
CAPÍTULO 3 - Ações especializadas

As operações em Logo, no que tange, ao universo da *geometria da tartaruga* vão além das primitivas apresentadas. Neste contexto, desenhar um quadrado ou triângulo não é difícil apesar de maçante, mas desenhar figuras geométricas com maior número de lados poderá se tornar inviável. É neste sentido que entram outras primitivas de apoio especializadas em facilitar o uso de certos recursos da linguagem, as quais são apresentadas ao longo deste capítulo.

3.1 - Repetições

O desenho de um quadrado pode ser produzido com uso básico da combinação das primitivas **FORWARD**, **BACK**, **RIGHT** ou **LEFT** repetidos quatro vezes. A combinação de uso das primitivas é de cunho pessoal ou da necessidade do que se deseja efetivamente fazer.

Para facilitar repetições Logo possui uma primitiva chamada **REPEAT** com a finalidade de repetir um grupo de instruções definidas entre colchetes um certo número de vezes. O comando (primitiva) **REPEAT** para ser usado deve seguir a seguinte estrutura sintática:

REPEAT <n> [<instruções>]

Onde, o indicativo "**n**" refere-se a quantidade de repetições e o indicativo "**instruções**" refere-se as ações que se deseja executar. Para desenhar um quadrado, por exemplo, use a instrução:

REPEAT 4 [FD 80 RT 90]

A figura 3.1 mostra a apresentação de um quadrado desenhado a partir do uso da instrução "**REPEAT 4 [FD 80 RT 90]**" formada pelas primitivas **REPEAT**, **FD** e **RT**.

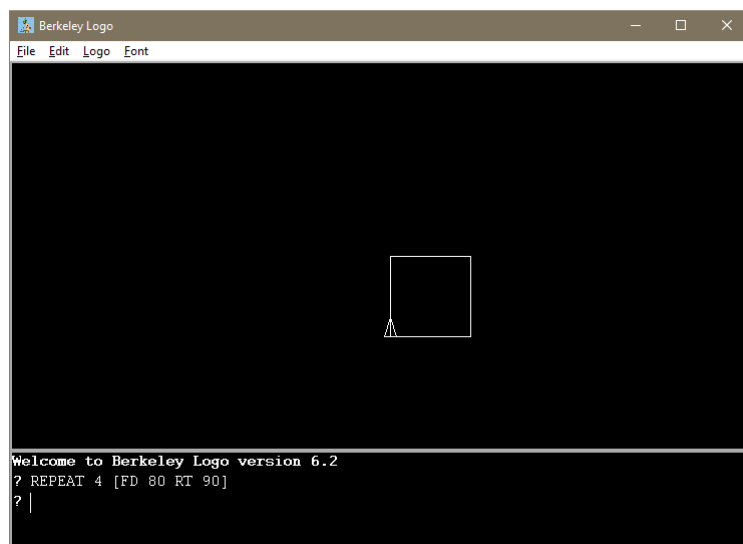


Figura 3.1 - Quadrado apresentado a partir do uso do comando "REPEAT"

Veja, por exemplo, a apresentação de um triângulo equilátero com o uso do comando **REPEAT**:

CS

REPEAT 3 [FD 80 RT 120]

A figura 3.2 mostra a apresentação de um triângulo equilátero desenhado a partir do uso do comando **REPEAT**.

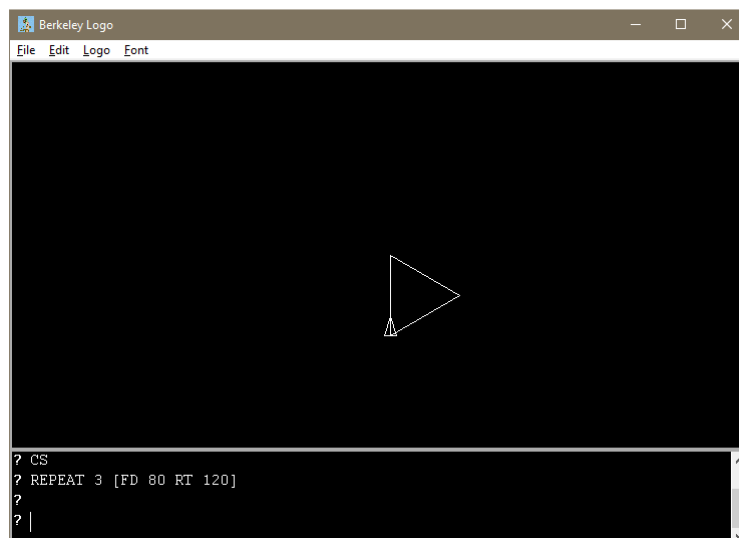


Figura 3.2 - Triângulo equilátero apresentado a partir do uso do comando "REPEAT"

Note que para desenhar um triângulo equilátero usou-se a medida de graus como "120". No ambiente Logo usa-se a medida externa da figura. Se fosse usado a medida interna o valor para um triângulo equilátero seria "60".

Agora veja o desenho de um pentágono. Assim sendo, execute a instrução:

```
CS
REPEAT 5 [FD 80 RT 72]
```

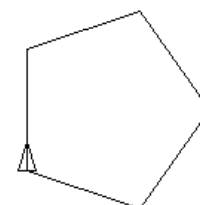


Figura 3.3 - Pentágono

A figura 3.3 mostra o resultado obtido.

Agora pode estar em sua mente a pergunta que não quer calar. *Como saber exatamente o valor dos graus a ser usado para a definição de certa figura geométrica?* Recorde que os comandos **RIGHT (RT)** e **LEFT (LT)** podem ser usados com valores entre "0" e "360". No universo Logo a menor figura geométrica possui três lados, que é um triângulo, por sua vez a maior figura geométrica é a circunferência com trezentos e sessenta lados.

Veja a instrução:

```
CS
REPEAT 360 [FD 1 RT 1]
```

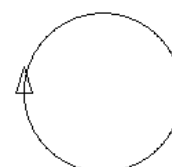


Figura 3.4 - Circunferência

A figura 3.3 mostra o resultado obtido.

A resposta à pergunta é que basta você pegar o número de lados desejados e dividi-los por "360" para obter o valor exato de graus a ser usado para desenhar determinada figura geométrica.

A partir dessas orientações fica fácil desenhar qualquer figura geométrica plana compreendida entre "3" e "360" lados, ou seja, desenhar um polígono. Como ilustração veja a tabela 3.1 com os nomes de alguns polígonos, seus respectivos lados e o valor calculado dos ângulos de rotação.

POLÍGONO	LADOS	360 / LADOS	ÂNGULO
TRIÂNGULO	3	360 / 3	120.00
QUADRILÁTERO	4	360 / 4	90.00
PENTÁGONO	5	360 / 5	72.00
HEXÁGONO	6	360 / 6	60.00
HEPTÁGONO	7	360 / 7	51.43
OCTÓGONO	8	360 / 8	45.00
ENEÁGONO	9	360 / 9	40.00
DECÁGONO	10	360 / 10	36.00
UNDECÁGONO	11	360 / 11	32.73
DODECÁGONO	12	360 / 12	30.00
TRIDECÁGONO	13	360 / 13	27.69
TETRADECÁGONO	14	360 / 14	25.71
PENTADECÁGONO	15	360 / 15	24.00
HEXADECÁGONO	16	360 / 16	22.50
HEPTADECÁGONO	17	360 / 17	21.18
OCTODECÁGONO	18	360 / 18	20.00
ENEDECÁGONO	19	360 / 19	18.95
ICOSÁGONO	20	360 / 20	18.00
TRIACONTÁGONO	30	360 / 30	12.00
TETRACONTÁGONO	40	360 / 40	9.00
PENTACONTÁGONO	50	360 / 50	7.20
HEXACONTÁGONO	60	360 / 60	6.00
HEPTACONTÁGONO	70	360 / 70	5.14
OCTOCONTÁGONO	80	360 / 80	4.50
ENEACONTÁGONO	90	360 / 90	4.00
HECTÁGONO	100	360 / 100	3.60
CIRCUNFERÊNCIA	360	360 / 360	1.00

Tabela 3.1 - Algumas medidas de ângulos

Além da produção de figuras geométricas planas há outras possibilidades de geração de imagens na linguagem. Mas este é um assunto a ser visto um pouco mais adiante.

3.2 - Procedimentos

Já foi comentado que o fato de Logo apresentar uma figura não significa em absoluto que Logo saiba fazer a figura. Para que Logo aprenda a fazer uma figura é necessário que você ensine Logo a fazer.

A ideia de "ensinar" um computador a realizar certa tarefa, de modo que o computador aprenda e tenha esse "conhecimento" armazenado para uso futuro chamasse comumente de "inteligência artificial".

A "inteligência artificial" é a junção de duas palavras distintas com significados absolutos: *inteligência* é a faculdade de conhecer, compreender e aprender a resolver problemas e de adaptá-los a novas situações e *artificial* significa aquilo que não revela naturalidade, sendo produzido pelo ser humano e não pela natureza. Desta forma, pode-se entender que a "inteligência artificial" é a fabricação de conhecimentos específicos processados por computadores e controlados dentro das bases da Ciência da Computação.

Uma forma de "ensinar" a linguagem Logo a realizar tarefas de modo que se desenvolva a ideias de "inteligência artificial" é fazer uso de rotinas de scripts de programas chamadas de **procedimentos**.

A criação de procedimentos em *Logo* é produzida com o uso do comando **TO** a partir da seguinte sintaxe:

TO <nome>

Onde, o indicativo "**nome**" refere-se a definição de um nome de identificação para o procedimento a ser usado no campo de comandos, dados e instruções como se fosse uma primitiva da linguagem. Note que se tem duas categorias de primitivas na linguagem Logo: as primitivas internas (pertencente a linguagem) e as primitivas externas (definidas por humanos) para adaptar a linguagem Logo a necessidades particulares.

Cada procedimento criado é a definição de um grau de conhecimento que o ambiente Logo pode artificialmente obter dando-lhe um nível de inteligência, além do que foi projetado.

Antes de realizar a próxima etapa efetue o fechamento do interpretador e proceda uma nova chamada para iniciar o desenvolvimento do procedimento a seguir em uma instância nova de trabalho. Assim sendo, veja a definição de um procedimento que desenhe um quadrado chamado "**QUAD1**". Para tanto, execute a instrução:

TO **QUAD1**

Ao ser executada a instrução anterior é apresentada ocorre a apresentação do símbolo ">" abaixo do símbolo "?" como mostra a figura 3.5. Isso indica que você está no modo de edição de procedimentos.

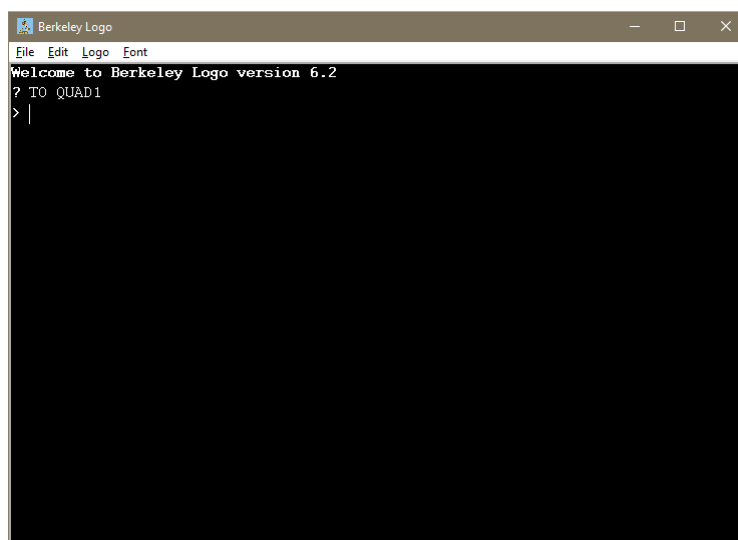
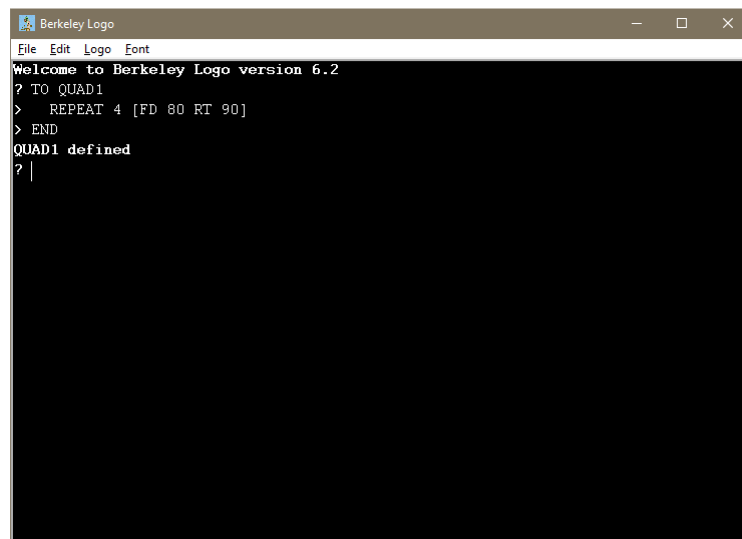


Figura 3.5 - Apresentação do modo de edição de procedimentos

Neste momento, escreva a instrução a executada no procedimento. Por exemplo, coloque a instrução **REPEAT 4 [FD 80 RT 90]** como mostrado na figura 3.6 após acionar dois espaços em branco para definir a endentação do código, acione "<Enter>". Para finalizar a edição escreva a primitiva **END** na primeira posição da linha apresentada.



```
Berkeley Logo
File Edit Logo Font
Welcome to Berkeley Logo version 6.2
? TO QUAD1
> REPEAT 4 [FD 80 RT 90]
> END
QUAD1 defined
? |
```

Figura 3.6 - Apresentação da finalização da edição do procedimento

Assim que o procedimento é devidamente registrado em memória ocorre a apresentação na área da mensagem "**QUAD1 defined**", ou seja "*QUADRADO* definido" e o *prompt* "?" volta a ser apresentado.

A partir deste instante o Logo sabe desenhar um quadrado por meio da primitiva derivada chamada "**QUAD1**". Assim sendo, no capo de comandos, dados e instruções execute a instrução:

QUAD1

A figura 3.7 mostra o resultado obtido a partir da definição interna do conhecimento do que é um quadrado para a linguagem Logo.



Figura 3.7 - Quadrado desenhado a partir da definição de um conhecimento artificial

Assim como é possível repetir primitivas internas também é possível repetir procedimentos (que são rotinas definidas a partir de ações derivadas). Por exemplo, imagine querer repetir o

procedimento "**QUAD1**" por quatro vezes com ângulo de giro de "**90**" graus para a esquerda. Assim sendo, execute a instrução:

```
CS
CT
REPEAT 4 [QUAD1 LT 90]
```

A figura 3.8 mostra o resultado desta operação.

Dependendo do valor de ângulo de giro é possível formar imagens geométricas muito diferentes como um conjunto de quadrados intercalados a partir da execução da instrução:

```
CS
CT
REPEAT 8 [QUAD1 LT 45]
```

A figura 3.9 mostra o resultado desta operação.

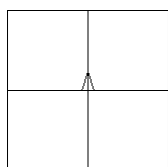


Figura 3.8 - Quadrado repetido

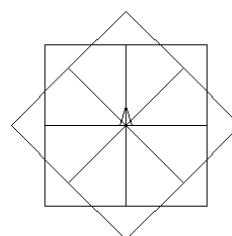


Figura 3.9 - Quadrados intercalados

Na sequência execute a instrução "**TO PENTA1**" e informe como já orientado as seguintes instruções "**REPEAT 5 [FD 50 RT 360 / 5]**" e "**END**". Veja na figura 3.10 esta ocorrência.

```
? TO PENTA1
> REPEAT 5 [FD 50 RT 360 / 5]
> END
PENTA1 defined
?
```

Figura 3.10 - Definição do procedimento "PENTA1"

Agora execute a instrução:

```
CS
CT
PENTA1
```

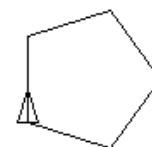


Figura 3.11 - Pentágono

A figura 3.11 mostra o resultado desta operação.

A partir da definição da imagem de um pentágono veja os dois exemplos de imagens geradas a partir do procedimento "**PENTA1**".

Execute primeiro a instrução:

```
CS
CT
REPEAT 5 [PENTA1 RT 72]
```

A figura 3.12 mostra o resultado desta operação.

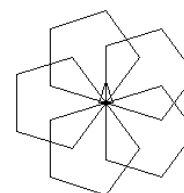


Figura 3.12 - Pentágono 1

Depois execute a instrução:

```
CS
CT
REPEAT 10 [PENTA1 RT 36]
```

A figura 3.13 mostra o resultado desta operação.

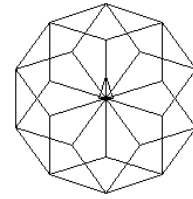


Figura 3.13 - Pentágono 2

3.3 - Sub-rotinas

A definição de procedimentos trás para a linguagem Logo a possibilidade de se fazer a criação de comandos derivados. Todo comando derivado é baseado sobre o uso de alguma primitiva padrão da linguagem.

O fato que se definir procedimentos e dar-lhes a mesma capacidade que possuem os comandos traz muita mobilidade, pois um procedimento pode ser usado da mesma forma que um comando é usado no estabelecimento de instruções. Desta forma, torna-se naturalmente possível usar um procedimento dentro de outro procedimento, o que caracteriza a existência de sub-rotinas.

Para experimentar esta ideia considere realizar o desenho de uma flor com oito pétalas. Veja que uma flor, é grosso modo, um conjunto de pétalas. Uma pétala por sua vez pode ser a junção de duas meias circunferências.

Veja pelo que é descrito que o desenho de uma flor será realizado a partir de três procedimentos interligados. Assim sendo, considere os procedimentos **"MEIOCIRC"**, **"PETALA"** e **"FLOR"** indicados a seguir:

```
TO MEIOCIRC
  REPEAT 90 [FD 1 RT 1]
END
```

```
TO PETALA
  MEIOCIRC
  RT 90
  MEIOCIRC
END
```

```
TO FLOR
  CS
  REPEAT 8 [PETALA RT 45]
END
```

Note que o procedimento **"FLOR"** faz uso do procedimento **"PETALA"** por oito vezes, o procedimento **"PETALA"** faz uso do procedimento **"MEIOCIRC"** por duas vezes. Atente para o fato de que o efeito em cascata para se fazer o desenho da flor é o que se chama de sub-rotinas, ou seja, **"MEIOCIRC"** é uma sub-rotina usada por **"PETALA"** que, por sua vez, é uma sub-rotina usada por **"FLOR"**. Na sequência execute a instrução:

```
FLOR
```

A figura 3.14 mostra o resultado da execução do procedimento "FLOR".

Imagine desenhar uma flor mais encorpada com um número maior de pétalas, por exemplo vinte pétalas. Para tanto, execute a instrução:

CS

REPEAT 20 [PETALA RT 36]

A Figura 3.15 mostra a imagem de uma flor com vinte pétalas.

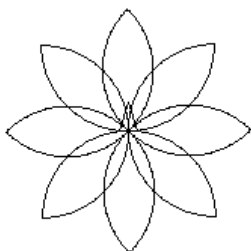


Figura 3.14 - Execução do procedimento "FLOR"

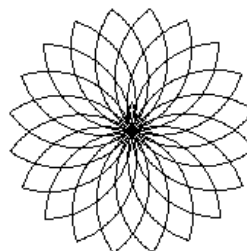


Figura 3.15 - Execução do procedimento "PETALA" para criar flor

A partir da definição de sub-rotinas é possível diminuir muito a carga de trabalho, pois é possível criar comandos derivados e especializados em desenhar parte de uma imagem e formar, a partir daí, imagens com as combinações dos comandos definidos em conjunto com as primitivas e mesmo as funções da linguagem Logo.

3.4 - Variável

Você viu anteriormente o uso de instruções baseadas em primitivas simples e primitivas com o uso de parâmetros.

No tópico anterior você aprendeu a escrever seus próprios comandos a partir da definição de procedimentos. Os procedimentos, então criados, se assemelharam aos comandos simples da linguagem Logo. É chegada a hora de aprender a definir os próprios parâmetros para certo procedimento. É aqui que entra a ideia de uso de *variáveis*.

Uma *variável* do ponto de vista mais amplo para a computação é uma região de memória que armazena determinado valor por certo espaço de tempo. O valor armazenado em memória poderá ou não ser usado em ações de processamento.

Para criar variáveis em memória usa-se a primitiva **MAKE** a partir da estrutura sintática:

MAKE <"variável"> <valor>

Onde, o indicativo "**variável**" refere-se ao nome de identificação da variável definido após o uso do símbolo de aspa inglesa (") e "**valor**" a definição do valor associado ao nome da variável. Os caracteres maiúsculos e minúsculos interferem no nome de uma variável.

Para apresentar o conteúdo de uma variável é importante que seja usado antes do nome da variável o símbolo de dois pontos (:). Por exemplo, veja a definição e apresentação do conteúdo de uma variável chamada "**PAISES**" com o valor **Brasil, Franca** (França, pois UCBLLogo não aceita acentuação) e **Argentina**.

```
MAKE "PAISES [Brasil Franca Argentina]
PRINT :PAISES
```


A figura 3.16 mostra o resultado da ação anterior na área de ação do modo texto.

```
? MAKE "PAISES [Brasil Franca Argentina]
? PRINT :PAISES
Brasil Franca Argentina
? |
```

Figura 3.16 - Definição e apresentação de variável

Veja outro exemplo de definição de variável e visualização do nome *Manzano*:

```
MAKE "NOME "Manzano
PRINT :NOME
```

Perceba que para imprimir o conteúdo de uma variável usa-se o símbolo dois pontos. Cuidado em não fazer uso do símbolo aspa inglesa. Caso use o nome de uma variável com aspa inglesa a linguagem Logo não entenderá que se trata de uma variável e considerará como sendo apenas uma palavra a ser escrita usando o nome da variável.

Escreva a instrução:

```
PRINT :PAISES
```

E em seguida, escreva a instrução

```
PRINT "PAISES
```

Veja na figura 3.17 a diferença de resultado apresentado para cada uma das formas de acesso.

```
? PRINT :PAISES
Brasil Franca Argentina
? PRINT "PAISES
PAISES
? |
```

Figura 3.17 - Diferença no uso dos símbolos dois pontos e aspa inglesa

No entanto, é possível fazer com que ocorra a apresentação do conteúdo de uma variável utilizando-se o símbolo aspa inglesa. Mas neste caso, é necessário fazer uso da primitiva **THING** antes no nome da variável identificada com aspa inglesa. Observe a instrução seguinte:

```
PRINT THING "PAISES
```

O efeito no uso da primitiva **THING** antes no nome da variável com aspa inglesa, ou seja, a instrução **PRINT THING "PAISES** é exatamente o mesmo resultado obtido a partir da execução da instrução **PRINT :PAISES**. Veja a figura 3.18.

```
PAISES
? PRINT THING "PAISES
Brasil Franca Argentina
? |
```

Figura 3.18 - Resultado da instrução "PRINT THING "PAISES"

A partir de uma visão geral da definição e uso de variáveis é possível criar um procedimento que, por exemplo, desenhe um quadrado em que o tamanho da figura será informado no uso e não dentro do procedimento. Assim sendo, informe o código do procedimento a seguir:

```
TO QUAD2 :TAMANHO
  REPEAT 4 [FD :TAMANHO RT 90]
END
```

Em seguida execute as seguintes instruções:

```
CT CS
QUAD2 40
QUAD2 60
QUAD2 80
QUAD2 90
```

Perceba que para cada chamada do procedimento "QUAD2" foi usado em conjunto um parâmetro diferente proporcionando efeitos diferenciados como apresenta a figura 3.19.

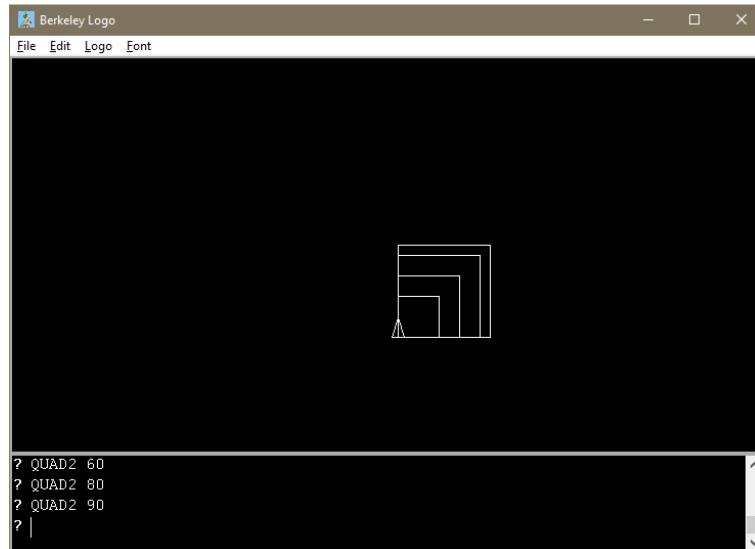


Figura 3.19 - Resultado a partir do uso de procedimento com parâmetro

A partir desses recursos é possível criar um procedimento mais "inteligente" que a partir da informação do tamanho de traço e quantidade de lados desenha qualquer figura entre "3" e "360" graus. Para tanto, entre o seguinte código:

```
TO POLIS :TAMANHO :LADO
  REPEAT :LADO [FD :TAMANHO RT 360 / :LADO]
END
```

A partir desta definição pode-se fazer uso, por exemplo, das seguintes instruções:

```
POLIS 80 3 - Desenha triângulo
POLIS 80 4 - Desenha quadrado
POLIS 80 5 - Desenha pentágono
POLIS 80 6 - Desenha hexágono
POLIS 1 360 - Desenha circunferência
```

A partir dessas orientações você já tem em mãos os recursos essenciais para a definição de parâmetros e de procedimentos. Assim sendo, já é possível criar comandos simples e comandos com parâmetros.

3.5 - Decisão

Logo é uma linguagem que possui internamente um nível de "inteligência" interessante. Entre as diversas possibilidades da linguagem há a primitiva **IF** que permite tomar decisões a partir do estabelecimento de duas formas de citação. Observe a seguinte estrutura sintática:

```
IF <condição> <[ação verdadeira]>
IFELSE <condição> <[ação verdadeira]> <[ação falsa]>
```

Onde, o indicativo "**condição**" (não importando **IF** ou **IFELSE**) refere-se a definição de uma relação lógica que devolve como resposta um valor "*falso*" representado pelo valor lógico "**FALSE**" ou um valor "*verdadeiro*" representado pelo valor lógico "**TRUE**". O indicativo "**[ação verdadeira]**" corresponde a definição da ação a ser realizada dentro de uma lista caso a condição seja verdadeira, tanto para **IF** como para **IFELSE**. O indicativo "**[ação falsa]**" é executado quando a condição para **IFELSE** for falsa. A primitiva **IF** é usada na tomada de decisão simples e a primitiva **IFELSE** é usada na tomada de decisão composta.

Para realizar a definição da condição usada com as primitivas **IF** e **IFELSE** é necessário levar em consideração o uso de operadores específicos para o estabelecimento das relações entre variáveis com variáveis ou de variáveis com constantes. A tabela 3.2 descreve o conjunto de *operadores relacionais* usados no estabelecimento de condições.

OPERADOR	SIGNIFICADO
=	IGUAL A
>	MAIOR QUE
<	MENOR QUE
>=	MAIOR OU IGUAL A
<=	MENOR OU IGUAL A
<>	DIFERENTE DE

Tabela 3.2 - Operadores relacionais

Para realizar um teste rápido sobre o uso de *operadores relacionais* execute as seguintes instruções observado a apresentação dos valores **FALSE** e **TRUE**:

```
PRINT 1 = 1 (mostra: TRUE como "verdadeiro")
PRINT 1 <> 1 (mostra: FALSE como "falso")
PRINT 1 < 2 (mostra: TRUE como "verdadeiro")
PRINT 1 > 2 (mostra: FALSE como "falso")
PRINT 1 >= 1 (mostra: TRUE como "verdadeiro")
PRINT 2 <= 2 (mostra: TRUE como "verdadeiro")
```

Como exemplo no uso de tomada de decisão composta considere o desenvolvimento de um procedimento chamado "**MAXIMO**" que retorne o maior valor numérico a partir de dois valores. Desta forma, entre o seguinte código:

```
TO MAXIMO :A :B
  IFELSE :A > :B [PR :A] [PR :B]
END
```

Em seguida execute separadamente as instruções:

```
MAXIMO 9 11
MAXIMO 15 8
```

Veja na figura 3.20 a apresentação do resultado das operações.



```
? MAXIMO 9 11
11
? MAXIMO 15 8
15
?
```

Figura 3.20 - Resultado de uma tomada decisão

No sentido de demonstrar outro exemplo de tomada de decisão composta considere um procedimento chamado **"PAR"** que apresente a mensagem **"Ok"** se o valor numérico for par ou mostre a mensagem **"Erro"** caso o valor numérico não seja par. Desta forma, entre o seguinte código:

```
TO PAR :N
  IFELSE (MODULO :N 2) = 0 [PRINT [Ok]] [PRINT [Erro]]
END
```

Em seguida execute separadamente as instruções:

```
PAR 2
PAR 3
```

Veja na figura 3.21 a apresentação do resultado das operações.



```
? PAR 2
Ok
? PAR 3
Erro
?
```

Figura 3.21 - Resultado do procedimento "PAR"

Observe a partir desses dois exemplos a definição da condição em vermelho. No procedimento **"MAXIMO"** tem-se a definição de uma condição simples de uma variável com outra variável, mas no procedimento **"PAR"** a condição é mais complexa, pois para saber se um valor é par é necessário validar o resto da divisão com a função **MODULO** e comparar este valor com a constante **"0"** (zero). Os trechos marcados em verde referem-se a ação se a condição for verdadeira e ocre se a condição for falsa.

Como exemplo de tomada de decisão simples considere o procedimento chamado **"IMPAR"** que apresenta a mensagem **"Ok"** se o valor numérico for par e não mostre nada, caso contrário. Desta forma, entre o seguinte código:

```
TO IMPAR :N
  IF (MODULO :N 2) <> 0 [PRINT [Ok]]
END
```

Em seguida execute separadamente as instruções:

```
IMPAR 3
IMPAR 2
```

Veja na figura 3.22 a apresentação do resultado das operações.



```
? IMPAR 3
Ok
? IMPAR 2
?

```

Figura 3.22 - Resultado do procedimento "IMPAR"

A partir do conhecimento de tomada de decisão com o comando **IF** considere, como exemplo, um procedimento chamado **"FLORAL"** que aceite a entrada apenas de valores entre **"1"** e **"7"**.

Qualquer valor abaixo de **1** ou acima de **7** deve fazer com o procedimento mostre a mensagem de erro "**Use valores entre 1 e 7**". Para tanto, entre o seguinte código (não se preocupe com a primitiva **OR**, adiante a sua explicação:

```
TO FLORAL :N
  IFELSE (OR :N < 1 :N > 7) [
    PRINT [Use valores entre 1 e 7]
  ] [
    REPEAT 8 [
      RT 45
      REPEAT :N [
        REPEAT 90 [
          FD 2
          RT 2
        ]
        RT 90
      ]
    ]
  ]
END
```

O procedimento "**FLORAL**" é uma adaptação de exemplo Logo encontrado num material de aula do Professor Carlos Eduardo Aguiar do Centro de Educação Superior a Distância do Estado do Rio de Janeiro: "<http://omnis.if.ufrj.br/~carlos/infoenci/notasdeaula/roteiros/aula10.pdf>".

Veja que no procedimento "**FLORAL**" a estrutura do código está sendo definida em outro estilo de escrita. Neste caso, baseando-a com o uso de endentações no sentido de indicar visivelmente quem está dentro de quem (observe as cores). Nada impede, por exemplo de escrever o procedimento "**FLORAL**" de outras maneiras, mas a forma apresenta é um estilo que deixa o emaranhado de instruções mais claras.

Em seguida execute separadamente as instruções:

```
CS FLORAL 0
CS FLORAL 1
CS FLORAL 2
CS FLORAL 3
CS FLORAL 4
CS FLORAL 5
CS FLORAL 6
CS FLORAL 7
CS FLORAL 8
```

Veja na figura 3.23 a apresentação dos vários resultados obtidos a partir dos valores numéricos entre **1** e **7** válidas para o procedimento "**FLORAL**".

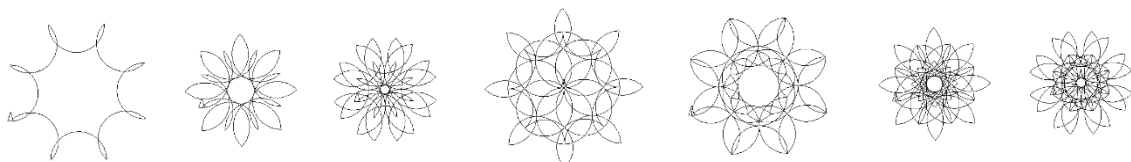


Figura 3.23 - Resultado do procedimento "**FLORAL**"

Além dos operadores relacionais são encontrados os operadores lógicos que auxiliam ações de tomada de decisão como é o caso do operador **OR** usado no procedimento "FLORAL". Além do operador **OR** existem os operadores lógicos **AND** e **NOT**. Atente para a tabela 3.3.

OPERADOR	SIGNIFICADO	RESULTADO
AND	CONJUNÇÃO	VERDADEIRO quando todas as condições forem verdadeiras.
OR	DISJUNÇÃO	VERDADEIRO quando pelo menos uma das condições for verdadeira.
NOT	NEGAÇÃO	VERDADEIRO quando condição for falsa e FALSO quando condição for verdadeira.

Tabela 3.3 - Operadores lógicos

Os operadores lógicos **AND** e **OR** permitem vincular em uma tomada de decisão duas ou mais condições. Já o operador lógico **NOT** faz a inversão do resultado lógico da condição a sua frente. A ordem de prioridade entre os operadores lógicos é: **NOT**, **AND** e **OR**. Observe as tabelas verdades a seguir para cada um dos operadores lógicos (**AND**, **OR** e **NOT**) e confronte o que é apresentado com o resumo descrito na tabela 3.3.

Operador de conjunção

A conjunção é a relação lógica entre duas ou mais condições que gera resultado lógico verdadeiro quando todas as proposições forem verdadeiras. A tabela 3.4 indica os resultados lógicos que são obtidos a partir do uso do operador lógico de conjunção "**AND**".

CONDIÇÃO 1	CONDIÇÃO 2	RESULTADO
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	FALSO

Tabela 3.4 - Tabela verdade para operador "AND"

Para demonstrar o uso do operador lógico de conjunção **AND** considere um procedimento chamado "TESTE_E" que receba um valor numérico e informe se este valor está ou não na faixa de "1" a "9". Desta forma, e entre o código:

```
TO TESTE_E :N
  IFELSE (AND :N >= 1 :N <= 9) [
    PR [Valor esta na faixa de 1 a 9.]
  ] [
    PR [Valor nao esta na faixa de 1 a 9.]
  ]
END
```

Em seguida execute separadamente as instruções a seguir:

```
TESTE_E 5
TESTE_E 0
TESTE_E 11
```

Procure fazer mais testes com outros valores.

Operador de disjunção

A disjunção é a relação lógica entre duas ou mais condições de tal modo que seu resultado lógico será verdadeiro quando pelo menos uma das proposições for verdadeira. A tabela 3.5 apresenta os resultados lógicos que são obtidos a partir do uso do operador lógico de conjunção "OR".

CONDIÇÃO 1	CONDIÇÃO 2	RESULTADO
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

Tabela 3.5 - Tabela verdade para operador "OR"

Para demonstrar o uso do operador lógico de disjunção **OR** considere um procedimento chamado "TESTE_OU" que receba um valor textual "FRIO" ou "QUENTE" e informe respectivamente as mensagens "Tempo ruim, proteja-se." ou "Tempo bom, aproveite.". Desta forma, entre o código:

```
TO TESTE_OU :TEMPO
  IFELSE (OR UPPERCASE :TEMPO = "FRIO UPPERCASE :TEMPO = "CHUVOSO) [
    PR [Tempo ruim, proteja-se.]
  ] [
    PR [Tempo bom, aproveite.]
  ]
END
```

Em seguida execute separadamente as instruções a seguir:

```
TESTE_OU "SOL
TESTE_OU "FRIO
TESTE_OU "QUENTE
TESTE_OU "CHUVOSO
```

Procure fazer mais testes com outros valores. Note o uso da função **UPPERCASE** para garantir que não importando o formato do texto informado este será transformado em texto maiúsculo antes de ser verificado na condição.

Operador de negação

A negação é a rejeição ou a contradição do todo ou de parte de um todo. Pode ser a relação entre uma condição "p" e sua negação "não-p". Se "p" for verdadeira, "não-p" é falsa e se "p" for falsa, "não-p" é verdadeira. A tabela 3.6 mostra os resultados lógicos que são obtidos a partir do uso do operador lógico de negação "NOT".

CONDIÇÃO	RESULTADO
VERDADEIRO	FALSO
FALSO	VERDADEIRO

Tabela 3.6 - Tabela verdade para operador "NOT"

Para demonstrar o uso do operador lógico de negação **NOT** leve em consideração um procedimento **"TESTE_NAO"** que receba um valor numérico e mostra o quadrado deste valor caso este valor *não seja maior que "5"*. Assim sendo, entre o código:

```
TO TESTE_NAO :V
  IF (NOT :V > 5) [
    PR POWER :V 2
  ]
END
```

Em seguida execute separadamente as instruções a seguir:

```
TESTE_NAO 3
TESTE_NAO 5
TESTE_NAO 7
```

Ao ser executado o procedimento serão apresentados apenas resultados para entradas menores ou iguais a 5 (que são a forma de respeitar a condição *valores não maiores que 5*).

3.6 - Recursão

A ação de recursão é um recurso que na linguagem Logo permite a um procedimento chamar a si mesmo certo número de vezes. Veja que, um procedimento recursivo não pode chamar a si mesmo infinitamente, pois se assim o fizer entrará em um processo infinito de chamadas sucessivas podendo interromper a ação operacional do computador como um todo no momento em que os recursos de memória se tornarem esgotados. É importante que o procedimento recursivo tenha a definição de uma condição de encerramento (*aterramento*) controlada com o comando **IF** e com o auxílio do comando **STOP**.

A ação de aterramento deve ser definida antes da linha de código que efetua a recursão, pois ao contrário pode ocasionar o efeito colateral de execução infinita da recursividade inviabilizando seu uso (TOMIYAMA, 2016, p.2).

De modo prático um procedimento recursivo efetua ações de repetições semelhantemente as ações produzidas com o comando **REPEAT**. A diferença é que o comando **REPEAT** repete um trecho arbitrário de instruções e a recursão repete o próprio procedimento.

Para demonstrar um efeito de recursão considere um procedimento chamado **"ESPIRAL"** que produza a apresentação de imagens espiraladas. Vale ressaltar que uma espiral é o desenho de uma linha curva que se desenrola num plano de modo regular a partir de um ponto, dele gradualmente afastando-se. Assim sendo, entre o seguinte código:

```
TO ESPIRAL :TAMANHO :ANGULO :NUMERO
  IF :NUMERO = 0 [
    STOP
  ]
  FD :TAMANHO
  RT :ANGULO
  ESPIRAL (:TAMANHO + 8) :ANGULO (:NUMERO - 1)
END
```


Em seguida execute separadamente as instruções após fazer uso do comando **CS**:

```
CS ESPIRAL 4 122 60
```

```
CS ESPIRAL 4 92 50
```

Veja na figura 3.24 a apresentação do resultado das operações.

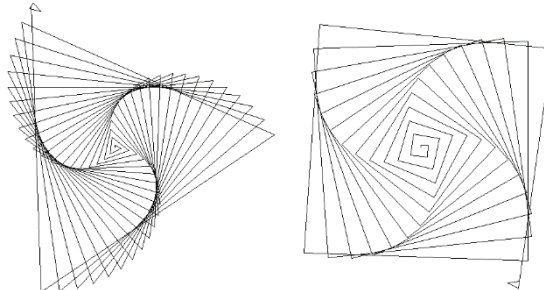


Figura 3.24 - Resultado do efeito recursivo sobre imagens espiraladas

A partir do código do procedimento **"ESPIRAL"** note a definição em cor verde da decisão de aterramento que detecta, neste caso, se o valor do parâmetro (variável) **:NUMERO** é igual a **"0"** (zero) e sendo faz a parada da execução do procedimento com o comando **STOP**. Note que se este trecho não estiver definido a função **ESPIRAL** chamará a si mesma sucessivamente.

Enquanto o valor da variável **:NUMERO** é diferente de zero ocorre a execução do procedimento que a cada chamada a si mesmo aumenta o valor de **:TAMANHO** em **"8"** (oito), mantém constante a distorção do valor de **:ANGULO** e diminui em **"1"** (um) o valor de **:NUMERO**. A cada chamada uma parte da imagem é desenhada com as instruções **FD :TAMANHO** e **RT :ANGULO** definidas na cor ocre. Veja também que a soma e subtração de valores sobre o valor inicial dos parâmetros tem que ser definida entre parênteses.

O efeito espiralado para a imagem de um triângulo é conseguido com o valor de giro de ângulo **"122"** para o parâmetro **:ANGULO** e a imagem do quadrado é conseguido com o valor **"92"**. Veja que esses valores são próximos ao valor real de giro para a apresentação das figuras geométricas planas.

O que aconteceria, por exemplo, se fossem usados os valores de ângulos **"120"** e **"90"** no procedimento **"ESPIRAL"**? Então, mãos à obra, execute separadamente as instruções após fazer uso do comando **CS**:

```
CS ESPIRAL 4 120 60
```

```
CS ESPIRAL 4 90 60
```

Veja os resultados na figura 3.25:

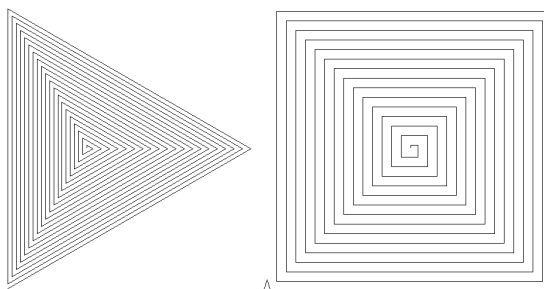


Figura 3.25 - Mais efeitos recursivos sobre imagens espiraladas

No sentido de apresentar mais ações sobre recursões veja na próxima sequência de procedimentos os recursos para se desenhar uma treliça (unidade definida a partir de cinco triângulos) como base de sustentação para uma ponte (adaptado, MULLER, 1998 p. 345). Assim sendo, entre o seguinte código:

```
TO TRIANGULAR
  FD 40 RT 135
  FD 40 / SQRT 2 RT 90
  FD 40 / SQRT 2 RT 135
END

TO TRELICA :X
  IF :X = 0 [STOP]
  REPEAT 4 [TRIANGULAR FD 40 RT 90]
  RT 90 FD 40 LT 90
  TRELICA :X - 1
END
```

Em seguida execute e a instrução:

```
CS TRELICA 10
```

Veja na figura 3.26 a apresentação do resultado do efeito de recursividade.

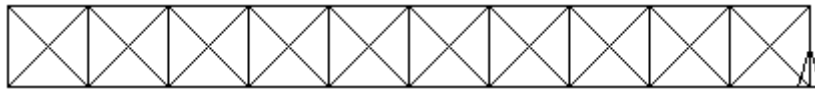


Figura 3.26 - Resultado do efeito de recursividade do procedimento "TRELICA"

Perceba que foram apresentados até este momento duas formas de se fazer a repetições de trechos de códigos. Uma com o comando **REPEAT** e a outra com o efeito de recursividade. Se a recursividade for implementada com o cuidado da definição da condição de aterramento será um mecanismo útil de repetição.

3.7 - Memorização e amnésia

Todo o desenvolvimento de procedimentos em prol da definição da estruturação da "inteligência artificial" da linguagem Logo para poder ter o efeito esperado precisa ser gravado na forma de arquivos. A gravação dos procedimentos criados permite estabelecer um recurso de "memorização", pois do contrário se nada for gravado quando o ambiente for executado posteriormente ao seu uso não se "lembrará" de nada entrando em estado de "amnésia".

Um arquivo Logo gravado pode ser formado por um único procedimento ou a partir de um conjunto de procedimentos vinculados por meio de sub-rotinas ou não.

Durante os tópicos anteriores foram desenvolvidos alguns procedimentos, os quais devem ser gravados. Para tanto, selecione a opção **"Save Logo Session"** do menu **"File"**. Será apresentada a caixa para gravação do arquivo de procedimento indicada na figura 3.27.

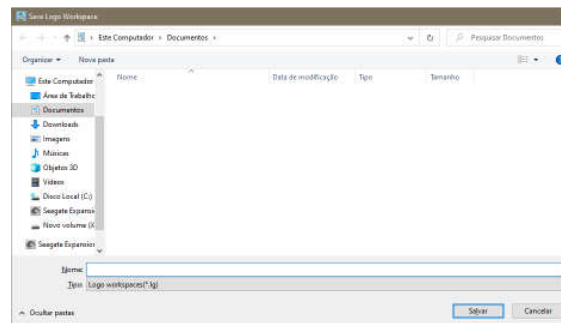


Figura 3.27 - Caixa para gravação de arquivos

No campo "**Nome**" informe um nome para a gravação do arquivo. Neste caso entre, por exemplo, o nome "**Cap03_Aprendizagem**" e acione o botão **Salvar**.

Para realizar um teste, encerre o ambiente "*UCBLogo*" e em seguida carregue-o novamente para a memória. Assim que o ambiente for carregado tente executar o procedimento "**FLORAL 5**" e veja a apresentação da mensagem de erro "**I don't know how to FLORAL**". Perceba que neste momento o ambiente Logo está em estado de "amnésia". Para fazer *Logo* "lembrar-se" do conhecimento que tem é necessário carregar para a memória o arquivo de procedimentos que se tenha anteriormente gravado. Para tanto, selecione a opção "**Load Logo Session**". Será apresentada a caixa para abertura do arquivo de procedimento que esteja gravado como mostra a figura 3.28.

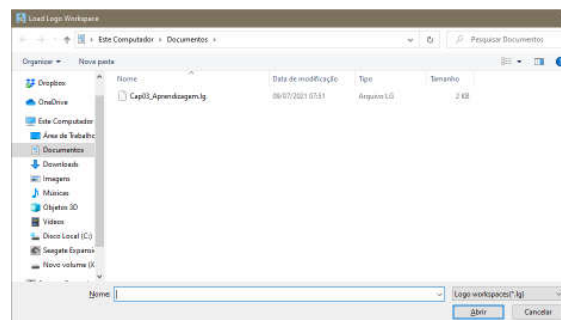


Figura 3.28 - Caixa para leitura de arquivos

Selecione, com o ponteiro do mouse, o arquivo desejado, neste caso "**Cap03_Aprendizagem**" e acione o botão "**Abrir**". A partir deste instante, execute a chamada da instrução "**FLORAL 5**" e veja o resultado apresentado e observe que sempre será necessário gravar procedimentos que se deseja tê-los para uso futuro.

Além do acesso a gravação e carregamento de conteúdo em memória por meio do menu superior do interpretador há a possibilidade de efetuar gravação e carregamento diretamente do *prompt* do ambiente. Para tanto, use respectivamente para gravar e carregar arquivos os comandos:

```
SAVE "<nome_do_arquivo>"
LOAD "<nome_do_arquivo>"
```

O arquivo a ser gravado ou lido não necessita possuir extensão de identificação. No entanto, fazer uso da extensão ".lg" ajuda a identificar qual é o conteúdo do arquivo. A gravação e leitura são executadas no diretório padrão do usuário.

Antes de passar ao próximo tópico efetue o encerramento do interpretador "*UCBLogo*" e abra-o novamente.

3.8 - Edição da área de trabalho

O ambiente interno do interpretador "UCBLogo" mantém enquanto ativo o registro temporário da definição dos dados armazenados em variáveis e também os procedimentos que controlam esses dados. Enquanto as variáveis podem ser indiscriminadamente redefinidas o mesmo não ocorre com os procedimentos. Os detalhes aqui indicados funcionam igualmente nos sistemas operacionais *Windows* e *Linux*, mas não no *Mac OS X* onde é necessário proceder com configuração adicional, aqui não apresentada.

Um procedimento quando definido na área de entrada textual não pode ser alterado diretamente no *prompt* do ambiente, ou seja, a definição de um procedimento chamado "X" não pode ser redefinido diretamente com o mesmo nome. Veja na figura 3.29 essa ocorrência com a apresentação de uma mensagem e a tentativa de redefinição para escrever outra mensagem no mesmo procedimento. Veja a apresentação da mensagem "X is already defined" indicando que o procedimento "X" já encontra-se definido e não pode ser redefinido.

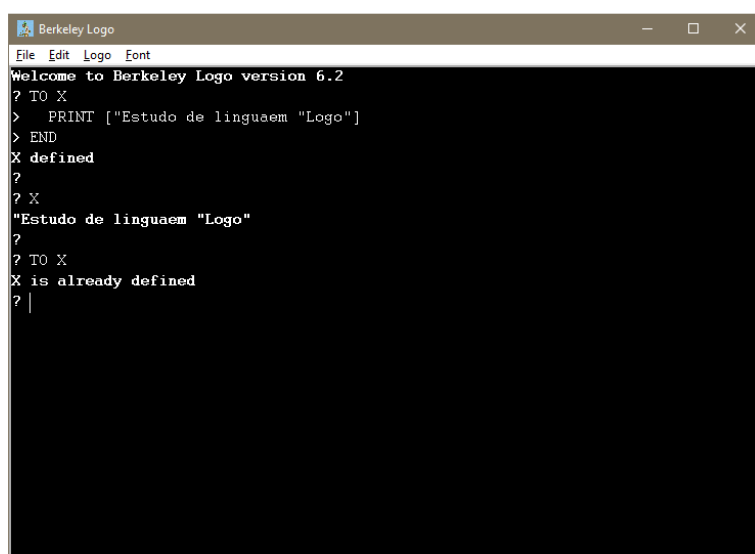


Figura 3.19 - Tentativa frustrada de redefinição de procedimento

A solução para esta questão é simples, basta fazer uso da primitiva de acesso ao programa de edição vinculado ao ambiente "UCBLogo" com a primitiva de edição de procedimento:

EDPS

Assim que a primitiva é operacionalizada apresenta-se a tela do programa editor com a apresentação do procedimento registrado em memória. De fato, estaria sendo apresentado, neste momento, todos os procedimentos existentes e ativos na memória. A figura 3.20 mostra a tela do modo de edição.

O ambiente de edição se mostra com uma interface básica mais útil em sua proposta. Observe a barra de menu com os comandos: *File*, *Edit* e *Font*.

O comando "File" possui as opções: "Close and Accept Changes" que encerra o modo editor e aceita todas as modificações realizadas; "Close and Revert Changes" que encerra o modo editor sem aceitar as modificações realizadas; "Page Setup" que abre o modo de configuração de página para impressão e "Print.." que permite imprimir o conteúdo do modo de edição.

O comando "Edit" possui as opções: "Cut" que corta um texto selecionado; "Copy" que copia um texto selecionado para a área de transferência; "Paste" que cola um texto existente na área

de transferência; **"Find..."** que faz a busca de algum detalhe nos procedimentos memorizados e **"Find Next"** que repete a ação definida em **"Find..."**. A seleção de texto ocorre tradicionalmente com o uso dos atalhos **"<Ctrl>+<A>"** para selecionar todo o texto ou a partir do uso da tecla **"<Shift>"** com as teclas de setas.

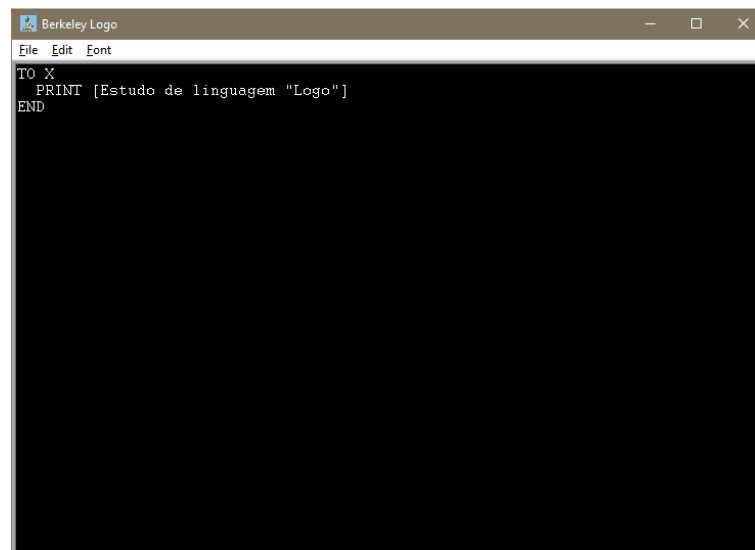


Figura 3.20 - Aparência do modo de edição "EDIT"

O comando **"Font"** possui as opções: **"Select Font..."** que permite alterar temporariamente o tipo de fonte de texto em uso; **"Increase Font Size"** que permite aumentar temporariamente o tamanho da fonte em uso e **"Decrease Font Size"** que permite diminuir temporariamente o tamanho da fonte em uso.

Neste momento pode-se proceder as alterações desejadas nos procedimentos existente e para registrados usar a opção de menu **"File/Close and Accept Changes"** ou então apenas fazer uso do atalho **"<Alt>+<A>"**.

Assim como procedimentos, variáveis também podem ter nomes e valores iniciais alterados. Para tanto, considerando a definição da variável:

```
MAKE "XPTO 10
```

Execute a primitiva de edição de nomes:

```
EDNS
```

Neste momento são apresentados os nomes que estiverem definidos em memória. Você poderá então fazer as mudanças desejadas e para registrar a aceitação das mudanças basta acionar o comando de menu **"File/Close and Accept Changes"**.

Caso deseje acesso total a memória para a edição de nomes de variáveis e procedimentos use a diretiva de mudança total:

```
EDALL
```

Caso queira codificar procedimentos diretamente no modo editor use a primitiva:

```
EDIT
```

Pode ocorrer quando do uso de **EDIT** a apresentação dos últimos procedimentos trabalhados. Isso ocorre pois o recurso grava as edições realizadas automaticamente. Caso deseje uma nova área de edição acione o atalho "<Ctrl>+<A>" e em seguida acione a tecla "" e proceda com as novas entradas.

Dentro das ações de gerenciamento da área de trabalho há certos comandos que são mais perigosos como as primitivas:

ERASE

ERALL

ERPS

ERN

ERNS

A primitiva **ERASE** apaga da área de trabalho um procedimento específico. Para seu funcionamento use a instrução "**ERASE** "<procedimento>", onde "**procedimento**" refere-se ao nome do procedimento a ser removido da memória. Há outros detalhes sobre **ERASE** que não vem, dentro do escopo deste trabalho, a ser apresentado.

A primitiva **ERALL** apaga da área de trabalho irrestritamente todos os procedimentos e variáveis existentes.

A primitiva **ERPS** apaga da área de trabalho todos os procedimentos existentes.

A primitiva **ERN** apaga da área de trabalho a variável indicada. Para seu funcionamento use a instrução "**ERN** "<variável>", onde "**variável**" refere-se ao nome da variável a ser removida da memória.

A primitiva **ERNS** apaga da área de trabalho todas as variáveis existentes.

Para remover especificamente um procedimento ou variável use a primitiva **EDIT** selecione na lista apresentada o que se deseja remover (desde que exista na memória) e utilize "".

3.9 - Exercícios de fixação

1. Quais são as figuras geométricas planas desenhadas a partir das seguintes instruções? Diga qual é a figura sem executar a instrução no ambiente de programação.

REPEAT 4 [FORWARD 100 RIGHT 90] _____

REPEAT 5 [FD 100 LT 72] _____

REPEAT 3 [PT 100 RT 120] _____

REPEAT 36 [FD 20 RT 10] _____

2. Criar procedimento chamado **RETANGULO1** (sem acento) que desenhe um retângulo com lados de tamanhos "**60**" e "**100**" para frente com giro de graus para à direita. O procedimento deve desenhar a imagem sem o uso do recurso **REPEAT**.

3. Criar procedimento chamado **RETANGULO2** (sem acento) que desenhe um retângulo com lados de tamanhos "**60**" e "**100**" para trás com giro de graus à esquerda. Usar a primitiva **REPEAT**.
4. Criar, sem uso da primitiva **REPEAT**, procedimento chamado **PENTAGONO** (sem acento) que construa um pentágono com tamanho "**40**". Avance para frente com giro a direita.
5. Criar procedimento chamado **DECAGONO** (sem acento) que construa uma figura com dez lados a partir do uso da primitiva **REPEAT** com giro de graus à esquerda com avanço para frente de "**30**" passos.
6. Criar procedimento chamado **ICOSAGONO** (sem acento) que desenhe a figura de mesmo nome a partir do uso da primitiva **REPEAT** com tamanho "**35**" movimentado para trás.

ANOTAÇÕES

[illegible]