# Apostila Visual Basic

## 1 - Uso do Visual Basic

A interface do Visual Basic consiste nos seguintes elementos :

- **Toolbar -** Providencia acesso rápido ao comandos mais usados no ambiente de programação. Você clica no ícone na Toolbar para que seja executada a ação representada por este ícone.

| Ação | Equivalência no Menu |
|---|---|
| Cria um novo *Form* ou janela de interface com o usuário | Comando *New Form* no menu *File* |
| Cria um novo módulo | Comando *New Module* no menu *File* |
| Abre um projeto já existente | Comando *Open Project* no menu *File* |
| Salva o projeto corrente | Comando *Save Project* no menu *File* |
| Mostra a janela *Menu Design* | Comando *Menu Design* no menu *Window* |
| Mostra a janela *Properties* | Comado *Properties* no menu *Window* |
| Começa a execução dentro do modo projeto | Comando *Start* no menu *Run* |
| Para a execução do programa durante a sua execução ( Pausa) | Comando *Break* no menu *Run* |
| Para a execução da aplicação e retorna ao modo de projeto | Comando *End* no menu *Run* |
| Ativa um breakpoint (ponto de parada) na linha corrente | Comando *Toggle Breakpoint* no menu *Debug* |
| Mostra o valor da seleção corrente na janela Code | Comando Instant Watch no menu Debug |
| Mostra a estrutura das chamadas ativas | Comando *Calls* no menu *Debug* |
| Executa uma linha de código de cada vez (passo a passo) | Comando *Single Step* no menu *Debug* |
| Executa um procedimento ou sentença de cada vez | Comando *Procedure Step* no menu *Debug* |

- **Toolbox** - Providencia um conjunto de ferramentas que você usa durante o projeto para colocar controles dentro de seu *form* (janela de interface com o usuário).
- **Menu Bar** - Mostra os comandos usados para construir a sua aplicação.
- **Form** - É a janela na qual voce irá construir a interface da sua aplicação. No form você irá adicionar controles , gráficos , figuras que irão criar a aprencia final da sua aplicação.
- **Project Window** - Lista os forms , módulos de código , arquivos *Custom Controls* que compoêm a sua aplicação.
- **Properties Window -** Lista as propriedades existentes para um form ou controle selecionado. Uma *property* (propriedade) é um valor ou característica associada a um objeto tais como tamanho , *caption* ou côr.

## 2 - Criando Aplicações com Visual Basic

- Diferença entre uma Aplicação Tradicional e outra Event-Driven ou seja ativada por eventos.

Em uma aplicação tradicional ou programada de forma *procedural* , a aplicação por si só controla que porções do código serão executadas. A execução começa na primeira linha de código e segue um caminho pré-definido através de toda a aplicação , chamando subrotinas conforme for sendo necessário.

Em uma aplicação controlada por eventos ou *event-driven ,* uma ação do usuário ou do sistema , ativa um procedimento associado a este evento. Assim a ordem através do qual o seu código de programa é executado depende de quais eventos ocorrem , que por sua vez estes eventos dependem das ações tomadas pelo usuário. Esta é a essência das Interfaces Gráficas e da Programaçao Ativada por Eventos.

- Passos para a criação de uma aplicação

A. Criar a interface
B. Setar as propriedades dos controles
C. Escrever o código

A. Criar a interface.

O primeiro passo na construção de uma aplicação Visual Basic é desenhar os objetos que irão compor a interface. Para inserir o controle no seu *form:*

1. Clique no ícone do controle dentro do *toolbox.*

2. Mova a seta do mouse para dentro da área do *form* , a seta vira uma cruz.

3. Ponha a cruz no ponto dentro do *form* onde irá ficar o canto superior esquerdo do controle escolhido

4. Arraste a cruz até que o controle fique do tamanho desejado (arrastar significa apertar o botão esquerdo do mouse e mante-lo apertado enquanto o objeto é movido com a mudança de posição do mouse)

5. Solte o botão do mouse e controle aparecerá no form.

B. Setar as propriedades dos controles

O próximo passo é setar ( colocar valores de inicialização ) as propriedades dos objetos que você criou. A janela *Properties* proporciona uma maneira fácil de inicializar as propriedades para todos os objetos do *form* . Para abrir a janela de propriedades , escolha o comando *Properties* no menu *Window* ou então clique no botão *Properties* na Barra de Ferramentas ( *Toolbar* )

Object Box - Mostra o nome do objeto para o qual você que inicializar as propriedades. Clique a seta sublinhada a direita do box para selecionar o form ou nome de controle a partir da lista de objetos presentes no form atual.

Settings box - Permite que você edite a inicialização da propriedade selecionada na lista de propriedades. Algumas inicializações podem ser trocadas clicando-se na seta sublinhada existente a direita do box ; será mostrada então uma lista de opções . Você poderá clicar em um item da lista para seleciona-lo

Lista de Propriedades - A coluna esquerda mostra todas as propriedades para um ojeto selecionado, a coluna da direita mostra a inicialização atual para cada uma das propriedades.

C. Escrever o código.

A janela de código é o local onde você escreve o código Visual Basic para a sua aplicação. *Código* consiste em sentenças da linguagem , constantes e declarações. Usando a janela de código ( *Code Window* ) , você pode rápidamente ver e editar qualquer parte do seu código dentro de sua aplicação.

Para abrir a janela de código você deve clicar duas vezes no form ou no controle para o qual você deseja escrever código ou então a partir da janela *Project* , selecione o nome do form e clique no botão *View Code.*

A janela de código ( *Code Window* ) contém os seguintes elementos :

Object Box - Mostra o nome do objeto selecionado. Clique na seta sublinhada a direita para mostrar a lista de todos os objetos associados a este *form*.

Procedures List Box - Lista os procedimentos existentes para um objeto. O box mostra o nome do procedimento selecionado - no exemplo acima *Click* . Clique na seta sublinhada a direita do box para mostrar todos os procedimentos associados a este objeto.

O código ( ou programa) em uma aplicação Visual Basic, é dividido em pequenos blocos chamados *procedures* (procedimentos). Um *event procedure* ( procedimento associado a um evento ) comtém código que é executado quando um evento ocorre ( como por exemplo quando o usuário clica um botão) .

Como criar um *event procedure* ( procedimento associado a um evento ) :

1. No *Object box* , selecione o nome do objeto no form ativo ( *form* que atualmente tem o foco)

2. Na *Procedure List box* , selecione o nome do evento desejado. No exemplo acima a *procedure Click* já estava selecionada, já que ela é a procedure default para o botão de comando. Note que uma *máscara* (template) para a escrita do código para este evento está sendo mostrada na janela de código

3. Digite o código desejado entre as sentenças *Sub* e *End Sub* e a procedure se parecerá com o texto abaixo.

```
Sub Command1_Click ( )

Text1.text = "Hello World!"

End Sub
```

**3 - Criação de Menus**

Menus são criados usando a janela *Menu Design*. Você adiciona items ao menu em tempo de projeto pela criação de controles de menu e setando propriedades que definirão sua aparência.

Para mostrar a janela de *Menu Design* , escolha a opção *Menu Design* no menu *Window* ou então escolha o botão *Menu Design* no *toolbar*.

**4 - Conectando Formulários**

A adição de novos formulários na sua aplicação é feito através da opção File na barra de menu , comando New Form.

Comandos para controle do form :

| Comando | Ação |
|---|---|
| Load form | Carrega o form mas o deixa invisivel |
| Form.Show [modo] | Mostra o form se ele estiver invisivel, se ele não estiver carrega então ele primeiro carrega o form e depois o mostra , estilo refere-se a se o o form que fez a carga do próximo form , fica parado até a desativação do novo form (modo =0 ou modeless) ou se continua a sua execução sem se importar com o proximo form (modo=1 ou modal) |
| Unload form | O form é descarregado da memória e a sua execução é encerrada |

**5 - Usando Controles**

O Toolbox do Visual Basic comtem as ferramentas necessárias para desenhar controles no seu *form* . Cada ferramenta no *Toolbox* representa um controle. Abaixo os controles mais usados :

| Controle | Descrição |
|---|---|
| Pointer | Executa a movimentação e mudança de tamanho um controle. |
| Picture Box | Mostra Bitmaps , ícones ou metafiles ou serve como container para outros controles. |
| Label | Mostra texto que não pode ser alterado pelo usuário. |
| Text Box | Fornece uma área de entrada de dados ou mostra texto. |
| Frame | Fornece um container visual para controles . |
| Command Buttom | Botão de Comando . Executa um comando ou ação quando clicado pelo usuário |
| Check Box | Mostra opção Falso/Verdadeiro ou Sim/Não. Qualquer numero de Check Box podem ser marcados ao mesmo tempo. |
| Option Button | Como parte de um grupo de opções junto com outras opções, mostra multiplas opções, no qual o usuário pode escolher só uma. |
| Combo Box | Combina uma Text Box com uma List Box. Permite ao usuário escolher uma opção a partir de uma lista Drop-Down. |
| List Box | Mostra uma lista de items no qual o usuário poderá fazer uma escolha. |
| Scroll Bar Horizontal | Permite que o usuário selecione um valor dentro de um limite de valores. |
| Scroll Bar Vertical | Permite que o usuário selecione um valor dentro de um limite de valores. |
| Timer | Executa eventos de tempo dentro de intervalos regulares. |
| Drive List Box | Mostra e permite ao usuário selecionar drivers de disco. |
| Directory List Box | Mostra e permite ao usuário selecionar diretórios. |
| File List Box | Mostra e permite ao usuário selecionar a partir de uma lista de arquivos. |
| Shape | Adiciona retângulos, circulos, elipses e circulos ao form de interface com o usuário. |
| Line | Adiciona um segmento de linha ao form. |
| Image | Mostra bitmaps, ícones ou arquivos Metafile; age como um botão de comando quando clicado. |
| Data | Habilita a conexão com um banco de dados e mostra informção dele no seu form. |
| Grid | Mostra uma série de linhas e colunas e permite ao usuário manipular dados em suas células . |
| OLE | Adiciona dados em uma aplicação Visual Basic. |
| Common Dialog | Fornece caixas de dialogo padrão para operações como abrir, salvar, e imprimir arquivos e selecionar fonts de letras e cores. |

Propriedades mais comuns para os controles :

| Propriedade | Função |
|---|---|
| BackColor | Determinar a cor de fundo de um objeto |
| ForeColor | Determina a cor de primeiro plano de um objeto |
| FontName | Determina a font usada para mostrar texto no controle |
| FontSize | Determina o tamanho do font a ser utilizado |
| TabIndex | Determina ordem de salto entre os controles quando se tecla Tab , dentro de um form |
| Enabled | Determina se um controle pode responder a eventos gerados pelo usuário |
| Visible | Determina se um controle é visivel ou não |
| Name | Especifica o nome a ser usado no programa para identificar o objeto |
| BorderStyle | Determina o estilo da borda de um objeto |
| Text | Determina o texto contido na area de edição |
| Caption | Determina o texto mostrado dentro ou próximo ao controle |

Principais Eventos :

| Evento | Descrição |
|---|---|
| Click | Ocorre quando o usuário pressiona e solta o botão do mouse |
| DblClick | Ocorre quando o usuário pressiona o botão do mouse 2 vezes seguidamente |
| GotFocus | Ocorre quando um objeto recebe o foco ( habilitação para receber eventos) |
| LostFocus | Ocorre quando um objeto perde o foco por uma ação de usuário tal como a tecla Tab ou clicando em outro objeto ou pelo comando **SetFocus** |
| KeyPress | Ocorre quando o usuário pressiona e solta uma tecla no teclado com exceção das teclas de função |
| Change | Indica que o conteúdo de um controle foi trocado |
| MouseMove | Ocorre quando o mouse é movido |

## 6 - Tratamento de Arquivos

O Visual Basic possui treis tipos de acesso a arquivo :

. Randomico

. Sequencial (Input , Output e Append)

. Binário

Acesso Randomico - Um arquivo aberto como *random acess* é assumido como sendo composto de uma série de registros (records) de tamanho idêntico. Apesar de um registro corresponder a um unico tipo de dados, podem existir tipos de dados definidos pelo usuário que podem ser usados para criar registros compostos de numerosos campos, onde cada um dele podem ter diferentes tipos de dados. O comprimento de cada registro necessita ser informado como parametro para o comando *Open* usado para abrir o arquivo para acesso randomico , ou então o Visual Basic assumirá o tamanho de 128 bytes. O comprimento é utilizado para calcular a posição do registro dentro do arquivo. Tudo que você precisa para acessar um registro em particular é especificar o numero do registro.

Abertura do Arquivo Randomico :

**Open** arquivo **For Random As** numero_de_arquivo **Len =** tamanho_de_registro

Leitura de dados :

**Get #**numero_de_arquivo, numero_de_registro, registro

Gravação de Dados :

**Put #**numero_de_arquivo, numero_de_registro, registro

Acesso Sequencial - O acesso sequencial é projetado para uso com arquivos texto. Cada caracter no arquivo representa um caracter do texto ou sequencia de formatação de texto, como por exemplo um salto de linha (*newline*). Arquivos sequenciais permite que você use procedimentos especialmente projetados para escrita e leitura de linhas ou strings de texto. Isto facilita o trabalho com arquivos produzidos por um editor de texto , isto é arquivos com dados que não estão divididos em uma série de registros.

Abertura do Arquivo Sequencial :

**Open** arquivo **For [Input | Output | Append] As** numero_de_arquivo **Len =** tamanho_do_buffer

Leitura de dados :

**Input** #numero_de_arquivo, variável1, variável2

Gravação de Dados :

**Print** #numero_de_arquivo, expressão1, expressão2

Acesso Binário - O acesso binário permite que você use arquivos para guardar dados no formato que você desejar , não são assumidos nenhum tipo de formato de dados ou necessidades de informar o tamanho do registro. Você precisa saber com exatidão onde seus dados foram escritos para poder recupera-los corretamente. Apesar do acesso binário fornecer poucas funções e procedimentos para ajudar na manipulação de dados, ele providencia grande flexibilidade. Por exemplo ele pode conservar espaço em disco pela construção de registros de tamanho variável.

Abertura do Arquivo Binário :

**Open** arquivo **For Binary As** numero_de_arquivo

Leitura de dados :

variável="ABCD"

**Get** #numero_de_arquivo, posição_inicial, variável ' Serão lidos 4 bytes do arquivo

Gravação de Dados :

**Put** #numero_de_arquivo, posição_inicial, variável

Observação : O parâmetro posição inicial é necessária apenas na primeira leitura ou gravação , as outras operações serão efetuadas nas posições posteriores.

## 7 - Uso dos Tipos de Dados Suportados pelo Visual Basic

A declaração de nomes de variáveis não é obrigatória no Visual Basic , mas é recomendada pois com esta precaução , evitam-se erros de digitação e atribuição de valores.

Na criação do nome da variável, devemos seguir as seguintes regras :

.Comece o nome com uma letra.

.O nome deve conter apenas letras, numeros e o caracter *underscore* , caracteres de pontuação e espaços não são permitidos.

.O nome não deve exceder 40 caracteres

.Não podem ser utilizadas palavras reservadas do Visual Basic

Dentro de uma *procedure* , a variável é declarada com a sentença :

**Dim** variável **As** tipo

Tipos fundamentais de váriaveis no Visual Basic :

| Tipo | Descrição | Caracter de declaração do Tipo | Limites |
|---|---|---|---|
| | | | |

| | | | |
|---|---|---|---|
| Integer | Inteiro de 2 bytes | % | -32.768 até 32.767 |
| Long | Inteiro de 4 bytes | & | -2.147.483.648 até 2.147.483.647 |
| Single | Número de ponto flutuante de 4 bytes | ! | -3,402823E38 até -1,401298E-45 e<br><br>1,401298E-45 até 3,402823E38 |
| Double | Número de ponto flutuante de 8 bytes | # | -1,79769313486232D-308 até -4,94065645841247D-324<br><br>4,94065645841247 D324 até 1,79769313486232D-308 |
| Currency | Numero de ponto decimal fixo com 8 bytes | @ | -922337203685477.5808 até 922337203685477.5807 |
| String | String de caracteres | $ | Comprimento de 0 até 65.500 caracteres |
| Variant | Pode conter date/time, numeros de ponto flutuante ou strings | Nenhum | Datas : de 1 de Janeiro de 0000 até 31 de Dezembro de 9999<br><br>Valores numéricos : igual ao tipo Double<br><br>Strings : igual ao tipo String |

Escopo das Variáveis

Quando você declara uma variável dentro de uma procedure , apenas o código contido nesta procedure pode acessar ou trocar o valor desta variável, significando que o escopo ou alcance daquela variável é restrito ou é local aquela procedure. As vezes voce pode necessitar usar a variavel com um alcance maior , de forma que o seu conteúdo esteja disponível para todas as procedures contidas dentro de um form , ou mesmo que ela possa ser vista por todos os módulos em todos os *forms* de uma aplicação. O Visual Basic permite que você especifique o escopo ou alcance de uma variável dependendo da forma que você a declarar.

| Escopo | Declaração da variável |
|---|---|
| Local | Dim , Static ou Redim - declaração dentro da procedure |
| Módulo | Dim - declaração na seção Declarations de um form ou de um módulo de código |
| Global | Global - na seção Declarations de um módulo de código |

Declaração implicita de variáveis :

Você não precisa declarar uma variável antes de usá-la.Por exemplo, você pode escrever uma função como abaixo:

**Function SafeSqr(num)**

**TempVal=Abs(num)**

**SafeSqr=Sqr(tempVal)**

**End Function**

Você não tem que declarar TempVal antes de usá-lo na função. O Visual Basic automáticamente cria uma variável com o seu nome, de forma que você pode usa-lo como se houvesse explicitamente declarado a variável. Ao mesmo tempo que esete procedimento é conveniente , ele tambem pode levar a erros em seu código se voê escrever uma variável de forma errada.

Declaração explicita de variáveis :

Para evitar o problema de variáveis escritas de forma errada , você pode estipular que o Visual Basic sempre gera uma

mensagem de erro quando encontra um nome não préviamente declarado explicitamente como uma variável. Para fazer isto, coloque a sentença abaixo na seção Declarations do form ou do módulo de código :

**Option Explicit**

Declaração de Constantes

Utilizada quando um valor ou uma String repete-se contantemente dentro do código com a finalidade de aumentar a legibilidade do código

Sintaxe :

**[Global] Const** nome_constante = expressão

Exemplos:

Const PI=3.141592654

Global Const MAX_PLANETS=9

Const PI2 = PI * 2

Tipos de variáveis definidos pelo usuário (estruturas)

Você pode combinar variáveis de diversos tipos de forma a criar novos tipos de variáveis. Esta situação assemelha-se ao comando *struct* do C ou ao *record* em Pascal. Você cria tipos definidos pelo usuário (user-defined types) com o uso da sentença **Type** que deve ser colocada na seção de declarações de um módulo de código. Um tipo criado pelo usuário é sempre visto de forma global dentro do código Visual Basic apesar de que as variáveis declaradas com este tipo possam ser globais , locais a funções ou módulos form.

Você pode criar por exemplo , um tipo definido pelo usuário que guarda informações sobre sistemas de computadores.

' Declarations ( de um módulo de código)

Type SystemInfo

CPU as Variant

Memory as Long

VideoColors as Integer

Cost as Currency

PurchaseDate as Variant

End Type

Você pode declarar uma variável global ou local ou a nivel de módulo com o tipo SystemInfo.

Dim MySystemas as SystemInfo , YourSystem as SystemInfo

Você pode assinalar valores de elementos dentro da variável de forma semelhante a que é usada para setar-se propriedades de controles.

MySystem.CPU = "486"

If MySystem.PurchaseDate > #1/1/92# then

**8 - Codificando em Visual Basic**

Comentários: '

Numeros : Decimal - 9 , Octal &O11 , Hexadecimal &H9

Sentenças do Visual Basic são normalmente colocadas apenas uma por linha e não tem nenhum terminador de linha. Apesar disto você pode colocar mais de uma sentença em uma linha se você colocar dois pontos ":" entre elas

Text1.Text = "HEllo" : Red=255 : Text1.Backcolor = Red

Nomes de **Sub** ou **Functions** seguem as regras abaixo

.Comece o nome com uma letra.

.O nome deve conter apenas letras, numeros e o caracter *underscore* , caracteres de pontuação e espaços não são permitidos.

.O nome não deve exceder 40 caracteres

.Não podem ser utilizadas palavras reservadas do Visual Basic

Palavras reservadas

Consulte o Help do Visual Basic em *Programming Language*

Dando valores a propriedades ou variaveis

Destino = Origem ' Este formato é usado para assinalar o valor de uma variavel , pegar o valor de uma propriedade de um controle ou guardar/pegar o valor de uma variavel.

Setando o valor de uma propriedade :

Text1.Text = "Seu nome aqui"

Text1.BackColor = 0

Pegando o valor de uma propriedade :

VariavelString = Text1.Text

Pegando ou assinalando o valor de propriedades de controles existentes em outros forms:

Form2!text1.Text = "Seu nome Aqui"

VariavelString=Form2!Text1.text

**9 - Loops e Condicionais**

As sentenças que controlam decisões e loops no Visual Basic são chamados de estruturas de controles. As mais utilizadas estruturas são :

- Blocos If_Then

Utilizados para executar **uma** ou mais sentenças de forma condicional.Você pode usar a sintaxe de uma linha única ou a sintaxe de bloco multilinhas:

**If** condição **Then** comando

**If** condição **Then**

comandos

**End If**

As condições são comparações mas podem ser qualquer expressão que ao final resulte em valores numéricos. Visual Basic interpreta estes valores como True (verdadeiro) ou False (falso). Um valor zero é considerado **False** e um valor não zero é considerado **True.** Se a condição é verdadeira o Visual Basic executa todos os comandos depois da palavra **Then.**

**If** Anydate **<** Now **Then** Anydate=Now

**If** Anydate **<** Now **then**

Anydate=Now

**End If**

- Blocos If_Then_Else

Esta estrutura é usada para controlar diversos blocos de comando, onde apenas um deles será executado.

**If** condição1 **Then**

comandos-1

**ElseIf** condição2 **Then**

comandos-2

**Else**

comandos-3

**End If**

- Sentenças Select Case

O Visual Basic providencia a estrutura Slect Case como uma alternativa a estrutura If_Then_ElseIf para seletivamente

executar um bloco de comandos dentro de multiplos blocos de comandos. O Select_Case torna o código escrito mais eficiente e fácil de ler.

A estrutura Select_Case funciona com uma simples expressão de teste que é avaliada no topo da estrutura. O resultado é então comparado com os valores para cada **Case** dentro da estrutura. Se houver uma coincidência , será executado o bloco de comandos associado com aquele **Case :**

**Select Case** expressão_de_teste

**Case** item_de_teste1

bloco_de_comandos1

**Case** item_de_teste2

bloco_de_comandos2

**Case** item_de_teste3

bloco_de_comandos3

**Case** item_de_teste4

bloco_de_comandos4

**Case Else**

bloco_de_comandos5

**End Select**

Cada item de teste é uma lista de um ou mais valores. Se houver mais de uma valor , eles estarão separados por virgulas. Cada bloco de comandos comtêm um , mais de um comando ou nenhum comando. Se mais de um **Case** coincide com a expressão testada, apenas o bloco de comandos associado com o primeiro **Case** coincidente será executado. O Visual Basic executará comandos no bloco **Case Else** (que é opcional) se nenhum dos valores na lista de itens não coincidir com a expressão de teste.

- Sentenças Do ... Loop

Use o Do ... Loop para executar um bloco de comandos por um numero indefinido de vezes. Existem variações da sentença Do... , mais cada uma avalia uma condição numérica para determinar quando continuar a execução. Assim como acontece com o If_Then a condição precisa resultar em um valor numérico que possa ser traduzido como True(não zero) ou False (zero).

O Do ... Loop abaixo é executado enquanto a condição for **True**:

**Do While** condição

comandos

**Loop**

Outra variação do Do...Loop executa primeiro os comandos e depois testa a condição. Esta variação garante pelo menos uma vez a execução do bloco de comandos :

**Do**

comandos

**Loop While** condição

As duas variações abaixo são analogas as anteriores , com a exceção de que elas ficam em loop enquanto a condição de teste for **False**

| Executa zero ou mais vezes | Executa pelo menos uma vez |
|---|---|
| Do Until condição<br><br>comandos<br><br>Loop | Do<br><br>comandos<br><br>Loop Until condição |

Observer que **Do Until** *condição* é equivalente a **Do While Not** *condição*.

- For ... Next

Esta é uma estrutura de repetição utilizada quando se sabe préviamente o numero vezes que um bloco de codigos sera executado. O For...Next utiliza um contador que é incrementado ou decrementado durante cada repetição do bloco de comandos. Sua sintaxe é :

**For** *contador* **=** *início* **To** *final* [ **Step** *incremento* ]

*comandos*

**Next** [ *contador* ]

Os argumentos *contador , inicio , final* e *incremento* são todos numéricos. O argumento *incremento* pode ser positivo ou negativo . Se *incremento* é positivo, *inicio* deve ser menor ou igual a *final* ou os comandos nos limites do For...Next não serão executados. Se *incremento* é negativo, *inicio* deve ser maior ou igual a *final* , para que os comandos sejam executados. Se o *incremento* não for informado então ele terá seu valor assumido como 1.

- Saindo de estruturas de controle

O comando **Exit** permite que você saia diretamente de um bloco For..Next, Do...Loop, Sub procedure, ou Function procedure. Sintáticamente, o comando **Exit** é simples:

**Exit For** pode aparecer quantas vezes você precisar dentro de um bloco **For** e **Exit Do** pode aparecer o numero de vezes que for necessário dentro de um bloco **Do :**

**For** *contador* **=** *início* **To** *final* [ **Step** *incremento* ]

*comandos*

**Exit For**

*comandos*

**Next** [ *contador* ]

**Do While** *condição*

*comandos*

**Exit Do**

*comandos*

**Loop**

- Saindo de procedures **Sub** ou **Functions**

**Exit Sub** e **Exit Function** é útil quando a procedure tiver feito tudo que necessita fazer e pode encerrar a execução imediatamente.

- Comando **Go To**

Faz com que o programa vá incondicionalmente para uma linha especificada dentro de uma procedure. Sintaxe :

**Go To** { rótulo | número de linha }

Rótulo marca a linha que deve ser executada em seguida . O rótulo deve começar com um caracter alfabético e encerrar com dois pontos ":". Cada rótulo precisa ser único dentro da procedure.

Número de linha que deve ser executada em seguida. O número de linha pode ser qualquer numero com até 40 caracteres , deve conter só numeros e não deve terminar com dois pontos ":"

## 10 - Depuração de Código

O Visual Basic não pode diagnosticar ou consertar erros para você, mas fornece ferramentas que o ajudarão a analizar. As ferramentas de depuração incluem passo a passo em linhas na execução do programa , breakpoints , break em expressões , pasos a passo em procedures e visualização de variáveis e propriedades. Visual Basic tambem inclui facilidades especiais de depuração tais como editar-e-continuar , alterando a proxima sentença a ser executada e testando o procedimento enquanto a aplicação está parada.

Durante a digitação do código o Visual Basic faz uma checagem prévia da sintaxe assinalando os erros em relação as palavras chaves da linguagem. Isso só acontecerá se estiver ativada a opção **Syntax Checking** para **Yes** em **Options , Environment** no menu do Visual Basic.

- Ferramentas de depuração no ToolBar

| | |
|---|---|
| Ativa um breakpoint (ponto de parada) na linha corrente | Comando *Toggle Breakpoint* no menu *Debug* |
| Mostra o valor da seleção corrente na janela Code | Comando Instant Watch no menu Debug |
| Mostra a estrutura das chamadas ativas | Comando *Calls* no menu *Debug* |
| Executa uma linha de código de cada vez (passo a passo) | Comando *Single Step* no menu *Debug* |
| Executa um procedimento passo a passo sem passar pelas subrotinas | Comando *Procedure Step* no menu *Debug* |

- Identificando o modo corrente

Barra de títulos quando o Visual Basic está no modo de projeto (*design*)

Barra de títulos quando o Visual Basic está no modo de execução (*run*)

Barra de títulos quando o Visual Basic está no modo parada (*break*)

Note que o modo corrente tambem determina quais ferramentas de depuração estão disponíveis , as ferramentas não disponíveis aparecem com o as cores do botão com a intensidade reduzida.

- Entrando no Break Mode

Automáticamente - Você entra no break mode automáticamente quando ocorre uma das situações abaixo :

1 - Um comando gera um erro de execução não tratado por rotinas internas

2 - Durante a execução do programa é atingida uma linha que contem um breakpoint

3 - Durante a execução do programa é encontrada o comando **Stop**

4 - Uma expressão de break definida na caixa de dialogo Add Watch mudou de valor ou tornou-se verdadeira True , dependendo de como você definiu

Manualmente - Você ativa o break mode manualmente quando se você executar uma das ações abaixo enquanto a aplicação estiver executando :

1 - Pressionar CTRL+BREAK

2 - Escolher a opção **Break** no menu **Run**

3 - Clicar no botão **Break** no **ToolBar**

## 11 - Impressão em Forms , Picture Box e Impressoras

O comando Print é o comando principal para apresentação de saídas de programas. A saída é escolhida colocando-se o objeto na qual deseja-se a impressão antes do comando Print.

Sintaxe :

[objeto].**Print** [lista_de_expressões] { ; | ,}

O objeto é opcional , se for omitido o Visual Basic assume que a impressão será no form.

Exemplos :

Impressão em um Form chamado MeuForm :

MeuForm**.Print** "Este é o meu form"

Impressão em um Picture Box chamado MeuPictureBox :

MeuPictureBox**.Print** "Este é um picture box"

Impressão no form corrente :

**Print** "Este é o form corrente"

Impressão na impressora :

**Printer.Print** "Este texto vai para a impressora"

Use o ponto e virgula ";" e a virgula "," para separar os itens a serem impressos dentro da linha de impressão. O ";" faz com que os itens sejam impressos um após o outro , a "," faz com que a impressão seja feita saltando-se as tabulações. Exemplos :

x=2 : y=7

Print "O valor de X é "; x; " e o valor de Y é "; Y

Por default , cada vez que o comando Print é utilizado , o texto desejado é impresso e salta-se para a próxima linha , se não houverem dados a serem impressos , o comando Print fará que se salte uma linha em branco , se for colocado um ";" no final da linha de comando , o comando Print não saltará linhas :

Print "Isto tudo aparece " ;

Print "na mesma linha"

A instrução **Format$/Format** é utilizada converte valores numéricos em strings de forma que você tenha controle sobre a forma com que os numeros serão impressos/visualizados . O comando Format$ transforma os valores numéricos em strings e o Format transforma em tipo Variant.

Sintaxe :

Format[$] ( expressão_numérica [,formato$] )

Expressão numérica especifica o numero a ser convertido e fmt$ é a string formada de simbolos que irão formatar o numero. O simbolos mais usados eestão abaixo :

| Simbolo | Descrição |
|---|---|
| 0 | Caracter posicionador ; imprime um zero anterior ou postrior ao numero na posiçao se apropriado |
| # | Caracter posicionador ; nunca imprime zeros anteriores ou posteriores |
| . | Separador de decimais |
| , | Separador de milhares |
| - + $ ( ) espaço | Caracteres literais ; mostra cada um desses caracteres exatamente como digitado dentro da string de formatação |

Exemplos :

Format$(83514.4 , "00000.00") resulta em 08315.40

Format$(83514.4 , "#####.##") resulta em 8315.4

Format$(83514.4 , "##,##0.00") resulta em 8,315.40

Format$(315.4 , "$##0.00") resulta em $315.40

Format$(Now , "d/m/yy") resulta em 12/09/95

Note que a representação da separação de milhares e decimais deverá no programa ser colocada na forma utilizada nos Estados Unidos , mas o Windows mostrará na tela os caracteres configurados no Painel de Controle , opção Internacional.

Imprimindo com o objeto Printer.

| Comando | Ação |
|---|---|
| **Printer.Print "texto"** | Enviar os dados para a impressora |
| **Printer.NewPage** | Mudar de página |
| **Printer.EndDoc** | Iniciar a impressão |
| **Printer.CurrentX = 0** | Seta a coordenada horizontal de impressão |
| **Printer.CurrentY = 0** | Seta a coordenada vertical de impressão |
| **Printer.ScaleMode** | Define a unidade a ser usada para posicionamento da impressão |
| **Printer.Print Spc(10);"Texto"** | Pula 10 espaços antes de imprimir o texto |
| **Printer.Print Tab(40);"Texto"** | Posiciona na coluna 40 antes de imprimir o texto |

## 12 - Uso do Controle de Acesso a Dados

Com o **Data Control** você pode criar aplicações que mostram , editam e atualizam informações a partir de diversos tipos de banco de dado. Primeiro você adiciona ao form o **Data Control** e especifica o banco de dados a utilizar. Após você adiciona os controles tais como text boxes ao form , seta as propriedades de ligação ao banco de dados , através dos quais será feito o acesso ao banco de dados.Quando você executar o programa , este controles estarão ligados ao banco de dados e automáticamente irão mostrar os dados.

O Visual Basic faz acesso aos bancos de dados , através do mesmo mecanismo de acesso implementado no Microsoft Acess que encontra-se embutido no Visual Basic.

As propriedades principais do Data Control são :

| Propriedade | Determina | Observação |
|---|---|---|
| Connect | O tipo de banco de dados | Não é necessário para banco de dados Access |
| DatabaseName | O nome da fonte dos dados. Identifica a localização do arquivo de banco de dados | Seta o path e nome de arquivo para o arquivo de banco de dados |
| Exclusive | Acesso Monousuário ou Multiusuário ao banco de dados | True (monousuário); False (multiusuário) . Default False |
| ReadOnly | Acesso de leitura e escrita ao banco de dados | True(só leitura) ou False (leitura/gravação). Defaul é False |
| RecordSource | O nome da tabela do banco de dados ou texto de um pergunta SQL | Tem que ser um nome de tabela válido dentro do database especificado ou um SQL query válido |

A propriedade Connect do Data Control pode assumir os seguintes valores :

| Database Format | DatabaseName | Connect |
|---|---|---|
| Access | drive:\path\arquivo.MDB | não é necessário |
| FoxPro versão 2.0 | drive:\path\ | FoxPro 2.0; |
| FoxPro versão 2.5 | drive:\path\ | FoxPro 2.5; |
| DBASE III | drive:\path\ | dbase III; |
| DBASE IV | drive:\path\ | dbase IV; |
| Paradox | drive:\path\ | paradox;pwd=password |
| Btrieve | drive:\path\.arquivo.DDF | btrieve; |

Usando Controles Ligados.

Os controles Check Box , Image , Label , Picture Box, Text Box , etc. são ligados ao Data Control através das propriedades abaixo :

| Propriedade | Descrição |
|---|---|
| DataChanged | Indica quando o valor mostrado em um controle foi alterado |
| DataField | Especifica o nome do campo no conjunto de registros criado pelo DataControl |
| DataSource | Especifica o Nome do DataControl ao qual este controle está ligado |