

## CAPÍTULO 5 - Ações complementares

A linguagem Logo vai além da possibilidade de apenas fazer belas figuras. Logo é uma linguagem como outra qualquer e possibilita muitos recursos para o desenvolvimento de programação lógica, funcional e estruturada. Este capítulo distancia-se do mundo gráfico da tartaruga e mostra a linguagem em uma outra dimensão.

### 5.1 - Funções matemáticas

Já foram apresentadas algumas funções matemáticas da linguagem, como: **DIFFERENCE**, **INT**, **POWER**, **PRODUCT**, **QUOTIENT**, **SQRT**, **MODULO** e **SUM**. A linguagem possui também: **COS**, **EXP**, **LN**, **LOG10**, **MINUS**, **ROUND** e **SIN**, indicadas na tabela 5.1.

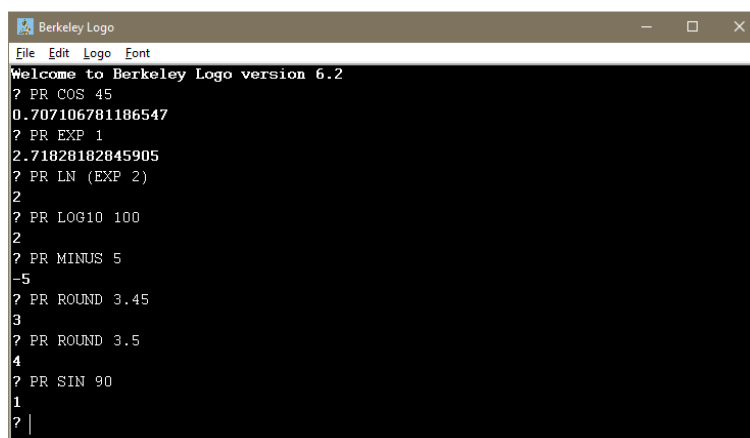
FUNÇÃO	OPERAÇÃO
COS	Calcula o cosseno de um ângulo medido em graus.
EXP	Calcula o exponencial de um número
LN	Calcula o logaritmo natural (base "e") de número.
LOG10	Calcula o logaritmo na base 10 de número.
MINUS	Retorna valor positivo como negativo.
ROUND	Arredonda o número para o inteiro mais próximo.
SIN	Calcula o seno do ângulo medido em graus.

Tabela 5.1 - Funções matemáticas e suas ações de ângulos

Veja a seguir alguns exemplos de apresentação de resultados gerados a partir do uso das funções matemáticas apresentadas.

```
PR COS 45
PR EXP 1
PR LN (EXP 2)
PR LOG10 100
PR MINUS 5
PR ROUND 3.45
PR ROUND 3.5
PR SIN 90
```

Veja os resultados gerados a partir das funções comentadas na figura 5.1.



```
Berkeley Logo
File Edit Logo Font
Welcome to Berkeley Logo version 6.2
? PR COS 45
0.707106781186547
? PR EXP 1
2.71828182845905
? PR LN (EXP 2)
2
? PR LOG10 100
2
? PR MINUS 5
-5
? PR ROUND 3.45
3
? PR ROUND 3.5
4
? PR SIN 90
1
? |
```

Figura 5.1 - Apresentação dos resultados no uso de funções matemáticas

É sabido que um computador efetua três ações essenciais: entrada de dados, processamento de dados e saída de dados na forma de informação. As operações de entrada na linguagem Logo podem ser realizadas com o uso de parâmetros junto aos procedimentos ou a partir das primitivas **READCHAR**, **READLIST** ou **READWORD**; já as operações de saída ocorrem nas áreas de ação dos modos gráfico e texto a partir das primitivas **PRINT**, **SHOW** ou **LABEL**. Em relação ao processamento este evento pode ocorrer em duas dimensões que se misturam: uma dimensão matemática e outra dimensão lógica.

As funções matemáticas e os operadores aritméticos são responsáveis por atenderem a necessidade de execução de cálculos aritméticos na resolução de problemas matemáticos. As funções matemáticas podem ser utilizadas no desenvolvimento de figuras, especialmente as funções trigonométricas. Veja um exemplo de imagem com o uso da função **SIN**:

```
REPEAT 360 [SETXY (SIN(2 * REPCOUNT)) * 150 (SIN(3 * REPCOUNT)) * 100]
```

A figura 5.2 mostra o resultado da execução da instrução com uso da função **SIN**.

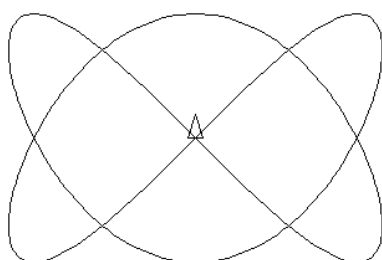


Figura 5.2 - Imagem obtida a partir do uso da função "SIN"

## 5.2 - Funções lógicas

Além das funções matemáticas a um conjunto de funções lógicas que auxiliam diversas ações de processamento. Funções do tipo lógicas são recursos que retornam como resposta de sua operação os valores **TRUE** ou **FALSE** sendo ótimas no estabelecimento de condições para as tomadas de decisão. Veja algumas funções lógicas encontradas na linguagem Logo: **BEFOREP**, **GREATERP**, **MEMBERP**, **LESSP**, **NUMBERP** e **EQUALP**, indicadas na tabela 5.2.

FUNÇÃO	OPERAÇÃO
<b>BEFOREP</b>	Retorna <b>TRUE</b> se parâmetro 1 vem à frente de parâmetro 2.
<b>GREATERP</b>	Retorna <b>TRUE</b> se parâmetro 1 é maior que parâmetro 2.
<b>MEMBERP</b>	Retorna <b>TRUE</b> se parâmetro 1 for membro de parâmetro 2.
<b>LESSP</b>	Retorna <b>TRUE</b> se parâmetro 1 é menor que parâmetro 2.
<b>NUMBERP</b>	Retorna <b>TRUE</b> se valor for um número.
<b>EQUALP</b>	Retorna <b>TRUE</b> se parâmetros forem iguais.

Tabela 5.2 - Funções lógicas (algumas)

Veja a seguir alguns exemplos de apresentação de resultados gerados a partir do uso das funções lógicas apresentadas.

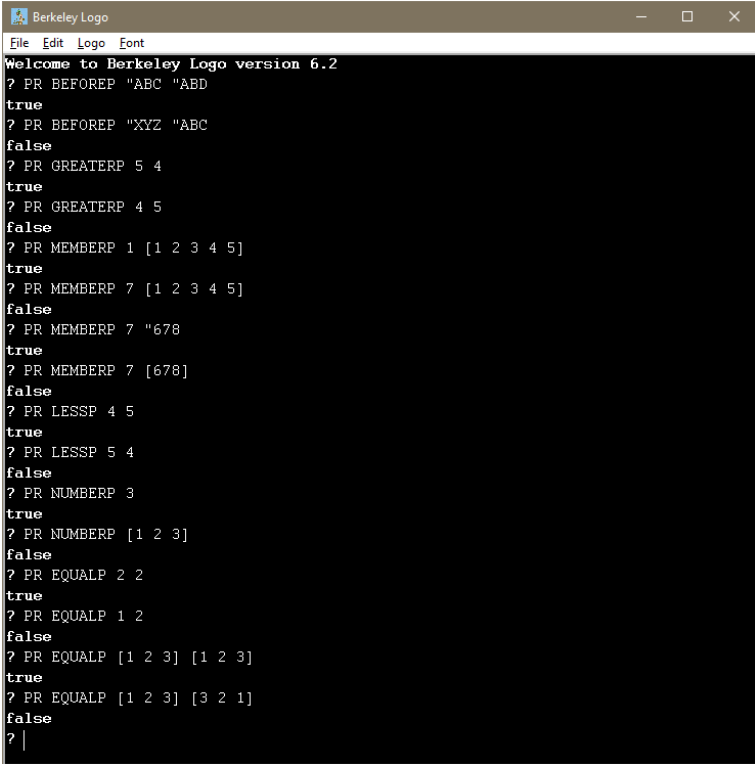
```
PR BEFOREP "ABC "ABD
PR BEFOREP "XYZ "ABC
PR GREATERP 5 4
PR GREATERP 4 5
PR MEMBERP 1 [1 2 3 4 5]
PR MEMBERP 7 [1 2 3 4 5]
```

```

PR MEMBERP 7 "678
PR MEMBERP 7 [678]
PR LESSP 4 5
PR LESSP 5 4
PR NUMBERP 3
PR NUMBERP [1 2 3]
PR EQUALP 2 2
PR EQUALP 1 2
PR EQUALP [1 2 3] [1 2 3]
PR EQUALP [1 2 3] [3 2 1]

```

Veja os resultados gerados a partir das funções comentadas na figura 5.3.



```

Berkeley Logo
Welcome to Berkeley Logo version 6.2
? PR BEFOREP "ABC "ABD
true
? PR BEFOREP "XYZ "ABC
false
? PR GREATERP 5 4
true
? PR GREATERP 4 5
false
? PR MEMBERP 1 [1 2 3 4 5]
true
? PR MEMBERP 7 [1 2 3 4 5]
false
? PR MEMBERP 7 "678
true
? PR MEMBERP 7 [678]
false
? PR LESSP 4 5
true
? PR LESSP 5 4
false
? PR NUMBERP 3
true
? PR NUMBERP [1 2 3]
false
? PR EQUALP 2 2
true
? PR EQUALP 1 2
false
? PR EQUALP [1 2 3] [1 2 3]
true
? PR EQUALP [1 2 3] [3 2 1]
false
? |

```

Figura 5.3 - Apresentação dos resultados no uso de funções matemáticas

As funções lógicas são úteis no estabelecimento de controles em tomadas de decisão e execução de laços condicionais exatamente por retornarem como resposta a sua ação um valor **TRUE** ou **FALSE**. Essas funções podem ser usadas para auxiliar diversas ações da linguagem seja no estabelecimento de operações matemáticas ou na elaboração de figuras.

As operações de processamento lógico são efetivadas a partir do uso dos operadores relacionais, operadores lógicos e funções especializadas.

### 5.3 - Algumas funções complementares

O conjunto de funções na linguagem Logo é extenso e torna-se inadequado exemplificar todas elas em um livro, uma vez que o ambiente "*UCBLogo*" possui boa documentação em português de todos os recursos disponíveis na linguagem. Veja algumas funções complementares que não foram anteriormente qualificadas, sendo: **NAMEP** (função lógica) **WORD**, **FIRST**, **KEYP** (função lógica), **LAST** e **THING**. Observe a tabela 5.3.

FUNÇÃO	OPERAÇÃO
NAMEP	Retorna TRUE se conteúdo for o nome de uma variável.
WORD	Concatena palavras formando outra palavra.
FIRST	Retorna o primeiro elemento de uma série de valores.
KEYP	Retorna TRUE se certa tecla esperada é acionada.
LAST	Retorna o último elemento de uma série de valores.
THING	Retorna o conteúdo que esteja associado a uma variável.

Tabela 5.3 - Funções complementares (algumas)

As funções **WORD**, **FIRST** e **LAST** podem ser demonstradas em uma linha de instrução como foram demonstradas as funções matemáticas e lógicas. Veja a seguir alguns exemplos e atente ao uso das chaves com a função **FIRST** e colchetes com a função **LAST** quando usadas na manipulação de conjuntos de elementos.

```
PR WORD "Lo "go
PR WORD "UCB "Logo
PR FIRST [1 2 3 4 5]
PR FIRST "Logo
PR FIRST [[L][o][g][o]]
PR FIRST {A B C D E}
PR LAST [1 2 3 4 5]
PR LAST "Computador
PR LAST [[L][o][g][o]]
PR LAST [A B C D E]
```

Veja os resultados gerados a partir das funções comentadas na figura 5.4.

```

Berkeley Logo
File Edit Logo Font
Welcome to Berkeley Logo version 6.2
? PR WORD "Lo "go
go
? PR WORD "UCB "Logo
UCBLogo
? PR FIRST [1 2 3 4 5]
1
? PR FIRST "Logo
L
? PR FIRST [[L][o][g][o]]
L
? PR FIRST {A B C D E}
1
? PR LAST [1 2 3 4 5]
5
? PR LAST "Computador
r
? PR LAST [[L][o][g][o]]
o
? PR LAST [A B C D E]
E
? |

```

Figura 5.4 - Apresentação dos resultados no uso de funções auxiliares

As funções **NAMEP**, **KEYP** e **THING** para serem demonstradas necessitam de mais passos que as demais funções. Por esta razão estão sendo demonstradas em separado.

A função **THING** opera sua ação sobre variáveis. Desta forma, é necessário que haja na memória definição de variáveis que possam ser utilizadas pela função. Assim sendo, considere o seguinte trecho de instruções (ATARI, 1983, p. 76) que definem as variáveis de memória:

```
MAKE "VENCEDOR "COMPUTADOR
MAKE "COMPUTADOR [100 PONTOS]
```

Observe um detalhe que pode passar despercebido. Veja que na primeira instrução são definidas duas variáveis: **VENCEDOR** e **COMPUTADOR**. Veja que a variável **COMPUTADOR** ao ser definida após a variável **VENCEDOR** tornou-se conteúdo da variável **VENCEDOR**.

A partir dessa ocorrência, perceba, que o conteúdo da variável **COMPUTADOR** está vinculado a variável **VENCEDOR**. O acesso ao conteúdo vinculado entre as variáveis é realizado com o uso da função **THING**. Veja as próximas instruções:

```
PR :VENCEDOR
PR THING :VENCEDOR
PR THING "VENCEDOR
```

Veja os resultados gerados com o uso da função **THING** indicados na figura 5.5. Atente a diferença de uso dos símbolos [:] dois pontos e ["] aspa inglesa no acesso ao conteúdo da variável indicada.



Figura 5.5 - Apresentação de resultados com função "THING"

Observe na sequência o procedimento **"INC"** que acrescenta o valor **"+ 1"** a uma variável indicada e faz uso da função **NAMEP**. Desta forma, entre o seguinte código:

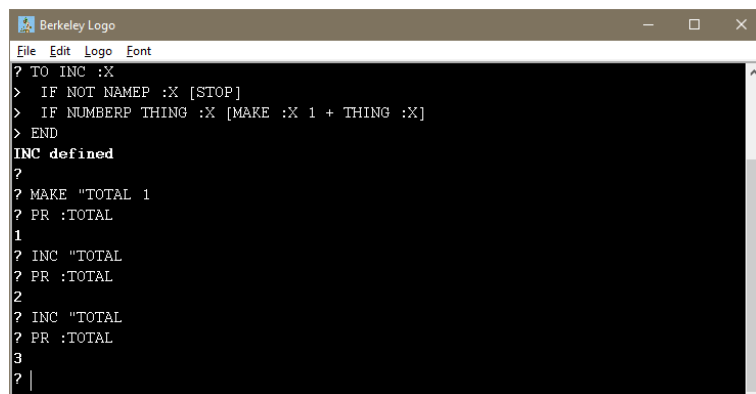
```
TO INC :X
  IF NOT NAMEP :X [STOP]
  IF NUMBERP THING :X [MAKE :X 1 + THING :X]
END
```

Execute na sequência as instruções a seguir e veja o resultado indicado na figura 5.6.

```
MAKE "TOTAL 1
PR :TOTAL
INC "TOTAL
PR :TOTAL
INC "TOTAL
PR :TOTAL
```

Observe a evolução dos valores de **"1"** a **"3"** nos passos indicados. Note que o procedimento **"INC"** recebe um parâmetro (seu conteúdo) representado pela variável **X**.

Veja que a primeira instrução do procedimento **"INC"** é verificar se o conteúdo passado para o argumento **X** é de fato uma variável. Isso é detectado pela função **NAMEP** com auxílio do operador lógico de negação **NOT**, ou seja, não sendo o nome fornecido uma variável **STOP**. A função **NAMEP** retorna o valor **TRUE** se a condição for satisfeita ou **FALSE** caso contrário.



```

Berkeley Logo
File Edit Logo Font
? TO INC :X
> IF NOT NAMEP :X [STOP]
> IF NUMBERP THING :X [MAKE :X 1 + THING :X]
> END
INC defined
?
? MAKE "TOTAL 1
? PR :TOTAL
1
? INC "TOTAL
? PR :TOTAL
2
? INC "TOTAL
? PR :TOTAL
3
? |

```

Figura 5.6 - Apresentação de resultados com função "THING" em procedimento

A segunda instrução do procedimento "INC" valida se o **THING** da variável **X** informado é de fato um número (**NUMBERP**), sendo efetue a atribuição (**MAKE**) de "+ 1" sobre o conteúdo do **THING** armazenado na memória a partir da variável informada em **X**.

Uma função lógica interessante é **KEYP** que pode ser usada na identificação do acionamento de teclas do teclado.

Observe a seguir o procedimento "DIRECAO" adaptado de ATARI (1983, p. 128) que risca a mão livre um desenho na área de ação do modo gráfico. Para tanto, entre o seguinte código:

```

TO DIRECAO
  IF KEYP [REALIZE RC]
  DIRECAO
END

TO REALIZE :DIR
  IF UPCASE :DIR = "C [CS] ; Limpa área de trabalho.
  IF UPCASE :DIR = "D [RT 10] ; Direção para direita.
  IF UPCASE :DIR = "E [LT 10] ; Direção para esquerda.
  IF UPCASE :DIR = "L [PD] ; Ativa o lápis.
  IF UPCASE :DIR = "N [PU] ; Desativa o lápis.
  IF UPCASE :DIR = "F [FD 2] ; Anda para frente.
  IF UPCASE :DIR = "T [BK 2] ; Anda para trás.
END

```

Execute na sequência o comando **DIRECAO** e na caixa de diálogo apresentada informe os comandos "C, D, E, L, N, F e T" para gerenciar o desenvolvimento de um desenho a mão livre. Veja um exemplo de resultado na figura 5.7.

Como não foi previsto nenhum comando para o encerramento do procedimento, sua operação fica "presa" na memória. Para encerrar a execução: no *Windows* acione o atalho "<Ctrl>+<Q>"; no *UNIX (Linux)* acione o atalho "<Ctrl>+<C>" e no *Mac OS* acione o atalho "<Command>+<. >". Após este acionamento é apresentada a caixa de diálogo "Quit Logo" perguntando se deseja salvar o trabalho na área de trabalho após o encerramento, como mostra a figura 5.8. Neste momento, acione o botão "Sim" e abrir-se-á a caixa de diálogo para salvar. Caso contrário, pode acionar "Não" ou "Cancelar". O botão "Não" efetiva a saída sem gravar e o botão "Cancelar" mantém o procedimento em execução.

Observe no procedimento "REALIZE" além da definição das teclas de acionamento e das ações a elas associadas o uso do símbolo ponto e vírgula com a descrição da ação de cada instrução

do procedimento. O símbolo dois pontos é usado dentro do código quando há o desejo de declarar linhas de comentários dentro do código com o objetivo de definir uma linha de documentação interna do próprio código.

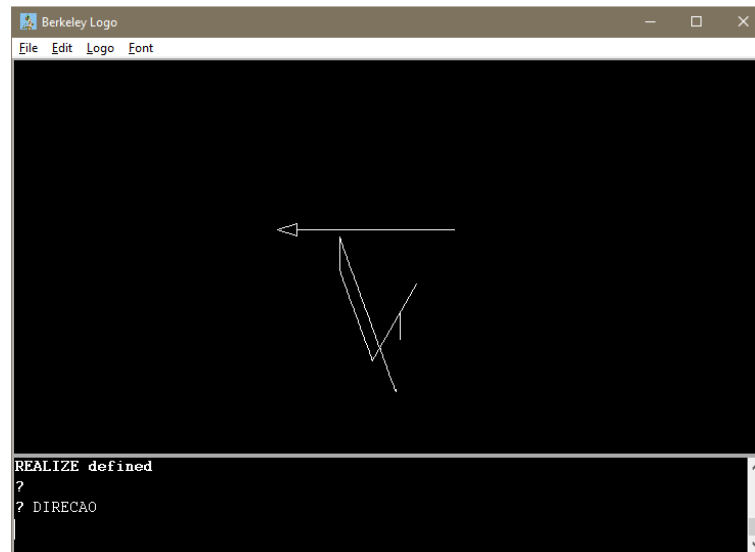


Figura 5.7 - Apresentação de desenho a mão livre

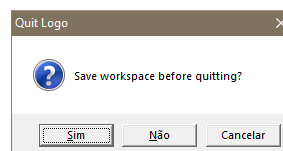


Figura 5.8 - Apresentação da caixa de diálogo para encerramento de execução

Caso deseje apenas pausar a execução de um procedimento: no *Windows* acione o atalho "<Ctrl>+<W>"; no *UNIX (Linux)* acione o atalho "<Ctrl>+<\>" e no *Mac OS* acione o atalho "<Command>+<, >".

#### 5.4 - Programação sem figuras

Logo é uma linguagem que possui muito mais do que aparentemente mostra. É possível desenvolver várias categorias de aplicação com Logo, mesmo não sendo usada no universo comercial. As aplicações Logo são usadas mais intensamente no universo educacional e podem ser amplamente usadas industrialmente junto a controle de robôs em produção crítica que colocam a vida humana em risco.

A primeira versão da linguagem foi escrita em computadores que não são usados domesticamente, ou seja, computadores hoje chamados de grande porte. Sua interface de acesso para crianças pré-alfabetizadas era intermediada por um console de Terminal (equipamento com monitor de vídeo e teclado) e para crianças não alfabetizadas a partir de um console com botões conectados ao computador ligado por um cabo (cordão umbilical) a um robô mecânico, chamada tartaruga que percorria sobre uma grande folha de papel traçando imagens geométricas planas (figura 5.9).

Sem sombra de dúvidas pensar em Logo nas décadas de 1960 e 1970 era muito divertido, pois era possível ver um robô sobre uma folha de papel produzir desenhos incríveis (figura 5.10).

Depois veio a fase do controle remoto de periféricos onde o robô já não possuía mais um cordão umbilical e sim controlado por ondas de rádio, mais emocionante ainda.

Possivelmente por questões de custos o controle da tartaruga diretamente na tela de um computador se tornou bastante popular afastando fisicamente a linguagem da tartaruga.

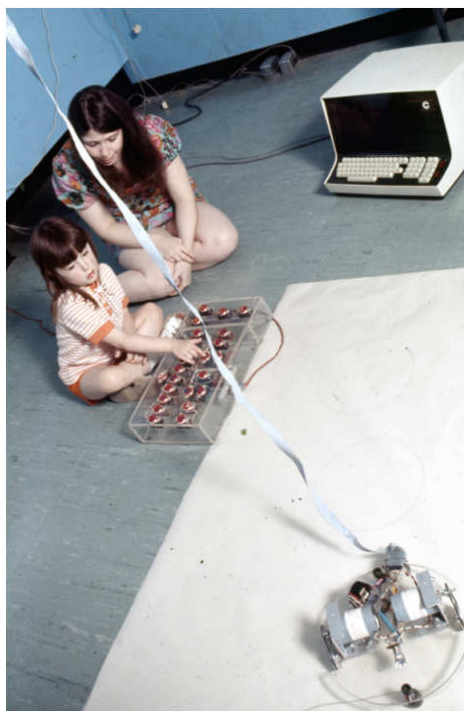


Figura 5.9 - Criança desenhando com Logo (SOLOMON, 2020, p. 39)



Figura 5.10 - Tartaruga remota (CATLIN, 2016)

A linguagem Logo vem sendo, há muito tempo, erroneamente vinculada a aprendizagem apenas de crianças. Isso pode até ter sido verdade, como as figuras anteriores mostram, há muito tempo, em meados de 1960, por ter sido inicialmente apresentada a esse grupo de pessoas. No entanto, a partir de 1975 com a introdução dos microcomputadores sua popularidade decolou e Logo se tornou bastante popular atraindo a atenção de outros públicos. Por controlar as ações de um robô, Logo se mostrou uma excelente linguagem para uso em ações de automação industrial e mecatrônica, indo além do que simplesmente fazer desenhos. Em certos momentos deste livro você deve ter notado exemplos de procedimentos que não elaboraram desenhos na área de ação do modo gráfico, como o procedimento "**TABUADA**" do capítulo "4" que mostra o resultado de sua operação na área de ação do modo texto. Esse é um tipo de ação que muitas vezes passa despercebido com o uso da linguagem Logo, pois é comum ver seu foco voltado a aplicações geométricas, até mesmo na quantidade de material produzido para apresentar a linguagem. Para exemplificar a linguagem fora do eixo geométrico considere a seguir alguns exemplos aplicativos simples que efetuam algumas ações computacionais comuns a qualquer linguagem de programação.

Considere para o primeiro exemplo um procedimento chamado "**AREACIRC**" que receba a entrada de um valor real para a medida do raio de uma circunferência e apresente o resultado da área desta circunferência sem desenhá-la com três casa decimais. Para tanto, entre o seguinte código:



TO AREACIRC

PR [Informe a medida do raio de uma circunferencia:]

MAKE "RAIO READWORD

MAKE "AREA 3.1415926 \* (POWER :RAIO 2)

PR (SENTENCE "Area "= FORM :AREA 0 2)

END

Execute o procedimento "AREACIRC" e informe para um primeiro teste o valor "5" e veja a apresentação do resultado "78.54" como mostra a figura 5.11. Observe cada detalhe do procedimento "AREACIRC". Note que são usados diversos elementos integrados na ideia de um todo com cada componente realizando uma ação que complementa outro componente.

```

Berkeley Logo
File Edit Logo Font
? TO AREACIRC
> PR [Informe a medida do raio de uma circunferencia:]
> MAKE "RAIO READWORD
> MAKE "AREA 3.1415926 * (POWER :RAIO 2)
> PR (SENTENCE "Area "= FORM :AREA 0 2)
> END
AREACIRC defined
?
? AREACIRC
Informe a medida do raio de uma circunferencia:
5
Area = 78.54
? |

```

Figura 5.11 - Entrada e saída do procedimento "AREACIRC"

O segundo exemplo é fundamentado a partir de um procedimento chamado "DECISAO" que recebe a entrada separada de dois valores numéricos representados pelas variáveis "A" e "B", soma-os armazenando seu resultado na variável "X" e mostra o resultado da soma na variável "X" caso este seja maior que "10". Para somas obtidas menores que "10" não deve ser realizada nenhuma operação. Desta forma, entre o seguinte código:

TO DECISAO

PR [Entre um valor numerico para a variavel <A>:]

MAKE "A READWORD

PR [Entre um valor numerico para a variavel <B>:]

MAKE "B READWORD

MAKE "X :A + :B

IF :X > 10 [

PR (SE "Resultado "= :X)

]

END

Execute o procedimento "DECISAO" e informe separadamente os valores "5" e "6" e veja a apresentação do resultado "11", depois execute novamente o procedimento e informe os valores "5" e "5" e observe que nada é apresentado. As figuras 5.12 e 5.13 mostram os resultados desses testes.

```

Berkeley Logo
File Edit Logo Font
? DECISAO
Entre um valor numerico para a variavel <A>:
5
Entre um valor numerico para a variavel <B>:
6
Resultado = 11
? |

```

Figura 5.12 - Testes de execução do procedimento "DECISAO"

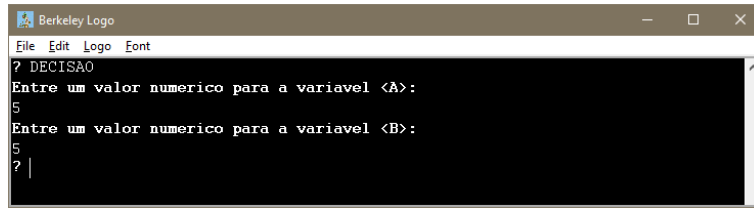


Figura 5.13 - Testes de execução do procedimento "DECISAO"

No terceiro exemplo considere um procedimento chamado "**CALCULAQUAD**" que apresente apenas o resultado do quadrado de um valor numérico inteiro e positivo (neste caso, maior ou igual a zero) lido pelo teclado sem armazená-lo em memória. Neste código considere validar a entrada recusando valores que não sejam numéricos e que não sejam positivos, além de considerar apenas a parte inteira do valor caso um valor real seja fornecido. Assim sendo, entre o seguinte código:

```
TO CALCULAQUAD
  DO.UNTIL [
    PR [Entre um valor numérico:]
    MAKE "VLR INT READWORD
  ][:VLR >= 0]
  PR (SE "Resultado "= POWER :VLR 2)
END
```

Execute o procedimento e teste as entradas com valores positivos, negativos e reais. Note o controle de negação da entrada de valores negativos com o uso da primitiva **DO.UNTIL**.

Como quinto e último exemplo de programação sem figuras considere um procedimento chamado "**FATORIAL**" que apresente o resultado da fatorial de um número. Este procedimento deve apresentar ao final para seu usuário a opção de fazer ou não novos cálculos, mas esta opção somente pode aceitar as respostas **SIM** ou **NÃO** em formato maiúsculo (qualquer outra forma de entrada deve ser recusada). Para tanto, entre o seguinte código:

```
TO FATORIAL
  DO.UNTIL [
    MAKE "FAT 1
    DO.UNTIL [
      PR [Entre um valor numerico:]
      MAKE "N INT READWORD
    ][:N >= 0]
    FOR [I 1 :N] [
      MAKE "FAT :FAT * :I
    ]
    PR (SE WORD :N "! "= :FAT)
    DO.UNTIL [
      PR [Deseja continuar? Responda: SIM ou NAO]
      MAKE "RESP READWORD
    ][OR UPCASE :RESP = "SIM UPCASE :RESP = "NAO]
  ][UPCASE :RESP <> "SIM]
END
```

Execute o procedimento e faça seus testes.

## 5.5 - Manipulação básica de dados

Em alguns outros momentos deste estudo foram apresentados certos detalhes sobre o uso e apresentação de textos e o tratamento de dados. Cabe junto a este tópico rever alguns detalhes e apresentar outros que complementem este conhecimento. Observe a tabela 5.5, adaptado de Downey & Gay (2003, p. 31-32, 55, 57-58, 60 e 63).

FUNÇÃO	OPERAÇÃO
ASCII	Retorna código ASCII do caractere informado.
CHAR	Retorna o caractere correspondente ao código ASCII.
COUNT	Conta caracteres de uma palavra ou palavras de uma frase.
BEFOREP	Retorna TRUE se primeiro conteúdo vem a frente do segundo.
SUBSTRINGP	Retorna TRUE se primeiro conteúdo faz parte do segundo.
REVERSE	Inverte uma sequência de caracteres.
ITEM	Mostra um elemento de uma coleção a partir da posição.
LOWERCASE	Mostra caracteres em formato minúsculo.
BUTFIRST	Mostra todos os elementos, exceto o primeiro.
BUTFIRSTS	Mostra os elementos de listas, exceto os primeiros.
BUTLAST	Mostra todos os elementos, exceto o último.
ISEQ	Gera lista de números inteiros customizada de 1 em 1.
RSEQ	Gera lista de números racionais quantificados.

Tabela 5.5 - Funções complementares (mais algumas)

Veja alguns detalhes indicados por instruções em azul e seus efeitos como resultados gerados em vermelho.

PR FIRST "ABCDE

A

PR FIRST 54321

5

PR LAST "ABCDE

E

PR LAST 54321

1

PR ASCII "A

65

PR CHAR 65

A

PR BEFOREP "ABC "ABD

TRUE

```
PR BEFOREP "ABD "ABC  
FALSE
```

```
PR BUTFIRST 1.2345 ; "BUTFIRST" pode ser escrito como "BF".  
.2345
```

```
PR BF 54321  
4321
```

```
PR BUTFIRSTS [[5 4 3 2 1] [A B C B D E]] ; "BUTFIRSTS" = "BFS".  
[4 3 2 1] [B C B D E]
```

```
PR SUBSTRINGP "AB "ABC  
TRUE
```

```
PR SUBSTRINGP "AC "ABC  
FALSE
```

```
PR REVERSE "ABCDE  
EDCBA
```

```
PR LOWERCASE "ABCDE  
abcde
```

```
PR BUTLAST "ABCDE ; "BUTLAST" = "BL".  
ABCD
```

```
PR BL 54321  
5432
```

```
PR ITEM 2 "ABCDE  
B
```

```
PR ITEM 4 54321  
2
```

```
PR WORD "Linguagem "Logo  
LinguagemLogo
```

```
PR SENTENCE "Linguagem "Logo  
Linguagem Logo
```

```
(PR "Linguagem "Logo)  
Linguagem Logo
```

```
PR (SENTENCE "Estudo "de "Linguagem "Logo)  
Estudo de Linguagem Logo
```

```
PR (SENTENCE 1 2 3 4 5)  
1 2 3 4 5
```

```
PR (PRODUCT 2 3 4)
```

```
24
```

```
PR (SUM 1 2 3 4 5)
```

```
15
```

```
PR ISEQ 2 9
```

```
2 3 4 5 6 7 8 9
```

```
PR ISEQ 9 2
```

```
9 8 7 6 5 4 3 2
```

```
PR RSEQ 2 9 8
```

```
2 3 4 5 6 7 8 9
```

```
PR RSEQ 9 2 8
```

```
9 8 7 6 5 4 3 2
```

```
PR COUNT [Estudo de Linguagem Logo]
```

```
4
```

```
PR COUNT "ABCDE
```

```
5
```

```
PR COUNT (SENTENCE "Estudo "de "Linguagem "Logo)
```

```
4
```

```
PR COUNT (WORD "Estudo "de "Linguagem "Logo)
```

```
21
```

```
PR COUNT WORD FIRST "ABCD BUTFIRST 1234567890
```

```
10
```

Um detalhe que deve ter sido percebido, ao longo do que foi até aqui apresentado, é a capacidade da linguagem operar com tipos de dados diferenciados, como números e textos, além do processamento aritmético e lógico. Um importante detalhe relacionados a manipulação de dados é o agrupamento de elementos em conjuntos contextualizados na forma de listas (elementos entre parênteses) e vetores (elementos entre chaves). Os dados de um tipo de agrupamento podem ser convertidos aos dados de outro tipo de agrupamento com o uso específicos das funções **ARRAYTOLIST** e **LISTTOARRAY**. Observe exemplos de conversão.

```
SHOW ARRAYTOLIST {1 2 3 4 5}
```

```
[1 2 3 4 5]
```

```
PR LISTTOARRAY [1 2 3 4 5]
```

```
{1 2 3 4 5}
```

No ambiente "UCBLogo" os dados de uma lista são apresentados sem a indicação dos parênteses quando em uso a primitiva **PRINT (PR)**.

Veja a seguir o procedimento "**TEXTOTRIANG1**" que a partir de uma palavra informada como parâmetro a apresenta de forma triangular diminuindo-se a cada apresentação um caractere

removido do lado esquerdo, adaptado de Muller (1998, p. 502). Para tanto, entre o seguinte código:

```
TO TEXTOTRIANG1 :CONTEUDO
  IF :CONTEUDO = " [STOP] ; Quando CONTEUDO for vazio pare.
  PR :CONTEUDO
  TEXTOTRIANG1 BUTFIRST :CONTEUDO
END
```

Na sequência execute a instrução **TEXTOTRIANG1 "COMPUTADOR** e veja o resultado apresentado na área de ação do modo texto.

```
COMPUTADOR
COMPUTADO
COMPUTAD
COMPUTA
COMPUT
COMPU
COMP
COM
CO
C
```

Considere outro procedimento chamado "**TEXTOTRIANG1**" que apresente a palavra informada como parâmetro do último caractere para o primeiro, adaptado de Muller (1998, p. 504). Assim sendo, entre o seguinte código:

```
TO TEXTOTRIANG2 :CONTEUDO
  IF :CONTEUDO = " [STOP] ; Quando CONTEUDO for vazio STOP.
  TEXTOTRIANG2 BUTFIRST :CONTEUDO
  PR :CONTEUDO
END
```

Na sequência execute a instrução **TEXTOTRIANG2 "COMPUTADOR** e veja o resultado apresentado na área de ação do modo texto.

```
C
CO
COM
COMP
COMPU
COMPUT
COMPUTA
COMPUTAD
COMPUTADO
COMPUTADOR
```

Veja que o efeito de apresentação entre os dois últimos procedimentos ocorre a partir da posição da instrução "**PR :CONTEUDO**" antes e após a instrução "**TEXTOTRIANG2 BUTFIRST :CONTEUDO**".

## 5.6 - Escopo e visibilidade de variáveis

O espaço de memória pode ser controlado com a definição do escopo de comportamento das variáveis, que podem ser globais e locais, dependendo apenas da primitiva de definição de variável em uso. As variáveis globais são definidas com a primitiva **MAKE (MAKE)** amplamente usada neste livro e as variáveis locais são definidas com a primitiva **LOCAL**. Quando uma variável global é definida dentro de um procedimento ela se torna visível fora do procedimento e para todos os subprocedimentos relacionados o que é diferente para uma variável definida como local, pois neste caso, esta variável é visível dentro do procedimento em que foi definida e também aos subprocedimento relacionados, mas não será visível fora do procedimento.

Observe o procedimento chamado "**VARGLOBA1**" com a definição de uma variável global e o acesso a esta variável de forma interna e externa ao procedimento. Para tanto, entre o seguinte código:

```
TO VARGLOBA1
  MAKE "CONTADORG 1
  PR :CONTADORG
  MAKE "CONTADORG :CONTADORG + 1
  PR :CONTADORG
END
```

Ao término, feche o programa **Editor** com "**Arquivo/Guardar e Sair**" ou acione as teclas "<Ctrl> + <D>". Em seguida execute a instrução de chamada do procedimento:

```
VARGLOBA1
```

Observe os resultados apresentados e na sequência na linha de comando do campo de entrada de comandos, dados e instruções informe a seguinte instrução:

```
PR SUM :CONTADORG 1
```

Veja na figura 5.14 a apresentação dos resultados de uso de variável global. Note que a instrução "**PR SOMA :CONTADORG 1**" soma mais "**1**" ao valor da variável "**CONTADORG**" tornando seu resultado "**3**" por ser "**CONTADORG**" uma variável global.

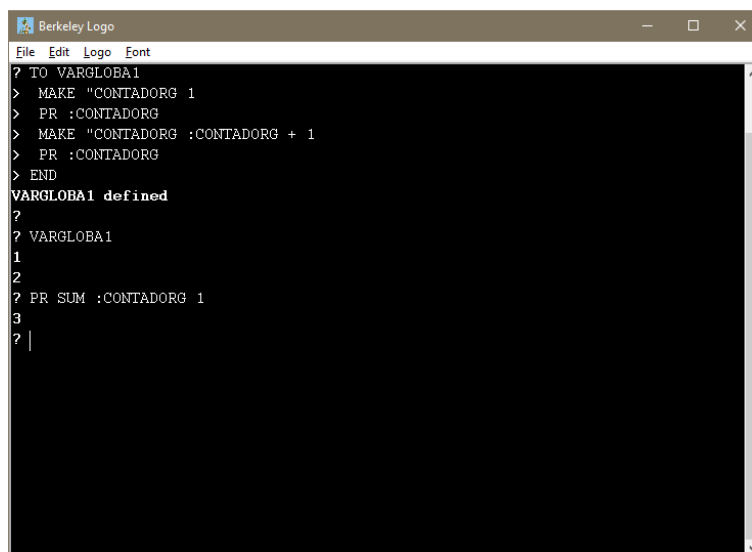


Figura 5.14 - Resultado no uso de variável global

Observe o procedimento chamado "**VARLOCAL1**" com a definição de uma variável local com acesso apenas interno ao procedimento. Desta forma, entre o seguinte código:

```
TO VARLOCAL1
  LOCAL "CONTADORL
  MAKE "CONTADORL 1
  PR :CONTADORL
  MAKE "CONTADORL :CONTADORL + 1
  PR :CONTADORL
END
```

Em seguida execute a instrução de chamada do procedimento:

```
VARLOCAL1
```

Observe os resultados apresentados e na sequência na linha de comando do campo de entrada de comandos, dados e instruções informe a seguinte instrução:

```
PR SUM :CONTADORL 1
```

Veja na figura 5.15 a apresentação dos resultados de uso de variável global. Note que a instrução "**PR SOMA :CONTADORL 1**" tenta somar mais "1" ao valor da variável "**CONTADORL**" que retorna como resposta a informação "**CONTADORL has no value**" por ser "**CONTADORL**" uma variável local.

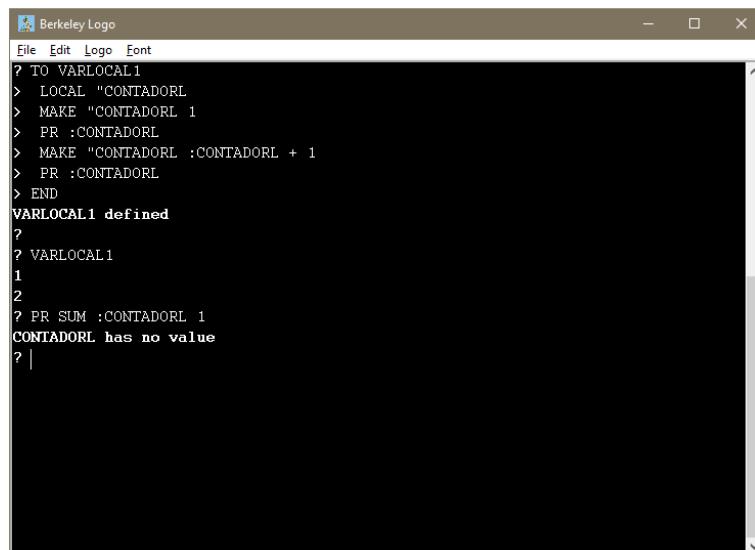


Figura 5.15 - Resultado no uso de variável local

Observe que a primitiva **LOCAL** permite que seja definida uma variável com estado de visibilidade local, mas após esta definição se faz uso da primitiva **MAKE (MAKE)** normalmente. Então, atente ao fato de que se uma variável definida com **MAKE (MAKE)** sem o uso prévio de "**LOCAL**" é global e com o uso prévio de "**LOCAL**" é local.

Veja em seguida o comportamento de variáveis globais e locais a partir do uso de subprocedimento (sub-rotinas).

Observe o procedimento chamado "**VARGLOBA2**" com a definição global de variável e uso de sub-rotina para a ação de contagem. Para tanto, entre o seguinte código:



```

TO VARGLOBA2
  MAKE "CONTADORGX 1
  PR :CONTADORGX
  SUBVARGLOBA2
END

TO SUBVARGLOBA2
  MAKE "CONTADORGX :CONTADORGX + 1
  PR :CONTADORGX
END

```

Em seguida execute a instrução de chamada do procedimento:

VARGLOBA2

Observe os resultados apresentados e na sequência na linha de comando do campo de entrada de comandos, dados e instruções informe a seguinte instrução:

PR SUM :CONTADORGX 1

Veja a apresentação dos resultados "1", "2" e "3" a partir da execução do procedimento principal com chamada de subprocedimento (sub-rotina) utilizando-se variável global, indicado na figura 5.16.

```

Berkeley Logo
File Edit Logo Font
? TO VARGLOBA2
> MAKE "CONTADORGX 1
> PR :CONTADORGX
> SUBVARGLOBA2
> END
VARGLOBA2 defined
?
? TO SUBVARGLOBA2
> MAKE "CONTADORGX :CONTADORGX + 1
> PR :CONTADORGX
> END
SUBVARGLOBA2 defined
?
? VARGLOBA2
1
2
? PR SUM :CONTADORGX 1
3
? |

```

Figura 5.16 - Resultado no uso de variável global com procedimento e sub-rotina

Na sequência veja o procedimento chamado "VARLOCAL2" com a definição local de variável e uso de sub-rotina para a ação de contagem. Assim sendo, e entre o seguinte código:

```

TO VARLOCAL2
  LOCAL "CONTADORLX
  MAKE "CONTADORLX 1
  PR :CONTADORLX
  SUBVARLOCAL2
END

```

```
TO SUBVARLOCAL2
  MAKE "CONTADORLX :CONTADORLX + 1
  PR :CONTADORLX
END
```

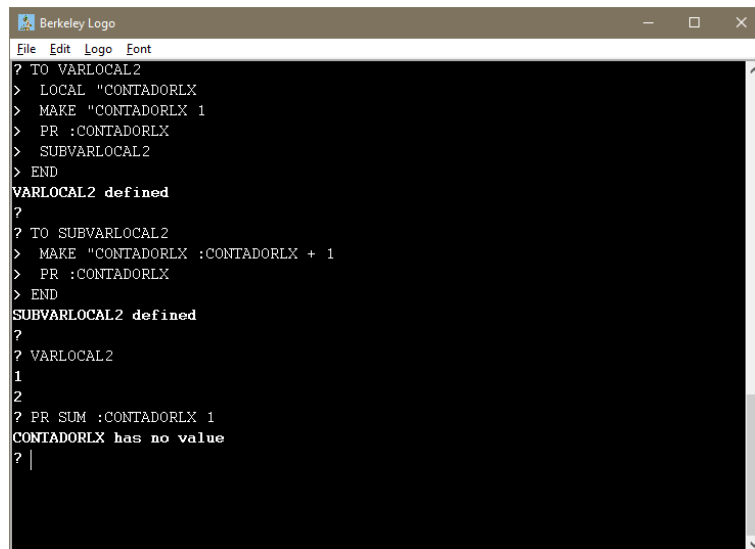
Em seguida execute a instrução de chamada do procedimento:

```
VARLOCAL2
```

Observe os resultados apresentados e na sequência na linha de comando do campo de entrada de comandos, dados e instruções informe a seguinte instrução:

```
PR SUM :CONTADORLX 1
```

Veja a apresentação dos resultados "1", "2" e "CONTADORLX não possui um valor" a partir da execução do procedimento principal com chamada de sub procedimento (sub-rotina) utilizando-se variável local, indicado na figura 5.17.



```
Berkeley Logo
File Edit Logo Font
? TO VARLOCAL2
> LOCAL "CONTADORLX
> MAKE "CONTADORLX 1
> PR :CONTADORLX
> SUBVARLOCAL2
> END
VARLOCAL2 defined
?
? TO SUBVARLOCAL2
> MAKE "CONTADORLX :CONTADORLX + 1
> PR :CONTADORLX
> END
SUBVARLOCAL2 defined
?
? VARLOCAL2
1
2
? PR SUM :CONTADORLX 1
CONTADORLX has no value
? |
```

Figura 5.16 - Resultado no uso de variável local com procedimento e sub-rotina

## 5.7 - Exercícios de fixação

1. Criar procedimento chamado **CAP0501** que efetue a leitura de um valor numérico inteiro e apresente o resultado do valor lido elevado ao quadrado sem efetuar o armazenamento do resultado em memória. A variável que receberá a entrada do dado deve ser definida como local.
2. Criar procedimento chamado **CAP0502** que efetue a leitura de um valor numérico inteiro e apresente o resultado do valor lido elevado ao cubo com armazenamento do resultado calculado em memória. As variáveis devem ser definidas como local.
3. Criar procedimento chamado **CAP0503** que efetue a leitura de uma temperatura em graus Celsius e apresente essa temperatura em graus Fahrenheit, sua conversão. A fórmula de conversão é " $F \leftarrow C * 9 / 5 + 32$ ", sendo "F" a temperatura em Fahrenheit e "C" a temperatura em Celsius. Armazene em memória o resultado calculado. Use variáveis locais. Formate a saída numérica com duas casas decimais.

4. Criar procedimento chamado **CAP0504** que efetue a leitura de uma temperatura em graus Fahrenheit apresente essa temperatura em graus Celsius, sua conversão. A fórmula de conversão é  $C \leftarrow ((F - 32) * 5) / 9$ , sendo "F" a temperatura em Fahrenheit e "C" a temperatura em Celsius. Armazene em memória o resultado calculado. Use variáveis locais. Formate a saída numérica com duas casas decimais.
5. Criar procedimento chamado **CAP0505** que efetue a leitura de dois valores numéricos inteiros (representados pelas variáveis locais "A" e "B") e mostre o resultado armazenado em memória do quadrado da diferença do primeiro valor (variável "A") em relação ao segundo valor (variável "B") junto a variável local "R".
6. Criar procedimento chamado **CAP0506** que efetue a leitura de um número inteiro qualquer em uma variável local e multiplique este número por "2" armazenando o resultado em memória. Apresentar o resultado da multiplicação somente se o resultado for maior que "30".
7. Criar procedimento chamado **CAP0507** que efetue a leitura de dois valores numéricos reais representados pelas variáveis locais "A" e "B" e apresente o resultado da diferença do maior valor pelo menor valor sem armazenar o resultado em memória.
8. Criar procedimento chamado **CAP0508** que efetue a leitura de dois valores numéricos reais representados pelas variáveis locais "A" e "B" e apresente o resultado da diferença do maior valor pelo menor valor com armazenamento do cálculo em memória.
9. Criar procedimento chamado **CAP0509** que efetue a leitura de três valores numéricos inteiros desconhecidos representados pelas variáveis locais "A", "B" e "C". O procedimento deve somar esses valores, armazenar o resultado em memória e apresentar este resultado somente se for "**maior ou igual**" a "100".
10. Criar procedimento chamado **CAP0510** que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se a primitiva "**REPEAT**".
11. Criar procedimento chamado **CAP0511** que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se a primitiva "**WHILE**".
12. Criar procedimento chamado **CAP0512** que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se a primitiva "**DO.UNTIL**".
13. Criar procedimento chamado **CAP0513** que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se a primitiva "**FOR**".
14. Criar procedimento chamado **CAP0514** que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use a primitiva "**REPEAT**".
15. Criar procedimento chamado **CAP0515** que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use a primitiva "**WHILE**".
16. Criar procedimento chamado **CAP0516** que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use a primitiva "**DO.UNTIL**".

17. Criar procedimento chamado **CAP0517** que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use a primitiva "FOR".
18. Criar procedimento chamado **CAP0518** que mostre todos os valores numéricos divisíveis por "4" *menores* que "20" iniciando a contagem em "1". Use a primitiva "REPEAT".
19. Criar procedimento chamado **CAP0519** que mostre todos os valores numéricos divisíveis por "4" *menores* que "20" iniciando a contagem em "1". Use a primitiva "WHILE".
20. Criar procedimento chamado **CAP0520** que mostre todos os valores numéricos divisíveis por "4" *menores* que "20" iniciando a contagem em "1". Use a primitiva "DO.UNTIL".
21. Criar procedimento chamado **CAP0521** que mostre todos os valores numéricos divisíveis por "4" *menores* que "20" iniciando a contagem em "1". Use a primitiva "FOR".
22. Criar procedimento chamado **CAP0522** que apresente os valores numéricos inteiros compreendidos na faixa de "0" e "4". Use a primitiva "REPEAT".
23. Criar procedimento chamado **CAP0523** que apresente os valores numéricos inteiros compreendidos na faixa de "0" e "4". Use a primitiva "WHILE".
24. Criar procedimento chamado **CAP0524** que apresente os valores numéricos inteiros compreendidos na faixa de "0" e "4". Use a primitiva "DO.UNTIL".
25. Criar procedimento chamado **CAP0525** que apresente os valores numéricos inteiros compreendidos na faixa de "0" e "4". Use a primitiva "FOR".
26. Criar procedimento chamado **CAP0526** que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use a primitiva "REPEAT".
27. Criar procedimento chamado **CAP0527** que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use a primitiva "WHILE".
28. Criar procedimento chamado **CAP0528** que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use a primitiva "DO.UNTIL".
29. Criar procedimento chamado **CAP0529** que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use a primitiva "FOR".