Curso Básico de Logic Basic NG

Introdução

Apresentação

O Logic Basic é uma linguagem Básica de programação para facilitar o desenvolvimento de programas no ambiente Windows, destinada a desenvolver a lógica de programação em crianças, jovens e adultos iniciantes em computação, mas é ideal também para programadores profissionais que desejam desenvolver programas com maior rapidez e facilidade.

É uma linguagem simples, divertida e interessante, porém com poderosos recursos que poderão ser utilizados pelo programador à medida em que for se desenvolvendo dentro da linguagem.

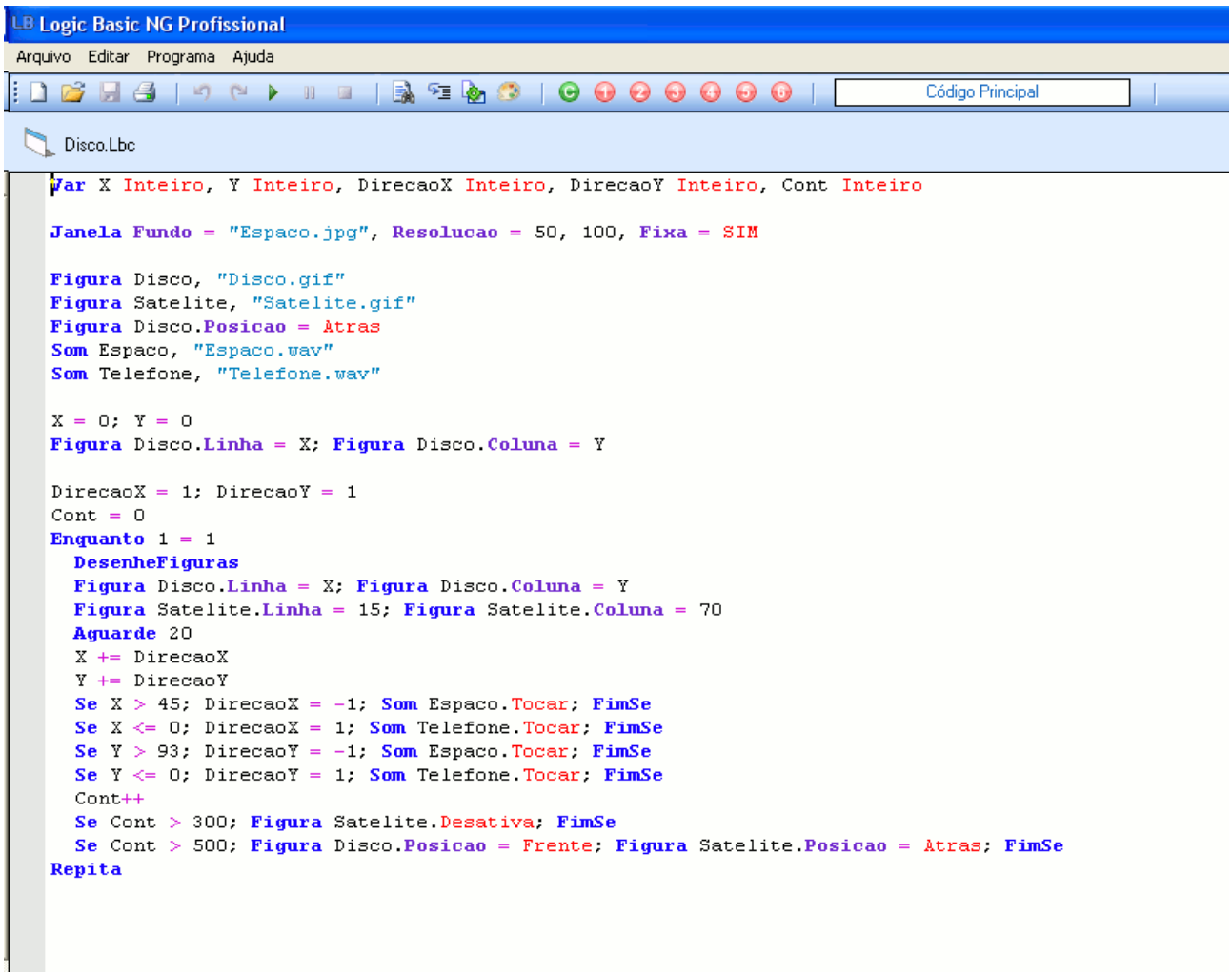
O Logic Basic é muito amigável, com um ambiente de programação claro e bastante estruturado, ou seja, ele incentiva o programador a fazer programas organizados, exercitando assim o seu raciocínio de acordo com as mais modernas linguagens de programação.

Com o Logic Basic, você poderá desenvolver programas comerciais, de animação gráfica, jogos, apresentações áudio-visuais, programas educativos, musicais, e tudo aquilo que sua mente puder imaginar.

Visão geral sobre o Logic Basic

Como funciona o Logic Basic?

O Logic Basic possui um ambiente de Código onde você escreverá o seu programa, que possui uma caixa de texto para o código principal e mais 6 caixas de textos para as extensões do código. Possui também o ambiente das janelas, onde os resultados da execução do programa serão apresentados.



Ao ser acionado o botão **Executar** (botão com uma seta verde para a direita), o Logic Basic executará o(s) comando(s) da primeira linha, depois o da segunda, depois o da terceira e assim por diante. Ou seja, ele executará o seu programa como se estivesse lendo um texto, onde você diz: "Faça isso", "Faça aquilo", ... na ordem da primeira linha para a última.

As Janelas são locais onde você, através dos comandos do Código, escreverá palavras, desenhará figuras, e também receberá informações do teclado, do mouse, do disco rígido do computador, etc. Nelas você poderá colocar fundos, que são imagens que ficarão fixas no plano de fundo da janela, escrever textos, colocar botões de comando, componentes como caixas de texto, caixas de listagem, caixas de checagem, etc., e também poderá desenhar figuras que poderão ser movidas ou modificadas através da programação. Há também sons que poderão ser tocados a qualquer momento durante a execução de seu programa.

Extensões de código

Há também as Extensões de Código, que são caixas de texto onde você poderá colocar alguns trechos do código para melhor organizar o seu programa. Para acessar o código principal, deve-se pressionar o botão **Código** (botão contendo a letra "C"), e para acessar as extensões, basta pressionar um dos 6 botões vermelhos do lado direito do botão de código.

Você pode imaginar essas extensões como uma continuação do código principal, e quando você executa o programa é como se elas estivessem concatenadas (emendadas) ao código principal. É aconselhável que nas extensões sejam colocados apenas funções e sub-rotinas que serão executadas a partir do código principal, como veremos mais adiante.

Ao se pressionar um dos botões de extensões, o Logic Basic apresentará na caixa de texto o código correspondente à respectiva extensão. Na parte superior esquerda da janela de código há uma caixa de texto denominada "Nome do código" onde poderá ser escrito um nome para identificar o trecho de código referente à extensão. Esse nome é opcional, não é necessário colocá-lo, mas é recomendável, para melhorar a clareza e o raciocínio do programa.

Salvando e abrindo programas

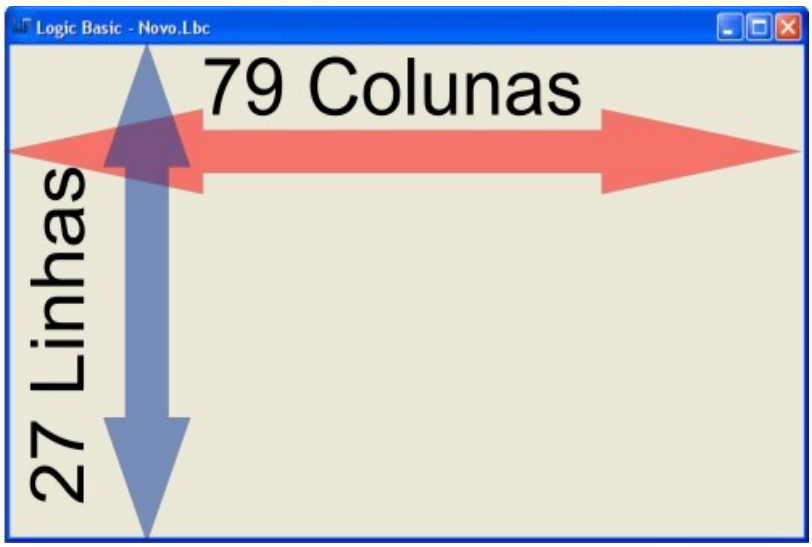
Os programas em Logic Basic poderão ser salvos em qualquer pasta selecionada pelo programador, e é aconselhável que se crie uma pasta para cada programa, onde serão gravados o código fonte, os arquivos de imagem e sons, etc. referentes ao programa. A extensão dos programas em Logic Basic é **.Lbc**, mas não é necessário você informá-la, pois o Logic Basic a colocará automaticamente.

Nome menu **Arquivo** temos quatro opções para criação, abertura e gravação de programas:

Novo programa: Quando esse botão for pressionado, O Logic Basic apagará o box de código e criará um novo programa com o nome de **Novo.Lbc**.
Abrir programa: O Logic Basic apresentará a você uma caixa de diálogo contendo os nomes dos programas gravados. Digite o nome do programa desejado ou dê um duplo-clique em cima do mesmo.
Salvar: Ao pressionar esse botão, o programa que estiver no ambiente de código será gravado no disco com o nome atual, sem perguntas.
Salvar como: O Logic Basic apresentará uma caixa de diálogo para que você informe o nome do programa a ser gravado. Caso o mesmo já tenha sido gravado anteriormente, basta dar um duplo-clique no programa da listagem.

Primeiro Programa em Logic Basic

A janela principal do Logic Basic é dividida em linhas e colunas. A quantidade padrão de linhas e colunas é 27 por 79 respectivamente, ou seja, o número da primeira linha é 0 (zero) e a última é 26, o número da primeira coluna é 0 e a última é 78.



O nosso primeiro programa será posicionar uma frase em uma determinada linha e em uma determinada coluna (a isso chamamos "Coordenada") e escrevê-la na janela.

Escrevendo na Janela do Logic Basic

A primeira coisa a fazer é escrever um comando no código para informar ao Logic Basic em qual linha e coluna ele deverá escrever ou desenhar alguma coisa. O nome desse comando é **Posicione**:

Posicione Linha, Coluna

Os valores **Linha e Coluna** são denominados **ARGUMENTOS** ou **PARÂMETROS**. Portanto, Linha é o primeiro argumento e Coluna é o segundo argumento do comando **Posicione**.

Os argumentos de um comando devem ser separados por vírgulas, e podemos escrever um comando por linha, ou vários na mesma linha, desde que sejam separados por **;** (ponto e vírgula). O próximo passo então é escrever uma frase na janela. Para isso utilizaremos o comando **Escreva**:

Escreva "Uma frase qualquer..."

Note que a frase deve estar entre aspas, para que o Logic Basic trate as palavras como apenas um argumento; sem as aspas, as palavras poderiam ser confundidas, no caso do exemplo acima, com uma expressão ou variável.

Executando o primeiro programa

Vamos então colocar em prática o que foi ensinado até agora. No ambiente de **Código** digite os seguintes comandos:

Posicione 11, 33
Escreva "Alô Mundo!"

Em seguida pressione o botão **Executar**, ou a tecla **F5**. O resultado deverá ser a frase **Alô Mundo!** escrita no centro da janela do Logic Basic.

Para voltar ao ambiente de código, pressione **F6**.

Exercício

Altere os valores dos argumentos Linha e Coluna do comando **Posicione**, e veja os resultados. Altere também a frase do comando **Escreva**.

Comandos e recursos básicos

Mensagem

O Logic Basic possui também um comando denominado **Mensagem**, idêntico ao comando **Escreva**, com a diferença que este não escreve diretamente na janela do Logic Basic, e sim em um pequeno box de mensagem centralizado no vídeo, e portanto, sua posição não obedece ao comando Posicione:

```
Mensagem "Uma frase qualquer..."
```

Janela

O comando **Janela** efetua vários tipos de configurações na janela principal do Logic Basic, como tamanho, cor de fundo, imagens de fundo, fonte de caracteres, etc.

Por exemplo, para definir a cor de fundo da janela como sendo branca, basta informar como argumento a palavra **Fundo**, em seguida o sinal de igual e a cor desejada:

```
Janela Fundo = Branco
```

Para definir uma imagem de fundo para a janela, ao invés da cor deve-se informar o caminho e nome de um arquivo de imagem, por exemplo:

```
Janela Fundo = "C:\Figuras\Paisagem.jpg"
```

Há varias outras opções de configuração, que podem ser informadas na mesma linha em sequência, desde que separadas por vírgula, por exemplo:

```
Janela Resolucao = 50, 150, Tamanho = 30, 100, Posicao = 0, 0
```

No exemplo acima o comando **Janela** definiu a resolução da janela com 50 linhas por 150 colunas, o tamanho da janela do Logic Basic com 30 linhas e 100 colunas e a posição superior esquerda da janela na área de trabalho do Windows na linha 0, coluna 0. Um detalhe importante é que o tamanho e a posição dependem da resolução da janela, que é calculada de acordo com o tamanho atual da janela, portanto essa deve ser definida em primeiro lugar.

Fonte

O comando **Fonte** define o fonte de caracteres da janela como nome, tamanho, cor, etc. Após definido o fonte, os textos escritos pelo comando **Escreva** terão todos os atributos definidos.

Como no Logic basic poderão existir novas janelas além da principal, que poderão ser criadas pelo comando **NovaJanela** (que explicaremos mais adiante) onde cada uma terá um nome, foi convencionado que o nome da janela principal será **Pai**. Então para definir um determinado atribudo de fonte na janela principal, deve-se informar o comando **Fonte**, o nome da janela a alterar o fonte, em seguida ponto, o nome do atributo e seu valor, por exemplo:

```
Fonte Pai.Nome = "Arial"
Fonte Pai.Cor = Azul
Fonte Pai.Tamanho = 20
Posicione 5,10; Escreva "Logic Basic"
```

LimpaJanela

O comando **LimpaJanela** apaga todos os textos da janela do Logic Basic, exceto a imagem de fundo e componentes colocados sobre a mesma. Esse comando não possui nenhum argumento:

```
LimpaJanela
```

Comentários

O Logic Basic permite você colocar comentários no código do programa, de modo que eles não tenham nenhum efeito no programa; é como se não existissem no código.

Para isso basta colocar o sinal ' (apóstrofo), no início da linha ou após alguma linha de comando, que qualquer texto digitado após ele será ignorado pelo Logic Basic:

```
Escreva "Alô mundo!"   'Escreve a frase Alô mundo! na janela do LB
```

No exemplo acima, o texto escrito após o comando **Escreva "Alô mundo"** será ignorado.

Colocando vários comandos em uma mesma linha

Quando for colocado **;** (ponto-e-vírgula) após um comando, poderá ser escrito outro comando após o mesmo, na mesma linha. Isso melhora a clareza do programa, e também diminui o número de linhas. Na verdade, para o Logic Basic, cada comando separado por ponto-e-vírgula é considerado como uma nova linha. Dessa maneira, você poderá colocar quantos comandos desejar em uma mesma linha, por exemplo:

```
Posicione 11, 33; Escreva "Alô Mundo!"
```

O exemplo acima escreve o nosso primeiro programa com dois comandos em uma mesma linha.

Comandos de finalização de programas

Os comandos **FimPrograma** e **FimJanela** encerram o programa na linha em que forem colocados. O primeiro apenas finaliza o programa, mas mantém a Janela principal do Logic Basic ativa. O segundo, finaliza o programa e fecha todas as Janelas do Logic Basic. Note que você deverá utilizar um ou outro, nunca os dois ao mesmo tempo.

Declaração de variáveis

No Logic Basic você poderá criar variáveis, que são memórias para guardar textos, números e caracteres. Existem três tipos de variáveis no Logic Basic: **String**, **Inteiro** e **Decimal**.

O tipo **String** deve ser utilizado para declarar variáveis de textos ou sequência de caracteres; o tipo **Inteiro** deve ser utilizado para declarar variáveis do tipo numérico inteiro que variem de -2.147.483.648 a 2.147.483.647; o tipo **Decimal** deve ser utilizado para declarar variáveis do tipo numérico decimal que podem variar de -1,79769313486232E308 a -4,94065645841247E-324 para valores negativos e 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos (note que a letra E significa "elevado a" ou seja exponenciação). Simplificando, variáveis do tipo **Decimal** podem ser utilizadas para guardar tanto valores inteiros quanto fracionários, e são utilizadas principalmente para trabalhar com valores de moeda.

Para declarar variáveis deve-se utilizar o comando **Variavel** ou simplesmente **Var** e em seguida o nome da variável seguido de seu tipo. Caso não seja informado o tipo da variável, o Logic Basic assumirá como sendo do tipo **String**. Pode-se declarar várias variáveis em uma mesma linha, por exemplo:

```
Variavel NomeCliente String, X Inteiro, Valor Decimal
```

ou

```
Var NomeCliente String, X Inteiro, Valor Decimal
```

O nome de uma variável não poderá conter espaços em seu corpo, por isso no exemplo acima, a variável "NomeCliente" não foi escrita "Nome Cliente". Nome de variáveis não podem ter o mesmo nome de comandos, palavras e variáveis reservadas do Logic Basic, e é recomendável que o nome das mesmas não contenham nome de comandos em seu corpo.

Após as variáveis terem sido criadas, poderemos colocar textos e números nas mesmas utilizando o operador **=** (igual), por exemplo:

NomeCliente = "João da Silva"
Valor = 1234.50

Note que textos devem ser colocados entre aspas, veremos o porquê a seguir...

Ao atribuirmos textos a uma variável, eles devem ser escritos entre aspas. Na maioria das linguagens de programação, uma sequência de letras ou caracteres são denominados **Strings**. A partir de agora, trataremos textos como Strings para facilitar o diálogo, e também para que o programador comece a se familiarizar com esse termo. Portanto, quando falarmos de Strings estaremos falando de textos, letras, e também números na forma de texto.

Um número, quando estiver entre aspas, será tratado como uma String, e quando estiver sem aspas, será tratado como um valor numérico; isso quer dizer que poderemos efetuar cálculos com ele.

Quando for atribuído um número explicitamente a uma variável, se o número for fracionário as casas decimais devem ser separadas por ponto e jamais por vírgula, isso é uma regra.

Uma variável pode ser atribuída a outra variável, por exemplo:

X = 7
Valor = X

No exemplo acima, a variável **X** recebeu o número **7**. Em seguida a variável **Valor** recebeu o valor da variável **X**, que é **7**. Portanto, o conteúdo da variável **Valor** é igual a **7**.

Para o Logic Basic, um número, ou uma variável contendo um número, são vistos como números. Ou seja, **7** é um número, e **X** também é um número. Só que você não pode ver o número que está em **X**, mas o Logic Basic sim, porque está em sua memória. O mesmo ocorre com strings, quando um texto for uma variável, o Logic Basic irá trabalhar com o seu conteúdo e não com o seu nome. Vamos exemplificar com o seguinte programa:

```
Escreva Nome
```

Ao pressionar o botão **Executar Programa**, o resultado será a seguinte palavra: **Nome.** O Logic Basic escreveu **Nome** porque essa palavra não é uma variável. Agora vamos modificar o programa para o seguinte código:

```
Variavel Nome String
Nome = "João da Silva"
Escreva Nome
```

Ao pressionar o botão **Executar Programa**, o resultado agora será o seguinte: **João da Silva**. O Logic Basic escreveu **João da Silva**, porque a palavra **Nome** foi declarada como uma variável ao Logic Basic, e ele sabendo disso, escreveu o seu contéudo.

Variáveis Globais e Locais

No Logic Basic NG foi introduzido o conceito de variáveis Globais e Locais. As variáveis Globais podem ser lidas em qualquer ponto do programa, elas manterão o seu valor dentro do código principal, extensões, sub-rotinas e funções. As variáveis Locais devem ser declaradas somente dentro de funções, e elas manterão o seu valor somente dentro da função onde ela for declarada e após a execução da função, as mesmas são destruídas pelo Logic Basic. Isso é importante pois economiza memória, além de evitar confusões em seu programa. As variáveis locais podem ter o mesmo nome em funções diferentes sem causar conflitos.

Regra para declaração de variáveis Locais

Para declarar uma variável local deve-se cumprir uma regra: elas só podem ser declaradas nas primeiras linhas de uma função, por exemplo:

```
Funcao Soma() Inteiro
Variavel X Inteiro, T Inteiro
Variavel Ret Inteiro

X = 10; Y = 20
Ret = X + T

Retorne Ret
FimFuncao
```

No exemplo acima, as variáveis **Ret**, **X** e **T** são Locais. Se uma variável for declarada no meio do código de uma função, ela será considerada como Global, por exemplo:

```
Funcao Soma() Inteiro
Variavel X Inteiro, T Inteiro

X = 10; Y = 20

Variavel Ret Inteiro

Ret = X + T

Retorne Ret
FimFuncao
```

No exemplo acima as variáveis **X** e **T** são locais, enquanto a variável **Ret** é global.

Operadores

A função das variáveis não é apenas guardar textos e números, elas podem ser

modificadas a qualquer momento pelo programador, através de operadores, comandos e novas atribuições. No Logic Basic temos quatro tipos de operadores: Operadores aritméticos, Operadores de incremento e decremento, de comparação e lógicos.

Operadores Aritméticos:

+	Soma
-	Diferença
*	Multiplicação
/	Divisão
%	Resto da divisão
^	Exponenciação

Operadores de incremento e decremento:

++	Soma 1 a ele mesmo
–	Diminui 1 dele mesmo
+=	Soma a ele um número
-=	Diminui dele um número

Operadores de comparação:

=	Igual
<	Menor que
>	Maior que
<=	Menor ou igual
>=	Maior ou igual
<>	Diferente

Operadores lógicos:

E	Conjunção lógica
OU	Disjunção lógica

Operador de união:

&	Une strings ou variáveis

O operador = é de comparação apenas quando estiver dentro de um comando de comparação, em outras situações ele é um operador de atribuição.
A seguir, alguns exemplos de atribuições e cálculos com variáveis e números:

```
Variavel X Inteiro, Y Inteiro, Total Inteiro

X = 3
Escreva "O valor de X é igual a ", X
Y = X + 7
Escreva "O valor de Y é igual a ", Y
Total = X * Y
Escreva "O Valor de X multiplicado por Y é igual a ", Total
X++
```

```
Escreva "O valor de X incrementado é ", X
X += 5
Escreva "O valor de X incrementado com 5 é ", X
```

Exercício

Digite o programa de exemplos de atribuições e cálculos com variáveis no ambiente de código do Logic Basic, e pressione o botão **Executar Programa**. Depois volte ao código, altere alguns valores, execute o programa novamente e veja o resultado.

Comandos Condicionais e de Controle de fluxo

Comando Condicional Se

Esse é um comando que existe em todas as linguagens de programação, e sua função é condicionar a execução de um trecho de código de acordo com o resultado de uma expressão: se a expressão for verdadeira, ele executa o trecho de código, se for falsa não executa, ou executa outro trecho de código. O nome desse comando é **Se**, e ele possui mais dois comandos auxiliares: **CasoContrario** e **FimSe**. O primeiro executa outro trecho de código caso o resultado da expressão seja falso, e o segundo deve ser colocado sempre no fim de uma cláusula **Se**. Por exemplo:

```
Variavel X Inteiro

X = 1
Se X = 1
   Escreva "X é igual a 1!"
CasoContrario
   Escreva "X não é igual a 1!"
FimSe
```

O comando **CasoContrario** não é obrigatório, mas o comando **FimSe** deverá ser colocado obrigatoriamente para finalizar seu respectivo comando **Se**.

Você pode colocar um comando **Se** dentro de outro comando **Se**, ou até mesmo em cascata, por exemplo:

```
Variavel X Inteiro, Y Inteiro

X = 1
Y = 2

Se X = 1
  Escreva "X é igual a 1"
  Se Y = 2
    Escreva "Y é igual a 2"
```

```
  CasoContrario
    Escreva "Y não é igual a 2"
  FimSe
CasoContrario
  Escreva "X não é igual a 1"
FimSe
```

Altere os valores de X e Y no exemplo acima e veja os resultados.

Comando direcional VaPara

Como já vimos anteriormente, o Logic Basic executa o programa linha por linha, da primeira até a última. Mas você pode direcionar a execução do programa para outra linha, para outro ponto do programa. Para isso o Logic Basic possui um comando denominado **VaPara** que faz com que a execução vá para uma determinada linha, bastando que você crie um rótulo (um nome qualquer seguido de dois pontos) em uma determinada linha do programa e direcione a execução para a linha desse rótulo com o comando VaPara, por exemplo:

```
Variavel X Inteiro

X = 2

Se X = 1
  VaPara POSICAO1
CasoContrario
  VaPara POSICAO2
FimSe

POSICAO1:
Escreva "Posicao 1"

POSICAO2:
Escreva "Posicao 2"
```

Note que no programa acima digitamos o nome de dois rótulos, o primeiro de nome POSICAO1: e o segundo de nome POSICAO2:, então dependendo do valor da variável **X** direcionamos a execução para o primeiro rótulo ou para o segundo rótulo, ou seja, se o valor de X for igual a 1, direcionamos a execução para a linha onde está o rótulo POSICAO1:, se o valor de X for igual a 2, direcionamos a execução para a linha onde está o rótulo POSICAO2:, execute o programa com o valor de X = 1, depois com o valor de X = 2 e veja os resultados.

Comando de controle de fluxo Enquanto

O Logic Basic possui um comando de controle de fluxo denominado **Enquanto** que permite executar repetidamente um trecho de código enquanto o resultado de uma expressão for verdadeiro. Cada comando **Enquanto** deve possuir seu respectivo comando **Repita**, que deve ser colocado no final do

trecho do código a ser executado. Quando o resultado da expressão do comando **Enquanto** for falso ele direcionará a execução para a próxima linha após o comando **Repita**, por exemplo:

```
Variavel X Inteiro

X = 0
Enquanto X < 10
  Escreva X
  X++
Repita
```

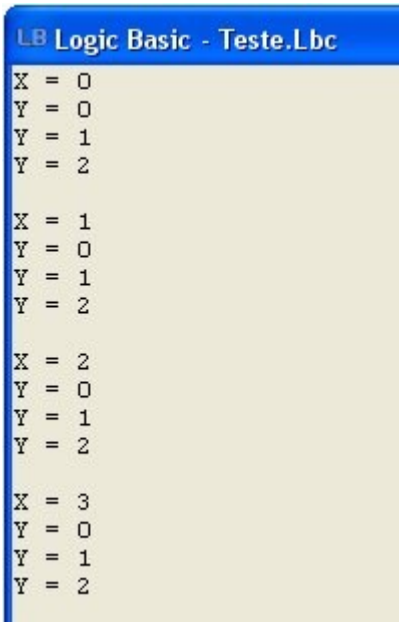
O código do exemplo acima irá escrever o valor de X enquanto ele for menor do que 10, ou seja, ele escreverá todos os valores de 0 a 9.

Você também poderá colocar um comando **Enquanto** dentro de outro comando **Enquanto**, ou até mesmo em cascata, por exemplo:

```
Variavel X Inteiro, Y Inteiro

X = 0
Enquanto X <= 3
  Escreva "X = ", X
  X++
  Y = 0
  Enquanto Y <= 2
    Escreva "Y = ", Y
    Y++
  Repita
  Escreva ""
Repita
```

No exemplo acima, para cada valor de X (0 a 3) será escrito três valores de Y (0 a 2):



Operadores lógicos

Os operadores lógicos **E** e **OU** podem ser utilizados na linha dos comando **SE** ou **ENQUANTO** para efetuar operações lógicas entre expressões. O operador **E** efetua uma operação lógica entre expressões de modo que ele retorna um resultado verdadeiro se todas as expressões forem verdadeiras ao mesmo tempo; se pelo menos uma das expressões for falsa, ele retorna falso. O operador **OU** efetua uma operação lógica entre expressões de modo que se pelo menos uma das expressões for verdadeira, ele retorna um resultado verdadeiro, e retorna falso somente se todas as expressões forem falsas. Você poderá efetuar operações lógicas com várias expressões em uma mesma linha e também colocar os dois operadores na mesma linha.

A seguir, exemplos dos operadores lógicos:

```
Variavel X Inteiro, Y Inteiro

X = 1
Y = 1

Se X = 1 E Y = 1          'Retorna Verdadeiro
  Escreva "Verdadeiro"
CasoContrario
  Escreva "Falso"
FimSe

Se X = 1 E Y = 2          'Retorna Falso
  Escreva "Verdadeiro"
CasoContrario
  Escreva "Falso"
FimSe

Se X = 1 OU Y = 2          'Retorna Verdadeiro
  Escreva "Verdadeiro"
CasoContrario
  Escreva "Falso"
FimSe

Se X = 2 OU Y = 2          'Retorna Falso
  Escreva "Verdadeiro"
CasoContrario
  Escreva "Falso"
FimSe
```

Matrizes

Matrizes são um conjunto de variáveis que possuem um único nome, e cada elemento de uma matriz possui um índice numérico de identificação. Uma matriz é declarada da seguinte forma: o nome da matriz juntamente com um valor numérico entre parênteses e o seu tipo:

```
Variavel Nome(20) String, Idade(20) Inteiro
```

A linha de comando acima cria duas matrizes: **Nome** com 20 variáveis e **Idade** com 20 variáveis. O índice de uma matriz começa com o valor 0 (zero); portanto no exemplo acima, o índice varia de 0 a 19. A seguir, um exemplo de como colocar e obter textos e números em matrizes:

```
Variavel Nome(20) String, Idade(20) Inteiro

Nome(0) = "Priscila"
Idade(0) = 10
Nome(1) = "Jorge"
Idade(1) = 20
Nome(2) = "Rossana"
Idade(2) = 16
```

Nesse exemplo, colocamos valores nas matrizes somente até o índice 2, mas poderemos colocar valores até o índice 19. As matrizes que não foram colocados valores ficarão com o conteúdo vazio.

Note que criamos duas matrizes com a mesma quantidade de índices, e associamos os índices da matriz **Nome** com os índices da matriz **Idade**, por exemplo, o Nome(0) corresponde à Idade(0), o Nome(1) corresponde à Idade(1) e assim por diante. Isso é importante porque podemos criar registros para fazer, por exemplo, um cadastro de clientes, associando cada cliente a um índice.

Para obter um valor de uma matriz, basta informar o nome da matriz e o índice entre parênteses, por exemplo:

```
Escreva Nome(2)
```

Se colocarmos o comando acima no final do nosso exemplo anterior, o resultado será **Rossana**.

No índice da matriz podemos colocar também variáveis numéricas, que como o próprio nome diz, por serem variáveis tornam as matrizes um poderoso recurso de programação. A seguir, um exemplo de como listar os nomes na janela, utilizando uma variável como índice:

```
Variavel Nome(20) String, Idade(20) Inteiro, X Inteiro

Nome(0) = "Priscila"
Idade(0) = 10
Nome(1) = "Jorge"
Idade(1) = 20
Nome(2) = "Rossana"
Idade(2) = 16

X = 0
Enquanto X <= 2
```

```
  Escreva Nome(X), " - ", Idade(X)
  X++
Repita
```

Nesse exemplo serão listados na janela do Logic Basic os três primeiros nomes com suas respectivas idades.

Uma regra importante é que no índice da matriz só poderemos colocar números ou variáveis, mas não expressões complexas. Caso necessite colocar uma expressão no índice da matriz, atribua a mesma a uma variável e depois informe-a como índice.

Exercício

Crie uma matriz do tipo Decimal com 10 elementos e faça uma rotina para atribuir as raízes quadradas de todos os números de 1 a 10 para os elementos da matriz, em seguida faça outra rotina para escrever os valores de todos os elementos da matriz na janela do Logic Basic.

Entrada de textos

Além de escrever textos nas janelas, você também pode capturar textos digitados para utilizá-los em seu programa. Para isso você deverá criar uma caixa de texto utilizando o componente denominado **Texto,** e posicioná-la em algum ponto na janela, por exemplo:

```
Posicione 1, 3; Escreva "Digite seu nome:"
Texto SeuNome, 1, 21
```

Ao criar a caixa de texto, componente esse que atribuímos o nome de "SeuNome", e que foi posicionada na linha 1, coluna 21 da janela, o Logic Basic criará automaticamente uma variável cujo nome será **SeuNome.Txt**, ou seja, o nome do componente seguido de **.Txt**, que conterá o texto digitado na caixa de texto. Sendo assim você poderá trabalhar com essa variável em seu programa do mesmo modo que uma variável normal do tipo String.

Após a criação da caixa de texto é necessário colocar algum componente que faça com que o programa aguarde na mesma linha até que o texto seja totalmente digitado na mesma, e a melhor solução para isso é criar um botão e colocar no nosso código o comando **Aguarde CliqueBotao** para que a execução do programa aguarde na mesma linha até que o botão seja pressionado, por exemplo:

```
Posicione 1, 3; Escreva "Digite seu nome:"
Texto SeuNome, 1, 21
Posicione 3,3; Escreva "Digite sua idade:"
Texto SuaIdade, 3, 21, 3
Botao Ok, 5, 3
Aguarde CliqueBotao
Mensagem "Seu nome é ", SeuNome.Txt, " e sua idade é ",
```

```
SuaIdade.Txt, " anos"
```

Note que utilizamos o componente **Botao** para criar um botão na janela do Logic Basic, o qual foi nomeado de **Ok**, e posicionado na linha 5, coluna 3.

Mouse

O Logic Basic permite que se posicione textos, figuras ou componentes na posição corrente do cursor do mouse. Para informar a posição do cursor do mouse, foram criadas duas variáveis reservadas do Logic Basic: **LinhaMouse** e **ColunaMouse**. Você também pode verificar se houve um clique ou duplo-clique no mouse sobre a janela do Logic Basic, bastando para isso checar o conteúdo da variável reservada **CliqueMouse**, se o seu conteúdo for igual a **"CLIQUE"** quer dizer que houve um clique, se o seu conteúdo for igual a **"DUPLOCLIQUE"** quer dizer que houve um duplo-clique. A seguir um exemplo de como posicionar textos na posição do cursor do mouse:

```
Enquanto 1 = 1
  Posicione LinhaMouse-1, ColunaMouse-1
  Se CliqueMouse = "CLIQUE"
    Escreva "+"
  CasoContrario
    Escreva "O"
  FimSe
  Aguarde 10
Repita
```

No exemplo acima é criado um loop de repetição infinito, e dentro desse loop posicionamos o texto na linha e coluna do cursor do mouse (menos 1 para posicionar um pouco mais à esquerda e acima da seta do mouse). Na próxima linha fazemos um teste com o comando **Se** para verificar se houve um clique do mouse, que em caso positivo, é escrito na janela o caractere "+" (mais), caso contrário é escrito na janela a letra "O". Na próxima linha foi colocado o comando **Aguarde 10** para que o programa aguarde 10 milissegundos antes de repetir o laço (loop).

Você também poderá verificar se o botão direito ou esquerdo do mouse está pressionado ou não. Para isso basta utilizar o comando **TestaMouse**, que informa 0 se o botão não estiver pressionado e 1 se ele estiver pressionado. Ele retorna uma string com duas posições, a primeira para o botão esquerdo do mouse e a segunda para o botão direito. Por exemplo:

```
Var SitMouse String

Enquanto 1 = 1
  LimpaJanela
  SitMouse = TestaMouse()
  Posicione 0,0; Escreva SitMouse
  Aguarde 10
Repita
```

SitMouse será igual a "00" se nenhum botão estiver pressionado, "10" se o botão esquerdo estiver pressionado e "01" se o botão direito estiver pressionado.

Exercício

No programa de posicionar textos no cursor do mouse, altere o programa para que ele escreva o caractere "+" somente quando houver um duplo-clique.

Aguarde

O Logic Basic possui um comando denominado **Aguarde** que paralisa a execução do programa na linha onde o mesmo for colocado durante um tempo pré-determinado, ou até que ocorra um determinado evento:

```
Escreva "Por favor, Aguarde 5 segundos..."
Aguarde 5000
Escreva "Ok"
```

O programa acima escreve o texto **"Por favor, Aguarde 5 segundos..."**, em seguida aguarda 5 segundos, e após transcorrido esse tempo, escreve **Ok** na janela do Logic Basic. Note que o valor passado como argumento foi 5000 porque o tempo deve ser informado em milissegundos.

Se o argumento do comando Aguarde for **CliqueBotao** ou **CliqueObjeto**, o Logic Basic aguardará até que um botão ou a opção de um menu sejam "clicados" ou pressionados pelo mouse, e colocará na variável reservada **Retorno** o nome do botão ou opção selecionada:

```
Aguarde CliqueBotao
Escreva Retorno
```

Obs.: as palavras acima que estão na cor vermelha são identificadores ou palavras reservadas do Logic Basic.

Há também o argumento **CliqueBox** que aguarda um clique no box de listagem do Logic Basic, cujo funcionamento será explicado mais adiante.

Data e Hora

O Logic Basic permite você obter a data e a hora do relógio do computador, para isso, você deve utilizar dois comandos: **Data** e **Hora**:

```
Variavel Hoje String, Agora String

Hoje = Data()
Agora = Hora()
Escreva "Hoje é", Hoje, "Agora são", Agora
```

O programa do exemplo acima cria duas variáveis: **Hoje** e **Agora**, e em seguida atribui o retorno do comando **Data** e **Hora** para as respectivas variáveis, e escreve o resultado na janela do Logic Basic.

Caso você necessite saber qual o dia da semana correspondente a uma determinada data, basta utilizar o comando **DiaSemana** informando como argumento a data desejada:

```
Variavel Hoje String

Hoje = Data()
Escreva DiaSemana(Hoje)
```

Componentes

Componentes são objetos que podem ser criados sobre as janelas do Logic Basic, como caixas de texto, botões de comando, botões de opção, caixas de checagem, calendários, barras de rolagem, imagens, etc. Há também componentes que não podem ser visualizados na janela do Logic Basic, somente o seu resultado, ou seja, são componentes "virtuais", como por exemplo os componentes **Rel**, **Imp** e **SQL**.

Cada tipo de componente possui um comando para sua criação e manipulação. Os componentes possuem também propriedades e métodos. **Propriedades** são características ou aspectos que podem ser atribuídos a um determinado componente ou mesmo retornados por ele. **Métodos** são comandos que são executados pelo componente para efetuar determinadas tarefas em relação a ele mesmo. Para você entender melhor, vamos dar alguns exemplos de propriedades e métodos:

Propriedades:

Visivel = Define o componente como visível.
Invisivel = Define o componente como invisível.
Txt = Define ou retorna o texto de um componente.
IndiceTab = Define o número (índice) de prioridade de tabulação do componente.

Métodos:

Seleciona = Seleciona o texto de uma caixa de texto.
Adiciona = Adiciona um item no componente.
MoveFoco = Move o foco para o componente em referência.
Remove = Remove o componente da janela.

Todo componente terá um nome para caracterizá-lo, por isso dois componentes, mesmo sendo de tipos diferentes, não poderão ter o mesmo nome. Uma boa técnica de programação é você colocar uma abreviação do tipo do componente antes de seu nome, por exemplo:

```
Texto TxtTeste, 3, 3
```

Botao BtnTeste, 7, 3

No exemplo acima, o comando **Texto** irá criar uma caixa de texto de nome **TxtTeste** na linha 3, coluna 3, e o comando **Botao** irá criar um botão de nome **BtnTeste** na linha 7, coluna 7.

Para atribuir um valor a uma propriedade de um determinado componente basta digitar o comando referente ao mesmo, o nome do componente seguido de ponto e o nome da propriedade, em seguida o sinal de igual e seu valor, por exemplo:

Texto TxtTeste.IndiceTab = 10

No exemplo acima, o valor 10 é atribuído à propriedade **IndiceTab** do componente **TxtTeste.**

Para executar um determinado método, a regra é praticamente a mesma, com a diferença que um comando será executado em relação ao componente, por exemplo:

Texto TxtTeste.Seleciona

No exemplo acima, o texto contido dentro da caixa de texto do componente **TxtTeste** será selecionado pelo Windows, permitindo você copiar e colar o texto marcado em outro local.

Em alguns componentes que possuem textos ou valores, como caixas de texto, barras de rolagem, botões de opção, calendários, etc. o Logic Basic cria uma variável para cada componente para armazenar os seus valores, e o nome dessa variável será o nome do componente seguido do nome da propriedade, por exemplo, para se obter o texto digitado dentro da caixa de texto de nosso exemplo basta ler o valor da propriedade **Txt** do seguinte modo:

Nome = TxtTeste.Txt

Você poderá trabalhar com essas propriedades em seu programa do mesmo modo que variáveis normais do tipo retornado pelo componente, que poderá ser **String**, **Inteiro** ou **Decimal.**

Figuras

Desenhando figuras na janela do Logic Basic

O comando **Figura** permite desenhar figuras na janela do Logic Basic, movimentá-las e sobrepô-las na ordem desejada.

Obs.: para que os exemplos desse capítulo funcionem corretamente, copie as figuras **Disco.gif** e **Satelite.gif** que estão não pasta **Disco** dos programas-exemplo do Logic Basic para dentro da pasta onde você irá salvar os seus programas.

Uma figura também é um componente que possui propriedades e métodos, e para criarmos um componente de figura deveremos informar o comando **Figura** e em seguida o seu nome, vírgula e o caminho do arquivo de imagem a carregar, por exemplo:

```
Figura FigDisco, "Disco.gif"
```

Para definirmos em que posição na janela (coordenadas) a figura deverá ser desenhada deveremos informar os valores das propriedades linha e coluna, conforme o exemplo a seguir:

```
Figura FigDisco.Linha = 5
Figura FigDisco.Coluna = 10
```

Desse modo, a figura será desenhada na linha 5, coluna 10, da janela do Logic Basic.

Note que até agora criamos a figura e definimos a sua posição na janela, mas somente esses comandos não farão com que a figura seja desenhada na janela, para isso é necessários executarmos o comando **DesenheFiguras**, que irá desenhar todas as figuras criadas de uma só vez, por exemplo:

```
Figura FigSat, "Satelite.gif"
Figura FigSat.Linha = 5
Figura FigSat.Coluna = 10

Figura FigDisco, "Disco.gif"
Figura FigDisco.Linha = 7
Figura FigDisco.Coluna = 17

DesenheFiguras
```

O código acima irá desenhar as imagens contidas nos arquivos **Disco.gif** e **Satelite.gif** de uma só vez nas suas respectivas coordenadas na janela do Logic Basic.

Quando as figuras são desenhadas na janela, caso ocorra uma sobreposição entre elas, as últimas figuras desenhadas sobreporão as primeiras, por isso no exemplo anterior a figura **FigDisco** será desenhada na frente da figura **FigSat**, mas você pode mudar a ordem de sobreposição das figuras a qualquer momento utilizando a propriedade **Posicao**, atribuindo-lhe as palavras-chaves **Atras** ou **Frente**, por exemplo:

```
Figura FigSat, "Satelite.gif"
Figura FigSat.Linha = 5
Figura FigSat.Coluna = 10

Figura FigDisco, "Disco.gif"
Figura FigDisco.Linha = 7
Figura FigDisco.Coluna = 17

Figura FigSat.Posicao = Frente
```

```
Figura FigDisco.Posicao = Atras

DesenheFiguras
```

No exemplo acima, a figura **FigSat** será desenhada na frente da figura **FigDisco**.

Movimentando figuras na janela

A cada chamada do comando **DesenheFiguras**, primeiramente o Logic Basic apaga todas as figuras e textos que estão sobre a janela, para depois desenhar as figuras. Sendo assim podemos facilmente fazer uma animação gráfica com as figuras, movimentando-as para novas posições.

Para fazer uma animação gráfica com uma figura, deveremos desenhá-la em uma determinada posição na janela, aguardar um determinado tempo e em seguida desenhá-la em outra posição.

O exemplo a seguir movimentará a figura **Disco.gif** do lado esquerdo para o lado direito da janela, para isso criaremos a variável **XCol** para incrementar a posição da coluna da figura a cada 50 milissegundos:

```
Variavel XCol Inteiro                    'Declara XCol do tipo inteiro

Figura FigDisco, "Disco.gif"             'Cria uma figura de nome FigDisco

XCol = 0                                 'Inicia XCol com o valor igual a zero
Enquanto XCol < 72                       'Enquanto XCol for menor do que
72
  DesenheFiguras                         'Desenha todas as figuras
  Figura FigDisco.Coluna = XCol          'Define a posição da coluna da
figura = XCol
  XCol++                                 'Incrementa mais 1 na variável
XCol
  Aguarde 50                             'Aguarda 50 milésimos de segundo
Repita                                   'Vai para a linha Enquanto XCol <
72
```

O comando **DesenheFiguras** possui um argumento denominado **Preserve** que se for informado fará com que esse comando não apague as figuras anteriores antes de desenhar as novas figuras, desse modo todas as figuras serão desenhadas na janela, execute o exemplo a seguir e veja o resultado:

```
Variavel XCol Inteiro

Figura FigDisco, "Disco.gif"

XCol = 0
Enquanto XCol < 72
  DesenheFiguras Preserve
  Figura FigDisco.Coluna = XCol
  XCol++
```

```
  Aguarde 50
Repita
```

Exercícios

1) Faça com que a figura **Disco.gif** movimente mais rapidamente da esquerda para a direita. Há duas maneiras: incrementando o valor da variável **XCol** com valores acima de 1, ou diminuindo-se o tempo de espera do comando **Aguarde**.

2) Faça com que a figura **Disco.gif** movimente verticalmente na janela (da primeira linha até a última). Para isso incremente a propriedade **Linha** da figura ao invés da propriedade **Coluna**.

Som e Vídeo

O Logic Basic possui dois componentes denominados **Som** e **Video** para reproduzir arquivos de som e vídeo respectivamente.

Para criar um novo componente de som basta informar o comando **Som**, em seguida o nome do componente a critério do programador, vírgula e o caminho do arquivo de som que pode ser qualquer arquivo de som válido (**wav, mid, mp3, etc.**), por exemplo:

```
Som Telefone, "Telefone.wav"
```

Para tocar o som que foi definido para esse componente, basta executar o método **Tocar** conforme o exemplo a seguir:

```
Som Telefone.Tocar
```

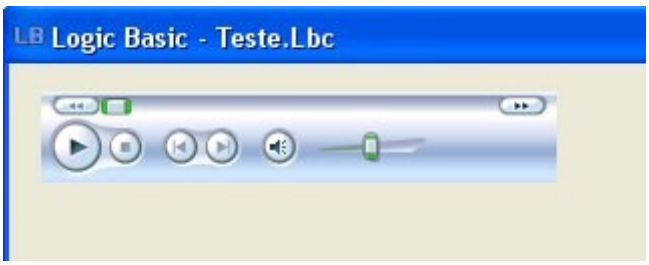
Pode-se criar vários componentes de som e tocá-los ao mesmo tempo, que os sons reproduzidos serão mixados.

Para encerrar o som de um determinado componente antes de seu término, deve-se executar o método **Fim:**

```
Som Telefone.Fim
```

Você também pode tornar o componente de som visível na janela do Logic Basic, para isso basta definir a propriedade **Visivel** igual a **SIM** e informar a linha e coluna onde o componente de som deverá ser posicionado na janela:

```
Som Telefone.Visivel = SIM
Som Telefone.Linha = 1; Som Telefone.Coluna = 2
```



Para criar um novo componente de vídeo o procedimento é praticamente o mesmo do componente de som, ou seja, deve-se informar o comando **Video**, em seguida o nome do componente, vírgula e o caminho do arquivo de imagem:

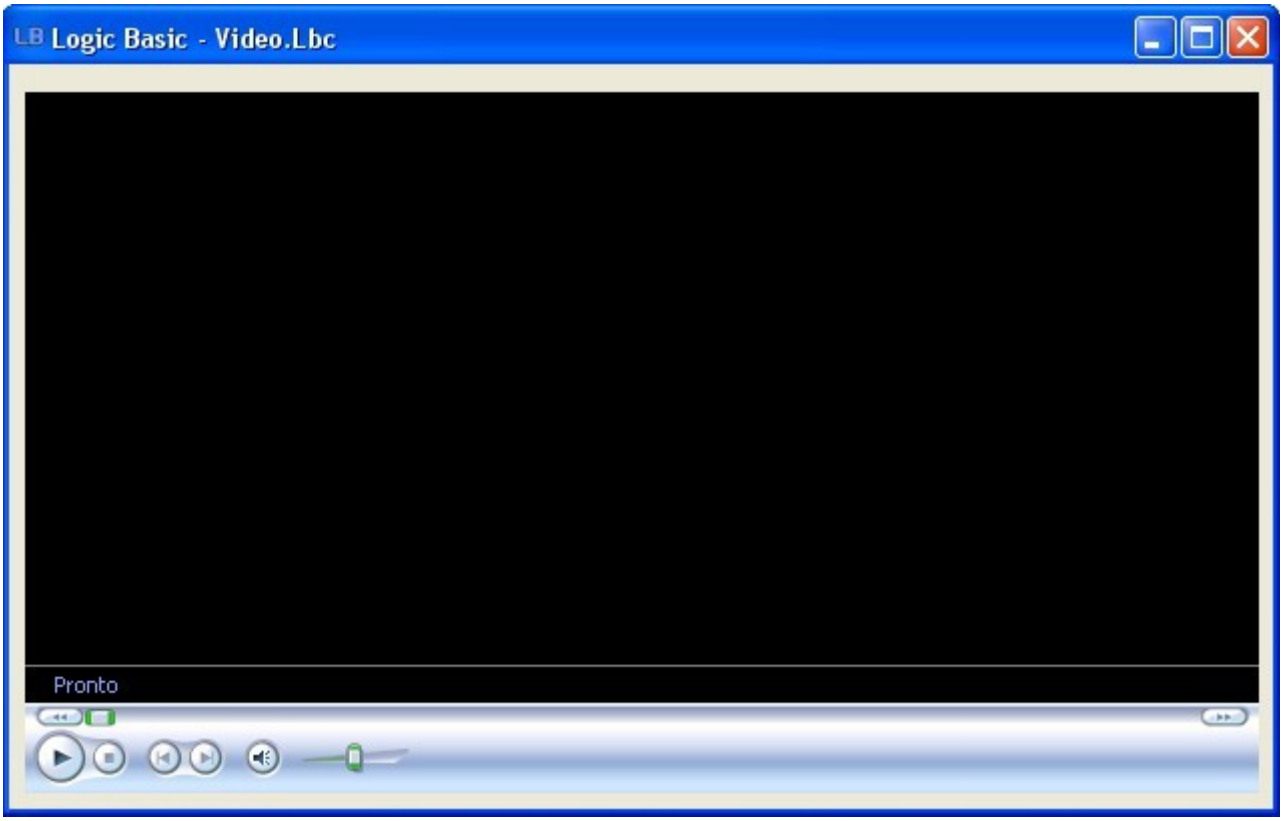
```
Video Teste, "D:\Videos\Filme.avi"
```

Em seguida deve-se definir a linha e coluna a posicionar na janela do Logic Basic, e a altura e largura do componente de vídeo, definindo-se as respectivas propriedades:

```
Video Teste.Linha = 1
Video Teste.Coluna = 1
Video Teste.Altura = 25      'altura em linhas
Video Teste.Largura = 77   'largura em colunas
```

Feito isso, basta definir o componente como visível e executar o método **Tocar** para que o vídeo seja apresentado na tela do componente:

```
Video Teste.Visivel = SIM
Video Teste.Tocar
```



Sub-rotinas, Funções e Temporizadores

Sub-rotinas e Funções são recursos que permitem ao programador criar rotinas (trechos de código que executarão uma determinada função), e que podem ser executadas a qualquer momento dentro do programa principal. A diferença entre Sub-rotina e Função é que essa última pode retornar valores, enquanto a sub-rotina simplesmente executa um trecho de código sem retornar valores.

Para se criar uma sub-rotina ou função primeiramente deve-se declarar as mesmas, e isso deve ser feito sempre após o comando **FimPrograma** ou **FimJanela**. Se elas forem colocadas dentro de uma das extensões de código do Logic Basic, não é necessário informar o comando **FimPrograma** no código principal, pois o Logic Basic já insere esse comando automaticamente no fim do código principal quando o programa é executado.

Sub-rotinas

Para declarar uma sub-rotina deve-se informar o comando **Sub**, em seguida o nome da sub-rotina e abre-fecha parênteses, conforme o exemplo abaixo:

```
Sub Teste()
'Código da sub-rotina
FimSub
```

Note que para todo comando **Sub** deve-se informar seu respectivo comando **FimSub**, que indica onde a sub-rotina termina.

Feito a declaração da sub-rotina, deve-se informar o código da mesma, que deve ficar entre o comando **Sub** e **FimSub**, por exemplo:

```
Sub Teste()
Escreva "Alô marciano!"
FimSub
```

Essa sub-rotina, nomeada de "Teste", irá escrever a frase "Alô marciano!" na janela do Logic Basic, então para executá-la basta informar seu nome seguido de abre-fecha parênteses em qualquer ponto do programa, por exemplo:

```
Teste()

Escreva "Fim do programa!"

FimPrograma

Sub Teste()
Escreva "Alô marciano!"
FimSub
```

No programa acima a sub-rotina foi chamada na primeira linha, então o Logic Basic executou o código da mesma escrevendo "Alô marciano", em seguida

retornou para a próxima linha após a sub-rotina e escreveu a frase "Fim do programa".

Funções

Uma função pode receber argumentos (parâmetros) e também retornar valores ou strings, enquanto sub-rotinas não possuem esses recursos.

Para declarar uma função deve-se informar o comando **Funcao** em seguida abrir parêntese e informar os nomes dos argumentos e o seu tipo (caso hajam argumentos), em seguida fecha-se o parêntese e informa-se o tipo de variável que ela retornará (caso ela retorne valores ou strings), por exemplo:

```
Funcao Exemplo(Argumento1 Tipo, Argumento2 Tipo) Tipo
```

Por exemplo, vamos criar uma função para somar 2 valores inteiros que serão passados como argumentos, e retornar o resultado da soma:

```
Funcao Soma(X Inteiro, T Inteiro) Inteiro
Variavel RET Inteiro

RET = X + T

Retorne RET
FimFuncao
```

Note no exemplo acima que para cada função declarada deve-se declarar seu respectivo comando **FimFuncao** que indica onde a função termina. Note também que utilizamos um comando denominado **Retorne** que efetua o retorno pela função dos valores ou expressões passados como argumento.

Um detalhe importante é que o valor retornado deve sempre ser do mesmo tipo declarado para a função, que no caso do nosso exemplo é do tipo **Inteiro**.

Quando declaramos os argumentos e seus respectivos tipos, a cada chamada da função o Logic Basic irá criar variáveis para esses argumentos, que serão destruídas após o encerramento da execução da função, ou seja, são variáveis locais, e o nome desses argumentos pode ser igual ao de outras funções sem causar conflitos.

Explicando o exemplo anterior, declaramos uma função de nome **Soma** que possui dois argumentos, **X** e **T** do tipo **Inteiro**, e essa função irá retornar valores do tipo **Inteiro**. Na próxima linha declaramos uma variável local de nome **RET** do tipo inteiro, e em seguida efetuamos a soma dos valores das variáveis **X** e **T** e atribuímos o resultado para a variável **RET**. Feito isso, efetuamos o retorno desse resultado passando a variável **RET** como argumento do comando **Retorne**.

Então, para executar a função deve-se atribuir o seu retorno para uma variável, ou passá-la como argumento para outro comando ou função do Logic Basic, e os argumentos devem ser informados sempre entre parênteses e na mesma ordem que foram declarados, por exemplo:

```
Variavel X Inteiro

X = Soma(5, 2)
Escreva "O valor da soma de 5 + 2 é igual a ", X

FimPrograma

Funcao Soma(X Inteiro, T Inteiro) Inteiro
Variavel RET Inteiro

RET = X + T

Retorne RET
FimFuncao
```

Temporizadores

Temporizadores são sub-rotinas que se auto executam em determinados intervalos de tempo, sem a necessidade de serem chamadas no código do programa. São úteis para monitorar eventos que ocorrem no programa em tempo real, como por exemplo verificar o que está sendo digitado em uma caixa de texto, ou até mesmo criar um relógio que fica ativo enquanto outras rotinas do programa são executadas.

Para declarar um temporizador deve-se informar o comando **Sub**, em seguida a palavra Temporizador, sub-linha e o nome do temporizador a critério do programador, a seguir coloca-se o valor do intervalo (em centésimos de segundo) entre parênteses, por exemplo:

```
Sub Temporizador_Teste(50)
```

Há também o comando **Temporizador** que possui dois métodos: **Ativa** e **Desativa**. No método **Ativa** deve-se informar como argumento o valor do intervalo em centésimos de segundo, por exemplo:

```
Temporizador Teste.Desativa
Temporizador Teste.Ativa, 20
```

No exemplo acima o temporizador de nome **Teste** foi desativado, e em seguida ativado novamente com um novo valor para o intervalo (20 centésimos de segundo).

Abaixo um exemplo de um temporizador que faz com que as letras digitadas em uma caixa de texto sempre sejam em maiúsculas:

```
Texto Teste, 5, 5

FimPrograma

Sub Temporizador_Maiuscula(10)
  Variavel Comp Inteiro
```

```
  Texto Teste.Txt = Formata(Teste.Txt, ">")  'Coloca o texto em maiúsculas
  Comp = Comprimento(Teste.Txt)
  Texto Teste.PosCursor = Comp  'Posiciona o cursor no final do texto
FimSub
```

Exercícios

1) Crie uma função de nome **Divide** que receba dois argumentos inteiros e retorne um resultado decimal resultante da divisão do primeiro argumento pelo segundo.
2) Crie um temporizador que apresente um relógio na janela do Logic Basic. Para isso crie um rótulo de nome **Relogio** e dentro do temporizador atribua o horário atual para o texto do rótulo.

Trabalhando com textos

O Logic Basic possui diversos comandos para trabalhar com textos, unindo-os, separando-os e modificando-os.

Unir

Para unir dois ou mais textos deve-se utilizar o comando **Unir**, para isso basta informar os textos ou strings a serem unidas como argumentos, e até mesmo funções que retornam strings podem ser passadas como argumento, que o comando irá retornar o resultado em uma nova string:

```
Variavel Gato String, Rato String, Resultado String

Gato = "O gato"
Rato = " e o rato"
Resultado = Unir(Gato, Rato, " são bons amigos!")
Escreva Resultado
```

Note que em todo comando que retorna valores ou strings, os argumentos DEVEM ser colocados entre parênteses. Quando um comando não retornar valores, os parênteses não devem ser colocados.

Operador de União

O Operador de união **&** permite executar a união de strings ou o conteúdo de variáveis, de uma maneira mais simples e com uma execução mais rápida do que o comando **Unir**, mas não permite informar funções na linha de comando, somente strings ou variáveis de qualquer tipo, por exemplo:

```
Var S1 String, Idade Inteiro

S1 = "Maria possui "
Idade = 20
```

```
S1 = S1 & Idade & " anos de idade"
Escreva S1
```

Após a união das strings e variáveis no programa acima, o conteúdo da variável S1 será "Maria possui 20 anos de idade".

Separe

O comando **Separe** separa trechos de um texto, de acordo com dois argumentos: o primeiro é a posição inicial no texto, e o segundo é o comprimento em caracteres ou letras do texto a ser separado. Por exemplo:

```
Variavel Resultado

Resultado = Separe("Hoje é um dia chuvoso!", 15, 7)
Escreva Resultado
```

A posição da primeira letra do texto "Hoje é um dia chuvoso!" é 1, portanto a posição da primeira letra da palavra "chuvoso" é 15; o seu comprimento em letras é 7, portanto, o conteúdo da memória **Resultado** será **chuvoso**.

Note que não informamos o tipo da variável **Resultado** na declaração da mesma, pois quando o tipo é omitido o Logic Basic assume o tipo da variável como sendo **String**.

Formata

O comando **Formata** formata uma string ou número de acordo com um determinado formato, denominado "máscara". A máscara pode escrita de duas formas: com o símbolo **#** ou com o número 0 (zero). A seguir alguns exemplos do comando **Formata**:

```
Variavel X Decimal, S String, Resultado String

X = 1123.57
Resultado = Formata(X, "##,##0.00")
Escreva Resultado

'O Resultado do exemplo acima é 1.123,57.

X = 10
Resultado = Formata(X, "00000")
Escreva Resultado

'O resultado do exemplo acima é 00010.

S = "123,5"
Resultado = Formata(S, "00000.00")
Escreva Resultado

'O resultado do exemplo acima é 00123,50.
```

```
S = "logic basic"
Resultado = Formata(S, ">")
Escreva Resultado

'o resultado do exemplo acima é LOGIC BASIC

S = "1234.5"
Resultado = Formata(S, Padrao)
Escreva Resultado
```

O resultado do exemplo acima é **1.234,50**. Nesse último caso, o número é formatado no padrão de moeda com duas casas decimais, sendo recomendável a sua utilização, pois esse formato não depende das configurações de moeda do Windows, que pode causar erros caso a configuração de moeda esteja em um padrão incompatível.

Exercícios

1) Crie mais algumas variáveis no programa de exemplo do comando **Unir**; atribua textos a elas e junte-as com os outros argumentos do comando **Unir**.

2) No exemplo do comando **Separe**, modifique os valores da posição inicial e comprimento, execute o programa e veja o resultado. Depois retire o segundo argumento **comprimento** e veja o resultado.

3) Digite os exemplos do comando **Formata** no código do Logic Basic e execute o programa; depois altere os números e as máscaras para ver o resultado.

Box de Listagem

O Logic Basic possui um componente denominado **Box** que cria uma caixa de listagem destinada a listar textos alinhadamente em colunas, com várias configurações. Para criar um novo box de listagem deve-se informar o comando **Box** e em seguida o seu nome, a linha e coluna inicial, e a altura do componente em linhas. A largura será definida automaticamente pela soma do total das colunas dos campos de título:

```
Box Alunos, 2, 35, 5
```

Titulos

Para definir as colunas com seus respectivos títulos no box de listagem, deve-se utilizar o método **Titulo** e em seguida a descrição do título e a sua largura em caracteres:

```
Box Alunos.Titulo, "Nome", 30
Box Alunos.Titulo, "Idade", 5, Centro, Numerico
```

Note que no título "Idade" foram informados mais dois argumentos: Justificação e Tipo, o primeiro pode ser Dir (Direita), Centro (Centro) ou Esq (Esquerda); o segundo informa que a coluna irá armazenar dados numéricos.

Ativa

Quando se define a posição do box de listagem e suas colunas, o box ainda não será apresentado na janela, isso somente acontecerá quando se executar o método **Ativa**:

```
Box Alunos.Ativa
```

Adicionando itens no box de listagem

Para adicionar novos itens na caixa de listagem deve-se utilizar o método **Adiciona** ou simplesmente **Add**. A cada chamada do método **Adiciona** será inserido o texto na próxima coluna da linha atual do box de listagem, na ordem da primeira para a última coluna. Ao serem preenchidas todas as colunas da linha, deve-se executar o método **NovaLinha** para que seja criada uma nova linha no box de listagem, por exemplo:

```
Box Alunos.Adiciona "Thaissa"
Box Alunos.Adiciona "17"
Box Alunos.NovaLinha
```

Após inseridos todos os dados no box de listagem, é necessário executar o método **Fim**, por exemplo:

```
Box Alunos.Fim
```

Feito isso o box de listagem já estará pronto para ser utilizado com todos seus recursos. A seguir um exemplo de criação e inserção de dados no box de listagem:

```
Box Alunos, 2, 35, 5
Box Alunos.Titulo, "Nome", 30
Box Alunos.Titulo, "Idade", 5, Centro, Numerico
Box Alunos.Ativa

Box Alunos.Add "Thaissa"; Box Alunos.Add "17"
Box Alunos.NovaLinha
Box Alunos.Add "Junara"; Box Alunos.Add "18"
Box Alunos.NovaLinha
Box Alunos.Add "Camila"; Box Alunos.Add "16"
Box Alunos.NovaLinha
Box Alunos.Fim

Retorno = "CLIQUEBOX"
Enquanto Retorno = "CLIQUEBOX"
  Aguarde CliqueBox
  Escreva "Retorno: ", Retorno, " - ", Alunos.Txt(0), ", ", Alunos.Txt(1)
```

```
Repita

Escreva "Selecionado: ", ValorObjeto, " - ", Alunos.Txt(0), ", ",
Alunos.Txt(1)
```

Quando os títulos são definidos, o Logic Basic cria uma matriz cujo nome será o nome do box seguido de .**Txt,** e que possui uma quantidade de elementos igual à quantidade de colunas do box de listagem. Sendo assim, quando uma linha do box for selecionada ou clicada, os textos de cada coluna serão atribuídos aos respectivos elementos dessa matriz, e na variável reservada **Retorno** a ação executada, que poderá ser **CLIQUEBOX** ou **DUPLOCLIQUEBOX**, ou seja, um clique na linha do box (ou selecionamento pelas teclas de setas), ou um duplo-clique na linha do box (ou pressionamento da tecla ENTER) respectivamente.

Como podem existir mais de um box de listagem na mesma janela, o Logic Basic atribui o nome do box que está sendo utilizado à variável reservada **ValorObjeto**.

O comando **Aguarde CliqueBox** faz com que a execução do programa aguarde até que haja um clique ou duplo-clique sobre o box, ou mesmo o selecionamento via teclado.

Escrevendo e desenhando na impressora

O Logic Basic possui um componente denominado **Imp** para imprimir textos e desenhos na impressora. Esse comando possui alguns métodos que são idênticos aos comandos para escrever na Janela, com algumas pequenas diferenças. A quantidade de linhas e colunas do papel depende da configuração da impressora, mas o padrão é aproximadamente 70 linhas por 100 colunas para um papel de tamanho A4.

Para definir a coordenada (linha, coluna) a imprimir, deve-se utilizar o método **Posicione**, e em seguida o número da linha e coluna separados por vírgula, por exemplo:

```
Imp.Posicione 10, 15
```

Para definir o fonte de caracteres dos textos a imprimir, deve-se utilizar o método **Fonte** e em seguida os atributos de fonte da seguinte maneira:

```
Imp.Fonte Nome = "Ms Sans Serif", Tamanho = 38, Negrito = SIM,
```

```
Italico = SIM
```

Para escrever no papel da impressora, deve-se utilizar o método **Escreva** e em seguida o texto a ser escrito na posição definida pelo método **Posicione:**

```
Imp.Escreva "Logic Basic"
```

Você poderá informar vários argumentos no método **Escreva**, assim como é feito no comando de mesmo nome.

Para desenhar uma figura na posição definida pelo comando **Posicione**, deve-se utilizar o método **Desenhe** e em seguida o caminho do arquivo de imagem:

```
Imp.Desenhe "Satelite.gif"
```

Para mudar de página, basta informar o método **NovaPagina**:

```
Imp.NovaPagina
```

No final do trabalho de impressão, deve-se executar o método **Fim**, para encerrar e descarregar todos os buffers de impressão:

```
Imp.Fim
```

A seguir, um exemplo de como escrever e desenhar na impressora:

```
Var S String

Imp.Fonte Nome = "Ms Sans Serif", Tamanho = 38, Negrito = SIM,
Italico = SIM

Imp.Posicione 0, 0
Imp.Escreva "Logic Basic"
Imp.Posicione 10, 0
Imp.Escreva "Teste de impressora"
Imp.Posicione 20, 0
Imp.Desenhe "Disco.gif"
Imp.Posicione 30, 0
Imp.Desenhe "Satelite.gif"

Imp.NovaPagina

S = "Fim do relatório!"
Imp.Fonte Nome = "Arial", Tamanho = 20, Negrito = NAO, Italico =
NAO
Imp.Posicione 0, 0
Imp.Escreva "Mensagem: ", S

Imp.Fim

Mensagem "Relatorio Impresso com Sucesso !"
```

Banco de Dados Nativo do Logic Basic

O Logic Basic possui uma implementação nativa de banco de dados muito simples e fácil de usar, mas que contém poderosos recursos que permitem inclusive sua utilização em ambiente de rede. As informações gravadas no banco de dados são protegidas por um algoritmo de criptografia super seguro, e que podem ser acessadas somente pelo Logic Basic através de senha.

Criando um Banco de Dados Nativo

Para criar um banco de dados em Logic Basic, deve-se utilizar o comando **BD**. Cada Banco de Dados se refere a um arquivo e uma tabela, podendo ser criados vários Bancos de Dados, um para cada tabela.

Para criar um banco de dados, deve-se digitar o comando **BD** e em seguida três argumentos: o primeiro é o nome do Banco de Dados a critério do programador, o segundo, o caminho onde será gravado o arquivo do Banco de Dados e o terceiro a senha do banco de dados. O nome do arquivo não deverá ser informado pois o mesmo será criado automaticamente pelo Logic Basic (nome do banco de dados com a extensão **.LBA**).

Nas próximas linhas deve-se definir os campos da tabela (um em cada linha), informando-se o nome do campo entre aspas, o seu tipo e o seu tamanho (esse último somente no caso de tipo String). Ao final da definição, deve-se colocar o comando **FimBD**.

Por exemplo, vamos criar um Banco de Dados de um arquivo de Clientes:

```
BD Clientes, "C:\LogicBasic\Arquivos", "12345"
  "Nome", String, 30
  "Idade", Inteiro
  "ValorCompras", Decimal
FimBD
```

Abrindo e Fechando um Banco de Dados

Após a criação de um Banco de Dados, deve-se abri-lo com o comando **AbreBD** para permitir a gravação e leitura de registros. Nesse comando deve-se informar como argumentos apenas o nome do arquivo e a senha, pois o seu caminho já está definido no comando **BD**.

```
AbreBD Clientes, "12345"
```

Para fechar um banco de dados, deve-se utilizar o comando **FechaBD** e em seguida o nome do banco de dados, por exemplo:

```
FechaBD Clientes
```

Você poderá abrir um novo arquivo com essa mesma estrutura do Banco de Dados, mas primeiramente você deverá fechar o anterior com o comando **FechaBD**. Por exemplo, vamos supor que você defina um banco de dados para armazenar os movimentos de um sistema de contabilidade do ano todo, mas que você queira gravar o movimento de cada mês em arquivos separados. Sendo assim você poderá abrir o banco de dados em um arquivo para gravar os movimentos em um mês, fechar o arquivo, e abrir o banco de dados com outro nome de arquivo para gravar o movimento de outro mês, para isso basta alterar o nome do arquivo no comando **BD**.

Gravando registros no Banco de Dados

Ao criarmos o Banco de Dados, o Logic Basic cria uma variável para cada campo da tabela que foi definida, para que sejam colocados os dados nos campos antes que o registro seja gravado no arquivo. Para colocar um dado em um campo, basta informar o nome do Banco de Dados seguido de ponto e o nome do campo, por exemplo, vamos preencher os campos do Banco de Dados do exemplo anterior:

```
Clientes.Nome = "João da Silva"
Clientes.Idade = 30
Clientes.ValorCompras = 1500.00
```

Essas variáveis tem as mesmas características de uma variável normal, e você poderá atribuir strings, números e expressões da mesma forma como se atribui a uma variável.

Após ter preenchido os campos com seus respectivos dados, para adicionar um novo registro ao Banco de Dados, basta utilizar o comando **AdicionaRegistro** e em seguida o nome do Banco de Dados, por exemplo:

```
AdicionaRegistro Clientes
```

Com esse comando, as informações contidas nas variáveis de campos do Banco de Dados **Clientes** serão gravados em um novo registro no arquivo.

Lendo registros do Banco de Dados

Para ler um registro do Banco de Dados é muito simples, basta executar o comando **LeRegistro**, em seguida o nome do Banco de Dados e a posição do registro entre parênteses. A posição do registro pode variar de 0 ao total de registros do arquivo menos um. Por exemplo:

```
S = LeRegistro(Clientes, 3)
```

O exemplo anterior lê o registro de número 3 do arquivo, e retorna uma string contendo todos os campos do registro lido ou uma string vazia caso o registro não exista ou tenha sido excluído.

Após a leitura do registro, os dados lidos estarão disponíveis nas variáveis de campos, por exemplo:

```
Escreva Clientes.Nome, " - ", Clientes.Idade, " - ",
        Clientes.ValorCompras
```

Para saber a quantidade de registros de um Banco de Dados, basta utilizar o comando **TotalRegistrosBD** seguido do nome do Banco de Dados entre parênteses:

```
Escreva TotalRegistrosBD(Clientes)
```

Um detalhe importante é que a quantidade de registros retornada pelo comando **TotalRegistrosBD** inclui registros excluídos, então caso o programador necessite saber a quantidade exata de registros ativos é necessário executar o comando **LimpaBD** antes de executar o comando **TotalRegistrosBD**.

A seguir, um exemplo de como ler todos os registros do Banco de Dados **Clientes**:

```
Variavel X Inteiro, T Inteiro, S String

T = TotalRegistrosBD(Clientes)
X = 0
Enquanto X < T
  S = LeRegistro(Clientes, X)
  Se S <> ""
          Escreva Clientes.Nome, " - ", Clientes.Idade, " - ",
           Clientes.ValorCompras
  FimSe
  X++
Repita
```

Note que após ler o registro, antes de escrevermos os campos na janela, verificamos se o retorno do comando **LeRegistro** é diferente de uma string vazia, para pular os registros excluídos caso existam.

Leitura ordenada de registros

A leitura de registros que fizemos no exemplo anterior, lê os registros na sequência em que foram gravados. Mas podemos ler os registros em qualquer ordem que desejarmos, bastando utilizar o comando **OrdemBD**, que ordena o arquivo por um campo, ou pela combinação de vários campos. Para isso, devemos passar como argumentos o nome do Banco de Dados e em seguida o nome do(s) campo(s) em que o arquivo será ordenado:

```
OrdemBD Clientes, "Idade", "Nome"
```

O comando acima ordena o Banco de Dados de Clientes por ordem de Idade e

Nome. Essa é uma ordenação composta, ou seja, por mais de um campo, e no caso desse exemplo, o arquivo será ordenado primeiramente por idade, e se houver um grupo de idades iguais, os nomes que fazem parte desse grupo serão ordenados em ordem alfabética. A seguir, um exemplo de leitura ordenada:

```
Variavel X Inteiro, T Inteiro, S String

OrdemBD Clientes, "Idade", "Nome"

T = TotalRegistrosBD(Clientes)
X = 0
Enquanto X < T
  S = LeRegistro(Clientes, X)
  Se S <> ""
          Escreva Clientes.Nome, " - ", Clientes.Idade, " - ",
           Clientes.ValorCompras
  FimSe
  X++
Repita
```

O comando OrdemBD gera um índice de ordenação para o Banco de Dados, e deve ser colocado sempre antes da rotina que irá utilizar o índice. Se você necessitar de um novo índice com outra ordenação, bastar executar o comando OrdemBD novamente antes da rotina que irá utilizar o índice. A partir do momento em que for definido um índice para o Banco de Dados, toda leitura que for feita será ordenada pelo índice. Se você desejar voltar a ler o arquivo sequencialmente, basta informar o argumento **Sequencial** após o comando OrdemBD:

```
OrdemBD Clientes, "Sequencial"
```

Procura de registros no Banco de Dados

Você pode procurar um determinado registro no Banco de Dados, de acordo com o índice atual do Banco de Dados, bastando executar o comando **Procure,** seguido do nome do Banco de Dados, uma das palavras-chave **Primeiro, Ultimo** ou **Proximo**, e a string a ser procurada, ou combinação de strings de acordo com o índice. Por exemplo, vamos supor que você deseje procurar o registro que contenha o nome "João da Silva" no Banco de Dados:

```
Variavel  Posicao Inteiro, S String

OrdemBD Clientes, "Nome"

Posicao = Procure(Clientes, "Primeiro", "João da Silva")
Se Posicao = -1
  Mensagem "Registro não encontrado !"
CasoContrario
  S = LeRegistro(Clientes, Posicao)
       Escreva Clientes.Nome, " - ", Clientes.Idade, " - ",
Clientes.ValorCompra
```

FimSe

No exemplo acima, o Banco de Dados é ordenado por "Nome", em seguida é executado o comando **Procure** que retorna -1 caso a string procurada não seja encontrada, ou a posição do registro, caso haja um registro contendo o campo "Nome" igual ao nome procurado. Nesse último caso, basta executar o comando **LeRegistro** informando a posição retornada.

Os identificadores **Primeiro** e **Ultimo** são utilizados no caso em que houverem vários registros com campos de ordenação iguais, por exemplo, vamos supor que no arquivo existam três nomes "João da Silva"; se for informado o identificador **Primeiro**, será retornada a posição do primeiro dos três registros, se for informado o identificador **Ultimo**, será retornada a posição do último dos três registros. O identificador **Proximo** retorna a posição do registro com campos de ordenação mais próximo(s) daquele que está sendo procurado.

Exclusão de registros do Banco De Dados

Para excluir um registro do Banco de Dados basta executar o comando **ExcluiRegistro**, seguido do nome do Banco de Dados e a posição do registro a ser excluído. A posição a ser excluída dependerá do índice atual do Banco de Dados, ou seja, poderá ser sequencial ou pelo índice. Assim, se o Banco de Dados não possuir ordenação, será excluído o registro da posição absoluta informada, se o Banco de Dados possuir ordenação, será excluída a posição relativa ao índice. Assim, você poderá excluir o registro da posição retornada pelo comando **Procure**. Por exemplo:

ExcluiRegistro Clientes, Posicao

O Logic Basic também possui um comando denominado **LimpaBD**, que pode ser utilizado para limpar os registros excluídos do arquivo, tornando-o menor e mais leve. Esse comando não é obrigatório de ser executado após a exclusão de registros no arquivo, somente quando houver um grande número de exclusões acumuladas, e pode ser utilizado a critério do programador. Para executar esse comando, basta informar como argumento o nome do Banco de Dados:

LimpaBd Clientes

Programas Executáveis

O Logic Basic permite que os programas sejam compilados para que seja gerado um programa executável, que pode ser copiado para outro computador e executado sem a necessidade da instalação do Logic Basic. Outra vantagem desse recurso, é que os programas executáveis não podem ser lidos por nenhum programa, nem mesmo o editor de texto do Logic Basic, pois estão em código binário.

Para compilar um programa é muito fácil, basta pressionar o botão **Compilar** no ambiente de código do Logic Basic, informar o nome do programa executável, que deverá ter a extensão **EXE**, se não for informada a extensão, o

Logic Basic a colocará automaticamente.

Para executar um programa compilado, basta dar um duplo-clique sobre o programa executável que o mesmo será executado imediatamente.

Depuração de programas

Depurar é sinônimo de "Tirar Defeitos", "Solucionar problemas", "Remover bugs", de um programa de computador. O Logic Basic oferece um recurso para facilitar esse trabalho, e para isso foi criado o botão **Depurar**. Ao ser pressionado esse botão, será ativado o "modo de duparação". Sendo assim, quando o programa for executado, o número do editor, a linha e código da linha irão sendo mostrados em uma pequena janela, à medida que o programa for sendo executado.

Dessa maneira, o programador irá visualizando passo a passo o que o interpretador do Logic Basic está executando, ou seja, verá ao mesmo tempo a linha de código e o seu resultado na janela. Com esse recurso, ficará fácil detectar algum erro no programa, pois se o Logic Basic não executar o programa de acordo com aquilo que é esperado, com certeza há algum erro na linha de código onde ocorre o erro. O programador deverá então verificar se não há algum comando escrito incorretamente, verificar se as variáveis utilizadas foram declaradas corretamente, etc.

Variáveis Reservadas e Cores

O Logic Basic possui algumas variáveis reservadas, ou seja, variáveis que já são criadas automaticamente assim que um programa é executado, por isso você não poderá criar variáveis, sub-rotinas ou funções com esses nomes, para evitar conflitos em seu programa.

Existem também várias variáveis contendo valores de cores, mas você poderá criar outras além dessas utilizando o comando RGB.

A seguir, a relação das variáveis reservadas do Logic Basic:

Amarelo Inteiro, AmareloEscuro Inteiro, AreaTrabalho String, Azul Inteiro, AzulClaro Inteiro, AzulEscuro Inteiro, Branco Inteiro, CliqueMouse String, ColunaMouse Inteiro, CorFundoPadrao Inteiro, GanhouFoco String, LinhaMouse Inteiro, MenuProgramas String,
PastaLB String, PastaPrograma String, PastaWindows String, PerdeuFoco String, PonteiroSQL String, Preto Inteiro, Retorno String, RetornoCampo Inteiro, TamLin Inteiro, TamCol Inteiro, TestaBotao String, TotalFontes Inteiro, ValorObjeto String, ValorReferencia String, ValorRetorno Inteiro, Verde Inteiro, VerdeEscuro Inteiro, Vermelho Inteiro, VermelhoEscuro Inteiro