INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
Departamento de Engenharia Informática

Tutorial de Visual Basic

Projecto realizado sob a orientação do
Eng. Carlos Vaz de Carvalho

António Alexandre Sousa Gouveia

Porto, Setembro de 2000

Agradecimentos

Agradeço ao meu orientador, Eng. Carlos Vaz de Carvalho pela sua colaboração, disponibilidade e orientação ao longo de todo o trabalho.

Ao Instituto Superior de Engenharia do Porto pela disponibilização dos recursos e instalações.

A todos quantos de alguma forma contribuíram para a realização deste projecto.

ÍNDICE

1

Introdução

Este trabalho, tal como é apresentado neste documento, foi realizado no âmbito da cadeira Projecto da Licenciatura em Engenharia Informática – ramo Computadores e Sistemas, do Instituto Superior de Engenharia do Porto.

O objectivo do trabalho foi o de fazer um estudo comparativo de alguns tutoriais disponíveis na Internet sobre programação em Visual Basic e implementar um tutorial que se baseasse na metodologia de ensino Learning by Example.

O facto do tutorial proposto ser baseado na metodologia de ensino Learning by Example significa que os alunos irão aprender através de exemplos explicados em pormenor e cuja complexidade aumentará progressivamente. Estes exemplos serão implementados pelos alunos à medida que eles forem evoluindo na aprendizagem.

Este trabalho foi aproveitado no contexto do Projecto WeMeet, projecto europeu apresentado no capítulo seguinte, no qual o DEI-ISEP participou pela segunda vez.

Do Projecto WeMeet fizeram parte vários sub-projectos, sendo que um deles tinha como objectivo iniciar os alunos de Engenharia (Química, Civil, Mecânica, ...) à Programação em Visual Basic. Este sub-projecto, intitulado "Programming for Engineers", teve como responsável o Eng. Carlos Vaz de Carvalho.

No capítulo 2 apresenta-se o Projecto WeMeet e as ideias que servem de base ao seu funcionamento.

No capítulo 3 faz-se o estudo de alguns tutoriais disponíveis na Internet e apresenta-se uma análise comparativa desses tutoriais.

No capítulo 4 apresenta-se o tutorial proposto.

No capítulo 5 tiram-se conclusões sobre o trabalho realizado e perspectivam-se desenvolvimentos futuros.

2

O Projecto WeMeet

O projecto **WeMeet** – **W**eb **E**xchange **M**ethodology for **E**ducation **E**ngineers by **T**elematics visa implementar acções cooperativas de auto-formação, promovendo a participação de alunos das Instituições participantes na elaboração de projectos de engenharia, mini-cursos multimédia, sínteses bibliográficas, etc... Toda a actividade dos alunos e respectivos tutores será suportada por metodologias de Ensino Aberto e a Distância.

Deste modo, o projecto procura satisfazer os objectivos do programa SOCRATES:

- A dimensão cooperativa de auto-aprendizagem proposta permite agrupar alunos de instituições distantes;

- Alunos e professores participantes estarão melhor informados sobre as potencialidade e ferramentas de Ensino a Distância. O trabalho de cada aluno será complementado com uma reflexão conjunta sobre estes temas;

- Como os resultados obtidos serão publicados na Internet, muitos outros alunos e professores terão acesso a este material educativo. Tornando-se uma referência de consulta, o *site* permitirá a outros professores tomarem conhecimento das potencialidades das tecnologias multimédia e das tecnologias de informação e comunicação.

Acções concretas do projecto:

- Implementação de acções de aprendizagem colaborativa entre Instituições de Ensino Superior. O projecto planeia criar uma rede transnacional inter-instituições para promover o intercâmbio científico entre os respectivos alunos, utilizando tecnologias de informação e comunicação.

- Desenvolvimento de uma abordagem original para o Ensino a Distância, juntamente com práticas educativas tradicionais. O ênfase é colocado na importância do trabalho cooperativo remoto e partilha de conhecimento, onde o Ensino tradicional prevalece. Esta nova abordagem tem uma estratégia mista.

- Desenvolvimento de uma metodologia para implementar esta forma particular de Ensino Aberto e a Distância. Neste contexto, o projecto visa desenvolver métodos e estruturas pedagógicas e ambientes organizacionais adaptados. Atenção particular será dedicada à promoção desta metodologia.

- Teste de qualidade e "amigabilidade" de ferramentas de EAD. Pretende-se ainda promover a utilização e domínio destas ferramentas por alunos e professores.

Sub-projectos

No decorrer do projecto **WeMeet** foram constituídos diversos grupos transnacionais de alunos, correspondendo a diversos sub-projectos. Da responsabilidade directa do DEI-ISEP constaram:

- Java Learning (responsável: Filipe Pacheco)
- Java Learning 2 (responsável: Filipe Pacheco)
- Programming for Engineers (responsável: Carlos Vaz de Carvalho)

O DEI-ISEP participou ainda nos projectos:

- FieldBus (responsável: Luís Lino Ferreira)
- Context Oriented Search Engines (responsável: Carlos Vaz de Carvalho)
- OLE Programming (responsável: Luís Lino Ferreira)

Apresentação de resultados

A apresentação final de resultados, na qual o autor deste trabalho participou, teve lugar no Institut Supérieur Industriel de Bruxelles, na Bélgica, no passado mês de Julho.

3

Estudo de Tutoriais

Com vista a desenvolver um estudo comparativo de alguns tutoriais sobre o ambiente e linguagem de programação do Visual Basic, foi realizada uma pesquisa na Internet que resultou em vários tutoriais. Aqui ficam os endereços:

- Tutorial I - http://members.tripod.com/~vkliew/vbtutor.html
- Tutorial II - http://vbtutorial.programmer.webjump.com
- Tutorial III - http://vb-world.net/beginning/vbtutorial
- Tutorial IV - http://www.ms-vb.com
- Tutorial V - http://www.dotcom2001.com/trainingvb
- Tutorial VI - http://lockledge.eng.wayne.edu/be101/tutorial/main.html
- Tutorial VII - http://www.xploiter.com/programming/vb/vb5_tutor
- Tutorial VIII - http://www.iessoft.com/scripts/beginner.asp
- Tutorial IX - http://www.dcs.napier.ac.uk/hci/vb50
- Tutorial X - http://www.vbinformation.com/tutor.htm
- Tutorial XI - http://klingon.cs.iupui.edu/~aharris/220vb/220s97.html
- Tutorial XII - http://www.cyber-matrix.com/lbasic.htm
- Tutorial XIII - http://www.geocities.com/~chuckeasttom/vb/vb.htm

Apesar do grande número de tutoriais encontrados, o certo é que poucos satisfazem os requisitos básicos para que o seu estudo possa ser considerado adequado a este trabalho.

Os requisitos básicos considerados necessários foram os seguintes:

- o tutorial deve ser de acesso público;
- deve abordar algoritmia;
- e deve ser orientado a alunos com pouca ou nenhuma experiência em Visual Basic

Sendo assim foram estudados os 3 primeiros tutoriais.

De acordo com as características de cada um destes tutoriais serão feitos alguns comentários sobre a forma como se encontram estruturados e apresentam a informação, realçando alguns aspectos que do ponto de vista do autor deste trabalho poderiam ter sido melhorados.

3.1 TUTORIAL I - http://members.tripod.com/~vkliew/vbtutor.html

Este tutorial encontra-se dividido em 8 partes:

1. O ambiente de desenvolvimento do Visual Basic
2. Construção de uma aplicação VB
3. Desenho da interface
4. Trabalhar com controlos
5. Lógica na programação em Visual Basic
6. Variáveis
7. Manipulação de variáveis (indisponível – link errado)
8. Métodos e propriedades (indisponível – link errado)

O autor do tutorial parte do princípio que o aluno já possui alguns conhecimentos básicos de programação em Basic.

1ª Parte - O ambiente de desenvolvimento do Visual Basic

Na primeira parte começa-se por comparar a execução sequencial dos programas com a programação por eventos. Depois explica-se o que é um projecto através dos seus módulos e apresentam-se as partes que compõem o ambiente de desenvolvimento (*toolbox*, *properties window*, ...) mas sem qualquer descrição a complementá-las. No final desta parte é proposto ao aluno que teste um pequeno e muito simples exemplo sendo para isso fornecidos alguns dos passos necessários e respectivo código. Não explica como se corre o programa.

2ª Parte - Construção de uma aplicação VB

A parte número 2 começa por referir os passos necessários à construção de uma aplicação em Visual Basic: desenhar a interface; definir as propriedades; e escrever o código.

O resto desta parte resume-se à apresentação de um exemplo que calcula o volume de um cilindro. Antes de tudo é apresentado o *form* com os controlos necessários e as propriedades já definidas, seguindo-se uma descrição das propriedades cujo valor é necessário alterar. Note-se que não é feita qualquer explicação de como se inserem os controlos no *form* nem para que servem os diferentes controlos usados no exemplo.

A seguir pede-se ao aluno que faça *duplo-click* sobre o botão e escreva o código necessário ao cálculo do volume de um cilindro e apresentação do resultado no ecrã. Este código é fornecido ao aluno.

Depois de explicar como usar o programa o autor resolve descrever o código usado através de pseudo-código. Segue-se a apresentação da sintaxe *Objecto.Propriedade* usando para o efeito uma *text-box* do exemplo e a propriedade *Text*.

Por fim é explicada a sintaxe de declaração de um procedimento de evento. Para isso recorreu-se à declaração do procedimento de evento *Click* do botão usado no exemplo: *"Private Sub OK_Click()"*. É explicado o conceito de *Private* e o facto de o nome do procedimento de evento estar de acordo com a acção do utilizador.

3ª Parte - Desenho da interface

Esta parte gira em volta do desenho da interface de uma máquina de calcular. O objectivo, implícito, consiste em através da apresentação do *form* da calculadora, o aluno ser capaz de construir uma réplica dessa calculadora.

Começa-se por, mais uma vez, fazer referência a alguns elementos do ambiente de trabalho do Visual Basic, sendo de seguida apresentado um conjunto de passos a seguir pelo aluno no desenho da interface.

Alguns destes passos são de carácter específico como por exemplo quando se pede para alterar a propriedade *BorderStyle* do *form* para *"Fixed Single"* ou quando se pede para apagar o valor da propriedade *Caption* do mostrador (um *label*).

Outros são de caracter mais geral como por exemplo quando se diz para redimensionar o *form* até se obter um tamanho satisfatório, ou se diz apenas para desenhar os botões no *form*.

4ª Parte - Trabalhar com controlos

Nesta parte são apenas apresentados alguns pontos importantes na definição das propriedades dos controlos. São eles:

- A propriedade *Caption* deve ser clara para não criar nenhuma situação de ambiguidade ao utilizador. É usado o exemplo da máquina de calcular para explicar este ponto.

- A propriedade *Name* deve ter um nome com o significado de acordo com a função do controlo.

- As propriedades *Visible* e *Enabled* são importantes na medida em que tornam o controlo visível ou invisível e disponível ou não.

5ª Parte - Lógica na programação em Visual Basic

Na parte 5 é apresentado mais um exemplo, desta vez com o intuito de explicar algumas operações aritméticas básicas.

Começa-se então por apresentar a interface do programa a implementar e por indicar quais as propriedades e valores a alterar. Neste exemplo as propriedades a alterar são sempre o *Name* e o *Caption* e isto acontece apenas em alguns controlos.

De seguida é apresentado todo o código do programa havendo uma pequena parte opcional e que é imediatamente explicada.

Depois da apresentação do código é feita uma descrição do funcionamento do programa. Após premir o botão respectivo são apresentados dois números calculados aleatoriamente. O objectivo é o utilizador introduzir o resultado da adição desses dois números para depois o programa indicar se o resultado está correcto ou errado.

No código do programa podem-se encontrar duas funções, *Rnd* e *Int*, que são explicadas através de um pequeno exemplo. Existem ainda as funções *Str$* e *Val* que são igualmente apresentadas.

Na validação do resultado é usada a estrutura de decisão *If ... Then ... Else* que no entanto o autor considera não ser importante explicar.

No final desta parte o aluno é convidado a testar o programa.

6ª Parte - Variáveis

Esta é a última parte disponível neste tutorial já que os *links* para as partes 7 e 8 não estão correctos.

Nesta parte começa-se por fazer uma breve descrição do que são variáveis por analogia às caixas de correio. De seguida enumeram-se algumas regras para nomes de variáveis e por fim apresentam-se os tipos de variável mais usados.

Nas regras para os nomes de variáveis pode-se ler que uma variável não pode ter mais de 255 caracteres, ou que não permite espaços nem pontos. São ainda apresentados alguns exemplos de nomes válidos e inválidos para variáveis.

São referidos 8 tipos de variável sempre com uma pequena descrição sobre que tipo de dados podem armazenar. Note-se que não é apresentada a sintaxe usada para declarar variáveis.

3.2 TUTORIAL II - http://vbtutorial.programmer.webjump.com

Este tutorial está dividido em:

1. Introdução
2. Estrutura da linguagem
3. Sub-rotinas, funções e módulos
4. ActiveX
5. Bases de dados

Uma vez que as partes 4 e 5 saem do âmbito deste estudo vamos apenas analisas as três primeiras.

1ª Parte - Introdução

Na introdução o autor começa por dar algumas dicas sobre como organizar a interface do próprio Visual Basic, como por exemplo deixar a *Toolbox* – Caixa de Ferramentas no lado esquerdo do ecrã, ou ainda redimensionar o *Project Explorer Window* – Janela de Projecto de forma a ocupar apenas o espaço necessário para mostrar 5 ou 6 linhas de itens.

De seguida são apresentados os conceitos *IntelliSense* e *AutoComplete*. O primeiro consiste em mostrar uma lista de eventos, métodos e propriedades sempre que escrever o nome de um objecto seguido de um ponto. O segundo consiste em formatar o código de uma linha (capitalizar, eliminar espaços duplos, detectar erros, etc) sempre que se muda para outra linha.

Após explicar estes conceitos é feita uma comparação entre as versões 4, 5 e 6 do Visual Basic onde o autor dá a sua opinião pessoal. É feita ainda uma comparação entre o Microsoft Visual Basic e o Inprise Delphi 4 onde se apresentam vantagens e desvantagens de ambos os ambientes de desenvolvimento.

Por fim o autor toma a liberdade de recomendar alguns livros.

2ª Parte - Estrutura da linguagem

O objectivo desta secção é o de ensinar a escrever código em Visual Basic. O autor parte do princípio que o aluno já possui alguma experiência em programação.

A primeira parte é muito importante para quem não está familiarizado com a programação orientada aos objectos uma vez que, com o auxilio de um pequeno exemplo, se explica o que é um evento e como os eventos são "disparados". É ainda explicado o conceito *"event driven"*.

O exemplo consiste em desenhar um botão no *form* para que ao ser premido faça surgir uma mensagem com o texto "Hi" (olá), isto durante a execução do programa.

Ainda com base neste exemplo são explicados de uma forma muito resumida os componentes da *Code Editor Window* que é a janela onde se escreve o código do programa.

De seguida é apresentada a estrutura de decisão *If ... Then* através de um pequeno exemplo, sendo feita a distinção entre a sintaxe de instrução única e a sintaxe de múltiplas instruções. Note-se que não é feita qualquer referência à estrutura If ... Then ... Else, o que não deixa de ser um ponto fraco neste tutorial.

Depois da estrutura de decisão *If ... Then* é a vez de outra estrutura de decisão ser apresentada: a estrutura *Case*. É feita uma analogia entre a estrutura *If ...Then* e a estrutura *Case* uma vez que se pode considerar a estrutura *Case* como sendo um conjunto de declarações *If*. É apresentado um exemplo de uma estrutura de decisão *Case*.

A seguir são apresentadas duas estruturas de controlo de fluxo: *For* e *Do*. Em ambos os casos são usados exemplos que complementam a breve apresentação destas estruturas. De notar que no caso da estrutura *For*, não é dada qualquer explicação sobre como o ciclo é executado, como funciona o contador, ou qual é a condição de paragem. Em relação à estrutura *Do* também não se faz qualquer referência a estes aspectos, nem se mostra as diferentes formas de usar esta estrutura.

Segue-se a simples apresentação dos tipos de variável existentes, sem qualquer descrição mas com alguns exemplos de declaração.

Por fim explica-se como inserir comentários no código: usando o apóstrofo ou a instrução *Rem*.

3ª Parte - Sub-rotinas, funções e módulos

Nesta secção do tutorial aborda-se a utilidade do uso de funções e sub-rotinas.

Começa-se por fazer a distinção entre *"event procedures"* – procedimentos de evento – e os outros procedimentos, explicando em que parte do código eles aparecem (*"general procedures"*). Depois explica-se o porquê de usar sub-rotinas através do exemplo de um procedimento que trata de ajustar a dimensão de uma *text-box* quando o *form* é carregado ou redimensionado. Este exemplo é resumidamente explicado.

Quanto às funções é referida a semelhança entre elas e os procedimentos com a diferença de retornarem um valor. Também para as funções se apresenta um exemplo que desta vez realça as limitações das sub-rotinas e vantagens das funções.

Depois de falar das funções e sub-rotinas é dada uma explicação sobre o que são e como se criam módulos. Isto porque, tal como o autor refere, pode haver a necessidade de criar funções ou sub-rotinas para serem usadas em *forms* diferentes e nesse caso têm que ser colocadas num módulo.

Por fim são apresentados dois exemplos que mostram como se podem criar funções que não sejam específicas a determinado objecto mas sim que através da passagem de parâmetros possam ser usadas para objectos diferentes.

3.3 TUTORIAL III - http://vb-world.net/beginning/vbtutorial

Este tutorial foi desenvolvido por Karl Moore, jornalista no Reino Unido, e encontra-se dividido em 6 partes:

1. Introdução ao Visual Basic
2. Objectos e propriedades
3. Variáveis e estruturas de decisão
4. Ciclos e sub-rotinas
5. Funções e revisão às propriedades, métodos e eventos
6. Perguntas mais frequentes

Todas as partes começam com uma introdução e terminam com uma conclusão onde se resume o que foi abordado nessa parte e se faz uma antevisão do que será tratado na parte seguinte.

1ª Parte - Introdução ao Visual Basic

A primeira parte começa com uma introdução na qual o autor se apresenta e assegura que não é necessário qualquer conhecimento prévio da linguagem Visual Basic para seguir o tutorial. O autor continua a introdução dizendo que irá usar termos simples e explicar alguns conceitos com analogias ao mundo real.

A seguir à introdução é feita uma breve descrição das origens do Visual Basic, como apareceu, como se tornou na linguagem de programação mais popular do mundo e que versões existem.

O passo seguinte consiste em explicar como abrir o Visual Basic a partir do Windows, passo a passo, para que o aluno possa criar o seu primeiro programa.

Chega-se assim ao primeiro exemplo deste tutorial. O autor começa por apresentar a *Toolbox* – Caixa de Ferramentas pedindo de imediato ao aluno que insira uma *label* no *form* e explicando a maneira de o fazer. Depois apresenta a Janela de Propriedades e pede ao aluno que altere a propriedade *Caption* da *label* para um valor indicado. Ainda com base na *label* que foi inserida no *form* explica-se como mover um controlo de um sítio para o outro convidando o aluno a testar o que aprendeu.

A seguir pede-se ao aluno que insira mais um controlo, desta vez um botão, que altere as propriedades *Caption* e *Font* para outros valores e que o redimensione.

Para terminar a criação do primeiro programa o aluno é convidado a escrever uma linha de código associada ao evento *Click* do botão, aproveitando-se para explicar o significado deste evento e em que situação o código inserido irá ser executado. Estes passos são todos explicados em pormenor.

Agora só falta mesmo correr e testar o programa criado. No tutorial segue-se então a explicação de como fazer correr um programa, pedindo-se ao utilizador que prima o botão inserido no *form* para assim ser executado o código do evento respectivo – o evento *Click*.

2ª Parte - Objectos e propriedades

Nesta parte do tutorial são apresentados alguns controlos que podem ser inseridos num *form* e é explicado como alterar as propriedades dos controlos durante a execução do programa.

Começa-se com a apresentação do controlo *check-box* sendo o aluno convidado a abrir o Visual Basic e inserir um controlo deste tipo no form. A seguir pede-se ao aluno que altere algumas propriedades aleatoriamente para ver a forma como elas afectam a *check-box*. É ainda apresentado um exemplo de um *form* com duas *check-box* de aparência bastante diferente entre si.

A seguir são apresentados mais alguns controlos com uma pequena descrição sobre a utilidade de cada um. São então apresentados os controlos: *Image*, *Label*, *TextBox*, *CommandButton*, *DriveListBox* e *Shape*. Segue-se um exemplo de um *form* no qual se despejaram estes controlos apenas para complementar graficamente a explicação, uma vez que o *form* não tem qualquer significado.

Depois de apresentados os controlos o autor lembra o facto de todos eles terem uma propriedade que os identifica, a propriedade *Name*. É referido que apesar de quando inserido no *form* o controlo ter um nome por defeito é importante atribuir-lhe um nome de acordo com a sua função. Ainda em relação aos nomes são apresentadas as *"naming conventions"* que são convenções pré-estabelecidas sobre os nomes a dar aos controlos e que consistem em incluir no início do nome uma abreviatura que identifica o tipo de controlo.

A segunda parte continua com a distinção entre *"design-time"* e *"runtime"*. Em *"design-time"* as propriedades podem ser directamente alteradas na janela de propriedades mas como se altera uma propriedade se o programa estiver a correr? O autor explica como o fazer com base num exemplo cuja implementação é explicada em pormenor.

O exemplo consiste em ter um *form* com dois botões que servem para alterar o titulo da janela durante a execução do programa. Pede-se por isso ao aluno que insira os dois botões no *form* e que escreva o código indicado nos eventos *Click* de cada botão. Este código resume-se a uma só linha que afecta a propriedade *Caption* do *form* com um determinado texto, diferente para cada botão.

Nesta altura o autor aproveita para explicar a sintaxe *Objecto.Propriedade=Valor* e encoraja o aluno a alterar outras propriedades por código apresentando algumas linhas de exemplos.

3ª Parte - Variáveis e estruturas de decisão

A terceira parte deste tutorial define o que são variáveis, apresenta o editor de código do VB e faz uma breve abordagem às estruturas de decisão *If ... Then* e *If ... Then … Else*.

Esta parte começa com a definição de variável fazendo uma analogia com caixas de texto invisíveis e dá-se um pequeno exemplo em que se demonstra a necessidade do uso de variáveis. Este pequeno exemplo servirá de base para o exemplo a implementar pelo aluno mas antes disso o autor decide apresentar a *Code Editor Window* que é a janela onde se escreve o código.

Para apresentar o editor de código começa-se por explicar o que representam as duas caixas que aparecem no cimo da janela – lista de objectos e lista de procedimentos (que

podem ou não ser eventos). Depois relacionam-se estas duas listas para se explicar onde se escreve o código para um determinado evento de um dado objecto.

Ainda na apresentação do editor de código faz-se referência à secção *"General"* que se encontra na lista de objectos e que não é mais do que o local onde se declaram variáveis.

A seguir mostra-se a sintaxe usada para declarar variáveis (locais) – instrução *Dim* e descrevem-se os elementos da sintaxe: nome da variável e tipo de dados. Antes de se passar à implementação do exemplo faz-se uma brevíssima descrição de alguns tipos de dados (apenas *String*, *Integer*, *Boolean* e *Date*) disponíveis no VB.

O exemplo consiste em inserir um botão no *form* que ao ser premido faça surgir no ecrã uma *Message Box* com o número de vezes que já foi premido. Para isso torna-se necessário declarar uma variável que sirva de contador e que vá sendo incrementada à medida que se vai premindo o botão. Todos os passos são explicados desde a localização da secção de declaração de variáveis até à escrita do código para incrementar a variável e mostrar no ecrã o número de vezes que se premiu o botão, passando pela própria declaração da variável. A seguir pede-se ao aluno que teste o programa.

Depois deste exemplo o autor continua com outro exemplo, desta vez para simular a validação da entrada num sistema por intermédio da introdução de uma *password*.

Mais uma vez todos os passos para implementar este programa são apresentados. Pede-se para inserir no *form* uma caixa de texto e alterar o seu nome, transformá-la numa caixa de entrada de *passwords* colocando um asterisco na propriedade *PasswordChar* e apagar a propriedade *Text*. A seguir pede-se também para inserir um botão alterando o seu nome e o *Caption*.

O passo seguinte consiste em declarar uma variável na secção de declarações do *form* e escrever o código apresentado no procedimento *Click* do botão. O código serve para validar a *password* introduzida e ao fim de 3 tentativas falhadas deixar de aceitar a *password* mesmo que correcta.

Com este exemplo o autor aproveita para apresentar as estruturas de decisão *If ... Then* e *If ... Then … Else* mas apenas como comentário ao código apresentado.

4ª Parte - Ciclos e sub-rotinas

Nesta parte do tutorial são apresentadas as estruturas de controlo de fluxo *For ... Next* e *Do ... Until*, ensinando-se de seguida uma forma simples de obter informações do utilizador, usando as *InputBox*. Por fim apresentam-se os *Sub* – sub-rotinas, a passagem de valores por parâmetro e os operadores aritméticos.

Começa com a apresentação da estrutura de controlo de fluxo *For ... Next*, explicando-se a utilidade deste tipo de estruturas. É apresentado um exemplo muito simples ao qual se segue a sua explicação – as instruções dentro do ciclo são repetidas *n* vezes de acordo com o intervalo definido.

A seguir apresenta-se outra estrutura de controlo de fluxo: *Do ... Until*. Esta estrutura é explicada com base num exemplo que consiste em pedir ao utilizador que introduza nomes de alunos usando para o efeito uma *InputBox* – caixa de entrada de dados. O utilizador termina a introdução dos nomes dos alunos escrevendo a palavra "FIM".

As *InputBox* são uma forma simples de obter informações do utilizador. Essa informação pode ser guardada numa variável para mais tarde ser usada. É analisando o valor desta variável que no exemplo apresentado se determina se o ciclo *Do ... Until* deve ou não parar.

Depois de apresentadas as estruturas de controlo de fluxo o autor decide desafiar o aluno a implementar um programa que, na sequência do exemplo anterior, guarde os nomes dos alunos numa *list-box*. O programa deve começar por perguntar o número de alunos a introduzir e só depois apresentar esse número de caixas de entrada, uma para cada aluno e à medida que os nomes forem sendo introduzidos deve guardá-los na referida *list-box*.

É evidente que o desafio é grande para um aluno que nunca tenha trabalhado com uma *list-box* e talvez por isso a solução, leia-se código, é de imediato apresentada. O código encontra-se comentado linha a linha para facilitar a aprendizagem.

O tutorial continua com a definição de *Sub* – sub-rotina e qual a sua utilidade. É apresentado um exemplo que mostra precisamente como pode ser útil a utilização das sub-rotinas. O exemplo consiste em criar um pequeno programa que faça aparecer no ecrã uma dada mensagem sempre que o utilizador termina a sua execução. O utilizador pode terminar a execução do programa de duas maneiras: fechando a janela do *form* ou premindo um botão.

Na apresentação do exemplo é explicado como criar o *Sub*, como chamá-lo e em que sítios deve ser chamado para produzir o efeito desejado (no evento *QueryUnload* do *form* e no evento *Click* do botão).

São ainda referidas algumas vantagens da utilização das sub-rotinas, como por exemplo evitar a duplicação de código e facilitar a sua manutenção.

Esta parte termina com mais um exemplo, desta vez para explicar a passagem de parâmetros. No exemplo existem duas caixas de texto e dois botões, um para cada caixa de texto. O que se pretende é que o utilizador introduza valores nas caixas de texto e ao premir o botão respectivo seja apresentado, por intermédio de uma *Message Box*, o resultado da multiplicação desse valor por 2.

Para implementar este exemplo é explicado como criar uma sub-rotina que aceite um valor como parâmetro e efectue a operação pretendida. É também explicado como passar um valor na chamada à sub-rotina.

Toda a implementação é explicada em pormenor desde a inserção dos controlos no *form*, à indicação do local a partir do qual a sub-rotina deve ser chamada, passando pela criação da própria sub-rotina. É também apresentada uma imagem do programa em execução.

Durante a explicação de como implementar este exemplo são apresentados os operadores aritméticos mais importantes (multiplicação, adição, subtracção e divisão).

5ª Parte - Funções e revisão às propriedades, métodos e eventos

Nesta parte é feita uma revisão às propriedades, métodos e eventos, sendo ainda apresentadas as funções.

Quase a chegar ao fim do tutorial o autor decide fazer uma revisão às propriedades, métodos e eventos, usando uma laranja para explicar estes conceitos.

A laranja tem uma propriedade que indica a sua cor. Por defeito é o cor-de-laranja mas em Visual Basic pode ser alterada para outra cor. É de novo apresentado um exemplo da sintaxe *Objecto.Propriedade=Valor*. Outras propriedades da laranja poderiam ser o seu tamanho ou se é venenosa ou não.

A laranja também tem métodos, diz o autor. Podemos ter o método *Comer*, tal como uma folha de cálculo pode ter o método *AlinharColunas*. É apresentado um exemplo da sintaxe *Objecto.Método*.

Por fim são apresentados os eventos e voltando ao exemplo da laranja podemos querer saber quando o utilizador come um gomo para depois lhe apresentar a conta. Isto é feito na resposta a um evento que se pode chamar *ComeuUmGomo*. Tal como para as propriedades e métodos é apresentado um exemplo de um evento.

A seguir apresentam-se as funções. O autor considera as funções como sendo provavelmente a coisa mais importante que se aprende em todo o curso. Considera-as úteis, eficientes e espertas. É feita a comparação com os *Sub* e explicado que a única diferença é que as funções devolvem sempre um valor. Mais à frente o autor volta a referir esta diferença mostrando o quão importante ela é.

Como não podia deixar de ser é apresentado mais um exemplo que desta vez consiste em ter no mesmo *form* 8 caixas de texto e para cada par de caixas de texto associar um botão. A ideia é usar cada botão para adicionar os valores das caixas de texto respectivas. Assim, o primeiro botão adiciona os valores das duas primeiras caixas de texto, o segundo adiciona os valores das caixas de texto seguintes, e assim sucessivamente.

Antes de explicar como criar uma função para resolver este problema, o autor decide mostrar como seria fastidioso implementar o exemplo se tivesse-mos que escrever o código para cada um dos botões. E se já não quisesse-mos adicionar os valor mas sim multiplicá-los? Tinha-mos que alterar o código em 4 sítios diferentes.

Usando funções a programação fica facilitada porque basta criar uma função que aceite como parâmetros dois valores e devolva a sua soma. Segue-se então a apresentação do código da função que executa este cálculo, bem como a sua explicação. Por fim mostra-se como se efectua a chamada à função cujo resultado é imediatamente usado numa *Message Box*.

O autor termina esta parte dizendo que as funções são muitas das vezes bastante mais complicadas do que a que foi criada para efectuar a soma de dois valores. Refere como exemplo uma função que tenha de aceder a uma base de dados para determinar se um dado utilizador se encontra ligado no sistema.

6ª Parte - Perguntas mais frequentes

Na última parte do tutorial o autor apresenta-nos uma espécie de FAQ – Frequently Asked Questions dividido em 5 partes – as partes que compõem o tutorial – em que se aproveita para abordar alguns assuntos que não foram apresentados antes.

3.4 Análise Comparativa

De forma a tornar possível uma análise comparativa entre os tutoriais estudados, foram definidos alguns tópicos de comparação que se agruparam em 3 áreas: aspectos genéricos, características específicas e avaliação dos tutoriais.

Aspectos Genéricos

	TUTORIAL I	**TUTORIAL II**	**TUTORIAL III**
Tutorial de acesso público	sim	sim	sim
Necessário conhecimentos prévios	sim	sim	não
Lingua	inglês	inglês	inglês

Características Específicas

	TUTORIAL I	**TUTORIAL II**	**TUTORIAL III**
Abordagem algorítmica	mínima	forte	suficiente
Abordagem ao ambiente de desenvolvimento	forte	mínima	suficiente
Conceitos algorítmicos abrangidos	mínimo	suficiente	forte
Conceitos do ambiente de desenvolvimento abrangidos	suficiente	mínimo	suficiente

Avaliação dos Tutoriais

	TUTORIAL I	TUTORIAL II	TUTORIAL III
Quantidade de exemplos	mínimo	suficiente	suficiente
Qualidade dos exemplos	razoável	fraco	razoável
Qualidade gráfica	fraco	fraco	bom
Aspectos negativos	links errados	exemplos isolados e não explica a implementação	controlos VB apresentados muito ligeiramente
Qualidade geral	fraco	fraco	razoável

Como aspecto negativo aplicável aos 3 tutoriais estudados pode-se referir ainda o facto de todos eles aparecerem escritos em inglês.

4

Tutorial Proposto

O objectivo que serviu de base ao desenvolvimento deste tutorial foi o de criar um conjunto de documentos que facilitasse a iniciação dos alunos de Engenharia (Química, Civil, Mecânica, ...) à Programação em Visual Basic. Assim, não são necessários quaisquer conhecimentos prévios para que os alunos possam segui-lo.

O tutorial baseia-se na metodologia de ensino Learning by Example o que significa que os alunos irão aprender através de exemplos explicados em profundidade e cuja complexidade aumentará progressivamente. Estes exemplos serão implementados pelos alunos à medida que eles forem evoluindo na aprendizagem.

Para praticar a implementação dos algoritmos será usado o ambiente de desenvolvimento do Visual Basic. Este é um ambiente de desenvolvimento integrado, o que significa que se podem usar alguns objectos (botões, listas, menus, ...) para criar as aplicações.

Em função da análise realizada no capítulo anterior desenvolveu-se um tutorial que respondesse às lacunas encontradas. Assim, decidiu-se documentar um tutorial em português, dirigido a alunos sem quaisquer conhecimentos de programação e que abordasse a algoritmia e o ambiente de desenvolvimento do Visual Basic de uma forma abrangente e ao mesmo tempo aprofundada.

O tutorial proposto divide-se em 5 partes e encontra-se estruturado da seguinte forma:

- **Tutorial – Parte I**
  - Introdução
  - Ambiente de desenvolvimento
  - Criar o projecto
  - Desenhar a interface
  - Definir as propriedades
  - Escrever o código
  - Correr e testar a aplicação
  - Gravar o trabalho

- **Tutorial – Parte II**
  - Introdução
  - Variáveis e Constantes
  - Operadores
  - Estruturas de decisão

- **Tutorial – Parte III**
  - Introdução
  - Arrays
  - Estruturas de controlo de fluxo

- **Tutorial – Parte IV**
  - Introdução
  - Procedimentos e Funções
  - Variáveis Locais e Globais

- **Tutorial – Parte V**
  - Introdução
  - Controlos – Parte I
  - Controlos – Parte II
  - Controlos – Parte III

A primeira parte faz uma abordagem ao ambiente de desenvolvimento do Visual Basic e usa um exemplo cuja implementação é explicada ao longo desta parte.

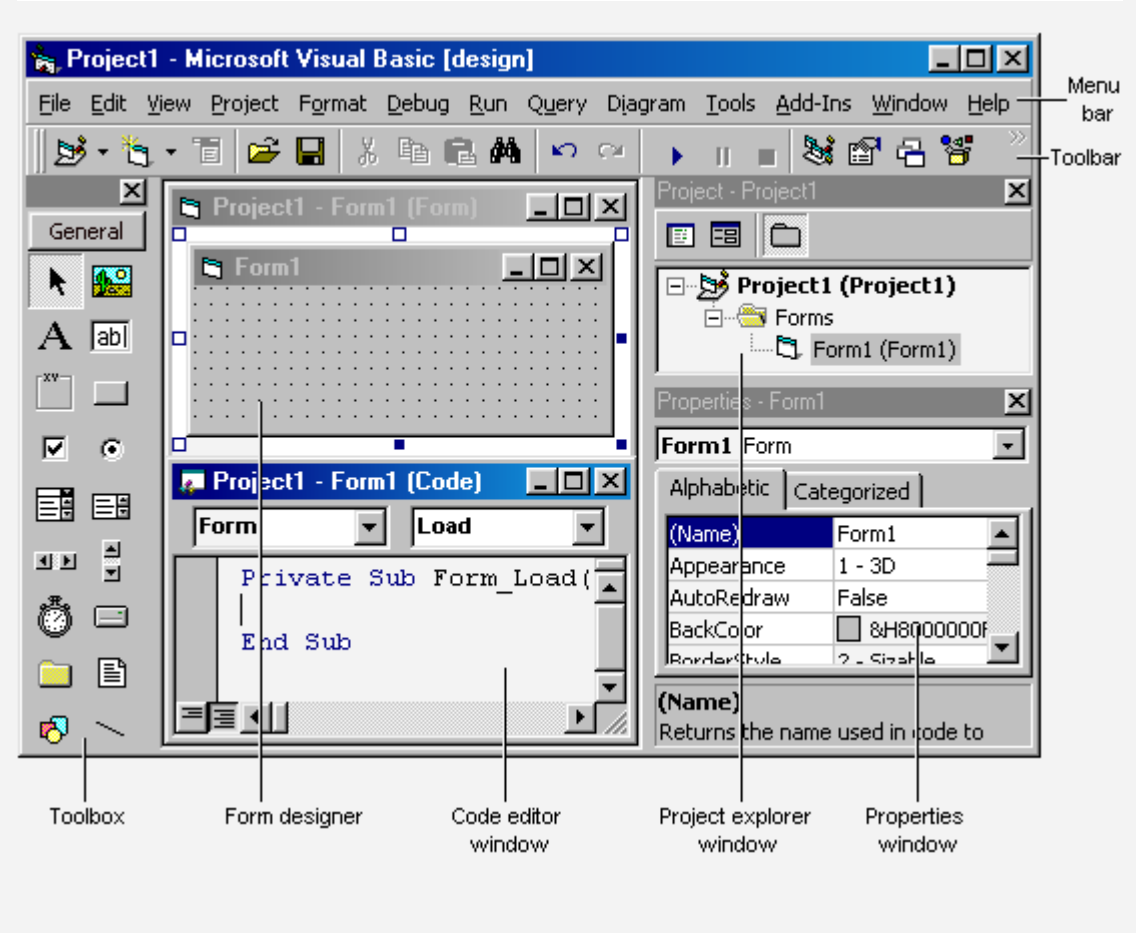
A partir da segunda parte são usados exemplos cuja implementação é explicada logo à partida e depois, à mediada que os conceitos algorítmicos vão sendo apresentados, vai sendo feita a explicação do código.

As partes sombreadas representam excertos tirados do tutorial desenvolvido e que se encontra publicado no endereço:

http://www.dei.isep.ipp.pt/~agouveia/tutorialvb

4.1 Tutorial – Parte I

A primeira parte começa por apresentar o ambiente de desenvolvimento integrado do Visual Basic (*menu bar*, *toolbars*, *toolbox*, *project explorer window*, *properties window*, *form designer* e *code editor window*).



O Ambiente de Desenvolvimento do Visual Basic é composto pelos seguintes elementos:

Menu Bar

Apresenta os comandos que se usam para trabalhar com o Visual Basic. Para além dos habituais menus *File*, *Edit*, *View*, *Window* e *Help*, existem outros menus que permitem o acesso a funções específicas da programação tais como *Project*, *Format* ou *Debug*.

Toolbars

Possibilitam um acesso rápido aos comandos mais frequentemente usados. Basta clicar uma vez num botão da toolbar para executar a acção associada a esse botão.

Toolbox

Fornece um conjunto de ferramentas usadas para inserir controlos nos forms.

Project explorer window

Lista os forms e módulos existentes no projecto actual. Um projecto é uma colecção de ficheiros usados para construir uma aplicação.

Properties window

Permite modificar a aparência ou comportamento do form ou controlo seleccionado. As propriedades são características dos objectos, tais como tamanho, texto ou cor.

Form designer

É a janela de trabalho que serve para desenhar a interface da aplicação com o utilizador. Inserem-se controlos, gráficos e imagens no form para se obter o resultado e aparência desejados. Cada form da aplicação tem o seu próprio "*form designer*".

Code editor window

Editor onde se escrevem as instruções que irão responder às acções do utilizador: botão premido, movimentos do rato, entrada de dados, etc. Existe um editor separado para cada form da aplicação.

Depois explica-se como se cria um projecto e detalham-se os passos necessários à construção de uma aplicação em VB. São eles:

- Desenhar a interface
- Definir as propriedades
- Escrever o código

No desenvolvimento de qualquer aplicação em VB começa-se por criar um projecto que reunirá todas as partes necessárias ao funcionamento da aplicação (forms, módulos, ...).

Para criar um projecto deve-se escolher a opção ***New Project*** do menu *File* e de seguida seleccionar ***Standard EXE*** na janela *New Project*.

Nota: Quando se corre o Visual Basic pela primeira vez, a janela *New Project* surge por defeito.

EXEMPLO – PARTE I

Crie então um projecto da forma indicada.

No "Desenhar a interface" explica-se, passo a passo, como se desenha um controlo no *form* e como mover um controlo, ou alterar a sua dimensão.

O primeiro passo no desenvolvimento de uma aplicação consiste em criar o form que será a base da interface dessa aplicação. De seguida desenham-se os controlos necessários no form criado.

Os controlos são caixas, botões ou texto desenhados no form para receber ou mostrar informação.

Como desenhar um controlo no form?

1. Seleccionar o controlo pretendido na *toolbox*
2. Mover o apontador do rato para o form - o apontador do rato passa a ser uma cruz
3. Colocar a cruz onde se deseja o canto superior esquerdo do controlo
4. Clicar no botão esquerdo do rato e sem largar efectuar um movimento de arrastamento até o controlo ter o tamanho desejado
5. Libertar o botão do rato - o controlo aparece no form

Para mover um controlo basta clicar com o botão esquerdo do rato sobre o controlo a mover e, sem largar, efectuar um movimento de arrastamento até chegar à posição pretendida.

Para alterar as dimensões selecciona-se o controlo e arrastam-se os *handles* (8 pequenos quadrados) em seu redor para dar o tamanho desejado.

EXEMPLO – PARTE I

Neste pequeno exemplo vamos usar 3 tipos de controlos:

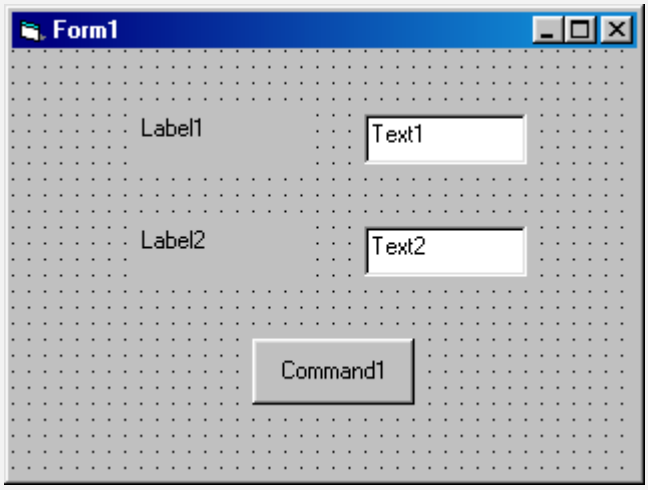
Controlo	Designação	Descrição
	Label (etiqueta)	Contém texto e é normalmente usado para descrever algo
	Text box (caixa de texto)	Permite a visualização e edição de dados
	Command button (botão)	Executa uma acção (previamente associada) quando premido

Insira os seguintes controlos no form:

- 2 label (etiquetas)
- 2 text box (caixas de texto)
- 1 command button (botão)

Após ter inserido estes controlos deverá obter algo semelhante ao form a seguir apresentado.

O próximo passo será o de ajustar as propriedades dos controlos e do próprio form.



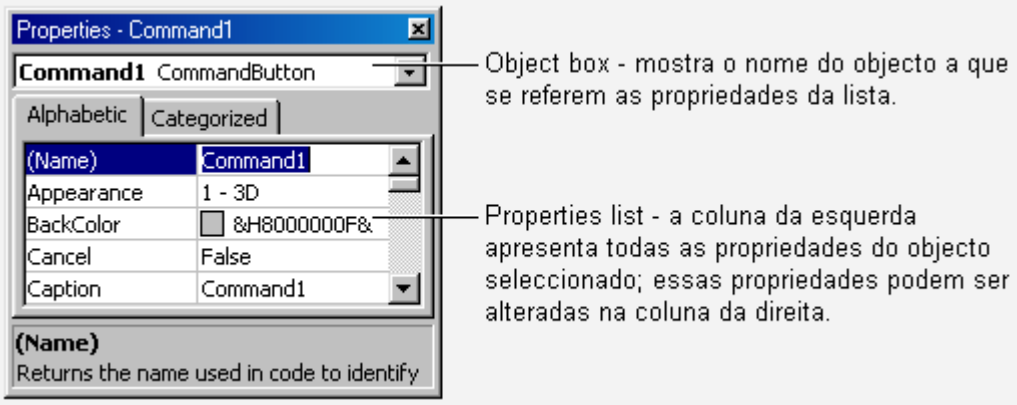
A seguir, na definição das propriedades, apresenta-se o conceito de propriedade de um objecto e descrevem-se os elementos que constituem a *Properties Window* – janela de propriedades.

Todos os objectos que constituem a parte visível de um programa em Visual Basic têm propriedades (o próprio form é um objecto e tem propriedades).

As propriedades de um controlo configuram-se na janela das propriedades - ***Properties Window***. Esta janela mostra sempre as propriedades do controlo (eventualmente um form) seleccionado. Por esta razão, para configurar as propriedades de um objecto é necessário

seleccioná-lo previamente.

Em alguns casos, o valor da propriedade pode ser escolhido de uma lista de opções pré-definidas.



EXEMPLO – PARTE I

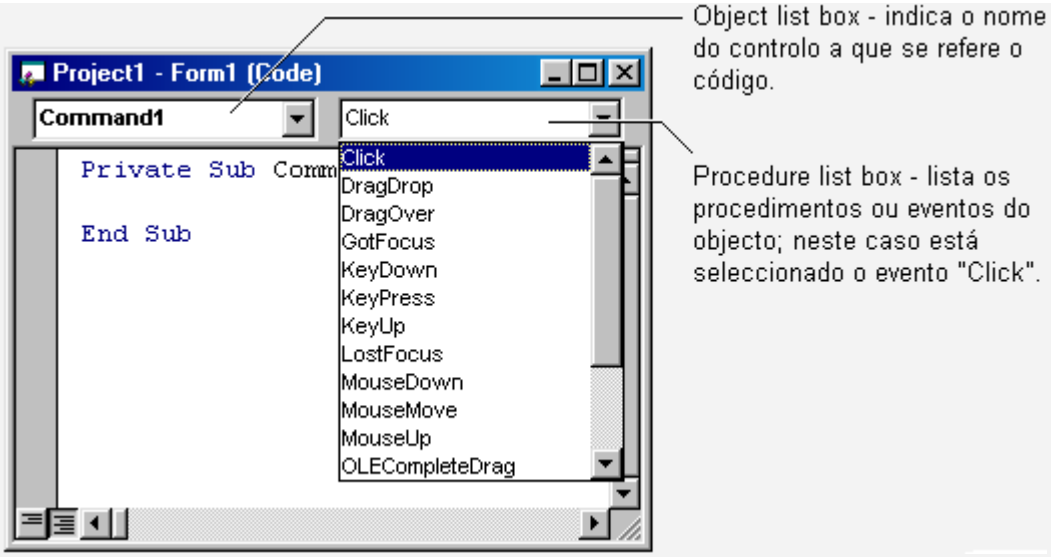
Altere então as seguintes propriedades:

Objecto	**Propriedade**	**Valor**
Form	Caption	Conversão Farenheit – Celcius
Label1	Caption	Graus Celcius:
Label2	Caption	Graus Farenheit:
Text1	Name	txtCelcius
	Text	(vazio)
Text2	Name	txtFarenheit
	Text	(vazio)
Command1	Name	cmdConverter
	Caption	Converter

Chegado à parte “Escrever o código”, o aluno fica a conhecer o *Code Editor* - editor de código do Visual Basic, e aprende como e onde se escreve o código que irá ser executado quando ocorrer um evento *Click*.

O editor de código ***- Code Editor -*** do Visual Basic é onde se escrevem as instruções de resposta às acções do utilizador. Através deste editor pode-se rapidamente ver ou editar o código da aplicação.

Para abrir o editor de código basta fazer duplo-clique no controlo ou form para o qual se pretende escrever o código.



O código numa aplicação VB está dividido em pequenos blocos a que se dá o nome de procedimentos. Um procedimento de evento **- event procedure -** contém código que é executado quando o evento ocorre como por exemplo quando se prime um botão.

Cada objecto possui um conjunto próprio de eventos aos quais reage. O nome dos procedimentos de evento é sempre composto pelos nomes do objecto e do evento separados por um caracter de *underscore* (_). Por exemplo, se quisermos que o botão **Command1** chame um evento quando premido, usamos o procedimento **Command1_Click**.

EXEMPLO – PARTE I

Faça duplo-clique no botão ***Converter***, escolha o evento ***Click*** e no espaço entre as declarações "Private Sub cmdConverter_Click()" e "End Sub" escreva a seguinte instrução:

txtFarenheit.Text = 32 + 1.8 * txtCelcius.Text

Por fim mostra-se como se corre uma aplicação...

EXEMPLO – PARTE I

Corra a aplicação seleccionando a opção ***Start*** que se encontra no menu *Run*, ou simplesmente premindo a tecla F5.

Para testar a aplicação introduza valores na caixa de texto relativa aos graus *Celcius*, prima o botão *Converter* e confirme o resultado que aparece na caixa de texto dos graus *Farenheit*.

...e se guarda o trabalho.

Para guardar o trabalho usa-se a opção ***Save Project*** do menu *File*. O Visual Basic pede então para indicar os nomes dos forms e do próprio projecto separadamente, bem como o local onde o trabalho ficará guardado.

O VB usa as extensões ".frm" para forms e ".vbp" para projectos.

Para efectuar alterações a um programa gravado bastará abrir o ficheiro do projecto, usando para o efeito a opção ***Open Project*** do menu *File*.

EXEMPLO – PARTE I

Como o nosso pequeno programa tem um único form, o Visual Basic usará apenas dois ficheiros para o guardar. Por defeito o VB atribuirá os nomes "Form1.frm" e "Project1.vbp" ao form e projecto, respectivamente. Estes nomes podem no entanto ser alterados no momento da gravação.

Proceda então à gravação do trabalho. Poderá escolher um nome mais apropriado para o projecto, como por exemplo "ConverteGraus".

Tudo isto foi complementado com um exemplo, explicado ao longo desta primeira parte e que o aluno foi sendo convidado a implementar. O exemplo consistiu em desenvolver um pequeno programa que efectuasse a conversão entre graus Celcius e graus Farenheit.

4.2 Tutorial – Parte II

Esta parte, com base num pequeno exemplo, aborda as seguintes áreas:

- Variáveis e Constantes
- Operadores
- Estruturas de decisão

4.2.1 Introdução

A segunda parte deste tutorial começa com a descrição de um exemplo que consiste em achar a média final de um aluno e, através dessa média, determinar se o aluno foi aprovado ou reprovado. A média é calculada tendo em conta os resultados dos seguintes componentes e respectivos pesos:

Mini-teste	20%
Trabalho prático	30%
Exame final	50%

A seguir apresenta-se o *form* que se poderia obter na implementação deste exemplo. Os nomes dos controlos mais importantes são também indicados, tal como se pode ver na figura seguinte.

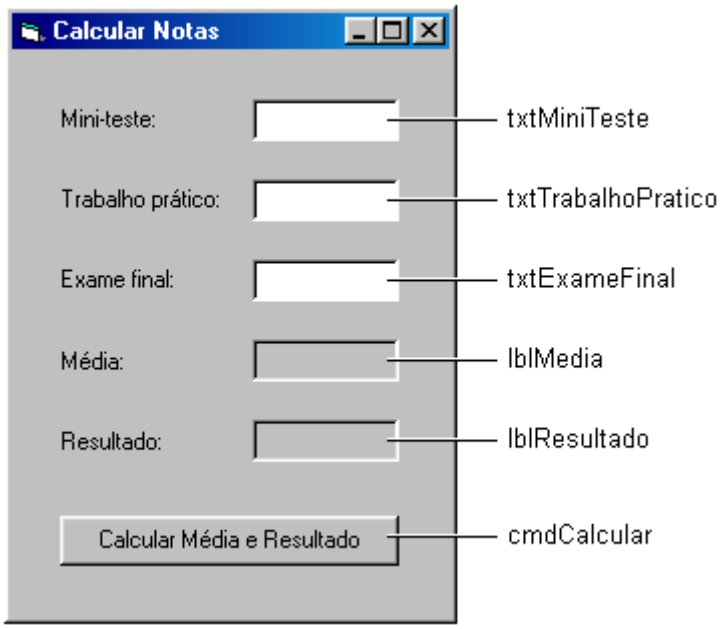


Figura 4.1 – *Form* do exemplo apresentado na segunda parte do tutorial

Depois de introduzir o valor da nota de cada componente na respectiva caixa de texto, o utilizador prime o botão e obtém o valor da média e o resultado do aluno (aprovado ou reprovado).

A descrição do exemplo termina com a apresentação do código necessário ao cálculo da média e verificação do resultado.

4.2.2 Variáveis e Constantes

Neste tópico começa-se por explicar a utilidade das variáveis, mostrando-se a seguir a forma como são declaradas. É ainda apresentada uma lista com os tipos de dados mais usados: Byte, Boolean, Integer, Long, Single, Double, Date, String e Variant.

Variáveis

No Visual Basic as variáveis são usadas para guardar valores temporariamente enquanto a aplicação é executada. As variáveis têm um nome pelo qual são identificadas e um tipo que determina o tipo de dados que a variável irá guardar.

Pode-se pensar numa variável como sendo um espaço de armazenamento na memória para guardar um valor inicialmente desconhecido.

Declaração de Variáveis

A instrução de declaração **Dim** serve para declarar variáveis e reservar espaço de armazenamento. A sintaxe para declaração de uma variável é a seguinte:

Dim <*nome da variável*> [As <*tipo*>]

exemplo: Dim Nome As String

Existe um conjunto de requisitos que o nome da variável deve satisfazer entre os quais a obrigatoriedade de começar por uma letra e de ser constituído apenas por letras, dígitos ou sublinhado (*underscore*).

O parâmetro ***tipo*** da instrução Dim indica o tipo de dados que se pretende colocar na variável para que o Visual Basic possa reservar o espaço adequado a cada situação.

Eis alguns dos tipos existentes:

Tipo	Descrição
Byte	Numérico inteiro (0 a 255)
Boolean	Lógico (True ou False)
Integer	Numérico inteiro (-32,768 a 32,767)
Long	Numérico inteiro (-2,147,483,648 a 2,147,483,647)
Single	Numérico decimal (-3.40E38 a -1.40E-45 para valores negativos; 1.40E-45 a 3.40E38 para valores positivos)
Double	Numérico decimal (-1.80E308 a -4.94E-324 para valores negativos; 4.94E-324 a 1.80E308 para valores positivos)
Date	Data (1-Janeiro-100 a 31-Dezembro-9999) ou hora (0:00:00 a 23:59:59)
String	Texto
Variant	Numérico, data, hora ou texto

EXEMPLO – PARTE II

No nosso exemplo apenas usamos uma variável para guardar o valor da média:

```
Dim Media As Single
```

Esta variável foi definida como sendo do tipo *Single* porque irá guardar valores numéricos decimais.

A seguir explica-se o que se entende por "declaração implícita", apresentando-se as vantagens e desvantagens deste tipo de declaração, bem como do uso do tipo de dados Variant.

O Tipo Variant e a Declaração Implícita

O Visual Basic permite que sejam usadas variáveis sem serem declaradas - são as variáveis com **declaração implícita**. Se por um lado esta facilidade é interessante por poupar o trabalho de declarar as variáveis antes de as usar, por outro lado pode conduzir a erros no programa: basta a troca de uma letra no nome da variável para que o Visual Basic a considere uma variável diferente.

O tipo **Variant** é o tipo de dados que o Visual Basic usa para as variáveis que não são declaradas. Este tipo de dados é muito flexível pois permite guardar diferentes tipos de informação (números, texto, datas ou horas). No entanto tem a desvantagem de tornar os programas mais lentos.

As constantes e a sua utilidade na programação são explicadas a seguir. Apresenta-se ainda a sintaxe para a declaração das constantes e os tipos de dados que podem guardar.

Constantes

Por vezes existem valores que aparecem repetidas vezes no código de um programa ou que são difíceis de memorizar. Nestes casos pode-se facilitar a leitura do código ou a sua actualização usando constantes.

Uma constante é um identificador que toma o lugar de um número ou texto e que não muda ao longo do funcionamento da aplicação. A grande diferença entre uma constante e uma variável é que a primeira não pode assumir outro valor para além do que lhe foi atribuído aquando da sua declaração.

Declaração de Constantes

A sintaxe para declaração de uma constante é:

- **Const <*nome da constante*> [As <*tipo*>] = <*expressão*>**

  exemplo: Const Pi = 3.14159265358979

Os requisitos a satisfazer pelo nome da constante são os que foram descritos para o caso das variáveis.

O parâmetro ***tipo*** indica o tipo de dados da constante e ***expressão***, apesar de na maior parte das vezes ser um número ou texto, pode também ser uma expressão (com operandos e operadores) que resulte num número ou texto.

exemplo: Const Perimetro = 2 * Pi * Raio

EXEMPLO – PARTE II

No exemplo apresentado definimos 3 constantes que representam os pesos de cada componente no cálculo da média:

```
Const PESO_MT As Single = 0.2
Const PESO_TP As Single = 0.3
Const PESO_EF As Single = 0.5
```

Estas constantes, tal como no caso da variável *Media*, são do tipo *Single* uma vez que representam valores numéricos decimais.

4.2.3 Operadores

Neste tópico são apresentados os operadores usados na programação em Visual Basic, divididos por 3 grupos:

- Operadores Aritméticos
- Operadores Lógicos
- Operadores Relacionais

É também apresentado o operador usado na concatenação de strings (**&**).

Uma expressão consiste numa fórmula que, depois de avaliada, produz um resultado. A expressão é constituída por operadores (que indicam as acções a efectuar) e operandos (valores sobre os quais são efectuadas as acções).

Os resultados das expressões dependem dos operadores e dos operandos: uma operação aritmética devolve um resultado numérico e uma operação relacional ou lógica produz um

resultado lógico.

Os operadores podem ser agrupados da seguinte forma:

Operadores Aritméticos	
+	Adição
-	Subtracção
*	Produto
/	Divisão
\	Divisão inteira
MOD	Resto da divisão inteira
^	Exponenciação

Operadores Lógicos	
AND	E
OR	Ou
NOT	Não
EQV	Equivalência lógica
IMP	Implicação lógica
XOR	Ou exclusivo

Operadores Relacionais	
=	Igual
<>	Diferente
<	Menor
>	Maior
<=	Menor ou igual
>=	Maior ou igual

Existe ainda um operador usado para concatenação de strings. Esse operador é o **&**.

EXEMPLO – PARTE II

No exemplo apresentado usamos operadores aritméticos (***** e **+**) e relacionais (**>=**):

```
Media = txtMiniTeste.Text * PESO_MT  +  txtTrabalhoPratico.Text * PESO_TP  +_
        txtExameFinal.Text * PESO_EF
...
If Media >= 10 Then
...
```

4.2.4 Estruturas de decisão

As estruturas de decisão apresentadas neste tópico são as seguintes:

- If ... Then
- If ... Then … Else

Começa-se por descrever a utilidade do uso deste tipo de estruturas e explica-se em que situações se usa uma ou a outra estrutura.

São apresentadas as sintaxes de ambos os tipos e explica-se como é que se pode controlar o fluxo de execução de um programa usando estas estruturas.

As estruturas de decisão permitem controlar o fluxo de execução de um programa. Estas estruturas baseiam-se no teste de condições para, de acordo com os resultados desse teste, executarem operações distintas.

Entre as estruturas de decisão que o Visual Basic suporta vamos estudar as seguintes:

- If ... Then
- If ... Then ... Else

If ... Then

Esta estrutura é usada quando se pretende executar uma ou mais instruções de forma condicionada. Pode-se usar a sintaxe de instrução única ou a sintaxe de múltiplas instruções, conforme o caso.

- **If <*condição*> Then <*instrução*>**
- **If <*condição*> Then**

  <conjunto de instruções>

  End If

A condição é normalmente uma comparação mas pode também ser qualquer expressão que resulte num valor numérico. O Visual Basic interpreta estes valores como **True** ou **False**. O valor zero é considerado *False* e qualquer valor diferente de zero é considerado *True*.

Se a condição for verdadeira (*True*), todas as instruções a seguir ao *Then* são executadas.

If ... Then ... Else

Esta estrutura usa-se quando se pretende definir mais do que um bloco de instruções e se pretende que um desses blocos seja sempre executado.

A sintaxe é a seguinte:

If <*condição 1*> Then

<*conjunto de instruções 1*>

[ElseIf <*condição 2*> Then

<*conjunto de instruções 2*>]

[Else

<*conjunto de instruções 3*>]

End If

O Visual Basic começa por testar a primeira condição. Se for falsa, passa a testar a segunda condição e assim sucessivamente até encontrar uma condição verdadeira. Quando encontra uma condição verdadeira o Visual Basic executa o conjunto de instruções respectivo. Se não encontrar nenhuma condição verdadeira então, caso tenha sido declarado, é executado o bloco de instruções do *Else*.

If ... Then ... ElseIf é apenas um caso especial do *If ... Then ... Else*. De notar que se pode ter qualquer número de instruções *ElseIf* e que se pode ou não ter a instrução *Else*.

EXEMPLO – PARTE II

No nosso exemplo usamos a estrutura de decisão *If ... Then ... Else* porque queremos que independentemente do resultado da condição seja sempre achado um resultado (Aprovado ou Reprovado):

```
If Media >= 10 Then
    lblResultado.Caption = "APROVADO"
Else
    lblResultado.Caption = "REPROVADO"
End If
```

A expressão *Media >= 10* é uma expressão lógica - descreve uma condição que desejamos testar. Se a condição é verdadeira então vai ser executada a instrução relativa ao *Then* (*lblResultado.Caption = "APROVADO"*). Se a condição é falsa vai ser executada a instrução relativa ao *Else* (*lblResultado.Caption = "REPROVADO"*).

4.3 Tutorial – Parte III

A terceira parte do tutorial, tal como acontece nas outras partes, começa por apresentar um exemplo que, neste caso, servirá de base à abordagem dos seguintes tópicos:

- Arrays
- Estruturas de controlo de fluxo

4.3.1 Introdução

O exemplo a implementar vem na sequência do exemplo da primeira parte só que desta vez pretende-se achar a média final de uma turma que tem no máximo 20 alunos. A média de cada aluno é calculada da mesma forma.

O *form* que se poderia obter na implementação deste exemplo é idêntico ao apresentado na parte anterior. Existe apenas mais um botão que servirá para ir guardando as notas dos alunos.

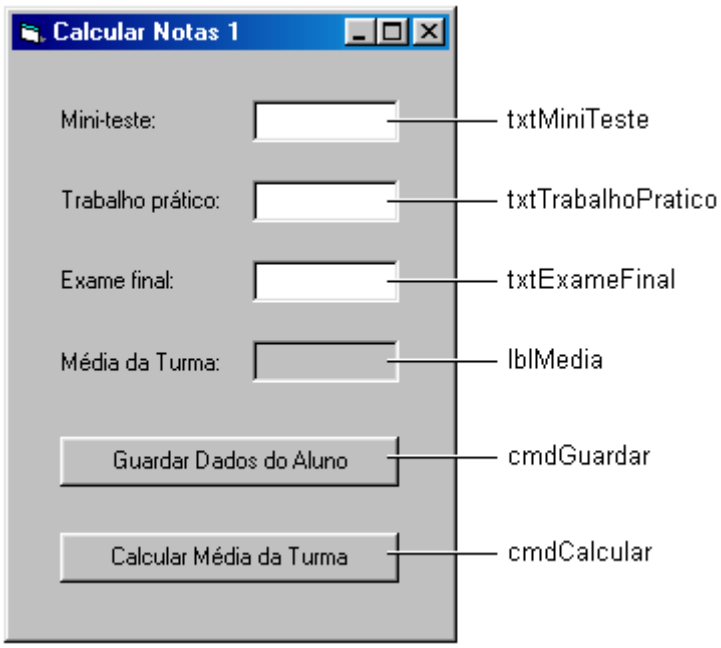


Figura 4.2 – *Form* do exemplo apresentado na segunda parte do tutorial

Depois de introduzir as notas do primeiro aluno, o utilizador prime o botão "Guardar Dados do Aluno". Este processo repete-se para todos os alunos e no fim, quando não houver mais alunos, o utilizador pode achar a média da turma carregando no botão respectivo.

A descrição do exemplo termina com a apresentação do código necessário ao cálculo da média da turma.

4.3.2 Arrays

Neste tópico começa-se por explicar as funcionalidades dos arrays, mostrando-se de seguida a forma como podem ser declarados. São ainda apresentados os arrays multi-dimensionais.

Os arrays permitem que se refira a um conjunto de variáveis usando sempre o mesmo nome e que se use um número (índice) para as diferenciar.

Um array é portanto um tipo de variável estruturado e constituído por um número fixo de elementos, todos do mesmo tipo. Claro que quando o tipo de dados é o Variant os elementos podem ser de tipos diferentes.

Declaração de Arrays

Tal como na declaração de variáveis, os arrays (um tipo de variável especial) são declarados recorrendo à instrução **Dim**. A sintaxe para declaração de um array é a seguinte:

- **Dim <*nome da variável*>(<*dimensão*>) [As <*tipo*>]**

  exemplo 1: Dim A(10) As Integer

  exemplo 2: Dim AA(5 To 10) As Integer

Como se pode verificar nestes exemplos, a dimensão do array pode ser indicada de duas formas:

a) com um único valor numérico que define quantos elementos tem o array (o primeiro elemento tem índice 0);

b) com a palavra-chave **To** que permite especificar os índices inicial e final do array.

Sendo assim, no 1º exemplo o array é declarado com 11 elementos, uma vez que o índice varia de 0 a 10. No 2º exemplo o array é declarado com 6 elementos, de 5 até 10, inclusive.

Os elementos dos arrays são acedidos da forma: ***<nome da variável>(<índice>)***.

Nota: O Visual Basic, contrariamente ao que acontece com as outras variáveis, exige que os arrays sejam previamente declarados para poderem ser usados.

Arrays Multi-dimensionais

Por vezes torna-se necessário guardar informação que de alguma forma está relacionada entre si. Por exemplo, quando estamos a trabalhar com sistemas de coordenadas bi-dimensionais cada ponto é referenciado através das suas coordenadas: *X* e *Y*. Se pretendêssemos guardar 10 pontos num array poderíamos usar um array com duas dimensões, como se mostra no exemplo seguinte.

exemplo: Dim Coord (1 To 10, 1 To 10) As Single

EXEMPLO – PARTE III

No nosso exemplo usamos um array bi-dimensional (20 x 3) para guardar as notas dos alunos. Isto porque, como foi dito, a turma terá no máximo 20 alunos, sendo cada aluno avaliado em 3 componentes.

Uma vez que o array irá guardar valores numéricos decimais, foi declarado do tipo *Single* da seguinte forma:

```
Dim Notas(1 To 20, 1 To 3) As Single
```

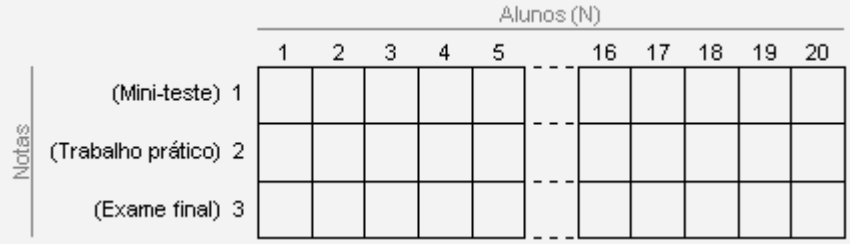
As notas dos alunos são guardadas no array através do código:

```
N = N + 1

Notas(N, 1) = txtMiniTeste.Text
Notas(N, 2) = txtTrabalhoPratico.Text
Notas(N, 3) = txtExameFinal.Text
```

N é uma variável contadora que vai sendo incrementada à medida que são introduzidas as notas dos alunos. Esta variável indica a coluna do array onde serão guardadas as notas do aluno actual.

O array pode ter uma representação gráfica do género da figura seguinte.



4.3.3 Estruturas de controlo de fluxo

As estruturas de controlo de fluxo apresentadas neste tópico são:

- Do ... Loop
- For ... Next

Na descrição da estrutura Do ... Loop começa-se por explicar em que situações deve ser utilizada, seguindo-se a apresentação da sintaxe e das variantes existentes: Do [While|Until] ... Loop, ou Do ... Loop [While|Until].

Do ... Loop

Esta estrutura baseia-se no teste de condições e deve ser usada sempre que se pretender executar um conjunto de instruções durante um número de vezes não definido. Existem

algumas variações desta estrutura mas todas elas verificam uma condição que indica se a execução deve ou não prosseguir.

Assim, a estrutura **Do ... Loop** pode tomar uma das seguintes formas:

- **Do [While | Until] <*condição*>**

  <*conjunto de instruções*>

  Loop

- **Do**

  <*conjunto de instruções*>

  Loop [While | Until] <*condição*>

A condição, tal como no caso do **If ... Then**, é normalmente uma comparação mas pode também ser qualquer expressão que resulte num valor numérico. O Visual Basic interpreta estes valores como **True** ou **False**. O valor zero é considerado *False* e qualquer valor diferente de zero é considerado *True*.

A principal diferença entre estas duas variações da estrutura **Do ... Loop** tem a ver com o facto de no segundo caso o conjunto de instruções ser sempre executado pelo menos uma vez. Isto porque a condição é testada apenas no fim do ciclo, ao passo que no primeiro caso a condição é testada logo à partida, o que pode levar a que o conjunto de instruções respectivo nunca chegue a ser executado.

A escolha entre as instruções **While** e **Until** deve ser feita de acordo com o resultado da condição. Isto quer dizer que se escolhermos **While** o conjunto de instruções vai ser executado **enquanto** a condição for verdadeira. Se nos decidirmos pela instrução **Until**, então neste caso o conjunto de instruções irá ser executado **até** a condição ser verdadeira.

Na apresentação da estrutura For ... Next explica-se em que situações o seu uso é o mais indicado, apresenta-se a sintaxe e descreve-se, passo a passo, o seu funcionamento.

For ... Next

A estrutura **Do ... Loop** é a indicada quando não se sabe à partida quantas vezes vai ser necessário executar o conjunto de instruções. Quando, por outro lado, se sabe que aquele conjunto de instruções tem que ser executado um número específico de vezes então a melhor escolha é a estrutura **For ... Next**.

Esta estrutura usa uma variável designada por contador que incrementa ou decrementa o seu valor a cada repetição do ciclo. A sintaxe é a seguinte:

- **For <*contador*> = <*início*> To <*fim*> [Step <*incremento*>]**

  <*conjunto de instruções*>

  Next [<*contador*>]

De notar que os argumentos *contador*, *start*, *end* e *incremento* são todos do tipo numérico.

Durante a execução do ciclo **For**, o Visual Basic faz o seguinte:

1. Inicializa o contador a início
2. Verifica se o contador é maior que **fim**. Se sim, o Visual Basic sai do ciclo (se o incremento for negativo, o Visual Basic verifica se o contador é menor que **fim**)
3. Executa o conjunto de instruções
4. Incrementa o contador em 1 unidade ou pelo valor do incremento, no caso de este ter sido definido
5. Repete os passos do 2 ao 4

EXEMPLO – PARTE III

No nosso exemplo usamos a estrutura de controlo de fluxo *For ... Next* para calcular a média da turma. Optou-se por esta estrutura porque sabemos logo à partida quantas vezes terão de ser executadas as instruções que se encontram dentro do ciclo. Como temos **N** alunos na turma é esse o número de vezes que temos de repetir o ciclo.

Eis o código usado:

```
For i = 1 To N
   MediaAluno = Notas(i, 1) * PESO_MT + Notas(i, 2) * PESO_TP + Notas(i, 3) * PESO_EF
   MediaAluno = Round(MediaAluno)

   MediaTurma = MediaTurma + MediaAluno
Next
```

Como se pode ver, **i** é a variável contadora. Começa em 1 e vai sendo incrementada em uma unidade de cada vez, a cada repetição do ciclo, até o seu valor ser maior do que **N**. Nesta altura o ciclo termina e a execução do programa continua na instrução imediatamente a seguir ao *Next*.

4.4 Tutorial – Parte IV

Esta parte, também com base num exemplo aborda as seguintes áreas da programação em Visual Basic:

- Procedimentos e Funções
- Variáveis Locais e Globais

4.4.1 Introdução

O exemplo a implementar consiste em alterar o exemplo anterior, onde se achava a média de uma turma, de forma a que a afectação das notas ao array seja feita usando um procedimento e a média da turma seja calculada através da chamada a uma função.

O *form* não sofre qualquer alteração em relação ao do exemplo da parte anterior.

Depois de introduzir as notas do primeiro aluno, o utilizador prime o botão "Guardar Dados do Aluno". Este processo repete-se para todos os alunos e no fim, quando não houver mais alunos, o utilizador pode achar a média da turma carregando no botão respectivo.

A descrição do exemplo termina com a apresentação do código necessário ao cálculo da média da turma usando funções e procedimentos.

4.4.2 Procedimentos e Funções

Neste tópico começa-se por fazer referência aos "*event procedures*" – procedimentos de evento, comparando-os aos procedimentos ditos gerais, a estudar nesta parte. A seguir apresentam-se as vantagens no uso de funções e procedimentos e explica-se a principal diferença entre estes dois tipos de sub-rotina.

No Visual Basic existem procedimentos que são executados em resposta às acções do utilizador, como por exemplo quando o utilizador prime um botão. Esses procedimentos são conhecidos por *"event procedures"* - procedimentos de evento.

No entanto, existem outros procedimentos que não estão directamente associados a qualquer evento e que para serem executados têm que ser explicitamente invocados. Estes são procedimentos mais gerais.

A principal razão para se usar este tipo de procedimentos tem a ver com o facto de, por vezes, diferentes procedimentos de evento necessitarem de executar o mesmo conjunto de acções (instruções). Nestas situações o ideal é colocar esse conjunto de acções num procedimento para evitar a duplicação de código e tornar a aplicação mais fácil de manter.

Para além dos procedimentos existem ainda as funções cuja finalidade é idêntica à dos procedimentos mas ao contrário destes devolvem sempre um valor.

Os procedimentos são os primeiros a ser explicados.

Procedimentos

A sintaxe é a seguinte:

Sub <*nome do procedimento*> (<*lista de argumentos*>)

<conjunto de instruções>

End Sub

A lista de argumentos é o conjunto de variáveis às quais são passados valores no momento da chamada do procedimento, valores esses que serão usados pelo procedimento durante as suas operações. Para cada argumento deve ser indicado o seu tipo e no caso de haver mais do que um argumento devem ser separados por vírgulas.

De cada vez que o procedimento é chamado, as instruções entre **Sub** e **End Sub** são executadas.

EXEMPLO – PARTE IV

No exemplo apresentado foi criado um procedimento que trata de guardar as notas dos aluno no array:

```
Sub Guardar_Notas(MT As Single, TP As Single, EF As Single)

    Notas(N, 1) = MT
    Notas(N, 2) = TP
    Notas(N, 3) = EF

End Sub
```

Este procedimento tem como parâmetros 3 variáveis do tipo Single às quais serão passadas as notas do aluno.

A chamada ao procedimento **Guardar_Notas** é feita da seguinte forma:

```
Guardar_Notas txtMiniTeste.Text, txtTrabalhoPratico.Text, txtExameFinal.Text
```

No momento em que é invocado o procedimento, as variáveis *MT*, *TP* e *EF* ficam com os valores de *txtMiniTeste*, *txtTrabalhoPratico* e *txtExameFinal*, respectivamente.

A seguir apresentam-se as funções.

Funções

A sintaxe para declarar uma função é a seguinte:

Function <*nome da função*> (<*lista de argumentos*>) [As *tipo*]

<conjunto de instruções>

End Function

O que foi descrito para os argumentos dos procedimentos aplica-se da mesma forma para os argumentos das funções.

As funções distinguem-se dos procedimentos porque devolvem um valor. Sendo assim, e tal como nas variáveis, as funções também têm um tipo de dados que será o tipo do valor a retornar.

Uma função cujo tipo de valor devolvido não seja explicitamente declarado devolverá um Variant.

O valor, normalmente resultante de algumas operações e/ou cálculos, é devolvido por atribuição a uma "variável" que é o próprio nome da função.

EXEMPLO – PARTE IV

No exemplo usou-se uma função para calcular a média da turma:

```
Function Achar_Media() As Byte

    Dim MediaAluno As Single
    Dim MediaTurma As Single

    Dim i As Byte

    MediaTurma = 0

    For i = 1 To N
       MediaAluno = Notas(i, 1) * PESO_MT + Notas(i, 2) * PESO_TP +_
                    Notas(i, 3) * PESO_EF
       MediaAluno = Round(MediaAluno)

       MediaTurma = MediaTurma + MediaAluno
    Next

    MediaTurma = Round(MediaTurma / N)

    Achar_Media = MediaTurma

End Function
```

Esta função retorna um valor do tipo *Byte* e não tem parâmetros de entrada. Como se pode ver a média é devolvida ao atribuir-se o seu valor à "variável" com o mesmo nome da

função: **Achar_Media**.

A chamada a esta função é feita da seguinte maneira:

```
lblMedia.Caption = Achar_Media()
```

O valor devolvido é imediatamente colocado no *label* respectivo (*lblMedia*).

De notar que, ao contrário da invocação de procedimentos onde nunca são usados parêntesis para agrupar os parâmetros, na invocação de funções é obrigatório o uso dos parêntesis mesmo que a função não tenha parâmetros, como é o caso.

4.4.3 Variáveis Locais e Globais

Neste tópico faz-se uma classificação das variáveis relativamente ao seu âmbito de acção: variáveis locais ao procedimento; variáveis locais ao módulo; e variáveis globais. Para cada tipo são indicados o local de declaração e a instrução a usar (*Dim* ou *Global*).

É ainda feita referência ao facto de também se poder declarar Constantes locais e globais sendo apenas diferente a forma como são declaradas (usando as instruções *Const* ou *Public Const*).

Quando se declara uma variável dentro de um procedimento, apenas o código desse procedimento pode aceder ou alterar o valor da variável. Podemos então dizer que a variável existe apenas dentro do procedimento onde foi declarada.

No entanto, torna-se por vezes necessário que uma variável tenha um âmbito de acção mais alargado. Podemos querer que essa variável esteja acessível a todos os procedimentos de um dado *form* ou módulo, ou acessível a toda a aplicação.

As variáveis devem ser declaradas com o menor âmbito de acção possível para não sobrecarregar a memória dos computadores e simplificar a estruturação dos programas.

Dependendo da forma como são declaradas, podemos ter variáveis:

- Locais ao procedimento
- Locais ao form ou módulo
- Globais

Variáveis Locais ao Procedimento

As variáveis locais ao procedimento são declaradas dentro dos procedimentos usando a instrução **Dim**. Estas variáveis são reconhecidas apenas dentro do procedimento onde forem declaradas e são por isso uma boa escolha para cálculos temporários.

Podem haver vários procedimentos em que existam variáveis com o mesmo nome. No

entanto, uma vez que estas variáveis são locais, cada procedimento tem a sua própria variável que pode ser alterada sem que isso afecte as variáveis dos outros procedimentos.

Este tipo de variáveis locais existem apenas enquanto o procedimento onde foram declaradas se encontrar em execução. Quando o procedimento termina, a variável deixa de existir perdendo-se o seu valor.

Variáveis Locais ao *form* ou módulo

As variáveis locais ao *form* ou módulo são declaradas na secção de declarações (*Declarations*) do respectivo *form* ou módulo. A declaração, tal como nas variáveis locais ao procedimento, é feita usando a instrução **Dim**.

Estas variáveis são reconhecidas em todo o *form* ou módulo onde forem declaradas. Desta forma é possível partilhar informação por todos os procedimentos existentes no *form* ou módulo.

As variáveis locais ao *form* ou módulo apenas deixam de existir quando a aplicação termina.

Variáveis Globais

As variáveis globais têm o maior âmbito de acção possível pois são reconhecidas em toda a aplicação. A declaração de uma variável global é feita na secção de declarações (*Declarations*) de um módulo, usando a instrução **Global** na vez do **Dim**.

As variáveis globais apenas deixam de existir quando a aplicação termina.

Constantes Locais e Globais

O que atrás foi dito sobre variáveis locais e globais no que se refere ao âmbito de acção e local de declaração, também se aplica às constantes.

A única diferença tem a ver com a forma como as constantes são declaradas. Assim, as constantes locais, quer sejam locais ao procedimento quer sejam locais ao *form* ou módulo são declaradas usando a instrução **Const**. As constantes globais são declaradas colocando a instrução **Public** imediatamente antes do **Const**.

Arrays

Aos arrays aplica-se tudo o que foi dito sobre variáveis locais e globais, uma vez que os arrays não são mais do que um tipo de variável estruturado.

EXEMPLO – PARTE IV

No exemplo apresentado foram usadas variáveis locais ao procedimento e locais ao *form*.

Como variáveis locais ao procedimento temos as variáveis *MediaAluno*, *MediaTurma* e a variável contadora *i*, todas elas da função **Achar_Media**:

```
Dim MediaAluno As Single
Dim MediaTurma As Single
Dim i As Byte
```

Estas variáveis apenas existem dentro da função onde foram declaradas e assim que a função termina elas deixam de existir.

Foram ainda usadas as seguintes variáveis e constantes locais ao *form*:

```
Const PESO_MT As Single = 0.2
Const PESO_TP As Single = 0.3
Const PESO_EF As Single = 0.5

Dim Notas(1 To 20, 1 To 3) As Single
Dim N As Byte
```

4.5 Tutorial – Parte V

Na última parte do tutorial dão-se a conhecer alguns dos controlos mais usados no desenvolvimento de aplicações em Visual Basic.

Assim, tendo como pano de fundo mais um exemplo, são apresentados os seguintes controlos:

- Label
- Text box
- Command button
- Option button
- Check box
- Frame
- List box
- Combo box

4.5.1 Introdução

O exemplo desta parte já nada tem a ver com os exemplos das partes anteriores. Agora, o exemplo consiste em criar uma aplicação para uma empresa do ramo das transportadoras que sirva para fazer a recolha de alguns dados dos seus funcionários.

Neste exemplo pretende-se que depois de introduzir os dados de um funcionário, esses dados sejam guardados numa "*list-box*" de forma a possibilitar a inserção de outros funcionários.

Os dados a recolher são os seguintes:

- Nome
- Categoria
  - Motorista Chefe
  - Motorista de 1ª
  - Motorista de 2ª
  - Estafeta de 1ª
  - Estafeta de 2ª
- Horário de trabalho
  - Fixo
  - Por turnos
- Veículos que pode conduzir
  - A – Motociclos
  - B – Automóveis ligeiros
  - C – Automóveis pesados
  - D – Pesados de passageiros
  - E – Veículos articulados

A seguir apresenta-se a o *form* que o aluno deve construir. Os nomes dos controlos mais importantes são também indicados.

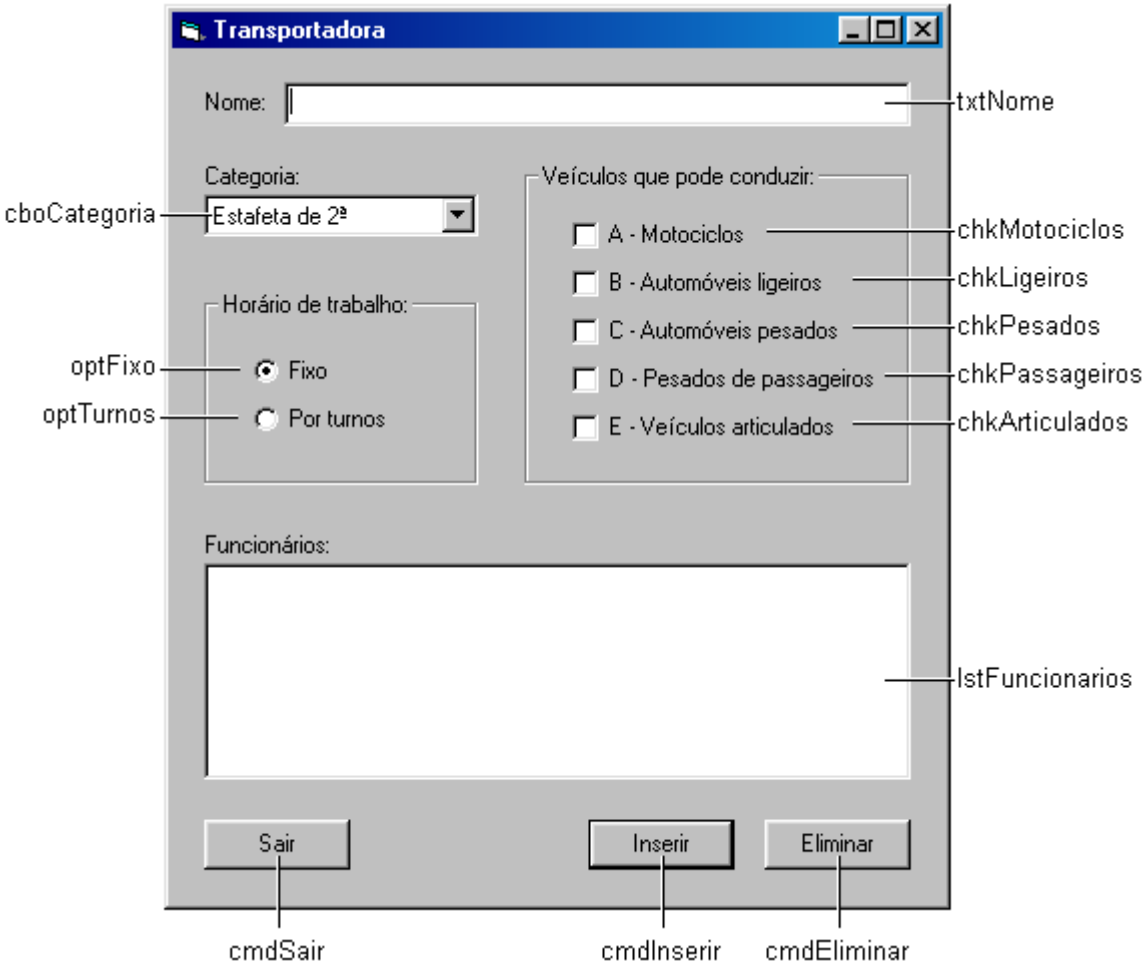


Figura 4.1 – *Form* do exemplo apresentado na última parte do tutorial

Deverá ser possível inserir e eliminar funcionários. Para eliminar um funcionário o utilizador tem primeiro que seleccioná-lo na lista de funcionários e depois premir o botão respectivo.

Antes do código para implementar este exemplo ser apresentado, indicam-se os controlos e respectivas propriedades a alterar.

Objecto	Propriedade	Valor
TxtNome	Text	(vazio)
cboCategoria	Style	2 - Dropdown List
	List	Motorista Chefe
		Motorista de 1ª
		Motorista de 2ª
		Estafeta de 1ª
		Estafeta de 2ª
optFixo	Value	True
lstFuncionarios	Sorted	True
cmdInserir	Default	True

4.5.2 Controlos – Parte I

Esta primeira sub-parte começa com a apresentação do controlo *label* e as suas propriedades mais importantes.

Label 

Contém texto normalmente usado para descrever algo. A sua aparência no ecrã é de texto vulgar. Este texto **não** pode ser modificado pelo utilizador durante a execução da aplicação - pode no entanto modificar-se dinamicamente por ordem do programador.

Principais propriedades:

Propriedade	Descrição
Caption	Contém o texto que é apresentado no *form*.
Alignment	Define o alinhamento do texto da *label* em relação ao espaço que esta ocupa no form. Os valores possíveis são: *0 - Left Justify*; *1 - Right Justify*; *2 - Center*

AutoSize	Determina se o espaço ocupado pela *label* é rigorosamente o necessário para o texto que contém. O valor por defeito é *False* mas quando configurado a *True*, faz com que a *label* aumente ou diminua de tamanho na horizontal, em função do texto que possui.
WordWrap	Quando configurado a *True* faz aumentar o tamanho da *label* na vertical para se ajustar à quantidade de texto.

EXEMPLO – PARTE V

No exemplo apresentado foram usadas 3 *labels* para identificar outros tantos controlos:

- **" Nome: "**, para identificar a *text box* na qual o utilizador deverá introduzir o nome do funcionário
- **" Categoria: "**, para identificar a *combo box* a partir da qual se define a categoria do funcionário
- **" Funcionários: "**, para identificar a *list box* onde serão guardados todos os funcionários da transportadora

De seguida apresenta-se o controlo *text box*, descrevendo também as principais propriedades.

Text box 

Permite a visualização e edição de dados. São uma das formas mais comuns para obter dados do utilizador.

As propriedades mais importantes das caixas de texto são:

Propriedade	**Descrição**
Text	Contém o texto propriamente dito.
Alignment	Define o alinhamento do texto dentro da *text box*. Os valores possíveis são: *0 - Left Justify*; *1 - Right Justify*; *2 - Center*
MaxLength	Permite definir o número máximo de caracteres que podem ser introduzidos na *text box*.
MultiLine	Permite definir se a *text box* irá aceitar ou não múltiplas linhas de texto.
PasswordChar	Permite definir se os caracteres introduzidos na *text box* serão visíveis ou se no seu lugar aparecerão asteriscos para ocultar o texto.

EXEMPLO – PARTE V

No exemplo usamos uma *text box* (**txtNome**) para o utilizador introduzir o nome do funcionário. O texto que existia na propriedade **Text** deste controlo foi eliminado para a caixa de texto a aparecer vazia quando se corre a aplicação.

Por fim, surge o *command button*. É feita a sua apresentação e descritas as principais propriedades.

Command button 

É um controlo que executa uma acção (previamente definida pelo programador) quando premido. A sua aparência no ecrã é de um botão, normalmente para ser premido pelo rato.

Propriedades mais usadas:

Propriedade	Descrição
Caption	Contém o texto que é apresentado no botão.
Default	Permite especificar se o botão em causa é o botão por defeito no *form*. Se esta propriedade estiver a *True* então sempre que o utilizador carregar na tecla **enter**, e o *focus* não estiver em nenhum outro botão, será executado o código associado a esse botão.
Enabled	Indica se o botão está disponível ou não. Quando não está disponível o texto do botão aparece cinzento.

EXEMPLO – PARTE V

No exemplo foram usados 3 botões:

- **Inserir**: usado para inserir os dados de um funcionário na lista de funcionários; quando premido executa o procedimento **cmdInserir_Click()**
- **Eliminar**: usado para remover os dados de um funcionário da lista; quando premido executa o procedimento **cmdEliminar_Click()**
- **Sair**: usado para terminar a aplicação; quando premido executa o procedimento **cmdSair_Click()**

No *form* consegue-se ver que o botão **Inserir** tem um rebordo preto diferente dos outros botões. Isto acontece porque foi definido que este botão seria o botão por defeito no form.

4.5.3 Controlos – Parte II

Nesta sub-parte apresentam-se os controlos *check box*, *option button* e *frame*. O primeiro a ser apresentado é precisamente o controlo *check box*.

Check box 

Este controlo funciona como um interruptor, ligando ou desligando uma opção. É normalmente usado para mostrar valores do tipo Sim/Não ou Verdadeiro/Falso. A sua aparência no ecrã é de um quadrado, normalmente para ser premido com o rato. Quando a opção está seleccionada (valor Sim ou Verdadeiro), o quadrado fica com um visto (√).

Uma vez que as *check boxes* são independentes umas das outras, o utilizador pode ligar ou desligar as *check boxes* que entender.

Principais propriedades deste controlo:

Propriedade	Descrição
Caption	Contém o texto que é apresentado ao lado do quadrado - interruptor
Value	Indica se a *check box* está seleccionada (*Value* = *1*) ou não (*Value* = *0*)

EXEMPLO – PARTE V

No exemplo foi usada uma *check box* para cada categoria de veículos, num total de 5:

- A – Motociclos
- B – Automóveis ligeiros
- C – Automóveis pesados
- D – Pesados de passageiros
- E – Veículos articulados

Foram usadas *check boxes* porque um funcionário pode estar habilitado a conduzir mais do que uma categoria de veículos.

Segue-se a apresentação do controlo *option button,* onde é feita uma comparação entre este controlo e o controlo *check box*.

Option button 

Este controlo funciona também como um interruptor, ligando ou desligando uma opção. A sua aparência no ecrã é de um circulo, normalmente para ser premido com o rato. Quando a opção está seleccionada, o circulo fica preenchido por um círculo negro, mais pequeno.

A grande diferença em relação às *check boxes* é que num grupo de *option buttons* apenas uma opção, num dado momento, pode estar seleccionada. Isto significa que a escolha da opção é feita em exclusão mútua (apenas uma).

Mais uma vez, aqui ficam as propriedades mais importantes:

Propriedade	Descrição
Caption	Contém o texto que é apresentado ao lado do circulo - interruptor
Value	Indica se o *option button* está seleccionado (*Value* = *True*) ou não (*Value* = *False*)

EXEMPLO – PARTE V

No exemplo apresentado foram usados dois *option buttons*, um para cada tipo de horário:

- Fixo
- Por turnos

Foram usados *option buttons* porque o que se pretende é que o utilizador escolha um, e só um, tipo de horário. Um funcionário não pode ter um horário fixo e ao mesmo tempo por turnos.

Por defeito (foi assim definido), a aplicação começa com a opção de horário fixo seleccionada. Quando se criam grupos de *option buttons* é normal que uma das opções apareça seleccionada por defeito uma vez que o utilizador é obrigado a escolher sempre uma das opções. Neste caso, para todos os funcionários tem que ser estipulado o seu horário.

Por fim apresentam-se as *frames* e explica-se a necessidade de as usar quando pretendemos ter mais do que um conjunto de *option buttons* no mesmo *form*.

Frame 

BorderStyle	Indica se o *frame* deverá possuir um rebordo a delimitar a sua área (*BorderStyle = 1*) ou não (*BorderStyle = 0*). Para que seja visível o texto definido no *Caption*, o *frame* terá que ter o rebordo.

EXEMPLO – PARTE V

No exemplo foram desenhados dois *frames*, um para cada grupo lógico:

- Horário de trabalho
- Veículos que pode conduzir

O primeiro é um grupo de *option buttons*, o segundo é um grupo de *check boxes*.

4.5.4 Controlos – Parte III

Para finalizar a apresentação dos controlos mais usados no desenvolvimento de aplicações em Visual Basic, falta apresentar os controlos *list box* e *combo box*.

O controlo *list box* vem em primeiro lugar, sendo dado especial destaque à forma como se insere e elimina itens da lista. São ainda descritas as propriedades mais importantes.

List box 

Este controlo mostra uma lista de opções passíveis de escolha, indicando com uma barra escura a selecção actual. Por defeito, as opções são apresentadas numa única coluna vertical, podendo-se no entanto definir várias colunas. Se o número de opções exceder a quantidade de itens que pode ser visto ao mesmo tempo na *list box* surgem barras de deslocamento verticais e/ou horizontais.

A introdução dos itens na lista pode ser feita enquanto se desenham os controlos no *form* ou durante a execução da aplicação.

Os elementos da lista estão guardados numa estrutura de dados acessível através da propriedade **List(n)** que simula um array de strings. Assim, o primeiro elemento encontra-se em **List(0)**, o segundo elemento em **List(1)** e assim sucessivamente.

É usando esta propriedade que se podem inserir elementos na lista enquanto se desenham os controlos no *form*. Como inserir elementos durante a execução da aplicação é o que vamos ver a seguir.

Inserir um item numa lista

A inserção de elementos na lista durante a execução da aplicação deve ser feita usando o método **AddItem**, cuja sintaxe é:

- **<*lista*>.AddItem <*item*>[, <*posição*>]**

Se **posição** for especificado, o item é inserido no local indicado, se não for especificado é inserido no fim da lista. A primeira posição da lista é a posição 0.

Remover um item de uma lista

O método usado na eliminação de um item da lista é o **RemoveItem** e a sintaxe é a seguinte:

- **<*lista*>.RemoveItem <*posição*>**

Aqui, **posição** representa o índice do elemento a eliminar.

A posição do item seleccionado é dada pela propriedade **ListIndex** que apenas se encontra disponível durante a execução do programa. O valor desta propriedade é **0** se estiver seleccionado o primeiro item da lista, **1** se o item seleccionado for o seguinte, e assim por diante. **ListIndex** tem o valor **-1** se nenhum item está seleccionado.

Para remover todos os elementos da lista pode-se fazer: **<*lista*>.Clear**

As propriedades mais usadas são:

Propriedade	Descrição
Style	Determina se devem ou não aparecer *check boxes* nos elementos da *list box*. Se tiver o valor 0 então não aparecem *check boxes*, se tiver o valor 1 aparece uma *check box* para cada item da lista.
Sorted	Indica se os elementos da lista devem aparecer por ordem alfabética
List	Contém os itens que compõem a lista quando se corre a aplicação
Columns	Indica se a lista de itens deve aparecer numa única coluna (*Columns* = *0*) ou em 2, 3, ... colunas
MultiSelect	Define se o utilizador pode seleccionar mais do que uma opção em simultâneo

EXEMPLO – PARTE V

No exemplo apresentado usou-se uma *list box* para guardar os funcionários da transportadora à medida que os seus dados forem sendo introduzidos.

Para inserir os dados de um funcionário na lista declarou-se uma variável do tipo string designada por **Dados**, à qual se vai concatenando os dados do funcionário. A concatenação de strings é feita usando o operador respectivo (**&**):

```
Dados = Trim(txtNome.Text) & ", " & cboCategoria.Text & ", "
...
```

Por fim, quando a string já possui todos os dados relativos ao funcionário é feita a inserção na lista:

```
lstFuncionarios.AddItem Dados
```

Para remover um funcionário da lista começa-se por verificar se existe algum funcionário

seleccionado:

```
If lstFuncionarios.ListIndex = -1 Then
...
```

No caso da condição anterior ser falsa então existe um funcionário seleccionado e pode-se então proceder à sua eliminação da lista:

```
lstFuncionarios.RemoveItem lstFuncionarios.ListIndex
```

Depois surge o controlo *combo box*. É feita referência aos tipos de *combo boxes* existentes e à semelhança entre este controlo e as *list boxes*.

Combo box 

Style	Pode ser um dos 3 tipos atrás explicados: *0 - Dropdown Combo*; *1 - Simple Combo*; *2 - Dropdown List*
Sorted	Indica se os elementos da *combo box* devem aparecer por ordem alfabética
List	Contém os itens que compõem a *combo box* quando se corre a aplicação

EXEMPLO – PARTE V

Neste exemplo usou-se uma *combo box* que indica as categorias existentes na empresa. Esta *combo box* é do tipo "drop-down list box" porque se pretende que na inserção de um funcionário se escolha a sua categoria de entre as categorias apresentadas.

No arranque da aplicação a *combo box* é "inicializada" de forma a que surja logo com uma categoria seleccionada por defeito - Estafeta de 2ª. Isso é feito com o código:

```
cboCategoria.ListIndex = 4
```

5

Conclusões

Neste capítulo tiram-se conclusões sobre o trabalho realizado e perspectivam-se desenvolvimentos futuros.

O tutorial proposto constitui o principal contributo deste trabalho. A sua principal característica resulta do facto de ser baseado na metodologia de ensino Learning by Example e por isso apresentar ao aluno exemplos que ele deve implementar à medida que vai evoluindo no tutorial.

Tal como já se disse, este tutorial foi desenvolvido com o intuito de vir a ser usado nos cursos de Engenharia (Química, Civil, Mecânica, ...). Sendo assim, fica-se à espera dos comentários dos alunos, no sentido de se poder vir a fazer alguns ajustamentos que naturalmente se justifiquem.

Como desenvolvimentos futuros pode-se referir o alargamento do tutorial a alguns conceitos mais avançados quer ao nível da algoritmia, quer ao nível do próprio ambiente de desenvolvimento.

Assim, pode-se mencionar como possível enriquecimento, a inclusão no tutorial de um capítulo sobre como trabalhar com ficheiros, outro sobre o acesso a base de dados, ou ainda outro que apresente mais alguns controlos disponíveis no Visual Basic (*image*, *scroll bar*, *data control*, *OLE*, etc).

BIBLIOGRAFIA

"Microsoft Visual Basic – Programmer's Guide"
Microsoft Corporation

"MSDN: Microsoft Developer Network Library – Visual Studio 6.0"
Microsoft Corporation

Internet

Tutoriais:
http://members.tripod.com/~vkliew/vbtutor.html
http://vbtutorial.programmer.webjump.com
http://vb-world.net/beginning/vbtutorial
http://www.ms-vb.com
http://www.dotcom2001.com/trainingvb
http://lockledge.eng.wayne.edu/be101/tutorial/main.html
http://www.xploiter.com/programming/vb/vb5_tutor
http://www.iessoft.com/scripts/beginner.asp
http://www.dcs.napier.ac.uk/hci/vb50
http://www.vbinformation.com/tutor.htm
http://klingon.cs.iupui.edu/~aharris/220vb/220s97.html
http://www.cyber-matrix.com/lbasic.htm
http://www.geocities.com/~chuckeasttom/vb/vb.htm

Projecto WeMeet:
http://www.uned.ipp.pt/wemeet
http://www.indutec.be/wemeet