Y THEN CASO. IGUAIS ELSE IF
    :X > :Y THEN CASO. MAIOR ELSE
    CASO. MENOR

```

```

END

```

```

TO INPUTNUMERO

```

```

  OUTPUT FIRST (REQUEST)

```

```

END

```

```

TO CASO. IGUAIS

```

```

  PRINT"

```

```

  PRINT [O PRIMEIRO NUMERO]

```

```

  PRINT"

```

```

  PRINT [E IGUAL AO SEGUNDO]

```

```

  PRINT"

```

```

END

```

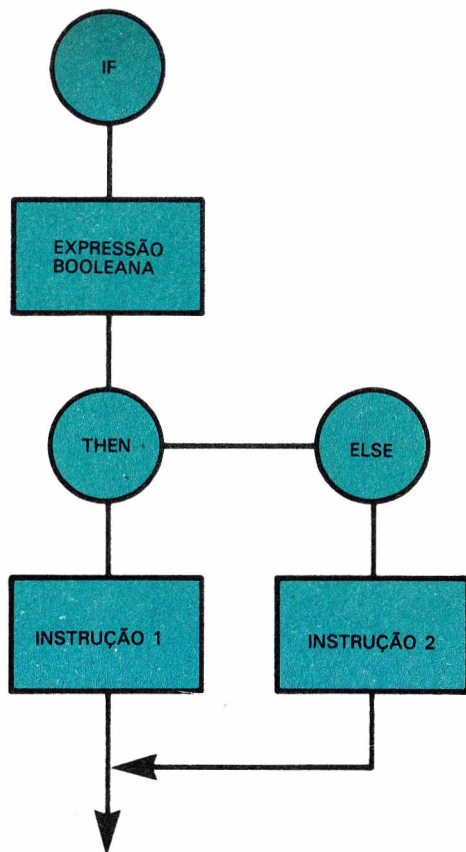


Figura 9. — Diagrama de blocos da instrução IF...THEN...ELSE.

```
TO CASO. MENOR
  PRINT"
  PRINT [O PRIMEIRO NÚMERO]
  PRINT"
  PRINT [E MENOR QUE O SEGUNDO]
  PRINT"
```

```
END
```

```
TO CASO. MAIOR
PRINT"
PRINT [O PRIMEIRO NUMERO]
PRINT"
PRINT [E MAIOR QUE O SEGUNDO]
PRINT"
```

END

TO INSTRUÇÕES

```
PRINT"
PRINT [ESCREVA UM NUMERO]
PRINT"
PRINT [EM SEGUIDA PRESSIONAR A TECLA DE]
PRINT"
PRINT [RETURN]
PRINT"
```

END

Seria um bom exercício que fariam vocês melhorarem os sub-procedimentos de saída de mensagens aproveitando a grande zona comum que tem.

TEST...IFTRUE...THEN...ELSE

Suponhamos que devemos calcular uma simples soma e que queremos chamar a um subprocedimento somente no caso particular em que esta soma seja igual a um valor anteriormente fixado.

Para conseguir isto podemos recorrer às instruções:

```
TESTE : A + B = :D
IFTRUE THEN SUBPROCEDIMENTO
```

Esclarecemos agora com a ajuda de um exemplo (procedimento VERDADEIRO. FALSO) quais são as distintas possibilidades da instrução IFTRUE.

Este procedimento pede quatro valores numéricos e arbitrários (x,y,z,s).

No caso em que a soma "X + Y + Z" vale "S", o programa im-

prime a mensagem “é verdadeira”; em caso contrário, teremos como saída “é falso”:

```
TO VERDADEIRO. FALSO
  MAKE "X INPUTNÚMERO
  MAKE "Y INPUTNÚMERO
  MAKE "Z INPUTNÚMERO
  MAKE "S INPUTNÚMERO
  MAKE "T :X + :Y + :Z
  TEST :T = :S
  IFTRUE THEN PRINT [E VERDADEIRO] ELSE
  PRINT [E FALSO]
END

TO INPUTNÚMERO
  OUTPUT [REQUEST]
END
```

TEST...IFFALSE...THEN...ELSE

Ao que se refere esta instrução, o tema é equivalente a IFTRUE, considerando, no entanto, as hipóteses ao contrário. Em outras palavras, o comando TEST verifica se a comparação é verdadeira ou falsa, porém as instruções que seguem a primitiva THEN levam a cabo só se o resultado do teste for falso. O programa anterior, portanto, seria modificado desta maneira:

```
TO VERDADEIRO. FALSO
  MAKE "X INPUTNÚMERO
  MAKE "Y INPUTNÚMERO
  MAKE "Z INPUTNÚMERO
  MAKE "S INPUTNÚMERO
  MAKE "T :X + :Y + :Z
  TEST :T = :S
  IFFALSE THEN PRINT [E FALSO] ELSE
  PRINT [E VERDADEIRO]

END

TO INPUTNÚMERO
  OUTPUT (REQUEST)
END
```

HEADING

Trata-se de um comando muito, particular. Consiste em uma medição da orientação angular da plumilha ou tartaruga (turtle) a respeito de sua direção inicial (olhando para a parte superior da tela).

Se, por exemplo, pressionamos:

```
HOME  
RIGHT 15  
PRINT HEADING
```

Obteremos como contestação

15

O valor gerado pela instrução HEADING é do tipo numérico e, portanto, é possível confrontar com qualquer outra (variável ou constante) através da acostumada relação do tipo:

IF HEADING (Operador relacional) e THEN...

"ANYOF" e "ALLOF"

A sintaxe desta instrução, relativamente sofisticada é:

ANYOF (expressão. booleana 1) (expressão booleana 2)

e

ALLOF (expressão booleana 1) (expressão booleana 2)

Portanto, ambas as instruções requerem como parâmetros duas expressões booleanas.

Em base ao anteriormente visto seguramente intui que este tipo de relação proporciona um resultado somente do tipo FALSE ou TRUE.

As duas primitivas em questão, segundo o que valem os argumentos (expressão 1 e expressão 2), geram a mensagem TRUE ou FALSE. A tabela 5 mostra todas as combinações possíveis.

A primeira coluna está constituída por resultados da expressão 1; a segunda, pelos relativos à expressão 2, e a terceira e a

quarta são os valores obtidos pelas primitivas ANYOF e ALLOF, respectivamente. Como veremos equivalem, simplesmente, aos operadores lógicos OR e EXOR.

A recursividade

O termo "RECURSIVIDADE" indica o processo lógico pelo qual um processo em execução chama a si mesmo.

EXPRES. 1	EXPRES. 2	ANYOF	ALLOF
FALSO	FALSO	FALSO	FALSO
FALSO	TRUE	TRUE	FALSO
TRUE	FALSO	TRUE	FALSO
TRUE	TRUE	TRUE	TRUE

O procedimento elementar:

TO EXEMPLO. RECURSIVIDADE

FORWARD 50

LEFT 120

EXEMPLO. RECURSIVIDADE

END

é um claro exemplo do que foi dito anteriormente.

É fácil ver que um programa destas características procede até o infinito; não existe, de fato, nenhuma condição para parar a execução.

Provavelmente o mais imediato e entusiasta campo de aplicação da técnica da recursividade é o aspecto gráfico (além de alguns cálculos matemáticos).

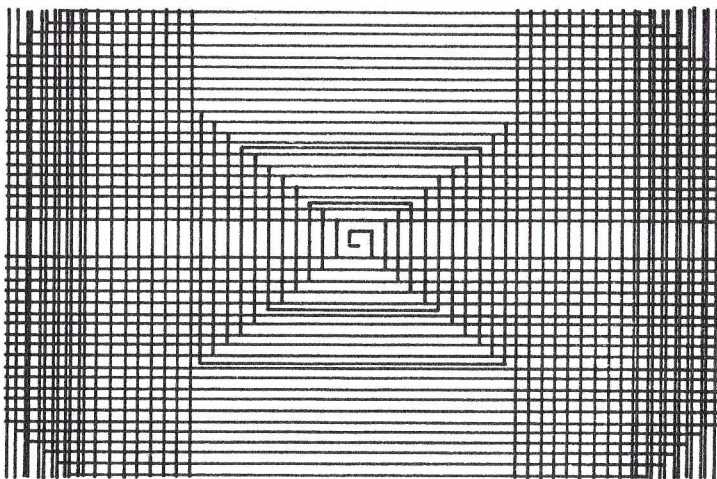
Pressionemos a breve seqüência seguinte (os números anteriores às instruções, não devem ser escritos; servem exclusivamente para poder comentar mais agilmente o funcionamento do programa)

```
01 TO DESENHA. ESPIRAL
02 FORWARD :LADO
03 RIGHT 90
04 ESPIRAL :LADO + 3
05 END
```

e a executamos com

```
RUN DESENHA. ESPIRAL1
```

A imagem que aparece é a mostrada na Figura 10.



Estudemos linha por linha o funcionamento do programa em questão.

- LINHA 01

Está constituída pela declaração do procedimento

ESPIRAL e do parâmetro variável :LADO, sobre o qual trabalhará.

- LINHA 02

A instrução FORWARD :LADO faz com que a tartaruga vá até adiante um número de posições igual ao valor da variável :LADO

- LINHA 03

Gira a tartaruga 90 graus na direção a esquerda, determinando as arestas da espiral.

- LINHA 04

Neste momento efetua-se o processo de recursividade: chama-se ao procedimento DESENHA. ESPIRAL passando-o ao valor atual de :LADO (igual a :LADO + 3).

O ciclo inicia de novo a partir do ponto (1) sem terminar nunca.

O leitor seguramente terá percebido que um possível sistema para interromper a execução de um programa recursivo poderia ser recorrido às primitivas IF...THEN...

Estudemos o seguinte programa, um pouco mais difícil que os anteriores, que adota esta técnica (recurso + IF...) para calcular um número fatorial.

```
01 TO FAT :X
02 FATORIAL :X :X
03 END
04 TO FATORIAL :X :Y
05 IF :Y = 0 THEN OUTPUT (1)
07 OUTPUT :X * (FATORIAL :X-1 :Y-1)
08 END
```

Por fatorial de um número inteiro positivo “n” define-se matematicamente o produto dos números inteiros sucessivos a partir de 1 até “n”.

Por exemplo, o fatorial de 3 é:

$$1*2*3=6$$

e o fatorial de 6 é:

$$1*2*3*4*5*6=720$$

A regra anteriormente descrita admite uma exceção: por definição, o fatorial de zero é um (o programa FAT leva também em conta isso).

Para executar o procedimento atua-se dando o acostumado RUN, acompanhado pelo número do qual queremos que seja calculada a função fatorial.

Por exemplo escrevemos:

RUN FAT 3

e teremos o valor ($3*2*1$):

RESULT :6

Vejamos atentamente como funciona o programa (repetiremos de novo que as linhas foram numeradas somente com a finalidade de facilitar a busca das instruções; estes números não fazem parte do programa e, portanto, não devem ser pressionados) seguindo o processo de sua execução para o exemplo anterior.

- LINHA 01

Define o procedimento FAT; a variável "X" está colocada, com referência ao exemplo, no valor 3.

- LINHA 02

Leva-se a cabo um salto incondicional em direção ao subprocedimento FATORIAL, ao qual é passado um duplo valor (:X:X).

- LINHA 04

A variável "X" e a "Y" são colocados ao valor numérico 3.

- LINHA 05

"Y" vale 3: as instruções que seguem ao teste são ignoradas.

- LINHA 06

Deve-se mandar como saída o resultado de $X * \text{FATORIAL de } (X-1)$: De fato, o fatorial de 3 é igual a 3 ($= X$) pelo fatorial de 2 ($= X-1$).

Neste momento o computador deverá calcular quanto vale o fatorial de 2, que é a operação que fica pendente; chama-se então pela recursividade no subprocedimento FATORIAL passando o valor $X-1 (= 2)$ e o $Y-1 (= 2)$.

A execução do programa volta a linha 04.

- LINHA 05
As instruções posteriores ao IF são ignorados, já que o teste é TRUE ($Y = X = 2 < > 0$).
- LINHA 06
O fatorial de $X (= 2)$ pode ser visto como X pelo fatorial de $X-1 (= 1)$; é o segundo nível de recursividade: chama-se outra vez o subprocedimento FATORIAL passando o valor $X-1 (= 1)$.
- LINHA 05
O teste não está contudo satisfeito (se o mecanismo foi entendido compreenderão que o teste será TRUE, no seguinte passo).
- LINHA 06
O fatorial de 1 (X) é 1 pelo fatorial de 0 ($X-1$).
É o terceiro nível de recursividade; outra vez uma chamada ao subprocedimento FATORIAL com “X” e “Y” que valem, por fim, 0.
- LINHA 05
Verifica-se o teste: na saída é gerado o valor 1.
A seguinte instrução levada a cabo é o END da linha 7.
- LINHA 07
Neste ponto o computador pode calcular por fim o produto, anteriormente procurado, entre 2 e o fatorial de 2-1; o resultado é $2*1 = 1$.
Este valor permite o cálculo do fatorial de $3 = 3* \text{fatorial de dois} = 3*2 = 6$.
- LINHA 03
O programa pode assim finalmente, encerrar sua execução.
O leitor que encontrou algum problema ao seguir a li-

na de raciocínio, deve repetí-lo lembrando da ilustração na Figura 10.

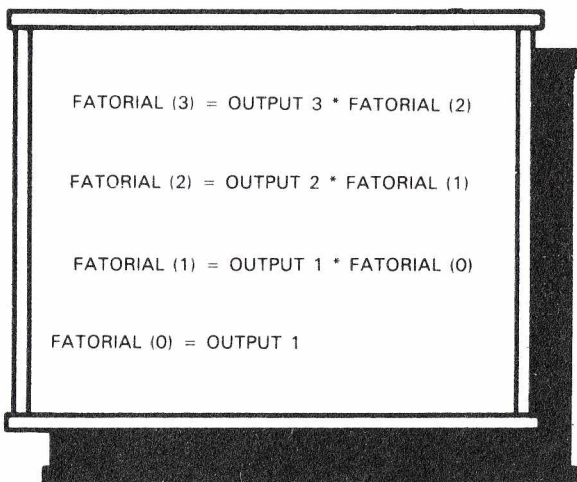


Figura 10. — Esquema para seguir o processo de cálculo do fatorial de 3.

Programas resumidos

O programa TRAJETORIA simula o movimento de um projétil do qual deve ser proporcionado o valor da velocidade inicial, o ângulo de inclinação inicial com respeito ao eixo "x" e o efeito da gravidade.

TO TRAJETORIA: OMEGA: VELOCIDADE: ALFA

FORWARD :VELOCIDADE/10

RIGHT :ALFA

MAKE "ALFA :ALFA + 0.9

IF :ALFA > :OMEGA THEN MAKE "ALFA
:ALFA-1

MAKE "VELOCIDADE :VELOCIDADE-1

IF HEADING > 180 THEN STOP

TRAJETORIA : OMEGA: VELOCIDADE: ALFA

END

TO SIMULA :OMEGA :VELOCIDADE :ALFA

SINGLECOLOR

HOME

CLEARSCREEN

PENUP

LEFT 90

FORWARD 130

RIGHT 180

LEFT :OMEGA

PENDOWN

TRAJETORIA : OMEGA: VELOCIDADE :ALFA

END

Introduzindo os dados 45 50 0 (ou seja, RUN SIMULA 45 50 0) é obtido o resultado da Figura 11.

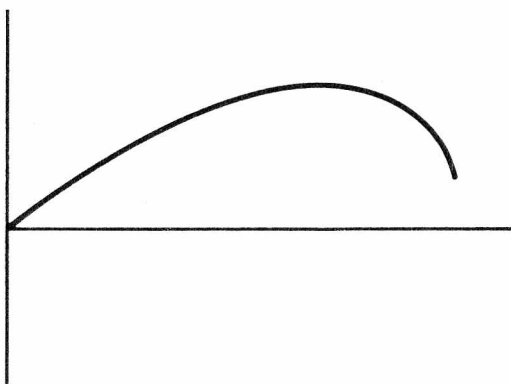


Figura 11. — Trajetória do projétil com os valores 45, 50 e 0.

Seguidamente incluímos as listagens de alguns procedimentos que sintetizam o que estudamos até agora:

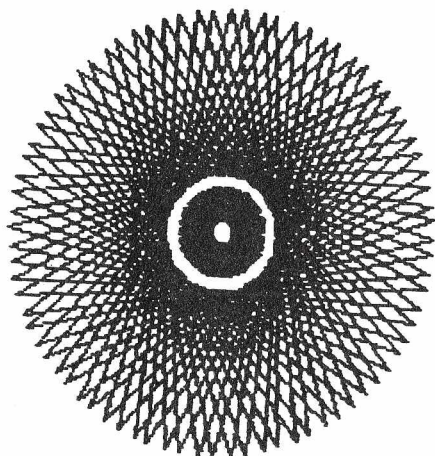


Figura 12. — Uma das figuras obtidas com estes procedimentos.

Para executá-los utilize o comando RUN acompanhado, se é necessário, pelos dados de entrada.

PROGRAMA "BORDADO"

TO BORDADO

HOME

CLEARSCREEN

ALFA 10 20 30

END

TO ALFA :LADO :ANGULO :PASSO

BACK :LADO

LEFT :ANGULO

ALFA :LADO (:ANGULO + :PASSO) :PASSO

END

PROGRAMA "ROSAO"

TO ROSAO

MAKE "LADO 110
POLIARESTA :LADO

END

TO POLIARESTA :LADO

RIGHT :LADO
REPEAT :LADO/3 [ARESTA :LADO]

END

TO ARESTA :LADO

FORWARD :LADO
RIGHT 170
FORWARD :LADO

END

PROGRAMA "INTEGRAL"

TO INTEGRAL

HOME
CLEARSCREEN
PENUP
FORWARD 40
PENDOWN
RIGHT 60
BETA 20

END
TO BETA :ANGULO
FORWARD 15
RIGHT :ANGULO
BETA: ANGULO + 5
END

PROGRAMA "PETALAS"

TO PETALAS
HOME
LEFT 90
FORWARD 30
RIGHT 90
FORWARD 40
CLEARSCREEN
REPEAT 12 [RIGHT 120 ARCO 1]
END

TO ARCO :PASSO
FORWARD :PASSO
RIGHT 6
IF :PASSO < 15 ARCO :PASSO + 1
END

PROGRAMA "FLOR"

TO CAULE :X
BACK 17
LEFT :X
IF :X < 20 THEN CAULE:X + 2
END

TO FLOR

HOME

FORWARD 30

CLEARSCREEN

LEFT 20

REPEAT 3 [MAKE "CONTADOR 1 PETALA 1]

RIGHT 40

CAULE 1

END

TO PETALA :START

FORWARD 10

RIGHT :START

IF :START < 20 THEN PETALA : START + 2

MAKE "CONTADOR : CONTADOR + 1

IF CONTADOR > 2 THEN STOP

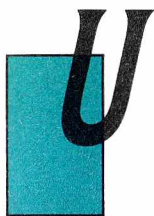
PETALA 1

END

CAPÍTULO V

OPERAÇÕES NUMÉRICAS

Representação binária



um conjunto das primitivas da linguagem está inteiramente dedicada à manipulação da informação numérica.

Vamos esclarecer antes de tudo o significado deste termo, ainda que possam recorrer a volumes anteriores da B.B.I. (especialmente o 1) para mais detalhes. Sabemos que nosso computador está preparado para resolver uma notável quantidade de cálculo, porém nem todos nós conhecemos a técnica que a máquina utiliza, para representar em seu interior a informação numérica. Por causa das limitações da eletrônica convencional **todas as informações** (também as não numéricas) **estão codificadas em grupos de bits**.

Um BIT é a mínima unidade de informações; pode estar exclusivamente em dois estados: ativado (nível 1) ou desligado (nível 0). Mediante o agrupamento de bits podemos codificar diversos tipos de informações.

Um número decimal, portanto, será representado pelo computador como uma coleção de “n” bits.

Estudemos agora um algoritmo que nos permita também levar a cabo a conversão a partir de um valor numérico (para simplificar a coisa, inteiro e positivo) ao correspondente equivalente binário.

Tomaremos como exemplo o número decimal 131. O processo de conversão será:

Dividir por 2.

131/2 - cociente 65 resto 1

Dividimos de novo o cociente que obtivemos por 2: $65/2 =$ cociente 32 resto 1

Procedamos igual até obter um cociente ZERO.

$32/2 =$ cociente 16 resto 0

$16/2 =$ cociente 8 resto 0

$8/2 =$ cociente 4 resto 0

$4/2 =$ cociente 2 resto 0

$2/2 =$ cociente 1 resto 0

$1/2 =$ cociente 0 resto 1

O cociente é nulo e, portanto, podemos terminar as operações.

Neste momento transcrevemos em ordem inversa (a partir do final ao princípio) os restos das sucessivas divisões, obtendo:

1 0 0 0 0 0 1 1

Pois bem, esta é a representação binária do número decimal 131.

Operadores lógicos binários

A linguagem LOGO, como dissemos anteriormente, dispõe de instruções preparadas para o tratamento da informação numérica binária; iremos estudá-las atentamente.

BITAND Argumento 1 Argumento 2

Efetua a função lógica END entre os argumentos. Em outros termos, compara o formato binário dos dois valores e sobre a base da tabela verdade representada na tabela 1 gera o valor binário apropriado.

Por exemplo, pressionemos de maneira direta:

PRINT BITAND 244 35

tentaremos compreender por que o resultado é

RESULT :32

Argumento 1	Argumento 2	Resultado
0	0	0
0	1	0
1	0	0
1	1	1

 Tabela 1. — Função lógica AND.

Convirtamos o número decimal 244 a binário, façamos o mesmo com o 35 e coloquemos um debaixo do outro como segue:

Decimal 244 = binário	1	1	1	1	0	1	0	0
Decimal 35 = binário	0	0	1	0	0	0	1	1

Examinemos o primeiro bit (pela esquerda) do número 244 (vale 1) e realizemos a operação AND (fazendo referência a sua tabela verdade) com o primeiro bit do número 35 (vale 0).

A linha da tabela 1 que nos interessa é, portanto, a terceira: as entradas que teremos são: 1 e 0, como saída obteremos um bit desativado (vale 0).

Repetiremos o mesmo raciocínio para os 7 bits restantes dos dois números:

Bit n.º	Número 1	Número 2	Resultado
2	1	0	0
3	1	1	1
4	1	0	0
5	0	0	0
6	1	0	0
7	0	1	0
8	0	1	0

A quarta coluna recolhe os valores que são derivados das sucessivas operações.

Esta é a representação binária do resultado procurado por nós (BITAND 244 35).

Coloquemos de forma horizontal os bits obtidos na última coluna, acrescentando o do bit número 1:

0 0 1 0 0 0 0 0

Veremos agora como podemos converter este número binário obtido no correspondente decimal, que, como esperamos terá que ser 32.

O algoritmo a seguir é o seguinte: volta-se a escrever em ordem inversa a sequência de bits encontrada, obtendo:

0 0 0 0 0 1 0 0

Recolhemos então cada bit, procedendo desde a esquerda à direita, e o multiplicamos por potências crescentes de dois, começando pela potência "zero" ($2^0 = 1$).

BIT	VALOR	RESULTADO
PRIMEIRO	0	$0 * (2^0) = 0$
SEGUNDO	0	$0 * (2^1) = 0$
TERCEIRO	0	$0 * (2^2) = 0$
QUARTO	0	$0 * (2^3) = 0$
QUINTO	0	$0 * (2^4) = 0$
SEXTO	1	$1 * (2^5) = 32$
SÉTIMO	0	$0 * (2^6) = 0$
OITAVO	0	$0 * (2^7) = 0$
Soma		= 32

Ao somar os resultados dos produtos teremos o valor decimal correspondente, que é igual a 32.

BITOR Argumento 1 Argumento 2

Realiza a função lógica OR entre os argumentos. Contrariamente à função AND, a tabela verdade que caracteriza o operador "OR" é a expressão na tabela 2.

Argumento 1	Argumento 2	Resultado
0	0	0
0	1	1
1	0	1
1	1	1

Tabela 2. — Tabela verdade da função lógica OR.

Para que possamos ver praticamente pressionemos de maneira direta.

```
PRINT BITOR 244 35
```

(que são os mesmos números utilizados na discussão da função AND).

O computador contestará com uma mensagem

```
RESULT :247
```

Recordemos que os valores decimais 244 e 35 equivalem aos binários:

Decimal 244 = binário 1 1 1 1 0 1 0 0

Decimal 35 = binário 0 0 1 0 0 0 1 1

Efetuada o mesmo procedimento que para o operador AND, examinemos o bit mais a esquerda do número 244 (= 1) e o comparamos com o primeiro do número 35, fazendo neste caso referência à tabela da verdade da função OR.

A linha a tomar em consideração é outra vez a terceira: como entradas teremos 1 e 0, e como saída, um bit a nível 1 (ligado).

Se repetirmos o mesmo raciocínio para os bits restantes dos dois números obteremos a tabela que segue:

Bit n.º	Argumento 1	Argumento 2	Resultado
2	1	0	1
3	1	1	1
4	1	0	1
5	0	0	0
6	1	0	1
7	0	1	1
8	0	1	1

A última coluna, como na função passada AND, é a expressão binária do número que nós procuramos (247 = BITOR 244 35).

Se, quisermos, finalmente, converter em decimal o valor 1 1 1 0 1 1 1 1 (obtido depois de ter invertido a sequência de bits presentes na quarta coluna), escreveremos:

BIT	VALOR	RESULTADO
PRIMEIRO	1	$1 * (2^0) = 1$
SEGUNDO	1	$1 * (2^1) = 2$
TERCEIRO	1	$1 * (2^2) = 4$
QUARTO	0	$0 * (2^3) = 0$
QUINTO	1	$1 * (2^4) = 16$
SEXTO	1	$1 * (2^5) = 32$
SÉTIMO	1	$1 * (2^6) = 64$
OITAVO	1	$1 * (2^7) = 128$
SOMA		= 247

BITXOR Argumento 1 Argumento 2

A primitiva BITXOR gera como saída um valor obtido ao aplicar a função OR exclusiva aos dois argumentos.

A tabela verdade relativa a este operador é a tabela 3.

Argumento 1	Argumento 2	Resultado
0	0	0
0	1	1
1	0	1
1	1	0

 Tabela 3. — Função OR exclusiva (XOR) aplicada a dois argumentos.

No que se refere às técnicas necessárias para a utilização deste operador valem as mesmas considerações utilizadas para as funções AND e OR.

Seguidamente lhes oferecemos a listagem do programa CONVERSAO. Após ter escrito de maneira direta:

```
RUN CONVERSAO NUMERO 1 NUMERO 2
```

O programa imprimirá na tela, em formato binário, os dois números de entrada e, sempre em binário, o resultado realizado com estas operações AND, OR e XOR.

O programa durante a fase da conversão de decimal para binário segue o algoritmo das divisões sucessivas utilizada por nós manualmente antes, empregando a primitiva QUOTIENT que veremos no seguinte item:

TO CONVERTE: NÚMERO

```
MAKE "CONTADOR 0
MAKE "BINARIO [ ]
MAKE "RESTO 0
DECIM. BIN :NÚMERO
PRINT :BINARIO
```

END

TO CONVERTE :NÚMERO 1 :NÚMERO 2

```
NODRAW
PRINT [ ]
PRINT [A REPRESENTAÇÃO BINÁRIA]
PRINT [ ]
PRINT [DO PRIMEIRO NÚMERO E:]
PRINT [ ]
CONVERTE: NÚMERO 1
PRINT [ ]

PRINT [ E A DO SEGUNDO NÚMERO:]
PRINT [ ]
CONVERTE: NÚMERO 2
PRINT [ ]
PRINT [A OPERAÇÃO "AND" DA COMO
  RESULTADO:]
PRINT [ ]
CONVERTE BITAND: NÚMERO 1 : NÚMERO 2
PRINT [ ]
PRINT [A OPERAÇÃO "OR" PRODUZ O SEGUINTE
  RESULTADO]
PRINT [ ]
CONVERTE BITOR: NÚMERO 1 :NÚMERO 2
PRINT [ ]
PRINT [AO USAR A "OR EXCLUSIVA" NOS EN-
```

```

CONTRAMOS COM:]
PRINT [ ]
CONVERTE BITXOR: NUMERO 1 : NUMERO 2
PRINT [ ]

END

TO DECIM. BIN :NUMERO

MAKE "CONTADOR :CONTADOR + 1
IF CONTADOR = 9 THEN STOP
MAKE "RESTO :NUMERO-2 QUOTIENT
:NUMERO 2 ;[VER SEGUINTE ITEM PARA
QUOTIENT]
MAKE "BINARIO SENTENCE :BINARIO :RESTO
DECIM. BIN QUOTIENT :NUMERO 2

END

```

Operadores algébricos

A linguagem LOGO põe à disposição do programador uma série de instruções cuja função desenvolve-se toda no campo da matemática.

Além dos acostumados operadores algébricos +, *, -, /, podemos dispor das primitivas:

INTEGER Argumento

O argumento pode ser um número, uma variável numérica ou uma expressão.

Cada vez que é utilizada esta instrução gera-se como saída um valor numérico igual à parte inteira do argumento.

Obteremos por exemplo:

```

INTEGER 5.1 → 5
INTEGER 5.9 → 5
INTEGER - 7.2 → 7
INTEGER - 7.8 → 7

```

Ou também

INTERGER 65 → RESULT 65

ROUND Argumento

Também a primitiva ROUND requer como entrada um valor numérico (constante, variável ou também uma expressão).

Se o valor de entrada é inteiro não será alterado; em caso contrário o argumento arredonda o inteiro mais próximo. De fato, por exemplo, resultará que:

ROUND 4.3 → 4

ROUND 4.6 → 5

ROUND -7.8 → -8

ROUND -7.2 → -7

e que

ROUND 5654 → 5654

QUOTIENT Elemento 1 Elemento 2

A utilidade da primitiva nota-se cada vez que é necessário calcular um cociente entre dois elementos (constantes e/ou variáveis).

O resultado é o valor inteiro do cociente; portanto, não terá significado uma ordem do tipo:

PRINT INTEGER QUOTIENT 30 4

Aqui alguns exemplos práticos do emprego da primitiva da qual falamos:

PRINT QUOTIENT 8 4 → 2

PRINT QUOTIENT -14 3 → -4

PRINT QUOTIENT 6 (-24) → 0

PRINT QUOTIENT 0 534 → 0

Note que se o argumento 2 é negativo deverá ser colocado entre parênteses.

É importante que o programador, durante o desenrolar de um algoritmo que utilize a primitiva QUOTIENT, lembre-se que

a segunda entrada (divisor) não pode ser cancelada e que, em consequência, se o for, surgirá uma mensagem de erro que passa necessariamente da execução para o procedimento que está sendo efetuado.

REMAINDER Argumento 1 Argumento 2

Ao contrário da função desenvolvida por QUOTIENT, a primitiva REMAINDER calcula o resto da divisão entre os dois argumentos inteiros que podem ser constantes e/ou variáveis numéricas; no caso de que o segundo argumento seja negativo, é necessário ser colocado entre parênteses.

Aqui alguns exemplos:

REMAINDER 10 2 → RESULT: 0

REMAINDER 14 3 → 2

REMAINDER -5 2 → 1

REMAINDER 4 (-8) → 0

SQR Argumento

Por último, falemos do operador SQR: este **dá a raiz quadrada do argumento**.

Uma advertência para o programador mais inexperiente: só pode ser calculado com SQR o valor da raiz quadrada de um número positivo ou, no máximo, nulo.

Funções trigonométricas

A finalidade deste item é guiar o leitor para que repasse as principais funções trigonométricas.

É necessário recordar a definição de um **arco goniométrico**: trata-se de uma **circunferência com o centro na origem de um sistema de eixos cartesianos ortogonais e de raio arbitrário que convencionalmente supõe-se unitário** (Fig. 1).

A circunferência goniométrica é dividida pelos eixos cartesianos em quatro QUADRANTE.

PRIMEIRO QUADRANTE: Arco AB

SEGUNDO QUADRANTE: Arco BA'

TERCEIRO QUADRANTE: Arco A'B'

QUARTO QUADRANTE: Arco B'A

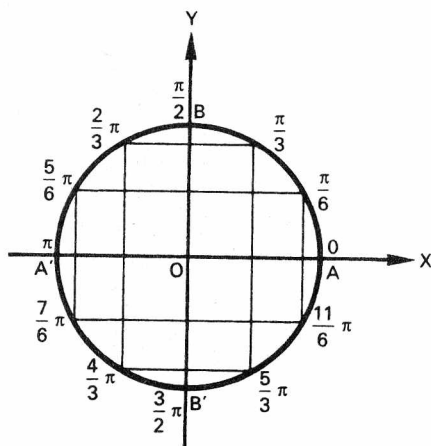


Figura 1. — Circunferência goniométrica.

Fazendo referência a esta circunferência com raio unidade podemos afirmar por definição que:

O seno de um arco é a ordenada do extremo de dito arco e que:

O cosseno de um arco é a abscissa do extremo do dito arco.

Na Figura 2 pode-se ver como seno e cosseno são funções da amplitude do arco e variam segundo a variação deste. Para o ângulo escolhido (x) o seno seria "Y" e o cosseno "X".

Pode ser desenhado um diagrama que ponha em relação o valor do ângulo e o seno ou cosseno correspondente.

Se sobre as ordenadas levamos a medida do ângulo e sobre as abscissas o correspondente valor da função do seno, obteremos o que se representa na Figura 3.

Com o mesmo sistema para a função cosseno obtemos a figura 4.

Estes gráficos ressaltam as características mais notáveis de ambas as funções trigonométricas, que podem ser sintetizadas em:

- ambas são periódicas (com um período de 360°);
- são funções limitadas; variam, como dissemos, desde um mínimo (-1) até um valor máximo (+1);

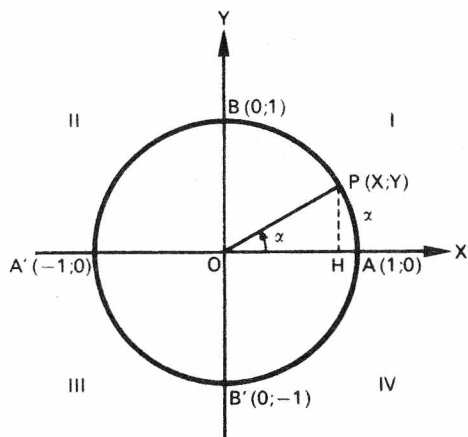


Figura 2. — Visualização gráfica da relação entre seno, cosseno e o arco correspondente.

- a função seno é positiva no primeiro e no segundo quadrantes e negativa no terceiro e quarto. Além disso, a senóide é crescente no primeiro e quarto quadrantes e decrescente nos restantes;
- o cosseno é positivo no primeiro e quarto quadrantes, e negativo no segundo e terceiro. Além disso é simples verificar no diagrama que é crescente nos quadrantes terceiro e quarto e decrescente nos restantes.

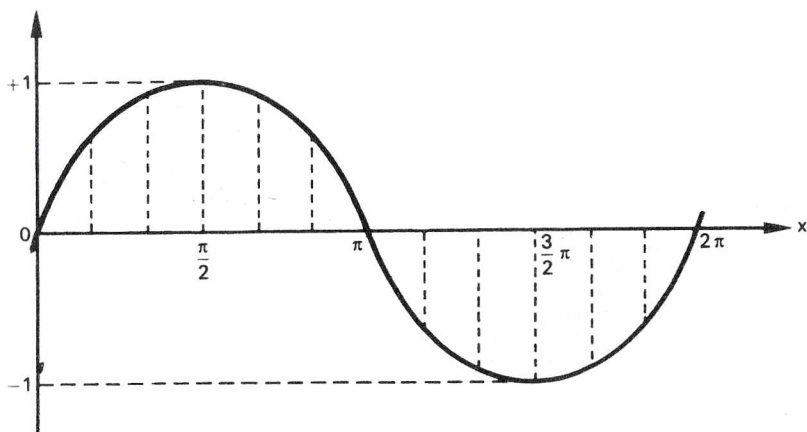


Figura 3. — Diagrama da função SENO.

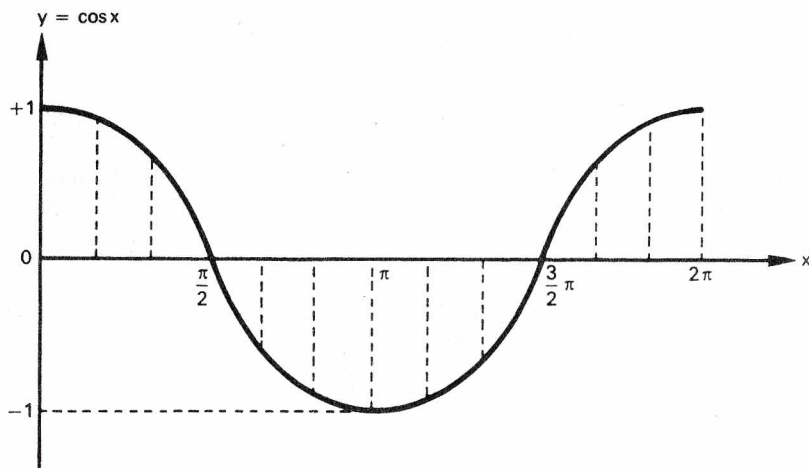


Figura 4. — Diagrama da função COSENO.

O breve programa seguinte, traça na tela o diagrama da função SENO e o eixo das abscissas (para facilitar uma referência cômoda):

```
TO SENO :X
  IF :X > 360 HOME STOP
  SET XY :X / 2 - 100 100 * SIN :X
  SENO :X + 3
END
```

Prestemos atenção à terceira linha do procedimento: “SIN :X” calcula o seno do argumento,

```
SETXY :X/2-100 100*SIN :X”
```

leva a tartaruga até a posição especificada do sistema de eixos cartesianos ortogonais fixados na tela.

Já que recordamos o conceito do seno e cosseno, examinemos outra função trigonométrica fundamental: a tangente.

Se chama **tangente trigonométrica** de um ângulo a relação entre o seno e o cosseno do mesmo.

Em termos matemáticos seria:

$$\text{TANGENTE (alfa)} = \text{SIN (alfa)} / \text{COS (alfa)}$$

O diagrama relativo à função da qual falamos está representado na Figura 5.

A tangente não está definida nos ângulos $\pi/2, 3\pi/2$, etc. Como veremos, são os mesmos nos quais anula-se o cosseno; isto acontece porque “ $\tan = \text{sen}/\text{cos}$ ” e, portanto, não pode ser definido nos pontos em que tem zero no denominador.

Um programa simples que nos dá a possibilidade de traçar o diagrama da função tangente é:

TO GRAFICO :X

IF :X > 360 HOME STOP

IF (COS :X) = 0 HOME STOP

MAKE “Y 100 * (SIN :X) / (COS :X)

IF :Y < -100 THEN PENUP GRAFICO :X + 5

STOP

IF :Y > 100 THEN PENUP GRAFICO :X + 5

STOP

SETXY :X / 2 - 100 :Y

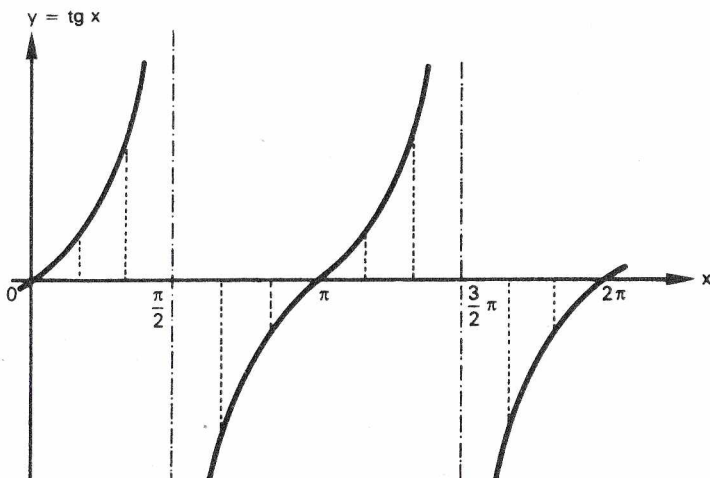


Figura 5. — Diagrama da função TANGENTE.

```

PENDOWN
TANGENTE :X + 3

END

TO TANGENTE

HOME
CLEARSCREEN
GRAFICO 0

END

```

Geração de números aleatórios

A linguagem LOGO coloca à disposição do programador uma interessante e, em alguns casos, insubstituível função numérica: trata-se da primitiva “RANDOM n”, que gera como saída um valor numérico aleatório, compreendido entre 0 e (n-1).

É importante recordar que o argumento desta instrução deve ser necessariamente positivo e inteiro.

Um exemplo de utilização desta instrução está constituído pelo programa ADIVINHE que apresentaremos em seguida:

```

TO TRANSNUMERO

OUTPUT FIRST :TENTATIVA

END

TO OK

NODRAW
REPEAT 4 [REPEAT 33 [PRINT1 “*]
PRINT [] ]

```

```

PRINT [ ]
PRINT [FELICIDADES! VOCE CONSEGUIU]
PRINT [ ]
PRINT 1 SENTENCE [EM SO] :TEN
PRINT [TENTATIVAS]
PRINT [ ]
REPEAT 4 [REPEAT 33 [PRINT1 "*"
    PRINT [ ] ]
PRINT [ ]
PRINT [ ]
PRINT [F = FINAL]
PRINT [C = CONTINUAR]
PRINT [ ]
MAKE "RESPOSTA READCHARACTER
    IF: RESPOSTA = "C THEN ADIVINHE

END

```

TO ENTRADA

```

NODRAW
REPEAT 3 [PRINT [ ] ]
PRINT [INTRODUZA SEU NUMERO]
PRINT [ ]
MAKE "TENTATIVA REQUEST

```

END

TO JOGO

```

MAKE "TEN :TEN + 1
ENTRADA
MAKE "TENT TRANSNUMERO
IF :TENT = :NUMERO.PENSADO THEN OK

```

```

IF :TENT > :NÚMERO.PENSADO THEN
    PRINT [ ] PRINT [MEU NÚMERO É MENOR
    QUE O SEU] PRINT [ ] PRINT REQUEST
    JOGO
IF :TENT < :NÚMERO.PENSADO THEN
    PRINT [ ] PRINT [MEU NÚMERO É
    MAIOR QUE O SEU ]PRINT [ ] PRINT REQUEST
    JOGO
END
TO ADIVINHE
    MAKE "TEN 0
    NODRAW
    PRINT [ ]
    PRINT [EU PENSO UM NÚMERO]
    PRINT [ ]
    PRINT [COMPREENDIDO ENTRE 0 E 100]
    PRINT [ ]
    PRINT [VOCE DEVERA ADIVINHA-LO]
    PRINT [ ]
    PRINT [NO MENOR NÚMERO POSSIVEL DE]
    PRINT [ ]
    PRINT [TENTATIVAS, LEMBRANDO DE]
    PRINT [ ]
    PRINT [TODAS AS SUGESTOES QUE]
    PRINT [ ]
    PRINT [FIZER]
    PRINT [ ]
    PRINT REQUEST
    MAKE "NÚMERO.PENSADO RANDOM 100
    JOGO
END

```

A finalidade do programa é conseguir no menor número de tentativas a individualização de uma cifra gerada aleatoriamente pelo computador.

Além disso, o programa guia o usuário através das tentativas advertindo se o número proposto por ele é menor ou maior que o escolhido.

Um exemplo de execução do programa poderá ser:

```
RUN ADIVINHE
EU PENSO UM NÚMERO
COMPREENDIDO ENTRE 0 E 100
VOCE DEVERA ADIVINHA-LO
NO MENOR NÚMERO POSSIVEL DE
TENTATIVAS, LEMBRANDO DE
TODAS AS SUGESTOES
QUE FIZER
<RETURN>
INTRODUZA SEU NÚMERO
40
MEU NÚMERO É MAIOR
QUE O SEU
<RETURN>
INTRODUZA SEU NÚMERO
50
MEU NÚMERO É MENOR
QUE O SEU
<RETURN>
INTRODUZA SEU NÚMERO
45
*****
*****
*****
*****
FELICIDADES! VOCE CONSEGUIU
EM SO 3 TENTATIVAS
*****
*****
*****
*****
F = FINAL
C = CONTINUAR
F
```

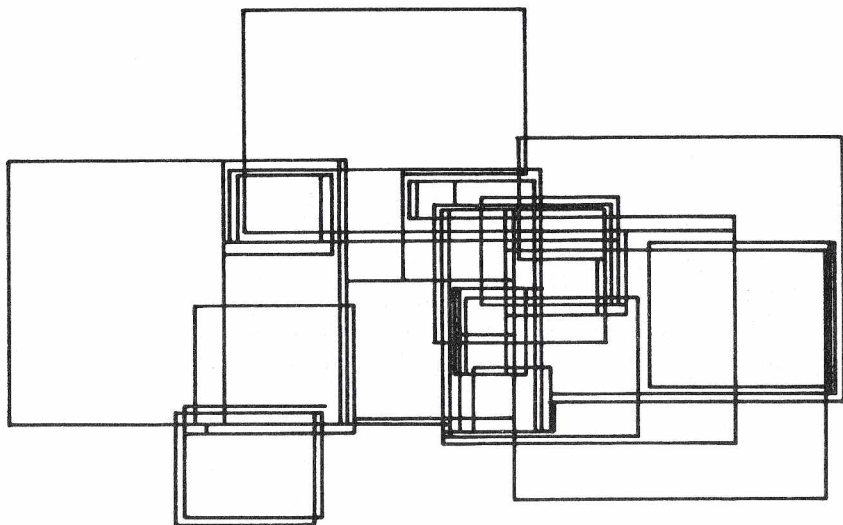
e a execução do programa termina.

Voltando a utilização da primitiva RANDOM é interessante usá-la em aplicações do tipo gráfico. Ainda que aparentemente simples, este emprego é, contudo, muito difícil. Fazer gráficos

agradáveis aos olhos humanos quer dizer unir ao aspecto “regular” da imagem, alguns parâmetros aleatoriamente variáveis: criar um tema de fundo sobre o qual atua a fantasia da primitiva RANDOM.

Em caso contrário, se confiarmos a execução da imagem somente ao acaso, deveremos nos conformar com verdadeiras “confusões”, como a que é apresentada na Figura 6, obtida com o procedimento DESENHA.

```
TO DESENHA
  MAKE "C 0
  ESPIRAL RANDOM 100
  DESENHA
END
TO ESPIRAL :X
  FORWARD :X
  RIGHT 90
  MAKE "C :C + 1
  IF +C 4 THEN STOP
  ESPIRAL +X +1
END
```



 *Figura 6. — Resultado do procedimento DESENHA, no qual somente atua o fator aleatório.*

CAPÍTULO VI

CÍRCULO, PARÁBOLAS E ELIPSES



leitor, sem dúvida, já viu que o intérprete da LOGO é rico em instruções gráficas ou não fundamentais cuja função, ainda que elementar, é de indiscutível e insubstituível valor.

Primitivas como as clássicas FORWARD, LEFT, etc., são muito pouco sofisticadas, porém para estas encomendam-se tarefas irrenunciáveis por parte da mesma linguagem.

Nas páginas anteriores falamos a fundo de como pode ser construído (definido) através destas primitivas um conjunto de ordens mais complexas que satisfaçam as específicas exigências do programador.

Este capítulo irá apresentar, neste sentido, uma ajuda válida para o programador que tenha a intenção de enfrentar problemas do tipo gráfico, como desenho de arcos, círculos ou figuras mais complexas.

Círculos e arcos

O procedimento mais simples que podemos escrever para desenhar uma circunferência é o que segue:

TO CÍRCULO

HOME


```

CLEARSCREEN
PENUP
LEFT 90
FORWARD 100
PENDOWN
REPEAT 360 [FORWARD 1 RIGHT 1]
END

```

O procedimento CIRCULO gera a figura geométrica requerida, sem recorrer à equação matemática típica da circunferência, que, além disso, não é nada fácil; **utiliza exclusivamente o princípio do "polígono regular"**.

Em outras palavras, criar um polígono de 360 lados tão pequeno que ao olho humano parece uma circunferência.

Este método, desde já, oferece seus melhores resultados quando o lado do polígono tende a 0 (teoricamente um segmento de comprimento nulo é igual a um ponto).

Podemos dizer, portanto, que **uma circunferência é aproximável por um polígono regular que tenha como características estes parâmetros:**

N = número de lados (tão grande quanto seja possível).

L = dimensão de um lado (tão pequeno quanto seja possível).

$LIMITE\ N \cdot L$ = perímetro da circunferência.

N inversamente proporcional a L de maneira que:

$N = 1/L$

Neste contexto, e lembrando sempre a resolução gráfica dos computadores, pode ser obtido um bom resultado tomando como medidas do lado e do ângulo a unidade.

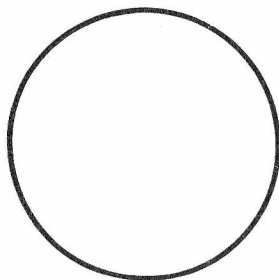


Figura 1. — Círculo obtido como um polígono regular de 360 lados.

Obteremos assim o gráfico de um círculo como o da Figura 1.

Se quisermos uma melhor aproximação seria suficiente trocar a instrução REPEAT do programa desta maneira:

```
REPEAT 720 (FORWARD 0.5 RIGHT 0.5)
```

Esta maneira de traçar um círculo é, sem dúvida, a mais simples, porém não permite controle algum por parte do usuário nas dimensões e características da mesma.

Tentemos agora criar um **procedimento que pode levar a cabo uma série de círculos com distintos diâmetros.**

Pressionemos:

```
TO CIRCULO :LADO
```

```
  HOME
```

```
  CLEARSCREEN
```

```
  PENUP
```

```
  LEFT 90
```

```
  FORWARD 100
```

```
  PENDOWN
```

```
  REPEAT 360 [FORWARD :LADO RIGHT 1]
```

```
END
```

Desta maneira poderemos variar as características da circunferência desenhada introduzindo um parâmetro (lado do polígono) variável.

Porém esta solução resolve nossos problemas só em parte: se quisermos desenhar simplesmente um arco de circunferência com um certo ângulo com respeito ao centro não seria possível.

O programa que segue permite estas operações além das anteriores.

```
TO CIRCUNFERENCIA :ARCO
```

```
  :VARIACAO.ANGULAR
```

```
  REPEAT :ARCO / :VARIACAO.ANGULAR
```

```
    [FORWARD :VARIACAO.ANGULAR RIGHT
```

```
    :VARIACAO.ANGULAR]
```

```
END
```

Tentemos examinar o funcionamento deste programa, que à primeira vista pode parecer simples porque é curto, e que, no entanto, tem suas dificuldades.

Proporcionam-se como entradas os parâmetros relativos à

medida do arco (em graus) e ao valor da variação angular desejada entre um lado e seus adjacentes do polígono regular que gera o círculo (se cresce este parâmetro aumentarão as dimensões do diâmetro).

A instrução REPEAT tem como primeiro argumento (números de ciclos) o resultado da divisão entre ARCO e VARIAÇÃO.ANGULAR que fica fixada antes de variar este último parâmetro.

Em outras palavras, se introduzirmos estes valores

ARCO: 100 graus

VARIACAO.ANGULAR: 2 graus

A repetição determinada por (ARCO/VARIAÇÃO.ANGULAR) será de 50.

Para obter uma circunferência (recordar que não é, em definitivo, nada mais que um arco de 360 graus) deveríamos pressionar

RUN CIRCUNFERENCIA 360 1

Estudaremos agora, antes de examinar os aspectos mais complexos das curvas, alguns programas demonstrativos.

CIRCUNFERÊNCIAS TANGENTES

TO CIRC.TANG

HOME

CLEARSCREEN

PENUP

LEFT 90

FORWARD 100

PENDOWN

RIGHT 90

REPEAT 360 [FORWARD 1 RIGHT 1]

REPEAT 360 [FORWARD 1.5 RIGHT 1]

REPEAT 240 [FORWARD 1 RIGHT 1.5]

END

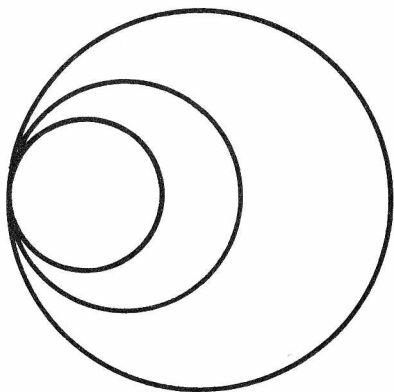


Figura 2. — Circunferências tangentes interiores.

Gera a Figura 2

CIRCUNFERÊNCIAS EXTERIORES

TO CIRC.EXT

REPEAT 360 [FORWARD 1 RIGHT 1]

REPEAT 360 [FORWARD 1 LEFT 1]

END

Desenha a Figura 3.

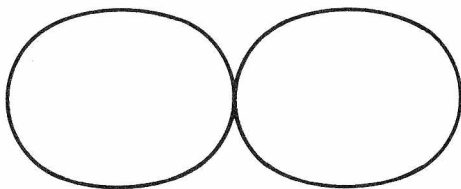


Figura 3. — Duas circunferências tangentes exteriores.

Até agora falamos de parâmetros que tem a possibilidade de fazer variar as dimensões e as características de uma circunferência; no entanto, nenhum destes pode ser tão controlado de maneira perfeita em seus resultados como o que segue, que desenha um círculo cujo raio é variável.

CIRCUNFERÊNCIA CON RADIO VARIÁVEL

```
TO PLOTTER :X
```

```
  HIDETURTLE
```

```
  PENUP
```

```
  FORWARD :X
```

```
  PENDOWN
```

```
  FORWARD 1
```

```
  PENUP
```

```
  BACK :X + 1
```

```
  LEFT 1
```

```
END
```

```
TO RAIO :X
```

```
  REPEAT 360 [PLOTTER :X]
```

```
END
```

Executando este programa com:

```
RUN RAIO 40
```

teremos o exposto na Figura 4.

Parábolas

A parábola é, a exemplo da circunferência, uma curva no plano. **Matematicamente, a parábola é o lugar (conjunto) dos pontos do plano eqüidistante de um ponto fixo (f) chamado foco e de uma reta chamada diretriz.**

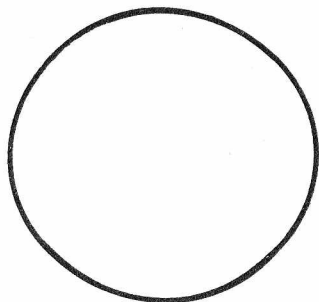


Figura 4. — Círculo obtido com o procedimento RAIIO.

Os mais experientes sabem, sem dúvida, que a equação que a define em um sistema de eixos cartesianos é:

$$Y = A*(X*X) + B*(X) + C$$

Utilizaremos esta equação (simplificada e resolvida de forma particular) para traçar a figura desejada.

```
TO PARABOLA :A
```

```
  EIXOS
```

```
  PENUP
```

```
  PLOTTER (- SQR 118 * 4* :A) :A
```

```
END
```

```
TO PLOTTER :X :A
```

```
  LOCAL "Y
```

```
  MAKE "Y (:X * :X) / (4 * :A)
```

```
  IF :Y > 126 STOP
```

```
  SETXY :X :Y
```

```
  PENDOWN
```

```
  PLOTTER :X + 5 :A
```

```
END
```

TO EIXOS

HOME

CLEARSCREEN

SETXY 155 0

HOME

SETXY -155 0

END

Ao realizar

RUN PARABOLA 20

obteremos o que mostra a Figura 5.

Note como ao incrementar-se o valor do parâmetro "A" diminui o crescimento hiperbólico dos lados da parábola. Em outros termos, um número pequeno significa uma figura mais "fechada", e vice-versa.

Elipses

Matematicamente a elipse é o lugar geométrico dos pontos do plano cujas distâncias a dois pontos fixados (f_1 e f_2) chamados focos tem uma soma constante.

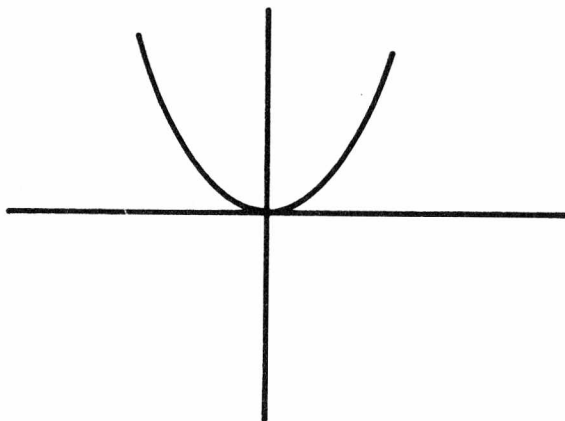


Figura 5. — Parábola.

A equação geral é:

$$(Y^2 / A^2) - (X^2 / B^2) = 1$$

Usaremos o seguinte procedimento para desenhá-la:

TO ELIPSE :A :B

 EIXOS

 PENUP

 HIDETURTLE

 PLOTTER — :A :A :B 1

END

TO PLOTTER :X :A :B :INC

 IF (:X * :X) > (:A * :A) STOP

 IF :X = :A THEN MAKE "INC (- :INC)

 SETXY :X :INC * :B * SQRT (1 - (:X *
 :X) / (:A * :A))

 PENDOWN

 PLOTTER :X + :INC :A :B :INC

END

TO EIXOS

 HOME

 CLEARSCREEN

 SETXY 155 0

 HOME

 SETXY -155 0

END

Para executar o que escrevemos pressione:

```
RUN ELIPSE 100 180 10 10
```

que nos dará o resultado representado na Figura 6.

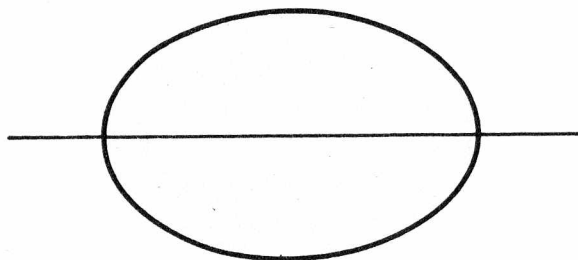


Figura 6. — *Ellipse.*

Lembre-se sempre que o primeiro parâmetro correspondente à medida do eixo maior e que o segundo refere-se à medida do eixo menor.

Desta maneira pode ser obtida uma elipse com o eixo maior paralelo ao das abcissas:

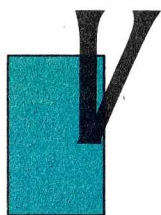
```
RUN 100 100 10 10
```

ou paralelo ao das ordenadas:

```
RUN 30 100 10 10
```

CAPÍTULO VII

OS ARQUIVOS



eremos aqui os comandos dedicados no LOGO ao tratamento dos arquivos, típicos das versões baseadas no COMMODORE 64 e no Apple II.

Versão COMMODORE 64

SAVE "nomearquivo

Transfere todos os procedimentos presentes em memória ao disco, dando ao arquivo o nome especificado.

SAVE "nomearquivo [Proce 1 Proce 2...]

Abre sobre o disco um arquivo com o nome especificado e transfere a ele exclusivamente os procedimentos especificados como argumentos, na condição que estão presentes na memória.

READ "nomearquivo

Carrega do disco e arquivo desejado as primitivas contidas que, se são distintas às presentes na memória, são acrescentadas a estas; em caso contrário, as substituem.

Uma técnica particularmente interessante que pode ser feita, depois de uma instrução READ, é que o programa carregado

em memória seja executado automaticamente.

Suponhamos ter residente em memória o programa PRG que desejamos registrar sobre o disco com a técnica de execução automática (AUTOSTART).

Pressionamos o comando:

```
MAKE "STARTUP [PRG]
```

que cria e coloca ao fundo da listagem de nosso programa a variável STARTUP que contém a lista [PRG].

Neste momento gravemos sobre o disco o programa (PRG).

```
SAVE "PRG
```

e a operação estará concluída.

Se agora queremos carregar nosso programa (melhor depois de um ERASE ALL) é suficiente escrever:

```
READ "PRG
```

Depois de ter levado a cabo o carregamento será executado automaticamente sem necessidade de nossa intervenção.

```
BSAVE "nomearquivo star end + 1
```

Esta primitiva salva em disco uma região inteira de memória, mais exatamente desde a direção "star" até a "end" dando-lhe o nome "nomearquivo.LOGO". As direções "star" e "end" devem ser dois valores decimais inteiros e positivos.

```
BLOAD "nomearquivo
```

Executa a função contrária de BSAVE. Coloca em memória o conteúdo do arquivo indicado, gerado anteriormente graças ao comando BSAVE.

```
SAVEPICT "nomearquivo
```

Copia no arquivo de nome especificado o conteúdo da página de alta resolução.

A opção SAVEPICT cria sobre o disco dois arquivos: o primeiro contém as informações gráficas, o segundo as cores.

```
READPICT "nomearquivo
```

Se anteriormente registramos uma figura com o comando SAVEPICT podemos visualizá-la de novo utilizando esta primitiva. Evidentemente só tem sentido no caso de que o arquivo procurado exista; em caso contrário, o sistema gera uma mensagem de erro.

CATALOG

Visualiza a lista dos nomes de todos os arquivos presentes no disco introduzido na unidade.

ERASEFILE "nomearquivo"

Apaga do disco o arquivo especificado no argumento. Se não existe arquivo com o nome procurado gera uma mensagem de erro.

DOS [NEW:nomedisco, ID]

Executa a operação de formato de disco necessária para preparar um disco virgem, nunca utilizado, para as futuras gravações. A identificação (ID) é um número inteiro de duas cifras que se atribui ao disco junto com o nome, para facilitar buscas futuras.

DOS [RENAME:novonome = velhonome]

Suponhamos ter registrado sobre o disco um arquivo chamado "ANTES" e que queremos trocar seu nome pelo de "DEPOIS", será suficiente utilizar a primitiva RENAME pressionando

DOS [RENAME:DEPOIS = ANTES]

A operação é automática.

DOS [COPY:nomecopia = arquivoacopiar]

A primitiva COPY abre o arquivo chamado "nomecopia" e copia nele o conteúdo do arquivo "arquivoacopiar".

A operação é possível se no disco está presente o arquivo a duplicar e se o nome do arquivo duplicado não corresponde a nenhum arquivo já presente no disco.

DOS [SCRATCH:nomearquivo]

Esta instrução é análoga à primitiva ERASEFILE; apaga do disco o arquivo cujo nome está especificado no argumento.

Versão APPLE II

Falemos agora das primitivas do LOGO APPLE, cuja função está dedicada ao processamento dos arquivos.

SAVE "nomearquivo

Cria em disco um arquivo chamado "nomearquivo.LOGO" e transfere a ele os procedimentos que podem estar presentes na memória.

SAVE "nomearquivo "Proce1 "Proce2...

Como a ordem anterior, abre em disco o arquivo chamado "nomearquivo.LOGO", porém agora neste são copiados, desde a memória do computador, só os procedimentos especificados (PROCE1, PROCE2...).

LOAD "nomearquivo

Carrega na memória o arquivo indicado no argumento; os procedimentos contidos neste são acrescentados aos que estão presentes na memória. O sistema gera um erro em caso de tentar carregar desde o disco a um arquivo que não exista.

CATALOG

Imprime na tela a lista de todos os arquivos anteriormente registrados em disco.

ERASEFILE "nomearquivo

Apaga do disco o arquivo especificado no argumento. Se o arquivo em questão não existe, o sistema gera uma mensagem de erro.

DISK

Recorrendo a esta instrução geram-se como saída três valo-

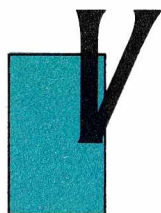
res; o primeiro indica o número de unidade; o segundo, o número de slot típico da unidade utilizada e, finalmente, o último número correspondente ao identificador presente no disco.

SET DISK

Troca o número de disco por default.

CAPÍTULO VIII

COMMODORE 64 E LOGO



amos falar agora, a título de ilustração, das primitivas da linguagem LOGO típicas da versão baseada sobre os sistemas COMMODORE e sobre as quais não discutimos nas páginas anteriores do texto .

ATAN X Y

A instrução ATAN X Y calcula o cociente dos argumentos e em seguida efetua a função ARCO TANGENTE do valor obtido; a utilidade de uma primitiva deste tipo desenvolve-se principalmente no campo gráfico quando devemos, por exemplo, calcular trajetórias de objetos em movimento.

No caso de que o segundo argumento seja número negativo deverá necessariamente estar fechado entre um par de colchetes.

CLEARTEXT

Limpa a tela de baixa resolução e coloca o cursor no ângulo superior esquerdo da tela.

CURSOR X Y

Os valores "x" e "y" são atribuídos à abcissa e à ordenada do cursor, respectivamente. Este comando desenvolve a função oposta à primitiva CURSORPOS.

CURSORPOS

Determina como saída uma listagem com somente dois elementos: o primeiro é o valor atual da abcissa do cursor, o segundo constitui a ordenada deste.

DOUBLECOLOR X

Dispõe a tela gráfica de um modo duplo de cor; a resolução horizontal diminui (150 pontos) porém podem ser utilizadas até 16 cores diferentes (e também duas simultaneamente na mesma região de 8x8 pontos) em base à tabela 1 do primeiro capítulo.

Lembrando que, se trabalhar em maneira de dupla cor a instrução STAMPCHAR provoca resultados irregulares (caracteres ilegíveis) pela troca da resolução horizontal (e por fatores mais complexos que agora não é oportuno tratar).

DRAWSTATE

Potente comando que gera uma listagem que contém todas as informações atuais da tartaruga na seguinte ordem:

- estado da tartaruga (FALSE = Plumilha levantada/TRUE = Plumilha baixada);
- visibilidade da tartaruga (FALSE = tartaruga não está visível/TRUE = tartaruga visível);
- código de cor para tela (número inteiro desde 0 à 15);
- cor da tartaruga;
- modo de cor (standard ou dupla cor);
- modo de visualização (gráfica, texto ou gráfica e texto);
- código da cor atribuída a tela em baixa resolução;
- cor dos caracteres.

A listagem gerada por DRAWSTATE depois de ter ligado a máquina é: [TRUE TRUE 11 1 DRAW SINGLECOLOR SPLITS-SCREEN 14 1]

FPRINT X

No caso de que o argumento "X" seja uma tabela imprime-se na tela com parênteses contrariamente ao que acontece para o clássico e mais utilizado PRINT (ou PRINT1).

HEADING

Gera o valor em graus do ângulo tomado pela tartaruga. Recordamos que depois das instruções do tipo HOME, CS e DRAW a instrução HEADING gera o valor 0 (zero graus).

JOYSTICK X

A variável "X" deve assumir exclusivamente os valores 0 ou 1; a primitiva determina a atual posição do joystick introduzindo na porta o número "X".

JOYBUTTON X

Como JOYSTICK requer que o parâmetro "x" valha 0 ou 1; o comando JOYBUTTON gera a mensagem TRUE se pressionar o botão do joystick; em caso contrário teremos o sinal de FALSE.

NOPRINTER

Depois de uma operação sobre a impressão iniciada graças ao comando PRINTER, NOPRINTER fecha os canais de comunicação abertos anteriormente.

NOWRAP

No modo gráfico standard (WRAP), no caso de que a tartaruga saia por um lado da tela, voltará a entrar imediatamente pelo lado contrário. É possível graças à instrução NOWRAP evitar isto.

Se tentamos escrever

```
HOME NOWRAP FORWARD 5000
```

obteremos de fato a mensagem de erro "TURTLE OUT OF BOUND" (uma expressão muito próxima do português "tartaruga fora da tela"). Para retornar ao modo gráfico normal é suficiente recorrer à expressão WRAP.

PADDLE X

É possível controlar a atual posição do Paddle número "x" graças a esta interessante primitiva.

PADDLEBUTTON x

Gera a mensagem TRUE ou FALSE, dependendo de que a tecla do Paddle número "x" seja pressionada ou não.

PRINTER

Ao executá-lo, as acostumadas primitivas de impressão sobre tela atuarão diretamente sobre a impressora. Para voltar ao modo standard basta pressionar NOPRINTER.

SETH X, SETHEADING X

Fixa o ângulo da tartaruga no valor especificado pelo argumento "X".

SETSHAPE X

O parâmetro "x" deve ser do tipo numérico e incluído entre 0 e 7 (inteiro); este comando atribui a plumilha ativa (no total são 8). No momento de ligar, a plumilha número 0 corresponde ao sprite 0, ao número 2 o sprite 2, e assim sucessivamente.

SHAPE

Dá o valor do argumento da última instrução SETSHAPE que foi efetuado; para o caso de pressionarmos SHAPE sem ter levado a cabo a primitiva SETSHAPE anteriormente, obteremos o valor 0.

SINGLECOLOR

Leva o modo de visualização de alta resolução ao formato standard se anteriormente tinha sido alterado pela primitiva DOUBLECOLOR.

TELL x

O valor da variável "x" deve ser inteiro e incluído entre 0 e 7. Provavelmente nem todos sabem que o LOGO baseado em sistemas COMMODORE coloca à disposição do programador 8 sprites. É possível usá-los todos, porém só pode mover-se um a um. Por exemplo, escrevendo TELL 3, todas as instruções que pressio-

narmos depois serão atribuídas automaticamente à terceira plumilha. No momento de ligar a máquina trabalha com o SPRITE número 0 tendo disposto o estado dos 7 restantes (veja DRAWSTATE) ao valor (TRUE, TRUE...); em outras palavras se desejamos desenhar por meio da plumilha número “x” (com $x \neq 0$) antes de tudo faz falta pressionar a seqüência TELL x PENDOW (baixa plumilha “x”) SHOWTURTLE (torna visível a tartaruga).

TOWARDS x y

Potente primitiva gráfica. Gera o valor do ângulo que a tartaruga irá tomar deslocando-se desde a posição em que se encontra até o ponto de coordenadas “x” e “y”. Se escrevemos, por exemplo, HOME INTEGER TOWARD 100 100 obteremos o valor inteiro 45, já que a reta que une o ponto (0,0), onde encontrará a tartaruga depois de DRAW, com o ponto (100, 100), localizado pelos argumentos da primitiva TOWARDS, está inclinada justamente 45 graus com respeito aos eixos cartesianos.

WHO

Devolve um valor numérico inteiro e incluído entre 0 e 7 que corresponde ao número da plumilha com a que atualmente estamos trabalhando. Veja a este propósito o comentário relativo à primitiva TELLx. Como no momento de ligar o sistema, trabalha com a plumilha número 0, pressionando WHO obteremos portanto 0.

WRAP

Ver NOWRAP.

XCOR

Devolve o valor inteiro atual da abcissa da tartaruga.

YCOR

Devolve o valor inteiro atual da ordenada da tartaruga.

.ASPECT x

Potente instrução com possibilidade de trocar a escala da definição gráfica horizontal. Frequentemente ocorre que se trabalha

com monitores que tem como características uma tela mais larga do que alta: a imagem que resulta está claramente distorcida e incômoda além de ser esteticamente imprecisa (círculos que parecem elipses ou quadrados que tornam-se retângulos). Para evitar este inconveniente o intérprete LOGO coloca à disposição do usuário esta primitiva: se pressionarmos "ASPECT x" poderemos (a condição de que "x" seja distinto do valor standard 0,76), AUMENTAR ou DIMINUIR a escala resolutiva vertical.

CALL x

Passa todo o controle do sistema a uma subrotina em linguagem máquina residente na memória a partir da direção decimal "x" especificada no argumento; o usuário que tenha capacidade de levar a cabo o programa em linguagem máquina sabe que para voltar ao ambiente do intérprete LOGO é suficiente uma instrução do tipo RTS.

.CONTENTS

Lista os nomes de todos os procedimentos e das variáveis atualmente residentes na memória do computador.

.DEPOSIT x y

Os argumentos desta primitiva devem ser ambos inteiros e positivos (no máximo nulos); em particular, a variável "x" deve estar incluída entre um mínimo de 0 e um máximo de 256×256 , enquanto o valor "y" deve ser necessariamente menor de 256. A instrução "DEPOSIT x y" copia o conteúdo do argumento "x" na posição "y" da memória RAM.

.EXAMINE x

Gera o valor inteiro, incluído entre 0 e 255, lido na posição de memória ROM ou RAM número "y".

.GCOLL

O intérprete necessita periodicamente melhorar a disposição das variáveis de memória; esta operação chamada garbage collection, é levada a cabo automaticamente pelo sistema; no entanto, deixa ao programador a possibilidade de utilizar se for necessário.

.NODES

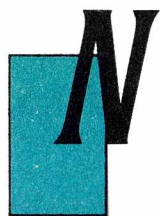
Interessante comando cuja função é colocar em conhecimento do programador quanta memória RAM dispõe todavia para desenvolvimento e a execução de seus programas. Aconselha-se utilizar a instrução **.NODES** fazendo-a preceder pela primitiva **.GCOLL**.

.SPRINT x

A primitiva **SPRITE-PRINT** é, sem dúvida, de grande ajuda prática para o programador que tenha a intenção de adquirir uma preparação completa acerca da manipulação dos sprites. Pressionando **".SPRINT x"** o intérprete LOGO transfere à tela de baixa resolução o que está codificado na memória de definição relativa ao sprite número "x".

CAPÍTULO IX

APPLE II E LOGO



este capítulo examina-se as principais primitivas da linguagem com referência à versão baseada no APPLE II.

Uma boa parte destas instruções foi amplamente discutida; neste caso, ao lado de cada uma nomeia-se o capítulo no qual se falou delas.

Primitivas gráficas

BG, BACKGROUND X

Leia a página gráfica com a cor correspondente ao código “x” proposto.

CHAR X

Gera o carácter cujo valor ASCII é “x”; o argumento desta primitiva, portanto, deve de ser positivo (no máximo, nulo), inteiro e menor de 255.

CLEAN

Apaga o conteúdo da página gráfica sem trocar os parâmetros (posição, ângulo, etc.) da tartaruga.

CS. CLEARSCREEN (Capítulo 1)

CLEARTEXT

Como CLEARSCREEN, porém atua na tela de baixa resolução gráfica.

CURSOR

Dá como saída a posição atual do cursor.

DOT x

Acende um ponto na tela em correspondência com o valor de "x".

FD. FORWARD x (Capítulo 1).

FULLSCREEN (Capítulo 1).

HEADING

Devolve o ângulo tomado pela tartaruga.

HT. HIDETURTLE (Capítulo 1).

HOME (Capítulo 1).

PEN

Gera uma listagem que contém o tipo e cor atualmente característicos da plumilha.

PC. PENCOLOR (Capítulo 1).

A tabela de códigos-cores típica do sistema APPLE II é a seguinte:

CÓDIGO	COR
00	Preto
01	Branco
02	Cinza
03	Violeta
04	Laranja
05	Azul marinho

PD, PENIDOWN (Capítulo 1).

PE, PENERASE (Capítulo 1).

PX, PENREVERSE

Coloca em vídeo inverso o percurso da tartaruga.

PU, PENUP (Capítulo 1).

SETH X, SETHEADING X

Coloca a plumilha dando-lhe um ângulo de “x” graus; as coordenadas, contudo, não variam.

SETPC n

Dá a plumilha a cor do código “x”.

SETPEN X

X deve ser uma lista que contenha o tipo e cor da plumilha. A primitiva encarrega-se da atribuição destes parâmetros à tartaruga.

SETPOS X

Move a tartaruga até a posição definida pelo parâmetro “x”.

SETXx (Capítulo 1).

SETYy (Capítulo 1).

SHOWNP

Comprova se a tartaruga é visível; em tal caso gera a mensagem TRUE; se é ao contrário, a saída é FALSE.

ST, SHOWTURTLE (Capítulo 1).

SPLITSCREEN (Capítulo 1).

TEXTSCREEN (Capítulo 1).

Dispõe a tela em modo texto.

WINDOW

Cria o chamado “modo janela”. A tartaruga tem a possibilidade de mover-se em um espaço mais amplo do representado pela tela.

WRAP

Depois desta ordem, ao sair a tartaruga por um determinado lado da tela, volta a entrar pelo lado contrário.

XCOR

Devolve a abcissa da tartaruga.

YCOR

Devolve a ordenada da tartaruga.

Outras primitivas

$\neq$ AND x y (como BITAND, Capítulo 5).

ARCTAN x

Calcula qual ângulo tem uma tangente cujo valor está especificado pelo argumento “ x ”.

ASCII x (Apêndice B).

BURY pkg

Apaga todos os procedimentos contidos em pkg.

BF x , BUTFIRST x (Capítulo 3).

BL x , BUTLAST x (Capítulo 3).

BUTTO x

Controla se a tela do paddle número “x” está pressionada; em caso afirmativo gera uma mensagem TRUE, ao contrário teremos FALSE.

CATCH nomelista

Executa a lista de instruções.

CO

Volta a “pegar” a execução do procedimento momentaneamente suspenso pelo comando PAUSE.

COPYDEF X Y

Potente primitiva que permite a imediata duplicata do conteúdo do procedimento “x” no procedimento “y”.

COS x (Capítulo 5).

COUNT x

Se “x” é uma lista que contém “n” elementos, ao aplicar esta ordem nos devolverá o valor “n”.

EDNS x

Introduz o operador de modo EDITOR porém só para as variáveis contidas pelo procedimento “x”; é conveniente proporcionar como argumentos de EDNS dois ou mais nomes de procedimentos.

EQUALP x y (como $x = y$).

ERALL (como ERASE ALL, Capítulo 2).

ERASE x (Capítulo 2).

ERN X

Apaga da memória a variável “x”. É permitida uma ordem do tipo ERNS x y z; as três (ou mais) variáveis serão canceladas em ordem.

ERNS x

O parâmetro “x” deve ser o nome de um procedimento atualmente definido; todas as variáveis contidas nele são canceladas. Estão permitidas várias entradas (por exemplo: ERNS x y z).

FIRST x (Capítulo 3).

FPUT x y (Capítulo 3).

GO x

Efetua um salto incondicional da execução do programa até a direção (label) chamada “x”. O programador experiente NÃO deverá utilizar esta primitiva.

IF (Capítulo 4)

IFF, IFFALSE (Capítulo 4).

IFT, IFTRUE (Capítulo 4).

INT n (Capítulo 5).

ITEM x

Se o argumento “x” está composto por “y” elementos (y x) então é gerada como saída o elemento que ocupa a posição “x” especificada pelo argumento.

LAST (Capítulo 3).

LT x, LEFT x (Capítulo 1).

≠LIST x y (como LIST, Capítulo 3).

≠LOCAL x (como LOCAL, Capítulo 3).

LPUT x y (Capítulo 3).

MAKE x y (Capítulo 3).

MEMBER x y

É gerada a mensagem TRUE se o elemento “x” está compreendido na lista “y”; em caso contrário resulta FALSE.

NOT Predicado

Gera uma decisão do tipo TRUE se o predicado for FALSE, e vice-versa.

NUMBER x

Gera uma decisão do tipo TRUE se o objeto “x” for um número.

≠OR (como BITOR, Capítulo 5).

OP, OUTPUT (Capítulo 3).

PAUSE

Introduz uma pausa na execução do procedimento.

PO x (Capítulo 2).

POALL (como PO ALL, Capítulo 2).

POS

Devolve a atual posição da tartaruga.

POTS x (Capítulo 2).

PRIMITIVEP x

No caso de que o argumento “x” corresponda ao nome de uma primitiva da linguagem é gerada a mensagem TRUE (FALSE no caso contrário).

≠PRINT x (como PRINT, Capítulo 3).

≠PRODUCT x y

Calcula o produto dos argumentos.

QUOTIENT x y

Efetua a divisão (x/y) e trunca seu resultado.

RANDOM n (Capítulo 5).

RC, READCHARACTER (Capítulo 3).

REMAINDER X Y

Devolve o resto da divisão entre os argumentos (“x” dividido por “y”).

REPEAT X Y (Capítulo 4).

RT x, RIGHT x (Capítulo 1).

ROUND X (Capítulo 5).

RUN X (Capítulo 2).

≠SENTENCE x y (como SENTENCE, Capítulo 3).

SETBEG x

Dá a cor correspondente ao código “x” à página gráfica.

SETCURSOR x

Conduz o cursor à posição “x”.

SHOW x

Imprime o argumento acompanhado por RETURN.

SQRT x (Capítulo 5).

STOP

Pára a execução do procedimento e devolve o controle ao programa que o chamou.

≠SUM x y

Calcula e devolve o valor obtido somando os dois argumentos.

BPT

Introduz o operador em ambiente monitor, permitindo a programação em linguagem máquina.

.CONTENTS

Gera as listas dos nomes de todas as listagens dos procedimentos e outras palavras definidas.

.DEPOSIT x y

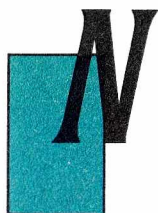
Copia o valor inteiro positivo (ou nulo) “y” na célula de memória número “x”.

.EXAMINE x

Devolve o valor lido na célula de memória número “x”.

APÊNDICE A

MENSAGENS DE ERRO



este apêndice examina-se as mensagens com as quais nos avisa o LOGO no caso de que o programa apresente erros sintáticos e/ou lógicos.

As análises são desenvolvidas sobre os sistemas do Commodore 64 e Apple II.

Commodore 64

(01): (Procedimento) didn't output

Esta mensagem aparece quando um procedimento ou uma primitiva que tem necessidade de argumentos estão sendo chamadas com outros procedimentos ou primitivas em lugar dos valores numéricos requeridos.

Exemplo:

INSTRUÇÃO: QUADRADO FORWARD 100

erro: QUADRADO didn't output

(02): (Primitiva) doesn't like (dados) as input

Indica uma operação entre dois tipos de dados incompatíveis, por exemplo, uma multiplicação entre um número e uma lista.

(03): (Primitiva) doesn't like (dados) as input it expects true or false

Aparece nas instruções if...then... else... not, allof, anyof, em que o computador espera uma posição verdadeira ou falsa e encontra-se na situação de dever valorizar uma expressão do tipo:

if: x — 10 then print [final]

que não é verdadeira nem falsa; a contestação será:

if doesn't like 2 as input. it expects true or false

(04): (mensagem) in line
(linha) at level (nível) of (nome procedimento)

Este é um tipo de mensagem que o intérprete vai construindo; expõe o nível no qual foi parado o programa, a linha equivocada e o tipo de erro.

O erro que segue é um exemplo deste.

(05): there is no procedure named (procedimento), in line (linha do Programa) at level (x) of (procedimento).

Foi chamado um procedimento inexistente na linha do programa listado, a nível de operações (x) do procedimento chamado. Por exemplo:

There is no procedure named fd 100, in line fd 100
at level of quadrado

(06): (nome) is a logo primitive.

O LOGO adverte que não é possível definir nosso procedimento com o nome escolhido, já que representa uma primitiva da linguagem.

(07): (Procedimento) needs more inputs.

O procedimento chamado tem necessidade de mais dados.

(08): (Primitiva) needs more inputs.

A primitiva chamada tem necessidade de mais dados.

(09): (Operador aritmético) needs something before it.

Caso tenha esquecido um operador aritmético em um cálculo numérico.

Exemplo:

Print/21

Contestação: “/needs something before it”.

Isto é, falta o número que tem que ser dividido por 21.

(10): (Primitiva) should be used only inside a Procedure.

Adverte que a primitiva empregada por nós não pode ser utilizada de maneira direta, mas só em um procedimento.

Exemplo:

output should be used inside a Procedure.

(11): can't divide by zero

A mensagem 11 indica que está sendo levada a cabo uma divisão por zero (absurdo desde o ponto de vista matemático) que o computador não pode efetuar.

(12): disk error

Erro devido a unidade de disco.

(13): end should be used only in the editor

Indica o uso equivocado da primitiva, “END”, como por exemplo “Print 8 end” pressionando de maneira direta.

(14): else is out of place

Com esta mensagem assinala-se a presença de um “else” no interior de uma instrução sem “if”.

Recordemos que, de fato, não é possível escrever um else fora da instrução “if... else”.

(15): file not found

O arquivo requerido não foi encontrado; aconselha-se a examinar o diretório (catálogo) para estarem seguros da presença do arquivo.

(16): line given to define too long

O argumento de “define” é superior a 256 caracteres e, portanto, é recusado pelo computador.

(17): line given to repeat too long

O mesmo erro anterior com “repeat” no lugar de “define”.

(18): line given to run too long

Como o número 16 com “run” no lugar de “define”.

(19): no storage left!

Esgotou a memória que se encontrava à nossa disposição. O computador nos comunica para não tentarmos memorizar novas coisas antes de termos apagado alguns procedimentos com ERASE.

(20): number too large

O argumento numérico da operação é demasiadamente grande.

(21): number too small

O argumento numérico da operação é demasiadamente pequeno.

(22): procedure nesting too deep

Foi introduzido mais de 200 procedimentos, número máximo aceito pelo Commodore 64.

(23): the disk is full

O disco em que estamos registrando os programas está completo.

(24): the disk is write protected

O disco está protegido da escrita; não gravou nada.

(25): there is no disk in the disk drive

O disco não está presente em sua unidade.

(26): turtle out of bounds

A tartaruga passou dos limites de altura e/ou largura da tela.

Apple II

(01): (Procedimento) is already define

O procedimento já foi definido.

(02): number too big

O valor numérico é excessivamente grande.

(03): (Símbolo) isn't a procedure

Não é uma palavra do vocabulário LOGO, isto é, não existe este procedimento.

(04): (Símbolo) isn't a word

O especificado entre parênteses não é uma primitiva.

(05): (Procedure) can't be use in a procedure

Não foi utilizado corretamente o procedimento nomeado entre parênteses.

(06): (Símbolo) is a Primitive

Esta mensagem aparece quando se tenta definir um procedimento com o nome de uma primitiva.

(07): (Símbolo) is undefined

O que estamos buscando não foi definido.

(09): I'm having trouble with the disk

Há problemas no que se refere à leitura de disco.

(10): disk full

O disco está completo; é necessário introduzir outro ou abrir espaço no mesmo.

(11): Can't divide

Se está efetuando uma divisão por zero ou uma operação matematicamente impossível.

(12): end of data

Terminaram os dados.

(13): file already exists

O arquivo que queremos criar (em disco) já existe.

(14): file locked

Assinala que o arquivo já encontra-se “gravado” em disco.

(15): file not found

O arquivo requerido não foi encontrado, não está no disco introduzido ou é necessário colocar “on line” a unidade de disco.

(16): file is wrong type

Foi requerido (ou introduzido) um arquivo de tipo equivocado.

(17): no more file buffers

Não é possível acrescentar mais instruções. O buffer está completo.

(18): (Símbolo) not found

Não foi encontrado o que se pediu.

(19): (Símbolo) is not true or false

A primitiva não é verdadeira nem falsa. Aparece nas propostas de “if... then... else” e de maneira particular, em “anyof” e “allof”.

(20): Pausing...

Sinal de espera.

(21): stopped

Interrompemos um procedimento em marcha pressionando CTRL + 9.

(22): not enough inputs to (Procedimento)

Não são suficientes as entradas proporcionadas ao procedimento em marcha.

(23): too many inputs to (Procedure)

O contrário do anterior.

(24): too much inside parentheses

Demasiadas instruções estão fechadas entre parênteses.

(25): can only do that in a Procedure

É possível operar de tal maneira só no interior de um procedimento.

(26): turtle out of bounds

A tartaruga passou dos limites da página gráfica.

(27): don't know how to (símbolo)

O computador não reconhece esta instrução como um procedimento nem como uma primitiva: controlar a sintaxe.

(28): (Símbolo) has no value

O “símbolo” não tem um valor definido.

(29): don't know what to do with (símbolo)

O computador não sabe como executar esta instrução: controlar a sintaxe.

(30): disk is write protected

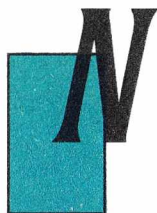
O disco está protegido da escrita. Não gravou nada.

(31): (Procedure) doesn't like (símbolo) as input

Este procedimento não é admissível como entrada.

APÊNDICE B

O CÓDIGO ASC II



este apêndice são mostradas as tabelas relativas aos códigos ASC II do COMMODORE e do APPLE II.

Os comandos que permitem a utilização dos caracteres ASC II são:

CHAR

Gera o valor numérico correspondente ao caracter desejado; sua sintaxe é:

CHAR caracter

ASC II

Devolve o caracter correspondente ao valor proposto segundo a sintaxe:

ASC II número

Na primeira coluna das tabelas é dado o valor numérico decimal, e na segunda, o caracter ASC II correspondente.

COMMODORE

CODIGO	CARACTER	CODIGO	CARACTER
00		26	
01		27	
02		28	
03		29	
04		30	
05		31	
06		32	
07		33	!
08	desactiva 	34	"
09	activa 	35	#
10		36	\$
11		37	%
12		38	&
13		39	.
14		40	(
15		41	)
16		42	*
17		43	+
18		44	,
19		45	-
20		46	.
21		47	/
22		48	0
23		49	1
24		50	2
25		51	3

COMMODORE (continuação)

CODIGO	CARACTER		CODIGO	CARACTER
52	4		78	N
53	5		79	O
54	6		80	P
55	7		81	Q
56	8		82	R
57	9		83	A
58	:		84	T
59	;		85	U
60	<		86	V
61	=		87	W
62	>		88	A
63	?		89	Y
64	@		90	Z
65	A		91	[
66	B		92	£
67	C		93	]
68	D		94	↑
69	E		95	←
70	F		96	
71	G		97	
72	H		98	
73	I		99	
74	J		100	
75	K		101	
76	L		102	
77	M		103	

COMMODORE (Continuação)

CODIGO	CARACTER	COGIDO	CARACTER
104		130	
105		131	
106		132	
107		133	f1
108		134	f3
109		135	f5
110		136	f7
111		137	f2
112		138	f4
113		139	f6
114		140	f8
115		141	SHIFT RETURN
116		142	MAIUSCULAS
117		143	
118		144	BLK
119		145	CRSR
120		146	RVS OFF
121		147	CLR HOME
122		148	INST DEL
123		149	
124		150	
125		151	
126		152	
127		153	
128		154	
129		155	

COMMODORE (Continuação)

CODIGO	CARACTER	CODIGO	CARACTER
156		182	
157		183	
158		184	
159		185	
160		186	
161		187	
162		188	
163		189	
164		190	
165		191	
166		192	
167		193	
168		194	
169		195	
170		196	
171		197	
172		198	
173		199	
174		200	
175		201	
176		202	
177		203	
178		204	
179		205	
180		206	
181		207	

COMMODORE (continuação)

CODIGO	CARACTER	CODIGO	CARACTER
208		232	
209		233	
210		234	
211		235	
212		236	
213		237	
214		238	
215		239	
216		240	
217		241	
218		242	
219		243	
220		244	
221		245	
222		246	
223		247	
224		248	
225		249	
226		250	
227		251	
228		252	
229		253	
230		254	
231		255	

APPLE

CODIGO	CARACTER	CODIGO	CARACTER
00	@	26	^Z
01	^A	27	ESC
02	^B	28	^ \
03	^C	29	^]
04	^D	30	^ ↑
05	^E	31	^ _
06	^F	32	SPACE
07	BELL	33	!
08	^H	34	"
09	^I	35	#
10	^J	36	\$
11	^K	37	%
12	^L	38	&
13	RETURN	39	'
14	^N	40	(
15	^O	41	)
16	^P	42	*
17	^Q	43	+
18	^R	44	,
19	^S	45	-
20	^T	46	.
21	^U	47	/
22	^V	48	0
23	^W	49	1
24	^X	50	2
25	^Y	51	3

APPLE (Continuação)

CODIGO	CARACTER		CODIGO	CARACTER
54	4		78	N
53	5		79	O
54	6		80	P
55	7		81	Q
56	8		82	R
57	9		83	S
58	:		84	T
59	;		85	U
60	<		86	V
61	=		87	W
62	>		88	X
63	?		89	Y
64	@		90	Z
65	A		91	[
66	A		92	\
67	C		93	]
68	D		94	^
69	E		95	-
70	F		96	\
71	G		97	a
72	H		98	b
73	I		99	c
74	J		100	d
75	K		101	e
76	L		102	f
77	M		103	g

APPLE (Continuação)

CODIGO	CARACTER		CODIGO	CARACTER
104	h		117	u
105	i		118	v
106	j		119	w
107	k		120	x
108	l		121	y
109	m		122	z
110	n		123	{
111	o		124	
112	p		125	}
113	q		126	~
114	r		127	DEL
115	s		129	@
116	t			

BIBLIOGRAFIA

1. LOGO - Introdução ao poder do ensino através da programação
Goodyear
2. LOGO for the Apple II
Abelson
3. Apple LOGO
Abelson



Uma das linguagens de programação mais "moderna" nos últimos tempos, é sem dúvida, o LOGO. Nasceu como consequência de diversas pesquisas sobre inteligência artificial, porém agora está disponível em versões para os mais conhecidos e populares computadores domésticos e pessoais.

Fácil de assimilar e com uma aprendizagem muito rápida o LOGO é, além disso, uma linguagem extremamente eficaz. Junto a uma capacidade gráfica invejável, que permite a realização de complexos desenhos, dispõe de um manejo simples, clareza de textos e cadeias de caracteres, além de peculiaridades tão interessantes como a recursividade e outras muitas que neste volume da B.B.I. lhes explicaremos em profundidade com uma grande quantidade de provados exemplos práticos.

B.B.I.

GO

10