

PROGRAMAÇÃO DE COMPUTADORES PARA INICIANTES COM SMALL BASIC

Introdução com turtle graphics e um pouco mais



JOSÉ AUGUSTO N. G. MANZANO

Augusto N. G. Manzano



ORCID: 0000-0001-9248-7765

PROGRAMAÇÃO DE COMPUTADORES PARA INICIANTES COM SMALL BASIC

Introdução com turtle graphics e um pouco mais

Versão 1.2

**São Paulo
2021 - Propes Vivens**

ISBN 978-65-00-27582-7
9 786500 275827

© Copyright 2021 by José Augusto N. G. Manzano / Propes Vivens.

Todos os direitos reservados. Proibida a reprodução total ou parcial, por qualquer meio ou processo, especialmente por sistemas gráficos, microfilmicos, fotográficos, reprogramáticos, fonográficos, videográficos, internet, e-books. Vedada a memorização e/ou recuperação total ou parcial em qualquer sistema de processamento de dados e a inclusão de qualquer parte da obra em qualquer programa jus cibernetico existentes ou que a venham existir. Essas proibições aplicam-se também às características gráficas da obra e à sua editoração, exceto pelo exporto no próximo parágrafo. A violação dos direitos autorais é punível como crime (art. 184 e parágrafos, do Código Penal, conforme Lei nº 10.695, de 07.01.2003) com pena de reclusão, de dois a quatro anos, e multa, conjuntamente com busca e apreensão e indenizações diversas (artigos 102 e 103, parágrafo único, 104, 105, 106 e 107 itens 1, 2 e 3 da Lei nº 9.610, de 19.06.1998, Lei dos Direitos Autorais).

Esta obra é distribuída gratuitamente em formato digital (somente em PDF) apenas e tão somente no sítio do autor (www.manzano.pro.br) e na forma impressa comercialmente e disponibilizada nas plataformas **Clube de Autores** e **Agbook**. Nenhum outro local da Internet ou fora dela está autorizado a distribuir, seja gratuitamente ou comercialmente este material. Não é permitido o compartilhamento deste material em qualquer lugar ou por qualquer meio exceto o exposto neste parágrafo, bem como outro formato digital. Os infratores estão sujeitos a processo judicial.

O Autor acredita que as informações aqui apresentadas estão corretas e podem ser utilizadas para qualquer fim legal. Entretanto, não existe qualquer garantia, explícita ou implícita, de que o uso de tais procedimentos conduzirá sempre ao resultado desejado. O autor e a editora não poderão ser responsabilizados civilmente ou criminalmente. Os nomes de sítios e empresas, mencionados, foram utilizados apenas como ilustração, não havendo nenhum vínculo com a obra, não garantindo a sua existência nem divulgação a posteriori.

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Manzano, Augusto N. G.
Programação de computadores para iniciantes com
small basic : introdução com turtle graphics e um
pouco mais / Augusto N. G. Manzano. -- 1. ed. -- São
Paulo : Ed. do Autor, 2021.

ISBN 978-65-00-27582-7

1. Algoritmos de computadores 2. Ciência da
computação 3. Linguagem de programação (Computadores)
I. Título.

21-74842

CDD-005.133

Índices para catálogo sistemático:

1. Linguagem de programação : Computadores :
Processamento de dados 005.133

Aline Grazielle Benitez - Bibliotecária - CRB-1/3129

ED.VER - DATA
1.00 - 31/07/2021

Produção e Editoração: José Augusto Navarro Garcia Manzano
Capa: canva.com / Espiral hexagonal (veja apêndice)

Edição: Propes Vivens

FERRAMENTA UTILIZADA

Produto: **Small Basic**

Sítio: <http://smallbasic.com>

Desenvolvedor: Microsoft Corporation (Brasil)

Av. Presidente Juscelino Kubitscheck, 1909 - Torre Sul, 16° andar

São Paulo - SP (Vila Nova Conceição)

CEP: 04543-907

Microsoft Corporation (USA)

One Microsoft Way

Redmond, WA (Washington)

ZIP: 98052-6399

REQUISITOS DE HARDWARE E SOFTWARE

- ◆ Processador com 1 GHz
- ◆ 512 MB de memória RAM;
- ◆ 5 GB de espaço em disco para instalação do ambiente;
- ◆ Sistema Operacional Windows 8, 8.1 e 10;
- ◆ Monitor com 1024 x 768 pontos ou superior para melhor visualização;
- ◆ Mouse ou outro periférico de apontamento;
- ◆ Modem e acesso à Internet.

Está sendo considerado o uso da plataforma .Net de 4.5 até 4.8

CARTA AO ESTUDANTE

Olá, estudante de programação.

Espero que você esteja bem e com excelente animo para desenvolver habilidades na arte da programação de computadores a partir de uma linguagem de programação imperativa "*Small Basic*". Este livro vem de encontro a sua aprendizagem fornecendo diversos aspectos práticos no uso da linguagem como suporte as aulas de lógica de programação, podendo e devendo ser usado por pessoas de todas as idades.

A linguagem "*Small Basic*" é baseada na famosa linguagem BASIC em estilo estruturado. Sua aprendizagem proporciona excelente visão sobre detalhes lógicos de forma divertida e descontraída partindo-se de exemplos focados em imagens geométricas e avançado para formas contextualizadas de programação que se assemelham ao estilo usado nas grandes organizações. Neste livro se faz uma introdução a linguagem e seus principais recursos de operação desde o uso da simulação da geometria da tartaruga (lembrando a linguagem "*Logo*") até o desenvolvimento de programas independentes da operação gráfica.

O livro encontra-se dividido em cinco capítulos, mais alguns apêndices complementares que abordam diversas características de uso da linguagem, onde praticamente todos os capítulos, exceto o primeiro possuem como reforço um conjunto de exercícios de fixação. São fornecidas orientações desde a obtenção e instalação da fermenta de trabalho a apresentação de diversas ações que podem ser produzidas com a linguagem, como: sub-rotinas; operadores aritméticos, lógicos e relacionais; ações de entrada e saída; decisões; recursividades; simulação de uso de variáveis locais e passagem de parâmetro, entre outros. São demonstradas ações que produzem formas geométricas a partir do uso do objeto **Turtle**, bem como a demonstração de outros objetos da linguagem como: **Array**, **Text**, **Math**, entre outros.

Cabe destacar que este livro é distribuído gratuitamente no site do autor ou disponibilizado comercialmente em sua forma impressa a partir das plataformas de publicação Clube de Autores e Agbook para quem desejar ter o material no formato livro impresso.

Este trabalho não passou por revisões de língua portuguesa e está em sua forma bruta, versão pré-alpha. Por isso, poderão ser encontrados erros de escrita, concordância ou outros que passaram "batidos", exceto os códigos de programas apresentados os quais foram exaustivamente testados. Auxílios, neste sentido, são sempre bem vindos, desde que sejam realizados sem nenhum interesse financeiro ou comercial.

Para auxiliar parte do estudo você encontrará os arquivos dos códigos dos exercícios de aprendizagem disponíveis para aquisição em: https://github.com/J-AugustoManzano/livro_Small-Basic-1.2.

Espero que este conteúdo possa lhe ser bastante útil.

Com grande abraço.
Augusto Manzano.

AGRADECIMENTOS

À minha esposa Sandra e à minha filha Audrey, motivos de constante inspiração ao meu trabalho. Pela paciência nos momentos de ausência que tenho, quando estou absorto em escrever, dedicando-me ao ensino.

Há você que me lê neste livro e a todos os estudantes que passaram e passam por minhas mãos, que acreditaram e acreditam na minha pessoa e seguiram e seguem as orientações passadas; por me incentivarem continuamente quando me questionam sobre temas que ainda não conheço, por me levarem a um patamar maior, por exigirem assim que eu pesquise mais e retorno a você conhecimentos na forma de livros.

Vida longa e próspera.

SUMÁRIO

1.	INTRODUÇÃO AO SMALL BASIC	
1.1.	Linguagem BASIC.....	13
1.2.	Linguagem Small Basic	14
1.3.	Obtenção e instalação	16
1.4.	O ambiente de trabalho	20
2.	AÇÕES INICIAIS	
2.1.	A origem da tartaruga e o universo Small Basic	25
2.2.	Objetos Small Basic	26
2.3.	Programação sequencial	31
2.4.	Exercícios de fixação	35
3.	AÇÕES ESPECIALIZADAS	
3.1.	Repetições	37
3.2.	Sub-rotinas	41
3.3.	Simulação de passagem de parâmetro.....	47
3.4.	Programação com decisão	49
3.5.	Recursão	53
3.6.	Exercícios de fixação	56
4.	AÇÕES ESPECÍFICAS	
4.1.	Randomização	57
4.2.	Coordenadas	58
4.3.	Cores	63
4.4.	Fractais	66
4.5.	Mais repetições	70
4.6.	Exercícios de fixação	75
5.	AÇÕES COMPLEMENTARES	
5.1.	Decisões sequenciais	77
5.2.	Variáveis indexadas	81
5.3.	Manipulação de texto	85
5.4.	Efeitos sonoros	92
5.5.	Programação gráfica	96
5.6.	Endentação automatizada	105
5.7.	Uso de arquivos	107
5.8.	Exercícios de fixação	121

APÊNDICES

A.	Imagen da capa	123
B.	Tabela de cores	125
C.	Arranjos e pilhas	129
D.	Passagem de parâmetro e retorno	137
E.	Variáveis locais	141
F.	Classificação de dados	145
G.	Extensão LitDev	153
H.	Gabarito	157
	REFERÊNCIAS BIBLIOGRÁFICAS	171

CAPÍTULO 1

Introdução ao Small Basic

Programar é a forma pela qual se faz o desenvolvimento de certo software para computadores a partir do uso de linguagens de programação. São muitas as linguagens de programação existentes. Assim como temos diversos idiomas falados no mundo, temos também diversas linguagens para programar computadores. Este livro abrange o uso de uma delas: neste caso a linguagem "Small Basic" desenvolvida em 2008 pela empresa Microsoft.

1.1 - Linguagem BASIC

Antes de falarmos sobre a linguagem "Small Basic" é importante falarmos um pouco sobre uma famosa linguagem de programação chamada "BASIC" desenvolvida em 1964 no *Dartmouth College* por *Mary Kenneth Keller, John George Kemeny e Thomas Eugene Kurtz*, professores da instituição (SONYMASTER, 2020). O *Dartmouth College* (Faculdade de Dartmouth) é uma instituição de ensino privada estadunidense fundada no ano de 1769, localizada na cidade de *Hanover*, no Estado de *New Hampshire* (DARTMOUTH, 2021).

O nome "BASIC" apesar de sugerir "BÁSICO" é na verdade uma sigla para *Beginner's All-purpose Symbolic Instruction Code* (Código de Instrução Simbólica de Uso Geral para Iniciantes - CISUGI). BASIC é uma linguagem que se destacou por ter sido desenvolvida exclusivamente para o auxílio na aprendizagem de lógica e programação de computadores, tornou-se popular após 1975 com o surgimento dos primeiros microcomputadores de 8 bits, pois todo o microcomputador dessa época vinha com uma versão da linguagem pré-instalada em sua ROM (*Read Only Memory*). Apesar da linguagem ter caído em certo desuso ela passou por três gerações evolucionárias sendo usada até os dias de hoje.

Na primeira geração o código BASIC para ser usado necessitava ser definido em linhas numeradas, normalmente de dez em dez. Apesar de ser uma linguagem imperativa e muito fácil de se usar não possuía nenhuma elegância sintática no que tange a escrita de programas estruturados. Para dar-lhe uma ideia veja o código seguinte de um programa que apresenta os resultados da tabuada de um número inteiro qualquer entre "2" e "9". Valores diferentes da faixa de aceite devem ser recusados e nova entrada deve ser solicitada. Não se preocupe em entender o que está efetivamente escrito, observe apenas o emaranhado de instruções e comandos. Ressaltando que "comando" é a forma de se passar uma ordem de execução a ser executada em um computador, sendo que o conjunto de comandos que especificam isoladamente uma ideia ou ordem são comados de "instruções". Por sua vez um conjunto de instruções (formadas por um ou mais comandos) caracterizam-se em ser um "programa".

```
10 CLS
20 PRINT "PROGRAMA: TABUADA" : PRINT
30 INPUT "ENTRE UM VALOR NUMERICO INTEIRO ENTRE '2' E '9': "; N
40 IF (N < 2 OR N > 9) THEN GOTO 30
50 FOR I = 1 TO 10 STEP 1
60 LET R = N * I
70 PRINT USING "## X ## = ###"; INT(N), I, R
80 NEXT I
90 END
```

Veja que cada linha do programa é uma instrução. A linha 10, por exemplo, tem uma instrução formada por um só comando (**CLS**). Já a linha 40 tem sua instrução formada por dois comandos (um **IF/THEN** e um **GOTO**).

Observe que o programa possui a definição de uma ação de entrada de dados na linha 30, de ações de processamento nas linhas 30 (processamento lógico) e 60 (processamento aritmético) e ação de saída na linha 70. Isto mostra que um programa de computador executa três ações básicas: a entrada, o processamento e a saída. A instrução da linha 70 (**PRINT USING**) garante a apresentação dos resultados inteiros mesmo que um valor real seja fornecido.

O estilo anterior de escrita de programa "BASIC" foi com o tempo remodelado, surge a *segunda geração* no início da década de 1980 trazendo uma mudança radical na estrutura de escrita do código, passando a permitir o uso de código estruturado sem a necessidade de usar comando "**GOTO**" (apesar deste existir nesta nova geração) como mostra a linha 40 do código anterior. Desta forma o código do programa de tabuada pôde ser escrito como:

```
CLS
PRINT "PROGRAMA: TABUADA" : PRINT
DO
    INPUT "ENTRE UM VALOR NUMERICO INTEIRO ENTRE '2' E '9': "; N
    LOOP WHILE (N < 2 OR N > 9)
    FOR I = 1 TO 10 STEP 1
        R = N * I
        PRINT USING "## X ## = ###"; N, I, R
    NEXT
END
```

Nessa época a Microsoft lança o ambiente de desenvolvimento "*QuickBASIC*" que posteriormente foi substituído pelo ambiente "*Visual Basic*". Veja que o código passa a ser escrito em algumas partes com um leve deslocamento a direita, característica da escrita de códigos estruturados a partir de uma técnica conhecida como *endentaçāo*.

Por volta do ano 2000 a linguagem "BASIC" passa por mais uma repaginada chegando a sua *terceira geração*, agora com suporte à *Programação Orientada a Objetos*. Com as alterações ocorridas a linguagem se distanciou de sua raiz: ser uma linguagem de programação educacional para o ensino de lógica e programação de computadores simplificada. Nesse interim, por volta do ano de 2008, a Microsoft lança a linguagem "*Small Basic*" distribuída sem custo de licença de uso, ou seja, é um programa em estilo *freeware*.

Na primeira geração temos o que se pode char de "*Classic BASIC*" e nas segunda e terceira gerações têm-se o que se pode chamar de "*Structured BASIC*".

1.2 - Linguagem Small Basic

A linguagem "Small Basic" é uma versão simplificada da linguagem "BASIC" existente no ambiente "Visual Basic". A linguagem foi desenvolvida com o objetivo de retomar as raízes da era do "Classic BASIC", sendo uma ferramenta voltada ao ensino de principiantes independentemente da idade que possuam, sejam crianças, adolescentes ou adultos, executada sobre a plataforma ".Net Framework". Cuidado em não confundir "Small Basic" com o projeto "SmallBASIC", por serem propostas diferentes.

A linguagem "Small Basic" possui um conjunto pequeno de operações, se comparada com outras linguagens de programação inclusive em relação a sua irmã mais velha "Visual Basic". Seus recursos estão acondicionados em vinte grupos de ações chamados **Objetos** (não confundir com *objetos* do paradigma da *Programação Orientada a Objetos*). Os objetos são: **Array, Clock, Controls, Desktop, Dictionary, File, Flickr, GraphicsWindow, ImageList, Math, Mouse, Network, Program, Shapes, Sound, Stack, TextWindow, Text, Timer e Turtle**.

Os *objetos* são como bibliotecas de módulos que agrupam recursos chamados: **Operações, Propriedades e/ou Eventos**. Um objeto para ser usado será descrito como sendo: **Objeto.Operação, Objeto.Propriedade ou Objeto.Evento**.

Para auxiliar o uso dos *objetos* há um pequeno conjunto de *comandos* formado por quatorze palavras-chave: **Else, Elself, EndFor, EndIf, EndSub, EndWhile, For, Goto, If, Step, Sub, Then, To e While**. Além de dois operadores lógicos para auxiliar operações lógicas, sendo: **And e Or**.

O foco deste trabalho está direcionado as ações da *geometria da tartaruga* (turtle graphics) e por esta razão serão usados mais intensamente os *objetos* **Math, TextWindow, Text e Turtle**. Eventualmente são usados outros objetos. Em conjunto com os objetos serão usadas, de alguma forma, as palavras-chave e os operadores lógicos.

Por ser uma versão simplificada a linguagem "Small Basic" não é projetada para o desenvolvimento de aplicações comerciais e industriais, mas em seu universo de atuação: a área educacional permite que sejam simuladas ações profissionais simplificadas o que permite a obtenção de uma visão mais apurada do que efetivamente é a programação de computadores.

A título de ilustração veja a proposta do programa de apresentação da tabuada de um número inteiro qualquer entre "2" e "9" codificado em "Small Basic"

```
TextWindow.Clear()
TextWindow.WriteLine("PROGRAMA: TABUADA")
TextWindow.WriteLine("")
Loop:
    TextWindow.Write("ENTRE UM VALOR NUMERICO INTEIRO ENTRE '2' E '9': ")
    N = Math.Floor(TextWindow.ReadNumber())
    If (N < 2 Or N > 9) Then
        Goto Loop
    EndIf
EndLoop:
For I = 1 To 10 Step 1
    R = N * I
    TextWindow.WriteLine(N + " X " + I + " = " + R)
EndFor
```

Uma das principais diferenças na forma de escrita do código é o estilo no formato dos caracteres. Na primeira e segunda gerações os comandos são escritos em caracteres maiúsculos, na terceira geração são escritos, mesmo que informados em contrário, obrigatoriamente seguindo o estilo conhecido como "*Pascal Case*", ou seja, *notação Pascal*, criado no ano de 1813 por Jacob Berzelius para simplificar a definição de fórmulas químicas (KITAMURA, 2017).

Outra diferença é o uso do objeto **Math** com a operação **Floor()** para garantir que o número informado seja inteiro, mesmo quando não for. Veja que essa questão foi anteriormente definida com "**PRINT USING**". Diferentemente dos dois códigos apresentados este código não tem

como apresentar os valores numéricos alinhados da direita para a esquerda, pois a linguagem não possui diretamente este recurso. É possível resolver esta questão mas o código para isso não é assunto para este livro.

1.3 - Obtenção e instalação

Para fazer a aquisição do programa "Small Basic" abra seu programa de navegação e informe na barra de endereço a URL (*Uniform Resource Locator*): <http://smallbasic.com>. Veja na figura 1.1 a imagem do sitio do programa aberto.

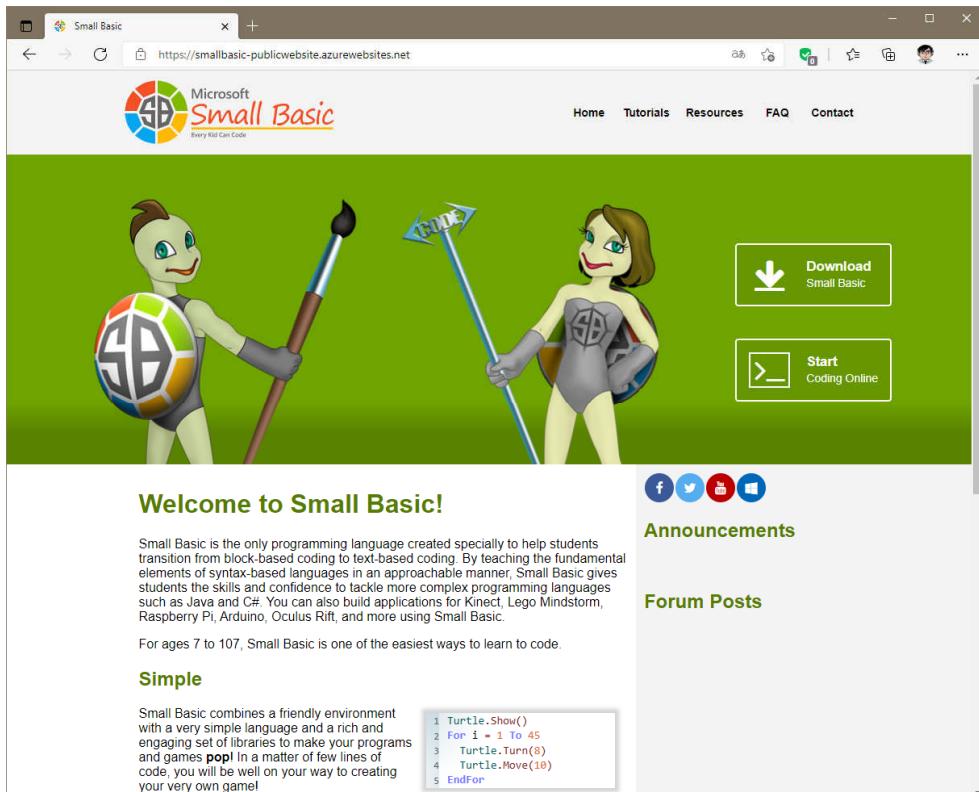


Figura 1.1 - Sítio oficial do programa "Small Basic"

Veja ao lado direito, como indicado na figura 1.2, há a indicação de um botão verde denominado "**Download Small Basic**", acione-o com o ponteiro do *mouse*.



Figura 1.2 - Seleção do botão "Download Small Basic"

Aguarde o arquivo ser copiado para seu computador. Isso, normalmente, é realizado automaticamente junto a pasta "**Downloads**". Feche o programa de navegação e abra programa de gerenciamento de arquivos "**Explorer**" e vá até a pasta "**Downloads**", selecionando com um duplo clique do ponteiro do *mouse* o arquivo "**SmallBasic_v1.2**" (versão disponível quando este texto foi produzido) para colocá-lo em execução.

Provavelmente a primeira ocorrência a surgir é a apresentação da caixa de advertência "**Abrir Arquivo - Aviso de Segurança**" pedindo autorização para instalação, como indica a figura 1.3. Neste momento, se há o desejo de instalar acione o botão "**Executar**". Se for acionado o botão "**Cancelar**" a instalação é interrompida.

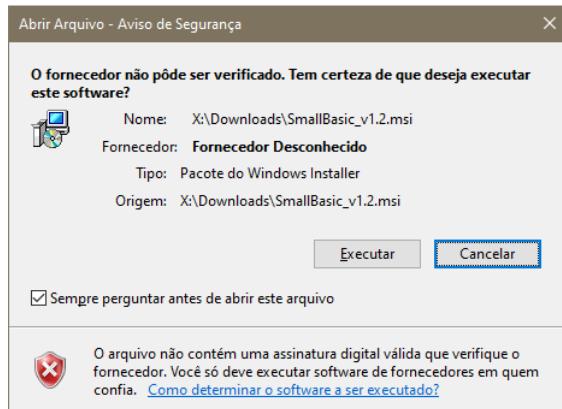


Figura 1.3 - Caixa de advertência "Abrir Arquivo - Aviso de Segurança"

Ao ser acionado o botão "**Executar**" ocorre a apresentação da primeira tela do programa de instalação como indica a figura 1.4.



Figura 1.4 - Tela inicial do programa de instalação

Para iniciar a instalação acione o botão "**Next**". Será apresentada a tela com a indicação do contrato de uso que para poder instalar o programa deve ser aceito. Assim sendo, leia o contrato, marque a opção "**I accept the terms in the License Agreement**", como apresentado na figura 1.5 e acione o botão "**Next**".

A próxima tela é muito importante pois é onde você escolhe o idioma de apresentação do ambiente de programação. Desta forma dê um clique com o ponteiro do *mouse* no símbolo "+" ao lado esquerdo da indicação "**Main Files**" e veja que será aberta uma lista de opções de idiomas a serem selecionados. É possível selecionar mais de um idioma, se assim desejar. Para o nosso caso, basta selecionar "**Português (Brasil)**". Quando um idioma é selecionado apresenta-se um pequeno menu com algumas opções, selecione neste momento a opção "**Will be installed on local hard drive**" como indicado na figura 1.6. Na sequência acione o botão "**Next**".

Será apresentada a última tela antes do início da instalação (figura 1.7). Desejando rever alguma seleção ou opção anterior acione o botão "**Back**", caso contrário acione o botão "**Install**".

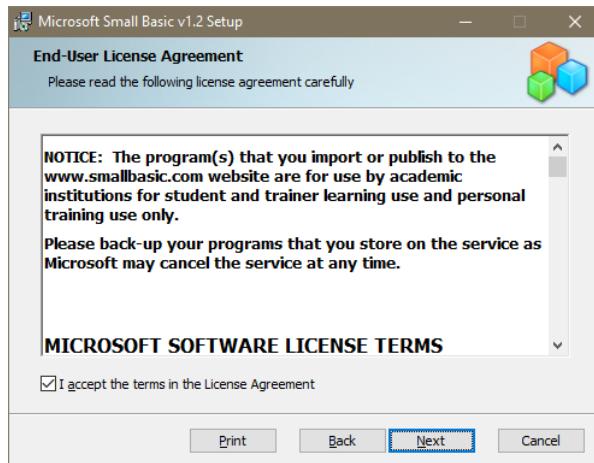


Figura 1.5 - Contrato de licença

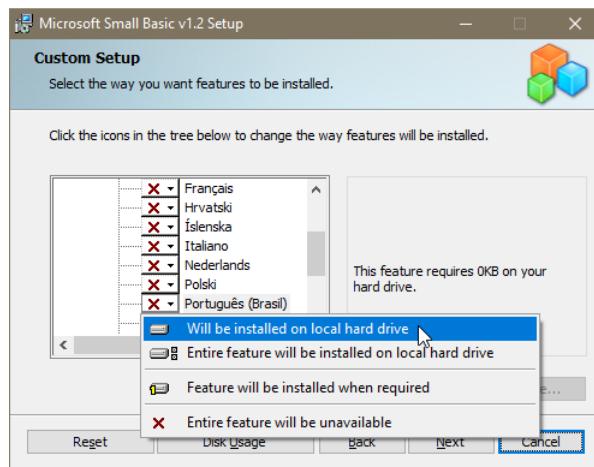


Figura 1.6 - Seleção do idioma da interface de uso



Figura 1.7 - Tela antes da instalação

Assim que o botão "Install" é acionado ocorre a apresentação da tela indicada na figura 1.8 mostrando a evolução do processo de instalação. Aguarde o término desta ocorrência que poderá ser um pouco demorada, dependendo da configuração do ambiente computacional em uso.

Assim que o processo de instalação for encerrado ocorrer a apresentação da última tela de instalação como mostra a figura 1.9. Neste momento, basta apenas acionar a execução do botão "Finish".

Para executar o programa acesse no "**Windows 10**" o botão "**Iniciar**", localize na lista do menu a pasta "**Small Basic**", selecione o ícone "**Microsof Small Basic**" e será apresenta a imagem da figura 1.10.

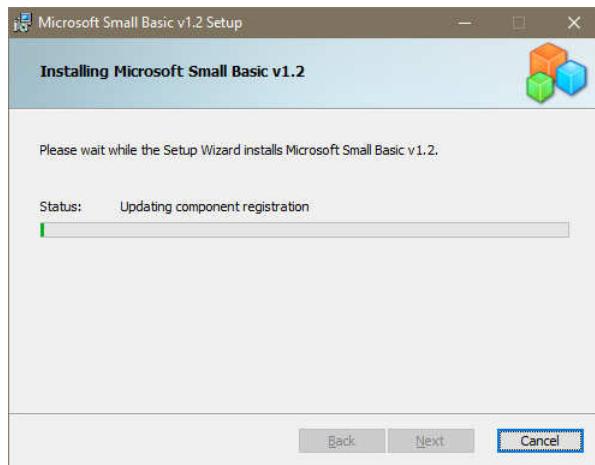


Figura 1.8 - Tela com andamento da instalação

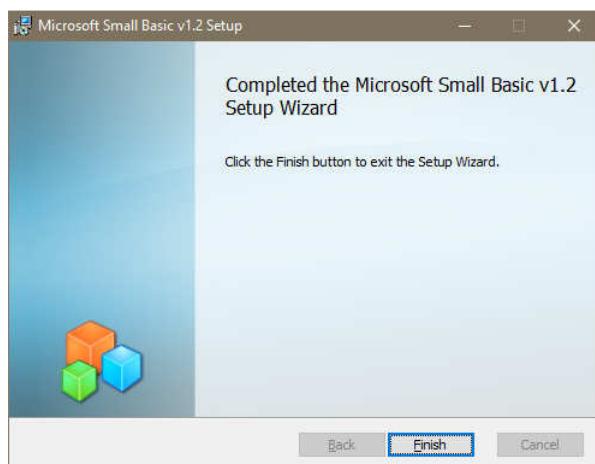


Figura 1.9 - Tela com encerramento da instalação

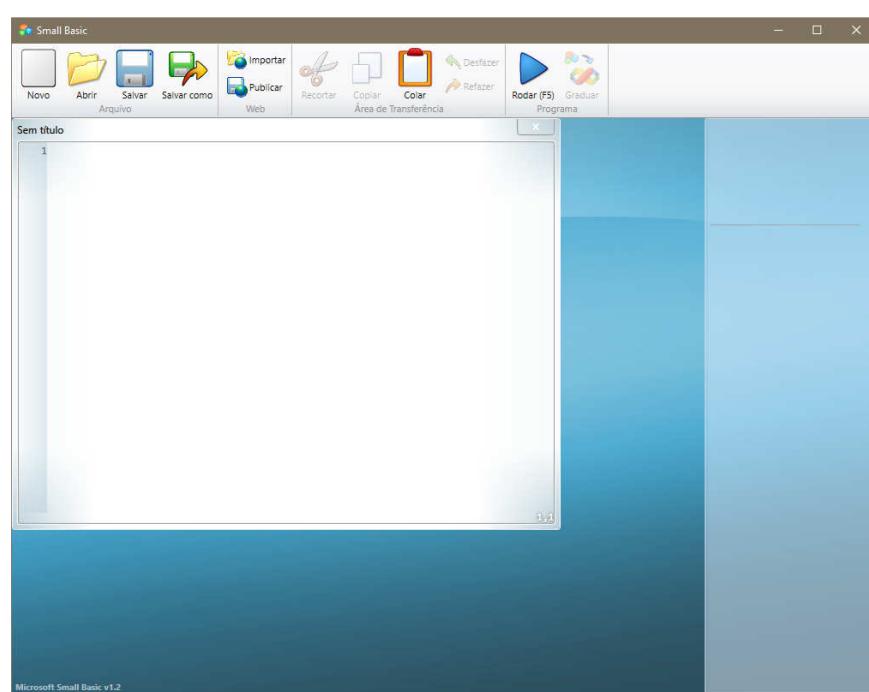


Figura 1.10 - Tela do ambiente de programação "Microsoft Small Basic"

Para encerra o ambiente use o botão "X" da barra de título ou acione o atalho "<Alt>+<F4>".

1.4 - O ambiente de trabalho

A linguagem "*Small Basic*" para ser usada depende do ambiente "*Microsoft Small Basic*". Assim sendo, é necessário sempre estar com o ambiente carregado para ser possível programar nesta linguagem. Como ambiente de desenvolvimento esta é uma das ferramentas mais simples existentes possuindo uma interface minimalista e simplificada. Veja os detalhes apontados na figura 1.11.

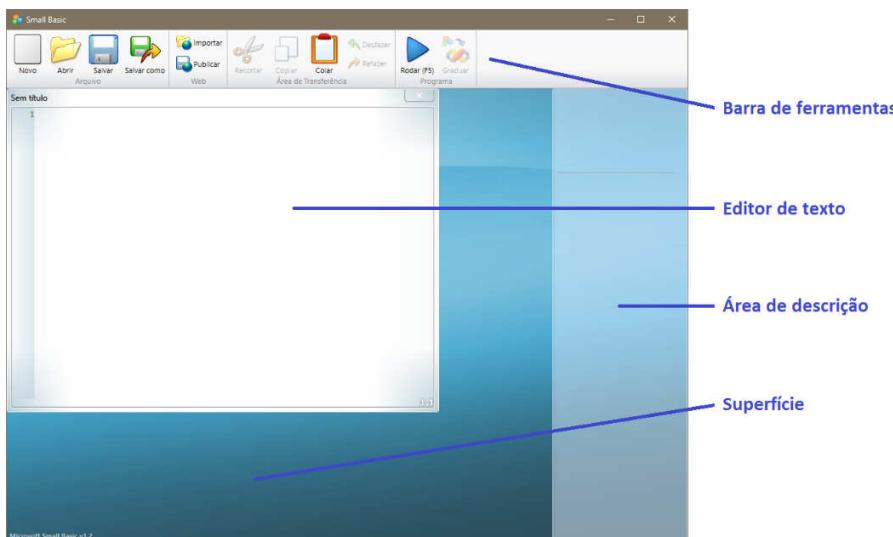


Figura 1.11 - Elementos contextuais de operação

Na "**Barra de ferramentas**" você tem acesso aos comandos que controlam o ambiente, estando dividida em área de ação: "**Arquivo**", "**Web**", "**Área de Transferência**" e "**Programas**". Cada área possui um conjunto de opções:

- ❖ Em "**Arquivo**" é possível criar um novo projeto (botão "**Novo**" ou "<Ctrl>+<N>"), abrir um projeto existente (botão "**Abrir**" ou "<Ctrl>+<O>"), gravar um projeto em uso (botão "**Salvar**") "<Ctrl>+<S>" ou gravar um projeto com outro nome (botão "**Salvar como**");
- ❖ Em "**Web**" é possível importar um programa da Web (botão "**Importar**" ou "<Ctrl>+<Shift>+<O>") ou publicar um programa na Web (botão "**Publicar**" ou "<Ctrl>+<Shift>+<S>");
- ❖ Em "**Área de Transferência**" é possível recortar parte de um código selecionado (botão "**Recortar**" ou "<Ctrl>+<X>"), copiar um trecho de código selecionado (botão "**Copiar**" ou "<Ctrl>+<C>"), colar um trecho de código que tenha sido recortado ou copiado (botão "**Colar**" ou "<Ctrl>+<V>"), desfazer uma ação (botão "**Desfazer**" ou "<Ctrl>+<Z>") e refazer uma ação (botão "**Refazer**" ou "<Ctrl>+<Y>");
- ❖ Em "**Programas**" é possível executar um programa (botão "**Executar**") que pode ser substituído pelo atalho "**<F5>**" e exportar o programa para o Visual Basic (botão "**Graduar**").

No "**Editor de texto**" é onde escrevemos nossos programas e onde podemos editá-los. Os programas abertos para uso são apresentados nesta área, onde é possível fazer uso das seguintes ações:

- ❖ As teclas de setas permitem posicionar o cursor dentro do código escrito;
- ❖ A tecla "**<Home>**" posiciona o cursor no início da linha;

- ❖ A tecla "<End>" posiciona o cursor no final da linha;
- ❖ As teclas "<Ctrl>+<Home>" posiciona o cursor no início do código (primeira linha);
- ❖ As teclas "<Ctrl>+<End>" posiciona o cursor no fim do código (última linha)
- ❖ As teclas "<Ctrl>+<Seta para cima>" rola tela para cima;
- ❖ As teclas "<Ctrl>+<Seta para baixo>" rola tela para baixo;
- ❖ As teclas "<Ctrl>+<Seta para frente>" posiciona cursor no início da próxima palavra;
- ❖ As teclas "<Ctrl>+<Seta para trás>" posiciona cursor no início da palavra anterior;
- ❖ As teclas "<Ctrl>+<A>" seleciona todo o texto;
- ❖ As teclas "<Ctrl>+<F>" abre caixa de pesquisa;
- ❖ As teclas "<F3>" abre caixa de pesquisa dando continuidade na busca;
- ❖ As teclas "<Ctrl>+<L>" apaga linha;
- ❖ As teclas "<Ctrl>+<W>" seleciona a palavra a frente do cursor;
- ❖ A tecla "" remove caractere a frente do cursor ou as partes selecionadas de um texto;
- ❖ A tecla "<Ctrl>+" remove palavra a frente do cursor;
- ❖ A tecla "<Ctrl>+<Shift>+<U>" coloca em maiúsculo o trecho de texto selecionado;
- ❖ A tecla "<Ctrl>+<T>" coloca caractere a frente do cursor na próxima posição do texto;
- ❖ A tecla "<Ctrl>+<U>" coloca em minúsculo o trecho de texto selecionado;
- ❖ A tecla "<Ctrl>+<Espaço>" apresenta o modo *IntelliSense*;
- ❖ A tecla "<Ctrl>+<+>" ou "<Ctrl>+<roda do mouse para cima>" aumenta o zoom da tela de edição;
- ❖ A tecla "<Ctrl>+<->" ou "<Ctrl>+<roda do mouse para baixo>" diminui o zoom da tela de edição;
- ❖ A tecla "<Ins>" ativa ou desativa os modos de inserção e sobreposição;
- ❖ As setas acionadas com a tecla "<Shift>" selecionam partes do texto escrito.

A "Área de descrição" apresenta explicações sobre os recursos em uso ou utilizados da linguagem.

A "Superfície" é o local onde as janelas do editor se encontram e onde são apresentadas eventuais mensagens ao programador.

Para um teste escreva na primeira linha do editor de texto do ambiente o comando baseado no uso de um **Objeto** e sua **Operação** que efetua a limpeza de uma tela de texto em uso pausadamente:

[TextWindow.Clear\(\)](#)

Observe que ao escrever "**Text**" ocorre a apresentação de um conjunto de informações como indica a figura 1.12.

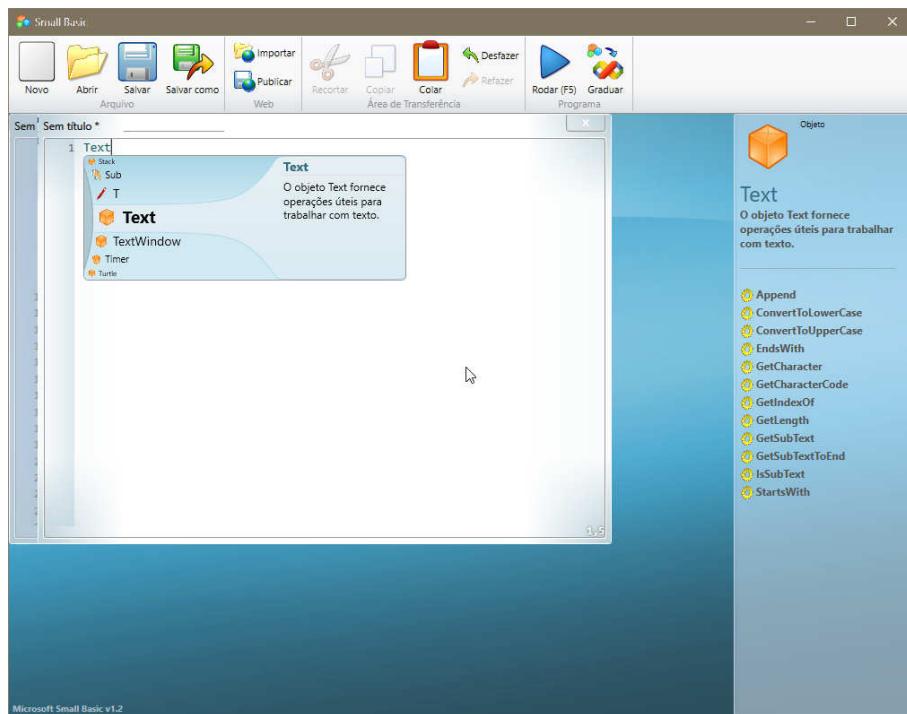


Figura 1.12 - Tela com início da escrita do comando "TextWindow.Clear()"

Veja que abaixo da escrita da palavra "**Text**" ocorre a apresentação no editor de texto de um menu rolante com a indicação do termo escrito. Ao lado direito na área de descrição ocorre a apresentação das informações do que é o objeto "**Text**" e a apresentação de suas operações. Termine de escrever a palavra "**TextWindow**" e observe a apresentação na área de descrição da lista de propriedades e operações como indica a figura 1.13.

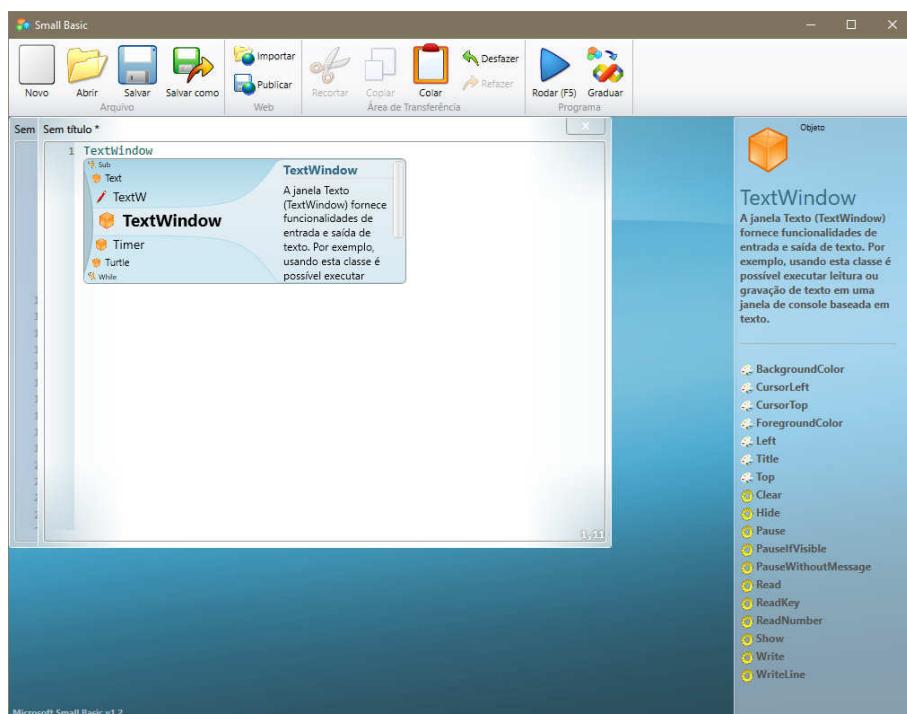


Figura 1.13 - Tela com início da escrita do comando "TextWindow.Clear()"

As *propriedades* do objeto "**TextWindow**" são identificadas com o ícone de um godê e as *operações* são identificadas com o ícone da uma engrenagem como apresentada na figura 1.14.



Figura 1.14 - Ícones de identificação de recursos

Veja na figura 1.15 a apresentação do significado da operação "**Clear**". Esta é a forma típica de apresentação de informações executada pelo ambiente "*Microsoft Small Basic*".

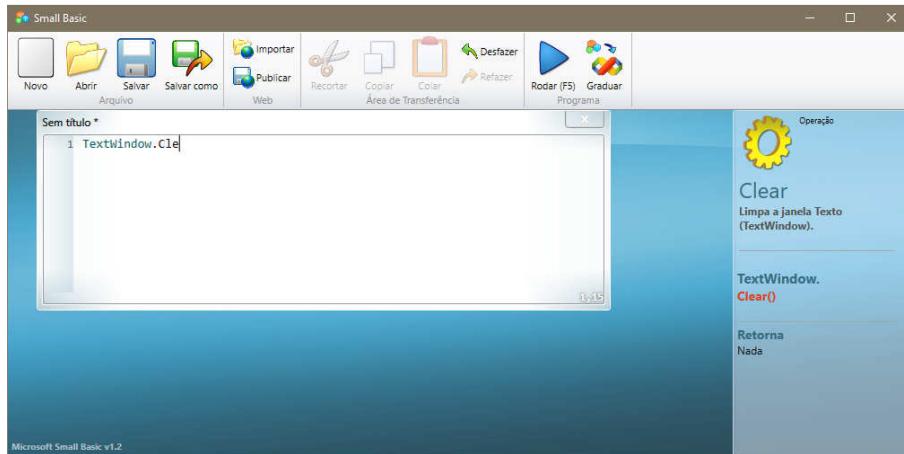


Figura 1.15 - Indicação da ação da operação "Clear"

Então mantenha sempre a atenção nos elementos informativos apresentados pelo ambiente. Isto ajudará você a entender melhor os recursos oferecidos e disponibilizados pelo ambiente.

ANOTAÇÕES

CAPÍTULO 2

Ações iniciais

A ação de programar computadores baseia-se em três ações operacionais básicas: a entrada, o processamento e a saída. Essas três ações podem ser realizadas de forma direta com intervenção do usuário do programa ou indireta com quase toda a ação sendo controlada pelo próprio programa. Neste capítulo fazemos uma introdução inicial há alguns recursos mínimos para o desenvolvimento de aplicações baseadas no foco da geometria da tartaruga (*turtle graphics*).

2.1 - Origem da tartaruga e o universo Small Basic

O conceito de programação baseado no uso de tartaruga vem de uma época, década de 1960, em que o Professor, Filósofo, Matemático e Pesquisador Seymour Papert que trabalhava com Jean Piaget dirigiu uma equipe multidisciplinar de estudo em *Inteligência Artificial* no desenvolvimento de uma linguagem de programação educacional inicialmente voltada para crianças. Essa linguagem inicial comandava um robô tipo tartaruga que desenhava figuras geométricas sobre uma folha de papel. A linguagem "Logo" foi desenvolvida com coautoria de Cynthia Solomon no *Massachusetts Institute of Technology* (MIT) com a participação direta do Professor Marvin Minsky e participação indireta de Daniel Bobrow e Wally Feurzeig.

"Logo" é uma linguagem para programação de computadores que ficou muito famosa, mas ser conhecida como uma linguagem para crianças também trouxe inconvenientes. Há quem ache que Logo ou qualquer coisa associada a "Logo" é ferramenta lúdica para ensinar apenas crianças, quando pessoas de outras idades podem se beneficiar dessa proposta.

A Microsoft ao lançar a linguagem "Small Basic" demonstrou entender os benefícios propostos pela de uma linguagem de programação educacional como "Logo" não só para crianças e fez a portabilidade do núcleo de operações do robô tartaruga para sua linguagem. Foi feliz ao criar uma ferramenta de trabalho educacional que se assemelha a estrutura de outras de suas linguagens como: Visual Basic, C#, F# e Visual C++ permitindo que um iniciante em programação, de qualquer idade, inicie sua jornada com uma linguagem simples, mas que já apresenta recursos operacionais que mais para a frente serão essenciais dando-lhe condições mentais de ir se familiarizando com certos detalhes.

O suporte ao efeito da geometria da tartaruga em "Small Basic" é obtido com o objeto especial chamado **Turtle**. Muito cuidado com o termo "objeto", pois é uma palavra em voga no universo da programação de computadores. É importante não confundir a linguagem "Small Basic" com linguagens orientadas a objeto como "Visual Basic". Para que uma linguagem seja considerada orientada a objetos ela deve deixar o programador criar seus próprios objetos. "Small Basic" faz uso de objetos internos, que estão prontos dentro de seu *kernel*, então apesar de "Small Basic" parecer-se com uma linguagem orientada a objetos não o é, nem de fato e nem de direito.

É importante entender que a ideia de *objeto* em "Small Basic" parece-se com a ideias do mesmo termo na *Programação Orientada A Objetos* e é por isso que se deve ter o cuidado de não fazer a confusão conceitual. Um "objeto" pode ser entendido como algo material percebido pelos sentidos ou algo mental (ou físico) que converge o pensamento, o sentimento ou ação. A Programação Orientada a Objetos baseia-se na ideia do material por isso dá a possibilidade de ser

criar os objetos, já "*Small Basic*" fica na esfera mental por disponibilizar os objetos que possui para uso, para ser percebido (sentido) e para ver sua ação.

O objeto **Turtle** é um recurso que fornece a linguagem "*Small Basic*" recursos que a aproximam da linguagem "*Logo*" no que tange a geometria da tartaruga, mas não nos demais recursos de "*Logo*". Este objeto é uma espécie de biblioteca ou módulo estático que possui um conjunto de funcionalidades para manipular desenhos.

2.2 - Objetos *Small Basic*

Um objeto é um componente que agrupa em seu interior um conjunto de recursos caracterizados por *propriedades*, *operações* e *eventos*. Não necessariamente um objeto terá as três categorias de recursos. Há objeto que possui apenas um conjunto de *operações*, há objeto que possui apenas um conjunto de *propriedades*, há objeto que possui um conjunto de *propriedades*, *eventos* e *operações* e há objeto que possui um conjunto de *propriedades* e *operações*.

Cada um desses recursos (*propriedades*, *operações* e *eventos*) estabelece alguma característica comportamental ao objeto, a saber:

- ❖ Uma *propriedade* é um recurso que define algum valor que altera o comportamento do objeto;
- ❖ Uma *operação* é um recurso que estabelece certo comportamento ao objeto;
- ❖ Um *evento* é um recurso que ocorre quando certa ação é executada sobre o objeto.

O objeto **Turtle** é composto pelas propriedades (*Speed*, *Angle*, *X* e *Y*) e pelas operações (*Show*, *Hide*, *PenDown*, *PenUp*, *Move*, *MoveTo*, *Turn*, *TurnRight* e *TurnLeft*) que serão apresentadas ao longo deste trabalho. Veja na tabela 2.1 a descrição resumida de cada um desses recursos para começar a ter uma ideia do que pode ser efetivamente realizado.

Além do objeto **Turtle** são existentes outros objetos em "*Small Basic*", dos quais alguns objetos são vistos neste trabalho, mas não todos eles. O principal objetivo desta obra é manter ao máximo o foco sobre a ótica da geometria da tartaruga, mas há no sítio oficial da linguagem um bom conjunto de atividades de programação que poderá ser executada para a ampliação de conhecimento.

É importante ressaltar para aqueles que já conhecem a linguagem "*Logo*" que **Turtle** não possui todos os recursos existentes no módulo da geometria da tartaruga. Este objeto possui os recursos essenciais que são suficientes para simular em "*Small Basic*" grande parte do efeito "*Logo*" atendendo o quesito de se ter um instrumento para auxiliar de maneira lúdica a aprendizagem de programação e lógica para iniciantes. Isso significa dizer que o robô tartaruga do "*Small Basic*" é um elemento virtual de operação, ou seja, só é implementado em uma tela gráfica que mostra o seu plano de operação, diferentemente de algumas implementações "*Logo*" que possibilitem além do modo virtual fazer uso do modo real com um robô tartaruga sendo controlado fisicamente sobre folhas de papel.

A partir dessa visão conceitual podemos passar a nossa primeira experiência artística. Abra o ambiente de programação "*Small Basic*" como orientado e escreva dentro da janela do editor de textos o código de um pequeno programa que tem por finalidade desenhar um quadrado no plano de ação da tartaruga, prestando muita atenção no que é escrito para evitar erros de sintaxe, pois esse tipo de erro faz com que o programa não seja executado e alguma mensagem de erro será apresentada na área de superfície do ambiente.

OPERAÇÃO	TIPO	RESULTADO
Angle	Propriedade	Retorna e define o ângulo de posicionamento da tartaruga
Speed	Propriedade	Define a velocidade pela qual a tartaruga se movimenta no plano
X	Propriedade	Retorna e define a posição cartesiana da tartaruga em X
Y	Propriedade	Retorna e define a posição cartesiana da tartaruga em Y
Show	Operação	Apresenta a tartaruga no plano
Hide	Operação	Esconde a tartaruga
PenDown	Operação	Abaixa a caneta para traçar desenho
PenUp	Operação	Levanta a caneta
Move	Operação	Movimenta a tartaruga no plano uma distância especificada
MoveTo	Operação	Movimenta a tartaruga para determinada coordenada do plano
Turn	Operação	Gira a tartaruga certo valor de ângulo especificado
TurnRight	Operação	Gira a tartaruga 90 graus a direita
TurnLeft	Operação	Gira a tartaruga 90 graus a esquerda

Tabela 2.1 - Operações e propriedade "Turtle"

Escreva apenas as linhas de instrução em azul do código a seguir. Não escreva os números em preto, pois estes são apresentados automaticamente na janela do editor de texto. A explicação de cada instrução é descrita após o código:

```

1 | Turtle.Move(80)
2 | Turtle.TurnRight()
3 | Turtle.Move(80)
4 | Turtle.TurnRight()
5 | Turtle.Move(80)
6 | Turtle.TurnRight()
7 | Turtle.Move(80)
8 | Turtle.TurnRight()

```

Antes de executar o programa vamos proceder sua gravação. Desta forma, acione na barra de ferramentas o botão "**Salvar**" e entre o nome do projeto como "**Cap02Ex01**" (capítulo dois, exemplo um).

Agora vamos colocá-lo em execução. Assim sendo, acione o botão "**Rodar**" da barra de ferramentas. Se tudo estiver escrito corretamente o programa entra em operação em alguns segundos e para finalizá-lo é necessário acionar o botão vermelho "X" da barra de título da janela gráfica ou acionar o atalho "<Alt>+<F4>". Veja o resultado na figura 2.1.

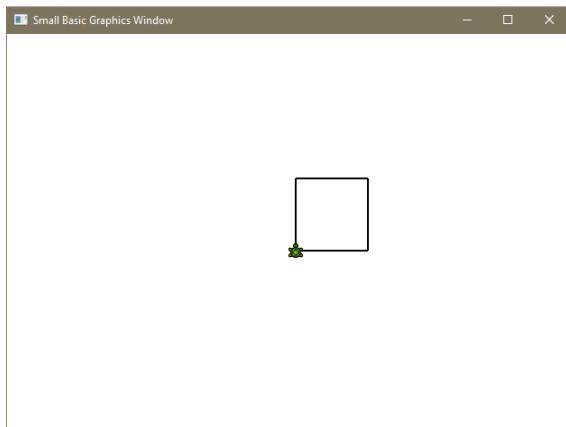


Figura 2.1 - Resultado da ação do programa "Cap02Ex01"

Veja que ao ser apresentado o desenho do quadrado, temos a impressão de ver a tartaruga se deslocando "vagarosamente" sobre seu plano de ação. Essa velocidade de deslocamento pode se tornar insatisfatória e deverá ser alterada. Em momento oportuno voltaremos a esta questão.

Observe que o programa "**Cap02Ex01**" está composto por um conjunto de oito instruções definidas em duas categorias comportamentais, sendo: instruções simples nas linhas pares e instruções com parâmetros nas linhas impares. As instruções simples são formadas pela definição de um objeto e uma operação com parênteses vazios e as instruções com parâmetros são formadas pela definição de um objeto e uma operação com um valor determinado dentro dos parênteses.

As instruções simples são iterativas e executam suas ações sem nenhuma interferência do programa. Já as instruções com parâmetros são interativas pois aceitam a definição de algum valor de apoio para que elas atuem dentro do programa.

A operação **TurnRight()** faz com que a tartaruga vire a direita 90 graus e a operação **Move(n)** faz sua movimentação certo número de passos identificado no parâmetro "**n**", que pode receber valores numéricos decimais positivos ou negativos de "**0**" até confortavelmente "**230**" em sua janela padrão. Valores de distanciamento maior podem ser estabelecidos mas isso dependerá de se fazer o ajuste do tamanho da janela que mostra o plano de ação da tartaruga. Para a maior parte dos exemplos deste livro são consideradas a faixa de valores indicada. Os números positivos de **Move(n)** avançam a tartaruga para a frente e os números negativos de **Move(-n)** avançam a tartaruga para trás.

Veja que o indicativo "**n**" ou "**-n**" representam a definição de um parâmetro, o qual determina a distância a ser percorrida pela tartaruga. O valor da distância efetivamente fornecido no programa foi "**80**" para **Move(n)**, sendo este valor chamado de argumento. Desta forma, temos aqui dois conceitos importantes em programação que são: parâmetros (valor esperado) e argumentos (valor passado).

A operação **TurnRight()** sempre faz um giro de 90 graus a direita e sua operação irmã **TurnLeft()** faz o giro 90 graus sempre a esquerda. Mas e se houver o desejo de operar uso de graus diferentes? Para este caso devemos usar a operação **Turn()** que aceita o uso de valores numéricos decimais positivos ou negativos entre "**0**" e "**360**". Valores positivos fazem o giro ocorrer a direita e valores negativos fazem o giro ocorrer a esquerda.

Veja na sequência um programa que apresenta a figura de um triângulo equilátero desenhado na direção esquerda da tela.

Para desenhar um triângulo é necessário saber de antemão o valor do grau de giro a ser usado. Para tanto, pegue um transferidor, de preferência redondo, aquele que apresenta a escala com 360 graus, o de 180 graus também serve, mas o de 360 é mais interessante. Veja na figura 2.2 os modelos de transferidores comentados.

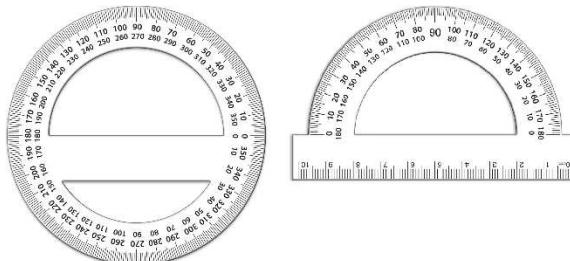


Figura 2.2 - Transferidores de 360 e 180 graus

Divida o valor "360" por "3" e teremos o resultado "120". Na sequência escreva o código seguinte:

- 1 `Turtle.Move(80)`
- 2 `Turtle.Turn(-120)`
- 3 `Turtle.Move(80)`
- 4 `Turtle.Turn(-120)`
- 5 `Turtle.Move(80)`
- 6 `Turtle.Turn(-120)`

Grave o projeto com o nome "**Cap02Ex02**" e coloque-o em execução como orientado. Veja a apresentação da imagem indicada na figura 2.3.

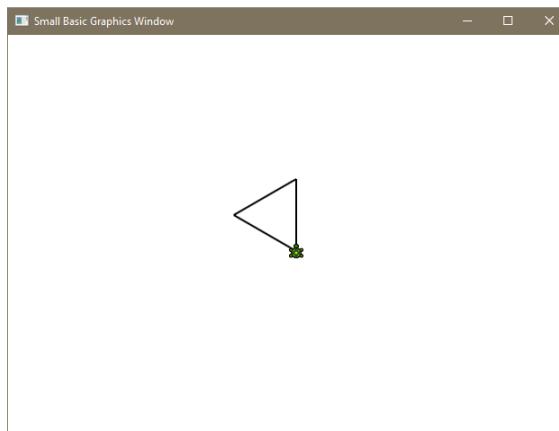


Figura 2.2 - Resultado da ação do programa "Cap02Ex02"

Veja que neste projeto todas as instruções são baseadas no uso de operações com parâmetros. Em especial a operação **Turn(n)** está fazendo uso de valores negativos para produzir seu giro a esquerda. Mas o que pode causar estranheza a alguns é o uso do valor de graus 120 para definir o giro para criação de um triângulo equilátero.

Você deve ter visto nas aulas de Geometria na disciplina de Matemática que um *triângulo equilátero* é um tipo de triângulo que possui seus lados congruentes, ou seja, possuem a mesma medida. Talvez você tenha fixado, nessas aulas, que um triângulo equilátero possui seus ângulos de 60 graus e esteja agora *bugando* sua mente e se perguntando "de onde veio esse 120?". É simples, se você pensou em 60 não está erado, pois 60 são os ângulos internos do triângulo, mas 120 são seus ângulos externos. O desenho produzido pela geometria da tartaruga baseia-se no uso dos ângulos externos. Por isso 120. Veja a figura 2.3.

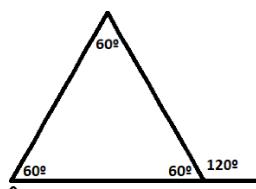


Figura 2.3 - Ângulos internos (60) e externo (120) de um triângulo equilátero

Note que a sobreposição da tartaruga sobre a figura desenhada pode atrapalhar um pouco sua visualização. Neste sentido, é possível pedir que a tartaruga seja ocultada a partir do uso da instrução:

`Turtle.Hide()`

Veja o código seguinte que após desenhar um traço oculta a apresentação da tartaruga:

```
1 | Turtle.Move(80)
2 | Turtle.Hide()
```

Grave o projeto com o nome "**Cap02Ex03**" e coloque-o em execução como orientado. Veja a apresentação do traço desenhado pela tartaruga e em seguida seu ocultamento. A figura 2.4 indica o resultado da ação.



Figura 2.4 - Ocultamento da tartaruga

Para retornar a apresentação da tartaruga em uma figura use a instrução:

`Turtle.Show()`

Quando a tartaruga anda, ela por padrão desenha, pois carrega com ela uma caneta posicionada sobre o plano de trabalho. Mas nem sempre se deseja que a tartaruga ande desenhando. Por vezes é interessante que a tartaruga ande sem desenhar.

Para andar sem desenhar, ou seja, com a caneta erguida é necessário antes do movimento pedir a execução da instrução:

`Turtle.PenUp()`

Para voltar a desenhar basta usar a instrução:

`Turtle.PenDown()`

A fim de demonstrar o uso desses recursos considere o programa a seguir:

```

1 | Turtle.Move(40)
2 | Turtle.Turn(-90)
3 | Turtle.PenUp()
4 | Turtle.Move(-40)
5 | Turtle.Turn(90)
6 | Turtle.PenDown()
7 | Turtle.Move(40)

```

Grave o projeto com o nome "**Cap02Ex04**" e coloque-o em execução como orientado. Veja a apresentação da imagem indicada na figura 2.5.



Figura 2.5 - Resultado da ação do programa "Cap02Ex04"

Após executar as instruções anteriores temos na figura 2.5 a imagem de duas linhas verticais deslocadas.

2.3 - Programação sequencial

Vamos agora deixar um pouco de lado o desenvolvimento de desenhos e vamos pensar em uma programação com interatividade com o usuário. Já é sabido que um programa de computador executa as ações de entrada, processamento e saída. Para programarmos efetivamente um computador é necessário quando efetuamos a entrada de dados, fazer o armazenamento desses dados em memória. O armazenamento desses dados ocorre dentro de uma estrutura (área de memória) denominada **variável**.

Para facilitar a operação as variáveis são componentes que podem ser livremente nomeadas desde que sigamos algumas regras fundamentais:

1. O nome de uma variável deve ser iniciado por letra;
2. O nome de uma variável pode a partir do segundo caractere conter números/sublinhados;
3. O nome de uma variável não pode colidir com o nome dos recursos da ação do programa;
4. Para facilitar os nomes devem ser significativos ao seu conteúdo.

Para teste considere um programa simples que solicite o nome de quem está usando o programa e apresente na sequência uma mensagem de saudação com o nome informado. Veja o código com a indicação do uso da variável "**Nome**" na cor vermelha:

```

1 | TextWindow.Clear()
2 | TextWindow.WriteLine("Olá, sou o Small Basic.")
3 | TextWindow.Write("Qual é o seu nome? ")
4 | Nome = TextWindow.Read()

```

```
5 | TextWindow.WriteLine("")  
6 | TextWindow.WriteLine("Olá " + Nome + ", tenha excelente aprendizagem.")
```

Grave o projeto com o nome "**Cap02Ex05**" e coloque-o em execução como orientado. Se tudo estiver escrito corretamente o programa entra em operação. Entre o nome solicitado e veja a resposta apresentada pelo programa na figura 2.6.

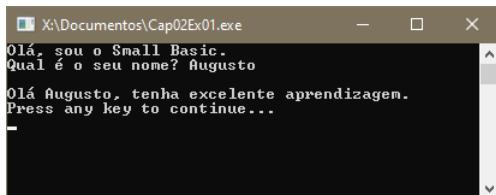


Figura 2.6 - Resultado da ação do programa

Vamos entender o que aconteceu no programa linha a linha. Na linha "1" ocorre a limpeza da tela de texto a ser usada com a operação "**Clear()**" do objeto "**TextWindow**", na linha "2" é feita a apresentação da primeira mensagem de saudação com um salto de linha ao final a partir da operação "**WriteLine**", na linha "3" é feita a apresentação da segunda mensagem sem que haja o salto de linha final com a operação "**Write**", na linha "4" é colocada a variável "**Nome**" com a operação "**Read()**" do objeto "**TextWindow**" que tem por finalidade executar a leitura de um dados alfanumérico informado no teclado, na linha "5" faz-se a apresentação de uma linha em branco e na linha "6" faz-se a operação de apresentação de uma mensagem de saudação com o nome informado. Veja que na linha "6" é usada uma ação de processamento, muito sutil, chamada *concatenação*. Concatenação é o efeito de junção de partes separadas como uma só parte, sendo isso executado em "*Small Basic*" por meio do símbolo "+".

Agora que conhecemos como fazer uso de variáveis, já é possível pensar em um programa que efetue a leitura de dois valores numéricos desconhecidos e apresente o resultado da soma desses valores. Assim sendo, considere o seguinte código:

```
1 | TextWindow.Clear()  
2 | TextWindow.Write("Entre o 1º valor numérico: ")  
3 | Valor1 = TextWindow.ReadNumber()  
4 | TextWindow.Write("Entre o 2º valor numérico: ")  
5 | Valor2 = TextWindow.ReadNumber()  
6 | Resultado = Valor1 + Valor2  
7 | TextWindow.WriteLine("")  
8 | TextWindow.WriteLine("Soma = " + Resultado)
```

Grave o programa com o nome "**Cap02Ex06**" e coloque-o em execução. Depois entre dois valores numéricos e veja o resultado da soma apresentado. Veja que os dois programas apresentados realizam as ações de entrada de dados, processamento desses dados e a saída da informação a partir do dado informado. A operação "**ReadNumber()**" é ideal para a recepção da entrada de dados numéricos, pois se algum dado não numérico for fornecido este recurso zera a entrada evitando que ocorra apresentação de saídas confusas. Para ver o que estamos falando troque as operações de entrada de "**ReadNumber()**" para "**Read**" e observe que ao invés de fazer o cálculo o programa faz uma ação de concatenação.

Observe que o programa "**Cap02Ex02**" a partir da entrada numérica dos valores estabeleceu uma linha de processamento com operação de adição. Veja outras operações, na tabela 2.2, que podemos obter com a linhagem.

SÍMBOLO	OPERAÇÃO	RESULTADO
+	Adição	Soma
-	Subtração	Diferença
*	Multiplicação	Produto
/	Divisão	Quociente

Tabela 4.2 - Operadores aritméticos

Observe o código seguinte que mostra os resultados das quatro operações aritméticas sobre os números "3" e "2".

```

1 | TextWindow.Clear()
2 | TextWindow.WriteLine(3 + 2) ' mostra a soma
3 | TextWindow.WriteLine(3 - 2) ' mostra a diferença
4 | TextWindow.WriteLine(3 * 2) ' mostra o produto
5 | TextWindow.WriteLine(3 / 2) ' mostra o quociente

```

Grave o programa com o nome "**Cap02Ex07**" e coloque-o em execução e veja os resultados apresentados. O trecho de código grafado com o símbolo "**aspas simples**" indica que a linha a sua frente é um comentário de código que serve para documentar o código do programa, mas não é executado pela linguagem. Veja os resultados apresentado junto a figura 2.7

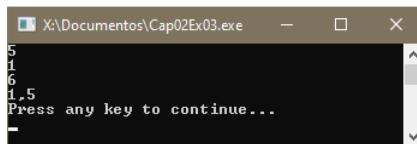


Figura 2.7 - Resultado de operações aritméticas

No entanto, o conjunto dos quatro operadores aritméticos consegue atender apenas a uma parte muito básica e limitada do processamento matemático. Para outras operações matemáticas é necessário o uso do objeto "**Math**" que apresenta uma série maior de operações.

No objeto "**Math**" encontra-se um conjunto com vinte operações diversas e uma propriedade para o retorno do valor da constante "**π**". A tabela 4.3 mostra esses.

OPERAÇÃO	TIPO	RESULTADO
Pi	Propriedade	3,14159265358979
Abs	Operação	Retorna o positivo de um número
ArcCos	Operação	Retorna o ângulo em radianos do cosseno de um número
ArcSin	Operação	Retorna o ângulo em radianos do seno de um número
ArcTan	Operação	Retorna o ângulo em radianos da tangente de um número
Ceiling	Operação	Retorna inteiro maior ou igual do número decimal

OPERAÇÃO	TIPO	RESULTADO
Cos	Operação	Retorna o cosseno do ângulo em radianos de um número
Floor	Operação	Retorna inteiro menor ou igual do número decimal
GetDegrees	Operação	Converte radianos em graus
GetRadians	Operação	Converte graus em radianos
GetRandomNumber	Operação	Retorna número aleatório entre 1 e o número especificado
Log	Operação	Retorna logaritmo na base 10 de um número
Max	Operação	Compara dois números e retorna o maior
Min	Operação	Compara dois números e retorna o menor
NaturalLog	Operação	Retorna logaritmo natural de um número
Power	Operação	Retorna a potência de uma base sobre um expoente
Remainder	Operação	Retorna o resto de uma divisão
Round	Operação	Arredonda número para inteiro mais próximo
Sin	Operação	Retorna o seno do ângulo em radianos de um número
SquareRoot	Operação	Retorna a raiz quadrada de um número
Tan	Operação	Retorna a tangente do ângulo em radianos de um número

Tabela 4.3 - Operações e propriedade "Math"

Observe o código seguinte com algumas demonstrações de cálculos efetuados com os recursos disponibilizados pelo objeto "Math".

```

1 TextWindow.Clear()
2 TextWindow.WriteLine(Math.Pi)           ' 3,14159265358979
3 TextWindow.WriteLine(Math.Abs(-1))      ' 1
4 TextWindow.WriteLine(Math.ArcCos(1))    ' 0
5 TextWindow.WriteLine(Math.ArcSin(1))    ' 1,5707963267949
6 TextWindow.WriteLine(Math.ArcTan(1))    ' 0,785398163397448
7 TextWindow.WriteLine(Math.Ceiling(1.7)) ' 2
8 TextWindow.WriteLine(Math.Cos(1))       ' 0,54030230586814
9 TextWindow.WriteLine(Math.Floor(1.7))   ' 1
10 TextWindow.WriteLine(Math.GetRandomNumber(5)) ' um número entre 1 e 5
11 TextWindow.WriteLine(Math.Log(2))        ' 0,301029995663981
12 TextWindow.WriteLine(Math.Max(2,3))     ' 3
13 TextWindow.WriteLine(Math.Min(2,3))     ' 2

```

```
14 | TextWindow.WriteLine(Math.NaturalLog(2))      ' 0,693147180559945
15 | TextWindow.WriteLine(Math.Power(2,3))          ' 8
15 | TextWindow.WriteLine(Math.Remainder(5,3))       ' 2
17 | TextWindow.WriteLine(Math.Sin(1))              ' 0,841470984807896
18 | TextWindow.WriteLine(Math.SquareRoot(25))       ' 5
19 | TextWindow.WriteLine(Math.Tan(1))              ' 1,5574077246549
```

Grave o programa com o nome "**Cap02Ex08**" e coloque-o em execução e veja os resultados apresentados.

A partir das instruções apresentadas temos em mãos algumas ferramentas básicas para o desenvolvimento de algumas operações interessantes de programação, além da possibilidade de poder desenhar diversas formas geométricas básicas.

2.4 - Exercícios de fixação

1. Desenvolver programa 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. Dica: considere usar "**Math.Floor()**".
2. Desenvolver programa 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.
3. Desenvolver programa 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.
4. Desenvolver programa 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.
5. Desenvolver programa que efetue a leitura de dois valores numéricos (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. Desenvolver programa que efetue a leitura de um valor numérico qualquer e apresente como resultado os valores sucessor e antecessor do valor numérico fornecido. Não armazene em memória os valores calculados.
7. Desenvolver programa que efetue a leitura de qualquer valor para as variáveis "A" e "B", efetue a troca dos valores de modo que no final a variável "A" tenha o valor da variável "B" e a variável "B" tenha o valor da variável "A". Apresentar os novos valores de cada uma das variáveis.

ANOTAÇÕES

CAPÍTULO 3

Ações especializadas

As operações do universo da *geometria da tartaruga* vão além dos recursos já 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 feito com uso das operações **Turtle.Move(n)** e **Turn(n)**, além do uso das operações **TurnRight()** ou **TurnLeft()** do objeto **Turtle** ao invés de **Turn(n)** repetindo-se quatro vezes as combinações de instruções necessárias. No entanto, a linguagem "Small Basic" possui um comando representado pelas palavras-chave **For/To/Step/EndFor** que dá conta muito bem deste recado. Veja a estrutura sintática do comando:

```
For <variável> = <valor inicial> To <valor final> [Step <incremento>]  
    <instruções e comandos a serem repetidos>  
EndFor
```

Onde, "<variável>" é a indicação de uma variável usada para contar as repetições. "<valor inicial>" é o estabelecimento do valor inicial da contagem, "<valor final>" é o estabelecimento do valor final da contagem e "<incremento>" é a definição do salto de contagem entre os valores inicial e final.

O conjunto de palavras-chave **For/To/Step/EndFor** é usado para estabelecer uma instrução de repetição de recursos entre o comando **For** (para) e o comando **EndFor** (fim para). O trecho sinalizado com o comando **To** (até) estabelece a contagem entre as definições de início e fim. O trecho com o comando **Step** é opcional quando a contagem possui incremento unitário (de 1 em 1) por isso estar grafado entre parênteses. O conteúdo grafado entre os símbolos "<" e ">" é considerado obrigatório.

Veja em seguida o programa para desenhar um quadrado a partir do uso da instrução de repetição **For/To/Step/EndFor**:

```
1 | For I = 1 to 4 Step 1  
2 |     Turtle.Move(80)  
3 |     Turtle.TurnRight()  
4 | EndFor  
5 | Turtle.Hide()
```

Grave o programa com o nome "**Cap03Ex01**", coloque-o em execução e veja na figura 3.1 seu resultado de operação.

Observe que entre os comandos **For** e **EndFor** está sendo definida a escrita do código com um deslocamento de dois espaços em branco. Este deslocamento chama-se *endentaçāo* e serve para nos mostrar de forma simplificada o que está dentro de certa operação, ou seja, um bloco

de ação. Apesar deste estilo não ser necessário para o programa, é para nós humanos. Isso facilita a nossa leitura e é uma prática utilizada profissionalmente em todo o mundo.

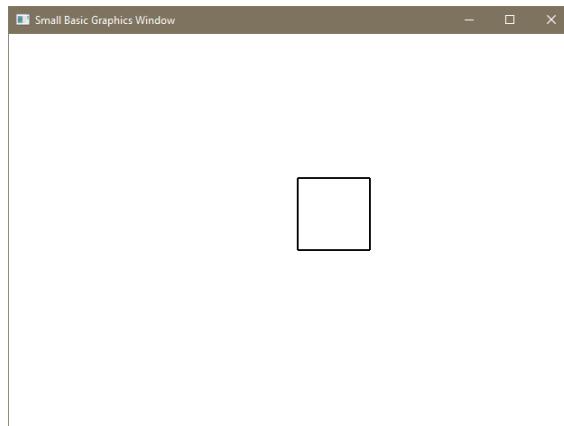


Figura 3.1 - Resultado da ação do programa "Cap03Ex01"

Pensando no triângulo equilátero veja o código seguinte com a definição de mudança de velocidade no traço do desenho da tartaruga a partir do uso propriedade **Speed** do objeto **Turtle**:

```
1 | Turtle.Speed = 8
2 | For I = 1 to 3
3 |   Turtle.Turn(120)
4 |   Turtle.Move(80)
5 | EndFor
```

Grave o programa com o nome "**Cap03Ex02**" e coloque-o em execução. Perceba que a tartaruga agora se locomove mais rapidamente no plano. Veja na figura 3.2 seu resultado de operação.

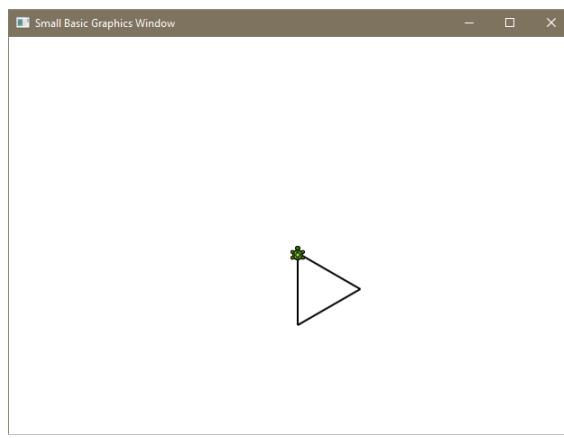


Figura 3.2 - Resultado da ação do programa "Cap03Ex02"

Veja que a tartaruga se locomoveu mais rápido com o uso do valor "8" associado a propriedade **Speed**. Perceba que uma propriedade é diferente de uma operação. A operação segue o uso de uma palavra anexada aa um conjunto de parênteses que pode ou não possuir a definição de certo parâmetro. Já a propriedade é definida com a atribuição direta de certo valor.

A propriedade **Speed** aceita valores entre 1 e 10, sendo 1 a velocidade padrão (mais lento) e 10 a velocidade mais rápida com o desenho sendo produzido o mais rápido possível. A velocidade mais lenta possibilita ver efetivamente o que está acontecendo durante a execução do programa.

Agora vejamos o desenho de um pentágono. Assim sendo, considere o programa seguinte:

```

1 | Turtle.Speed = 8
2 | For I = 1 to 5
3 |   Turtle.Move(80)
4 |   Turtle.Turn(72)
5 | EndFor

```

Grave o programa com o nome "**Cap03Ex03**", coloque-o em execução e observe o resultado indicado junto a figura 3.3.

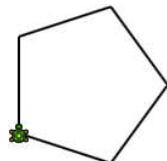


Figura 3.3 - Pentágono.

A partir dos desenhos de um quadrado, um triângulo equilátero e um pentágono pode estar em sua mente a pergunta "como desenhar uma circunferência?". Bom, a regra é a mesma comentada no capítulo anterior. Basta dividir o valor 360 graus pela quantidade de ângulos desejada. Lembre-se que um transferidor redondo nos mostra que uma circunferência possui 360 graus. Isto posto, significa dizer que basta fazer a divisão e usar então o valor "**1**" para a operação **Turn(n)**. Veja o programa a seguir:

```

1 | Turtle.Speed = 10
2 | For I = 1 to 360
3 |   Turtle.Move(1)
4 |   Turtle.Turn(1)
5 | EndFor

```

Grave o programa com o nome "**Cap03Ex04**", coloque-o em execução e veja o resultado apresentado junto a figura 3.4.



Figura 3.4 - Circunferência.

Veja a partir de uma circunferência a criação de outras circunferências com o uso de duas operações de repetição com o código seguinte:

```

1 | Turtle.Speed = 10
2 | For I = 1 to 6
3 |   For J = 1 To 360
4 |     Turtle.Move(1)
5 |     Turtle.Turn(1)
6 |   EndFor
7 |   Turtle.Turn(60)
8 | EndFor

```

Grave o programa com o nome "**Cap03Ex05**", coloque-o em execução e veja o resultado apresentado junto a figura 3.5.

Observe a definição no programa "**Cap03Ex05**" de uma repetição (vermelho) definida dentro de outra repetição (azul). Este efeito é conhecido como *encadeamento* de repetições.

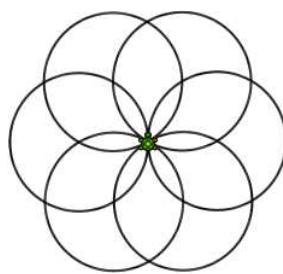


Figura 3.5 - Junção de circunferências.

Neste modelo a repetição em vermelho espera que a repetição em azul cumpra todo seu ciclo para que ela de continuidade a próxima ação. Isto significa que para a repetição em vermelho cumprir seus 6 ciclos aguarda antes a repetição em azul executar seus 360 passos. Observe que para se conseguir o efeito de circunferências sobrepostas usa-se a repetição em vermelho com 6 repetições com um desvio de grau a direita de 60, que se multiplicado 6 x 60 gera o resultado 360 que permite o efeito entrelaçado dentro de uma circunferência.

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
TRIDECA GONO	13	360 / 13	27.69
TETRADECÁGONO	14	360 / 14	25.71
PENTADECÁGONO	15	360 / 15	24.00

POLÍGONO	LADOS	360 / LADOS	ÂNGULO
HEXADECÁGONO	16	360 / 16	22.50
HEPTADECÁGONO	17	360 / 17	21.18
OCTODECÁGONO	18	360 / 18	20.00
ENEADECÁGONO	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 - Sub-rotinas

O programa "**Cap03Ex05**" pode ser simplificado com o uso de uma técnica de programação que visa dividir um programa em sub-rotinas. Cada sub-rotina pode executar sua ação e ser usado por outras sub-rotina para auxiliar sua ação. Neste caso o programa "**Cap03Ex05**" pode ser recomposto em duas partes: uma sub-rotina que desenha a circunferência e outra parte do programa que executa a chamada da sub-rotina, como sendo a rotina principal.

O uso da técnica de programação baseada em sub-rotina nos permite duas considerações: primeira que podemos organizar melhor a estrutura do programa dividindo-o em partes conceituais sem deixar tudo misturado em uma mesma sequência de código; segundo pelo fato de fazermos o reaproveitamento de código, pois aquilo que está definido em uma sub-rotina não necessita ser reescrito se houver a necessidade de fazer uso mais de uma vez.

Pode até não parecer mas o programa "**Cap03Ex05**" faz com a repetição em vermelho a chamada de seis execuções da repetição em azul que é uma excelente candidata a se tornar uma

sub-rotina. Assim sendo, observe o código de programa a seguir que executa a mesma ação do programa "**Cap03Ex05**", só que agora de forma estruturada:

```
1 | Turtle.Speed = 10
2 |
3 | Sub Circunferencia
4 |   For I = 1 To 360
5 |     Turtle.Move(1)
6 |     Turtle.Turn(1)
7 |   EndFor
8 | EndSub
9 |
10| For I = 1 to 6
11|   Circunferencia()
12|   Turtle.Turn(60)
13| EndFor
```

Grave o programa com o nome "**Cap03Ex06**", coloque-o em execução e veja o resultado apresentado. Perceba que exatamente o mesmo indicado na figura 3.5.

Observe que a chamada da sub-rotina na linha de código "11" é feita com a indicação de nome seguido de parênteses ao estilo de uso das operações de um objeto que caracterizam instruções simples. Quanto ao uso de parâmetros em sub-rotina aguarde mais um pouco.

Cada sub-rotina criada dentro de um programa estabelece um grau de "conhecimento" que a linguagem pode fazer uso, dando-lhe um toque de *inteligência artificial*.

Veja agora no próximo código a definição de uma sub-rotina chamada "**Quadrado**" que desenha obviamente um quadrado de tamanho "80" com "90" graus a direita. O programa deverá desenhar a partir desse quadrado mais quatro outros com deslocamento de "90" graus a esquerda. Veja o código:

```
1 | Turtle.Speed = 10
2 |
3 | Sub Quadrado
4 |   For I = 1 To 4
5 |     Turtle.Move(80)
6 |     Turtle.Turn(90)
7 |   EndFor
8 | EndSub
9 |
10| For J = 1 To 4
11|   Quadrado()
12|   Turtle.Turn(-90)
13| EndFor
```

Grave o programa com o nome "**Cap03Ex07**", coloque-o em execução e veja o resultado apresentado junto a figura 3.6.

Veja que para realizar a apresentação do desenho proposto são definidos dois trechos de código. O primeiro trecho controlado pela variável "I" executa a ação da sub-rotina "**Quadrado**"

e o segundo trecho controlado pela variável "J" caracteriza-se por ser a parte principal do programa que faz a chamada da sub-rotina, neste caso quatro vezes.

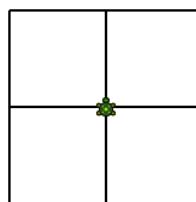


Figura 3.6 - Quadrado repetido.

Em outras linguagens de programação fazer o uso sequencial da mesma variável para definição de repetições não causa problemas, mas em "*Small Basic*" isso não é possível, pois a variável carrega dentro de repetição fica "presa" e não permite a sobreposição de valores após o encerramento de sua operação.

Dependendo do valor de ângulo é possível formar imagens geométricas muito diferentes como um conjunto de quadrados intercalados com "8" repetições com "45" graus a esquerda. Veja, então, o próximo código:

```

1 | Turtle.Speed = 10
2 |
3 | Sub Quadrado
4 |   For I = 1 To 4
5 |     Turtle.Move(80)
6 |     Turtle.Turn(90)
7 |   EndFor
8 | EndSub
9 |
10| For J = 1 To 8
11|   Quadrado()
12|   Turtle.Turn(-45)
13| EndFor

```

Grave o programa com o nome "**Cap03Ex08**", coloque-o em execução e veja o resultado apresentado junto a figura 3.7.

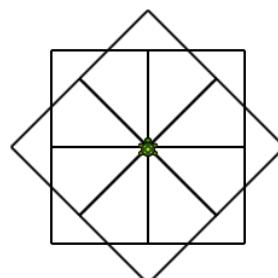


Figura 3.7 - Quadrados intercalados.

No próximo projeto é apresentada uma sub-rotina que desenha um pentágono a partir de um cálculo efetuado dentro do parâmetro da operação **Turn(n)**. Essa é outra maneira que podemos fazer uso dos recursos da linguagem, sendo totalmente válida. Essa ação abre outras possibilidades como será apresentado mais adiante.

Observe atentamente no código a seguir a indicação da parte sinalizada em vermelho:

```
1 | Turtle.Speed = 10
2 |
3 | Sub Pentagono
4 |   For I = 1 To 5
5 |     Turtle.Move(80)
6 |     Turtle.Turn(360 / 5)
7 |   EndFor
8 | EndSub
9 |
10| Pentagono()
```

Grave o programa com o nome "**Cap03Ex09**", coloque-o em execução e veja o resultado apresentado junto a figura 3.8.

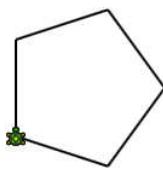


Figura 3.8 - Pentágono

A partir do conhecimento de como produzir um pentágono é possível criar outros efeitos. Observe a seguir o próximo código que mostra a sobreposição de 5 imagens de um pentágono, com efeito semelhante ao apresentado na figura 3.5:

```
1 | Turtle.Speed = 10
2 |
3 | Sub Pentagono
4 |   For I = 1 To 5
5 |     Turtle.Move(80)
6 |     Turtle.Turn(72)
7 |   EndFor
8 | EndSub
9 |
10| For J = 1 To 5
11|   Pentagono()
12|   Turtle.Turn(72)
13| EndFor
```

Grave o programa com o nome "**Cap03Ex10**", coloque-o em execução e veja o resultado apresentado junto a figura 3.9.

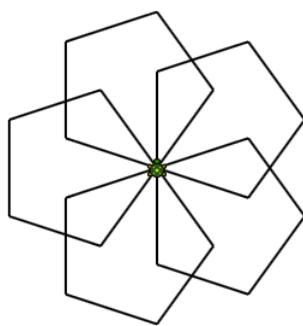


Figura 3.9 - Pentágono sobreposto

Veja agora o mpróximo código:

```

1 Turtle.Speed = 10
2
3 Sub Pentagono
4   For I = 1 To 5
5     Turtle.Move(80)
6     Turtle.Turn(72)
7   EndFor
8 EndSub
9
10 For J = 1 To 10
11   Pentagono()
12   Turtle.Turn(36)
13 EndFor

```

Grave o programa com o nome "**Cap03Ex11**", coloque-o em execução e veja o resultado apresentado junto a figura 3.10.

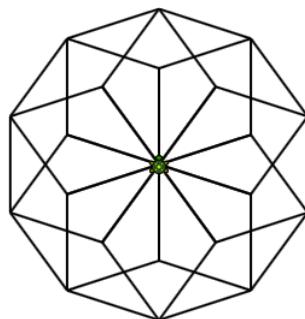


Figura 3.10 - Pentágono sobreposto maior

Veja a seguir um projeto em que uma sub-rotina é usada por outra sub-rotina:

```

1 Turtle.Speed = 10
2 Sub MeioCirc
3   For I = 1 To 90
4     Turtle.Move(1)
5     Turtle.Turn(1)
6   EndFor
7 EndSub
8
9 Sub Petala
10   MeioCirc()
11   Turtle.Turn(90)
12   MeioCirc()
13 EndSub
14
15 Sub Flor
16   For J = 1 To 8
17     Petala()
18     Turtle.Turn(45)
19   EndFor
20 EndSub
21
22 Flor()

```

Grave o programa com o nome "**Cap03Ex12**", coloque-o em execução e veja o resultado apresentado junto a figura 3.11.

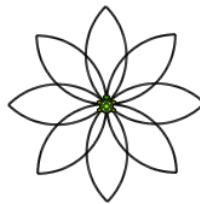


Figura 3.11 - Execução de sub-rotina que desenha flor.

A partir de um código semelhante ao anterior veja a apresentação de uma flor com vinte pétalas:

```
1 | Turtle.Speed = 10
2 | Sub MeioCirc
3 |   For I = 1 To 90
4 |     Turtle.Move(1)
5 |     Turtle.Turn(1)
6 |   EndFor
7 | EndSub
8 |
9 | Sub Petala
10|   MeioCirc()
11|   Turtle.Turn(90)
12|   MeioCirc()
13| EndSub
14|
15| Sub Flor
16|   For J = 1 To 20
17|     Petala()
18|     Turtle.Turn(36)
19|   EndFor
20| EndSub
21|
22| Flor()
```

Grave o programa com o nome "**Cap03Ex13**", coloque-o em execução e veja o resultado apresentado junto a figura 3.12.

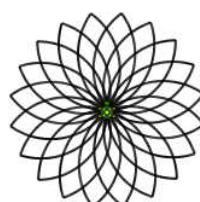


Figura 3.12 - Flor com vinte pétalas.

A partir da definição de sub-rotinas é possível diminuir muito a carga de trabalho, pois é possível criar comandos, ou seja, operações derivadas e especializadas em desenhar parte de uma imagem e formar, a partir daí, imagens com as combinações dos comandos definidos em conjunto com os demais recursos da linguagem.

3.3 - Simulação de passagem de parâmetro

A partir da visão já fornecida sobre definição e uso de variáveis é possível criar sub-rotinas que, por exemplo, desenhe um quadrado em que o tamanho da figura seja informado pelo usuário do programa.

Por ser a linguagem "*Small Basic*" simplificada para a aprendizagem ela não opera a passagem de parâmetro como linguagens de programação mais robustas. Neste caso, sua forma de tratamento baseia-se na definição de uma variável que receberá o valor desejado e processará a sub-rotina com o valor fornecido. Assim sendo, considere o código seguinte:

```

1 Turtle.Speed = 10
2
3 Sub Quadrado
4   For I = 1 To 4
5     Turtle.Move(Tamanho)
6     Turtle.Turn(90)
7   EndFor
8 EndSub
9
10 TextWindow.Clear()
11 TextWindow.Write("Informe o tamanho do quadrado: ")
12 Tamanho = TextWindow.ReadNumber()
13 Quadrado()

```

Grave o programa com o nome "**Cap03Ex14**", coloque-o em execução. Quando solicitado informe o tamanho do quadrado a ser desenhado, experimente o valor "80" e observe o resultado indicado na figura 3.13.

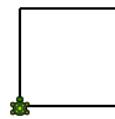


Figura 3.13 - Quadrado com tamanho definido pelo teclado

O código do programa "**Cap03Ex14**", bem como os outros códigos que operam neste capítulo com o uso de sub-rotinas foram produzidos a partir de uma técnica clássica de codificação chamada "*bottom-up*", ou seja, de baixo para cima. Neste caso, primeiro coloca-se a sub-rotina e depois as sub-rotinas que a chamam ou a rotina principal, sendo isto um modelo de codificação.

Existem linguagens de programação que obrigam que a programação seja efetivamente escrita de baixo para cima, mas a outras que não fazem esta questão como é o caso do "*Small Basic*".

É óbvio que em linguagens mais flexíveis tanto faz a forma de escrita, mas independentemente disso a pessoa que programa deve manter um comportamento coerente com as boas práticas de programação. É uma questão de educação e respeito com outros profissionais da área, e esta prática orienta o uso do modelo "*bottom-up*", principalmente quando o código deve atender aplicações em modo **CLI** (*Command-Line Interface*), ou seja, programas escritos em modo texto. No entanto, programas escritos para aplicações **GUI** (*Graphical User Interface*), ou seja, programas que usam o modo gráfico normalmente seguem o estilo de codificação "*top-down*".

Desta forma, o próximo código é escrito em estilo inverso ao modelo "*bottom-up*", ou seja, escrito no modelo "*top-down*" (de cima para baixo), sendo então um código de programa "*mal-educado*". O modelo "*top-down*" é usado na Computação com o objetivo de pensar o projeto

do sistema, mas não seu desenvolvimento. Observe que uma casa é construída usando o modelo "*bottom-up*", ou seja, do alicerce ao telado e não o inverso.

Veja o código seguinte:

```
1  Turtle.Speed = 10
2  TextWindow.Clear()
3  Tamanho = 40
4  Quadrado()
5  Tamanho = 60
6  Quadrado()
7  Tamanho = 80
8  Quadrado()
9  Tamanho = 90
10 Quadrado()
11
12 Sub Quadrado
13   For I = 1 To 4
14     Turtle.Move(Tamanho)
15     Turtle.Turn(90)
16   EndFor
17 EndSub
```

Grave o programa com o nome "**Cap03Ex15**", coloque-o em execução e observe a apresentação do resultado indicado na figura 3.14.

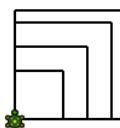


Figura 3.14 - Quadrado desenhado de modo "top-down"

Veja que a partir dos detalhes apresentados é possível criar uma sub-rotina mais "inteligente" que a partir da informação do tamanho de traço e quantidade de lados desenhe qualquer figura entre "3" e "360" graus. Para tanto, entre o seguinte código ao bom estilo "*bottom-up*":

```
1  Turtle.Speed = 10
2
3  Sub Polis
4    For I = 1 To Lado
5      Turtle.Move(Tamanho)
6      Turtle.Turn(360 / Lado)
7    EndFor
8  EndSub
9
10 TextWindow.Clear()
11 TextWindow.Write("Informe o tamanho do quadrado: ")
12 Tamanho = TextWindow.ReadNumber()
13 TextWindow.Write("Informe a quantidade de lados: ")
14 Lado = TextWindow.ReadNumber()
15 Polis()
```

Grave o programa com o nome "Cap03Ex16", coloque-o em execução. Quando solicitado informe o tamanho desejado e a quantidade de lados. Experimente fazer algumas entradas para ver diferentes figuras. Veja algumas sugestões:

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

A partir dessas orientações você já tem em mãos os recursos essenciais para a definição de parâmetros e de sub-rotinas.

3.4 - Programação com decisão

Uma das ações mais curiosas executadas por um computador é a capacidade de efetuar tomadas de decisão, evidentemente, controladas dentro de certos parâmetros.

Para que uma decisão seja tomada é necessário ter primeiro o estabelecimento de uma condição. Condição caracteriza-se, por sua natureza, como sendo uma situação, estado ou circunstância de algo. No universo da computação condição é um estado lógico estabelecido a partir de certa relação lógica entre certos elementos que proporciona uma resposta, podendo ser: *falso* ou *verdadeiro*.

Para se estabelecer uma relação lógica os elementos a serem considerados são variáveis e/ou constantes, podendo-se definir relações entre variáveis com variáveis e variáveis com constantes. A definição dessas relações necessita do uso de *operadores relacionais*. A tabela 3.2 mostra os operadores relacionais encontrados na linguagem "Small Basic".

OPERADOR	SIGNIFICADO
=	Igual a
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
<>	Diferente de

Tabela 3.2 - Operadores relacionais

Com o uso de operadores relacionais têm-se como condições válidas em uma ação de tomada de decisão do tipo variável com variável as relações: $A = B$, $A > B$, $A < B$, $A \geq B$, $A \leq B$ e $A \neq B$. Para a tomada de decisão do tipo variável com constante tem-se como relações: $A = 5$, $A > 5$, $A < 5$, $A \geq 5$, $A \leq 5$ e $A \neq 5$.

A partir do estabelecimento de certa condição usa-se comandos especiais para esta ação representados pelas palavras-chave: **If/Then/Else/EndIf** que pode ser usado a partir das seguintes sintaxes:

```
If <condição> Then  
    <instruções e comandos executados se condição for verdadeira>  
EndIf
```

Ou

```
If <condição> Then  
    <instruções e comandos executados se condição for verdadeira>  
Else  
    <instruções e comandos executados se condição for falsa>  
EndIf
```

Onde, "<condição>" é o estabelecimento de uma relação lógica entre elementos associados com o uso dos operadores relacionais.

A primeira forma de escrita, sem o uso do comando **Else** caracteriza-se por ser a definição de uma tomada de decisão simples e a segunda forma com o uso de **Else** caracteriza-se por ser a definição de uma tomada de decisão composta.

Como exemplo de uso de tomada de decisão simples considere um programa que mostre a mensagem "**Você é maior de idade**" se a idade fornecida for maior ou igual a 18. Caso contrário o programa não deve apresentar nenhuma outra mensagem. Desta forma, considere o código seguinte:

```
1 TextWindow.Clear()  
2 TextWindow.Write("Informe sua idade: ")  
3 Idade = TextWindow.ReadNumber()  
4 If (Idade >= 18) Then  
5     TextWindow.WriteLine("Você é maior de idade")  
6 EndIf
```

Grave o programa com o nome "**Cap03Ex17**", coloque-o em execução. Faça a entrada de um valor maior ou igual a "**18**" e veja a mensagem apresentada. Depois faça uma nova execução e entre um valor menor que "**18**" e veja que nada é apresentado.

Agora, considere um programa em que um valor numérico inteiro é informado é que haja a apresentação de mensagem informado se o valor é par ou ímpar. Veja a diferença entre o uso de tomada de decisão simples com tomada de decisão composta. Observe o código a seguir:

```
1 TextWindow.Clear()  
2 TextWindow.Write("Entre valor numérico inteiro: ")  
3 Valor = TextWindow.ReadNumber()  
4 If (Math.Remainder(Math.Floor(Valor), 2) = 0) Then  
5     TextWindow.WriteLine("Valor par")  
6 Else  
7     TextWindow.WriteLine("Valor ímpar")  
8 EndIf
```

Grave o programa com o nome "**Cap03Ex18**", coloque-o em execução. Faça a entrada de um valor numérico qualquer e veja se o valor fornecido é par ou ímpar. Faça outros testes, inclusive usando valores decimais.

Veja que na linha "4" usa-se duas operações do objeto **Math** sendo **Floor(n)** para garantir o uso de um valor inteiro e **Remainder(dividendo, divisor)** que calcula o resto da divisão. Veja que um valor numérico é par quando seu resto de divisão é igual a "0".

A partir do conhecimento sobre tomada de decisão com de **If/Then/Else/EndIf** considere, como exemplo, um programa chamado "**Cap03Ex19**" que exulta o uso de uma sub-rotina chamada "**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 programa mostre a mensagem de erro "**Use valores entre 1 e 7**". Para tanto, e entre o seguinte código (não se preocupe com o uso do operador lógico **Or**, adiante a sua explicação:

```

1 TextWindow.Clear()
2 TextWindow.Write("Entre valor numérico entre '1' e '7': ")
3 N = TextWindow.ReadNumber()
4 If (N < 1 Or N > 7) Then
5   TextWindow.WriteLine("Use valores entre 1 e 7")
6 Else
7   Turtle.Speed = 10
8   For I = 1 To 8
9     Turtle.Turn(45)
10    For J = 1 To N
11      For K = 1 to 90
12        Turtle.Move(2)
13        Turtle.Turn(2)
14      EndFor
15      Turtle.Turn(90)
16    EndFor
17  EndFor
18 EndIf

```

A sub-rotina "**Floral**" é uma adaptação de exemplo similar codificado em "*Logo*" disponibilizado a partir do material de aula do Professor Carlos Eduardo Aguiar do Centro de Educação Superior a Distância do Estado do Rio de Janeiro, disponível no endereço: "<http://omnis.if.ufrj.br/~carlos/infoenci/notasdeaula/roteiros/aula10.pdf>".

Em seguida execute o programa fornecendo os valores entre "1" e "7" e observe as imagens apresentadas junto a figura 3.15. Seja paciente pois a construção das imagens é um pouco demorada.

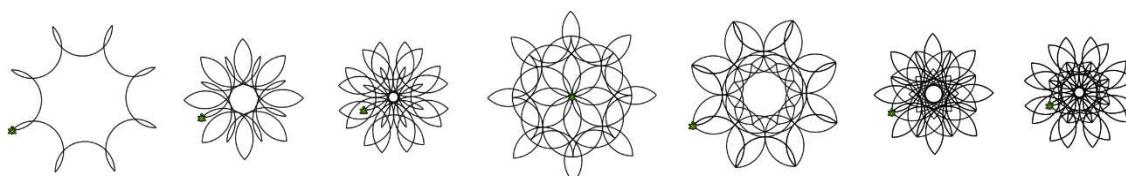


Figura 3.15 - Resultado do procedimento "FLORAL".

Além dos operadores relacionais são encontrados os operadores lógicos que auxiliam ações de tomada de decisão como é o caso do operador **Or** de disjunção usado na sub-rotina "**Floral**", além do operador **And** de conjunção indicados na 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

Tabela 3.3 - Operadores lógicos

Os operadores lógicos **And** e **Or** permitem vincular em uma tomada de decisão duas ou mais condições. Observe as tabelas verdadeas a seguir para cada um dos operadores lógicos.

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 programa que receba um valor numérico e informe se este valor está ou não na faixa de "1" a "9". Desta forma, entre o código:

```
1 | TextWindow.Clear()
2 | TextWindow.Write("Entre valor numérico: ")
3 | N = TextWindow.ReadNumber()
4 | If (N >= 1 And N <= 9) Then
5 |   TextWindow.WriteLine("Valor está na faixa de 1 a 9")
6 | Else
7 |   TextWindow.WriteLine("Valor não está na faixa de 1 a 9")
8 | EndIf
```

Grave o programa com o nome "**Cap03Ex20**", coloque-o em execução. Faça a entrada de diversos valores numéricos para ver a saídas apresentadas.

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 programa 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:

```

1 | TextWindow.Clear()
2 | TextWindow.WriteLine("Entre condição do tempo: ")
3 | Tempo = Text.ToUpperCase(TextWindow.Read())
4 | If (Tempo = "FRIO" Or Tempo = "CHUVOSO") Then
5 |     TextWindow.WriteLine("Tempo ruim, proteja-se ")
6 | Else
7 |     TextWindow.WriteLine("Tempo bom, aproveite ")
8 | EndIf

```

Grave o programa com o nome "**Cap03Ex21**", coloque-o em execução. Faça as entradas **FRIO**, **CHUVOSO**, **CALOR** ou **QUENTE** e observe as saídas apresentadas. Procure fazer mais testes com outros valores. Note o uso da operação **ConvertToUpperCase(n)** do objeto **Text** para garantir que não importando o formato do texto informado este será transformado em texto maiúsculo antes de ser verificado na condição.

3.5 - 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 a definição de uma para forçada. 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 **For**. A diferença é que o comando **For** repete um trecho arbitrário de instruções e a recursão repete a própria sub-rotina. Para demonstrar recursão considere um programa 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:

```
1 | Turtle.Speed = 10
2 |
3 | Sub Espiral
4 |   If (Numero = 0) Then
5 |     Parada = TextWindow.Read()
6 |   EndIf
7 |   Turtle.Move(Tamanho)
8 |   Turtle.Turn(Angulo)
9 |   Tamanho = Tamanho + 8
10 |  Numero = Numero - 1
11 |  Espiral()
12 | EndSub
13 |
14 | TextWindow.Clear()
15 | TextWindow.Write("Informe o tamanho .....: ")
16 | Tamanho = TextWindow.ReadNumber()
17 | TextWindow.Write("Informe o ângulo .....: ")
18 | Angulo = TextWindow.ReadNumber()
19 | TextWindow.Write("Informe a quantidade ...: ")
20 | Numero = TextWindow.ReadNumber()
21 | Espiral()
```

Grave o programa com o nome "**Cap03Ex22**" e coloque-o em execução. Entre primeiro os valores 4, 122 e 57 para ver um triângulo espiralado e depois entre os valores 4, 92 e 47 para ver um quadrado espiralado. A figura 3.16 apresenta as imagens geradas.

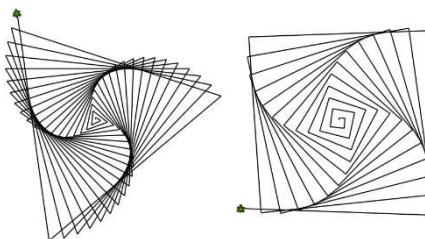


Figura 3.16 - Resultado do efeito recursivo sobre imagens espiraladas.

A partir do código da sub-rotina "**Espiral**" note a definição em cor verde da decisão de aterrramento que detecta, neste caso, se o valor do parâmetro variável **NUMERO** é igual a "0" e sendo faz a parada da execução com a instrução **Parada = TextWindow.Read()**, aguardando que alguma tecla seja acionada no teclado (isso é feito para que a imagem seja apresentada na tela e haja tempo de visualizá-la). Note que se este trecho não estiver definido a sub-rotina "**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 tem antes de sua chamada a atualização dos valores das variáveis **TAMANHO** em "8" (oito), **ANGULO** com valor mantido e a diminuição de "1" (um) sobre o valor da variável **NUMERO**. A cada chamada uma parte da imagem é desenhada pelas instruções grafadas na cor ocre.

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" como medida de ângulo para a sub-rotina "**Espiral**"? Então, mãos à obra, execute o programa duas vezes e entre respectivamente os valores 4, 120 e 57 e depois entre 4, 90 e 47. Veja os resultados apresentados junto a figura 3.17:

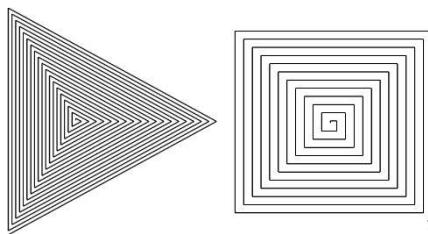


Figura 3.17 - Mais efeitos recursivos sobre imagens espiraladas.

No sentido de apresentar mais ações sobre recursões veja na próxima sequência recursos para 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:

```
1 Turtle.Speed = 10
2
3 Sub Triangular
4   Turtle.Move(40)
5   Turtle.Turn(135)
6   Turtle.Move(40 / Math.SquareRoot(2))
7   Turtle.Turn(90)
8   Turtle.Move(40 / Math.SquareRoot(2))
9   Turtle.Turn(135)
10 EndSub
11
12 Sub Trelica
13   If (X = 0) Then
14     Parada = TextWindow.Read()
15   EndIf
16   For I = 1 To 4
17     Triangular()
18     Turtle.Move(40)
19     Turtle.Turn(90)
20   EndFor
21   Turtle.Turn(90)
22   Turtle.Move(40)
23   Turtle.Turn(-90)
24   X = X - 1
25   Trelica()
26 EndSub
27
28 X = 10
29 Trelica()
```

Grave o programa com o nome "**Cap03Ex23**" e coloque-o em execução. Veja na figura 3.18 a apresentação do resultado do efeito de recursividade.

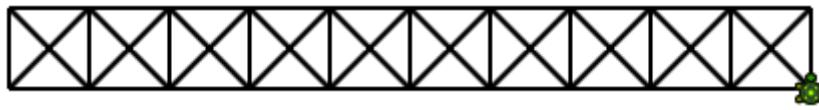


Figura 3.18 - Resultado do efeito de recursividade

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

3.6 - Exercícios de fixação

1. Desenvolver programa 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 de qualquer comando de repetição.
2. Desenvolver programa que desenhe um retângulo com lados de tamanhos "**60**" e "**100**" para trás com giro de graus à esquerda. Usar o comando **For**.
3. Criar, sem uso de qualquer comando de repetição, um programa que construa um pentágono com tamanho "**40**". Avance para frente com giro a direita.
4. Desenvolver programa que construa uma figura com dez lados a partir do uso do comando **For** com giro de graus à esquerda com avanço a frente de "**30**" passos.
5. Desenvolver programa que efetue a leitura de dois valores numéricos representados pelas variáveis "**A**" e "**B**" e apresente o resultado da diferença do maior valor pelo menor valor sem armazenar o resultado em memória.
6. Desenvolver programa que efetue a leitura de dois valores numéricos representados pelas variáveis "**A**" e "**B**" e apresente o resultado da diferença do maior valor pelo menor valor com armazenamento do cálculo em memória.
7. Desenvolver programa que efetue a leitura de três valores numéricos inteiros desconhecidos representados pelas variáveis "**A**", "**B**" e "**C**". Efetue a soma desses valores, armazene o resultado em memória e apresente o resultado somente se este for "**>=**" a "**100**".
8. Desenvolver programa que efetue a leitura de três valores numéricos inteiros desconhecidos representados pelas variáveis "**A**", "**B**" e "**C**". Apresentar os valores dispostos em ordem crescente. Dica: use para este exercício como inspiração o mesmo mecanismo que você usa para aplicar propriedade distributiva, ou seja, compare a variável "**A**" com as variáveis "**B**" e "**C**", depois compare a variável "**B**" com "**C**", lembrando que a cada comparação em que o primeiro valor for maior que o segundo valor deverá esses valores serem trocados.

CAPÍTULO 4

Ações específicas

Anteriormente foram apresentados alguns recursos da linguagem que proporcionam experiências interessantes. A linguagem "*Small Basic*" mesmo sendo uma linguagem simplificada disponibiliza um bom conjunto de recursos. Veja mais alguns deles.

4.1 - Randomização

Um recurso que pode ser explorado em computadores é a capacidade destes conseguirem sortear valores, principalmente numéricos, de forma aleatória. Valores aleatórios também são chamados de *valores randômicos*. Daí o termo *randomização*.

A linguagem "*Small Basic*" possui a operação **GetRandomNumber(n)** do objeto **Math** que possibilita definir o sorteio de número entre "1" e o número especificado em "n". Veja no programa a seguir a apresentação do sorteio de cinco valores entre "0" e "100":

```
1 | TextWindow.Clear()
2 | For I = 1 to 5 Step 1
3 |   Sorteio = Math.GetRandomNumber(101)
4 |   TextWindow.WriteLine(Sorteio - 1)
5 | EndFor
```

Grave o programa com o nome "**Cap04Ex01**", coloque-o em execução e veja na figura 4.1 seu resultado de operação.

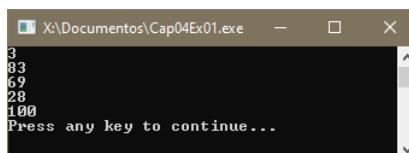


Figura 4.1 - Resultado do sorteio de valores entre "0" e "100"

A partir do recurso de randomização veja o próximo programa que a partir da sub-rotina "**Maquinho**" desenha uma figura geométrica desformada, pois avança para a direita aleatoriamente entre os valores "0" e "60" e gira para a esquerda aleatoriamente entre "0" e "360".

```
1 | Turtle.Speed = 10
2 | For I = 1 to 100 Step 1
3 |   Turtle.Move(Math.GetRandomNumber(61) - 1)
4 |   Turtle.Turn(Math.GetRandomNumber(361) - 1)
5 | EndFor
```

Grave o programa com o nome "**Cap04Ex02**", coloque-o em execução e veja na figura 4.2 seu resultado de operação.

A geração de valores aleatórios é uma ação ideal para realizar a simulação na obtenção de valores quando há a necessidade de realizar testes de entrada em um programa. Outra aplicação para este recurso pode ser o desenvolvimento de jogos onde as situações enfrentadas pelos personagens podem ocorrer de forma imprevisível tornado a dinâmica do jogo mais atraente e emocionante.

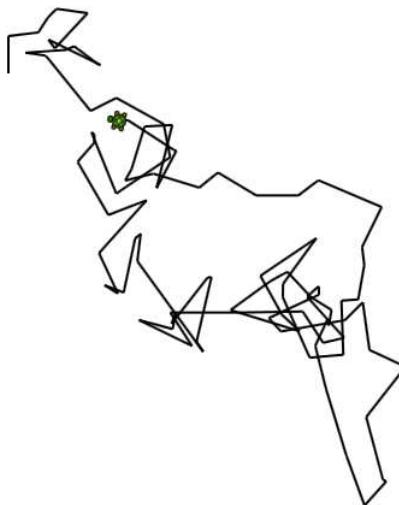


Figura 4.2 - Resultado de figura geométrica aleatória

4.2 - Coordenadas

Todos as figuras geométricas planas apresentadas até este momento foram desenhadas a partir de determinado ponto cartesiano no plano de ação gráfica da linguagem "*Small Basic*". O que não sabemos ainda é em que posição cartesiana do plano isso ocorreu. Para saber, basta fazer uso das propriedades **X** e **Y** do objeto **Turtle**. Veja o código seguinte:

```
1 | TextWindow.Clear()  
2 | TextWindow.WriteLine("[ " + Turtle.X + "," + Turtle.Y + "]")
```

Grave o programa com o nome "**Cap04Ex03**", coloque-o em execução e veja na figura 4.3 seu resultado de operação.

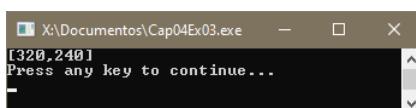


Figura 4.3 - Apresentação das coordenadas de posicionamento da tartaruga

Veja que é apresentado os valores "[**320,240**]" informando que por padrão a tartaruga está posicionada na coordenada "X" com "**320**" e coordenada "Y" com "**240**". Isto indica que essa é a posição inicial de ação da tartaruga, de onde todos os desenhos anteriores foram realizados.

Antes de realizar posicionamentos no plano gráfico é importante ter noção de como é sua organização e isso é conseguido com o uso de alguns recursos do objeto **GraphicsWindow**, tais como as propriedades **Height** (altura) e **Width** (comprimento). Observe o código seguinte:

```
1 | TextWindow.Clear()  
2 | TextWindow.WriteLine("Comprimento ...: " + GraphicsWindow.Width)  
3 | TextWindow.WriteLine("Altura .....: " + GraphicsWindow.Height)
```

Grave o programa com o nome "**Cap04Ex04**", coloque-o em execução e serão apresentadas duas janelas de saída: uma janela gráfica e outra janela texto, que são telas já conhecidas. Isso ocorre pelo fato do programa estar utilizando dois objetos diferentes, sendo **TextWindow** para a janela texto e **GraphicsWindow** para a janela gráfica. Veja na figura 4.4 o resultado da operação na janela texto indicando o tamanho de "**624**" pixels para o comprimento e "**441**" para a

altura. O *pixel* é o menor ponto luminoso acesso em seu monitor de vídeo, é o elemento que possibilita a apresentação de textos e imagens.

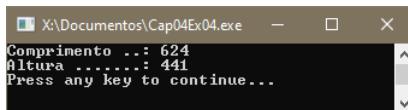


Figura 4.4 - Dimensão padrão da janela gráfica

Temos agora duas informações importantes: as coordenadas de posicionamento da tartaruga "[320,240]" e a dimensão da janela gráfica "[624,441]". Observe que o posicionamento da tartaruga é definido em quase que a metade da dimensão da janela, como apresentado na figura 4.5.



Figura 4.5 - Posicionamento quase que "centralizado" da tartaruga

Veja a partir da figura 4.5 que a posição aparentemente central da tartaruga depende da dimensão de tamanho da janela, pois as coordenadas são iniciadas nas posições "[0,0]" e se estendem para a direita e para baixo em relação ao tamanho do comprimento e da altura com valores de posicionamento positivos.

O tamanho máximo permitido para uma janela é o tamanho que seu sistema consegue definir para sua configuração de placa gráfica (adaptador de vídeo) e monitor de vídeo em uso. Para saber sobre essa dimensão abra em seu sistema o recurso **Configurações**, selecione **Sistema** e escolha **Vídeo** e veja o valor indicado no campo "**Resolução da tela**" no sistema utilizado para escrever este livro a dimensão máxima indicada era de "**1920 x 1080**", ou seja, "**1920**" de comprimento e "**1080**" de altura. No entanto, não é conveniente fazer uso de toda essa dimensão. Desta forma, trabalhe em um percentual de tamanho um pouco menor, por exemplo, **10%** menos. Tomando então por base a dimensão "**1728 x 972**" (10% menor que o máximo) observe o código do programa seguinte que tem por finalidade apresentar uma janela com tal tamanho:

```
1 | GraphicsWindow.Width  = 1728
2 | GraphicsWindow.Height =  972
3 | Turtle.Move(0)
4 | Turtle.Turn(0)
```

Grave o programa com o nome "**Cap04Ex05**", coloque-o em execução e veja na figura 4.6 seu resultado de operação.

A mudança da dimensão é necessária quando o desenho a ser produzido precisa de maior espaço. Caso contrário não é necessário fazer essa mudança.



Figura 4.6 - Janela com tamanho redimensionado

Veja na figura 4.6 que a tartaruga continua na posição "[320,240]" e considerando que se queira coloca-la centralizada na nova dimensão é necessário dividir cada valor da dimensão de comprimento e altura por dois. Desta forma o valor de centralização da tartaruga corresponde a "864" para a coordenada "X" e "461" para a coordena "Y". Caso o valor da metade seja retornado como decimal considere apenas o valor inteiro calculado. Veja o código seguinte:

```
1 | GraphicsWindow.Width  = 1728
2 | GraphicsWindow.Height =  972
3 | Turtle.X = 864
4 | Turtle.Y = 461
5 | Turtle.Move(0)
6 | Turtle.Turn(0)
```

Grave o programa com o nome "**Cap04Ex06**", coloque-o em execução e veja na figura 4.7 seu resultado de operação.



Figura 4.7 - Janela com tamanho redimensionado
e tartaruga centralizada

O objeto **GraphicsWindow** fornece opções de funcionalidades para o uso de ações de entradas, saídas e manipulações em modo gráfico, como a obtenção dos tamanhos da altura, do comprimento e limpeza de tela gráfica com a operação **Clear()**. Possui recursos próprios mas pode ser usada em conjunto com o objeto **Turtle** para aumentar as possibilidades de trabalho no modo geometria da tartaruga.

Antes de avançarmos vamos conhecer outra operação do objeto **Turtle** chamada **MoveTo(x, y)**. Esta operação tem por finalidade mover a tartaruga fazendo-a caminhar para certa posição. Se a caneta estiver abaixada, será desenhada uma linha conforme ela se move, então se a intenção é mover a tartaruga sem traçar o caminho é necessário manipular a caneta, levantando-a com a operação **PenUp()**. Lembre-se de após mover abaixa a caneta com a operação **PenDown()**.

A partir desses "novos" recursos e de alguns recursos conhecidos é possível extrapolar e criar desenhos e imagens mais elaboradas, como por exemplo a figura da face quadrada indicada junto a figura 4.8, adaptada de Harvey (1997, p. 184) e desenhada dentro da dimensão padrão de "624" pixels para o comprimento e "441" para a altura com a tartaruga posicionada em "X" com "320" e "Y" com "240".

Apesar de simples a imagem da figura 4.8 apresenta um grau de complexidade interessante, pois diversas ações são necessárias. Assim sendo, entre o seguinte código:

Assim sendo, entre o seguinte código:

```

1 GraphicsWindow.Clear()
2
3 Sub Cabeca
4     Turtle.PenDown()
5     For I = 1 To 4
6         Turtle.Move(100)
7         Turtle.Turn(90)
8     EndFor
9 EndSub
10
11 Sub PreparaOlho
12     Turtle.Turn(-90)
13     Turtle.Move(65)
14     Turtle.Turn(90)
15     Turtle.Move(20)
16 EndSub
17
18 Sub Olho
19     For I = 1 To 4
20         Turtle.Move(15)
21         Turtle.Turn(90)
22     EndFor
23 EndSub
24
25 Sub OlhoEsquerdo
26     Turtle.PenDown()
27     Olho()

```

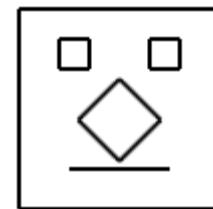


Figura 4.8 - Face quadrada

```
28  Turtle.PenUp()
29  Turtle.Move(45)
30 EndSub
31
32 Sub OlhoDireito
33  Turtle.PenDown()
34  Olho()
35  Turtle.PenUp()
36  Turtle.Move(-15)
37 EndSub
38
39 Sub Nariz
40  Turtle.Turn(90)
41  Turtle.Move(20)
42  Turtle.Turn(-45)
43  Turtle.PenDown()
44  For I = 1 To 4
45    Turtle.Move(29)
46    Turtle.Turn(90)
47  EndFor
48  Turtle.PenUp()
49  Turtle.MoveTo(320,240)
50  Turtle.Turn(-135)
51  Turtle.PenDown()
52 EndSub
53
54 Sub Boca
55  Turtle.PenUp()
56  Turtle.Move(20)
57  Turtle.Turn(90)
58  Turtle.Move(25)
59  Turtle.PenDown()
60  Turtle.Move(50)
61  Turtle.PenUp()
62  Turtle.Move(-75)
63 EndSub
64
65 Sub Face
66  Cabeca()
67  Boca()
68  PreparaOlho()
69  OlhoEsquerdo()
70  OlhoDireito()
71  Nariz()
72 EndSub
73
74 Face()
75 Turtle.TurnLeft()
76 Turtle.Hide()
```

Grave o programa com o nome "**Cap04Ex07**" e coloque-o em execução para ver o desenho do rosto quadrado ser apresentado. Veja que o objeto **GraphicsWindow** também possui recurso para limpeza da tela gráfica com a operação **Clear()**.

4.3 - Cores

A linguagem "*Small Basic*" permite fazer uso de cores nas janelas gráfica e texto. É possível manipular a cor do fundo da janela (cor de fundo) e a cor usada para desenhar ou escrever (cor de frente).

Como exemplo de manipulação de cores em texto considere um programa que pergunte seu nome em cor verde, que a entrada ocorra na cor ciano, que a mensagem de saudação apresentada esteja em amarelo e que a mensagem de paralização de tela esteja em magenta. Veja o código seguinte:

```

1 | TextWindow.Clear()
2 | TextWindow.ForegroundColor = "Green"
3 | TextWindow.WriteLine("Olá, qual é o seu nome? ---> ")
4 | TextWindow.ForegroundColor = "Cyan"
5 | Nome = TextWindow.Read()
6 | TextWindow.WriteLine("")
7 | TextWindow.ForegroundColor = "Yellow"
8 | TextWindow.WriteLine("Olá " + Nome)
9 | TextWindow.ForegroundColor = "Magenta"
10| TextWindow.WriteLine("Tecle <Enter> para encerrar...")
11| Tecla = TextWindow.Read()
12| Program.End()

```

Grave o programa com o nome "**Cap04Ex08**" e coloque-o em execução para ver o desenho do rosto quadrado ser apresentado. A figura 4.9 mostra a ocorrência.

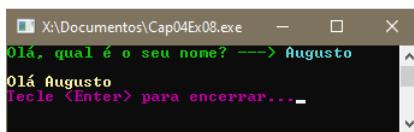


Figura 4.9 - Apresentação de texto colorizado

Atente para a definição das cores entre aspas inglesas junto as propriedades **ForegroundColor** do objeto **TextWindow** marcadas, no código, em vermelho. Veja que a partir da definição de uma cor esta fica ativa até que outra venha a ocorrer.

Os programas em janela texto anteriores apresentam uma mensagem padrão de pausa, que foi substituída neste programa com a instrução da linha "**10**" com uma parada de telado realizada na linha "**11**" e o encerramento do programa com a operação **End()** do objeto **Program** que fecha a janela de operação.

Veja no próximo exemplo uma mensagem sendo apresentada na cor cinza e fundo azul, ambas em tom escuro e definidas a partir de seus valores numéricos de referência:

```

1 | TextWindow.Clear()
2 | TextWindow.ForegroundColor = 8
3 | TextWindow.BackgroundColor = 1
4 | TextWindow.WriteLine("Estudo de Small Basic ")
5 | TextWindow.WriteLine("")

```

Grave o programa com o nome "**Cap04Ex09**", coloque-o em execução e veja na figura 4.10 o resultado.



Figura 4.10 - Apresentação de texto colorizado

Caso o fundo não seja totalmente apresentado de forma colorizada basta alterar o tamanho da janela com o ponteiro do *mouse* em suas bordas para que o efeito seja aplicado em toda a janela.

A tabela 4.1 apresenta o conjunto de cores, de acordo com seus valores numéricos e nomes, suportados para uso em janelas de texto.

VALOR	NOME	COR
0	Black	Preto
1	DarkBlue	Azul escuro
2	DarkGreen	Verde escuro
3	DarkCyan	Ciano escuro
4	DarkRed	Vermelho escuro
5	DarkMagenta	Magenta escuro
6	DarkYellow	Amarelo escuro
7	Gray	Cinza
8	DarkGray	Cinza escuro
9	Blue	Azul
10	Green	Verde
11	Cyan	Ciano
12	Red	Vermelho
13	Magenta	Magenta
14	Yellow	Amarelo
15	White	Branco

Tabela 4.1 - Índice e cores para janela texto

Os desenhos produzidos até este momento foram riscados em cor preta em um fundo branco. A linguagem "Small Basic" permite manipular o conjunto de cores do risco do desenho e do fundo da tela gráfica.

A definição de cores de fundo na janela gráfica é feita com a propriedade **BackgroundColor** do objeto **GraphicsWindow**. Para a definição da cor dos traços dos desenhos e efeitos de pintura sobre imagens usa-se a operação **GetColorFromRGB(r, g, b)** que deve estar associada a propriedade **BrushColor** ou propriedade **PenColor** do objeto **GraphicsWindow**.

A cor de fundo da janela gráfica fica limitada as cores apresentadas na tabela 4.1, mas as cores de fundo podem usar a relação **RGB** (**Red**, **Green** e **Blue**), onde cada cor a ser usada é a combinação do espectro de cores formado a partir das cores básicas definidas por meio de valores entre "**0**" e "**255**".

O sistema de cores **RGB** pode não ser intuitivo, mas é um sistema que possibilita a geração de **16.777.216** tons de cores diferentes, considerando-se os níveis de brilho e saturação dessas cores, mesmo que não seja possível ao olho humano distinguir tal espectro de cores.

Para um teste do uso de cores em janela gráfica considere o código de programa a seguir que desenha sol estilizado com o traço da caneta na cor laranja "**(255 165 000)**" em fundo azul escuro "**DarkBlue**" (adaptado de Abelson, 1981, p. 22). A Figura 4.11 apresenta a imagem gerada pela execução do programa "**Cap04Ex10**".

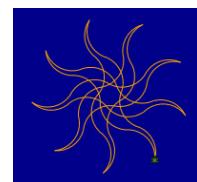


Figura 4.11 - Sol colorizado

```

1 Turtle.Speed = 10
2 GraphicsWindow.Width  = 1728
3 GraphicsWindow.Height =  972
4 Turtle.X = 864
5 Turtle.Y = 461
6
7 Sub Arco1
8   For I1 = 1 To Grau / 10
9     Turtle.Move(Tamanho)
10    Turtle.Turn(10)
11  EndFor
12 EndSub
13
14 Sub Arco2
15   For I2 = 1 To Grau / 10
16     Turtle.Move(Tamanho)
17     Turtle.Turn(-10)
18   EndFor
19 EndSub
20
21 Sub Raio
22   For I3 = 1 To 2
23     Grau = 90
24     Arco2()
25     Arco1()
26   EndFor
27 EndSub

```

```
28
29 Sub Sol
30   GraphicsWindow.BackgroundColor = "DarkBlue"
31   GraphicsWindow.PenColor = GraphicsWindow.GetColorFromRGB(255, 165, 0)
32   For I4 = 1 to 9
33     Raio()
34     Turtle.Turn(160)
35   EndFor
36 EndSub
37
38 Tamanho = 10
39 Sol()
```

Veja no programa "**Cap04Ex10**" a definição da cor de fundo em tom vermelho utilizando-se a propriedade **BackgroundColor** do objeto **GraphicsWindow** na definição da cor azul escuro idêntica a propriedade de mesmo nome do objeto **TextWindow**.

Para definir a cor da caneta em tom laranja se usa a propriedade **PenColor** com atribuição da operação **GetColorFromRGB(r, g, b)**, ambas pertencentes ao objeto **GraphicsWindow**. Há três possibilidades de definição de cores quando se utiliza a propriedade **BrushColor** ou a propriedade **PenColor** do objeto **GraphicsWindow**, sendo:

1. pelo nome da cor predefinida como 4.1, ressaltando que existem mais nomes de cores;
2. pelo número hexadecimal equivalente a cor em RGB;
3. pelo número decimal de identificação da cor em RGB.

Assim sendo, podemos usar as formas:

```
(Caso 1) - GraphicsWindow.PenColor = "DarkBlue"
(Caso 2) - GraphicsWindow.PenColor = "#00008B"
(Caso 3) - GraphicsWindow.PenColor = GraphicsWindow.GetColorFromRGB(0,0,139)
```

4.4 - Fractais

Fractal é uma estrutura geométrica de forma complexa não regular que possui normalmente propriedades que se repetem em diversas escalas. Não é objetivo deste tópico, explorar o tema em profundidade, mas demonstrá-lo. Para tanto, considere um programa que possua algumas sub-rotinas especializadas em apresentar fractais. O programa a seguir receberá implementações momentâneas até chegar ao projeto final. Assim sendo, observe o código seguinte:

```
1 Sub Ponta
2   Turtle.Move(Lado)
3   Turtle.Turn(-60)
4   Turtle.Move(Lado)
5   Turtle.Turn(120)
6   Turtle.Move(Lado)
7   Turtle.Turn(-60)
8   Turtle.Move(Lado)
9 EndSub
10
11 Lado = 40
12 Ponta()
```

Grave o programa com o nome "**Cap04Ex11**", coloque-o em execução e veja na figura 4.12 o resultado.



Figura 4.12 - Ponta de um fractal

A partir da sub-rotina "**Ponta**" observe o próximo programa com o uso de uma segunda sub-rotina chamada "**Fractal**", como segue:

```

1 Sub Ponta
2   Turtle.Move(Lado)
3   Turtle.Turn(-60)
4   Turtle.Move(Lado)
5   Turtle.Turn(120)
6   Turtle.Move(Lado)
7   Turtle.Turn(-60)
8   Turtle.Move(Lado)
9 EndSub
10
11 Sub Fractal
12   For I = 1 To 6
13     Turtle.Turn(120)
14     Ponta()
15     Turtle.Turn(-60)
16   EndFor
17 EndSub
18
19 Lado = 30
20 Fractal()
```

Grave o programa com o nome "**Cap04Ex12**", coloque-o em execução e veja na figura 4.13 o resultado.

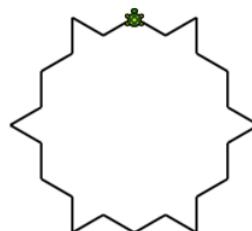


Figura 4.13 - Resultado de um "Fractal"

Com algumas pequenas mudanças na direção do programa anterior consegue-se uma imagem fractal diferente. Assim sendo, considere o código a seguir onde se definem mudanças na direção de deslocamento da tartaruga dentro da repetição gerenciada pelo comando **For**.

Observe no código seguinte as linhas sinalizadas em vermelho e compare-as com a versão anterior do programa. Perceba onde foi e quais as mudanças indicadas.

```
1 Sub Ponta
2   Turtle.Move(Lado)
3   Turtle.Turn(-60)
4   Turtle.Move(Lado)
5   Turtle.Turn(120)
6   Turtle.Move(Lado)
7   Turtle.Turn(-60)
8   Turtle.Move(Lado)
9 EndSub
10
11 Sub Fractal
12   For I = 1 To 6
13     Turtle.Turn(-120)
14     Ponta()
15     Turtle.Turn(60)
16   EndFor
17 EndSub
18
19 Lado = 30
20 Fractal()
```

Grave o programa com o nome "**Cap04Ex13**", coloque-o em execução e veja na figura 4.14 o resultado.

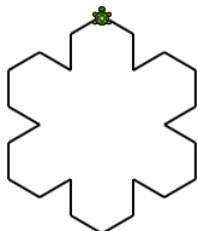


Figura 4.14 - Resultado de mudanças no "Fractal"

É possível a partir da sub-rotina "**Ponta**" fazer a apresentação de formas fractais mais complexos. Observe o programa a seguir que apresenta um trecho de imagem fractal bastante tradicional, como indicado na figura 4.15.

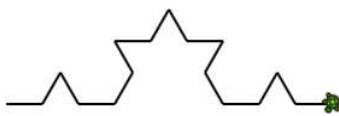


Figura 4.15 - Fractal tradicional (base do flocos de neve)

Para a apresentação do fractal da figura 4.15 considere os detalhes do próximo código de programa. Atente ao fato de que neste e os exemplos anteriores deste tópico a sub-rotina "**Ponta**" é a base, o pivô, de toda as ações. Veja como um elemento aparentemente simples pode ser usado para o desenvolvimento de formas complexas.

A diferença entre os programas apresentados está no posicionamento da tartaruga para a continuidade do desenho. Para este caso, faz-se o deslocamento de direção da tartaruga após desenhar "**Ponta**" pela primeira vez "**60**" graus para a esquerda e faz-se novo desenho de "**Ponta**" e gira "**120**" graus para a direita, observe o próximo código:

```
1 Sub Ponta
2   Turtle.Move(Lado)
3   Turtle.Turn(-60)
4   Turtle.Move(Lado)
5   Turtle.Turn(120)
6   Turtle.Move(Lado)
7   Turtle.Turn(-60)
8   Turtle.Move(Lado)
9 EndSub
10
11 Sub Fractal
12   Turtle.TurnRight()
13   For I = 1 To 2
14     Ponta()
15     Turtle.Turn(-60)
16     Ponta()
17     Turtle.Turn(120)
18   EndFor
19 EndSub
20
21 Lado = 30
22 Fractal()
```

Grave o programa com o nome "**Cap04Ex14**", coloque-o em execução e veja que o resultado apresentado assemelha-se a imagem da figura 4.15.

A partir da imagem da figura 4.15 pode-se extrapolar um pouco para obter o desenho de um floco de neve, propriamente dito, a partir da "**Fractal**" que usa como base a sub-rotina "**Ponta**". Neste caso, considere o código a seguir e observe atentamente a estrutura de código da sub-rotina "**Floco**":

```
1 Sub Ponta
2   Turtle.Move(Lado)
3   Turtle.Turn(-60)
4   Turtle.Move(Lado)
5   Turtle.Turn(120)
6   Turtle.Move(Lado)
7   Turtle.Turn(-60)
8   Turtle.Move(Lado)
9 EndSub
10
11 Sub Floco
12   Turtle.TurnRight()
13   For I = 1 To 2
14     Ponta()
15     Turtle.Turn(-60)
16     Ponta()
17     Turtle.Turn(120)
18   EndFor
19 EndSub
20
```

```
21 Sub Floco
22   For J = 1 To 4
23     Fractal()
24     Turtle.Turn(-120)
25     Turtle.Turn(30)
26   EndFor
27 EndSub
28
29 Lado = 30
30 Floco()
```

Grave o programa com o nome "**Cap04Ex15**", coloque-o em execução e veja na figura 4.16 o resultado.

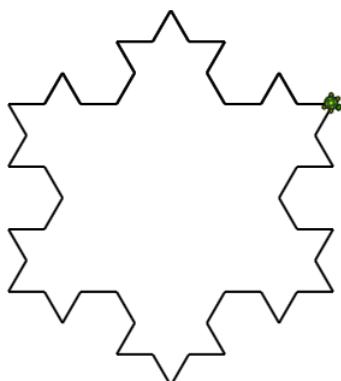


Figura 4.16 - Floco de neve

4.5 - Mais repetições

Além das repetições produzidas com o comando **For** e pelas operações de recursão a linguagem "Small Basic" possui outras duas formas de realizar operações de repetição com os comandos: **While/EndWhile** e **Goto**.

O comando **While** (laço condicional pré-teste com fluxo verdadeiro) realiza repetições de trechos de código de forma condicional sem o uso de recursividade enquanto a condição de verificação permanece verdadeira. No momento em que a condição torna-se falsa o laço é automaticamente encerrado. Isso é útil em momentos que se necessite de laço interativo onde não se conhece antecipadamente o número de repetições.

Para verificar o efeito de funcionamento do comando **While** considere um programa que apresente os valores de contagem entre "1" e "5" de "1" em "1". Assim sendo, entre o código:

```
1 I = 1
2 While (I <= 5)
3   TextWindow.WriteLine(I)
4   I = I + 1
5 EndWhile
```

Grave o programa com o nome "**Cap04Ex16**", coloque-o em execução e veja na figura 4.17 o resultado.

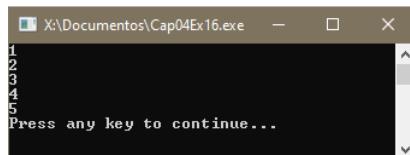


Figura 4.17 - Apresentação da contagem de "1" a "5" com "While"

Veja que para o comando **While** funcionar a condição deve permanecer verdadeira por certo tempo, pois a repetição somente é encerrada quando a condição se torna falsa.

Observe que no programa a variável "I" é iniciada com o "1" (linha "1") e cada escrita é somada a ela mais "1" por meio da instrução "I = I + 1" (linha "4") fazendo com que a variável "I" atinja um valor que fará a condição "(I <= 5)" se tornar falsa encerrando as repetições.

A partir deste recurso veja o próximo programa que tem por finalidade apresentar uma estrela de cinco pontas. Para tanto, e entre o seguinte código:

```

1 | Turtle.Turn(18)
2 | I = 1
3 | While (I <= 5)
4 |   Turtle.Move(100)
5 |   Turtle.Turn(144)
6 |   I = I + 1
7 | EndWhile

```

Grave o programa com o nome "**Cap04Ex17**", coloque-o em execução e veja na figura 4.18 o resultado.



Figura 4.18 - Apresentação de imagem gerada com comando "While"

Os trechos marcados em vermelho mostram a estrutura a ser usada na definição de repetições iterativas que venham a substituir o uso do comando **For**. Usar as definições (**I = 1**) para a inicialização e incremento (**I = I + 1**) do programa são obrigatórias.

Para se fazer o desenho da estrela de cinco pontas é necessário usar um ângulo de "144" graus como apresentado. Talvez a pergunta em sua mente, seja: como saberei esse valor de grau? A resposta é basicamente simples. Divida o valor de graus máximos que é "360" por "5" e obter-se-á o valor "72" que é o ângulo interno de uma figura geométrica, multiplique o valor "72" por "2" e obter-se-á o valor "144" que é o valor do ângulo externo considerado e você terá o valor correspondente ao ângulo de giro necessário.

Veja que a linguagem "*Small Basic*" fornece dois comandos para ações de repetição: o comando **For** (não condicional) voltado as operações *iterativas* e o comando **While** (condicional) voltado a execução de repetições *iterativas* e *interativas*. Quanto a execução de ações de repetições *interativas*, aguarde um pouco. Estamos chegando lá.

No entanto, a linguagem deixa de lado o oferecimento de um comando de repetição que opere em sentido contrário ao **While**, ou seja, um comando que realize as repetições em estilo pós-teste até que a condição de encerramento torne-se verdadeira. Mas, oferece outro comando que em conjunto com o comando **If** pode simular este tipo de ação, trata-se do comando **Goto**

que deve ser usado com muito cuidado e dentro de certos limites de comportamento operacional.

Para testar uma simulação de repetição com fluxo falso que seja encerrada no momento em que a condição se torna verdadeira considere um programa que receba a entrada de um valor numérico inteiro positivo maior ou igual a zero. Caso o número fornecido seja menor que zero o programa não poderá aceitar tal entrada. Após a entrada do valor correto o programa deve apresentar o resultado do número informado ao quadrado. Assim sendo, considere o código a seguir:

```

1 | Repita:
2 |   TextWindow.WriteLine("Entre um valor inteiro positivo: ")
3 |   Vlr = Math.Floor(TextWindow.ReadNumber())
4 |   If (Vlr >= 0) Then
5 |     Goto FimRepita
6 |   EndIf
7 |   Goto Repita
8 | FimRepita:
9 | TextWindow.WriteLine("Resultado = " + Math.Power(Vlr, 2))

```

Grave o programa com o nome "**Cap04Ex18**", coloque-o em execução e veja na figura 4.19 o resultado.

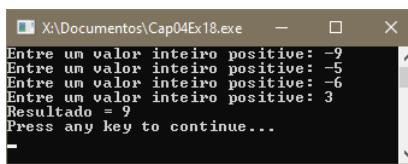


Figura 4.19 - Resultado da simulação de repetição pós-teste

Veja que o programa simula a proposta do que seria a estrutura de repetição ao estilo pós-teste chamada **Repeat/EndRepeat**, neste caso, representada pelos rótulos "**Repita**" e "**FimRepita**". Veja que se a condição verificada entre as linhas "4" e "6" sendo verdadeira desvia o fluxo do programa (**Goto**) para a linha "8" identificada pelo rótulo "**FimRepita:**". Se a condição for falsa a instrução da linha "7" faz o desvio diretamente para a linha "1". A definição de um rótulo segue as mesmas regras para definição de variáveis, tendo como diferença o uso do símbolo de dois pontos ":" após sua definição.

O programa anterior pode ser simplificado eliminando-se um comando **Goto**, mas neste caso, não será mais uma simulação de **Repeat/EndRepeat** e sim um comando **DoWhile/EndDoWhile**. Então observe a nova proposta:

```

1 | FacaEnquanto:
2 |   TextWindow.WriteLine("Entre um valor inteiro positivo: ")
3 |   Vlr = Math.Floor(TextWindow.ReadNumber())
4 |   If (Vlr < 0) Then
5 |     Goto FacaEnquanto
6 |   EndIf
7 | FimFacaEnquanto:
8 | TextWindow.WriteLine("Resultado = " + Math.Power(Vlr, 2))

```

Grave o programa com o nome "**Cap04Ex19**", coloque-o em execução e veja que o resultado apresentado assemelha-se ao resultado indicado na figura 4.19.

O rótulo definido na linha "7" pode ser retirado do código do programa. Foi usado apenas para indicar o final do bloco de repetição. Veja que os dois programas anteriores demonstram o uso de repetição interativa. Uma repetição é considerada interativa quando não se sabe de antemão quantas vezes a repetição deve ocorrer.

Como mais um exemplo considere como problema computacional a necessidade de criar um programa de computador que apresente o resultado da factorial de um número inteiro positivo fornecido qualquer. Este programa deve após indicar a saída do cálculo apresentar ao 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 pelo programa.

A situação proposta é um caso em que o comando **While** não atenderá adequadamente o que deve ser feito. Você terá diversos problemas a serem resolvidos. Neste caso, o aconselhável é fazer uso de repetições pós-teste, onde você entra primeiro e verifica após a entrada a validade do que deseja ser feito. Como a linguagem "*Small Basic*" não possui o recurso a solução é simulá-lo com o comando **Goto** controlado por um trecho **If/Then/EndIf**. Veja esta solução no código a seguir:

```

1 | Repita1:
2 |   Repita2:
3 |     TextWindow.WriteLine("Entre um valor inteiro positivo: ")
4 |     N = Math.Floor(TextWindow.ReadNumber())
5 |     If (N < 0) Then
6 |       Goto Repita2
7 |     EndIf
8 |     Fat = 1
9 |     For I = 1 To N
10 |       Fat = Fat * I
11 |     EndFor
12 |     TextWindow.WriteLine(N + "!" = " + Fat)
13 |   Repita3:
14 |     TextWindow.WriteLine("Deseja continuar? Responda: SIM ou NÃO ")
15 |     Resp = Text.ConvertToUpperCase(TextWindow.Read())
16 |     If (Resp <> "SIM" And Resp <> "NÃO") Then
17 |       Goto Repita3
18 |     EndIf
19 |     If (Resp = "SIM") Then
20 |       Goto Repita1
21 |     EndIf

```

Grave o programa com o nome "**Cap04Ex20**", coloque-o em execução e veja que o resultado apresentado assemelha-se ao resultado indicado na figura 4.20.

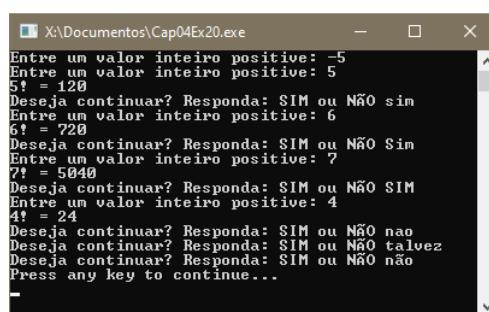


Figura 4.20 - Cálculo de factorial

A figura 4.20 mostra alguns momentos de uso do programa. Atente para cada detalhe indicado.

Veja que o programa é formado por três momento operacionais colorizados em vermelho, verde e ocre.

Em vermelho encontra-se a parte principal do programa, ou seja, seu efetivo processamento que é apresenta o resultado da factorial dos valores numéricos inteiros e positivos eventualmente fornecidos.

O trecho sinalizado em verde corresponde a validação e aceite da entrada de valores numéricos inteiros e positivos. Se um valor numérico negativo é fornecido o usuário fica "preso" neste trecho de programa até que efetue a entrada correta.

O trecho em ocre valida a entrada das respostas "**SIM**" e "**NÃO**". Qualquer resposta diferente das duas esperadas é recusada pelo programa. São aceitas variações de respostas esperadas tanto em formato maiúsculo, minúsculo ou combinações entre esses formatos.

Já que foi desenhada uma estrela de cinco pontas (pentagrama) com o comando **While** será feita uma estrela com várias pontas com a simulação produzida com o comando **Goto**. Para tanto, entre o seguinte código:

```
1 | Turtle.Speed = 10
2 | I = 1
3 | Repita:
4 |   Turtle.Move(200)
5 |   Turtle.Turn(-170)
6 |   I = I + 1
7 |   If (I <= 40) Then
8 |     Goto Repita
9 | EndIf
```

Grave o programa com o nome "**Cap04Ex21**", coloque-o em execução e veja que o resultado apresentado assemelha-se ao resultado indicado na figura 4.21.

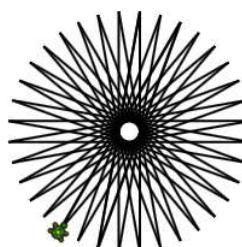


Figura 4.21 - Apresentação de estrela

Os comandos: **For**, **While** e a simulação de repetição pós-teste feita com o comando **Goto** são recursos complementares e não necessariamente substitutos um dos outros. Esses comandos podem ser trabalhados juntos dentro de um mesmo projeto. Os comandos podem ser usados sequencialmente ou encadeados. Sequencialmente quando um é definido após o outro e encadeado quando é definido um dentro do outro.

Veja a seguir um programa que faz uso de repetições sequenciais utilizando-se para a ação os comandos **For** e **While** no mesmo projeto na forma sequencial. Assim sendo, entre o seguinte código:

```

1 | Turtle.Speed = 10
2 | For I = 1 To 6
3 |   J = 1
4 |   While (J <= 120)
5 |     Turtle.Move(1)
6 |     Turtle.Turn(1)
7 |     J = J + 1
8 |   EndWhile
9 |   Turtle.Turn(60)
10| For K = 1 to 180
11|   Turtle.Move(1)
12|   Turtle.Turn(1)
13| EndFor
14| Turtle.Turn(60)
15| EndFor

```

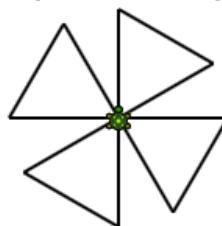
Grave o programa com o nome "**Cap04Ex22**", coloque-o em execução e veja que o resultado apresentado assemelha-se ao resultado indicado na figura 4.22.



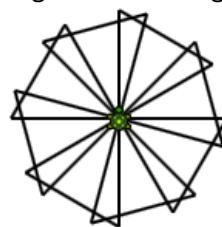
Figura 4.22 - Apresentação de imagem gerada com repetições sequenciais e encadeadas

4.6 - Exercícios de fixação

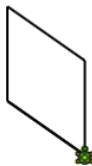
1. Criar programa que desenhe um retângulo cujo lado menor (comprimento) seja a metade do lado maior (altura) em que o tamanho seja informado pelo usuário. Movimente o desenho para traz com sentido a direita.
2. Crie um programa que apresente um triângulo equilátero com lado de tamanho "70" para trás a partir de giros a esquerda.
3. Crie um programa que a partir de uma sub-rotina que desenhe um triângulo equilátero com lado de tamanho "70" para a frente e giro a direita e gere a imagem.



4. Crie um programa que a partir de uma sub-rotina que desenhe um triângulo equilátero com lado de tamanho "70" para a frente e giro a direita e gere a imagem.



5. Crie um programa que desenhe um losango com tamanho fornecido pelo usuário a partir de giro a direita. Assim que o desenho for concluído faça com que a tartaruga seja posicionada na parte inferior a direita como mostra a imagem seguinte.



6. Crie programa que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se o comando **For**.
7. Crie programa que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se o comando **While**.
8. Crie programa que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use o comando **For**.
9. Crie programa que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use o comando **While**.
10. Crie programa que mostre todos os valores numéricos divisíveis por "4" menores que "20" iniciando a contagem em "1". Use o comando **For**.
11. Crie programa que mostre todos os valores numéricos divisíveis por "4" menores que "20" iniciando a contagem em "1". Use o comando **While**.
12. Crie programa que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use o comando **For**.
13. Crie programa que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use o comando **While**.

CAPÍTULO 5

Ações complementares

A linguagem "Small Basic" apesar de simplificada em relação a outras linguagens de programação oferece uma série de recursos que auxiliam a aprendizagem de sofisticadas técnicas de programação. Neste capítulo são apresentados alguns detalhes complementares da linguagem que facilitam este estudo de introdução apresentado.

5.1 - Decisões sequenciais

Uma das tarefas que aprendemos nesta jornada de estudo foi que computadores tomam decisões a partir do uso dos comandos **If/EndIf** com auxílio do comando **Else** quando a tomada de decisão efetua ação composta. No entanto, há ainda mais um componente de tomada de decisão que pode ser usado com os comandos **If/EndIf**, trata-se do comando **ElseIf** que facilita o estabelecimento de sequenciais de decisão.

Para vermos este "novo" comando vamos considerar duas situações uma sem o uso de **ElseIf** e o mesmo caso com uso de **ElseIf**. Vejamos um programa que a partir da entrada do número de um mês entre "1" e "12" apresente o nome do mês por extenso.

O primeiro programa dessa situação faz uso de uma solução baseada em tomadas de decisão sequenciais com **If/EndIf** e o segundo programa faz uso de uma solução baseada na estrutura de comando **If/Else/EndIf**.

Veja o primeiro exemplo, atente para a parte em vermelho:

```
1 TextWindow.Clear()
2 TextWindow.WriteLine("Entre o número de um mês (entre '1' e '12'): ")
3 Mes = TextWindow.ReadNumber()
4 If (Mes < 1 Or Mes > 12) Then
5   TextWindow.WriteLine("Mês inválido")
6 Else
7   If (Mes = 1) Then
8     TextWindow.WriteLine("Janeiro")
9   EndIf
10  If (Mes = 2) Then
11    TextWindow.WriteLine("Fevereiro")
12  EndIf
13  If (Mes = 3) Then
14    TextWindow.WriteLine("Março")
15  EndIf
16  If (Mes = 4) Then
17    TextWindow.WriteLine("Abril")
18  EndIf
19  If (Mes = 5) Then
20    TextWindow.WriteLine("Maio")
21  EndIf
22  If (Mes = 6) Then
23    TextWindow.WriteLine("Junho")
24  EndIf
```

```
25 If (Mes = 7) Then
26     TextWindow.WriteLine("Julho")
27 EndIf
28 If (Mes = 8) Then
29     TextWindow.WriteLine("Agosto")
30 EndIf
31 If (Mes = 9) Then
32     TextWindow.WriteLine("Setembro")
33 EndIf
34 If (Mes = 10) Then
35     TextWindow.WriteLine("Outubro")
36 EndIf
37 If (Mes = 11) Then
38     TextWindow.WriteLine("Novembro")
39 EndIf
40 If (Mes = 12) Then
41     TextWindow.WriteLine("Dezembro")
42 EndIf
43 EndIf
```

Grave o programa com o nome "**Cap05Ex01**" e coloque-o em execução. Entre os valores de "**1**" a "**12**" e veja todas as saídas apresentadas. Entre também valores fora da faixa de aceitação.

Veja o segundo exemplo, atente para a parte em verde e para a quantidade de linhas que este programa tem em relação ao primeiro. Perceba que o comando **ElseIf** permite uma economia de código muito atraente:

```
1 TextWindow.Clear()
2 TextWindow.Write("Entre o número de um mês (entre '1' e '12'): ")
3 Mes = TextWindow.ReadNumber()
4 If (Mes < 1 Or Mes > 12) Then
5     TextWindow.WriteLine("Mês inválido")
6 Else
7     If (Mes = 1) Then
8         TextWindow.WriteLine("Janeiro")
9     ElseIf (Mes = 2) Then
10        TextWindow.WriteLine("Fevereiro")
11    ElseIf (Mes = 3) Then
12        TextWindow.WriteLine("Março")
13    ElseIf (Mes = 4) Then
14        TextWindow.WriteLine("Abril")
15    ElseIf (Mes = 5) Then
16        TextWindow.WriteLine("Maio")
17    ElseIf (Mes = 6) Then
18        TextWindow.WriteLine("Junho")
19    ElseIf (Mes = 7) Then
20        TextWindow.WriteLine("Julho")
21    ElseIf (Mes = 8) Then
22        TextWindow.WriteLine("Agosto")
23    ElseIf (Mes = 9) Then
24        TextWindow.WriteLine("Setembro")
```

```
25 | ElseIf (Mes = 10) Then
26 |     TextWindow.WriteLine("Outubro")
27 | ElseIf (Mes = 11) Then
28 |     TextWindow.WriteLine("Novembro")
29 | ElseIf (Mes = 12) Then
30 |     TextWindow.WriteLine("Dezembro")
31 | EndIf
32 | EndIf
```

Grave o programa com o nome "**Cap05Ex02**" e coloque-o em execução. Entre os valores de "1" a "12" e veja todas as saídas apresentadas. Entre também valores fora da faixa de aceitação.

Bom, agora a partir desta visão temos o conjunto completo e essencial de comandos para desenvolver diversas aplicações. Vale relembrar que um programa de computador é um componente que opera segundo três ações:

1. Entrada de dados;
2. Processamento de dados (matemático e/ou lógico);
3. Saída de dados.

As ações de entrada e saída podem ocorrer de diversas formas e isso foi demonstrado em várias vezes neste livro; a ação de processamento matemático ocorre a partir do uso de operadores aritméticos ("+", "-", "*" e "/") e operação matemáticas a partir do objeto **Math** e a ação de processamento lógico ocorre a partir do uso dos operadores relacionais ("=,>,<,>=,<=" e "<>") e operadores lógicos ("**And**" e "**Or**") operacionalizados com os comandos **If/Then/Else/ElseIf/EndIf** e **While**.

Vejamos então um programa calculadora que a partir de um menu apresente a possibilidade de se efetuar as quatro operações entre dois números fornecidos. Veja que este simples programa em modo texto praticamente utiliza quase tudo que foi apresentado anteriormente (entrada, processamento, saída, sub-rotinas, repetições e decisões) e a "novo" operação **End()** do objeto **Program** que tem por finalidade encerrar sumariamente a execução de um programa:

```
1 Sub Pausa
2     TextWindow.Write("Tecle <Enter> para voltar ao menu... ")
3     Enter = TextWindow.Read()
4 EndSub
5
6 Sub Entrada
7     TextWindow.WriteLine("")
8     TextWindow.Write("Entre o 1o. valor numérico: ")
9     A = TextWindow.ReadNumber()
10    TextWindow.Write("Entre o 2o. valor numérico: ")
11    B = TextWindow.ReadNumber()
12 EndSub
13
14 Sub Saida
15     TextWindow.WriteLine("")
16     TextWindow.WriteLine("Resultado = " + R)
17     TextWindow.WriteLine("")
18     Pausa()
19 EndSub
20
```

```
21 Sub Titulo
22   TextWindow.Clear()
23   TextWindow.WriteLine("PROGRAMA: CALCULADORA")
24   TextWindow.WriteLine("-----")
25   TextWindow.WriteLine("")
26 EndSub
27
28 Sub Adicao
29   Titulo()
30   TextWindow.WriteLine("CÁLCULO DE ADIÇÃO")
31   Entrada()
32   R = A + B
33   Saída()
34 EndSub
35
36 Sub Subtracao
37   Titulo()
38   TextWindow.WriteLine("CÁLCULO DE SUBTRAÇÃO")
39   Entrada()
40   R = A - B
41   Saída()
42 EndSub
43
44 Sub Multiplicacao
45   Titulo()
46   TextWindow.WriteLine("CÁLCULO DE MULTIPLICAÇÃO")
47   Entrada()
48   R = A * B
49   Saída()
50 EndSub
51
52 Sub Divisao
53   Titulo()
54   TextWindow.WriteLine("CÁLCULO DE DIVISÃO")
55   Entrada()
56   If (B = 0) Then
57     TextWindow.WriteLine("")
58     TextWindow.WriteLine("Resultado = ERRO")
59     TextWindow.WriteLine("")
60     Pausa()
61   Else
62     R = A / B
63     Saída()
64   EndIf
65 EndSub
66
67 Opcão = 9
68 While (Opcão <> 5)
69   Titulo()
70   TextWindow.WriteLine("[1] - Adição")
```

```
71  TextWindow.WriteLine("[2] - Subtração")
72  TextWindow.WriteLine("[3] - Multiplicação")
73  TextWindow.WriteLine("[4] - Divisão")
74  TextWindow.WriteLine("[5] - Fim de programa")
75  TextWindow.WriteLine("")
76  TextWindow.Write("Entre sua opção: ")
77  Opcao = TextWindow.ReadNumber()
78  If (Opcao >= 1 And Opcao <= 5) Then
79    If (Opcao = 1) Then
80      Adicao()
81    ElseIf (Opcao = 2) Then
82      Subtracao()
83    ElseIf (Opcao = 3) Then
84      Multiplicacao()
85    ElseIf (Opcao = 4) Then
86      Divisao()
87    EndIf
88  Else
89    TextWindow.WriteLine("")
90    TextWindow.WriteLine("Opção inválida - tente novamente.")
91    TextWindow.WriteLine("")
92    Pausa()
93  EndIf
94 EndWhile
95 Program.End()
```

Grave o programa com o nome "**Cap05Ex03**" e coloque-o em execução. Entre as opções indicadas no menu e realize todos os testes desejados. Teste entradas de opções invalidas e veja o comportamento do programa.

5.2 - Variáveis indexadas

Variáveis indexadas, também são chamadas de *variáveis compostas*, *variáveis subscritas*, *vetores*, *listas*, *arranjos*, *tabelas*, *matrizes* e etc. Este recurso tem a capacidade de facilitar o armazenamento e controle de uma quantidade grande de dados.

Por exemplo, imagine que precisamos armazenar na memória dez nomes de pessoas. É óbvio que vamos precisar de cinco variáveis, mas isso começa a ficar problemático, pois e se tivermos 10, 100, 1.000, 10.000 nomes ou mais? É aqui que entra o recurso de uso de variáveis indexadas, onde é possível definir uma única variável que aceite mais de um dado controlado por um índice de acesso.

Para exemplificar melhor vejamos dois exemplos: o primeiro programa recebe a entrada de dez nomes e os apresenta em seguida com o uso de *variáveis simples*; já o segundo programa recebe a entrada e fornece a saída de cinco nomes por meio de *variáveis indexadas*.

Veja o primeiro exemplo de entrada e saída de cinco nomes com o uso de *variáveis simples*:

```
1  TextWindow.Clear()
2
3  TextWindow.Write("Entre o 1o. nome: ")
4  Nome1 = TextWindow.Read()
5  TextWindow.Write("Entre o 2o. nome: ")

---


```

```
6 | Nome2 = TextWindow.Read()
7 | TextWindow.WriteLine("Entre o 3o. nome: ")
8 | Nome3 = TextWindow.Read()
9 | TextWindow.WriteLine("Entre o 4o. nome: ")
10| Nome4 = TextWindow.Read()
11| TextWindow.WriteLine("Entre o 5o. nome: ")
12| Nome5 = TextWindow.Read()
13|
14| TextWindow.WriteLine("")
15| TextWindow.WriteLine("1o. nome: " + Nome1)
16| TextWindow.WriteLine("2o. nome: " + Nome2)
17| TextWindow.WriteLine("3o. nome: " + Nome3)
18| TextWindow.WriteLine("4o. nome: " + Nome4)
19| TextWindow.WriteLine("5o. nome: " + Nome5)
20| TextWindow.WriteLine("")
21| TextWindow.WriteLine("Tecle <Enter> para encerrar... ")
22| Enter = TextWindow.Read()
23| Program.End()
```

Grave o programa com o nome "**Cap05Ex04**" e coloque-o em execução. Entre os nomes solicitados e os veja sendo apresentados.

Observe no código a indicação das variáveis **Nome1**, **Nome2**, **Nome3**, **Nome4** e **Nome5** grafadas em vermelho como representantes de *variáveis simples*.

Veja o primeiro exemplo de entrada e saída de cinco nomes com o uso de *variáveis simples*:

```
1 | TextWindow.Clear()
2 |
3 | For I = 1 To 5
4 |   TextWindow.WriteLine("Entre o " + I + "o. nome: ")
5 |   Nome[I] = TextWindow.Read()
6 | EndFor
7 |
8 | TextWindow.WriteLine("")
9 | For J = 1 To 5
10|   TextWindow.WriteLine(J + "o. nome: " + Nome[J])
11| EndFor
12|
13| TextWindow.WriteLine("")
14| TextWindow.WriteLine("Tecle <Enter> para encerrar... ")
15| Enter = TextWindow.Read()
16| Program.End()
```

Grave o programa com o nome "**Cap05Ex05**" e coloque-o em execução. Entre os nomes solicitados e os veja sendo apresentados.

Veja que no programa exemplo "**Cap05Ex04**" são usadas cinco variáveis explicitamente chamadas: **Nome1**, **Nome2**, **Nome3**, **Nome4** e **Nome5** e no programa exemplo "**Cap05Ex05**" em que são usadas apenas uma variável chamada **Nome[I]** (variável **Nome** índice **I** na entrada) ou **Nome[J]** (variável **Nome** índice **J** na saída), indicando uma variável dimensionada e com a capacidade de armazenar dinamicamente a quantidade que estiver estabelecida, por exemplo, em

um comando **For** como ocorre nas linhas "3" e "8" com a contagem de vezes entre "1" e "5". Veja também o tamanho em linhas dos programas e note que para fazermos a mesma coisa não foi preciso a quantidade de linha do programa "**Cap05Ex04**".

Para um exemplo prático vamos considerar um programa que efetue a leitura do número correspondente a um mês do calendário de "1" a "12" e apresente o nome por extenso do mês. Observe o código a seguir:

```
1 TextWindow.Clear()
2
3 Meses[ 1] = "Janeiro"
4 Meses[ 2] = "Fevereiro"
5 Meses[ 3] = "Março"
6 Meses[ 4] = "Abril"
7 Meses[ 5] = "Maio"
8 Meses[ 6] = "Junho"
9 Meses[ 7] = "Julho"
10 Meses[ 8] = "Agosto"
11 Meses[ 9] = "Setembro"
12 Meses[10] = "Outubro"
13 Meses[11] = "Novembro"
14 Meses[12] = "Dezembro"
15
16 TextWindow.Write("Entre o número de um mês (entre '1' e '12'): ")
17 Mes = TextWindow.ReadNumber()
18 If (Mes < 1 Or Mes > 12) Then
19   TextWindow.WriteLine("Mês inválido")
20 Else
21   TextWindow.WriteLine(Meses[Mes])
22 EndIf
23
24 TextWindow.WriteLine("")
25 TextWindow.Write("Tecle <Enter> para encerrar... ")
26 Enter = TextWindow.Read()
27 Program.End()
```

Grave o programa com o nome "**Cap05Ex06**" e coloque-o em execução. Entre os números dos meses e veja seu funcionamento. Veja que com a técnica de variáveis indexadas é possível criar na memória tabelas de valores.

O uso de variáveis indexadas em "*Small Basic*" permite que além de índices numéricos sejam usados índices alfabéticos (e mesmo alfanuméricos). Veja o próximo programa exemplo:

```
1 TextWindow.Clear()
2
3 TextWindow.Write("Entre seu nome .....: ")
4 Pessoa["Nome"] = TextWindow.Read()
5 TextWindow.Write("Entre sua idade .....: ")
6 Pessoa["Idade"] = TextWindow.Read()
7 TextWindow.Write("Entre sua escolaridade ...: ")
8 Pessoa["Escolaridade"] = TextWindow.Read()
9
```

```
10 TextWindow.WriteLine("")  
11 TextWindow.WriteLine("Nome .....: " + Pessoa["Nome"])  
12 TextWindow.WriteLine("Idade .....: " + Pessoa["Idade"])  
13 TextWindow.WriteLine("Escolaridade ...: " + Pessoa["Escolaridade"])  
14  
15 TextWindow.WriteLine("")  
16 TextWindow.Write("Escolha um índice - Nome, Idade, Escolaridade: ")  
17 Indice = TextWindow.Read()  
18  
19 TextWindow.WriteLine("")  
20 TextWindow.Write("Você está vendo o dado '" + Pessoa[Indice] + "'")  
21 TextWindow.WriteLine(" do índice [" + Indice + "]")  
22  
23 TextWindow.WriteLine("")  
24 TextWindow.Write("Tecle <Enter> para encerrar... ")  
25 Enter = TextWindow.Read()  
26 Program.End()
```

Grave o programa com o nome "**Cap05Ex07**" e coloque-o em execução. Entre os dados solicitados e após serem apresentados escolha um campo a ser visto.

Os exemplos indicados estão usando variáveis indexadas de uma única dimensão na forma de listas de valores. No entanto, é possível com o acréscimo de mais uma dimensão fazer uso de estruturas na forma de tabelas como tradicionais matrizes. Para experimentar este efeito vamos considerar um programa que estabeleça uma tabela em memória com três linhas e quatro colunas que aceite valores numéricos e apresente o resultado dos valores informados multiplicados por dois sem guarda-los em memória. Veja o código seguinte:

```
1 TextWindow.WriteLine("Matriz de duas dimensões")  
2 TextWindow.WriteLine("")  
3 For I = 1 To 3  
4     TextWindow.WriteLine("Linha ...: " + I)  
5     For J = 1 To 4  
6         TextWindow.Write("Coluna ...: " + J + " --> ")  
7         Valor[I][J] = TextWindow.ReadNumber()  
8     EndFor  
9 EndFor  
10 TextWindow.WriteLine("")  
11 For I = 1 To 3  
12     For J = 1 To 4  
13         TextWindow.Write ("Valor[" + I + "," + J + "]: ")  
14         TextWindow.WriteLine(Valor[I][J] * 2)  
15     EndFor  
16 EndFor  
17 TextWindow.WriteLine("")  
18 TextWindow.Write("Tecle <Enter> para encerrar... ")  
19 Enter = TextWindow.Read()  
20 Program.End()
```

Grave o programa com o nome "**Cap05Ex08**" e coloque-o em execução. Entre os dados solicitados e veja o resultado dos cálculos apresentados.

5.3 - Manipulação de texto

Um dos pontos tratados em linguagens de programação não são só as ações de processamento numérico ou lógico, mas a manipulação de textos. Por exemplo, houve ocorrência de exemplo neste livro em que foi usada a operação **ConvertToUpperCase()** do objeto **Text**: converter para o formato maiúsculo uma sequência de caracteres foi uma ação de manipulação de texto. Vamos agora conhecer com mais profundidade outras operações do objeto **Text**.

Vejamos um programa que faz a concatenação das palavras "COMPU" e "TADOR" sem o operador "+" muito utilizado. Além de fazer a concatenação o programa mostra a quantidade de caracteres da palavra formada.

```

1 Palavra = Text.Append("COMPU", "TADOR")
2 TextWindow.WriteLine("A palavra " + Palavra)
3 TextWindow.WriteLine(" possui " + Text.GetLength(Palavra))
4 TextWindow.WriteLine(" caracteres.")
5 TextWindow.WriteLine("")
6 TextWindow.WriteLine("Tecle <Enter> para encerrar... ")
7 Enter = TextWindow.Read()
8 Program.End()

```

Grave o programa com o nome "**Cap05Ex09**", coloque-o em execução e veja na figura 5.1 o resultado da palavra concatenada com a quantidade de "10" caracteres retorna com a operação **GetLength()**.

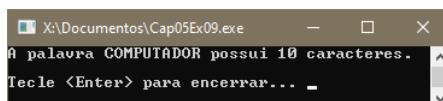


Figura 5.1 - Resultado da ação de concatenação e contagem de caracteres

Vejamos no próximo exemplo a definição de uma palavra formada com caracteres maiúsculos e minúsculos sendo apresentada totalmente em formato maiúsculo e totalmente em formato minúsculo.

```

1 Palavra = "CoMpUTadOr"
2 TextWindow.WriteLine("Maiúsculo: " + Text.ConvertToUpperCase(Palavra))
3 TextWindow.WriteLine("Minúsculo: " + Text.ConvertToLowerCase(Palavra))
4 TextWindow.WriteLine("")
5 TextWindow.WriteLine("Tecle <Enter> para encerrar... ")
6 Enter = TextWindow.Read()
7 Program.End()

```

Grave o programa com o nome "**Cap05Ex10**", coloque-o em execução e veja na figura 5.2 o resultado da palavra "CoMpUTadOr" escrita em formatos maiúsculo e minúsculo respectivamente com as operações **ConvertToUpperCase()** e **ConvertToLowerCase()**.

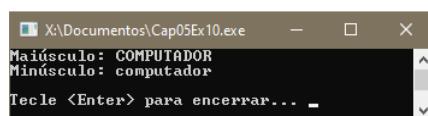


Figura 5.2 - Palavra em maiúsculo e minúsculo

Outro exemplo curioso que podemos realizar é a identificação do código ASCII (*American Standard Code for Information Interchange*) ou Unicode, que são os padrões dos caracteres suportados em um computador. Este código é quem estabelece os valores internos de cada caractere

que o computador processa, tendo valor internacional e usado por toda a indústria de desenvolvimento de equipamentos para a área da computação. Veja na figura 5.3 a tabela completa com a descrição de seus caracteres.

ASCII TABLE											
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	:	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figura 5.3 - Palavra em maiúsculo e minúsculo

<https://pt.wikipedia.org/wiki/Ficheiro:ASCII-Table-wide.svg><https://commons.wikimedia.org/wiki/File:ASCII-Table.svg>

Veja o próximo programa que pede a entrada de um valor numérico entre o decimal 32 e 126 (não checando a validação da entrada) e apresenta o caractere ASCII correspondente e pede a entrada de um caractere e mostra o código ASCII do caractere informado.

```

1 TextWindow.WriteLine("Entre valor ASCII (32 a 126): ")
2 CodASCII = TextWindow.ReadNumber()
3 TextWindow.WriteLine(CodASCII + " = " + Text.GetCharacter(CodASCII))
4 TextWindow.WriteLine("Entre caractere: ")
5 Caract = TextWindow.Read()
6 TextWindow.WriteLine(Caract + " = " + Text.GetCharacterCode(Caract))
7 TextWindow.WriteLine("Tecle <Enter> para encerrar... ")
8 Enter = TextWindow.Read()
9 Program.End()
```

Grave o programa com o nome "**Cap05Ex11**", coloque-o em execução e veja na figura 5.4 o resultado da apresentação da entrada do código ASCII "97" que retorna o caractere "a" a partir da operação **GetCharacter()** e a entrada do caractere "A" que retorna o valor "65" a partir da operação **GetCharacterCode()**.

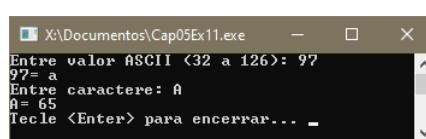


Figura 5.4 - Código ASCII

Outro recurso é a localização lógica de partes de um subtexto em um texto maior a partir das operações **StartsWith()** e **EndsWith()** do objeto **Text**. Essas duas operações são idênticas e adequadas para operarem respectivamente das extremidades esquerda e direita de um texto ou palavra verificando se o subtexto indicado faz ou não parte do texto maior.

O programa seguinte mostra se subtextos da palavra "**PARALELEPIPEDO**" fazem ou não parte de seu contexto.

```

1 Palavra = "PARALELEPIPEDO"
2 Parte1  = "PARA"
3 Parte2  = "DO"
4 TextWindow.WriteLine(Text.StartsWith(Palavra, Parte1))
5 TextWindow.WriteLine(Text.StartsWith(Palavra, Parte2))
6 TextWindow.WriteLine(Text.EndsWith(Palavra, Parte1))
7 TextWindow.WriteLine(Text.EndsWith(Palavra, Parte2))
8 TextWindow.Write("Tecle <Enter> para encerrar... ")
9 Enter = TextWindow.Read()
Program.End()

```

Grave o programa com o nome "**Cap05Ex12**", coloque-o em execução e veja na figura 5.5 o resultado de sua apresentação.



Figura 5.5 - Partes de um texto maior

Por serem as operações **StartsWith()** e **EndsWith()** do tipo lógicas elas retornam os resultados **False** (falso) ou **True** (verdadeiro) se a ocorrência indicada em seu segundo parâmetro faz parte do primeiro parâmetro.

Outro efeito que podemos usar é a localização da posição dentro de um texto que certa ocorrência acontece a partir do uso da operação **GetIndexOf()**. Por exemplo, saber a partir de que posição o subtexto "**LELE**" da palavra "**PARALELEPIPEDO**" efetivamente inicia-se. Veja o código:

```

1 Palavra = "PARALELEPIPEDO"
2 Parte   = "LELE"
3 Posicao = Text.GetIndexOf(Palavra, Parte)
4 TextWindow.WriteLine("A ocorrência LELE está na posição " + Posicao)
5 TextWindow.Write("Tecle <Enter> para encerrar... ")
6 Enter = TextWindow.Read()
7 Program.End()

```

Grave o programa com o nome "**Cap05Ex13**", coloque-o em execução e veja na figura 5.6 o resultado de sua execução.

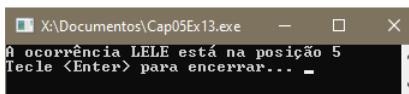


Figura 5.6 - Localização de texto

Se não for possível a operação **GetIndexOf()** localizar a posição em que certo subtexto está em um texto maior retornará como resposta o valor "**0**".

Além da operação **GetIndexOf()** há a operação **IsSubText()** do tipo lógica que retorna **True** ou **False** caso o subtexto exista ou não no texto maior. Veja seu exemplo:

```

1 Palavra = "PARALELEPIPEDO"
2 Parte   = "LELE"
3 TextWindow.WriteLine(Text.IsSubText(Palavra, Parte))
4 TextWindow.Write("Tecla <Enter> para encerrar... ")
5 Enter = TextWindow.Read()
6 Program.End()
    
```

Grave o programa com o nome "**Cap05Ex14**", coloque-o em execução e veja a indicação do resultado "**True**" pelo fato de "**LELE**" estar dentro de "**PARALELEPIPEDO**".

Vimos até aqui, praticamente, todas as operações do objeto **Text** faltando apenas duas operações usadas para trabalhar na extração de subtextos de um texto maior. Para tanto, considere as operações **GetSubText()** e **GetSubTextToEnd()** de certa forma parecidas.

A operação **GetSubText()** retorna as partes de um subtexto de um texto indicado a partir de três parâmetros: primeiro a indicação do texto, segundo a posição a partir da qual se deseja extrair o subtexto e o terceiro a indicação da quantidade de caracteres a ser extraído do texto. Veja o exemplo seguinte:

```

1 Palavra = "COMPUTADOR"
2 TextWindow.WriteLine(Text.GetSubText(Palavra, 1, 3))
3 TextWindow.WriteLine(Text.GetSubText(Palavra, 4, 4))
4 TextWindow.WriteLine(Text.GetSubText(Palavra, 8, 4))
5 TextWindow.Write("Tecla <Enter> para encerrar... ")
6 Enter = TextWindow.Read()
7 Program.End()
    
```

Grave o programa com o nome "**Cap05Ex15**", coloque-o em execução e veja na figura 5.7 o resultado de sua execução.

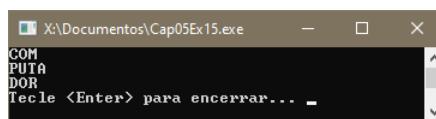


Figura 5.7 - Subtextos de um texto

A operação **GetSubTextToEnd()** se parece, em parte, com a operação **GetSubText()** com a diferença que sua extração ocorre da posição indicada até o final do texto. Esta operação poderá ser substituída pela ação: **Text.GetSubText(texto, posição, Text.GetLength(texto))**. Veja exemplos de uso dessas operações:

```

1 Palavra = "COMPUTADOR"
2 For I = 1 To Text.GetLength(Palavra)
3     TextWindow.WriteLine(Text.GetSubTextToEnd(Palavra, I))
4 EndFor
5 TextWindow.Write("Tecla <Enter> para encerrar... ")
6 Enter = TextWindow.Read()
7 Program.End()
    
```

Grave o programa com o nome "**Cap05Ex16**", coloque-o em execução e veja na figura 5.8 o resultado de sua execução.

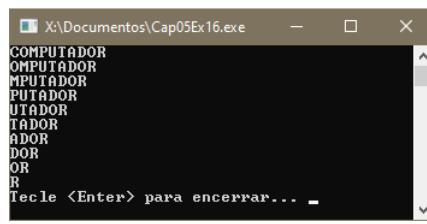


Figura 5.8 - Efeito pirâmide invertida

Para um teste substitua na linha "3" o trecho "**Text.GetSubTextToEnd(Palavra, I)**" indicado em "**TextWindow.WriteLine()**" por "**Text.GetSubText(Palavra, I, Text.GetLength(Palavra))**" e observe a similaridade no resultado apresentado.

Agora como exemplo vamos considerar apresentar a mensagem em forma de pirâmide em sentido contrário. Observe os detalhes no código a seguir:

```
1 Palavra = "COMPUTADOR"
2 Tamanho = Text.GetLength(Palavra)
3 For I = 1 To Tamanho
4   TextWindow.WriteLine(Text.GetSubTextToEnd(Palavra, (Tamanho + 1) - I))
5 EndFor
6 TextWindow.Write("Tecle <Enter> para encerrar... ")
7 Enter = TextWindow.Read()
8 Program.End()
```

Grave o programa com o nome "**Cap05Ex17**", coloque-o em execução e veja na figura 5.9 o resultado de sua execução.

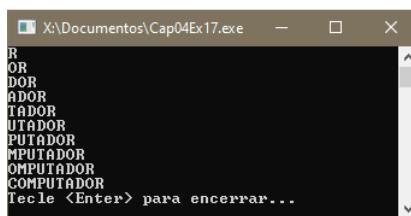


Figura 5.9 - Efeito pirâmide normal

Observe atentamente os detalhes indicados nas linhas "2", "3" e "4", principalmente na definição da variável "**Tamanho**" e como essa variável está sendo utilizada nas partes indicadas em ocre.

Vamos considerar um programa que mostre a pirâmide da palavra incisando-a da esquerda para a direita e não da direita para a esquerda. Neste caso a operação **GetSubTextToEnd()** deve ser substituída pela operação **GetSubText()**. Veja o código a seguir:

```
1 Palavra = "COMPUTADOR"
2 Tamanho = Text.GetLength(Palavra)
3 For I = 1 To Tamanho
4   TextWindow.WriteLine(Text.GetSubText(Palavra, 1, (Tamanho + 1) - I))
5 EndFor
6 TextWindow.Write("Tecle <Enter> para encerrar... ")
7 Enter = TextWindow.Read()
8 Program.End()
```

Grave o programa com o nome "**Cap05Ex18**", coloque-o em execução e veja na figura 5.10 o resultado de sua execução.

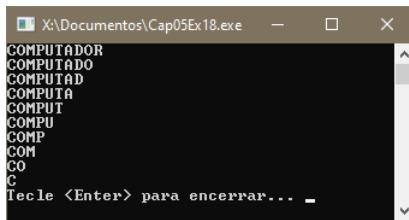


Figura 5.10 - Efeito pirâmide invertida com texto a esquerda

Veja neste próximo exemplo a apresentação do texto no formato piramidal iniciando a esquerda e se estendendo a direita com o uso de repetição com contagem decrescente:

```

1 Palavra = "COMPUTADOR"
2 Tamanho = Text.GetLength(Palavra)
3 For I = Tamanho To 1 Step -1
4     TextWindow.WriteLine(Text.GetSubText(Palavra, 1, (Tamanho + 1) - I))
5 EndFor
6 TextWindow.Write("Tecle <Enter> para encerrar... ")
7 Enter = TextWindow.Read()
8 Program.End()

```

Grave o programa com o nome "**Cap05Ex19**", coloque-o em execução e veja na figura 5.11 o resultado de sua execução.

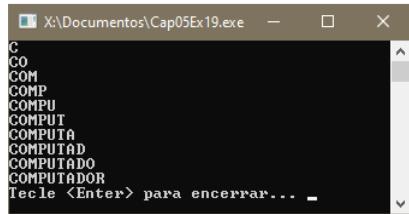


Figura 5.11 - Efeito pirâmide normal com texto a esquerda

Observe que os efeitos de manipulação de texto em formato piramidal tiveram pequenas ocorrência de mudanças em certos pontos dos códigos dos programas apresentados. No entanto, todos os exemplos indicam meras demonstrações despretensiosas. Para um entendimento contextualizado destes recursos considere um programa que receba a entrada de um verbo regular da primeira conjugação (terminados em "ar") e o presente conjugado no tempo presente do indicativo. Para esta proposta buscou-se fazer uso de alguns dos recursos textuais apresentados ao longo deste livro, até então. Veja o código:

```

1 P[1] = "Eu "
2 P[2] = "Tu "
3 P[3] = "Ele(a) "
4 P[4] = "Nós "
5 P[5] = "Eu "
6 P[6] = "Eles(as) "
7
8 T[1] = "o"
9 T[2] = "as"
10 T[3] = "a"
11 T[4] = "amos"
12 T[5] = "ais"
13 T[6] = "am"

```

```

14
15 TextWindow.WriteLine("Entre um verbo regular: ")
16 Verbo = Text.ConvertToLowercase(TextWindow.Read())
17
18 Tamanho = Text.GetLength(Verbo)
19 Radical = Text.GetSubText(Verbo, 1, Tamanho - 2)
20 Terminacao = Text.GetSubTextToEnd(Verbo, Tamanho - 1)
21
22 TextWindow.WriteLine("")
23 If (Terminacao = "ar") Then
24   For I = 1 To 6
25     TextWindow.WriteLine(P[I] + Radical + T[I])
26   EndFor
27 Else
28   TextWindow.WriteLine("O verbo fornecido não é válido.")
29 EndIf
30
31 TextWindow.WriteLine("")
32 TextWindow.Write("Tecle <Enter> para encerrar... ")
33 Enter = TextWindow.Read()
34 Program.End()

```

Grave o programa com o nome "**Cap05Ex20**", coloque-o em execução e veja na figura 5.12 o resultado de sua execução.

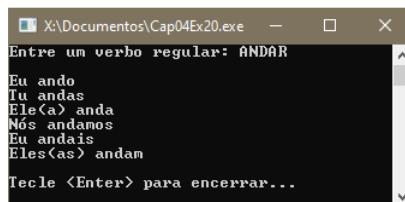


Figura 5.12 - Conjugação de verbo regular na primeira conjugação no presente do indicativo

Veja neste exemplo que a partir de alguns efeitos de processamento matemático e lógico com o uso de operações especializadas na manipulação de textos fez-se o processamento de sequências de caracteres alfabéticos.

Entre as linhas "1" até "6" e "8" até "13" faz-se a definição das variáveis indexadas "P" para as pessoas verbais e "T" para as terminações verbais para os verbos regulares de primeira conjugação no presente do indicativo.

Na linha "16" faz-se a conversão da entrada do verbo para o formato minúsculo. Veja o uso da operação **Text.ConvertToLowercase()**.

Na linha "18" é realizado o cálculo do tamanho do verbo. Esse cálculo auxiliar as ações das linhas "19" e "20" na separação do radical do verbo e de sua terminação com o uso respectivo das operações **GetSubText()** e **GetSubTextToEnd()**.

Entre as linhas "23" e "29" faz-se o tratamento para a apresentação da conjugação do verbo como planejado. Se o verbo não for válido ocorre a execução da linha "28", caso seja válido ocorre a execução do trecho de repetição entre as linhas "24" e "26".

5.4 - Efeitos sonoros

A linguagem "*Small Basic*" possui diversos recursos e entre eles há a possibilidade de se fazer a criação de sons como efeitos sonoros e música e para esta finalidade há na linguagem a operação **PlayMusic()** do objeto **Sound**. O objeto **Sound** possui um conjunto de operações que executam sons e músicas, além de tocar arquivos dos tipos MP3, WAV ou WMA.

Por ser este um livro de introdução a linguagem "*Small Basic*" não serão vistos todos estes recursos, apenas os principais.

Veja na sequência um programa que tocas os sons de sino, alarme e clique:

```
1 Sound.PlayBellRingAndWait() ' Toca sino  
2 Sound.PlayChimeAndWait()    ' Toca alarme 1  
3 Sound.PlayChimesAndWait()   ' Toca alarme 2  
4 Sound.PlayClickAndWait()    ' Toca clique
```

Grave o programa com o nome "**Cap05Ex21**", coloque-o em execução e ouça os efeitos sonoros apresentados.

Para criar e tocar músicas com a linguagem "*Small Basic*" é necessário que você tenha conhecimento sobre teoria musical e tenha certa intimidade com os sons das notas musicais. Neste livro faz-se apenas uma pequena introdução desprestensiosa ao tema.

Considere que para escrever uma música é necessário combinar notas musicais. Isto significa, grosso modo, que pode-se considerar música um conjunto de sons e silêncios organizados de forma melódica e harmoniosa a partir da marcação de certo ritmo. Melodia é o que pode ser cantado, é a voz da música. Já a harmonia é a sobreposição das notas de modo que se tenha uma base para a melodia e o ritmo é a marcação do tempo de uma música. Veja que a linguagem "*Small Basic*" por meio da operação **PlayMusic()** do objeto **Sound** aproxima-se da definição da harmonia. Quanto a melodia? Esta fica por sua conta.

Uma nota musical é o elemento mínimo que compõe uma música podendo ser representada por letras ou por partitura. Veja o padrão de representação de notas musicais por letras junto a tabela 5.1.

LETRA	NOTA MUSICAL
C	DÓ
D	RÉ
E	MÍ
F	FÁ
G	SOL
A	LÁ
B (ou H em alemão)	SI

Figura 5.1 - Representação das notas musicais

Veja que são "7" as notas musicais. Essas notas são chamadas de *notas naturais*. No entanto, o conjunto de notas é composto por outras notas chamadas *notas acidentadas* que são notas que possuem a alteração na altura de seu tom em meio toma acima ou meio toma abaixo, formando um conjunto de mais "5" notas, perfazendo um total de "12" notas. As notas "MI (E)" e "SI (B)" não possuem acidentes. Veja essa ocorrência junto a tabela 5.2.

MEIO TOM ABAIXO (b)	NOTA MUSICAL	MEIO TOM ACIMA (#)
-	C	C#
D _b	D	D#
E _b	E	-
-	F	F#
G _b	G	G#
A _b	A	A#
B _b	B	-

Figura 5.2 - Conjunto das notas naturais e acidentadas

Note que a nota *sustenido* de um tom é igual a nota *bemol* do tom de uma nota anterior e vice versa, sendo estas, de fato, as mesmas notas: C# = D_b, D# = E_b, F# = G_b, G# = A_b e A# = B_b com notações diferentes. Devido a esta característica "Small Basic" pode usar sustenidos com os indicativos "+" ou "#" e bemóis com o indicativo "-".

Para ver esse conjunto de notas sendo tocado considere o programa seguinte que aceita a definição das notas dentro de uma cadeia alfanumérica delimitada entre aspas inglesas (não se preocupe com o código "04" na linha "1", isso será visto adiante):

```

1 Sound.PlayMusic("04")
2 Sound.PlayMusic("C")
3 Sound.PlayMusic("C#")
4 Sound.PlayMusic("D")
5 Sound.PlayMusic("D+")
6 Sound.PlayMusic("E")
7 Sound.PlayMusic("F")
8 Sound.PlayMusic("F#")
9 Sound.PlayMusic("G")
10 Sound.PlayMusic("G#")
11 Sound.PlayMusic("A")
12 Sound.PlayMusic("A+")
13 Sound.PlayMusic("B")

```

Grave o programa com o nome "**Cap05Ex22**", coloque-o em execução e ouça os efeitos sonoros apresentados.

Considerando a notação de sustenidos veja as frequências das notas graves, médias e agudas obtidas a partir de um piano (<https://www.intmath.com/trigonometric-graphs/music.php>) indicadas na tabela 5.3.

NOTAS	FREQUÊNCIA		
	AGUDO	MÉDIO	GRAVE
C	130.82	261.63	523.25
C# / Db	138.59	277.18	554.37
D	146.83	293.66	587.33
D# / Eb	155.56	311.13	622.25
E	164.81	329.63	659.26
F	174.61	349.23	698.46
F# / Gb	185.00	369.99	739.99
G	196.00	392.00	783.99
G# / Ab	207.65	415.30	830.61
A	220.00	440.00	880.00
A# / Bb	233.08	466.16	932.33
B	246.94	493.88	987.77

Figura 5.3 - Frequências das notas musicais

Além das notas musicais indicadas a operação **PlayMusic()** faz uso de outros elementos de configuração essenciais sempre à frente da nota a ser tocada como o código: "O" que especifica a oitava a ser tocada; "L" que define a duração do tom a ser tocado e "R" ou "P" que estabelece um tempo de silêncio da música.

O código "O" é usado a partir da indicação "On", onde "n" é um valor que determina a escala a ser usada, sendo aceitos valores entre "1" e "8", podendo-se também fazer uso das indicações "O>" e "O<" para respectivamente subir ou descer uma escala a partir da escala em uso. Em relação ao código "L" usa-se a indicação "Ln", onde "n" é um valor que determina a duração do tom a ser tocado, sendo aceitos valores entre "1" e "64", em que "1" é a duração máxima da nota. Os valores de "1" a "64" podem ser usados com os códigos "O", "L", "R" ou "P".

Para um teste de uso dos códigos "O" e "L" considere o programa seguinte que toca a sequência de notas "**DÓ-RÉ-MÍ-FÁ-FÁ-FÁ**" por duas vezes em diferentes configurações:

```

1 | Sound.PlayMusic("05 C2 D2 E2 F2 F3 F R2")
2 | Sound.PlayMusic("05 L2C L2D L2E L2F L3F L1F P2")

```

Grave o programa com o nome "**Cap05Ex23**", coloque-o em execução e ouça os efeitos sonoros apresentados.

Veja que tanto a linha "1" como linha "2" tocam a mesma sequência e na mesma tonalidade. A diferença está na forma de uso da definição das notas e da duração do tom. Veja que tanto faz fazer uso da nota como "C2" ou "L2C" ou mesmo usar as pausas como "R2" ou "P2".

A partir deste exemplo pode-se configurar a mesma nota a ser tocada de formas diferentes. Por exemplo, um **SOL bemol** pode ser tocado com a definição "**G-1**" ou com a definição "**L1G-**" ou um **DÓ sostenido** tocada ao tempo de 1/16, ou seja semicolcheia pode ser escrito "**L16C#**", "**C#16**" ou "**C+16**".

O próximo exemplo de programa toca a sequência de notas "**DÓ-RÉ-MÍ-FÁ-FÁ-FÁ**" entre as oito possíveis oitavas:

```
1 | For I = 1 To 8
2 |   TextWindow.WriteLine("DÓ-RÉ-MÍ-FÁ-FÁ-FÁ na oitava " + I)
3 |   Sound.PlayMusic("0" + I + "C2 D2 E2 F2 F3 F R2")
4 | EndFor
```

Grave o programa com o nome "**Cap05Ex24**", coloque-o em execução e ouça os efeitos sonoros apresentados.

A partir de uma noção geral de como produzir e configurar os sons gerados na linguagem é importante conhecer como fazer a composição de acordes. Um acorde é a composição de três ou mais notas que são tocadas ao mesmo tempo e de forma harmônica. A linguagem "*Small Basic*" não opera diretamente com a possibilidade de tocar acordes, mas é possível simulá-los a partir de um macete sutil.

Para um breve teste considere tocar um acorde de **LÁ maior** que é composto a partir das notas "A", "C#" e "E" (**LÁ+DÓ#+MÍ**), ou seja, um acorde maior é formado pela nota tônica (a própria nota), a terça maior desta nota e sua quinta justa. O próximo programa toca um acorde de **LÁ maior**. Veja o código:

```
1 | Tonica = "05 A1"    ' LÁ - Tônica
2 | Terca  = "05 C64+"  ' DÓ# - Terça maior
3 | Quinta = "05 E64"   ' MÍ - Quinta justa
4 | Acorde = Quinta + Terca + Tonica
5 | Sound.PlayMusic(Acorde)
```

Grave o programa com o nome "**Cap05Ex25**", coloque-o em execução e ouça os efeitos sonoros apresentados.

O macete para esta funcionalidade é definir os sons da tônica, da terça e da quinta separadamente. Os sons da terça e da quinta podem até ficar juntos na mesma cadeia, mas não faça isso com a cadeia da tônica.

Os sons da terça e da quinta são definidos com o valor menor de duração possível, neste caso, foi usado o valor "**64**", pode-se até aumentar a duração do tempo um pouco, mas não além de "**45**" e a tônica é definida com o tempo de duração desejado. É adequado que a terça e a quinta tenham sua duração sempre iguais.

Definidas as notas do acorde estas devem ser somadas na ordem "**Quinta + Terca + Tonica**" ou na ordem "**Terca + Quinta + Tonica**", mas não faça esta soma com a nota da "**Tonica**" a frente das demais, pois não surtirá o efeito desejado.

Fica como desafio a proposta de escrever um programa que toque a música de autoria desconhecida do folclore brasileiro "Marcha Soldado". Veja sua partitura básica na figura 5.13.

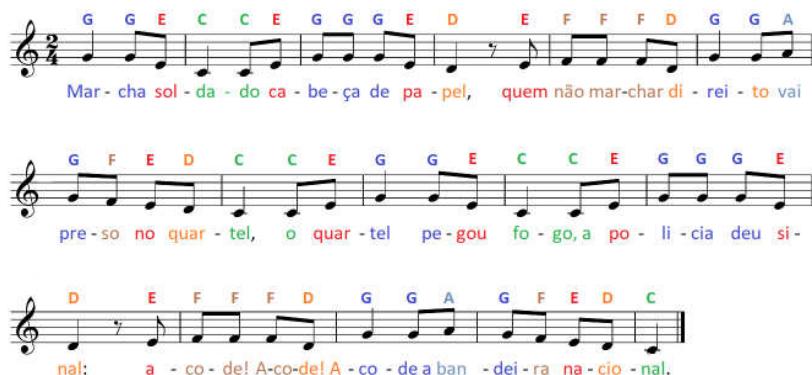


Figura 5.13 - Partitura música "Marcha Soldado"

5.5 - Programação gráfica

Uma das ações mais exploradas nesta obra foram os efeitos gráficos com o uso da geometria da tartaruga. Foram apresentados ao longo deste trabalho diversos recursos do objeto **Turtle** e alguns recursos do objeto **GraphicsWindow** complementando o objeto **Turtle**.

O objeto **GraphicsWindow** possui alguns outros recursos que serão apresentados neste tópico como a possibilidade de desenhar figuras geométricas, como: circunferências, elipses, triângulos, quadrados, retângulos, linhas, criar interfaces gráficas com o usuário e etc.

O primeiro detalhe para fazer uso dos recursos sobre uma janela gráfica é conhecer as operações **Show()** que apresenta uma tela gráfica e **Clear()** que limpa o conteúdo gráfico existente na tela gráfica (já indicada) do objeto **GraphicsWindow**. Desta forma, para apresentar uma janela gráfica considere o programa seguinte:

- ```

1 | GraphicsWindow.Show()
2 | GraphicsWindow.Clear()

```

Grave o programa com o nome "**Cap05Ex26**", coloque-o em execução e veja junto a figura 5.14.



Figura 5.14 - Tela gráfica

A imagem apresentada na figura 5.14 é semelhante as telas que foram apresentadas com o uso do objeto **Turtle**. No entanto, deixando o objeto **Turtle** de lado, vejamos mais exclusivamente o objeto **GraphicsWindow** que possui funcionalidades interessantes.

Um ponto importante antes de realizar operações em modo gráfico é ter noção do dimensionamento e formato da janela gráfica. Atente para a figura 5.15.

Uma das aplicações mais populares na área de desenvolvimento de software é o desenvolvimento de programas com interfaces gráficas com seus campos de entrada e botões de operação. Para atender a estas necessidades deve-se usar além do objeto **GraphicsWindow** o objeto complementar **Controls**.

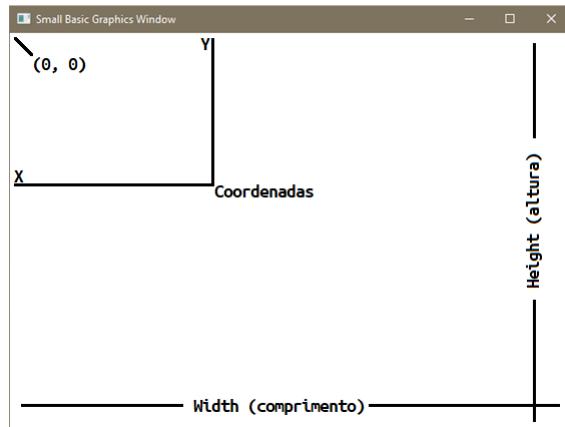


Figura 5.15 - Dimensionamento da janela gráfica

No sentido de demonstrarmos o uso de interface gráfica considere um programa que receba a entrada de um valor numérico e apresente como resultado seu dobro. O programa deve possuir dois botões de comando: um para realizar o cálculo e outro para encerrar a aplicação. Assim sendo, considere o código seguinte e observe as colorizações dos recursos em uso, até então, desconhecidos:

```

1 GraphicsWindow.Show()
2 GraphicsWindow.Clear()
3
4 GraphicsWindow.BackgroundColor = "DarkOliveGreen"
5 GraphicsWindow.Title = "Calculador"
6 GraphicsWindow.Width = 230
7 GraphicsWindow.Height = 104
8 GraphicsWindow.FontName = "Consolas"
9 GraphicsWindow.FontSize = 12
10
11 GraphicsWindow.BrushColor = "White"
12 GraphicsWindow.DrawText(10, 14, "Entre um valor numérico:")
13
14 GraphicsWindow.BrushColor = "Black"
15 N = Controls.AddTextBox(175, 10)
16 Controls.SetSize(N, 40, 20)
17
18 Calc = Controls.AddButton(" Calcular ", 10, 40)
19 Sair = Controls.AddButton(" Encerrar ", 95, 40)
20 Controls.ButtonClicked = BotaoAcionado
21
22 GraphicsWindow.BrushColor = "White"
23 GraphicsWindow.DrawText(10, 75, "Resultado =")
24
25 Sub BotaoAcionado
26 UltimoBotao = Controls.LastClickedButton
27 If (UltimoBotao = Calc) Then
28 R = Controls.GetTextBoxText(N) * 2

```

```
29 GraphicsWindow.DrawText(88, 75, R)
30 ElseIf (UltimoBotao = Sair) Then
31 Program.End()
32 EndIf
33 EndSub
```

Grave o programa com o nome "**Cap05Ex27**", coloque-o em execução. Entre um valor numérico no campo apresentado, por exemplo, "6" e veja o resultado "12" como indica a figura 5.16.

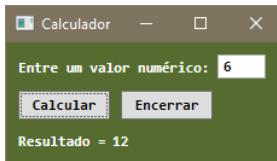


Figura 5.16 - Aplicação com interface gráfica

O programa anterior faz uso de alguns recursos desconhecidos.

Veja em **GraphicsWindow** o uso da: propriedade **FontName** (linha "8") usada para apresentar na tela o texto escrito de acordo com a fonte indicada e que esteja instalada em seu sistema; propriedade **FontSize** (linha "9") para estabelecer o tamanho da fonte em uso; da operação **DrawText()** que coloca um texto (terceiro parâmetro) em uma posição da tela gráfica, sendo o primeiro parâmetro o valor da coordenada "X" e o segundo parâmetro o valor da coordenada "Y" nas linhas "12", "23" e "29".

Para auxiliar as operações do programa em modo gráfico usa-se como apoio o objeto **Controls** que coloca em uso os elementos de controle para as ações de entrada, processamento e saída do programa.

Para efetivar a entrada do valor numérico usa-se do objeto **Controls** a operação **AddTextBox()** que apresenta um campo para a entrada de dados a partir do uso de dois parâmetros, sendo o primeiro a definição da coordenada "X" e o segundo a definição da coordenada "Y" para posicionamento do campo na tela. Atente para o fato de que um campo de entrada de dados deve ser associado a uma variável que receberá seu conteúdo. Neste caso, a variável "**N**", semelhantemente ao uso das operações **ReadNumber()** ou **Read()** do objeto **TextWindow**. Veja a linha "15".

A operação **SetSize()** do objeto **Controls** na linha "16" é usada para definir o tamanho de algum campo. Neste caso, o campo associado a variável "**N**". Veja que esta operação faz uso de três parâmetros: o primeiro a variável, o segundo a largura do campo e o terceiro a altura do campo.

Definida a parte que receba a entrada de dados, passa-se a definição da parte relacionada ao processamento do programa, sendo basicamente o trecho mais trabalhoso. Veja nas linhas "18" e "19" a definição dos botões de controle com a operação **AddButton()** do objeto **Controls**. Note que cada botão deve ser associado a uma variável. A ação da operação **AddButton()** faz uso de três parâmetros: primeiro o rótulo de identificação do botão, segundo a posição da coordenada "X" e o terceiro a posição da coordenada "Y".

Após definir os botões é necessário estabelecer o tratamento do disparo desses botões. É aqui que entra o tratamento do que se chama *evento*. Quando um botão é selecionado ocorre o "disparo" de certo evento. Por exemplo, o botão "**Calcular**" quando acionado dispara um evento para efetuar o cálculo, assim como o botão "**Encerrar**" quando acionado deve fechar o programa.

O evento do acionamento dos botões é detectado na linha "20" a partir de **ButtonClicked** do objeto **Controls**. Observe que esta linha indica que o evento **ButtonClicked** é associado a sub-rotina, neste caso, "**BotaoAcionado**" que possui seu código definido entre as linhas "25" e "33".

Parte da informação de saída do programa está definida na linha "23" e o restante definido na linha "29" dentro do código da sub-rotina "**BotaoAcionado**".

Tanto o processamento do cálculo quanto do encerramento do programa encontra-se na sub-rotina "**BotaoAcionado**" que verifica qual dos botões foi acionado e procede a ação correspondente. Veja que na linha "26" usa-se a propriedade **LastClickedButton** do objeto **Controls** para detectar o botão acionado atribuindo esta ação junto a variável "**UltimoBotao**".

Se o botão acionado e armazenado na variável "**UltimoBotao**" for aquele que está associado a variável "**Calc**" (linha "27") pegar-se-á o valor informado junto ao campo de entrada com a operação **GetTextBoxText()** do objeto **Controls** na linha "28", multiplicar-se-á por "2" e atribuir-se-á o resultado a variável "**R**", fazendo sua apresentação junto a linha "29". Caso contrário, se o botão acionado e armazenado na variável "**UltimoBotao**" for aquele associado a variável "**Sair**" (linha "30") far-se-á o encerramento do programa indicado na linha "31".

Veja mais efeitos. O programa mostra o texto "**Atenção**" em formato negrito, itálico, em tom amarelo no fundo vermelho, em tamanho "50" com fonte "**Courier New**":

```

1 GraphicsWindow.Show()
2 GraphicsWindow.Clear()
3 GraphicsWindow.Title = "Mostra texto"
4 GraphicsWindow.Width = 260
5 GraphicsWindow.Height = 150
6 GraphicsWindow.FontBold = "True"
7 GraphicsWindow.FontItalic = "True"
8 GraphicsWindow.BrushColor = "Yellow"
9 GraphicsWindow.BackgroundColor = "Red"
10 GraphicsWindow.FontSize = 50
11 GraphicsWindow.FontName = "Courier New"
12 GraphicsWindow.DrawText(20, 40, "Atenção")

```

Grave o programa com o nome "**Cap05Ex28**", coloque-o em execução e veja junto a figura 5.17.



Figura 5.17 - Texto com formatações

Veja no programa as definições das propriedades **FontBold** e **FontItalic** com a definição dos valores **True** para indicar que o efeito deve ser aplicado. Essas propriedades operam apenas com os valores **True** (para ativar o efeito) e **False** (para desativar o efeito).

A partir de uma visão geral sobre a definição de interface gráfica vamos ver um pouco do estabelecimento de algumas ações geométricas sobre a janela gráfica. Para começar considere o recurso para desenhar linhas na janela gráfica. Linhas são desenhadas com a operação **DrawLine()** do objeto **GraphicsWindow**. Esta operação faz uso de quatro parâmetros, podendo ser entendidos a partir da seguinte sintaxe:

**GraphicsWindow.DrawLine(x1, y1, x2, y2)**

Onde, "**x1**" é a coordenada "X" do ponto inicial, "**y1**" é a coordenada "Y" do ponto inicial, "**x2**" é a coordenada "X" do ponto final e "**y2**" é a coordenada "Y" do ponto final.

Observe o programa seguinte que desenha linhas: horizontal, vertical, perpendicular da esquerda para a direita e perpendicular da direita para a esquerda:

```

1 | GraphicsWindow.Title = "Linhas"
2 | GraphicsWindow.Show()
3 | GraphicsWindow.Clear()
4 | GraphicsWindow.Width = 220
5 | GraphicsWindow.Height = 115
6 | GraphicsWindow.DrawLine(15, 10, 95, 10) ' horizontal
7 | GraphicsWindow.DrawLine(30, 19, 30, 99) ' vertical
8 | GraphicsWindow.DrawLine(50, 59, 99, 99) ' perpendicular: esq -> dir
9 | GraphicsWindow.DrawLine(40, 99, 99, 40) ' perpendicular: dir -> esq

```

Grave o programa com o nome "**Cap05Ex29**", coloque-o em execução e veja junto a figura 5.18.



Figura 5.18 - Desenho de linhas

As linhas traçadas de uma imagem podem ser colorizadas a partir da propriedade **PenColor** do objeto **GraphicsWindow** que permite mudar a cor da caneta em uso. Assim sendo considere o próximo código que mostra em fundo preto e as linhas do programa anterior colorizadas. Atente para as marcações em tom vermelho:

```

1 | GraphicsWindow.Title = "Linhas"
2 | GraphicsWindow.Show()
3 | GraphicsWindow.Clear()
4 | GraphicsWindow.BackgroundColor = "Black"
5 | GraphicsWindow.Width = 220
6 | GraphicsWindow.Height = 115
7 | GraphicsWindow.PenColor = "Green"
8 | GraphicsWindow.DrawLine(15, 10, 95, 10) ' horizontal
9 | GraphicsWindow.PenColor = "Yellow"
10 | GraphicsWindow.DrawLine(30, 19, 30, 99) ' vertical
11 | GraphicsWindow.PenColor = "Cyan"
12 | GraphicsWindow.DrawLine(50, 59, 99, 99) ' perpendicular: esq -> dir
13 | GraphicsWindow.PenColor = "Magenta"
14 | GraphicsWindow.DrawLine(40, 99, 99, 40) ' perpendicular: dir -> esq

```

Grave o programa com o nome "**Cap05Ex30**", coloque-o em execução e veja junto a figura 5.19.

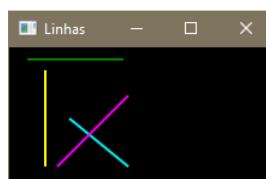


Figura 5.19 - Desenho de linhas colorizadas

As linhas traçadas podem ter suas espessuras alteradas a partir da propriedade **PenWidth** do objeto **GraphicsWindow**. Essa propriedade aceita valores numéricos a partir de "1". Veja o programa a seguir e atente para as linhas marcadas em tom vermelho:

```

1 GraphicsWindow.Title = "Linhas"
2 GraphicsWindow.Show()
3 GraphicsWindow.Clear()
4 GraphicsWindow.BackgroundColor = "Black"
5 GraphicsWindow.Width = 220
6 GraphicsWindow.Height = 115
7 GraphicsWindow.PenColor = "Green"
8 GraphicsWindow.PenWidth = 5
9 GraphicsWindow.DrawLine(15, 10, 95, 10) ' horizontal
10 GraphicsWindow.PenColor = "Yellow"
11 GraphicsWindow.PenWidth = 10
12 GraphicsWindow.DrawLine(30, 19, 30, 99) ' vertical
13 GraphicsWindow.PenColor = "Cyan"
14 GraphicsWindow.PenWidth = 15
15 GraphicsWindow.DrawLine(50, 59, 99, 99) ' perpendicular: esq -> dir
16 GraphicsWindow.PenColor = "Magenta"
17 GraphicsWindow.PenWidth = 20
18 GraphicsWindow.DrawLine(40, 99, 99, 40) ' perpendicular: dir -> esq

```

Grave o programa com o nome "**Cap05Ex31**", coloque-o em execução e veja junto a figura 5.20.



Figura 5.20 - Alteração na espessura de linhas

A partir desta visão podemos passar para a parte de desenhar figuras geométricas planas como: quadrados, retângulos, elipses, circunferências e triângulos. Para desenhar essas formas geométricas a linguagem "*Small Basic*" oferece a partir do objeto **GraphicsWindow** seis operações: três para desenhar e três para preencher a imagem com determinada cor. Veja a tabela 5.4.

| OPERAÇÃO             | AÇÃO                                           |
|----------------------|------------------------------------------------|
| <b>DrawEllipse</b>   | Usado para desenhar elipses ou circunferências |
| <b>FillEllipse</b>   | Usado para colorizar elipses e circunferências |
| <b>DrawRectangle</b> | Usado para desenhar quadrados e retângulos     |
| <b>FillRectangle</b> | Usado para colorizar quadrados e retângulos    |
| <b>DrawTriangle</b>  | Usado para desenhar triângulos                 |
| <b>FillTriangle</b>  | Usado para colorizar triângulos                |

Figura 5.4 - Recursos para definição de formas geométricas

As operações de desenhos são realizadas com as instruções:

```
GraphicsWindow.DrawEllipse(x, y, comprimento, largura)
GraphicsWindow.DrawRectangle(x, y, comprimento, largura)
GraphicsWindow.DrawTriangle(x1, y1, x2, y2, x3, y3)
```

As operações de preenchimento dos desenhos são realizadas com as instruções:

```
GraphicsWindow.FillEllipse(x, y, comprimento, largura)
GraphicsWindow.FillRectangle(x, y, comprimento, largura)
GraphicsWindow.FillTriangle(x1, y1, x2, y2, x3, y3)
```

Onde, para as figuras circulares e quadrangulares os parâmetros "x" são as coordenadas "X", "y" são as coordenadas "Y", "comprimento" e "largura" são o tamanho da figura. Para a figura triangular os parâmetros "x1", "y1", "x2", "y2", "x3" e "y3" que marcam as posições dos pontos definidos para um triângulo de qualquer tipo.

Veja a demonstração do uso das operações **DrawRectangle()** e **FillRectangle()** ao desenhar um retângulo com bordas pretas em cor cinza. Veja o código e atente para os detalhes usados:

```
1 GraphicsWindow.Title = "Retângulo"
2 GraphicsWindow.Show()
3 GraphicsWindow.Clear()
4 GraphicsWindow.Width = 300
5 GraphicsWindow.Height = 200
6 GraphicsWindow.PenColor = "Black"
7 GraphicsWindow.BrushColor = "Gray"
8 GraphicsWindow.FillRectangle(50, 50, 200, 100)
9 GraphicsWindow.DrawRectangle(50, 50, 200, 100)
```

Grave o programa com o nome "**Cap05Ex32**", coloque-o em execução e veja junto a figura 5.21.

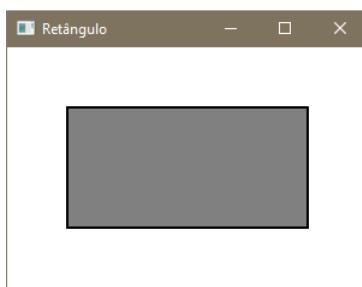


Figura 5.21 - Apresentação de retângulo

Para desenhar um quadrado usa-se as operações **DrawRectangle()** e **FillRectangle()** que desenham um retângulo, pois todo quadrado é por sua natureza um retângulo e também um losango.

Para desenhar um quadrado é necessário definir os parâmetros de *comprimento* e *largura* deverão com o mesmo tamanho. Observe o programa seguinte que desenha um quadrado amarelo com borda azul:

```
1 GraphicsWindow.Title = "Quadrado"
2 GraphicsWindow.Show()
3 GraphicsWindow.Clear()
4 GraphicsWindow.Width = 300
5 GraphicsWindow.Height = 200
```

```

6 | GraphicsWindow.PenColor = "Blue"
7 | GraphicsWindow.BrushColor = "Yellow"
8 | GraphicsWindow.FillRectangle(50, 50, 100, 100)
9 | GraphicsWindow.DrawRectangle(50, 50, 100, 100)

```

Grave o programa com o nome "**Cap05Ex33**", coloque-o em execução e veja junto a figura 5.22.

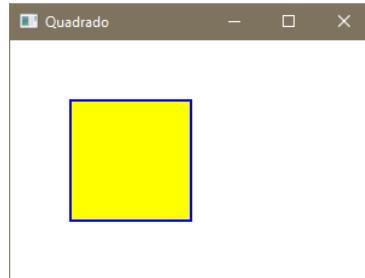


Figura 5.22 - Apresentação de quadrado

Assim como acontece com quadrados e retângulos, elipses e circunferências usam as operações **DrawEllipse()** e **FillEllipse()**. Veja a seguir um programa que mostra a imagem de um círculo circundado com borda vermelha pintado na cor ciano com a sobreposição de uma elipse pintada de amarelo e circundado em laranja com espessura de borda maior:

```

1 | GraphicsWindow.Title = "Circular"
2 | GraphicsWindow.Show()
3 | GraphicsWindow.Clear()
4 | GraphicsWindow.Width = 300
5 | GraphicsWindow.Height = 250
6 | GraphicsWindow.PenColor = "Red"
7 | GraphicsWindow.BrushColor = "Cyan"
8 | GraphicsWindow.FillEllipse(50, 50, 100, 100)
9 | GraphicsWindow.DrawEllipse(50, 50, 100, 100)
10 | GraphicsWindow.PenColor = "Orange"
11 | GraphicsWindow.BrushColor = "Yellow"
12 | GraphicsWindow.PenWidth = 5
13 | GraphicsWindow.FillEllipse(100, 100, 150, 100)
14 | GraphicsWindow.DrawEllipse(100, 100, 150, 100)

```

Grave o programa com o nome "**Cap05Ex34**", coloque-o em execução e veja junto a figura 5.23.

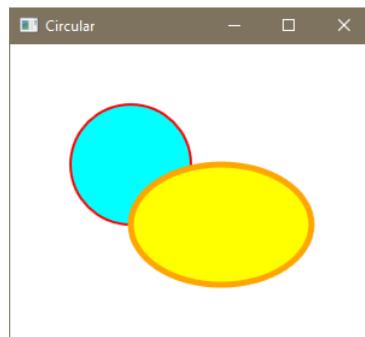


Figura 5.22 - Apresentação de figuras circulares

Para completar esta série falta apenas desenhar a imagem de um triângulo. Para tanto, observe o código seguinte que faz uso das operações **DrawTriangle()** e **FillTriangle()** para desenhar um quadrado com borda azul:

```
1 GraphicsWindow.Title = "Triângulo"
2 GraphicsWindow.Show()
3 GraphicsWindow.Clear()
4 GraphicsWindow.Width = 300
5 GraphicsWindow.Height = 200
6 GraphicsWindow.PenWidth = 3
7 GraphicsWindow.PenColor = "Blue"
8 GraphicsWindow.DrawTriangle(200, 50, 50, 145, 250, 160)
```

Grave o programa com o nome "**Cap05Ex35**", coloque-o em execução e veja junto a figura 5.24.

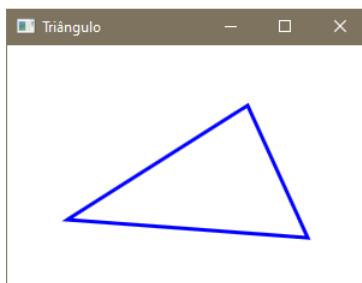


Figura 5.24 - Apresentação de triângulo

O uso de interface gráfica para criação de programas "*Small Basic*" oferece um recurso de interatividade muito útil. Trata-se do uso de caixa de mensagem. Uma caixa de mensagem é um componente que tem por finalidade apresentar uma mensagem em um quadro de advertência com a apresentação de um botão "**OK**" para encerramento da mensagem. Veja na figura 5.25 os modelos existentes.

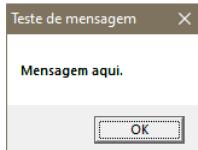


Figura 5.25 - Caixas de mensagem

A caixa de mensagem apresentada na figura 5.25 é apresentada a partir do uso da instrução:

```
GraphicsWindow.ShowMessage("Mensagem aqui.", "Teste de mensagem")
```

Veja que a operação **ShowMessage()** do objeto **GraphicsWindow** faz uso de dois parâmetros: o primeiro para definir a mensagem a ser apresentada e o segundo para definir o título da caixa de mensagem.

Veja o uso do recurso de apresentação de caixa de mensagem em um programa que apresenta um efeito de barras coloridas em janela gráfica com fundo cinza escuro. Veja o código a seguir:

```
1 GraphicsWindow.Title = "Cortina de cores"
2
3 GraphicsWindow.BackgroundColor = "DimGray"
4 GraphicsWindow.Width = 325
5 GraphicsWindow.Height = 200
6
7 Cor[1] = "#5B0E5B"
8 Cor[2] = "#B40158"
9 Cor[3] = "#C3090D"
```

```

10 Cor[4] = "#DA030C"
11 Cor[5] = "#E26100"
12 Cor[6] = "#F2A000"
13 Cor[7] = "#F8EC3A"
14 Cor[8] = "#F8EC3A"
15 Cor[9] = "#3D932D"
16 Cor[10] = "#51B17A"
17 Cor[11] = "#6BBFDF"
18 Cor[12] = "#003E92"
19 Cor[13] = "#002E73"
20 Cor[14] = "#00134D"
21 Cor[15] = "#00063E"
22
23 For I = 1 To 15
24 GraphicsWindow.PenColor = Cor[I]
25 GraphicsWindow.PenWidth = I
26 GraphicsWindow.DrawLine(I * 20, 10, I * 20, 180)
27 EndFor
28
29 GraphicsWindow.ShowMessage("Ação concluída", "Cortina de cores")

```

Grave o programa com o nome "**Cap05Ex36**", coloque-o em execução e veja junto a figura 5.26.

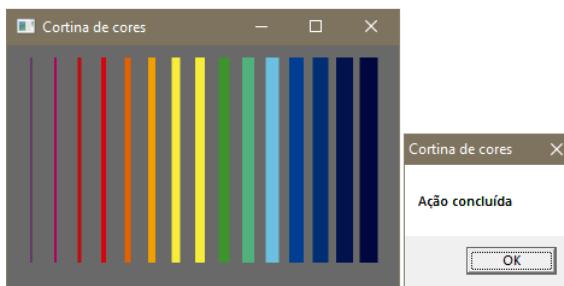


Figura 5.26 - Apresentação de triângulo

Caixas de mensagens podem ser usadas de muitas formas, principalmente para a indicação de ocorrências de erros em tomadas de decisão.

### 5.6 - Endentação automatizada

Você deve ter percebido que em vários programas escritos foi utilizado uma forma de tabulação no código a fim de mostrar o que está dentro do que. Esta forma de tabulação é conhecida como *endentaçāo*.

Computadores não necessitam de endentação para executar programas. Este formato de escrita é usado por nós humanos para nos facilitar a identificação de blocos de códigos. O efeito de endentação é aplicado apenas e tão comente após os comandos **If / Then**, **Else**, **Elself**, **For**, **While** e **Sub**.

O uso de endentação é considerado uma prática no desenvolvimento de bons códigos. É educado escrever programas com essa formatação. Muitas organizações levam isso em extrema consideração.

O que você, talvez, não saiba é que o ambiente de programação da linguagem "*Small Basic*" possui um recurso que efetua automaticamente a endentação do código escrito. Para um teste, observe o programa a seguir escrito sem endentação:

```
1 TextWindow.Clear()
2 TextWindow.WriteLine("PROGRAMA FATORIAL")
3 TextWindow.WriteLine("")
4 TextWindow.Write("Entre um valor inteiro positivo: ")
5 N = Math.Floor(TextWindow.ReadNumber())
6 Fat = 1
7 For I = 1 To N
8 Fat = Fat * I
9 EndFor
10 TextWindow.WriteLine(N + "!" = " + Fat)
11 TextWindow.WriteLine("")
12 TextWindow.Write("Tecle <Enter> para encerrar... ")
13 Enter = TextWindow.Read()
14 Program.End()
```

Grave o programa com o nome "**Cap05Ex37**", coloque-o em execução e veja que o programa é executado normalmente, pois a existência ou não da endentação não afeta sua funcionalidade. A figura 5.27 mostra o ambiente com o código sem endentação.

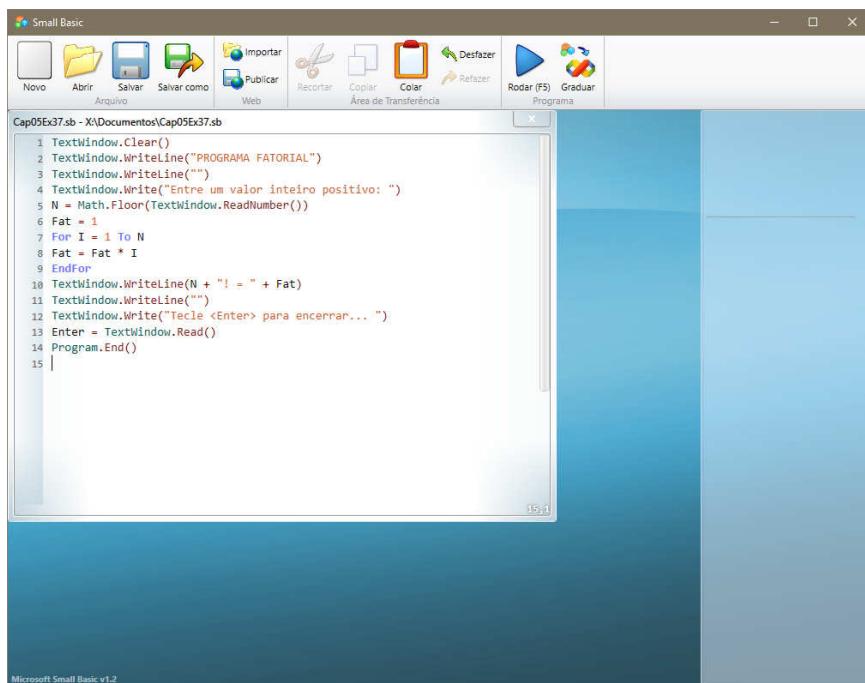


Figura 5.27 - Programa sem endentação

Para fazer a endentação de forma automatizada basta acionar o botão de contexto do *mouse* (normalmente o botão direito) e selecionar no menu apresentado a opção "**Formatar Texto do Programa**". As figuras 5.28 e 5.29 mostram respectivamente a opção a ser selecionada e o efeito após sua ação.

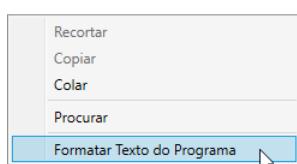


Figura 5.28 - Seleção de endentação automatizada

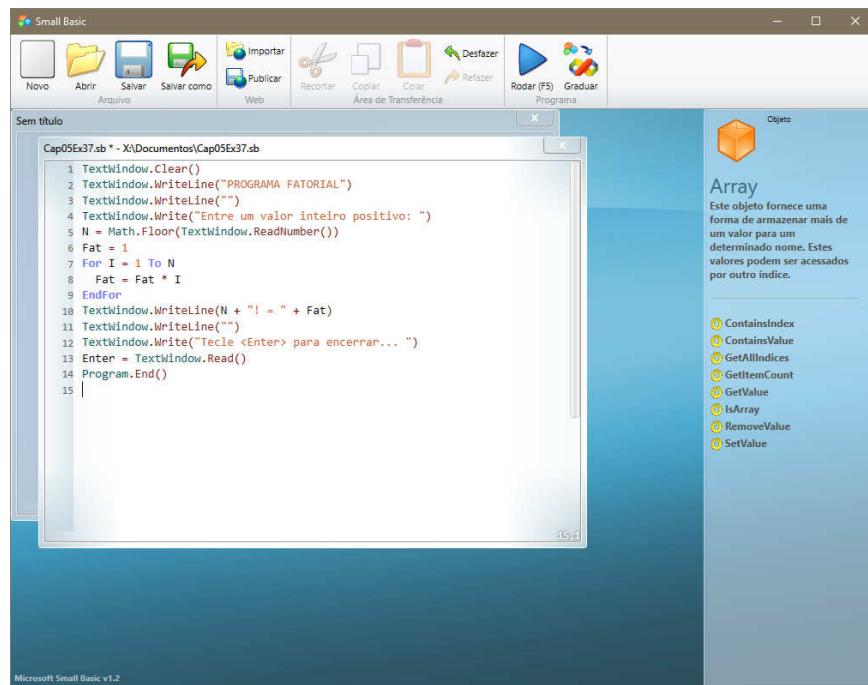


Figura 5.29 - Programa com endentação

O fato da ferramenta possuir este recurso não significa em absoluto que é para deixar de escrever programas com endentação. O recurso serve para auxiliar na correção de endentações mal definidas. Salve a alteração com o nome "**Cap05Ex38**". Para tanto, use o botão "**Salvar como**".

### 5.7 - Uso de arquivos

A finalidade de uma linguagem de programação é criar aplicações para computadores. As aplicações computacionais, por sua vez, são responsáveis pela manipulação de dados. Daí o fato de um computador executar as ações de entrada de dados, processamento de dados (matemático e lógico) e saída de dados que podem se tornar informações.

Os dados são, por sua natureza própria, a matéria prima principal para o desenvolvimento de programas. Esses dados devem ser armazenados em memória para poderem ser utilizados a qualquer momento e, nesse sentido, a dois tipos de memória usadas pelos dados: a memória principal e a memória secundária (ou memória de massa).

A memória principal é o espaço onde ocorrem o armazenamento de dados temporários. É esse espaço que usamos até o presente momento para o armazenamento de dados no computador por meio das variáveis simples e indexadas. Se o computador for desligado ou o ambiente de programação encerrado os dados armazenados nessa memória são permanentemente perdidos.

A memória secundária por sua vez é um espaço onde os dados podem ser gravados e utilizados mesmo após o desligamento do computador ou encerramento do ambiente de programação. Essa memória se caracteriza pelo uso dos componentes de armazenamento como: discos rígidos, pen-drives, CDs, DVDs e etc. É nesse tipo de mídia que se define o uso de arquivos.

É importante ressaltar que a memória primária é extremamente rápida e o mesmo não se pode dizer da memória secundária. Por esta razão usamos as memórias secundárias para manutenção do armazenamento de dados e a memória principal é usada para efetivar o processamento desses dados.

A linguagem "*Small Basic*", assim como outras tantas linguagens de programação, dá suporte ao uso de arquivos. Neste caso, a partir do objeto **File**.

O objeto **File** oferece um conjunto de recursos para o tratamento de operações sobre arquivos de dados baseados em formato texto. Como comentado a linguagem foi desenvolvida exclusivamente para uso educacional e por esta razão possui certas limitações, entre elas o fato de não fazer uso de arquivos do tipo binário, pois não foi projetada para o uso comercial, industrial ou científico, ou seja, dentro do escopo que a linguagem atua o uso de arquivos binários não é necessário.

Para realizarmos os exemplos deste tópico é necessário ter um local para a criação e manipulação de arquivos. Para esta finalidade será criada na raiz de seu disco principal (normalmente identificado pela unidade "C:") a pasta (ou diretório) "**Dados**". Isso pode ser feito manualmente ou por um programa (que é mais prático). Veja o código seguinte:

```

1 | TextWindow.Clear()
2 | TextWindow.WriteLine("CRIAÇÃO DE DIRETÓRIO (PASTA)")
3 | TextWindow.WriteLine("")
4 |
5 | File.CreateDirectory("C:\Dados")
6 |
7 | TextWindow.WriteLine("Diretório (pasta) criado")
8 | TextWindow.WriteLine("")
9 | TextWindow.Write("Tecle <Enter> para encerrar... ")
10| Enter = TextWindow.Read()
11| Program.End()

```

Grave o programa com o nome "**Cap05Ex39**", coloque-o em execução. Depois por meio do programa "**Explorer**" localize a pasta em seu disco rígido. A figura 5.30 mostra o diretório criado.

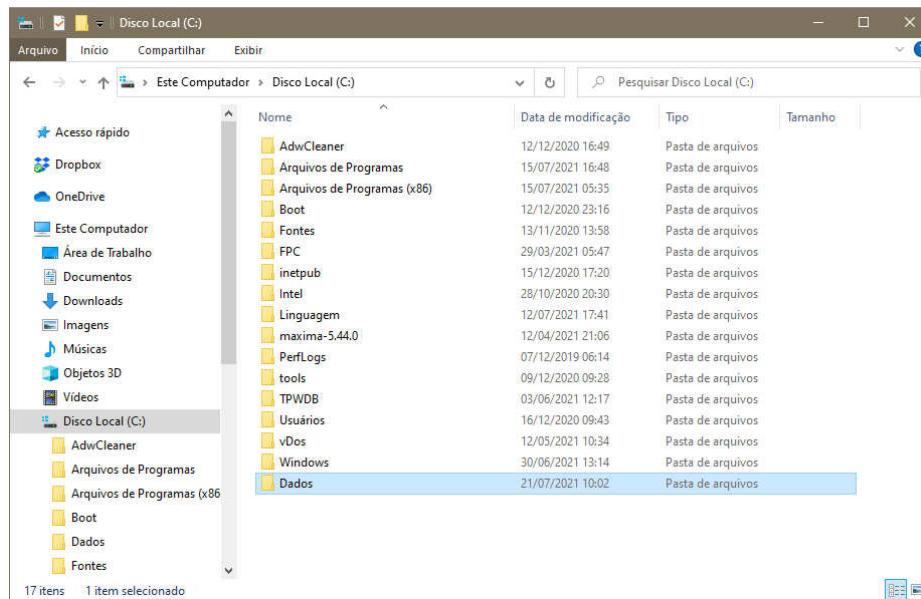


Figura 5.30 - Diretório "Dados" criado na raiz do disco "C:"

Atente ao uso da instrução "**File.CreateDirectory("C:\Dados")**" no programa "**Cap05Ex39**". Veja que a operação **CreateDirectory()** do objeto **File** cria um diretório a partir do parâmetro passado e definido entre aspas inglesas. Note que é necessário indicar a unidade de disco, colocar dois pontos e contra barra e informar o nome do diretório a ser criado.

As operações do objeto **File** que efetuam escrita de dados criam o arquivo caso este não exista. Veja o código a seguir que tem por finalidade escrever algo em um arquivo. Como o arquivo não existe este é automaticamente criado:

```

1 TextWindow.Clear()
2 TextWindow.WriteLine("CRIAÇÃO DE ARQUIVO")
3 TextWindow.WriteLine("")
4
5 Arquivo = "C:\Dados\arqteste.txt"
6 File.WriteAllText(Arquivo, "")
7
8 TextWindow.WriteLine("Um arquivo de exemplo foi criado")
9 TextWindow.WriteLine("")
10 TextWindow.Write("Tecla <Enter> para encerrar... ")
11 Enter = TextWindow.Read()
12 Program.End()

```

Grave o programa com o nome "**Cap05Ex40**", coloque-o em execução e veja que será apenas apresentada na tela a mensagem "**Um arquivo de exemplo foi criado**". Neste instante na pasta "**Dados**" foi criado o arquivo "**arqteste.txt**" ocupando "**0 KB**" de espaço no disco, ou seja um arquivo vazio com a escrita do conteúdo ("""), como mostra a figura 5.31.

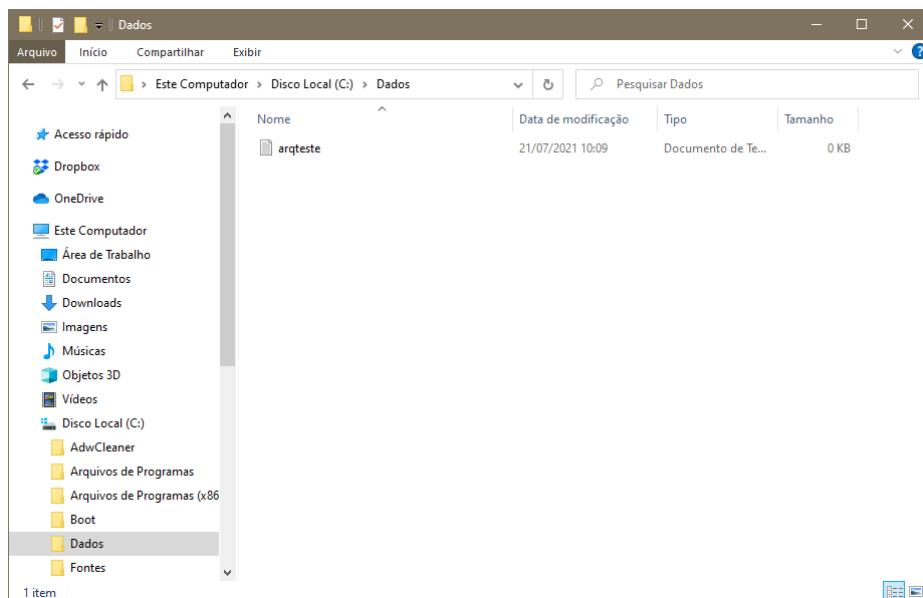


Figura 5.31 - Arquivo de dados criado

O programa "**Cap05Ex40**" define na linha "**5**" uma variável que armazena o caminho (**Dados**) e o nome do arquivo (**arqteste.txt**). É uma prática muito comum, em certas linguagens de programação, na manipulação de arquivos usar essa estratégia. Na linha "**6**" encontra-se a instrução que cria o arquivo com um conteúdo vazio.

A operação **WriteContents()** do objeto **File** é operada a partir do fornecimento de dois parâmetros: o primeiro é o local e nome do arquivo e o segundo é o conteúdo a ser escrito, que neste caso é nada ou vazio pela indicação ("") gravada no arquivo sinalizado pela variável "**Arquivo**".

A partir da criação do arquivo vamos colocá-lo em uso. Para tanto, observe o uso da operação **AppendContents()** do objeto **File** que tem por finalidade acrescentar "novas" linhas ao final do arquivo. Veja como exemplo o código seguinte:

```

1 TextWindow.Clear()
2 TextWindow.WriteLine("ESCRITA EM ARQUIVO")
3

```

```
4 Arquivo = "C:\Dados\arqteste.txt"
5 File.AppendContents(Arquivo, "Linguagem Small Basic")
6
7 TextWindow.WriteLine("")
8 TextWindow.Write("Tecla <Enter> para encerrar... ")
9 Enter = TextWindow.Read()
10 Program.End()
```

Grave o programa com o nome "**Cap05Ex41**" e coloque-o em execução por três vezes. Vá até a pasta "**Dados**" e abra o arquivo "**arqteste.txt**" com o programa "**Bloco de notas**" e veja o conteúdo existente como mostra a figura 5.32.

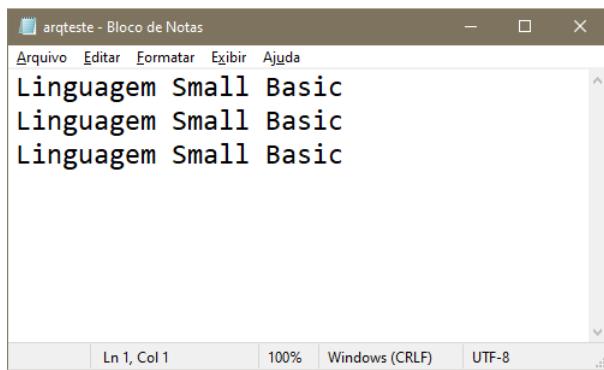


Figura 5.32 - Arquivo de dados criado

O programa "**Cap05Ex41**" a cada execução cria ao final do arquivo uma nova linha de texto com a mensagem "**Linguagem Small Basic**". Observe que a operação **AppendContents()** faz o acréscimo (**Append**) dos dados indicados (**Contents**).

Imagine que você tenha, por exemplo, na pasta "**C:\Dados**" vários arquivos e queira obter uma lista dos arquivos existentes neste diretório. Neste caso, podemos usar a operação **GetFiles()** que lê os arquivos do diretório e armazena-os em uma variável na forma de um arranjo (variável indexada). Observe o código seguinte:

```
1 TextWindow.Clear()
2 TextWindow.WriteLine("ARQUIVO(S) DE UM DIRETÓRIO")
3 TextWindow.WriteLine("")
4
5 Diretorio = "C:\Dados"
6 Conteudo = File.GetFiles(Diretorio)
7
8 For I = 1 To Array.GetItemCount(Conteudo)
9 TextWindow.WriteLine(Conteudo[I])
10 EndFor
11
12 TextWindow.WriteLine("")
13 TextWindow.Write("Tecla <Enter> para encerrar... ")
14 Enter = TextWindow.Read()
15 Program.End()
```

Grave o programa com o nome "**Cap05Ex42**", coloque-o em execução e veja o conteúdo do diretório indicado apresentado como mostra a figura 5.33. Perceba que o programa apresenta

linearmente o conteúdo encontrado no diretório indicado na variável indexada "Conteúdo". Junto a linha "9".

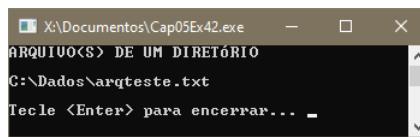


Figura 5.33 - Arquivos de um diretório

O programa "**Cap05Ex42**" para realizar a ação que necessita usa a operação **GetItemCount()** do objeto **Array** que detecta a quantidade de elementos de uma variável indexada, ou seja, neste caso, a quantidade de arquivos existentes no diretório. Para saber mais sobre o objeto **Array** consulte o apêndice deste livro.

Outro recurso de obtenção de informação é a operação **GetDirectories()** do objeto **File** que obtém todos os caminhos de todos os diretórios de um diretório especificado. Veja o código seguinte em que estamos obtendo a lista de todos os diretórios da raiz do disco principal:

```

1 TextWindow.Clear()
2 TextWindow.WriteLine("DIRETÓRIOS DA RAIZ")
3 TextWindow.WriteLine("")
4
5 Diretorio = "C:"
6 Conteudo = File.GetDirectories(Diretorio)
7
8 For I = 1 To Array.GetItemCount(Conteudo)
9 TextWindow.WriteLine(Conteudo[I])
10 EndFor
11
12 TextWindow.WriteLine("")
13 TextWindow.Write("Tecle <Enter> para encerrar... ")
14 Enter = TextWindow.Read()
15 Program.End()

```

Grave o programa com o nome "**Cap05Ex43**", coloque-o em execução e veja o conteúdo do diretório indicado apresentado como mostra a figura 5.34.

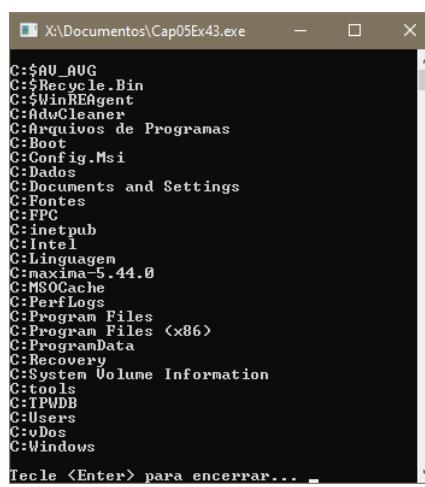


Figura 5.34 - Conteúdo do diretório raiz

O conteúdo indicado na figura 5.34 com certeza será diferente do conteúdo de seu sistema, pois alguns diretórios dependem das configurações e programas instalados.

Assim como arquivos e diretórios podem ser criados também podem ser apagados. É claro que este tipo de operação é mais arriscada e deve ser realizada com muito cuidado.

Para apagar um arquivo de um diretório basta fazer uso da operação **DeleteFile()** e para apagar um diretório basta fazer uso da operação **DeleteDirectory()**. Observe o código a seguir de um programa que remove o arquivo e diretório anteriormente criados:

```
1 | TextWindow.Clear()
2 | TextWindow.WriteLine("REMOÇÃO DE ARQUIVO E DIRETÓRIO")
3 | TextWindow.WriteLine("")
4 |
5 | Arquivo = "C:\Dados\arqteste.txt"
6 | Diretorio = "C:\Dados"
7 |
8 | File.DeleteFile(Arquivo)
9 | File.DeleteDirectory(Diretorio)
10|
11| TextWindow.WriteLine("Arquivo e diretório removidos")
12| TextWindow.WriteLine("")
13| TextWindow.Write("Tecle <Enter> para encerrar... ")
14| Enter = TextWindow.Read()
15| Program.End()
```

Grave o programa com o nome "**Cap05Ex44**" e coloque-o em execução. Depois busque na raiz de seu disco principal e veja se localiza o diretório. Se o diretório não existir o arquivo também não existirá. Veja que o programa remove primeiramente o arquivo (linha **8**) e posteriormente remove o diretório (linha **9**), não tende fazer ao contrário pois para remover um diretório este deve estar sem nenhum arquivo.

Um arquivo além de ser criado e removido pode ser copiado de um local do disco para outro local do disco ou para outra mídia de gravação. Veja o programa seguinte que cria os diretórios "**Dados1**" e "**Dados2**", depois cria em "**Dados1**" o arquivo "**teste.txt**" com uma linha de conteúdo definida a partir da entrada do usuário. Após as ações anteriores o programa efetua a cópia do arquivo "**teste.txt**" do diretório "**Dados1**" para o "**Dados2**" e remove do diretório "**Dados1**" o arquivo "**teste.txt**":

```
1 | TextWindow.Clear()
2 | TextWindow.WriteLine("CÓPIA DE ARQUIVO ENTRE DIRETÓRIOS")
3 | TextWindow.WriteLine("")
4 |
5 | File.CreateDirectory("C:\Dados1") ' Cria diretório "Dados1"
6 | File.CreateDirectory("C:\Dados2") ' Cria diretório "Dados2"
7 | File.WriteAllText("C:\Dados1\teste.txt", "") ' Cria arquivo
8 |
9 | TextWindow.Write("Entre algum texto: ")
10| Entrada = TextWindow.Read() ' Recebe uma entrada
11|
12| File.AppendText("C:\Dados1\teste.txt", Entrada) ' Grava entrada
13|
14| File.CopyFile("C:\Dados1\teste.txt", "C:\Dados2")
```

```

16 File.DeleteFile("C:\Dados1\teste.txt")
17
18 TextWindow.WriteLine("")
19 TextWindow.Write("Tecle <Enter> para encerrar... ")
20 Enter = TextWindow.Read()
21 Program.End()

```

Grave o programa com o nome "**Cap05Ex45**" e coloque-o em execução. Concluída a execução do programa a partir do uso do programa "**Explorer**" veja os diretórios criados e o arquivo operacionalizado definido apenas no diretório "**Dados2**".

Em relação a apresentação de informações sobre o gerenciamento de arquivos, além das operações obtidas por **GetFiles()** e **GetDirectories()** temos a operação **GetSettingsFilePath()** que obtém o caminho completo do arquivo de configurações do programa em execução e a operação **GetTemporaryFilePath()** que cria um arquivo temporário em um diretório temporário e retorna o caminho completo do arquivo temporário. Veja o programa a seguir apresentando essas funcionalidades:

```

1 TextWindow.Clear()
2 TextWindow.WriteLine("INFORMAÇÕES GERAIS")
3 TextWindow.WriteLine("")
4
5 ArqConf = File.GetSettingsFilePath()
6 ArqTemp = File.GetTemporaryFilePath()
7
8 TextWindow.WriteLine("Arquivo de configuração")
9 TextWindow.WriteLine(ArqConf)
10
11 TextWindow.WriteLine("")
12 TextWindow.WriteLine("Arquivo temporário")
13 TextWindow.WriteLine(ArqTemp)
14
15 TextWindow.WriteLine("")
16 TextWindow.Write("Tecle <Enter> para encerrar... ")
17 Enter = TextWindow.Read()
18 Program.End()

```

Grave o programa com o nome "**Cap05Ex46**" e coloque-o em execução. Veja os resultados apresentados na figura 5.35.

Essas funcionalidades mostram informações internas de execução dos programas que são usadas pelo programa e computador para que os programas surtam os efeitos desejados.

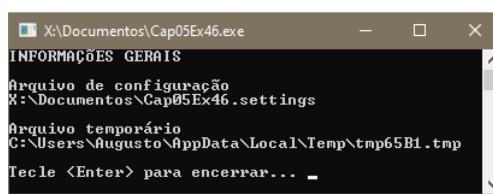


Figura 5.35 - Apresentação de informações internas

Além do exposto há ainda mais algumas operações importantes em relação ao uso de arquivos, entre elas a operação **ReadContents()** que faz a leitura do conteúdo de um arquivo. Para um teste considere um programa que efetua a leitura de cinco valores numéricos gravando esses

valores em no arquivo "arqnum.txt" no diretório "Dados1", limpe a tela após a as entradas e apresente os valores cadastrado:

```

1 TextWindow.Clear()
2 TextWindow.WriteLine("ENTRADA EM ARQUIVO")
3 TextWindow.WriteLine("")
4
5 Arquivo = "C:\Dados1\arqnum.txt"
6
7 For I = 1 To 5
8 TextWindow.Write("Entre o " + I + "o. valor numérico: ")
9 Valor = TextWindow.ReadNumber()
10 File.AppendContents(Arquivo, Valor)
11 EndFor
12
13 TextWindow.Clear()
14 TextWindow.WriteLine("SAÍDA DO ARQUIVO")
15 TextWindow.WriteLine("")
16
17 Leitura = File.ReadContents(Arquivo)
18 TextWindow.WriteLine(Leitura)
19
20 TextWindow.Write("Tecle <Enter> para encerrar... ")
21 Enter = TextWindow.Read()
22 Program.End()

```

Grave o programa com o nome "**Cap05Ex47**" e coloque-o em execução. Veja os resultados apresentados na figura 5.36 com as etapas de entrada a esquerda e a saída a direita. Se o programa for executado mais de uma vez irá acrescentar continuamente ao arquivo cinco valores por vez.

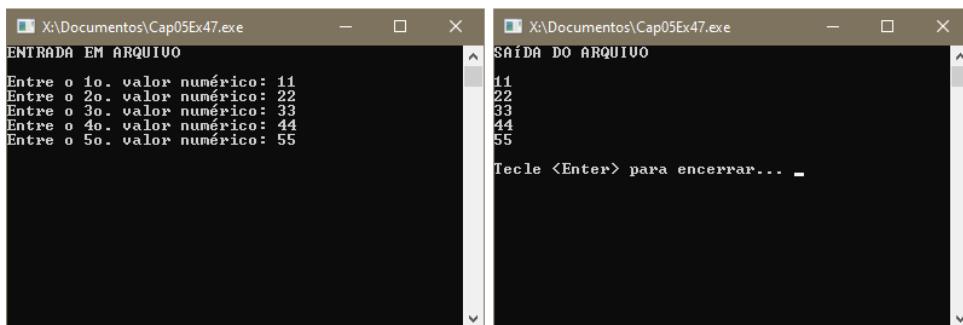


Figura 5.36 - Uso da operação "ReadContents()"

A operação **ReadContents()** faz a leitura de todo o conteúdo do arquivo de uma só vez, como mostra, a ação da linha "17" do programa.

Os exemplos anteriores usaram o arquivo com um todo. Apesar de útil isso pode ser um pouco inconveniente principalmente quando se deseja entrar informações posicionadas e controladas como se fossem registros.

A linguagem "*Small Basic*" tem disponível algumas operações que permitem fazer acesso controlado a posição dentro do arquivo, como se fosse um registro, a partir da indicação do número de certa linha. Para a realização dessas operações são oferecidas junto ao objeto **File** as operações **InsertLine()**, **ReadLine()** e **WriteLine()**.

A operação **InsertLine()** efetua a abertura ou criação do arquivo indicado e insere certo conteúdo em uma linha especificada. A inserção de conteúdo em determinada linha não substitui o conteúdo existente se uma posição usada for apontada. Neste caso, a linha com a indicação de posição usada é inserida na posição e o dado existente é "empurrado" para a frente.

A operação **ReadLine()** efetua a abertura ou criação do arquivo indicado e lê o conteúdo especificado pela posição.

A operação **WriteLine()** efetua a abertura ou criação do arquivo indicado e insere certo conteúdo em uma linha especificada. Se for usada uma posição ocupada o seu conteúdo será substituído.

Observe que as operações **InsertLine()** e **WriteLine()** devem ser usadas com cuidado, pois a vantagem de uma pode ser a desvantagem da outra e vice-versa. Para um teste dessas operações observe os dois próximos programas.

O primeiro programa faz uso de duas sequências com uso da operação **InsertLine()** que opera a entrada de três valores numéricos nas posições "1", "2" e "3", depois repete uma segunda etapa de entrada e apresenta os seis valores cadastrados. Observe o código seguinte:

```
1 TextWindow.Clear()
2 TextWindow.WriteLine("ACESSO CONTROLADO - V#1.0")
3
4 Arquivo = "C:\Dados1\arqteste1.txt"
5
6 TextWindow.WriteLine("")
7 TextWindow.WriteLine("Primeira bateria de entrada")
8 TextWindow.WriteLine("")
9 For I = 1 To 3
10 TextWindow.Write("Entre o " + I + "o. valor: ")
11 ValorEntra[I] = TextWindow.ReadNumber()
12 File.InsertLine(Arquivo, I, ValorEntra[I])
13 EndFor
14
15 TextWindow.WriteLine("")
16 TextWindow.WriteLine("Segunda bateria de entrada")
17 TextWindow.WriteLine("")
18 For J = 1 To 3
19 TextWindow.Write("Entre o " + J + "o. valor: ")
20 ValorEntra[J] = TextWindow.ReadNumber()
21 File.InsertLine(Arquivo, J, ValorEntra[J])
22 EndFor
23
24 TextWindow.WriteLine("")
25 TextWindow.WriteLine("Apresentação dos dados informados")
26 TextWindow.WriteLine("")
27 For K = 1 To 6
28 ValorSai[K] = File.ReadLine(Arquivo, K)
29 TextWindow.Write(K + "o. valor: ")
30 TextWindow.WriteLine(ValorSai[K])
31 EndFor
32
```

```
33 | TextWindow.WriteLine("")
34 | TextWindow.Write("Tecle <Enter> para encerrar... ")
35 | Enter = TextWindow.Read()
36 | Program.End()
```

Grave o programa com o nome "**Cap05Ex48**", coloque-o em execução e entre a sequência de valores solicitadas. Veja os resultados apresentados na figura 5.37.

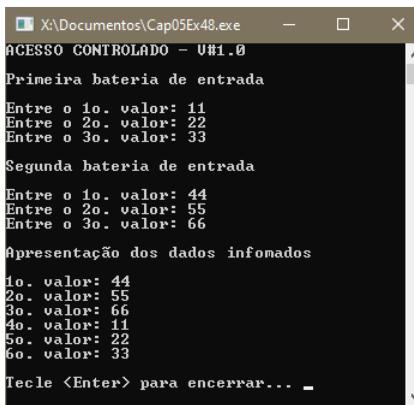


Figura 5.37 - Resultado da operação "InsertLine()"

O segundo programa faz uso de duas sequências com uso da operação **WriteLine()** que opera a entrada de três valores numéricos nas posições "1", "2" e "3", depois repete uma segunda etapa de entradas e apresenta tenta apresentar os seis valores cadastrados. Observe o código seguinte:

```
1 | TextWindow.Clear()
2 | TextWindow.WriteLine("ACESSO CONTROLADO - V#2.0")
3 |
4 | Arquivo = "C:\Dados1\arqteste2.txt"
5 |
6 | TextWindow.WriteLine("")
7 | TextWindow.WriteLine("Primeira bateria de entrada")
8 | TextWindow.WriteLine("")
9 | For I = 1 To 3
10 | TextWindow.Write("Entre o " + I + "o. valor: ")
11 | ValorEntra[I] = TextWindow.ReadNumber()
12 | File.WriteLine(Arquivo, I, ValorEntra[I])
13 | EndFor
14 |
15 | TextWindow.WriteLine("")
16 | TextWindow.WriteLine("Segunda bateria de entrada")
17 | TextWindow.WriteLine("")
18 | For J = 1 To 3
19 | TextWindow.Write("Entre o " + J + "o. valor: ")
20 | ValorEntra[J] = TextWindow.ReadNumber()
21 | File.WriteLine(Arquivo, J, ValorEntra[J])
22 | EndFor
23 |
24 | TextWindow.WriteLine("")
25 | TextWindow.WriteLine("Apresentação dos dados informados")
26 | TextWindow.WriteLine("")
```

```

27 | For K = 1 To 6
28 | ValorSai[K] = File.ReadLine(Arquivo, K)
29 | TextWindow.WriteLine(K + "o. valor: ")
30 | TextWindow.WriteLine(ValorSai[K])
31 | EndFor
32 |
33 | TextWindow.WriteLine("")
34 | TextWindow.Write("Tecle <Enter> para encerrar... ")
35 | Enter = TextWindow.Read()
36 | Program.End()

```

Grave o programa com o nome "**Cap05Ex49**", coloque-o em execução e entre a sequência de valores solicitadas. Veja os resultados apresentados na figura 5.38.

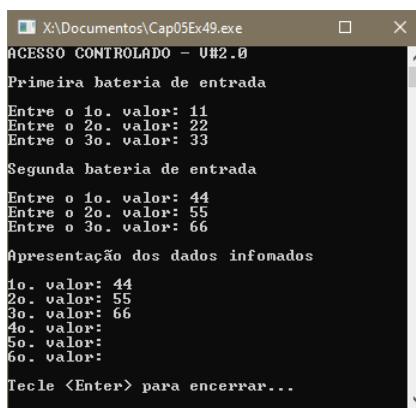


Figura 5.38 - Resultado da operação "WriteLine()"

Ao comparar os resultados apresentados nas figuras 5.37 e 5.38 ficam evidente a diferença entre o uso das operações **InsertLine()** e **WriteLine()**.

Observe o próximo programa que efetua a entrada do cadastro de valores numéricos inteiros e armazena este conteúdo em arquivo por meio da operação **WriteLine()**. A entrada do programa é controlada por ação de repetição interativa em que o usuário determina quando parar a entrada de valores.

Outro detalhe deste programa é a apresentação da lista de valores cadastrada no arquivo por meio da operação **Text()** do objeto **TextWindow** ao invés de fazer uso da operação **TextLine()**. Isso cria o inconveniente de após apresentar um dado não ser realizado o salto de linha para o próximo valor, e nesse sentido é que entramos com um macete curioso realizando o salto manualmente.

O programa também é operado a partir de dois arquivos: um chamado "**arqdat.txt**" que armazena os valores numéricos e o outro chamado "**arqdat.txt**" que mantém a quantidade de registros cadastrados no arquivo "**arqdat.txt**". Desta forma, tem a quantidade de registros sempre atualizada e disponível para uso pelo trecho de apresentação da listagem de valores.

Atente para o fato de que para o programa exemplo estamos utilizando diversos recursos da linguagem que foram apresentados ao longo das páginas anteriores deste livro. Veja que chegamos a um ponto em que temos em uso quase todos os recursos apresentados.

Observe atentamente o código seguinte e os detalhes indicados:

```

1 | TextWindow.Clear()
2 | TextWindow.WriteLine("ACESSO CONTROLADO A ARQUIVO")
3 | TextWindow.WriteLine("")

```

```
4
5 CR = Text.GetCharacter(13) ' Código para Carriage Return
6 LF = Text.GetCharacter(10) ' Código para Line Feed
7
8 Arquivo1 = "C:\Dados1\posdat.txt"
9 Arquivo2 = "C:\Dados1\arqdat.txt"
10
11 Posicao = File.ReadLine(Arquivo1, 1)
12
13 Resp = "S"
14 While (Resp = "S")
15 Posicao = Posicao + 1
16 TextWindow.WriteLine("Entre o " + Posicao + "o. dado: ")
17 Valor = Math.Floor(TextWindow.ReadNumber())
18 File.WriteLine(Arquivo2, Posicao, Valor)
19 TextWindow.WriteLine("")
20 Refaça:
21 TextWindow.WriteLine("Deseja continuar? Responda: 'S' ou 'N': ")
22 Resp = Text.ToUpperCase(TextWindow.Read())
23 If (Resp <> "S" And Resp <> "N") Then
24 Goto Refaça
25 EndIf
26 TextWindow.WriteLine("")
27 EndWhile
28
29 TextWindow.WriteLine("EXIBIÇÃO DE DADOS")
30 TextWindow.WriteLine("")
31 For J = 1 To Posicao
32 Valor = File.ReadLine(Arquivo2, J)
33 TextWindow.WriteLine(J + "o. dado: " + Valor + CR + LF)
34 EndFor
35 File.WriteLine(Arquivo1, 1, Posicao)
36
37 TextWindow.WriteLine("")
38 TextWindow.WriteLine("Tecle <Enter> para encerrar... ")
39 Enter = TextWindow.Read()
40 Program.End()
```

Grave o programa com o nome "**Cap05Ex50**", coloque-o em execução e entre a sequência de valores solicitadas. Veja os resultados apresentados de duas execuções do programa em separado junto a figura 5.39.

Um detalhe importante no programa é a definição das variáveis "**CR**" e "**LF**" respectivamente nas linhas "5" e "6" que são usadas na linha "33" com a concatenação após a escrita na tela do dado armazenado na variável "**Valor**". Este recurso serve para executar a mudança de linha após a impressão. Normalmente este efeito é produzido pelo uso da operação **WriteLine()** do objeto **Text**, mas foi a oportunidade para mostrar como o efeito é produzido. A variável "**CR**" representa o efeito "*Carriage Return*" (retorno de carro, que posiciona o cursor no início da linha) e "**LF**" representa o efeito "*Line Feed*" (mudança do cursor da linha atual para a próxima

linha). Vale salientar que os efeitos "CR" e "LF" também ocorrem quando se utiliza a tecla "<Enter>".

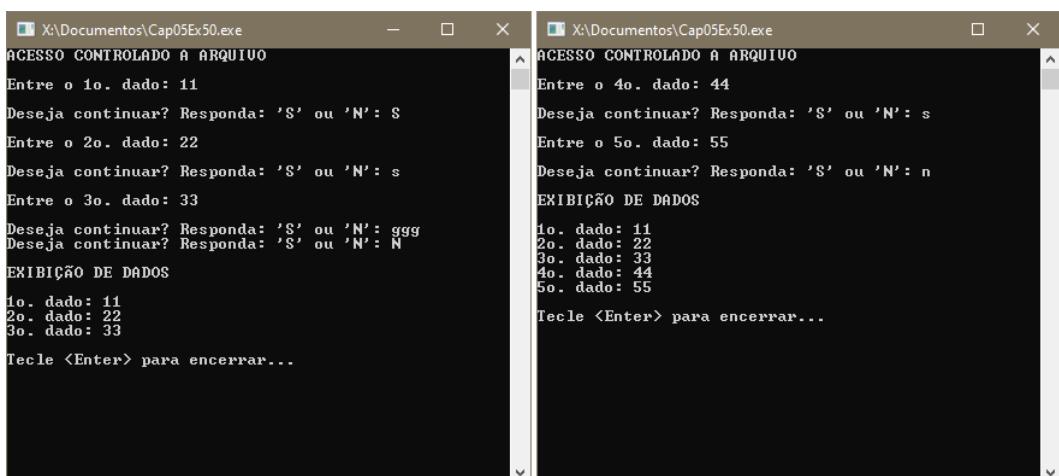


Figura 5.39 - Resultado da execução do programa

As linhas **"7"** e **"8"** definem as variáveis "**Arquivo1**" para o arquivo "**posdat.txt**" (quantidade das posições de dados usadas no segundo arquivo) e "**Arquivo2**" para o arquivo "**arddat.txt**" (cadastro dos dados propriamente ditos) de modo que tenha acesso e controle a esses arquivos.

A linha **"11"** faz a leitura do único registro do arquivo "**posdat.txt**" e atualiza com o valor lido a variável "**Posicao**". O arquivo "**posdat.txt**" tem por finalidade manter gravado a quantidade de dados que o arquivo "**arddat.txt**" possui.

Entre as linhas **"13"** e **"27"** encontra-se o trecho de cadastro de valores no arquivo "**arddat.txt**". Veja que a primeira ação mais importante deste trecho é fazer na linha **"15"** a atualização do contador de posições que foi lido e estabelecido na linha **"11"**.

Após a entrada do dado na linha **"17"** a instrução da linha **"18"** efetiva a gravação do valor informado no arquivo "**arddat.txt**".

As linhas de **"21"** a **"25"** controlam a parte interativa do programa.

Quando encerrada a interação de entradas o programa é executado a partir da linha **"29"** e se estende até a linha **"34"**. Neste trecho o arquivo é lido e seu conteúdo é apresentado na forma de uma listagem.

Observe que a linha **"31"** faz uso para o comando **For** do valor limite atualizado da variável "**Posicao**". É este o controle usado para gerar a lista de valores a partir da leitura do dado no arquivo "**arddat.txt**" por intermédio da linha **"32"** e a apresentação do valor na linha **"33"**.

Outra ação muito importante do programa é a execução da linha **"35"** que grava a quantidade de registros do arquivo "**arddat.txt**" no arquivo "**posdat.txt**". É aqui que a mágica acontece. É onde se faz o preparado do valor da quantidade de registros de "**arddat.txt**" para uma próxima execução do programa.

Além do conjunto de operações apresentados o objeto **File** possui uma propriedade chamada **LastError** que detecta ocorrência de erro sobre arquivos devolvendo o valor "**FAILED**" quando algo de errado ocorre ou "**SUCCESS**" quando ocorre a ação de acesso a um arquivo como esperado. Além de capturar um erro esta propriedade pode indicar o motivo da ocorrência do erro. Erros podem ocorrer por vários motivos, os três mais comuns, são:

1. Caminho informado para acesso ao arquivo inexistente;

2. O arquivo a ser usado está aberto em outro programa;
3. Falta de espaço em disco para salvar o arquivo.

Veja no programa seguinte a apresentação dos valores "SUCCESS" quando uma ação é bem sucedida ou "FAILED" quando certa ação de acesso a determinado arquivo não é bem sucedida a partir do uso da propriedade **LastError**. Para validar acessos tanto faz usar o valor "SUCCESS" como o valor "FAILED":

```
1 TextWindow.Clear()
2 TextWindow.WriteLine("VERIFICAÇÃO DE ACESSO")
3 TextWindow.WriteLine("")
4
5 Arquivo1 = "C:\Dados1\arqteste1.txt"
6
7 If (File.LastError = "FAILED") Then
8 TextWindow.WriteLine("FALHA: Arquivo não existe - " + Arquivo1)
9 Else
10 TextWindow.WriteLine("SUCESSO: Arquivo ok - " + Arquivo1)
11 EndIf
12
13 Arquivo2 = "C:\Dados1\arqtesteX.txt"
14
15 If (File.LastError = "SUCCESS") Then
16 TextWindow.WriteLine("SUCESSO: Arquivo ok - " + Arquivo2)
17 Else
18 TextWindow.WriteLine("FALHA: Arquivo não existe - " + Arquivo2)
19 EndIf
20
21 TextWindow.WriteLine("")
22 TextWindow.Write("Tecle <Enter> para encerrar... ")
23 Enter = TextWindow.Read()
24 Program.End()
```

Grave o programa com o nome "**Cap05Ex51**", coloque-o em execução e veja na sequência as indicações de falha e sucesso no acesso aos arquivos indicados, como indicado na figura 5.40.

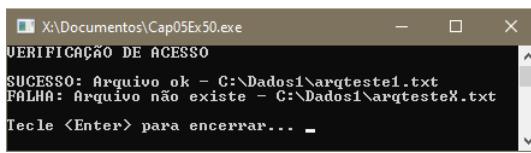


Figura 5.40 - Resultado de acessos a arquivos

Mas, não é só a propriedade **LastError** que devolve "SUCCESS" ou "FAILED", há operações que fazem o mesmo, como por exemplo: *AppendContents()*, *CopyFile()*, *CreateDirectory()*, *DeleteDirectory()*, *DeleteFile()*, *InserLine()*, *WriteContents()* e *WriteLine()*. Há outras que retornam apenas "FAILED", como: *GetDirectories()* e *GetFiles()*.

Veja no programa seguinte a tentativa de abertura de um arquivo em um diretório inexistente:

```
1 Arquivo = "C:\xpto\arqdat.txt"
2 Resultado = File.WriteAllText(Arquivo, "Small Basic")
3 TextWindow.WriteLine(Resultado + ": " + File.LastError)
4
```

```
5 | TextWindow.WriteLine("")
6 | TextWindow.Write("Tecla <Enter> para encerrar... ")
7 | Enter = TextWindow.Read()
8 | Program.End()
```

Grave o programa com o nome "**Cap05Ex52**", coloque-o em execução e veja o resultado da tentativa de acessar um arquivo inexistente em um diretório inexistente, como indicado na figura 5.41.



Figura 5.41 - Erro no acesso ao arquivo

### 5.8 - Exercícios de fixação

1. Desenvolver programa que leia oito elementos inteiros em uma matriz "**A**" de uma dimensão. Construir matriz "**B**" com os elementos da matriz "**A**" multiplicados por três. Apresentar a matriz "**B**".
2. Desenvolver programa que leia duas matrizes "**A**" e "**B**" de uma dimensão com cinco elementos numéricos. Construir uma matriz "**C**", sendo cada elemento a subtração de um elemento correspondente da matriz "**A**" com um elemento correspondente da matriz "**B**". Ao final, apresentar os elementos da matriz "**C**".
3. Desenvolver programa que leia cinco elementos numéricos em uma matriz "**A**" unidimensional e construa uma matriz "**B**" com os mesmos elementos da matriz "**A**" dispostos de forma invertida. Ou seja, o primeiro elemento da matriz "**A**" passa a ser o último da matriz "**B**", o segundo elemento da matriz "**A**" passa a ser o penúltimo da matriz "**B**", e assim por diante. Apresentar os elementos das matrizes "**A**" e "**B**".
4. Desenvolver programa que efetue a leitura de uma palavra ou frase qualquer e apresente o conteúdo da entrada de forma invertida. Dica: considere usar "**Text.GetSubText()**".
5. Desenvolver programa que efetue a leitura de uma frase qualquer e apresente a quantidade de caracteres da frase, excetuando-se os espaços em branco.
6. Desenvolver programa que efetue a leitura de um valor numérico inteiro maior que zero e escreva cada um de seus dígitos em extenso. Para "123", apresente "um, dois, três.".
7. Desenvolver programa que efetue a leitura de uma frase e conte a quantidade de palavras existentes na frase informada.
8. Desenvolver programa que efetue a leitura de uma palavra e apresente-a em sentido vertical (um caractere por linha).
9. Desenvolver programa que efetue a leitura de uma frase e apresente frase e a quantidade de vogais existentes. Considere a frase em maiúsculo mesmo que não seja fornecida.
10. Desenvolver programa que efetue a leitura de uma frase e apresente a frase informada sem espaços em branco e em formato maiúsculo.

ANOTAÇÕES

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Apêndice A

### Imagen da capa

A imagem usada para o desenho da capa deste livro foi inspirada a partir da proposta de código de programa em "Python" no sítio "ProgrammerSough" disponível no endereço: <https://www.programmersought.com/article/92794745509/> (figura A.1).

```
#Drawing colorful spirals
import turtle
def draw_spin():
 colores = [
 'red', 'purple', 'blue',
 'green', 'yellow', 'orange'
]
 turtle.bgcolor('black')
 for x in range(200):
 turtle.pencolor(colores[x % 6])
 turtle.width(x / 100 + 1)
 turtle.forward(x)
 turtle.left(59)
draw_spin()
input()
```

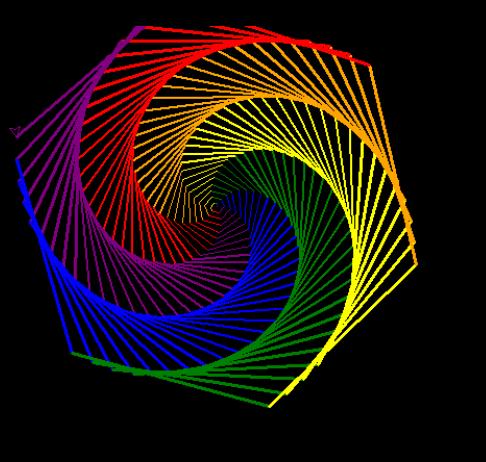


Figura A.1 - Programa Python e imagem do sítio "ProgrammerSough"

Veja a versão do programa "Python" escrito e adaptado para a linguagem "Small Basic":

```
1 GraphicsWindow.Width = 1728
2 GraphicsWindow.Height = 972
3 Turtle.Speed = 10
4 Turtle.X = 864
5 Turtle.Y = 461
6 GraphicsWindow.BackgroundColor = "Black"
7 Sub DrawSpin
8 For x = 0 To 199
9 If (Math.Remainder(x, 6) = 0) Then
10 GraphicsWindow.PenColor = "Red"
11 ElseIf (Math.Remainder(x, 6) = 1) Then
12 GraphicsWindow.PenColor = "Purple"
13 ElseIf (Math.Remainder(x, 6) = 2) Then
14 GraphicsWindow.PenColor = "Blue"
15 ElseIf (Math.Remainder(x, 6) = 3) Then
16 GraphicsWindow.PenColor = "Green"
17 ElseIf (Math.Remainder(x, 6) = 4) Then
18 GraphicsWindow.PenColor = "Yellow"
19 ElseIf (Math.Remainder(x, 6) = 5) Then
20 GraphicsWindow.PenColor = "Orange"
21 EndIf
22 GraphicsWindow.PenWidth = x / 100 + 1
23 Turtle.Move(x)
24 Turtle.Turn(-59)
```

```
25 | EndFor
26 | Turtle.Hide()
27 | EndSub
28 | DrawSpin()
```

Grave o código com o nome "**ApendA01**", execute-o e veja junto figura A.2 o resultado apresentado:

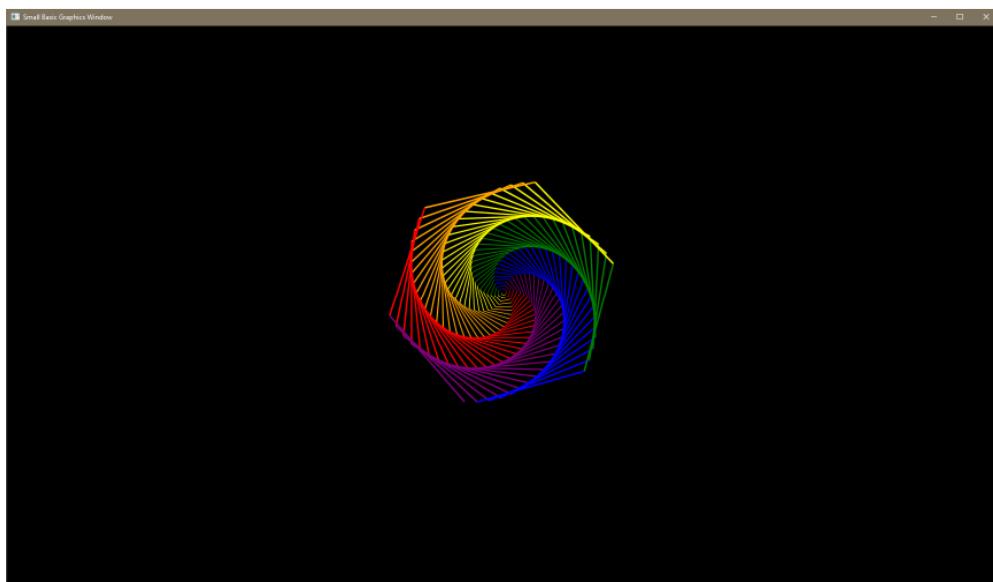


Figura A.1 - Imagem em "Small Basic"

Veja o que realiza cada instrução do programa a partir dos comandos "*Small Basic*" e sua relação com os comandos "*Python*":

- **Sub** e **EndSub** são usadas para definir o escopo de escrita do código da sub-rotina, sendo isso compatível ao comando **def**;
- A instrução **GraphicsWindow.BackgroundColor = "Black"** faz a mudança da cor do fundo da área de trabalho para a cor preto, similar a instrução **turtle.bgcolor('black')**;
- O comando **For** efetua a execução de duzentas passagens do grupo de instruções subordinadas contadas de "0" até "199" antes de encontrar o comando **Turtle.Hide()** estando de acordo com a instrução "**for x in range(200):**" que conta de "0" até "199";
- Dentro do escopo de ação do comando **For** encontra-se definida uma sequência de instruções **If** com **ElseIf** que detectam os valores do resto da divisão do conteúdo da variável "x" por "6" (que é a quantidade de cores em uso) que geram valores de resto entre "0" e "5" para a detecção e uso da cor definida para cada linha traçada do hexágono, estando este conjunto de instruções consoante a instrução "**turtle.pencolor(colores[x % 6])**". Apesar de "*Small Basic*" operar com o uso de variáveis compostas (matrizes), não aceita uma instrução semelhante a "*Python*" como **colores[x % 6]**;
- A instrução **GraphicsWindow.PenWidth = x / 100 + 1** tem por finalidade a partir da propriedade **PenWidth** mudar a largura do traço do desenho em andamento estando está instrução em consonância com a instrução **turtle.width(x/100 + 1)**;
- A instrução **Turtle.Move(x)** faz a apresentação do traço com valores crescentes na variável "x" de "0" até "199" sendo compatível com a instrução **turtle.forward(x)**;
- A instrução **Turtle.Turn(-59)** faz o giro em graus a esquerda para a formação de uma figura hexagonal, pois "59" é o valor numérico inteiro mais próximo de "60" que são os graus de formação de um hexágono estando de acordo com a instrução **turtle.left(59)**.

## Apêndice B

### Tabela de cores

Foi apresentado, neste livro, o uso de algumas cores junto a linguagem "Small Basic" tanto na área de texto como na área gráfica.

Para a área de texto **TextWindow** são oferecidas 16 cores indexadas e nomeadas como já documentado, mas para a área gráfica **GraphicsWindow** a quantidade de cores é bem maior e é importante tomar alguns cuidados, pois entre essas áreas existem cores iguais com nomes diferentes, como: a cor **Green** em **TextWindow** é igual a **Lime** em **GraphicsWindow**. A recomendação que se utilize nomes de tons diferentes quando se tem as duas janelas em operação. Outro detalhe é que escrever os nomes dos elementos da linguagem em notação camelCase ou outra forma não os diferencia.

A área gráfica opera com 140 cores nomeadas (não há a possibilidade de usar as cores a partir da indicação do valor de índice como se faz na área de texto). Além das 140 cores nomeadas é possível a partir do código **RGB** fazer uso de todo o espectro de cores suportadas. Veja as 140 cores nomeada disponíveis com seus respectivos códigos **RGB** em notação hexadecimal.

| Tom púrpura | Lavender        | #E6E6FA | Tom verde | GreenYellow       | #ADFF2F |
|-------------|-----------------|---------|-----------|-------------------|---------|
|             | Thistle         | #D8BFD8 |           | Chartreuse        | #7FFF00 |
|             | Plum            | #DDA0DD |           | LawnGreen         | #7CFC00 |
|             | Violet          | #EE82EE |           | Lime              | #00FF00 |
|             | Orchid          | #DA70D6 |           | LimeGreen         | #32CD32 |
|             | Fuchsia         | #FF00FF |           | PaleGreen         | #98FB98 |
|             | Magenta         | #FF00FF |           | LightGreen        | #90EE90 |
|             | MediumOrchid    | #BA55D3 |           | MediumSpringGreen | #00FA9A |
|             | MediumPurple    | #9370DB |           | SpringGreen       | #00FF7F |
|             | BlueViolet      | #8A2BE2 |           | MediumSeaGreen    | #3CB371 |
|             | DarkViolet      | #9400D3 |           | SeaGreen          | #2E8B57 |
|             | DarkOrchid      | #9932CC |           | ForestGreen       | #228B22 |
|             | DarkMagenta     | #8B008B |           | Green             | #008000 |
|             | Purple          | #800080 |           | DarkGreen         | #006400 |
|             | Indigo          | #4B0082 |           | YellowGreen       | #9ACD32 |
|             | SlateBlue       | #6A5ACD |           | OliveDrab         | #6B8E23 |
|             | DarkSlateBlue   | #483D8B |           | Olive             | #808000 |
|             | MediumSlateBlue | #7B68EE |           | DarkOliveGreen    | #556B2F |
|             |                 |         |           | MediumAquamarine  | #66CDAE |
|             |                 |         |           | DarkSeaGreen      | #8FBC8F |
|             |                 |         |           | LightSeaGreen     | #20B2AA |
|             |                 |         |           | DarkCyan          | #008B8B |
|             |                 |         |           | Teal              | #008080 |

|              |                 |         |            |                 |         |
|--------------|-----------------|---------|------------|-----------------|---------|
| Tom azul     | Aqua            | #00FFFF | Tom marrom | Cornsilk        | #FFF8DC |
|              | Cyan            | #00FFFF |            | BlanchedAlmond  | #FFEBCD |
|              | LightCyan       | #E0FFFF |            | Bisque          | #FFE4C4 |
|              | PaleTurquoise   | #AFEEEE |            | NavajoWhite     | #FFDEAD |
|              | Aquamarine      | #7FFFD4 |            | Wheat           | #F5DEB3 |
|              | Turquoise       | #40E0D0 |            | BurlyWood       | #DEB887 |
|              | MediumTurquoise | #48D1CC |            | Tan             | #D2B48C |
|              | DarkTurquoise   | #00CED1 |            | RosyBrown       | #BC8F8F |
|              | CadetBlue       | #5F9EA0 |            | SandyBrown      | #F4A460 |
|              | SteelBlue       | #4682B4 |            | Goldenrod       | #DAA520 |
|              | LightSteelBlue  | #B0C4DE |            | DarkGoldenrod   | #B8860B |
|              | PowderBlue      | #B0E0E6 |            | Peru            | #CD853F |
|              | LightBlue       | #ADD8E6 |            | Chocolate       | #D2691E |
|              | SkyBlue         | #87CEEB |            | SaddleBrown     | #8B4513 |
|              | LightSkyBlue    | #87CEFA |            | Sienna          | #A0522D |
|              | DeepSkyBlue     | #00BFFF |            | Brown           | #A52A2A |
|              | DodgerBlue      | #1E90FF |            | Maroon          | #800000 |
|              | CornflowerBlue  | #6495ED |            |                 |         |
|              | MediumSlateBlue | #7B68EE |            |                 |         |
|              | RoyalBlue       | #4169E1 |            |                 |         |
|              | Blue            | #0000FF |            |                 |         |
|              | MediumBlue      | #0000CD |            |                 |         |
|              | DarkBlue        | #00008B |            |                 |         |
|              | Navy            | #000080 |            |                 |         |
|              | MidnightBlue    | #191970 |            |                 |         |
| Tom vermelho | IndianRed       | #CD5C5C | Tom rosa   | Pink            | #FFC0CB |
|              | LightCoral      | #F08080 |            | LightPink       | #FFB6C1 |
|              | Salmon          | #FA8072 |            | HotPink         | #FF69B4 |
|              | DarkSalmon      | #E9967A |            | DeepPink        | #FF1493 |
|              | LightSalmon     | #FFA07A |            | MediumVioletRed | #C71585 |
|              | Crimson         | #DC143C |            | PaleVioletRed   | #DB7093 |
|              | Red             | #FF0000 |            |                 |         |
|              | FireBrick       | #B22222 |            |                 |         |
|              | DarkRed         | #8B0000 |            |                 |         |

|                    |                     |          |                    |                             |         |
|--------------------|---------------------|----------|--------------------|-----------------------------|---------|
| <b>Tom laranja</b> | <b>LightSalmon</b>  | #FFA07A  | <b>Tom amarelo</b> | <b>Gold</b>                 | #FFD700 |
|                    | <b>Coral</b>        | #FF7F50  |                    | <b>Yellow</b>               | #FFFF00 |
|                    | <b>Tomato</b>       | #FF6347  |                    | <b>LightYellow</b>          | #FFFFE0 |
|                    | <b>OrangeRed</b>    | #FF4500  |                    | <b>LemonChiffon</b>         | #FFFACD |
|                    | <b>DarkOrange</b>   | #FF8C00  |                    | <b>LightGoldenrodYellow</b> | #FAFAD2 |
|                    | <b>Orange</b>       | #FFA500  |                    | <b>PapayaWhip</b>           | #FFEFD5 |
| <b>Tom branco</b>  | <b>White</b>        | #FFFFFF  | <b>Tom cinza</b>   | <b>Moccasin</b>             | #FFE4B5 |
|                    | <b>Snow</b>         | #FFFFAFA |                    | <b>PeachPuff</b>            | #FFDAB9 |
|                    | <b>Honeydew</b>     | #F0FFF0  |                    | <b>PaleGoldenrod</b>        | #EEE8AA |
|                    | <b>MintCream</b>    | #F5FFFA  |                    | <b>Khaki</b>                | #F0E68C |
|                    | <b>Azure</b>        | #F0FFFF  |                    | <b>DarkKhaki</b>            | #BDB76B |
|                    | <b>AliceBlue</b>    | #F0F8FF  |                    | <b>Gainsboro</b>            | #DCDCDC |
|                    | <b>GhostWhite</b>   | #F8F8FF  |                    | <b>LightGray</b>            | #D3D3D3 |
|                    | <b>WhiteSmoke</b>   | #F5F5F5  |                    | <b>Silver</b>               | #C0C0C0 |
|                    | <b>Seashell</b>     | #FFF5EE  |                    | <b>DarkGray</b>             | #A9A9A9 |
|                    | <b>Beige</b>        | #F5F5DC  |                    | <b>Gray</b>                 | #808080 |
|                    | <b>OldLace</b>      | #FDF5E6  |                    | <b>DimGray</b>              | #696969 |
|                    | <b>FloralWhite</b>  | #FFFAF0  |                    | <b>LightSlateGray</b>       | #778899 |
|                    | <b>Ivory</b>        | #FFFFFF0 |                    | <b>SlateGray</b>            | #708090 |
|                    | <b>AntiqueWhite</b> | #FAEBD7  |                    | <b>DarkSlateGray</b>        | #2F4F4F |
|                    | <b>Linen</b>        | #FAF0E6  |                    | <b>Black</b>                | #000000 |

Para saber mais sobre aspectos de colorização na linguagem "Small Basic" acesse a página no endereço: <https://social.technet.microsoft.com/wiki/contents/articles/23227.small-basic-color.aspx>.

**ANOTAÇÕES**

## Apêndice C

### Arranjos e pilhas

Uma das ações apresentadas neste livro foi a utilização de variáveis indexadas. Você deve lembrar que uma variável indexada representa a possibilidade de armazenar diversos valores basicamente em uma mesma variável ou quase isso.

Uma variável simples armazena um só valor em sua posição e tem seu valor sobreposto por outro que nela seja colocado. Já as variáveis indexadas conseguem armazenar mais de um valor, mas não na mesma posição de memória. Uma variável indexada é uma região de memória que recebe um nome de identificação e cria a partir desse nome *slots* (posições de armazenamento) que são usadas para o armazenamento de valores. Cada posição tem um índice de identificação, que no caso da linguagem "*Small Basic*" pode ser definido a partir de um valor numérico iniciado em "1" ou definido a partir de um rótulo alfanumérico. Desta forma, entenda como sendo uma variável indexada uma coleção de valores armazenados em mesmo nome de forma contigua.

Não há uma quantidade previamente definida de elementos suportados por variáveis indexadas na linguagem "*Small Basic*", pois esta linguagem opera suas matrizes dinamicamente. Ao mesmo tempo em que isso é atraente pois facilita muito o desenvolvimento de programas, traz outras questões como, por exemplo, como saber a quantidade de elementos já informados? É aqui que entra o uso do objeto **Array** que auxilia o gerenciamento de parte do uso das variáveis indexadas na forma de listas (uma dimensão).

O objeto **Array** é uma ferramenta de apoio no armazenamento de mais de um elemento em variáveis, neste caso, indexadas que disponibiliza nove operações comentadas na tabela seguinte:

| OPERAÇÃO        | SIGNIFICADO                                         |
|-----------------|-----------------------------------------------------|
| ContainsIndex() | Checa se a matriz possui o índice especificado      |
| ContainsValue() | Checa se a matriz possui o valor especificado       |
| GetAllIndices() | Retorna em uma matriz os índices da matriz indicada |
| GetItemCount()  | Retorna o número de elementos existentes na matriz  |
| IsArray()       | Detecta se a variável indica é uma matriz           |

Veja a seguir alguns exemplos que demonstram isoladamente cada uma das operações apontadas.

O programa a seguir verifica se uma variável é ou não uma matriz:

```
1 | Nome = "José das Couves"
2 | Idade[1] = 25
3 |
4 | TextWindow.WriteLine(Array.IsArray(Nome))
```

```

5 | TextWindow.WriteLine(Array.isArray(Idade))
6 |
7 | TextWindow.WriteLine("")
8 | TextWindow.Write("Tecle <Enter> para encerrar... ")
9 | Enter = TextWindow.Read()
10| Program.End()

```

Grave o programa com o nome "**ApendB01**", coloque-o em execução e veja na figura B.1 seu resultado. Note que na primeira linha é apresentado "**False**" indicando que a variável "**Nome**" não é matriz, mas mostra na segunda linha "**True**" apontando que a variável "**Idade**" é matriz.

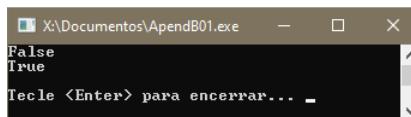


Figura B.1 - Identificação do que é ou não matriz

O programa a seguir confirma se determinado índice fornecido está ou não presente na matriz:

```

1 | Valor[1] = "Banana"
2 | Valor[2] = "Abacate"
3 |
4 | TextWindow.WriteLine(Array.ContainsIndex(Valor, 1))
5 | TextWindow.WriteLine(Array.ContainsIndex(Valor, 3))
6 |
7 | TextWindow.WriteLine("")
8 | TextWindow.Write("Tecle <Enter> para encerrar... ")
9 | Enter = TextWindow.Read()
10| Program.End()

```

Grave o programa com o nome "**ApendB02**", coloque-o em execução e veja na figura B.2 seu resultado. Note que na primeira linha é apresentado "**True**" indicando que a variável "**Valor**" possui um índice denominado "**1**", mas mostra na segunda linha "**False**" indicando que a variável "**Valor**" não possui um índice denominado "**3**".

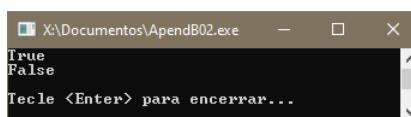


Figura B.2 - Identificação de existência de índice

O programa a seguir confirma se determinado elemento (valor) fornecido está ou não presente na matriz:

```

1 | Valor[1] = "Banana"
2 | Valor[2] = "Abacate"
3 |
4 | TextWindow.WriteLine(Array.ContainsValue(Valor, "Ferrari"))
5 | TextWindow.WriteLine(Array.ContainsValue(Valor, "Banana"))
6 |
7 | TextWindow.WriteLine("")
8 | TextWindow.Write("Tecle <Enter> para encerrar... ")
9 | Enter = TextWindow.Read()
10| Program.End()

```

Grave o programa com o nome "**ApendB03**", coloque-o em execução e veja na figura B.3 seu resultado. Note que na primeira linha é apresentado "**False**" indicando que o elemento "**Banana**" está na matriz.

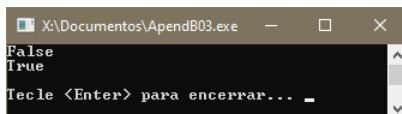


Figura B.3 - Identificação de existência de valor (elemento)

O programa a seguir retorna a quantidade de elementos existentes na matriz:

```
1 | 1 Valor[1] = "Banana"
2 | 2 Valor[2] = "Abacate"
3 | 3 Valor[3] = "Maça"
4 |
5 | 5 TextWindow.WriteLine(Array.GetItemCount(Valor))
6 |
7 | 7 TextWindow.WriteLine("")
8 | 8 TextWindow.Write("Tecle <Enter> para encerrar... ")
9 | 9 Enter = TextWindow.Read()
10|10 Program.End()
```

Grave o programa com o nome "**ApendB04**", coloque-o em execução e veja na figura B.4 seu resultado com a indicação da quantidade de elementos, neste caso "**3**".



Figura B.4 - Quantidade de elementos na matriz

O programa a seguir duplica os conteúdos de uma matriz em outra:

```
1 | 1 Valor[1] = "Banana"
2 | 2 Valor[2] = "Abacate"
3 | 3 Valor[3] = "Maça"
4 |
5 | 5 Produto = Array.GetAllIndices(Valor)
6 |
7 | 7 For I = 1 To Array.GetItemCount(Valor)
8 | 8 TextWindow.WriteLine("Produto[" + I + "] = " + Produto[I])
9 | 9 EndFor
10|
11|11 TextWindow.WriteLine("")
12|12 TextWindow.Write("Tecle <Enter> para encerrar... ")
13|13 Enter = TextWindow.Read()
14|14 Program.End()
```

Grave o programa com o nome "**ApendB05**", coloque-o em execução e veja na figura B.5 seu resultado.

Veja que os índices da matriz "**Valor**" foram duplicados junto a matriz "**Produto**" a partir do uso da operação  **GetAllIndices()** na linha "**5**". Veja que na linha "**7**" está sendo usado para detectar a quantidade de elementos na matriz "**Produto**" a operação  **GetItemCount()**. Atente para o fato de que a matriz "**Produto**" possui os índices da matriz "**Valor**" e não seus elementos.

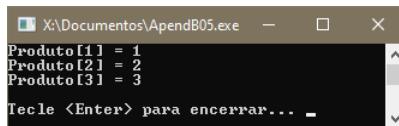


Figura B.5 - Resultado da duplicação de uma matriz

Como exemplo de uso de algumas operações do objeto **Array** considere um programa que receba a entrada de valores numéricos em uma matriz "A" em que os elementos vão sendo fornecidos e confirmados pelo usuário se deseja mais alguma entrada na matriz. Assim que as entradas forem encerradas o programa deve construir a partir da matriz "A" uma matriz chamada "B" e apresentar sem armazenar na memória os valores da matriz "B" multiplicados por "2". Veja o código a seguir, atente para as marcações em tom vermelho:

```
1 TextWindow.WriteLine("ENTRADA DE VALORES")
2 TextWindow.WriteLine("")
3
4 I = 1
5 Resp = "S"
6 While (Resp = "S")
7 TextWindow.Write("Entre A[" + I + "] --> ")
8 A[I] = TextWindow.ReadNumber()
9 Repita:
10 TextWindow.WriteLine("")
11 TextWindow.Write("Deseja continuar? Responda: 'S' ou 'N': ")
12 Resp = Text.ConvertToUpperCase(TextWindow.Read())
13 If (Resp <> "S" And Resp <> "N") Then
14 Goto Repita
15 EndIf
16 I = I + 1
17 TextWindow.WriteLine("")
18 EndWhile
19
20 TextWindow.WriteLine("SAIDA DE VALORES")
21 TextWindow.WriteLine("")
22
23 For J = 1 To Array.GetItemCount(A)
24 B[J] = A[J]
25 EndFor
26
27 For K = 1 To Array.GetItemCount(B)
28 TextWindow.WriteLine("B[" + K + "] = " + B[K] * 2)
29 EndFor
30
31 TextWindow.WriteLine("")
32 TextWindow.Write("Tecle <Enter> para encerrar... ")
33 Enter = TextWindow.Read()
34 Program.End()
```

Grave o programa com o nome "**ApendB06**", coloque-o em execução e veja na figura B.6 seu resultado.

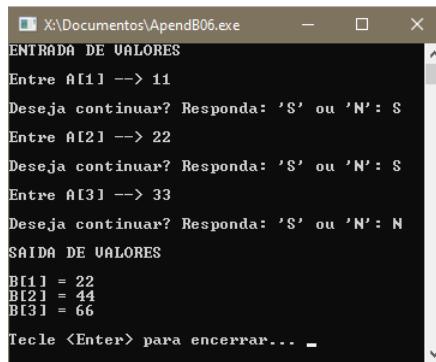


Figura B.6 - Manipulação de matrizes

Note que no trecho entre as linhas "4" e "18" é estabelecida a entrada de valores na matriz "A". Veja que não é definido quantos elementos serão acrescidos na matriz. A quantidade de entradas depende do usuário controlado pelas linhas de "11" até "15".

Vale neste ponto de nossa apresentação salientar um detalhe. Todas as operações existentes na versão 1.2 da linguagem "Small Basic" para manipulação de matrizes foram demonstradas. Mas se você fizer uma pesquisa pela Web encontrará a definição de mais operações que não foram até então citadas. Trata-se das operações **GetValue()** que retorna o elemento de um determinado índice de uma matriz, **RemoveValue()** que remove um determinado elemento a partir do índice de uma matriz e **SetValue()** que insere um elemento a partir de um índice de uma matriz.

Essas operações são consideradas na versão 1.2 da linguagem obsoletas (ou talvez preteridas), ou seja, foram deixadas de lado e não são mais indicadas para uso. Quando uma linguagem indica em certa versão a definição de recursos como obsoletos ou preteridos dá a indicação de que em um futuro esses recursos poderão não estar mais disponíveis, principalmente se o recurso estiver considerado preterido. Veja a descrição de uso dessas operações:

```
Array.GetValue(matriz, índice)
Array.RemoveValue(matriz, índice)
Array.SetValue(matriz, índice, valor)
```

Onde, o parâmetro "**matriz**" é o nome da matriz em operação definida entre aspas inglesas, o parâmetro "**índice**" é a posição da matriz que realizará a operação sobre determinado elemento e no caso particular da operação **SetValue()** o parâmetro "**valor**" o elemento propriamente dito.

O programa a seguir demonstra o uso dessas operações obsoletas a título apenas de ilustração e curiosidade aplicados em uma matriz chamada "A" de cinco elementos numéricos, que terá a retira de um dos elementos e a apresentação dos elementos restantes. Veja o programa e atente para os trechos sinalizados em tom vermelho:

```
1 | TextWindow.WriteLine("ENTRADA DE VALORES")
2 |
3 | TextWindow.WriteLine("")
4 | For I = 1 To 5
5 | TextWindow.Write("Entre o elemento " + I + ": ")
6 | Array.SetValue("A", I, TextWindow.ReadNumber())
7 | EndFor
8 |
9 | Array.RemoveValue("A", 3)
10|
```

```
11 TextWindow.WriteLine("")
12 For J = 1 To 5
13 TextWindow.WriteLine("Elemento " + J + " = " + Array.GetValue("A", J))
14 EndFor
15
16 TextWindow.WriteLine("")
17 TextWindow.Write("Tecle <Enter> para encerrar... ")
18 Enter = TextWindow.Read()
19 Program.End()
```

Grave o programa com o nome "**ApendB07**", coloque-o em execução e faça a entrada de cinco valores e veja o resultado ocorrido como demonstra a figura B.7.

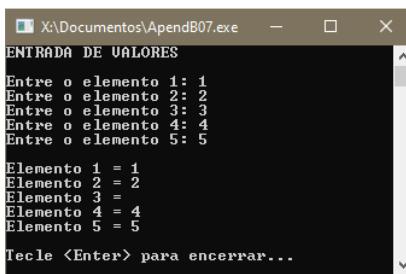


Figura B.7 - Uso de operações obsoletas

Quando elementos são armazenados em matrizes estes podem ser manipulados, por exemplo, sequencialmente como demonstrado em todos os programas apresentados até este ponto. Mas há outras formas de fazer manipulações de elementos armazenados em memória como, por exemplo, operacionalizar uma estrutura de dados como uma pilha e não como matriz, e neste caso, entra em uso o objeto **Stack**.

O conceito de pilha ocorre com a entrada de valores em uma matriz e sua retirada ocorre a partir do último elemento inserido. Uma pilha opera segundo o critério LIFO (*Last In First Out*), ou seja, o último que entra é o primeiro que sai.

O objeto **Stack** é o que possui o menor número de recursos na linguagem contanto com três operações como segue:

| OPERAÇÃO    | SIGNIFICADO                                             |
|-------------|---------------------------------------------------------|
| GetCount()  | Retorna a quantidade de elementos da pilha especificada |
| PopValue()  | Retira um elemento da pilha especificada                |
| PushValue() | Adiciona um elemento na pilha especificada              |

Se fornecidos para uma pilha os valores "1", "2", "3", "4" e "5" a saída ocorrerá automaticamente como sendo "5", "4", "3", "2" e "1".

Veja a seguir um programa que demonstra o uso da operação de dados em uma pilha. Atente para as partes sinalizadas em vermelho e verde. Atenta para o uso do nome da pilha referenciado entre aspas inglesas. O programa aceita valores numéricos inteiros diferentes de zero. Quando zero é informado o programa encerra a entrada e mostra os dados armazenados na pilha.

```

1 TextWindow.WriteLine("PILHA - ENTRADA DE VALORES")
2 TextWindow.WriteLine("-----")
3 TextWindow.WriteLine("")
4 TextWindow.WriteLine("Informe <0> para encerrar a entrada.")
5 TextWindow.WriteLine("")
6
7 I = 1
8 N = 9
9 While (N <> 0)
10 TextWindow.Write("Entre o " + I + "o. elemento: ")
11 N = TextWindow.ReadNumber()
12 If (N <> 0) Then
13 Stack.PushValue("A", N)
14 EndIf
15 I = I + 1
16 EndWhile
17 TextWindow.WriteLine("")
18
19 TextWindow.WriteLine("PILHA - SAIDA DE VALORES")
20 TextWindow.WriteLine("-----")
21 TextWindow.WriteLine("")
22
23 Tamanho = Stack.GetCount("A")
24
25 For J = 1 To Tamanho
26 TextWindow.WriteLine(J + "o. elemento = " + Stack.PopValue("A"))
27 EndFor
28
29 TextWindow.WriteLine("")
30 TextWindow.Write("Tecle <Enter> para encerrar... ")
31 Enter = TextWindow.Read()
32 Program.End()

```

Grave o programa com o nome "**ApendB08**", coloque-o em execução e faça a entrada de alguns valores e veja o resultado ocorrido como mostra a figura B.8.

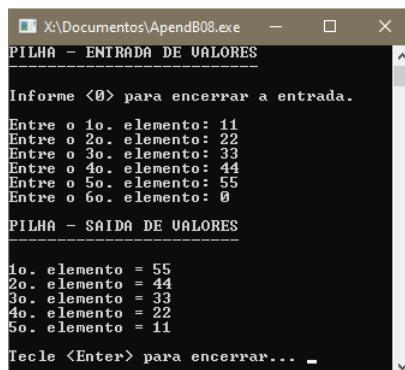


Figura B.8 - Operação de uso de pilha

Note que o programa "**ApendB08**" demonstra na linha "**13**" o uso da entrada de dados na pilha, na linha "**23**" detecta a quantidade de elementos da pilha e na linha "**26**" faz a retirada de cada elemento da pilha.

Um cuidado tomado no programa foi o uso da variável "**Tamanho**" na linha "**23**" que tem nela atribuído o valor da quantidade de elementos na pilha a partir do uso da operação **GetCount()** e serve como base para o limite da operação da linha "**25**". Não é possível fazer uso da operação **GetCount()** diretamente na linha "**25**", pois a operação **PopValue()** definida na linha "**26**" após sua execução retira um elemento da pilha e isso faz com que o tamanho da pilha seja modificado e interferiria na execução da linha "**25**".

## Apêndice D

### Passagem de parâmetro e retorno

Você deve ter percebido que a maioria das operações existentes nos vários objetos da linguagem "Small Basic" fazem uso de parâmetros. Um parâmetro na programação "Small Basic" é representado por uma variável usada exclusivamente por uma operação que recebe a definição de um valor fornecido dentro de parênteses. O valor quando passado é chamado de argumento, ou seja, o parâmetro é a indicação do que pode ser recebido por uma operação e o argumento é o valor efetivamente recebido. Por exemplo, a operação **WriteLine** do objeto **TextWindow** para sua ação recebe como parâmetro um determinado **dado** que representa uma mensagem ou resultado de alguma operação a ser escrito, podendo este ser expresso a partir da definição de concatenação.

O fato é que o efeito de passagem de parâmetro em "Small Basic" somente é usado nas operações definidas nos objetos da linguagem. Quando você escreve uma sub-rotina com os comandos **Sub / EndSub** (que é em essência, uma operação customizada por você) a linguagem não permite que você faça uso de parâmetros. Isso limita um pouco as coisas, mas há uma forma de contornar essa questão fazendo a *simulação de passagem de parâmetro*.

Para simular passagem de parâmetro basta criar dentro de certa sub-rotina as variáveis específicas que recebem os valores que são "passados" como "argumentos" e que a partir destes é realizada as operações.

No sentido de deixar mais claro essa proposta vamos considerar uma sub-rotina que apresente o resultado do cálculo da factorial de um valor numérico inteiro positivo qualquer. Veja o código de programa seguinte e atente as partes colorizadas:

```
1 Sub Fatorial
2 CalcFat = 1
3 If (LimiteFat < 0) Then
4 TextWindow.WriteLine("Entrada inválida")
5 ElseIf (LimiteFat = 0) Then
6 TextWindow.WriteLine(1)
7 Else
8 For Sequencia = 1 To LimiteFat
9 CalcFat = CalcFat * Sequencia
10 EndFor
11 TextWindow.WriteLine(CalcFat)
12 EndIf
13 EndSub
14
15 TextWindow.WriteLine("CALCULO DE FATORIAL")
16 TextWindow.WriteLine("")
17
18 TextWindow.Write("Entre um valor inteiro positivo: ")
19 N = Math.Floor(TextWindow.ReadNumber())
20
21 LimiteFat = N
```

```
22 | TextWindow.WriteLine("Resultado = ")
23 | Fatorial()
24 |
25 | TextWindow.WriteLine("")
26 | TextWindow.WriteLine("Tecla <Enter> para encerrar... ")
27 | Enter = TextWindow.Read()
28 | Program.End()
```

Grave o programa com o nome "**ApendC01**", coloque-o em execução, entre os valores "**5**", "**0**" e "**-3**" e veja na figura C.1 sua operação e resultado. O programa aceita valores entre "**1**" e "**27**". Valores acima de "**27**" abortam a execução do programa.

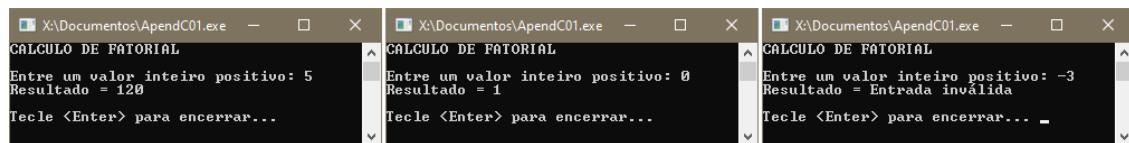


Figura C.1 - Simulação de passagem de parâmetro (resultados)

As sub-rotinas em "*Small Basic*" são estruturas de programas que ficam com seus "pés" fincados entre os universos dos procedimentos e das funções, se parecendo mais com procedimentos, exceto pelo fato de aceitar realizar operações de recursividade que são naturais as funções. Talvez, o uso desse tipo de recursividade advém do fato de "*Small Basic*" dar suporte a simulação da linguagem "*Logo*" que opera somente com sub-rotinas do tipo procedimento a partir do objeto **Turtle**.

Um procedimento é um estilo de sub-rotina usada para dividir um programa complexo em partes menores e simplificadas. Em que cada parte resolve uma etapa da solução, obtendo-se ao final a solução do todo. Além dessa característica sub-rotinas servem para a implementação do reaproveitamento de código, pois permite escrever uma vez certo grupos de operações que podem ser repetidas (reaproveitadas) quantas vezes forem necessárias. Uma sub-rotina pode receber passagem de parâmetros para integralizar internamente certas ações de processamento (e é isso que falta em "*Small Basic*").

Note entre as linhas "**1**" e "**13**" a definição do código para o cálculo da factorial a partir da sub-rotinas **Fatorial()**. Veja na linha "**2**" a definição da variável "**CalcFat**" assemelhando-se ao uso de uma variável local ("*Small Basic*" não opera com variáveis locais, mas é um recurso que pode ser simulado adequadamente com o uso do recurso de pilha - veja isso em outro apêndice) que é usada para auxiliar a obtenção do resultado do cálculo da factorial junto a linha "**9**".

O que caracteriza a sub-rotina **Fatorial()** como um "procedimento" é a apresentação dos resultados de sua operação de forma interna como indicados nas linhas "**4**", "**6**" e "**11**".

A simulação da passagem de parâmetro ocorre com o uso da variável "**LimiteFat**" nas linhas da sub-rotina "**3**" e "**5**" e principalmente seu uso na parte principal do programa na linha "**21**" ao ser atribuído com o valor informado para a variável "**N**". É como se a variável "**N**" estivesse passando seu valor (argumento) para a variável da sub-rotina "**LimiteFat**".

Uma sub-rotina do tipo função tem basicamente as mesmas características de uma sub-rotina do tipo procedimento com uso ou não de passagem de parâmetros. A diferença entre sub-rotinas procedimento e função está no fato de que as sub-rotinas de função sempre retornam uma resposta a sua ação tenha essa sub-rotina passagem ou não de parâmetro.

A linguagem "Small Basic" também não dá suporte à definição de sub-rotinas de funções customizadas, mas é possível simulá-las. Veja o código a seguir de um programa que simula o uso de uma função para o cálculo de fatorial, atente para as partes colorizadas:

```
1 Sub Fatorial
2 CalcFat = 1
3 If (LimiteFat < 0) Then
4 RetornaFat = "Entrada inválida"
5 ElseIf (LimiteFat = 0) Then
6 RetornaFat = 1
7 Else
8 For Sequencia = 1 To LimiteFat
9 CalcFat = CalcFat * Sequencia
10 EndFor
11 RetornaFat = CalcFat
12 EndIf
13 EndSub
14
15 TextWindow.WriteLine("CALCULO DE FATORIAL")
16 TextWindow.WriteLine("")
17
18 TextWindow.Write("Entre um valor inteiro positivo: ")
19 N = Math.Floor(TextWindow.ReadNumber())
20
21 LimiteFat = N
22 Fatorial()
23
24 TextWindow.WriteLine("Resultado = " + RetornaFat)
25
26 TextWindow.WriteLine("")
27 TextWindow.Write("Tecla <Enter> para encerrar... ")
28 Enter = TextWindow.Read()
29 Program.End()
```

Grave o programa com o nome "**ApendC02**", coloque-o em execução, entre os valores "**5**", "**0**" e "**-3**" e veja os resultados apresentados semelhantes a imagem da figura C.1. Respeite o limite de entrada de dados já mencionado.

A simulação da definição de uma função ocorre com o uso da variável "**RetornaFat**" nas linhas "**4**", "**6**" e "**11**" da sub-rotina em substituição a apresentação direta dos resultados simulando o retorno do valor calculado pela sub-rotina e apresentado diretamente na linha "**24**" do trecho que representa a parte principal do programa. Veja que neste caso o resultado retornado é concatenado a cadeia "**Resultado =**".

## **ANOTAÇÕES**

## Apêndice E

### Variáveis locais

A linguagem "Small Basic" somente usa variáveis de escopo global. Uma variável global é um tipo de variável que quando definida fica disponível durante todo o tempo de execução do programa. Ao mesmo tempo em que isso é atraente faz com que a linguagem ocupe mais espaço de memória. Por isso linguagens de programação profissionais operam com o conceito de escopo e visibilidade de variáveis permitindo a definição de variáveis globais e locais, mas não a linguagem "Small Basic" que por ser uma linguagem de programação educacional não necessita deste tipo de recurso. Na prática, você possivelmente não verá aplicações comerciais executadas empresas e organizações escritas com a linguagem "Small Basic". Não que isso não seja até possível, mas não faz sentido perto de outras linguagens como Visual Basic, C#, Java, C, C++ entre outras.

No entanto, o fato de não operar variáveis locais abre a possibilidade de colocar em prática a ação de simulá-las a partir do uso do objeto **Stack** que também pode ser usada para simular passagem de parâmetros como será apresentado mais adiante neste apêndice.

Veja essa "deficiência" da linguagem como outra grande oportunidade de ampliar suas habilidades nas técnicas de programação. Vale ressaltar que uma variável local é um tipo de variável, normalmente definida dentro de uma sub-rotina, usada apenas dentro da sub-rotina que deve ser destruída da memória antes do encerramento da sub-rotina. Isso garante que um programa ocupe um espaço de memória menor do que faria se estivesse usando apenas variáveis globais.

Para demonstrar a simulação de uso de variáveis locais considere como exemplo o programa a seguir que a partir de uma sub-rotina de função simulada retorne o resultado do cálculo de uma fatorial. Observe as partes colorizadas do programa a seguir, veja o uso das pilhas "**Fat**" e "**I**" simulando o uso de variáveis locais e o uso da variável global "**RetornaFat**" como retorno de ação da sub-rotina de função:

```
1 Sub Fatorial
2 Stack.PushValue("Fat", 1)
3 If (LimiteFat < 0) Then
4 RetornaFat = "Entrada inválida"
5 ElseIf (LimiteFat = 0) Then
6 RetornaFat = 1
7 Else
8 Stack.PushValue("I", 1)
9 RetornaFat = Stack.PopValue("I")
10 While (RetornaFat <= LimiteFat)
11 Stack.PushValue("Fat", Stack.PopValue("Fat") * RetornaFat)
12 RetornaFat = RetornaFat + 1
13 EndWhile
14 RetornaFat = Stack.PopValue("Fat")
15 EndIf
16 EndSub
17
18 TextWindow.WriteLine("CALCULO DE FATORIAL")
```

```
19 | TextWindow.WriteLine("")
20 |
21 | TextWindow.Write("Entre um valor inteiro positivo: ")
22 | N = Math.Floor(TextWindow.ReadNumber())
23 |
24 | LimiteFat = N
25 | Fatorial()
26 |
27 | TextWindow.WriteLine("Resultado = " + RetornaFat)
28 |
29 | TextWindow.WriteLine("")
30 | TextWindow.Write("Tecle <Enter> para encerrar... ")
31 | Enter = TextWindow.Read()
32 | Program.End()
```

Grave o programa com o nome "**ApendD01**", coloque-o em execução, entre valores de "**1**" até "**27**" e veja os resultados apresentado.

O programa a partir da manipulação das pilhas "**Fat**" e "**I**" simula a aplicação de variáveis locais, evitando que variáveis globais para a manipulação dos valores de "**Fat**" e "**I**" fossem criadas.

A linha "**2**" equivale a escrever a instrução "**Fat = 1**". Veja que a operação **PushValue()** coloca dentro da pilha "**Fat**" o valor numérico "**1**".

As únicas variáveis globais no código são "**LimiteFat**" usada como passagem de parâmetro e "**RetornaFat**" usada como retorno da sub-rotina na simulação da ação de uma função.

Na linha "**8**" encontra-se a definição da instrução "**Stack.PushValue("I", 1)**" equivalente a escrever "**I = 1**" que prepara a ação para a execução da repetição definida entre as linhas "**10**" e "**13**" a partir do uso da variável "**RetornaFat**" na linha "**9**" que assume o valor que encontra-se armazenado na pilha "**I**" a partir da instrução "**RetornaFat = Stack.PopValue("I")**".

O cálculo da fatorial equivalente a "**Fat = Fat \* I**" é produzido pela instrução da linha "**11**" que ao mesmo tempo que retira o valor da pilha "**Fat**" com o trecho "**Stack.PopValue("Fat")**" multiplicando este valor pelo conteúdo da variável "**RetornaFat**" faz imediatamente seu armazenamento na filha com a operação **PushValue()** que engloba toda a ação.

Concluída a ação da repetição até o limite especificado junto a variável "**LimiteFat**" ocorre o processamento da linha "**14**" em que o resultado da fatorial guardado na pilha "**Fat**" e retirado da pilha e atribuído a variável "**RetornaFat**" antes do encerramento da sub-rotina.

Vimos aqui que variáveis locais são ideias para serem usadas dentro de sub-rotinas, tanto em estilo procedimento, como em estilo função. No entanto, os parâmetros de uma sub-rotina também podem, por sua natureza operacional, serem definidos como variáveis locais. É importante ter em mente que nem sempre será possível implementar totalmente essa ideia, há limitações na linguagem "*Small Basic*" que podem impedir essa ação como a proposta do programa de cálculo de fatorial. Mas isso deixarei para você experimentar.

Para simular o uso de parâmetros como variáveis locais na linguagem "*Small Basic*" considere o programa seguinte que apresenta o resultado de uma raiz de base qualquer de um índice qualquer simplificada com a simulação de passagem de parâmetro com variáveis globais. Atente para os pontos colorizados do programa:

```
1 Sub RaizEnesima
2 Stack.PushValue("RetornaRaiz",
3 Math.Power(Stack.PopValue("BaseRaiz"),
4 1 / Stack.PopValue("IndiceRaiz")))
5 EndSub
6
7
8 TextWindow.WriteLine("CALCULO DE RAIZ ENÉSIMA")
9 TextWindow.WriteLine("")
10
11 TextWindow.Write("Entre o valor da base: ")
12 Stack.PushValue("BaseRaiz", TextWindow.ReadNumber())
13
14 RaizEnesima()
15
16 TextWindow.WriteLine("Resultado = " + Stack.PopValue("RetornaRaiz"))
17
18 TextWindow.WriteLine("")
19 TextWindow.Write("Tecle <Enter> para encerrar... ")
20 Enter = TextWindow.Read()
21 Program.End()
22
23
```

Grave o programa com o nome "**ApendD02**", coloque-o em execução, entre, por exemplo, os valores de "**2**" até "**3**" e veja o resultado da raiz cúbica do valor "**2**" como "**1,25992104989487**".

Entre as linhas "**1**" e "**3**" é definida a sub-rotina em estilo função "**RaizEnesima**" que efetua o cálculo de uma raiz enésima a partir da operação **Power()** do objeto **Math** respectivamente com os parâmetros da base e índice da raiz recuperados a partir dos valores armazenados nas pilhas "**BaseRaiz**" e "**IndiceRaiz**" quando da ação das linhas "**9**" e "**12**". Após o cálculo o resultado é inserido na pilha "**RetornaRaiz**".

Veja que neste exemplo está se fazendo uso totalmente de pilhas. Nenhuma variável foi declarada para a operação do programa.

---

---

ANOTAÇÕES

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Apêndice F

### Classificação de dados

Uma das ações muito requisitadas na computação são as operações de classificação de dados usadas para colocar listas de valores em determinada ordem, podendo-se aplicar em dados numéricos, alfanuméricicos e alfabéticos.

A classificação numérica pode ocorrer de forma crescente (de "1" a "9") ou decrescente (de "9" a "1"). Já a classificação alfanumérica ou alfabética é dita de forma ascendente (de "A" a "Z") ou descendente (de "Z" a "A").

Para esta operação existem algoritmos muito populares. Uns mais rápidos e outros nem tanto. Os mais rápidos são mais complexos e, normalmente, mais difíceis de serem entendidos por iniciantes em programação. Os mais lentos são mais simples e consequentemente mais fáceis de serem entendidos.

O algoritmo aqui apresentado é um dos mais lentos, mas possui numa estrutura lógica bastante simples. Trata-se de um algoritmo que utiliza permuta entre valores a partir da aplicação do mesmo princípio usado para o estabelecimento de propriedades distributivas. Este método é idêntico ao algoritmo aplicado no exercício de fixação "7" do capítulo "2".

No sentido de demonstrar o uso dessas operações de modo contextualizado considere um programa que efetue a entrada de valores numéricos em uma matriz e os apresente classificados de modo crescente a partir de quantidade informada pelo usuário. Veja o programa seguinte:

```
1 TextWindow.WriteLine("Quantos elementos a entrar: ")
2 N = Math.Floor(TextWindow.ReadNumber())
3 TextWindow.WriteLine("")
4
5 For X = 1 To N
6 TextWindow.Write("Valor " + X + ": ")
7 Valor[X] = TextWindow.ReadNumber()
8 EndFor
9
10 For I = 1 To Array.GetItemCount(Valor) - 1
11 For J = I + 1 To Array.GetItemCount(Valor)
12 If (Valor[I] > Valor[J]) Then
13 V = Valor[I]
14 Valor[I] = Valor[J]
15 Valor[J] = V
16 EndIf
17 EndFor
18 EndFor
19
20 TextWindow.WriteLine("")
21 TextWindow.WriteLine("Os valores classificados são:")
22 TextWindow.WriteLine("")
23 For Y = 1 To Array.GetItemCount(Valor)
24 TextWindow.WriteLine(Valor[Y])
25 EndFor
```

```
26
27 TextWindow.WriteLine("")
28 TextWindow.Write("Tecle <Enter> para encerrar... ")
29 Enter = TextWindow.Read()
30 Program.End()
```

Grave o programa com o nome "**ApendF01**", coloque-o em execução e veja na figura F.1 seu resultado.

Observe no programa o trecho entre as linhas "6" e "9" que efetua a entrada dos dados na matriz. Veja que na linha "9" usa-se para controlar o número de repetições do comando **For** a variável "N" definida na linha "3".

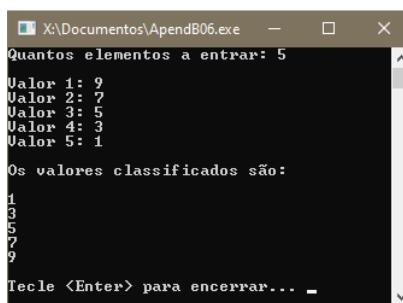


Figura F.1 - Resultado de matriz com números classificados

O trecho entre as linhas "11" e "19" é responsável pelo processamento da classificação, neste caso, crescente como descreve a condição do comando **If/Then** da linha "13" a partir do uso do operador relacional ">" (maior que). Veja que as linhas "11" e "12" definem os modos de repetição que aplicam a ideia da aplicação da propriedade distributiva. Considerando a figura B.6 que mostra a entrada de "5" valores as comparações realizadas são do índice "1" com os índices "2", "3", "4" e "5"; depois é realizada a comparação do índice "2" com os índices "3", "4" e "5"; em seguida do índice "3" com os índices "4" e "5"; e pôr fim do índice "4" com o índice "5". A cada ocorrência em que a condição da linha "13" é verdadeira ocorre a troca efetuada de valores entre as linhas "14" e "16".

O trecho de código entre as linhas "24" e "26" demonstra a apresentação dos valores da matriz devidamente classificados.

Agora que você viu como classificar uma lista de valores em certa ordem pode estar em sua mente a vontade de adaptar o programa "**ApendF01**" para classificar sequências alfabéticas ou alfanuméricas em ordem ascendente (de "A" até "Z") ou descendente (de "Z" até "A").

E em tese bastaria apenas mudar a forma de entrada de dados no programa e a saída classificada deveria funcionar, mas não na linguagem "*Small Basic*".

Essa classificação direta de dados com cadeias de caracteres é possível em outras linguagens de programação, mas "*Small Basic*" assim como a linguagem "C" não efetuam essa ação desta maneira, precisam de outro tratamento. Neste caso, é necessário escrever uma sub-rotina que trata separadamente cada caractere da cadeia fornecida.

Para dar uma ideia observe as palavras "**AMOR**" e "**AMAR**" que ao serem apresentadas classificadas de forma ascendente deveriam ser expressas como "**AMAR**" e "**AMOR**".

Para um teste inicial considere o programa seguinte que "tenta" realizar a apresentação das palavras em ordem ascendente:

```

1 A = "AMAR"
2 B = "AMOR"
3
4 If (A > B) Then
5 TextWindow.WriteLine(A)
6 TextWindow.WriteLine(B)
7 Else
8 TextWindow.WriteLine(B)
9 TextWindow.WriteLine(A)
10 EndIf
11
12 TextWindow.WriteLine("")
13 TextWindow.Write("Tecle <Enter> para encerrar... ")
14 Enter = TextWindow.Read()
15 Program.End()

```

Grave o programa com o nome "**ApendF02**", coloque-o em execução e veja na figura F.2 seu resultado.

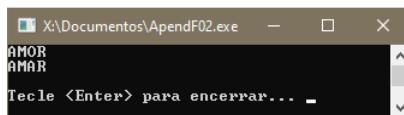


Figura F.2 - Resultado de classificação não efetuada

Veja que a saída esperada era a apresentação das palavras como "**AMAR**" e "**AMOR**" e não como "**AMOR**" e "**AMAR**". Isso mostra que a linguagem "*Small Basic*" não consegue avaliar duas cadeias de caracteres, ou seja, não "sabe" fazer isso, só "sabe" avaliar valores numéricos.

Quando uma linguagem não faz o que queremos é hora de entramos com nosso conhecimento e resolver o problema. Aliás é para isso que nos tornamos profissionais na área. Para resolver problemas.

Mas antes de ensinar um computador a fazer algo por meio de alguma linguagem é necessário que saibamos fazer esse algo. A propósito é o que esperamos neste ponto depois de estudarmos todo este conteúdo.

Observe como uma operação de classificação ascendente de cadeia alfabética ou mesmo alfanumérica deve fazer seu trabalho. Considere as variáveis "**A**" e "**B**" respetivamente com as cadeias "**ANDOR**" e "**ANDAR**". Veja como são armazenados esses valores na memória:

|   | 1   | 2   | 3   | 4   | 5   |
|---|-----|-----|-----|-----|-----|
| A | "A" | "N" | "D" | "O" | "R" |
| B | "A" | "N" | "D" | "A" | "R" |

Veja que uma cadeia é de fato um conjunto de caracteres, ou seja, é uma matriz de caracteres em que cada caractere ocupa determinada posição. Desta forma, o que precisa ser feito é pegar isoladamente cada caractere em sua respectiva posição, detectar o valor do seu código ASCII, fazer a comparação do valor do caractere da primeira cadeia com o valor do caractere da segunda cadeia e verificar se o primeiro caractere tem valor maior que o segundo caractere, sendo a condição verdadeira, faz-se a troca das cadeias entre as variáveis para conseguir uma

apresentação ascendente, mas se não for passa-se para a próxima posição. Durante a verificação de cada caractere da cadeia pode acontecer dessas cadeias terem tamanhos diferentes. Se os tamanhos entre as cadeias forem iguais ou a segunda cadeia for maior que a primeira basta seguir pelo tamanho da primeira cadeia e tudo fica certo. Mas se a primeira cadeia for maior que a segunda deve-se focar a atenção na segunda cadeia. Se "A[1] > B[1]" então troca-se as cadeias de modo que a variável "A" fique o valor da variável "B" e a variável "B" fique com o valor da variável "A", como realizado no exercício de fixação "7" do capítulo "2". Essa mesma operação deve ser realizada para as demais posições enquanto a condição de cada verificação não for verdadeira, pois no momento em que uma posição é verdadeira faz-se a troca dos valores das cadeias. Observe que a ocorrência de troca ocorre para as cadeias "ANDOR" e "ANDAR" quando a verificação chegar na condição "A[4] > B[4]".

Vejamos os valores ASCII de cada um dos caracteres das cadeias "ANDOR" e "ANDAR":

|   | 1   | 2  | 3   | 4  | 5   |    |     |    |     |    |
|---|-----|----|-----|----|-----|----|-----|----|-----|----|
| A | "A" | 65 | "N" | 78 | "D" | 68 | "O" | 79 | "R" | 82 |
| B | "A" | 65 | "N" | 78 | "D" | 68 | "A" | 65 | "R" | 82 |

Veja que as posições "1", "2" e "3" possuem valores iguais e nada será realizado. Mas a posição "4" possui para "A[4]" o valor ASCII "79" e para "B[4]" o valor ASCII "65". Veja que o valor "79" é maior que "65" e, neste caso, as cadeias serão trocadas entre as variáveis "A" e "B" proporcionando o efeito de classificação.

Agora que temos uma ideia de como isso pode ser feito vamos a parte em que devemos orientar a linguagem a realizar essa tarefa. Para tanto considere a seguir apenas o trecho de código da sub-rotina que faz a validação da verificação condicional entre duas cadeias:

```

1 Sub ComparCad
2 TamPriCad = Text.GetLength(PriNome)
3 TamSegCad = Text.GetLength(SegNome)
4 RetornoCmpCad = "False"
5 Pivo = "False"
6 Pos = 1
7 While Pos <= TamPriCad And Pos <= TamSegCad And Pivo <> "True"
8 CarPriCad = Text.GetCharacterCode(Text.GetSubText(PriNome, Pos, 1))
9 CarSegCad = Text.GetCharacterCode(Text.GetSubText(SegNome, Pos, 1))
10 If CarPriCad <> CarSegCad Then
11 Pivo = "True"
12 If CarPriCad >= CarSegCad Then
13 RetornoCmpCad = "True"
14 EndIf
15 EndIf
16 Pos = Pos + 1
17 EndWhile
18 If TamPriCad > TamSegCad And Pivo = "False" Then
19 RetornoCmpCad = "True"
20 EndIf
21 EndSub

```

Por uma questão de maior simplicidade a sub-rotina **ComparCad()** de comparação de cadeias definida entre as linhas "1" e "21" está fazendo uso de passagem de parâmetro e retorno apenas com o uso de variáveis globais. Mas, isso não impede de você tentar escrever a rotina com as técnicas já apresentadas neste tipo de implementação para simular uso de variáveis e parâmetros locais.

Veja nas linhas "2" e "3" a obtenção do tamanho de cada uma das cadeias respectivamente com as variáveis "**TamPriCad**" e "**TamPriCad**". Isso é importante para que se "saiba" até onde é possível ir.

Na linha "4" encontra-se a definição da variável "**RetornoCmpCad**" usada para simular o retorno de uma sub-rotina de função. Veja que a variável é iniciada com o valor "**False**" e tornar-se-á verdadeira com o valor "**True**" quando um caractere da primeira cadeia for maior que o caractere da segunda cadeia (linha "13") ou quando o tamanho da primeira cadeia for maior que o tamanho da segunda cadeia (linha "19").

A linha "5" faz uso da variável "**Pivo**" que tem por finalidade auxiliar a operação da sub-rotina. Essa variável é iniciada com valor "**False**" e só torna-se "**True**" se a operação da linha "10" for verdadeira, ou seja, se o caractere da primeira cadeia tiver maior valor que o caractere da segunda cadeia. A variável "**Pivo**" é importante para garantir a realização da operação definida na linha "18", pois se "**Pivo**" for "**False**" e o tamanho da primeira cadeia for maior que o tamanho da segunda cadeia considera o retorno da segunda cadeia a partir da definição da variável "**RetornoCmpCad**" com valor "**True**". Veja que mesmo o tamanho da primeira cadeia sendo maior que o tamanho da segunda cadeia ocorrerá a atualização da variável "**RetornoCmpCad**" com valor "**True**" somente se a variável "**Pivo**" estiver sinalizada com o valor "**False**".

Na linha "6" temos a definição da variável "**Pos**" usada para "pegar" cada uma das posições das cadeias em operação.

Veja que na linha "7" a busca por caracteres entre as cadeias somente ocorre se a condição da repetição for verdadeira, ou seja, a variável "**Pos**" deverá ser menor ou igual aos tamanhos da primeira e segunda cadeias e ter a variável "**Pivo**" diferente de "**True**". A ação da linha "7" se estende até a linha "17".

As linhas "8" e "9" são responsáveis por pegar o código ASCII de cada um dos caracteres das cadeias. A operação **GetSubText()** retorna o caractere e a operação **GetCharacterCode()** retorna o código ASCII do caractere atualizando respectivamente as variáveis "**CarPriCad**" caractere da primeira cadeia e "**CarSegCad**" caractere da segunda cadeia.

O trecho de linhas entre "10" e "15" verifica primeiro se os caracteres das cadeias em uso são diferentes, pois se forem diferentes a variável "**Pivo**" é atualizada para o valor "**True**" (linha "11") e tem-se o valor para encerrar a repetição e faz-se uma segunda verificação na linha "12" que se tiver o primeiro caractere com valor maior ou igual ao valor do segundo caractere deve atualizar a variável "**RetornoCmpCad**" com valor "**True**" para que a troca entre as variáveis seja efetiva no programa principal.

Na linha "16" é realizada a atualização do contador de posições que percorre as cadeias em verificação.

A última etapa da operação da sub-rotina é após ter passado pela verificação anterior é ver se a primeira cadeia é menor que a segunda cadeia e retomar a segunda cadeia desde que a variável "**Pivo**" tenha valor "**False**".

Na sequência veja o programa que mostra as palavras "**ANDOR**" e "**ANDAR**" classificadas.

```
1 Sub ComparCad
2 TamPriCad = Text.GetLength(PriCad)
3 TamSegCad = Text.GetLength(SegCad)
4 RetornoCmpCad = "False"
5 Pivo = "False"
6 Pos = 1
7 While (Pos <= TamPriCad And Pos <= TamSegCad And Pivo <> "True")
8 CarPriCad = Text.GetCharacterCode(Text.GetSubText(PriCad, Pos, 1))
9 CarSegCad = Text.GetCharacterCode(Text.GetSubText(SegCad, Pos, 1))
10 If (CarPriCad <> CarSegCad) Then
11 Pivo = "True"
12 If (CarPriCad >= CarSegCad) Then
13 RetornoCmpCad = "True"
14 EndIf
15 EndIf
16 Pos = Pos + 1
17 EndWhile
18 If (TamPriCad > TamSegCad And Pivo = "False") Then
19 RetornoCmpCad = "True"
20 EndIf
21 EndSub
22
23 A = "ANDOR"
24 B = "ANDAR"
25
26 PriCad = A
27 SegCad = B
28 ComparCad()
29 If (RetornoCmpCad = "True") Then
30 X = A
31 A = B
32 B = X
33 EndIf
34
35 TextWindow.WriteLine(A)
36 TextWindow.WriteLine(B)
37
38 TextWindow.WriteLine("")
39 TextWindow.Write("Tecle <Enter> para encerrar... ")
40 Enter = TextWindow.Read()
41 Program.End()
```

Grave o programa com o nome "**ApendF03**", coloque-o em execução e veja na figura F.3 seu resultado.



Figura F.3 - Resultado de classificação efetuada

Veja nas linhas "27" e "27" do trecho principal do programa a definição dos parâmetros "**PriCad**" e "**SegCad**" que recebem os valores definidos nas variáveis "A" e "B". Depois na linha "28" faz-se a chamada da sub=rotina que ao avaliar os conteúdos passados com "**PriCad**" e "**SegCad**" devolve o resultado de sua operação junto a variável de retorno "**RetornoCmpCad**" que ao estar com valor "**True**" faz entre as linhas "29" e "33" a troca dos valores permitindo apresentar os textos em ordem ascendente de classificação junto as linhas "35" e "36".

A partir do conhecimento de como fazer a classificação de dados alfabéticos ou alfanuméricos considere o programa a seguir que apresenta certa quantidade de nomes informados em ordem ascendente de classificação. Veja o código e observe os pontos colorizados:

```
1 Sub ComparCad
2 TamPriCad = Text.GetLength(PriCad)
3 TamSegCad = Text.GetLength(SegCad)
4 RetornoCmpCad = "False"
5 Pivo = "False"
6 Pos = 1
7 While (Pos <= TamPriCad And Pos <= TamSegCad And Pivo <> "True")
8 CarPriCad = Text.GetCharacterCode(Text.GetSubText(PriCad, Pos, 1))
9 CarSegCad = Text.GetCharacterCode(Text.GetSubText(SegCad, Pos, 1))
10 If (CarPriCad <> CarSegCad) Then
11 Pivo = "True"
12 If (CarPriCad >= CarSegCad) Then
13 RetornoCmpCad = "True"
14 EndIf
15 EndIf
16 Pos = Pos + 1
17 EndWhile
18 If (TamPriCad > TamSegCad And Pivo = "False") Then
19 RetornoCmpCad = "True"
20 EndIf
21 EndSub
22
23 TextWindow.Clear()
24 TextWindow.Write("Quantos nomes a entrar: ")
25 N = Math.Floor(TextWindow.ReadNumber())
26 TextWindow.WriteLine("")
27
28 For X = 1 To N
29 TextWindow.Write("Entre o " + X + "o. nome: ")
30 Nome[X] = TextWindow.Read()
31 EndFor
32
33 For I = 1 To Array.GetItemCount(Nome) - 1
34 For J = I + 1 To Array.GetItemCount(Nome)
35 PriCad = Nome[I]
36 SegCad = Nome[J]
37 ComparCad()
38 If (RetornoCmpCad = "True") Then
39 NomeTrc = Nome[I]
40 Nome[I] = Nome[J]
```

```
41 Nome[J] = NomeTrc
42 EndIf
43 EndFor
44 EndFor
45
46 TextWindow.WriteLine("")
47 TextWindow.WriteLine("Os nomes classificados são:")
48 TextWindow.WriteLine("")
49 For Y = 1 To Array.GetItemCount(Nome)
50 TextWindow.WriteLine(Nome[Y])
51 EndFor
52
53 TextWindow.WriteLine("")
54 TextWindow.Write("Tecle <Enter> para encerrar... ")
55 Enter = TextWindow.Read()
56 Program.End()
```

Grave o programa com o nome "**ApendF04**", coloque-o em execução e entre inicialmente a quantidade cinco nomes. Informe os dados "Silvio", "Antonia", "Mario", "Silvia" e "Maria". Veja na figura F.4 o resultado da classificação apresentado.

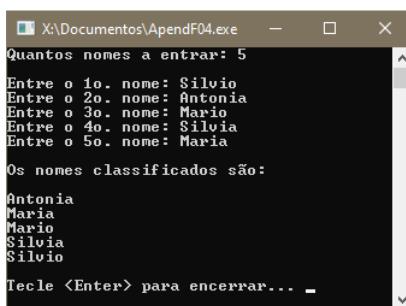


Figura F.4 - Resultado de classificação realizada

Perceba que é assim que um profissional de programação deve agir. Não existe a funcionalidade então cria-se. O que você jamais pode dizer é coisas do tipo: "*isso a linguagem não faz*"; "*nunca ninguém fez isso*" ou outra cosa parecida. Esse tipo de comportamento com certeza irá depor contra você e a mais ninguém.

Como dito desde o início do estudo deste livro é que a linguagem "*Small Basic*" não é comercial e sim educacional. Mas até mesmo em uma linguagem educacional como a "*Small Basic*" é possível realizar a implementação de novos recursos, até mesmo usá-la como ferramenta comercial, apesar de não ser indicada, pois existem linguagens mais apropriadas para isso. Depois, pesquise sobre a extensão "*LitDev*" disponível em "<http://litdev.co.uk/>" que acrescenta diversos recursos a linguagem como: física 2D; renderização 3D; controles; diálogos; gráficos; matrizes; listas; classificação e pesquisa (regex); conexão USB/COM; SQLite; MySQL; SqlServer; Odbc; OleDb; webcam; mais recursos para uso de GraphicsWindows; barras de rolagem; processamento de imagem; manipulação de data e hora; depurador; interação cliente-servidor; criptografia, área de transferência; música e gravação de som; arquivo zip/ftp; chamadas de sub-routine assíncronas e de argumento; código inline em C#/Javascript e muito mais.

## Apêndice G

### Extensão LitDev

Podemos dizer que a linguagem "*Small Basic*" é um grande sucesso da Microsoft, tanto que gerou uma boa legião de fãs, o suficiente para que fosse criado pela comunidade uma extensão de recursos para a linguagem dando-lhe um potencial acima do que fora projetada. Trata-se do projeto "*LitDev*" que pode ser obtido no endereço "<http://litdev.co.uk/>".

Não é o objetivo deste livro apresentar esta extensão, pois merece um livro a parte. O objetivo é mostrar a você que existe um pouco mais para usar "*Small Basic*" do que foi apresentado. A seguir é informado como obter a extensão e fazer sua instalação em seu computador. Assim sendo, acesse seu programa de navegação e entre no endereço indicado. Essa ação deve apresentar uma página semelhante à figura G.1.

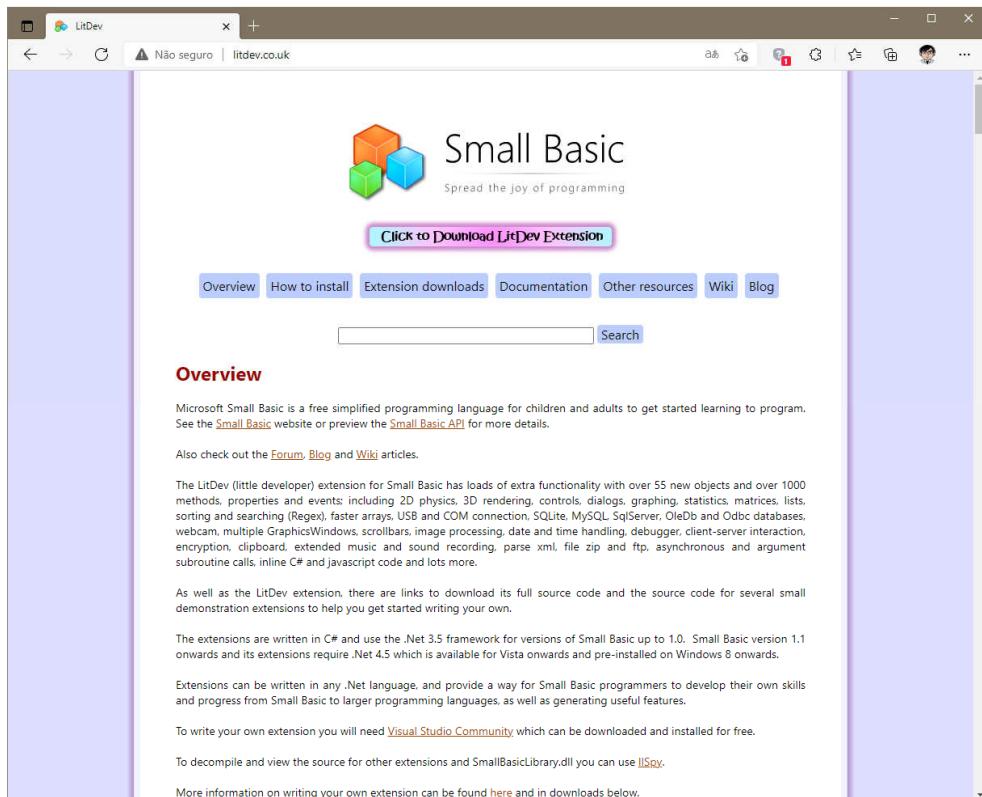


Figura G.1 - Sítio oficial do projeto de extensão "LitDev"

Role a página um pouco para baixo e aione a imagem indicada na figura G.2 para dar início ao processo de cópia da extensão. Aguarde a operação de cópia terminar.

O processo de instalação não é automático, então tenha muita atenção no que está fazendo e siga exatamente o que é descrito.

Vá até a pasta "**Downloads**" é descompacte o arquivo "*zipado*". Entre na pasta descompactada e copie os arquivos "*LitDev.dll*" e "*LitDev.xml*" para o local onde o ambiente "*Small Basic*" está instalado, ou seja, "**C:\Program Files (x86)\Microsoft\Small Basic\Lib**". Talvez seja necessário criar a pasta "**Lib**" caso não seja localizada em "**C:\Program Files (x86)\Microsoft\Small Basic\**".

Os demais arquivos podem ser copiados livremente para outro local. Um bom lugar é a partir da localização "**C:\Program Files (x86)\Microsoft\Small Basic\**".

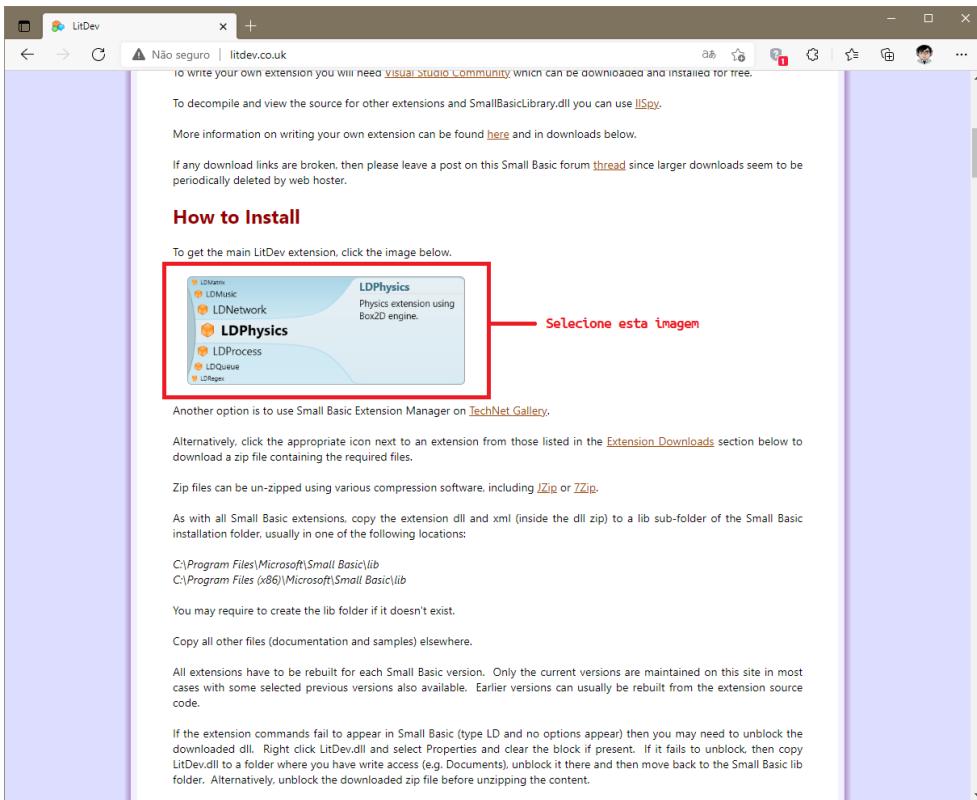


Figura G.2 - Opção para obtenção da extensão "LitDev"

Agora carregue o ambiente "*Small Basic*" escreva no editor de textos as letras "**LD**". Veja a apresentação dos objetos iniciados com essas letras. Note que neste momento você possui os recursos da extensão com as explicações grafadas em inglês e provavelmente os recursos da linguagem em português.

Para maiores detalhes e outros esclarecimentos consulte o sítio do projeto "*LitDev*".

A título de ilustração veja o programa seguinte que apresenta uma mensagem criptografada de acordo com o algoritmo "**AES**" (*Advanced Encryption Standard*) após ser fornecida via teclado. Para o uso do recurso além da mensagem informada é necessário a definição de uma senha para a operação:

```

1 TextWindow.WriteLine("CRIPTOGRAFIA - AES")
2 TextWindow.WriteLine("")
3
4 TextWindow.Write("Entre uma mensagem ...: ")
5 Mensagem = TextWindow.Read()
6
7 TextWindow.Write("Entre uma senha: ")
8 Senha = TextWindow.Read()
9
10 CrptEAS = LDEncryption.AESEncrypt(Mensagem, Senha)
11 DcptEAS = LDEncryption.AESDecrypt(CrptEAS, Senha)
12
13 TextWindow.WriteLine("")
14 TextWindow.WriteLine("EAS - Cripto: " + CrptEAS)
15
16 TextWindow.WriteLine("")
```

```

17 | TextWindow.WriteLine("EAS - Descri: " + DcptEAS)
18 |
19 | TextWindow.WriteLine(""))
20 | TextWindow.Write("Tecle <Enter> para encerrar... ")
21 | Enter = TextWindow.Read()
22 | Program.End()

```

Grave o programa com o nome "**ApendG01**", coloque-o em execução e veja na figura G.1 seu resultado. Para um teste, entre por exemplo a mensagem "**Mensagem teste para criptografia.**" E informe a senha "**Bololo**".

Figura G.1 - Resultado de uso de criptografia e descriptografia

Veja que o uso da extensão "*LitDev*" ocorre de forma transparente como se fosse um recurso natural do próprio ambiente "*Small Basic*".

Além de operações de criptografia com o objeto **LDEncryption** há operações para a criação de *hashs* como o exemplo seguinte utilizando-se o algoritmo "**MD5**":

```

1 | Nome1 = "Augusto"
2 | Nome2 = "Augusta"
3 | Nome3 = "Auguste"
4 |
5 | TextWindow.WriteLine("HASHS - PADRÃO: MD5")
6 | TextWindow.WriteLine(""))
7 | TextWindow.WriteLine(Nome1 + " = " + LDEncryption.MD5Hash(Nome1))
8 | TextWindow.WriteLine(Nome2 + " = " + LDEncryption.MD5Hash(Nome2))
9 | TextWindow.WriteLine(Nome3 + " = " + LDEncryption.MD5Hash(Nome3))
10 |
11 | TextWindow.WriteLine(""))
12 | TextWindow.Write("Tecle <Enter> para encerrar... ")
13 | Enter = TextWindow.Read()
14 | Program.End()

```

Grave o programa com o nome "**ApendG02**", coloque-o em execução e veja na figura G.2 seu resultado. Perceba que mesmo que os textos sejam parecidos o *hash* é completamente diferente.

Figura G.2 - Resultado de uso de hash

---

---

ANOTAÇÕES

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Apêndice H

## Gabarito

### CAPÍTULO 2

1. Desenvolver programa 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.

```
TextWindow.Write("Entre um valor numérico inteiro: ")
N = Math.Floor(TextWindow.ReadNumber())
TextWindow.WriteLine("Resultado = " + Math.Power(N, 2))
```

2. Desenvolver programa 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.

```
TextWindow.Write("Entre valor um valor numérico inteiro: ")
N = Math.Floor(TextWindow.ReadNumber())
R = Math.Power(N, 3)
TextWindow.WriteLine("Resultado = " + R)
```

3. Desenvolver programa 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.

```
TextWindow.Write("Entre graus Celsius: ")
C = TextWindow.ReadNumber()
F = C * 9 / 5 + 32
TextWindow.WriteLine("Resultado em Fahrenheit = " + F)
```

4. Desenvolver programa 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.

```
TextWindow.Write("Entre graus Fahrenheit: ")
F = TextWindow.ReadNumber()
C = ((F - 32) * 5) / 9
TextWindow.WriteLine("Resultado em Celsius = " + C)
```

5. Desenvolver programa que efetue a leitura de dois valores numéricos (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".

```
TextWindow.WriteLine("Entre um valor numérico para <A>: ")
A = TextWindow.ReadNumber()
TextWindow.WriteLine("Entre um valor numérico para : ")
B = TextWindow.ReadNumber()
R = Math.Power(A - B, 2)
TextWindow.WriteLine("Resultado = " + R)
```

6. Desenvolver programa que efetue a leitura de um valor numérico qualquer e apresente como resultado os valores sucessor e antecessor do valor numérico fornecido. Não armazene em memória os valores calculados.

```
TextWindow.WriteLine("Entre um valor numérico: ")
N = TextWindow.ReadNumber()
TextWindow.WriteLine("Sucessor: " + (N + 1))
TextWindow.WriteLine("Antecessor ...: " + (N - 1))
```

7. Desenvolver programa que efetue a leitura de qualquer valor para as variáveis "A" e "B", efetue a troca dos valores de modo que no final a variável "A" tenha o valor da variável "B" e a variável "B" tenha o valor da variável "A". Apresentar os novos valores de cada uma das variáveis.

```
TextWindow.WriteLine("Entre um valor <A>: ")
A = TextWindow.Read()
TextWindow.WriteLine("Entre um valor : ")
B = TextWindow.Read()
X = A
A = B
B = X
TextWindow.WriteLine("<A> = " + A)
TextWindow.WriteLine(" = " + B)
```

**CAPÍTULO 3**

---

1. Desenvolver programa 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 de qualquer comando de repetição.

```
Turtle.Move(60)
Turtle.Turn(90)
Turtle.Move(100)
Turtle.Turn(90)
Turtle.Move(60)
Turtle.Turn(90)
Turtle.Move(100)
Turtle.Turn(90)
```

2. Desenvolver programa que desenhe um retângulo com lados de tamanhos "60" e "100" para trás com giro de graus à esquerda. Usar o comando **For**.

```
For I = 1 To 2
 Turtle.Move(60)
 Turtle.Turn(90)
 Turtle.Move(100)
 Turtle.Turn(90)
EndFor
```

3. Criar, sem uso de qualquer comando de repetição, um programa que construa um pentágono com tamanho "40". Avance para frente com giro a direita.

```
Turtle.Move(40)
Turtle.Turn(72)
Turtle.Move(40)
Turtle.Turn(72)
Turtle.Move(40)
Turtle.Turn(72)
Turtle.Move(40)
Turtle.Turn(72)
Turtle.Move(40)
Turtle.Turn(72)
```

4. Desenvolver programa que construa uma figura com dez lados a partir do uso do comando **For** com giro de graus à esquerda com avanço a frente de "30" passos.

```
For I = 1 To 10
 Turtle.Move(30)
 Turtle.Turn(36)
EndFor
```

5. Desenvolver programa que efetue a leitura de dois valores numéricos representados pelas variáveis "A" e "B" e apresente o resultado da diferença do maior valor pelo menor valor sem armazenar o resultado em memória.

```
TextWindow.WriteLine("Entre valor <A>: ")
A = TextWindow.ReadNumber()
TextWindow.WriteLine("Entre valor : ")
B = TextWindow.Read()
If (A > B) Then
 TextWindow.WriteLine("Resultado = " + (A - B))
Else
 TextWindow.WriteLine("Resultado = " + (B - A))
EndIf
```

6. Desenvolver programa que efetue a leitura de dois valores numéricos representados pelas variáveis "A" e "B" e apresente o resultado da diferença do maior valor pelo menor valor com armazenamento do cálculo em memória.

```
TextWindow.WriteLine("Entre valor <A>: ")
A = TextWindow.ReadNumber()
TextWindow.WriteLine("Entre valor : ")
B = TextWindow.Read()
R = A - B
If (A > B) Then
 TextWindow.WriteLine("Resultado = " + R)
Else
 TextWindow.WriteLine("Resultado = " + R)
EndIf
```

7. Desenvolver programa que efetue a leitura de três valores numéricos inteiros desconhecidos representados pelas variáveis "A", "B" e "C". Efetue a soma desses valores, armazene o resultado em memória e apresente o resultado somente se este for ">=" a "100".

```
TextWindow.WriteLine("Entre valor <A>: ")
A = TextWindow.ReadNumber()
TextWindow.WriteLine("Entre valor : ")
B = TextWindow.Read()
TextWindow.WriteLine("Entre valor <C>: ")
C = TextWindow.Read()
R = A + B + C
If (R >= 100) Then
 TextWindow.WriteLine("Resultado = " + R)
EndIf
```

8. Desenvolver programa que efetue a leitura de três valores numéricos inteiros desconhecidos representados pelas variáveis "A", "B" e "C". Apresentar os valores dispostos em ordem crescente.

```
TextWindow.WriteLine("Entre valor <A>: ")
A = TextWindow.ReadNumber()
TextWindow.WriteLine("Entre valor : ")
B = TextWindow.Read()
TextWindow.WriteLine("Entre valor <C>: ")
C = TextWindow.Read()
If (A > B) Then
 X = A
 A = B
 B = X
EndIf
If (A > C) Then
 X = A
 A = C
 C = X
EndIf
If (B > C) Then
 X = B
 B = C
 C = X
EndIf
TextWindow.WriteLine("Os valores são: " + A + ", " + B + " e " + C)
```

**CAPÍTULO 4**

---

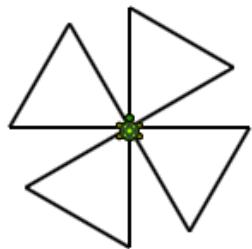
1. Criar programa que desenhe um retângulo cujo lado menor (comprimento) seja a metade do lado maior (altura) em que o tamanho seja informado pelo usuário. Movimente o desenho para traz com sentido a direita.

```
TextWindow.WriteLine("Entre um valor: ")
Tamanho = TextWindow.ReadNumber()
For I = 1 To 2
 Turtle.Move(-Tamanho)
 Turtle.Turn(90)
 Turtle.Move(-Tamanho / 2)
 Turtle.Turn(90)
EndFor
```

2. Crie um programa que apresente um triângulo equilátero com lado de tamanho "70" para frente a partir de giros a esquerda.

```
For I = 1 To 3
 Turtle.Move(-70)
 Turtle.Turn(-120)
EndFor
```

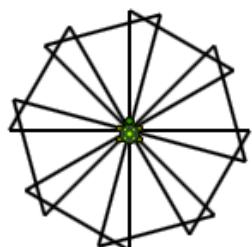
3. Crie um programa que a partir de uma sub-rotina que desenhe um triângulo equilátero com lado de tamanho "70" para a frente e giro a direita e gere a imagem.



```
Sub Triangulo
 For I = 1 To 3
 Turtle.Move(70)
 Turtle.Turn(120)
 EndFor
EndSub

For J = 1 To 4
 Triangulo()
 Turtle.Turn(90)
EndFor
```

4. Crie um programa que a partir de uma sub-rotina que desenhe um triângulo equilátero com lado de tamanho "70" para a frente e giro a direita e gere a imagem.

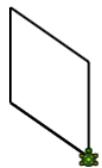


```
Sub Triangulo
 For I = 1 To 3
 Turtle.Move(70)
 Turtle.Turn(120)
 EndFor
EndSub

For J = 1 To 8
 Triangulo()
 Turtle.Turn(-45)
EndFor
```

5. Crie um programa que desenhe um losango com tamanho fornecido pelo usuário a partir de giro a direita. Assim que o desenho for concluído faça com que a tartaruga seja posicionada na parte inferior a direita como mostra a imagem seguinte.

```
TextWindow.Write("Entre um valor: ")
Tamanho = TextWindow.ReadNumber()
For I = 1 To 2
 Turtle.Move(Tamanho)
 Turtle.Turn(125)
 Turtle.Move(Tamanho)
 Turtle.Turn(55)
EndFor
Turtle.Turn(-55)
Turtle.Move(-Tamanho)
Turtle.Turn(55)
```



6. Crie programa que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se o comando **For**.

```
S = 0
For I = 1 To 100
 S = S + I
EndFor
TextWindow.WriteLine(S)
```

7. Crie programa que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se o comando **While**.

```
S = 0
I = 1
While (I <= 100)
 S = S + I
 I = I + 1
EndWhile
TextWindow.WriteLine(S)
```

8. Crie programa que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use o comando **For**.

```
S = 0
For I = 1 To 500
 R = Math.Remainder(i, 2)
 If (R = 0) Then
 S = S + I
 EndIf
EndFor
TextWindow.WriteLine(S)
```

9. Crie programa que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use o comando **While**.

```
I = 1
S = 0
While (I <= 500)
 R = Math.Remainder(i, 2)
 If (R = 0) Then
 S = S + I
 EndIf
 I = I + 1
EndWhile
TextWindow.WriteLine(S)
```

10. Crie programa que mostre todos os valores numéricos divisíveis por "4" *menores* que "20" iniciando a contagem em "1". Use o comando **For**.

```
For I = 1 To 19
 R = Math.Remainder(i, 4)
 If (R = 0) Then
 TextWindow.WriteLine(I)
 EndIf
EndFor
```

11. Crie programa que mostre todos os valores numéricos divisíveis por "4" *menores* que "20" iniciando a contagem em "1". Use o comando **While**.

```
I = 1
While (I <= 19)
 R = Math.Remainder(i, 4)
 If (R = 0) Then
 TextWindow.WriteLine(I)
 EndIf
 I = I + 1
EndWhile
```

12. Crie programa que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use o comando **For**.

```
For I = 0 To 15 Step 3
 TextWindow.WriteLine(I)
EndFor
```

13. Crie programa que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use o comando **While**.

```
I = 0
While (I <= 15)
 TextWindow.WriteLine(I)
 I = I + 3
EndWhile
```

**CAPÍTULO 5**

1. Desenvolver programa que leia oito elementos inteiros em uma matriz "A" de uma dimensão. Construir matriz "B" com os elementos da matriz "A" multiplicados por três. Apresentar a matriz "B".

```
TextWindow.Clear()
For I = 1 To 8
 TextWindow.Write("Entre o " + I + "o. número inteiro: ")
 A[I] = Math.Floor(TextWindow.ReadNumber())
EndFor
For J = 1 To 8
 B[J] = A[J] * 3
EndFor
TextWindow.WriteLine("")
For K = 1 To 8
 TextWindow.WriteLine("B[" + K + "] = " + B[K])
EndFor
TextWindow.WriteLine("")
TextWindow.Write("Tecle <Enter> para encerrar... ")
Enter = TextWindow.Read()
Program.End()
```

2. Desenvolver programa que leia duas matrizes "A" e "B" de uma dimensão com cinco elementos numéricos. Construir uma matriz "C", sendo cada elemento a subtração de um elemento correspondente da matriz "A" com um elemento correspondente da matriz "B". Ao final, apresentar os elementos da matriz "C".

```
TextWindow.Clear()
For I1 = 1 To 5
 TextWindow.Write("A[" + I1 + "] = ")
 A[I1] = Math.Floor(TextWindow.ReadNumber())
EndFor
TextWindow.WriteLine("")
For I2 = 1 To 5
 TextWindow.Write("B[" + I2 + "] = ")
 B[I2] = Math.Floor(TextWindow.ReadNumber())
EndFor
For J = 1 To 5
 C[J] = A[J] - B[J]
EndFor
TextWindow.WriteLine("")
For K = 1 To 5
 TextWindow.WriteLine("C[" + K + "] = " + C[K])
EndFor
TextWindow.WriteLine("")
TextWindow.Write("Tecle <Enter> para encerrar... ")
Enter = TextWindow.Read()
Program.End()
```

3. Desenvolver programa que leia cinco elementos numéricos em uma matriz "A" unidimensional e construa uma matriz "B" com os mesmos elementos da matriz "A" dispostos de forma invertida. Ou seja, o primeiro elemento da matriz "A" passa a ser o último da matriz "B", o segundo elemento da matriz "A" passa a ser o penúltimo da matriz "B", e assim por diante. Apresentar os elementos das matrizes "A" e "B".

```

TextWindow.Clear()
For I = 1 To 5
 TextWindow.WriteLine("A[" + I + "] = ")
 A[I] = Math.Floor(TextWindow.ReadNumber())
EndFor
For J = 1 To 5
 B[J] = A[6 - J]
EndFor
TextWindow.WriteLine("")
For K = 1 To 5
 TextWindow.WriteLine("C[" + K + "] = " + B[K])
EndFor
TextWindow.WriteLine("")
TextWindow.Write("Tecle <Enter> para encerrar... ")
Enter = TextWindow.Read()
Program.End()

```

4. Desenvolver programa que efetue a leitura de uma palavra ou frase qualquer e apresente o conteúdo da entrada de forma invertida. Dica: considere usar "Text.GetSubText()".

```

TextWindow.Clear()
TextWindow.Write("Entre um texto: ")
Texto = TextWindow.Read()
TextWindow.WriteLine("")
For I = Text.GetLength(Texto) To 1 Step -1
 TextWindow.WriteLine(Text.GetSubText(Texto, I, 1))
EndFor
TextWindow.WriteLine("")
TextWindow.WriteLine("")
TextWindow.Write("Tecle <Enter> para encerrar... ")
Enter = TextWindow.Read()
Program.End()

```

5. Desenvolver programa que efetue a leitura de uma frase qualquer e apresente a quantidade de caracteres da frase, excetuando-se os espaços em branco.

```

TextWindow.Clear()
TextWindow.Write("Entre uma frase: ")
Frase = TextWindow.Read()
TextWindow.WriteLine("")
Conta = 0
For I = 1 To Text.GetLength(Frase)
 If (Text.GetSubText(Frase, I, 1) <> " ") Then
 Conta = Conta + 1
 EndIf
EndFor
TextWindow.WriteLine("A frase possui " + Conta + " caracteres.")
TextWindow.WriteLine("")
TextWindow.Write("Tecle <Enter> para encerrar... ")
Enter = TextWindow.Read()
Program.End()

```

6. Desenvolver programa que efetue a leitura de um valor numérico inteiro maior que zero e escreva cada um de seus dígitos em extenso. Para "123", apresente "um, dois, três".

```
Extenso[1] = "um"
Extenso[2] = "dois"
Extenso[3] = "três"
Extenso[4] = "quatro"
Extenso[5] = "cinco"
Extenso[6] = "seis"
Extenso[7] = "sete"
Extenso[8] = "oito"
Extenso[9] = "nove"
TextWindow.Clear()
TextWindow.WriteLine("Entre um valor numérico inteiro: ")
N = Math.Floor(TextWindow.ReadNumber())
QuantDigitos = 1
While (N > 0)
 Dígito[QuantDigitos] = Math.Remainder(N, 10)
 QuantDigitos = QuantDigitos + 1
 N = Math.Floor(N / 10)
EndWhile
TextWindow.WriteLine("")
For J = 1 To QuantDigitos
 Número[J] = Dígito[QuantDigitos - J]
 TextWindow.WriteLine(Extenso[Número[J]])
 If (J < QuantDigitos - 1) Then
 TextWindow.WriteLine(",")
 ElseIf (J = QuantDigitos) Then
 TextWindow.WriteLine(".")
 EndIf
EndFor
TextWindow.WriteLine("")
TextWindow.WriteLine("Tecle <Enter> para encerrar... ")
Enter = TextWindow.Read()
Program.End()
```

7. Desenvolver programa que efetue a leitura de uma frase e conte a quantidade de palavras existentes na frase informada.

```
TextWindow.Clear()
TextWindow.WriteLine("Entre uma frase: ")
Frase = TextWindow.Read()
TextWindow.WriteLine("")
Conta = 1
For I = 1 To Text.GetLength(Frase)
 If (Text.GetSubText(Frase, I, 1) = " ") Then
 Conta = Conta + 1
 EndIf
EndFor
TextWindow.WriteLine("A frase possui " + Conta + " palavras.")
TextWindow.WriteLine("")
TextWindow.WriteLine("Tecle <Enter> para encerrar... ")
Enter = TextWindow.Read()
Program.End()
```

8. Desenvolver programa que efetue a leitura de uma palavra e apresente-a em sentido vertical (um caractere por linha).

```

TextWindow.Clear()
TextWindow.Write("Entre uma palavra: ")
Palavra = TextWindow.Read()
TextWindow.WriteLine("")
For I = 1 To Text.GetLength(Palavra)
 TextWindow.WriteLine(Text.GetSubText(Palavra, I, 1))
EndFor
TextWindow.WriteLine("")
TextWindow.Write("Tecle <Enter> para encerrar... ")
Enter = TextWindow.Read()
Program.End()

```

9. Desenvolver programa que efetue a leitura de uma frase e apresente frase e a quantidade de vogais existentes. Considere a frase em maiúsculo mesmo que não seja fornecida.

```

Vogal[1] = "A"
Vogal[2] = "E"
Vogal[3] = "I"
Vogal[4] = "O"
Vogal[5] = "U"
Vogal[6] = "Á"
Vogal[7] = "É"
Vogal[8] = "Í"
Vogal[9] = "Ó"
Vogal[10] = "Ú"
Vogal[11] = "À"
Vogal[12] = "Ã"
Vogal[13] = "Ê"
Vogal[14] = "Ô"
Vogal[15] = "Ã"
Vogal[16] = "Õ"
Vogal[17] = "Ü" ' Apesar de fora do acordo ortográfico
Vogal[18] = "Y"
TextWindow.Clear()
TextWindow.Write("Entre uma frase: ")
Frase = Text.ConvertToUpperCase(TextWindow.Read())
TextWindow.WriteLine("")
ContaVog = 0
For I = 1 To Text.GetLength(Frase)
 For J = 1 To 18
 If (Text.GetSubText(Frase, I, 1) = Vogal[J]) Then
 ContaVog = ContaVog + 1
 EndIf
 EndFor
EndFor
TextWindow.WriteLine("A frase: " + Frase + ",")
TextWindow.WriteLine("possui " + ContaVog + " vogais.")
TextWindow.WriteLine("")
TextWindow.Write("Tecle <Enter> para encerrar... ")
Enter = TextWindow.Read()
Program.End()

```

10. Desenvolver programa que efetue a leitura de uma frase e apresente a frase informada sem espaços em branco e em formato maiúsculo.

```
TextWindow.Clear()
TextWindow.Write("Entre uma frase: ")
FraseEntra = Text.ToUpperCase(TextWindow.Read())
TextWindow.WriteLine("")
FraseSaida = ""
For I = 1 To Text.GetLength(FraseEntra)
 If (Text.GetSubText(FraseEntra, I, 1) = " ") Then
 I = I + 1
 Else
 FraseSaida = FraseSaida + Text.GetSubText(FraseEntra, I, 1)
 EndIf
EndFor
TextWindow.WriteLine("A frase sem espaço é:")
TextWindow.WriteLine(FraseSaida)
TextWindow.WriteLine("")
TextWindow.Write("Tecle <Enter> para encerrar... ")
Enter = TextWindow.Read()
Program.End()
```

## **ANOTAÇÕES**

---

## Bibliografia

- ABELSON, H. **TI Logo**. New York: McGraw-Hill, 1984.
- DARTMOUTH. **Dartmouth at a Glance**. Site institucional, 2021. Disponível em <<https://home.dartmouth.edu/dartmouth-glance>>. Acesso 12 jul. 2021.
- HARVEY, B. **Computer science Logo style: Symbolic computing**. 2nd. Massachusetts: MIT Press, 1998, v. 1.
- KITAMURA, C. **Pascal Case e Camel Case: O que é e como usar? - [C#]**. Site pessoal. 2017. Disponível em: <<https://celsokitamura.com.br/pascal-case-e-camel-case/>>. Acesso 13 jul. 2021.
- MULLER, J. **The Great Logo Adventure: Discovering Logo on and Off the Computer**. Madison, AL, United States: Doone Pubns, 1998.
- SONYMASTER. **MSX BASIC - Programação da Linguagem BASIC no MSX**. xenForo: Adrenaline. 2020. Disponível em <<https://forum.adrenaline.com.br/threads/msx-basic-programacao-da-linguagem-basic-no-msx.671772/>>. Acesso 12 jul. 2021.

## Referências bibliográficas

- CONROD, P. **Beginning Microsoft Small Basic: Computer Programming Tutorial**. Maple Valley: Kidware Software. 2014.
- \_\_\_\_\_. **The Developer's Reference Guide to Microsoft Small Basic**. Maple Valley: Kidware Software. 2010.
- MARJI, M & PRICE, E. **Learn to Program with Small Basic: An Introduction to Programming with Games, Art, Science, and Math**. XXX: No Starch Press, 2016

## **ANOTAÇÕES**

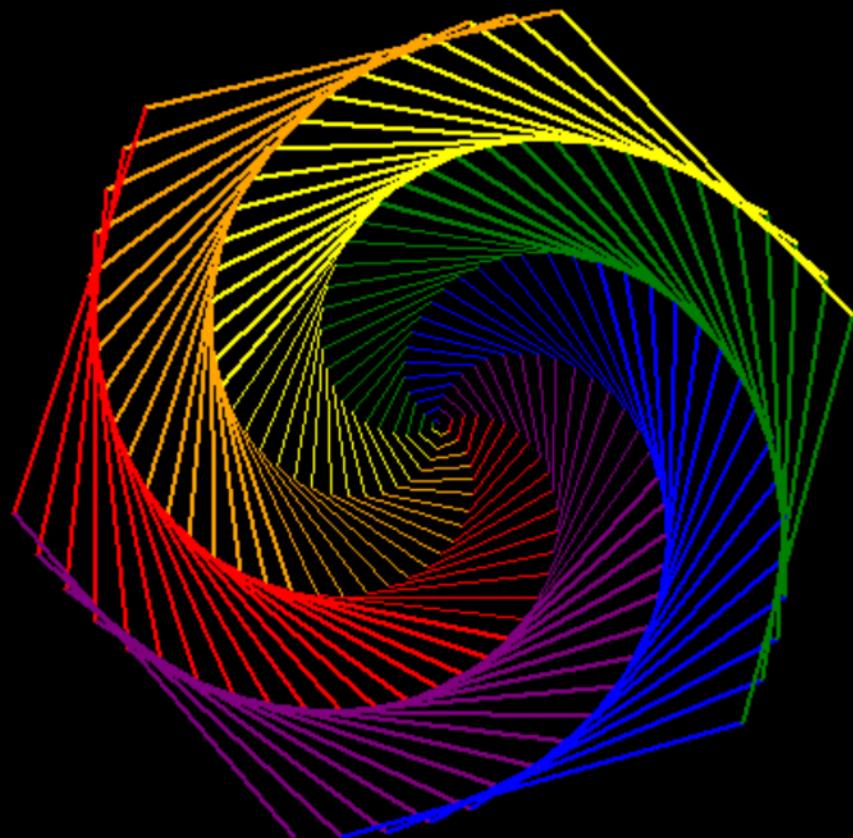






# PROGRAMAÇÃO DE COMPUTADORES PARA INICIANTES COM SMALL BASIC

Introdução com turtle graphics e um pouco mais



ESTE LIVRO APRESENTA A LINGUAGEM DE PROGRAMAÇÃO EDUCACIONAL "SMALL BASIC" DA MICROSOFT. O ESTUDO DESTE LIVRO É APRESENTADO DE FORMA DINÂMICA E PRÁTICA A PARTIR DE EXEMPLOS DE APLICAÇÃO FOCADOS NOS PRINCÍPIOS DE LÓGICA DE PROGRAMAÇÃO.

NESTE TRABALHO A LINGUAGEM É APRESENTADA A PARTIR DO PONTO DE VISTA DAS AÇÕES DA GEOMETRIA DA TARTARUGA SE ESTENDENDO A DIVERSAS PRÁTICAS DE PROGRAMAÇÃO.



ISBN 978-65-00-27582-7



9 786500 275827